Empirical Software Engineering
using R

Derek M. Jones
derek@knosof.co.uk

Contents

Read me 1st

This book aims to discuss all of what is currently known about software engineering, based on an analysis of all publicly available software engineering data.

This aim is not as ambitious as it sounds because there is not a great deal of data publicly available. Until recently researchers in software engineering concentrated on producing work that gave readers mathematical orgasms, rather than anything useful to industry based on experimental evidence.

As work progressed, it became obvious that the best way to organise the material was as two parts, one covering software engineering and the second the statistics used in the analysis of software engineering data.

Life would have been easier if your author could have pointed readers at other books to learn about statistics. Unfortunately existing statistics books are not suitable, for reasons which include:

- they contain too much implementation detail. Developers are casual users of statistics and don't want to spend time learning lots of mathematics; they want to use the techniques, not implement them, and are only interested in the pre and post conditions,
- they target the largest market for introductory statistics books, the social sciences; the characteristics of the data encountered in the social sciences are very different from the data encountered in software engineering, e.g., the Normal distribution is commonly encountered in social science data and not that common in software engineering data, while the exponential distribution is common in software engineering and much less common in the social sciences,
- continuing to use statistical techniques that were designed to be practical for manual implementation (because electronic computers had not yet been invented). If fast computers are available, it is possible to use more powerful techniques which are impractical for manual implementation.

It is assumed that developer time is expensive and computer time is cheap. Where possible a single, general, statistical technique is described and a single way of coding something in R is used. This minimal, but general approache focuses on what developers need to know and the price paid is that their R code may be slower (in many cases the performance slowdown is unlikely to be noticeable).

The approach used is similar to that of a Doctor examining a patient for symptoms that can be matched against known underlying processes.

In some cases the questions I have asked about a particular set of measurements and the techniques I have used are very different from those performed by the researchers who made the original measurements. Reasons for this include needing to illustrate a particular technique and making use of more powerful techniques to extract more information.

The statistics for software engineering material has settled down and hopefully will be useful to people; there are some "...", "???" and places where the document production chain throws a wobbly. The rest of the material is not yet in a state that would be useful and chapters will be released as they settle down.

The completed pdf will be made available for free online.

Formatting of large values on graph axis is courtesy of a time-machine from the 1970s; this is on the TODO list.

If you have questions about empirical software engineering that this material does not answer, please let me know.

Even better, if you know of some interesting software engineering data, please tell me where I can download a copy.

All the code and data can be downloaded at: github.com/Derek-Jones/ESEUR-code-data

The names of data files usually share the same sequence of initial characters as the pdf file names of the corresponding research paper downloaded from the Internet.

Chapter 1

Introduction

Software systems and their host, the electronic computer, entered the human ecosystem around 70 years ago.[i] During this growing up period the price of computer equipment has continually declined, averaging 17.5% per year,[572] an economic hurricane that has continually increased the number of tasks handled by software systems. Figure 1.1 shows the dramatic fall in the cost of performing compute operations. However, without cheap mass storage computing would be a niche market; the continued reduction in the cost of storage created many new ways of employing this cheap processing power.

Since computers were first invented[182] people have learned how to do software development through experience of what works, it is a craft activity. Most activities that build things start with people learning through their own and sometimes others' experience. In some cases what is being created is important enough to be worth investing in research to develop an effective engineering/scientific approach; the benefits of using such an approach are greater control and predictability.

Why has software engineering not progressed from a craft to an engineering activity?

The engineering/scientific approach is based on theories that have been validated using empirical data. Until recently measurement data relating to the creation of software systems has either been unavailable or somewhat insubstantial. Over the last few years there has been an explosive growth in the collection and analysis of empirical data; it is now possible to write a book such as this one.

The lack of available data on commercial software development is a consequence of the longstanding relationship that has existed between customers and vendors. Change is driven by those with the power to make it happen and software systems has been a sellers market. The benefits of an engineering approach, are primarily reaped by customers, why should vendors invest in change if customers are willing to continue paying for systems developed using existing practices?

Those involved in building systems that use software want to control the process. Control requires understanding; understanding of the many processes involved in building software systems is the goal of software engineering research.

This is a data driven book that treats software engineering as an economic and cultural activity; it is intended to be useful to those involved in building software systems. A topic is only discussed if measurement data is publicly available to ground the discussion. Keeping to this requirement means that readers are likely to be dismayed at the scant coverage of many topics often covered at length in other books on software engineering.

Software is written within a particular development culture, by people having their own unique and changeable behavior patterns. Measurements of the products and processes in this environment are intrinsically noisy and are likely to include a variety of variables that are not measured. This does not mean that measurement and analysis is a futile activity, just that the uncertainty and variability is likely to be much larger than typically found in other engineering disciplines.

Measurement data is analysed using statistics, with statistical techniques being wielded as weaponised pattern recognition; those wanting to discuss statistics in a way that acts as a stimulant for mathematical orgasms will not find satisfaction here.

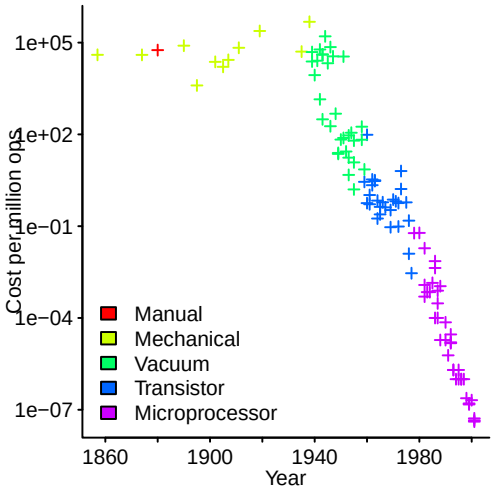


Figure 1.1: Total cost of one million computing operations over time. Data from Nordhaus.[418] code

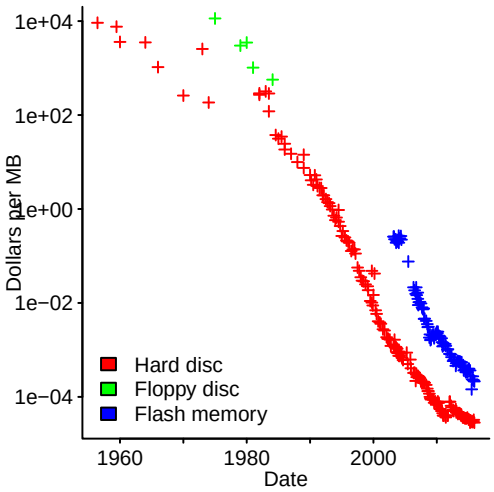


Figure 1.2: Storage cost, in US dollars per Mbyte, of mass market technologies over time. Data from McCallum.[375] code

[i] The verb "to program" was first used in its modern sense during 1946.[232]

Statistics does not assign a meaning to any of the patterns it uncovers; interpreting the patterns thrown up by statistical analysis, to give them meaning, is your job dear reader (based on your knowledge and experience of the problem domain).

The tool used for statistical analysis is the R system.[469] R was chosen because of its ecosystem; there are many books, covering a wide range of subject areas, using R and very active online forums discussing R usage (answers to problems can often be found by searching the Internet or if none are found a question can be posted with a reasonable likelihood of receiving an answer).

All the source code and data used in this book is freely available for download from the book's website.[304]

Like programming, data analysis contains a craft component and the way to improve craft skills is to practice.

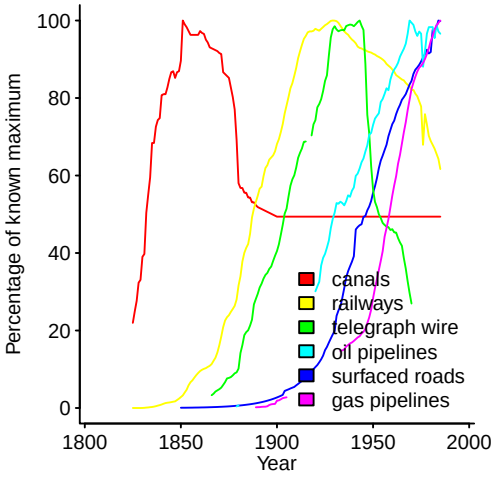


Figure 1.3: Growth of transport and product distribution infrastructure in the USA (underlying data is measured in miles). Data from Grübler et al.[235] code

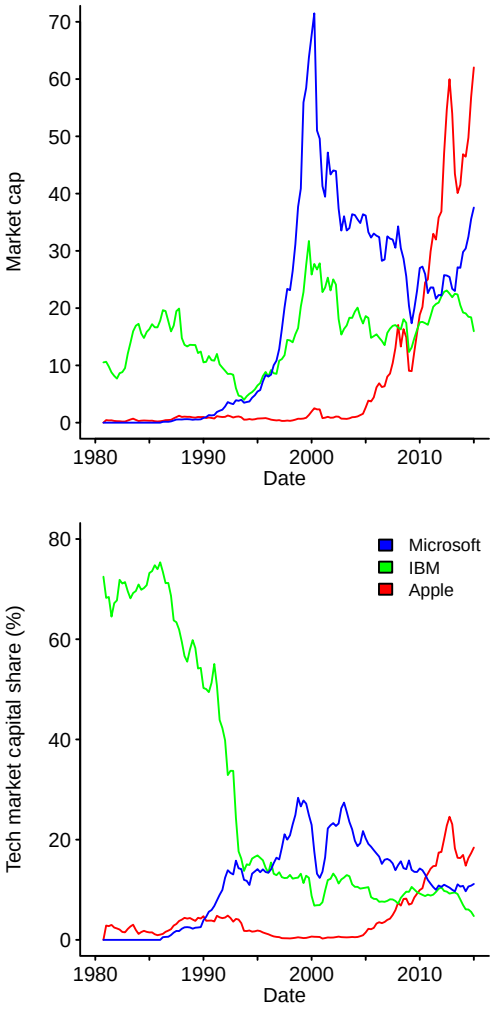


Figure 1.4: Market capitalization of IBM, Microsoft and Apple (top) and expressed as a percentage of the top 100 listed US tech companies (bottom). Data extracted from the Economist website.[155] code

1.1 Software ecosystems

The last 50 years or so has been a sellers market. Software vendors learned that the only way to survive in a rapidly evolving unpredictable market was to create products quickly, before things moved on; customers were engulfed and became red queens who had to keep running to stay in the same place or be eclipsed by competitors. Provided vendors delivered something that was good enough, customer complaints could be ignored; standing still was seen as a risky option.

Experience from the introduction of earlier high-tech industries suggests that it takes many decades for major new technologies to settle down and reach market saturation. For instance, the evolution from the use of wood to steel for battleships[431] which started in 1858 and reached it zenith during the second world war, and there is a long history of growth and decline various forms of infrastructure in the US (see Figure 1.3).

Over the last 70 years a succession of companies have dominated the computing industry, and all the related major niche markets. The continual reduction in the cost of computing platforms created new markets and occassionaly one of these grew to be the largest market, financially, for computing resources. A company dominates the computer industry when it dominates the market that dominates the industry. Companies that lose computing industry dominance often continue to grow, only declining when the market they continue to dominate declines.

Figure 1.4 illustrates the impact of the growth of new markets on the market capitalization of three companies; IBM dominates the mainframe market (which is no longer the dominant form of computing device), Microsoft the desktop software market (until mobile phones removed the need for computers to sit on desks) and Apple the mobile handset market (Google don't charge for Android, but profit from it indirectly, making it very difficult to estimate its contribution to Google's market capitalization). It shows market capitalization (top) and as a percentage of the top 100 listed US tech companies (bottom), as of the first quarter of 2015.[155]

The three major eras, each with its own particular product and customer characteristics, have been:

- the IBM era (with mainframes as its main money spinner); very expensive computers sold, or rented, to large organizations who either rented software from the hardware vendor or paid for software to be developed specifically for their own needs. In this ecosystem a single entity usually paid for software to be created, maintained and incurred the costs of any faulty operation of that software.

  When the actual cost of software faults experienced by one organization is very high (with potential for even greater costs if things go wrong) and the same organization is paying all or most of the cost of create the software, that organization can see a problem that is costing it money, that it thinks it should have control over. Very large organizations are in a position to influence research agendas to target the problems they want solved.

  Large organizations tend to move slowly. The rate of change was slow enough for experience and knowledge of software engineering to be considered essential to do the job (this is not to say that anybody had to show that their skill was above some a minimum level before they would be employed as a software developer).

- the Wintel era (with Microsoft Windows as the dominant software vendor and Intel the dominant hardware vendor); money to be made selling software packages to very many customers. The direct cost of software maintenance is not visible to these customers, but they are paying for the costs of the consequences of faulty operation of that software.

  Microsoft's mantra of a PC on every desk required that people write software to cover niche markets. The idea that anyone could create an application was promoted as a means of spreading Windows, by encouraging people with application domain knowledge to create software running under MS-DOS and later Windows.

  Programming languages, libraries and user interface experienced very rapid rates of change, which meant that developers found themselves on a learning treadmill. One lesson that many learned was that it was that the likelihood of change did not make it worthwhile investing in becoming an expert in a particular area.

- the Internet era (some large niches have dominant vendors, e.g., mobile phones, but no single vendor dominates the Internet).

  It is difficult to judge whether the rate of change has been faster than in previous eras, or the volume of discussion about the changes has been higher because the changes have been visible to more people, or the lack of a dominant vendor to prevent change occurring too quickly.

Figure 1.5 shows unit sales by platform and product using particular computing devices.[217,266]

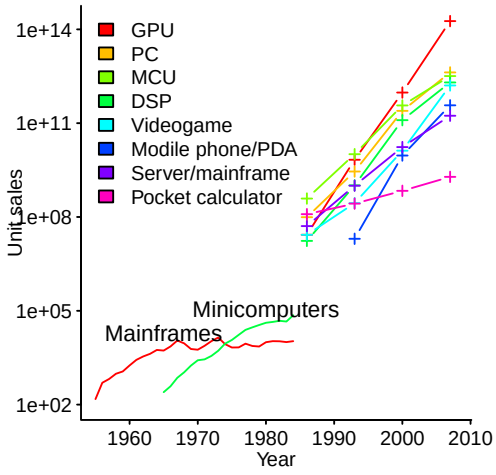


Figure 1.5: Unit sales of processors used in various ecosystems. Data from Gordon[217] (mainframes and minicomputers) and Hilbert et al[266] (post 1985 hardware). code

The ongoing history of new software systems and computing platforms has created an environment where people are willing to invest their time and energy creating what they believe will be the next big thing. Those with the time and energy to do this tend to be young and inexperienced, outsiders in the sense that they don't have any implementation experience with existing systems. If any of these new systems take off, the developers involved will have made, or will make, many of the same mistakes made by the developers involved in earlier systems. The rate of decline of major software platforms is slow enough that employees with significant accumulated experience and expertise can continue to enjoy their seniority in well-paid jobs and have no incentive to jump ship to apply their expertise to an emerging system.

Software systems have yet to reach a stable market equilibrium in many of the ecosystems they have colonised.

Many software systems are still new enough that they are expected to adapt when the ecosystem in which they operate changes. The operating characteristics of such systems are not yet sufficiently entrenched that they enjoy the influence of making it easier to change the world to operates around them.

Embedded systems is a software ecosystem that does not appear to have attracted much attention from the academic software engineering research community. Figure 1.6 shows that in terms of microprocessor sales volume the embedded systems market is significantly larger than what might be called the desktop computer market. mobile market...

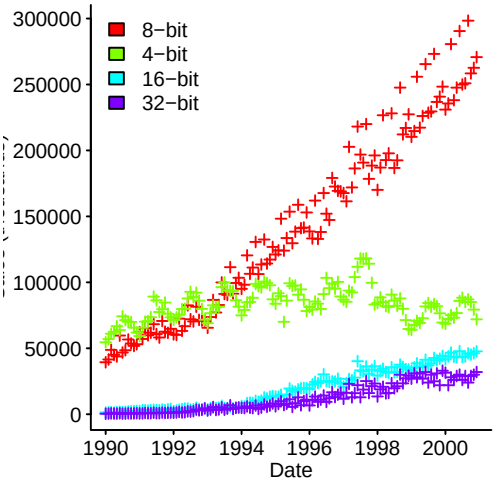


Figure 1.6: Monthly unit sales (in thousands) of microprocessors having a given bus width. Data kindly supplied by Turley.[575] code

A study by Wang[601] found that while firms associated with the current IT fashion have a higher reputation and pay their executives more, they do not have a higher performance. As companies in other industries have discovered, continuing commercial success requires a steady stream of sales.[527] Given that software does not wear, fashion provides a mechanism for encouraging users to obtain the latest version.

To what extent can results obtained from measurements of Open Source be applied to commercial software development? Software developers probably have a larger say in what features are added to the software they are working on... Successful Open Source projects probably have many more users than most commercial systems (which may only be used internally or have a few large customers)...

A study by Zhang[627] of changes made, over 12 years, to 100 programs used internally by a large company found that the rate of change was orders of magnitude smaller...

No Kondratieff waves found in the US data, only Kuznets swings...[?,?]

1.1.1 What activities are part of software engineering?

Software engineering is the collection of activities performed by people involved in producing and maintaining software executed by computer to solve a problem.

These activities will have some path dependency, that is, once the know-how and infrastructure for performing some activity becomes established the existing practice is likely to continue.

Perhaps the most entrenched path dependency in software development is the use of two-valued logic, i.e., binary. The most efficient radix, in terms of representation space (number of digits times number of possible values of each digit), is, 2.718...;[253] the closest integral value is 3. The use of binary, rather than ternary, was driven by the characteristics of the available electronic switching devices.[ii] Given the vast quantity of software making an implicit, and in some cases explicit, assumption that binary representation is used, a future switching technology that would support the use of a ternary representation might not be adopted or be limited to resource constrained environments.[604]

Traditionally activities have included: obtaining requirements, creating specifications, design at all levels, writing and maintaining code, writing manuals fixing problems and providing user support. Employees within large organizations specialise within a particular area and these activities are the areas where software developers specialise.

In small companies there is greater opportunity, and sometimes a need, for employees to become involved in tasks that would not be considered part of the job of a software developer in larger companies. For instance, being involved in any or all of a company's activities from the initial sales inquiry through to customer support of the delivered system; the financial aspect of running a business is likely to be much more visible in a small company.

Software is created and used within a variety of ecosystems and software engineering activities can only be understood in the context of the ecosystem in which it operates.

While this is an overly broad definition, let's be ambitious and run with it, allowing the constraints of data availability and completing a book to provide the filtering.

...[49]

1.2 Brief history of software engineering research

The fact that software often contained many faults and took much longer than expected to produce was a surprise to those working at the start of electronic computing, after World War II. Established practices for measuring and documenting the performance of electronics were in place and ready to be used for computer hardware.[328,399] It was not until the end of the 1960s that a summary of the major issues appeared in print.[407]

Until the early 1980s most software systems were developed for large organizations and over 50% of US government research funding for mathematics and computer science came from the Department of Defense,[181] an organization that built large systems over time-frames of many years.

Large organizations, such as the DOD, spend so much on software that it is worth their while investing in research aimed at reducing costs and learning how to better control the development process. During the 1970s project data, funding and management by the Rome Air Development Center,[iii] RADC, came together to produce the first collection of wide-ranging empirically based reports analysing the factors involved in the development of large software systems.[567,579]

For whatever reason the data available at RADC was not widely distributed or even known about; the only people making use of this data in the 1980s and 1990s appear to be Air Force officers writing Master's thesis.[429,518]

[ii] In a transistor switch, Off is represented by very low-voltage/high-current and On represented by saturated high-voltage/very low-current. Transistors in these two states consume very little power (power equals voltage times current). A third state would have to be represented at a voltage/current point that would consume significantly more power. Power consumption, or rather the heat generated by it, is a significant limiting factor in processors built using transistors.

[iii] The main US Air Force research lab.

The legacy of this first 30 years was a research agenda oriented towards building, in their day, very large systems.

Most published software engineering research since around 1980 has not made use of empirical evidence; unless empirical research does not include theoretical contribution it may fail to be accepted for publication.[3] A review[363] in the early 1990s, of published papers relating to software engineering, found an almost total lack of empirical analysis of its engineering characteristics; a systematic review of 5,453 papers published between 1993 and 2002[246] found 2% reporting experiments. When experiments had been performed, they suffered from small sample sizes[318] (a review[599] using papers from 2005 found that little had changed), had statistical power falling well below norms used in other disciplines[154] or simply failed to report an effect size (for the 92 controlled experiments published between 1993 and 2002 only 29% reported an effect size[318]).

Why have academics working in an engineering discipline not based their research on empirical data? Perhaps the difficulty of obtaining realistic data[463] was an important factor (commercial companies are loath to have outsiders measuring what they are doing and many do not measure themselves, so even confidential is hard to find). The lack of data discouraged anybody wanting to do intellectually sound research, with those investigating the field learning how difficult it is to make worthwhile progress without reliable data (a few intrepid souls did run experiments using professional developers). Anybody with a talent for software engineering either moved on to other research areas or found a job in industry, leaving the field to those without this talent (or perhaps talents in the less employable areas of mathematical theory, literary criticism (of source code) or folklore[81]).

A lack of data has not prevented researchers creating plausible sounding theories that have in some cases have become widely regarded as true. For instance, it was once claimed, without any empirical evidence, that the use of source code clones (i.e., copying code from another part of the project) is bad practice (e.g., clones are likely to be a source of faults, perhaps because only one of the copies was updated).[191] In practice research has shown that the opposite is true,[471,569] clones are less likely to contain faults than *uncloned* source.

Given the sparsity of available empirical data relating to industrial software systems, it is no surprise that academics researchers have failed to create any reliable predictive or descriptive software engineering theories that had a connection to reality.[iv]

Many of the theories relating to software engineering processes commonly encountered in academic publications are based on ideas created many years ago by somebody who was able to gain access to a relevant (often very small) dataset. For instance, Perry[448] divided software interface faults into 15 categories using a data set of just 85 modification requests to draw conclusions; this work is still being cited in papers 25 years later. These fossil theories have continued to exist because of the sparsity of data needed to refute or improve on them.

Human psychology and sociology continue to be completely ignored as major topics of software research, a fact pointed out over 35 years ago.[513]

Quantity of published papers should not be confused with progress towards an effective model of behavior. For instance there has been a great deal published on software process improvement, SPI, (635 papers were found by a recent study[336] of research over the last 25 years), with experience reports and proposed solutions making up two-thirds of the publications; however, the proposed solutions were barely evaluated, there were no studies evaluating advantages and disadvantages of proposals and the few testable theories are waiting to be tested.

Over the last 10 years or so there has been an explosion of empirical research, driven by the availability of large amounts of data extracted from open source software. This significant change in the availability of data and the resulting boom in the fortunes of empirical research does not mean that existing theories and ideas will change overnight. Simply ignoring all research published before 2005 (roughly when the data deluge started) does not help, earlier research has seeded old wives tales that have become embedded in the folklore of software engineering creating a legacy that is likely to be with us for sometime to come.

It is inevitable that some early papers will make claims about software engineering that we now believe to be true. It is possible to search though the contents of old papers looking for wording that can be interpreted as a connection with a recent discovery in the same way that it is possible to search through the prophecies of Nostradamus to find one that can be interpreted as predicting the same discovery.

[iv] The data that was available was generally very small, with little supporting context. See (am I really going to create this collection???) for a collection of historical data sets.

This book takes the approach that empirically verified theories in software engineering don't yet exist, the subject is a blank slate. Knowing that certain patterns of behavior occur on a regular basis is an empirical observation, a theory would make verifiable predictions that include the observed patterns. In some cases existing old wives tales are discussed if it is felt that their use in an engineering environment would be seriously counter-productive. For instance, while various software metrics (e.g., Halstead's metric) are widely known, your authors' experience is that practicing developers do not invest effort in using them; they are famous for being famous and so there is little to be gained in spending much effort debunking them.

1.2.1 The academic ecosystem

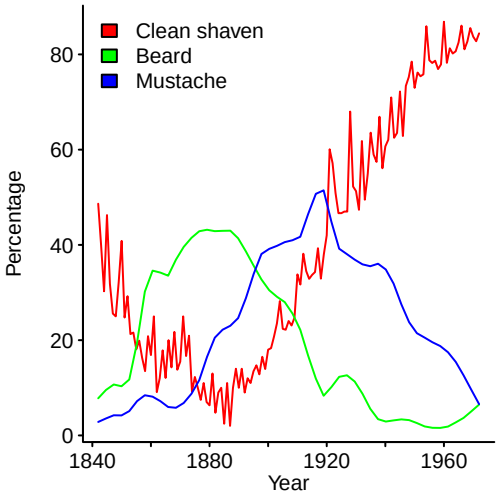


Figure 1.7: Changing habits in men's facial hair. Data from Robinson.[485] code

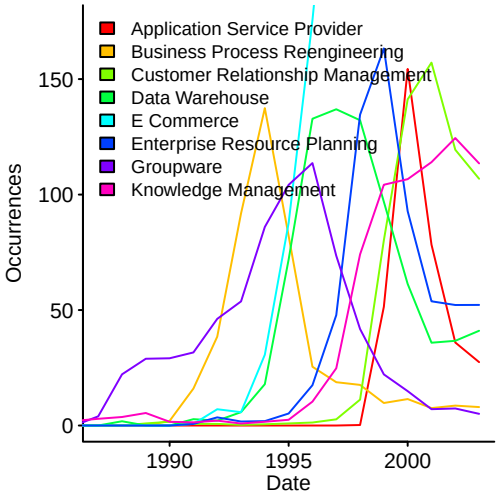


Figure 1.8: Number of papers, in each year between 1987 and 2003, associated with a particular IT topic. The E-commerce paper count peaks at 1,775 in 2000 and in 2003 is still off the scale compared to other topics. Data kindly provided by Wang.[601] code

Interactions between people in industry and academia has often suffered from a misunderstanding of each other's motivations and career pressures.

Few people in industry have much interaction with the academic ecosystem that researches their field. The quaint image of researchers toiling away for years before publishing a carefully crafted manuscript is long gone. Although academics still hold positions in a feudal based system of patronage and reputation, they are incentivised by the motto 'publish or perish',[438] with science perhaps advancing one funeral at a time.[30] Hopefully the change to empirically based research will happen faster than the change in men's facial hair (most academics researching software engineering are men); see Figure 1.7.

Academic research projects share many of the characteristics of commercial startups. They involve a few people attempting to solve a sometimes fuzzily defined problem, trying to make an improvement in one area of an existing *product* and they often fail, with the few that succeed producing spectacular returns. Researchers are serial entrepreneurs in that they tend to only work on funded projects, moving onto other projects when funding runs out (and often having little interest in previous projects). Like commercial product development, the choice of research topics is fashion driven; see Figure 1.8.

The visible output from academic research are papers published in journals and conference proceedings. It is important to remember that: "...an article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures."[84]

In the past most researchers have made little, if any, effort to archive the data they obtain for future use. One study[595] requested the datasets relating to 516 biology related studies, with ages between 2 and 22 years; they found that the probability of the dataset being available fell by 17% per year. Your author's experience of requesting data from researchers is that it often fails to survive beyond the lifetime of the computer on which it was originally held.

Some researchers may have reasons, other than carelessness, for not making their data generally available. For instance, a study[613] of 49 papers in major psychology journals found that the weaker the evidence for the researcher's hypothesis the less likely they were to be willing to share their data.

The importance of making code and data available is becoming widely acknowledged, with a growing number of individual researchers making code/data available for download and journals having explicit policies about authors making this information available.[553] In the UK researchers who receive any funding from a government research body are now required to archive their data and make it generally available.[481]

Discovering an interesting pattern in experimental data is the start, not the end of experimentation involving that pattern (the likelihood of obtaining a positive result is as much to do with the subject studied as the question being asked[172]). The more often a behavior is experimentally observed the greater the belief that the effect seem actually exists. Replication is the process of duplicating the experiment(s) described in a report or paper to confirm the findings previously obtained. For replication to be practical researchers need to be willing to share their code and data, something that a growing numbers of software engineering researchers are starting to do; those attempting replication are meeting with mixed success.[120]

Replication is not a high status activity, with high impact journals interested in publishing research which reports new experimental findings and not wanting to dilute their high impact status by publishing replications (which, when they are performed and fail to replicate previous results considered to be important discoveries, can often only find a home for publication

in a low impact journal). A study[425] replicating 100 psychology experiments found that while 97% of the original papers reported p-values less than 0.05, only 36% of the replicated experiments obtained p-values this low; a replication study[100] of 67 economics papers was able to replicate 22 (33%) without assistance from the authors, 29 with assistance.

Replication is necessary, i.e., the results claimed by one researcher must be reproduced by another if they are to be accepted.

These days academic performance is often measured by number of papers published and the impact factor of the journal in which they were published[337] (scientific journals publish a percentage of the papers submitted to them, with prestigious high impact journals publishing a lower percentage than those with lower impact factors; journals and clubs share a common economic model[459]). Organizations that award grants to researchers often include the number of published papers and the impact factor of the publication journal when deciding whether to award a grant.

One consequence of the important role of published paper count in an academic's career is an increase in scientific fraud,[171] most of which goes undetected;[118] one study[173] found that most retracted papers (67.4%) were attributable to misconduct, around a 10-fold increase since 1975. More highly ranked journals have also been found to contain a higher percentage of retracted papers,[74] it is not known whether this represents an increased in flawed articles or an increase in detection.[545] Only a handful of software engineering papers have been retracted, perhaps because a lack of data makes it very difficult to verify any of the claims made by the authors. The website retractionwatch.com reports on possible and actual paper retractions.

Commercial research labs Many large companies have research groups. While the researchers working in these groups often attend the same conferences as academics and publish papers in the same journals, their performance is often measured by the number of patents granted. The number of software patents continues to grow, with 109,281 granted in 2014[416] (36% of all utility patents).

1.2.1.1 Paper citation practices

This book attempts to provide citations to any claims made, so that readers can perform background checks. To be cited papers have to be freely available for public download, unless published before 2000 (or so), and when data is analysed it has to be free for public distribution.[v]

When academics claim that their papers can be freely downloaded what they sometimes mean is that the university that employs them has paid for a site-wide subscription that enables university employees to download copies of papers published by various commercial publishers. Tax payers pay for the research and university subscriptions and may not have free access to it. Things are slowly changing. In the UK researchers in receipt of government funding are now incentivized to publish in journals that will make papers produced by this research freely available after a period of 6-12 months from publication date. Your author's attitude is that academics are funded by tax payers and if they are unwilling to provide a freely downloadable copy on their web page, they should not receive any credit for the work.[vi]

Software developers are accustomed to regarding documents that are more than a few years old as being out-of-date; a consequence of the fast changing environment in which they operate. Some fields, such as cognitive psychology, are more established and people do not feel the need to keep repeating work that was first performed decades ago (although it might be replicated as part of the process of testing new hypothesis). In these more established fields it is counter-productive to treat date of publication as a worthiness indicator.

1.3 What can be learned from the data analysed in this book?

The gold standard of empirical research is the controlled randomised experiment, followed by replication of the results by others (which may contradict the finding of the original study[286]).

[v] The usual academic procedure is to cite the first paper that proposes a new theory, describes a pattern of behavior, etc.

[vi] In fact they should not have been funded in the first place; if an academic refuses to make a copy of their papers freely available to you please report their behavior to your elected representative.

All the factors that could influence the outcome of an experiment are either controlled or randomly divided up such that any unknown effects add noise to the signal rather than ghost patterns.

A handful of the 210 datasets analysed in this book came from controlled experiments, some of which used randomization, and a handful of this handful have been replicated.

The patterns of behavior seen in the analysis results should be treated as suggestions for what might be. Perhaps their most practical application is in helping to dismiss behaviors that might otherwise be thought possible.

and...

1.4 Overview of contents

This book has two parts:

- a discussion of various topics in software engineering, driven by an analysis of publicly available data. The intent is to cover the currently available empirical evidence, enumerating patterns of behavior that have been found in software engineering activities. Many topics that are usually covered in software engineering textbooks are not discussed because the necessary data could not be found,
- illustration of statistical techniques applicable to the kinds of measurement data and problems likely to be encountered by professional software developers. This provides the foundation for the statistical analysis in the second part.

  The intent is to provide pointers to statistical and experimental techniques that might be used to help analyse a sample of measurements or provide guidance for solving a data driven problem in software engineering.

  While readers will learn something about statistics the material is not intended to provide an introduction to statistics as such, but rather an introduction to the use of statistical concepts and methods, along with the assumptions that underlie them. A suggested reading list is provided...

Human cognitive characteristics The operating characteristics of the human brain is an essential part of any study of software engineering. Characteristics such as ability to expend cognitive effort, memory storage/recall and learning, personality (e.g., propensity to take risks) and processing of visual information are discussed.

Culture and in particular human language are learned characteristics that come preloaded in software developers.

Economics The approach to software economics is from the perspective of the software engineer or software company rather than the perspective of the customer or user of software (which differs from a lot of existing work which implicitly adopts the customer or user perspective).

Basic introductory economic issues are discussed.

Software ecosystems Software is created in a development ecosystem built and maintained by many people and is used in a customer ecosystem that generally involves many people. Ecosystems evolve over time.

Various components of developer (e.g., careers) and development (e.g., APIs) ecosystems are discussed, along with non-software issues having a direct impact on developers, e.g., sales and bidding on projects.

Software development projects A walk through the various stages of building a software system, from requirements to maintenance after delivery. This chapter is short because of a lack of data.

Reliability How can the reliability of a software system be estimated and what is the contribution made by the various components of a system? Why and where do faults occur and when are they worth fixing? Economics and marketing drive the effort invested in searching for and removing faults, as well as deciding which faults are considered acceptable in a shipped product.

Source code Patterns of language usage in source code. What do they tell us about the habits and characteristics of developers who wrote the code?

Stories told by data There is no point analysing data unless the results can be effectively communicated to the intended audience. Possible techniques for communication a story found in data are covered, along with issues to look out for in stories told by others.

Probability An overview and examples involving basic principles in probability, along with commonly used constructs, e.g., probability distributions, means and variance, permutation and randomizations tests.

Statistics for software engineering No statistical knowledge is assumed on the part of the reader, but it is assumed that readers have basic algebra skills and can interpret graphs.

The approach used is similar to that of a Doctor examining a patient for symptoms that provide pointers to underlying processes. Developers are casual users of statistics and don't want to spend time learning lots of mathematics; they want to make use of techniques, not implement them.

In many practical situations the most useful expertise to have is knowledge of the application domain that generated the data and how results might be applied.

It is better to obtain an approximate answer to the correct problem than an exact answer to the wrong problem.

Because empirical software engineering is only just starting to develop, there is uncertainty about which statistical techniques are most likely to be generally applicable. Therefore, an all encompassing approach is taken and a broad spectrum of topics are covered, including:

- statistical concepts: sampling, describing data, p-value, confidence intervals, effect size, statistical power, model building, comparing two or more groups, bootstraping,
- regression modeling: fitting data to an equation can answer questions about which variables have the power to explain measurable behavior. Software engineering measurements come in a wide variety of forms and while ordinary least squares might be widely used in the social sciences it is not always suitable for modeling datasets encountered in software engineering; more powerful techniques such as generalized least squares, nonlinear models, mixed models, additive models, structural equation models and others sometimes have to be used,
- time series: analysis of date where measurements at time t is correlated with measurements made at time $t-1$ is handled by time series analysis (regression modeling assumes that successive measurements are independent of each other).
- circular statistics: measurements where the scale used wraps around, e.g., time of day wraps from 24 hrs to 0 hrs.
- survival analysis: measurements of time to an event occurring (e.g., death) are handled by survival analysis.
- machine learning: various techniques for finding patterns in data when having no knowledge of the data or specific application domain. Sometimes we are clueless button pushers and machine learning can be a useful guide.

Experiments General issues involving the design of experiments are covered.

Probably the most common experiment performed by developers is hardware and software benchmarking. The many difficulties and complications involved in performing reliable benchmarks are illustrated.

Another form of experiment is product testing, e.g., website design. In a small company a software developer may have a lot of influence on the company website and knowing how to test the effectiveness of different designs is important.

Surveys: Measurements obtained from asking people questions.

Data cleaning Garbage in garbage out. Data cleaning if often only mentioned in passing, but is often the most time-consuming part of data analysis and a very necessary activity for obtaining reliable results.

Common data cleaning tasks, along with possible techniques for detecting potential problems and solving them using R are discussed.

Overview of R An overview of R aimed at developers who are fluent in at least one other computer language. The discussion concentrates on those language features likely to be commonly used, but behave very differently from languages the reader is likely to be familiar with.

Obtaining and installing R: If you cannot figure out how to obtain and install R, this book is not for you.

RStudio is a widely used R IDE sometimes used by your author.

Many add-on libraries (over 7,000 at the time of writing) are available on CRAN, the Comprehensive R Archive Network. Some packages are only available on R-Forge and Github.

1.4.1 Why use R?

The main reasons for selecting R as the language+support library in which to write the statistical analysis programs in this book were:

- it is possible to quickly write a short program that solves the kind of problems that often occur when analysing software engineering data. The process often follows the sequence: read data from one of a wide variety of sources, operate on it using functions selected from a huge library of existing packages and finally graphically displaying the results or printing values,
- lots of people are using it: there is a very active ecosystem with many R books, active discussion forums where examples can be found, answers to old questions read and new questions posted,
- accuracy and reliability: a comparison of the reliability of 10 statistical software packages[423] found that GAUSS, LIMDEP, Mathematica, MATLAB, R, SAS, and Stata provided consistent reliable estimation results, and a comparison of the statistical functions in Octave, Python, and R[10] found that R yielded the best results. A study[11] of the precision of five spreadsheets (Calc, Excel, Gnumeric, NeoOffice and Oleo) running under Windows Vista, Ubuntu and Mac OS found that no one spreadsheet provided consistently good results (it was recommended that none of these spreadsheets be used for nonlinear regression and/or Monte Carlo simulation); another study[379] found significant errors in the accuracy of the statistical procedures in various versions of Microsoft Excel.
- an extensive library of add-on packages: CRAN, the official library of packages contains over seven thousand packages implementing various statistical techniques.

A freely available open source implementation is always nice to have.

1.5 Terminology, concepts and notation

Much of the basic terminology used in probability and statistics in common use today derives from gambling and experimental research in medicine and agriculture, because these were the domains where researchers working in the early days of statistics were employed.

- *group*: each sample is sometimes referred to as a group,
- *treatment*: The operation or process performed is often referred to as a treatment.
- *response variable* also known as a *dependent variable*, responds/depends to/on changes of values of explanatory variables; their behavior depends on the variables that are independent (at least of them),
- *explanatory variables* (also known as *independent*, *stimulus* or *predictor variables*), are used to explain, predict or stimulate the value of response variables; they are independent of the response variable,

- Data is *truncated* when values below or above some threshold are unobserved (or removed from the dataset).
- Data is *censored* when values below or above some threshold are set equal to the threshold.

between subjects: when samples are obtained from different groups of subjects, often with the different groups performing a task under different experimental conditions,

within subjects: Comparing two or more samples obtaining using the same group of subjects, often with the different subjects performing a task under two or more different experimental conditions,

Some commonly encountered symbols and notation:

- $n!$ (n factorial), denotes the expression $n(n-1)(n-2)\cdots 1$, available in the `factorial` function,
- $\binom{n}{r} = \frac{n!}{r!(n-r)!}$, a commonly occurring quantity in the analysis of problems in probability; available in the `choose` function,
- a hat above a variable, $\hat{y}$, denotes an estimate, in this case an estimate of y's value,
- μ (mu), commonly denotes the mean value, available in the `mean` function,
- σ (sigma), commonly denotes the standard deviation, available in the `sd` function. The terms 1-sigma, 2-sigma, etc are sometimes used to refer to the probability of an event occurring. Figure 1.9 shows the sigma multiplier for various probabilities,
- $n \to \infty$ as n goes to infinity, i.e., becomes very very large,
- $n \to 0$, as n goes to zero, i.e., becomes very very small,
- $P(x)$, the probability of x occurring and sometimes used to denote the Poisson distribution with parameter x (however, this case is usually written using λ, e.g., $P(\lambda)$),
- $P(a < X)$, the probability that $a < X$. The functions `pnorm`, `pbinom` and `ppois` can be used to obtain the probability of encountering a value less than or equal to x for the respective distribution (e.g., Normal, Binomial and Poisson, as suggested by the naming convention),
- $P(|a-X|)$, the probability of the absolute value of the difference between a and X,
- $\prod_{i=1}^{6} a_i$, the product: $a_1 \times a_2 \times \cdots \times a_6$,
- $\sum_{i=1}^{6} P(a_i)$, the sum: $P(a_1) + P(a_2) + \cdots + P(a_6)$,

  The sum the probabilities of all the mutually exclusive things that could happen, when an action occurs, is always one. For instance, when a die is rolled the six probabilities of a particular number occurring sum to one irrespective of whether the die is fair or has been tampered with in some way.
- $P(D|S)$, the probability of D occurring given that S is true; known as the *conditional probability*. For instance, S might be the event that two dice have been rolled and their face-up numbers sum to five and D the event that the value of the first die is four.

  The value can be calculated using the following:

  $$P(D|S) = \frac{P(DS)}{P(S)}$$

  where $P(DS)$ is the probability of both D and S occurring together.

  If D and S are independent of each other, then we have:

  $P(DS) = P(D)P(S)$

  and the above equation simplifies to: $P(D|S) = P(DS)$

  A lot of the theory in statistics assumes that variables are independent. Characteristics unique to each die means using a different die for each roll will produce values having slightly more independence than using the same die twice.

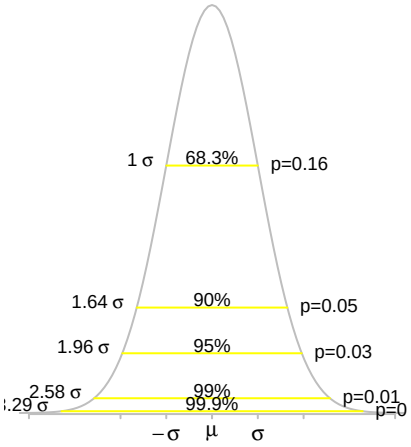


Figure 1.9: Normal distribution with total percentage of values enclosed within a given number of standard deviations. code

Chapter 2

Human cognitive characteristics

2.1 Introduction

Software systems are built and maintained by the creative output of the human brain, which supplies the cognitive effort that directs and performs the activities required; the operating characteristics of this machine are an essential component of any study of software engineering.

Modern humans evolved from earlier humanoids who in turned evolved from earlier members of the ape family who in turn evolved from etc., etc. The collection of cognitive characteristics present in homo sapien brains is the end result of a particular sequence of survival requirements that occurred over millions of years of evolutionary history; with the last common ancestor of the great apes and the line leading to modern humans living 5 to 7 million years ago,[188] the last few hundred thousand years spent as hunter-gatherers roaming the African savannah, followed by 10,000 years or so having a lifestyle that involved farming crops and raising domesticated animals.

Our skull houses a computing system that evolved to provide responses to problems that occurred in the stone age ecosystem. However, neural circuits in the brain established for one purpose can be redeployed,[22] during normal development, for to different uses, often without losing their original functions, e.g., in many people learning to read and write involves repurposing the neurons in the ventral visual occipito-temporal cortex (an area of the barin involved in face processing and mirror-invariant visual recognition).[140] Reuse of neural circuitry is a central organizational principle of the brain.

The collection of cognitive characteristics supported by an animal's brain only makes sense in the context of the problems that had to be handled in the environment in which it evolved; Simon's scissors...

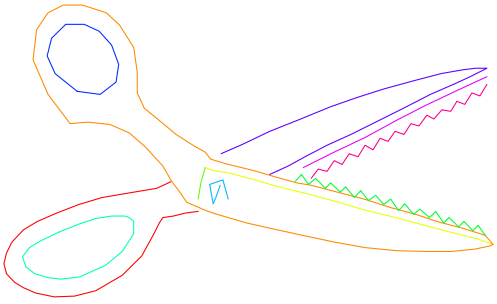


Figure 2.1: Simon's scissors... code

- The structure of the natural environment places constraints on optimal performance (rational analysis),
- Cognitive, perception, and motor operations have their own sets of constraints (bounded cognition).

Cognition and the environment are like the two blades of a pair of scissors (see Figure 2.1), both have to mesh together for things to work...

Suboptimal human performance results when cognitive systems violate the design assumptions of the environment they evolved to operate within. Optical illusions can be produced by preferential biases in the processing of visual inputs that, in most cases, are beneficial (in that they simplify the processing of ecologically common inputs). For instance, the human visual system assumes light shines from above because it has evolved in an environment where this is generally true. A consequence of the assumption of light shining from above in Figure 2.2 is that the top row appears as mounds while the lower row appears as depressions.

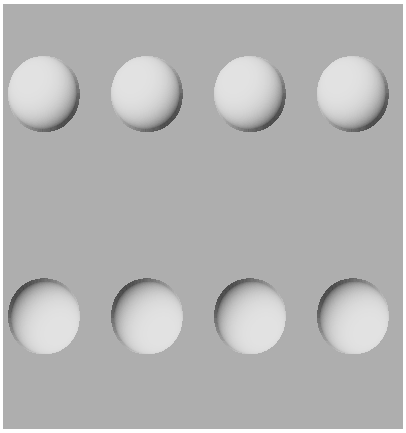


Figure 2.2: Could be more convincing hemispheres with light shining from above and below... code

This effect is not caused by low-level processing of the input from the optic nerve, but by high-level processing of the scene (recognizing the recurring pattern and that some squares are within a shadow).

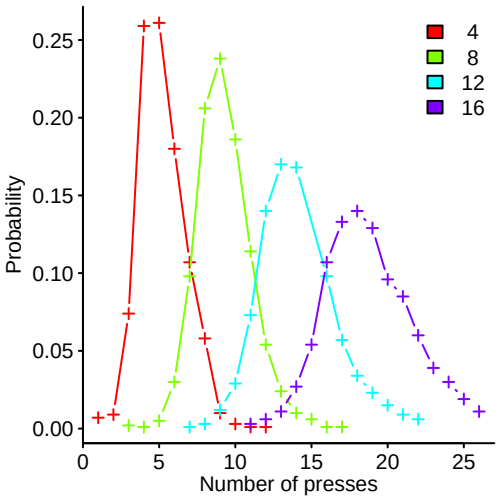


Figure 2.3: Probability that rat N1 will press a lever a given number of times before pressing a second lever to obtain food, when the target count is 4, 8, 12 and 16. Data extracted from Mechner.[383] code

Optical illusions are accepted as curious anomalies of the eye/brain system; there is no rush to conclude that human eyesight is faulty. Failures of the cognitive system to produce answers that agree with some mathematical principle should not be treated as wrong...

This book is written from the viewpoint that how people produce software systems needs to be fitted around the characteristics of the computing platform in our head (the view that developers should aspire to be omnipotent logicians is driven by human self-image and is a counter-productive mindset to hold).[i] Builders of bridges do not bemoan the lack of unbreakable materials available to them, they have learned how to work within the limitations of the materials available.

Evolutionary psychology[42,44] is an approach to psychology which uses knowledge and principles from evolutionary biology to held understand the operation of the human mind. Of course physical implementation details, the biology of the brain,[315] also have an impact on psychological performance.

The fact that cognitive abilities have benefits that outweigh their costs means they are to be found in many creatures,[50] e.g., use of numbers by monkeys[249] and syntax by birds.[557] A study by Mechner[383] rewarded rats with food if they pressed a lever N times (with N taking one of the values 4, 8, 12 or 16), followed by pressing a second lever. Figure 2.3 shows that rat N1 has the ability to use an approximate number system.

Table 2.1 shows a division of human time scales by the kind of action that fits within each interval.

Scale (sec)	Time Units	System	World (theory)
10000000	months		
1000000	weeks		Social Band
100000	days		
10000	hours	Task	
1000	10 min	Task	Rational Band
100	minutes	Task	
10	10 sec	Unit task	
1	1 sec	Operations	Cognitive Band
0.1	100 msec	Deliberate act	
0.01	10 msec	Neural circuit	
0.001	1 msec	Neuron	Biological Band
0.0001	100s	Organelle	

Table 2.1: Time scales of human action. Based on Newell.[409]

Does the brain contain a collection of modules (each handling particular functionality) or a general purpose processor? This question is the nature vs. nurture debate argument, rephrased using implementation details, i.e., a collection of modules pre-specified by nature or a general purpose processor can be influenced by nurture. The term *modularity of mind* refers to a model of the brain[187] containing a general purpose processor attached to special purpose modules that handle perceptual processes (e.g., hearing and sight); The term *massive modularity hypothesis* refers to a model[126] that only contains modules.

Consciousness is the tip of the iceberg, most of what goes on in your mind is handled by the unconscious.[130,609] Problems that are experienced as easy to solve may actually require very complicated neural circuitry and be very difficult for computers to solve...

The only software engineering activities that could be said to be natural, in that there are prewired biological structures in the brain, involve social activities. The exactitude needed for coding is at odds with the fast and frugal approach of our unconscious mind,[204] whose decisions our conscious mind later does its best to justify.[609] Reading and writing are not natural in the sense that specialist brain structures have evolved to perform these activities; it is the brain's generic ability to learn that enables this skill to be acquired.

[i] An analysis of the operation of human engineering suggests that attempting to modify our existing cognitive systems is a bad idea,[71] e.g., it is better to rewrite spaghetti code than try to patch it.

What are the likely differences in cognitive performance between males and females? A study by Strand, Deary and Smith[555] analyzed Cognitive Abilities Test (CAT) scores from over 320,000 school pupils in the UK. Figure 2.4 provides a possible explanation for the prevalence of males at the very competent/incompetent ends of the scale and shows that women outnumber men in the middle competency band.

A study by Jørgensen and Grimstad[310] asked subjects from three Asian and three east European countries to estimate the number of lines of code they wrote per hour and the effort needed to implement a specified project (both as part of a project investigating cognitive biases). A regression model built using the results included both country and gender as significant predictors (see rexample[estimation-biases.R])...

While much has been written on how society exploits women, relatively little has been written on how society exploits men.[46] There are far fewer women than men directly involved in software engineering.[ii]

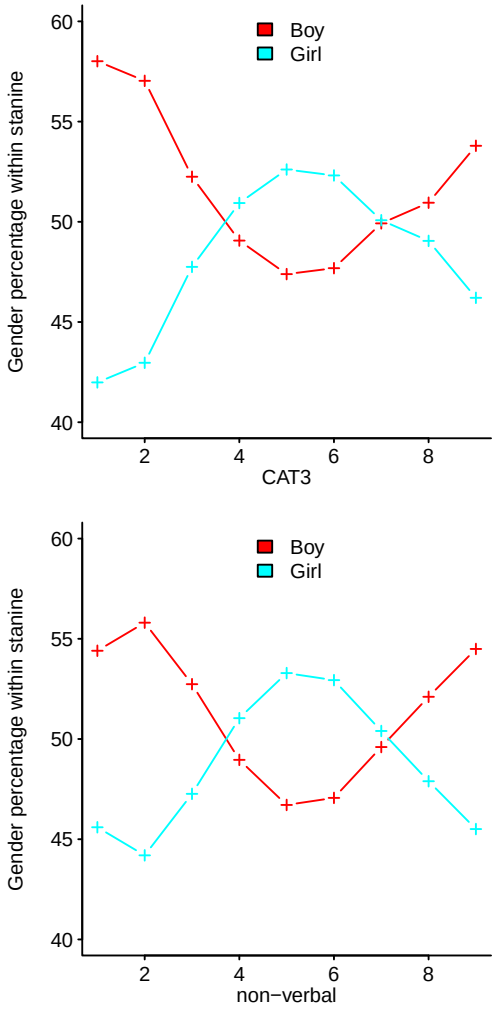


Figure 2.4: Boy/girl (aged 11-12 years) verbal reasoning, quantitative reasoning, non-verbal reasoning and mean CAT score over the three tests; each stanine band is 0.5 standard deviations wide. Data from Strand et al.[555] code

2.1.1 Models of human cognitive performance

Models of human cognitive performance are based on the results of experiments using samples drawn almost entirely from Western, Educated, Industrialized, Rich and Democratic (WEIRD) societies.[258] While this is a problem for those seeking to uncover the characteristics of humans in general, it is not a problem for software engineering because those involved have often had a WEIRD society education.

Characteristics of WEIRD people that appear to differ from the general population include:

- WEIRD people are willing to think about and make inferences about abstract situations, without the need for having had direct experience, while studies[364] of non-WEIRD people have found they are unwilling to discuss situations where they don't have direct experience,
- when mapping numbers onto space WEIRD people have been found to use a linear scale for mapping values between one and ten, while studies of non-WEIRD people have found they often use a logarithmic scale,[143]
- WEIRD people have been found to have complex, but naive, models of the mechanisms that generate every day random events they observe.[18]

Much of the research carried out in cognitive psychology draws its samples from people between the ages of 18 and 21, studying some form of psychology degree. There has been discussion[41] on the extent to which these results can be extended to the general populace, but again results obtained by sampling from this subpopulation are likely to be good enough for dealing with software engineering issues.

The reasons why students are not appropriate subjects to use in software engineering experiments whose results are intended to be applied to professional software developers are discussed in the Experiments chapter.

Several executable models of the operation of human cognitive processes have been created. The ACT-R model[20] has been applied to a wide range of problems, including learning, the visual interface, perception and action, cognitive arithmetic, and various deduction tasks.

Studies[192] have found poor correlation between an individual's estimate of their own cognitive ability and measurements of their ability.

2.1.2 Embodied cognition

Embodied cognition is the theory that aspects of a person's cognitive processing are dependent on or shaped by features of their physical body.

A study by Presson and Montello[465] asked two groups of subjects to memorize the locations of objects in a room. Both groups were then blindfolded and asked to point to various objects;

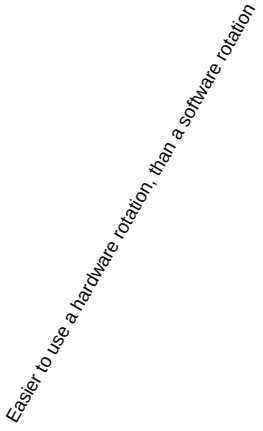


Figure 2.5: Rotate text in the real world, by tilting the head, or in the mind? code

[ii] When your author started working in software development, if there was a woman working on a team the chances were that she would be at the very competent end of the scale (male/female ratio back then was what, 10/1?). These days, based on my limited experience, women are less likely to be as competent as they once were but still a lot less likely, than men, to be completely incompetent; is the small number of incompetence women caused by a failure of equal opportunity regulations or because the underlying population is small?

their performance was found to be reasonably fast and accurate. Subjects in one group were then asked to imagine rotating themselves 90°, they were then asked to point to various objects. Their performance was found to be much slower and less accurate. Subjects in the other group were asked to actually rotate 90°; while still blindfolded, they were then asked to point to various objects. The performance of these subjects was found to be as good as before they rotated. These results suggest that mentally keeping track of the locations of objects, a task that might be thought to be cognitive and divorced from the body, is in fact strongly affected by body position.

Tetris players have been found to prefer rotating an item on screen, as it descends, rather than mentally perform the rotation.[324]

Figure 2.6: Two objects paired with another object that may be a rotated version. Based on Shepard et al.[516] code

A study by Shepard and Metzler[516] showed subjects pairs of figures and asked if they were the same. Some pairs were different, while others were the same, but had been rotated relative to each other. The results showed a linear relationship between the angle of rotation (needed to verify that two objects were the same) and the time taken to make a matching comparison. Readers might like to try rotating, in their mind, the pairs of images in Figure 2.6 to find out if they are the same.

A related experiment by Kosslyn[333] showed subjects various pictures and asked questions about them. One picture was of a boat and subjects were asked a question about the front of the boat and then asked a question about the rear of the boat. The response time, when the question shifted from the front to the rear of the boat, was longer than when the question shifted from one about portholes to one about the rear. It was as if subjects had to scan their image of the boat from one place to another to answer the questions.

2.2 Motivation

Motivation to complete a task can have a powerful effect on an individuals priorities, causing them to change in ways that they would not choose in more relaxed circumstances. A study by Darley and Batson[132] asked subjects (theological seminary students) to walk across campus to deliver a sermon. Some subjects were told that they were late and the audience was waiting, the remainder were not told this. Their journey took them past a victim moaning for help in a doorway. Only 10% of subjects who thought they were late stopped to help the victim, while 63% of the other subjects stopped to help. These results do not match the generally perceived behavior pattern of theological seminary students.

Hedonism is an approach to life that aims to maximise personal pleasure and happiness. Many theories of motivation take as their basis that people intrinsically seek pleasure and avoid pain, i.e., they are driven by *hedonic motivation*.

People involved in work that requires creativity can choose to include personal preferences and desires in their decision-making process, i.e., they are subject to hedonic motivation. Todate, most of the research on hedonic motivation research has involved studies of consumer behavior...

One widely cited theory that has been gathering experimental support is *Regulatory focus theory*. Regulatory focus is based around the idea that people's different approaches to pleasure and pain influences the approach they take towards achieving an end state (or an end goal). The theory contains two end states, one concerned with aspirations and accomplishments (a *promotion focus*), and the other concerned with attainment of responsibilities and safety (a *prevention focus*).

A promotion focus is sensitive to presence and positive outcomes, seeks to insure hits and insure against errors of omission. A prevention focus is sensitive to absence and negative outcomes, seeks to insure against correct rejections and insure against errors of commission.

People are able to exercise some degree of executive control over the priorities given to cognitive processes, e.g., deciding on speed/accuracy trade-offs. Studies[190] have found that subjects with a promotion focus will prefer to trade-off accuracy for speed of performance and those with a prevention focus will trade-off speed for improved accuracy.

The concept of *Regulatory fit*[265] has been used to explain why people engage more strong in some activities and "feel right" about it (because the activity sustains, rather than disrupts, their current motivational orientation or interests).

A study by Luthiger and Jungwirth[366] investigated fun... rexample[LuthigerJungwirth.R]

What motivations are developers attempting to satisfy when they read and write source code? Possible aims and motivations include:

- Performing their role in a development project (with an eye on promotion, for the pleasure of doing a good job, or doing a job that pays for other interests),
- Minimizing cognitive effort; for instance, using heuristics rather than acquiring all the necessary information and using deductive logic,
- maximizing the pleasure they get out of the time spent developing...
- belief maintenance: studies have found that people interpret evidence in ways that will maintain their existing beliefs...

2.3 Memory systems

Memory evolved to supply useful, timely information to an organism's decision-making systems.[326] Memory subsystems are scattered about the brain, with each subsystem/location believed to process and store distinct kinds of information. Figure 2.7 illustrates the current model of known long-term memory subsystems and the region of the brain where they are believed to operate.

Declarative memory has two components that process specific instances of information, one handles facts about the world, while the other episodic memory, deals with events (i.e., the capacity to experience an event in the context in which it occurred; it is not known if non-human brains support episodic memory). We are consciously aware of declarative memory, facts and events can be consciously recalled; it is the kind of memory that is referred to in everyday usage as *memory*. Declarative memory is representational, it can be true or false.

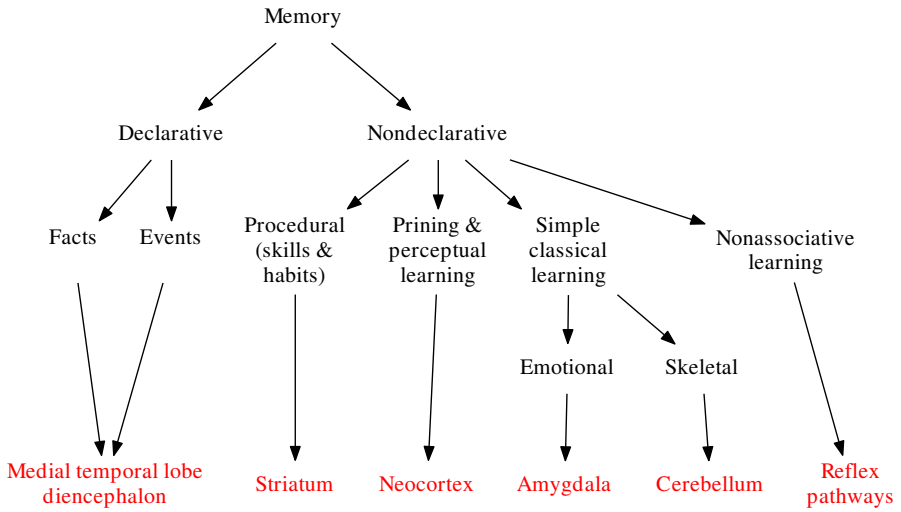


Figure 2.7: Structure of mammalian long-term memory subsystems; brain areas in red. Based on Squire et al.[538]

Nondeclarative memory (also known as *implicit memory*) extracts information from recurring instances of an experience to form skills (e.g., speaking a language) and habits, simple forms of conditioning, priming (response to a stimulus is modified by a preceding stimulus;[479] an advantage in a slowly changing environment where similar things are likely to occur on a regular basis), and perceptual learning (gradual improvement in the detection or discrimination of visual stimuli with practice).

Information in nondeclarative memory is extracted through unconscious performance, e.g., riding a bike; it is an unconscious memory and is not available for conscious recall (information use requires reactivation of the subsystem where the learning originally occurred).

3 3 3 3
a a a a a a
8 8 8
z z
1 1
t t t t
6 6 6 6 6

These subsystems operate independently and in parallel, which creates the possibility of conflicting inputs to higher level systems. For instance, a sentence containing the word blue may be misinterpreted because information about the word and the color in which it appears, green, is returned by different memory subsystems (known as the *Stroop effect*).

A Stroop effect has also been found to occur with lists of numbers. Readers might like to try counting the number of characters occurring in each row in the outside margin. The effort of

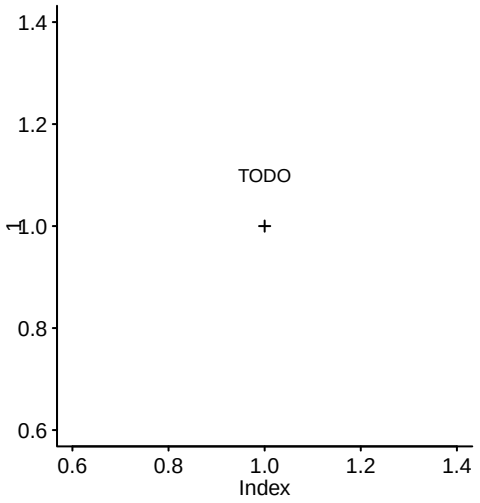


Figure 2.8: Percentage correct answers to questions about binary operator precedence against occurrence in source code. Data from Jones.[300] code

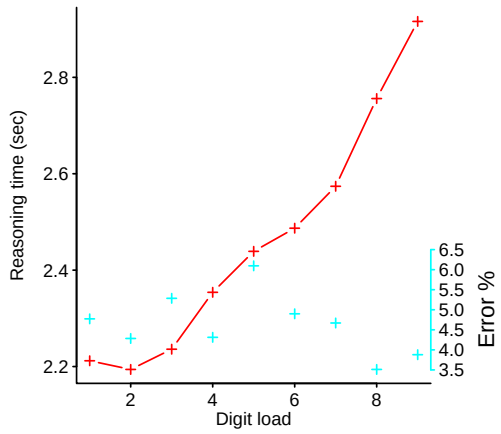


Figure 2.9: Response time (left axis) and error percentage (right axis) on reasoning task with given number of digits held in memory. Data extracted from Baddeley.[34] code

8704
2193
3172
57301
02943
73619
659420
402586
542173
6849173
7931684
3617458
27631508
81042963
07239861

counting the digit sequences is likely to have been greater and more error prone than for the letter sequences.

Studies[439] have found that when subjects are asked to enumerate visually presented digits, the amount of Stroop-like interference depends on the arithmetic difference between the magnitude of the digits used and the number of those digits displayed. Thus, a short, for instance, list of large numbers is read more quickly and with fewer errors than a short list of small numbers. Alternatively a long list of small numbers (much smaller than the length of the list) is read more quickly and with fewer errors than a long list of numbers where the number has a similar magnitude to the length of the list.

Being able to take advantage of any patterns that occur in an environment is a useful skill.

A study by Reber and Kassin[476] asked subjects to memorize sets of words. Some words had been generated using a finite state grammar and the rest had not been generated according to the rules of this grammar. There were two groups of subjects, one group was not told about the existence of a pattern in the letter sequences, the other group was told and it was suggested that deducing this pattern could help them to remember the words. The results showed that performance of the group that had not been told about the presence of a pattern almost exactly mirrored that of the group who had been told, on all sets of words (pattern words only, pattern plus non-pattern words, non-pattern words only).

A study by Jones[300] investigated developer beliefs about binary operator precedence. Subjects were show an expression containing two binary operators and had to specify the relative precedence of these operators by adding parenthesis to the expression, e.g., `a + b | c`. The hypothesis was that answer accuracy would be proportional to the occurrence of the respective operator pairs in source code, i.e., the more often developers have to make a particular precedence decision when reading code, the more likely they are to know the correct answer. Measurements of binary operator usage in a large amount of code was used as a proxy for developer experience of making binary operator decisions. . . . Figure 2.8. . .

There has been some research on the interaction between human memory and software development. For instance, Altmann[13] built a computational process model, based on SOAR, and fitted it to 10.5 minutes of programmer activity (debugging within an <tool>emacs</tool> window); the simulation was used to study the memories, called near-term memory by Altmann, built up while trying to solve a problem.

2.3.1 Short term memory

As its name implies, *short term memory* (STM) is the ability to hold information in memory for short periods of time. Short term memory is the popular term for what cognitive psychologists call *working memory*, named after its function rather than the relative duration it can hold information. Early researchers explored its capacity and a paper by Miller[390] introduced the now-famous 7±2 rule. Things have moved on in the 60 years since the publication of his paper[298] (not that Miller ever proposed 7±2 as the capacity of STM; he simply drew attention to the fact that this range of values fitted the results of several experiments).

Readers might like to try measuring their STM capacity using the list of numbers in the outside margin. Any Chinese-speaking readers can try this exercise twice, using the English and Chinese words for the digits (use of Chinese should enable readers to apparently increase the capacity of STM) in the outside margin. Slowly and steadily read the digits in a row, out loud. At the end of each row, close your eyes and try to repeat the sequence of digits in the same order. If you make a mistake, go on to the next row. The point at which you cannot correctly remember the digits in any two rows of a given length indicates your capacity limit—the number of digits in the previous rows.

The performance impact of reduced working memory capacity can be shown by having people perform two tasks simultaneously. A study by Baddeley[34] measured the time taken to solve a simple reasoning task (e.g., $B \rightarrow A$: 'A follows B' True or False?) while remembering a sequence of digits (the number of digits is known as the *digit load*). Figure 2.9 shows response time (left axis) and percentage of incorrect answers (right axis) for various digit loads. . .

Measuring memory capacity using sequences of digits relies on a variety of assumptions, such as assuming all items consume that same amount of memory resources (e.g., digits and letters are interchangeable), that relative item ordering is implicitly included in the measurement and

that individual concepts are the unit of storage. Subsequent studies resulted in completely different models of STM being created. What the preceding exercise measured was the amount of *sound* that could be held STM. The sound used to represent digits in Chinese is shorter than in English and using Chinese should enable readers to maintain information on more digits (average 9.9[277]) using the same amount of sound storage. A reader using a language for which the sound of the digits is longer would be able to maintain information on fewer digits (e.g., average 5.8 in Welsh[158]). The average for English is 6.6.

The 7±2 capacity limit has been replaced by the limit of two seconds of sound. The two seconds estimate is based on the requirement to remember items and their relative order. The contents of STM do not get erased after two seconds, this limit is the point at which degradation of its contents start to become noticeable.[400] If recall of item order is not relevant, then the limit increases because loss of this information is not relevant.

Studies[422] involving multiple tasks have been used to distinguish the roles played by various components of working memory (e.g., storage, processing, supervision, and coordination). Figure 2.10 shows the components believed to make up working memory, each with its own independent temporary storage areas, each holding and using information in different ways.

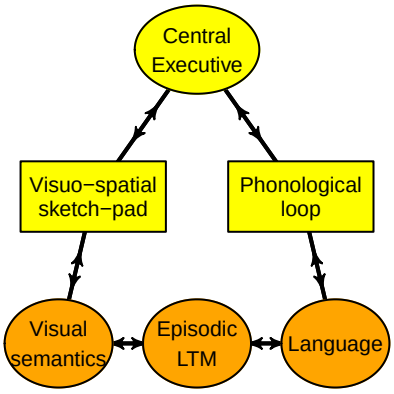


Figure 2.10: Major components of working memory: working memory in yellow, long-term memory in orange. Based on Baddeley.[35] code

The central executive is assumed to be the system that handles attention, controlling the phonological loop, the visuo-spatial sketch pad, and the interface to long-term memory. The central executive needs to remember information while performing tasks such as text comprehension and problem solving. It has been suggested that the focus of attention is capacity-limited, but that the other temporary storage areas are time-limited (without attention to rehearse them, they fade away)[127]...

Visual information held in the visuo-spatial sketch pad decays very rapidly. Experiments have shown that people can recall four or five items immediately after they are presented with visual information, but that this recall rate drops very quickly after a few seconds. From the source code reading point of view, the visuo-spatial sketch pad is only operative for the source code currently being looked at.

Mental arithmetic provides an example of how different components of working memory can be combined to solve a problem that is difficult to achieve using just one component; multiply 23 by 15 without looking at this page. All information has to be held in short term memory and the central executive. Now perform another multiplication, but this time look at the two numbers being multiplied (see outer margin for values) while performing the multiplication. 26
12

Looking at the numbers reduces the load on working memory. Multiplying while being able to look at the numbers being multiplied seems to require less cognitive effort.

Table 2.2 contains lists of words; those at the top of the table contain a single syllable, those at the bottom multiple syllables. Readers should have no problems remembering a sequence of five single-syllable words, a sequence of five multi-syllable words should prove more difficult. As before, read each word slowly out loud.

List 1	List 2	List 3	List 4	List 5
one	cat	card	harm	add
bank	lift	list	bank	mark
sit	able	inch	view	bar
kind	held	act	fact	few
look	mean	what	time	sum
ability	basically	encountered	laboratory	commitment
particular	yesterday	government	acceptable	minority
mathematical	department	financial	university	battery
categorize	satisfied	absolutely	meaningful	opportunity
inadequate	beautiful	together	carefully	accidental

Table 2.2: Words with either one or more than one syllable (and thus varying in the length of time taken to speak).

It has been found that fast talkers have better short-term memory. The connection is the phonological loop. Short-term memory is not limited by the number of items that can be held, but the length of sound that can be stored (about two seconds[36]). Faster talkers can represent more information in that two seconds than those who do not talk as fast.

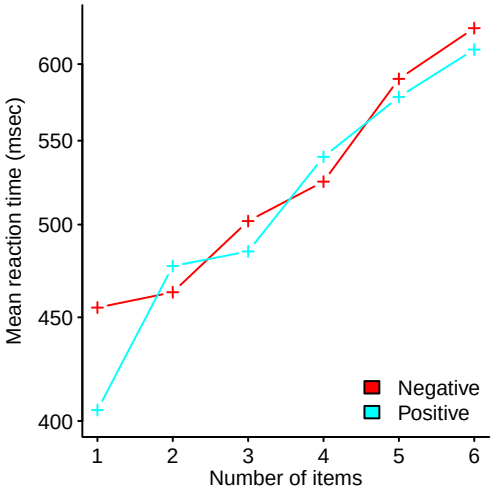


Figure 2.11: Yes/no response time (in milliseconds) as a function of the number of digits held in memory. Data extracted from Sternberg.[551] code

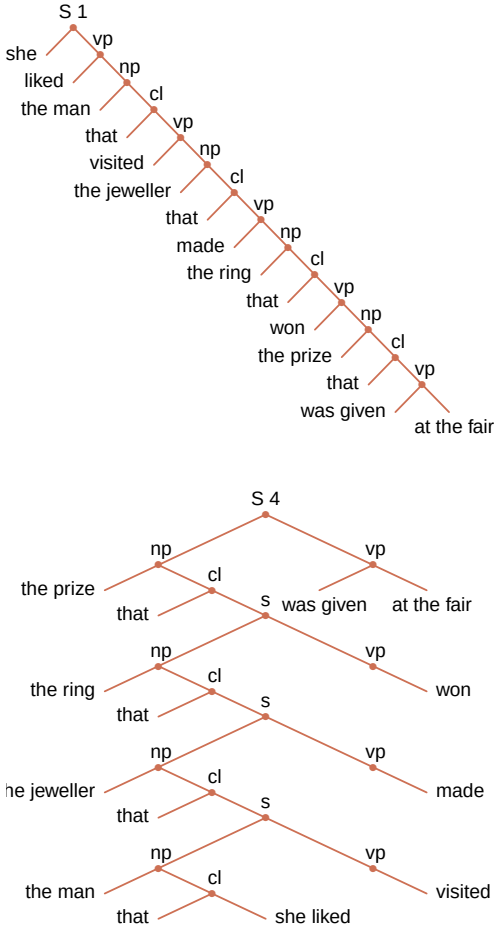


Figure 2.12: Parse tree of a sentence with no embedding, upper "S 1", and a sentence with four degrees of embedding, lower "S 4". Based on Miller et al.[391] code

An analogy between *phonological loop* and a loop of tape in a tape recorder, suggests the possibility that it might only be possible to extract information as it goes past a *read-out point*. A study by Sternberg[551] asked subjects to hold a sequence of digits in memory, e.g., 4185 and measured the time taken to respond yes/no on whether a particular digit was in this sequence. Figure 2.11 shows that as the number of digits increased, the time taken for subjects to respond increases. The other result was that response time was not affected by whether the answer was yes or no. It might be expected that a yes answer would enable searching to terminate, but the behavior found suggests that all digits were always being compared. Different kinds of information has different search response times.[95]

The ability to comprehend syntactically complex sentences is correlated with working memory capacity.[323] A study by Miller and Isard[391] investigated subjects' ability to memorize sentences that varied in their degree of embedding. The following sentences have increasing amounts of embedding (Figure 2.12 shows the parse tree of two of them):

```
She liked the man that visited the jeweller that made the ring that won the prize
that was given at the fair.

The man that she liked visited the jeweller that made the ring that won the prize
that was given at the fair.

The jeweller that the man that she liked visited made the ring that won the prize
that was given at the fair.

The ring that the jeweller that the man that she liked visited made won the prize
that was given at the fair.

The prize that the ring that the jeweller that the man that she liked visited made
won was given at the fair.
```

Subjects' ability to correctly recall wording decreased as the amount of embedding increased, although performance did improve with practice. People have significant comprehension difficulties when the degree of embedding in a sentence exceeds two.[65]

Human language has a grammatical structure that enables it to be parsed serially (e.g., as it is spoken).[451] A consequence of this expected characteristic of sentences is that so called *garden path* sentences, where one or more words at the end of a sentence changes the parsing of words read earlier, generate confusion (requiring conscious effort to reason about what has been said):

```
The old train their dogs.
The patient persuaded the doctor that he was having trouble with to leave.
While Ron was sewing the sock fell on the floor.
Joe put the candy in the jar into my mouth.
The horse raced past the barn fell.
```

2.3.2 Episodic memory

Episodic memory is memory for personally experienced events that are remembered as such, i.e., the ability to recollect specific events or episodes in our lives. When the remembered events occurred sometime ago, the term *autobiographical memory* might be used.

What impact does the passage of time have on episodic memories?

A study by Altmann, Trafton and Hambrick[15] investigated the effects of interruption on a task involving seven steps. Subjects performed the same task 37 times and were interrupted at random intervals during the 30-50 minutes it took to complete the session. Interruptions required subjects to perform a simple typing task that took, on average, 2.8, 13, 22 and 32 seconds. Figure 2.13 shows the percentage of sequencing errors made immediately after an interruption and under normal working conditions (a sequence error occurs when an incorrect step is performed, e.g., step 5 is performed again after performing step 5, when step 6 should have been performed; the `offset` on the x-axis is the difference between the step performed and the one that should have been performed; the sequence error rate, as a percentage of the total number of tasks performed at each interruption interval, was 2.4, 3.6, 4.6 and 5.1%). The lines are predictions made by a model fitted to the data.

2.3.3 Forgetting

People are unhappy when they forget things, however not forgetting may be worse.[417] The Russian mnemonist Shereshevskii found that his ability to remember everything cluttered up his mind.[365] Having many similar, not recently used, pieces of information matching during a memory search would be counterproductive; forgetting is a useful adaptation. For instance, a driver returning to a car wants to know where it was last parked, not the location of all previous parking locations. It has been proposed that human memory is optimized for information retrieval based on the statistical properties of likely need for the information,[21] in people's everyday lives; Burrell investigated the pattern of book borrowings in several libraries; which were also having items added to their stock.[88] The rate at which the mind forgets seems to mirror the way that information tends to lose its utility in the real world over time.

Forgetting, like learning, follows a power law[494] (the results of some studies are also well fitted by an exponential equation). The general relationship between the retention of information, R, and the time, T, since the last access has the form $R = aD^{-b}$ (where a and b are constants). It is known as the *power law of forgetting*.

emailed for data...?

2.3.4 Information recognition and recall

Kinds of information retrieval from memory...

- recognition: studies[330] have found that people can often make a reasonably accurate judgement about whether they know a piece of information or not, even if they are unable to recall that information at a particular instant; the so-called *feeling of knowing* is a good predictor of subsequent recall of information,
- recall: studies[282] have consistently found a variety of patterns in recall of recently lists of remembered items, including better recall performance for items at the start (the *primacy effect*) and end (the *recency effect*) of a list (known as the *serial position effect*,[52] see Figure 2.14), and when prompted by an entry on the list being most likely to recall the item following it[278] rexample[HKJEP99.R].

The environment in which information was learned can have an impact on recall performance. A study by Godden and Baddeley[207] investigated subjects' recall of memorized words in two different environments. Subjects were divers and learned a list of spoken words either while submerged underwater wearing scuba apparatus or while sitting at a table on dry land. The results showed that subjects recall performance was significantly better when performed in the environment in which the word list was learned.

When asked to retrieve members of a category, people tend to produce a list of semantically related items, before switching to list another cluster of semantically related items and so on. This pattern of retrieval is similar to optimal food foraging strategies,[267] however the same behavior would also emerge from a random walk on a semantic network built from human word-association data.[1]

Chunking is a technique commonly used by people to help them remember information. A chunk is a small set of items (4±1 is seen in many studies) having a common, strong, association with each other (and a much weaker one to items in other chunks). For instance, Wickelgren[614] found that people's recall of telephone numbers is optimal if numbers are grouped into chunks of three digits. An example from random-letter sequences is **fbicbsibmirs**. The trigrams (**fbi**, **cbs**, **ibm**, **irs**) within this sequence of 12 letters are well-known acronyms. A person who notices this association can use it to aid recall. Several theoretical analyses of memory organizations have shown that chunking of items improves search efficiency (optimal chunk size 3–4,[150] number items at which chunking becomes more efficient than a single list, 5–7[367]).

A study by Klahr, Chase, and Lovelace[325] investigated how subjects stored letters of the alphabet in memory. Through a series of time-to-respond measurements, where subjects were asked to name the letter that appeared immediately before or after the presented probe letter, they proposed the alphabet-storage structure shown in Figure 2.15.

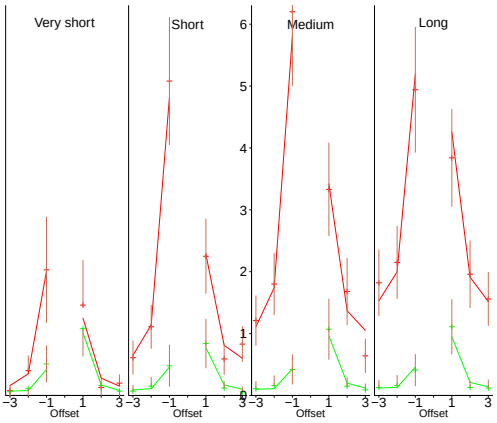


Figure 2.13: Sequencing errors (as percentage) after interruptions of various length (red), including 95% confidence intervals, normal sequence error rate in green; lines are fitted model predictions. Data from Altmann et al code

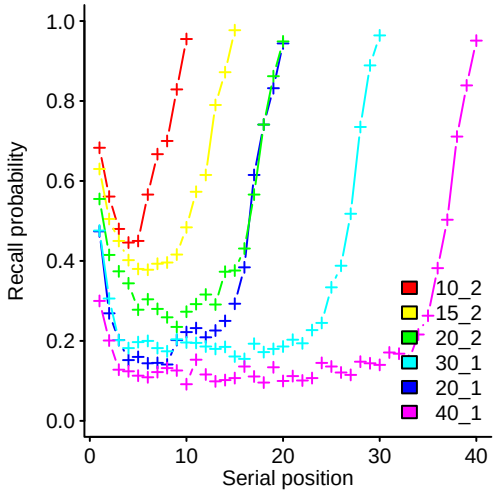


Figure 2.14: Probability of correct recall of words by serial presentation order (each word visible for 1 or 2 seconds, last digit in legend). Data extracted from Murdoch.[52] code

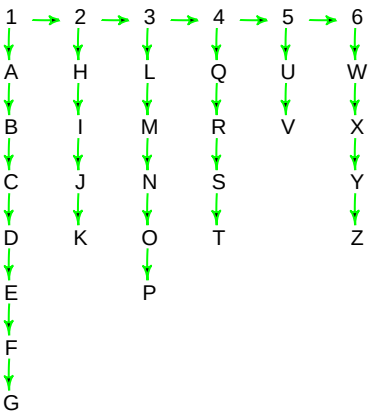


Figure 2.15: Semantic memory representation of alphabetic letters (the numbers listed along the top are place markers and are not stored in subject memory). Readers may recognize the structure of a nursery rhyme in the letter sequences. Derived from Klahr.[325] code

A study by Pennington[447] found that developers responded slightly faster to questions about a source code statement when its immediately preceding statement made use of closely related variables...

2.4 Learning and experience

People have the ability to implicitly and explicitly learn and in human performance on many tasks improves with practice; many studies have fitted a power law to practice performance measurements (the term *power law of learning* is often used). If chunking is assumed to play a part in learning, a power law is a natural consequence;[410] the equation has the form:

$$RT = a + bN^{-c}$$

where RT is the response time; N is the number of times the task has been performed; and a, b, and c are constants.

There are also good theoretical reasons for expecting the measurements to be fitted by an equation having an exponential form and such as equation has been fitted to many data sets;[255] the equation has the form:

$$RT = a + be^{-cN}$$

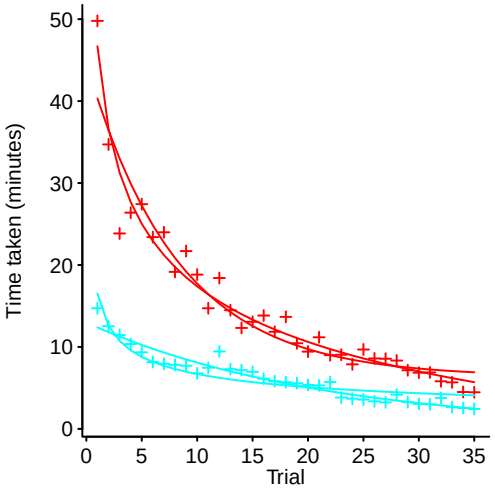


Figure 2.16: Time taken to solve the same jig-saw puzzle 35 times, followed by a two-week interval and then another 35 times, with power law and exponential fits. Data extracted from Alteneder.[12] code

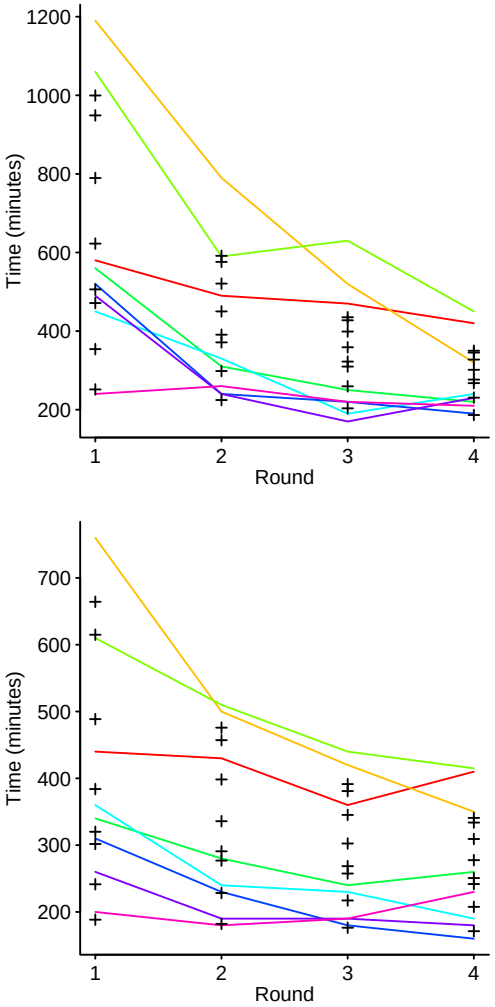


Figure 2.17: Completion times of eight solo (upper) and eight pairs (lower) for each implementation round, along with fitted equation.... Data kindly provided by Lui.[362] code

Implicit learning occurs when people perform a task containing information that is not explicitly obvious to those performing it. A study by Reber and Kassin[476] compared implicit and explicit pattern detection. Subjects were asked to memorize sets of words, with the words in some sets containing letters generated using a finite state grammar. One group of subjects thought they were just taking part in a purely memory-based experiment, while the second group were told of the existence of a letter sequence pattern in some words and that it would help their performance if they could deduce this pattern. The performance of the two groups on the different sets of words (i.e., pattern words only, pattern plus non-pattern words, non-pattern words only) matched each other. Without being told to do so, subjects had used patterns in the words to help perform the memorization task.

Explicit learning occurs when the task contains obvious patterns that can be remembered and used on subsequent performances. A study by Alteneder[12] recorded the time taken by the author to solve the same jig-saw puzzle 35 times (over a four-day period). After two weeks, the same puzzle was again solved 35 times. The exponent of the fitted power law, for the first series, is -0.5; Figure 2.16 show both a fitted power law and exponential.

The source code for an application does not need to be rewritten every time somebody wants a copy; it is unusual for the same developer to be asked to implement exactly the same application again. Having a developer reimplemented the same application many times provides some insight into the underlying

A study by Lui and Chan[362] asked 24 developers to implement the same application four times; 16 developers worked in pairs, i.e., eight pair programming teams, and eight worked solo. Before starting to code the subjects took a test involving 50 questions from a computer aptitude test; subjects were rank by number of correct answers and pairs selected such that both members were adjacent in the ranking.

Learning occurs every-time the application is written. Each subject has existing knowledge and skill, which means everybody starts the experiment at a different point on the learning curve. In the following analysis the test score is used as a proxy for this each subject's initial point on the learning curve.

Figure 2.17 shows the completion times, for each implementation round, for solo and pairs, along with fitted... The following equation is used to fit a regression model to the data:

$$Completion\_time = a \times (b \times Test\_score^3 + Round)^c$$

where: $Completion\_time$ is time to complete an implementation of the application, $Test\_score$ the test score (cubing this value amplifies the difference between subjects) and $Round$ is the number of times the application has been implemented; a, b and c are constants chosen by the model building process.

The improvement in performance obtained through practice starts to decline when a person stops performing a learned activity.[290] Clerical task[519]... emailed for data

A study by Mockus and Weiss[396] found that the probability of developer introducing a fault into an application, when modifying the software, decreased as the log of the total number of changes made by the developer (i.e., their experience or expertise).

Time taken to implement the same algorithm reduces with practice...[631]

binary operator precedence...

Job advertisements often specify that a minimum number of years of experience is required. Number of years is known not to be a measure of expertise, but it provides some degree of comfort that a person has had to deal with many of the problems that might occur within a given domain.

To quote Herbert Simon:[522] 'Intuition and judgement–at least good judgement–are simply analyses frozen into habit and into the capacity for rapid response through recognition. . . . Every manager needs also to be able to respond to situations rapidly, a skill that requires the cultivation of intuition and judgement over many years of experience and training.'

2.4.1 Belief

Knowledge could be defined as belief plus complete conviction and conclusive justification.

- The *foundation approach* argues that beliefs are derived from reasons for these beliefs. A belief is justified if and only if (1) the belief is self-evident and (2) the belief can be derived from the set of other justified beliefs (circularity is not allowed).

  This is very costly. in cognitive effort, to operate, e.g., the reasons for beliefs need to be remembered and applied when considering new beliefs. Studies[490] show that people exhibit a belief preservation effect; they continue to hold beliefs after the original basis for those beliefs no longer holds. The evidence suggests that people use some form of *coherence approach* for creating and maintaining their beliefs.

- the *coherence approach* argues that where beliefs originated is of no concern. Instead, beliefs must be logically coherent with other beliefs (believed by an individual). These beliefs can mutually justify each other and circularity is allowed. A number of different types of coherence have been proposed, Including *deductive coherence* (requires a logically consistent set of beliefs), *probabilistic coherence* (assigns probabilities to beliefs and applies the requirements of mathematical probability to them), *semantic coherence* (based on beliefs that have similar meanings), and *explanatory coherence* (requires that there be a consistent explanatory relationship between beliefs).

The Belief-Adjustment model A belief may be based on a single piece of evidence, or it may be based on many pieces of evidence. How is an existing belief modified by the introduction of new evidence? The belief-adjustment model of Hogarth and Einhorn[274] offers an answer to this question; the basic equation for their model is:

$$S_k = S_{k-1} + w_k[s(x_k) - R]$$

where: $0 \leq S_k \leq 1$ is the degree of belief in some hypothesis or impression after evaluating k items of evidence; S_{k-1} is the anchor, or prior opinion (with $S_0 i$ denoting the initial belief); $s(x_k)$ is a subjective evaluation of the k'th item of evidence (different people may assign different values for the same evidence, x_k); R is the reference point, or background, against which the impact of the k'th item of evidence is evaluated; $0 \leq w_k \leq 1$ is the adjustment weight for the k^{th} item.

When presented with new information people can process the evidence it contains in several ways, including:

- using an *evaluation* process, which encodes new evidence relative to a fixed point—the hypothesis addressed by a belief. If the new evidence supports the hypothesis, a person's belief is increased, and if it is not supported by the hypothesis the belief is decreased. The increase/decrease occurs irrespective of the current state of a person's belief; for this case: $-1 \leq s(x_k) \leq 1$ and $R = 0$, and the belief-adjustment equation simplifies to:

  $$S_k = S_{k-1} + w_k s(x_k)$$

  An example of an evaluation process might be the belief that the object `X` always holds a value that is numerically greater than `Y`.

- using an *estimation* process, which encodes new evidence relative to the current state of a person's beliefs. For this case $R = S_{k-1}$, and the belief-adjustment equation simplifies to:

  $S_k = S_{k-1} + w_k[s(x_k) - S_{k-1}]$

  where: $0 \leq s(x_k) \leq 1$

  In this case the degree of belief, in a hypothesis, can be thought of as a moving average. For an estimation process, the order in which evidence is presented can be significant. While reading source code written by somebody else, a developer will form an opinion of the quality of that person's work. The judgement of each code sequence will be based on the readers current opinion (at the time of reading) of the person who wrote it.

The $s(x_k)$ could represent either the impact of a single piece of evidence (known as *Step-by-Step*, SbS), or several pieces of evidence that have been combined to have a single impact (known as *End-of-Sequence*, EoS).

Hogarth and Einhorn proposed that when people are required to provide an EoS response they use an EoS process when the sequence of items is short and simple. As the sequence gets longer, or more complex, they shift to an SbS process, to keep the peak cognitive load (of processing the evidence) within their capabilities.

Order effects Use of an SbS process when $R = S_{k-1}$ leads to a recency effect.[274] When $R = 0$, a recency effect only occurs when there is a mixture of positive and negative evidence (there is no recency effect if the evidence is all positive or all negative).

The use of an EoS process leads to a primacy effect; however, a task may not require a response until all the evidence is seen. If the evidence is complex, or there is a lot of it, people may adopt an SbS process. In this case, the effect seen will match that of an SbS process.

A study by Hogarth and Einhorn[274] investigated order, and response mode, effects in belief updating. Subjects were presented with a variety of scenarios (e.g., a defective stereo speaker thought to have a bad connection or a baseball player whose hitting improved dramatically after a new coaching program). Subjects read an initial description followed by two or more additional items of evidence. The additional evidence was either positive (e.g., 'The other players on Sandy's team did not show an unusual increase in their batting average over the last five weeks') or negative (e.g., 'The games in which Sandy showed his improvement were played against the last-place team in the league'). The positive and negative evidence was worded to create either strong or weak forms.

The evidence was presented in three combinations: strong-positive and weak-positive, upper plot in Figure 2.18; strong-negative and weak-negative, middle plot of Figure 2.18; positive-negative and negative-positive, lower plot of Figure 2.18. Subjects were asked, 'Now, how likely do you think X caused Y on a scale of 0 to 100?' In some cases, subjects had to respond after seeing each item of evidence: in other cases, subjects had to respond after seeing all the items.

Other studies have duplicated these results, for instance, professional auditors have been shown to display recency effects in their evaluation of the veracity of company accounts.[437]

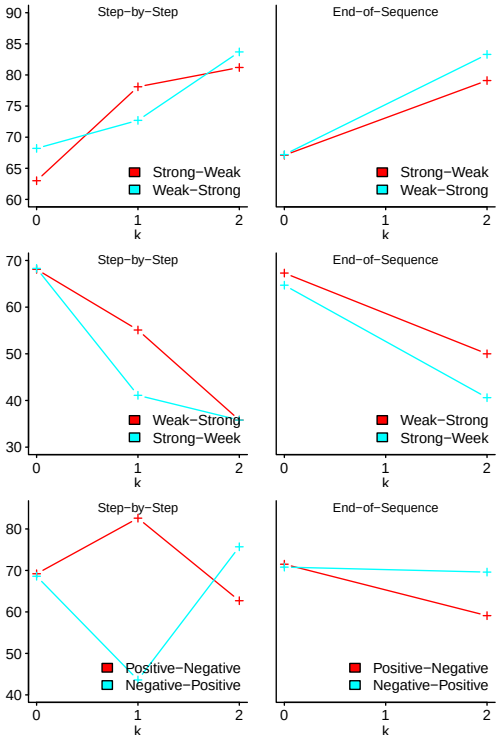


Figure 2.18: Subjects belief response curves for positive weak–strong, negative weak–strong, and positive–negative evidence. Based on Hogarth et al.[274] code

2.4.2 Category knowledge

Children as young as four have been found to use categorization to direct the inferences they make,[198] and many different studies have shown that people have an innate desire to create and use categories (they have also been found to be sensitive to the costs and benefits of using categories[369]). By dividing items they encounter into categories, people reduce the amount of information they need to learn[458] and to generalize based on prior experience.[414] Probably information about the characteristics of a newly encountered item is obtained by matching it to one or more known categories and then extracting characteristics common to previously encountered items in those categories. For instance, a flying object with feathers and a beak might be assigned to the category *bird*, which suggests the characteristics of laying eggs and being migratory.

Categorization is used to perform inductive reasoning (the derivation of generalized knowledge from specific instances), and also acts as a memory aid (remembering the members of a category). Categories provide a framework from which small amounts of information can be used to infer, seemingly unconnected (to an outsider), useful conclusions.

Several studies have shown that people use around three levels of abstraction in creating hierarchical relationships. Rosch[488] called the highest level of abstraction the *superordinate-level*—for instance, the general category furniture. The next level down is the *basic-level*; this is the level at which most categorization is carried out—for instance, car, truck, chair, or table. The lowest level is the *subordinate-level*, denoting specific types of objects. For instance, a family car, a removal truck, my favourite armchair, a kitchen table. Rosch found that the basic-level categories had properties not shared by the other two categories; adults spontaneously name objects at this level, it is the abstract level that children acquire first, and category members tend to have similar overall shapes.

When categories have hierarchical structure it is possible for an attribute of a higher-level category to affect the perceived attributes of subordinate categories. This is illustrated in a study by Stevens and Coupe[552] in which subjects were asked to remember the information contained in a series of maps (see Figure 2.19). They were asked questions such as: 'Is X east or west of Y?' and 'Is X north or south of Y?' Subjects gave incorrect answers 18% of the time for the congruent maps, but 45% of the time for the incongruent maps (15% for homogeneous). These results were interpreted as information about the relative locations of countries influencing the answer questions about the city locations.

Studies[530] have found that people do not consistently treat subordinate categories as inheriting the properties of their superordinates, i.e., category inheritance need not be a tree.

How categories should be defined and structured is a long-standing debate within the sciences. Some commonly used category formation techniques, their membership rules and attributes include:

- in the defining-attribute theory members of a category are characterized by a set of defining attributes. Attributes divide objects up into different concepts whose boundaries are well-defined and all members of the concept are equally representative. Also, concepts that are a basic-level of a superordinate-level concept will have all the attributes of that superordinate level; for instance, a sparrow (small, brown) and its superordinate bird (two legs, feathered, lays eggs),
- in the prototype theory categories have a central description, the prototype, that represents the set of attributes of the category. This set of attributes need not be necessary, or sufficient, to determine category membership. The members of a category can be arranged in a typicality gradient, representing the degree to which they represent a typical member of that category. It is also possible for objects to be members of more than one category (e.g., tomatoes as a fruit, or a vegetable),
- in the exemplar-based theory of classification specific instances, or *exemplars*, act as the prototypes against which other members are compared. Objects are grouped, relative to one another, based on some similarity metric. The exemplar-based theory differs from the prototype theory in that specific instances are the norm against which membership is decided. When asked to name particular members of a category, the attributes of the exemplars are used as cues to retrieve other objects having similar attributes,
- in the explanation-based theory of classification there is an explanation for why categories have the members they do. For instance, the biblical classification of food into *clean* and *unclean* is roughly explained by saying that there should be a correlation between type of habitat, biological structure, and form of locomotion; creatures of the sea should have fins, scales, and swim (sharks and eels don't) and creatures of the land should have four legs (ostriches don't).

  From a predictive point of view, explanation-based categories suffer from the problem that they may heavily depend on the knowledge and beliefs of the person who formed the category; for instance, the set of objects a person would remove from their home if it suddenly caught fire.

Figure 2.20 shows the possible combinations of three, two-valued attributes, color/size/shape; there are eight possibilities. It is possible to create six unique categories by selecting four items from these eight possibilities (see Figure 2.21; there are 70 different ways of taking four things from a choice of eight, $(8!/(4!4!))$, and taking symmetry into account reduces the number to unique categories to six).

A study by Shepard, Hovland, and Jenkins[515] measured subject performance in assigning objects to these six categories. Subject error rate decreased with practice.

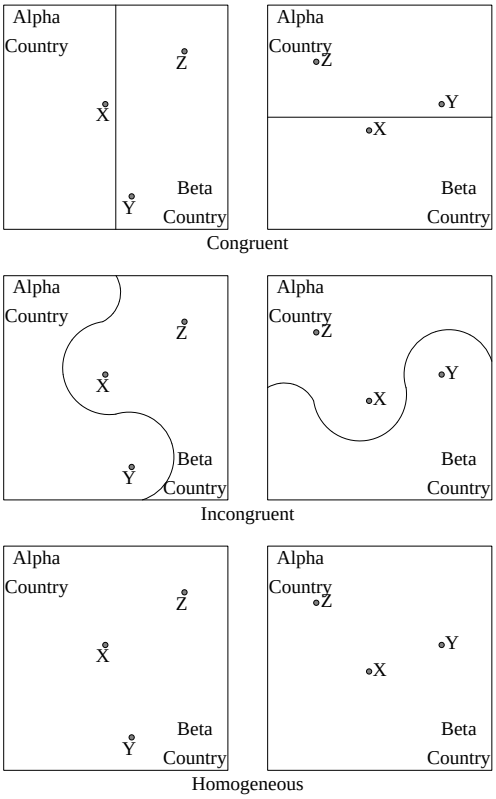


Figure 2.19: Country boundaries distort judgement of relative city locations. Based on Stevens et al.[552]

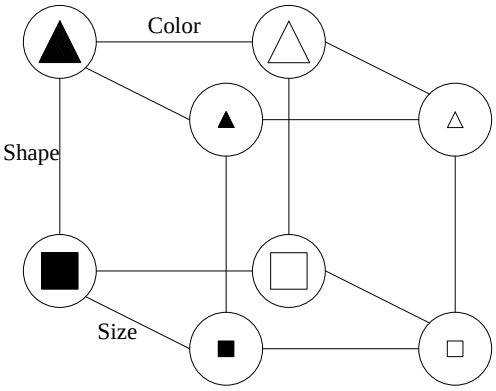


Figure 2.20: Orthogonal representation of shape, color and size stimuli. Based on Shepard.[515]

A method of calculating the similarity of two objects, proposed by Estes,[167] is in general agreement with the Shepard et al results. A matching similarity coefficient, $0 \leq t \leq \infty$ is assigned for matching attributes and a nonmatching similarity coefficient, $0 \leq s_i \leq 1$ (potentially different for each nonmatch), is assigned for each nonmatching attribute. The similarity of two objects is calculated by multiplying the similarity coefficients. When comparing objects within the same category the convention is to use, $t = 1$ and to give the attributes that differ the same similarity coefficient, s.

As an example, for simplicity consider the two attributes shape/color in Figure 2.20, giving the four combinations black/white—triangles/squares; assign black triangle and black square to category A, and the white triangle and white square to category B, i.e., category membership is decided by color. The similarity of each of the four possible object combinations to category A and B is listed in Table 2.3. Looking at the top row: Black triangle is compared for similarity to all members of category A ($1+s$, because it does not differ from itself and differs in one attribute from the other member of category A) and all members of category B($s+s^2$, because it differs in one attribute from one member of category B and in two attributes from the other member).

Stimulus	Similarity to A	Similarity to B
Black triangle	$1+s$	$s+s^2$
Black square	$1+s$	$s+s^2$
White triangle	$s+s^2$	$1+s$
White square	$s+s^2$	$1+s$

Table 2.3: Similarity of a Stimulus object to two categories (category A: black triangle and black square; category B: white triangle and white square).

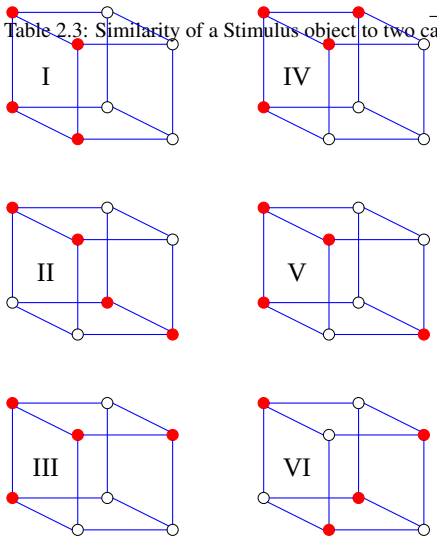


Figure 2.21: The six unique configurations of selecting four times from eight possibilities, i.e., it is not possible to rotate one configuration into another within these six configurations. Based on Shepard.[515]

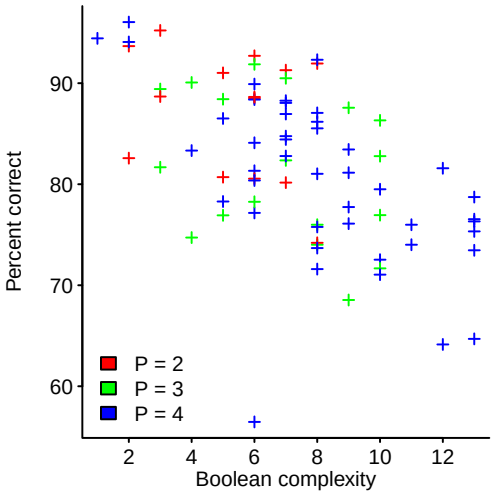


Figure 2.22: Percentage of correct answers given by one subject, against boolean-complexity of category, colored by number of positive cases needed to define the category. Data kindly provided by Feldman.[176] code

If a subject is shown a stimulus that belongs in category A, the expected probability of them assigning it to this category is:

$$\frac{1+s}{(1+s)+(s+s^2)} \rightarrow \frac{1}{1+s}$$

When s is 1, the expected probability is no better than a random choice; when s is 0, the probability is a certainty.

A study by Feldman[176] involved categories containing objects having either three or four attributes. Categories containing objects having various combinations of attributes were created and used to measure subjects' classification accuracy.

Feldman[177] specified category membership algebraically, e.g., membership of category IV in the top right of Figure 2.21 is specified by the expression: $S\overline{HC}+S\overline{HC}+\overline{S}\overline{HC}+\overline{S}\overline{HC}$, where: S is size, H is shape, C is color and an $\overline{overline}$ indicates negation. The number of terms in the minimal boolean formula specifying the category (a measure of category complexity which Feldman terms *boolean complexity*) was found to predict the trend in subject error rate (i.e., number of errors in selecting the correct category increased with boolean complexity, see Figure 2.22).

2.4.2.1 Categorization consistency

Obtaining the benefits of category usage requires some degree of usage consistency. In the case of an individual's personal categories, a person has to be able to assign the appropriate items to the correct category; within groups there has to be some level of agreement between people using the same category.

Cross-language research has found that there are very few concepts that might be claimed to be universal (they mostly relate to the human condition).[597,615]

Human languages encode views of the world. The extent to which the language used by a person influences their thought processes continues to be hotly debated.[199] The proposal that language does influence thought is commonly known as the *Sapir-Whorf* or *Whorfian* hypothesis. Some people hold what is known as the *strong language-based* view, believing that the language used does influence its speakers' conceptualization process, while people holding the so-called *weak language-based* view believe that linguistic influences occur in some cases (e.g., English contains count nouns and speakers have to pay attention to whether one or more than one item is being referred to; languages without contain count nouns do

not require speakers to pay attention to this detail, but perhaps require them to pay attention to the shape of the object being discussed, e.g., Japanese). The *language-as-strategy* view holds that language affects speakers performance by constraining what can be said succinctly with the set of available words (a speed/accuracy trade-off, approximating what needs to be communicated in a brief sentence rather than using a longer sentence to be more accurate).

While different languages may contain different ways of describing the world, common usage patterns can be found. A study by Berlin and Kay[55] isolated what they called the *basic color terms* of 98 languages. They found that the number and kind of basic color terms in languages followed a consistent pattern (see Figure 2.23); while the boundaries between color terms varied, the visual appearance of the basic color terms was very similar across languages. Simulations of the evolution of color terms[43] suggest that it takes time for the users of a language to reach consensus on the naming of colors and over time languages accumulate more color terms.

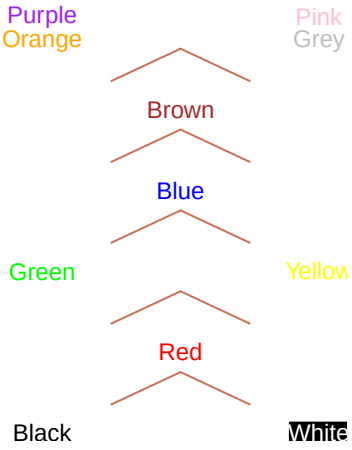


Figure 2.23: The Berlin and Kay[55] language color hierarchy. The presence of any color term in a language implies the existence, in that language, of all terms below it. Papuan Dani has two terms (black and white), while Russian has eleven (Russian may also be an exception in that it has two terms for blue.) code

Context plays an important role in the creation and use of categories.

A study by Bailenson, Shum, Atran, Medin and Coley[37] compared the categories created for two sets (US and Maya) of 104 bird species by three groups, asked US bird experts (average of 22.4 years bird watching), US undergraduates, and ordinary Itzaj (Maya Amerindians people from Guatemala). The categorization choices made by the three groups of subjects were found to be internally consistent within each group. The US experts correlated highly with the scientific taxonomy for both sets of birds, the Itzaj only correlated highly for Maya birds, and the nonexperts had a low correlation for either set of birds. The reasons given for the Maya choices varied between the expert groups; US experts were based on a scientific taxonomy, Itzaj were based on ecological justifications (the bird's relationship with its environment). Cultural differences were found in that, for instance, US subjects were more likely to generalise from songbirds, while the Itzaj were more likely to generalize from perceptually striking birds.

Context can play an important role in classification. A study by Labov[340] showed subjects pictures of items that could be classified as either cups or bowls (see upper plot in Figure 2.24). These items were presented in one of two contexts—a neutral context in which the pictures were simply presented and a food context (they were asked to think of the items as being filled with mashed potatoes).

Figure 2.24, lower plot, shows that as the width of the item seen was increased, an increasing number of subjects classified it as a bowl. By introducing a food context subjects responses shifted towards classifying the item as a bowl at narrower widths.

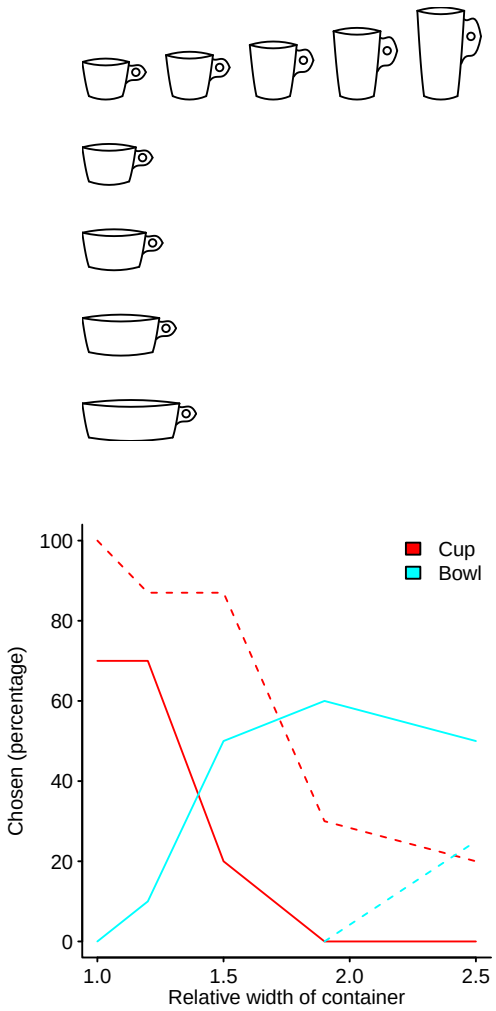


Figure 2.24: Cup- and bowl-like objects of various widths (ratios 1.2, 1.5, 1.9, and 2.5) and heights (ratios 1.2, 1.5, 1.9, and 2.4). The percentage of subjects who selected the term *cup* or *bowl* to describe the object they were shown (the paper did not explain why the figures do not sum to 100%). Based on Labov.[340] code

The same situation can often be viewed from a variety of different points of view (the term *frame* is sometimes used); for instance, commercial events include buying, selling, paying, charging, pricing, costing, spending, and so on. Figure 2.25 shows four ways (i.e., buying, selling, paying, and charging) of looking at the same commercial event.

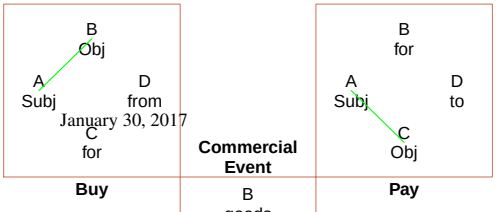
A study by Jones[302] investigated the extent to which different developers make similar decisions when creating data structures to represent the same information... organization of struct members, what information closely associated in the data structs...

2.4.3 Expertise

The term *expert* might be applied to a person because of their professional standing[iii] or because of what they can do (i.e., they know a great deal about a particular subject, or can perform at a qualitatively higher level than a novice in a specific domain, or some combination of the two).[206]

There are domains where professional experts do not perform significantly better than non-experts. For instance, in typical cases the performance of medical experts was not much greater than those of doctors after their first year of residency, although much larger differences were seen for difficult cases,[?] and in some cases expertise can constrain the search for solutions.[617]

This section discusses expertise as a high-performance skill; something that requires many years of training and substantial numbers of individuals fail to develop proficiency.



[iii] Industrial countries use professionalism as a way of institutionalising expertise.

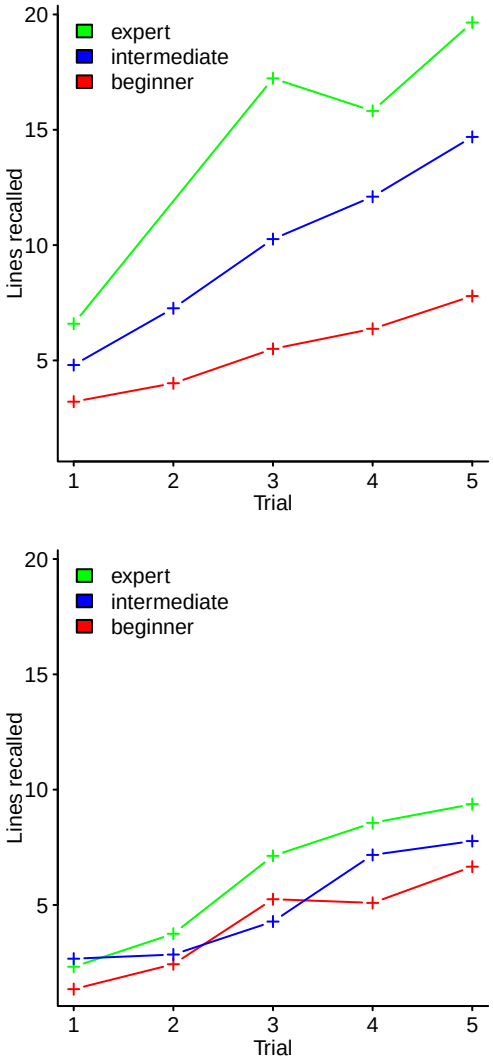


Figure 2.26: Lines of code correctly recalled after a given number of 2 minute memorization sessions; upper plot actual program, lower plot line order scrambled. Data extracted from McKeithen et al.[381] code

In many fields expertise is acquired by memorizing a huge amount of domain-specific information, organizing it for rapid retrieval based on patterns that occur when problem solving within the domain and refining the problem solving process.[165]

Chess players were the subject of the first major study of expertise by de Groot,[134] and techniques used to study Chess along with the results obtained continues to dominate the study of expertise. In a classic study, de Groot briefly showed subjects the position of an unknown game and asked them to reconstruct it. The accuracy and speed of experts (e.g., Grand Masters) was significantly greater than non-experts when the pieces appeared on the board in positions corresponding to a game, but was not much greater when the pieces were placed at random. An explanation of the significant performance difference is that experts are faster to recognise relationships between the positions of pieces and make use of their large knowledge of positional patterns to reduce the amount of working memory needed to remember what they were briefly shown.

A study by McKeithen, Reitman, Ruster and Hirtle[381] gave subjects two minutes to study the source code of a program and then gave them three minutes to recall the 31 lines of the program. They were then given another two minutes to study the same source code and asked to recall the code; this process was repeated for a total of five trials.

Figure 2.26 shows the number of lines recalled by experts (over 2,000 hours of general programming experience), intermediates (just completed a programming course) and beginners (about to start a programming course) over the five trials. The upper plot are the results for the 31 line program and the lower plot a scrambled version of the program.

There is a belief that experts have some innate ability or capacity that enables them to do what they do so well. Research over the last two decades has shown that while innate ability can be a factor in performance (there do appear to be genetic factors associated with some athletic performances), the main factor in acquiring expert performance is time spent in *deliberate practice*;[163] deliberate practice does not explain everything[245]...

Deliberate practice is different from simply performing the task. It requires that people monitor their practice with full concentration and obtain feedback[275] on what they are doing (often from a professional teacher). It may also involve studying components of the skill in isolation, attempting to improve on particular aspects. The goal of this practice being to improve performance, not to produce a finished product.

Studies of the backgrounds of recognized experts, in many fields, found that the elapsed time between them starting out and carrying out their best work was at least 10 years, often with several hours of deliberate practice every day of the year. For instance, a study of violinists[164] (a perceptual-motor task), found that by age 20 those at the top-level had practiced for 10,000 hours, those at the next level down 7,500 hours, and those at the lowest level of expertise had practiced for 5,000 hours; similar quantities of practice were found in those attaining expert performance levels in purely mental activities (e.g., chess).

People often learn a skill for some purpose (e.g., chess as a social activity, programming to get a job) without the aim of achieving expert performance. Once a certain level of proficiency is achieved, they stop trying to learn and concentrate on using what they have learned (in work, and sport, a distinction is made between training for and performing the activity). During everyday work, the goal is to produce a product or to provide a service. In these situations people need to use well-established methods, not try new (potentially dead-end, or leading to failure) ideas to be certain of success. Time spent on this kind of practice does not lead to any significant improvement in expertise, although people may become very fluent in performing their particular subset of skills.

Expertise within one domain does not confer any additional skills within another domain,[?] e.g., statistics (unless the problem explicitly involves statistical thinking within the applicable domain) and logic[106] and subjects who learned to remember long sequences of digits (after 50–100 hours of practice they could commit to memory, and recall later, sequences containing more than 20 digits) did not transfer their expertise to learning sequences of other items.[103]

What of individual aptitudes? In the cases studied the effects of aptitude, if there are any, have been found to be completely overshadowed by differences in experience and deliberate practice times. Willingness to spend many hours, every day, studying to achieve expert performance is certainly a necessary requirement. Does an initial aptitude or interest in a subject lead to praise from others (the path to musical and chess expert performance often starts in childhood), which creates the atmosphere for learning, or are other issues involved? IQ scores do correlate to performance during and immediately after training, but the correlation

reduces over the years. The IQ scores of experts has been found to be higher than the average population, at about the level of college students.

Education can be thought of as trying to do two things (of interest to us here)—teach students skills (procedural knowledge) and providing them with information, considered important in the relevant field, to memorize (declarative knowledge).

Does attending courses in particular subjects have any measurable effect on students' capabilities, other than being able to answer questions in an exam? That is, having acquired some skill in using a particular system of reasoning, do students apply it outside of the domain in which they learnt it?

A study by Lehman, Lempert, and Nisbett[353] measured changes in students' statistical, methodological, and conditional reasoning abilities (about everyday-life events) between their first and third years. They found that both psychology and medical training produced large effects on statistical and methodological reasoning, while psychology, medical, and law training produced effects on the ability to perform conditional reasoning; training in chemistry had no effect on the types of reasoning studied. An examination of the skills taught to students studying in these fields showed that they correlated with improvements in the specific types of reasoning abilities measured.

2.5 Visual processing

An understanding of human visual processing is of interest to software development because it consumes cognitive resources and is a source of information input error. The 2-D image that falls on the retina does not contain enough information to build the 3-D model we *see*, the mind creates this model by making assumptions about how objects in our environment move and interact.[273]

The perceptual systems of an organism have evolved to detect information in the environment that is relevant to its survival and ignore the rest. The relevant information is about opportunities *afforded* by the world...

Some inputs to the visual system appear to *pop-out* from their surroundings. Preattentive processing, so called because it occurs before conscious attention,[467] is automatic and apparently effortless. Figure 2.27 shows some examples of features that *pop-out* at the reader.

Preattentive processing is independent of the number of distractors; a search for the feature takes the same amount of time whether it occurs with one, five, ten, or more other distractors. However, it is only effective when the features being searched for are relatively rare. When a display contains many different, distinct features (the mixed category in Figure 2.27), the *pop-out* effect does not occur.

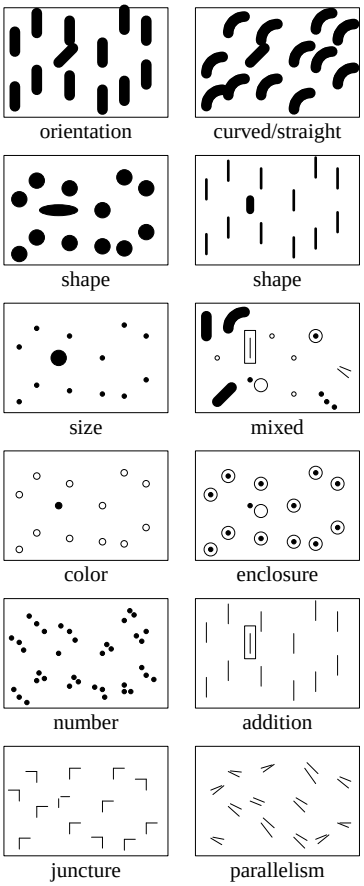
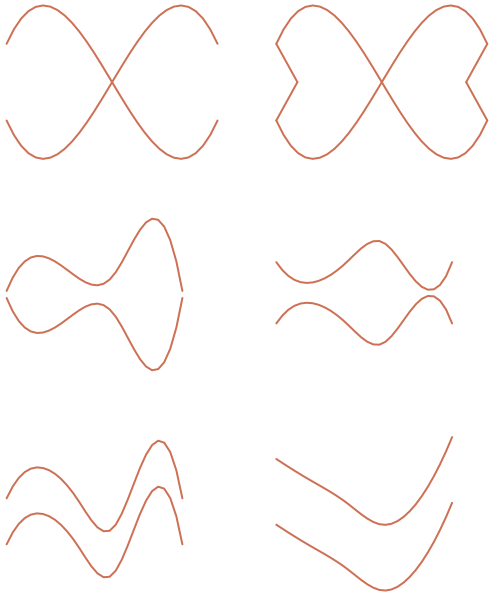


Figure 2.27: Examples of features that may be preattentively processed (parallel lines and the junction of two lines are the odd ones out). Based on Ware.[603]

The *Gestalt laws of perception* ('gestalt' means 'pattern' in German, also known as the *laws of perceptual organization*)[598] are based on the underlying idea that the whole is different from the sum of its parts. These so-called *laws* do not have the rigour expected of a scientific law, and really ought to be called by some other term (e.g., principle). The Gestalt principles are preprogrammed (i.e., there is no conscious cognitive cost). The following are some commonly occurring principles

- Continuity, also known as Good continuation: Lines and edges that can be seen as smooth and continuous are perceptually grouped together, see upper plot in Figure 2.28,
- Closure: elements that form a closed figure are perceptually grouped together, see upper plot in Figure 2.28,
- Symmetry: treating two, mirror image lines as though they form the outline of an object, see second down plot in Figure 2.28. This effect can also occur for parallel lines.
- Proximity: elements that are close together are perceptually grouped together, see second from bottom plot in Figure 2.28,
- Similarity: elements that share a common attribute can be perceptually grouped together, see lower plot Figure 2.28.
- Other: principles include grouping by connectedness, grouping by common region, and synchrony.[433]



The organization of visual grouping of elements in a display, using these principles, is a common human trait. However, different people can make different choices in the perceptually grouping of the same collection of items. Figure 2.29 shows items on a horizontal line, which readers may group by shape, color or relative proximity. A study by Kubovy and van den Berg[335] created a model that calculated the probability of a particular perceptual grouping (i.e., shape, color or proximity in two dimensions) being selected for a given set of items.

When mapping the prose specification of a mathematical relationship to a formula, the error rate has been found to be affected by the visual proximity of the applicable words.[343]

A number of studies have found that people are more likely to notice the presence of a distinguishing feature than the absence of a distinguishing feature. This characteristic affects performance when searching for an item when it occurs among visually similar items. It can also affect reading performance—for instance, substituting an e for a c is more likely to be noticed than substituting a c for an e.

A study by Treisman and Souther[571] found that visual searches were performed in parallel when the target included a unique feature (i.e., search time was not affected by the number of background items), but were performed serially when the target had a unique feature missing (i.e., search time was proportional to the number of background items).

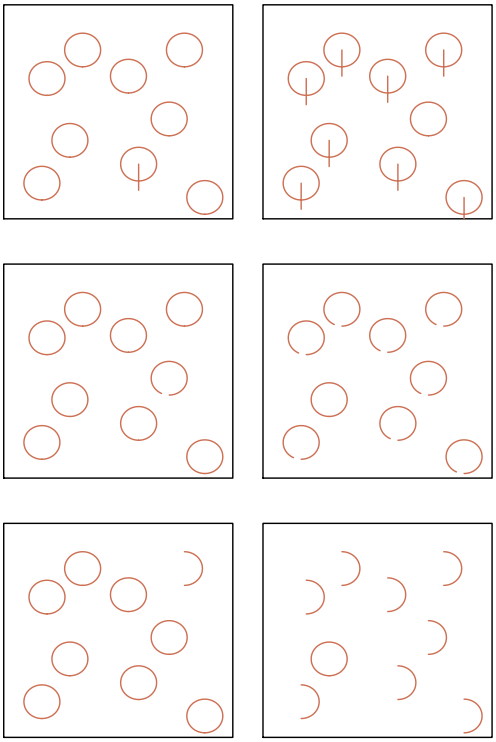


Figure 2.30: Examples of unique items among visually similar items. Those at the top include an item that has a distinguishing feature (a vertical line or a gap); those underneath them include an item that is missing this distinguishing feature. Based on displays used by Treisman et al.[571] code

What is a unique feature? Subjects searched for circles that differed in the presence or absence of a gap (see Figure 2.30). The results showed that subjects were able to locate a circle containing a gap, in the presence of complete circles, in parallel. However, searching for a complete circle, in the presence of circles with gaps, was performed serially. In this case the gap was the unique feature. Performance also depended on the proportion of the circle taken up by the gap...

The visual aspect of creating a software system involves the reading code and text. Research on the cognitive processes involved in reading prose written in human languages has uncovered the basic processes and various models have been built.[473]

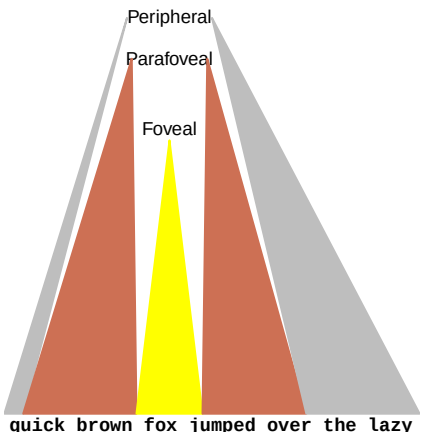
During reading, a person's eyes make short rapid movements, known as *saccades*, taking 20 ms to 50 ms to complete; a saccade typically moves the eyes forward 6 to 9 characters. No visual information is extracted during a saccade and readers are not consciously aware of them. Between saccades the eyes are stationary, typically for 200 ms to 250 ms, these stationary periods are known as *fixations*; a study of consumer eye movements[452] while comparing multiple brands found a fixation duration of 354 ms when subjects were under high time pressure and 431 ms when under low time pressure.

Individual readers can exhibit considerable variations in performance, a saccade might move the eyes by one character, or 15 to 20 characters; fixations can be shorter than 100 ms or longer than 400 ms (there is also variation between languages[428]). The content of the fixated text has a strong effect on performance.

The eyes do not always move forward during reading—10% to 15% of saccades move the eyes back to previous parts of the text. These backward movements, called *regressions*, are caused by problems with linguistic processing (e.g., incorrect syntactic analysis of a sentence) and oculomotor error (for instance, the eyes overshooting their intended target).

Saccades are necessary because the eyes' field of view is limited. Light entering an eye hits light-sensitive cells in the retina, where cells are not uniformly distributed. The visual field (on the retina) can be divided into three regions: foveal (the central 2°, measured from the front of the eye looking toward the retina), parafoveal (extending out to 5°), and peripheral (everything else). Letters become increasingly difficult to identify as their angular distance from the center of the fovea increases.

Two processes during the fixation period: identifying the word (or sequence of letters forming a partial word) and planning the next saccade, when to make it and where to move the eyes. Reading performance is speed limited by the need to plan and perform saccades (removing the need to saccade by presenting words at the same place on a display, there is a threefold speed increase in reading aloud and a two-fold speed increase in silent reading). The time needed to plan and perform a saccade is approximately 180 ms to 200 ms (known as the *saccade latency*), which means that the decision to make a saccade occurs within the first 100 ms of a fixation.



The contents of the parafoveal region are partially processed during reading and this increases a reader's perceptual span. When reading words written using alphabetic characters, the perceptual span extends from 3 to 4 characters on the left of fixation to 14 to 15 letters to the right of fixation. This asymmetry in the perceptual span is a result of the direction of reading, attending to letters likely to occur next being of greater value. Readers of Hebrew (which is read right-to-left) have a perceptual span that has opposite asymmetry (in bilingual Hebrew/English readers the direction of the asymmetry depends on the language being read, showing the importance of attention during reading).[472]

Characteristics used by the writing system affect the asymmetry of the perceptual span and its width, e.g., the span can be smaller for Hebrew than English (Hebrew words can be written without the vowels, requiring greater effort to decode and plan the next saccade). It is also much smaller for writing systems that use ideographs, such as Japanese (approximately 6 characters to the right) and Chinese.

The perceptual span is not hardwired, but is attention-based. The span can become smaller when the fixated words are difficult to process. Also, readers obtain more information in the direction of reading when the upcoming word is highly predictable (based on the preceding text).

Models of reading have achieved some level of success include: Mr. Chips:[352] an ideal-observer model of reading (it is not intended to model of how humans read, but to establish the pattern of performance when optimal use is made of available information) which attempts to calculate the distance, in characters, of the next saccade; it combines information from visual data obtained by sampling the text through a retina, lexical knowledge of words and their relative frequencies, and motor knowledge of the statistical accuracy of saccades and uses the optimization principle of entropy minimization. SWIFT:[160] attempts to provide a realistic model of saccade generation, E-Z Reader:[455] attempts to account for how cognitive and lexical processes influence the eye movements of skilled readers; it can handle the complexities of *garden path* sentences[478][iv].

English text is written on lines down a page and read left to right. The order in which the components of a formula are read depends on its contents, with the visual processing of subexpressions by experienced users driven by the mathematical syntax,[293,294] and the extraction of syntax happening in parallel.[502]

A study by Pelli, Burns, Farell, and Moore[445] found that 2,000 to 4,000 trials were all that was needed for novice readers to reach the same level of efficiency as fluent readers in the letter-detection task (efficiency was measured by comparing human performance compared to an ideal observer). They tested subjects aged 3 to 68 with a range of different (and invented) alphabets (including Hebrew, Devanagari, Arabic, and English). Even fifty years of reading experience, over a billion letters, did not improve the efficiency of letter detection. They also found this measure of efficiency was inversely proportional to letter perimetric complexity (defined as, inside and outside perimeter squared, divided by *ink* area).

Studies have found that peoples memory for objects within their visual field of view is organized according to the relative positions of the objects. For instance, a study by McNamara, Hardy, and Hirtle[382] gave subjects two minutes to memorize the location of objects on the floor of a room (see upper plot in Figure 2.32). The objects were then placed in a box and subjects were asked to place the objects in their original position. The memorize/recall cycle was repeated, using the same layout, until the subject could place all objects in their correct position.

The order in which each subject recalled the location of objects was used to create a hierarchical tree (one for each subject). The resulting trees (see lower plot in Figure 2.32) showed how subjects' spatial memory of the objects seen had a hierarchical organization, with the spatial distance between items being a significant factor in its structure.

Choice of display font is something that many developers are completely oblivious to. The use of Roman, rather than Helvetica (or serif vs. sans serif), is often claimed to increase reading speed and comprehension. The issues involved in selecting fonts are covered in a report detailing 'Font Requirements for Next Generation Air Traffic Management Systems'.[78]

Vision provides information about what people are thinking about; our gaze follows shifts of visual attention. Tracking a subject's eye movements and fixations when viewing images or

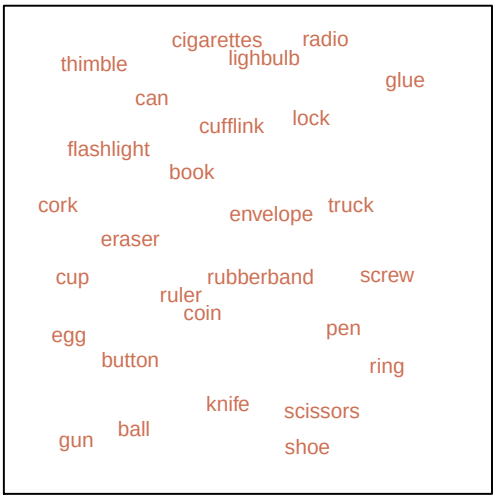


Figure 2.32: Example object layout and the corresponding ordered tree produced from the answers given by one subject. Data extracted from McNamara et al.[382] code

[iv] In 'Since Jay always jogs a mile seems like a short distance.' readers experience a disruption that is unrelated to the form or meaning of the individual words; the reader has been led down the syntactic garden path of initially parsing the sentence such that `a mile` is the object of `jogs` before realizing that `a mile` is the subject of `seems`.

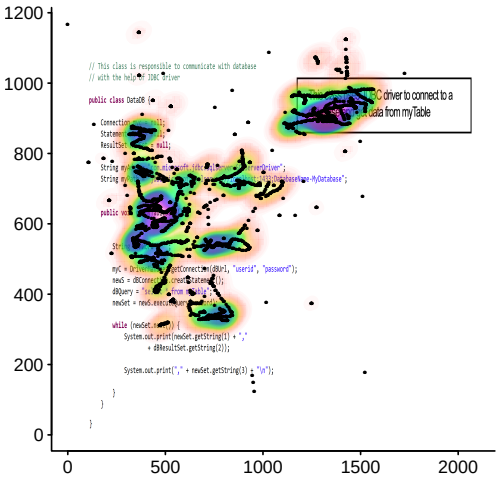


Figure 2.33: Heat map of one subject's cumulative fixations (black dots) on a screen image. Data kindly provided by Ali.[9] code

real-life scenes is an established technique in fields such as marketing and reading research; this technique is now starting to be used to research developer code reading behavior (see Figure 2.33).

2.6 Reasoning

The purpose of reasoning is to extract information from the available data; adding constraints on the data (e.g., known behaviors) can enable more specific information to be extracted.

What survival advantages does an ability to reason provide, or is it primarily an activity WEIRD people need to learn to pass school tests?

The kinds of questions asked in studies of reasoning appear to be uncontentious. However, studies[364,509] of reasoning using illiterate subjects from remote parts of the world received answers to verbal reasoning problems that were based on personal experience and social norms, rather than the western ideal of logic. The answers given by subjects, in the same location, who had received several years of schooling were much more likely to match those demanded by mathematical logic; the subjects had learned how to be WEIRD and *play the game*. The difficulty experienced by those learning formal logic suggests that there is no innate capacity for this task (innate capacity enables the corresponding skill to be learned quickly an easily). The human mind is a story processor, not a logic processor.[243]

The Wason selection task[605] is to studies of reasoning like the fruit fly is to studies of genetics. Wason's paper was published in 1968 and considered mathematical logic to be the norm against which human reasoning performance should be judged. The reader might like to solve this selection task:

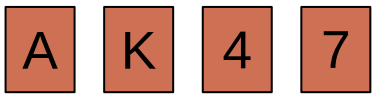


Figure 2.34: The four cards used in the Wason selection task. Based on Wason.[605] code

- Figure 2.34 depicts a set of four cards, of which you can see only the exposed face but not the hidden back. On each card, there is a number on one of its sides and a letter on the other.
- Below there is a rule which applies only to the four cards. Your task is to decide which if any of these four cards you must turn in order to decide if the rule is true.
- Don't turn unnecessary cards. Tick the cards you want to turn.

Rule: If there is a vowel on one side, then there is an even number on the other side.

The failure of many subjects to give the expected answer surprised many researchers and over the years a wide variety of explanations, experiments, thesis and books have attempted to explain what is going on.

Explanations include: detecting people who cheat[126] in groups where mutual altruism is the norm, interpreting the wording of questions in a pragmatic way based on how natural language is used rather than as logical formula[268] (i.e., assuming a social context; people are pragmatic virtuosos rather than logical defectives), and adjusting norms to take into account cognitive resource limitations (i.e., computational and working memory).[421]

Dual process theories[529,540,541] treat people as having two systems: unconscious reasoning and conscious reasoning.

Understanding spoken language requires reasoning about what is being discussed[385] and in a friendly shared environment it is possible to fill in the gaps by assuming that what is said is relevant,[537] with no intent to trick. In an adversarial context skeptical reasoning of the mathematical logic kind is useful for enumerating all possible interpretations of what has been said.

Outside of classroom problems, a real world context in which people explicitly reason is decision-making and here 'fast and frugal algorithms'[201,204] which provide answers quickly within the, often, limited constraints of time and energy. Context and semantics are crucial inputs to the reasoning process.[546]

Reasoning and decision-making appear to be closely related; however, reasoning researchers tied themselves to using the norm of mathematical logic for many decades and created something of research ghetto,[547] while decision-making researchers have been involved in explaining real-world problems.

Some Wason task related studies used a dialogue protocol (i.e., subjects' discuss their thoughts about the problem with the experimenter) and transcriptions[548] of these studies read like people new to programming having trouble understanding what it is they have to do to solve a problem by writing code.

People have different aptitudes and this can result in them using different strategies to solve the same problem,[540] e.g., an interaction between a subject's verbal and spatial ability and the strategy used to solve linear reasoning problems.[550] However, a person having high spatial ability, for instance, does not necessarily use a spatial strategy. A study by Roberts, Gilmore, and Wood[483] asked subjects to solve what appeared to be a spatial problem (requiring the use of a very inefficient spatial strategy to solve). Subjects with high spatial ability used non-spatial strategies, while those with low spatial ability used a spatial strategy. The conclusion made was that those with high spatial ability were able to see the inefficiency of the spatial strategy and selected an alternative strategy, while those with less spatial ability were unable to make this evaluation.

A study by Bell and Johnson-Laird[51] investigated the effect of the kind of questions asked on reasoning performance. Subjects had to give yes/no responses to two kinds of questions, asking about what is possible or what is necessary. The hypothesis was that subjects would find it easier to infer a *yes* answer to a question about what is possible, compared to one about what is necessary, because only one instance needs to be found (all instances need to be checked to answer *yes* to a question about necessity). For instance, in a game in which only two can play and the following information:

```
If Allan is in then Betsy is in.
If Carla is in then David is out.
```

answering *yes* to the question 'Can Betsy be in the game?' (a possibility) is easier than giving the same answer to 'Must Betsy be in the game?' (a necessity); see Table 2.4.

Question	**Correct *yes***	**Correct *no***
is possible	91%	65%
is necessary	71%	81%

Table 2.4: Percentage of correct answers to the two kinds of questions and two kinds of response. Data from Bell et al.[51]

However, subjects would find it easier to infer a *no* answer to a question about what is necessary, compared to one about what is possible, because only one instance needs to be found, whereas all instances need to be checked to answer *no* to a question about possibility. For instance, in another two person game and the following information:

```
If Allan is out then Betsy is out.
If Carla is out then David is in.
```

answering *no* to the question 'Must Betsy be in the game?' (a necessity) is easier than giving the same answer to 'Can Betsy be in the game?' (a possibility).

Conditionals in English In natural languages the conditional clause generally precedes the conclusion, in a conditional statement;[229] an example where the conditional follows the conclusion is 'I will leave, if you pay me' given as the answer to the question 'Under what circumstances will you leave?'. In one study of English[189] the conditional preceded the conclusion in 77% of written material and 82% of spoken material. There is a lot of variation in the form of the conditional.[58,94]

2.6.1 Deductive reasoning

The following are some factors that have been found to affect peoples performance in solving deductive reasoning problems:

- Belief bias: people are more willing to accept a conclusion, derived from given premises, that they believe to be true than one they believe to be false; see Table 2.5. A study by Evans, Barston, and Pollard[168] gave subjects two premises and a conclusion and asked them to state whether the conclusion was true or false (based on the premises given; the conclusions were rated as either believable or unbelievable by a separate group of subjects).

Status-context	Example	Accepted
Valid-believable		
	No Police dogs are vicious	
	Some highly trained dogs are vicious	
	Therefore, some highly trained dogs are not police dogs	88%
Valid-unbelievable		
	No nutritional things are inexpensive	
	Some vitamin tablets are inexpensive	
	Therefore, some vitamin tablets are not nutritional things	56%
Invalid-believable		
	No addictive things are inexpensive	
	Some cigarettes are inexpensive	
	Therefore, some addictive things are not cigarettes	72%
Invalid-unbelievable		
	No millionaires are hard workers	
	Some rich people are hard workers	
	Therefore, some millionaires are not rich people	13%

Table 2.5: Percentage of subjects accepting that the stated conclusion could be logically deduced from the given premises. Based on Evans et al.[168]

- Form of premise: a study by Dickstein[146] measured subjects performance on the 64 possible two premise syllogisms (a premise being one of the propositions: *All S are P*, *No S are P*, *Some S are P*, and *Some S are not P*). For instance, the following syllogisms show the four possible permutations of three terms (the use of S and P is interchangeable):

```
All M are S     All S are M     All M are S     All S are M
No P are M      No P are M      No M are P      No M are P
```

  The results showed that performance was affected by the order in which the terms occurred in the two premises of the syllogism.

  The order in which the premises are processed may affect the amount of working memory needed to reason about the syllogism, which in turn can affect human performance.[205]

- Individual differences.[33]

2.6.2 Linear reasoning

The ability to make relational comparisons provides a number of benefits, including selecting which of two areas contains the largest amount of food. Some animals, including humans, have a biologically determined representation of numbers, including elementary arithmetic operations, what one researcher has called the *number sense*.[141]

Being able to make relational decisions is a useful skill for animals living in hierarchical social groups where aggression is used to decide status.[441] Aggression is best avoided, as it can lead to injury; the ability to make use of relative dominance information (obtained by watching interactions between other members of the group) may remove the need for aggressive behavior during an encounter between two group members who have not recently contested dominance (i.e., there is nothing to be gained in aggression towards a group member who has previously been seen to dominate a member who is dominant to yourself).

Humans have a cognitive system capable of making approximate numerical estimates...

A study by van Oeffelen and Vos[584] investigated subject's ability to estimate the number of dots in a briefly (100 ms, i.e., not enough time to be able to count the dots) displayed image. Subjects were given a target number and had to answer yes/no whether they thought the image they saw contained this number of dots. Figure 2.35 shows the probability of a correct answer for various target numbers and a given difference between target number and number of dots displayed.

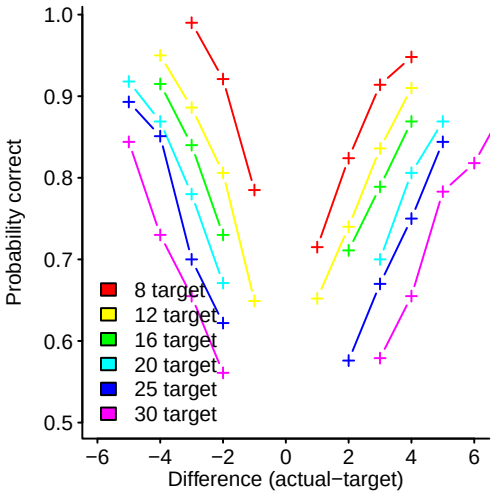


Figure 2.35: Probability a subject will successfully distinguish a difference between the number of dots displayed and a specified target number (x-axis is the difference between these two values). Data extracted from van Oeffelen et al.[584] code

The use of relational operators have an obvious interpretation in terms of linear syllogisms. A study by De Soto, London, and Handel[136] investigated a task they called *social reasoning*, using the relations *better* and *worse*. Subjects were shown two relationship statements involving three people, and a possible conclusion (e.g., 'Is Mantle worse than Moskowitz?) *and had 10 seconds to answer* `yes', `no', or 'don't know'. The British National Corpus[351]

lists *better* as appearing 143 times per million words, while *worse* appears under 10 times per million words and is not listed in the top 124,000 most used words.

	Relationships	Correct %		Relationships	Correct %
1	A is better than B		5	A is better than B	
	B is better than C	60.5		C is worse than B	61.8
2	B is better than C		6	C is worse than B	
	A is better than B	52.8		A is better than B	57.0
3	B is worse than A		7	B is worse than A	
	C is worse than B	50.0		B is better than C	41.5
4	C is worse than B		8	B is better than C	
	B is worse than A	42.5		B is worse than A	38.3

Table 2.6: Eight sets of premises describing the same relative ordering between A, B, and C (peoples names were used in the study) in different ways, followed by the percentage of subjects giving the correct answer. Based on De Soto et al.[136]

There are two patterns of performance behavior visible in Table 2.6.

- A higher percentage of correct answers were given when the direction was better-to-worse (case 1), than mixed direction (case 2, 3), and were least correct in the direction worse-to-better (case 4),
- A higher percentage of correct answers were given when the premises stated an end term (better or worse) followed by the middle term, than a middle term followed by an end term.

A second experiment in the same study gave subjects printed statements about people. For instance, 'Tom is better than Bill'. The relations used were *better*, *worse*, *has lighter hair*, and *has darker hair*. The subjects had to write the peoples names in two of four possible boxes; two arranged horizontally and two arranged vertically.

The results showed 84% of subjects selecting a vertical direction for better/worse, with better at the top (which is consistent with the *up is good* usage found in English metaphors[341]). In the case of lighter/darker 66% of subjects used a horizontal direction, with no significant preference for left-to-right or right-to left.

A third experiment in the same study used the relations *to-the-left* and *to-the-right*, and *above* and *below*. The above/below results were very similar to those for better/worse. The left-right results showed that subjects learned a left-to-right ordering better than a right-to-left ordering.

The results of this study show the effect that operand order has on the accuracy of peoples responses. However, the interpretation placed on the operator also plays a significant role. It appears that without knowing what interpretation a reader is likely to give to the operands and operators in the following two (logically equivalent) conditional expressions, for instance, it is not possible to select the one that is most likely to minimize incorrect reasoning on the part of readers.

```
if ((x <= y) && (x => z))
if ((x >= z) && (x <= y))
```

ACCU relational if-condition study...

Subject performance on linear reasoning improves the greater the distance between the items being compared; this *distance effect* is discussed in the section covering Numerosity.

2.6.3 Causal reasoning

A question often asked by developers, while reading source, is 'what causes this event/situation to occur?' Causal questions such as this are also common in everyday life. However, there has been relatively little mathematical research on causality (Pearl[442] is the standard reference; statistics deals with correlation) and little psychological research on causal reasoning.[531] It is sometimes possible to express a problem in either a causal or conditional form. A study by Sloman, and Lagnado[532] gave subjects one of the following two reasoning problems and associated questions:

- Causal argument form:

```
A causes B
A causes C
B causes D
C causes D
D definitely occurred
```

with the questions: `If B had not occurred, would D still have occurred?, *or `If B had not occurred, would A have occurred?*.

- Conditional argument form:

```
If A then B
If A then C
If B then D
If C then D
D is true
```

with the questions: `If B were false, would D still be true?, *or `If B were false, would A be true?*.

Table 2.7 shows that subject performance depended on the form in which the problem was expressed (more cases to be added...).

Question	Causal	Conditional
D holds?	80%	57%
A holds?	79%	36%

Table 2.7: Percentage 'yes' responses to various forms of questions (based on 238 responses). Based on Sloman et al.[532]

interpretation of causal verbs[210,528]...

2.7 Numerosity

Having a sense of quantity and being able to judge the relative size of two quantities provides a number of survival benefits, including being able to perform an approximate number of repetitions of some task (e.g., pressing a bar, see Figure 2.3) and deciding which of two clusters of food is the largest.

Studies have found that a variety of animals have the ability to make use of an approximate mental number system (sometimes known as the *number line*); the extent to which brains have a built-in number line or existing neurons are repurposed through learning is an active area of research.[139,419] Humans are the only creatures known to have a second system, one that can be used to represent numbers exactly: language.

What are the operating characteristics of the approximate number system? The characteristics that have most occupied researchers are the scale used (e.g., linear or logarithmic), the impact of number magnitude on cognitive performance and when dealing with two numbers the effect of their relative difference in value on cognitive performance.[141]

Studies of how single digit numbers are mentally represented[143] have found a linear scale used by subjects from western societies and a logarithmic scale for subjects from indigenous cultures that have not had formal schooling.

Engineering and science sometimes deal with values spanning many orders of magnitude, a difference that people are unlikely to encounter in everyday life. How do people mentally represent large value ranges?

A study by Landy, Charlesworth and Ottmar[344] asked them to click the position on a line (labeled with one thousand at the left end and one billion on the right end) which they thought was the appropriate location for each of the 182 they saw (selected from 20 evenly spaced values between one thousand and one million, and 23 evenly spaced values between one million and one billion).

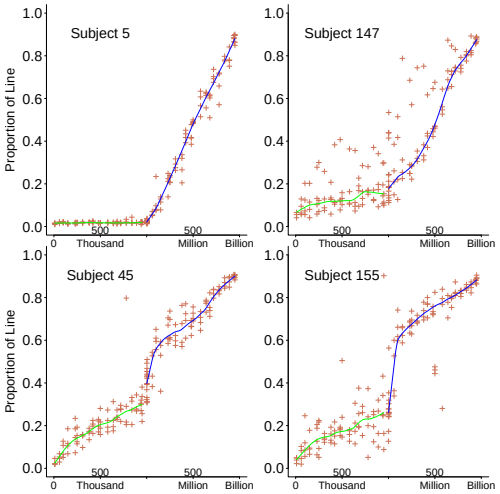


Figure 2.36: Line locations chosen for the numeric values seen by each of four subjects; color of fitted loess line changes at one million boundary. Data kindly provided by Landy.[344] code

Various patterns occurred in subject responses, with the top left plot in Figure 2.36 showing one of the most common. Most subjects placed one million at the halfway point (as-if using

a logarithmic scale), placing values below/above a million on separate linear scales. Landy et al developed a model based on Category Adjustment theory,[?] where subjects selected a category boundary (e.g., one million, creating the categories: the thousands and the millions), a location for the category boundary along the line and a linear(ish)[v] mapping of values to relative position within their respective category.

Studying the learning and performance of simple arithmetic operations has proven complicated;[186,585] models of simple arithmetic performance[348] have been built. Working memory capacity has an impact on the time taken to perform mental arithmetic operations and the likely error rate, e.g., remembering carry or borrow quantities during subtraction;[285] the human language used to think about the problem is also has an impact.[284]

How do people compare multi-digit integer constants? For instance, do they compare them digit by digit (i.e., a serial comparison), or do they form two complete values before comparing their magnitudes (the so-called *holistic* model)? Studies show that the answer depends on how the comparisons are made, with results consistent with the digit by digit[626] and holistic[142] approaches being obtained.

Other performance related behaviors include:

- *Split effect*: taking longer to reject false answers that are close to the correct answer (e.g., $4 \times 7 = 29$) than those that are further away (e.g., $4 \times 7 = 33$),
- *associative confusion* effect: answering a different question from the one asked (e.g., giving 12 as the answer to $4 \times 8 =?$, which would be true had the operation been addition),
- plausibility judgments:[354] using of a rule rather than retrieving a fact from memory to verify the answer to a question; for instance, adding an odd number to an even number always produces an odd result,

2.7.1 Problem size and symbolic distance effect

The time taken to produce an answer and the error rate increase as the numeric value of operands increases (e.g., subjects are slower to solve $9+7$ than $2+3$); this is known as the *problem size effect*.

A study by Campbell[92] subject performance when multiplying numbers between two and nine, and dividing a number (that gave an integer result) by a number between two and nine. Figure 2.37 shows that the number of errors generally increases with operand and result value (both operands have the same value appears to be a special case, see blue points). The time taken for subjects to add[435] or multiply[434] two single digit values, e.g., $p \times q$, is proportional to $\min(p,q)$ (the time taken to confirm an answer follows a similar pattern).

The *symbolic distance effect* is a general effect that occurs when people compare two items sharing a distance-like characteristic which can be easily estimated; the larger the distance between two items the faster people are likely to respond to a comparison question involving this characteristic. For instance, comparisons of social status[108] and geographical distance[270] (also see Figure 2.32). Inconsistencies between the symbolic distance and actual distance can increase the error rate,[257] e.g., is the following relation true? $3 > 5$.

In a study by Tzelgov, Yehene, Kotler and Alon[577] subjects trained one-hour per day for six days, learning the relative order of nine graphical symbols. A symbolic distance effect was seen in subject performance, when answering questions about relative graphical symbol order.

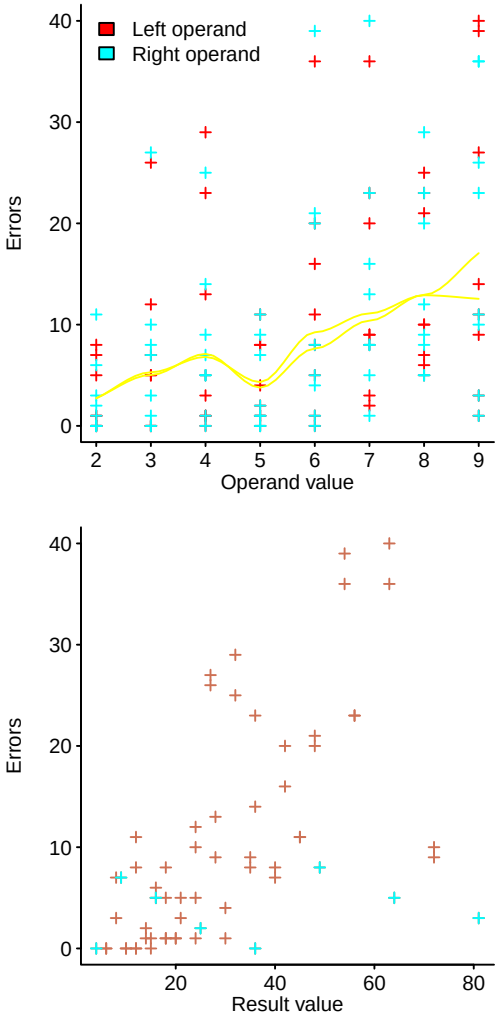
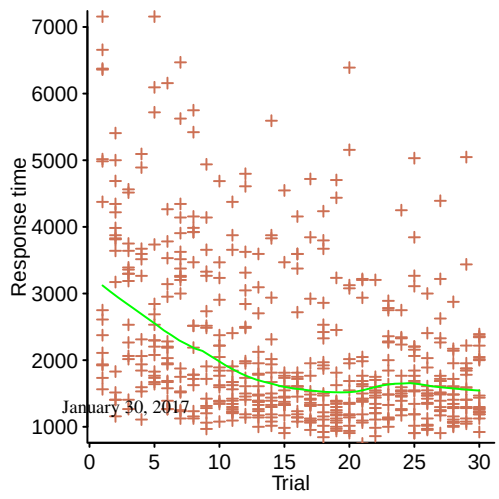


Figure 2.37: Number of errors, in 132 simple multiplication trials (e.g., 3×7), upper plot shows operand values (a loess fit in yellow) and lower plot result value (points where both operands have the same value are in blue). Data from Campbell.[92] code

2.8 Human factors

Not only do Humans come in various shapes and sizes, an individual's performance can significantly change from one moment to the next.

A study by Remington, Yuen and Pashler[480] compared subject performance between using a GUI and a command line (with practice, there was little improvement in GUI performance, but command line performance continued to improve and eventually overtook GUI performance). Figure 2.38 shows the command line response time for one subject over successive blocks of trials and a fitted loess line.



[v] Category Adjustment theory supports curvaceous lines.

2.8.1 People make mistakes

Evolution makes use of survival of the fittest, i.e., it is based on relative performance; a consistent flawless performance is not only unnecessary, but a waste of precious resources.

Socially, making mistakes is an accepted fact of life and people are given opportunities to correct mistakes, if that is considered necessary.

Reason[475] is a readable introduction to human error.

For a given task, obtaining information on the kinds of mistakes that are likely to be made (e.g., entering numeric codes on a keyboard[430]) and modeling the behavior (e.g., subtraction mistakes[585] made by children learning arithmetic) requires a lot of effort, even for simple tasks. Modeling the mistakes people make has proven to be very difficult. Researchers are still working to build good models[533] for the apparently simple task of text entry.

One technique for increasing the number of errors made by subjects in an experiment is to introduce factors that will increase the number of mistakes made. For instance, under normal circumstances the letters/digits viewed by developers are clearly visible and the viewing time is not constrained. In experiments run under these conditions subjects make very few errors. To obtain enough data to calculate letter similarity/confusability, studies[401] have to show subjects images of single letters/digits that have been visually degraded in some way, or given a limited amount of time to make a decision, or both until a specified error rate is obtained.[570] While such experiments may be provide the only available information on the topic of interest, their ecological validity has to be addressed (compared to say asking subjects to rate pairs of letters for similarity[524]).

How often do people make mistakes?

A lower bound on human error rate, when performing over an extended period, is probably that of astronauts in space; making an error during a space mission can have very serious consequences and a huge amount of resources are devoted to astronaut training. NASA maintains several databases of errors made by operators during simulation training and actual missions; measurements of human error rates, for different missions, of between 1.9×10^{-3} and 1.05×10^{-4} have occurred.[96]

Touch typists, who are performing purely data entry:[373] with no error correction 4% (per keystroke), typing nonsense words (per word) 7.5%.

A number of human reliability analysis methods[97] for tasks in safety critical environments are available. The Cognitive Reliability Error Analysis Model (CREAM) is widely used; Calhoun et al[91] work through a calculation of the probability of an error during the International Space Station ingress procedure, using CREAM.

What causes error rates to increase?... Are some people more error prone than others?...

How do people respond when their mistakes are discovered?

A study by Jørgensen and Moløkken[312] interviewed employees, from one company, with estimation responsibilities. Analysis of the answers showed a strong tendency for people to perceive factors outside their control as important reasons for inaccurate estimates, while factors within their control were typically cited as reasons for accurate estimates.

2.8.2 Cognitive effort

When attempting to solve a problem, a person's cognitive system makes cost/accuracy trade-offs. The details of how it forms an estimate of the value, cost, and risk associated with an action, and carries out the trade-off analysis is not known (various models have been proposed[228]). An example of the effects of these trade-offs is provided by a study by Fu and Gray,[193] where subjects had to copy a pattern of colored blocks (on a computer-generated display). Remembering the color of the block to be copied and its position in the target pattern created a memory effort; a perceptual-motor effort was introduced by graying out the various areas of the display where the colored blocks were visible; these grayed out areas could be made temporarily visible using various combinations of keystrokes and mouse movements. Subjects had the choice of expending memory effort (learning the locations of different colored blocks) or perceptual-motor effort (using keystrokes and mouse movements to uncover different areas of the display). A subject's total effort is the sum of the perceptual motor effort and the memory storage and recall effort.

The subjects were split into three groups; one group had to expend a low effort to uncover the grayed out areas, the second acted as a control, and the third had to expend a high effort to uncover the grayed out areas. The results showed that the subjects who had to expend a high perceptual-motor effort, uncovered grayed out area fewer times than the other two groups. These subjects also spent longer looking at the areas uncovered, and moved more colored blocks between uncoverings. The subjects faced with a high perceptual-motor effort reduced their total effort by investing in memory effort. Another consequence of this switch of effort investment, to use of memory, was an increase in errors made.

incentives improve cognitive performance...

Cognitive load theory...

What are the physical processes that generate the feeling of mental effort? The processes involved remain poorly understood.[514] Proposals include: metabolic constraints (the brain accounts for around 20% of heart output, and between 20% to 25% of oxygen and glucose requirements), but the energy consumption of the visual areas of the brain while watching television are higher than the consumption levels of those parts of the brain associated with difficult thinking, that the body's reaction to concentrated thinking is to try to conserve energy, for use in other opportunities that could arise in the immediate future, by creating a sense of effort.[339]

2.8.3 Time discounting

People seek to consume pleasurable experiences sooner and to delay their appointment with painful experiences, i.e., people have a tendency to accept less satisfaction in the short-term than could be obtained by pursuing a longer-term course of action. Studies have found that animals, including humans, use a hyperbolic discount function for time variable preferences.[144] The hyperbolic delay discount function is:

$$v_d = \frac{V}{1+kd}$$

where: v_d is the delayed discount, V the undiscounted value, d the delay and k some constant.

some ok data...[?]

2.8.4 Personality & intelligence

Perhaps the most well-known personality test is the *Meyer-Briggs type indicator*, or MBTI (both registered trademarks of Consulting Psychologists Press). A lot has been published about the reliability and validity of this test;[378,454] commercial considerations have also made it research difficult. The International Personality Item Pool (IPIP)[209] is a public domain measure containing over three thousand items that is growing in popularity.

Tests measuring something called IQ have existed for over 100 years. The results are traditionally encapsulated in a single number, but tests claiming to measure the various components of intelligence are available;[138] the model is that people possess a general intelligence *g* plus various domain specific intelligences, e.g., in tests involving maths the results would depend on *g* and maths specific intelligence and in tests involving use of language the results would depend on *g* and language specific intelligence.

The five factory theory of personality is a popular basis for personality and intelligence tests... it has its critics[125]...

Some personality traits are likely to be beneficial and others detrimental in those involved in some development activities... Without reliable techniques for measuring these characteristics...

Perhaps one of the most important traits is a willingness to spend large amounts of time engaged in a single activity...

emailed asking for data, may be on its way...[?,?]

The results of personality and intelligence tests are sometimes included in the job selection process and sometimes in studies investigating developer mental characteristics.

Cultural intelligence hypothesis, specific set of social skills for exchanging knowledge in social groups children and chimpanzees have similar cognitive skills for dealing with the physical world, but children have more sophisticated skills for dealing with the social world.[260]

2.8.5 Attention

Most of the sensory information received by the brain is handled without the need for conscious processing. Conscious attention is like a spotlight shining the available resources on a chosen area. In today's world, there is often significantly more information available to a person than they have available attention resources, we live in an age of attention.

Much of the psychology research on attention has investigated how inputs from our various senses are handled. It is known that they operate in parallel and at some point there is a serial bottleneck, beyond which point it is not possible to continue processing input stimuli in parallel. The point at which this bottleneck occurs is a continuing subject of debate; there are early selection theories, late selection theories, and theories that combine the two.[436] People can also direct attention to their internal thought processes and memories.

Read the bold print in the following paragraph:

Somewhere **Among** hidden **the** in **most** the **spectacular** Rocky Mountains **cognitive** near **abilities** Central City **is** Colorado **the** an **ability** old **to** miner **select** hid **one** a **message** box **from** of **another.** gold. **We** Although **do** several **this** hundred **by** people **focusing** have **our** looked **attention** for **on** it, **certain** they **cues** have **such** not **as** found **type** it **style**.

What do you remember from the regular, non-bold, text? What does this tell you about selective attention? and...

A study by Rogers and Monsell[487] used the two tasks of classifying a letter as a consonant or vowel, and classifying a digit as odd or even. The subjects were split into three groups. One group was given the letter classification task, the second group the digit classification task, and the third group had to alternate (various combinations were used) between letter and digit classification. The results showed that having to alternate tasks slowed the response times by 200 to 250 ms and the error rates went up from 2% to 3% to 6.5% to 7.5%. A study by Altmann[14] found that when the new task shared many features in common with the previous task (e.g., switching from classifying numbers as odd or even, to classifying them as less than or greater than five) the memories for the related tasks interfered, causing a reduction in subject reaction time and an increase in error rate.

and this relates to software engineering...

2.8.6 Risk taking

Making a decision based on incomplete or uncertain information involves an element of risk. How do people handle risk?

The term *risk asymmetry* refers to the fact that people have been found to be *risk averse* when deciding between alternatives that have a positive outcome, but are *risk seeking* when deciding between alternatives that have a negative outcome.[vi]

While there is a widespread perception that women are more risk averse than men, existing studies are either not conclusive or show a small effect[179] (many suffer from small sample sizes and dependencies on the features of the task subjects are given). For the case of financial risks the evidence[102] that men are more willing to take financial risks than women is more clear cut. The evidence from attempts to improve road safety is that 'protecting motorists from the consequences of bad driving encourages bad driving'.[2]

Utility function...

analysis of BART data?...?

A study by Jones[303] investigated the possibility that some subjects in an experiment involving recalling information about previously seen assignment statements were less willing to risk giving an answer when they had opportunity to specify that in real-life they would refer back to previously read code. Previous studies[300][301] had found that a small percentage of subjects consistently gave much higher rates of "would refer back". One explanation was that these subjects had a smaller short term memory capacity than other subjects (STM capacity does vary between people), another explanation is that these subjects are much more risk averse than the other subjects.

[vi] While studies[259] based on subjects drawn from non-WEIRD societies sometimes produce different results, this book assumed developers are WEIRD.

The Domain-Specific Risk-Taking (DOSPERT) questionnaire[64,608] was used to measure subject's risk attitude. The results found no correlation between risk attitude (at least as measured by DOSPERT) and "would refer back".

2.8.6.1 Overconfidence

Overconfidence is a false belief which can create unrealistic expectations and lead to hazardous decisions being made (e.g., to allocate insufficient resources to complete a job).

There is evidence from simulation[296] that in some cases, on average, there are benefits to being overconfident (see rexample[0909.R]). A study[29] of commercialization of new inventions found that while inventors are significantly more confidence and optimistic than the general population, the likely return on their investment of time and money in an invention is negative. A few provide a huge payoff. While overconfidence at the individual level can have fatal consequences, at the group level overconfidence can increase group fitness.

A study by Lichtenstein and Fishhoff[357] asked subjects general knowledge questions, with the questions divided into two groups, hard and easy. Figure 2.39 shows that subjects' overestimated their ability (x-axis) to correctly answer hard questions, but underestimated their ability to answer easy questions; solid line denotes perfect self-knowledge.

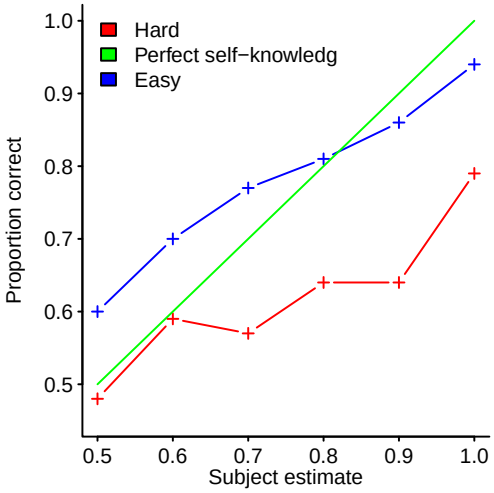


Figure 2.39: Subjects' estimate of their ability (x-axis) to correctly answer a question and actual performance in answering on the left scale. The responses of a person with perfect self-knowledge is given by the solid line. Data extracted from Lichtenstein et al.[357] code

These, and subsequent results, show that the skills and knowledge that constitute competence in a particular domain are the same skills needed to evaluate one's (and other people's) competence in that domain. *Metacognition* is the term used to denote the ability of a person to accurately judge how well they are performing.

Peoples' belief in their own approach to getting things done can result in them ignoring higher performing alternatives;[25] this behavior has become known as *the illusion of control*.[346]

It might be thought that people who have previously performed some operation would be in a position to make accurate predictions about future performance on those operations. However, studies have found[492] that while people do base predictions of future duration on their memories of how long past events took, these memories are systematic underestimates of past duration. People appear to underestimate future event duration because they underestimate past event duration.

2.8.7 Decision-making

Human decision-making often takes place in an environment of incomplete information and limited decision-making resources (e.g., working memory capacity and thinking time); people have been found to adopt various strategies to handle this situation,[372] balancing the predicted cognitive effort required to use a particular decision-making strategy against the likely accuracy achieved by that strategy.

The term *bounded rationality*[521] is used to describe an approach to problem solving used when limited cognitive resources are available. A growing number of studies[204] have found that the methods used by people to make decisions and solve problems are often close enough to optimal, given the resources available to them; even when dealing with financial matters.[380] If people deal with money matters in this fashion, how can their approach to software development decisions fare any better?

Consumer research into understanding how shoppers decide among packets of soap powder on a supermarket shelf has uncovered a set of mechanisms that are applicable to decision-making in general, e.g., decision-making around the question of which soap will wash the whitest is no different from the question of where should an **if** statement or a **switch** statement be used.

Before a decision can be made, a decision-making strategy has to be selected. People have been found to use a number of different decision-making strategies and this section discusses some of these strategies and the circumstances under which people might apply them.[vii]

While coding a developer may first attempt to evaluate alternatives mentally, which constrains the decision-making process to operating within the constraints of working memory...

[vii] 'The Adaptive Decision Maker' by Payne, Bettman, and Johnson[440] covers this in detail.

A developer who has an hour to write a program knows there is not enough time to make complicated trade-offs among alternatives. There is a hierarchy of responses for how people deal with time pressure:[440] they work faster; if that fails, they may focus on a subset of the issues; if that fails, they may change strategies (e.g., from alternative based to attribute based).

Decision strategies differ in several ways, for instance, some make trade-offs among the attributes of the alternatives (making it possible for an alternative with several good attributes to be selected instead of the alternative whose only worthwhile attribute is excellent); they also differ in the amount of information that needs to be obtained and the amount of cognitive processing needed to make a decision.

The weighted additive rule: this requires the greatest amount of effort, but delivers the most accurate result (strong attachments to a particular point of view has been found to influence the weight given to evidence that contradicts this view[566]). The steps are as follows:

- build a list of attributes for each alternative,
- assign a value to each of these attributes,
- assign a weight to each of these attributes (these weights could, for instance, reflect the relative importance of that attribute to the person making the decision, or the probability of that attribute occurring),
- for each alternative, sum the product of each of its attributes' value and weight,
- select the alternative with the highest sum.

When dividing two numbers in a loop, a developer has to decide whether it is worthwhile testing the denominator against zero before the division. The obvious attributes are performance and reliability. A comparison would decrease performance, an attribute whose weight will depend on the time-critical nature of the loop. The benefit from making a comparison is the potential to improve reliability by detecting an error case.

The equal weight heuristic: this is a simplification of the weighted additive rule that assigns the same weight to every attribute. This heuristic might be applied when accurate information on the importance of each attribute is not available and a decision is made to use equal weights.

The frequency of good and bad features heuristic: estimating a numerical measure for an attribute may be very difficulty and a binary measurement based on good/bad is sometimes good enough (i.e., looking at things in black and white). This heuristic is a simplification of the equal weight heuristic.

The majority of confirming dimensions heuristic: estimating an absolute measure for an attribute may be very difficult, but a yes/no answer can be given to the question: 'Is the value of attribute X greater (or less) for alternative A compared to alternative B?'. The answer can be used to determine which alternative has the most (or least) of each attribute.

The algorithm starts by selecting a pair of alternatives, compare each matching attribute of the alternatives and select the alternative that has the greater number of winning attributes, the winning alternative is then paired with an uncompared alternative and the compare/select steps are repeated until one of the alternatives is left.

The satisficing heuristic: the result of this heuristic can depend on the order in which alternatives are checked and often does not check all alternatives. The steps are as follows:

- assign a cutoff, or aspirational, level that must be met by each attribute,
- for each alternative:
  - check each of its attributes against the cutoff level, rejecting the alternative if the attribute is below the cutoff,
  - if there are no attributes below the cutoff value, accept this alternative,
- if no alternative is accepted, revise the cutoff levels associated with the attributes and repeat the previous step.

When selecting a library function to obtain some information, the list of attributes might include the amount of information returned and the format it is returned in (relative to the format it is required to be in). Once a library function meeting the developer's minimum aspirational level is found, additional effort need not be invested in finding a better alternative.

The lexicographic heuristic: has a low effort cost, but it might not be very accurate. It can also be intransitive; with X preferred to Y, Y preferred to Z, and Z preferred to X. The steps are as follows:

- determine the most important attribute of all the alternatives,
- find the alternative that has the best value for the selected most important attribute,
- if two or more alternatives have the same value, select the next most important attribute and repeat the previous step using the set of alternatives whose attribute values tied,
- the result is the alternative having the best value on the final, most important, attribute selected.

An example of the intransitivity that can occur, when using this heuristic, might occur when writing software for embedded applications. Here the code has to fit within storage units that are available in fixed-size increments (e.g., 8 K chips). It may be possible to increase the speed of execution of an application by writing code for special cases, or have generalized code that is more compact, but slower. Table 2.8 shows some commonly seen, alternatives:

Alternative	Storage Needed	Speed of Execution
X	7 K	Low
Y	15 K	High
Z	10 K	Medium

Table 2.8: Storage/Execution performance alternatives.

Based on storage minimization, X is preferred to Y. Because storage comes in 8 K increments there is no preference, based on the storage attribute, between Y and Z. Based on speed of execution, Y is preferred to Z, and Z is preferred to X.

The habitual heuristic: looks for a match of the current situation against past situations, it does not contain any evaluation function (although there are related heuristics that evaluate the outcome of previous decisions). The single step for this heuristic is: select the alternative chosen last time for that situation.

Going with a winning solution suggests one of two possibilities:

- so little is known that once a winning solution is found, it is better to stick with it than to pay the cost (time and the possibility of failure) of looking for a better solution that might not exist,
- the developer has previously performed an extensive analysis and the best solution is known.

Copying others: Find good solutions to problems is hard, doing what others do can be a cost effective strategy. *When Strategies*: copy when established behavior is unproductive, copy when asocial learning is costly, copy when uncertain. *Who Strategies*: copy the majority, copy if rare, copy successful individuals, copy if better, copy if dissatisfied, copy good social learners, copy kin, copy "friends", copy older individuals.

Studies have found that having to justify a decision can affect the choice of decision-making strategy used.[565] One strategy that handles accountability is to select the alternative that the perspective audience is thought most likely to select.[564] People who have difficulty determining which alternative has the greatest utility tend to select the alternative that supported the best overall reasons (for choosing it).[523]

Requiring developers to justify why they have not followed existing practice can be a two-edged sword. Developers can respond by deciding to blindly follow everybody else (a path of least resistance), or they can invest effort in evaluating alternatives (not necessarily cost

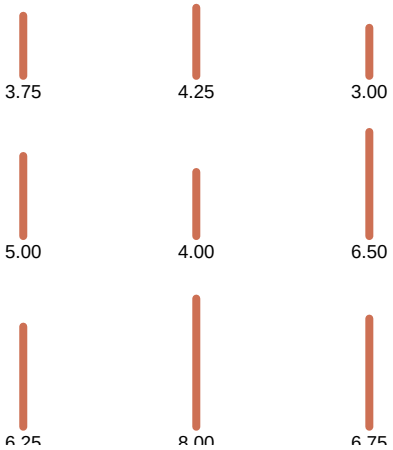


Figure 2.40: Each row shows a scaled version of the three stripes, along with actual lengths in inches, from which subjects were asked to select the longest. Based on Asch.[28] code

effective behavior, since the invested effort may not be warranted by the expected benefits). The extent to which some people will blindly obey authority was chillingly demonstrated in a number of studies by Milgram.[389]

Social pressure can cause people to go along with the decision voiced by those currently in the room. A study by Asch[28] asked a group of seven to nine subjects to individually state which of three black strips they considered to be the longest (see Figure 2.40), the group sat together in front of the stripes subjects and could interact with each other; all the subjects, except one, were part of the experiment and in 12 of 18 questions selected a stripe that was clearly not the longest (i.e., the majority gave an answer clearly at odds with visible reality). It was arranged that the actual subject did not have to give an answer until hearing the answers of most of the other subjects.

The actual subject in 24% of groups always gave the correct answer, in 27% of groups the subject agreed with the incorrect majority answer between eight and twelve times and just under half varied in the extent to which they followed the majority decision. When the majority selected the most extreme incorrect answer, i.e., the shortest stripe, subjects giving an incorrect answer selected the less extreme incorrect answer in 20% of cases.

Social conformity as a signal of belonging... Corporate IT fashion Figure 1.8...

Informational cascades[59]...

Without any evidence about the efficiency of existing practice, it is not unreasonable to assume that the practice arose out of a random event that became dominant. A more efficient way of performing the same practice may not become established because it does not provide a sufficient benefit to overcome existing practices...

How a problem is posed can have a large impact on the decision made.

A study by Regnell, Höst, och Dag, Beremark and Hjelm[477] asked 10 subjects to assign a relative priority to two lists of requirements (subjects' had a total budget of 100,000 units and had to assign units to requirements). One list specified that subjects should prioritize the 17 high level requirements and the other list specified that a more detailed response, in the form of prioritizing every feature contained within each high level requirement, be given.

Comparing the totals for the 17 high level requirements (summing the responses for the detailed list), showed that the correlation was not very strong (the mean across 10 subjects was 0.46, see rexample[prioritization.R]).

Anchoring is a cognitive bias where people assign too much weight to the first piece of information they obtain, relating to the topic at hand. A study by Jørgensen and Sjøberg[313] asked professionals and students to estimate the development effort for a project, with one group of subjects being given a low estimate from the *customer* another a low estimate and the third group no customer estimate. The results (see rexample[anchor-estimate.R]) showed that estimates from subjects given a high/low customer estimate where much higher/lower than subjects who did not receive any customer estimate.

A study by Jørgensen and Grimstad[310] asked subjects to estimate the number of lines of code they wrote per hour, with subjects randomly split into two groups; one anchored with the question: 'Did you on average write more or less than 1 Line of Code per work-hours in your last project?' and the other with: 'Did you on average write more or less than 200 Lines of Code per work-hours in your last project?' Fitting a regression model to the results showed the form of the question changed the mean estimate by around 72 lines (sd 10)... rexample[estimation-biases.R]

2.8.8 Miscellaneous characteristics

There are a variety of human characteristics that probably have some impact on the software development process, but the size of the impact... including the following:

- Reaction time has an ex-Gaussian distribution... emailed for data...
- Fritt's law: ...
- Hick's law: Time taken to decide which item, from a known list of items, is currently visible[251]...

- Ageing effects: The Seattle Longitudinal Study[501] has been following the intellectual development of over six thousand people since 1956 (surveys of the individuals in the study are carried out every seven years). Findings include: " . . . there is no uniform pattern of age-related changes across all intellectual abilities, . . . " and " . . . reliable replicable average age decrements in psychometric abilities do not occur prior to age 60, but that such reliable decrements can be found for all abilities by age 74 . . . " An analysis[70] of the workers on the production line at a Mercedes-Benz assembly plant found that productivity did not decline until at least up to age 60. + Years of experience vs. experimental performance. . . .

  Changes in a person's social and economic standing over time are likely to have a much larger impact than changes in mental ability. For instance, commitments outside work (e.g., family) reduce the time available for acquiring new skills, people who are paid more (an individual's salary is likely to increase over the years) are expected to deliver more (people promoted to their maximum level of incompetence; its cheaper to hire younger people who are already familiar with new technologies than retrain more expensive older people). and the data backing up these claims. . .

  Figure 3.12 shows an age distribution of developers.

2.9 Developer performance

Companies want to hire those people who will give the best software development performance. At the moment the only reliable way of measuring developer performance is to measure developer output (a reasonably accurate model of the workings of human mental operations is still in the future).

One working characteristic of the brain that can estimated in the number of operations it can potentially perform per second (a commonly used method of measuring the performance of silicon-based processors, however, processors evolve to contain ever more special purpose subsystems and these subsystems in turn evolve, the correlation between MIPS and commonly seen performance declines).

The brain might simply be a very large neural net, so there will be no instructions to count as such; Merkle[386] used the following approaches to estimate the number of synaptic operations per second (supplying the energy needed to fire neurons limits the number that can be simultaneously active in a local region to between 1% and 4% of neurons in that region[355]):

- multiplying the number of synapses (10^{15}) by their speed of operation (about 10 impulses/second) gives 10^{16} synapse operations per second,
- the retina of the eye performs an estimated 10^{10} analog add operations per second. The brain contains 10^2 to 10^4 times as many nerve cells as the retina, suggesting that it can perform 10^{12} to 10^{14} operations per second,
- a total brain power dissipation of 25 watts (an estimated 10 watts of useful work) and an estimated energy consumption of 5×10^{-15} joules for the switching of a nerve cell membrane provides an upper limit of 2×10^{15} operations per second.

A synapse switching on and off is the same as a transistor switching on and off in that they both need to be connected to other switches to create a larger functional unit. It is not known how many synapses are used to create functional units, or even what those functional units might be. The distance between synapses is approximately 1 mm and sending a signal from one part of the brain to another part requires many synaptic operations, for instance, to travel from the front to the rear of the brain requires at least 100 synaptic operations to propagate the signal. So the number of synaptic operations per high-level, functional operation is likely to be high. Silicon-based processors can contain millions of transistors; the potential number of transistor-switching operations per second might be greater than 10^{14}, but the number of instructions executed is significantly smaller.

Although there have been studies of the information-processing capacity of the brain (e.g., visual attention[589] and storage rate into long-term memory[72,342]), we are a long way from being able to deduce the likely work rates of the components of the brain used during code comprehension.

Processing units need a continuous energy to function. The brain does not contain any tissue that stores energy and obtains all its energy needs through the breakdown of blood-borne

glucose. Consuming a glucose drink has been found to increase blood glucose levels and enhance performance on various cognitive tasks.[321] Also, fluctuations in glucose levels have an impact on an individual's ability to exert self-control,[195] with some glucose intolerant individuals not always acting in socially acceptable ways.[viii]

Do people vary in the ability to supply energy to the brain and remove waste products?...

How do developers differ in their measurable output performance?

Although much talked about, there has been little research on individual developer productivity. Claims of a 28-to-1 productivity difference between developers is sometimes still bandied about. The, so called *Grant-Sackman study*[224] is based on an incorrect interpretation of a summary of the data.[461] The data shows a performance difference of around 6-to-1 between developers using different systems for creating software (i.e., batch vs. online).

Most organizations do not attempt to measure the mental characteristics of developer job applicants; unlike many other jobs for which individual performance is an important consideration. Whether this is because of the existing non-measurement culture, lack of reliable measuring procedures, or fear of frightening off prospective employees is not known.

A study of development and maintenance costs of programs written in C and Ada[623] found no correlation between salary grade (or employee rating) and rate of bug fix or add feature rate.

One metric used in software testing is number of faults found. In practice non-failing tests, written by software testers, are useful because they provide evidence that particular functionality behaves as expected.

A study by Iivonen[283] analysed the defect detection performance of those involved in testing software at several companies. Table 2.9 shows the number of defects detected by six testers (apart from the first, all columns show percentages), along with self-classification of the seriousness, followed by the default status assigned by others.

Tester	Defects	Extra Hot	Hot	Normal	Open	Fixed	No fix	Duplicate	Cannot reproduce
A	74	4	1	95	12	62	26	12	0
B	73	0	56	44	15	87	6	2	5
C	70	0	29	71	36	71	24	0	4
D	51	0	27	73	33	85	6	0	9
E	50	2	16	82	30	89	9	0	3
F	18	0	22	78	22	64	14	0	21

Table 2.9: Defects detected by six testers (some part-time and one who left the company during the study period) and their status. Data from Iivonen.[283]

A performance comparison, based on defects reported, requires combining these figures (and perhaps others, e.g., likelihood of being experienced by a customer) into a value that can be reliably compared across testers. Is the weight assigned to defects classified as 'No fix' is larger than that given to 'Cannot reproduce' or 'Duplicate'?

To what extent would a tester's performance, based on measurements involving one software system in one company be transferable to another system in the same company or another company? Iivonen interviewed those involved in testing to find out what characteristics were thought important in a tester. Knowledge of customer processes and use cases was a common answer; this usage knowledge enables testers to concentrate on those parts of the software that customers are most likely to use and be impacted by incorrect operation, it also provides the information needed to narrow down the space of possible input values.

Knowledge of the customer ecosystem and software development skills are the two blades that have to mesh together to create useful software systems.

[viii] There is a belief in software development circles that consumption of chocolate enhances cognitive function. A systematic review of published studies[503] found around a dozen papers addressing this topic, with three finding some cognitive benefits and five some improvement in mood state.

Chapter 3

Stories told by data

3.1 Introduction

An appreciation of the general patterns present in the data is the starting point for understanding the stories the data has to tell and this chapter starts with a discussion of the techniques that might be used to uncover these patterns. The person performing the analysis may have to communicate the story they have discovered to other people and the second part of the chapter deals with communicating stories about data.

Ideally you will have a clear idea of the questions for which answers are sought, in practice there may be a lot of uncertainty about exactly what questions are.

Ideally you will have the time and resources needed to obtain the data needed to answer the questions asked, in practice obtaining data is often time-consuming and/or expensive and you often have to make do with what data is cheaply and quickly available and this may only indirectly relate to the questions being asked.

Ideally the data contains little noise, in practice the available data may be very noisey and cleaning may be very time-consuming.

Ideally the statistical analysis techniques used is capable of providing answers to the desired level of certainty, in practice it may not be possible to draw any meaningful conclusions from the data or more questions will be uncovered.

Data analysis is like programming in that you get better with practice, there are a few basic techniques that can be used to solve many problems and doing what you did on a previous successful project can save lots of time.

For the most part this book assumes that you have a dataset and collecting of data is only discussed as a secondary issue.

Perhaps the information contained in the measurements is not good and need to presented in a favourable light. This book aims to improve understanding, not obscure understanding.

There are many ways in which information can be presented in a way that misleads. This books tries to highlight techniques that aid understanding and without empirical data on the most common mistakes there is nothing to guide any discussion.

At a bare minimum, the story told by an analysis of data needs to meet the guidelines for truthfulness in advertising specified by the national advertising standards' authorities. It manufacturers of soap powder have to meet these requirements in what they tell the public, then so should you.

Check assumptions derived from visualizations Assumptions suggested by a visualization of a data need to be checked numerically. For instance, Figure 3.1 shows professional software development experience, in years, of subjects taking part in an experiment using a particular language; it suggests that as a group, the PHP subjects are more experienced than the Java subjects.

Comparing years of experience for the PHP and Java developers, using a permutation test, shows that the difference in mean values is not significant (there are only nine subjects in each group and the variation in experience is within the bounds of chance; see rexample[ESEUR/communicating/postmortem-answers.R]).

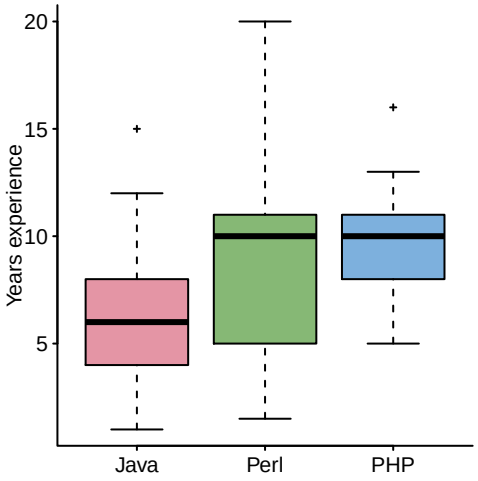


Figure 3.1: Years of professional experience in a given language for experimental subjects. Data from Prechelt.[462] code

3.2 Finding stories in data

You have some data. Perhaps you are expecting to see a particular pattern of behavior (e.g., an exponential relationship between two variables), perhaps you have some questions you would like answered or perhaps you would like to know what, if any, *interesting* stories the data has to tell.

This section covers initial data exploration. Examples of more detailed analysis that might be performed after a basic appreciation has been achieved are covered in subsequent chapters. The table of contents and index can be used to locate possible techniques to analyse whether your data contains the expected pattern of behavior.

If you have questions you would like answered, then you will need to translate the questions into expected patterns of behavior that can be extracted from the data. Of course there is never any guarantee that the data contains the information needed to answer any of the questions you have.

If you don't have specific questions, you can explore the data looking for commonly occurring patterns and then try to weave anything you find into a consistent story.

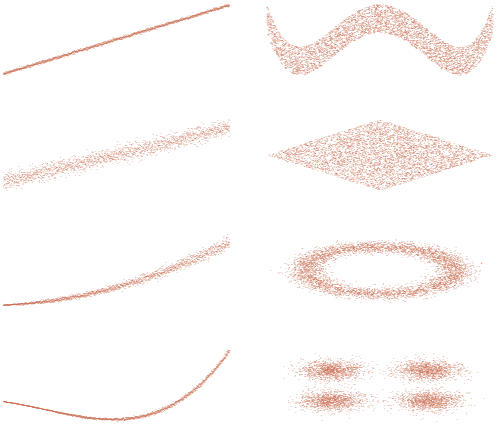


Figure 3.2: Plots of sample values having various visual patterns. code

Figure 3.2 shows some common and less commonly seen patterns in data. The left column shows data forming lines of various shapes; a straight line is perhaps the most commonly encountered pattern in data and points may all be close to the line or form a band of varying width. The right column shows data clustering together in various regular shapes. Uncovering a pattern is the next step along the path to understanding the processes that generated the sample measurements.

Interestingness is in the eye of the beholder. Reading through the data analysis examples in this book will give you some idea of the patterns that might be found in data and hopefully suggest some questions whose answer are of interest to you; if you still don't have a question to ask of your data and require an answer, then you might as well accept 42.

Compelling. Sometimes the numbers are so compelling that statistical analysis seems unnecessary. For instance, relative spacing is sometimes used within the visible form of expressions to highlight the relative precedence of binary operators (e.g., more whitespace around the addition operator when it appears adjacent to a multiplication, e.g., `5 + 2*3`). Table 3.1 shows that when relative spacing is used it nearly always occurs in a form that gives the operator with higher precedence greater proximity to its operands (relative to the operator having lower precedence). The number of cases where the reverse occurs is so small, that the either the developer who wrote the code did not know the correct relative precedence or there is a fault in the code.

	Total	High-Low	Same	Low-High
no-space	34866	2923	29579	2364
space no-space	4132	90	393	3649
space space	31375	11480	11162	8733
no-space space	2659	2136	405	118
total	73032	16629	41539	14864

Table 3.1: Number of expressions containing two binary operators having the specified spacing (i.e., no spacing, no-space, or one or more whitespace characters (excluding newline), space) between a binary operator and both of its operands. High-Low are expressions where the first operator of the pair has the higher precedence, Some are expressions where the both operators of the pair have the same precedence, Low-High are expressions where the first operator of the pair has the lower precedence. For instance, x + y*z is space no-space because there are one or more space characters either side of the addition operator and no-space either side of the multiplication operator, the precedence order is Low-High. Based on the visible form of the .c files. Data from Jones.[299]

A study by Landy and Goldstone[345] found that subjects were more likely to give the correct answer (and answer more quickly) to simple arithmetic expressions when there was greater visual proximity between the operands separated by the binary operator having the higher precedence.

3.2.1 Initial data exploration

Initial data exploration starts with the very messy subject of how data is formatted (lines containing a fixed number of delimited values is the ideal form because lots of tools can accept this as input, if a database is provided it may be worth extracting the required data into this form).

A programmer's text editor is probably the best tool for an initial look at data, unless the filename suggests it is a known binary format (such as a spreadsheet or database). For data held in spreadsheets exporting the required values to a csv file is often the simplest solution.

This initial look at the data will reveal some characteristics of the values measured, such as:

- number of measurement points (often the number of lines) and number of attributes measured (often the number of columns),
- what kind of attributes have been recorded (e.g., date, time, lines of code, language, cost estimated, email addresses, etc),
- the range of values taken by each attribute, i.e., minimum/maximum values (the R functions `min`, `max` and `range` process numeric or character vectors),
- any unexpected or missing values,
- duplicate or very similar rows or columns.

One call to `read.csv` will read the entire contents of a text file into a data frame (what R calls a structure or record type). The file is assumed to contain rows of delimited values (there is an option to change the default delimiter); spurious characters or missing values can cause values to appear in the wrong column. The data cleaning chapter provides some suggestions for finding and correcting problems such as this. The `foreign` package contains functions for reading data stored in a variety of proprietary binary forms.

Having read the file into a variable the following functions are useful for exploring what has been read (unless the dataset is small enough to be displayed on a screen in its entirety):

- `str` returns type information about its argument (most usefully the names, types and first few values of each column in a data frame),
- `NROW` and `NCOL` return the number of rows and columns in a vector or data frame (`nrow` and `ncol` are also available but do not behave sensibly when the argument is a vector),
- `head` and `tail` print six rows from the start/end of their argument respectively,
- `table` prints a count of the number of occurrences of each value in its argument, e.g., a particular column of a data.frame (by default NAs are not included).

When one or two columns are of specific interest a call to `plot` can be used to quickly visualize the specific data of interest.

For instance, limits imposed by a system can have a significant impact on the usage characteristics of that system. The upper plot in Figure 3.3 shows a very noticeable change in the distribution pattern of C source line length (i.e., number of characters on a line). This change occurs at around the line length commonly supported by non-GUI non-flat screen terminals (these measurements were of C source that is over 10 years old, i.e., before flat screen monitors became available) with wider modern displays and GUI editors the distribution of line lengths seen in more recently written code may be different.

The impact of external limits is not always immediately obvious. The number of tokens per line (lower plot) does not have any dramatic visible changes of behavior, but knowledge that average token length is around 3-4 characters provides a possible explanation for the slight change in the downward slope of the pluses just visible at around 25 tokens.

When a sample contains many variables, plotting one pair of variables at a time is an inefficient use of time. If passed a data frame containing three or more columns, `plot` creates nested plots showing the relationship between every pair of columns. Figure 3.4 shows four sets of related measurements; some measurement pairs appear to have a roughly linear relationship, while no obvious visual pattern is apparent for other pairs.

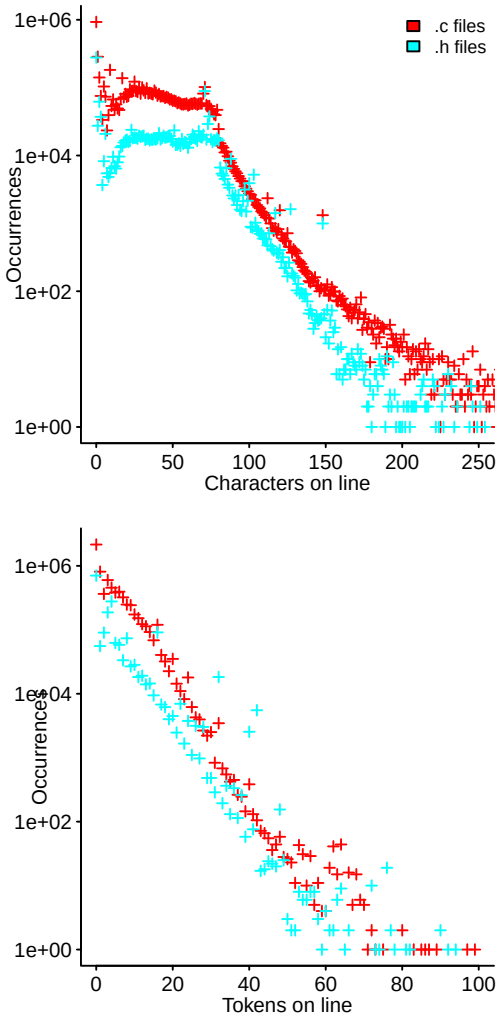


Figure 3.3: Total number of lines of C code, in .c and .h files, having a given length, i.e., containing a given number of characters (upper) and tokens (lower). Data from Jones.[299] code

```
work=read.csv(paste0(ESEUR_dir, "communicating/pub-fs-fp.csv.xz"),
                                                    as.is=TRUE)
# -1 removes the first column
plot(work[, -1], col=point_col, cex.labels=1.2)
```

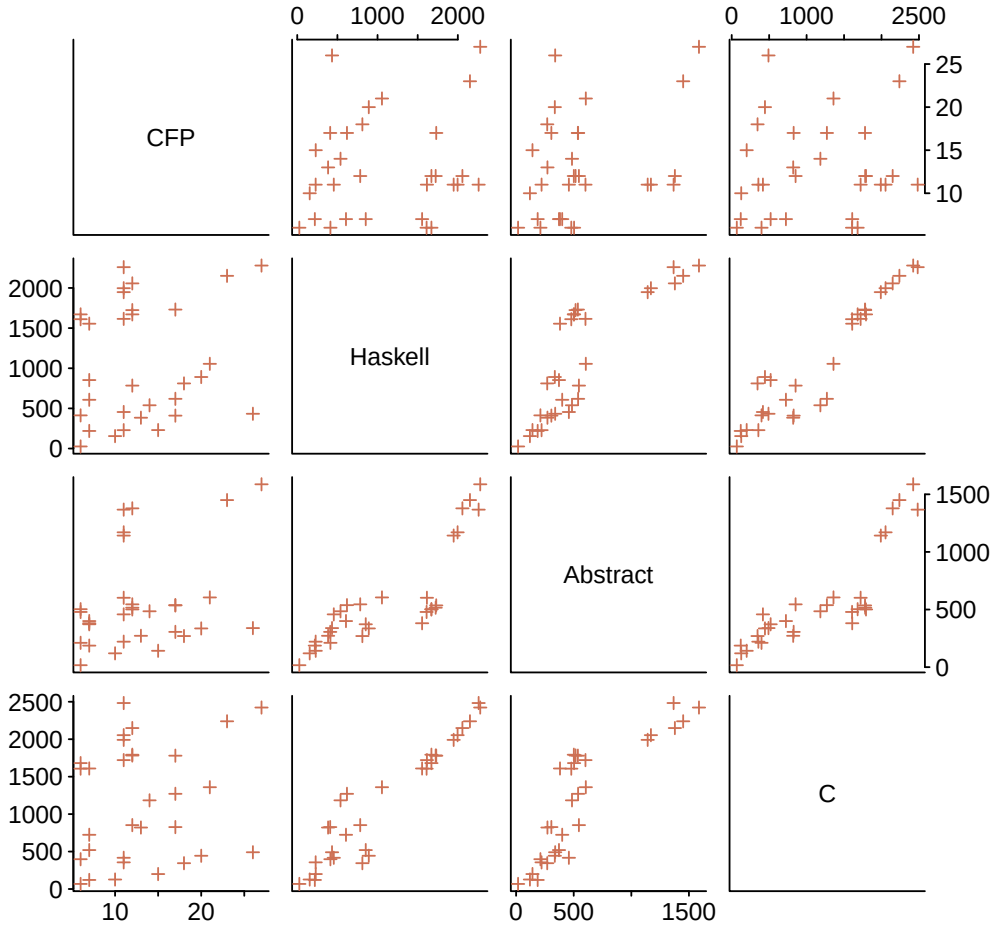


Figure 3.4: Various measurements of work performed implementing the same functionality, number of lines of Haskell and C implementing functionality, CFP (COSMIC function points; based on user manual) and length of formal specification. Data kindly provided by Staples.[542]
code

The `pairs` function supports a variety of options for producing more tailored visualizations of pairs of columns. Separating out and highlighting subsets of a sample (known as *stratifying*) can highlight interesting differences and similarities. Figure 3.5 separates out measurements of Ada and Fortran projects. The lines come from fitting points using loess, a simple regression modeling technique (see below and covered in more detail in a later chapter).

```
panel.language=function(x, y, language)
{
par(cex.axis=1.1)
Ada=(language == "Ada")
points(x[Ada], y[Ada], col=pal_col[2])
lines(loess.smooth(x[Ada], y[Ada], span=0.7), col=pal_col[2])

Fortran=(language != "Ada")
points(x[Fortran], y[Fortran], col=pal_col[1])
lines(loess.smooth(x[Fortran], y[Fortran], span=0.7), col=pal_col[1])
}

# rows 28 and 30 are zero and we only want columns 16:19
pairs(log(nasa[-c(28, 30) , 16:19]), cex.labels=1.3,
                panel=panel.language, language=nasa$language)
```

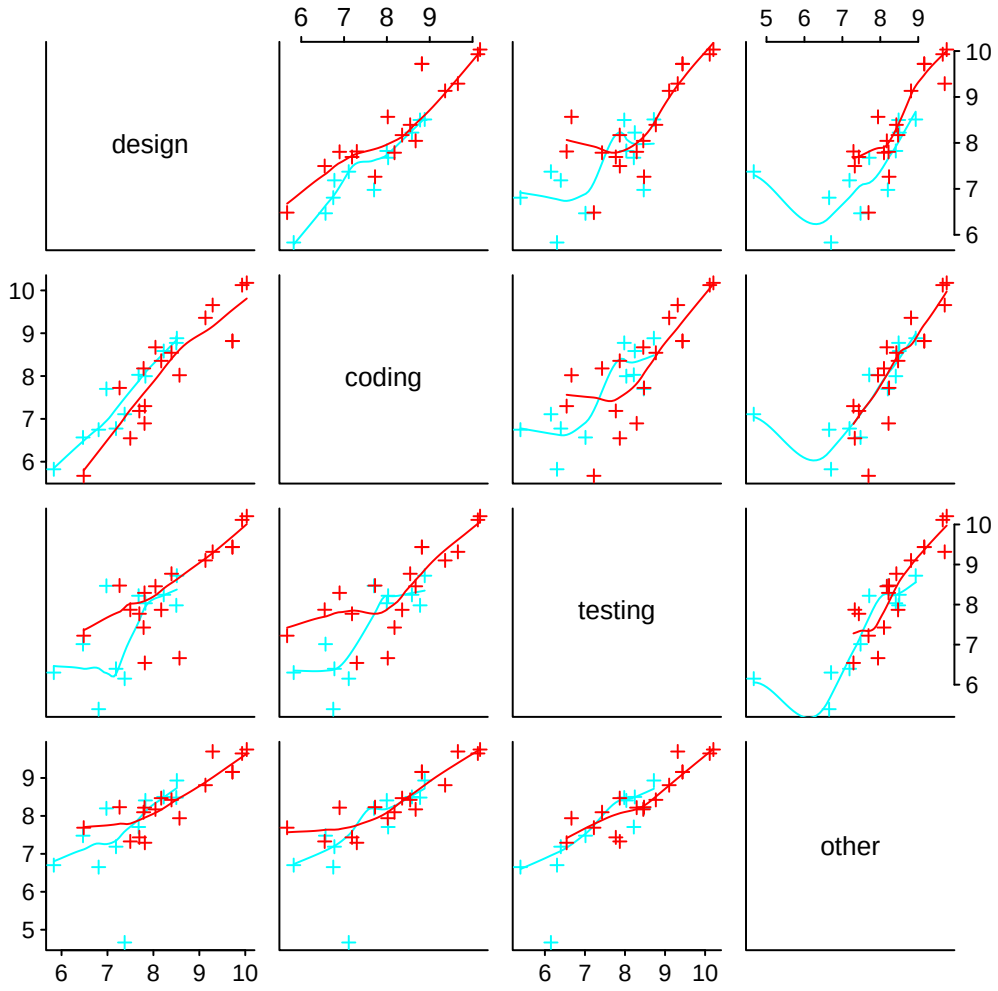


Figure 3.5: Effort, in hours (log scale), spent in various development phases of projects written in Ada (blue) and Fortran (red). Data from Waligora et al.[600] code

The default `pairs` plot contains redundant information; it is possible to display different information in the upper and lower halves of the plot, and along the diagonal. Figure 3.6 shows expert and novice performance (time taken to complete various tasks and final test coverage) in a test driven development task, with a boxplot along the diagonal and correlation between each pair of attributes, for the two kinds of subjects, in the lower half of the plot. This plot, which primarily uses the default values for its visual appearance, would need more work before being presented to customers.

```
panel_user=function(x, y, user)
{
expert=(user == "e")
points(x[expert], y[expert], col=pal_col[1])
points(x[!expert], y[!expert], col=pal_col[2])
}

panel_correlation=function(x, y, user)
{
expert=(user == "e")
r_ex=cor(x[expert], y[expert])
r_nov=cor(x[!expert], y[!expert])
txt = paste0("e= ", round(r_ex, 2), "\n", "n= ", round(r_nov, 2))
text(0.1, 0.5, txt, pos=4)
}

panel_boxplot=function(x, user)
{
t=data.frame(x, user)
boxplot(x ~ user, data=t, border=pal_col, add=TRUE)
}

pairs(tdd[ , c("duration.min", "changes", "TDD",
                "average.cycle.length", "development.cycle.length",
                "cycle", "line.coverage", "block.coverage")],
```

```
                upper.panel=panel_user, lower.panel=panel_correlation,
                                diag.panel=panel_boxplot, user=tdd$user)
```

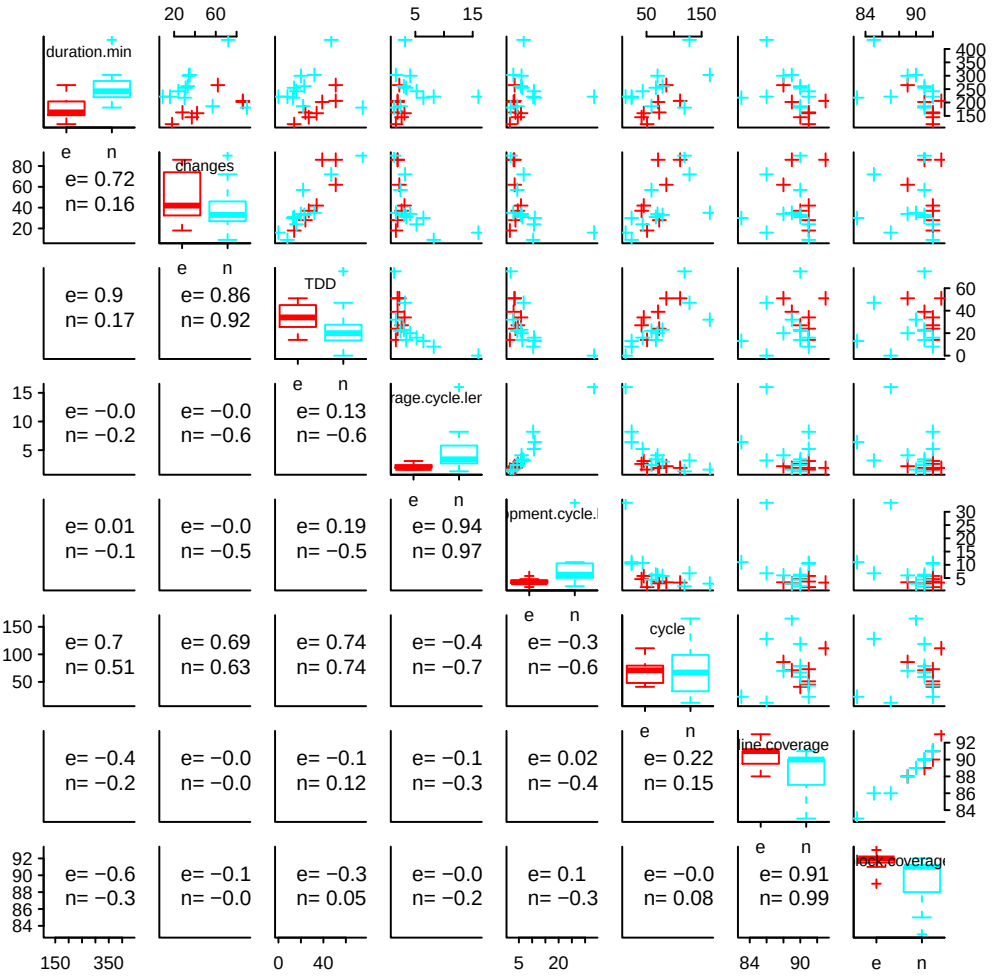


Figure 3.6: Performance of experts (e) and novices (n) in a test driven development experiment. Data from Muller et al.[402] code

The `splom` function in the `lattice` package allows more complex pair-wise plots to be created.

As the number of columns increases, the amount of detail visible in a `pairs` plot decreases. The `plotcorr` function in the `ellipse` package produces a visualization of the correlation between the values in pairs of columns and because correlation is a single value (the extent to which a linear relationship exists) it is possible to create a usable plot containing lots of columns (there are 27 in Figure 3.7). The correlation controls the color and eccentricity of the ellipse, with blue/right slant denoting a positive correlation and red/left slant a negative one.

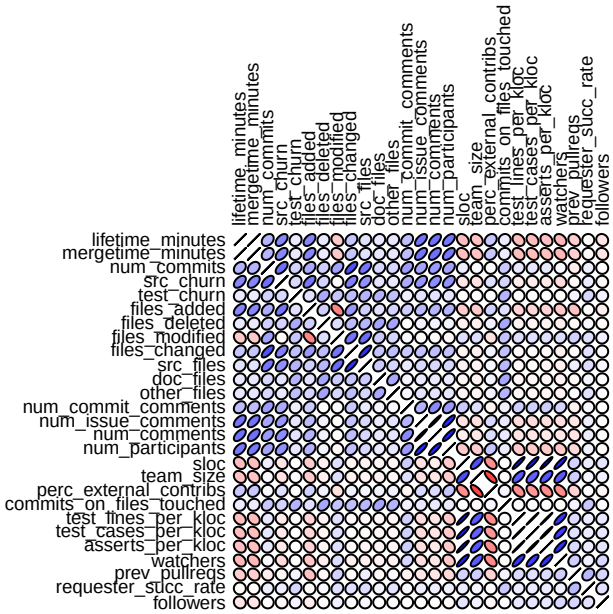


Figure 3.7: Correlations between pairs of attributes of 12,799 Github pull requests to the Homebrew repo, represented using colored ellipses. Data from Gousios et al.[221] code

```
library(ellipse)

# Cross correlation ellipses
ctab = cor(used, method = "spearman", use="complete.obs")

# Map the range 0..1 to colors
colorfun = colorRamp(c("#ff0000","white","#0000ff"), space="rgb")
plotcorr(ctab, col=rgb(colorfun((ctab+1)/2), maxColorValue=255),
                                  outline = FALSE, cex.lab=1.0)
```

The `corrgram` package supports a variety of functions for displaying correlation information. Figure 3.8 shows one alternative to displaying the data in Figure 3.7. Having variable names appear along the diagonal creates a compact plot; when many variables are involved this form of display is better suited to situations where variable identity follows a regular pattern.

```
library("corrgram")

corrgram(ctab, upper.panel=panel.pie, lower.panel=panel.shade)
```

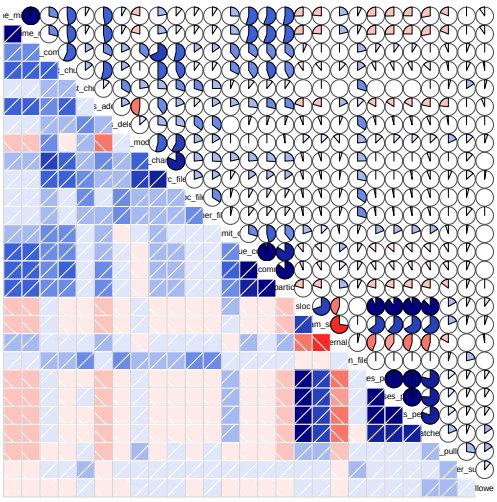


Figure 3.8: Correlations between pairs of attributes of 12,799 Github pull requests to the Homebrew repo, represented using pie charts abd shaded boxes. Data from Gousios et al.[221] code

In a sample containing multiple variables there will be varing degrees of similarity in the patterns of behavior shared by pairs of variables. Hierarchical clustering is one technique for arranging items such that those closer to each other, based on a user supplied distance metric, are closer to each other in the created hierarchy.

The `hclust` function is included in the base system and requires the user to handle the details; the `varclus` function in the `Hmisc` package provides a high level interface. In the following code `as.dist` is used to map the cross-correlation matrix returned by `cor` to a distance, see Figure 3.9::

```
ctab = cor(used, method = "spearman", use="complete.obs")

pull_dist=as.dist((1-ctab)^2)
t=hclust(pull_dist)
plot(t, main="", sub="", xlab="Pull related variables", ylab="Height\n")
```

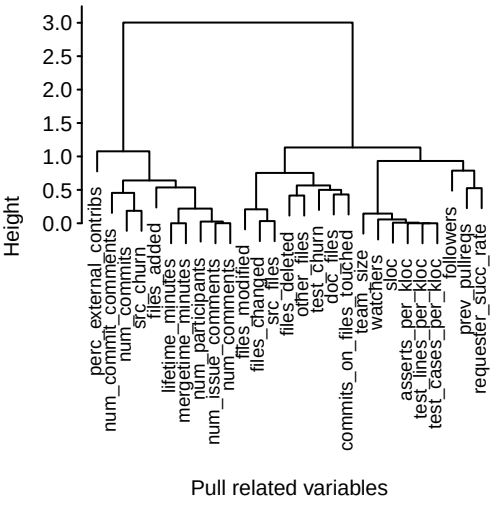


Figure 3.9: Hierarchical cluster of correlation between pairs of attributes of 12,799 Github pull requests to the Homebrew repo. Data from Gousios et al.[221] code

3.2.2 Guiding the eye through the data

When looking at a plot of a collection of points, it is not always possible to reliably estimate the path of line through them (according to some goodness of fit criteria). One way to quickly obtain a rough idea of the likely trajectory of such a line is to use the `loess.smooth` function (loess is discussed later). Including such a fitted line in a plot is a way of visually showing that expectations of a pattern of behavior is being followed (or not). Figure 3.10 shows a loess

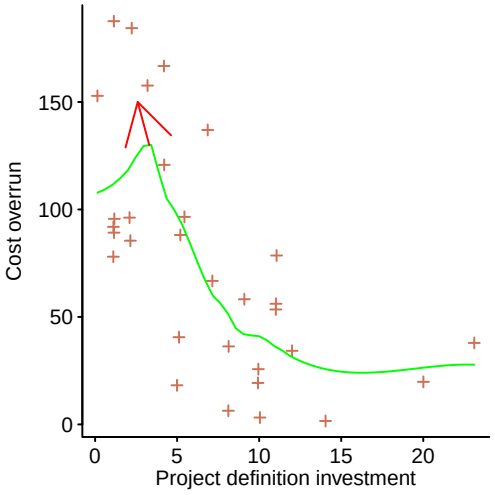


Figure 3.10: Effort invested in project definition (as percentage of original estimate) against cost overrun (as percentage of original estimate). Data extracted from Gruhl.[236] code

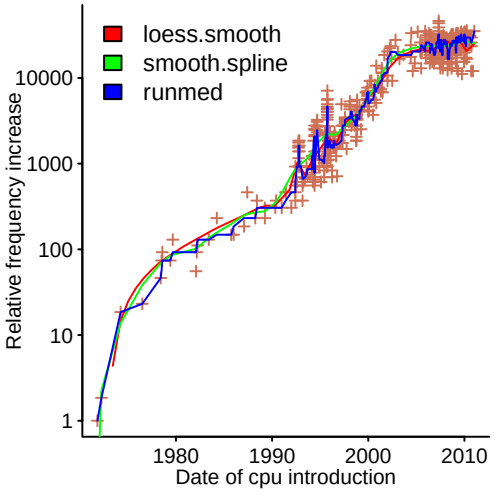


Figure 3.11: Relative clock frequency of cpus when first launched (1970 == 1). Data from Danowitz et al.[131] code

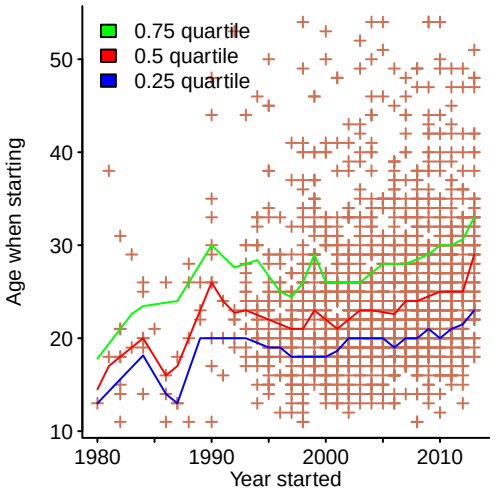


Figure 3.12: Year and age at which survey respondents started contributing to FLOSS, i.e., made their first FLOSS contribution. Data from Robles et al.[486] code

fit (green) to NASA data[236] on cost overruns for various space probes against effort invested in upfront project definition; the upward arrow shows the continuing direction of the line seen in the original plot created by NASA users of this data (who are promoting a message that less investment is always bad).

There are a variety of techniques for calculating a smooth line that is visually less noisy than drawing a line through all the points. Splines are invariably suggested in any discussion of fitting a smooth curve to an arbitrary set of points; the `smooth.spline` function will fit splines to a series of points and return the x/y coordinates of the fitted curve.

Splines originated as a method for connecting a sequence of points by a visually attractive smooth curve, not as a method of fitting a curve that minimises the error in some measurement. LOESS is a regression modeling technique for fitting a smooth curve that minimises the error between the points and the fitted curve; the `loess.smooth` function fits a loess model to the points and return the x/y coordinates of the fitted curve.

Both splines and loess can be badly behaved when asked to fit points that include extreme outliers or have regions that are sparsely populated with data. The running median (e.g., `median(x[1:k])`, `median(x[(1+1):(k+1)])`, `median(x[(1+2):(k+2)])` and so on for some `k`) is a smoothing function that is very robust to outliers; the `runmed` function calculates the running median of the points and return these values (the points need to be in sequential order).

Figure 3.11 shows the maximum clock frequency of cpus introduced between 1971 and 2010; the various lines were produced using the values returned by the `smooth.spline`, `loess.smooth` and `runmed` functions. Don't be lulled into a false sense of security by the lines looking very similar, the *smoothing parameter* provided by each function was manually selected to produce a pleasing looking fit in each case; the mathematics behind the functions can produce curves that look very different and the choice of function will depend on the kind of curve required and perhaps be driven by the characteristics of the data.

```
plot(x_vals, y_vals, log="y", col=point_col,
                        xlab="Date of cpu introduction",
                        ylab="Relative frequency increase\n")
lines(loess.smooth(x_vals, y_vals, span=0.05), col=pal_col[1])

# smooth.spline and runmed don't handle NA
t=!is.na(x_vals) ; x_vals=x_vals[t] ; y_vals=y_vals[t]
t=!is.na(y_vals) ; x_vals=x_vals[t] ; y_vals=y_vals[t]

lines(smooth.spline(x_vals, y_vals, spar=0.7), col=pal_col[2])

t=order(x_vals)
lines(x_vals[t], runmed(y_vals[t], k=9), col=pal_col[3])
```

Lines drawn through a sample of measurements values often follow the path specified by some central location metric, e.g., the mean value. In more cases it may be more informative to fit a line such that 25% of the measurements are below/above it, or some other percentage (the fitting processing is known as quantile regression). Figure 3.12 is based on 2,183 replies from a survey of FLOSS developers;[486] two of the questions being the year and age at which those responding first contributed to FLOSS.

If you find yourself writing lots of algorithmic R code during initial data exploration, you are either investing too much effort in one area or you have found what you are looking for and have moved past initial exploration.

3.2.3 Smoothing data

Sample values sometimes fluctuate widely around a general trend (the data is said to be *noisey*). Smoothing the data can make it easier to see a pattern in the clutter of measured values. The traditional approach is to divide the range of measurement values into a sequence of fixed width bins and count the number of data points in each bin; the plotted form of this binning process is known as a *histogram*.

Histograms have the advantage of being easy to explain to people who do not have a mathematical background and widespread usage means they are likely to have been encountered

before. Until the general availability of computers, histograms also had the advantage of keeping the human effort needed to smooth data within reasonable bounds.

In the upper plot of Figure 3.13 shows the number of computers having the same SPECint result value, in the lower plot the data has been aggregated into 13 fixed width bins (the number of bins selected by the `hist` function for this data).

The `hist` function accepts a vector of values, automatically selects a bin width and plots a histogram; there are options to change the number of bins and to explicitly specify the location of each bin boundary (which allows variable width bins to be supported, useful when a logarithmic scale is used and bin widths have to follow a geometric progression). The `histogram` package offers automatically support for a wider range of functionality and more optional support that is available in the base system functions. When plotting is not required the `cut` function can be used to divide the range of its argument into intervals and return the bounds of the intervals and the corresponding counts in each interval.

When dealing with measurements that span several orders of magnitude, a log scale is often used. Creating a histogram using a log scale requires the use of bin widths that grow geometrically (coding is needed to get the `hist` function to use variable width bins; the `histogram` package contains built-in support for this functionality) and bin contents has to be expressed as a density (rather than a count). A histogram based on counts, rather than density, can produce misleading results; see rexample[slash_hist.R].

The advantage of the binning approach to aggregating data is that it is an easy to perform manual task and for this reason it has a long history. The disadvantages of histograms are: 1) changing the starting value of the first bin can dramatically alter the visual outline of the created histogram and 2) they do not have helpful mathematical properties.

A technique that removes the arbitrariness of histogram bins' starting position is averaging over all starting positions, for a given bin width (known as a *average shifted histogram*); this is exactly the effect achieved using kernel density with a rectangular kernel function.

It often makes sense for the contribution made by each value to be distributed across adjacent measurement points, with closer points getting a larger contribution than those further away. This kind of smoothing calculation is too compute intensive to be suited to manual implementation, but is trivial when a computer is available.

The distribution of values across close measurement points is known as *kernel density*; histograms are the manual labourer's poor approximation to Kernel density, if a computer is available use the better technique.

The `density` function returns a kernel density estimate (which can be passed to `plot` or `lines`; the following code produced Figure 3.14).

```
plot(density(cint$Result))
```

Density plots also perform well when comparing rapidly fluctuating measurements of related items. Figure 3.15 shows the number of commits of different lengths (in lines of code; upper) to the Linux filesystem code, for various categories of changes and the lower plot shows a density plot of the same data.

The kernel density approach generalizes to more than one dimension; see the `KernSmooth` and `ks` packages.

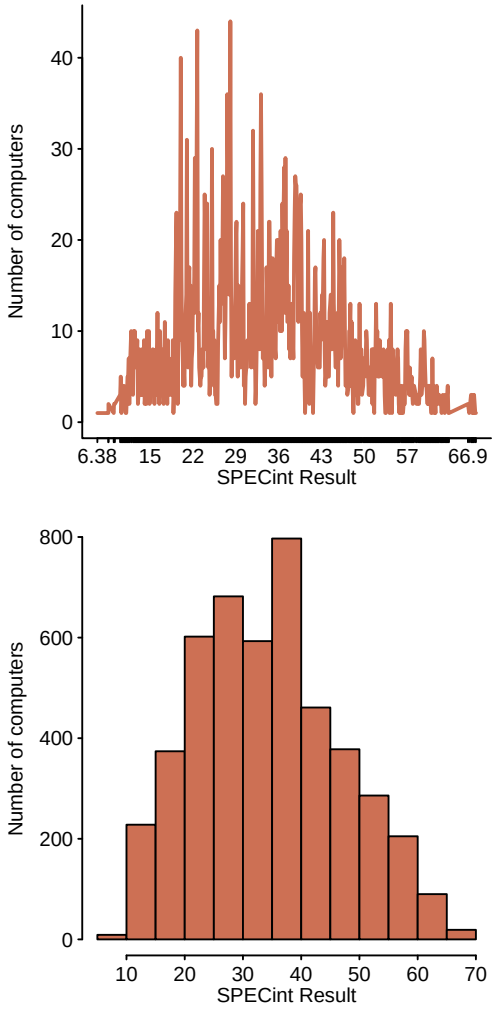


Figure 3.13: SPECint results, summed over all distinct values (upper) and summed within equal width bins (lower). Data from SPEC website.[534] code

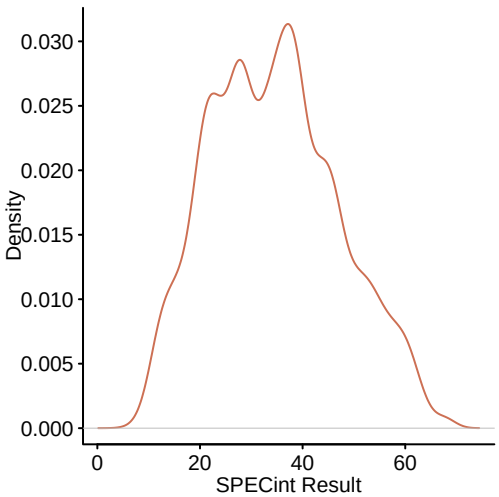
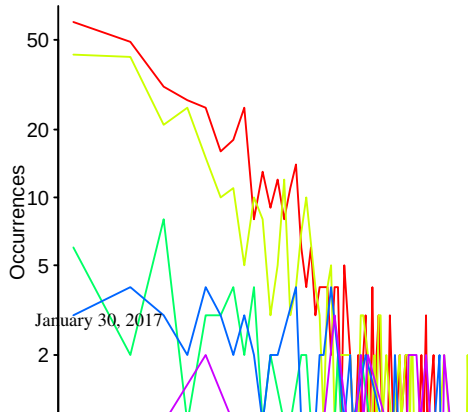


Figure 3.14: Kernel density plot of the number of computers having the same SPECint result. Data from SPEC.[534] code

3.2.4 Densely populated measurement points

Some samples contain data having characteristics that are result in plots containing lots of ink and little visual information. Common characteristics of the density of values, such as the following:

- adjacent values on the x-axis having widely different values on the y-values, e.g., the SPECint in the upper plot of Figure 3.13,
- multiple points having the same x/y, visually value appearing as a single value,
- many very similar values that merge into a formless blob when plotted.



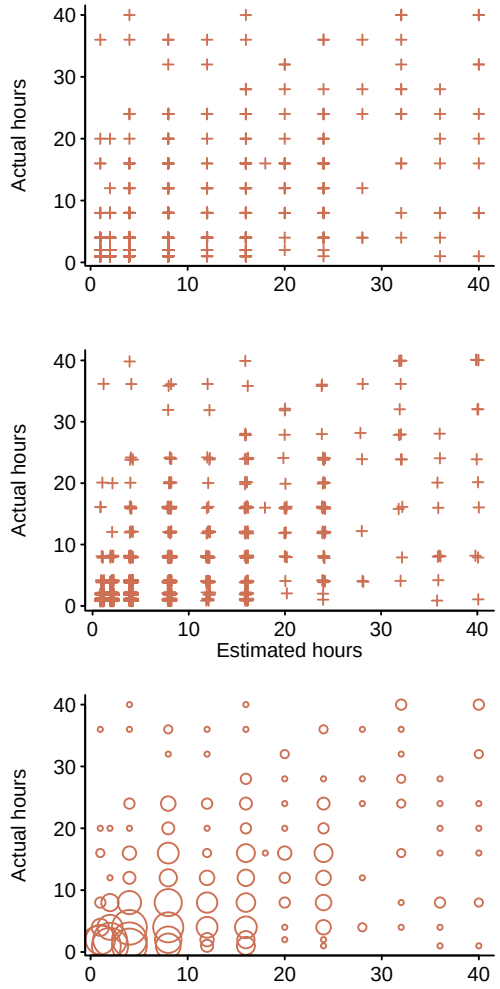


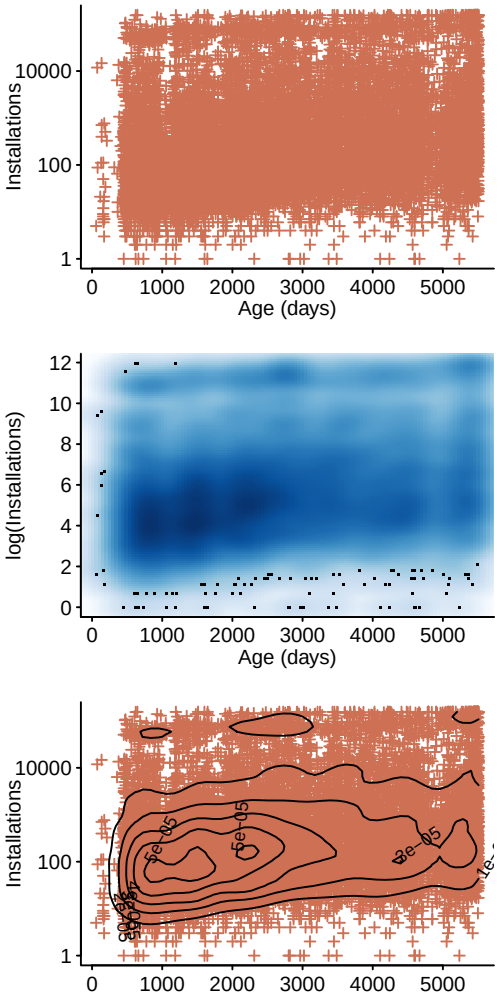
Figure 3.16: Developer estimated effort against actual effort (in hours), for various maintenance tasks, e.g., adaptive, corrective and perfective; upper as-is, middle jittered values and lower size proportional to the log of the number measurements. Data from Hatton.[248] code

A plot of discrete values may give a misleading impression because what appears to be single points may actually be multiple measurements having the same values (see top plot in Figure 3.16). The `jitter` function returns its argument with a small amount of added random noise; the middle plot of Figure 3.16 shows the effect of jittering the data seen in the upper plot. Another possibility is for the size of the symbol used to vary with the number of measurements at a given point (lower plot of Figure 3.16); as discussed elsewhere, people are poor at estimating the relative area and so size should not be treated as anything more than a rough indicator.

```
plot(maint$est_time, maint$act_time, col=point_col, xlab="",
                                        ylab="Actual hours\n")

plot(jitter(maint$est_time), jitter(maint$act_time), col=point_col,
                xlab="Estimated hours", ylab="Actual hours\n")

library("plyr")
t=ddply(maint, .(est_time, act_time), nrow)
plot(t$est_time, t$act_time, cex=log(1+t$V1), pch=1, col=point_col,
                                    xlab="", ylab="Actual hours\n")
```

A different kind of problem occurs when data points are so densely packed together that any patterns which might be present are hidden by the fog (upper plot in Figure 3.17). One technique for uncovering patterns in what appears to be a uniform color surface is to display the density of points. The `smoothScatter` function calculates a kernel density over the points to produce a color representation (middle plot), or contour lines can be drawn with `contour` using the 2-D kernel density returned by `kde2d` (lower plot).

```
plot(udd$age, udd$insts, log="y", col=point_col,
                xlab="Age (days)", ylab="Installations\n")
# Bug in support for log argument :-(
smoothScatter(udd$age, log(udd$insts),
                xlab="Age (days)", ylab="log(Installations)\n")

library("MASS")

plot(udd$age, udd$insts, log="y", col=point_col,
        xlab="Age (days)", ylab="Installations\n")

# There is no log option, so we have to compress/expand ourselves.
d2_den=kde2d(udd$age, log(udd$insts+1e-5), n=50)
contour(d2_den$x, exp(d2_den$y), d2_den$z, nlevels=5, add=TRUE)
```



The `hexbin` package is available for those who insist on putting values into bins, in this case using hexagonal binning to support two dimensions.

One solution to a high density of point in a plot is to stretch the plot over multiple lines; the `xyplot` function, in the `lattice` package, can produce a strip-plot such as the one in Figure 3.18.

```
library("lattice")
library("plyr")

cfl_week=ddply(cfl, .(week),
                function(df) data.frame(num_commits=length(unique(df$commit)),
                                        lines_added=sum(df$added),
                                        lines_deleted=sum(df$removed)))

# Placement of vertical strips is sensitive to the range of values
# on the y-axis, which may have to be compressed, e.g., sqrt(...).
t=xyplot(lines_added ~ week | equal.count(week, 4, overlap=0.1),
                cfl_week, type="l", aspect="xy", strip=FALSE,
                xlab="", ylab="Weekly total",
                scales=list(x=list(relation="sliced", axs="i"),
                            y=list(alternating=FALSE, log=TRUE)))
plot(t)
```

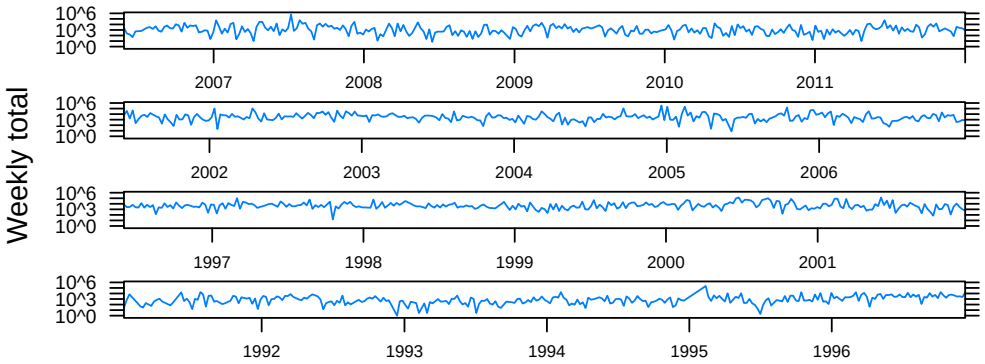


Figure 3.18: Number of lines added to glibc each week. Data from González-Barahona et al.[212] code

3.2.5 Visualizing the distribution of values

A *box-and-whiskers* plot (or *boxplot* as it is more generally known) provides a simple visualization of the distribution the values in a sample (see Figure 3.19). The following characteristics are highlighted:

- median, i.e., the point that divides the number of values in half,
- first/third or lower/upper quartile, the 25th/75th percentiles respectively,
- lower/upper hinges, the points at a distance $\pm 1.5 \times IQR$ where IQR is the interquartile range (the difference between the lower quartile and the upper quartile). The dotted line joining the hinges to the quartile box are the whiskers,
- outliers, all points outside the range of the lower/upper hinge.

The `boxplot` function produces a boxplot and passing the argument `notch=TRUE` creates a plot that includes a *notch* indicating the 95% confidence interval of the median (right plot in Figure 3.19).

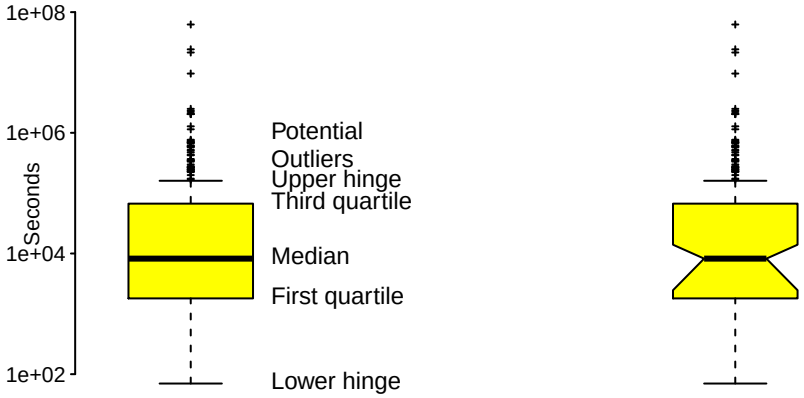


Figure 3.19: Boxplot of time between a bug in Eclipse being reported and the first response to the report; right plot is notched. Data from Breu et al.[75] code

```
box_inf=boxplot(eclipse_rep$min.response.time, log="y",
                boxwex=0.25, col="yellow", yaxt="n",
                notch=TRUE, xlim=c(0.9, 1.3), ylab="")
```

The ideas behind the boxplot and kernel density can be combined to create what is known as a *violin plot*. The curve in Figure 3.20 is based on the kernel density of the data points.

The `vioplot` function in the `vioplot` package has limited functionality and for non-trivial displays needs to be used in conjunction with a previous call to, say, the `plot` function. The `beanplot` function in the `beanplot` package supports a lot of functionality, of which producing the envelope of a violin plot is one item.

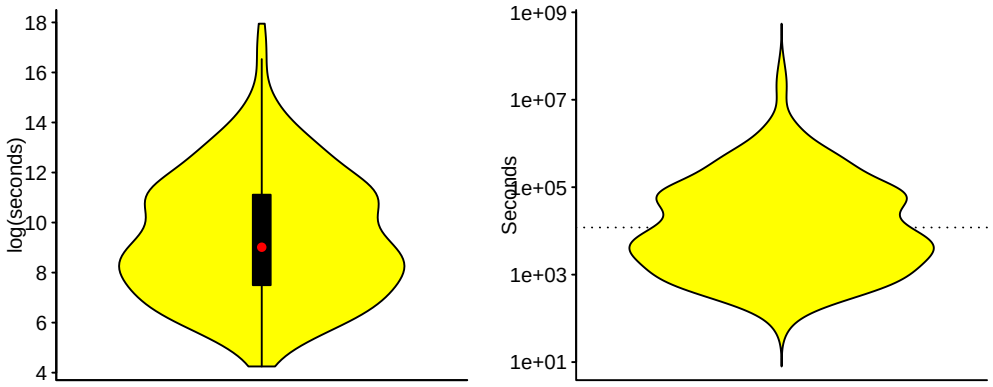


Figure 3.20: Violin plots (left using `vioplot`, right using `beanplot`) of time between bug being reported in Eclipse and first response to the report. Data from Breu et al.[75] code

```
library("vioplot")

# vioplot raises error for attempts to specify axis labels.
# So use plot to control layout and use add=TRUE in vioplot.
# log="y" produces weird results.
plot(x=0.1, xlab="", ylab="log(seconds)", xaxt="n",
       xlim=c(0.2, 1.8),
       ylim=range(log(eclipse_rep$min.response.time)))

vioplot(log(eclipse_rep$min.response.time), col="yellow",
                        colMed="red", wex=1.5, add=TRUE)

library("beanplot")

beanplot(eclipse_rep$min.response.time, col="yellow", log="y",
                        what=c(1, 1, 0, 0), ylab="Seconds\n")
```

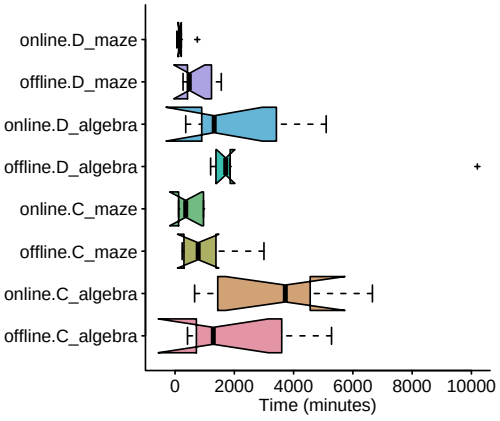


Figure 3.21: Time taken for developers to debug various programs using batch processing or online (i.e., time-sharing) systems. Data kindly provided by Prechelt.[461] code

The `boxplot` function supports multiple, related, boxplots in the same plot using formula notation, with the following code producing Figure 3.21 (portions of the notch are offset to avoid obscuring the hinge):

```
boxplot(time ~ group+task, data=gs, notch=TRUE, horizontal=TRUE,
                                        xlab="Time (minutes)")
```

A bar chart with error bars is regularly used to visually summarise values (sometimes known as *dynamite plots*). A study[124] investigating the effectiveness of various ways of visually summarizing data (including boxplots, violin plots and others) found that when extracting information from bar charts (with or without error bars) subjects did not perform as well as they did when using the other techniques.

3.2.6 Relationships between items

The relationship between two entities may be the data attribute of interest. The data structure commonly associated with relationships is the graph. The `igraph` package contains numerous functions for processing graphs.

When displaying graphs containing large numbers of nodes, potentially useful information in the visual presentation can be swamped by many nodes having relatively few connections. Figure 3.22 is an attempt to show which languages commonly occur, in the same project, with another language, in a sample of 100,000 GitHub projects. The number of projects making use of a given pair of languages is represented using line width and to stop the plot being an amorphous blob the color and transparency of lines also changes with number of occurrences.

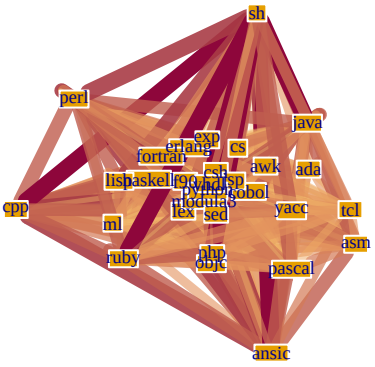


Figure 3.22: Pairs of languages used together in the same GitHub project with connecting line width, color and transparency related to number of occurrences. Data kindly supplied by Bissyande.[62] code

Perhaps those nodes having relatively few connections are the ones of interest. The Microsoft Server protocol specification[621] contains over 16 thousand pages across 130 documents (the client specification documents are also numerous). The upper plot in Figure 3.23 shows dependencies between the documents (based on cross-document references in the 2009 release[621]). The lower plot shows the dependencies after excluding the 18 most often referenced documents.

```
library("igraph")
library("sna")

interest_gr=graph.adjacency(interest, mode="directed")

# V(interest_gr)[names(in_deg)]$size=3+in_deg^0.7
V(interest_gr)$size=1
# V(interest_gr)$color=brew_col[4]
V(interest_gr)$label.cex=0.75
E(interest_gr)$arrow.size=0.3

plot(interest_gr)
```

Depending on the question being asked, either the identity of the most frequently referenced documents or the dependencies between those remaining after these have been removed may be included in the story communicated.

It is possible to use R to draw presentable graphs, however, if your primary interest is drawing visually attractive graphs containing lots of information, then there other systems that may be easier to use (e.g., GraphViz[225]). Yes, an R interface to these systems may be available, but if statistical analysis is not the primary purpose, why is R being used?

Alluvial plots are a technique for visualizing the flow between connected entities. Figure 3.24 shows factors used to prioritize the application of Github pull requests and the relative orders in which they appear in a dataset of pull requests.[222]

3.2.7 3-dimensions

Three dimensions is only one more than two dimensions and various techniques for enhancing a flat surface to display information about one more dimension (i.e., measurements of a new attribute) are available.

Heatmaps are a technique that use of color to display information about a third quantity within a 2-D plot. Figure 3.25 shows the L3 cache bandwidth of an Intel Sandy Bridge processor when running at various clock frequencies and using various combinations of cores.

Both the `heatmap` function in the base system and `heatmap.2` function in the `gplots` package clusters the rows/columns and display a dendrogram; various arguments have to be set to switch off this default behavior, with `heatmap` doing its best to make life difficult including not coexisting with other plots in the same image; `heatmap.2` is more reasonable.

The `levelplot` function in the `lattice` package provides straightforward functionality for producing heat maps and it is used to produce all the heatmaps in this book.

```
library("lattice")

t=levelplot(L3_band,
                col.regions=rainbow(100, end=0.9),
                xlab="Clock frequency (Mhz)", ylab="Cores used",
                scales=list(x=list(cex=0.70, rot=35),
                            y=list(cex=0.65)),
                panel=function(...)
                        {
                        panel.levelplot(...)
                        panel.text(1:11, rep(1:8, each=11),
                                        L3_band, cex=0.55)
                        })

plot(t, panel.height=list(3.8, "cm"), panel.width=list(6.2, "cm"))
```

A contour plot can be used for visualizing the relationship between a response variable and two explanatory variables; the `contour` function is part of the base system.

A study by Thereska, Doebel, Zheng and Nobel[568] measured the performance of various applications running on a variety of desktop computers; the cpu speed and memory capacity of the computer hosting each of the 4,924,467 user sessions was recorded. The contours in Figure 3.26 are derived from the number of user sessions measured on a computer having a given processor speed and memory capacity.

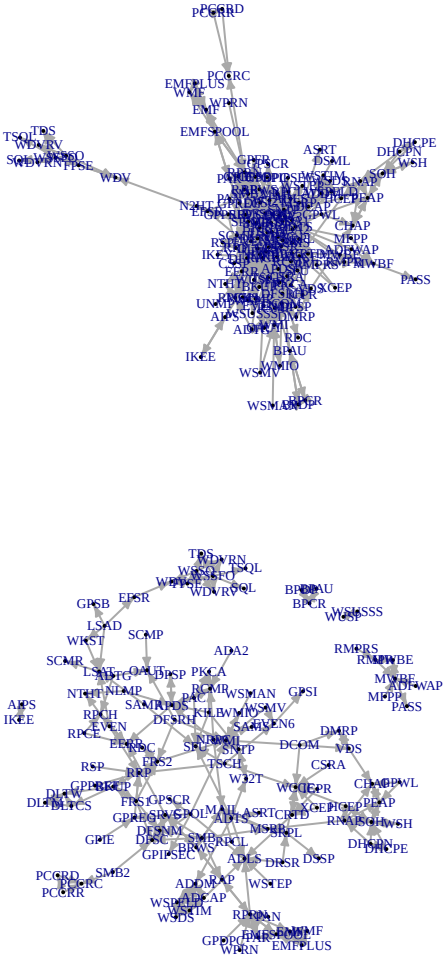


Figure 3.23: References from one document to another in the Microsoft Server Protocol specifications. Data extracted by the author from the 2009 document release.[621] code

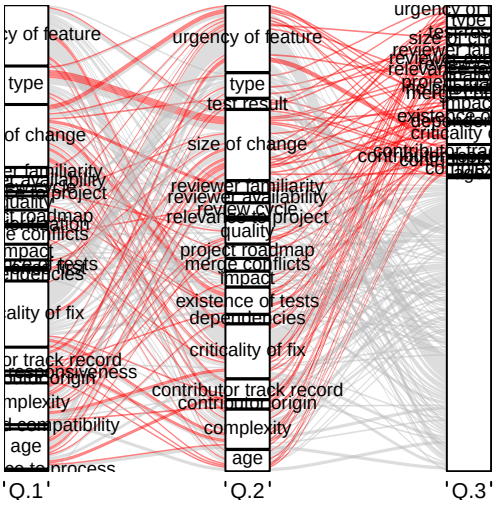


Figure 3.24: Alluvial plot of relative prioritization order of selection and application of Github pull requests. Data from Gousios et al.[222] code

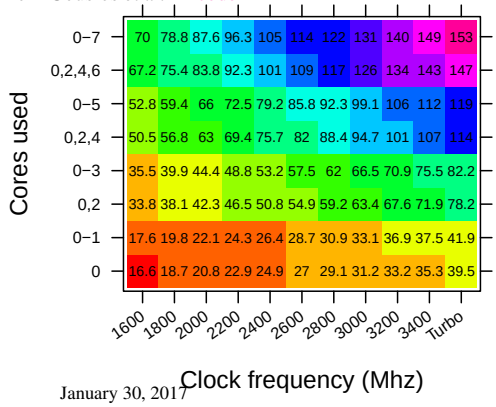


Figure 3.25: Intel Sandy Bridge L3 cache bandwidth in GB/s at various clock frequencies and using combinations of cores (0-3 denotes cores zero-through-three, 0,2,4 de-

```
library("plyr")

Um=unique(memcpu$MemorySize)
M_map=mapvalues(memcpu$MemorySize, from=Um, to=rank(Um))

Us=unique(memcpu$ProcSpeed)
S_map=mapvalues(memcpu$ProcSpeed, from=Us, to=rank(Us))

cnt_mat=matrix(data=0, nrow=length(Us), ncol=length(Um))

cnt_mat[cbind(S_map, M_map)]=log(memcpu$Session_Count)

contour(x=seq(min(Us)/max(Us), 1, length.out=length(Us)),
        y=seq(min(Um)/max(Um), 1, length.out=length(Um)),
        z=cnt_mat, col=pal_col, nlevels=10, axes=FALSE,
        xlim=c(min(Us)/max(Us), 1), ylim=c(min(Um)/max(Um), 1),
        xlab="Processor speed (GHz)",
        ylab="Memory size (Mbyte)\n")

axis(1, at=sort(Us)/max(Us), labels=sort(Us))
axis(2, at=sort(Um)/max(Um), labels=sort(Um))
```

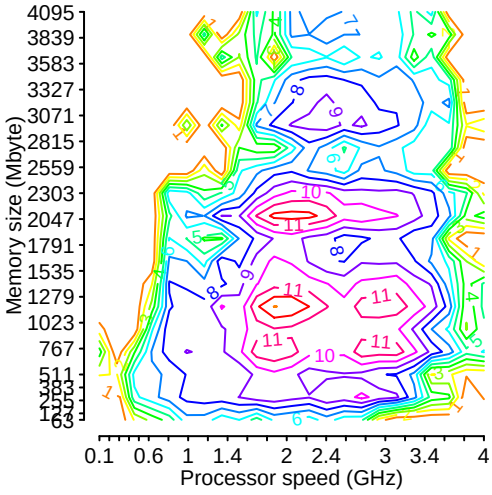


Figure 3.26: Contour plot of the number of sessions executed on a computer having a given processor speed and memory capacity. Data kindly provided by Thereska.[568] code

A variety of functions are available for representing a 3-D plot as a 2-D plot, including the `scatterplot3d` function in the `car`, the `plot3d` function in the `rgl` package.

The `plot.design` function can be used to display the effects of each factor on the mean value of the response variable (see Figure 8.7 in the experiment chapter).

Histograms in 3-dimensions provide more opportunities than histograms in 2-dimensions for looking impressive with little data and misleading viewers. As such, they can be a useful visualization technique.

A study by Hamill and Goseva-Popstojanov[23] investigated the origin of 1,257 faults in 21 large safety critical applications, recording where the fixes were made (e.g., requirements, design, code or supporting files). Figure 3.27 shows a 3-D histogram of root cause/fix location on the x-y axis and a count of occurrences on the z-axis. Color has the effect of enhancing the visual appeal of the plot and makes it easier to locate pairs having similar values, but it is very difficult to obtain detailed information from this plot. Adding numeric values would provide detail, but the real issue is what information is the plot intended to communicate?

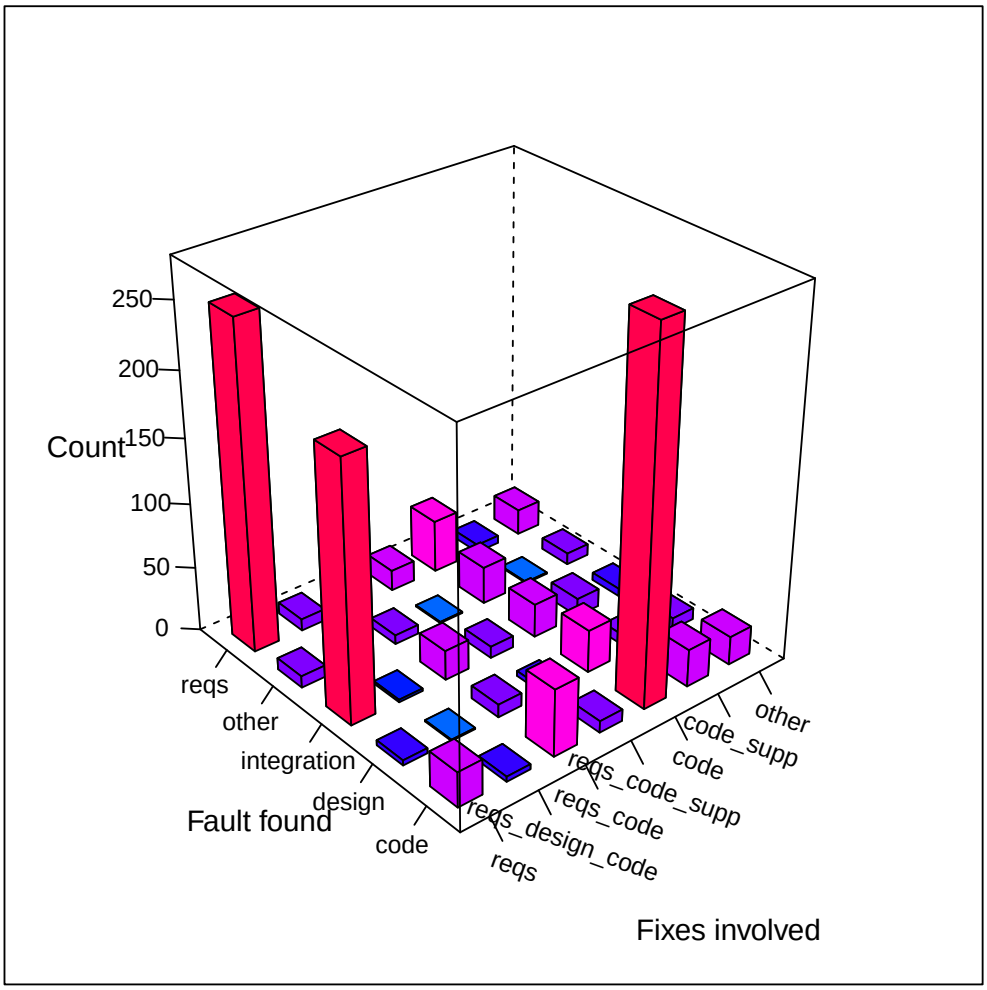


Figure 3.27: Root source of 1,257 faults and where fixes were applied for 21 large safety critical applications. Data from Hamill et al.[23] code

```
library("lattice")
library("latticeExtra")

# log transform pulls out small differences in majority of counts
transform_breaks= exp(do.breaks(range(log(1e-4+STVR_col$occurrences)), 20))
t=cloud(occurrences ~ fix+fault, STVR_col,
                panel.3d.cloud=panel.3dbars,
                xlab="Fixes involved", ylab="Fault found", zlab="Count",
                xbase=0.5, ybase=0.5, aspect=c(1, 1),
                col.facet = level.colors(STVR_col$occurrences,
                                         at = transform_breaks,
                                         col.regions = rainbow),
                scales=list(arrows=FALSE, distance=c(2, 1.1, 1),
                            x=list(rot=-20) # Rotate tick labels
                       ))
plot(t)
```

A ternary, or triangle, plot has three axes. The axes are inclined at an angle of 60°, rather than 90°, to each other, and some effort is needed to work out the coordinates of any point. Figure 3.28 shows two ways of labelling a ternary plot (with the three coordinates summing to 100%), with labels appearing at the vertex rather than along the axis and axis scales drawn either perpendicular to the axis or labeled along the axis and within the triangle as a grid. The upper plot shows how lines perpendicular to the appropriate axis are used to find the location of a point (at 10, 35, 55 in this case).

Points appearing close to a vertex have a higher

The closer points are to a vertex the larger the value of the corresponding variable, the closer points are to an axis the smaller the value of the corresponding variable.

Ternary plots are used to visualize compositional data. The `compositions` and `vcd` packages include support for creating ternary plots.

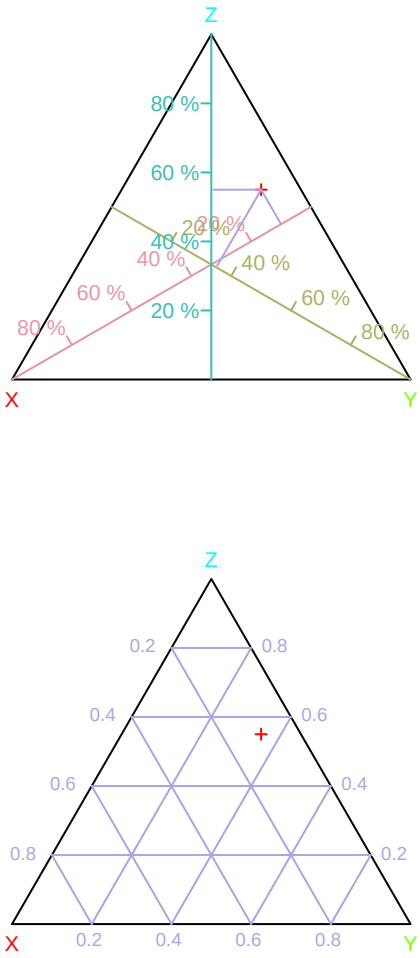


Figure 3.28: Ternary plots drawn with two possible visual aids for estimating the position of a point (red plus at x=0.1, y=0.35, z=0.55); axis names appear on the vertex opposite the axis they denote. code

In the following code `rcomp` normalises its argument (so that rows sum to 100) using an interval scale and returns an object having class `rcomp` (the `compositions` has overloaded functions for handing objects of this type):

```
library("compositions")

xyz=c(10, 35, 55)
plot(rcomp(xyz), labels="", col="red", mp=NULL)
ternaryAxis(side=-1:-3, labels=paste(seq(20, 80, by=20), "%"),
                pos=c(0.5,0.5,0.5), col.axis=hcl_col, col.lab=pal_col,
                small=TRUE, aspanel=TRUE,
                Xlab="X", Ylab="Y", Zlab="Z")

lines(rcomp(rbind(xyz, c(10, 45, 45))), col=hcl_col[4])
lines(rcomp(rbind(xyz, c(32, 35, 33))), col=hcl_col[4])
lines(rcomp(rbind(xyz, c(22, 23, 55))), col=hcl_col[4])

plot(rcomp(xyz), labels="", col="red", mp=NULL)

isoPortionLines(col=hcl_col[4])
ternaryAxis(side=0, col.axis=hcl_col, small=TRUE, aspanel=TRUE,
                Xlab="X", Ylab="Y", Zlab="Z")
```

3.3 Communicating a story

Results from data analysis have no value until they have been reliably communicated to the target audience.[110] Reliably communicating information to other people is difficult; the intended message may be misunderstood or important parts may simply be overlooked by its audience.

No known algorithm is available for selecting a method that communicates in a way that is correctly interpreted by its audience, telling the story that is part of the intended message. The main techniques available for presenting numerate information and how they might be implemented using R are covered in the rest of this chapter.

There are a wide variety of different ways in which information can be presented, e.g., tables, pie charts, bar charts and scatter plots. Which of these is best at communicating information to readers? The answer from a wide range of studies is that it depends on what information readers are trying to obtain. The following is a brief summary of some research findings:

- graph or table? Studies have found that except for reading-off specific values (and recall of these values later) subjects perform better with line graphs than tables. However, while graphs have better performance when presenting a given perspective (e.g., by selection of the axis), tables may be preferable[73] when wanting to present data in a way that does not favour any one perspective on the data; it boils down to selecting the best cognitive fit,[590]

- the ability of pie charts to communicate information has been questioned over the years.[128] A study[536] comparing subject performance using pie charts, a horizontal divided bar chart, a vertical bar charts and a table, found that except when direct magnitude estimation was required pie charts were comparable to bar charts, but for combinations of proportions pie charts were superior,

- adding a third dimension to a graph has been found to slow down reader performance,[264] i.e., subjects take longer to extract information and may be less accurate. The conclusion would appear to be not to use three dimensions when two would do. While subjects have expressed a preference for using 3-D graphs to impress others, no studies have investigated whether they have this effect,

- human judgement of the relative sizes of surface areas has been found not to be based on a linear scale, e.g., a circle of radius two is likely to be around three times larger, rather than four times larger, than a circle of radius one.[i] Encoding information as an area can lead to

[i] Stevens' power law states that psychological magnitude is proportional to the actual magnitude raised to some power. Unlike many other quantities used to express information graphically, for area this power is 0.8 rather than 1 (although values close to 1 have been found for some tasks).

reader communication breakdown; around a quarter of people focus on the areas of each slice in pie charts rather than either the length of the outer arc or the angle at the center of the chart,

- studies by Cleveland are often cited in R related publications: one study[115] asked subjects to make judgements about graphical information encoded in various ways; the results showed that accuracy of subjects' answers varied slightly between encoding methods, ordered from most accurate to least accurate: position along a common scale, positions along nonaligned scales, length/direction/angle, area, volume/curvature and shading/color saturation. Later studies[536] suggest that things are not so well-defined, with some effects seen being influenced by the structure of the experiments or performance with a particular encoding depending on the task subjects perform.

The thinking behind some layout details used by R's `plot` function are based on experimental work by Cleveland.[114] Although not explicitly stated the aim appears to have been to present data in a workman-like way that avoids the possibility of plotted data values being obscured by plot markings (e.g., tick marks).

The `plot` function is a workhorse for handing the graphical display of data in R; it does a good job of producing a reasonable looking plot from whatever it is passed. Based on this book's implementation goal of using one implementation technique, where-ever possible, the plots in this book were generated using the `plot` function.

The `lattice`[498] and `ggplot`[101] packages provide alternative world views on the plotting of data; `lattice` is based on the Trellis graphics system[48] from Bell Labs and has an emphasis on multivariate data, while the design of `ggplot` is derived from the work of Wilkinson.[618][ii] Both `lattice` and `ggplot` provide a great deal of control over the created plot through the use of user supplied functions. While `ggplot` is widely used by experienced R developers, its inability to sensibly handle whatever nonsense is thrown at it prevents this package being recommended for casual use. A detailed technical overview of the R graphics subsystems is available in 'R Graphics' by Paul Murrell.[403]

In some cases the intent of a plot may be to communicate that life is complicated. For instance, Figure 3.29 shows an estimate of the market share of Android devices in use in 2015, by brand/company and product name, based on the 682,000 unique devices that downloaded an App from OpenSignal.[427]

```
library("treemap")

and_tree=treemap(android, c("brand", "model"), "august2015",
                 title="", palette=pal_col,
                 border.col="white", border.lwds=c(0.5, 0.25))
```

[ii] The title of this book 'The Grammar of Graphics' refers to the structure of software written to display graphics rather than the structure of the displayed information.

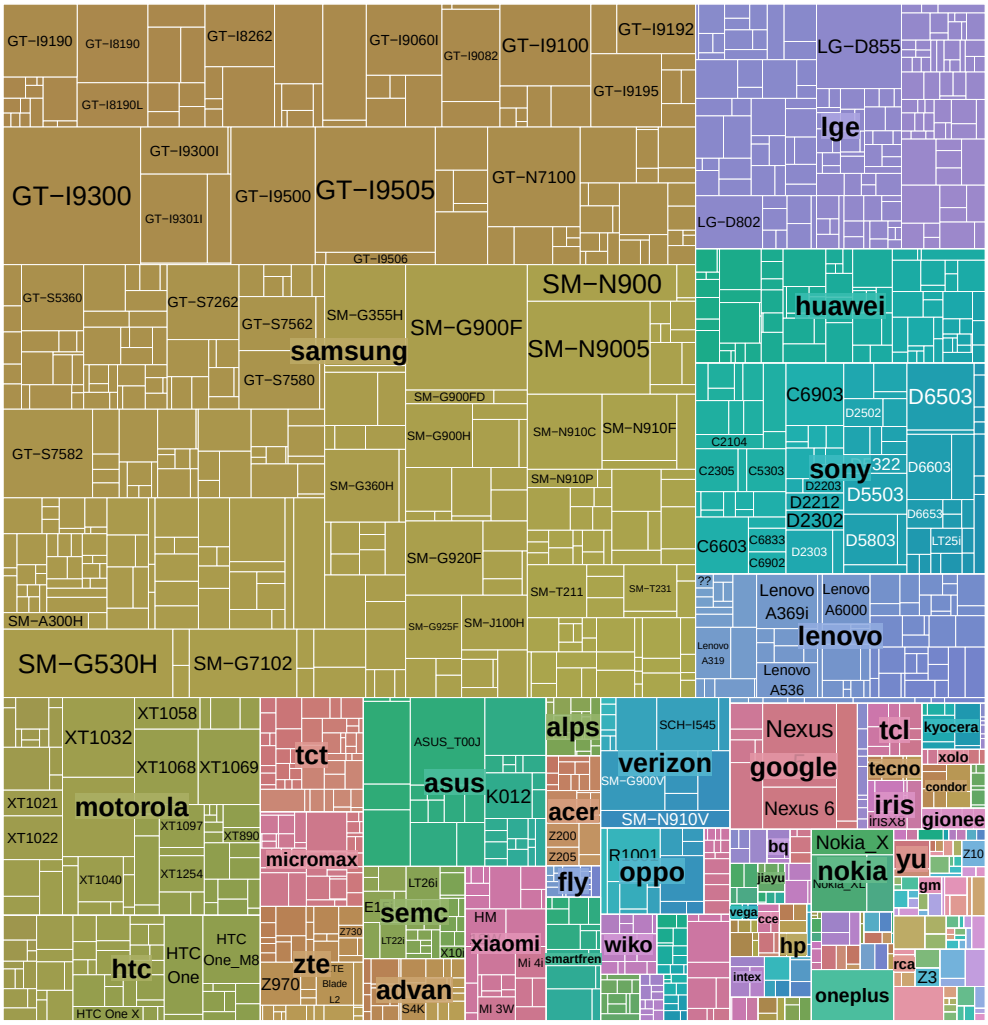


Figure 3.29: Estimated market share of Android devices by brand and product, based on downloads from 682,000 unique devices in 2015. Data from OpenSignal.[427] code

3.3.1 What kind of story?

The kinds of output from statistical data analysis include the following:

- a description of the data, e.g., its mean and variance, how measurements cluster, an equation that summarises the data. A descriptive model, built from the data, can be used to help gain insights into the system that was measured, for comparing different descriptions (e.g., benchmark results) and for building similar systems (e.g., automatically creating file system contents[6] for benchmarking purposes)
- a model built to mimic the behavior of a system (as expressed in the measurements made), e.g., a simulator,
- a predictive model capable of making appropriately accurate predictions using values not in the set of measurements used to build the model. The possible range of prediction values may be within the range of values used to build the model or outside the range of these values, e.g., making predictions about a future time,

A standard reply to any complaints about the adequacy of a model built using data is the adage 'All models are wrong but some are useful.'

An example of the different kinds of models that can be built and how their usefulness depends on the problem they are intended to solve is provided by a question involving the usage of local variables in the source code of a function definition.

If the source code contains a total of N read accesses to variables defined locally within the function, what percentage of variables will be read from once, how many twice and so on (based on a static count of the visible source code, not a dynamic count obtained by executing the function)?

Using data from an analysis of C source[299] we have a description of "what is".

Plotting the data shows that a few variables account for most of the accesses (i.e., read from). After some experimentation the following equation was found to be a good fit to the data; see Figure 3.30:

$$pv = 34.2e^{-0.26acc-0.0027N}$$

where pv is the percentage of variables, acc is the number of read accesses to a given variable and N is the total number of accesses to all local variables within a function. For example, if a function contains a total of 30 read accesses of local variables the expected percentage of variables accessed twice is: $34.2e^{-0.26\times 2-0.0027\times 20}$.

Are there other ways of building a model to answer the question asked?

This problem has a form that parallels a model of the growth of new pages and links to existing pages on the world wide web. Each access of a local variable could be thought of as a link to the information contained in that variable. One algorithm that has been found to do a reasonable job of modeling the number of links between web pages is *Preferential attachment*.

With some experimentation an iterative algorithm, based on these ideas, was created that produced a pattern of behavior close to that seen in the data. The algorithm is as follows:

Assume we are automatically generating code for a function and from the start of the function to the current point in the code L distinct local variables exist (and have been accessed), with each accessed R_i times ($i = 1, \ldots, L$). The following weighted preferential attachment algorithm is used to select the next local variable to access (global variables are ignored in this analysis):

- With probability $\frac{1}{1+0.5L}$ select a new variable to access,[iii]
- with probability $1 - \frac{1}{1+0.5L}$ select a variable that has previously been accessed in the function, select an existing variable with probability proportional to $R + 0.5L$ (where R is the number of times the variable has previously been read from; e.g., if the total accesses up to this point in the code is 12, a variable that has had four previous read accesses is $\frac{4+0.5\times 12}{2+0.5\times 12} = \frac{10}{8}$ times as likely to be chosen as one that has had two previous accesses).

The red points in Figure 3.30 were calculated using the above algorithm.

This preferential attachment model provides an insight into local variable usage that is very different from that provided by the fitted exponential equation. neither of them could not be said to be realistic descriptions of the process used by developers when writing code. Both models are descriptions of the end result of the emergent process of writing a function definition. Each model has its own advantages and disadvantages, including the following:

- the fitted equation is fast and simple to calculate, while the output from the iterative model is slow (an average over 1,000 runs in the example code) and requires more work to implement,
- the iterative model automatically generates a possible sequence of accesses (for machine generated source), while a fitted equation does not provide any obvious method of generating a sequence of accesses,
- multiple executions of an iterative model can be used to obtain an estimate of standard deviation, while the equation does not provide a method for estimating this quantity (it may be possible to fit another equation model that provides this information),
- the equation provides an end result way of thinking while the iterative model provides a choice-based way of thinking about variable usage.

A common technique for devising a model for a new problem is to find a very similar problem that has a proven model, and to adapt this existing model to the new problem. A model based on existing practice is often easier to sell to an audience than a completely new model.

Some multiprocessor system have a "shared nothing" architecture, which minimises the sharing of hardware resources. Benchmark performance measurements of such a system under

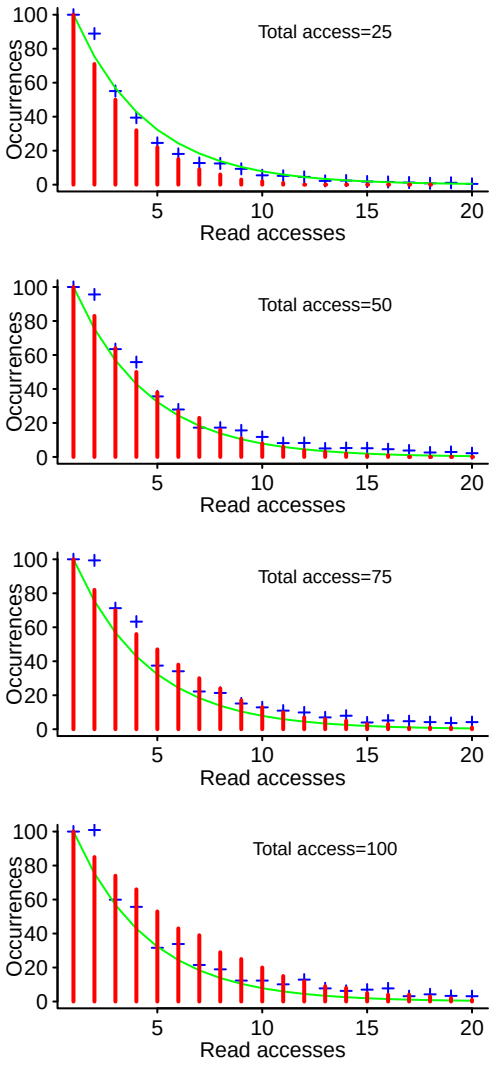


Figure 3.30: Variables having a given number of read accesses, given 25, 50, 75 and 100 total accesses, calculated from running the weighted preferential attachment algorithm (red), the smoothed data (blue) and a fitted exponential (green). code

[iii] The unweighted preferential attachment algorithm uses a fixed probability to decide whether to access a new variable.

various loads shows that even when tasks can be evenly distributed across all X processors in the system, performance is rarely X times faster. What model provides a good explanation of the performance seen?

Amdahl's law predicts changes in multiprocessor performance as the number of processors used changes, where the multiprocessor system has a shared hardware architecture. Gunther[239] extended this "law" to cover multiprocessors having "shared nothing" architecture; the adapted model, plus a further adaption, are not good fits to the data (see Figure 3.31).

Gunther[240] then created a model based on queuing theory and ran simulations to model performance (with each job waiting in a queue for time t_1 and executing for time t_2). The argument for using queuing theory was that data sharing between different programs can create a resource contention that the "shared nothing" hardware architecture cannot unblock.

Figure 3.31 shows that the queuing model more accurately follows the pattern measured. Given the small amount of data available it would be unwise to attempt further model tuning.

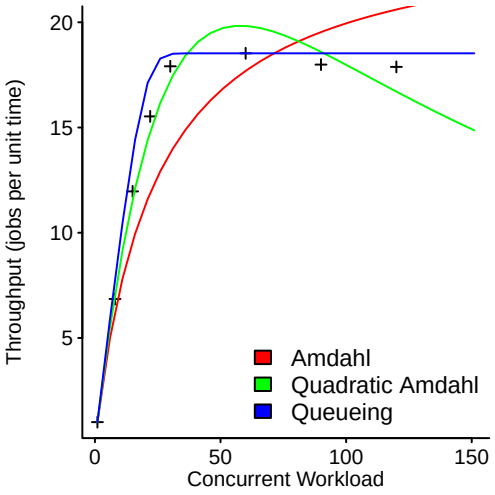


Figure 3.31: Throughput when running the SPEC SDM91 benchmark on a Sun SPARCcenter 2000 containing 8 CPUs, with the predictions from three fitted queuing models. Data from Gunther.[240] code

The R language does not contain features designed with simulation in mind[iv], but like most languages it can be used to solve problems outside of its core domain; see the `simFrame` package.

Finding a good model for the data can sometimes involve many iterations over a long time. For instance, modeling the growth of the size and number of files/directories in a filesystem has a long history, with current models[395] either involving a mixture of two distributions for the equation fitting approach or a generative approach based on simulating the way new files are created from existing files.

Perhaps the most important question to answer when proposing any model is the purpose to which it will be put. A model intended to gain insight might not be of any use in making practical recommendations and a model used to make predictions might not provide any useful insight. For instance, modeling the connection between modifications to files and the introduction of new faults may be used to predict fault rates based on modification history, but this model has limited scope for directly deriving techniques for reducing faults (e.g., reduce faults by reducing file modifications, is of no use when customers want new or modified behavior in the applications they use).

In most cases a great deal of domain knowledge is required to build a model that has the desired level of performance. There is no guarantee that any created model will be sufficiently accurate to be useful for the problem at hand; this is a risk that occurs in all model building exercises. Ideally model building is driven by a theory describing the behavior of the system being modeled. When a theory is complete there is no need for new models, the fact that the creation of a new model is being considered implies that existing models are lacking in some respect.

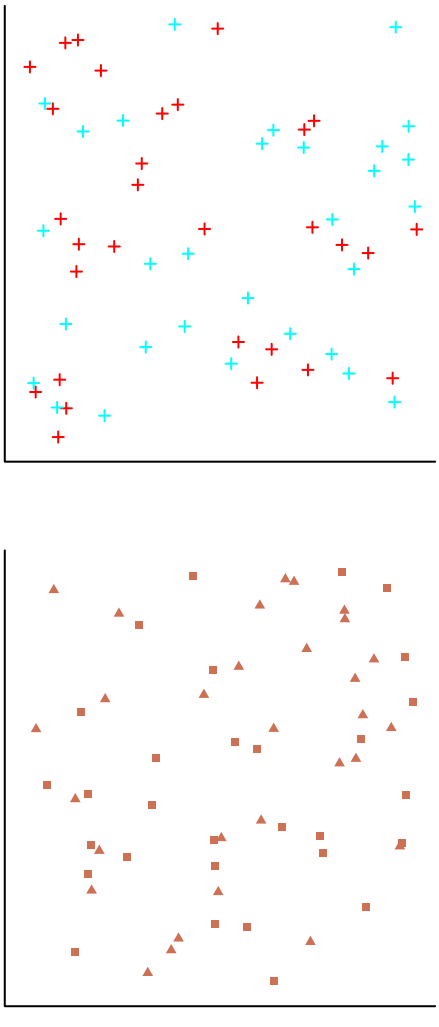


Figure 3.32: Illustration of the difference in cognitive effort needed to locate points differing by shape or color. code

3.4 Technicalities should go unnoticed

The machinery of information presentation should not get in the way of reader's access to that information.

There are many books offering tips, suggests and recommendations for how best to present visual information to readers; the only book highly recommended by your author (it is based on a wide range of empirical research) is 'Graph Design for the Eye and Mind' by Stephen Kosslyn.[332] Sometimes multiple plots are used to tell an evolving story, McCloud[376] is a great introduction to this art form.

3.4.1 People have color vision

Until the mid 1980s most people used computer terminals that could only display black and white (or green and black). Thirty years later the look-and-feel of computer usage in the mid-1980s still predominates in serious works involving statistical visualization.

[iv] Interfacing to Netlogo[464]

This book treats color as an essential component of numeric story telling. Color provides an extra dimension that can provide more information within the same space and help the viewer extract information from a plot.[v]

Selecting the most appropriate colors to use requires skill and experience. The `colorspace` and `RColorBrewer` packages both include functions that automatically select a color palette based on the arguments passed;[vi] the `colorspace` package provides a wider range of functionality than `RColorBrewer` and is used to select the colors for the plots in this book. The default usage of the color palette generating functions in this package is to pass the number of colors required in the palette, and assign the returned vector (whose values can be passed as the color argument to plotting functions).

The Hue-Chroma-Luminence (HCL) color space is claimed[624] to provide a better mapping to the human color perceptual system (hue: dominant wavelength; chroma: colorfulness, intensity compared to gray; and luminence: brightness, amount of gray) than alternative spaces.[vii] The color palettes generated by the `rainbow_hcl` function are considered to be qualitative palettes, that is suitable for depicting different categories; those generated by the `sequential_hcl` function to suitable for coding numerical information that ranges over a given interval, with the `diverge_hcl` function also encoding numerical information but including a neutral value.

The `choose_palette` function provides an interactive, slider based, method for developers to define their own color palettes.

Approximately 10% of men and 1% of women have some form of color blindness. The `dichromat` package provides a way of showing how a plot that contains color would appear to a viewer having some form of color blindness. It does this by making use of experimental data[593] to simulate the effects of different kinds of color blindness, modifying the requested colors to appear, to normal sighted viewers, like they would to a viewer having the selected kind of color blindness.

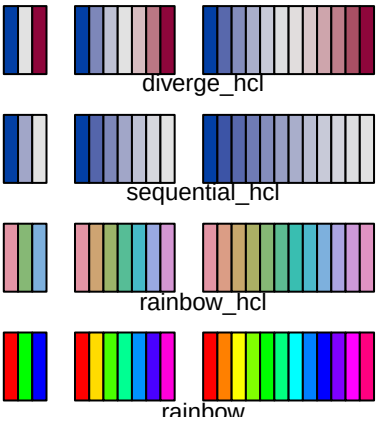


Figure 3.33: The three, seven and twelve color palettes returned by calls to the `diverge_hcl`, `sequential_hcl`, `rainbow_hcl` and `rainbow` functions. code

3.4.2 Color palette selection

Figure 3.34 shows how time varying data involving related items (in this case market share of successive versions of Android) can be displayed in a way that preferentially highlights one aspect of the data; the left plot highlighting individual versions while the right plot shows each version contribution to the overall market share. Bold colors are effective at drawing attention to individual lines, but can be overpowering when there is a large area of color in the plot; the opposite is often the case for pastell colors.

[v] R contains 657 built-in color names (the `colors` function lists them) and also supports hexadecimal RGB literals.

[vi] The selection process is based on theories derived from the use of color in maps,[76] which has a long history.

[vii] Red-Green-Blue (RGB) is a specification based on the display of color on computer screens; Hue-Saturation-Value (HSV) is a transformation of RGB that attempts to map to the human perceptual system and is used by some other software packages.

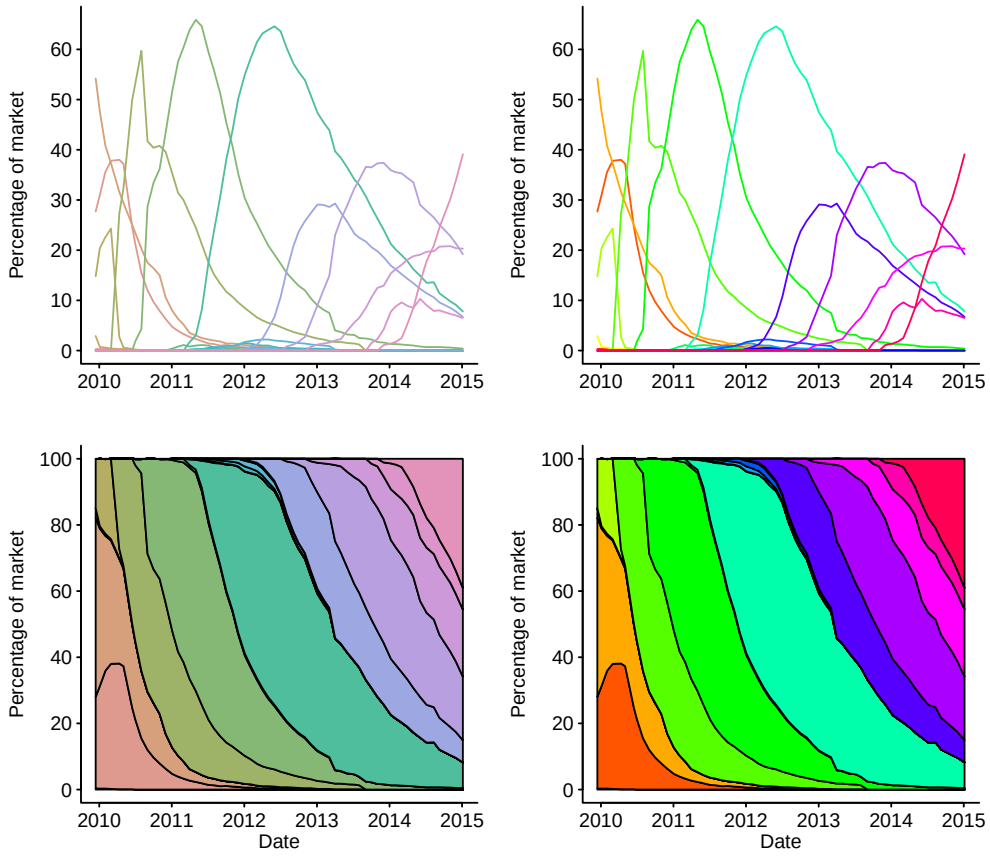


Figure 3.34: Percentage share of the Android market by successive Android releases between 2010 and 2015. Data from Bidouille.[594] code

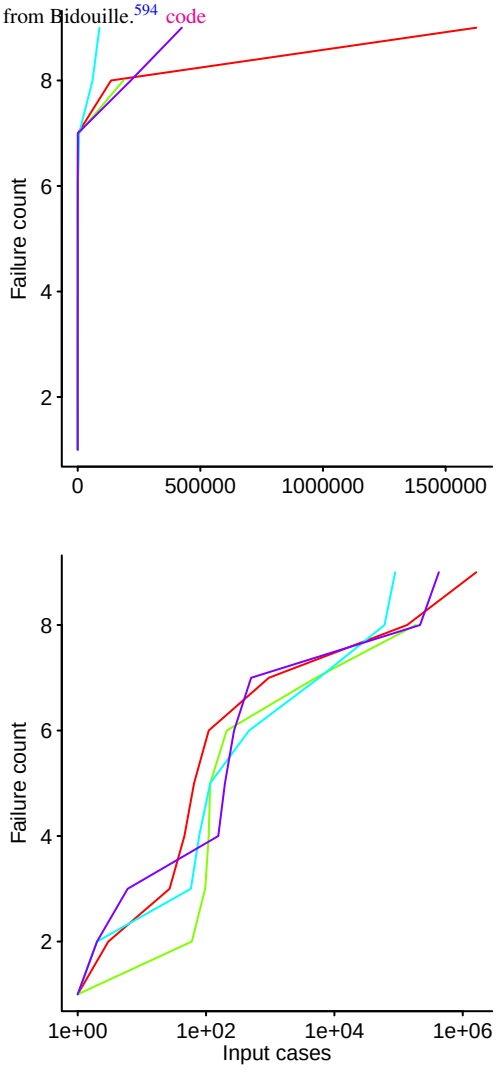


Figure 3.35: Values plotted using a linear (upper) and logarithmic (lower) x-axis. Data from Dunham et al.[152] code

3.4.3 Plot axis: what and how

The choice of plotting axis can have a dramatic impact on the visual perception of the displayed data.

The two most commonly used methods for mapping values to points along an axis are linear and logarithmic. When the range of plotted values spans several orders of magnitude, using a logarithmic axis often produces a more informative visualization; compare the use of linear and log axis in Figure 3.35. Plotting values drawn from an exponential or power law-like distribution on a linear scale axis often results in many of the points being visually clumped together in a small area of the plot; use of a log scaled axis has the effect of expanding these clumped values.

The `plot` function (and many other plotting functions) automatically select the minimum/-maximum range of each axis based on the range of the data passed; by default 4% is added to each end of the range.

The choice of quantity plotted along each axis is driven by the desire to highlight a relationship between the two quantities; the purpose of a plot is to help viewers appreciate this relationship.

Care needs to be taken to ensure that artificial relationships are not created by the choice of quantity used for an axis. An example of the wasted effort that can occur when the relationship implied by poorly selected quantities is provided by the sorry saga of bug density vs. lines of code.

It was noticed that when bug density (i.e., number of faults divided by lines of code in a function) was plotted against lines of code (in a function), the distribution of points followed a U-shape pattern. Some people proposed that the minimum of this U represented an optimum for the length of a function.[247]

A study by El Emam, Benlarbi, Goel, Melo, Lounis and Rai[157] showed that this U-shape was an artefact generated by the choice of quantities plotted along each axis. Plotting the ratio $\frac{F}{LOC}$ against LOC, with F constant, will produce a tilted U-shape (blue line in Figure 3.36). If the number of faults grows faster than the number of lines of code (which has been found to occur for large line counts) then U-shaped curves such as the red line in Figure 3.36 can occur (a growth rate was picked to illustrate one possibility).

```
x=1:100 ; inv.x=1/x

plot(x, 3*inv.x, type="l", col=pal_col[1],
                        xlab="LOC", ylab="Faults/LOC\n")
lines(x, ((x+50)^3/5e4)*inv.x, col=pal_col[2])
```

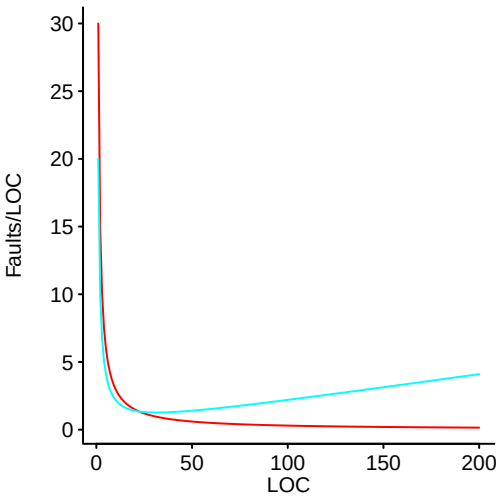


Figure 3.36: Illustration of U-shape created when y-axis values are a ratio calculated from x-axis values. code

The idea suggested by the U-shape pattern in this plot, that there might be an optimal function length, is purely a misinterpretation of the behavior of a ratio quantity plotted against one of the values used in the ratio calculation.

A log transform of an axis can sometimes hide potentially useful information, rather than help reveal it. Figure **??** shows Figure 8.3 from a study by Putnam and Myers;[466] in both cases the x-axis is log transformed. In the right plot the y-axis is linear and there is a visually distinct cluster of measurements, across the top; in the lower plot, where both axis are log transformed, this cluster is visually less prominent.

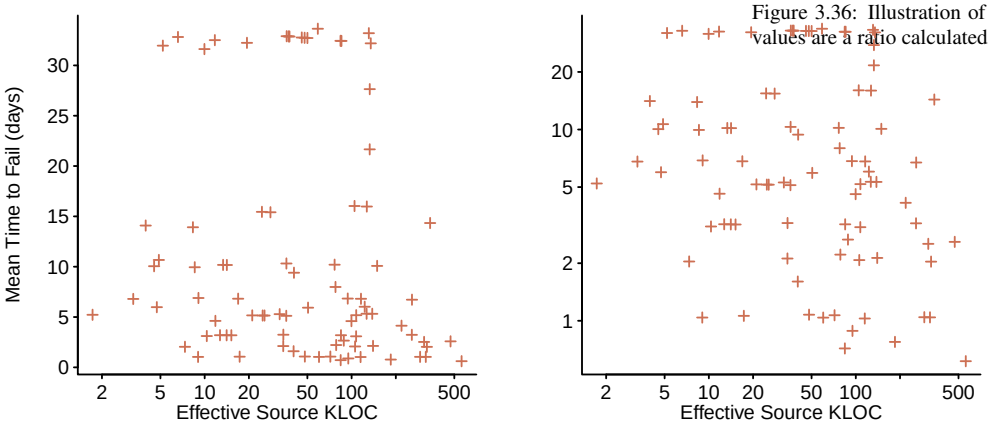


Figure 3.37: Mean time to fail for systems of various sizes (measured in lines of code); linear y-axis left, log y-axis right. Data extracted from Figure 8.3 of Putnam et al.[466] code

3.5 Communicating numeric values

The output from statistical analysis can include visual plots and a small collection of numbers. What is the best way to communicate a story involving a small collection of numbers?

The form of result communication depends on the sophistication of the target audience, with a single value often being preferred for simplicity of selling the result.

A table of numbers covering a very wide range of values (e.g., Table 3.2) can be difficult to interpret quickly unless this is something readers regularly do. An alternative representation separates out the mantissa and exponent, and combines them using area and color, allowing a same/different comparison to be made (see Figure 3.38).

Operation	**Approximate runtime**
L1 cache reference	1 ns
Branch mispredict	3 ns
L2 cache reference	4 ns
Mutex lock/unlock	17 ns
Main memory reference	100 ns
Send 2K bytes over commodity network	177 ns
Compress 1K bytes with Zippy	2,000 ns
Read 1 MB sequentially: memory	7,000 ns
SSD random read	16,000 ns
Round trip within same datacenter	500,000 ns
Read 1 MB sequentially: magnetic disk	1,000,000 ns
Seek: magnetic disk	3,000,000 ns
Send packet CA→Netherlands→CA	150,000,000 ns

Table 3.2: Numbers Everyone Should Know, circa 2016. Data from Scott.[119]

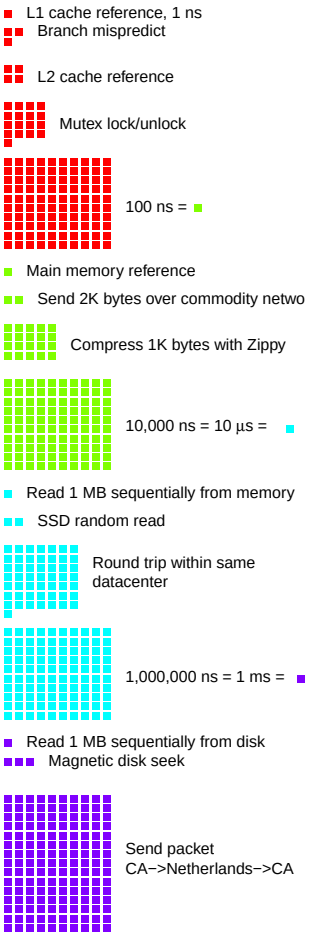


Figure 3.38: Alternative representation of numeric values in Table 3.2. Data from Scott.[119] code

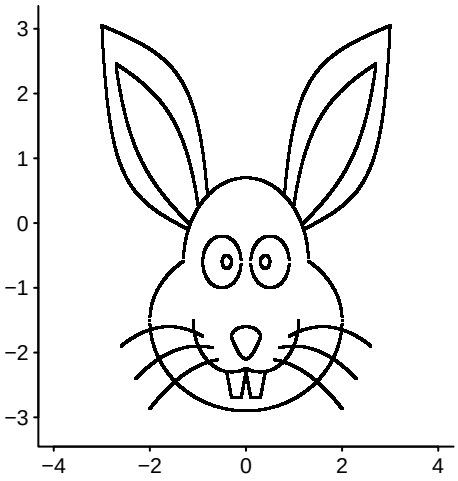


Figure 3.39: What's up doc? Not the fitted model you were expecting. Equations from White.[612] code

Confidence bounds are a good way of communicating uncertainty... p-values are often misunderstood...

Multi-way contingency tables using the `vcd` package...

Some statistical procedures find the values of parameters in an equation that result in the best fit to the data. While the numeric values are the output of from the model building the information being communicated is equation+parameter values, i.e., the final fitted equation should be shown. This suggestion is illustrated in the section on building regression models, as is the use of functions in the `sjPlot` package plot the parameters of various regression models.

Packages are available for integrating the output from R programs into the workflow of various document preparation systems, for instance, the `ascii` package provides functions for producing Asciidoc compatible output and the `knitr` package produces LaTeX output.

```
x=seq(-4.7, 4.7, by=0.002)

plot(0, type="n", col=point_col,
       xlim=c(-4.0, 4.0), ylim=c(-3.2, 3.1),
       xlab="", ylab="")

dummy=sapply(x, function(X) points(rep(X, 18), c(
       c(1, -0.7, 0.5)*sqrt(c(1.3, 2, 0.3)^2-X^2) - c(0.6, 1.5 ,1.75),
       0.6*sqrt(4 - X^2)-1.5/as.numeric(1.3 <= abs(X)),
       c(1, -1, 1, -1, -1)*sqrt(c(0.4, 0.4, 0.1, 0.1, 0.8)^2-(abs(X)-c(0.5, 0.5, 0.4,
                                                                  c(0.6, 0.6, 0.6
       (c(0.5, 0.5, 1, 0.75)*tan(pi/c(4, 5, 4, 5)*(abs(X)-c(1.2, 3, 1.2, 3)))+c(-0.1,
                          as.numeric(c(1.2, 0.8, 1.2, 1) <= abs(X) & abs(X) <= c(3,3,
       (1.5*sqrt(X^2+0.04) + X^2 - 2.4) / as.numeric(abs(X) <= 0.3),
       (2*abs(abs(X)-0.1)+2*abs(abs(X)-0.3)-3.1)/as.numeric(abs(X) <= 0.4),
       (-0.3*(abs(X)-c(1.6, 1, 0.4))^2-c(1.6, 1.9, 2.1))/
                          as.numeric(c(0.9, 0.7, 0.6) <= abs(X) & abs(X) <= c(2.6, 2.3
                                                                              ),
                                                                    pch="."))
```

3.5.1 Percentages vs frequencies

Experiments have found that people are much better at extracting certain kinds of information when it is presented in the form of frequency of occurrence rather than as a percentage[202]...

pie charts... inline charts...

Chapter 4

Probability

4.1 Introduction

What are the chances of an event occurring?

Probability is the mathematics involved in answering this question and reasons for being interested in the estimation of probabilities include:

- betting, does a particular case need to be handled, making an insurance decision and for all other decisions and predictions,
- deciding the extent to which an event is surprising. The level of surprise might be used to decide whether something is going wrong or when performing statistical analysis discriminating between hypotheses.

Readers are assumed to have some basic notion of the concept of probability and have encountered the idea of probability in the form of likelihood of an event occurring; classic examples involve calculating the probability of a given combination or sequence of values occurring when flipping a coin or rolling a die, e.g., two heads or rolling two sixes, or the probability of having to make N flips/rolls before some event occurs.

What is the difference between probability and statistics?

Probability makes inferences about individual events based on the characteristics of the population, while statistics makes inferences about the population based on the characteristics of a sample of the population.

Another way to compare the two is that probability makes use of deductive reasoning while statistics makes use of inferential reasoning.

Probability and statistics are intertwined in that ideas and techniques from probability about individual events may be used when solving problems involving statistics and results about the characteristics of a population obtained from statistical analysis may be used to help solve problems involving probability.

This book is empirically driven and so primarily makes use of statistical analysis. The following is an example of a problem solved primarily using probability.

The vendor of a static analysis tool wants to add support for detecting a newly discovered potential fault pattern. An occurrence of this pattern in code is not always a fault, what is the upper bound on the probability of generating a false positive that keeps the likelihood that developers will stop using the tool below some limit (say 10%)?[i]

Answering this question requires knowledge of the mental model used by developers to evaluate analysis tool performance. Two possible mental models include the following (which assumes zero correlation between difference warning occurrences and that developers assign the same importance to all warning messages):

[i] Experience shows that these false-positives are sufficiently unpopular with developers (they are a source of wasted effort) a developer will often stop using the tool concerned if they are encountered too often. Higher false-positive rates for Tornado warnings result in more deaths and injuries,[520] through people ignoring the warning.

- an *economic* developer who tracks the benefit of processing each warning (e.g., false positive warning -1 benefit, else $+1$ benefit), starting in an initial state of zero benefit this economic developer stops processing warnings if the current sum of benefits ever goes negative.

  The Ballot theorem can be used to calculate the probability that for a sequential processing of warnings, the number of correct warnings is always greater than the number of false positive warnings (assuming equal weight is given to both cases, the alternative being more complex to analyse). Let C be the number of correct warnings and F the number of false positive warnings and assume $C > F$, then the probability is given by:

  $$\frac{C-F}{C+F}$$

  rewriting in terms of probability of the two kinds of warning we get:

  $$C_p - F_p$$

  so, for instance, when the false positive rate is 0.25 the probability of a developer processing all the warning generated by a tool is 0.75-0.25 $\rightarrow$ 0.5, and does not depend on the total number of warnings.

- an *instant gratification* developer who processes each warning and stops when a sequence of N consecutive false positive warnings have been encountered. This kind of thinking is analogous to that of the *hot hand in sports* (what psychologists call the clustering illusion).

  What is the probability that a sequence of N consecutive false positive warnings is not encountered?

  If the total number of warnings is k and q is the probability of a false positive occurring, then the probability of a run of N consecutive false positive warnings occurring can be calculated using the following recurrence:

  $$P(k,q,N) = P(k-1,q,N) + q^N(1-q)(1-P(k-N-1,q,N))$$

  with initial values:

  $$P(j,q,N) = 0, \text{ for } j = 0,1,\ldots,N-1$$

  $$P(j,q,N) = q^N, \text{ for } j = N$$

  Figure 4.1 shows the probability of not encountering a sequence of three (red) or four (blue) consecutive false positive warnings when processing some total number of warning messages, for various underlying false positive rates (ranging from 0.5 to 0.2).

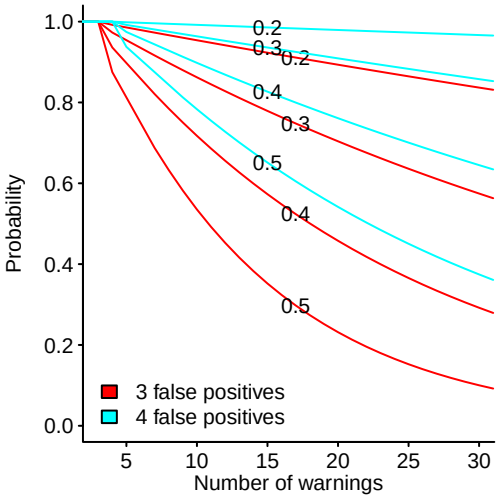


Figure 4.1: Probability that three (red) or four (blue) consecutive false positive warnings occur in some total number of warnings (false positive rate appears on line). code

When dealing with warnings involving complex constructs a developer may be unwilling to put the effort into understanding what is going on and either go along with the what the static analysis tool says, thus underestimating the actual false positive rate, or default to assuming the warning is a false positive, thus overestimating the actual false positive rate.

Finding an equation or technique to use in solving a problem involving probability requires some knowledge of the terminology used in this field. Possible phrases to try in search queries include: birth and death process, coin tossing, colored balls, combination, ergodic, event, fair games, first passage time, generating function, Markov chain, Markov process, occupancy problem, partitions, permutation, random walk, stochastic, trials and urn model.

Finding a closed form solution can be difficult, even when one exists. Simulation using Monte Carlo methods can provide usable estimates of the value of interest.

4.1.1 Useful rules of thumb

If the distribution of the values taken by some attribute, in a population, is not known the following inequalities may be of use as worst case estimates of the probability of various relationships being true. Both inequalities are distribution independent (the price of this generality is that the bounds are loose).

Markov inequality

The Markov inequality uses the sample mean, X, to calculate the maximum probability that X (which is required to be nonnegative) is larger than some constant. The inequality does not make any assumptions about the sample distribution:

$$P(X \geq k) \leq \frac{\mu}{k}$$

where μ is the sample mean.

Example. If a sample of measurements has $\mu = 10$, then the probability of the sample containing a value greater than or equal to 20 (i.e., twice the mean) is $\frac{10}{20}$.

Chebychev's inequality

If the standard deviation (σ) of the sample is known, then Chebychev's inequality can be used to obtain a tighter bound than that given by the Markov inequality, as follows:

$$P(|X-\mu| \geq k\sigma) \leq \frac{1}{k^2}$$

alternatively:

$$P(|X-\mu| \geq k) \leq \frac{\sigma}{k^2}$$

Using the above example the probability of the sample containing a value that differs from the mean by at last 10 is less than or equal to: $\frac{\sigma}{10^2}$.

Example: an analysis of the number of mutants needed to estimate test suite adequacy to within a specified error and confidence bounds.[215]

Fréchet inequalities Bounds on the union and disjunction of two or more probabilities are given by the Fréchet inequalities, as follows:

Logical conjunction: $\max(0, P(a_1)+P(a_2)-1) \leq P(a_1 \& a_2) \leq \min(P(a_1), P(a_2))$

Logical disjunction: $\max(P(a_1), P(a_2)) \leq P(a_1 \vee a_2) \leq \min(1, P(a_1)+P(a_2))$

Correlation between three variable pairs If the correlation between two pairs of three variables is known, say r_{12} and r_{13}, the bounds on the correlation of the remaining pair r_{23} is given by:

$$r_{12}r_{13} - \sqrt{(1-r_{12}^2)(1-r_{13}^2)} \leq r_{23} \leq r_{12}r_{13} + \sqrt{(1-r_{12}^2)(1-r_{13}^2)}$$

As the number of variables involves increases, the expressions become more complicated.[85]

4.1.2 Measurement scales

Mathematically measurement values can be characterised by whether they are discrete or continuous, and the properties of the scale used. Possible scales include the following:

- Discrete
  - nominal scale: each measurement value has an arbitrary number or name. Because the choice of number/name is arbitrary, no ordering relationship exists between different numbers/names. A nominal scale is not really a scale in the usual sense of the word.

    Examples: the numbers on the back of footballers' shirt or the various sales regions in which a product is sold.

- ordinal scale: each measurement value is a number or name of an item and an ordering relationship exists between the numbers/names. The distance between distinct values need not be the same.

  Example: Classifying faults by their severity, e.g., minor, moderate, serious.

  If a minor fault is considered less important than a moderate fault, and a moderate fault is less important than a serious fault we can deduce that a minor fault is less important than a serious fault.

  The address of the members of a structure type increases for successive members, but the difference between member addresses is not fixed because different members can have different types.

  When names are assigned to entities, there may be cultural differences in the selection process. Figure 4.2 shows how words are assigned to tracts of trees having various areas.

  - Continuous

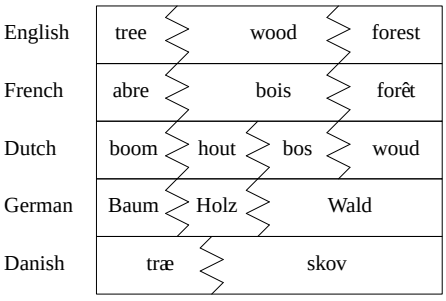


Figure 4.2: The relationship between words for tracts of trees in various languages. The interpretation given to words (boundary indicated by the zigzags) in one language may overlap that given in other languages. Adapted from DiMarco et al.[149]

- interval scale: each measurement value is a number and not only does a relative ordering exist but a fixed length interval of the scale denotes the same amount of quantity being measured.

  A data point of zero does not indicate the absence of what is being measured.

  Example: the start date of some event is an interval scale. If the start date of events A, B and C are known, and difference in start date between events A and B is the same as between events C and D, then it is possible to calculate the start date of event D.

  Addition and subtraction can be applied to values on an interval scale but not multiplication or division (e.g., it makes no sense to say that the start date of event A is twice that of event C).

- ratio scale: each measurement assigns a number to an item and this numeric scale preserves: the ordering of items, the size of the interval between items and the ratios between items. It differs from the interval scale in that a measurement of zero denotes the lack of the attribute being measured.

  The time difference between two events is a ratio scale.

The kinds of statistical analysis that can be legitimately performed on the values in a sample will depend on the kind of scale used to measure values.

4.2 Probability distributions

Probability distributions are mathematical descriptions of the properties of values obtained by following a consistent pattern of behavior, e.g., the flipping a coin pattern of behavior generates one of two results, a fixed probability of either result, with each result being independent of the previous one and a count of the number of heads and tails has a binomial probability distribution.

If a sample of values can be fitted to a known probability distribution, then information about the pattern of behavior that generated them can be inferred from what is known about processes generating values having that particular distribution. For instance, given a list of pairs of numbers, if the ratio formed from each pair (i.e., $\frac{a}{a+b}$) can be fitted to a binomial distribution, then there is strong evidence that the pairs are counts of a process producing one of two possible values (e.g., heads/tails, yes/no etc) and the probability of producing each value can be calculated from the fitted distribution.

Fitting a distribution to a sample is a step towards understanding the processes that generated the measurements, not an end in itself.

Failure to fit values to a known distribution may mean that more than one distribution is involved, e.g., two different coins are being used and both are biased in some way. Given enough data it is sometimes possible to obtain a reasonable fit that involves two or more distributions.

If there is a reason for believing that the measured processes are driven by particular behaviors, the quality of fit of the predicted probability distribution to the sample can be compared against the quality of fit of other distributions.

If there is no expectation of a particular behavior, then finding an acceptable fit of a probability distribution to the sample values is a starting point for understanding the processes that are driving the measurements observed.

There are a very large number of probability distributions[109] but only a handful of them crop up regularly; R packages tend to support commonly occurring distributions with a few packages supporting a wide range of distributions.[109]

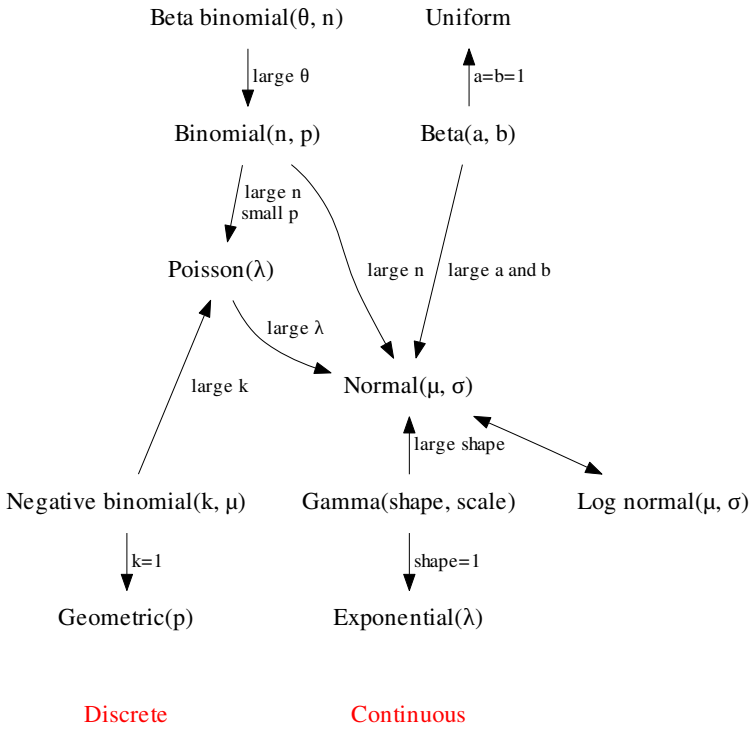


Figure 4.3: Relationships between common discrete and continuous probability distributions.

Every family of probability distributions is completely characterised by a small set of numbers (often one or two) and a formula that the numbers parameterise. For instance everything about a Normal probability distribution can be calculated by plugging values for the mean, μ, and standard deviation, σ, into the formula: $P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$ (this formula is often abbreviated as $N(\mu, \sigma)$). Fitting data to a Normal distribution involves finding appropriate values for μ and σ.

In practice a few probability distributions are encountered much more frequently than others. One of these common cases will often fit reasonably well to a wide spectrum of commonly encountered samples and unless there are theoretical reasons for expecting one of the less commonly encountered distributions then there is nothing to be gained by searching through all known distributions to find the one that is the best fit for a sample.

Some of the characteristics of plotted sample values, that may or may not correspond to a known probability distributions include:

- mean value (sometimes called the *arithmetic mean*, *central tendency* or *location value*, the last two terms may also be used to refer to the median): many distributions have a finite mean (some that don't include power laws with an exponent greater than or equal to -1 and the Cauchy distribution),
- scale parameter, variance (standard deviation is the square-root of variance): how spread out the distribution is, a few distributions do not have a finite variance, e.g., power laws with an exponent between zero and -2,
- symmetrical/asymmetrical about the mean, the *skew* of a distribution is a measure of how asymmetrical it is; a symmetric distribution has a skew of zero, while a positive skew has a tail pointing towards larger positive values and a negative skew has a tail pointing towards negative values.
- where most of the density resides, i.e, around the mean or in the tails. The *kurtosis* of a distribution is a measure of how spiky the distribution is, possibilities include tall and slim (known as *leptokurtic*; slender-curved), short and flat (known as *platykurtic*) or medium-curved (known as *mesokurtic*; the Normal distribution has a Kurtosis of three).
- number of distinct peaks, known as the *modality* of a distribution; a distribution with one distinct peak is said to be unimodal, two distinct peaks bimodal (such as measurements from two different distributions, e.g., height of men/women),

The `moments` package contains functions for calculating skewness, kurtosis and moment related attributes of a numeric vector.

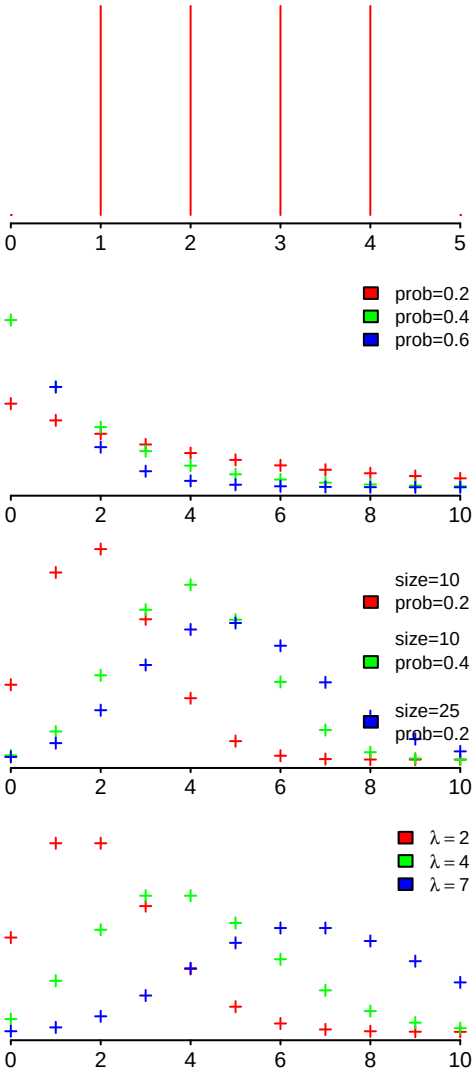
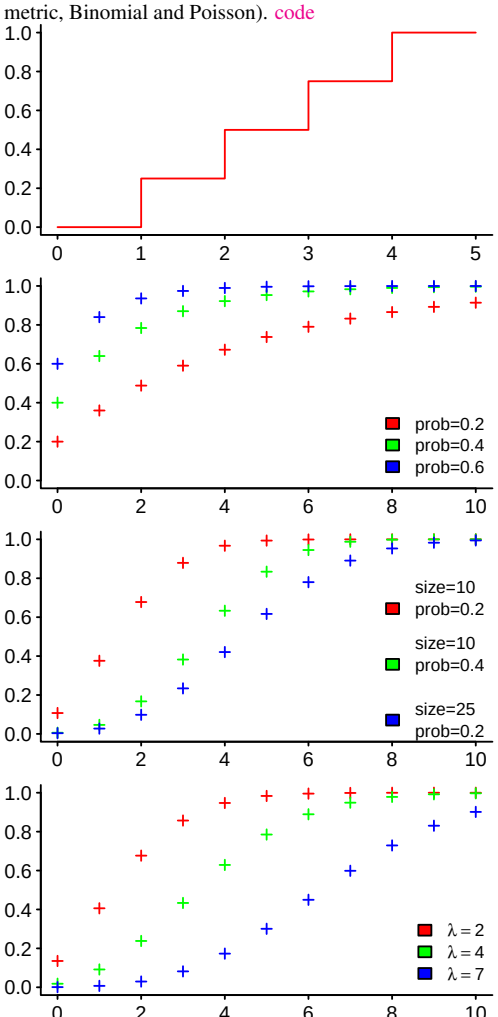


Figure 4.4: Shapes of commonly encountered discrete probability distributions (upper to lower: Uniform, Geometric, Binomial and Poisson). code



Probability distributions can be divided into discrete and continuous distributions, with discrete distributions only being defined at specific points (usually integer values). In R functions that involve discrete distributions usually require integer values while functions involving continuous distributions takes floating-point values.

There are various ways of representing a probability distribution and the following are often encountered:

- density function: for discrete distributions, see Figure 4.4, this can be viewed as the probability that x will have a given value, $P(x = value)$; for continuous distributions, see Figure 4.6, the probability of any particular value occurring is zero, however there is a finite probability of a measurement returning a value within a specified interval.
- cumulative density function: the probability that x will be less than or equal to a given value $P(x < value)$, see Figure 4.5,
- equation: an equation for the probability distribution. For the majority of people, except for a few distributions, this is little more than eye candy, e.g., the equation, $\frac{\lambda^k e^{-\lambda}}{k!}$, is very difficult to visualize and is only of use to developers wanting to implement the Poisson distribution.

Discrete distributions: commonly encountered discrete distributions include the following (see Figure 4.4):

- Binomial distribution: for a random variable X,
  1. the process involves a sequence of independent trials,
  2. each trial produces two possible outcomes, e.g., heads/tails,
  3. the probability of either outcome (say p for heads) does not change,
  - X counts the number of success (where success might be defined as a head occurring) in n fixed trials.
    The Binomial distribution is complete described by two parameters: $B(n, p)$,
    The above process is sometimes described as the process of drawing n objects from a pool containing a finite number of two kinds of object, where the object is placed back in the pool after it has been drawn (the draws are said to be *with replacement*).
    The Hypergeometric distribution is obtained if objects are not returned to the pool once they are drawn (the draws are said to be *without replacement*).
    A distribution that takes more than two discrete values is known as a *Multinomial distribution* (again with a fixed probability of each value occurring). The `XNomial` package provides support for multinomial distributions,
- Negative Binomial distribution: this has the same three requirements as the Binomial distribution, and differs in what is counted,
  - X counts the number of trials up to and including the k^{th} success (where success might be defined as a head occurring after a continuous sequence of tails).
    Another process that produces values having a Negative Binomial distribution is randomly drawing from a mixture of Poisson distributions, where the mean of the mixture of Poisson distributions has a Gamma distribution,
    This distribution is a generalised version of the Geometric distribution (which is based on the probability of observing the first success on the n^{th} trial).
- Poisson distribution: for a random variable X,
  1. the process involves independent events,
  2. only one event can happen at any time,
  - X counts the number of events that occur within a specified time.
    The Poisson distribution is complete described by one parameter (λ, the distribution mean): $P(\lambda)$,
    The sum of two independent Poisson distributions $P(\lambda_1)$ and $P(\lambda_2)$ is the Poisson distribution $P(\lambda_1 + \lambda_2)$.
    The Binomial and Poisson distributions are related in that as $n \to \infty$ and $p \to 0$, then $B(n, p) \to P(np)$, i.e., The Poisson distributions is a limit case of a Binomial distribution having a very low probability of success over a long period.

Continuous distributions: commonly encountered continuous distributions include the following (see Figure 4.6):

In all but one case the generating process clusters the values around a single peak.

- Uniform distribution: all values between the lower and upper bounds of the interval have an equal probability of occurring, i.e., no value is more likely to occur than any other. For discrete values between 1 and n the probability of any value occurring is $\frac{1}{n}$.

  One process that generates a uniform distribution is a random number generator, such as calling the `runif` function.

- Normal distribution: this can be generated by adding together contributions from many independent processes, a consequence of the Central limit theorem. This distribution crops up with great regularity, it has a mathematical form that is much easier to analytically manipulate than many other distributions resulting in it being widely used before computers reduced the need for analytic solutions to problems. This distribution is described by its mean and variance.

  While the Normal is the result of adding contributions from many independent processes, it is not true to say that adding contributions from many different kinds of processes will result in this distribution, similarly for multiplicative contributions and a lognormal distribution. For instance, given the right conditions, adding values drawn from many different Poisson distributions can result in a Negative Binomial distribution, a Geometric distribution and many other distributions,[320]

- Lognormal distribution: the logarithm of a Normal distribution and it can be thought of as being generated by multiplying together the sum of contributions from many independent processes;[394] samples drawn from a Lognormal distribution can produce a straight line, over some of their range, when plotted using log-log axis,

- Exponential distribution: generated by a memoryless process, e.g., when the waiting time for an event to occur is independent of the amount of time that has passed since the last event. This is the continuous form of the Geometric distribution, and like it, is described by a single parameter.

  Over some of its range the exponential distribution is visually very similar to a Power law, which has led researchers to incorrectly claim that their sample fits a power law (a fashionable distribution to have one's sample following).

  The sum of a reasonably large number of independent exponential distributions has an Erlang distribution, e.g., the interval between incoming calls to a telephone exchange, where the interval between calls from an individual have an exponential distribution,

- Beta distribution: applies to processes where the explanatory variable is restricted to a finite interval, e.g., zero to one. This distribution is defined by two shape parameters.

- Gamma distribution (Γ is the Greek uppercase Gamma, the symbol often used to denote the Gamma function; γ is the lowercase Gamma): used to describe waiting times, e.g., `Gamma(shape=3, scale=2)` is the distribution of the expected waiting time (in some units) for three events to occur given that the average waiting time is 2 time units (yes, the `Gamma` function differs from most other distribution names in the base system by starting with an uppercase letter).

  When `shape=1`, the Gamma distribution reduces to the Exponential distribution.

  The Gamma distribution is the continuous equivalent of the Negative binomial distribution.

- Chi-squared distribution (sometimes written using χ, the Greek lowercase letter of that name): this is more often encountered in the mathematical analysis of statistics than as a distribution of a sample. A random variable has a chi-squared distribution with d degrees of freedom if it is produced by a process which generates the values: $Z_1^2 + Z_2^2 + \cdots + Z_d^2$, where Z_i are independent random variables having a Normal distribution.

  The chi-squared distribution is a special case of the gamma distribution.

- Weibull distribution: this distribution drops out as the solution to various problems in hardware reliability, e.g., time to failure, and is often used as the hazard function in survival analysis. The Exponential and Rayleigh distributions are special cases of the Weibull distribution,

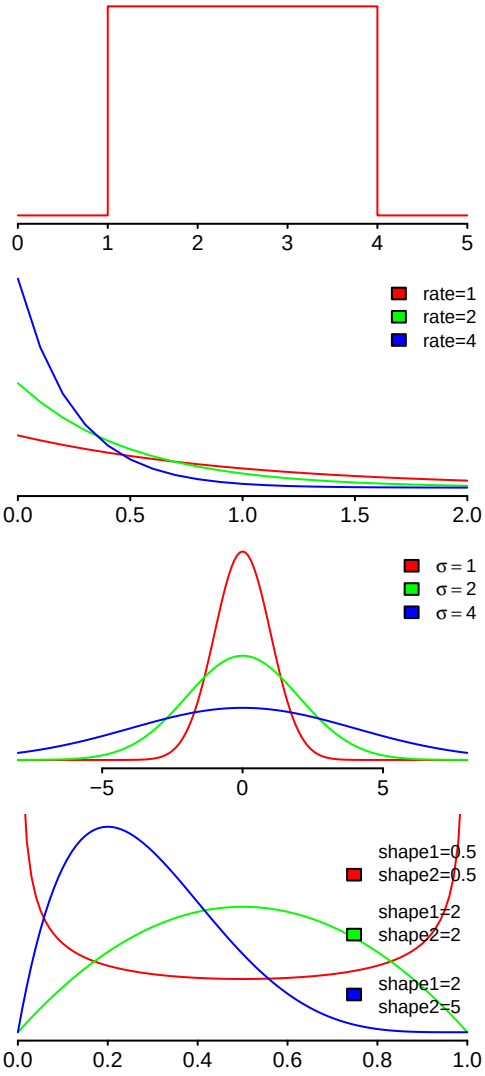


Figure 4.6: Commonly encountered continuous probability distributions (upper to lower: Uniform, Exponential, Normal, beta). code

- Cauchy distribution: this distribution is more famous for its unusual characteristics, e.g., having an undefined mean and variance (because of its very fat tail), than because of its uses. The density function for the average of two random variables each having a Cauchy density is a random variable with a Cauchy density; this self mapping is unique to the Cauchy distribution. One consequence is that if the error in a measurement has a Cauchy density, then the average of many measurements will not be more accurate than the individual measurements.

A chi-squared test is the traditional technique for testing whether data has a multinomial distribution, but has a higher false positive than an exact calculation. Use to a computer and the

4.2.1 Comparing probability distributions for equality

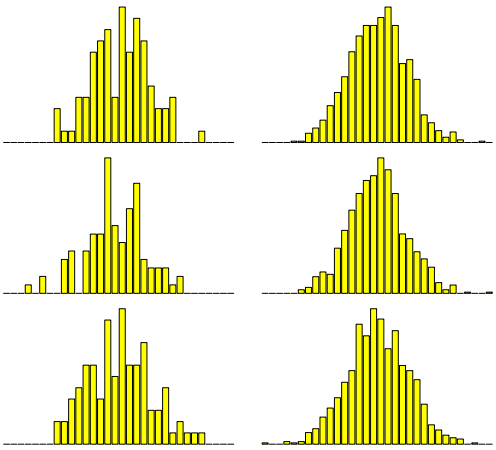


Figure 4.7: Samples of randomly selected values drawn from the same normal distribution (left: 100 points in each sample, right 1,000 points in each sample). code

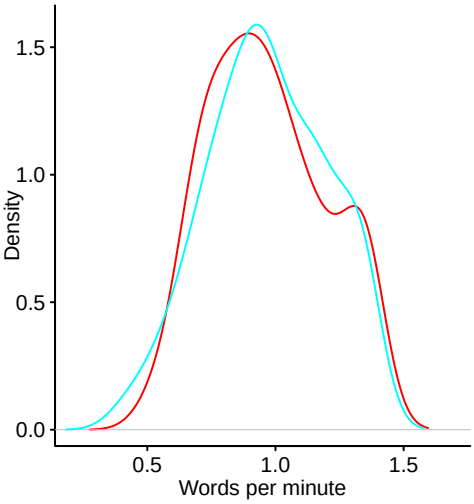


Figure 4.8: Reading rate for text printed using a serif (blue) and sans-serif (red) font, data has been normalised and displayed as a density. Data from Veytsman et al.[592] code

The question of interest is actually whether two or more samples are drawn from the same population, but the mathematics of sample comparison is framed in terms of comparing the measurable characteristics of probability distributions. These measurable characteristics include mean, standard deviation, skew and kurtoise (a probability distribution is defined in terms of a formula and parameters that get plugged into this formula; a formula is not a measurable).

Here we are interested in the distribution as a whole, not individual characteristics. The various comparison techniques are based on some measure of difference between the *shape* of the sample distributions.

As always visualization is a useful first step in obtaining information about whether two samples might be drawn from the same distribution. However, be warned that small datasets can produce visualizations showing little resemblance to the distribution from which they were drawn, as can be seen from Figure 4.7, where all the samples are drawn from the same Normal distribution.

A study by Veytsman and Akhmadeeva[592] measured subject reading rate, in words per minute, for text printed using a Serif or Sans Serif font. Words per minute is a discrete distribution and subject performance is likely to be similar, i.e., there will be many duplicates. Figure 4.8 shows a density plot of the normalised data.

The following tests are based on comparing the edf (empirical distribution function) of the sample values.

- The Anderson-Darling test is based on the largest difference between the edf of the two distributions, it uses weights to ensure that the tails of the distribution have as much influence as other parts of the distribution; it is possible to use this test to compare more than two distributions. While the Kolmogorov-Smirnov test is often encountered it has been found to be less sensitive than the Anderson-Darling test[549] because it primarily detects differences in the main body of the distribution, rather than over the complete range of values.

  The `ad.test` function in the `kSamples` package implements the Anderson-Darling test for two or more samples.

  The `ks.test` function, part of the base system, implements the Kolmogorov-Smirnov test; other implementation include the `ks.test` function in the `dgof` package whose interface is the same but includes support for discrete distributions.

  Samples drawn from a continuous distribution are very unlikely to contain identical values and many implementations warn if a sample contains duplicate values.

- The Cramér-von Mises test is based on summing (the square of) differences between edfs, rather than using a single maximum value, and can be more powerful against a large class of alternative hypothesis.[26]

  The `cvm.test` function in the `dgof` package implements the Cramér-von Mises test.

The choice of statistical test depends on whether differences over the range of values in the samples are of interest, whether tail values are uninteresting (perhaps because there are few measurements in the tail and so what is there is noisey) or the amount of difference between samples is the primary differentiator.

Support for comparing samples drawn from discrete distributions is provided by: the `WRS` package (on Github) implements a version of the Kolmogorov-Smirnov test (the `ks` function) that supports discrete data and also the `bmpmul` function which uses the Brunner-Munzel test; the `ks.test` function in the `dgof` package.

The following code performs various tests checking whether the two sample are likely to have been drawn from the same population (see rexample[tb104veytsman-dist.R]):

```
library("dgof")
library("kSamples")
library("WRS")

# From WRS
ks(serif$Standard_WPM, sansserif$Standard_WPM)
# In fact unscaled measurements give the same result, i.e., not different
ks(serif$WordsPerMinute, sansserif$WordsPerMinute)

dgof::ks.test(serif$Standard_WPM, ecdf(sansserif$Standard_WPM))

# From base system
ks.test(serif$Standard_WPM, sansserif$Standard_WPM)

# Only applicable to continuous distributions
ad.test(serif$Standard_WPM, sansserif$Standard_WPM)
```

In this case the hypothesis that the samples are drawn from different distributions is rejected.

Practical manual techniques often require that samples be drawn from a Normal distribution and this test commonly performed by other people.[ii]

The result of testing whether a small sample is drawn from a Normal distribution has a high degree of uncertainty. Figure 4.9 was obtained by testing samples of various sizes, all drawn from the same distribution (e.g., a call to `rexp`), using the `shapiro.test` function. The y-axis shows the probability of the Shairo-Wilk test detecting (p-value < 0.05) that the sample values are not drawn from a Normal distribution; for the case when the values are drawn from a Normal distribution (e.g., a call to `rnorm`) the y-axis gives the probability of this fact not being detected.

Note that many points may be needed to distinguish a difference when one exists. For instance, two samples of 150 points are needed to obtain a 95% confidence level, using `ad.test`, that the samples are drawn from different distributions, when one sample is drawn from an Exponential distribution and the other from a Normal distribution (550 points are needed when the samples are drawn from Normal and Uniform distributions); see rexample[ad-check.R].

There is no guarantee that the values in a sample will have a distribution that even closely resembles any of the known probability distributions.

A study by Berger, She, Czarnecki and Wąsowski[53] investigated the use of feature macros used in the configuration of software product lines. Figure 4.10 shows the number of conditionally compiled sections of source code that were dependent on a given number of feature macros.

The Cullen and Frey graph, produced by `descdist`, shows that the characteristics of neither sample are close to matching any of the common discrete distributions. A Kolmogorov-Smirnov test considers them to be sufficient different that they are likely to have been drawn from different distributions (see rexample[2010-berger.R]).

Samples may appear to have a similar shape, but have different mean values. Technically, samples with different mean values (or standard deviations) are considered to be drawn from different distributions. There may be theoretical reasons for believing that samples have been generated by the same processes and normalizing mean values (or even variance) is of enabling the shape of the sample distributions to be compared.

A study by Zhu, Whitehead, Sadowski and Song[630] counted the number of various kinds of statements in a corpus of C, C++ and Java programs (approximately 100 programs, around 10 million lines, for each language). Figure 4.11 shows the distribution of occurrence (expressed

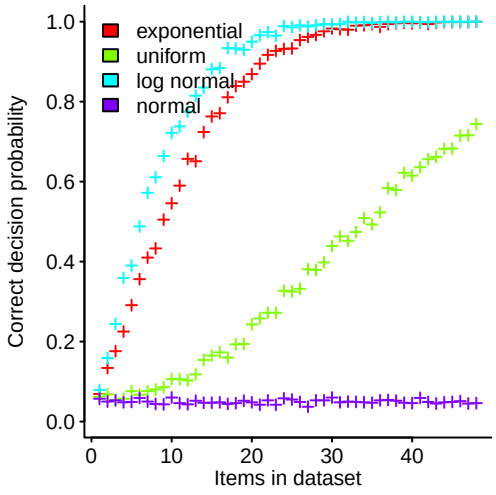


Figure 4.9: Probability, with 95% confidence, that `shapiro.test` correctly reports that samples drawn from various distributions are not drawn from a Normal distribution, and probability of an incorrect report when the sample is drawn from a Normal distribution. code

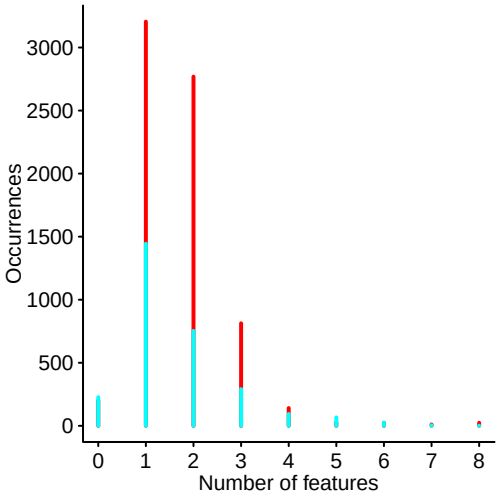


Figure 4.10: Number of conditionally compiled code sequences dependent on a given number of feature macros (red overwritten by blue: Linux, blue: FreeBSD). Data from Berger et al.[53] code

[ii] Readers of this book know about more powerful techniques that do not have this precondition.

as a density on the y-axis) of various statements (expressed as a percentage on the x-axis) over the programs measured; a different color for each language, figure out which is which before looking at the code.

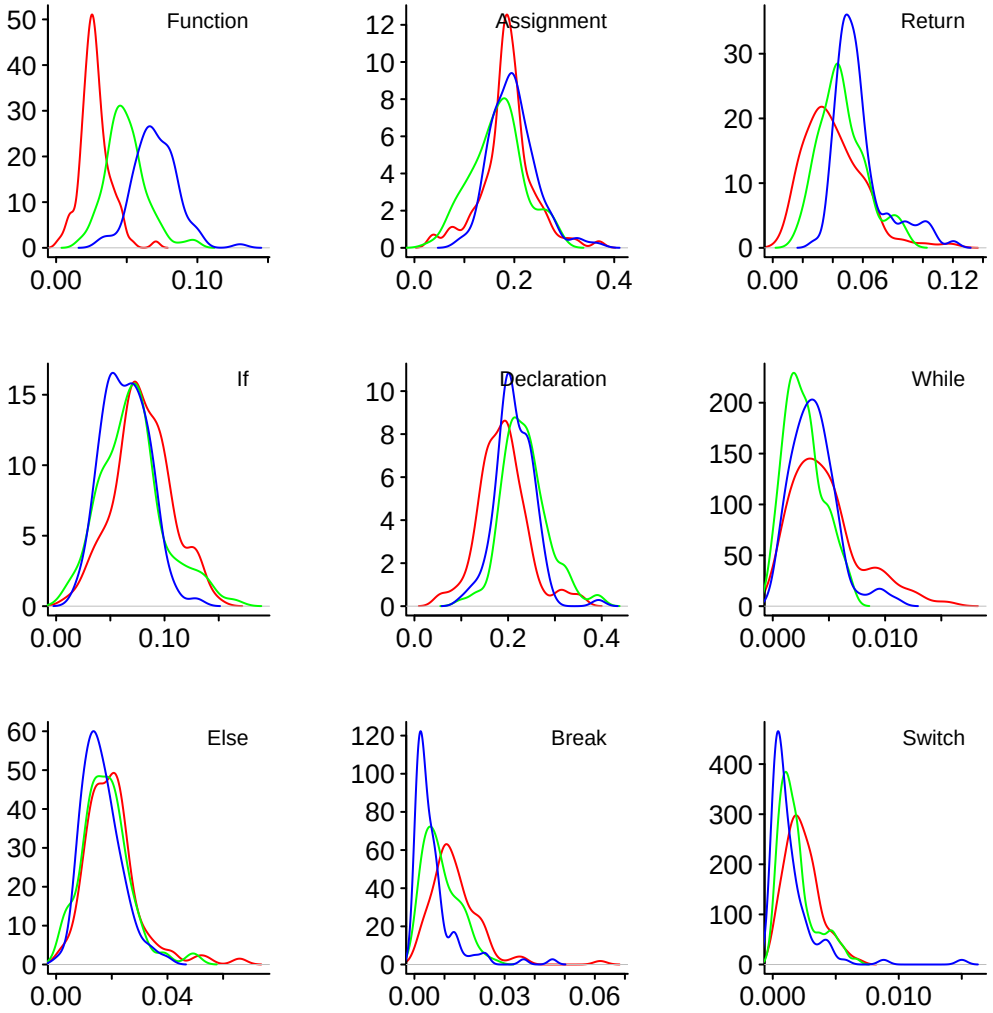


Figure 4.11: Percentage occurrence of statements for each of 100 or so C, C++ and Java programs, plotted as a density on the y-axis. Data from Zhu et al.[630] code

Developers are pushed towards using particular statements by the characteristics of the application/language/algorithm. Differences in the probability of various kinds of statements being used, over a sample of programs written in various languages, is evidence that language has an impact on what code gets written (either because particular kinds of applications are written using a given language, particular algorithm selection is influenced by language, or the impact of differences in language semantics).

Might two or more of the languages be said to have the same distribution of if-statement and/or assignment-statement usage? The interactions between different statements makes the analysis non-trivial.

The takeaway from this section is that for small sample sizes distribution comparison produces unreliable answers and for large samples comparison may be complicated.

Comparison of particular characteristics of sample distributions, e.g., sample means, is covered in the chapter on running experiments.

4.3 Fitting a probability distribution to a sample

Given a sample of values of a single attribute, which of the known, supported by R,[109] probability distributions is the best fit?

There is no universal best-test statistic for goodness-of-fit of a sample to a probability distribution. The performance of the available tests depends on the (unknown) distribution from which the sample was drawn.[544]

The Normal distribution is often the default answer given, when people are asked about the distribution of a sample. There are several reasons for this, including: historically many of the techniques were designed to be performed by a human calculator were derived from theory that assumed normally distributed data (which often appeared to work reasonably well when the data only approximated a Normal distribution) and a misunderstanding of what the Central Limit theorem is about driving a belief that measurements of a complex process provides the mixing needed to produce a Normal distribution.

As always, knowledge of the processes driving the production of the measured values can be very useful. For instance, measurements of arrival times that are driven by a Poisson process will result in inter-arrival times that are exponentially distributed, values created via the multiplicative effect of many contributions may have a Lognormal distribution and a preferential attachment process often results in links or what they link following a power law.

If there is no theoretical justification for a particular distribution, limiting the selection process to those distributions having some degree of name recognition is likely make the final choice an easier sell to readers. For instance, the Delaporte distribution[iii] might happen to fit a particular sample slightly better than the Negative Binomial distribution, but its lack of name recognition will mean that extra effort will have to be invested on justifying its use.

A study by van der Meulen[583] posted the $3n+1$ problem on a programming competition website: 95,497 solutions were submitted and van der Meulen kindly sent me a copy of these solutions (11,674 solutions were written in Pascal, the rest written in C). The $3n+1$ problem is to write a program that takes a list of integers and outputs the *length* of each value, where length is the number of iterations of the following algorithm:

```
for input integer ++pass:[n]++;
   while (n != 1)
      n = (is_even(n) ? n/2 : n*3+1);
```

Which distribution is a good approximation to the number of lines of code contained in the programs submitted as answers to this problem?

The first step of visualizing the sample provides basic information about the shape of the distribution, e.g., decreasing/increasing, single/multiple peak, symmetric/skewed or appearing to be nothing but random noise.

One technique for narrowing down the list of possible distributions is to plot its Cullen and Frey graph. The `descdist` function in the `fitdistrplus` package plots this graph and returns some descriptive distribution characteristics of the values (mean, median, sd, skewness and kurtosis). Skew and kurtosis are not reliable estimators and `descdist` includes an option to create and test bootstrap samples.

The blue and yellow points in Figure 4.12 denote the sample and various bootstrap results for the $3n+1$ program lengths, assuming a continuous distribution (the average number of lines is large enough that the difference between discrete/continuous is likely to be very small).

```
library(fitdistrplus)

# Default is to check continuous distributions
# dummy=descdist(li, discrete=TRUE, boot=500)
dummy=descdist(li, boot=500)
```

The `fitdist` function[iv] in the `fitdistrplus` package can be used to fit a distribution to the data, i.e., find values of the specified distribution's parameters, such as mean and variance, that minimise some measure of goodness-of-fit (the AIC of the fit is returned). The `gamlss` package supports a wider range of distributions (the help information for the `gamlss.family` function lists them) that can be used by `fitdist` to fit data.

Figure 4.13 shows fits for the Normal, Poisson, Lognormal and Negative binomial distributions.

```
library(fitdistrplus)

tp=fitdist(li, distr="pois")
```

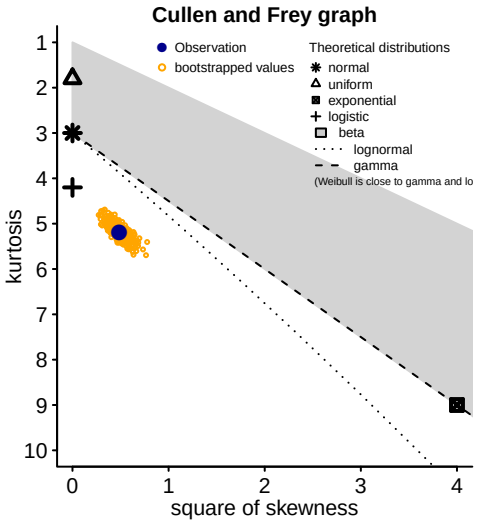


Figure 4.12: A Cullen and Frey graph for the $3n+1$ program length data. Data kindly provided by van der Meulen.[583] code

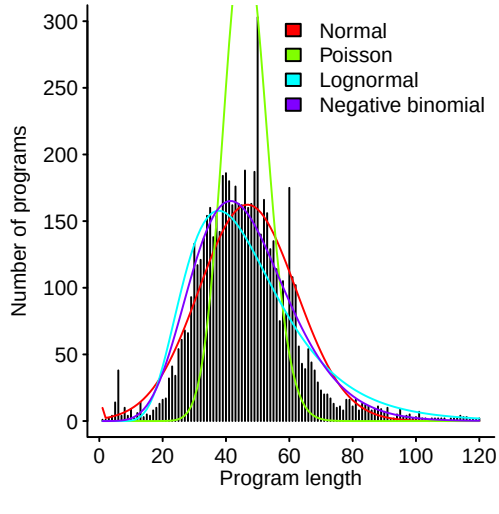


Figure 4.13: Number of 3n+1 programs containing a given number of lines and four distributions fitted to this data. Data kindly provided by van der Meulen.[583] code

[iii] A compound distribution derived from a Poisson distribution whose mean has a shifted Gamma distribution.

[iv] The `MASS` package contains the `fitdistr` function and the `gamlss` package contains the `fitDist` function both of which fit distributions to data.

```
tn=fitdist(li, distr="norm")
tln=fitdist(li, distr="lnorm")
tnb=fitdist(li, distr="nbinom")

# gofstat is a way of getting all the values used for plotting
theo_vals=gofstat(list(tn, tp, tln, tnb), chisqbreaks=1:120,
                        fitnames=c("Normal", "Poisson",
                               "Lognormal", "Negative binomial"))

plot_distrib=function(dist_num)
{
lines(theo_vals$chisqbreaks, head(theo_vals$chisqtable[, 1+dist_num], -1),
                col=pal_col[dist_num])
}

plot(theo_vals$chisqbreaks, head(theo_vals$chisqtable[, 1], -1), type="h",
             xlab="Program length", ylab="Number of programs\n")
plot_distrib(1)
plot_distrib(2)
plot_distrib(3)
plot_distrib(4)
```

The large spike at 50 lines might be caused by a group of solutions all doing the same thing but with different statement orderings or multiple submissions derived from a common solution.

Based on minimizing AIC the Normal distribution is the best fit, with the Negative binomial distribution a close second. Should either distribution be chosen as the best fitting, or is it worthwhile attempting to fit other distributions? The answer depends on what the fitted distribution will be used for, e.g., making predictions or building models. Jumping to any conclusions based on one data-point (i.e., set of length measurements for one problem) is always problematic.

4.3.1 Zero-truncated and zero-inflated distributions

While zero is a very common lower bound for measurement values, other lower bounds occur, e.g., the number of minutes to complete a task (the zero time tasks, that are never started, are not measured). Many potentially useful distributions describe variables whose values start at zero, e.g., the Poisson distribution.

It is possible to adjust the equations that describe zero-based distributions to have a non-zero lower bound. Rebasing a distribution to start at one (rather than zero) is the common case and after such an adjustment the distribution is said to be *zero-truncated*, e.g., *zero-truncated Poisson distribution*.

The `gamlss.tr` package contains functions that support the creation of zero-truncated (or truncation to the right or left of any value) distribution functions. The following call creates a set of functions relating to the zero-truncated type II Negative binomial distribution; the name of the created function is `NBIItr` and like other distribution functions in R there the associated density, distribution, quantile and random functions are obtained by prefixing the letters `d`, `p`, `q` and `r` respectively to `NBIItr`:

```
library(gamlss)
library(gamlss.tr)

gen.trun(par=0, family=NBII) # Bring various functions into existence
```

The 7Digital data[482] (covered in more detail in the section on Agile) contains information on 3,238 features implemented between April 2009 and July 2012; the information consists of three dates (Prioritised/Start Development/Done) from which a non-zero duration can be calculated.

The Cullen and Frey graph suggests a negative binomial distribution as a good fit (see rexample[???]).

The functions returned by `gen.trun` do not have a form that can be used in calls to the `fitdist` function. The `gamlss` function in the `gamlss.tr` package has a special form for handling these created functions, as shown in the following call (where `day_list` contains

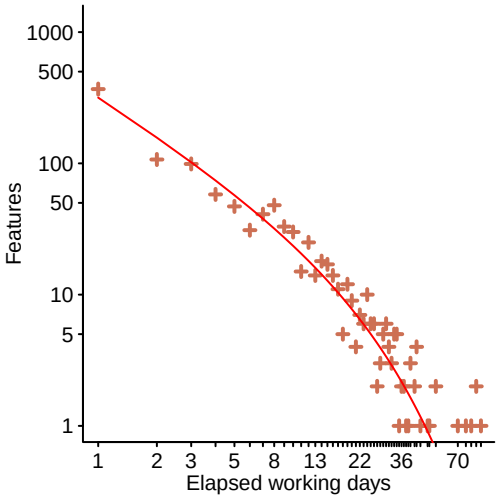


Figure 4.14: A zero-truncated Negative Binomial distribution fitted to the number of features whose implementation took a given number of elapsed workdays: first 650 days

the list of values and `NBIItr` was created by an earlier call to `gen.trun`). The following code was used in the production of Figure 4.14:

```
library(gamlss)
library(gamlss.tr)

g.NBIItr=gamlss(day.list ~ 1, family=NBIItr)

NBII.mu=exp(coef(g.NBIItr, "mu"))       # get mean coefficient
NBII.sigma=exp(coef(g.NBIItr, "sigma")) # standard deviation

plot(table(day.list), log="xy", type="p", col=point_col,
                xlab="Elapsed working days", ylab="Features\n")

lines(dNBIItr(1:93, mu=NBII.mu, sigma=NBII.sigma)*length(day.list),
                                                    col="red")
```

One process generating values having a Negative binomial distribution is based on a mixture of Poisson distributions whose means have a Gamma distribution. It is possible to generate other distributions by combining a mixture of Poisson distributions, are any of these a better fit of the data? The Delaporte distribution sometimes fits very slightly better and sometimes slightly worse (see the source code chapter for details); the difference is not large enough to warrant switching from a relatively well-known distribution to one that is rarely covered in text books or supported in software; if data from other projects is best fitted by a Delaporte distribution then it may be worthwhile spending time analysing how this distribution might be a better model of project scheduling.

If the processes generating these values involve a mixture of Poisson distributions, then there unlikely to be a single subprocess responsible for a large percentage of the behavior, many subprocesses are involved.

Sometimes count data measurements contain many more zero values than are expected from the discrete distribution that the generating process is believed to be following. Two of the processes that can generate extra zeroes include:

- a subset of the population is not involved in the process measured and so always has a zero value. This situation can be modeled by what is known as *zero-inflated model*, which combines a model of always zero vs. maybe non-zero values and a model of maybe non-zero values.

  The `gamlss` package...

- the measurements involve two processes, one in which the values are zero or non-zero, and the other where values are always non-zero.

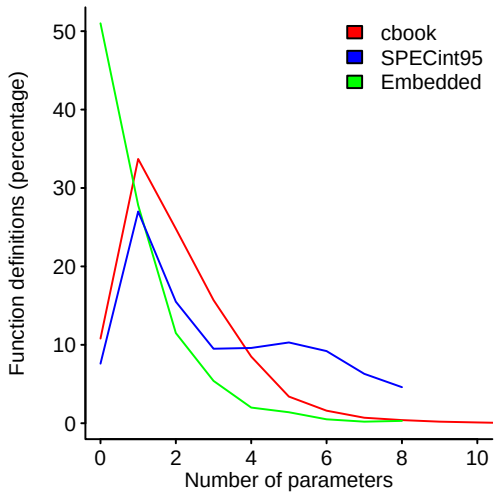


Figure 4.15: Percentage of function definitions in embedded applications, the SPECint95 benchmark, and the translated form of C source benchmark programs declared to have a given number of parameters. Data for embedded and SPECint95 kindly supplied by Engblom,[161] C book data from Jones.[299] code

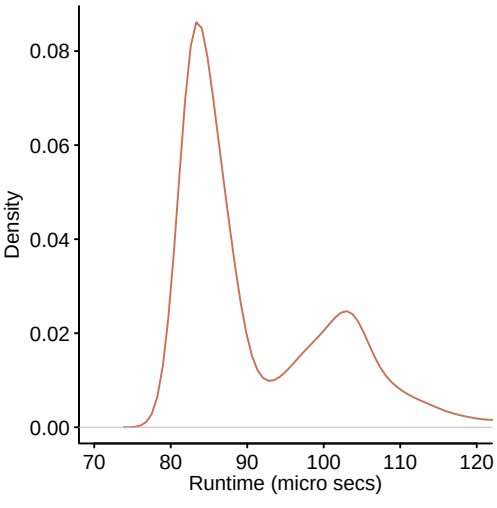
Sharp increases in value at the origin are not necessarily generated by a second process. In Figure 4.15 the green line continues to rise, suggesting that another process is causing functions with no parameters to be created. An analysis shows that a Poisson distribution provides a reasonable fit to both the green and red lines (with λ equal to 0.8 and 1.98 respectively).

Just need some data... have emailed a few people...

4.3.2 Mixtures of distributions

Sometimes sample values are generated by two or more distinct processes, resulting in measurements that appear to be drawn from two or more distinct distributions, e.g., a plot shows multiple peaks. A model built using a mixture, or weighted sum, of distributions is known as a *finite mixture model* or just a *mixture model*; a continuous mixture of distributions is known as a *compounded distribution* (the Negative Binomial distribution is a common compounded distribution).

The `mixtools` and `rebmix` packages contain functions for fitting samples drawn from two or more distributions, but they have to be from the same distribution family, e.g., multiple Normal distributions. The two packages differ in the structure of their API, e.g., one having many functions and the other one main function taking many arguments (neither would win a prize for user interface design).



A study by Hunold, Carpen-Amarie and Träf[281] investigated the impact of external factors on the performance of an MPI micro-benchmark. Figure 4.16 shows the runtime variation of two MPI calls, with each having two distinct peaks.

The lower Figure 4.16 shows two peaks with each appearing to be symmetrical and perhaps a mixture of two Normal distributions is a good fit. Figure 4.17 shows the two distributions fitted by a call to the `normalmixEM` function (in the `mixtools` package).

```
library("mixtools")

scan_dist=normalmixEM(fig1_Allreduce$time)

plot(scan_dist, whichplots=2, main2="", col2=pal_col,
       xlab2="Time (micro secs)", ylab2="Density\n")
```

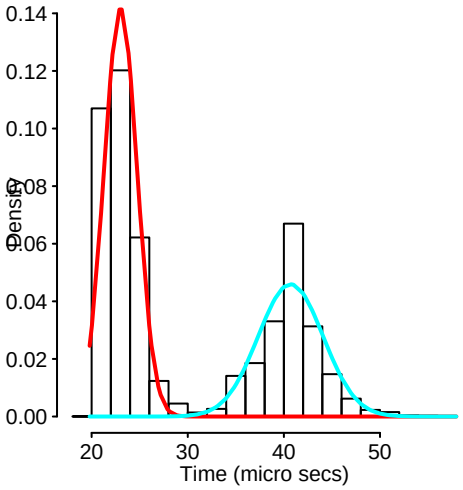


Figure 4.17: Mixture model fitted by the `normalmixEM` function to the performance data from calls to `MPI_Allreduce`. Data kindly supplied by Hunold.[281] code

A call to `summary` returns the parameters of the fitted model. The first row (prefixed by `lambda`) is the fraction contributed by each distribution, followed by the mean, standard deviation and log likelihood (rather than AIC): code

```
number of iterations= 19
summary of normalmixEM object:
          comp 1    comp 2
lambda  0.611002  0.388998
mu     23.011364 40.703294
sigma   1.720528  3.378664
loglik at estimate:  -28873.39
```

A plot of a sample drawn from a mixture of distributions does not always show visually distinct peaks.[v]

A study by Kaltenbrunner, Gómez, Moghnieh, Meza, Blat and López[317] analysed the pattern of user activity of the Slashdot technical community news site. The black line in Figure 4.18 shows the density of the number of accesses to one article in each minute after first publication (a total of 1,567 accesses).

A possible explanation for the multiple upticks in number of accesses is the article being linked to from other web sites, driving a fresh batch of readers to Slashdot; the number of linking websites is not known, but this explanation sounds reasonable enough to build a mixture model containing a relatively large number of different distributions. Which mixture of distributions might best fit the access times of this Slashdot article? The Poisson distribution is often used to model arrival times and is the obvious first choice, but in this particular case turns out not to provide the best fit.

Figure 4.18 shows several Normal distributions fitted to data, on a log scale, using functions from the `rebmix` and `mixtools` packages. The algorithms used by packages do not guarantee to find the globally optimal solution and differences in the mix of distributions selected can occur because of differences during the search process.

```
library("rebmix")

slash_mod=REBMIX(Dataset=list(data.frame(users=log(slash$users))),
                 Preprocessing="histogram", cmax=5,
                 Variables="continuous", pdf="normal", K=7:45)

plot_REBMIX_dist=function(dist_num)
{
y_vals=dnorm(x_vals, mean=as.numeric(slash_mod$Theta[[1]][2, dist_num]),
                  sd=as.numeric(slash_mod$Theta[[1]][3, dist_num]))
lines(x_vals, slash_mod$w[[1]][1, dist_num]*y_vals, col=pal_col[dist_num])
}

plot(work_den, main="", xlim=c(0, 10), ylim=c(0, 0.36),
       xlab="", ylab="Access density\n")
plot_REBMIX_dist(1)
plot_REBMIX_dist(2)
```

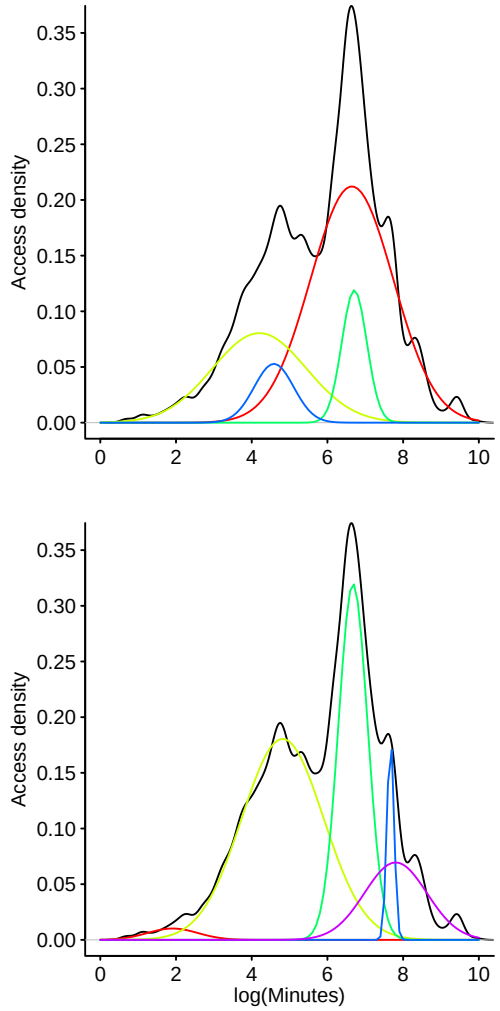


Figure 4.18: Density plot of accesses to one article on Slashdot, in minutes since its publication. The distinct Normal distributions (colored and fitted to the log of the data) contained in the mixture models fitted by the `REBMIX` (upper) and `normalmixEM` (lower) functions. Data kindly supplied by Kaltenbrunner.[317] code

[v] If f_1 and f_2 are normal densities with means μ_1 and μ_2, respectively, and both have the same variance σ^2, then the mixture density $f = 0.5f_1 + 0.5f_2$ will have a single peak if, and only if, $abs(\mu_2 - \mu_1) \leq 2\sigma$.

```
plot_REBMIX_dist(3)
plot_REBMIX_dist(4)
```

Fitting a Normal distribution to log scaled data means that the sample actually has a Lognormal distribution. Is the Lognormal distribution a good representation of the processes driving readers to access Slashdot articles? As always in model building the answer depends on what the model will be used for. If the purpose is to make predictions, then the accuracy of prediction is of more interest than any underlying assumptions. If the purpose is to understand what is going on, then a theory that contains processes generating Lognormal distributed behavior is needed.

It can take a lot of analysis over many years to settle on the distribution, or combinations of distributions, that best describes the measured properties of some system. The study of file-system characteristics[7] is an example of how researchers' ideas and models changed over time,[151,288,395] becoming more sophisticated as more data became available, from various platforms,[133] and more analysis was carried out.

4.3.3 Heavy/Fat tails

Heavy tailed is the term used to describe distributions where the majority of measured events occur a long way from the mean value (*fat tails* and *long tail* are also used).[vi] When the 80/20 rule applies the distribution is heavy tailed and the frequency with which this rule is encountered suggests that such data is not rare.

The `poweRlaw` package supports operations involving a variety of heavy tailed distributions, including power laws.

The Pareto distribution is the mathematical name of a particular instance of a heavy tailed distribution (sometimes going by the name *power law* in popular culture); Zipf's law is a particular instance of this distribution.

The mean of a heavy tailed distribution may not exist (because it is infinite). Obviously any finite dataset has a finite mean and if a sample is drawn from a heavy tailed distribution, its mean value will jump around erratically.

It is more difficult to narrow down which distribution might best fit a sample drawn from a heavy tailed distribution (because several fit equally well), compared to one without a heavy tail, because measurements are spread out rather than clumped around a central location.

Figure 4.19 is from a survey[288] of file sizes and shows that a small percentage of files account for most of the disk spaced occupied (the vertical line meets the bytes line where 89.9% of disk space has yet to be consumed and the files line where 12.5% of files still remain to be accounted for). Another way of describing the situation is to say that there is a mass/count disparity, i.e., a few files occupy most of the space.

Care needs to be taken to separate out concepts that are popularly associated with power laws, e.g., *scale invariant*, that are a property of the distribution, not the generating process. The process generating data that fits a power law can be remarkably random, e.g., the length of words in text produced by monkeys typing.[121]

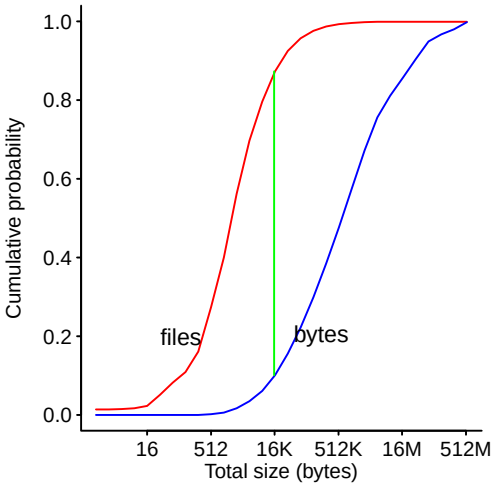


Figure 4.19: Cumulative probability distribution of files size (red) and of number of bytes occupied in a file system (blue). Data from Irlam.[288] code

4.4 Markov chains

A finite state machine (FSM) is a machine represented by a set of distinct states connected by edges denoting the possible transitions that can occur when a given event occurs, such as when a particular character is input (FSMs are deterministic).

A Markov chain (MC) also a machine represented by a set of distinct states connected by edges, but the possible transition is chosen at random based on the transition probability of each edge (the transition probabilities out of any state, that is not an absorbing state, add to one); the next state only depends on the current state, i.e., the system is memoryless.

A Markov chain is a *discrete-time Markov chain* (DTMC) if the transition between states occurs at fixed time intervals; if the time interval between state transitions is not fixed, the

[vi] The term *sub-exponential* is sometimes used to describe tails that decay slower than exponential and *super-exponential* for tails that decay faster than exponential.

Markov chain is a *continuous-time Markov chain* (CTMC) (the memoryless requirement means that transition times must have an exponential distribution). If the transition time depends on how long the system has been in the current state, it is a semi-Markov process (SMP).

Multi-state models, the `msm` package...

Finite state machines provide a useful abstraction for modelling user interfaces. A study by Oladimeji[424] investigated the user interface of the Alaris volumetric infusion pump (a medical device used for controlled automatic delivery of fluid medication, or blood transfusion, to patients); the user interface includes 14 buttons and an LCD display. Figure 4.20 shows the available transitions between states.

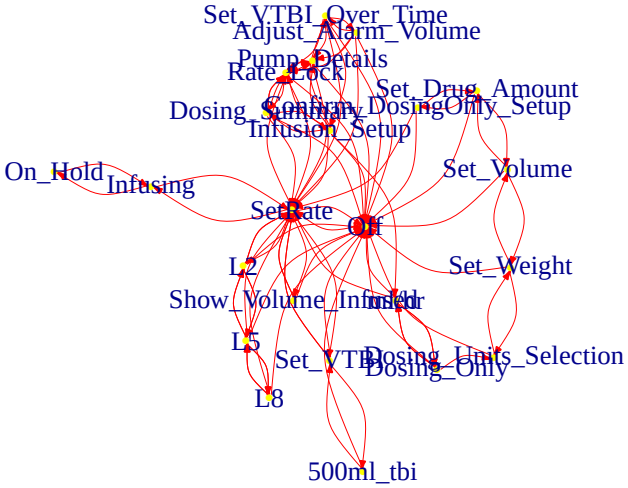


Figure 4.20: Graph of available state transitions for Alaris volumetric infusion pump (the button presses that cause transitions between states are not shown). Data kindly supplied by Oladimeji.[424] code

Representing the FSM as a control flow graph, functions in the `igraph` package can be used to answer questions such as: the maximum number of button presses needed to get to any state (12; the `path.length.hist` function returns a count of all possible path lengths) and the average number of presses to transition between any two states (4; using the `average.path.length` function).

If the behavior of a system (that can be represented using a FSM) is monitored, the probability of occurrence of every transition between states can be calculated. If the behavior represents typical user interaction with the system, then the probabilities can be used to create a Markov chain for this typical behavior.

A study by Tarasov, Mudrankit, Buik, Shilane, Kuenning and Zadok[562] used data on the lifetime of source files in various systems, such as the Linux kernel, to generate realistic filesystem contents (for deduplication analysis). Figure 4.21 shows a Markov chain for the source files in the Linux kernel (from being Initialised to new, through modified/unmodified to deleted and reaching the Terminal state). The measurement snapshot occurred at each of the 40 releases between versions 2.6.0 and 2.6.39, with an average of 23k files per snapshot; the time between releases is roughly constant, so this might be considered a discrete-time Markov chain.

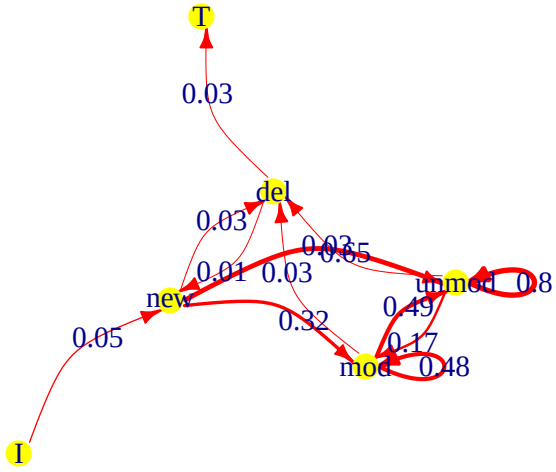


Figure 4.21: Discrete-time Markov chain for created/modified/deleted status of Linux kernel files at each major release from versions 2.6.0 to 2.6.39. Data from Tarasov.[562] code

```
library("igraph")

atc=read.csv(paste0(ESEUR_dir, "probability/atc12-gra.csv.xz"), as.is=TRUE)
atc_gra=graph.data.frame(atc, directed=TRUE)

V(atc_gra)$frame.color=NA
V(atc_gra)$size=12 ; V(atc_gra)$color="yellow"
E(atc_gra)$arrow.size=0.5 ; E(atc_gra)$color="red"
E(atc_gra)$weight=E(atc_gra)$linux
E(atc_gra)$label=E(atc_gra)$weight/100

# layout.lgl outperforms the default layout for this graph
plot(atc_gra, edge.width=0.3*sqrt(E(atc_gra)$weight),
                        edge.curved=TRUE, layout=layout.lgl)
```

The `graph.data.frame` function assumes there is a link between the row values in two columns (`from` and `to` vertices) and builds a graph using this information. The `V` and `E` functions access the vertices and edges of the graph and various attributes can be set and may be subsequently used by `plot`.

The algorithm used by `plot` to layout a graph makes use of randomization, which means that the layout returned by every call will be different.

4.4.1 A Markov chain example

A study by Perugupalli[223,449] investigated the reliability of gcc based on the reliability of its major subsystems. Information on the probability of a subsystem experiencing a failure was obtained using the regression suite for gcc version 3.3.3 (which contains tests for 110 faults present in gcc version 3.2.3, out of 2,126 tests, of which 55 were traced back to the source code of a single subsystem; the others faults involved multiple subsystems). The researchers did not attempt to analyse failures involving more than one subsystem and assumed that subsystems fail independently of each other.

Subsystems were identified by instrumenting gcc to count the number of calls between pairs of functions performed while executing the regression suite (this is not really Markov chain-like behavior because the called functions return, which is not transition-like behavior). The 1,759 traced functions were manually assigned to one of 13 internal subsystems (e.g., parsing, tree optimization and register allocation),

The reliability of gcc version 3.2.3 might be estimated using:

$$R = 1 - \frac{F_c}{T_c} = 1 - \frac{110}{2126} = 0.948$$

where F_c is the number of source files that it did not correctly compile and T_c is the total number of files compiled.[vii]

This approach has the advantage of being simple to calculate, but it does not provide any information on the impact of individual subsystems on overall reliability, for instance, what is the sensitivity of overall system reliability to behavioral changes to one subsystem?

The probability of reaching subsystem n from subsystem 1 after k transitions is Q^k (where Q is the matrix of transition probabilities). Summing over all transitions (using an infinite upper bound for the total number of transitions simplifies the mathematics) we get:[573]

$$S = \sum_{k=0}^{\infty} Q^k = (I - Q)^{-1}$$

where: I is the identity matrix. The expression $(I - Q)^{-1}$ is easily calculated (i.e., inverting the result of a matrix subtraction).

The matrix S is known as the *fundamental matrix* and can be used to calculate a variety of properties of systems modeled by the Markov chain.

The composite and hierarchical methods are two techniques for combining information on subsystem usage (i.e., subsystem transition probabilities and subsystem reliability calculated using the above formula) to obtain a value for the reliability of a complete system:

[vii] The calculated reliability is very low because it is based on compiling a test suite of short code samples designed to reveal faults.

- composite method:[107] calculates the probabilities a successful transition between each subsystem by multiplying the transition probabilities of each subsystem by the probability of the subsystem executing successfully. These individual successful transition probabilities are used to calculate the successful transition probability from the initial subsystem to the final subsystem (i.e., the systems fundamental matrix). The estimated reliability obtained for gcc is 0.9972,[viii] (see rexample[gcc-reliability.R]).

- hierarchical method: if R_i is the reliability of a subsystem, the probability of all executions of that subsystem being successful is $R_i^{N_i}$, where N_i is the number of transitions to subsystem i during one execution of the system. Assuming that subsystems fail independently, the expected value of the system reliability is:

$$R = E\left[\prod_{i=1}^{n} R_i^{N_i}\right]$$

Assuming subsystems are highly reliable and the variance in the number of subsystem transitions is very small, the first order Taylor approximation can be used:

$$R \simeq \prod_{i=1}^{n} R_i^{V_i}$$

where $V_i = E[N_i]$ is the expected number of times a transition occurs to subsystem i during a single execution of the complete system; V_i is obtained by solving:

$$V_i = q_i + \sum_{j=1}^{n} V_j p_{ji}$$

where q_i is the probability that execution starts with subsystem i and the p_{ji} are obtained from the subsystem transition probability matrix (see rexample[gcc-reliability.R]).

There is a plentiful supply of books discussing the use of Markov chains to solve problems. The kinds of problems attacked using a Markov chain approach don't occur very often in this book.

4.5 Social network analysis

The popularity of web based social networks has made the mathematics of social network analysis a fashionable research topic. Unfortunately many of the results involve little more than claiming to have found a power law, with only pretty pictures and blustering hand waving to show.

An example of the kind of descriptive statistics encountered in a social network analysis appears in the ecosystem chapter and the section on regression modeling discusses building models based on network data.

Social networks are represented as graphs and the `igraph` package supports reading the common graph data representation formats, along with a wide range of operations and analysis on graphs.

Both FreeBSD and OpenBSD were forked from a common base and not only continue to share common code but faults fixed in one of them are often applied (some time later) to the other. A study by Canfora, Cerulo, Cimitile and Di Penta[93] analysed the developer's mailing lists for these systems to obtain information on what they called *Cross-System-Bug-Fixings*; the data contains information on 861 unique developers sending email and 1,062 unique developers receiving email. Figure 4.22 was produced using code very similar to that used earlier for the Markov chains.

[viii] If the 55 fault count used in this analysis is plugged into the formula used above, the reliability estimate is 0.974.

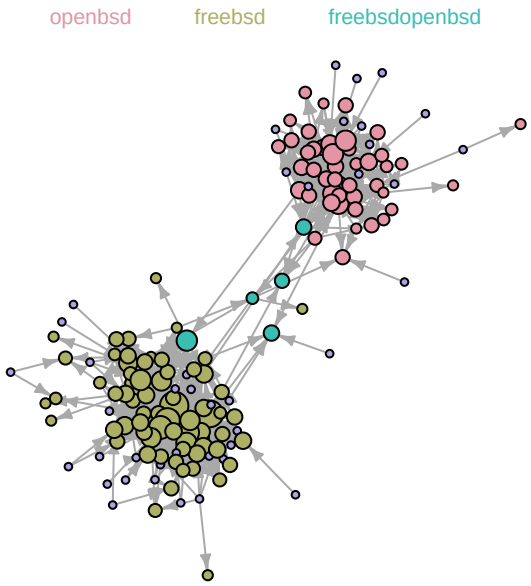


Figure 4.22: Directed graph of emails between FreeBSD and OpenBSD developers, plus a few people involved in both discussions, with developers who sent/received less than four emails removed. Data from Canfora et al.[93] code

Many real world linked collections contain subgroups (e.g., clusters of developers or related code modules) and there are a variety of algorithms for detecting these subgroups. Care needs to be exercised in interpreting the clusters returned by these algorithms as there may be many distinct high-scoring solutions and a clear global maximum may not exist.[213]

4.6 Combinatorics

The analysis of some systems makes it necessary to consider combinations of various items and there is a need to enumerate all possible sequences, to calculate the total number of different sequences of items that could occur or other related questions. The mathematics used to solve this kind of problem is known as *combinatorics*.

A few of the functions frequently used in combinatorial problem solving are included in R's base system, including:

- `choose` takes two arguments, n and k and returns the value $\frac{n!}{k!(n-k)!}$, often written as $\binom{n}{k}$; the number of ways of selecting k items from n items,
- `combn` takes two arguments, x and k and returns an array containing all combinations of the elements of x taken k at a time.

When an item is drawn, with replacement, from a pool of items the probability of drawing the same item again is unchanged, when drawn without replacement the probability will decrease by the appropriate amount. An item is distinct if it is treated as being different from all other items in the pool (even when drawing with replacement), e.g., there are four items in the pool `x=c("a", "a", "b", "c")`, but only three of them are distinct.

Table 4.1 show how the `iterpc` function in the `iterpc` package can be used to generate sequences based on the distinctness of the items and whether they are drawn with replacement or not.

		Distinct	
		True	**False**
Replacement	**True**	`I=iterpc(5, 2, replace=TRUE)`	`x=c("a", "a", "b", "c")` `I=iterpc(table(x), 2, replace=TRUE)`
	False	`I=iterpc(5, 2)`	`x=c("a", "a", "b", "c")` `I=iterpc(table(x), 2)`

Table 4.1: Example `iterpc` calls generating particular kinds of sequences of length two (by passing the value returned to `getall`, e.g., `getall(I)`).

The treatment of item ordering is another factor when considering all possible permutations; is the ordering of items significant or not, e.g., are the sequences `a,b` and `b,a` treated as different or equivalent? When the ordering of items is significant calls to `iterpc` need to set the optional argument `ordered` to `TRUE`, e.g., `I=iterpc(5, 2, ordered=TRUE)`.

4.6.1 A combinatorial example

A study by Jones[302] investigated developer preferences for ordering the members of C `struct` types. The hypothesis was that members having the same type are grouped together within the same `struct` type.

There are enough instances of `struct` types containing between three and eight members in the sample to be analyse with a reasonable level of confidence.[ix]

If a `struct` contains n members, then the number of possible member sequences is $n!$. However, we are only interested in member types and don't care about permutations of members having the same type. The number of different member type sequences is $\frac{n!}{n_1!n_2!\cdots}$ where $n = n_1 + n_2 + \cdots$ and n_1, n_2, etc are the number of members having a given unique type.

Taking the example of a `struct` containing four members, two of type `x` and two of type `y` the possible sequences of member types within a `struct` type are:

```
xxyy xyxy yxxy xyyx yxyx yyxx
```

and if two members are of type `x`, one of type `y` and one of type `z` the possible member type sequences are:

```
xxyz xxzy xyxz xyzx xzxy xzyx yxxz yxzx yzxx zxyx zxxy zyxx
```

In the first case members are grouped together in $\frac{1}{3}$ of cases and in the second in $\frac{1}{2}$ of cases.

If there are t different types, there are $t!$ possible unique sequences of types. If the ordering of `struct` members is random, the probability of encountering a definition in which all members having the same type are grouped together is:

$$\frac{t!}{\frac{n!}{n_1!n_2!\cdots n_t!}}$$

For the two examples above the probabilities of having member ordering where identical types are grouped together are: $\frac{2!}{\frac{4!}{2!2!}}$ and $\frac{3!}{\frac{4!}{2!1!1!}}$ (which we already knew from writing out all possible sequences).

When a `struct` contains four members, as in the above examples, it is not possible to distinguish between a developer intentionally choosing an order and random selection. Repeating the calculation for `structs` types containing five members shows that the probability of random selection of member order producing the same member types being grouped together is surprisingly high (see the fifth column in Table 4.2).

Total members	Type sequence	structs seen	Grouped occurrences	Random probability	Occurrence probability
4	1 1 2	239	185	0.50	2.83e-18
4	1 3	185	146	0.50	4.75e-16
4	2 2	98	61	0.33	4.58e-09
5	1 1 1 2	57	50	0.40	1.03e-13
5	1 1 3	94	61	0.30	3.13e-12
5	1 2 2	86	49	0.20	5.18e-14

Table 4.2: Various forms of `struct` types containing a given number of members, one possible type grouping, number of actual `struct` types measured, number having grouping, probability that one type will contain this grouping and probability that number grouped out of total seen will be so grouped. Data from Jones.[302] code

Table 4.2 shows source code measurement counts and calculated probabilities for `struct` types containing four and five members: the column *Total members* lists the number of members in the type, *Type sequence* is a possible grouping of member types for the given number

[ix] The number of `struct` types containing a given number of members decreases approximately logarithmicly with increasing number of members,[299] i.e., most member sequences are relatively short.

of members, *structs seen* is the number of measured `structs` containing the given number of members/types, *Grouped occurrences* is the number of measured `structs` having the grouping listed in the first column, *Random probability* is the probability that this grouping will occur randomly in one `struct` declaration containing that number of members and types, *Occurrence probability* is the probability that *Grouped occurrences* out of *structs seen* will occur when the probability of a single instance occurring is *Random probability*.

These calculations show that it is not possible to confidently distinguish between random and intentional ordering for individual `struct` types. However, programs contain many such types and if we label each one "Yes" or "No", depending on whether their member types are grouped or not, this list of Yes/No labels have a binomial distribution and the probability of the given number of Yes/No labels occurring through chance can be calculated.

Taking the example of a `struct` containing four members, two of type `x` and two of type `y`, the sample contains 98 such types with 61 of them having grouped member types (see columns 3 and 4 of Table 4.2). The probability of this occurring, when the random occurrence probability is $\frac{1}{3}$, is calculated using `pbinom(61-1, 98, 1/3, lower.tail=FALSE)`, whose value is `4.58272e-09` (the `lower.tail=FALSE` option is used because we are interested in the probability of seeing 60 or more occurrences).

Figure 4.23 shows the measured percentage of `struct` types whose members are grouped by type (red pluses) and the percentage that would occur with random ordering (blue line). The green line is the 99.9% probability bound for the likelihood that 100 `structs`, all sharing the same member types, will all have their members grouped by type when member ordering is chosen at random. The distance of the red crosses from the 99.9% bound strongly shows that grouping of members by type has not been driven by random selection.

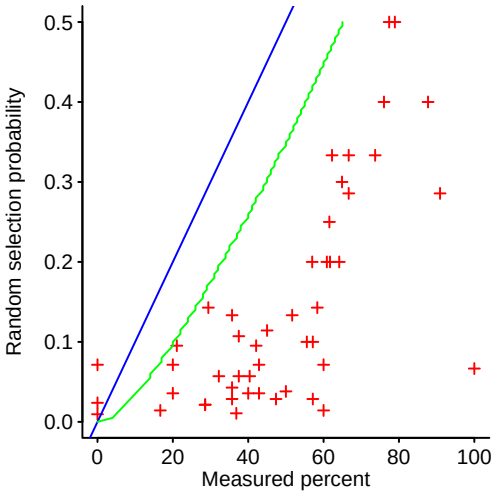


Figure 4.23: Expected probability of a single instance (y-axis) against the probability of a measured **struct** type having grouped member types (x-axis); when both probabilities are the same points will be along the blue line. Data from Jones.[302] code

4.6.2 Generating functions

Generating functions are discussed here purely to inform readers about a very powerful technique that is significantly different from the traditional approach to solving probability problems using factorials; this technique is capable of solving problems that appear to be otherwise intractable. If you cannot derive an expression specifying how many possibilities can occur in some situation, then a search for the appropriate generating function may provide an answer.

Generating functions are starting to be covered in texts on probability; some mathematical sophistication is required.

A generating function is a polynomial $a_0x^0 + a_1x^1 + \cdots + a_nx^n$ where the coefficients a_n encode information about the quantity of interest.

The following is a simple example that could just as easily be calculated using factorials, but illustrates the idea. How many ways can five items be selected if `A` can be selected 0 or 1 times, `B` can be selected 0, 1 or 2 times and `C` can be selected 0, 1, 2, 3 or 4 times? The generating function is (see the suggested reading for why this works):

$$(1+x)(1+x+x^2)(1+x+x^2+x^3+x^4) = x^7+3x^6+5x^5+6x^4+6x^3+5x^2+3x+1$$

the coefficient of x^5 is 5, so five different items orderings are possible.

A more complicated example is when items have a particular value and sequences that sum to a specific total are required. If `A` is worth 1, `B` is worth 3 and `C` is worth 5, the generating function is:

$$(1+x+x^2+\cdots)(1+x^3+x^6+\cdots)(1+x^5+x^{10}+\cdots) = 7x^{11}+7x^{10}+6x^9+5x^8+4x^7+4x^6+ 3x^5+2x^4+2x^3+x^2+x+1$$

the coefficient of x^{10} is 7, so there are seven different ways of selecting items that sum to ten.

The `polynem` package...

Chapter 5

Statistics for software engineering

5.1 Introduction

The output of statistical analysis should be treated as a guide and not a mandate.

Correlation does not imply causation, a common mantra that is worth repeating.

Traditionally statistical techniques have had to be practical to perform manually. This has resulted in general problems being split into a profusion of specific subproblems and the creations of techniques tailored to do a good job of handling each case. Doing statistics involved mapping the sample characteristics to a particular subproblem and then applying the corresponding technique. Using a computer makes it practical to apply general solution techniques and there are a few more powerful and robust statistical techniques available;[162] many users of statistical techniques have simply switched from manual to computer based calculation of the familiar historical techniques, without appreciating the original design rational for these techniques. Many of the statistical techniques appearing in this book are impractical to apply manually, e.g., the bootstrap, a computer is required.

The developer input to the data analysis process is to apply domain knowledge to suggest possible patterns of behavior to search for and to provide one or more interpretations of any other patterns that might be uncovered.

This chapter attempts to illustrate the use of techniques likely to be of use in analysing data that is commonly encountered in a software engineering context. It starts by discussing some general ideas that are used in statistical analysis.

Many books using statistical techniques invest a lot of effort massaging data into a form that permits the use of techniques that assume the data has a Normal distribution, a.k.a. Gaussian distribution. The reasons for this are historical (assuming Normality made the analysis tractable in the days before computers) and data in the Social sciences (early adopters of statistical techniques and a huge market for statistical books) appearing to be drawn from a Normal distribution (despite the claims made, data in this field often does not have a Normal distribution[387]). It could be said that nobody ever got fired for assuming a Normal distribution.

Software engineering measurements often involve values that do not have a Normal distribution; the Exponential and Poisson distributions are relatively common; measurements best described using a Normal distribution do occur, but they do not have the dominant market share encountered in other, non-software related, domains (e.g., the social sciences).

The input to any statistical analysis is a sample and some expectations of behavior; the expectations may be explicit (e.g., measurements are independent of each other) or implicit assumptions (e.g., the choice of a statistical technique that only produces reliable results for samples drawn from a population having certain characteristics).

The terms *parametric test* and *nonparametric test* are sometimes used to describe statistical tests; a parametric test assumes that the sample has some known distribution (i.e., an expression containing configuration parameters such as mean and standard deviation), while a nonparametric test makes no assumptions about the distribution of the sample (it is sometimes said to be *distribution free test*).

When a sample has the required distribution, parametric tests have greater power for the same number of measurements relative to a corresponding nonparametric test.

Historically, the reliability of the results produced by statistical techniques has depended on mathematical analysis; analytic solutions are only known for a relatively small number of cases. Once computers became available they provided another tool that could be used to evaluate the performance of techniques on data having a wide variety of characteristics, e.g., how poorly some parametric techniques perform when a sample does not have the required probability distribution.

Ideally the person doing the analysis has a good understanding of the processes that generated the sample being analysed, this understanding can be used to create a model against which the measured values can be compared. In practice the understanding of the processes involved is often poor or non-existent and in this case the available sample is used to guide the analysis, which is almost as dangerous as not using it.

Having found a pattern that matches to some desired level of certainty, the next question is the size of the effect (e.g., mountain or molehill; the statistical term is *effect size*). If the effect size is large enough to be of interest, the next step is to obtain some assurance that it really exists and is not the result of some random combination of events. Confidence intervals and *p-value* are two of the quantities used to compare the results obtained against random behavior.

The ability to detect patterns that might be present in a sample depends on the quality and quantity of the measurement data:

- quality: noise in the measurement process and errors in post measurement processing (e.g., incorrect conversion of file formats or inaccurate calculations of values derives from the raw data) are some of the problems that affect the quality of the sample data,
- quantity: the number of measurements impacts the power and significance of statistical tests and the error bounds of statistical algorithms.

Some statistical techniques divide variables into two classes, those that explain and those that respond. As their name suggests *explanatory variables* are used to explain something (the term *predictor variables* is used when the variables are used to make predictions, *control variables* is sometimes used in an experimental setting and *independent*, *stimulus*, *factor* and are also used). The *response variable* (also known as a *dependent variable*) is the variable (usually there is only one) whose behavior is explained or predicted using explanatory variable(s).

5.1.1 Statistical inference

The most commonly used statistical inference technique is based on *frequentist* methods, i.e., how often events occur and long-run averages. All techniques have problems associates with their use and frequentist being the most widely used has the greatest number of detractors; a common problem is misuse of the concept of p-value, a problem likely to befall any widely used technique simply because of the varying skills of the people using it. The p-value is the fall-guy of the frequentist approach and the issues surrounding this quantity are discussed below.

The frequentist approach is the technique predominantly used in this book because it is commonly used in statistical books and articles, it is used by most of the R packages and readers are likely to encounter it when interacting with other people involved in analysing and using data.

Another inference technique is *Bayesian statistics*. A major issue with the Bayesian approach is that it requires an estimate for the probability of an event occurring, i.e., a reasonable value for the probability of the event occurring estimated prior to any measurements being made (the measurements get factored in later); selecting a suitable prior opens the door to the bias of opinion and policy guidelines e.g., a Bayesian approach to deciding whether the accused is guilty runs into the problem that many legal systems assume that people are innocent until proven guilty (i.e., the prior is zero), which steps through the calculation to always producing a non-guilty answer.

Maximum likelihood estimation, MLE, is a technique that finds the set of parameters for a model that makes the observed data most likely to have occurred. It has been said that academics like maximum likelihood estimation because it is a way of showing off their calculus within the statistics syllabus.

5.2 Samples and populations

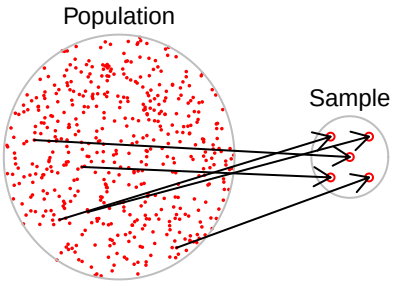


Figure 5.1: Example of a sample drawn from a population. code

It is not always possible to measure every member of a population and the subset of a population measured is known as a *sample* (see Figure 5.1).[i] Depending on the question being asked, a set of measurements may be a population or a sample. For instance, measurements of one particular program yields the parameters of a population when the questions being asked concern just that one program, but they become the statistics of a sample when generalizing the findings to questions about other programs (including future versions of the one measured).

A sample is selected as a proxy of the entire population. There are a variety of sampling techniques, including:

- a *prospective* study collects data as events unfold. Figure 5.2 shows the date of introduction of a cpu against its commercial lifetime, in years.[295] Processors that ceased production in 2000 or 2010 would appear along one of the two colored lines,[ii]
- a *retrospective* study collects data after events have taken place,
- a *convenience sample*, as its name implies, makes do with what is available,
- *snowball sampling*, or *chain sampling* starts with an initial list of subjects who are asked to propose other subjects whom to them, with the process iterating until the number of new subjects falls below some threshold,
- stratified sampling divides the population into what are known as *strata*, with the strata chosen so that similar cases cluster tend to within each one; each of these strata are then sampled (using say random sampling) to produce the final sample (which is a set of distinct stratum, see Figure 5.3),
- sequential sampling is covered in the chapter on Experiments,
- interval sampling divides the measurement interval into a series of fixed points and samples at just these points. The width of the sampling intervals puts a lower bound on the behavior that can be resolved. An experimental study[iii] by Kistowski et al[581] measured power consumption, using programs from SPEC's Server Efficiency Rating Tool, at load level increments of 2% (crosses) and 10% (lines); see Figure 5.4. A cost/benefit analysis would compare the greater accuracy obtained using finer measurement intervals against the likelihood of sudden jumps in the value of the response that could have a noticeable impact on the results.

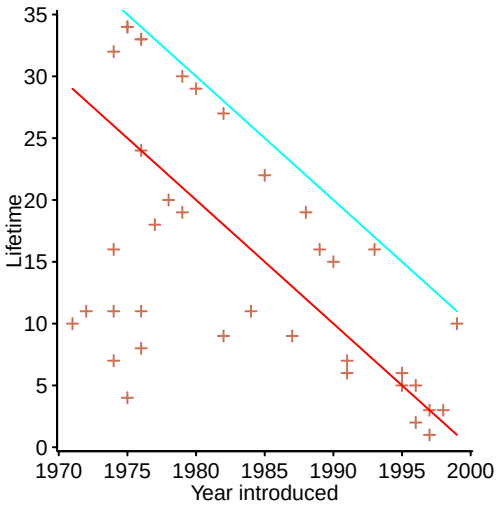


Figure 5.2: Date of introduction of a cpu against its commercial lifetime. Data from Culver.[295] code

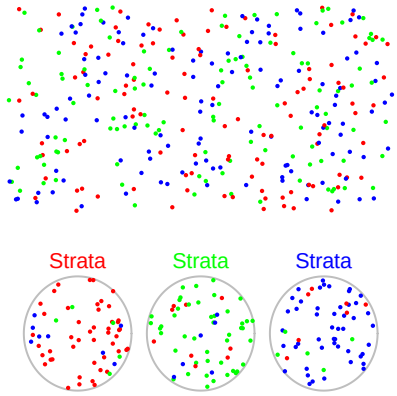
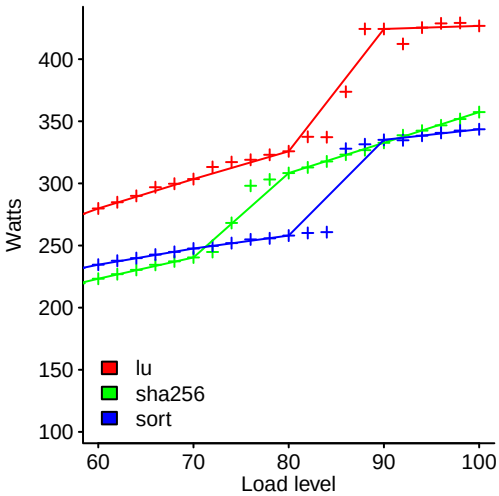


Figure 5.3: A population of items having one of three colors and three strata sampled from it. code

Occasionally the subjects of interest are not present in the sample. For instance, the damage experienced by aircraft returning from combat, during the second world war, was analysed with a view to improving aircraft survival rate. One of the statisticians involved pointed out that important subjects were missing from the sample,[371] aircraft that had not returned. The return of a damaged aircraft provided evidence that the damaged areas were not critical to survival. It was those areas not damaged in returning aircraft that were likely to be critical to survival.

While the aim is often to gather a representative sample of the population as a whole, sometimes samples having other characteristics are of interest, e.g., being representative of diversity.[405]



January 30, 2017

Figure 5.4: Power consumed by three SERT benchmark programs at various levels of system load; crosses at 2% load intervals, lines based on 10% load intervals. Data

[i] The term *statistic* applies to values calculated from a sample, while the term *parameter* applies to values calculated from a population.

[ii] Email discussion with the author confirmed that the data had not been updated since 2010.

[iii] The study was experimental because it did not meet all the requirements for an official SERT run.

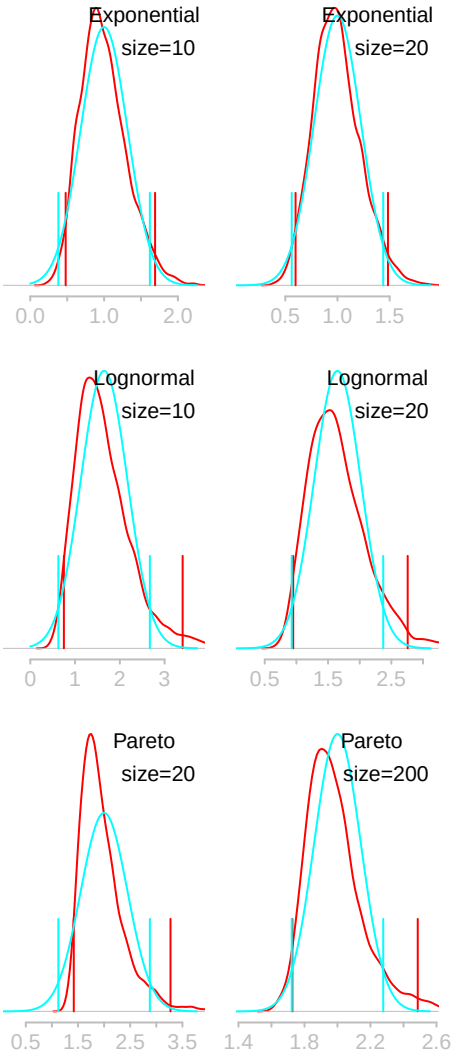


Figure 5.5: Distribution of 4,000 sample means for two sample sizes drawn from exponential (left), lognormal (center) and Pareto (right) distributions, vertical lines are 95% confidence bounds. The blue curve is the Normal distribution predicted by theory. code

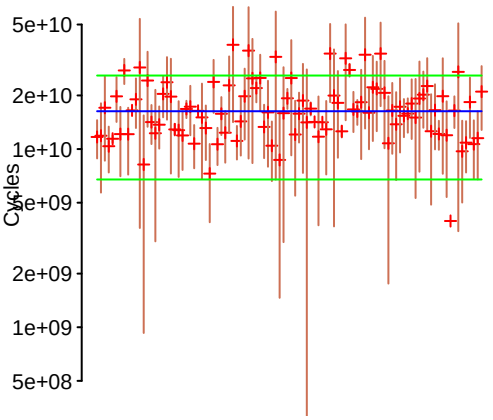
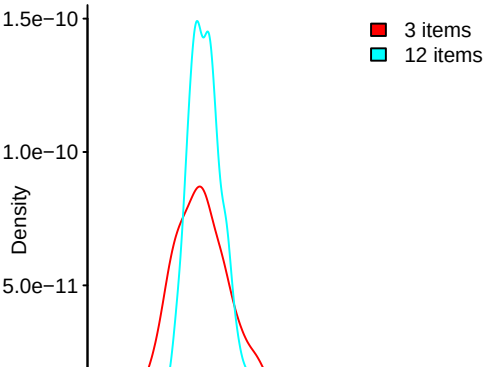


Figure 5.6: Mean (red) and standard deviation (grey lines; they are not symmetrical because of the log scaling) of samples of 3 items drawn from a population of 1,000 items (blue line mean, green line standard deviation). Data kindly provided by Chen.[104] code



5.2.1 Sampling error

If the reader is happy to agree that sampling error is an important issue, this section can be skipped. Otherwise, read on and be frightened into agreeing.

The Central Limit Theorem is a statement about the mean value of samples drawn from a population. If the population has a finite variance (power laws with an exponent between zero and two have an infinite variance), then the distribution of sample means converges to a Normal distribution as the sample size, N, increases (it does not matter which distribution the population has, the distribution of the sample means converges to the Normal).

How quickly does the distribution of sample means converge? The Berry-Esseen theorem provides the best known estimate of the convergence of the distribution of the mean of independent, identically distributed variables to a Normal distribution:

$$|F_n(x) - \Phi(x)| \leq \frac{0.34(\rho + 0.43\sigma^3)}{\sigma^3\sqrt{N}}$$

where: F_n is the cumulative distribution function of the means, Φ the cumulative distribution function of a Normal distribution, ρ the third moment of x (and less than infinity), N the sample size and σ the standard deviation.

The only control parameter available for influencing the error is the number of measurements; the error is proportional to: $\frac{1}{\sqrt{N}}$, e.g., to halve the error in the sample mean, the sample size needs to increase by a factor of four.

Experiments have found that for some distributions a sample size of 20 is sufficient to provide a reasonable approximation to the population mean. However, for other distributions (e.g., those whose tails decay more slowly than exponential) a sample size of over 200 may not be good enough.

Figure 5.5 shows the distribution of mean values for samples drawn from three different distributions (using two sample sizes). The vertical lines are 95% confidence bounds and show that a small increase in sample size is enough for the exponential distribution the bounds to noticeably converge improve (i.e., the confidence bounds of a normal distribution), while a much larger increase in sample size has had no overall impact on the confidence bounds for the Pareto distribution.[iv]

A study by Chen, Chen, Guo, Temam, Wu and Hu[104] measured the performance of programs in the SPEC CPU2006 benchmark using 1,000 sets of input data for each program. As an exercise in sampling let's assume we only have access to three of the possible 1,000 input datasets, what range of execution times might we expect to see from processing just three datasets?

Figure 5.6 was obtained by randomly sampling three items from the population of 1,000 and repeating the process 100 times. The red cross is the sample mean and the vertical grey lines each sample's standard deviation; the blue line is the mean for the population of 1,000 input sets and the green lines the bounds of its standard deviation.

Figure 5.7 shows the distribution of sample means for sample sizes of 3 and 12 items. As expected, the larger samples show less variation in mean value.

The sources of random variability in a sample include the following:

- measurement error caused by the imperfect tools used to make measurements, which can include faults in the programs used to count constructs,
- demographic variability, e.g., particular kinds of programs, or developers working in one location or for one company are measured,
- environmental variability is the sea in which developers swim now or have swum in the past, e.g., company culture or habits acquired from early teachers.

Figure 5.8 shows the number of commits to glibc[212] for each day of the week, separated out by year. The plot with the large values on the right shows combined counts over all years. The combined plot suggests that most commits occur near the middle of the week, with the number falling off towards the beginning and end of the week. However, the yearly plots

[iv] There will be random fluctuations from the random drawn of the sample.

rarely show anything like this pattern; is any interpretation of the pattern of commits in the combined plot a just-so story?

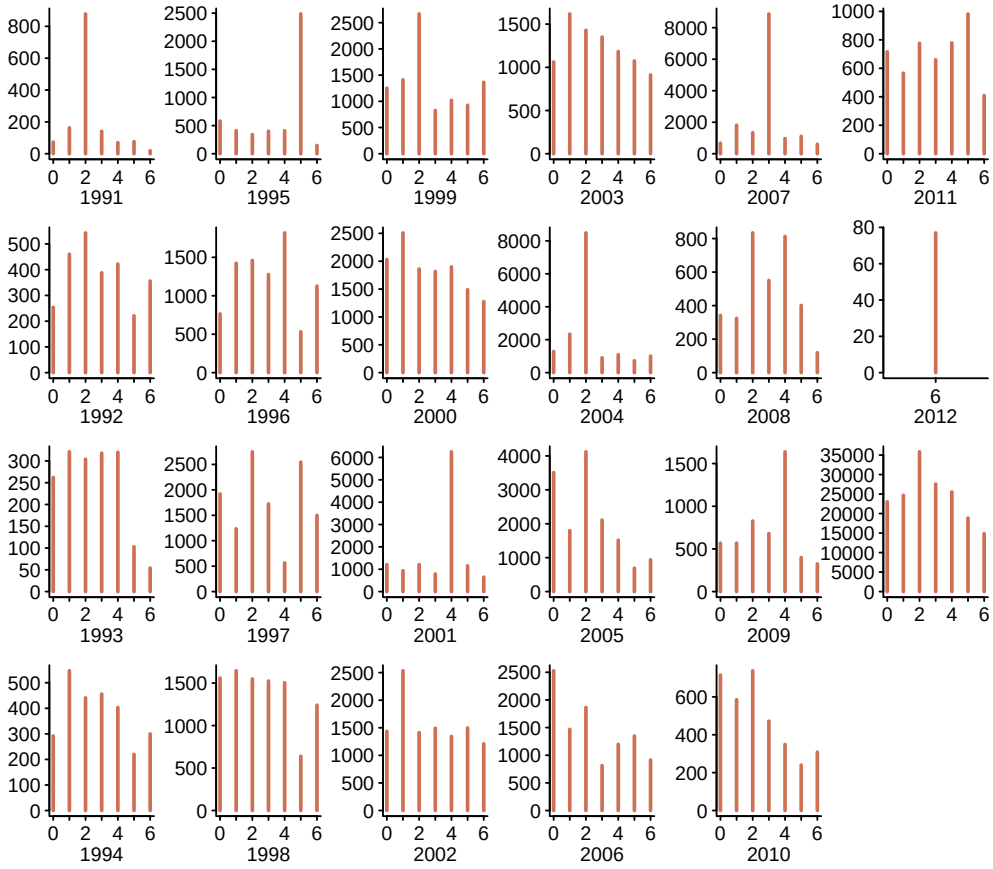


Figure 5.8: Number of commits to glibc for each day of the week, for the years from 1991 to 2012. Data from González-Barahona et al.[212] code

5.3 Describing a sample

A list of values can be overwhelming and compressing many values into a few values, often just one value, is a commonly used approach.[v] The few compressed values are known as *descriptive statistics*. The following are some commonly used algorithms for producing the values that are said to describe a sample:

- a point estimate of a central value and its variability, e.g., mean and standard deviation,
- an equation that closely fits the sample data, such that some condition is met (e.g., minimising mean squared error),
- quartiles cluster measurements based on where values are relative to other values in the sample, e.g., a box-and-whiskers plot such as Figure 3.19.

The mean and standard deviation are the two most commonly used descriptive statistics about a sample. It is incorrect to think that two distributions having the same mean and standard deviation will be very similar; see Figure 5.9.

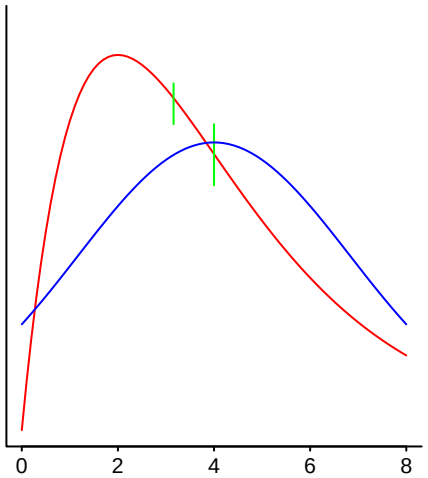


Figure 5.9: A Normal distribution with mean=4 and variance=8 and a Chi-squared distribution with four degrees of freedom having the same mean and variance (the vertical lines are at the distributions' median value). code

5.3.1 A central location

Perhaps the most widely used single value summary of the values in a sample derives from the idea of a *middle* or *central* location.

- the mean, obtained by adding together the values in a sample and dividing by the number of values, is perhaps the most commonly used central location,

[v] Plotting is a technique that often uses all the values and is the primary focus of the communicating chapter.

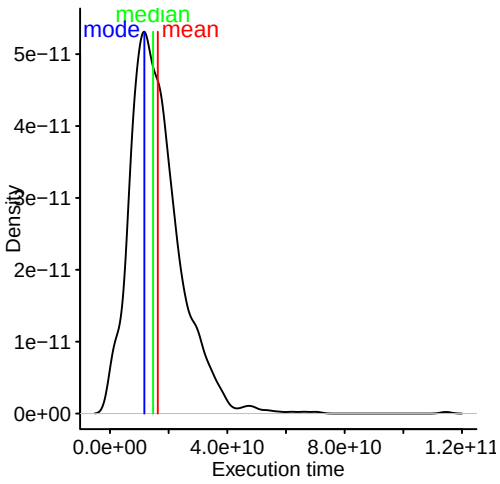


Figure 5.10: Density plot of execution time of 1,000 input data sets, with lines marking the mean, median and mode. Data kindly supplied by Chen.[104] code

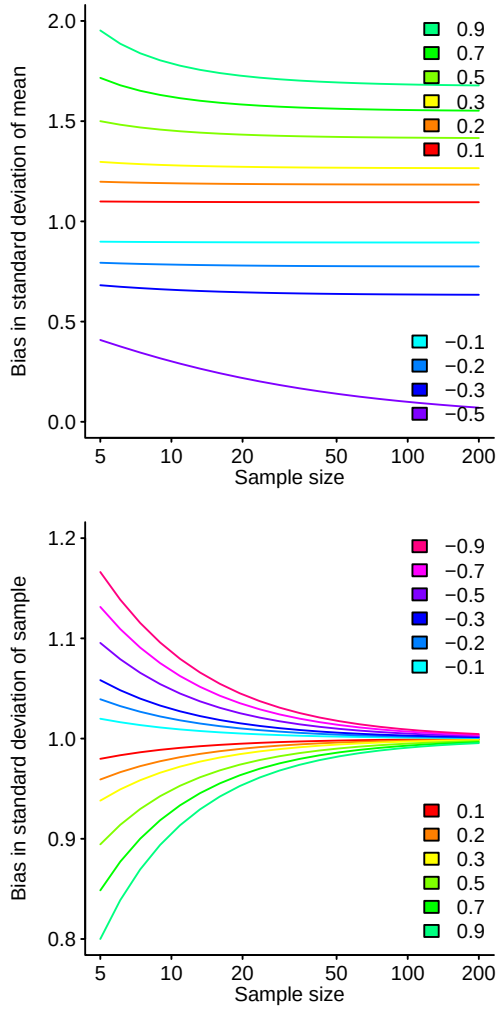


Figure 5.11: Impact of serial correlation, AR(1) in this example, on the calculated mean (upper) and standard deviation (lower) of a sample (the legends specify the amount of serial correlation). code

- the *median* is obtained by sorting the N values into numerical order and selecting the value of the $\frac{N+1}{2}$th element (if N is even the average of the middle two values is used),
- the *mode* is the value most likely to be sampled (the `mode` function is unrelated to the statistical algorithm of that name, it returns the type or storage mode of an object). The `modeest` package contains functions for estimating various kinds of mode.

For symmetric distributions the values of the mean, median and mode are equal, and for asymmetric distributions the three values can be very different.

For any sample values drawn from a unimodal distribution, the difference between the median and mean is less than or equal to $\sqrt{0.6}\sigma$, and for other distributions less than σ.

The difference between the median and mode is less than or equal to $\sqrt{3}\sigma$.

Unless the sample distribution is symmetric, it is not possible to sum multiple modes, e.g., cost estimates. For nonsymmetric distributions, adding underestimates the true value, e.g., for a gamma distribution the mean is $k\theta$ and the mode is $(k-1)\theta$, where k and θ describe the distribution.

Figure 5.10 shows the distribution of execution times of the 1,000 input data sets from Chen et al.[104] If we are interested in an estimate of the execution time of a randomly chosen input data set, it makes sense to use the median value, the point that equally divides the number of input data sets. If we are interested in an estimate of the execution time most likely to be encountered, the value to use is the mode.

Some distributions have such fat tails that the mean is infinite, e.g., the Cauchy distribution. In practice the regularity with which very large values occur causes the mean value of a sample to jump around erratically, as new measurements are added to it. A distribution that does not have a finite mean may still have a median. This is because the median is not affected by extreme values in the way the mean is, so any extreme values that do appear in a sample do not prevent the median converging to a fixed value.

The well-known algorithms for calculating the mean and standard deviation of a sample require that each value be independent of the others. When a sequence of values is serially correlated, i.e., the value of a measurement is related to the value of the one or more immediately previous measurements, the calculated mean and standard deviation is biased. In the case of the mean, the uncertainty in its value grows for positive correlation and decreases for negative correlation. The upper plot in Figure 5.11 shows the fraction of this change for various sample sizes; it is based on an AR(1) model, where each value correlates with the immediately preceding value by an amount given in the legends on the right of the plot (see the time-series subsection for details of AR models). A positive correlation causes the ratio of the sample standard deviation, relative to the population standard deviation, to be underestimated, while a negative correlation causes it to be overestimated (the lower plot in Figure 5.11 shows the fraction of this change).

The `sandwich` package supports the calculation of various error measures caused by serial correlation (e.g., the `lrvar` function calculates the error in the long term mean of a series).

5.3.2 Sensitivity of central location algorithms

Samples sometimes include extreme values (which may or may not be the result of noise). The percentage of observations that can cause a statistical estimator to produce an arbitrarily large (positive or negative) value is known as the *breakdown point*.

The breakdown point for the mean is proportional to $\frac{1}{N}$, i.e., no matter how many observations are made it only takes one extreme value to produce a completely spurious result for the mean; the mean has the smallest breakdown point it is possible to have.

At the other end of the scale the median has a breakdown point of 0.5 (i.e., half of the measurements can have extreme value without affecting the value of the result) and for this reason the median is often recommended, over the mean, when measurements values are known to be very noisy. However, the median cannot be recommended for universal use because there are situations where it does not perform as well as the mean. For instance, when values are drawn from a discrete distribution whose mean is roughly half-way between measurable points and the sample includes duplicate values, then most samples will have a median value slightly larger/smaller than the actual mean, i.e., the median is not evenly distributed across

possible values in the way the Central limit theorem says that the mean is distributed; see Figure 5.12.

The probability of an outlier occurring depends on the reliability of the measurement process and the characteristics of the population being sampled from. The following two techniques are robust in the presence of extreme values and often used when a single mean value is involved:

- *trimmed mean* removes a percentage of the largest and smallest values before calculating the mean of the remaining values (it has been found that 20% is a good value for general use). The `mean` function includes a `trim` argument for specifying the percentage to be trimmed.
- *winsorized mean* replaces rather than remove values. The values of the lowest X% are replaced with the lowest value that is just not within percentage, and the values of the highest X% are replaced with the highest value just not within this percentage; the Winsorized mean is calculated using the updated list of values. The `psych` package contains functions that calculate various quantities using the Winsorizing algorithm.

The Trimmed and Winsorized means do not produced biased results when applied to samples drawn from a population having a symmetric distribution.

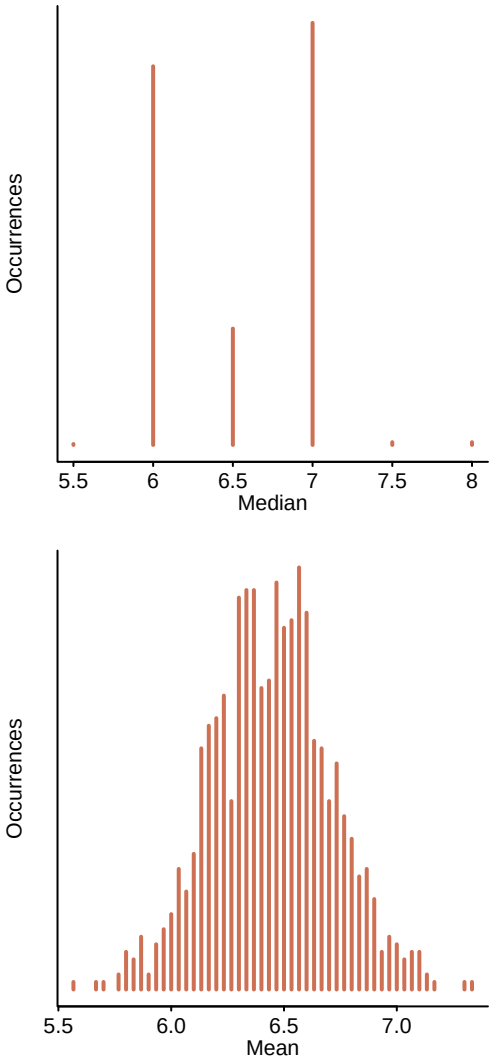


Figure 5.12: Occurrence of sample median and mean values for 1,000 samples drawn from a binomial distribution. code

5.3.3 Geometric mean

The *geometric mean* is defined as:

$$Mean_g = \left(\prod X\right)^{\frac{1}{N}}$$

e.g., the geometric mean of 10, 100, 1000 is $(10 \times 100 \times 1000)^{\frac{1}{3}} \rightarrow 100$.

The geometric mean is preferred to the arithmetic mean when ranking ratios or normalised data (which is a kind of ratio) because it gives consistent results.

Consider the (invented) benchmark performance of the three systems in Table 5.1. Treating a as the base performance, what is the relative performance improvement of b and c?

If the arithmetic mean is used, the performance ranking of b and c, relative to a, depends on whether the calculation used is a ratio of the means or the mean of the ratios. The arithmetic means of the benchmark performance for each system is 50.5, 53.5 and 53: the ratio of these mean values is listed in the fourth column; the individual benchmark ratios for a and b are: $\frac{2}{1}$ and $\frac{105}{100}$, and for a and c: $\frac{3}{1}$ and $\frac{103}{100}$. The mean of these ratios is listed in the fifth column. Comparing columns four and five shows that the ranking of b and c depends on the method of calculation.

system	integer	float	ratio of means	mean of ratios	geometric mean
a	1	100			10
b	2	105	$\frac{53.5}{50.5} \rightarrow 1.0594$	`mean(2/1+105/100) -> 3.05`	14.49
c	3	103	$\frac{53}{50.5} \rightarrow 1.0495$	`mean(3/1+103/100) -> 4.03`	17.58

Table 5.1: Invented benchmark performance measurements of three systems and various methods of calculating relative performance. The relative performance of b and c depends on which mean is used.

If the geometric mean is used, the relative order of the final ratio is not order dependent.

Sometimes the arithmetic and geometric means produce the same benchmark rankings, e.g., a benchmark[166] of eight Intel IA32 processors used the arithmetic mean of ratios to compare results, the results from using the geometric means was not large enough to affect the relative ranking of processors for a given performance characteristic (see rexample[powervperfasplos2011.R]).

The Geometric mean might be used when values cover several orders of magnitude, e.g., a geometric or logarithmic series (e.g.,2, 4, 8, 16, 32, 64, ...)

Two ways of calculating the geometric mean include the code `exp(mean(log(x)))` and the `geometric.mean` function in the `psych` package.

5.3.4 Harmonic mean

The *harmonic mean* is defined as:

$$Mean_h = \frac{N}{\sum X^{-1}}$$

e.g., the harmonic mean of 1, 2, 3, 4, 5 is $\frac{5}{\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}} \rightarrow 2.189781$

The harmonic mean is used to find the "average" of a set of ratios or proportions.

When there are two values the formula becomes:

$$Mean_h = 2\frac{x \times y}{x+y}$$

which has the same form as the F_1 score or F-measure used in information retrieval to combine the precision and recall:

$$F_1 = (1+\beta^2)\frac{precision \times recall}{\beta^2 precision + recall}$$

Two ways of calculating the harmonic mean include the code `1/mean(1/x)` and the `harmonic.mean` function in the `psych` package.

5.3.5 Contaminated distributions

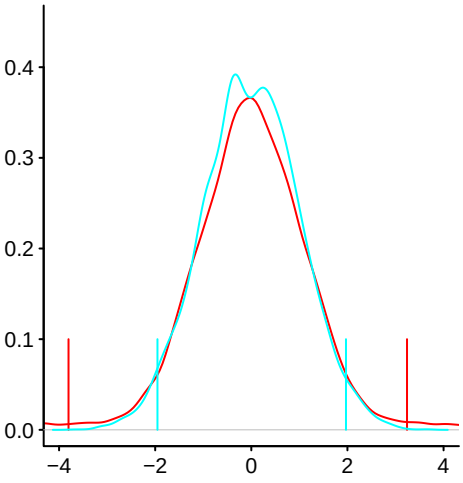


Figure 5.13: A contaminated normal, values drawn from two normal distributions with 10% of values drawn from a distribution having a standard deviation five times greater than the other. code

That a sample, or the error present in the measurements it contains, can be fitted to a Normal distribution is a common assumption that often goes unquestioned. Textbooks are filled with techniques that only exhibit the cited desirable attributes when the Normality assumption holds. There is also the lure of analytic solutions to problems, which again are often only available when the Normality assumption holds.

Even when sample values appear to be drawn from a Normal distribution, a small percentage of contaminated values can have a dramatic effect on the value returned by a statistical algorithm.

The Contaminated Normal distribution is a mixture of values drawn from two Normal distributions both having the same mean, but with 10% of the data points drawn from a distribution whose standard deviation is five times greater than the other. Figure 5.13 shows the sample density where 10,000 values are drawn from a Normal distribution and 10,000 values from a Contaminated Normal distribution; visually they seem very similar (see rexample[contamnorm.R] to learn the color used to plot each sample).

This contaminated Normal distribution has a standard deviation that is more than three times greater than the Normal distribution from which 90% of them are drawn. This illustrates that a Normal distribution contamination by just 10% of values from another distribution can appear to be Normal, but have very different descriptive statistics.

There are a number of tests for estimating whether sample values are drawn from a Normal distribution. The Shapiro-Wilk test (the `shapiro.test` function), the Kolmogorov-Smirnov Test (the `ks.test` function)[vi] and the Anderson-Darling test are common suggested techniques. A comparison of four normality tests[474] found the Shapiro-Wilk test to be the most powerful normality test; see Figure 4.9.

Recommendation: Use the Shapiro-Wilk test for testing whether sample values are drawn from a Normal distribution.

When a data set contains very few values, even the Shapiro-Wilk test may fail to detect (e.g., p-value < 0.05) that sample values are not drawn from a Normal distribution. In the case of the Contaminated Normal distribution, samples containing only 10 values are considered to have a Normal distribution in around 30% of cases (based on p-value > 0.05), with the percentage dropping to 10% for samples containing 20 values.

For an in depth analysis of the potential problems that outliers, skewed distributions and fat tails can cause see "Introduction to Robust Estimation & Hypothesis Testing" by Rand Wilcox.[616]

[vi] Both included in the R base system.

5.4 Statistical error

The outputs from a statistical technique generally includes probabilities and it is the responsibility of the user of the technique to decide the cut-off probability below/above which an event is considered to have/not occurred.

The two kinds of error that can be made are:

- treating a hypothesis as true when it is actually false (known as a *False positive*), the technical statistical term is making a *Type I* error; $P(TypeIerror = P(RejectH_0|H_0true))$,
- treating a hypothesis as false when it is actually true (known as a *False negative*), the technical statistical term is making a *Type II* error; $P(TypeIIerror = P(DonotrejectH_0|H_Atrue)$, where H_A is an alternative hypothesis).

		Decision made	
		Reject H	Fail to reject H
Actual	H true	Type I error	Correct
	H false	Correct	Type II error

Table 5.2: The four combinations of circumstances and their outcomes in hypothesis testing.

The consequences of making an error will depend on the perspective of those affected by the outcome of the decision. For instance, consider the consequences of a manager's decision on whether to invest more time and money testing the reliability of a software system; an incorrect decision can lead to more than losing the original investment (i.e., losing market share to a competitor), but the likely bearer of the loss will depend on the vendor/customer relationship world in which the decision plays out, as Table 5.3 illustrates:

		Decision made	
		Finish testing	Do more testing
Actual	More testing needed	Customer loss	Ok
	Testing is sufficient	Ok	Vendor loss

Table 5.3: Finish/do more testing decision and likely outcome based on who incurs the loss.

5.4.1 Hypothesis testing

A hypothesis is an unverified explanation of why things are the way they are. Hypothesis testing is the process of collecting and evaluating evidence that may or may not be consistent with the hypothesis, i.e., positive and negative testing.[vii] Once enough evidence consistent with the hypothesis has been collected people may feel confident enough to start referring to it as a theory or law.

The most commonly used hypothesis testing technique is based on what is known as the _null hypothesis_[viii], which works as follows:

- a hypothesis, H, having testable prediction(s) is stated,
- an experiment to test the prediction(s) is performed, producing data D,
- assuming the hypothesis is true, calculate the probability of obtaining the data produced by the experiment. The calculation made is: $P(D|H)$; that is the probability of obtaining the data D assuming that the hypothesis H is true.

  If the calculated probability is less than or equal to some prespecified value, the hypothesis is rejected, otherwise it is said that *the null hypothesis has not been rejected* (since the result of the experiment is not conclusive evidence that the null hypothesis is true).

[vii] Gigerenzer[201] is a readable discussion of how people cope with uncertainty.

[viii] As the market leader in hypothesis testing techniques, over many decades, this technique attracts regular criticism.[117] The criticism is invariably founded on widespread misuse of the null hypothesis ritual; misuse is the fate of all widely used techniques.

Expressed in code, the null hypothesis testing algorithm is as follows:

```
void null_hypothesis_test(void *result_data, float p_value)
{
// H is set by reality, only accessed by running experiments
if (probability_of_seeing_data_when_H_true(result_data) < p_value ||
    !H)
   printf("Willing to assume that H is false\n");
else
   printf("H might be true\n");
}

null_hypothesis_test(run_experiment(), 0.05);
```

A test statistic is said to be *statistically significant* when it allows the null hypothesis to be rejected. The phrase "statistically significant" is often shortened to just "significant", a word whose common usage meaning is very different from its statistical one; this shortened usage is likely to be misconstrued when the audience is not aware that the statistical definition is being used and assumes the word is being used in its everyday meaning sense.

Statistical significance can roughly be treated as meaning *likelihood of occurring by chance*, it does not mean the results highlighted by the statistical analysis have any practical significance, i.e., the magnitude of the pattern detected may still be so small as to make it useless for practical applications.

Runing one experiment that produces a surprisingly high/low p-value is a step in the process of increasing everybodies confidence that a hypothesis is true/false.

Replication of the results (i.e., repeating the experiment and getting the very similar measurements) provides evidence that the first experiment was not a chance effect; a further boost in confidence Replication by others, who independently set up and run an experiment, is the ideal replication (it reduces the possibility that unknown effects specific to a person or group influenced the outcome); an even larger boost in confidence.

There is a great deal of confusion surrounding how the results from a null hypothesis test should be interpreted. Studies have found[203] that people (incorrectly) think that one or more of the following statements apply:

- *Replication fallacy*: The level of significance measures the confidence that the results of an experiment would be repeatable under the conditions described. This is equivalent to saying: $P(D|H) == 1 - P(D)$, and would apply if the hypothesis was indeed true.
- the significance level represents the probability of the null hypothesis being true. This is equivalent to saying: $P(D|H) == P(H|D)$.

The Bayesian approach to hypothesis testing is growing in popularity and works as follows:

- two hypotheses, H_1 and H_2, having testable prediction(s) are stated (the second hypothesis may just be that H_1 is false),
- a non-zero probability is stated for the hypotheses being true, $P(H_1)$ and $P(H_2)$, known as the *prior* probabilities,
- an experiment to test the prediction(s) is performed (producing data D),
- update the previously estimated probability that H_1 and H_2 are true. The calculation uses Bayes theorem, which for H_1 is:

$$P(H_1|D) = \frac{P(H_1)P(D|H_1)}{P(H_1)P(D|H_1) + P(H_2)P(D|H_2)}$$

The updated prior probability, on the basis of the experimental data, is known as the *posterior probability* of the hypothesis being true.

5.4.2 p-value

In a randomized experiment the *p-value* is the probability that random variation alone produces a test statistic as extreme or more extreme than the one observed.

In a commercial environment the choice of p-value should be regarded as an input parameter to a risk assessment comparing the costs and benefits of all envisioned possibilities.

The p-value can be thought of as a distance, measured in standard deviations, between the data and the null hypothesis.

In many social sciences the probability of the Null hypothesis being true must be less than 0.05 (i.e., 5%, or slightly greater than 2σ),[ix] while in civil engineering a paper describing a new building technique that created structures having a 1-in-20 chance of collapsing would not be considered acceptable. High energy physics requires a p-value below $5\sigma \rightarrow 5.7 \times 10^{-7}$, before the discovery of a new particle is accepted.

As sample size increases, p-values will always become smaller. For instance, some aspect of flipping a coin may very slightly favour heads and given enough coin flips a sufficiently small p-value, for the hypothesis that the coin is not a fair one, will be obtained. There is no procedure for adjusting p-values for hypothesis analyses using very large amounts of data.

When lots of measurement data about many variables is available it is possible to go on a fishing expedition, looking for relationships between variables.[484] The probability of finding one significant result when comparing n pairs of variables, using a p-value of 0.05, is $1-(1-0.05)^n$ (which is 0.4 when $n=10$). When multiple comparisons are made the base p-value needs to be adjusted to take account of the increased probability of treating noise as signal.

Perhaps the most common technique is the *Bonferroni correction*, which divides the base p-value by the number of tests performed. In the above example, the base p-value would be adjusted from 0.05 to $\frac{0.05}{10} \rightarrow 0.005$, for each of the ten tests.

The `p.adjust` function supports p-value adjustment using a variety of different techniques.

Researchers know their work only has a chance of being accepted for publication if the reported results have p-values below the journal's cut-off value. Given the use of published paper counts as a measure of academic performance, there is an incentive for researchers to run many slightly different experiments to find a combination that produces a sufficiently low p-value that the work can be written up and submitted for publication[309] (a process known as *p-hacking*).[x] One consequence of only publishing papers containing studies achieving a minimum p-value is that many of the results are likely to be false (while a theoretical analysis suggests most are false,[287] an empirical analysis suggests around 14% of false positives for medical research[292]).

5.4.3 Confidence intervals

Many statistical techniques return a single number, a point value. What makes this number so special, would a value close to this number be almost as good an answer? If an extra measurement was made and added to the sample, is this number likely to dramatically change; what is one measurement were excluded from the sample, how much would that change the answer?

A confidence interval is an upper and lower bound on the result of a statistical technique. A common choice is the 95% confidence bound, default value used by many R library functions.

The numeric values associated with a *confidence interval* can be translated into visual form by including them in a plot. Figure 5.14 illustrates how confidence intervals provide an easier to digest insight into the uncertainty of a fitted regression model, compared to the single number that is the p-value. The red line shows a fitted regression model, whose predictor has a p-value of 0.02, with 95% confidence intervals in blue, showing how wide a range of lines could be said to fit the sample just as well.

A confidence interval is a random variable, it depends on the sample drawn. If many 95% confidence intervals are obtained (one from each of many samples), the true fitted model is

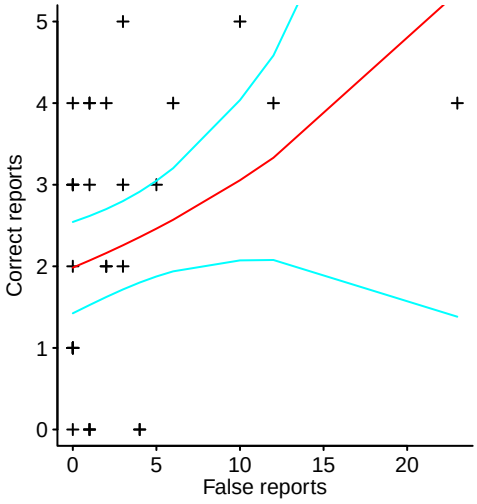


Figure 5.14: Regression model (red line; pvalue=0.02) fitted to the number of correct/false security code review reports made by 30 professionals; blue lines are 95% confidence intervals. Data from Edmundson et al.[156] code

[ix] Journals with high impact ratings can be more choosy and some specify a p-value of 0.01.

[x] Which commercial company would not willing add warts to software to keep an important customer happy?

expected to be included in this set of intervals 95% of the time (it is a common mistake to think that the confidence interval of one sample has this property). The probability that the next sample will be within the 95% confidence interval of the current sample, for a Normal distribution, is 84% or around 5 out of 6.[129]

A closed form formula for calculating confidence intervals is only known for a few cases, e.g., the mean of samples drawn from a Normal distribution, for a Binomial distribution a variety of different approximations have been proposed.[453]

Built-in support for calculating confidence intervals, in R packages, is sporadic. A Monte Carlo algorithm can be used to calculate a confidence interval from the sample, e.g., the bootstrap. This approach has the advantage that it is not necessary to assume that sample values are drawn from any particular distribution. Figure 5.15 was created by fitting many models, via bootstrapping, and using color to indicate density of fitted regression lines.

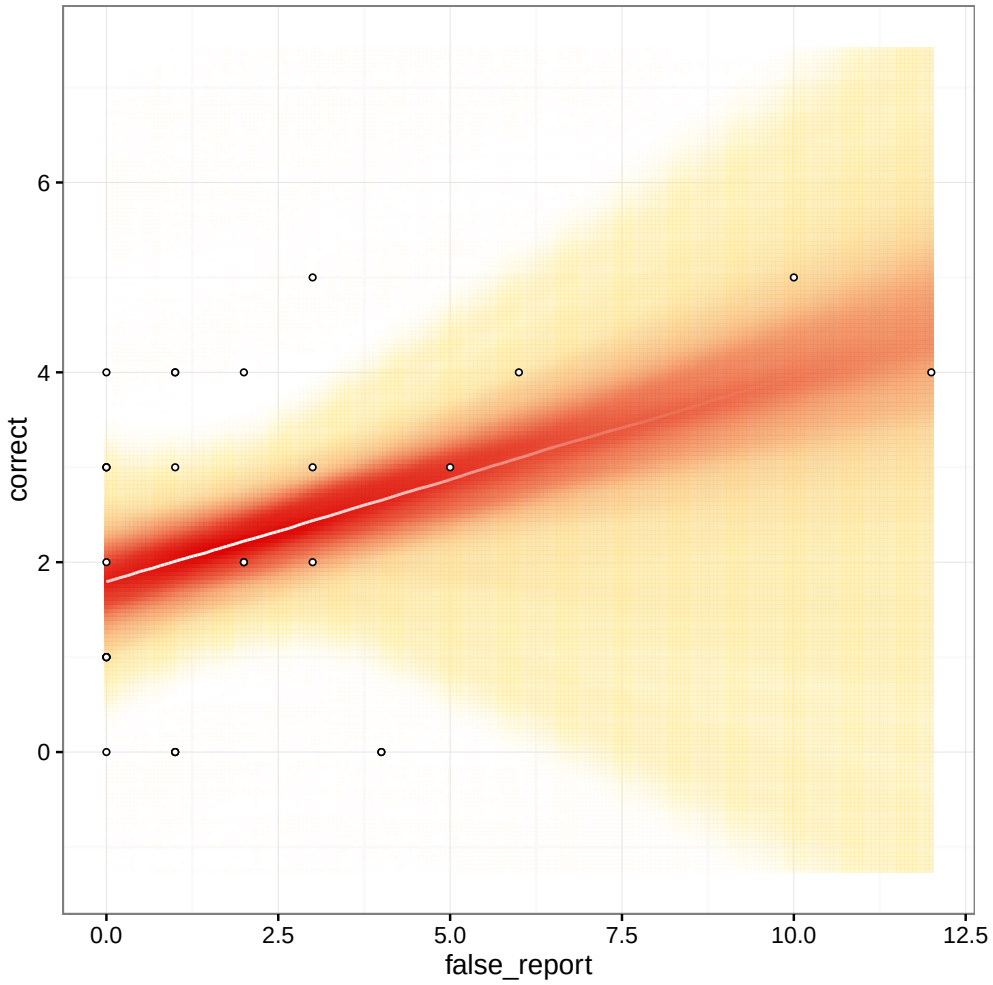


Figure 5.15: Bootstrapped regression lines fitted to random samples of the number of correct/false security code review reports made by 30 professionals. Data from Edmundson et al.[156] code

5.4.4 The bootstrap

The bootstrap is a general technique for answering questions about the uncertainty in the estimate of a statistic calculated from a sample (e.g., calculating a confidence interval or standard error).[263] Bootstrap techniques operate on a sample drawn from a population and cannot extract information about the population that is not contained in the sample, e.g., if the population contains reds and greens and a sample only contains reds, then the bootstrap will not provide any information about the greens.

Bootstrapping is a term applied in software development to systems that start themselves. In statistics, it is used to describe a process where new samples are created from an existing sample; the term *resampling* is sometimes used.

Estimating the confidence interval for the mean value of a sample is a good example of the basic bootstrap algorithm. The following are the steps involved:

- create a sample by randomly drawing items from the original sample. Usually the items are selected with replacement (i.e., an item can be selected multiple times). When items are selected without replacement (i.e., can only be selected once), the term *jacknife* is used,
- obtain the mean value of the created sample,
- iterate, say, 5,000 times.
- analyse the 5,000 mean values to obtain the lowest and highest 2.5%. The 95% confidence interval for the mean of the original sample is calculated from this lowest/highest band (several algorithms, giving slightly different answers, are available);

The `boot` package support common bootstrap operations, including `boot.ci` for obtaining a confidence interval from a bootstrap sample.

The distribution of the sample from which the bootstrap algorithm draws values is known as the *empirical distribution*.

The *bootstrap distribution* contains m^n possible samples, when sampling with replacement from m possible items to create samples containing n items; when the order of items does not matter, there are $\binom{2m-1}{n}$ possible samples (a much smaller number).

The same bootstrap procedure can be applied to obtain confidence intervals on a wide range of metrics. Figure 5.16 shows confidence intervals for kernel density in Figure 3.14 and was produced by `sm.density`, in the `sm` package, using the following code:

```
library("sm")

res_sample=sample(cint$Result, size=1000) # generate 1000 samples

sm.density(res_sample, h=4, col=point_col, display="se", rugplot=FALSE,
        ylim=c(0, 0.03),
        xlab="SPECint Result", ylab="Density\n")
```

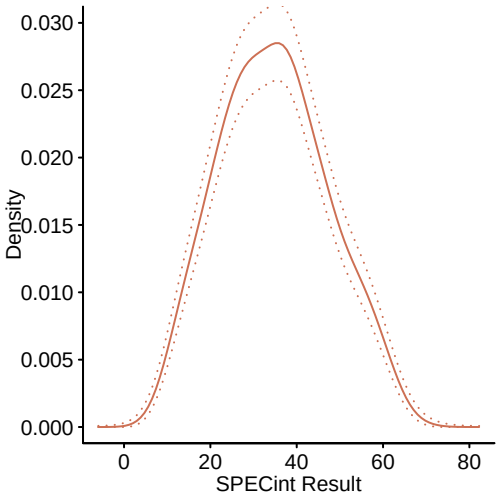


Figure 5.16: Kernel density plot, with 95% confidence interval, of the number of computers having the same SPECint result. Data from SPEC.[534] code

5.4.5 Permutation tests

For small sample sizes, computers are fast enough for it to be practical to calculate a statistic (e.g., the mean) for all possible permutations of the items in a sample. This kind of test is known as a *permutation test*.

Some techniques designed for manual implementation (e.g., Student's t-test) are approximations to the exact answer produced by a permutation test.

Permutation tests do not have any preconditions on the distribution of the sample, other than it is representative of the population, and produce an exact answer.

The `coin` package contains infrastructure for creating permutation tests and functions that perform common tasks (the names of these functions are derived from the names of the tests designed for manual implementation, e.g., `spearman_test` and `wilcox_test`).

The following is a permutation test of whether the professional experience of the two samples of subjects shown in Figure 3.1 are likely to have different mean values:

```
library("coin")

# The default is alternative="two.sided",
# an option not currently listed in the Arguments section.
oneway_test(experience ~ as.factor(language), data=Perl_PHP,
                                              distribution="exact")
```

5.5 Effect-size

Effect-size is the degree to which the characteristic of interest is present in the population (from which a sample is drawn), e.g., if we are interested in the difference in the mean performance of developers before and after attending a training course, how big is the difference (answering this question may be the reason for obtaining measurements)?

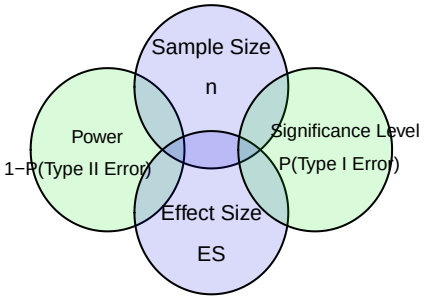


Figure 5.17: The four related quantities in the design of experiments. code

The question to ask about a given effect-size is: 'Does it matter?' The larger the effect-size the more likely it is to be of interest in practice; in some cases a small effect-size may be of interest (e.g., a small difference multiplied over a large population can have a large impact), while in other cases only a large effect-size is of interest (e.g., when the population is small a large effect-size could be needed to have a large impact).[xi]

Smaller effect-sizes are likely to be more costly to detect because more measurements are needed to isolate small effect-sizes compared to larger ones.

Figure 5.18 shows how percentage differences in the presence of a condition in a population can have a dramatic effect on the false positive rate (in red), for the same statistical power and p-value.

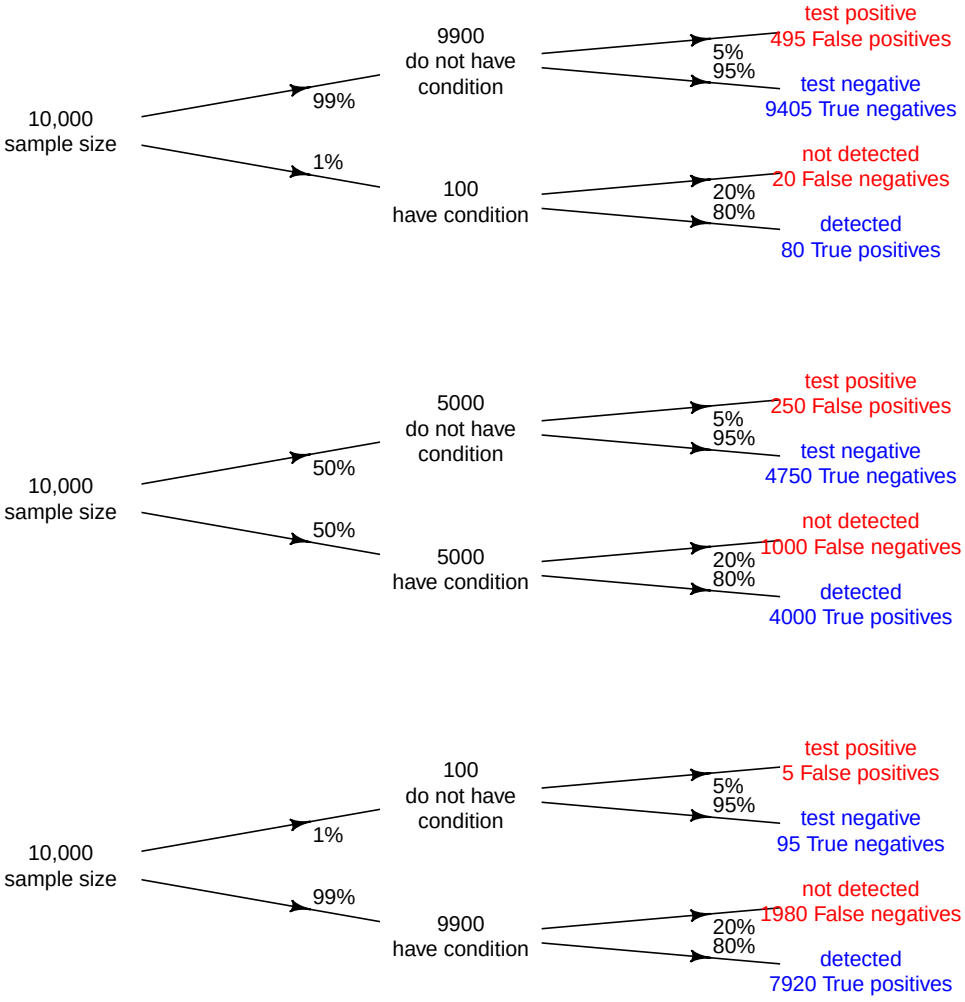


Figure 5.18: Examples of the impact of population prevalence, statistical power and p-value on number of false positives and false negatives. code

Methods for calculating effect-size depend on how data is being analysed[159] and include the following:

- correlation, e.g., the Pearson correlation coefficient, is a measure of effect-size,
- combining information on the mean and standard deviation of two samples into a single value. For instance, Cohen's d is one measure used when the samples have similar standard deviations and is given by (other approaches adjust the calculation of the standard deviation): $d = \frac{\mu_1 - \mu_2}{\sigma_{pooled}}$.

  Figure 5.19 illustrates how differences in mean and standard deviation, of two distributions, result in a given Cohen's d,

[xi] Statistical books[116] and papers sometimes concern themselves with questions of where to draw the lines that delimit large/medium/small effect-sizes; an approach that might be applicable when researchers are more interested in publishing papers than making useful discoveries.

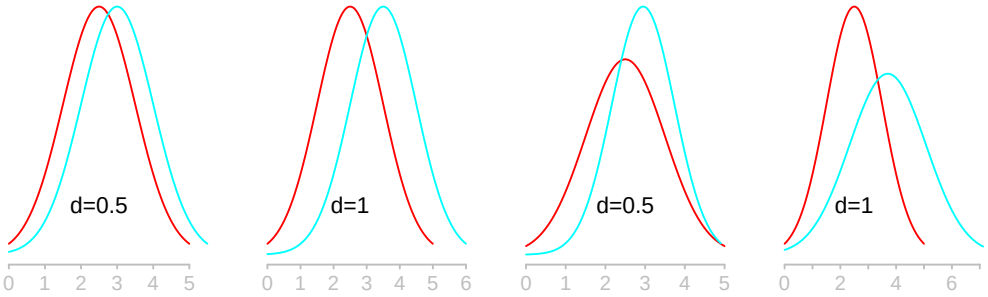


Figure 5.19: Visualization of Cohen's d for two normal distributions having different means and the same standard deviation (two left) and both different (right). code

- odds ratio, that is the ratio of the odds (i.e., $\frac{p}{1-p}$) of an event occurring in one sample divided by the ratio of the same event occurring in the other sample (perhaps a control group),
- other methods are discussed where applicable.

5.6 Statistical power

If an effect exists and an experiment is performed to measure it, what is the likelihood that the effect will be detected? The numeric answer to this question is known as *statistical power*, of the experiment. The *power* of a statistical test is its ability to detect a difference when one actually exists in the data. Failing to detect an effect when one exists is known as making a *Type II error*, or more commonly as a false negative (β is commonly used to denote the Type II error rate). Techniques for reducing Type II errors include:

- being willing to accept a larger *Type I error* (α is commonly used to denote the Type I error rate),
- sample from a population that has a higher probability of containing the sought after characteristics. For instance, Vasa[586] excluded releases with less than 30 changed classes in a study of class change dynamics. If a subset of a population is selected to maximise detection rate, care must be taken to ensure that any statement of statistical power refers to the subset population, not the larger population from which it was subsetted,
- increasing the number of measurements made.

The area of the unknown distribution excluding β (i.e., $1-\beta$) is known as the power of the test.

A power of 80% is often quoted[116] as being an acceptable lower limit of a test having a high power, just like 5% is often quoted as an acceptable significance level in many disciplines.

As an example, Figure 5.20 shows the distribution of measurements in two populations: X (red) and Y (green) (e.g., the time taken to execute all possible programs, with all possible input, on two different computers). The upper and middle plot only differ in mean value, while the middle and lower plot only differ in standard deviation. The false positive rate, α, is shaded in red, and the false negative rate, β, in green. The two rates are connected in that increasing one decreases the other, and vice versa.

Measuring an entire population is not usually practical, (e.g., all programs and over all input data) will not be available; a sample of the population is measured.

When there is a large overlap between populations (middle plot), most of the measurements in the Y sample may have values that suggest that were drawn from the X population. There is less overlap in the upper and lower plots and sample values are more likely to appear to be drawn from different distributions.

In the upper plot, the larger difference in the population mean makes it more likely that sample measurements from Y will have values that are more likely to appear to be drawn from a different distribution than samples from X. In the lower plot, there is less overlap because of the smaller population standard deviations.

If there is a need to find out whether an effect exists (e.g., one computer is faster than another or a new algorithm uses less memory), the first question is whether the effect is likely to

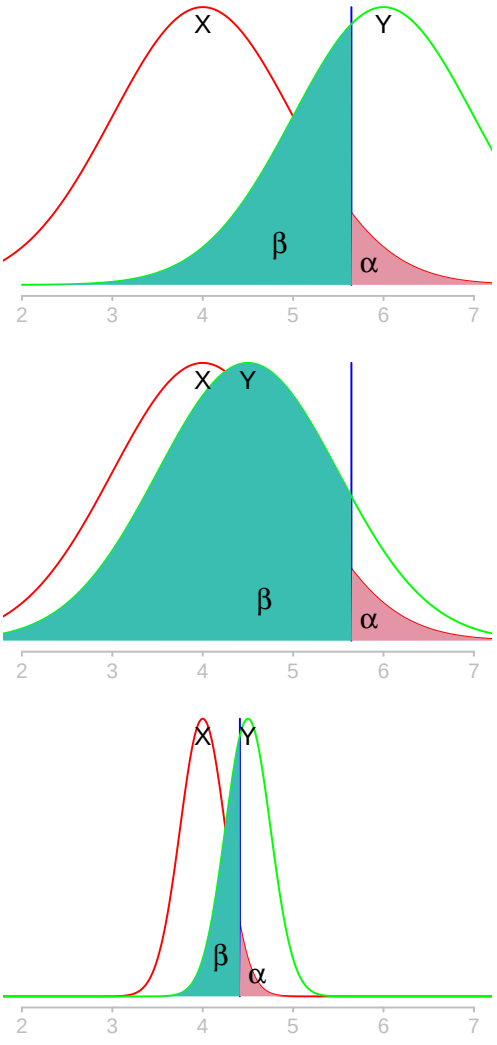


Figure 5.20: The impact of differences in mean and standard deviation on the overlap between two populations (α: probability of making a false positive error, and β: probability of making a false negative error). code

be detected using the available resources (e.g., time and effort needed to obtain a measurement sample). A statistical power calculation enables the tradeoffs between sample size and probability of detecting an effect (assuming a given population mean, standard deviation and amount of difference between two or more samples) to be analysed.

The Monte Carlo simulation can be used to obtain an estimate of the likelihood that a particular test will detect an effect in a sample of the population. The algorithm works by simulating the experiment under consideration by, obtaining samples from the population(s) that are thought to exist and performing the analysis on each sample, counting each success/failure to detect the difference.

The following code creates two populations and then compares two samples drawn from these populations. The user written function `some_test_statistic` compares two samples and returns the probability that they have some property (rexample[boot-power.R] contains an example that checks for a difference in mean value between samples drawn from two populations, see Figure 5.21):

```
boot_power=function(pop_1, pop_2, sample_size, test_stat, alpha=0.05)
{
num_samples=5000 # Number of times to run the 'experiment'.
results=sapply(1:num_samples, function(X)
                {
                sample_1=sample(pop_1, size=sample_size, replace=TRUE)
                sample_2=sample(pop_2, size=sample_size, replace=TRUE)
                return(test_stat(sample_1, sample_2, alpha))
                })

return(sum(results<alpha)/num_samples) # percentage detected
}

# Create two slightly different populations.
population_1=rnorm(100000, mean=0, sd=1)
population_2=rnorm(100000, mean=0+0.5, sd=1)

boot_power=function(population_1, population_2, 20,
                            some_test_statistic, alpha=0.05)
```

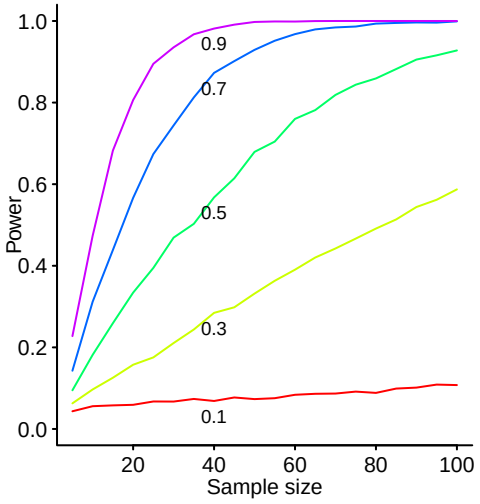


Figure 5.21: The power of a statistical test at detecting that a difference exists between the mean value of two sample drawn from two populations, both having a Normal distribution. code

In the case where both populations have a Normal distribution, an analytic solution is available for calculating the statistical power of using the t-test to detect a difference in two sample means (available in the `power.t.test` function). However, except for a few special cases, analytic solutions are not known. The lack of an analytic solution is not a problem if a computer is available, a Monte Carlo approach can be used.

Figure 5.21 shows a plot of the results of a Monte Carlo simulation testing for a difference in the mean of two samples drawn from two populations (randomly generated, using `rnorm`, to have various differences in their mean), using various sample sizes (see rexample[response-power.R] for the values returned by the analytic solution).

Obtaining good enough accuracy from a power analysis requires a good approximation of the characteristics of the population distribution. This information might come from the results from the analysis of related measurements, a preliminary study or theory.

An analysis of the data from a study by Syed, Robinson and Williams[558] of the correlation between hardware characteristics and intermittent faults in Firefox, failed to find any correlation. A statistical power analysis shows that the experiment had a low probability of detecting such a correlation if it existed...

Benchmarking to test impact of code-checking options...?

5.7 Meta-Analysis

Some issues are sufficiently interesting that they have been the subject of multiple studies. Meta-analysis is the process of systematically reviewing all the evidence available from multiple studies to produce a combined result.

Historically only descriptive statistics of samples has been available, rather than the raw data, and meta-analysis has primarily involved combining these values to produce a big picture

result. When the raw data from more than one study is available, it may be possible to combine it into a form that can be used to repeat the analysis with a larger sample size.

The `meta` package ...

A study by Sabherwal, Jeyaraj and Chowa[495] performed a meta-analysis to compute a correlation matrix based on 612 findings from 121 studies published between 1980 and 2004...

Chapter 6

Regression modeling

6.1 Introduction

Regression modeling is the default hammer used in this book to solve data analysis problems in software engineering.[i] The tree diagram, Figure 6.1, gives a high level overview of the various kinds of hammers available in the regression modeling toolkit. Concentrating on a single, general technique, removes the need for developers to remember how to select from, and use, many special case techniques (which in many cases only return a subset of the information returned from regression modeling).

The tree diagram connects the various regression techniques using the characteristics of the data they are designed to handle, with the techniques in red being the common use cases for their respective data characteristics.

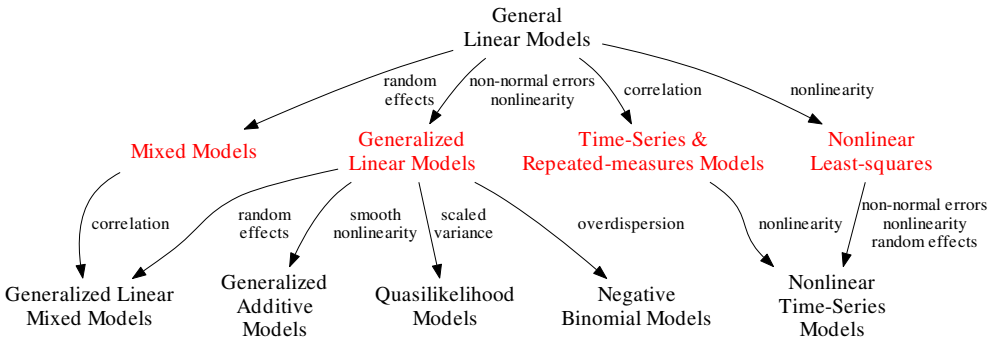


Figure 6.1: Relationship between data characteristics (edge labels) and applicable techniques (node labels) for building regression models.

Regression modeling is powerful enough to fit almost any data to any desired degree of accuracy, which means overfitting is an ever present danger and model validation (e.g., how well a model might fit new data or an estimation of the benefit obtained from including each coefficient in a model) is an important self-correcting step.

As always, it is necessary to remember the adage: 'All models are wrong, but some are useful.'

The two main reasons for building a regression model are:

- understanding: by combining the explanatory variables into an equation that can be used to interpret why the response variable behaves the way it does,
- prediction: that is predicting the value of the response variable, for given values of the explanatory variables.

The focus of predictive modeling is accuracy of prediction and there is a willingness to trade-off understanding what is going on for greater accuracy, while the focus of interpretive modeling is understanding why and this creates a willingness to trade-off prediction accuracy for model simplicity.

[i] Machine learning is a not a regression modeling technique and is covered later.

Understanding is the primary focus for most of the model building in this book; builders of computing systems are generally interested in controlling what is happening, with predicting being a fall back position, and control requires understanding. Model building for prediction is often easier than building for understanding, so readers should not find it difficult switching to a predictive focus.

Regression models contain a *response variable*, one or more *explanatory variables*[ii] and some form of error term.

The *response variable* is modeled as some combination of *explanatory variables* and an additive or multiplicative error term (the error term associated with each explanatory variable represents behavior not accounted for by the explanatory variable; different kinds of regression model make different assumptions about the characteristics of the error term).

It is always possible to build a model that fits the data to within some degree of error, i.e., the amount of variation in the measurements that the model does not explain. It is very important to always ask how well a model fits the known data, not just the data used to build it.

If a sample contains many variables, then it is sometimes possible to build a model that has an impressive fit to the chosen response variable using only a few of the other variables. A study by Zeller, Zimmermann and Bird[625] built a fault prediction model whose performance was comparable to the best available at the time. The model used four explanatory variables to predict the probability of a fault being reported in the source code contained in each file; the explanatory variables were the percentage occurrence of each of the letters `IROP` in the source code of each file. The model was *discovered* by checking how good a job every possible source code character did at predicting fault probability.

6.2 Linear regression

The simplest form of regression model is linear regression, where the *response variable* is modeled as a linear combination of *explanatory variables* and an additive error (the error terms are assumed to be independent and identically distributed; ε denotes the total error). The equation is:

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \varepsilon$$

Note that the term *linear* refers to the coefficients of the model, i.e., β, not the form taken by the explanatory variables which may have a non-linear form, as in:

$$y = \alpha + \beta x^2 + \varepsilon$$

or:

$$y = \alpha + \beta \log(x) + \varepsilon$$

A linear model is perhaps the most commonly used regression model, reasons for this include:

- many real world problems exhibit linear behavior, or a good enough approximation to it for practical purposes, over their input range,
- they are much easier to fit manually than more sophisticated models and until recently software to build other kinds of models was not widely available,
- they can generally be built with minimal input from the user (apart from having to decide which column of the data is the response variable).

The `glm` function[iii] builds a linear model and the use common case requires two argument value, a formula expressing a relationship between variables (response variable on the left and explanatory variable(s) on the right) and an object containing the data (this object is required to contain columns whose names match the identifiers appearing in the formula).

[ii] Books that concentrate on the predictive aspect of models use the term *prediction variable* or just *predictor*, while those that concentrate on running experiments use terms such as *control variables* or just the *controls*.

[iii] Many statistics books start by discussing the `lm` function, rather than `glm`, because the mathematics that underpins it is easier to learn; if you dear reader want to learn this mathematics I recommend taking this approach. As its name implies the Generalised Linear Method has a wider range of applicability and its use here is in line with the aim of teaching one technique that can be used everywhere. Also, the mathematics behind `glm` make fewer assumptions about the sample characteristics, e.g., it does not require the variance in the error to be constant (which `lm` does).

The formula has the form of an equation, with the = symbol replaced by ~ (pronounced *is distributed according to*) and the coefficients α and β are implicitly present, i.e., they do not need to be explicitly specified.

The following code uses `glm` to build a model showing the relationship between the number of lines of source code (*sloc*) in FreeBSD and the number of days elapsed since the project started (in 1993):

```
BSD_mod=glm(sloc ~ Number_days, data=bsd_info)
```

The fitted equation is:

$$E[sloc] = \alpha + \beta \times Number\_days$$

where $E[sloc]$ is the expected value of *sloc* (the error term is discussed below).

Figure 6.2 shows the measurement data points and the straight line whose coefficients are contained in the object returned by `glm`.

The `summary` function takes the object returned by `glm` and prints details about the fitted model;[iv] the following is for the model fitted to the FreeBSD data: code

```
Call:
glm(formula = sloc ~ Number_days, data = kind_bsd)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-82990  -32136   -3609   35389   87324

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.139e+05  1.171e+03   97.24   <2e-16 ***
Number_days 3.937e+02  4.205e-01  936.33   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 1657283104)

    Null deviance: 1.4610e+15  on 4826  degrees of freedom
Residual deviance: 7.9964e+12  on 4825  degrees of freedom
AIC: 116172

Number of Fisher Scoring iterations: 2
```

The table following the `Coefficients:` header, in the `summary` output, lists estimated fitted values for α and β (Intercept and `Number_days` respectively), the standard error in these estimates (`Std.Error`) and (in the `Pr(>|t|)` column) the probability that if the true value of the coefficient was zero the estimated value would have occurred by chance.

The values listed in the `summary` output can be plugged into the model formula to give the following fitted equation:

$$sloc = 1.139 \times 10^5 + 3.937 \times 10^2 Number\_days$$

The fit between the model and the data is not perfect and the following are the two forms of uncertainty, or variation, present in the model:

1. Uncertainty in the values of the model coefficients. The values listed in the `Std. Error` column denote one standard deviation, which when added to the model gives the following:

   $$sloc = (1.139 \times 10^5 \pm 1.171 \times 10^3) + (3.937 \times 10^2 \pm 4.205 \times 10^{-1}) Number\_days$$

2. Uncertainty caused by the inability of the explanatory variable used in the model to explain everything. This uncertainty is the ε appearing in equation <???>; the term *residual* is used to denoted this quantity. In the general case it is unlikely that ε will have a fixed value over the range of values supported by a model and `glm` does not generate a single value.

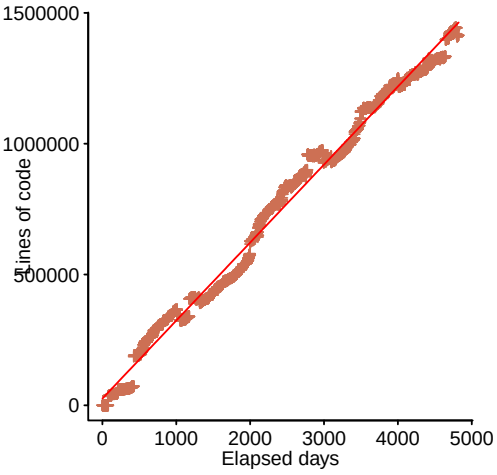


Figure 6.2: Total lines of source code in FreeBSD by days elapsed since the project started (in 1993). Data from Herraiz.[559] code

[iv] Only a few digits of the estimated values are printed by default.

In Figure 6.2 the variations in the unexplained error, ε, appear to be small. Assuming a fixed value, a call to `aov` returns 40,710 as the residual standard error. The equation including this estimate of the residual is:

$$sloc = 1.139 \times 10^5 + 3.937 \times 10^2 Number\_days \pm 4.071 \times 10^4$$

In other words the difference between measured values and values predicted by this fitted model will have a standard error of 4.071×10^4.

The object returned by the call to `glm` can be used to make predictions and these can be overlaid on the output from an earlier call to `plot`, as follows:

```
BSD_pred=predict(BSD_mod)  # predict using measured values
lines(BSD_pred, col="red") # x-axis start at 1 and increment
```

The `predict/lines` approach follows this book's aim of using techniques that work for the general case. Plotting a fitted straight line is such a common operation that there is a function for doing just that, e.g., `abline(reg=BSD_mod, col="red")`, but this does not always work when the axis have been scaled in some way and is of no use for fitted models that are more complicated than a straight line.

Before being carried away with the high degree of agreement between this model and the data it is important to remember that the model has a number of characteristics that do not reflect reality, including:

- source code does not spontaneously grow of its own accord and the only justification for treating *number of days* as an explanatory variable is that the resulting model provides potentially interesting insight into the rate of growth of these software systems.
- when it started the BSD project contained zero lines of code, but this model has an Intercept of 1.39×10^5,
- the model shows the number of lines increasing for ever, at a constant rate, whereas at some point in the future growth must slow down and eventually stop,
- it says nothing about large amounts of code being added/removed over very short periods of time (known to exist because of the very visible breaks in the connectedness of the plotted values).

While the model has various disconnects with reality, it does provide strong evidence that growth has been remarkable constant over a long period. Unless there are seismic changes within the FreeBSD development world the constant rate of code growth would be expected to continue to hold for a non-trivial number of days into the future.

The call to `summary`, passing the value returned by `glm`, is an example of function overloading in action. The value returned by `glm` has class `glm`, which when passed as an argument to `summary` results in `summary.glm` being called; a call to `predict` results in `predict.glm` being called (function overloading is the most common use of object oriented constructs in R programs; the use of a period in the function name is a naming convention followed by the implementers and not something automatically added by the compiler).

Some of the factors and processes that might be involved in driving a fixed rate of growth over 20 years include:

- developers working on the system have continually found new functionality to add,
  - if there has always been functionality to add, why haven't more developers become involved to increase the rate of growth until there is less to do?
  - to what extent is the continual stream of new hardware devices responsible for driving growth?
- what are the bottlenecks that have prevented increases in growth rate when the resources have been available?
  - has growth rate remained constant because the developers working on the systems have remained constant?
  - is there a buffer of code waiting to be released, whose growing and shrinking hides an internal growth rate that is much more variable than the externally visible rate?

6.2.1 Scattered measurement values

In the previous example the measurements ran together in a way that created a visually recognizable line. The common case is not always so accommodating and when many samples are plotted a scattering of disjoint points often appears; viewed as a whole a general trend may emerge.

A study by Kampstra and Verhoef[319] investigated the estimated cost and duration of 73 large Dutch federal IT projects.[v] Figure 6.3 shows that very few of the measurement points are on the (red) line specified by the model returned by `glm`; the variability of the measured values is much larger than that for the FreeBSD model. While numeric estimates of the uncertainty present in the fitted model are readily available, interpreting these numeric values requires a degree of effort and some experience. A confidence interval provides an easy to interpret visual representation of the uncertainty in a fitted model.

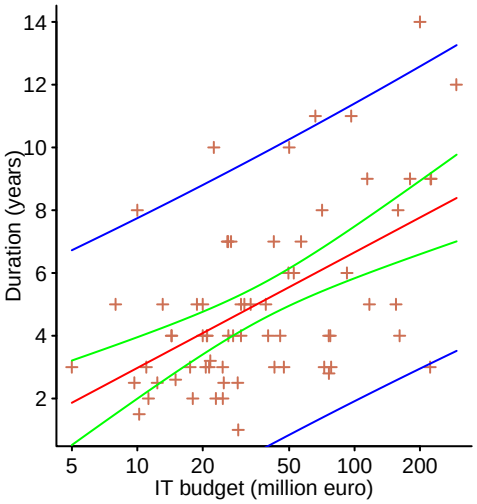


Figure 6.3: Estimated cost and duration of 73 large Dutch federal IT projects, along with fitted model and 95% confidence intervals. Data from Kampstra et al.[319] code

The kind of uncertainty of interest will depend on whether the model is built to gain understanding or make predictions:

- when understanding is the priority, the confidence interval of interest involves the estimated model coefficients:
  - a call to `predict` with the `se.fit=TRUE` argument returns the standard error for each fitted value. Multiplying `se.fit` by `qnorm`[vi] converts the returned value to a 95% confidence interval (2.5% above and below the fit; the two `qnorm` values differ only in sign because the Normal distribution is symmetrical), i.e., there is a 95% expectation that the actual model fits within the interval enclosed by these lower/upper bounds. `qnorm(0.975)==1.96` and the literal value is often used (in fact the value 2 is often seen as a sufficiently close approximation).[vii]

```
fed_pred=predict(fed_mod, newdata=list(log.IT=1:7, log.IT_sqr=(1:7)^2),
                                                    se.fit=TRUE)
lines(fed_pred$fit, col="green")     # fitted line
# CI above and below
lines(fed_pred$fit+qnorm(0.975)*fed_pred$se.fit, col="green")
lines(fed_pred$fit+qnorm(0.025)*fed_pred$se.fit, col="green")
```

  - the `confint` function in the `MASS` package or the `boot.ci` function in the `boot` package can be used to obtain a point estimate of the confidence interval of the fitted model coefficients.

- when prediction is the priority the interval is known as the *prediction interval*; the bounds between which newly measured values are expected to appear. Two sources of uncertainty are added to calculate the prediction interval: uncertainty in the model coefficients (i.e., the confidence interval) plus the variance in the data not explained by the fitted model.

```
# print.aov also calculates it from residuals returned by glm...
MSE=sum(fed_mod$residuals^2)/(length(fed_mod$residuals)-2)
# Variances, but not sd, can be added
pred_se=sqrt(fed_pred$se.fit^2+MSE)
lines(fed_pred$fit+1.96*pred_se, col="blue")
lines(fed_pred$fit-1.96*pred_se, col="blue")
```

When measurement values and a fitted regression line are plotted, it is easy to visually fixate on the line and forget about the associated uncertainties. Including a confidence band as part of the plot provides a vivid visual reminder of the quality of the fit.

Plotting values does not always reveal an obvious pattern in the points. A visual pattern may not exist because no relationship exists between the response and explanatory variables or because the noise in the data is much greater than the signal (i.e., a relationship that does exist is swamped by the noise present in the measurements).

How much random scattering of measurement values has to exist before a fitted regression model can be said to be not worth bothering about?

[v] They discovered that there was a lot of uncertainty in the estimates given.?

[vi] This calculation assumes that the measurement error has a Normal distribution, the default assumption made by `glm` when building a model.

[vii] For small sample sizes a call to `qt` may be more accurate.

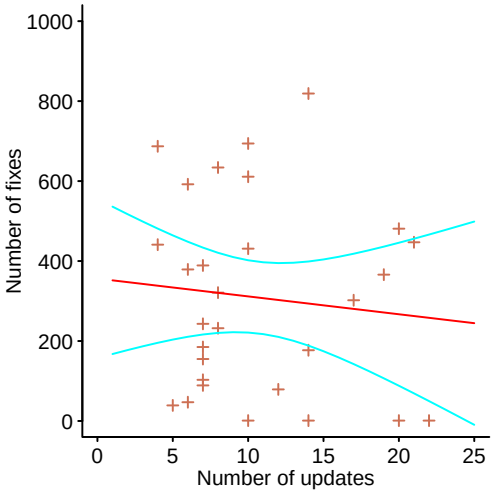


Figure 6.4: Number of updates and fixes in each Linux release between version 2.6.11 and 3.2. Data from Corbet et al.[123] code

The `glm` function and many other model building functions available in R are capable of fitting models to data points that are randomly distributed. For instance, Figure 6.4 shows the number of updates and fixes made in various Linux versions released between early 2011 and 2012. The standard error of the fitted line show that its slope could have a positive or negative value.

The output from `summary` shows how poor the fit actually is; the `Pr(>|t|)` column lists the p-value for the hypothesis that the coefficient in the corresponding row is zero, i.e., that no relationship was found to exist for that component of the model. code

```
Call:
glm(formula = Fixes ~ Total.Updates, data = cleaned)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-310.60  -223.67     0.48   184.51   525.26

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    356.233    101.522   3.509   0.0016 **
Total.Updates   -4.464      8.478  -0.526   0.6029
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 60685.71)

    Null deviance: 1655335  on 28  degrees of freedom
Residual deviance: 1638514  on 27  degrees of freedom
AIC: 405.62

Number of Fisher Scoring iterations: 2
```

6.2.2 Discrete measurement values

Regression models are not limited to using continuous numeric explanatory variables, variables having nominal values can also be used.

A study by Cook and Zilles[122] investigated the impact of compiler optimization flags on the ability of software to continue to operate correctly when subject to random bit-flips, i.e., simulating random hardware errors; 100 evenly distributed points in the program were chosen and 100 instructions from each of those points were used as fault injection points, giving a total of 10,000 individual tests run, for each of 12 programs from the SPEC2000 integer benchmark compiled using gcc version 4.0.2 (using optimization options: 00, 02 and 03) and the DEC C compiler (called *osf*).

The fitted model has percentage of correct benchmark program execution as the response variable and optimization level as the explanatory variable; the following is the call to `glm`:

```
bitflip_mod=glm(pass.masked ~ opt_level, data=bitflip)
```

The following is `summary` output of the fitted model (see rexample[comp-mask.R]): code

```
Call:
glm(formula = pass.masked ~ opt_level, data = bitflip)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-12.6689   -2.8454   -0.3478    4.4017    8.1100

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    28.589      1.825  15.665  < 2e-16 ***
opt_level02     9.161      2.581   3.550  0.00112 **
opt_level03     7.429      2.581   2.878  0.00677 **
opt_levelosf   11.642      2.414   4.822 2.74e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
(Dispersion parameter for gaussian family taken to be 29.97578)

    Null deviance: 1785.8  on 38  degrees of freedom
Residual deviance: 1049.2  on 35  degrees of freedom
AIC: 249.07

Number of Fisher Scoring iterations: 2
```

Plugging the model coefficients into the regression equation we get:

$pass.masked = 28.6 + 9.2 \times D_{O2} + 7.4 \times D_{O3} + 11.6 \times D_{osf}$

where: D_i, known as a *dummy variable* or *indicator variable*, take one of two values:

$$D = \begin{cases} 1 & \text{optimization flag used} \\ 0 & \text{optimization flag not used} \end{cases}$$

The value for optimization 00 is implicit in the equation, it occurs when all other optimizations are not specified, i.e., its value is that of the intercept.

The standard error in the 02 and 03 compiler options is sufficiently large for their respective confidence bounds to significantly overlap. This suggests that these two options have a similar impact on the behavior of the response variable.

6.2.3 Uncertainty only exists in the response variable

The algorithms used to fit regression models often attempt to minimise the difference between the measured points and a specified equation. For instance, least squares minimises the sum of the squares of the distance along one axis between each data point and the fitted equation;[viii] alternative minimization criteria are discussed later, e.g., giving greater weight to positive error than negative error.

An important, and often overlooked, detail, is that many regression techniques assume that the values of the explanatory variable(s) contain no uncertainty (e.g., measurements are exact), with all uncertainty, ε, occurring in the response variable (see the first equation at the start of this chapter).

A consequence of assuming that uncertainty only exists in the response variable is that the equation obtain by fitting a model that specifies, say, X is the explanatory variable and Y is the response variable will not be consistent with a model that assumes Y is the explanatory variable and X is the response variable. That is, algebraicly transforming the first equation produces an equation whose coefficients are different from the second.

Take as an example, data from Kroah-Hartman[230] who measured the number of commits made between the release of a version of Linux and the immediately previous version, and the number of developers who contributed code to that release, for the 67 major kernel releases between versions 2.6.0 and 4.6.

In the upper plot of Figure 6.5 the number of developers is treated as the explanatory variable (x-axis) and number of commits as the response variable (y-axis), with the fitted regression line in red and dashed lines showing the difference between measurement and fitted model. In the lower plot the explanatory/response roles played by the two variables when fitting a regression model is switched; to simplify comparison the axis denote the same variables in both plots, with the green line denoting the fitted model and dashed lines showing the difference between measurement and model (now on the x-axis response variable; the line fitted in the upper plot is also given for comparison, still in red).

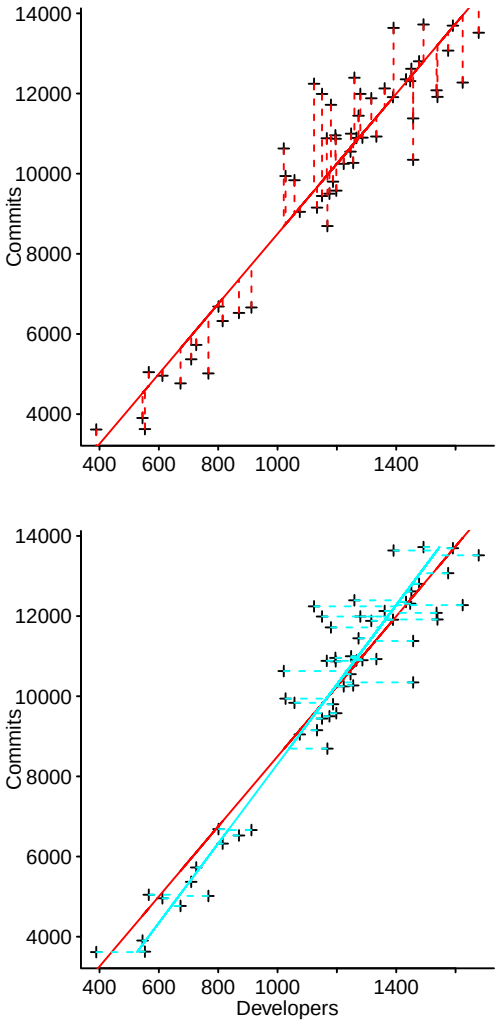


Figure 6.5: The number of commits made and the number of contributing developers for Linux versions 2.6.0 to 3.12. The green line in the right plot is the regression model fitted by switching the x/y values. Data from Kroah-Hartman.[230] code

In the first case the fitted equation is:

$commits = -237 \pm 523 + (8.7 \pm 0.44) \times Number\_devs$

transforming this equation we get:

$$\begin{aligned} Number\_devs &= \frac{237 + commits}{8.7} \\ &= 27 + 0.11 commits \end{aligned}$$

[viii] Minimising the sum of squares in the error has historically been popular because it is a case that can be analysed analytically.

However, if a new model is fitted by switching the roles of the two variables in the formula passed to `glm`, the returned model is described by the following equation:

$$Number\_devs = 162 \pm 52 + (0.10 \pm 0.005) \times commits$$

which is different from the equation obtained by transforming the first fitted model.

There is another difference between the two fitted models, the second model has a better fit to the data. Somebody only interested in the quality of fit might be tempted to select the second model, purely for this reason.

What is the procedure for deciding which measurement variables play the role of response and explanatory variable, e.g., should number of developers be considered an explanatory or response variable?

An important attribute of explanatory variable(s) is that their value is controlled by the person making the measurement. For instance, the model building process used to create Figure 6.2 had number of days as the explanatory variable; this choice was completely controlled by the person making the measurements.

The Kroah-Hartman commit measurements are based on the day of release of a version of the Linux kernel, a date that is outside the control of the measurement process. In fact both measurements have the characteristics of a response variables, that is, the value they have was not selected by the person making the measurement. The possibility of variation in Linux version release dates is a source of uncertainty that needs to be treated as a form of measurement error.

Building regression models using explanatory variables containing measurement error can result in models containing biased and inconsistent values, as well as inflating the Type I error rate.[82,512]

There are a variety of regression model building techniques that take into account error in the explanatory variable. These techniques are sometimes known as *model II* linear regression techniques (model I being the case where there is no uncertainty in the explanatory variables), *errors-in-variable models*, *total least squares* or *latent variable models*; using methods such as *major axis*, *standard major axis* and *ranged major axis*.

If all the variables used to build a model contain some amount of error, then it is necessary to decide how much error each variable contributes to the total error in the model. Some model II techniques are not scale invariant, that is they are only applicable if both axis are dimensionless or denote the same units, otherwise rescaling one axis (e.g., converting from kilometers to miles) will change its relative contribution; if each axis denotes a different unit it does not make sense to use a model building technique that attempts to minimise some measure of combined uncertainty.

SIMEX (SIMulation-EXtrapolation) is a technique for handling uncertainty in explanatory variables that works in conjunction with a range of regression modeling techniques. While the SIMEX approach does not suffer from many of the theoretical problems that other techniques suffer from, it does require that the model builder provide an estimate of the likely error in the explanatory variable. The `simex` package implements this functionality and supports a wide variety of regression models built by functions from various packages.

Continuing with the Linux developer/commit count example, to build a regression model using SIMEX we need an estimate of the uncertainty in the number of developers contributing at least one commit to any given release. The `simex` function taking a model built using `glm` (and by other regression model building functions) and an estimate of the uncertainty in one or more of the explanatory variables and returns an updated model that takes this uncertainty into account.

The following is a rough and ready approach to estimating the uncertainty in the Kernel attributes measured by Kroah-Hartman:

- the release date of a new version of Linux is assumed to have an uncertainty of ±14 days about the actual release date.[ix]
- the possible variation in the unique contributor count for any release is assumed to be uniformly distributed in the range: measured contributor count plus/minus number of developers contributing their first commit in the last 14 days.

[ix] Pointers to a more reliable, empirically derived, value are welcome.

- making the above assumptions we get a standard deviation of 41 for the number of unique developers making at least one commit, averaged over all versions (see rexample[dev-commit.R]).

This estimate of the standard deviation in the explanatory variable is integrated into a regression model that takes account of uncertainty in more than just the response variable as follows:

- first build a regression model using `glm` in the usual way, but with the optional named parameter `x` set to `TRUE` (`y` also needs to be `TRUE`, but this is its default value and so the assignment below is redundant),
- pass the model returned by `glm` to `simex`, along with the name of the explanatory variable and its estimated standard deviation.

```
yx_line = glm(commits ~ developers, x=TRUE, y=TRUE)

sim_mod=simex(yx_line, SIMEXvariable="developers", measurement.error=41)
```

The `summary` output (see rexample[regression/dc-simex.R]) shows that the model returned by `simex` is the following:[x]

$$commits = -387 \pm 453 + (8.9 \pm 0.4) \times Number\_devs$$

The error in each individual explanatory variable measurement can be specified by assigning a vector to `measurement.error` (the argument `asymptotic=FALSE` is also required); see Figure 6.35.

There are techniques, and R packages, for building complete models starting from the data, rather than refitting an existing regression model. For those wanting to a build model from scratch the `lmodel2` package provides functions that implement many of the available methods.

How reliable is a fitted model that has been built by ignoring any uncertainty/error in explanatory variable measurements? The only way to answer this question is to build a model that takes this error into account and compare it with one that does not.

It is incorrect to assume that the model fitted by switching response/explanatory variables will fall within the 95% confidence intervals of the existing model, as Figure 6.6 shows. One Effort/Size fit has a slope greater than one and the other less than one, with the value one being an important dividing line in the interpretation of behavior (i.e., do economies of scale exist for software development).

Many of the measurement values treated as explanatory variables in this book were not under the control of the person who measured them. For instance, lines of code, number of files and reported problems measured at a given point in time are all response variables. To reduce your authors workload most of the model fitting in this book does not make any adjustments for errors in the explanatory variables.

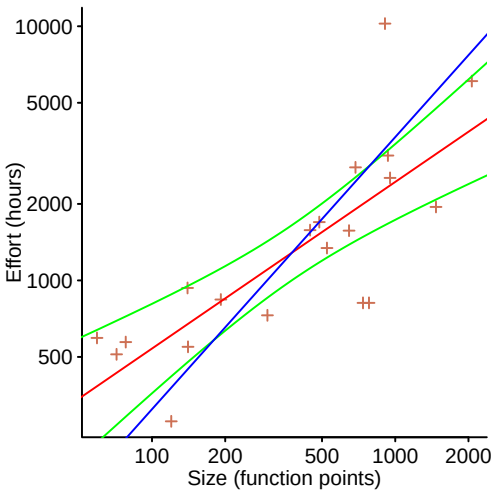
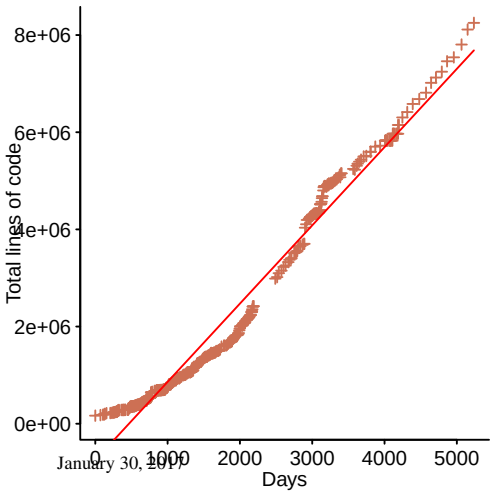


Figure 6.6: Effort/Size of various projects and regression lines fitted using Effort as the response variable (red, with green 95% confidence intervals) and Size as the response variable (blue). Data from Jørgensen et al.[?] code

6.2.4 Modeling data that curves

A model based on a straight line is a wonderful thing to behold, it is simple to explain and often aligns with people's expectations (many useful real world problems are well fitted by a straight line). However, life is complicated and throws curved data at us.

Having encountered an operating system having constant lines of code growth over many years, it is tempting to draw a conclusion about the growth rate of other operating systems. However, the way in which the data points curve around the fitted line in the upper plot of Figure 6.7 suggests that some of the processes driving the growth of the Linux kernel are different from those driving FreeBSD; perhaps a quadratic or exponential equation would be a better fit (these possibilities were chosen because they are two commonly occurring forms for upwardly curving data).



[x] Readers might like to experiment with the value of the `measurement.error` to see the impact on the model coefficients.

This section is about fitting linear models, so the possibility of an exponential fit is put to one side for the time being; building non-linear models, including better fitting non-linear models to this data, is discussed later.

The following call to `glm` fits an equation that is quadratic in the variable `Number_days`; the righthand side of the formula contains `Number_days+I(Number_days^2)`. The `I` (sometimes known as *as-is*) causes its argument to remain unevaluated and is treated as a distinct explanatory variable (without the `I`, `Number_days` would be squared, added to the first instance and the sum treated as a single explanatory variable). An alternative way of including a squared explanatory variable in the model is to assign the value `Number_days^2` to a new variable and include this new variable's name on the righthand side of the formula. The result of fitting this equation is shown on the lower plot of Figure 6.7.

```
linux_mod=glm(sloc ~ Number_days+I(Number_days^2), data=linux_info)
```

The quadratic fit looks like it could be better than linear, but perhaps a cubic, quartic or higher degree polynomial would be even better. The higher the order of the polynomial used, the smaller the error between the fitted model and the data used. The error decreases because the additional terms are used to adjust the model to do a better job of following the random fluctuations in the data. An Occam's razor method is needed to select the number of terms that produces the simplest model consistent with the data and having an acceptable error.

The Akaiki Information Criterion, AIC, is a commonly used metric for comparing two or more models (available in the `AIC` function). It takes into account both how well a model fits the data and the number of free coefficients in the model; free coefficients have to pay their way by providing an appropriate improvement in a model's fit to the data.[xi] AIC can also be viewed as the information loss when the true model is not among those being considered.[87]

One set of selection criteria[87] are that models whose AIC differ by less than 2 are more or less equivalent, those that differ by between 4 and 7 are clearly distinguishable, while those differing by more than 10 are definitely different.

How much better does a quadratic equation fit Linux SLOC growth compared to a straight line and how much better do higher degree polynomials fit? The following lists the AIC for models fitted using polynomials of degree 1 to 4 (lower values of AIC are better). After initially decreasing the AIC starts to increase once a fourth degree polynomial is reached; the third degree polynomial is thus the better fitting linear polynomial of those tested, i.e., other forms of equation could be better. code

```
[1] Degree 1, AIC= 13998.0004739753
[1] Degree 2, AIC= 13674.6883243397
[1] Degree 3, AIC= 13220.8542892188
[1] Degree 4, AIC= 13221.7072389496
```

The following is the `summary` output of the fitted cubic model: code

```
Call:
glm(formula = LOC ~ Number_days + I(Number_days^2) + I(Number_days^3),
    data = latest_version)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-428217   -80061     6503    64889   620500

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)       3.432e+05  2.876e+04  11.935  < 2e-16 ***
Number_days      -3.664e+02  5.144e+01  -7.123 3.79e-12 ***
I(Number_days^2)  8.167e-01  2.456e-02  33.258  < 2e-16 ***
I(Number_days^3) -9.184e-05  3.371e-06 -27.242  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 23001633959)
```

[xi] A negative value may be the result of a nominal explanatory variable having many values, e.g., dates are represented as strings that are best converted using `as.Date`.

```
    Null deviance: 1.8867e+15  on 494  degrees of freedom
Residual deviance: 1.1294e+13  on 491  degrees of freedom
AIC: 13221

Number of Fisher Scoring iterations: 2
```

Regression modeling finds the best fit for an equation over the range of the data used and AIC helps prevent overfitting. No claims are made about how well the model is likely to fit data outside the range that was used to build it. Using a model optimized to fit the available data to make predictions outside the interval of the data used can produce very surprising results.

What is the behavior of the cubic model outside its fitted range and in particular what predictions does it make about future growth? This model predicts a future decrease in the number of lines of code (see Figure 6.8). A decreasing number of lines is the opposite of previous behavior this prediction is unlikely to be believed by many people (if this behavior were predicted by a more detailed model of code growth that closely mimicked real-world development by using information on the number of developers actively involved and a list of functionality that is likely to be implemented, it might be more believable).

A quadratic equation might not fit the data as well as a cubic equation, but the form of its predictions (increasing growth) is consistent with expectations.

If the purpose of the model is understanding, then the quadratic model maps more closely to anticipated behavior; if the purpose is prediction within the interval of the fitted data, then the cubic model is likely to have a smaller error.

What about fitting other kinds of equations to the data? Equations such as $Y = \alpha e^{\beta X} + \varepsilon$ an $Y = \alpha X^{\beta} + \varepsilon$ are nonlinear in β; non-linear model building is covered later.

For a software system to grow more code has to be added to it than is deleted. A constant rate of growth suggests either a constant amount of developer effort or a bottleneck holding things up; an increasing rate of growth (i.e., quadratic) suggests an increasing rate of effort. The different code growth pattern seen in the Linux kernel, compared to NetBSD/FreeBSD and various other applications, has been tracked down to device driver development;[208] new hardware devices often share many similarities with existing devices and for Linux developers tend to copy an existing driver, modifying it to handle the hardware differences; it is this reuse of existing code that is the source of what appears to be a non-linear growth in developer effort.

This method of creating a new device driver, performed by many developers working independently, can continue for as long as there are new devices coming to market; the evolution of Linux device drivers is discussed elsewhere...

A linear regression model is not limited to using polynomials of explanatory variables, any function can be used as long as the coefficients of the model occur in linear form. For instance, the FreeBSD model plotted in Figure 6.2 might include a seasonal term that varies with time of year; while a model containing the term $A\sin(2\pi ft + \phi)$ is nonlinear (because of ϕ, the phase shift[xii]) it can be written in the following linear form:

$$A\sin(2\pi ft + \phi) = \alpha_s \sin(2\pi ft) + \alpha_c \cos(2\pi ft)$$

where: $\alpha_s = A\cos\phi$ and $\alpha_c = A\sin\phi$; $A = \sqrt{\alpha_s^2 + \alpha_c^2}$ and $\phi = \arctan\frac{\alpha_s}{\alpha_c}$.

The call to `glm` is now (the argument to the trig functions has to be expressed in radians):

```
rad_per_day=(2*pi)/365
freebsd$rad_Number_days=rad_per_day*freebsd$Number_days
season_mod=glm(sloc ~ Number_days+
                      I(sin(rad_Number_days))+I(cos(rad_Number_days)),
                                              data=freebsd)
```

The `summary` output from the fitted model shows that while a seasonal component probably exists, its overall contribution is very small (see rexample[Herraiz-BSD-season.R]).

While fitting a model using all available measurements points is a reasonable first step, subsequent analysis may suggest that the sample might best be treated as two or more disjoint samples. There may be time dependent factors that have a strong influence on growth patterns.

[xii] Some texts use $A\cos(2\pi ft + \phi)$, which changes the phase by 90&degree and changes some signs

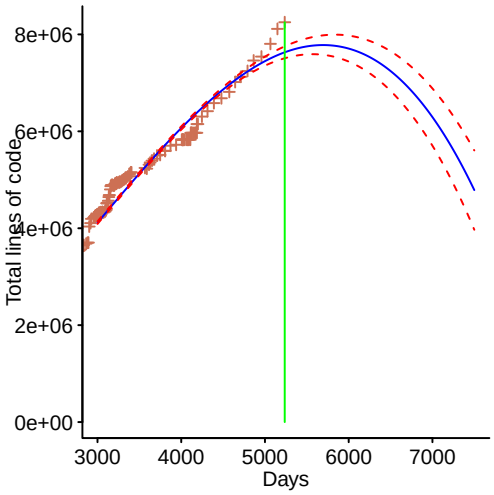


Figure 6.8: Actual (left of vertical line) and predicted (right of vertical line) total lines of code in Linux at a given number of days since the release of version 1.0, derived from a regression model built from fitting a cubic polynomial to the data (dashed lines are 95% confidence bounds). Data from Israeli et al.[289] code

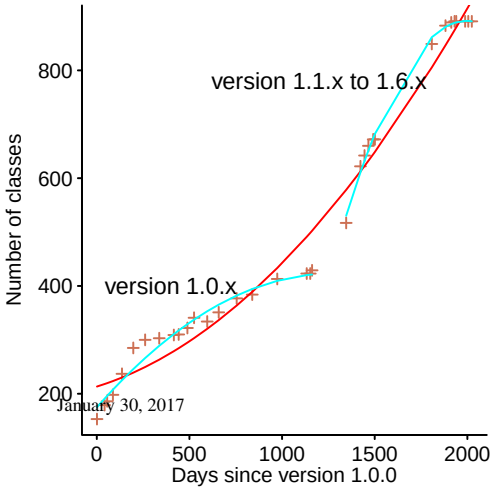


Figure 6.9 shows the number of classes in the Groovy compiler at each release, in days since version 1.0. There are noticeable kinks in the growth rate at around 1,300 and 1,500 days. Fitting a model to the complete sample shows upward trending quadratic growth in the number of classes over time, while fitting separate models to two halves of the sample shows quadratic growth that flattens out.

Some investigation finds that the kink occurs at a transition between version numbers. It is easy to invent a variety of explanations for the pattern of behavior seen, but treating the measurements as-if they came from a single continuously developed code base is probably not one of them; further investigation of the circumstances behind the development of the Groovy compiler is needed if a reasonable level of confidence is required in whatever model is finally selected.

6.2.5 Visualizing the general trend

Even when the measurement points are scattered in what appears to be an obvious general direction, it is worthwhile quickly obtaining an estimate of the trend followed by the data.

A general technique for highlighting a trend follows by data is to fit a regression model to a consecutive sequence of small intervals of the data and join this sequence of fits together to form a continuous line. Two methods based on this idea (both fitting so the lines smoothly run together) are LOWESS (LOcally WEighted Scatterplot Smoothing) and LOESS (LOcal regrESSion); `lowess` and `loess` are the respective functions, with `loess` being used in this book.

A study by Kunst[338] counted, for 148 languages, the number of lines committed to Github (between February 2013 and July 2014) and the number of questions tagged with that language name on Stackoverflow.

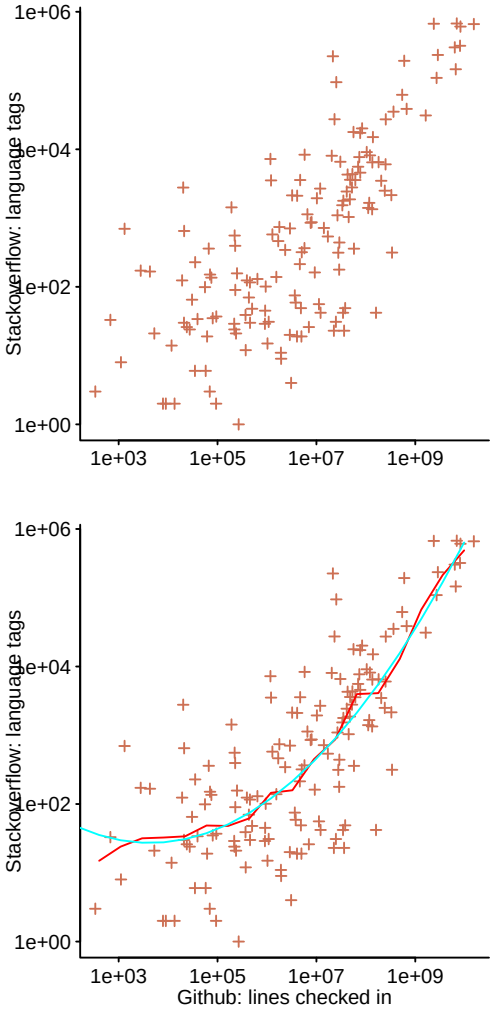


Figure 6.10: For each distinct language, the number of lines committed on Github and the number of questions tagged with that language. Data from Kunst.[338] code

The upper plot of Figure 6.10 shows lots of points which look as-if they trend in a straight line. The `loess` fit, red line in lower plot, shows the trend having a distinct curve. Experimenting with a quadratic equation in `log(lines_committed)` shows, blue line in lower plot, that this more closely follows the loess fit than a straight line (a quadratic fit has a lower AIC than a linear one; see rexample[langpop-corger-nl.R] for details).

A call to `loess` has the same pattern as a call to `glm`, with the possible addition of an extra argument; `span` is used to control the degree of smoothing:

```
loess_mod=loess(log(stackoverflow) ~ log_github, data=langpop, span=0.3)
x_points=1:max(langpop$log_github)
loess_pred=predict(loess_mod, newdata=data.frame(log_github=x_points))
lines(exp(x_points), exp(loess_pred), col=pal_col[1])
```

A study by Edmundson, Holtkamp, Rivera, Finifter, Mettler and Wagner[156] investigated the effectiveness of web security code reviews and asked professional developers to locate vulnerabilities in code.

The `lowess` fit, blue line in Figure 6.11, suggests that the percentage of vulnerabilities found increases as the number of years working in security increases, but then rapidly decreases; this performance profile seems unrealistic. A fitted straight line, in red, shows a decreasing percentage with years of work in the security field (its p-value is 0.02).

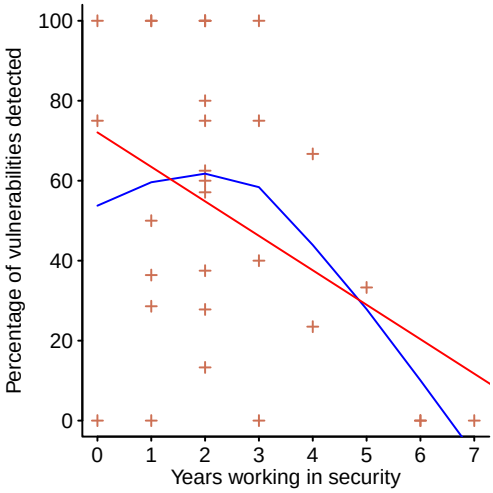


Figure 6.11: Percentage of vulnerabilities detected by developers working a given number of years in security.

Perhaps the correct interpretation of this sample is that average performance does increase with years of working in the field, but that the subjects with many years working in security, who took part in the study, were more managerial and customer oriented people who had time available to take part in the experiment, i.e., this is a subject sampling problem. At the time of the study software security work was rapidly expanding, so the experience profile will be skewed with more subjects being less experienced.

When neither argument has been transformed, the value returned by the `loess.smooth` function can be passed directly to `lines`.

```
lines(loess.smooth(dev$experience, dev$written, span=0.5), col=pal_col[2])
```

A `loess` visualization can also helpful when the number of data points is so large they coalesce into formless blobs. The Ultimate Debian Database discussed later, see Figure **??**, is an example.

The default behavior of the `loess` implementation is to divide the range of x-axis values into fixed intervals. When the range of x-values various by many orders of magnitude the fitted curve can look over stretched at the low values and compressed at high values.

One solution is to reduce the range of x-values by, for instance, taking the log, smoothing and then expanding (see rexample[java-api-size.R]); the following code is used later:

```
t=loess.smooth(log(API$Size), API$APIs, span=0.3)
lines(exp(t$x), t$y, col=loess_col)
```

6.2.6 Influential observations and Outliers

Influential observations are observations that have a disproportionate impact on the values of the model coefficients, e.g., a single observation significantly changing the slope of the fitted straight line. The terms *leverage* or *hat-value* describes the amount of influence a data point has on a fitted model; the `hatvalues` function takes the model returned by `glm` and returns the leverage of each point.

Influential observations might be removed or modified, or regression techniques used that reduce the weight given to what are otherwise overly influential points (e.g., the `glmrob` function in the `robustbase` package).

Outliers are discussed as a general issue in the data cleaning chapter, this subsection discusses outliers in the context of regression modeling. In the context of regression modeling an outlier might be defined as a data point having a disproportionately large standardized residual (here Studentized residuals are used).

To repeat an important point made in the data cleaning chapter: excluding any influential observations or outliers from the analysis is an important decision that needs to be documented in the results.

Cook's distance (also known as *Cook's D*) combines leverage and outlierness into a single number that is a commonly used metric.

A study by Fenton, Neil, Marsh, Hearty, Radliński and Krause[178] involved data from 31 software systems for embedded consumer products. Figure 6.12 shows development effort against the number of lines of code, along with a fitted straight line and standard error bounds. At the right edge of the plot are two projects that consumed over 50,000 hours of effort and the number of lines of code for these projects looks very small in comparison with other projects. Is the fitted model overly influenced by these two projects and should they be ignored or adjusted in some way?

As the number of points in a sample grows there is an increasing probability that one or more of them will be some distance away from the fitted line; in any large sample a few apparent outliers are to be expected as a natural consequence of the distribution of the error. The following example illustrates the dangers of not taking sample size into account when making judgements about the outlier status of a measurement.

Figure 6.13 shows the results of building a model and removing measurements having both a high Cook's distance and Studentized residuals, and the repeating the process until points stop being removed. At the end of the process most of the measurements have been removed.

Removing overly influential points until everything looks respectable is seductive, it is an easy to follow process that does not require much thought about the story that the data might have to tell. For those who don't want to think about their data, the `outlierTest` function in the `car` package can be used to automate outlier detection and removal (it takes a model returned by `glm` and returns the Studentized residuals of points whose Bonferroni corrected p-value is below a cutoff threshold; default `cutoff=0.05`).

A method of visualizing the important influential observation and outlier information is required. The `influenceIndexPlot` function in the `car` package takes the model returned by `glm` and plots the Cook's distance, Studentized residual, Bonferroni corrected p-value and hat-value for each data-point; Figure 6.15 is for the Fenton et al data.

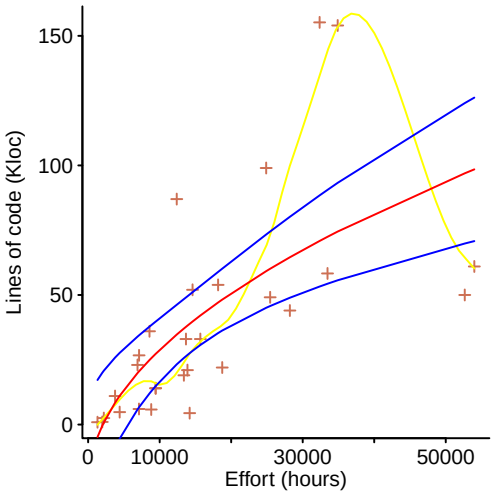


Figure 6.12: Hours to develop software for 29 embedded consumer products and the amount of code they contain, with fitted regression model and loess fit (yellow). Data from Fenton el al.[178] code

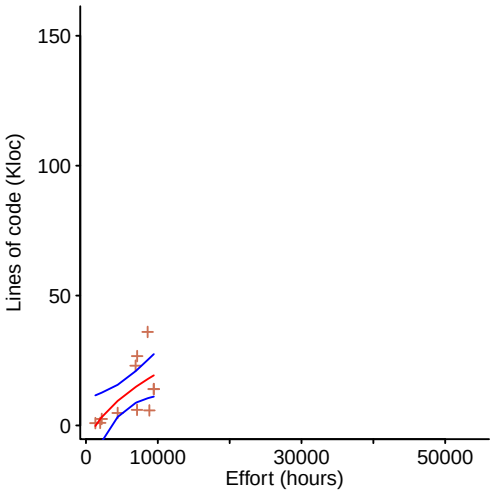


Figure 6.13: Points remaining after removal of overly influential observations, repeatedly applying Cook's distance and Studentized residuals. Data from Fenton el al.[178] code

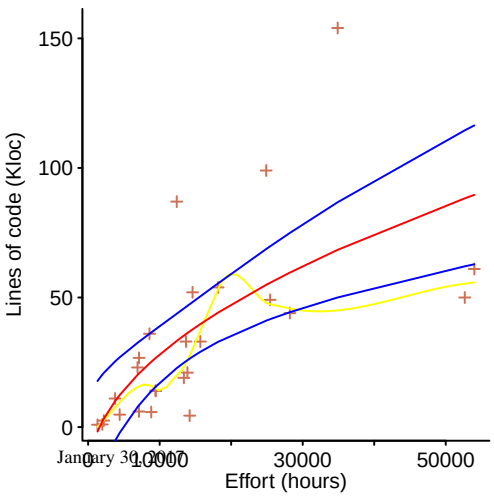


Figure 6.14: Points remaining after removal of overly in-

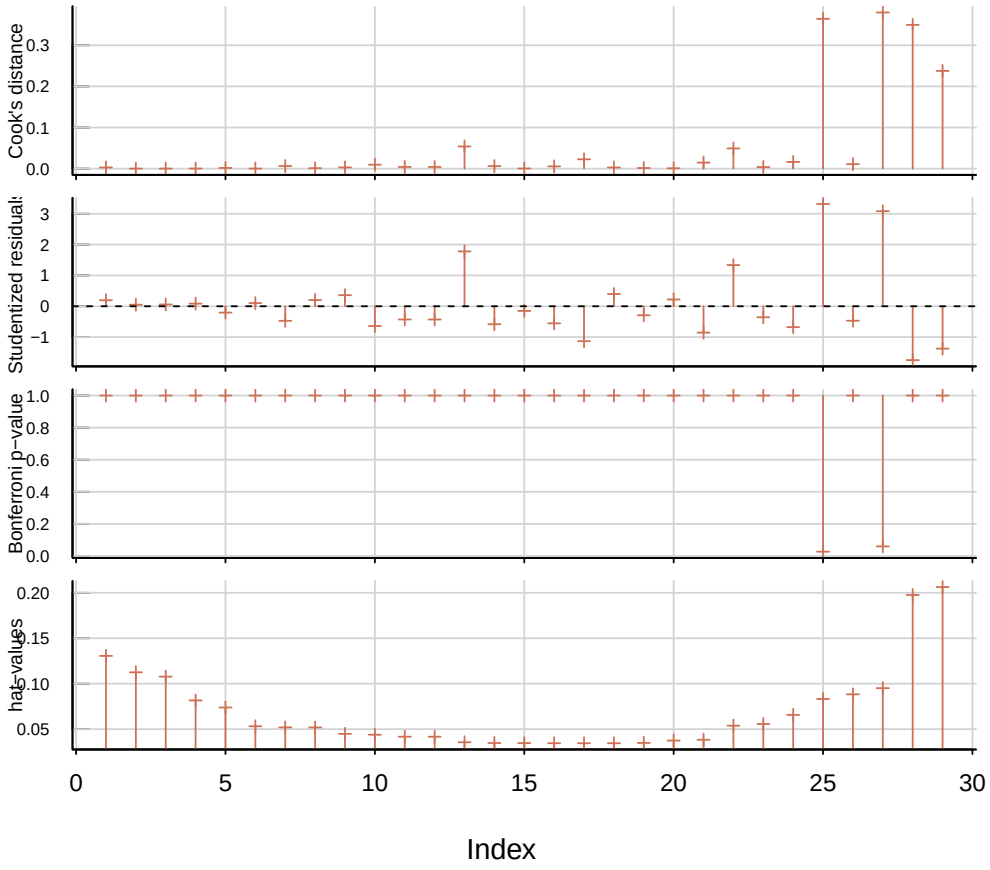


Figure 6.15: `influenceIndexPlot` for the model having the fitted line shown in Figure 6.12. Data from Fenton el al.[178] code

```
all_mod=glm(KLoC ~ I(Hours^0.5), data=loc_hour)
influenceIndexPlot(all_mod, main="", col=point_col, cex.axis=0.9, cex.lab=1.0)
```

The top plot shows four data points having a large Cook's distance, but only two of them have a significant corrected p-value (second plot up). These two data points were removed and the process of building a model and calling `influenceIndexPlot` is repeated; this time one point is removed and repeating the process shows that no other data points are worthwhile candidates for removal.

Figure 6.14 shows the results of removing data points having both a high Cook's distance and Studentized residuals whose corrected p-value is below the specified limit.

Outliers are loners, appearing randomly scattered about a plot. When multiple points appear to be following a different pattern than the rest of the data, the reason for this may be a new process driving behavior, or a change of behavior in what went before.

A study by Alemzadeh, Iyer, Kalbarczyk and Raman[8] analysed safety-critical computer failures in medical devices between 2006 and 2011 (as reported by the US Food and Drug Administration). Figure 6.16 shows the number of devices recalled for computer related problems (20-30% of all recalls), binned by two week intervals.

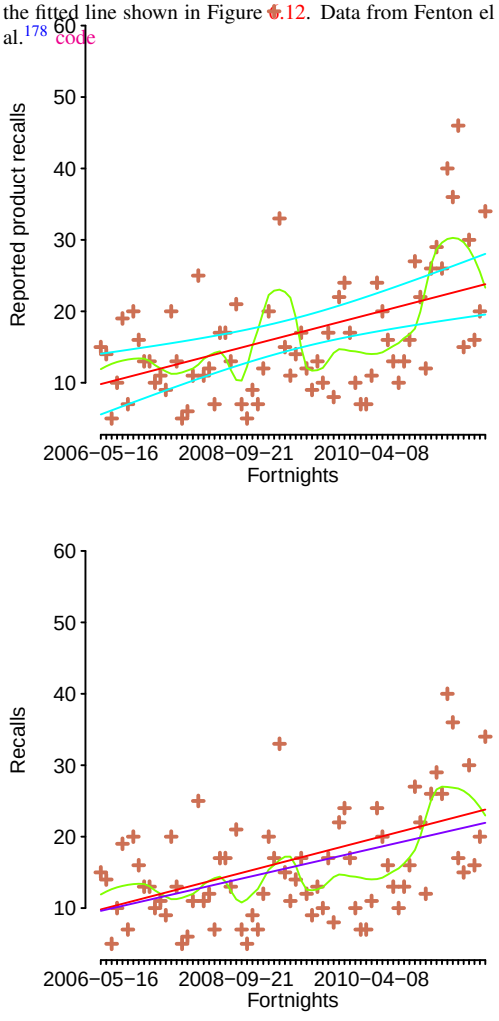


Figure 6.16: Number of medical devices reported recalled by the US Food and Drug Administration, in two week bins. Upper: fitted straight line and confidence bounds, with loess fit (green); Lower: straight line (purple) fitted after two outliers replaced by mean and original fit (red). Data from Alemzadeh et al.[8] code

Points that stand out in the upper plot of Figure 6.16 are the two large recall rates in the middle of the measurement interval and recall rates at later dates appearing to increase faster than earlier; plotting a loess fit (green) shows peaks around the two suspicious periods.

The fitted straight line shows a distinct upward trend. Is this fitted line being overly influenced by the middle period or end of period recall rates?

The measurements occur at regular intervals and deleting a measurement in one of these time slots does not make sense; replacing a value with the mean of all observations is one solution for handling this situation.

After replacing two outliers one of the peaks in the fitted loess line is reduced, but there is still a noticeable hump after 2010 (see lower plot). There is little change in the fitted regression model (red and purple lines), showing that the two outliers had little influence. Did a substantive change in the processes driving recalls, or recording of recalls, occur around the start of 2011? Domain knowledge is needed to answer this question.

In Figure 6.17, the lower plot shows two fitted models one using data up until the end of 2010 and the other using the data after 2010.

This is an example where blindly fitting a straight line to a complete sample produces a misleading model. A change occurred around the end of 2010 that had a significant impact on reported recalls (work is needed to uncover the reason for this change) and fitting data up to the end of 2010 shows a much smaller increase, and perhaps even no increase, in recall rates compared to when measurements after this date are included.

A change-point analysis of this data is discussed in the section on time series.

When combining results from multiple studies it is possible for an entire study to be an outlier, relative to other related studies.

A study by Amiri and Padmanabhuni[17] analysed the methods used by various other studies to convert between two common methods of counting function points.[xiii] Many of the studies included in the analysis have small sample sizes, include both student and commercial projects, and the function points are sometimes counted by academics rather than industrial developers.

Figure 6.18 shows function points counted using the COSMIC and FPA algorithms (counts made by students have been excluded). Both lines are loess fits, with red used for industry points and blue for academic researchers; the academic line overlays the industry line if one sample (i.e., Cuadtado_2007) is excluded.

The impact of influential observations on a fitted model can vary enormously, depending on the form on the equation being fitted. Figure 6.19 shows five equations fitted to the Embedded subset of the COCOMO 81[67] data, with the upper plot using the original data and the lower plot the data after three influential observations have been removed from the sample.

In some cases outlier removal has had little impact on the fitted model, while in other cases there has been a dramatic change in the coefficients of the fitted model.

Don't transform variables to reduce the effect of outliers... rexample[74-267-1-PB.R]

6.2.7 Diagnosing problems in a regression model

The commonly used regression modeling functions will build a model from almost any sample without reporting an error (some functions are so user-friendly they gracefully handle data that produces a singular matrix, an error that is traditionally flagged as it suggests that something somewhere is badly wrong). It is the users' responsibility to diagnose any problems in the model returned.

It is immediately obvious from Figure 6.20 that at least two of the fitted regression lines completely fail to capture the pattern present in the data. This is a famous data set, known as the Anscombe quartet,[24] whose four samples each contain two variables, with each sample having the same mean, standard deviation, Pearson correlation coefficient and are fitted by linear regression with a line having the same slope and intercept.

Problems with a regression model are not always as obvious as the Anscombe quartet case and diagnosing the cause of the problem can be difficult. As always, domain knowledge is very useful for suggesting possible changes to the model.

The difference between the measured value of the response variable and the value predicted by the fitted model is known as the *residual*. Many regression model diagnosis techniques involve the use of the residual. Some of these techniques require a lot more knowledge of the mathematics of regression modeling than is covered in this book.[xiv] Various visualization based techniques are discussed here.

The upper plot in Figure 6.21 shows the residual of the straight line fitted to the Linux kernel growth data analysed earlier. Ideally the residual is randomly scattered around zero and the V-shape seen in this plot is typical of a straight line fitted to values that curve around it (the smallest residual is in the center, where the model fits best, and is greatest at the edges; the smaller peak is a localised change of behavior and may explain why a cubic produces a slightly better fit). This plot shows one of the four diagnostic visualizations produced by `plot` when passed a regression model, as follows:

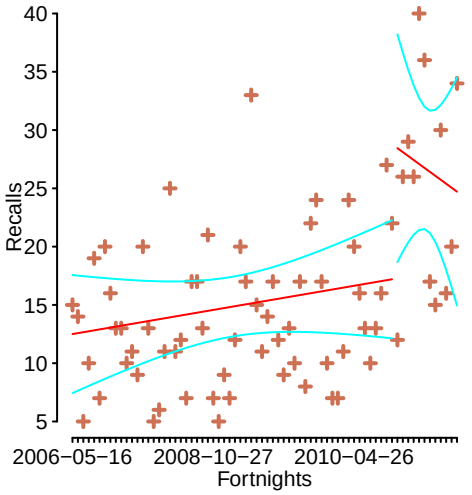


Figure 6.17: Two fitted straight lines and confidence intervals, one up to the end of 2010 and one after 2010. Data from Alemzadeh et al.[8] code

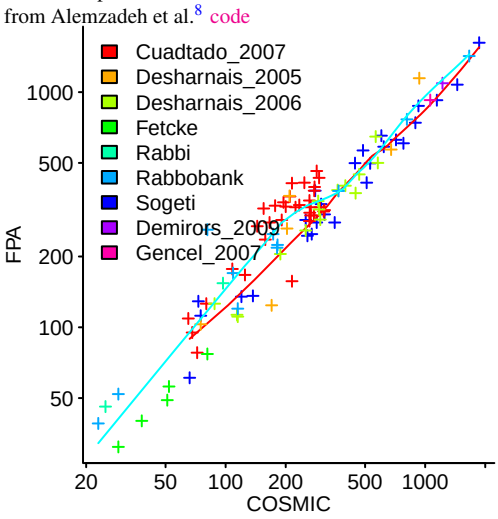
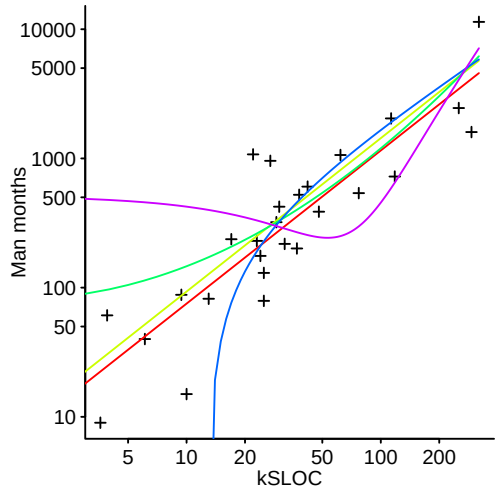
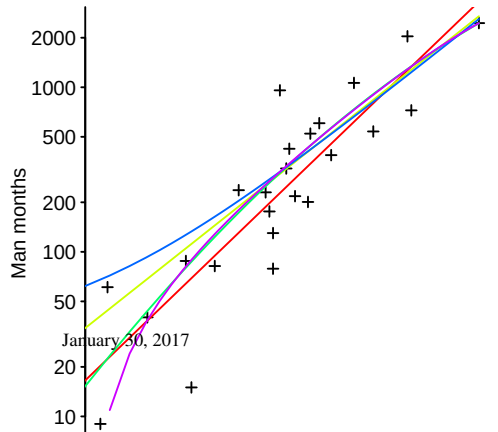


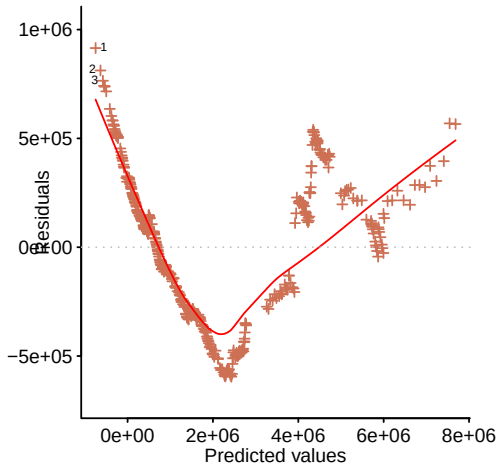
Figure 6.18: Results from various studies of software requirements function points counted using COSMIC and FPA; lines are loess fits to studies based on industry and academic counters. Data from Amiri et al.[17] code





[xiii] Function point counting is a technique for estimating development effort by counting the functionality contained in the software requirements specification.

[xiv] It is not obvious that the cost/benefit of learning the necessary mathematics is worthwhile (but it is a good source of homework exercises for students).



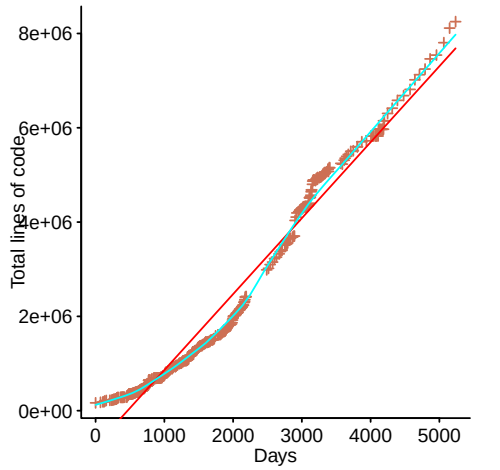
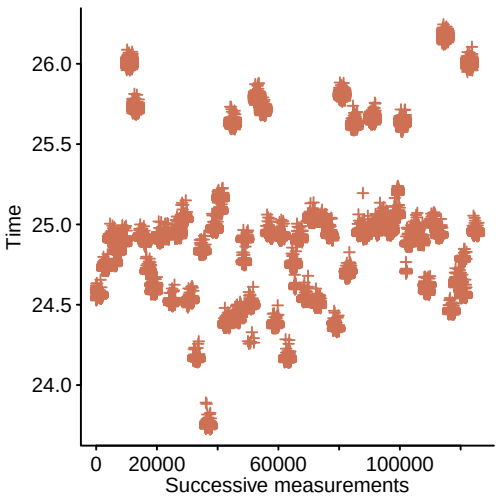
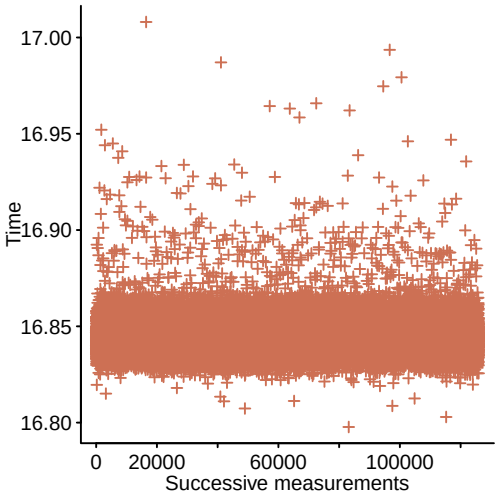


Figure 6.21: Residual of the straight line fit to the Linux growth data analysed in Figure 6.7 (upper) and data+straight line fit (red) and loess fit (blue). Data from Israeli et al.[289] code






```
m1=glm(LOC ~ Number_days, data=latest_version)
plot(m1, which=1, caption="", col=point_col)
```

The lower plot of Figure 6.21 shows the original data, straight line fit (red) and loess fit (blue). Both the residual plot and loess fit express the same pattern of curvature around about the straight line fit. Both visualizations have their advantage, the loess line can be drawn before any model is created and the details are easier to extract from a residual plot (e.g., values for the difference).

The mathematics behind linear regression requires that each measurement be independent of all the other measurements in a sample. A common form of dependence between measurements is serial correlation, i.e., correlation between successive measurements. The `durbinWatsonTest` function, in the `car` package, tests a fitted regression model for serial correlation.

A study by Flater and Guthrie[185] measured the time taken to assign a value to an array element in C and C++ using twelve different techniques, some of which checked that the assignment was within the defined bounds of the array (two array sizes were used, large and small); the programs benchmarked were compiled using seven different compiler optimization options.

Figure 6.22 shows the timings from 2,000 executions of one method of assigning to an array element, compiled using gcc with the 00 option (upper) and 03 option (lower). The results for 00 clearly show a clustering of execution times for groups of successive measurements.

A Durbin Watson test confirms that the 00 measurements are correlated (see rexample[array-durbanwatson.R]).

When a non-trivial correlation exists between successive measurements, the appropriate technique to use is time-series analysis; this is covered in a later section.

6.2.8 A model's goodness of fit

This section discusses the various ways in which the error in a regression model is measured and the uses of this value in model selection. The term *goodness of fit* is often used in this context.

When dealing with a single explanatory variable, it is possible to get a very good idea of how well a model fits the data through visualization, e.g., by plotting them both. Does the fitted line look correct and how wide are the confidence intervals? However, for data contains more than one explanatory variable accurate visualizations becomes problematic.

Meeting expectations of behavior is an important model characteristic when building to gain understanding. A model whose behavior goes against expectations is going to have to do a much better job of minimising the differences between the model and the measurement sample than one that is consistent with expectations. In this case error is measured in terms of the difference between expectation and the behavior of the fitted model.

When making predictions, the primary quantity of interest is the accuracy of new predictions, i.e., the amount of expected error in predictions for values that are not in the sample used to build the model. The error structure is also a consideration; is the priority to minimise total error, worse case error, to prefer over estimates to under estimates (or vice versa) or does some complicated weighting (over the range of values that explanatory variables might take) have to be taken into account?

To create a model by fitting it to data is to create a just so story. The predictions made by a model outside of the range of the data used to build it are only a reasonable subject for discussion when considering expectations of behavior derived from a theory of the processes involved in generating the data used to fit the model.

Confidence intervals, discussed earlier, provide information about the goodness of fit at every point. The following discussion looks at some of the ways of producing a single numeric value to represent goodness of fit.

The leftover variation in the sample that is not accounted for by the fitted model, the residual, is invariably a component in any equation calculating one or more numbers for the error present in a model. Some of the metrics that readers are likely to encounter include:

- deviance is reported by `glm`, in the `summary` output from a fitted model: `null deviance` is a measure of the difference between the data and the mean of the data, `deviance` is a measure of the difference between the data and the fitted model; a relative comparison shows how much better the model performs,
- R-squared (also known as the *coefficient of determination* and commonly written R^2) can be interpreted as the amount of the variance in the data (as measured by the residuals) that is explained by the model. It takes values between zero and one (which has the advantage of being scale invariant) and is a measure of correlation, not accuracy.

  Sometimes the adjusted R^2, $\bar{R}^2$, is used, which takes into account the number of explanatory variables, p, and sample size, n: $\bar{R}^2 = R^2 - (1-R^2)\frac{p-1}{n-p}$
- mean squared error (MSE): the mean squared error is the mean value of the square of the residuals and as such has no upper bound (and will be heavily influenced by outliers); root mean squared error (RMSE) is the square-root of MSE.

  The following equation shows how MSE and R^2 are related: $R^2 = 1 - \frac{MSE}{\sigma^2}$
- mean absolute error (MAE): the mean absolute error is the mean value of the absolute value of the residuals. This measure is more robust in the presence of outliers than MSE.

Apart from R-squared metric, the metrics listed (plus AIC) are scale dependent, e.g., mapping measurements from centimeters to inches changes their value; transforming the scale (e.g., taking logs) will also change values.

The choice of a metric is driven by what information is available and what model characteristics are considered important (e.g., how important is outlier handling). In a competitive situation people might not be willing to reveal details about their model and so any metric has to be made on predictive accuracy (e.g., models builds provide the predictions made by their model to a test data set).

R-squared is the only scale invariant metric and provides an indication of how much improvement might be possible over an existing model.

It is possible for the coefficients of a fitted model to be known with a high degree of accuracy and yet for this model to explain very little of the variance present in the data, and for there to appear to be little chance of improving on the model given the available data.

The Ultimate Debian Database project[578] collects information about packages included in the Debian Linux distribution from users who have opted-in to the Debian Popularity Contest. Figure 6.23 shows the age of a packaged application plotted against the number of systems on which that application is installed, for 14,565 applications in the "wheezy" version of Debian. Also, see Figure **??**.

The fitted linear model (red line hidden by the 95% confidence interval in green overwriting it; loess fit in blue) has a very low p-value, a consequence of the large number of, and uniform distribution of, data points. The predictive accuracy of this model is almost non-existent, the only information it contains is that older packages are a little more likely to be installed that younger ones.

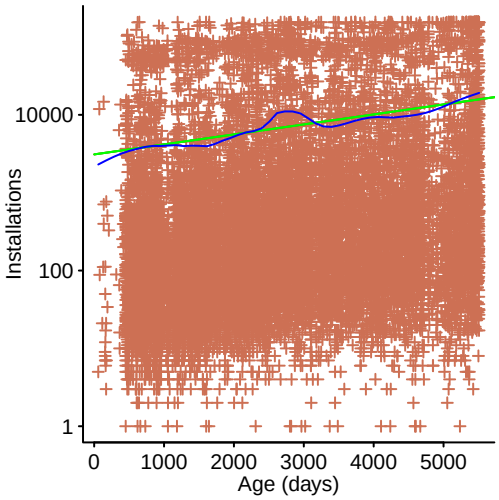


Figure 6.23: Number of installations of Debian packages against the age of the package, plus fitted model and loess fit. Data from the "wheezy" version of the Ultimate Debian Database project.[578] code

A study by Jørgensen and Sjøberg[314] investigated developers ability to predict whether any major unexpected problems would occur during a software maintenance task. Building a regression model using the available measured attributes showed that lines of code was the only explanatory variable having a p-value less than 0.05. However, only 3.3% of the variance in the response variable was explained by lines of code; so while the explanatory variable was statistically significant, its practical significance was negligible (see rexample[10.1.1.37.38.R]).

6.2.9 Low signal-to-noise ratio

Sample measurements sometimes contain a large amount of noise relative to the signal that is present, i.e., a low signal-to-noise ratio. Fitting a model to data having a low signal to noise ratio can be difficult because many different equations do an equally good job.

The top row of Figure 6.24 shows data generated from a quadratic equation containing noise and two fitted models. The following equation was used to generate the two sets of data:

$$y = x^2 + K \times (5 + rnorm(length(x)))$$

where: K was 10^3 (left column) or 10^2 (right column).

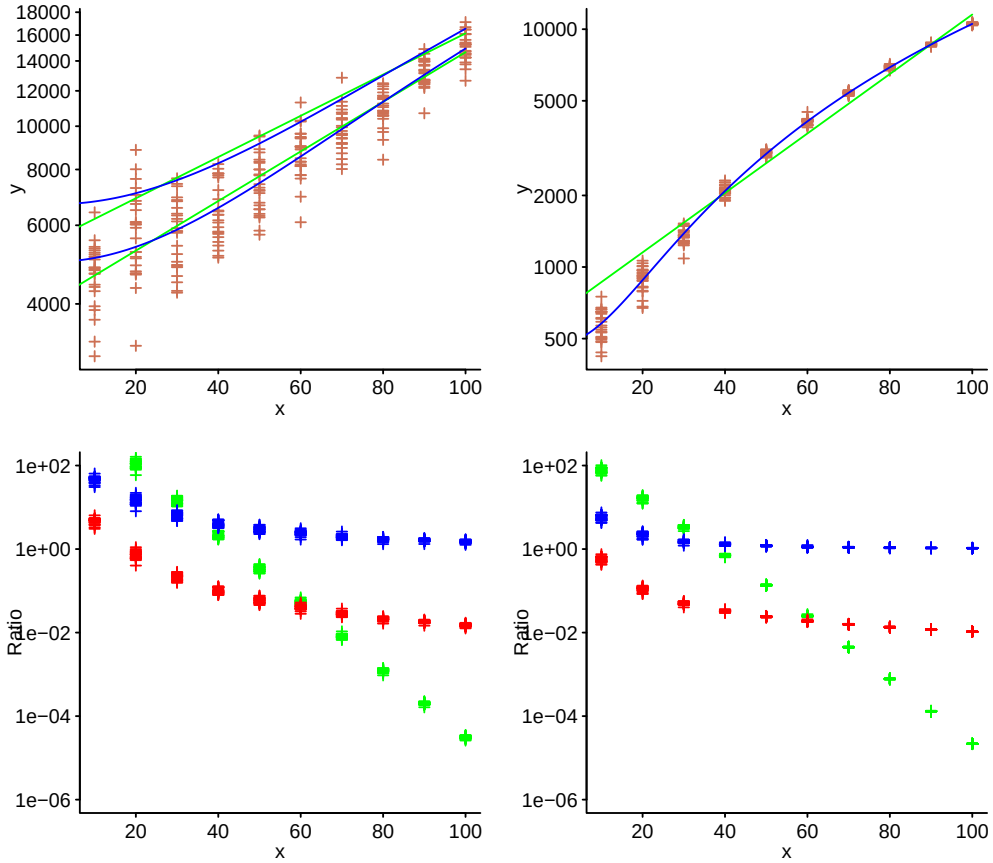


Figure 6.24: Quadratic relationship with various amounts of added noise fitted using a quadratic and exponential model. code

It is not possible to tell by looking at the top left plot whether a quadratic (blue) or an exponential (green) is a better fit; the output from `summary` is not much help (see rexample[noisey-data.R]). The top right plot contains less noise and it is easier to see that the exponential fit does not follow the data as well as the quadratic.

Sometimes the peaks (or troughs) can be a better indicator of the shape of the data. The top left plot shows a quadratic and exponential fit to the three largest values at each x-value. The fit does not seem to reduce the uncertainty in this case.

The *ratio test* is a technique that can help rule out some possible model equations. If $f(x)$ is the function being fitted to the data and this data was generated by the function $g(x)$, the ratio $\frac{g(x)}{f(x)}$ will converge to a constant as x becomes small/large enough such that the signal dominates the noise.

The bottom row in Figure 6.24 show ratio tests for quadratic (blue), cubic (red) and exponential (green) equations. The exponential equation shows no sign of converging to a constant, while a quadratic is closer to doing this than a cubic (which can be ruled out because it does a poor job of fitting the data).

The ratio test rules out an exponential equation being a good model for the sample data.

A study by Vasilescu, Serebrenik, Goeminne and Mens[587] investigated contributions to the Gnome ecosystem, from the point of view of workload (measured by counting the number of file touches, e.g., commits), breaking it down by projects, authors and number of activity types (e.g., coding, testing, documentation, etc).

The upper plot of Figure 6.25 shows, for individual authors, workload and the number of activity types they engaged in. There is a large amount of noise in the data. The lower plot of Figure 6.25 showing a ratio test, with an exponential equation clearly failing to level off, the linear equation slowly growing and the quadratic looking like it's trying to grow.

Perhaps the situation becomes clearer with more activity types, but none of the commonly occurring equations looks like they may be good fits.

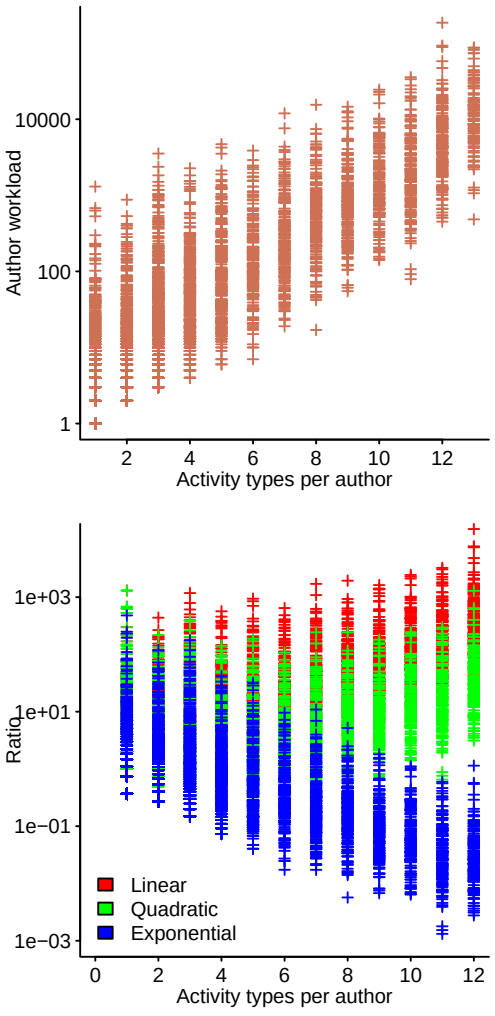


Figure 6.25: Author workload against number of activity types per author (upper) and ratio test (lower). Data from Vasilescu et al.[587] code

6.2.10 Weighting data

Ho hum, some data where weighting has a meaningful interpretation...

6.2.11 Sharp changes in a sequence of values

When the processes generating the measured values changes, the statistical properties of the post-change sequence of values may change. The point at which the statistical properties of a value sequence changes is known as a *change point*.

The `changepoint` package includes functions for performing change point detection of the mean, variance and both mean and variance of a sequence of values. The `cpt.mean` function checks for significant shifts in the mean value; the two options are `method="AMOC"` (At Most One Change; other values support searching for a specified maximum number of changes, with `method="AMOC"` selecting what it considers to be the optimum number of changes) and `test.stat="Normal"` (assume the measurement error has a Normal distribution).

An earlier analysis of electronic device recall frequency suggested that a significant shift in processes driving recalls occurred at the end of 2010. Figure 6.26 shows the output from the following calls to `cpt.man`:

```
change_at=cpt.mean(as.vector(t2))
plot(change_at, col=point_col,
        xlab="", ylab="Reported product recalls\n")

change_at=cpt.mean(as.vector(t2), method="PELT")
plot(change_at, col=point_col,
        xlab="Fortnights", ylab="Reported product recalls\n")
```

See rexample[hpc-read-write.R] for an example of detecting changes in variance and changes in both mean and variance.

The existence of a change point is not always obvious and when dealing with a few discrete measurement points change point analysis might not be able to provide a reliable answer. As always, a loess curve can provide useful information.

A study by Berger, She, Lotufo, Wąsowski and Czarnecki[54] analysed the variability models for 13 open source projects. Figure 6.27 shows, for systems containing a total number of possible features, the number of these features depending on 1, 2, 3, etc. build-flags (some optional features require other optional features to be enabled before they can be enabled).

The loess curve (in red) is flat and then suddenly jumps and appears to flatten off again. A fitted regression line (in green) appears to have been shifted up by the jump in values. Perhaps the processes driving optional features in the few larger systems that were measured are different from the smaller systems.

A sequence of values containing discontinuities can be fitted by multiple regression models, one each over each interval between discontinuities. However, it is possible to build a single model that contains the discontinuity information.

Figure 6.28 shows a distinct change in the pattern of the sales volume of 4-bit microprocessors (green). Straight lines have been fitted to the two periods before/after April 1998 (red), with the yearly sales cycle modeled with a single sine wave (blue).

The technique for building a model to handle discontinuous patterns of behavior makes use of an interaction between the explanatory variable, `date`, and a dummy variable whose 0/1 value depends on `date` relative to the discontinuity point. The code for the straight line model (in red) is:

```
# discontinuity point
y_1998=as.Date("01-04-1998", format="%d-%m-%Y")

p4=glm(bit.4 ~ date*(date < y_1998)+date*(date >= y_1998), data=proc_sales)
```

and the `summary` output is: code

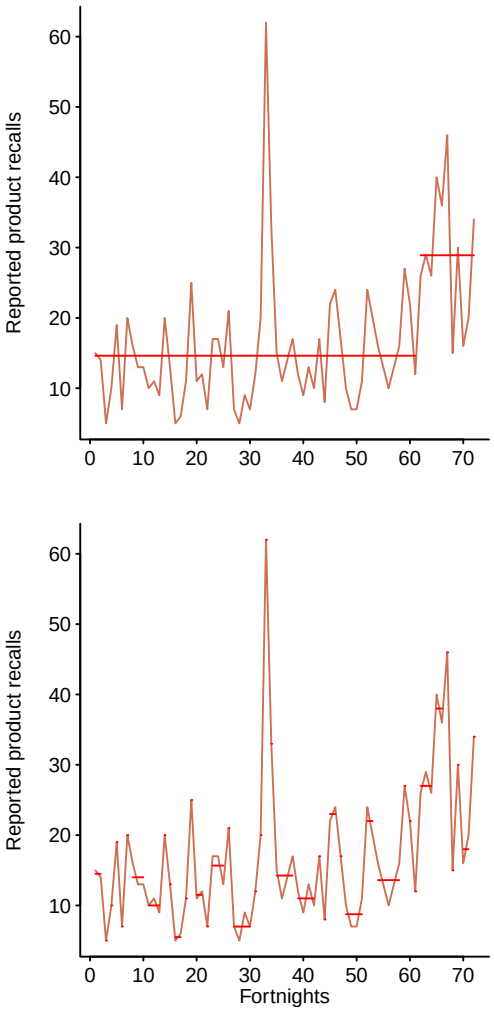


Figure 6.26: Change points detected by `cpt.mean`, upper using `method="AMOC"` and lower using `method="PELT"`. Data from Alemzadeh et al.[8] code

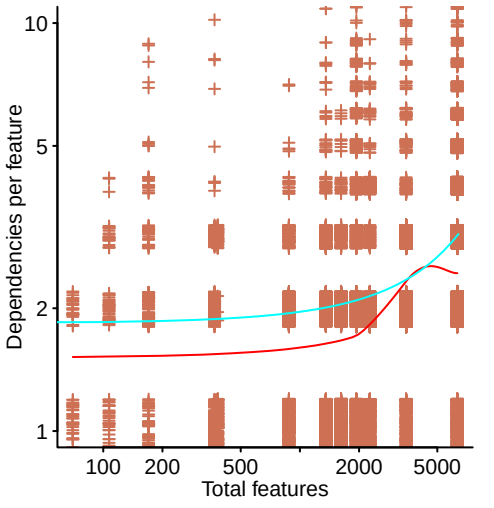


Figure 6.27: Number of flags (y-axis jittered) used to control the selection of optional features in system containing a total number of features, loess curve (red), regression line (green). Data from Berger et al.[54] code

```
Call:
glm(formula = bit.4 ~ date * (date < y_1998) + date * (date >=
    y_1998), data = proc_sales)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-19756.9   -6372.8    -558.7    6533.2   19086.4

Coefficients: (2 not defined because of singularities)
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)              7.050e+04  5.802e+04   1.215   0.2265
date                     7.072e-01  5.368e+00   0.132   0.8954
date < y_1998TRUE       -8.357e+04  5.873e+04  -1.423   0.1572
date >= y_1998TRUE              NA         NA      NA       NA
date:date < y_1998TRUE   1.045e+01  5.466e+00   1.912   0.0581 .
date:date >= y_1998TRUE         NA         NA      NA       NA
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 80003408)

    Null deviance: 2.0763e+10  on 131  degrees of freedom
Residual deviance: 1.0240e+10  on 128  degrees of freedom
AIC: 2782.6

Number of Fisher Scoring iterations: 2
```

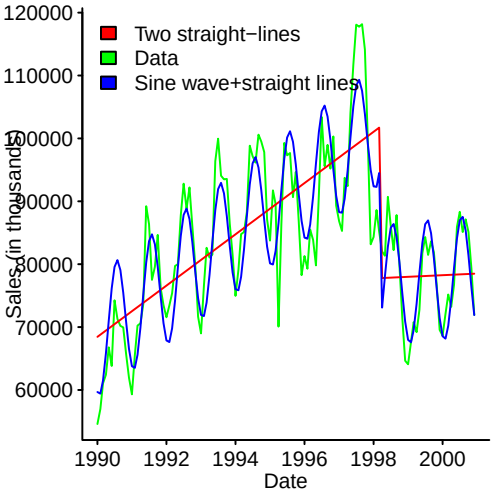


Figure 6.28: Monthly unit sales (in thousands) of 4-bit microprocessors. Data kindly supplied by Turley.[575] code

Sales follow a seasonal trend that can be approximated using a single 12-month frequency sine wave and adding this to the straight line model:

```
season_p4=glm(bit.4 ~ date*(date < y_1998)+date*(date >= y_1998)+
                                I(sin(rad_days))+I(cos(rad_days)),
                                                data=proc_sales)
```

6.3 Moving beyond the default Normal error

Some measurements in software engineering have properties that do not meet the requirements necessary for the mathematics behind `glm`s default argument values to work. These measurements often have one or more characteristics that require non-default argument values to be passed to `glm`, including the response variable having:

- values that span several orders of magnitude,
- values that can never go below zero, e.g., count data,
- values that can never go above some maximum value, e.g., a percentage can never be greater than one hundred.

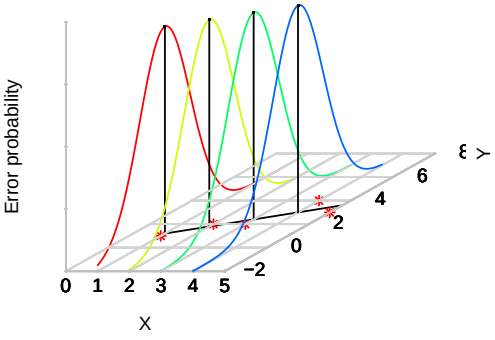


Figure 6.29: Fitted regression line to points (in red) and 3-D illustration of assumed Normal distribution of errors. code

By default, `glm` assumes that measurement error has a Normal distribution (also known as a _Gaussian distribution_[xv]). Figure 6.29 shows a fitted regression line with four of the data points (in red); the colored Normal curves over each point represents the probability distribution of the measurement error that is assumed to have occurred for that measurement (the center of each error curve is directly above the fitted line at each sample value of the explanatory variable).

In calls to `glm`, the `family` argument has the default value `family=gaussian(link="identity")` (which can be shortened in code to `family=gaussian` because `link="identity` is the default link function for `gaussian`).

The Normal distribution includes negative values and when a measurement cannot have a negative value, assuming an error distribution that allows negative values to occur can distort the fitted model. One possible alternative is the Poisson distribution which is zero for all negative values. The following call to `glm` specifies that the measurement error has a Poisson distribution:

[xv] This book uses the term Normal because it appears to be more widely used.

```
a_model=glm(a_count ~ x_measure, data=some_data, family=poisson)
```

The Poisson and Binomial distributions are the most common cases of non-default error distributions appearing in this book.

Calling `glm` using a non-default value for the `family` argument requires knowing more about the mathematics behind generalised regression model building. The equation actually being fitted by `glm` is:

$$l(y+\varepsilon) = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

which differs from the one given at the start of the chapter in having $l(y+\varepsilon)$ on the left-hand-side, rather than y. This l is known as the *link function* and for the Normal distribution is the identity function (which leaves its argument unmodified, which means the equation ends up looking like the equation given at the start of the chapter).

Once a regression model is fitted, the value of the response variable is calculated from:

$$y = l^{-1}(\alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots +) - \varepsilon$$

where: l^{-1} is the inverse of the link function used, e.g., the inverse of log is e raised to the appropriate power.

Every error distribution has what is known as its *canonical link* function, which is the function that pops out of the mathematical analysis for that distribution. By default, `glm` using the canonical link functions for each error distribution, and allows some alternatives to be specified. The default link function for the Poisson distribution is `log`.

When the link function is not `identity`, prediction values and confidence intervals need to be mapped as follows

```
a_pred=predict(a_model, se.fit=TRUE)
inv_link=family(a_model)$linkinv      # get the inverse link function

lines(x_values, inv_link(a_pred$fit)) # fitted line
# confidence interval above and below
lines(x_values, inv_link(a_pred$fit+1.96*a_pred$se.fit))
lines(x_values, inv_link(a_pred$fit-1.96*a_pred$se.fit))
```

The following sections analyse measurements having characteristics that may require the use of various error distributions and link functions.

6.3.1 Count data

Count data has two defining characteristics, it is discrete and has a lower bound of zero. The discrete distribution only taking on non-negative values, supported by `glm`, is the Poisson distribution.

In practice, when measurement values are sufficiently far away from zero (where far may be more than 10) there is very little difference between models fitted using the Normal and Poisson distributions. For measurements closer to zero the main difference between models fitted using different distributions is in the confidence intervals (which are usually not symmetric and may be larger/smaller).

The canonical link function for the Poisson distribution is `log` and the following two calls are equivalent:

```
p_mod=glm(y ~ x, data=sample, family=poisson)
p_mod=glm(y ~ x, data=sample, family=poisson(link="log"))
```

The `log` link function means that the equation being fitted is actually:

$$y = e^{\alpha+\beta x}$$

To fit the equation: $y = \alpha + \beta x$, using a Poisson error distribution the `identity` link function has to be used, as follows:

```
p_mod=glm(y ~ x, data=sample, family=poisson(link="identity"))
```

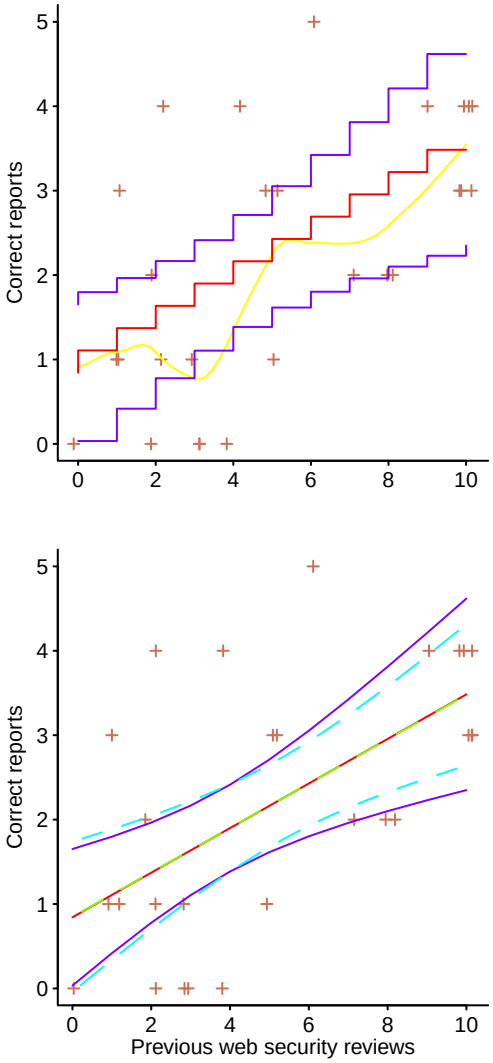


Figure 6.30: Number of vulnerabilities detected by professional developers with web security review experience; upper: technically correct plot of model fitted using a Poisson distribution, lower: easier to interpret curve representation of fitted regression models assume error has a Poisson distribution (continuous lines) or a Normal distribution (dashed lines). Data extracted from Edmundson.[156] code

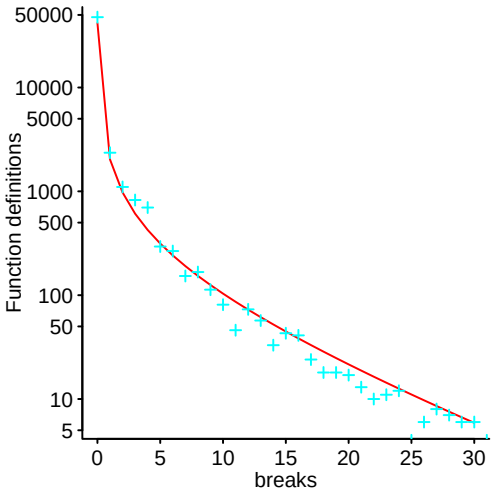


Figure 6.31: Number of functions containing a given number of `break` statements and a fitted Negative Binomial distribution. Data from Jones.[299] code

A study of the effectiveness of security code reviews by Edmundson, Holtkamp, Rivera, Finifter, Mettler and Wagner[156] asked subjects, professional developers with web security review experience, to locate vulnerabilities in web code. The number of vulnerabilities found can only ever be a non-negative integer value and in this study were single digit values.

The values fitted to a discrete distribution consists of a series of discrete steps, as the upper plot of Figure 6.30 shows (fitted line and 95% confidence intervals). While this plot is technically correct, it is ambiguous: are the values specified by the top left edge or the bottom right edge of the staircase?[xvi] Plots using continuous lines are easier for readers to interpret and these are used in this book.

The continuous lines in the lower plot of Figure 6.30 were fitted using the argument `family=poisson(link="identity")`, while the dashed lines were fitted using `glm` default argument values.[xvii]

The two fitted lines are virtually identical (the green dashed line is drawn over the continuous red line), the 95% confidence intervals are different. This pattern of behavior is very common unless the response variable has many values near zero.

When fitting models containing multiple explanatory variables (discussed later) and a response variable containing count data, it can be more difficult to see the difference between using a Poisson and Normal distribution for the errors. Using a Poisson distribution may be an inconvenience, but it removes uncertainty and is always worth trying.

Sometime a Poisson error distribution is used because a `log` link function is required to model an additive error.

The Negative Binomial distribution is probably the second most commonly occurring count distribution. A study by Jones[299] included counting the number of `break` statements in C functions. A `break` statement can occur zero or more times within a loop or `switch` statement, and these statements can occur zero or more times within a function definition. One process for generating a Negative Binomial distribution is to select values from multiple Poisson distributions, whose mean has a Gamma distribution. Figure 6.31 shows the number of functions containing a given number of `break` statements, along with a fitted Negative Binomial distribution.

The `gamlss` package supports a wide variety of probability distributions, including the `NBI` distribution used in the following call to the `gamlss` function:

```
breaks=rep(j_brk$occur, j_brk$breaks)
nbi_bmod=gamlss(breaks ~ 1, family=NBI)

plot(function(y) max(jumps$breaks, na.rm=TRUE)*     # Scale probability distribution
                     dNBI(y, mu=exp(coef(nbi_bmod, what="mu")),
                             sigma=exp(coef(nbi_bmod, what="sigma"))),
        from=0, to=30, n=30+1, log="y", col=pal_col[1],
        xlab="breaks", ylab="Function definitions\n")
points(jumps$occur, jumps$breaks, col=pal_col[2])
```

While zero is a very common lower bound, other values are sometimes encountered. When the lower bound is one, a zero truncated error distribution can be used. The `vglm` function in the `VGAM` package supports a wide variety of error distributions functions, including zero-truncated...

Calls to `vglm` follows the same pattern as calls to `glm`, except a wider variety of distribution families are supported (`pospoisson` is the zero-truncated Poisson distribution).

```
feat_mod=vglm(all ~ total, data=q, family=pospoisson())
```

The `quasipoisson` function...

Modeling zero inflated data... the `mhurdle` package... the `gamlss` package... Zero inflated poisson ... zipoisson()

[xvi] The choice is selectable via the `type` argument to `plot/lines`.
[xvii] To achieve an acceptable p-value, three outliers were removed.

6.3.2 Continuous response variable having a lower bound

Some continuous response variable measurements have a lower bound of zero, e.g., length or time measurements. The continuous distribution only taking on non-negative values, supported by `glm`, is the Gamma distribution.

In practice, when most measurement values are sufficiently far away from zero (where far away could be a large single digit value) there is very little difference between models fitted using the Normal and Gamma distributions. For measurements closer to zero the main difference between models fitted using different distributions is in the confidence intervals (which are usually not symmetric and may be larger/smaller).

The canonical link function for the Gamma distribution is `inverse` and the following two calls are equivalent:

```
G_mod=glm(y ~ x, data=sample, family=Gamma)  # Yes, capital G
G_mod=glm(y ~ x, data=sample, family=Gamma(link="inverse"))
```

The `inverse` link function means that the equation being fitted is actually (the `identity` link function is supported for this distribution):

$$y = \frac{1}{\alpha + \beta x}$$

Figure 6.32 comes from a code review study, discussed elsewhere, and shows meeting duration when reviewing various amounts of code. Meeting duration must be greater than zero and a Gamma error distribution is assumed to apply (the data has a linear relationship and the identity link function is used). The green line is the model fitted using a Gaussian error distribution.

The data contains a few points with high leverage and the loess fit suggests that there may be a change-point, so a more involved analysis is probably necessary.

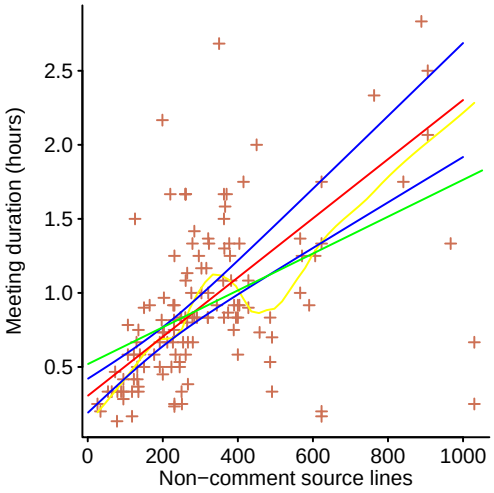


Figure 6.32: Code review meeting duration for a given number of non-comment lines of code. Fitted regression model, assuming errors have a Gamma distribution (red, with confidence interval in blue) or a Normal distribution (green). Data from Porter et al.[456] code

6.3.3 Transforming the response variable

When plotting sample points, values along one or both axes are sometimes transformed to compress or spread out the points, for the purpose of improving data visualization, e.g., using a log scale.

A regression model is fitted to a pattern (i.e., an equation) and if a visible pattern of behavior is present in a plot using transformed axis, it is worth investigating a model that uses similarly transformed values.

Applying a non-linear transform to the response variable produces a result that has a different error distribution to the original values. A regression model built using transformed response variable values may not be a natural fit to the processes that generated the measurements. Explanatory variables are assumed not to contain any error and transforming them does not change this assumption.

For example, in the following regression model the error, ε is additive:

$$y = \alpha + \beta x + \varepsilon$$

while fitting a log-transformed response variable:

$$\log y = \alpha + \beta x + \varepsilon$$

produces a model where the error is multiplicative (i.e., the error present is a percentage of the measured value):

$$y = e^{\alpha+\beta x} e^{\varepsilon}$$

The error in a model fitted using a log link function is additive, because the equation fitted is:

$$\log(y + \varepsilon) = \alpha + \beta x$$

which becomes:

$$y = e^{\alpha+\beta x} + \varepsilon$$

If the response variable is transformed, the decision on whether to transform it directly or via a link function is driven by whether the error is thought to be additive or multiplicative. As always, knowledge of the processes that produced the measured values is needed.

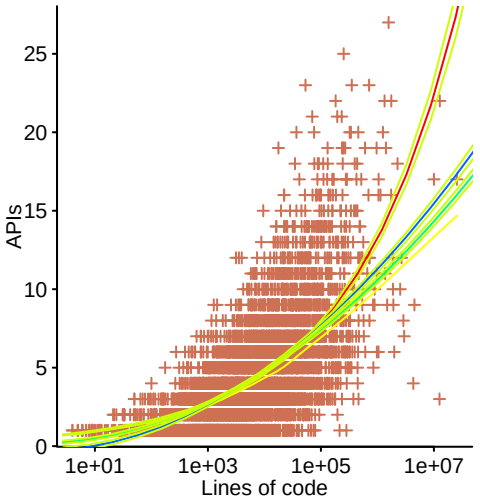


Figure 6.33: Number of APIs used in Java programs containing a given number of lines and three fitted models. Data from Starek.[543] code

A study by Starek[543] investigated API usage in Java programs. Figure 6.33 shows the number of APIs used in Java programs containing a given number of lines of code.

Many programs use a few APIs and the number of APIs is not large, suggesting that a Poisson error distribution may be applicable; the range in the number of APIs used does not really suggest a log scale.

The following three models were fitted (plot line color shown, with 95% confidence bounds sharing the same color and a yellow loess fit peaking through underneath):

```
ap_mod=glm(APIs ~ log(Size), data=API, family=poisson)  # Red

ap_id_mod=glm(APIs ~ I(log(Size)^2), data=API,
                                family=poisson(link="identity"),
                                start=c(1, 1))          # Green

ag_mod=glm(APIs ~ I(log(Size)^2), data=API)             # Blue
```

Which of these models is the best description the processes that produced the measurements? Perhaps none of them.

Choosing a model based on one of the goodness of fit metrics is one possibility. What is needed is a theory that explains the distribution of usage, along with measurements of other program characteristics that can be used to build a model that explains more of the variance in the data.

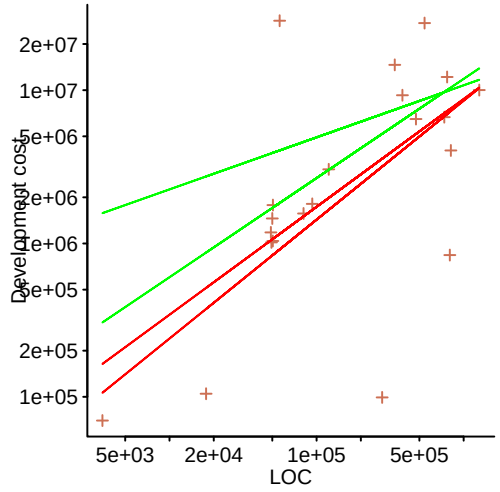


Figure 6.34: Yearly development cost and line of Fortran code delivered to the US Air Force between 1962 and 1984; with fitted regression models. Data extracted from NeSmith.[408] code

Except for when many sample values are close to zero, the Poisson distribution is a good enough approximation to the Gaussian distribution that it can be substituted when a log link function is required and the response variable take's integer values (see rexample[ICSE2010-cpp.R]). When the response variable contains many non-integer values (i.e., too many to ignore), then a log link function can be specified: `family=gaussian(link="log")` (see rexample[putnam-MTTF.R]).

One advantage of log transforming a response variable is that it reduces the influence of outlier values (because the range of values is compressed). Figure 6.34 illustrates the impact of removing one very influential value from the data used to fit a model using a log link function (green lines) and a model fitted to a log transformed response variable (red lines).

The visual appearance of outliers and influential observations plotted using log axis can be deceiving, i.e., they may not appear to be that far removed from the general trend. As always, assumptions based on visual appearance need to be checked using numerical methods.

The `robustbase` package includes the `glmrob` function for robust GLM model fitting, which is not always as robust as desired and manual help may be required (see rexample[a174454-reg.R]).

If the only available practical to-use regression modeling technique assumes the error in the response variable has a Normal distribution, then there is an incentive to transform the response variable so that it has this characteristic. When a computer, and the necessary software, is available to do the calculation, there is no need to transform the response variable to give it a normally distributed error.

The `boxcox` function in the `MASS` package and the `powerTransform` function in the `car` package suggest the transform that is most likely to produce a response variable whose error has a Normal distribution.

A traditional approach to simplifying a problem is to map a continuous variable to a number of discrete values (e.g., small/medium/large). Throwing away information may simplify a problem, but the cost is a considerable loss of statistical power and residual confounding.[493] Using a computer removes the need to simplify in order to reduce the manual effort needed to perform the analyse.

See rexample[melton-statics.R] for an example where building a regression model provides a lot more information about the characteristics of the continuous data compared to mapping values to large/small and running a chi-squared test.

Adjusting a model to handle uncertainty in explanatory variables, when the model contains a multiplicative error, requires specifying the measurement error for every value of an explanatory variable. The following option assigns a 10% error `measurement.error=maint$lins_up/10`.

A study by Jørgensen[307] investigated maintenance tasks and obtained developer effort and code change data. Figure 6.35 shows the effort (in days) and number of lines inserted and updated for 89 maintenance tasks. The original fitted regression line is in red and the SIMEX adjusted line is in blue. The following is the call to `simex`:

```
maint_mod=glm(EFFORT ~ lins_up, data=maint,
                        family=gaussian(link="log"), x=TRUE, y=TRUE)

y_err=simex(maint_mod, SIMEXvariable="lins_up",
                measurement.error=maint$lins_up/10, asymptotic=FALSE)
```

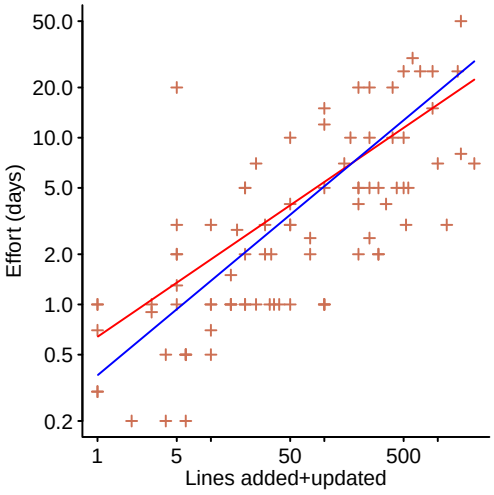


Figure 6.35: Maintenance task effort and lines of code added+updated, with fitted regression model (red) and SIMEX adjusted for 10% error (blue). Data from Jørgensen.[307] code

6.3.4 Binary response variable

When the response variable can only take one of two possible values, e.g., (false, true) or (0, 1), it has a binomial distribution. If the response variable, as the explanatory variable increases (or decreases), switches from 0 to 1 (or 1 to 0) and then always has that value for further increases in the explanatory variable, there is no need to build a regression model (simply find the switch point). When the response variable can have two possible values over some range of the explanatory variable, regression modeling will fit an equation that minimises the residual error.

The data in Figure 6.36 is continuous on the x-axis and has two discrete values on the y-axis. While it is possible to fit a regression model using a straight line, it is difficult to interpret this line as a model of the processes that generated the data. A Logistic equation starts low (or high) and eventually the high (or low) values dominate and it switches.

rexample[74-267-1-PB.R] from Höfer[272]...

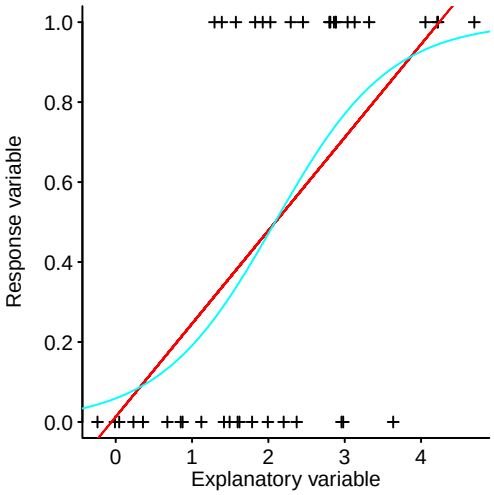


Figure 6.36: Regression modeling 0/1 data with a straight line and a logistic equation. code

The canonical link function for the Binomial distribution is `logit` and the following two calls are equivalent:

```
b_mod=glm(y ~ x, data=sample, family=binomial)
b_mod=glm(y ~ x, data=sample, family=binomial(link="logit"))
```

The equation for the `logit` link function is:

$$\log \frac{y}{1-y} = \alpha + \beta x$$

where the response has the form of log odds ratio.

The predicted values returned by `predict`, from a fitted binomial model, are in the range 0...1. The user has to make a decision about where to divide this continuous range, with predicted values on one side of the division treated as zero and all other values as one. A simple approach is to treat predictions greater than 0.5 as one and everything else as zero, while a more sophisticated approach looks at the distribution of predictions and makes an informed trade-off between true positive (in this context also known as *recall* and *hit rate*) and accuracy (i.e., false positive rate).

A ROC curve (receiver operating characteristics; named after a technique used to measure the performance of radio receivers) is a visualization technique for showing the trade-offs between two rates, e.g., true positive rate and false positive rate; it is a popular technique for displaying the trade-offs from predictions returned by machine learning models.

The `ROCR` package supports the creation and plotting of ROC curves.

The pair of columns in Table 6.1 is an example, from training data, of the impact of selecting particular cut-off values for distinguishing between true/false (for 10 data points). At a cut-off of 0.9 one correct prediction, a true positive, is made (20% of the available true responses), at a 0.81 cut-off another correct prediction occurs, while at 0.72 an incorrect prediction, a false positive, is made (at this cut-off the response rate for correct predictions is 40% and 20% for incorrect predictions).

t	**t**	**f**	**t**	**f**	**t**	**f**	**t**	**f**	**f**
0.90	0.81	0.72	0.60	0.53	0.44	0.39	0.28	0.16	0.09

Table 6.1: Example list of actual prediction outcome occurring at various cut-off values. code

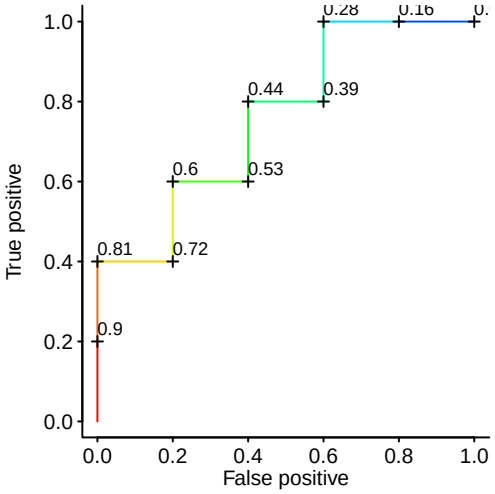


Figure 6.37: ROC curve for the data listed in Table 6.1. code

Figure 6.37 shows the ROC curve for this data...

AUC (Area Under the Curve) and select a cut-off value that provides the best trade-off between false-positive and....

Comparing the fitted model coefficients by plotting them using:

```
library("sjPlot")
...
sjp.glm(proc_mod)
```

In Figure **??**... REXAMPLE[fuzzer-mod.R] is a relatively complicated model... need something simpler...

6.3.5 Multinomial data

When discrete response variables take on more than two values, they have a *multinomial* distribution.

nominal : When a response variable can take N distinct values and π_i is the probability of the i^{th} value occurring then the baseline-category logit model, with one explanatory variable x, is (with the sum over all π equals one):

$$\log \frac{\pi_n}{\pi_N} = \alpha_n + \beta_i x$$

for $n = 1, \ldots, N-1$.

Building a model results in $N-1$ equations with separate coefficients for each.

The `mlogit` package supports the building of multinomial logit...

ordinal : fitting a logit model to each pair of adjacent values does not make use of all the information present and the logit equation can be extended to make use of the ordering information...

The model building process pairs each value of the multinomial response variable with a baseline value from that variable (where the baseline value is the last value *baseline-category logit...*)

The cumulative probability for Y is the probability that Y is at or falls below a given value:

$$P(Y \leq i) = \pi_1 + \pi_2 + \cdots + \pi_N$$

The *cumulative logits* treats the $P(Y \leq i)$ as the response variable in a logit model building process and creates a separate model for each value of i.

The `ordinal` package supports the building of *cumulative link models*, also known as *ordinal regression models*.

rexample[LuthigerJungwirth.R]...

Perhaps use CART when there are several categorical variables, early splits involving these variables may suggest that multiple models be created...

6.3.6 Rates and proportions response variables

When dealing with a response variable representing a rate or proportion there is a fixed lower and upper bound, often zero and one or 100. Measurements within a unit interval often share particular characteristics: they exhibit more variation around the mean and less variation towards the lower and upper bounds,[xviii] and they may have an asymmetrical distribution. These characteristics can be accommodated by the *Beta distribution* and a regression models where the response variable has a Beta distribution is known as a *Beta regression model*.

The `glm` function does not currently support a `family=beta` option; the `betareg` package contains function that support building Beta regression model. When fitting basic models calls to `betareg` have the same form as calls to `glm`; both functions include more sophisticated options that are not supported by the other.

[xviii] The measurement sample is heteroskedastic.

Figure 6.38 shows fitted curves from a beta regression model (red) and a call to `glm` (blue); the study from which the data is taken is discussed elsewhere.

The equation fitted is: $mutants_{killed} \propto \sqrt{coverage}$, and was chosen because it is something simple that works reasonably well. Searching for the best fitting exponent, using `nls` (the `betareg` package does not support fitting non-linear models), shows that 0.44 is a better fit than 0.5 for this sample.

Not a lot of difference in this case (or this one REXAMPLE[reuse_and_prod.R])... Need a better example...

The Beta regression model `summary` output includes extra information, as follows:

```
Call:
betareg(formula = y_measure ~ I(x_measure^0.5))

Standardized weighted residuals 2:
    Min      1Q  Median      3Q     Max
-2.6881 -0.6403 -0.1279  0.6399  3.1829

Coefficients (mean model with logit link):
                Estimate Std. Error z value Pr(>|z|)
(Intercept)      -2.5460     0.1891  -13.46   <2e-16 ***
I(x_measure^0.5)   4.7093     0.3502   13.45   <2e-16 ***

Phi coefficients (precision model with identity link):
      Estimate Std. Error z value Pr(>|z|)
(phi)   4.9641     0.5386   9.217   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Type of estimator: ML (maximum likelihood)
Log-likelihood: 80.37 on 3 Df
Pseudo R-squared: 0.6323
Number of iterations: 13 (BFGS) + 1 (Fisher scoring)
```

The ϕ variable is the second coefficient of the fitted Beta equation, $B(\mu, \phi)$.

Calls to `predict` return the expected value of the response variable, $E(y) = \mu$. The standard deviation in the expected value is given by the following equation (setting `se.fit` does not cause `predict` to return standard error information when passed a Beta regression model):

$$SD(y) = \sqrt{\frac{\mu(1-\mu)}{1+\phi}}$$

The default link function used by the `betareg` function is `logit`, the same default link function used by `glm` when passed the `family=binomial` argument.

$$\log \frac{p}{q} = \alpha + \beta x$$

p proportion of successes, q proportion of failures.

$$p = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$

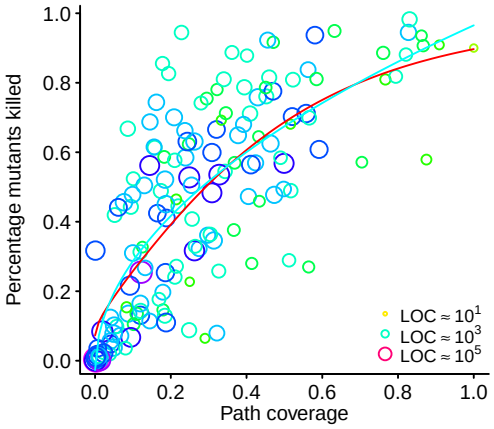


Figure 6.38: Percentage of mutants killed at various percentage of path coverage for 300 or so Java projects; fitted Beta (red) and `glm` (blue) regression models. Data from Gopinath et al.[216] code

6.3.7 Relational responses

List of relational comparisons... an experiment[300]

6.4 Multiple explanatory variables

Linear regression can be used to build models containing more than one single explanatory variable; *multiple regression* is the name given to modeling using more than one explanatory variable (the term *bivariate regression* is sometimes applied to the single explanatory variable+response variable case). In theory there is no limit on the number of explanatory

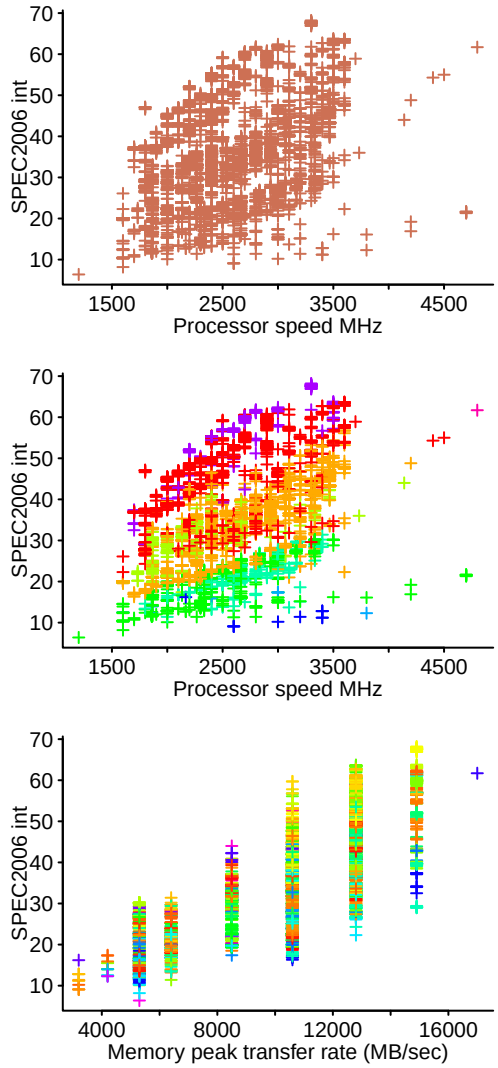
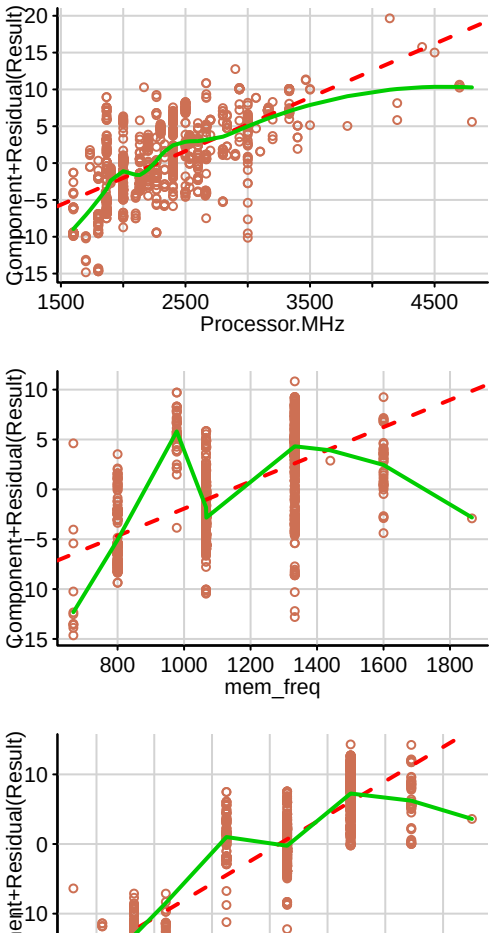


Figure 6.39: SPECint 2006 performance results for processors running at various clock rates, memory chip frequencies and processor family. Data from SPEC.[534] code



variables, but in practice available processing resources and by the need to hold data in storage set an upper bound.

Visualizing data is much more difficult when there are multiple explanatory variables. The communication stories chapter contains many examples for visualizing two variables and the general approach is to break down multiple regression visualization into pairs of variables.

System performance is affected by many factors and Figure 6.39 shows SPECint 2006 results for processors running at various frequencies (upper), color coded by memory chip frequency (center) and name of processor family (lower).

The SPECint results include 36 columns of information relating to the benchmarked system. Which of these columns contains information that can be used to succinctly model the performance of a system and what equation best describes the form of their contribution?

The R formula notation provides a means of specifying all columns in the data frame as explanatory variables, except the one specified as the response variable; the dot token is used as follows:

```
spec_mod=glm(Result ~ ., data=cint)
```

Given enough cpu power and memory, it can be more productive to start by considering all explanatory variables, removing underperforming variables, rather than starting with the explanatory variable believed to be the most important and then adding more variables.

The `stepAIC` function in the `MASS` package automates the process of removing underperforming explanatory variables from an existing model to create a model having a minimum AIC (the `step` function in the base system is a rather minimal implementation).[xix]

When some domain knowledge is available (e.g., performance usually correlates with clock rate and is not usually affected by date of execution), experimenting by building models containing those explanatory variables considered to be most likely to have a large impact on the response variable can help refine understanding of the impact of various subcomponent characteristics on overall performance.

For this SPEC dataset there is so much detail recorded in the `Processor` column of the Spec results, that each entry is often unique; making it possible to create an almost perfect, but completely uninformative, model using just this one explanatory variable.

The following model explains 80% of the variance in the `Result` values:

```
spec_mod=glm(Result ~ Processor.MHz+mem_rate+mem_freq, data=cint)
```

where: `Processor.MHz` is the processor clock rate, `mem_rate` the peak memory transfer rate and `mem_freq` the frequency at which memory is clocked.

The `+` binary operator, in the above formula, specifies that explanatory variables are added together. The `summary` output shows that the equation fitted by `glm` is:

$$Result = -2.4 \times 10^{1} + Processor.MHz 7.3 \times 10^{-3} + mem\_rate 2.5 \times 10^{-3} + mem\_freq 1.0 \times 10^{-2}$$

Is a linear combination of each explanatory variable the best model? With a single variable, it is easy to visually compare model predictions against measured values and a method of visualizing information for each explanatory variable in a fitted model is required.

The `crPlot` function, in the `car` package, produces a *component+residual* plot (also known as a *partial-residual* plot); the y-axis contains the predicted value plus the residual, the x-axis the value of the explanatory variable.

```
library("car")

spec_mod=glm(Result ~ Processor.MHz+mem_rate+mem_freq, data=cint)

crPlot(spec_mod, variable="Processor.MHz", col=point_col)
crPlot(spec_mod, variable="mem_freq", col=point_col)
crPlot(spec_mod, variable="mem_rate", col=point_col)
```

[xix] This fishing expedition approach to model building requires that p-values be suitably reduced, e.g., using a Bonferroni corrected value.

Figure 6.40 shows the component+residual plots produced using the code above (the red dotted line is derived from the fitted model and the green line a loess fit). If the form of an explanatory variable in the formula used to fit a model is close to reality, the two lines will be intertwined. For the SPEC model there is obvious curvature for two variables and perhaps some for a third.

Experience of computer behavior suggests that performance will not increase forever, as clock rates are increased. Adding quadratic forms of the explanatory variables to the model is an obvious modification to try in a linear model (an exponential is more realistic in that its value converges to a limit, but this form of modeling requires the use of non-linear regression, which is covered later).

Adding quadratic terms to the model shows that two of the three explanatory variables make worthwhile contributions; see Figure 6.41. The updated model explains another 4% of the variance; perhaps the techniques discussed below can produce further improvements...

Some systems contained error correcting memory, which might be expected to slightly reduce performance. An existing model can be updated to include, or remove, explanatory variables using the `update` function. The following code adds the variable `ecc` to the previously built model, `spec_mod`:

```
ecc_spec_mod=update(spec_mod, . ~ . + ecc)
```

The advantage of using `update` is a reduction in the system resources needed to build the model, compared with starting from the beginning again.

The `summary` output shows that systems using error correcting memory have slightly better performance. Before jumping to the conclusion that adding error correction improves system performance, it is worth noting that this kind of memory tends to be used in expensive systems where it is likely that money has been spent generally improving performance and reliability.

The cpu and memory frequency are decided by information that is not included in the SPEC results, the intended price point a computing system is designed to be sold at and the trade-off in the cost/performance of the components needed to build it....

How much does each explanatory variable contribute to a fitted model? Ways in which individual contributions can be measured include:

- the amount of variance, in the response variable, explained by an explanatory variable.

  The `calc.relimp` function in the `relaimpo` package calculates the contribution made by each explanatory variable to the variance explained by the fitted model,

- the impact each explanatory variable can have on the range of values taken by the response variable (with all other explanatory variables maintaining a fixed value).

  For very simple models, [xx] one way of calculating the maximum impact on the value of the response variable is by multiplying the minimum/maximum value taken by the explanatory variable by the corresponding coefficient in the fitted model. For instance, `range(cint$Processor.MHz)*7.3*10^-3` evaluates to `11.68 35.04`, a difference of `23.36`.

  The `visreg` function, in the `visreg` package, produces a visual representation of the impact of each explanatory variable on the response variable.

  Nomograms are a visual technique for calculating the value of a response variable when each explanatory variable has a particular value: the `DynNom` function in the `DynNom` package supports interactive exploration of model behavior in a web browser.

  Normalising values prior to building a model is sometimes suggested (using the `scale` function); the relative values of the model coefficients can then be directly compared. This technique only works when all explanatory variable values are drawn from a Normal distribution.

The `relaimpo` package supports a variety of functions[234] for calculating the relative contribution made to a model by each explanatory variable. For instance, the `calc.relimp` function can calculate: `first`, the variance explained by a model containing one variable, `last`, the variance explained when a variable is added to a model that already contains the other variables, `betasq`, the standardized coefficients of the model (i.e., one fitted after normalising the

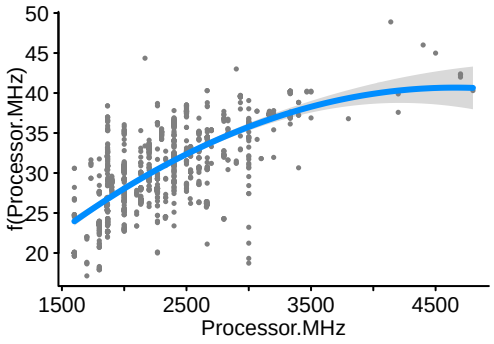
[xx] Those that are linear in the explanatory variable and no interactions between variables.

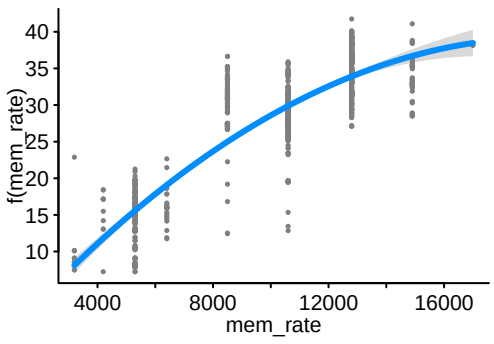
data; effectively a metric for the contribution of each explanatory variable to the value of the response variable), and `lmg` (named after the initials of its creators), the variance explained by each variable; the `boot.relimp` function returns confidence intervals for these values.

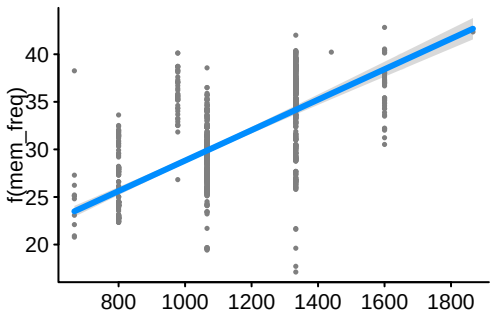
```
library("relaimpo")

spec_mod=glm(Result ~ Processor.MHz+I(Processor.MHz^2)+mem_rate
                          + I(mem_rate^2)+mem_freq, data=cint)

# How much does each explanatory variable contribute?
calc.relimp(spec_mod, type = c("first", "last", "betasq", "lmg"))
```

In the following output from `calc.relimp`, based on a model built using the SPECint data: code

```
Response variable: Result
Total response variance: 81.5614
Analysis based on 1346 observations

5 Regressors:
Processor.MHz I(Processor.MHz^2) mem_rate I(mem_rate^2) mem_freq
Proportion of variance explained by model: 83.77%
Metrics are not normalized (rela=FALSE).

Relative importance metrics:

                         lmg        last      first    betasq
Processor.MHz      0.06188975 0.017806784 0.04608568 0.6888167
I(Processor.MHz^2) 0.04555774 0.005697759 0.02962054 0.2201344
mem_rate           0.29379918 0.028909460 0.55553992 2.0853323
I(mem_rate^2)      0.29050403 0.006362584 0.58253084 0.4768264
mem_freq           0.14598416 0.067530522 0.28997162 0.1258352

Average coefficients for different model sizes:

                           1X           2Xs           3Xs           4Xs
Processor.MHz      4.308098e-03  1.289747e-02  1.567592e-02  1.823108e-02
I(Processor.MHz^2) 6.559513e-07 -4.893878e-07 -9.880091e-07 -1.727687e-06
mem_rate           2.343231e-03  1.561771e-03  2.235992e-03  3.303496e-03
I(mem_rate^2)      1.280338e-07  1.488238e-07  8.108289e-08 -1.285720e-08
mem_freq           2.429426e-02  2.012476e-02  1.557607e-02  1.456773e-02
                           5Xs
Processor.MHz       1.665538e-02
I(Processor.MHz^2) -1.788212e-06
mem_rate            4.539887e-03
I(mem_rate^2)      -1.158365e-07
mem_freq            1.600394e-02
```







the second set of columns, under the line starting `Average coefficients`, lists the model coefficients for each explanatory variable, if that variable were to appear in a model containing X variables (values are averaged over all combinations of other variables). The values in the last column (5Xs in this case) are the same as those produced by `summary`.

How do changes in the value of each explanatory variable effect the value of the response variable (when the other variables remaining constant)? Figure 6.41 shows the individual contribution made by each explanatory variable to the value of the response variable (along with confidence intervals), when the other variables are held constant, for the following model of SPECint performance:

```
library("visreg")

spec_mod=glm(Result ~ Processor.MHz + I(Processor.MHz^2)+mem_freq
                              +mem_rate+I(mem_rate^2), data=cint)

visreg(spec_mod)
```

Sometimes including an explanatory that has no correlation with the response variable improves the performance of a model; why does this happen? An explanatory variable may

correlate with the residual of a model, so adding this new variable has the effect of improving a model by reducing its residual.

6.4.1 Interaction between variables

In the models built so far, each explanatory variable has been independent of the others. The `glm` function and many other regression modeling functions provide mechanisms for specifying interactions between explanatory variables, using binary operators in the formula, such as `:`, `*` and `^`.

Operator	Effect
+	causes both of its operands to be included in the equation.
:	denotes an interaction between its operands, e.g., `a:b` or `a:b:c`.
*****	denotes all possible combinations of + and `:` operators, e.g., `a*b` is equivalent to `a+b+a:b`.
^	denotes all interactions to a specific degree, e.g., `(a+b+c)^2` is equivalent to `a+b+c+a:b+a:c+b:c`.
.	denotes all variables in the data-frame specified in the `data` argument except the response variable.
-	specifies that the right operand is removed from the equation, e.g., `a*b-a` is equivalent to `b+a:b`.
-1	specifies that an intercept is not to be fitted (many regression fitting functions implicitly include an intercept).
I()	"as-is", any operators in the enclosed expression are not treated as formula operators, their behavior that which occurs outside of a formula.

Table 6.2: Symbols that can be used within a formula to express relationships between explanatory variables.

As with all data analysis, the choice of interactions between explanatory variables should be driven by an understanding of the problem domain. When there is a great deal of uncertainty about which interactions are significant, it is often easiest to start by specifying all pairs of interactions between variables (or triple interactions if there are not too many variables) and then to simplify, either automatically using `stepAIC` or through manual inspection of `summary` output of the fitted models.

Stepwise regression techniques, such as that provided by `stepAIC`, can return models that suffer from a variety of problems, such as overfitting. There are techniques available to help avoid these problems; the `train` function in the `caret` package supports some of these techniques.

The `glmulti` package automates the process of finding an optimal, in a sense specified by the user (e.g., minimise AIC or some other measure), explanatory variable interaction; a list of variables is specified and the function permutes through the possibilities, e.g., `glmulti("y", c("a", "b", "c", "d"), data=some_data)`.

A study by Moløkken-Østvold and Furulund[398] investigated the impact of daily communication between customer and contractor on the accuracy of effort estimates, for 18 software projects. Figure 6.42 shows estimated vs. actual effort broken down by communication frequency (i.e., daily or not daily), along with individually fitted straight lines.

It is possible to build one regression model that simultaneously fits both straight lines to this data; the following code shows one possibility:

```
sim_mod=glm(Actual ~ Estimated+Estimated:Communication, data=sim)
```

The fitted equation in this case is:

$$\begin{aligned} Actual &= -270.1 + 1.18 Estimated + 0.51 Estimated \times D \\ &= -270.1 + (1.18 + 0.51 D) Estimated \end{aligned}$$

where: *Actual* is the actual and *Estimated* the estimated effort, and D has one of two values:

$$D = \begin{cases} 1 & \text{daily communication} \\ 0 & \text{not daily communication} \end{cases}$$

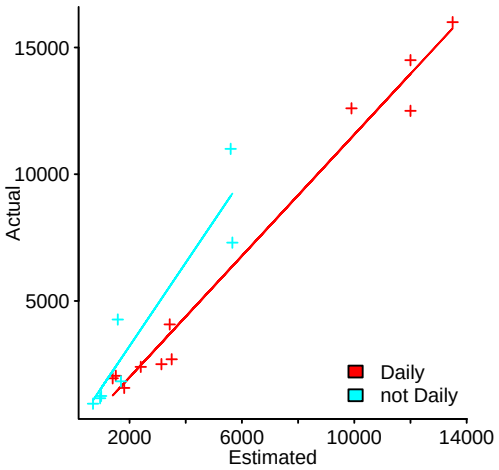


Figure 6.42: Estimated and actual effort broken down by communication frequency, along with individually fitted straight lines. Data from Moløkken-Østvold et al.[398] code

Is this formula the best fit possible using the available data? The formula used in this model was selected by your author because of a belief that the benefit of communication will increase as project size increases.

There are six data points for each of the 18 projects, computationally small enough for the brute force approach of examining all possible models; but with only 18 projects, some formula possibilities cannot be fitted because they contain more variables than available data points (a unique solution requires fewer variables than data points).

The formula in the following code fits four explanatory variables individually, plus each variable paired with every other variable (one at a time). `stepAIC` is used as a quick way of removing explanatory variables that are not paying their way (automatic model selection is fraught with problems, with perhaps the largest being that it causes users to stop thinking):

```
sim_mod=glm(Actual ~ (Estimated+Communication+Contract+Complexity)^2, data=sim)
min_sim=stepAIC(sim_mod)
summary(min_sim)
```

This book primary aim is to build models as a means of obtaining understanding, and minimising AIC is often a useful step along the way. Another possible way of removing low impact variables from a model is to consider the p-value of each fitted component.

The `summary` output for ordinal and nominal explanatory variables lists p-values for each value that these variables take in the data. The `Anova` function (in the `car` package) lists p-values at the variable level and its output for the above model is: code

```
Analysis of Deviance Table (Type II tests)

Response: Actual
                         LR Chisq Df Pr(>Chisq)
Estimated                  45.879  1  1.258e-11 ***
Communication              17.272  1  3.240e-05 ***
Contract                    6.767  3    0.07971 .
Complexity                  1.543  1    0.21423
Estimated:Communication     2.546  1    0.11060
Communication:Contract      5.020  3    0.17034
Contract:Complexity         3.197  2    0.20224
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The variable `Complexity` has the highest p-value and repeatedly removing the component having the highest p-value, for successively smaller models leads to the following model:

```
sim_mod=glm(Actual ~ Estimated+Communication+Communication:Contract,
                                                        data=sim)
```

which fits the equation:

$$\begin{aligned} Actual = -274.8 + 1.21Estimated + 2625\times !D + \\ C_{fp}(1862\times !D - 197.6\times D) + \\ C_{tp}(-2270\times !D - 462.2\times D) + \\ C_{ot}(-2298\times !D - 234.3\times D) \end{aligned}$$

where the new variables are: C_{fp} is a fixed price contract, C_{tp} is a target price contract and C_{ot} other kind of contract.

$$C_{fp} = \begin{cases} 1 & \text{fixed price contract} \\ 0 & \text{not fixed price contract} \end{cases} \quad C_{tp} = \begin{cases} 1 & \text{target price contract} \\ 0 & \text{not target price contract} \end{cases} \quad C_{ot} = \begin{cases} 1 & \text{other contract} \\ 0 & \text{not other contra} \end{cases}$$

This model explains more of the variance in the data than the first model built, it also has a slightly smaller AIC. While it makes use of extra information (i.e., the kind of contract), a more noticeable difference is that `Communication` has a constant effect (i.e., it does not increase with estimated size); the case of fixed price contracts with no daily communication cries out for attention.

Following the numbers has produced a model which better fits the data, but does not fit expectation (which may, of course, be wrong).

6.4.2 Correlated explanatory variables

The mathematics behind many of the approaches used to build linear regression models assumes that explanatory variables are independent of each other. If a linear relationship exists between one or more pairs of explanatory variables (i.e., a relationship of the form: $PV_1 = a + b \times PV_2$, where PV_1 and PV_2 are explanatory variables, and a is any constant and b is a non-zero constant), then this needs to be taken into account by the model building technique used.[xxi]

Multicolinearity is said to occur when a linear relationship exists between two or more explanatory variables, the term *colinearity* is often used when only two variables are involved.

Figure 6.43 illustrates how the variance in Y explained by combining X_1 and X_2 may be less than the sum of the variance explained by each individually, because the two variables are not independent; there is a shared contribution.

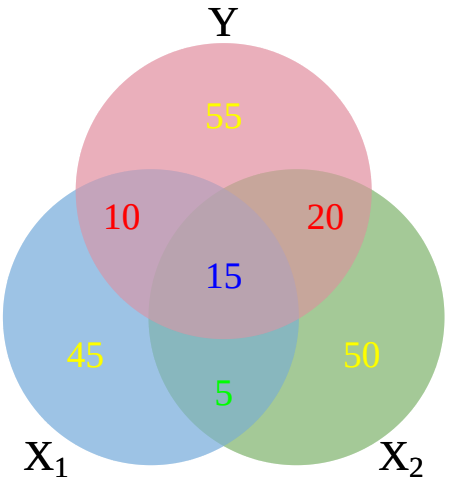


Figure 6.43: Illustration of the shared and non-shared contributions made by two explanatory variables to the response variable Y. code

The impact of multicolinearity is to increase the standard error in the calculated value of the coefficients of the fitted model (i.e., the β_n), potentially resulting in a model that is not considered acceptable or being unreliable (in the sense that small changes in the data result in large changes in the coefficients of the fitted model). The increased uncertainty in some variables will make it more difficult to isolate the effects of individual explanatory variables and will increase the width of the confidence intervals for the predicted values of the response variable.

The *Variance Inflation Factor* (VIF) is a measure of the uncertainty created by the presence of multicolinearity. The impact of VIF is the same as reducing the sample size (when no multicolinearity is present, VIF has a value of one):

$$\varepsilon_{standard} \propto \sqrt{\frac{VIF}{observations}}$$

When is a VIF value too large? A large VIF is more likely to be acceptable when there are many observations, compared to when there are few, e.g., the standard error is proportionally the same for 10,000 observations having a VIF of 400 and for 100 observations having a VIF of 4.

Suggested maximum VIF values do appear in print, e.g.,5 or 10 are sometimes suggested. As always, think about what the VIF value means in the context of how the results will be used; pick a value that makes sense given the sample size, the error in the measurements and the level of error that is acceptable in the business context.

The `car` and `rms` packages support a `vif` function, taking the model returned by a call to, for instance, `glm` and returning the VIF for each explanatory variable.

A study by Kroah-Hartman[230] investigated the amount of change in the Linux kernel source code occurring between each release. Figure 6.44 shows the number of lines added, modified and removed, plus overall growth, number of files and total number of lines at each initial release of the Linux kernel from version 2.6.0 to 3.9 (two outliers have been excluded).

[xxi] It is ok for a nonlinear relationship to exist e.g., $PV_1 = a + b \times PV_2^2$.

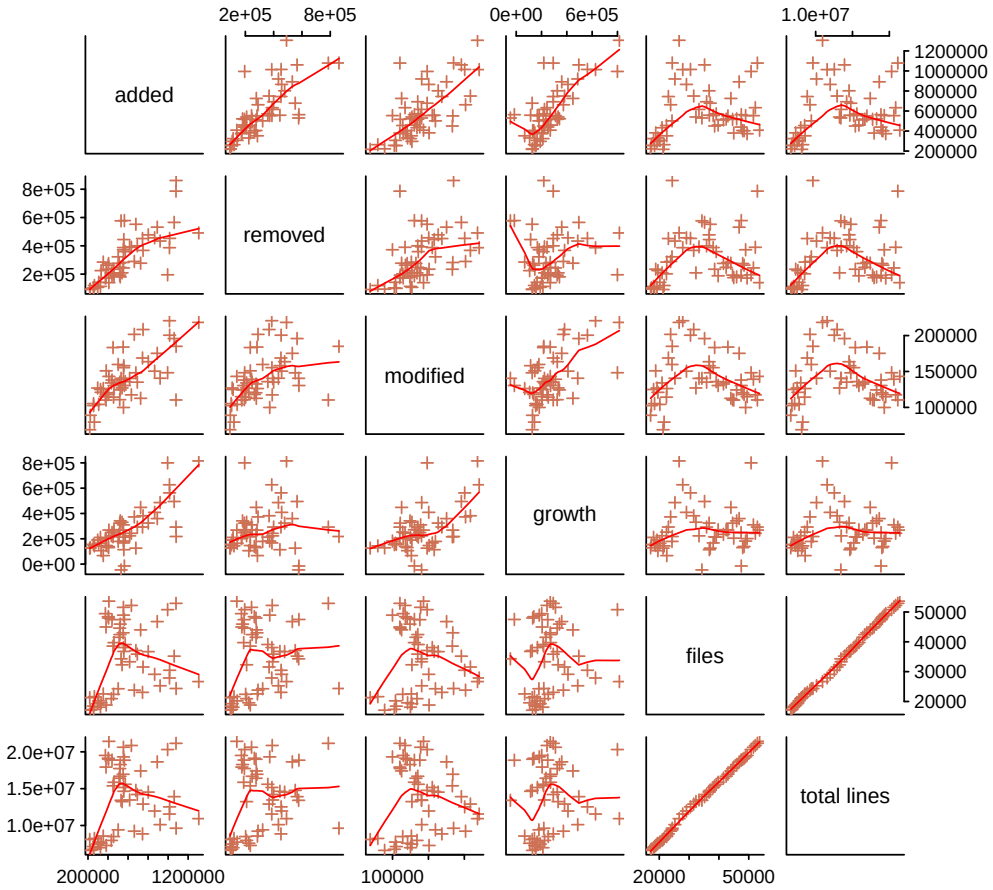


Figure 6.44: `pairs` plot of lines added/modified/removed, growth and number of files and total lines in versions 2.6.0 through 3.9 of the Linux kernel. Data from Kroah-Hartman.[230] code

Building a model of the growth of the Linux kernel is complicated by the potentially large amount of correlation between some of the measured variables, including:

- the growth in, lines of code, between releases is the difference between lines added and lines removed; these three variables are perfectly correlated in that knowing two of them enables the third to be calculated,
- lines added appears strongly correlated with lines removed. Perhaps existing functionality is being rewritten, rather than being completely new,
- the decision about whether a line has been modified or removed/added is made algorithmically (rather than asking the developer who made the change). The amount of misclassified lines is not known,
- system level measurements are also correlated, e.g., number of files and total lines of code.

Modeling the number of modified lines, using the Kroah-Hartman data, finds that both lines added and lines removed individually explain around half of the deviance (61% and 41% respectively). However, combining them both appear in a model does not produce any improvement; the following output from `summary` was obtained by including the argument `correlation=TRUE`. code

```
Call:
glm(formula = lines.modified ~ lines.added + lines.removed, data = amr_out)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-72376  -12049     321   11274   54964

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)    8.705e+04  8.625e+03  10.093 9.40e-14 ***
lines.added    9.958e-02  2.093e-02   4.759 1.64e-05 ***
lines.removed -1.500e-02  3.117e-02  -0.481    0.632
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
(Dispersion parameter for gaussian family taken to be 639477748)

    Null deviance: 6.2276e+10  on 53  degrees of freedom
Residual deviance: 3.2613e+10  on 51  degrees of freedom
AIC: 1253.1

Number of Fisher Scoring iterations: 2

Correlation of Coefficients:
              (Intercept) lines.added
lines.added   -0.54
lines.removed -0.06       -0.76
```

The correlation between the model coefficients appears at the end of the output and shows a high correlation between `lines.added` and `lines.removed`; `lines.added` is a better predictor of `lines.modified` and has been selected over `lines.removed` (whose p-value is significantly larger than when this variable appeared on its own in a model).

A call to the `vif` produces the following: code

```
  lines.added lines.removed
      2.39311       2.39311
```

With only two explanatory variables there is no ambiguity about which variables are involved in a linear relationship, but when more than two variables are involved things are not always so obvious. The correlation output from `summary` can be used to identify related variables; the `alias` function generates just this information when the argument `partial=TRUE` is specified.

Approaches to dealing with multicolinearity having an undesirable impact on the fitted model include:

- removing one or more of the correlated explanatory variables. The choice of which explanatory variables to remove might be driven by:
  - the cost of collecting information on a variable,
  - a VIF driven approach. The process builds a model using the current set of explanatory variables, removes the explanatory variable with the largest VIF (removing one variable effects the VIF of those that remain and may reduce the VIF of other variables to an acceptable level) and repeats until all explanatory variables have what is considered to be an acceptable VIF,
- combining the strongly correlated variables in a way that makes use of all the information they contain.

The disadvantages of excluding explanatory variables from a model include:

- ignoring potentially useful information present in the excluded variable,
- the resulting model may give a false impression about which explanatory variables are important, i.e., readers will assume that the variables appearing in the model are the only important ones, unless information about the excluded variables is given,
- it provides a means for the data analyst to select a model that favours the hypothesis they want to promote (by allowing them to select which explanatory variables appear in the model).

The SPEC power benchmark[535] is designed to measure single and multi-node server power consumption while executing a known load. The results contain 515 measurements of six system hardware characteristics, such as number of chips, number of cores and total memory, as well as average power consumption at various load factors.

A model of average power consumption, at 100% load, containing a linear combination of all explanatory variables, shows very high multicolinearity for the number of chips (its VIF is 27.5 and several other variables have a high VIF; see rexample[SPECpower.R]). Removing

this variable reduces the VIF of the remaining variables, but the AIC drops from 6798.7 to 7182.1. Whether this decrease in model performance is an important issue depends on the reason for building the model, i.e., prediction or understanding. Do the values of the model coefficients, after removing this variable, provide more insight than the coefficient values of the original model? These kinds of questions can only be answered by a person having detailed domain knowledge. This example illustrates that removing a variable solves one problem and raises others.

A study of fault prediction by Nagappan, Zeller, Zimmermann, Herzig and Murphy[406] produced data containing six explanatory variables having an exact linear relationship with other explanatory variables. The `glm` function detects the existence of this relationship and excludes the offending explanatory variables from the model (the value returned for their fitted coefficients is `NA`); see rexample[ESEUR/regression/change-burst-sum.R].

Two explanatory variables having an exact linear relationship will have a correlation of ±1, as a call to `alias` will show.

6.4.3 Penalized regression

Penalized regression handles multicolinearity by using a technique that automatically selects how much each explanatory variable should contribute to the model; explanatory variables are penalized based on their relative contribution to the model.

The traditional technique for fitting a regression model involves minimising some measure of the error, where the error is defined to be the difference between actual and predicted values, e.g., the sum of squared error, and the equation is:

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Penalised regression modifies this equation to include a penalty (the λ in the equation below) for the P coefficients in the model (β in the equation below).

$$SSE_{enet} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{P} |\beta_j| + \lambda_2 \sum_{j=1}^{P} \beta_j^2$$

This technique of using both first- and second-order penalties is known as the *elastic net*. When only the first order term, λ_1, is used it is known as the least absolute shrinkage and selection operator method (*lasso*). When only the second order term, λ_2, is used it is known as *ridge regression*.

In theory the penalization penalties, the values of λ_1 and λ_2, are chosen by the user. In practice packages provide a function that automatically finds values (using bootstrap) that minimise the error.

The lasso tends to pick one from each set of correlated variables and ignore the rest (by setting the corresponding βs to zero). Ridge regression has the effect of causing the coefficients, β, of the corresponding correlated variables to converge to a common value, i.e., the coefficient chosen for k perfectly correlated variables is $\frac{1}{k}$th the size chosen had just one of be used.

The calculation of mean squared error adds contributions from both variance and bias. The default regression modeling techniques are unbiased (i.e., attempt to minimise bias). It is possible to build models with lower MSE by trading off bias for variance; one consequence of correlation between variables is that the variance is high...

The `penalized` package...

now if we had some data where it makes a big difference (see rexample[SPECpower.R] for a case where this makes a small difference)...

6.5 Non-linear regression

The term linear is applied to the regression models built so far because the coefficients of the model (e.g., β_1 in the equation at the start of this chapter) are linear (the form of the explanatory variables is irrelevant). In a non-linear regression model one or more of the coefficients appear in a context that creates a non-linear equation, as in the following:

$y = \alpha + \beta_1 x^{\beta_2} + \varepsilon$

Table 6.3 lists some commonly occurring non-linear equations and Figure 6.45 illustrates some example instances of these equations.

The `nls` function (Nonlinear Least Squares) is part of the base system and can be used to build non-linear regression models. This function requires that the response variable error distribution be Normal (the default behavior used by `glm`). The `gnm` package (Generalized nonlinear models) contains support for other forms of error distribution.

Shape	**Name**	**Equation**
Asymptotic growth to a limit	Michaelis-Menten	$y = \frac{ax}{1+bx}$
Asymptotic growth to a limit	Exponential	$y = a(1 - be^{-bx})$
S-Shaped	Logistic	$y = a + \frac{b-a}{1+e^{(c-x)/d}}$
S-Shaped	Weibull	$y = a - be^{-cx^d}$
S-Shaped	Gompertz	$y = ae^{be^{-cx}}$
Humped	Bell-shaped	$y = ae^{-\lvert bx\rvert^2}$
Humped	Biexponential	$y = ae^{-bx} - ce^{-dx}$
Humped	Ricker curve	$y = axe^{-bx}$

Table 6.3: Some commonly encountered non-linear equations, see Figure 6.45.

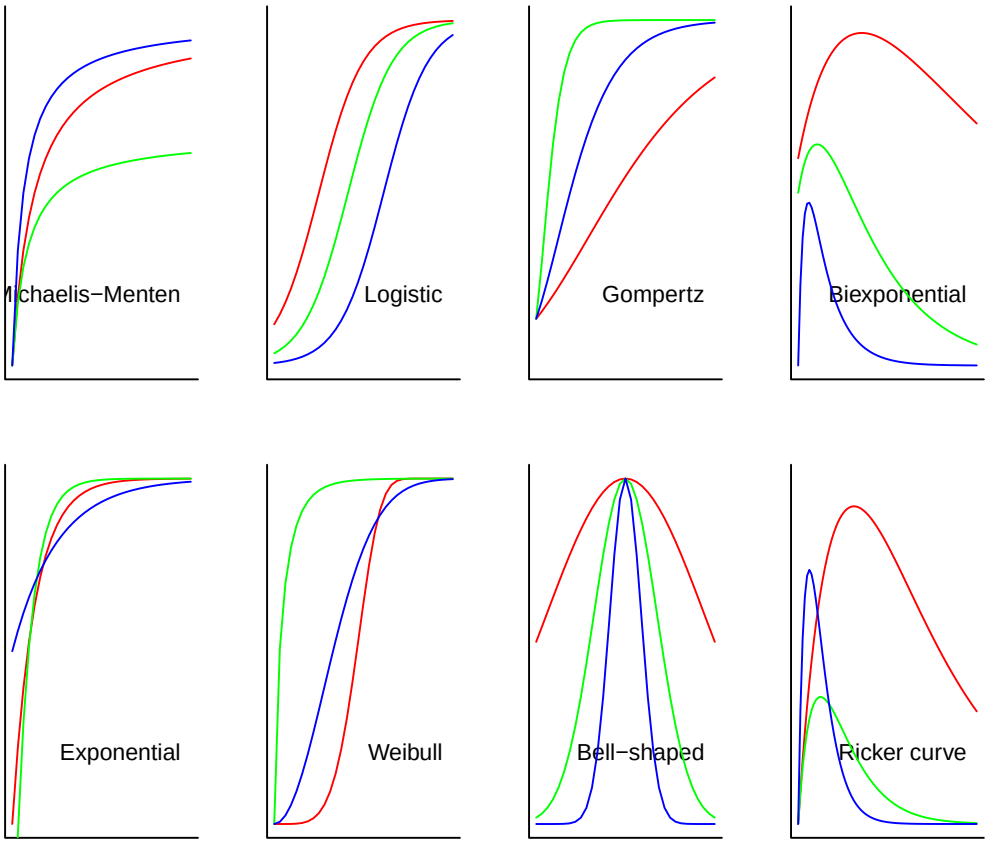


Figure 6.45: Example plots of functions listed in Table 6.3. These equations can be inverted, so they start high and go down. code

From the practical point of view there are several big differences between using `glm` and using `nls`, including:

- `nls` may fail to fit a model; the techniques used to find the coefficients of a non-linear model are not guaranteed to converge,
- `nls` may return a fitted model that differs from the actual solution; the techniques used to find the coefficients of a non-linear model may become stuck in a local minimum that is good enough and not find a better solution,
- `nls` often requires users to provide an estimate of the initial values for the model coefficients that is very close to the final values (the `start` argument),

- the names of the model coefficients being estimated have to explicitly appear in the formula,
- the operators appearing in the expression to the right of ~ have their usual arithmetic interpretation, i.e., the behaviors listed in Table 6.2 do not apply.

The biggest problem with fitting non-linear regression models is finding a combination of starting values that enable `nls` to converge to a fitted model. Possible techniques include:

- using a "self-start" function, if available (e.g., `SSlogis` for Logistic models), these attempt to find good starting values to feed into `nls`. These function, in turn, require starting values, but at least there is a known method for calculating them,
- fitting a linear model that is close enough to the non-linear model and using the coefficients of the fitted linear model as starting values,
- using the argument `trace=TRUE`, which outputs the list of model coefficients being tried internally, as a source of ideas to try,
- picking a few points in the plotted data that a fitted line is likely to pass through and calculate values that would result in an equation used passing close to these points.

A study by Hazelhurst[254] measured the performance of various systems running a computational biology program. Figure 6.46 shows four non-linear equations fitted to one processor characteristic (L2 cache size). The calls to `nls` are as follows:[xxii]

```
b_mod=nls(T1 ~ c+a*exp(b*L2), data=bench,
                start=list(a=300, b=-0.1, c=60))

mm_mod=nls(T1 ~ (1+b*L2)/(a*L2), data=bench,
                start=list(b=3, a=0.004))

gm_mod=nls(T1 ~ a/exp(b*exp(-c*L2)), data=bench,
                start=list(a=80, b=-1, c=0.1), trace=FALSE)
Asym = 0.0125
Drop = 0.002
lrc = -1.0
pwr = 2.5
# 1/SSweibull does not have the desired effect, so have to invert
# the response.
getInitial(1/T1 ~ SSweibull(L2, Asym, Drop, lrc, pwr), data=bench)
wb_mod=nls(1/T1 ~ SSweibull(L2, Asym, Drop, lrc, pwr), data=bench)
```

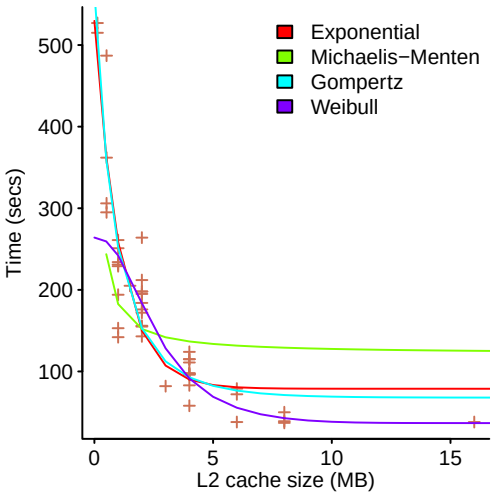


Figure 6.46: Time to execute a computational biology program on systems containing processors with various L2 cache sizes. Data kindly provided by Hazelhurst.[254] code

At the start of this chapter, Figure 6.7, various linear models were fitted to the growth of Linux; polynomials containing integer powers were used, perhaps the data is better fitted by a polynomial containing non-integer powers. The following call to `nls` attempts to fit such an equation using start values obtained from the quadratic model fitted earlier:

```
m1=nls(LOC ~ a+b*Number_days+Number_days^c, data=h2,
                        start=list(a=3e+05, b=-4e+2, c=2.0))
```

The `summary` and `AIC` output is: code

```
Formula: LOC ~ a + b * Number_days + Number_days^c

Parameters:
    Estimate Std. Error t value Pr(>|t|)
a -1.679e+05  2.969e+04  -5.656 2.61e-08 ***
b  7.319e+02  3.463e+01  21.131  < 2e-16 ***
c  1.806e+00  4.616e-03 391.211  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 231800 on 498 degrees of freedom

Number of iterations to convergence: 5
Achieved convergence tolerance: 4.299e-06

[1] "AIC =  13805.2100165816"
```

[xxii] It is difficult to separate inspiration from suck it and see in this process.

showing that the equation:

$$sloc = (-1.68 \times 10^5 \pm 3 \times 10^4) + (7.32 \times 10^2 \pm 3.5 \times 10^1) Number\_days + Number\_days^{1.81 \pm 4.6 \times 10^{-3}}$$

is a slightly better fit than a cubic equation (i.e., a lower AIC) and also predicts continuing growth (unlike the cubic equation).

It is possible that further experimentation will find a polynomial model with a lower AIC. However, the purpose of this analysis is to understand what is going on, not to find the equation whose fitted model has the lowest AIC.

A more practical issue to address is the creation of a model that makes what are considered to be more realistic future predictions. The growth in the number of lines of code in the Linux kernel will not continue forever, at some point the number of lines added will be closely matched by the number of lines deleted. One commonly seen growth pattern starts slow and then has a rapid growth period followed by a levelling off converging to an upper limit (i.e., an S-shaped curve). The Logistic equation is S-shaped and is often used to model this pattern of growth; the equation involves four unknowns (fourth row in Table 6.3).

```
# suck it and see...
m3=nls(LOC ~ a+(b-a)/(1+exp((c-Number_days)/d)), data=h2,
                start=list(a=-3e+05, b=4e+6, c=2000, d=800))
# no thinking needed, SSfpl works out of the box for this data :-)
m3=nls(LOC ~ SSfpl(Number_days, a, b, c, d), data=h2)
```

The AIC for the fitted Logistic equation is slightly worse than the cubic polynomial (13,273 vs 13,220), but a lot better than the quadratic fit and predicts a future trend that might be considered more likely to occur.

While the `predict` function includes parameters to request confidence interval and standard error information, support for both is currently unimplemented. The `confint` function in the `MASS` package, when passed a model built using `nls`, returns the confidence intervals for each of the model coefficients. Alternatively, bootstrapping can be used to find confidence intervals.

Figure 6.47 shows the fitted model predicting a slow down in growth and the maximum being reached at around 10,000 days. Who is to say whether this prediction is more likely to occur, over the specified number of days than the continuing increase predicted by the quadratic model? Given that the one explanatory variable used to fit the models, time, has no direct impact on the production of source it is no surprise that the predictions of future behavior made by the various models vary so wildly.

One technique for obtaining a rough idea of the accuracy of the future predictions made by a model is to fit models to subranges of the available data, and then check the predictions made against the known data outside the subrange. Figure 6.48 shows logistic equations fitted to various subranges of the data, e.g., all data up to 2900, 3650, 4200 number of days and all days.

The lesson to learn from Figure 6.48 is to be careful what you ask for, if you ask for a logistic equation fitted to the data, you may get one. The fitting process is driven by your expectations (in the form of a formula) and the data it is given.

The processes generating the data fitted by a Logistic equation may only broadly follow this pattern, with independent processes each making separate contributions. A study by Grochowski and Fontana[233] showed that increases in the density of data stored on hard disks could be viewed as a sequence of technologies that each rapidly improved (e.g., magneto-resistive and antiferromagnetically-coupled). Figure 6.49 shows the areal density (think magnetic domains) of various models of hard disk on first entering production. Improvements in each technology can be fitted with its own Logistic equation, as can the overall pattern of performance improvements.

A codebase showing some evidence of having completed its major expansion phase us glibc (i.e., its growth rate has levelled off), the GNU C library; see Figure 6.50. Plugging the fitted model coefficients into the Logistic equation we get:

$$y = -28168 + \frac{1114626 + 28168}{1 + e^{(3652-x)/935}}$$

The C Standard's committee, JTC1 SC22/WG14, have recently started work on revising the current specification. So the model's prediction that glibc will max out at around 1,115,000 lines is unlikely to come true.

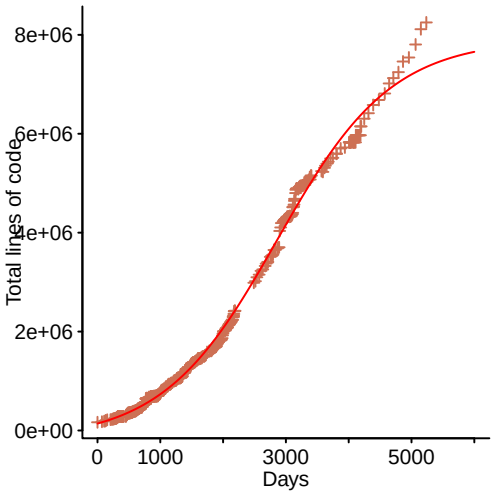


Figure 6.47: A logistic equation fitted to the lines of code in every non-bugfix release of the Linux kernel since version 1.0. Data from Israel et al.[289] code

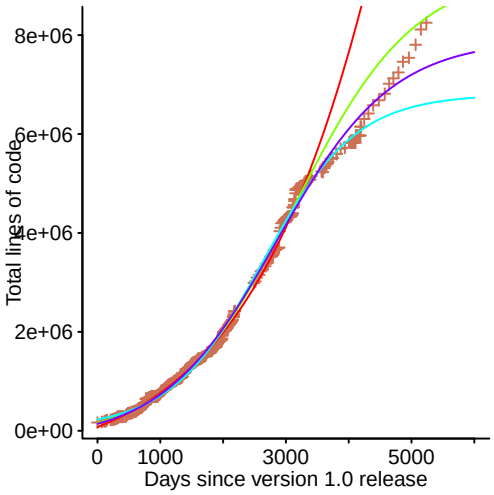


Figure 6.48: Predictions by logistic equations fitted to Linux SLOC data, using subsets of data up to 2900, 3650, 4200 number of days and all days since the release of version 1.0. Data from Israel et al.[289] code

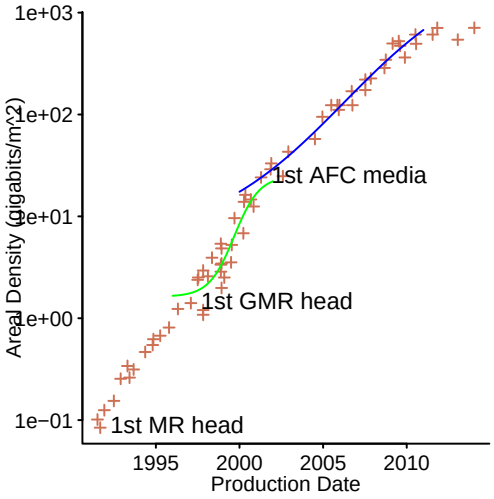


Figure 6.49: Increase in areal density of hard disks entering production over time. Data from Grochowski et al.[233] code

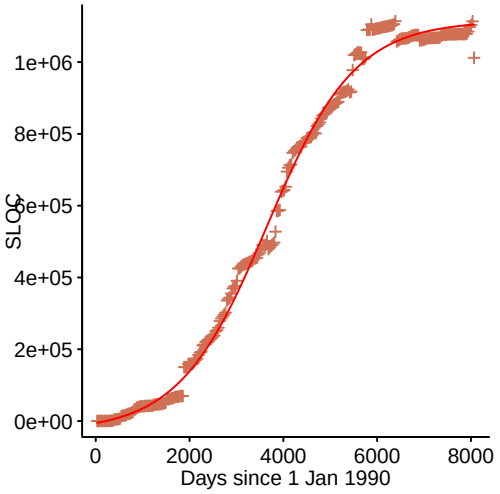


Figure 6.50: Lines of code in the GNU C library against days since 1 January 1990. Data from González-Barahona.[212] code

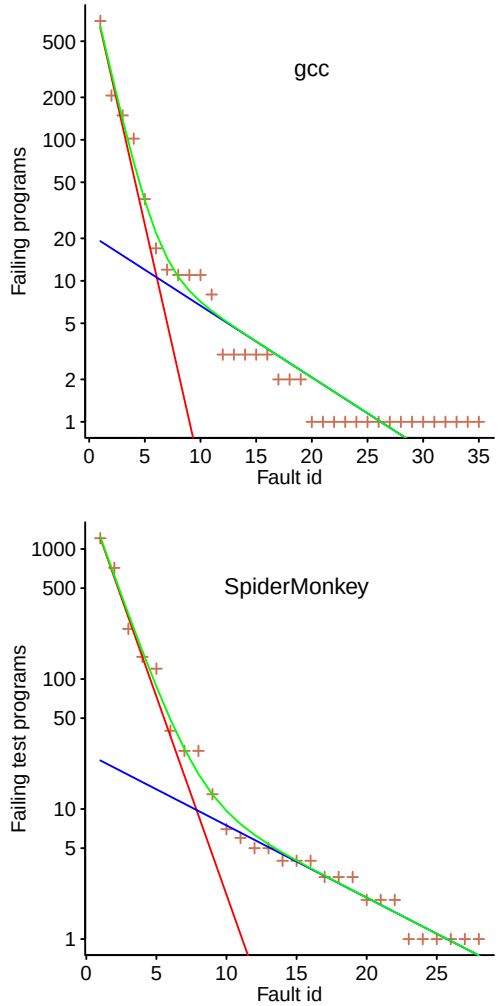


Figure 6.51: Number of failing programs caused by unique faults in gcc (upper) and SpiderMonkey (lower). Fitted model in green, with two exponential components in red and blue. Data kindly provided by Chen.[105] code

A study by Chen, Groce, Fern, Zhang, Wong, Eide and Regehr[105] investigated faults in a C compiler and JavaScript engine by having them process randomly generated programs. Some programs failed in some way (1,298 in gcc and 2,603 in Mozilla's SpiderMonkey) and many of these failures could be traced back to the same few underlying compiler faults, i.e., some faults were encountered more often than others. Figure 6.51 shows the number of failing programs that could be traced to the same compiler fault, a curved green line is the fit (a biexponential, or double exponential) and two straight lines are exponentials that are added to create the fit.

The `nls` has a `SSbiexp` starter function, which performs poorly (or, at least, I could not make it do better) for this data.

The sample counts count data, with many very small values. A Poisson error distribution probably applies. When the Normal distribution is not a good enough approximation for measurements errors in the response variable, the `gnm` function, from the `gnm` package, can be used.

The formula notation used by `gnm` is based on function calls,[576] rather than the binary operators used by `glm` and `nls`. The formula argument in the following call (used to fit the model plotted in Figure 6.51) contains two exponentials (specified using the `instances` function), which are specified as a constant (the literal `1` is a placeholder for an unknown constant) multiplied (the `Mult` function) by an exponential (the `Exp` function); like calls to `nls`, starting values are required:

```
library("gnm")

fail_mod=gnm(count ~ instances(Mult(1, Exp(ind)), 2)-1,
                data=wrong_cnt, verbose=FALSE,
                start=c(2000.0, -0.6, 30.0, -0.1),
                family=poisson(link="identity"))
```

Why is a biexponential model such a good fit? A speculative idea is that the two exponentials are driven by the two independent processes involved in creating the data: the tool generating the input programs and each compiler. The tool just happens to regularly generate particular patterns of source and certain paths through the compiler are more likely than others; each of these two patterns of behavior driving a process that can be modeled by a single exponential.[xxiii]

6.5.1 Power laws

Plotting values drawn from a power law distribution using a log scale for both axis, produces a straight line. This straight line characteristic is not unique to power laws and can also be seen in values drawn from other distributions over a wide range of values, e.g., an exponential distribution.[xxiv]

The `poweRlaw` package includes functions for fitting and checking whether a power law is likely to be a good fit for a sample.[113]

When the model being fitted contains only one explanatory variable having the form of a power law, use of functions from the `poweRlaw` package is the recommended approach. However, this package does not support more complicated models and so other functions have to be used when a power law is one of multiple components in a model, e.g., `nls`.

A study by Queiroz, Passos, Valente, Hunsen, Apel and Czarnecki[468] analysed the conditional compilation directives (e.g., `#ifdef`) used to control the optional features in 20 systems written in C. Researchers in this area use the term *feature constants* to denote macro names used to control the selection of optional features and *scattering degree* to describe the number of `ifdefs` that refer to a given feature constant, e.g., if the macro `SUPPORT_X` appears in two `ifdefs`, it has a scattering degree of two.

Figure 6.52 shows the total number of feature constants (y-axis) having a given scattering degree (x-axis) in these 20 systems. A power law (red) and exponential (blue) is fitted to the

[xxiii] Working out which process corresponds to which exponential appearing in the plots is left as an exercise to the reader (because your author has no idea).

[xxiv] Papers[360] claiming to have found a power law purely on the basis of a plot showing points scattered roughly along a straight line are surprisingly common.

data; the numbers are the p-values for the fit (higher is better, i.e., fail to reject the hypothesis). This is a fishing expedition involving 20 systems and a power law is suggested by the visual form of the plotted data and with multiple tests it is necessary to take into account the increased likelihood of a chance match.

If 0.05 is taken as the p-value cutoff, below which the distribution hypothesis is rejected for one test, then $(1-0.95^{20}) \rightarrow 0.64$ is the cutoff when 20 tests are involved. Some systems have p-values above the cutoff for one of the power law or exponential fitted models and so the chosen distribution is not rejected for these systems.

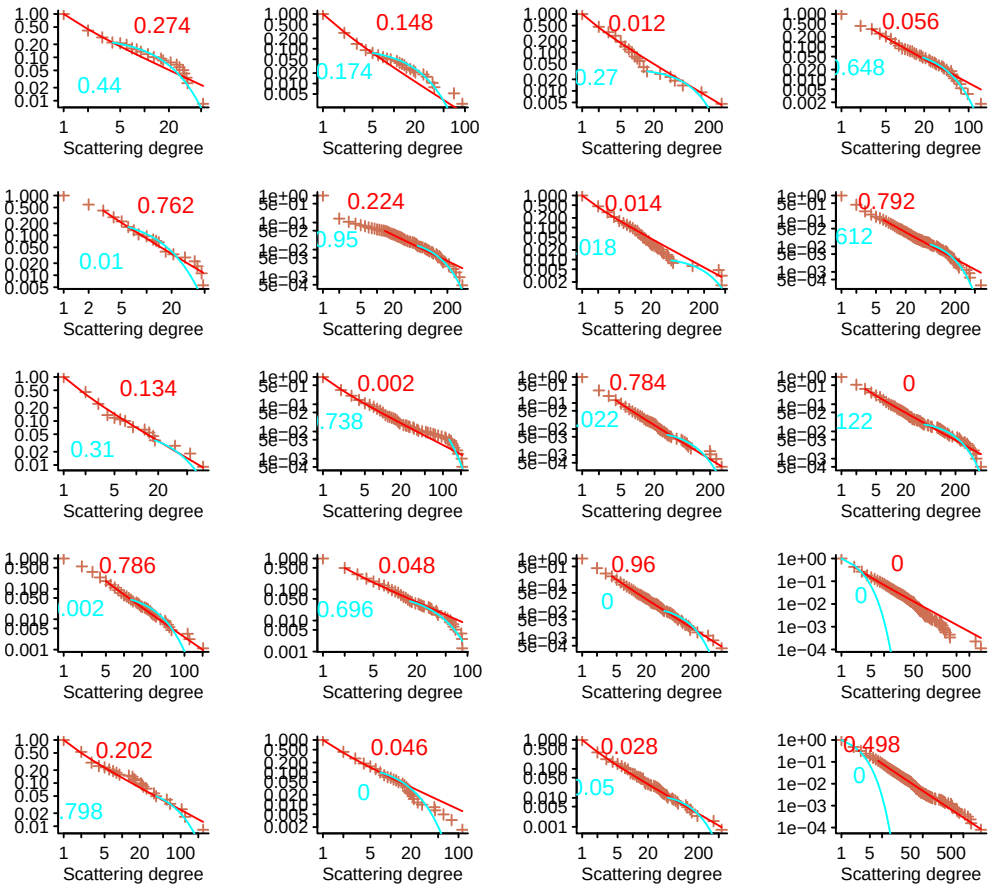


Figure 6.52: Power law (red) and exponential (blue) fits to feature macro usage in 20 systems written in C; fail to reject p-value for 20 systems is 0.64. Data from Queiroz et al.[468] code

The poweRlaw package supports discrete and continuous forms of heavy tailed distributions. The scattering degree is an integer value and the code below fits both a discrete power law and exponential to the data (the continuous forms are conpl and conexp respectively):

The power law equation includes a minimum value of x, scattering degree in this case, below which it does not hold. The estimate_xmin function estimates the value, x_{min}, that minimises the error between the fitted model and the data. The new function, called by the constructor, sets x_{min} to the minimum value present in the data. It is common for power laws to fit a subset of the data.

```
# Fit scattering degree
# displ is the constructor for the discrete power law distribution
pow_mod=displ$new(FS$sd)
exp_mod=disexp$new(FS$sd)  # discrete power exponential

# Estimate the lower threshold of the fit
pow_mod$setXmin(estimate_xmin(pow_mod))
exp_mod$setXmin(estimate_xmin(exp_mod))

# Plot sample values
plot(pow_mod, col=point_col, xlab="Scattering degree", ylab="")
lines(pow_mod, col=pal_col[1]) # Plot fitted line
lines(exp_mod, col=pal_col[2])

# Bootstrap to test hypothesis that sample drawn from a power law
bs_p=bootstrap_p(pow_mod, threads=4, no_of_sims=500)
```

```
text(40, 0.5, bs_p$p, pos=2, col=pal_col[1]) # Display value
```

6.6 Mixed-effect models

A study by Balaji, McCullough, Gupta and Agarwal[40] measured the power consumption of six different Intel Core i5-540M processors executing the SPEC2000 benchmark at various clock frequencies. The six processors measured are a sample of the entire population of Intel Core i5-540M processors. The power consumption characteristics of this might be modeled using the combined data from all six processors; the following is the `summary` output for this model: code

```
Call:
glm(formula = meanpower ~ frequency, data = power_bench)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5746  -0.1882   0.0413   0.1902   2.2965

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.12594    0.01506   141.2   <2e-16 ***
frequency    1.95248    0.00767   254.6   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.1429928)

    Null deviance: 10692.7  on 9980  degrees of freedom
Residual deviance:  1426.9  on 9979  degrees of freedom
AIC: 8916.2

Number of Fisher Scoring iterations: 2
```

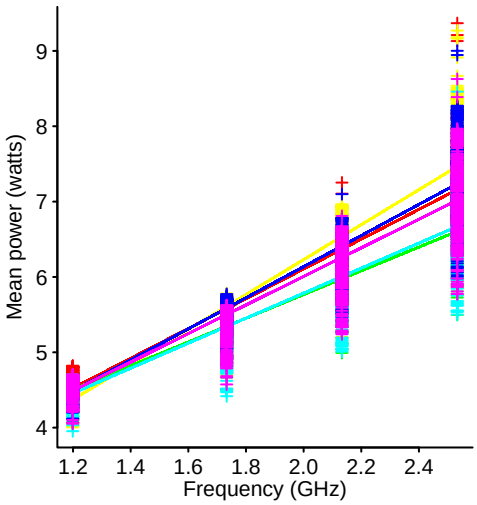


Figure 6.53: Power consumption of six different Intel Core i5-540M processors running at various frequencies; colored lines denote fitted regression models for each processor. Data from Balaji et al.[40] code

This model does not provide any information about how performance varies between processors. The identity of the processor measured could be included in the model (see rexample[hotpower-proc.R]), but this cannot be generalized to the entire population.

Possible techniques for estimating model variability caused by processor differences include:

- build a regression model for each processor and then average these six models in some way, e.g., use the coefficients from the six models to build a regression model that is a model of models,

  Electronic circuit theory tells us that processor power consumption is proportional to clock frequency and Figure 6.53 shows the results of fitting a separate straight line to the data for each processor.

- build a *mixed-effect model*. A mixed-effect model [xxv] might be viewed as a model of models; mathematically it uses a more direct approach that makes more effective use of the available data than the method described above.

In a mixed-model the explanatory variables are classified as either a *fixed-effect* or *random-effect* (sometimes called a *covariate*). Technically the effects are not fixed and are not random[xxvi]. One way to think about classifying the two kinds of explanatory variables is to look at the impact they have on the response variable:

- fixed effects influence the mean value of the response variable and are associated with the entire population,

- random effects influence the variance of the response variable and are associated with individual subjects.

[xxv] Also known as a *hierarchical model*.

[xxvi] Some authors point this out and then proceed to use what they consider to be more technically correct terms, this book follows common usage because it is common; these terms crop up as named parameters in functions and appear in output information.

Mixed-effect models are used to model measurements of clusters of related subjects amd multiple correlated measurements of the same subjects (e.g., before/after measurements of the same subject).

A number of different packages are available for building mixed-effect models, the one primarily used in this book is `lme4`, whose workhorse functions are the `glmer` and `lmer` functions[xxvii].

The `lme4` package extends the formula notation to support the specification of random effects. In the following code:

```
library("lme4")

# Express in Gigahertz (otherwise lmer does not converge)
power_bench$frequency=power_bench$frequency/1000000

p_mod=lmer(meanpower ~ frequency + (1 | processor), data=power_bench)
p_mod=lmer(meanpower ~ frequency + (frequency-1 | processor), data=power_bench)
p_mod=lmer(meanpower ~ frequency + (frequency   | processor), data=power_bench)
```

- first call to `lmer`: `frequency` is the fixed-effect and `(1 | processor)` is the random-effect; the `1` specifies there is variation in the value of the intercept and the source of this variation is the `processor` variable (i.e., the column having this name in the data frame). When plotted the models might look like those of the upper plot of Figure 6.54 with six lines intersecting the y-axis at different points, but all having the same slope,
- second call to `lmer`: the operand `(frequency-1 | processor)` specifies there is variation in the value of the slope and the source of this variation is the `processor` variable. When plotted the models might look like the lines in the middle of Figure 6.54, where all lines intersect the y-axis at the same point but have different slopes.
- third call to `lmer`: the operand `(frequency | processor)` specifies that variation in the `processor` variable causes both the intercept and the slope to vary. When plotted the models might look like the lines in the lower of Figure 6.54, where the lines have different intersections and slopes.

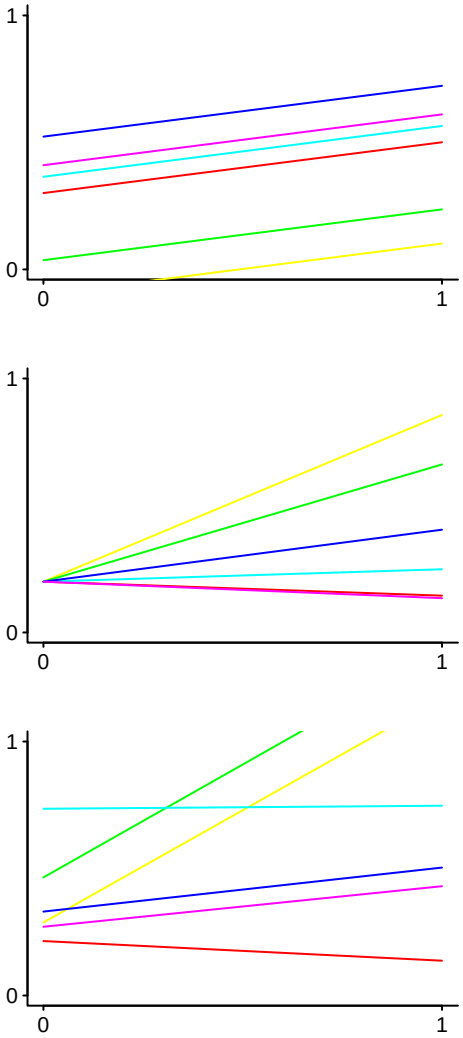


Figure 6.54: Example showing the three ways of structuring a mixed effects model, i.e., different intersections/same slope (upper), same intersection/different slopes (middle) and different intersections/slopes (lower). code

The following is the `summary` output from a mixed-effect model where the processor is a random effect on both the intercept and slope: code

```
Linear mixed model fit by REML ['lmerMod']
Formula: meanpower ~ frequency + (frequency | processor)
   Data: power_bench

REML criterion at convergence: 6300.3

Scaled residuals:
    Min      1Q  Median      3Q     Max
-4.0533 -0.4866  0.1453  0.4994  6.6743

Random effects:
 Groups    Name        Variance Std.Dev. Corr
 processor (Intercept) 0.12904  0.3592
           frequency   0.07383  0.2717   -0.99
 Residual              0.10941  0.3308
Number of obs: 9981, groups:  processor, 6

Fixed effects:
            Estimate Std. Error t value
(Intercept)   2.1740     0.1473   14.76
frequency     1.9156     0.1111   17.23

Correlation of Fixed Effects:
          (Intr)
frequency -0.993
```

[xxvii] A call to the `glmer` function that uses the default family distribution, i.e., gaussian, generates a warning that this usage is deprecated and `lmer` should be used.

The estimated coefficients, listed under `Fixed effects:`, for `(Intercept)` and `frequency` are very similar to the combined data model fitted earlier.

Annoyingly the `summary` output does not include p-values. These can be obtained using the `Anova` function from `car` package.

The `Random effects:` information lists the variation introduced by `processor` (listed in the `Groups` column, on the variables listed in the `Name` column); `Residual` lists the residual left after taking all the specified random effects into account.

Taking `frequency` as an example, there are two sources of uncertainty in its contribution to the response variable (as expressed in its coefficient), one from fixed effects and a random effect caused by processor variation.

Plotting the 95% confidence intervals for the intercept and slope of a mixed-effect model provides a visualization of the relative contribution of the sources of variation. Figure 6.55 was generated by the following code, using data from the six processors:

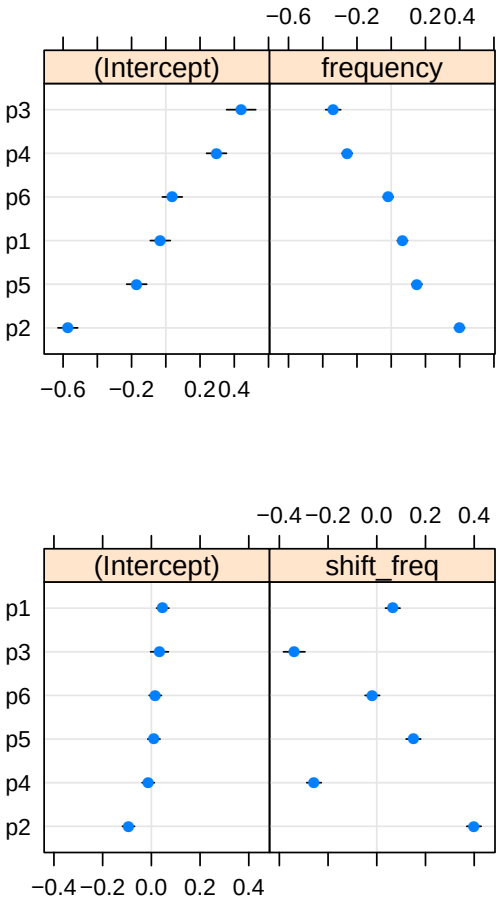


Figure 6.55: Confidence intervals, 95%, for within-subject intercept and slope (right plots) of mixed-effect models in the adjacent code. code

```
library("lattice")
library("lme4")
library("gridExtra")

proc_mod=lmer(meanpower ~ frequency +(frequency | processor),
                                                  data=power_bench)
dp_orig=dotplot(ranef(proc_mod, condVar=TRUE), main=FALSE)

power_bench$shift_freq=power_bench$frequency-min(power_bench$frequency)
proc_mod=lmer(meanpower ~ shift_freq +(shift_freq | processor),
                                                  data=power_bench)

dp_shift=dotplot(ranef(proc_mod, condVar=TRUE), main=FALSE)

# dotplot comes from the lattice package which uses grid layout
grid.arrange(dp_orig$processor, dp_shift$processor, nrow=2)
```

The upper plot is the model fitted using the original data; the intercept and slope (right plot) appear to be correlated. Looking at the straight lines fitted to each processor (Figure 6.53), they appear to share an origin starting at the lowest frequency measured; an intercept included as a random effect has a common origin assumed to start at zero (see Figure 6.54). Shifting frequency values down by the minimum measured value and refitting the model produces the confidence intervals in the lower plot. The correlation has disappeared; perhaps including the intercept as a random effect is not worthwhile.

Refitting a model without the intercept as a random effect, produces a model that only differs from previous models by a small amount (see rexample[ESEUR/regression/hotpower-mix-plot]).

There is a limit on the number of random effects (i.e., number of unknowns) that can occur in a model. The total number of unknown random effects must be less than the number of observations, otherwise the equations do not have a unique solution. A continuous explanatory variable counts as a single unknown, while a variable holding nominal or ordinal values contributes one unknown for each of the possible discrete values (there is no slope associated with fitting a variable that is not treated as being continuous)...

Other operators (a || b) (a | b / c)...

Example using bootstrap to calculate confidence intervals...

6.7 Generalised Additive Models

The regression modeling techniques discussed so far have required the user to provide an equation expressing every detail of the relationship between explanatory variables and the response variable (they are said to be parametric models). If there is no reason to believe that any particular equation applies, then a *Generalised additive model* (GAM) provides an alternative approach. A GAM only requires a list of explanatory variables and a response variable to be specified (these models are said to be nonparametric models).

A GAM is built by finding the best fit for a sequence of polynomial equations (e.g., some form of spline) that smoothly captures the shape of the data. These smooth equations might be used to make predictions, or when the fitted model is plotted may suggest possible parametric equations. The details of the fitted equations are not a source of understanding, but they can make very good predictions.

The `gam` function in the `mgcv` package can be viewed as extending the functionality of `glm` to support a variety of nonparametric smoothing functions (the `gam` package is simpler, but does not offer such a wide range of functionality). The following code shows formulas using a different smoothing function for each explanatory variable (first line below), a different smoothing function for some combinations of explanatory variables (second and third line), a combination of a smoothing function and parameterised form (fourth line) or an interaction between a smoothed and non-smoothed variable (fifth line; the `by` parameter rather than the `:` operator has to be used in this case):

```
mod=gam(y ~ s(x_1) + s(x_2) + s(x_3), data=foo_bar)
mod=gam(y ~ s(x_1) + s(x_2, x_3), data=foo_bar)
mod=gam(y ~ s(x_1) + s(x_2, x_3) + s(x_3, x_4) + s(x_4), data=foo_bar)
mod=gam(y ~ x_1 + s(x_2) + x_3, data=foo_bar, family="poisson")
mod=gam(y ~ x_1 + s(x_2, by=x_1) + x_3, data=foo_bar, family="poisson")
```

The smoothing function `s` supports a variety of options for controlling the fitting process. Two that are likely to be encountered are `k`, used to set an upper limit on the degrees of freedom that can be used in the fitted equation, and `bs` a string identifying the kind of smoother (e.g., "tp", the default, for a thin plate regression spline and "cr" for a cubic regression spline).

The value of `k` needs to be large enough to support the degrees of freedom needed by a polynomial capable of representing the underlying pattern in the data, but not to large as to require unacceptable computational resources. The `gam.check` function provides information about fitted models that can be used to help select a value for `k`.

The fitting procedure used by the `mgcv` version of `gam` tries to avoid overfitting by making every degree of freedom pay its way (using, for instance, *penalized regression splines*). Criteria used for measuring the *cost-effectiveness* of more complicated models include generalised cross-validation (GCV; the default) and AIC. The `select` argument provides support for *null space penalization*, see package documentation for details.

A study by Lee and Brooks[349] built a model to predict the performance and power consumed by applications running on processors having various hardware configurations (e.g., number of registers, size of cache and instruction latency).

The following additive model is based on the one proposed by Lee et al and explains over 95% of the variance in the data (see rexample[lee2006.R]). While this model is likely to be useful for prediction, it provides virtually no insight into the performance characteristics of the various hardware attributes.

```
l_mod=gam(sqrt(bips) ~ benchmark + fix_lat
                        +s(depth, k=4) + s(gpr_phys, k=10)
                        +s(br_resv, k=6) + s(dmem_lat, k=10) +
                                           s(fpu_lat, k=6)
                        +s(l2cache_size, k=5) + s(icache_size, k=3) +
                                          s(dcache_size, k=3)
                        +s(depth, gpr_phys, k=10)+s(depth, by=width, k=6)
                        +s(gpr_phys, by=width, k=10)
                  , data=lee)
```

An earlier chapter showed two approaches to modeling the number of accesses to a function's local variables. Without knowing anything about what relationships might exist between explanatory and response variables, and being willing to use very high degree polynomials, it is possible to build and use `gam` to build a prediction model

In the calls to `gam` below, the first assumes there is an interaction between the two explanatory variables (allowing up to 75 degrees of freedom) and the second assumes the variables are independent (allowing up to 50 degrees of freedom for each of them). While the fitted model (see rexample[obs-fit.R]) might make usable predictions, the use of such high degree polynomials suggests that the underlying model has a non-polynomial form.

```
locg_mod=gam(norm_occur ~ s(object.access, total.access, k=75),
                                    data=common_loc, family=Gamma)

locp_mod=gam(norm_occur ~ s(object.access, k=50)+s(total.access, k=50),
                                    data=common_loc, family=Gamma)
```

6.8 Miscellaneous

Stuff that has no other obvious home...

The p-value, for the coefficients of a fitted model, is a test of the hypothesis that the coefficient is zero, i.e., there is no association. When the actual value of a coefficient is close to zero, the reported p-value may be spurious. One solution is to rotate the axes, which will have the effect of increasing the value of the coefficient and removing any artefact from the p-value calculation.

6.8.1 Advantages of using `lm`

Many books using R introduce readers to regression through the `lm` function; one reason for this is herd mentality, it's what everybody else does. While this book promotes `glm` as a one stop solution, the `lm` function does have some advantages over `glm`, including:

- requiring less cpu time to fit a model. For extremely large datasets the performance difference may be worth considering,
- requiring less memory to fit a model. For extremely large datasets' memory requirements may be excessive for `glm`; possible solutions that continue to use `glm` are discussed below,
- the algorithm used by `lm` is always guaranteed to converge to a solution, singularities generated by correlation between explanatory variables excluded. There are edge cases where `glm` does not find a solution without being given some reasonable starting values...

The implementation of `lm` is based on the mathematics of *Ordinary Least Squares* (OLS) and the data has to satisfy more conditions for OLS to be applicable... constant variance...

The `lm` function requires that the error variance is constant (in practice close to constant). The `ncvTest` function, in the `car` package, checks that a fitted model meets this requirement; the `spreadLevelPlot` function provides some visualization. Also, see the `lmtest` function.

A user interface issue with models built using `glm` is that they do not come with an easy to understand goodness of fit number (`lm` has the R-squared value ... no r-squared supplied by `glm` and ???).

6.8.2 Network data

It is possible to build regression models over network data. But it does require some data to show how...

?

6.8.3 Alternative residual metrics

The default error metric used by many regression techniques is based on squaring the difference between the actual and predicted value. This choice is driven by the usefulness of the mathematical properties of sum of squares. Other error metrics could be used, for instance the absolute difference between actual and predicted values.

The `rlm` function in the `MASS` package supports user specified functions for calculating the residual to be minimised when fitting a model (the `psi`). Supported functions include... Huber...

6.8.4 Quantized regression

While removing 1% of outliers might make a significant difference to model accuracy, a quantized regression model might be more informative...

Minimises the residuals of the median (rather than the mean)...

?

6.8.5 Prediction vs. interpretation

A different set of trade-offs...

interested in minimising local residual error when predicting, rather than global residuals when interpreting...

variance bias tradeoff...

6.8.6 Solving systems of equations

a system of simultaneous equations, the `systemfit` package...

solve linear equality or inequality conditions `limSolve` package...

6.8.7 Very large datasets

The `biglm` package allows some kinds of regression models to be built using data that is too large to fit in memory all at once; the models are built using an incremental algorithm that only requires a subset of the data to be held in memory at any time. A variety of options are available for creating chunks of data to feed into the model building process, including incremental reading from files and databases.

The `biganalytics` package extends the `bigmemory` package by providing interfaces to various analytic packages, such as `biglm` (see rexample[bounds_chk.R] for an example)....

The `glm` function stores a lot of information in the object it returns, much of which is often not subsequently used...

6.8.8 Communicating model details

Information about fitted models needs to be communicated to the intended audience. The output produced by `summary` may appear cluttered in some contexts and it is not the easiest to interpret for casual users...

The `texreg` package contains function for mapping statistical model values to LaTeX and HTML tables...

6.9 Time series

Time series analysis deals with measurements that are sequentially correlated. An example of a measurement that is correlated with the previous measurement is current room temperature, which is likely to be similar to the temperature 10 minutes ago and the temperature in 10 minutes from now.

Techniques developed to analyse time-series can be used to analyse measurements of any quantity where a correlation exists between successive measurements.

A time series contains one or more of the following three components:

- underlying trend: which changes slowly,
- regular recurring pattern of changes (known as *seasonality*): for instance, expected daytime temperature throughout the year,

- random, irregular or fluctuating component.

The `stl` function (Seasonal Trend using Lowess) provides an easy way of splitting a time series (the argument must be an object of type `ts`, with a user specified frequency; the `stl` function does not automatically detect the recurrence period) into these three components (`plot` displays the individual components).[xxviii]

Figure 6.56, from a study by Eyolfson, Tan and Lam,[169] shows the three time-series components of the hourly rate of commits to the Linux kernel source tree, over the days of a week (the commits during the same hour of the same day were summed). The `stl` function assumes a fixed, recurring, pattern of seasonal behavior, a slowly changing trend and everything else is classified as random noise.

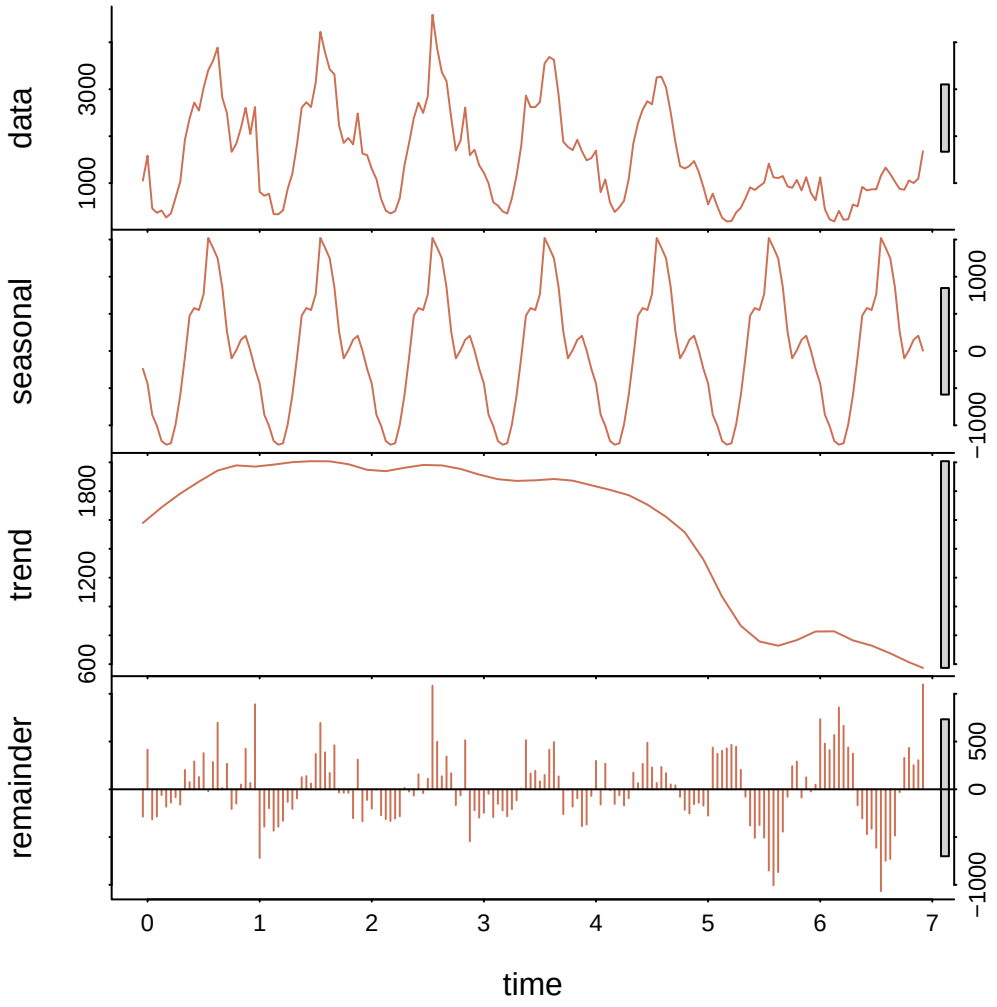


Figure 6.56: The three components of the hourly rate of commits, during a week, to the Linux kernel source tree; components extracted from the time series by `stl`. Data from Eyolfson et al.[169] code

rexample[commit-components.R]

```
# A seasonal frequency has to be specified
hr_ts=ts(linux_hr, start=c(0, 0), frequency=24)
plot(stl(hr_ts, s.window="periodic"))
```

Possible outputs from a time-series analysis include:

- a model of how the value of a quantity at time t depends on its values at an earlier time (often $t-1$),
- a regression model that take account of correlation between measured values,
- power spectrum showing the dominant frequencies present in the data,
- hierarchical clustering of multiple time series.

[xxviii] The `decompose` function, part of the base system, implements the same functionality in a less sophisticated way.

Structure is often added to the continuous, linear, nature of time by imposing repeating fixed length intervals, such as hours of the day and days of the week. Many time series analysis techniques require measurements to occur fixed length intervals; analysis of measurements at irregular intervals is not discussed here (but if we had some data...).

Some library functions use a time series datatype for representing time related measurements. The `ts` function, part of the base system, converts a vector to class `ts` (many time series functions will automatically convert vectors to this class).

The `xyplot` function, in the `lattice` package, can be used to create a time series strip chart, see Figure 3.18.

6.9.1 Cleaning time series data

Many time series techniques implicitly assume that measurement data occurs at regular intervals. A measurement process may only record events when they occur and if no event occurred in within an interval there may be no data-point for that interval. Part of the cleaning process involves ensuring that every interval contains a value (which may be zero or inferred from surrounding values).

A study by Buettner[86] gathered project staffing information for various commercial development software projects. On large commercial projects the amount of work done at weekends is likely to be zero (except for the weeks prior to major deliveries) and the autocorrelation of project activity is likely to show a recurring pattern involving two consecutive days separated by seven days, i.e., weekends and weekdays.

Figure 6.57 shows the autocorrelation of the number of defects found on a given day for one development project. The seven day recurring pattern involves three consecutive days, are the developers only working a four-day week? It turns out [xxix] that contractors on some projects work a two-week cycle, with extra hours worked one week and then not working the Friday of the following week.

The extent to which regular staffing level differences between Friday and other weekdays has to be taken into account will depend on the kind of analysis performed (weekends can be handled by excluding them from the analysis, focusing on where most effort occurs, i.e., week days).

Measurements made on public holidays, such as the New Year, are very likely to differ from normal work days. Removing public holidays from the data will scramble the association with day of the week. The extent to which day of the week is a more important factor in the analysis than public holidays has to be considered.

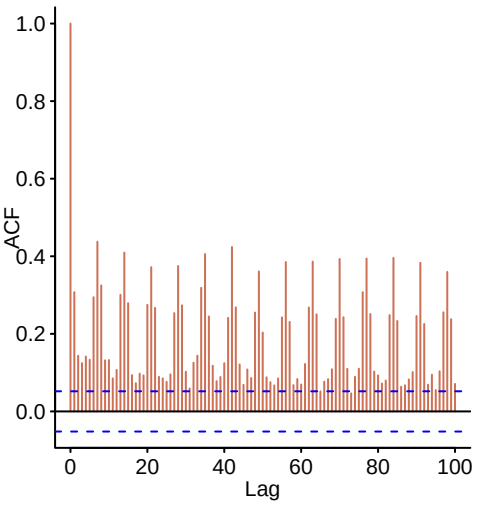


Figure 6.57: Autocorrelation of number of defects found on a given day, for development project C. Data kindly provided by Buettner.[86] code

6.9.2 Modeling time series

The expected mean of a time series can be modeled using one or both of the following two approaches (samples containing values where the variance is serially correlated are discussed later):

- the *Autoregressive model* (AR) is based on the idea that the value at time t can be modeled as a weighted combination of values measured at earlier time steps, plus some amount of added noise (w_t), for instance:

  $x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + w_t$

  is an autoregressive model of order 2, abbreviated AR(2), because it uses values from two time steps back (with weights ϕ_1 and ϕ_2).

  The `ar` function fits data to an autoregressive model.

- the *Moving Average model* (MA) is based on the idea that the value at time t can be modeled by as the sum of noise (w_t) and a weighted combination of the noise from earlier time steps, for instance:

  $x_t = w_t + \theta_1 w_{t-1}$

  is a moving average model of order 1, abbreviated MA(1), because it uses noise (with weight θ_1) from one step back.

[xxix] Email discussion with Buettner.

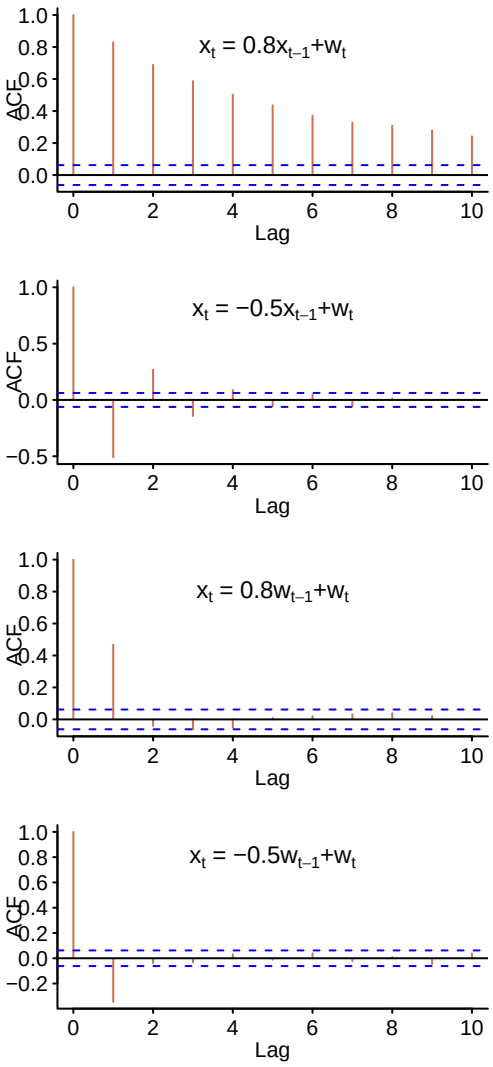
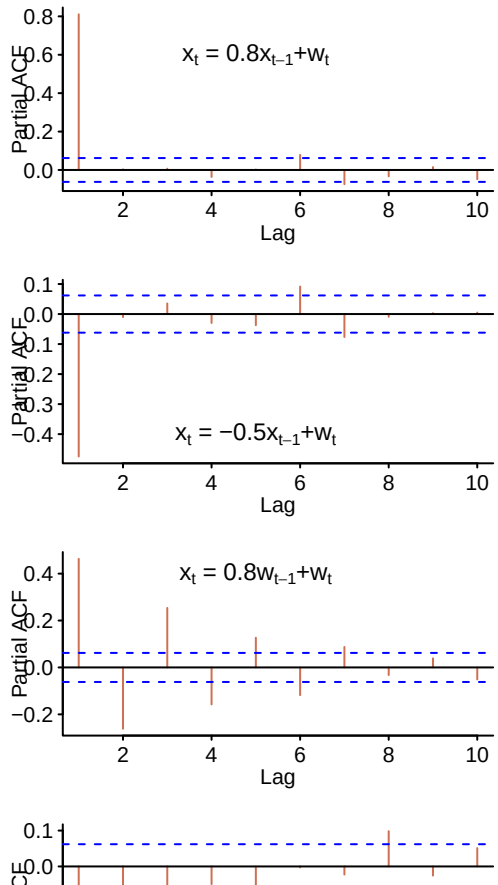


Figure 6.58: Autocorrelation of two AR models (upper plots) and two MA models (lower plots). code



The `arima` function with the first two values of the `order` argument set to zero can be used to fit data to a moving average model.

In both of these models there is a correlation between the value of the response variable x_t and values of this variable at earlier time steps; for the AR(1) model $x_t = \phi x_{t-1}$ the correlation between values separated by k time intervals has decreased by ϕ^k. The autocorrelation function returns the correlation of a time series with itself at successive intervals (i.e., the correlation of the measurement at time t with the measurement at time $t+n$; `n=1:25` is the default sequence of intervals); the `acf` returns and plots this function, see Figure 6.58.

The lag 0 autocorrelation is always one and the two dotted blue lines are 5% p-value bounds. Each lag is a hypothesis test and with 25 hypothesis tests (the default lag used) at least one value is expected to exceed a 5% p-value (with probability $1-0.95^{25} \rightarrow 0.72$), also successive measurements are correlated and so neighbouring lag points are likely to show similar significance levels.

The partial autocorrelation function (implemented in the `pacf` function) calculates and plots the correlation at lag k after removing the effect of any correlation generated by terms at shorter lags. The partial autocorrelation at lag k is the k^{th} coefficient of a fitted AR(k) model; the `pacf` returns and plots this function, see Figure 6.59.

The previous two plots illustrate how short range correlations in an AR model have a long range impact on the values returned by `acf`, but an MA model does not have a long range impact, while the opposite behavior is seen in the values returned by `pacf`. An ARMA model always behaves in the most unhelpful way.

An ARMA model (*Autoregressive Moving Average*) is a combination of an AR and MA model, e.g., ARMA(2, 1) is the sum of an AR(2) and MA(1) model, such as the following:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + w_t + \theta_1 w_{t-1}$$

The `ARMAacf` function takes a specification of an ARMA model and returns what `acf` would return when passed a time series following this model (the `pacf=TRUE` option switches the behavior to that of `pacf`).

A time series is said to be *stationary* if the expected mean value does not change over successive measurements, i.e., $E[t_i] == E[t_{i+k}]$, or in conceptual terms the probability of events driving a stationary process do not change over time. The mathematics behind both the AR and MA models assume a stationary time series.

ARIMA (*Autoregressive Integrated Moving Average*) handles non-stationary time series (implemented by the `arima` function) is a technique for handling certain kinds of non-stationary time series.

Many software engineering processes include components that change over time, e.g., more developers working on a project, more customers, larger systems, etc. Time dependent components having a significant impact on measured values create in a non-stationary time series. A variety of techniques are available for analyzing non-stationary time series (including converting them to a stationary form).

Time series analysis is not limited to data involving time, it can be used for any data that contains serial correlation between measurements.

A study by Hindle, Godfrey and Holt[269] measured the indentation of the first non-whitespace character on a line, for code written in a variety of languages. Figure 6.60 shows the autocorrelation of a list, ordered by indentation, of the total number of lines having a given indentation.

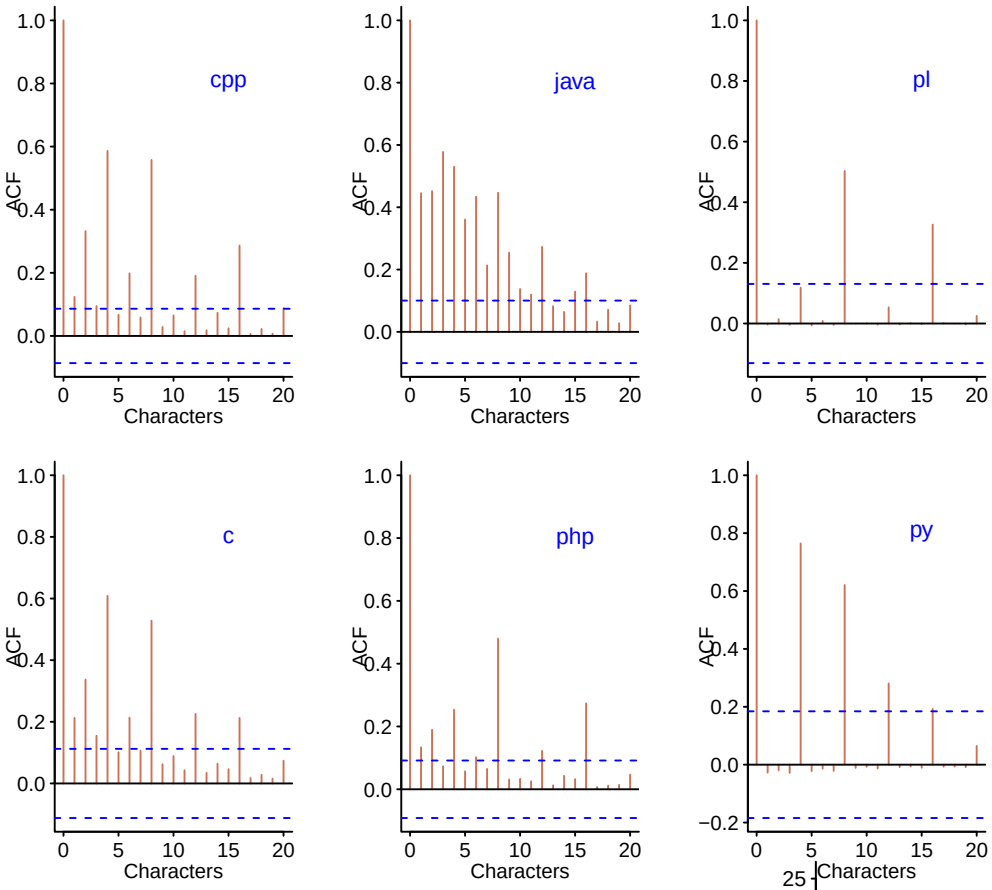


Figure 6.60: Autocorrelation of indentation of source code written in various languages. Data from Hindle et al.[269] code

6.9.2.1 Building an ARMA model

ARMA modeling takes as input a time series and a non-stationary time series needs to be converted to a stationary form before model building can begin. Common reasons for a time series not being stationary and possible transforms to a stationary series include:

- a non-zero trend: for instance, the following equation contains an increasing time dependent trend:

  $x_t = \alpha + \beta t + w_t$

  Differencing can be used to remove trends, but care needs to be taken because this can introduce signals that are not in the original data. For instance, differencing the above equation gives:

  $\Delta x_t = x_t - x_{t-1} = b + w_t - w_{t-1}$

  an MA(1) process which the original series does not contain.

  Subtracting the trend $\alpha + \beta t$ leaves just w_t (see Figure 6.61),

- seasonality: this is a cyclic trend, e.g., changes that recur every year. Implementations of ARMA often support for including a seasonal component in the model, e.g., the `seasonal` to the `arima` function,

- non-constant variance (known as *volatility* in the analysis of financial time series).

  If the growth in variance, over time, approximately follows the growth of the mean (perhaps there is a relatively consistent percentage change at each time step, e.g., $y_t = (1 + x_t)y_{t-1}$), then a log transform produces a time series with approximately constant variance (e.g., $\Delta(\log y_t) \approx x_t$).

  A log transform can only be applied when values are greater than zero; in the case of the 7digital data (which has an obviously increasing variance, as more developers are employed and time to implement features decreases) there are many zeroes and adding a tiny amount to every value allows a log transform to be made.

The plots produced by `acf` and `pacf` provide useful information about the likely structure and order of an ARMA model.

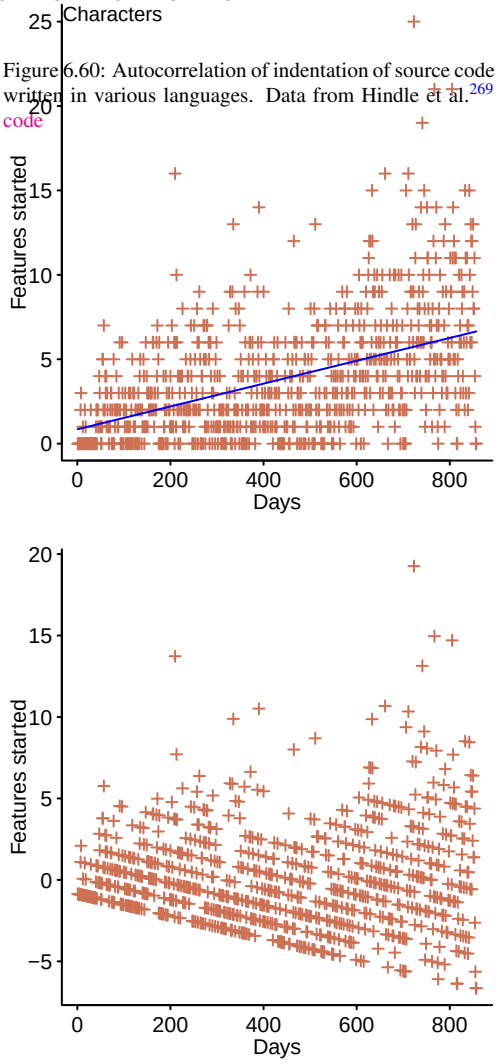


Figure 6.61: Number of features started for each day and fitted regression trend line (left) and number of features after subtracting the trend (right), over the entire period of the 7digital data. Data kindly supplied by 7Digital.[482] code

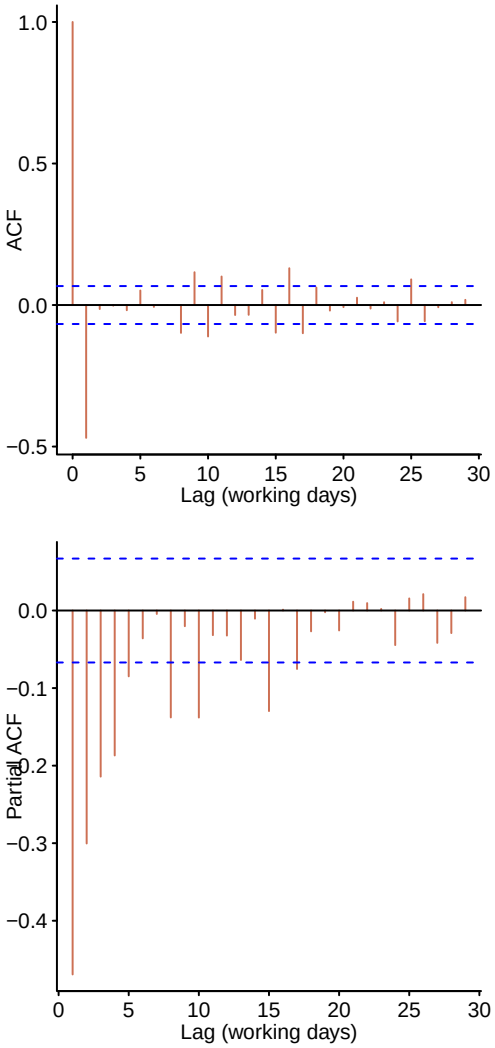


Figure 6.62: Autocorrelation (left) and partial autocorrelation (right) of the number of features started on a given day (after differencing the log transformed data), over the entire period of the 7digital data. Data kindly supplied by 7Digital.[482] code

- if the plot produced by `acf` shows a decreasing trend, while the `pacf` shows a sharp cut-off (see Figure 6.58), an AR model is a good place to start,
- if the plot produced by `acf` shows a sharp cut-off, while the `pacf` shows a decreasing trend (see Figure 6.59), an MA model is a good place to start,
- if both plots show a decreasing trend, then some combination of AR and MA model is likely to be needed.

```
# Add 1e-5 to handle zero values
acf(diff(log(weekdays+1e-5)), xlab="Lag (working days)")
pacf(diff(log(weekdays+1e-5)), xlab="Lag (working days)")
```

The `arima` function can be called with various settings of the `order` parameter, and the quality of fits compared using AIC (`arima` only returns the series mean when difference value of zero is passed to `order`; the `Arima` function in the `forecast` package is not limited in this way). The `auto.arima` function in the `forecast` package can be used to find ARIMA model values that minimise AIC.

```
arima(diff(log(weekdays+1e-8)), order=c(5, 0, 1))
auto.arima(weekdays, max.order=7)
arima(diff(log(weekdays+1e-8)), order=c(1, 0, 2))
```

The two models that best fit for the feature start data are ARMA(5, 1) and ARMA(1, 2).

The output from the last call to `arima` above is: code

```
Call:
arima(x = diff(log(weekdays + 1e-08)), order = c(1, 0, 2))

Coefficients:
         ar1      ma1     ma2  intercept
      0.8577  -1.6663  0.6663     0.0102
s.e.  0.0402   0.0579  0.0578     0.0022

sigma^2 estimated as 46.75:  log likelihood = -2862.8,  aic = 5735.59
```

The `Coefficients:` table lists the model coefficients and their standard error. The `intercept` column is actually the time series mean (which for a stationary series is zero). The equation for one of the models is:

$$x_t - 0.0102 = 0.8577(x_{t-1} - 0.0102) + w_t - 1.6663w_{t-1} + 0.6663w_{t-2}$$

which simplifies to:

$$x_t = 0.0102 \times (1 - 0.8577) + 0.8577x_{t-1} + w_t - 1.6663w_{t-1} + 0.6663w_{t-2}$$

with the constant increment per time step evaluating to 0.00145146.

Which of these two possible models provides the best explanation of the data? Features take different amounts of time to implement and work can only start on a new feature when enough people have been freed through completion of work on other features. The coefficients of the AR component of the ARMA(5, 1) model can be interpreted as a probability that people working on a feature started a given number of days earlier will become available to start work on a new feature (see Table 6.4).

	AR	Duration
ar1	0.19	0.32
ar2	0.11	0.16
ar3	0.09	0.11
ar4	0.07	0.07
ar5	0.10	0.05

Table 6.4: AR coefficients of ARMA(5, 1) model and percentage of features taking a given number of days to implement. Data kindly supplied by 7Digital.[482] code

This may be a just-so story, but stories are useful tools and your author cannot think of one for the ARMA(1, 2) model.

Handling seasonal trends A seasonal ARIMA model can include AR, difference and MA components at an offset equal to the number of measurement intervals in the season.

By default, the `auto.arima` function, in the `forecast` package, will return seasonal components (if any are found). The `seasonal` option can be used to specify seasonal components to the `arima` function.

The following code estimates a seasonal ARIMA model for hourly commits to the Linux kernel source tree (see Figure 6.56):

```
library("forecast")

hr_ts=ts(linux_hr, start=c(0, 0), frequency=24)

auto.arima(hr_ts)
arima(linux_hr, order = c(2,1,1), seas = list(order = c(1,0,1), period=24))
```

The coefficients of the fitted models (below) differ because of differences in the algorithms used, but are within each other's standard error: code

```
Series: hr_ts
ARIMA(2,1,1)(1,0,1)[24]

Coefficients:
          ar1      ar2     ma1    sar1     sma1
      -0.6985  -0.3524  0.2160  0.7811  -0.2838
s.e.   0.2235   0.0954  0.2523  0.0422   0.1085

sigma^2 estimated as 100511:  log likelihood=-1198.72
AIC=2397.22   AICc=2397.74   BIC=2415.93

Call:
arima(x = linux_hr, order = c(2, 1, 1), seasonal = list(order = c(1, 0, 1),
    period = 24))

Coefficients:
          ar1      ar2     ma1    sar1     sma1
      -0.8124  -0.2862  0.4483  0.8909  -0.4516
s.e.   0.2995   0.1177  0.3058  0.0546   0.1404

sigma^2 estimated as 129070:  log likelihood = -1229.02,  aic = 2470.03

Call:
arima(x = linux_hr, order = c(2, 1, 0), seasonal = list(order = c(1, 0, 1),
    period = 24))

Coefficients:
          ar1      ar2    sar1     sma1
      -0.3632  -0.1174  0.8980  -0.4727
s.e.   0.1007   0.0964  0.0519   0.1391

sigma^2 estimated as 129942:  log likelihood = -1229.71,  aic = 2469.42
```

The `sar1` is the seasonal AR coefficient and `sma1` the seasonal MA coefficient.

The output from `auto.arima` is a suggested model. In this case the `ar` and `sar` coefficients are pulling in opposite directions and the standard error for the `ma1` coefficient is very high. Removing the MA component produces a model (second call to `arima` above) where the coefficients are not almost cancelling each other out; the model is (24 is the seasonal period):

$$x_t = -0.4x_{t-1} - 0.1x_{t-2} + 0.9x_{t-24\times1} - 0.5w_{t-24\times1}$$

What happened 24 hours ago is a better predictor than what happened in the previous hour or two hours.

Predictions made using a fitted ARMA model A fitted ARIMA model can be used to predict the values likely to occur after a measurement at time t. However, the relatively large noise component present in some ARMA models means that the confidence bounds of the predicted values quickly become very wide.

Figure 6.63 shows how the uncertainty in the ARIMA model built from the 7digital data swamps the predicted values.

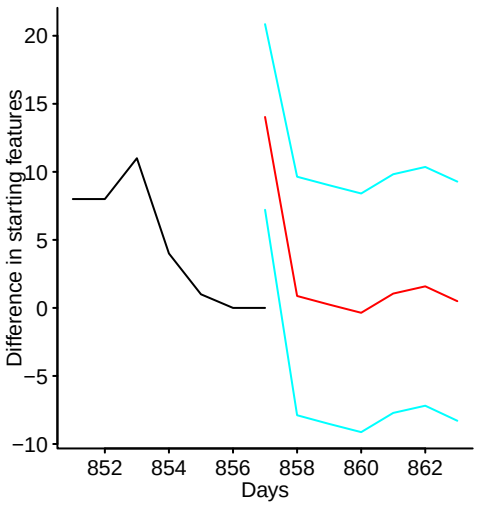


Figure 6.63: Predicted daily difference in the number of new feature starts (red) and 95% confidence intervals (blue). Data kindly supplied by 7Digital.[482] code

6.9.3 Non-constant variance

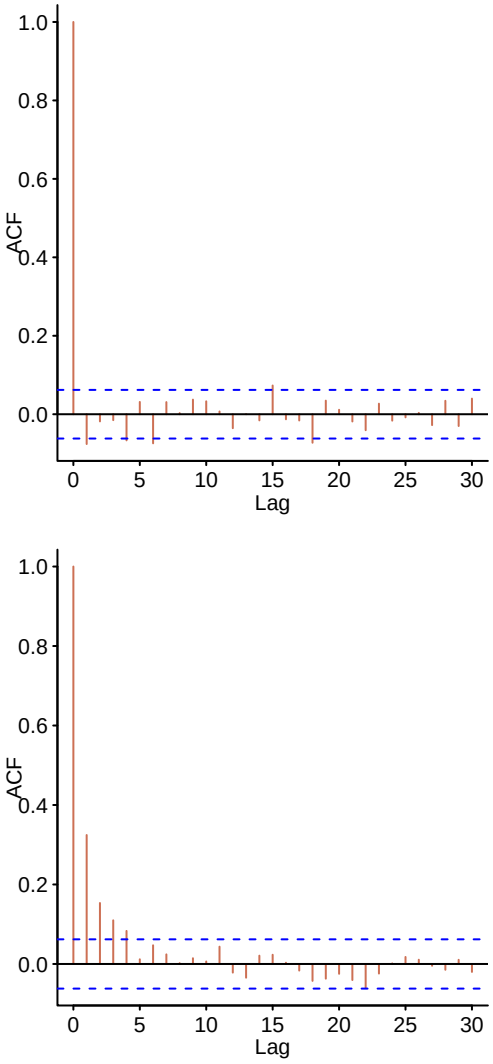


Figure 6.64: Time series whose values are uncorrelated (upper), but whose squared values are correlated (lower); see code for generation process. code

The variance of a time-series is primarily of interest when generating data for simulating processes. A time series that experiences rapid changes in variance is said to be *volatile*. Correlated variance is common during periods of volatility and techniques are available to build an autoregressive model for the variance, i.e. an *autoregressive conditional heteroskedastic* (ARCH) or a *generalised ARCH* (GARCH) model (a time series is *heteroskedastic* if the change in variance is regular and *conditionally heteroskedastic* if the change is irregular).

A time series experiencing changes in variance is not stationary.

An increase in frequency of commits leading up to a major new release is an example of behavior that can cause a change of variance in a time series.

The autocorrelation of a time series may show no correlation, but if its variance changes the square of the zero adjusted values will have a pattern of decreasing correlation in its acf, see lower plot of Figure 6.64.

```
acf(t_series)
acf((t_series-mean(t_series))^2)  # Check for changing variance
```

The `garch` function in the `tseries` package fits a GARCH model to data.

?

6.9.4 Long-memory processes

Correlation at high lags... just need the data... Fractal nature of LAN traffic... (is not SE)

Fractionally differences ARIMA processes (FARIMA)...

The `fracdiff` package...

6.9.5 Smoothing and filtering

Smoothing a time series can make it easier to visually identify larger scale patterns and also provides a simple approach to predicting the immediate future values.

Even when data does not contain a systematic trend or seasonal effects (perhaps because they have been removed), it may still be possible to make a good estimate of immediate future values based on immediate past values.

Smoothing using the *exponentially weighted moving average* (EWMA; also known as *exponential moving average*, EMA) uses the following formula:

$$EMA_t = \phi x_t + (1-\phi) EMA_{t-1}$$

where: x_t is the measured value of x at time t and ϕ determines the amount of smoothing. The *exponential moving standard deviation* (EMS) is given by:

$$EMS_t = \sqrt{\phi EMS_{t-1}^2 + (1-\phi)(x_t - EMA_t)^2}$$

EMA and EMS can be used to detect when a real-time data stream trends outside of prespecified bounds.

Holt-Winters smoothing is a generalization of exponential smoothing that uses three parameters: estimated level, slope and seasonality; the `HoltWinters` function can be used to both estimate and apply these parameters.

A call to `plot` will display the three components of a time series based on the value returned by the `HoltWinters` function... Seasonal component can vary...

The `filter` function can be used to apply a linear filter to a time series.

TODO some examples...

6.9.6 Missing data

Handling missing data in time series can be even more complicated and difficult than in regression modeling.

A study by Buettner[86] investigated project staffing and was not always able to obtain staffing information. Figure 6.65 shows a loess fit to the available data along with the standard error...

Some R functions support the use of splines for interpolation. Splines originated as a method for connecting a sequence of points by a smooth curve, not as a method of fitting a curve that minimises some error metric. Apart from their familiarity there is no reason to prefer the use of splines over other techniques (implementation issues also exist with the `bs` and `ns` functions in the `splines` package when building a model with the `predict.glm` function[588] when making predictions using new data points)...

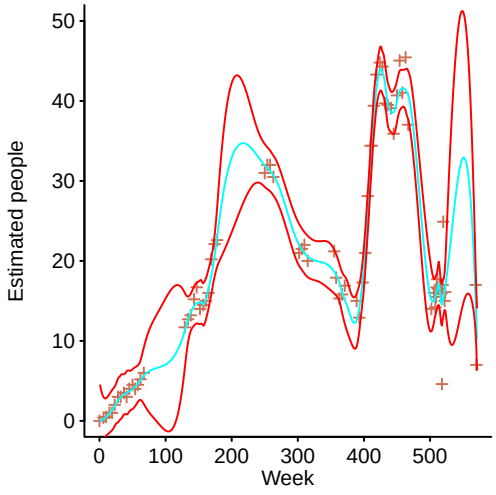


Figure 6.65: Estimated staff working on a project during every week. Data from Buettner.[86] code

6.9.7 Spectral analysis

It is possible to transform a series of measurements in the time domain into the frequency domain. A stationary time series does not contain components at specific frequencies, but it can be described in terms of an average frequency composition.

The `spectrum` function (default calls `spec.pgram` function)...

The `spec.arma` function takes a specification of an ARMA model and returns its power spectrum, i.e., behaves like a call to `spectrum` when passed a time series that follows this model.

6.9.8 Relationships between time series

Some of the relationships that can exist between two or more time series include:

Cross-correlation: The correlation, at various lags, between two stationary time series. Figure 6.66 shows the cross-correlation between the number of source lines added/deleted, per week, to the glibc library. In calls to the `ccf` function, the first argument is the one which is shifted. In the following call:

```
ccf(lines_added, lines_deleted, col=point_col, xlab="Weeks")
```

the plot produces shows correlation spikes well above the confidence bounds occurring between the sequence pairs $\texttt{lines\_added}_{t+2}$/$\texttt{lines\_deleted}_t$ and $\texttt{lines\_added}_{t+8}$/$\texttt{lines\_deleted}_t$ (changes involving `lines_deleted` is correlating with changes to `lines_added` two and 10 weeks later; a positive lag means the first argument follows the second, a negative lag that it is leads the second); there are small spikes at: $\texttt{lines\_added}_{t-8}$/$\texttt{lines\_deleted}_t$ and $\texttt{lines\_added}_{t-13}$/$\texttt{lines\_deleted}_t$. Your author has no explanation for this correlation behavior.

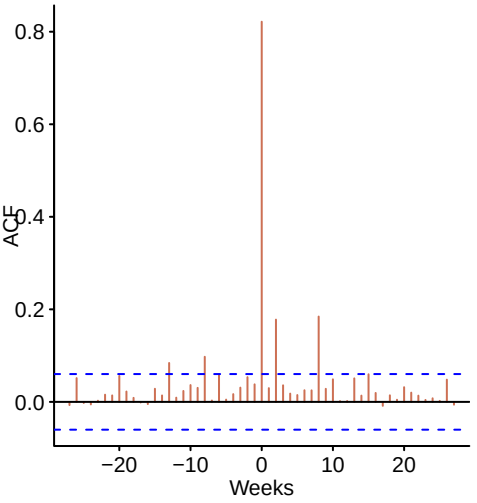


Figure 6.66: Cross-correlation of source lines added/deleted per week to the glibc library. Data from González-Barahona.[212] code

Alignment: A time series is a sequence of values, with each value being larger, smaller or equal to the value immediately before it. If two time series are generated by the same, or similar, process they may contain subsequences of values that share the same pattern of up, down and don't change. A non-time series application of this kind of subsequence matching is locating word sequences common to two documents.

Dynamic time warping (DTW) is a class of algorithms that compares two series of values by stretching or compressing one of them (treated as the reference series) so it resembles the other (treated as a query series). The `dtw` package contains functions to perform and support DTW alignment of two series.

A study by Herraiz[559] investigated the evolution of various long-lived software systems and measured the growth of NetBSD and FreeBSD (in lines of code). These two operating systems started from the same base, continue to share developers (see Figure 4.22) and code continues to be ported between them. Figure 6.67 shows the alignment, found by a call to `dtw`, between the weekly measurements of the lines of code in each OS.

```
library("dtw")

bsd_align=dtw(freebsd_weeks, netbsd_weeks, keep=TRUE,
              step=asymmetric, open.end=TRUE, open.begin=TRUE)
plot(bsd_align, type="twoway", offset=1, col=pal_col, xlab="Weeks")
```

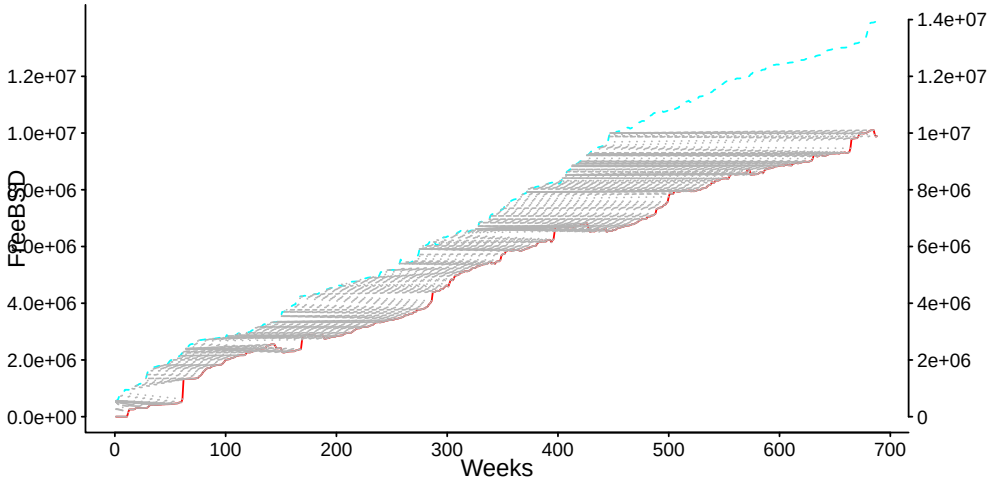
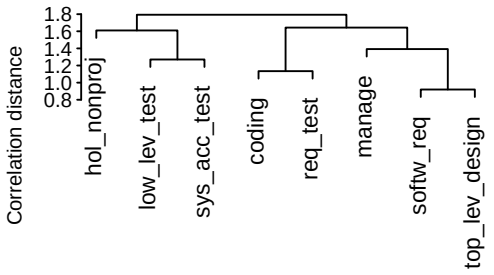


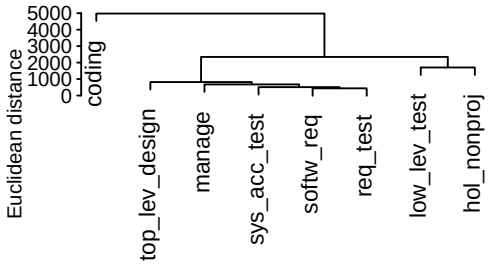
Figure 6.67: Visualization of alignment between weekly time series of lines code in NetBSD (blue) and FreeBSD (red). Data from Herraiz[559] code

Clustering: Many techniques for comparing two time series have been invented (at the time of writing the `diss` function, in the `TSclust` package, supports 22 distance metrics). The pair-wise similarity of multiple time series can be used to cluster them by. The `TSclust` package contains functionality for the clustering of time series.

A study by Powell[460] investigated task effort allocation in a development project at Rolls-Royce. Figure **??** shows effort (in person hours) spent on eight major tasks (lower, from the bottom up: s/w requirements, top-level design, coding, low level test, requirement test, system acceptance test, management and holiday/non-project) and a hierarchical clustering of each task by its effort time series, with pair-wise distance between time series calculated using correlation (upper) and Euclidean (middle) metrics.

```
library("TSclust")

eff_dist=diss(t(all_effort), METHOD="COR")
plot(hclust(eff_dist), main="", sub="", xlab="", ylab="Correlation distance")
```





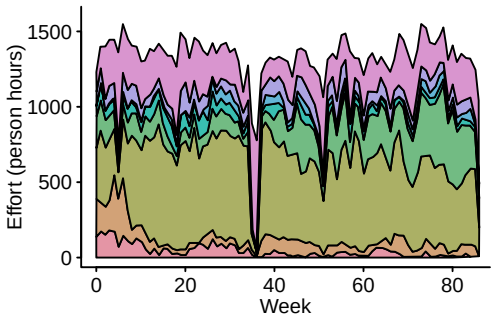


Figure 6.68: Effort distribution (person hours) over the eight main tasks of a development project at Rolls-Royce and a hierarchical clustering of each task effort time series based on pair-wise correlation and Euclidean distance metrics. Data extracted from Powell.[460] code

6.9.9 Regression models

The mathematics underpinning many regression modeling techniques requires each measurement to be independent of the other measurements in a sample. Time series data is often serially correlated.

Some regression modeling functions can adjust for the presence of serial correlation (information about the correlation is passed in an optional argument). The `gls` function, in the `nlme` package, supports a `correlation` option; the `dynlm` package supports the use of time series operators (e.g., diff and lag) in the specification of model formula; the `tscount` package supports the fitting of generalized linear models to time series of count data.

rexample[agile-week-acf.R]...

The number of source lines in FreeBSD is growing over time, see Figure 6.2...

6.9.10 Misc

Outlier detection...

Series containing irregular and regular processes?

The cpm package supports change point analysis of time series having a variety of distributions, e.g., exponential, poisson and the ecp package supports change point analysis of multivariate time series...

Granger causality...

6.10 Survival analysis

Survival analysis is the analysis of data where the response variable has the form of *time-to-event*. Historically this kind of model building has been used to compare the impact of different medical procedures, or drugs, on subject survival rate. In some cases the event of interest may not occur during the measurement period, the measurements in this case are said to be *censored*. A software example of censoring is measuring the time interval between a function definition being written and the first time it is modified; the measurement data is said to be *right censored* when one or more functions are not modified before the study ends.

By default, the analysis deals with one kind of event, which causes a transition to a terminal state, e.g., there is no coming back from the dead. Competing risk models deal with the situation where one of several risk events can cause the transition to the final state. Multistate models handle the situation where some transitions are to states that are not final, i.e., an appropriate event can cause a transition to another state.

Survival analysis makes greater use of the available information to produce estimates containing less error than other forms of regression modeling; a linear regression model comparing mean time-to-event between groups would have to ignore censored data, while a logistic regression model, using 0/1 to indicate whether a subject survived or not, would again have to ignore censored data.

Possible outputs from survival analysis include:

- survival function, $S(t)$, the probability of surviving a given amount of time, is used to estimate time-to-event for a group of subjects or compare time-to-event between subjects in two or more groups,
- hazard function, $h(t)$, the hazard rate, that is, the probability of an entity surviving to time t experiencing an event in the next time interval, e.g., having survived 69 years 11 months before reading this sentence the probability that you die in the next month (the interval used to denote an instant is small compared to the time spans involved). The survival and hazard functions can be derived from each other:

  $$h(t) = \frac{f(t)}{S(t)}$$

  where: $f(t)$ is a probability density function, the probability of the event occurring at exactly t time units in the future, e.g., the probability of a baby born 70 years ago living long enough to read this sentence but not before,
- a regression model showing the impact of explanatory variables on time-to-event. This may be a non-parametric model, such as the Cox proportional hazard model, because parametric models can be very difficult to build.

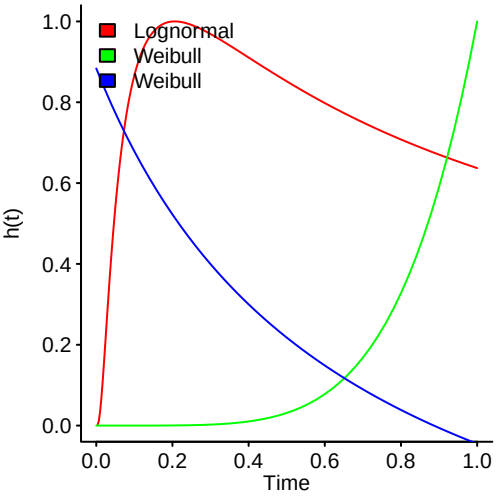


Figure 6.69: Two commonly used hazard functions; Weibull is monotonic (always increases, decreases or remains the same) and Lognormal which can increase and then decrease. code

Time-to-event is always positive and so has a skewed distribution (which means it cannot have a Normal distribution).

The survival package contains functions implementing the functionality needed to perform survival analysis.

6.10.1 Kinds of censoring

Ideally censoring is uninformative, i.e., the distribution of censoring times provides no information about the distribution of survival times. When a period of study is decided in advance, the censoring information is uninformative.

When censoring is not under the control of the experimenter, it is said to occur at random. For instance, a subject may decide to stop taking part in a study because they are not happy with their performance.

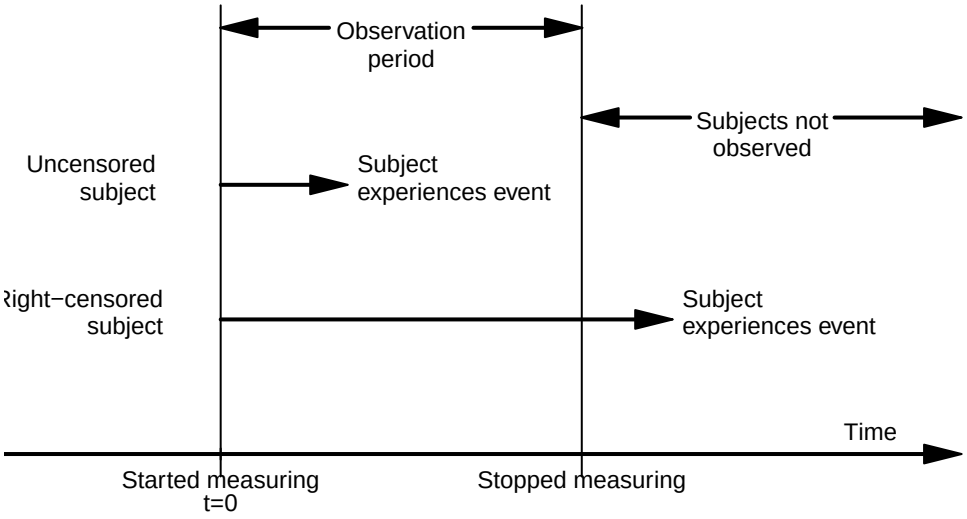


Figure 6.70: Observation period with events inside and outside the study period. code

The situations where censoring can occur include:

Left truncation: subject not observed before t_0 experienced an event before that time and is not included in the study (the event may have been such that it rendered the subject unable to join the study, e.g., developer left the company),

Left censored (also *left truncated*): occurs when a subject included in the study is known to have had the event prior to time t, but with the exact time not being known.

Right censored: described at the start of the subsection,

Interval censored: when measurements are made at regular intervals, the exact time of an event is not known, only that it occurred between two measurement points,

non-detect: the measurement process may fail to detect an event because the strength of the event is below the detection threshold. This kind of censoring is not covered here, see Helsel.[256]

6.10.1.1 Input data format

The `Surv` function creates a survival object from data and the object it returns plays the role of the response variable in formula passed to model building functions. The required data format depends on the kind of censoring and presence of time dependencies. The following is an example of the basic information required:

```
id,start_time,end_time,failure_status,explanatory_v1,explanatory_v2
```

where: `id` is a unique identifier denoting each subject (only needed when information on the same subject occurs on multiple lines), `start_time`/`end_time` the starting time (or date) of measurement and the end time (either when the event occurred, the end of the study or the last recorded time of a subject who was not seen again) and `failure_status` one of two values specifying whether an event occurred or not; followed by an optional list of explanatory variables.

The time of interest is the difference between the start/end time and the data may contain just this value.

6.10.2 Survival curve

The *Kaplan-Meier* curve is a descriptive statistic of time-to-event measurements, that can include censored data. It shows the percentage of subjects who have not experienced an event up to a point in time and an optional confidence interval.

A study by Businge, Serebrenik and van den Brand[90] investigated the number of releases of Eclipse third-party plug-ins (ETP) between 2003 and 2010; the history of each ETP was traced from the year of its first release and any releases in subsequent years were noted.

The Eclipse framework includes a published list of officially recognised APIs and each release of the Eclipse SDK also includes support for APIs considered to be for internal API use, i.e., not applications. The status difference between official/internal APIs is that internal APIs can be changed without notice, while the official APIs are intended to have some degree of permanence (they may change on major releases but are not intended to change on minor releases; starting in 2004 all yearly releases were minor).

At some point there are no new releases of an ETP in a year and this ceasation of new releases could be regarded as the *death* of development of the ETP (some ETP development died for one year only to be resurrected the following year; for simplicity the small number of such recurring events are ignored).

For this analysis ETP yearly release counts are divided into two groups, those that only made use of official APIs and those that made use of one or more internal APIs; Table 6.5 shows the number of ETPs using only the official API.

	2003	**2004**	**2005**	**2006**	**2007**	**2008**	**2009**	**2010**
2003	35	10	3	1	1	2	0	0
2004	0	33	4	4	2	2	0	0
2005	0	0	41	10	4	3	1	1
2006	0	0	0	61	7	1	0	2
2007	0	0	0	0	37	12	4	6
2008	0	0	0	0	0	38	7	2
2009	0	0	0	0	0	0	25	3
2010	0	0	0	0	0	0	0	16

Table 6.5: Total number of distinct ETPs released in a year; left column lists year of first release and releases in subsequent years. Data from Businge et al.[90]

Figure 6.71 shows the Kaplan-Meier curve for ETPs using only official APIs (blue) and ETPs that use internal APIs (red); the dotted lines are 95% confidence intervals.

The plot was created as follows:

- calling the `Surv` function to create a survival object containing time and censored information on each subject (this is the first step in most survival analysis when using R),
- calling the `survfit` function with a formula containing the object returned from `Surv` as the response variable and the explanatory variable `API`,
- calling `plot` (or rather the overloaded version) with the model returned by `survfit`:

```
library("survival")

api_surv=Surv(all_API$year_end-all_API$year_start,
                  event=(all_API$survived == 0), type="right")
api_mod=survfit(api_surv ~ all_API$API)
plot(api_mod, col=pal_col, conf.int=TRUE,  xlim=c(0,7), xlab="Years")
```

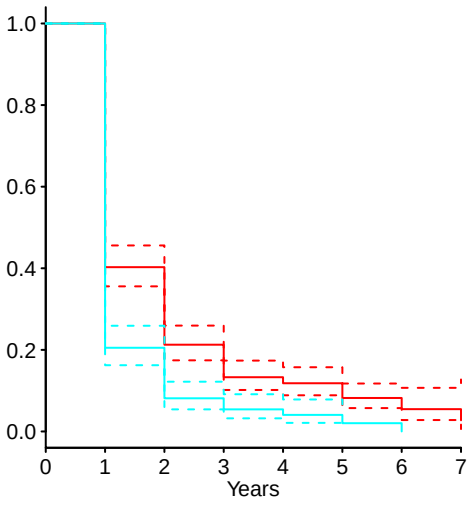


Figure 6.71: The Kaplan-Meier curve for survivability of new releases: (blue) ETPs using only official APIs, (blue) ETPs calling internal APIs (red); dotted lines are 95% confidence intervals. Data from Businge.[89] code

The `summary` function can be used to obtain the values of the survival curve at each time measurement point.

Comparing two survival curves Are the two survival curves statistically different? The `survdiff` function can be used to answer this question. The p-value returned by the call below (bottom right) shows that the two survival curves are very unlikely to be the same: code

```
Call:
survdiff(formula = Surv(year_end - year_start, event = (survived ==
    0), type = "right") ~ API, data = all_API)

        N Observed Expected (O-E)^2/E (O-E)^2/V
```

```
API=0 381      334      372      3.83       29
API=1 289      260      222      6.41       29

 Chisq= 29  on 1 degrees of freedom, p= 7.17e-08
```

By default, `survdiff` performs a *log-rank test*, which gives equal weight to all events. Passing the argument `rho=1` causes greater weight to be given to earlier events, while the argument `rho=-11` gives greater weight to later events. The hazard function is returned by `survfit` functions when is it passed the argument `type="fh"`.

Why, on average, do new releases of an ETP using internal APIs occur over a greater number of years? Is it because there are changes to the internal APIs that break the ETP, requiring the ETP to be updated to handle the change and a new version released, or is it because authors who use internal APIs are more committed to creating the best possible product and so continue to refine their ETP over more years?

Perhaps suspecting that changes to the SDK were a significant factor, Businge[89] investigated the source compatibility of ETPs with the Eclipse SDK across releases 1.0 to 3.7 (i.e., releases in every year from 2001 to 2011). Every ETP was built using each of these 11 SDK releases (yes, even SDKs created before an ETP was first released). To allow easy comparison with the ETP analysis above, the following analysis only considers SDK builds released after an ETP was first made available (see Table **??** for numeric values).

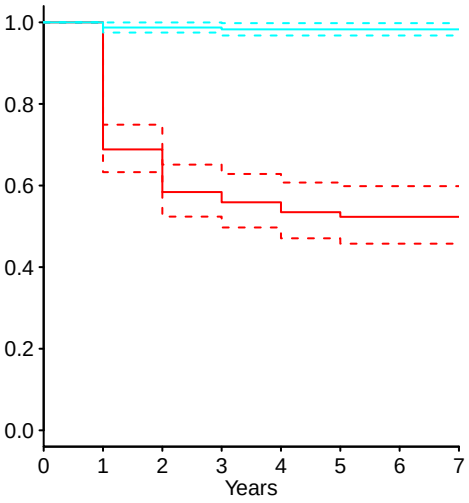


Figure 6.72: The Kaplan-Meier curve for survivability of ETPs ability to be built using SDK released in subsequent years: (blue) ETPs using only official APIs, (red) ETPs calling internal APIs; dotted lines are 95% confidence intervals, with plus signs, +, indicating censored data. Data from Businge.[89] code

The Kaplan-Meier curve in Figure 6.72 shows the survival of ETPs' ability to build under the Eclipse SDK released in each successive year. ETPs using internal APIs (red) are much more likely to fail to build (precompiled plug-ins may still function if they don't call any changed internal API) when a new Eclipse SDK is released than ETPs using only the official APIs (blue).

Figure 6.72 suggests that developers using internal APIs in their ETP are more likely to be forced to release an update if they want their ETP to continue to function with later releases. However, this data does not address the possibility that developers who make use of internal APIs are more committed to creating the best possible product.

The median is preferred over mean as the measure of central tendency for survival data, because the mean underestimates the true value when samples contain censored data. The median is measured as the point where the Kaplan-Meier curve falls before 0.5 and printing the model returned by `survfit` gives this value along with its 95% confidence intervals.

6.10.3 Regression modeling

Survival data implicitly contains information that is not present in ordinary regression modeling; the probability of an event occurring at a given time, the hazard function. Estimating the appropriate hazard function for survival data requires knowing the coefficients of the explanatory variables in the regression model, and estimating the coefficients of the explanatory variables requires knowing the hazard function.

Model building functions will attempt to model the shape of the hazard function that is specified and if the chosen hazard function is incorrect, the returned model may be substantially incorrect. Also, parametric models have been found to be very sensitive to the explanatory variables provided as input to the model building process.

There is no single statistic available for definitively selecting the best model (i.e., hazard function and appropriate explanatory variables).

The Cox proportional-hazards model does not require the specification of a hazard function, breaking the circularity in selecting regression coefficients and removing some of the dangers associated with use of an incorrect hazard function (the Cox modeling approach is not guaranteed to always build a reasonably accurate model). If there is any doubt about the appropriate parametric distribution, the Cox model is a safe choice.

While the Cox proportional hazards model has many advantages, a potentially big disadvantage is that without specifying a hazard function it is not possible to make predictions outside of the interval covered by the measurements.

The `censReg` package supports fitting regression models to censored data...

6.10.3.1 Cox proportional-hazards model

A Cox proportional-hazards regression model provides reasonably good estimates for the coefficients of the explanatory variables and hazard ratios (not absolute values, but ratios) for a wide variety of data. The Cox model is popular because it is robust, it will closely approximate the correct parametric model. If the correct parametric model has a Weibull hazard function (whose shape parameter is unknown), the Cox model will give similar results to those obtained from this parametric model; if the parameters of the Weibull hazard function are known, a model built using them will outperform a Cox model.

The Cox likelihood (known as a partial likelihood) is based on the observed order of events rather than the interval between them (so it only considers subjects' experiencing an event).

The equation for the basic Cox model is (note that time is not yet included as an explanatory variable, x_{ki}; the variables in this basic Cox model cannot be time dependent):

$$h_i(t) = h_0(t)e^{\beta_1 x_{1i}+\cdots+\beta_k x_{ki}}$$

where: $h_i(t)$ is the hazard function for subject i at time t, $h_0(t)$ is a baseline hazard function and the contents of the exponent expression are explanatory variables and their regression coefficients (β_0 is included as part of the baseline hazard).

This equation can be written as a log ratio of the hazard functions:

$$\log \frac{h_i(t)}{h_0(t)} = \beta_1 x_{1i} + \cdots + \beta_k x_{ki}$$

or as a hazard ratio for two subjects, i and j:

$$\frac{h_i(t)}{h_j(t)} = e^{\beta_1(x_{1i}-x_{1j})+\cdots+\beta_k(x_{ki}-x_{kj})}$$

Thus the Cox model assumes the effect of each explanatory variable is multiplicative.

The `coxph` function, in the `survival` package, builds Cox proportional-hazard models; the basic usage follows the pattern used by `glm`, with the object returned by `Surv` playing the role of the response variable. For example:

```
p_mod=coxph(Surv(patch_days, !is_censored) ~ log(cvss_score)+opensource,
                                                        data=ISR_disc)
```

The `cox.zph` function can be used to check the assumption that the explanatory variables are not time dependent (at least during the measurement period).

If two or more events occur at the same time the associated data is said to be *tied*. The default value of the option `ties="efron"` option, can handle some tied data, but if many events occur at the same time (e.g., the ETP data in Table 6.5) calls to `coxph` might need to use `ties="exact"`.

The techniques for formula specification and refinement used with `glm` can also be applied to models created with `coxph`, e.g., starting with a complicated model and using `stepAIC` to simplify it.

A study by Arora, Krishnan, Telang and Yang[27] investigated the time it took vendors to release patches to fix vulnerabilities reported in their product; explanatory variables included information about the software vendor and whether the vendor was privately notified about the vulnerability or the vendor first found out about it through a public disclosure.

The following is the `summary` output from a model fitted by `coxph` to the data for public disclosure vulnerabilities: code

```
Call:
coxph(formula = Surv(patch_days, !is_censored) ~ log(cvss_score) +
    opensource + y2003 + smallvendor + small_loge + log(cvss_score):y2002 +
    y2002:smallvendor + y2003:smallvendor, data = ISR_np)

  n= 945, number of events= 824

                        coef exp(coef) se(coef)      z Pr(>|z|)
log(cvss_score)      0.23283   1.26217  0.08570  2.717  0.00659 **
opensource           0.42235   1.52555  0.09167  4.607 4.08e-06 ***
y2003                0.83643   2.30811  0.10459  7.997 1.22e-15 ***
```

```
smallvendor           -0.40940   0.66405  0.17331 -2.362  0.01816 *
small_loge             0.02926   1.02969  0.01346  2.173  0.02975 *
log(cvss_score):y2002  0.23048   1.25920  0.04961  4.646 3.39e-06 ***
smallvendor:y2002      0.59685   1.81638  0.19540  3.054  0.00226 **
y2003:smallvendor      0.58999   1.80396  0.22502  2.622  0.00874 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

                      exp(coef) exp(-coef) lower .95 upper .95
log(cvss_score)           1.262     0.7923    1.0670    1.4930
opensource                1.526     0.6555    1.2747    1.8258
y2003                     2.308     0.4333    1.8803    2.8332
smallvendor               0.664     1.5059    0.4728    0.9326
small_loge                1.030     0.9712    1.0029    1.0572
log(cvss_score):y2002     1.259     0.7942    1.1425    1.3878
smallvendor:y2002         1.816     0.5505    1.2384    2.6640
y2003:smallvendor         1.804     0.5543    1.1606    2.8039

Concordance= 0.647  (se = 0.012 )
Rsquare= 0.19   (max possible= 1 )
Likelihood ratio test= 199  on 8 df,   p=0
Wald test            = 184.9  on 8 df,   p=0
Score (logrank) test = 198.3  on 8 df,   p=0
```

The first half of the output is very similar to the `summary` output produced by a model fitted by `glm`. The table of numbers in the middle are 95% confidence intervals, which are printed by default.

The numbers in the bottom part of the table are the R-squared of the fit[xxx] (0.19 in this case, showing that very little of the variance in the data is described by the model) and p-values for various tests of the null hypothesis that the coefficients are zero (abbreviated to a single letter, `p`).

The Cox model is a proportional-hazards model, the explanatory variable coefficients are proportions not absolute values. The coefficients specify the expected impact of the respective explanatory variable when the values of all the other variables are kept constant. On their own they cannot be used to predict response variable values, they can only be used to predict changes to known values.

Taking `log(cvss_score)` as an example, the value 1.26217 appears in its `exp(coef)` column. A ± 1 change in the value of $\log(cvss\_score)$ is expected to change the response variable (time taken to produce a patch) by $\pm(1.2621-1)\times 100 \rightarrow \pm 26.21$ percent (a value of less than one in the `exp(coef)` column reverses the sign of the percentage change, e.g., an increase in the value of the explanatory variable is predicted to decrease the value of response variable).

Model adequacy can be checked using Cox-Snell residuals and influential observations can be checked for using *score residuals*, which specify how each regression coefficient would change if a particular observation was removed (see rexample[patch-cph.R] for details).

Frailty of subjects The above analysis of time-to-patch has implicitly assumed that there is no difference between vendors in their ability to respond and fix reported vulnerabilities. Unobserved differences in subject performance means that there will be some variation in their hazard and *frailty* is the term used to denote these random changes in the hazard function. The effect of introducing the uncertainty implied by frailty, to a Cox model, is to add a random effect, v_j, the frailty of group j that x_i belongs to:

$$h_i(t) = h_0(t) v_j e^{\beta_1 x_{1i} + \cdots + \beta_k x_{ki}}$$

The `frailty` function can be included in a formula to specify explanatory variables that identify particular groups of subjects sharing the same frailty. In the case of the vulnerability study, vendors are treated as the frailty group:

```
fp_mod=coxph(Surv(patch_days, !is_censored) ~ log(cvss_score)+opensource
                                            +frailty(vendor), data=ISR_disc)
```

[xxx] The value printed is the Cox & Snell pseudo R-squared, which can be less than one and the maximum possible value for the data appears in the `summary` output.

The `summary` output includes the information: `Variance of random effect=0.374` (see rexample[patch-frailty.R] for details).

The v_j in the above equation is assumed to have a mean=1 and a variance that is calculated as part of the model building process (in this case it is 0.374). The main consequence of including frailty in a Cox model is to explicitly allocate some of the variance present in the data to a specific explanatory variable (the model coefficients of explanatory variables may also change).

The `frailtypack` package provides a wider range of frailty related options and functionality than is available in the `survival` package.

6.10.3.2 Time varying explanatory variables

The behavior of explanatory variables may change over time; the options are either to exclude all affected subjects from the analysis or to use a technique that handles the time dependent behavior

The Arora et al study investigated the impact of public disclosure of vulnerabilities on the time it took vendors to release patches for their product. Possible event sequences were:

- vendor was privately notified about a vulnerability and some time later a simultaneous announcement of the vulnerability and a vendor patch was made (213 of 755 private notifications),
- vendor was privately notified about a vulnerability, but information about the vulnerability was made public before a patch was available for release (the vendor's patch being released some time later in 542 of 755 private notifications); this is a time dependent change of a significant attribute.
- the vendor learned about a vulnerability when information about it was made public, and sometime later released a patch (945 cases),

If privately notified and public disclosure fix rates are compared using a Kaplan-Meier curve, any privately notified vulnerabilities that become public before a patch is available have to be treated as censored (simply ignoring them biases fix rates towards a lower value; see Figure 6.73).

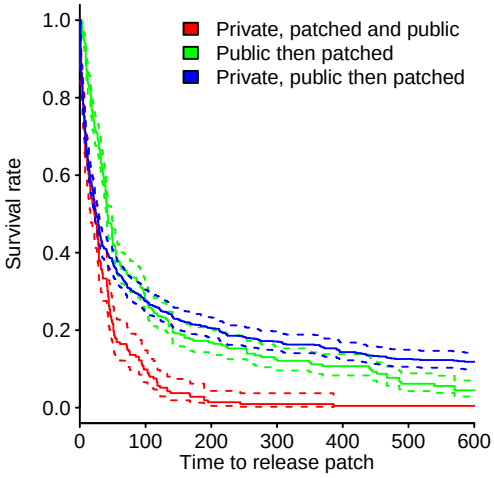


Figure 6.73: Kaplan-Meier curves for time-to-fix.... Data from Arora et al.[27] code

The first Cox model for the vulnerability data only used information for the case where the vendor found out about the vulnerability via public disclosure.

Building a regression model based on all the vulnerability data requires handling time dependent explanatory variables, which requires reformatting the data to make the time dependencies explicit. The time dependency, for this data, is a possible change of state from the vulnerability not being public to the information being public.

The original data looks something like the following:

```
notify,publish,patch,vendor,employee,os
2000-10-16,2000-11-18,2000-12-20,"abc",1000,unix
```

Publication of vulnerability information occurs before a patch is released and five columns are added, one to uniquely identify each vulnerability, the start/end dates the interval during which the information was private or disclosed, a flag each to specify private and disclosed, and whether an event (i.e., release of a patch) occurred in the interval. The first interval starts on the date the vendor was notified and ends on the date the vulnerability is made public, a second interval occurs for vulnerabilities that change state from private to disclosed before a patch is available and states on the date of disclosure and ends on the date a patch became available, as follows:

```
id,start,end,priv_di,notify,publish,patch,event,vendor,os
1,2000-10-16,2000-11-17,1,2000-10-16,2000-11-18,2000-12-20,0,"abc",unix
1,2000-11-18,2000-12-20,0,2000-10-16,2000-11-18,2000-12-20,1,"abc",unix
```

Treating `pr_di` as an explanatory variable (1 for private disclosure to vendor and zero for public disclosure) enables the impact of disclosure on patch time to be included in a model.

When all the measurement data has to be split on the same date, the `survSplit` function can be used to create the necessary rows, otherwise (as in this case) specific data mangling code has to be written.

The call to `coxph`, or `survreg`, has to include the term `cluster(id)`, which ties together (by vulnerability id in this case) the rows associated with the same subject. The call to `coxph` looks something like the following:

```
td_mod=coxph(Surv(patch_days, !is_censored) ~ priv_di*cvss_score
                                            +cluster(id), data=ISR_split)
```

It is not possible to use `cluster` and `frailty` in the same formula (`cluster` is based on GEE modeling building, while `frailty` is based on mixed-effects model building).

The `summary` output for the time dependent model is: code

```
Call:
coxph(formula = Surv(patch_days, !is_censored) ~ cluster(ID) +
    priv_di * (cvss_score + c_o + dis_by_s + os + y2 + smallvendor +
        small_loge) - c_o - dis_by_s - os - smallvendor + cvss_score:(c_o +
    dis_by_s + s_app) + opensource:c_o + opensource:dis_by_s +
    os:s_app + s_app:y2, data = ISR_split)

  n= 2242, number of events= 2081

                        coef exp(coef)  se(coef) robust se      z
priv_di              2.798750 16.424106  0.216150  0.209360 13.368
cvss_score           0.153926  1.166404  0.016806  0.017733  8.680
y2                   0.277421  1.319722  0.044042  0.044590  6.222
small_loge           0.037114  1.037811  0.007262  0.008817  4.210
priv_di:cvss_score  -0.114788  0.891555  0.017327  0.016795 -6.835
priv_di:c_o          0.644347  1.904743  0.228463  0.211989  3.040
priv_di:dis_by_s     0.475405  1.608665  0.116261  0.106601  4.460
priv_di:os          -0.331847  0.717597  0.098936  0.086976 -3.815
priv_di:y2          -0.614162  0.541094  0.063296  0.061954 -9.913
priv_di:smallvendor -0.440845  0.643492  0.138900  0.099310 -4.439
priv_di:small_loge  -0.082120  0.921161  0.016589  0.014449 -5.683
cvss_score:c_o      -0.060084  0.941685  0.012861  0.011990 -5.011
cvss_score:dis_by_s -0.061114  0.940716  0.008798  0.011002 -5.555
cvss_score:s_app    -0.096771  0.907764  0.014972  0.014853 -6.515
c_o:opensource       0.443978  1.558896  0.137952  0.118459  3.748
dis_by_s:opensource  0.414151  1.513086  0.091161  0.102359  4.046
os:s_app             0.815803  2.260991  0.077450  0.093536  8.722
y2:s_app             0.291007  1.337774  0.047420  0.045599  6.382
                    Pr(>|z|)
priv_di              < 2e-16 ***
cvss_score           < 2e-16 ***
y2                  4.92e-10 ***
small_loge          2.56e-05 ***
priv_di:cvss_score  8.22e-12 ***
priv_di:c_o         0.002369 **
priv_di:dis_by_s    8.21e-06 ***
priv_di:os          0.000136 ***
priv_di:y2           < 2e-16 ***
priv_di:smallvendor 9.03e-06 ***
priv_di:small_loge  1.32e-08 ***
cvss_score:c_o      5.42e-07 ***
cvss_score:dis_by_s 2.78e-08 ***
cvss_score:s_app    7.27e-11 ***
c_o:opensource      0.000178 ***
dis_by_s:opensource 5.21e-05 ***
os:s_app             < 2e-16 ***
y2:s_app            1.75e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Concordance= 0.654  (se = 0.007 )
Rsquare= 0.231   (max possible= 1 )
Likelihood ratio test= 590.1  on 18 df,   p=0
```

```
Wald test            = 376.9  on 18 df,   p=0
Score (logrank) test = 586.8  on 18 df,   p=0,   Robust = 454.6  p=0

  (Note: the likelihood ratio and score tests assume independence of
     observations within a cluster, the Wald and robust score tests do not).
```

There are two parts to the contribution made by `priv_di`; as a standalone variable it has a large impact, but its interactions with other variables create a large impact in the opposite direction (the model building process tries to minimise its error metric, not make it easy for us to understand what is going on).

The following is the component of the fitted equation of interest:

$$e^{priv\_di(2.8-0.11cvvs\_scorei+0.64c\_o+0.48dis\_by\_s-0.33os-0.61y2-0.44smallvendor-0.08small\_loge)}$$

where: *priv_di* is 0/1, *cvvs_score* varies between 1.9 and 10 (mean 7), *c_o* 0/1 NA other[xxxi] (mean 0.13), *dis_by_s* 0/1 disclosed by SecurityFocus (mean 0.38), *os* 0/1 vulnerability in O/S (mean 0.26), *y2* years since 2000 (mean 1.9), *smallvendor* 0/1 small vendor flag (mean 0.25) and *small_loge* zero for small vendors, otherwise the log of number of employees (mean 5).

Applying hand waving to average away the variables:

$$e^{priv\_di(2.8-0.11\times7+0.64\times0.13+0.48\times0.38-0.33\times0.26-0.61\times1.9-0.44\times0.25-0.08\times5)} \rightarrow e^{priv\_di(2.8-0.77+0.08+0.18-0.09-1.2-0.11-0.4)} \rightarrow e^{priv\_di\times0.49}$$

gives a (hand waving mean) percentage increase of $(e^{0.49}-1)\times100 \rightarrow 63\%$, when `priv_di` changes from zero to one. The percentage change for patches for vulnerabilities with a low `cvvs_score` is around 90% and for a high `cvvs_score` is around 13% (i.e., the patch time of vulnerabilities assigned a low priority improves a lot when they are publically disclosed, but patch time for those assigned a high priority is slightly affected).

The process of calculating the 95% confidence bounds, based on the values in the `summary` output, is fiddly and left to the reader.

Time dependencies can appear in various forms.

A study by Koru, El Emam, Zhang, Liu and Mathew[331] investigated the impact of class size, in LOC, on number of defects for ten products within the KOffice suite between April 1998 and January 2006. The following are some ways in which fault characteristics can change over time:

- the number of lines of code in files change as programs evolve. Every line of code added is a potential cause of a fault.

  The changing number of lines can be handled using the approach taken for the change of vulnerability visibility status. Each change in the number of lines in a class (or whatever the unit of measurement) appears as a separate row, with the rows for each class having the same id (specified in a formula as `cluster(id)`). The following data is for Kword:

  ```
  id,start,end,event,size,state
  204,0,163,1,31,1
  204,163,6372,1,29,2
  204,6372,11742,0,29,2
  204,11742,87259,0,32,2
  ```

- as existing faults are discovered the probability of discovering new faults decreases, i.e., the total number of faults is finite.

  To ensure consistency across classes the data is stratified by number of faults discovered. Given the wide variation in the number of faults discovered, from 0 to more than 25 per class, the number of cases in each stratification level would be small. Koru et al divided classes into four strata, containing 0, 1-5, 6-25 and >25 faults.

  Stratification variables are specified in a formula using the `strata` function, e.g., `strata(state)`.

  The term *recurring event* is used to describe situations where an event can occur more than once (and so is not a terminal event).

[xxxi] No information is available on what this variable represents.

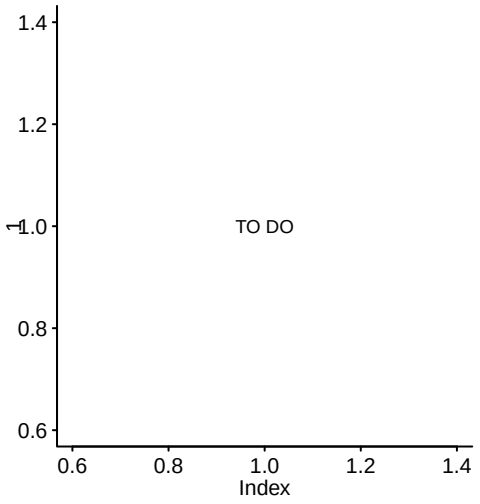


Figure 6.74: Survival curve after adjustment for explanatory variables... code

The following call to coxph builds a model for the Kword data (the log of lines of code has been found to provide a good fit in other contexts):

```
LOC_mod = coxph(Surv(start, end, event) ~ log(size)+strata(state)+cluster(id),
                                                    data=kw_data)
```

rexample[39_Koru_cox.R] TODO...

If size in LOC is a robust explanatory variable, its impact should be consistent across different component products of KOffice, with some variability due to differences in kind of program (frailty modeling...).

6.10.3.3 Parametric models

The first question that needs answering when building a parametric model is the form of the hazard function.

If the hazard is monotonic, i.e., continually increasing, decreasing or staying the same, then the Weibull function is the obvious first hazard function to try.

If the hazard function increases/decreases and then reverses to decreasing/increasing the Log-normal function is a reasonable first hazard function to try.

If the hazard function exhibits other growth patterns, then a piecewise approach...

rexample[patch-param.R]...

Accelerated failure time (AFT) and proportional hazards (PH) models...

6.10.4 Competing risks

When more than one possible kind of event can occur, i.e., there are multiple terminal states, a competing risk model can be used. Another way of handling this kind of data is to analyse each distinct event type separately from the other event types; data involving other events is marked as censored at the time other events occur.

The Kaplan-Meier plot for a single event, in a competing risk context, may give a misleading impression of the actual situation for events that rarely occur. The *cumulative incidence curve* (CIC) is a commonly used alternative that includes information on every event (when there is only one event, $CIC = 1 - KM$). CIC does not assume that competing risks are independent and estimates the marginal probability of an event.

The cmprsk package includes support for competing risk models.

A study by Di Penta, Cerulo and Aversano[145] investigated the history of problems in source code flagged by various static analysis tools. Newly written source code may contain a construct that is flagged and this code is tracked through subsequent versions. Possible competing events include the removal of the code containing the flagged construct and the flagged construct being modified such that it is no longer flagged (i.e., a bug fix).

Figure 6.75 shows the cumulative incidence curves (created by the cuminc function in the cmprsk package) for problems reported in the Samba and Squid source by the splint static analysis tool.

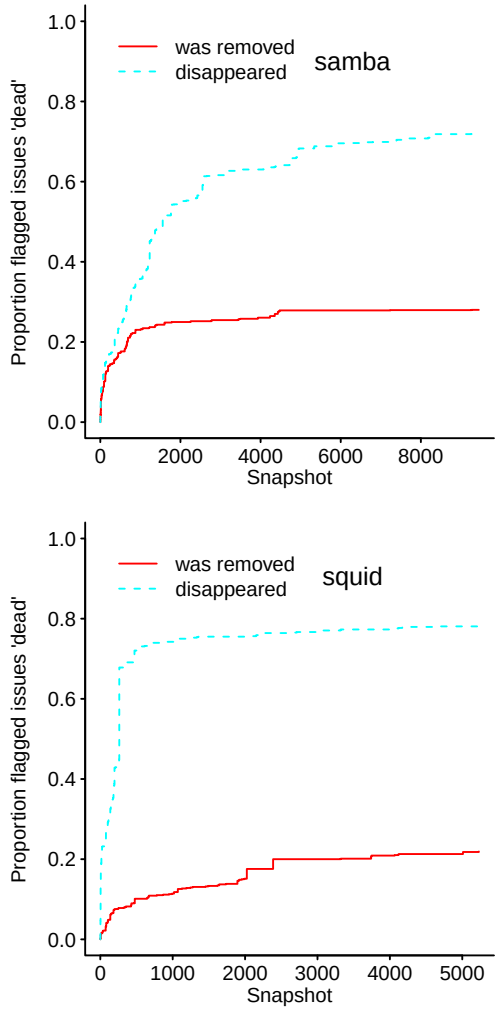


Figure 6.75: Cumulative incidence curves for problems reported by the splint tool in Samba and Squid (time is measured in number of snapshot releases). Data from Di Penta et al.[145] code

```
library("cmprsk")

plot_cif=function(sys_str)
{
t=cuminc(rats$failtime, rats$type, cencode=0, subset=(rats$SYSTEM == sys_str))

plot(t, col=pal_col, cex=1.25,
       curvlab=c("was removed", "disappeared"),
       xlab="Snapshot", ylab="Proportion flagged issues 'dead'\n")

text(max(t[[1]]$time)/1.5, 0.9, sys_str, cex=1.5)
}

plot_cif("samba")
plot_cif("squid")
```

Comparing competing risks...

6.10.5 Multistate models

Multistate models deal with time to event processes that involve multiple events and potentially changes of state between them... TODO

The `msm` package...

?

5,855 fault records from Chinese software house... used in?

Multiple licenses that software can switch to using...?

Function creation, zero or more modifications and possible deletions. function lifetime...

6.11 Structural Equation Models

?

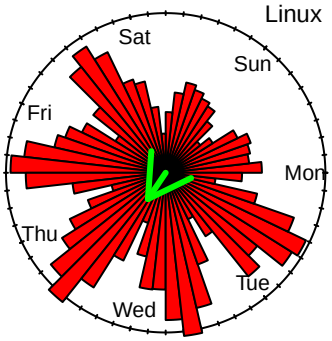
6.12 Circular statistics

Some measurements use a circular scale, with values wrapping back to the minimum value when incremented past the maximum value, e.g., time of day or days of the year. Circular statistics[450] is the name given to the analysis of data measured using such a scale. throughout this subsection. Circular statistics has only started to be more widely studied in the last 40 years or so and techniques for handling operations that are well-established in other areas of statistics are still evolving. Functions from the `circular` package are used

Differences between measurements on a circular and linear scale include the following:

- plotting uses a polar representation, rather than x/y-axis (the `circular` package includes support for the `plot`, `lines`, `points` and `curve` functions),
- the mean, if it exists, has two components: mean direction ($\overline{\theta}$, an angle) and mean resultant length ($\overline{R}$), returned by the `mean` and `rho.circular` functions respectively (the `trigonometric.moment` function provides another way of obtaining this information). The `median.circular` function returns a median (multiple medians may exist, but only one is returned),
- the term variance, on its own, is ambiguous. The *circular variance*, V, is defined as $V = 1-\overline{R}$ and varies between zero and one. Another measure is *angular variance* (returned by the `angular.variance` function) which varies between zero and two.

  The *circular standard deviation* is returned by the `sd.circular` function (it is not calculated by taking the square root of the variance; its formula is: $\sqrt{-2\log\overline{R}}$),
- the von Mises distribution plays a role similar to that filled by the Normal distribution on linear measurement scales.



The mean resultant length, $\overline{R}$, is a measure of how spread out data points are around the circle. If the points have a symmetric distribution $\overline{R}$ equals zero and if all the points are concentrated in one direction $\overline{R}$ equals one; for unimodal distributions the term *concentration* is applied to $\overline{R}$ to denote the extent to which measurements concentrate around the mean direction.

Figure 6.76 is a Rose diagram of the number of commits to Linux and FreeBSD for each 3 hour period of the days of the week (the same data is plotted using a linear scale in Figure **??**).

The `rose.diag` function plots Rose diagrams. By default, the area of each segment is proportional to the number of measurement points in the segment (the behavior used when plotting histograms).

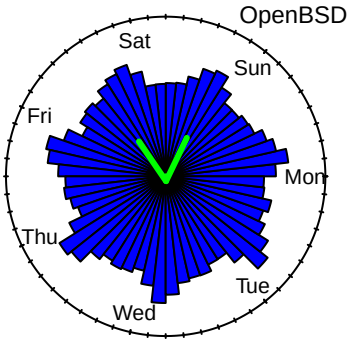


Figure 6.76: Rose diagram of number of commits in each 3 hour period of a day for Linux and FreeBSD. Data from Eyolfson et al.[169] code

```
library("circular")

# Map to a 360 degree circle
HoW=circular((360/hrs_per_week)*week_hr, units="degrees", rotation="clock")
rose.diag(HoW, bins=7*8, shrink=1.2, prop=5, axes=FALSE, col=col_str)
axis.circular(at=circular(day_angle, units="degrees", rotation="clock"),
```

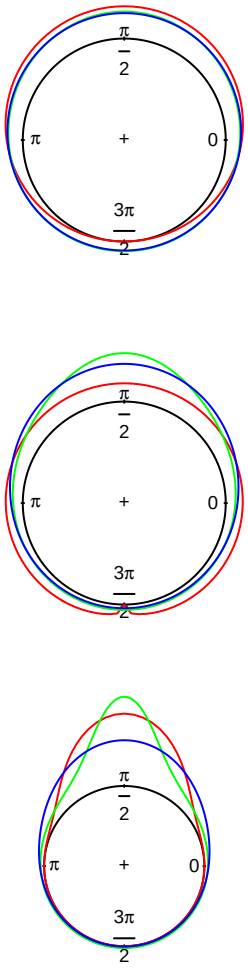
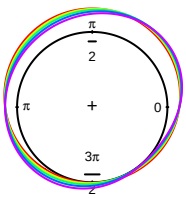
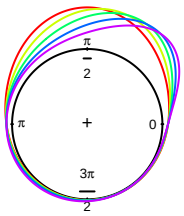


Figure 6.77: The Cartwright (red; `dcarthwrite`), wrapped Cauchy (green; `dwrappedcauchy`) and wrapped von Mises (blue; `dvonmises`) circular probability distributions for various values of their parameters. code





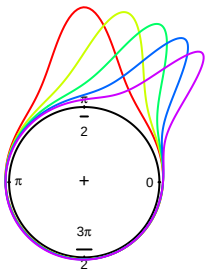


Figure 6.78: Asymmetric extended wrapped forms of the Cardioid (upper), von Mises (middle) and Cauchy (lower)

```
        labels=c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"))

text(0.8, 1, repo_str, cex=1.4)
arrows.circular(mean(HoW), y=rho.circular(HoW), col=pal_col[2], lwd=3)
```

The arrow at the center shows the direction of the mean and the length of its shaft is its resultant length. Linux has fewer commits at weekends, compared to weekdays and a mean direction near the middle of the week looks reasonable. The number of commits to FreeBSD does not seem to vary between days; the mean length is 0.03 (it almost does not have a mean), compared to Linux's mean length of 0.2.

If the measurement scale is very course (e.g., measuring commit time to an accuracy of day rather than hour or minute), then $\overline{R}$ will be underestimated and introduce errors in the calculation of the various location measures which use this value (see rexample[circle-bin.R]). The suggested correction to $\overline{R}$ for calculating circular standard deviation, when measuring in units of days rather minutes, is to use the calculated value of $\overline{R}$ multiplied by 1.034 (calculating higher order moments involves much larger values).

6.12.1 Circular uniformity

The *continuous circular uniform distribution* is the fundamental circular model; for this model no direction is any more likely than another. The `dcircularuniform` and `rcircularuniform` functions, but not `p` and `q` forms, are supported by the `circular` package.

The choice of circular uniformity test to use for measurements on a continuous scale, i.e., many possible measurement points around the circle, depends on how the data is thought to possibly deviate from uniformity. The two uniformity deviation possibilities are:

- a single peak over some range of values, i.e., a unimodal distribution. In this case the Rayleigh test is the most powerful known test, available in the `rayleigh.test` function,
- multiple peaks in the distribution of values around the circle. There are three tests that are more powerful than the Rayleigh test when the data distribution could be more complicated than a single peak, but no single one is superior to the others; available in the `kuiper.test`, `watson.test` and `rao.spacing.test` functions.

Unless there is a good reason to think that the measurements could have a single peak, one (or all) of the omnibus tests should be used.

When the measurements have been grouped into a few bins, e.g., months of the year, a grouped data test has to be used...

Bootstrapping rexample[grp-data-boot.R]...

Figure 6.77 shows how the shapes of three popular symmetrical single peak wrapped circular distributions differ from each other.[xxxii] The *Jones-Pewsey distribution* includes all of them, and others, as special cases.

Figure 6.78 shows asymmetric extended forms of some common circular distributions. The `circular` package does not include support for asymmetric distributions, but code is available in Pewsey et al.[450]

Compiler writers born in February... TODO

A study by Eyolfson, Tan and Lam[170] investigated the correlation between commit time and the likelihood of a fault being detected in the commit, for Linux and PostgreSQL. Figure 6.79 shows the number of non-fault commits (upper) and number of commits in which a fault was detected (lower), made in each hour of combined weekdays (the pattern of commits on weekdays differs from weekend days and the following analysis is based on weekdays only).

What differences, if any, exist between the two sets of daily commit times and in particular are commits made at certain times of the day more likely to have a fault detected in them?

[xxxii] The implementation of the Cartwright distribution, up to version 0.4.7 of the `circular` package, uses the spelling `carthwrite`.

- testing for a common mean direction: The `watson.williams.test` function, in the `circular` package, assumes that both samples are drawn from a von Mises distribution; the *Watson large sample non-parametric test* does not even require the samples to share a common shape (see rexample[common-mean.R]). When any of the samples has a size less than 25, a bootstrap version of these tests should be used. The daily commit times do not share a common mean direction (15.5 hours for fault commits and 16.2 hours for non-fault commits); the mean result lengths are 0.33 and 0.32 respectively,

- testing for a common concentration: Are the points concentrated around a common direction? The *Wallraff test* is not supported by the `circular` package, but is described in Pewsey et al[450] (see rexample[common-concen.R]). The two commit samples do not share a common concentration.

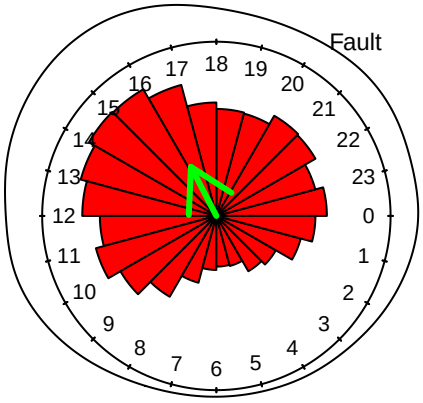


Figure 6.79: Number of commits (upper) and number of commits in which a fault was detected (lower) by hour of day of the commit, for Linux. Data from Eyolfson et al.[170] code

6.12.2 Fitting a regression model

When one or more variables are measured on a circular scale the technique used to build a regression model depends on whether the circular variable is an explanatory or response variable.

When the response variable is measured on a linear scale, existing techniques and functions can be used; there may be one or more circular or linear explanatory variables,

When the response variable is measured on a circular scale, the `lm.circular` function, in the `circular` package, can be used

6.12.2.1 Linear response with a circular explanatory variable

Circular explanatory variables can be modeling using periodic functions and the regression modeling techniques discussed in earlier sections. The sine and cosine functions can be combined to model any periodic function. As always, a model containing the fewest number of distinct parameters is desired.

The cosine function can be modified in various ways to change its shape:

$y = \alpha + \beta \cos(\omega x + \phi)$

higher order harmonics can be added:

$y = \alpha + \beta_1 \cos(\omega x + \phi) + \beta_2 \cos(2\omega x + \phi) \cdots$

The shape of the peaks and troughs can be modified by adding a sine wave to the angular argument. In the following a positive λ sharpens the peaks and flattens the troughs while a negative λ has the opposite effect.

$y = \alpha + \beta \cos(\omega x + \phi + \lambda \sin(\omega x + \phi))$

a skewed period (which is what asymmetrical distributions have) can be modeled by adding a cosine wave to the angular argument (provided $-\pi/6 \leq \lambda \leq \pi/6$, outside this range it also effects other shape characteristics):

$y = \alpha + \beta \cos(\omega x + \phi + \lambda \cos(\omega x + \phi))$

These are all non-linear equations and the `nls` function can be used to fit them.

The commit data is obviously asymmetrical and the following code fits an extended cosine regression model (the `gam` values were estimated from the height of the cycle and `omega` from fitting 24 hours into 2π radians):

```
basic_mod = nls(freq ~ gam0+gam1*cos(omega*hour-phi+nu*cos(omega*hour-phi)),
                        start=list(gam0=800, gam1=700,
                                   omega=0.3, phi=1, nu=0),
                data=week_basic)
```

The upper plot in Figure 6.80 shows the number of non-fault commits per hour for every week day and the fitted model, the lower plot shows the commits with detected faults.

Both fits handle the skewed period but not the sharp peak and flat trough. A sine contribution can be added to help handle this shape and improve the fit, the call to `nls` is below:

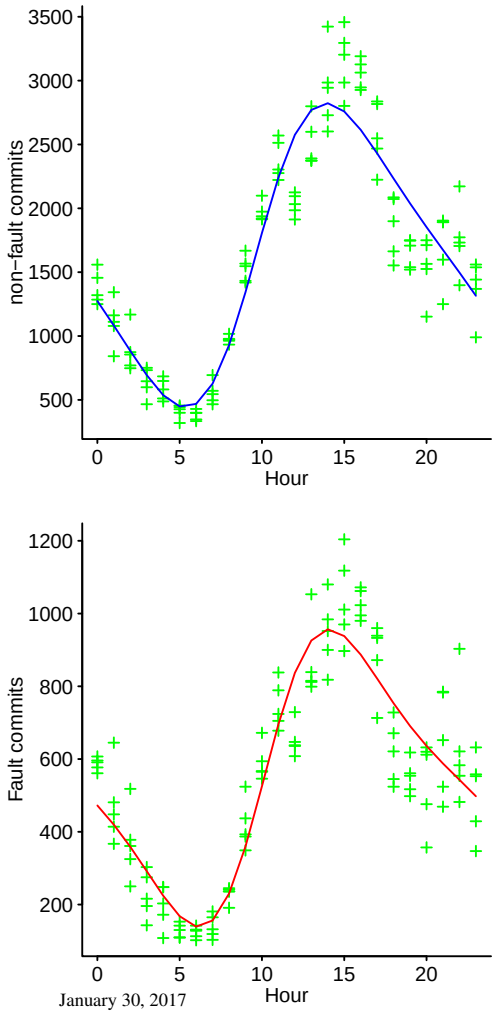


Figure 6.80: Number of commits per hour for weekdays and fitted model (upper) and number of commits in which a fault was detected (lower) for Linux. Data from Eyolf

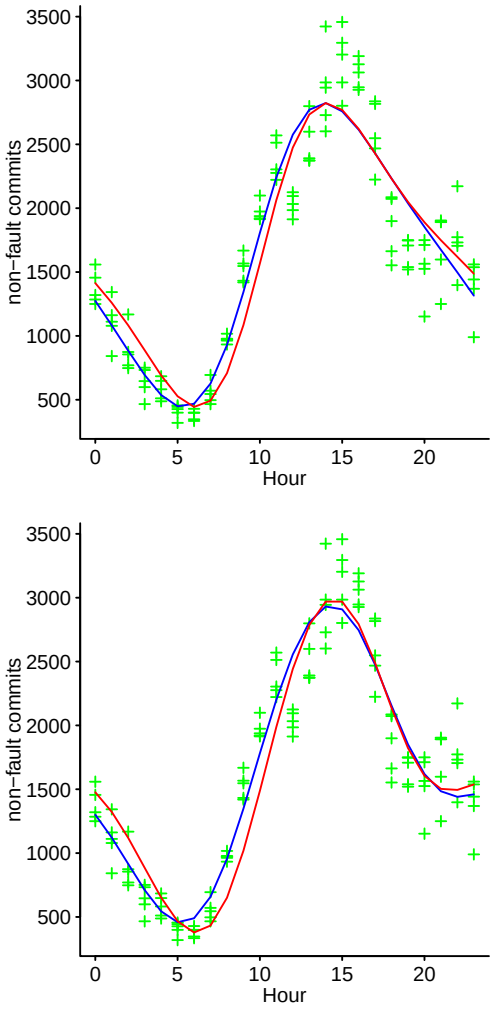


Figure 6.81: Number of commits per hour for each weekday, fitted using cos(...cos...) (upper) and cos(...cos+sin...) (lower), for Linux; in both cases the fitted fault model (red) has been rescaled to allow comparison. Data from Eyolfson et al.[170] code

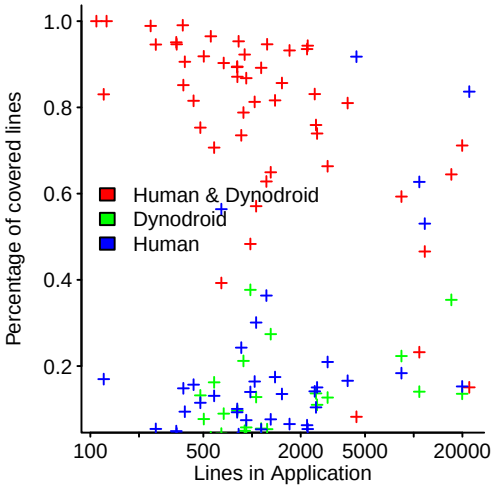


Figure 6.82: Application source lines against percentage of covered lines achieved by both Human & Dynodroid tests, by only by Dynodroid tests and only by Human tests. Data from Machiry et al.[368] code

```
basic_2mod = nls(freq ~ gam0
                        +gam1*cos(omega*hour-phi+nu*cos(omega*hour-phi))
                        +gam2*cos(2*omega*hour-phi+nu*sin(omega*hour-phi)),
                             start=list(gam0=800, gam1=700, gam2=100,
                                        omega=0.3, phi=1, nu=0),
                 data=week_basic)
```

Figure 6.81 overlays the fitted curve for non-fault and fault (red) commits over the non-fault hourly commits for each workday.

Confidence intervals...

6.12.2.2 Circular response variable

The `lm.circular` function supports circular response variables... if only we had some data...

6.13 Compositions

Compositional data is made up of components whose total contributions sum to 100 (or 1). The requirement of a total creates a correlation between the components, i.e., if one of them increases one or more of the others has to decrease. When this mutual correlation between variables is not taken into account, models with surprising can be fitted to data.

The theory needed to underpin techniques for handling compositional data became available at the start of the century; it is all very new and many issues are still unsolved.

The analysis in this section is based on the book by Boogaart and Tolosana-Deldago.[582]

A study by Machiry, Tahiliani and Naik[368] measured the performance of two application test generators by comparing the number of lines of program source code covered by the tests generated by each tool (50 Android apps were tested); human performance was also measured.

The application source lines covered by human and tool generated tests was recorded. The difficulty of creating tests to cover source lines is likely to vary across applications and within different parts of the same application. Normalizing coverage counts, to a percentage of source lines covered, allows performance across different applications to be compared.

Figure 6.82 shows that as the number of application source lines increases, coverage common to human and Dynodroid written tests decreases (measured as a percentage of all covered lines).

One measure of human vs. tool performance is to compare just those source lines that are covered by tests. What percentage, for each application, is covered by both human and tool generated tests and the percentage uniquely covered by human or tool tests? Figure **??** shows this information for the Dynodroid tool (the red tick marks on the axis are measurement points where one of the three components is zero), along with a (poorly fitting green line) regression line.

The clustering of points near the Human & Dynodroid vertex shows that tests created generated by these generators tend to cover the same source lines. More points are near the Dynodroid axis than the Human axis, suggesting that Dynodroid generated tests cover fewer unique source lines.

Fitting three regression models, one for each coverage percentage, using application source lines as the explanatory variable fails to make use of all the available information, i.e., the relationship between the three percentages.

A method of combining the three percentages into a single entity, that can be used as a response variable is required. The *isometric log-ratio transformation*, ilr, is one possibility and the `compositions` package supports the `ilr` function.

The `acomp` function normalises the listed columns using a ratio scale and returns an object having class `acomp` (named after Aitchison who pointed out the useful mathematical properties that a ratio scale bring to compositional analysis).

The `ilr` function is not currently handled by `glm`, so `lm` has to be used. Understanding the rest of the code requires a lot more background knowledge than is appropriate here; see Boogaart and Tolosana-Deldago[582] for more details.

```
library("compositions")

covered=acomp(dh, parts=c("LOC.covered.exclusively.by.Dyno..D.",
                          "LOC.covered.exclusively.by.Human..H.",
                          "LOC.covered.by.both.Dyno.and.Human..C."))

plot(covered, labels="", col=point_col, mp=NULL)
ternaryAxis(side=0, small=TRUE, aspanel=TRUE,
                Xlab="Dynodroid", Ylab="Human", Zlab="Human & Dynodroid")

dh$l_total_lines=log(dh$Total.App.LOC..T.)

comp_mod=lm(ilr(covered) ~ I(l_total_lines^2), data=dh)

d=ilrInv(coef(comp_mod)[-1, ], orig=covered)
straight(mean(covered), d, col="green")
```

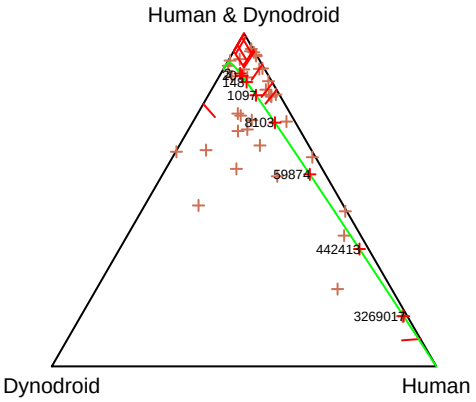


Figure 6.83: Percentage of source lines covered by both Human & Dynodroid tests, by only by Dynodroid tests and only by Human tests; fitted regression line and prediction points for various total source lines, red plus. Data from Machiry et al.[368] code

The explanatory variable is total source lines in the application and the red plus signs show predictions for various totals. The quality of the fit is very poor, with potentially many outliers and non-constant variance. Model building can only use the explanatory variables present in the data.

?

6.14 Extreme value statistics

Extreme value statistics deals with values that rarely occur during the normal operation of a system...

?

?

?

Chapter 7

Other techniques

7.1 Machine learning

Building a regression model requires an investment of somebodies time and expertise, potentially a lot of time.

Machine learning can often find patterns in large datasets with relatively little upfront investment of people time. The problem with models built using machine learning is often the difficulty of interpreting their behavior, i.e., they are designed for prediction performance, not ease of understanding.

The machine learning approach to model building is ideal for clueless button pushers, and being clueless about a data set that needs to be analysed happens to us all from time to time. The only input to a machine learning approach is the data,[i] there is no need to specify a functional form for the relationship between explanatory variables.

Possible uses for machine learning include:

- a filter that can quickly find the subset of important explanatory variables in a dataset containing very many variables. Once located, this subset can be fed into the regression modeling processes discussed earlier. This is essentially a fishing expedition and there will always be some subset that is better than the others. Cross validation needs to be used to check that the variables could have explanatory behavior with a different dataset,
- to build models for data whose characteristics regularly change so much that existing models become unusable.

A sample containing information about very many variables is useful to have when it is possible to use domain knowledge to select an appropriate subset. However, when all the variables are included in the analysis at the same time the *curse of dimensionality* is invoked by the fundamental mathematics often used to build models.

A common metric used by machine learning algorithms is the distance between points. Each measurement can be viewed as a point in an n-dimensional space, where n is the number of attributes associated with each measured item. For ease of comparison in the following analysis this n-dimensional space is normalised so that every side has length one, its volume is also one. In 3-dimensions the volume of a sphere of diameter one is $\frac{4}{3}\pi 0.5^3 \rightarrow 0.52$, that is the sphere occupies 52% of the unit cube. If the unit cube contains multiple points, then there is a 52% probability that a point at the center of the unit cube will be within 1-unit distance of another point. As the number of dimensions increases the sphere/unit cube volume ratio increases to a peak at five dimensions and then decreases rapidly. Figure 7.1 shows how the volume of a sphere changes as the number of dimensions increases.

As the number of dimensions increases the distance from a point to the point nearest to it approaches the distance to the point furthest from it;[57] an effect that can occur for as few as 10-15 dimensions. This behavior means that any algorithm relying on distance between points effectively ceases to work at higher dimensions.

[i] Implementations often support a variety of tuning parameters and the cost of ignoring them is often cpu time.

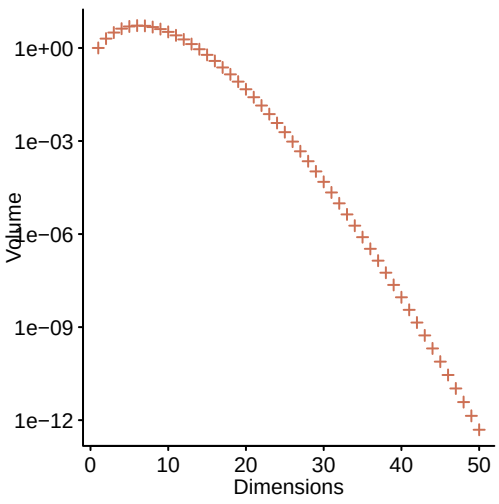


Figure 7.1: Volume of unit sphere in 1 to 50 dimensions, e.g., sphere has volume $\frac{4}{3}pi$ in three dimensions. code

7.1.1 Decision trees

As the name suggests decision trees have the form of a tree like structure, rather than that of an equation. Each node of the tree contains either an expression whose result is used to select which of the node branches to follow or a value denoting the result. The commonly used `rpart` package supports the creation of binary trees; the Weka machine learning system[619] supports nodes containing more branches and can be accessed from R using the `RWeka` package.

Tree models are popular in disciplines where they can be interpreted by people who need to make a decision based on what they observe, e.g., Doctors.

The predictions made by decisions tree models are generally not as accurate as other types of models and they do not use continuous variables effectively. But these are not important issues when the aim is to gain a better understanding the data.

The condition at each node involves one variable and a set of one or more constants. The binary value of the relationship selects which branch to go down to the next node (the displayed tree goes left when the condition is true and right when false), with the process continuing until a leaf node is reached.

The `rpart` function decides whether a leaf node should be split into a condition node and two leaf nodes using a method known as *cost complexity pruning*; any node split that does not improve the overall fit by a factor of `CP` is not attempted.

A study by Shihab et al[517] looked at reopened faults in the Eclipse project. Of the 18,312 bug reports 3,903 were resolved (i.e., closed at least once) and 1,530 of these could be linked to code changes. Of the 1,530 that could be linked to code changes, 246 had been reopened at the time of the study. Shihab et al cast their net very wide and extracted 22 factors, possible associated with reopened faults, from the source code repositories.

It is possible to plot a decision tree, but for non-trivial models the visualization has little practical use. Figure 7.2 shows the first few levels of the decision tree built from the Shihab et al data.

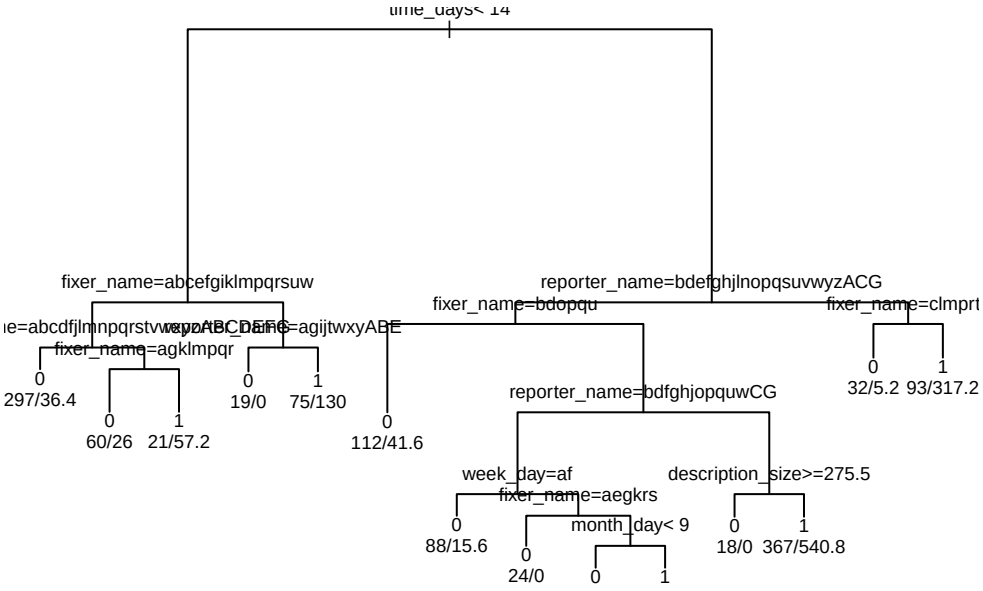


Figure 7.2: Top levels of the decision tree built from the reopened fault data. Data from Shihab et al.[517] code rexample[reopened.R]

Which variables were found to contribute most to the model? This information can be obtained by calling `summary` (the `cp=0.4` argument removes lots of details from the output; the `printcp` function does not provide any information on variable importance); the output below is for the reopened fault model:

```
> summary(weighted_model, cp=0.4)
Call:
rpart(formula = remod ~ time + week_day + month_day + month +
    time_days + severity + priority + pri_chng + num_fix_files +
    num_cc + prev_state + description_size + fixer_exp + fixer_name +
```

```
    reporter_exp + reporter_name, data = raw_data, weights = data_weight,
    method = "class", x = TRUE, parms = list(split = "information"))
  n= 1530

          CP nsplit rel error    xerror       xstd
1 0.17010632      0 1.0000000 1.1003752 0.01969308
2 0.02814259      1 0.8298937 0.9096310 0.01970490
3 0.02751720      2 0.8017511 0.8818011 0.01964631
4 0.02095059      5 0.6957473 0.8827392 0.01964854
5 0.01485303      6 0.6747967 0.8742964 0.01962785
6 0.01414947      7 0.6599437 0.9060350 0.01969820
7 0.01407129      9 0.6316448 0.9005629 0.01968751
8 0.01219512     10 0.6175735 0.8974359 0.01968114
9 0.01000000     13 0.5795810 0.8952470 0.01967656

Variable importance
      fixer_name    reporter_name        time_days         week_day
              32               32               13                6
description_size        fixer_exp     reporter_exp        month_day
               4                3                2                2
        priority         severity       prev_state    num_fix_files
               1                1                1                1
           month
               1

Node number 1: 1530 observations
  predicted class=0  expected loss=0.4990637  P(node) =1
    class counts:  1284 1279.2
   probabilities: 0.501 0.499
```

The variables found to make a useful contribution to the model and a measure of their relative importance appears at the end (at least when a large `cp` argument is passed).

For the identity of people fixing and reporting problems to play such a large role in the model suggests that either this subset of people have some characteristic or the faults they close have some characteristic that causes the faults they close to be reopened. A useful pointer for further investigation.

The columns of numbers in the middle of the output contain two measures of error for various values of `CP`. Decision trees are susceptible to overfitting and the `xerror` column estimates the error using ten-fold cross validation (the error listed in the `rel error` column does not use cross validation and gives a rosier estimate). The output above suggests that building a model using the `CP` value listed in the second row is likely to produce more accurate results than other values (the default value of `CP` is 0.01).

?

REXAMPLE[wcre2012-delaystudy.R] find out which variables are the biggest predictor of delayAfterChange...

7.2 Clustering

By dividing items in the world into categories of things, people reduce the amount of item specific information they need to learn,[458] information on an item that has not been encountered before can be inferred by deducing the category it is most likely to be a member of and then applying what is known about the chosen category; studies have found that people are sensitive to the costs and benefits of using categories[369]...

Many clustering algorithms will always produce a clustering of the data; the clustered returned may just be patterns that happen to exist in random data. The `VAT` and `iVAT` functions, Visual Assessment of (Clustering) Tendency and the improved version, in the `seriation` package can be used to obtain some idea about the number of clusters that may be present in data...

Sequence mining...?

Identifiers splitting/expansion by lots of subjects.[237] Are there clusters of developers making similar choices?...

Cluster of log file messages… Given the diverse nature of event log entries, often undocumented, the first step is to obtain a list of the message types present in the log file.

In the plot there appears to be vertical banding in the plot and a horizontal grouping of very popular installations…

languages used together, known by same person, emailed…?

7.2.1 Principle component analysis

Reduces the dimensionality of a multivariate dataset…

The principle components are transformed linear combinations of existing explanatory variables, these principle components are uncorrelated with each other…

Used when there are too large number of explanatory variables or when the explanatory variables are highly correlated with each other…

Primarily an exploratory technique, but… Sometimes the principle components are treated as an end in themselves and are then interpreted in the same way that explanatory variables might be *explanatory factor analysis*…

The `biplot` function…

Independent component analysis …

Some people claim factor analysis is not worth the time needed to understand it…

7.2.2 Seriation

Seriation involves finding a linear order for items that minimises/maximises some metric, with the hope that the linear order obtained has a structure that can be interpreted in a meaningful way.

The `seriation` package includes support for a range of algorithms that attempt to find some kind of optimal linear ordering of items. Evaluating all possible item orderings is impractical for all but the smallest samples (the number of possibilities grows as $n!$) and so heuristic algorithms are used.

A study by Jones[302] investigated the extent to which developers create similar data structures to hold information listed in a specification, i.e., grouping together identifiers containing related information in the same data structure. The hypothesis was that shared cultural and professional experiences would cause subjects to define similar data structures.

Subjects were given a list of items from the 'Department of Agriculture' and asked to design a C/C++ API containing this information. The results list, for each subject, the `structs` or `classes` defined and their fields/members (with each field containing one item of API information, e.g., 'Date crop harvested' and 'Organically produced').

Depending on the question asked and the form of the data various techniques are available.

Figure 7.3 shows which items are placed in the same data structure as one particular item, in this case 'Antibiotics used', by each subject. The matrix passed to `seriate` contained boolean values (in the same data structure or not) and subjects/fields are ordered.

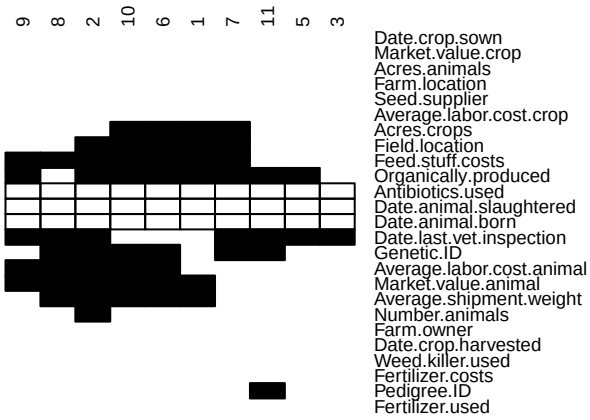


Figure 7.3: A Bertin plot for items included in the same data structure as 'Antibiotics used', for each subject, after reordering by `seriate`. Data from Jones.[302] code

```
library("seriation")

fser=seriate(fmat, method="BEA",  control = list(rep = 10))
bertinplot(fmat, fser, options=list(panel=panel.squares, spacing=0,
                        gp_labels=gpar(cex=0.6)))
```

A single item's pattern of association with all the other items can be generalised by counting the number of times every pair of items occurs together in the same data structure. A Robinson matrix has the property that the value of its matrix elements decrease, or stay the same, when moving away from the major diagonal and this matrix has been used to study commonality in subjects' categorization behavior.[489] Figure 7.4 shows a visualization of a Robinson matrix.

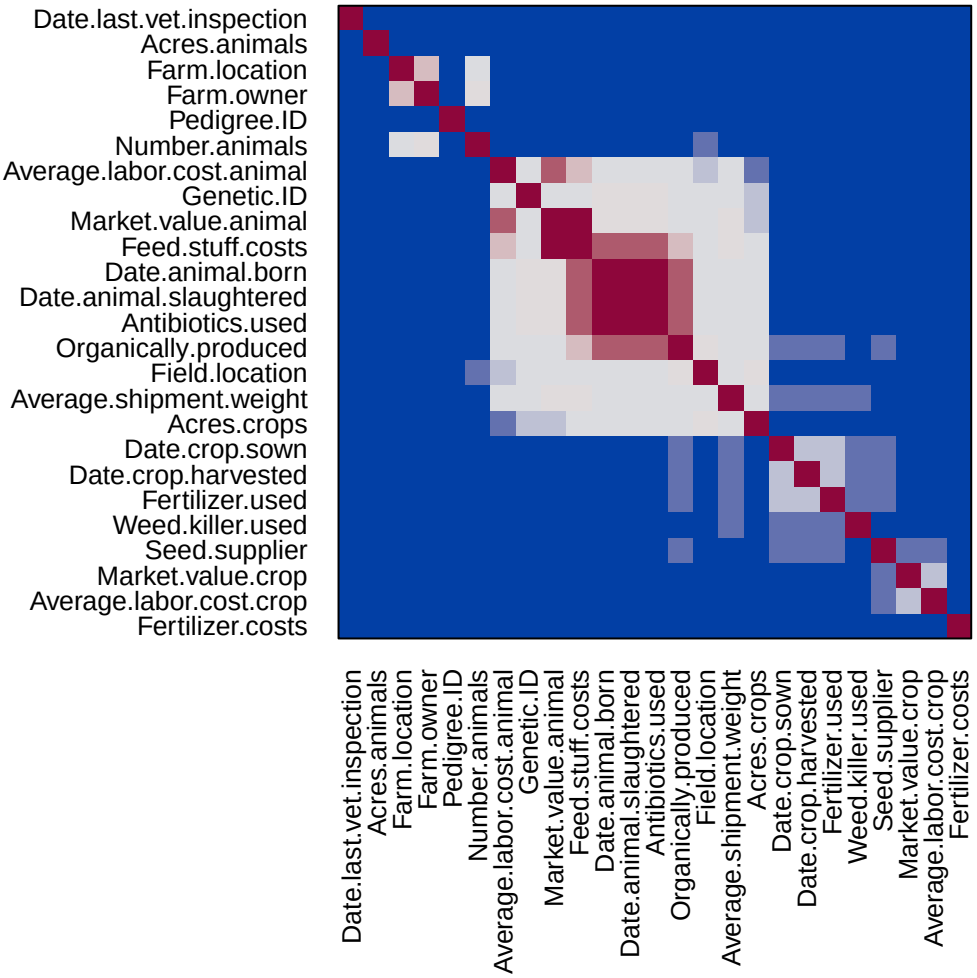


Figure 7.4: A visualization of the Robinson matrix based on number of times pairs of items co-occur in the same data structure (the closer to the diagonal the more often they occur together). Data from Jones.[302] code

```
library("seriation")

fdist = as.dist(1 - fmat/max(fmat)) # Normalise counts
fser = seriate(fdist, method="BBURCG")

pimage(fdist, fser, col=pal_col, key=FALSE, gp=gpar(cex=0.8))
```

7.3 Text analysis

Are there differences in the descriptions and comments associated with reopened and non-reopened faults that can be used to tell them apart using...

The file rexample[reopened_text.R] is an example of using the `tm` package to extract and process (e.g., remove punctuation, stop words and stem words) the words in each of the fault descriptions and comments.

Bayes rule tells us:

$$P(S|W) = \frac{P(W|S)P(S)}{P(W|S)P(S) + P(W|H)P(H)}$$

where: S is reopened, H non-reopened.

To combine the probabilities for each word associated with a given fault description/comment the naive Bayes classifier formula is:

$$P = \frac{psw_1 psw_2 \cdots psw_n}{psw_1 psw_2 \cdots psw_n + (1 - psw_1)(1 - psw_2) \cdots (1 - psw_n)}$$

Submitter correlation with description... Fixer correlation with comments...

Sentiment analysis is a popular technique for text by positive and negative opinions.

A study by Jongeling, Datta and Serebrenik[305] found that, when applied to different issue tracking datasets, different sentiment analysis tools assigned different labels and did not agree with the manually labels assigned to a software engineering dataset.... Labeled software engineering datasets are starting to appear...**?**

?,**?**,**?**

?

Chapter 8

Experiments

8.1 Introduction

Does doing X have a significant effect on S? Traditionally X might have been a new kind of fertiliser (or drug) and S the crop yield (or being cured of some illness). In software engineering the effect sought is often a performance improvement and X the latest snake oil.

In an observational study the researcher is a passive observer, simply recording what happened or is happening; in an experimental study the researcher actively attempts to control the values of the explanatory variables (a common technique is to vary the values of one explanatory variable while the others are held constant).

A controlled experiment is the technique used to obtain the data needed to test a hypothesis. The controlled experiment that most developers are likely to be familiar with is benchmarking.

Dramatic changes in performance, after doing X, are relatively common in software development and in such cases it is unnecessary to use statistics to confirm that a noticeable change has occurred. In some cases, a difference that requires statistical analysis to confirm is not a difference worth being concerned about.

Advice for running experiments often mimics the waterfall model of software development, with lots of planning and little or no feedback from production use until almost the end of the process. This advice has its roots in the environment in which experiments are carried out by the audience of many statistics text books, where running an experiment is costly (in money or time) or is a once only opportunity.[i]

Some questions in software engineering are amenable to iteration. Running quick, inexpensive experiments can be an efficient technique for filtering possible issues of interest and obtaining information on which of the myriad of variables have a large impact on the response of interest.

Finding the right question to ask is sometimes the most useful output from running an experiment.

An important point to remember is that, it is better to have an inexact answer to the right question than an exact answer to the wrong question.

Like all software development activities, experiments have to pay their way. Some of the answers needed for a cost-benefit analysis include the following:

- the cost of running an experiment capable of producing the information of interest within acceptable confidence intervals,
- the usefulness of information likely to be obtained by running an experiment using a given amount of resources (such as time and money)

[i] Experiments in the social sciences, major producer of experimental studies, are often grant funded, with limited opportunities for rerunning experiments that failed to produce data that can be published.

Many software engineering tasks are performed within complex environments. Controlling and measuring all the variables in the environment is so time consuming and expensive that few researchers are willing to attempt controlled experiments. Consequently, much of the hypothesis testing performed in commercial environments is based on convenience samples, obtained from experimentally uncontrolled, production software projects.

Experimenters want to use as many subjects as possible, because the greater the number of measurements the more accurate the results are likely to be. However, it is rare to have the luxury having access to more subjects than it is possible to make practical use. Obtaining experimental subjects generally involves being forced to make do with what's available.

Software engineering is not known as a subject where experiments are commonly performed. A study[526] of 5,453 papers in software engineering journals published between 1993 and 2002 found that only 1.9% reported controlled experiments (of which 72.6% used students only as subjects) and the statistical power of many of these experiments fell below expected norms.[154]

8.2 Design of experiments

Randomization is the foundation of any claims of causation involving experimental results, i.e., doing X caused Y might be considered a valid interpretation of the data. Without randomization the most that can be said is that there is a correlation between doing X and Y occurring.

An experiment measures the performance of subjects carrying out a task in a particular environment. If the information of interest is subject performance carrying out this task in the environment used in the experiment, then the results are likely to contain exactly the information of sought. However, in practice there is not always one-to-one mapping between the experiment and practice, with differences including:

- the subjects are a convenience sample of the population of interest,
- the task used does not share all the important characteristics of the tasks that is performed outside of the experiment, or those that are shared are in very different proportions,
- the environment in which the experiment is performed is different from the one likely to exist outside of the experiment (e.g., available time may be shorter in an experiment).

A critical component of experiment design is controlling all variables that could have a significant impact on the output. Failure to take into account and control variables having a significant impact can cause a tiny effect to appear to be a large effect and vice versa. One person's tongue in cheek advice on how to bias an experiment to obtain the desired outcome[393] is another person's list of thoughtless mistakes.

Ideally all the factors that could have a significant impact on the outcome of an experiment are controlled and when they cannot be controlled their impact is contained.

One technique for handling the problem of uncontrolled variables is to group subjects into blocks based on the variable that is suspected of influencing the response (a process known as *blocking*), randomization of subjects then occurs within each block. The identity of the block becomes another explanatory variable during analysis of the results.

Subject characteristics can sometimes interfere with good experimental design, e.g., human subjects have a memory of their previous experiences that they cannot choose to erase.

A study by Basili, Green, Laitenberger, Lanubile, Shull, Sørumgård and Zelkowitz[45] compared the performance of perspective based reading (which instructs reviewers to read a document from a specified perspective, e.g., a designer, tester or user) against the reading technique currently used by the professional developers who were the subjects.

The researchers thought it likely that training subjects to use the new technique would change their performance on whatever technique that currently use, and decided to measure subject performance when using their current technique first, before giving them any training in the new, perspective based reading, technique.

Being forced to have all subjects use the same techniques in the same order means it is not possible to separate ordering effects in the results, e.g., learning during the experiment and any random distraction effects that only occurred at certain times.

Other factors outside of the control of the researchers, that could affect the results, include:

- the time taken for people to become proficient at using a new technique, old habits die hard. How much practice do subjects need to obtain a reliable estimate of the performance of a new technique? In this study subjects were taught PBR two days after the first part of the experiment, trained on a test document, reviewed one document, received more training and then reviewed another document.
- the kind of review technique used by subjects in the first half of the experiment. A change in performance is expected, but the details of what the change is relative to are not known (it is assumed that adhoc techniques are being used),
- the characteristics of the seeded faults. Were more faults found in the NASA documents because readers were familiar with reading that kind of document, or perhaps the characteristics of the seeded faults was such that they were harder to detect in one kind of document than another?

The experimental output included, for each subject, the number of faults detected (which has a known upper limit and a yes/no detection status) and the number of false positives (which has no upper limit, in theory) in each document reviewed. These characteristics of the data select for a binomial distribution for faults found and a Poisson distribution for the false positive distribution in the respective regression models. See rexample[pbr-experiment.R]... more

Basing all experimental choices on random selection does not automatically create samples that maximise the information that can be obtained.

A study by Porter, Siy, Mockus and Votta[456] investigated software inspections. The structure of the inspection process was manipulated by varying the number of reviewers (1, 2 or 4), number of meeting (1 or 2) and for multiple meetings whether reported faults were repaired between meetings (88 inspections occurred, involving 130 meetings and 17 reviewers).

Selecting the treatment to use, for a review, from successive entries on a randomised list of all possible treatment structure combinations (created at the start of the study) would ensure that the results contain data that is balanced across the variables of interest. However, the choice of treatment to use was randomly selected from all possibilities as each unit of code became available for review, resulting in some combinations of reviewers/meetings/repaired not being used and some used very often. The sample contained an unbalanced set of experimental conditions, making it difficult to reliably fit a model (see rexample[inspection.R] and Figure 6.32).

8.2.1 Subjects

Experimental subjects might be people or artefacts such as computers. An essential requirement for generalising the results from an experiment to a larger population of subjects is that the applicable characteristics of the experimental subjects are representative of the population of interest.

Typically, a major limiting factor, when designing an experiment, is the amount of time subjects are likely to be willing to make available to participate.[525]

Perhaps the two main issues with human subjects that are much less of a problem with computers are peoples' ability to learn and their inability to quickly forget what they have learned.

The major issues involved in using computer hardware as subjects are covered later in this chapter (in the section on benchmarking), while the first chapter covered the major issues in human cognitive performance.

A study by Zislis[631] measured the amount of time taken (in minutes) by the same person to implement 12 algorithms, the process repeated three times using a different language for each algorithm and on the fourth repetition reusing the language used on the first iteration. Figure 8.1 shows that in nearly all cases performance improved between the first and last

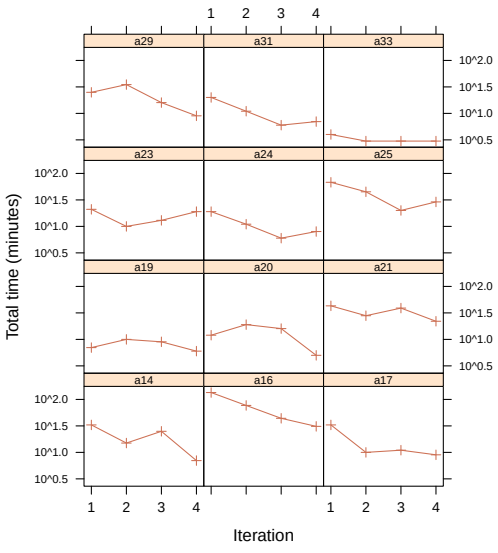


Figure 8.1: Time taken, by the same person, to implement 12 algorithms from the Communications of the ACM, with four iteration of the implementation process. Data from Zislis[631] code

implementation, with some continuously improving while others got worse before they got better.

Professional developers working in different ecosystems probably share a set of basic skills and knowledge, such as being able to fluently use at least one programming language.

Much of the published research involving human subjects in software engineering experiments has used students as subjects. Students are a convenience sample for many researchers and results based on student subjects are unlikely to be questioned, i.e., it is not an issue in getting the research published. However, the results from experiments using student subjects is unlikely to be applicable to professional software developers, reasons for this include:

- students' commercial software skills and knowledge is likely to be very poor in comparison to professional developers.[402] This lack of experience and know-how means that student subjects will have to spend time on activities that are second nature to professionals or they simply make noncommercial judgement calls,

- students, typically, have very little experience of writing software, perhaps 50 to 150 hours (and many have no basic coding skills[359, 377, 580]), while commercial software developers are likely to have between 1,000 to 10,000 hours of experience. This lack of programming fluency means that student programming performance is likely to contain the effects of a strong earning component, as well as student performance being much lower than professional developers,[397]

Industry is well aware that students' software engineering skills are not representative of professional developers; industry is where many graduates find employment after graduation and the abilities of these new employees is plain for everyone within industry to see.

In other areas of research students subjects may be more representative of the target population, because they have had many years of experience performing the activities and tasks used in those areas, e.g., processing text written in English and everyday image processing are activities used in cognitive psychology experiments.

When experimental results are intended to be applied to the population of university students studying a computing related subject, subjects drawn from this population can be representative of it.

In the US and UK students pay to attend university and like all businesses universities have to respond to customer demand. When a student decides to study a computing related subject, perhaps because they believe it will improve their job prospects, the University's interests are in ensuring that the student meets its minimum entry requirements and can pay; the likelihood of that person obtaining employment in a software related job is not a consideration.

Given the high failure rate for programming courses,[?] many students on such courses may not even have any software development skill or ability...

Note on terminology: many academic studies use the phrase *expert* to describe subjects who are final-year undergraduates or graduate students, with the term *novice* used to describe first-year undergraduates. In a commercial software development environment a recent graduate is considered to be a *novice* developer, while somebody with five or more years of commercial development experience might know enough to be called an *expert*.

Amazon's Mechanical Turk is becoming popular as a resource for finding subjects and running experiments. Subjects can stop taking part in a MTurk experiment at any time and care needs to be taken to ensure that the characteristics of subjects who remain does not bias the results.[628]

8.2.2 The task

A requirement for generalising the results from an experiment to the tasks performed under work conditions is that the characteristics of the experimental tasks performed by subjects share the same important characteristics as the work tasks. That is, the task needs to mimic realistic activities (the technical term is being *ecologically valid*), the two important factors are:

- being representative of real world intended usage requires obtaining reliable information about how a system will be used in real life and the inputs it is likely to experience; lack of resources to perform an analysis of real world usage often means that a convenience sample is used. In a rapidly changing environment it may not even be possible to specify usage patterns in sufficient detail and perhaps one of the important real world behaviors that needs to be benchmarked is adaptability to change.

  A study by Gregg and Hazelwood[231] provides an example where data usage characteristics are the deciding factor in the cost/benefit trade-off They measured the time taken to perform a matrix multiply when the CPU uses a local GPU (the SGEMM implementation in the nVidia CUBLAS package[420] was used). Figure 8.2 shows that moving data between the CPU and GPU (a Barracuda12 containing a GTX 480 with 1024MB) consumes a significant amount of time relative to the work done on the data once inside the GPU. Deciding whether GPU usage is worthwhile depends on the size of matrices encountered in the real world use case, the performance may be slower because of the data transfer overhead, or faster if the matrices are large to consume the largest amount of compute resources.

  Another example of the impact of variations in the input data is related to Figure 5.6.

- using a product that is identical to the one that will be used in production. Products that appear to be very similar often have very different performance characteristics (this is one reason why they exist as different products; the other common reason is marketing).

  In a study by Bird,[61] a performance optimization expert took the existing generic code of a library and created tuned versions for each of five different processors (IBM's Blue Gene P and four different members of Intel's x86 product line). The performance of the generic and all tuned versions of the code was measured on all processors. Figure 8.3 shows relative performance, with the x-axis listing the processor the code was tuned for and the y-axis the processor on which it was run; the performance impact of executing code tuned for one processor on a different processor is dramatic.

In practice, availability of resources is often a constraining factor; for example, benchmarking backup/restore tools or desktop search applications requires that realistic file system contents be used (e.g., the file system must contain a realistic number of files, directory depth, disk fragmentation, etc); getting to a position of being able to generate realistic file systems is a non-trivial task,[5] let alone realistic file content characteristics.[562]

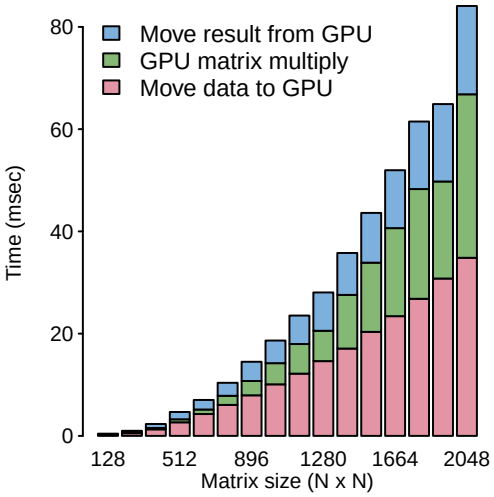


Figure 8.2: Time taken to transfer and multiply 2-dimensional matrices of various sizes on a GTX 480 GPU. Data kindly supplied by Gregg and Hazelwood.[231] code

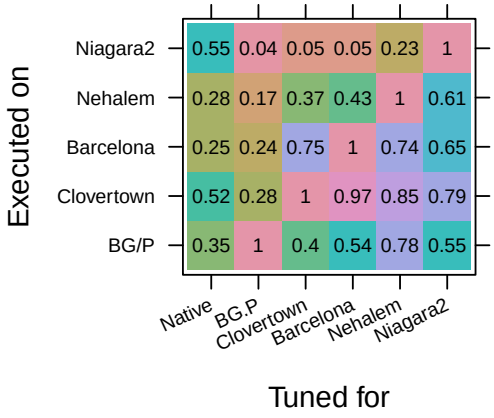


Figure 8.3: Relative performance (y-axis) of libraries optimized to run on various processors (x-axis). Data from Bird.[61] code

8.2.3 What is actually being measured?

Subjects may not solve the problems they are presented with, in an experimental context, in ways that were intended by the person who designed the experiment.

Subject motivation is an important factor in obtaining reliable experimental data. Subjects who feel they are being coerced may respond by providing spurious responses or simply attempt to minimise the time spent taking part in the experiment, without attracting attention by making too many errors.[252]

The history of research into human memory provides an example of how early experimental results were misinterpreted.[298] Experiments asked subjects to remember sequence of digits and the results suggested that STM has a capacity limit of 7 ± 2 items.[390] The 7 ± 2 digit limit model was later replaced by a model based on a limit of 2 seconds of sound[36] (in English this corresponds to around 7 digits, 5.8 in Welsh[158] and around 10 digits in Chinese:[276] the number of digits that can be held in memory when people use these languages).

Software developers are problem solvers and get plenty of practice in finding patterns that can be used to achieve a goal. Unless an experiment is carefully constructed, it would be naive to assume that developers will use any of the techniques anticipated by the person who designed the experiment.

Your author once ran several experiments[300] expected to find a *2 seconds of sound* effect in developers short term memory of source code (sequences of simple assignment statements were used). A great deal of attention went into creating code sequences whose spoken form required either more or less than 2 seconds of sound, but the results did not contain any evidence for the expected effect (i.e., a difference in performance caused by the length of sound in the spoken form of source code statements).

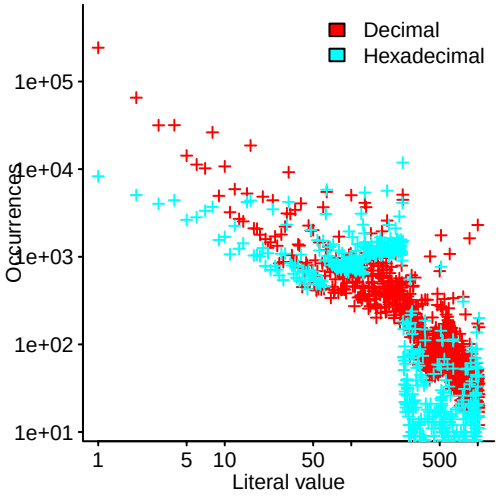


Figure 8.4: Number of integer constants having the lexical form of a decimal-constant (the literal 0 is also included in this set) and hexadecimal-constant that have a given value. Data from Jones.[299] code

At the end of one experiment a subject mentioned a strategy he had used to help improve his performance, remembering the first letter of each variable (I had not noticed that the variables in each list had unique first letters). Use of this strategy reduced the amount of STM the subject needed to use and is one explanation for why the expected effect was not found.

The last task on subsequent experiments asked subjects to list any strategies they used during the experiment.

A study[237] of how subjects source split code identifiers into components and expanded them, or not, into words measured individual performance against what was considered to be the definitive expansion of each identifier. The results of this experiment could be used to measure the performance of the researchers in creating a list of identifier expansions that maximised the likelihood of developers correctly decoding the intended information.

Source code is full of cases where what might be thought to be small differences in semantics have a large difference in usage. For instance, many languages allow numeric literals to be specified using decimal or hexadecimal notation and there is a large difference in the distribution of occurrence of literal values written using each notation; see Figure 8.4.

8.2.4 Stopping conditions

The cost of running an experiment means it can be very tempting to stop as soon as what is thought to be a reliable result is obtained. For instance, a researcher may process subjects in batches, running statistical tests after each batch of results becomes available (e.g., checking the p-value), to check progress.

Differences in subject performance, as an experiment progresses, will cause variations in the result of statistical tests and it is possible that some cut-off value (e.g., a p-value significance level) is temporarily achieved before later result cause it to revert to a less extreme value.

Terminating the testing of subjects before results from all the preplanned subjects are available...

In an experiment terminates before completing all the planned cases is an important material fact that needs to be reported, so readers can make their own assessment of its impact on their confidence in the results.

Writeups must include a list of all the variables collected in an experiment. It does not include information on variables that did not make it into the final model, however readers need to be able to judge whether variables have been cherry picked... bonferonni correction has been applied...

Examples of ways in which A/B testing on the web can go wrong[329]...

Sequential adaptive design for modifying an experiment as results become available... sequential sampling...

The `Sequential` and `AGSTest` packages...

Sequential probability ratio test `SPRT` package...

Using Instance selection and rough set theory to decide which options to use for the next sample...?

8.2.5 Selecting experimental options

The behavior of many systems depends on the setting of a wide variety of options and an estimate of the impact of individual options on system performance may be required. One way of obtaining this estimate is to measure system performance for all possible combinations of option values. This approach might be practical for a small numbers of options, each having relatively few values, e.g., Apache supports nine build time yes/no options giving 2^9 possible configurations (out of these 512 only 192 are valid). However, many large systems often support some many options that building and executing every configuration would be impractical (SQLite supports 3,932,160 valid options).

An experiment in which all possible permutations of option values (distinct options are known as *factors*) are tested is known as a *full factor design*. The `fac.design` function in the `DoE.`

base package takes a specification of the factor levels and produces a list of all combinations that need to be run to perform a full factor design (see rexample[design_fac.R).

A study by Citron and Feitelson[112] investigated the performance impact of adding what they called a Memo-Table (essentially a cache designed to store and reuse the results of previously executed instruction sequences) to the IBM Power4 cpu architecture. The configuration options for the Memo-Table were Size (1k or 32k), Associativity (1-way or 8-way), Mapping (indexing by program counter or operand+opcode) and Replacement method (random or least recently used).

Four configuration parameters, each having two possible values, gives $4^2 \rightarrow 16$ possible configurations. Citron and Feitelson benchmarked all 16 possibilities, enabling them to check for interactions between all factors. However, in many cases the number of interactions between factors is small and a common cost saving is to only consider interactions between pairs of factors.

Factor having just two possible values is a common case and is known as a two-factor factorial design: it is a *full two-factor design* when all combinations used and a *fractional two-factor design* when a subset is used.

The FrF2 function, in the FrF2 package, generates a list of the combination of factor values that need to be run to analyse N factors having a resolution of R (the ability to separate out main effects and interactions between factors; to be able to separate out main effects a resolution of 3 is required, a resolution of 4 enables detection of separate pairs of interactions). A call specifying the number of runs and number of factors produces a list of options to use for each run, along with the number of interactions that can be analysed:[ii]

```
> library("FrF2")
> FrF2(nfactors=4, resolution=3, alias.info=3)
   A  B  C  D
1  1  1 -1 -1
2 -1  1 -1  1
3 -1  1  1 -1
4  1 -1  1 -1
5 -1 -1 -1 -1
6  1 -1 -1  1
7  1  1  1  1
8 -1 -1  1  1
class=design, type= FrF2
```

The price paid for running a fractional, rather than full, factorial design experiment is that it is not possible to distinguish interactions between some combinations of factors. For instance, after running the eight combinations listed above it is not possible to distinguish between an effect caused by a combination of the AB factors and one caused by the combination CD; this combination is said to be *aliased*. The complete list of aliased factors is:

```
> design.info(FrF2(nfactors=4, resolution=3, alias.info=3))$aliased
$legend
[1] "A=A" "B=B" "C=C" "D=D"

$main
[1] "A=BCD" "B=ACD" "C=ABD" "D=ABC"

$fi2
[1] "AB=CD" "AC=BD" "AD=BC"

$fi3
character(0)
```

To distinguish between an effect caused by any of these combinations, all 16 combinations of factors have to be run.

Factorial designs require the number of runs to be a power of two, so the number of different runs grows very quickly as the number of factors increases. A Plackett & Burnam design only requires that the number of runs be a multiple of four...

See argument checking...

[ii] Some functions use +/− rather than 1/−1.

A study by Lee and Brooks[349] made use of three optional values per parameter; see rexample[lee.R]. Randomly selecting option values is an inefficient use of resources because some option values are over/under selected (see rexample[SQL_PWR.R] from a study by Guo et al[241]).

8.3 Analysing the results

The need to compare measurement samples obtained from running experiments kick started the development of statistics. The range of possible different experimental designs (e.g., one/two/k samples, parametric/non-parametric and between/within subject) and the need for practical manual solutions produced techniques designed to handle each specific case.[iii]

The chapter compares measurement samples using techniques that are only practical when a computer is available to do the calculations. The two techniques are regression modeling and Monte Carlo methods (or permutation tests when the sample is small enough for it to be practical to calculate an exact answer).[iv]

Many data analysis techniques assume that each measurement in a sample is independent of the other measurements in the sample. There is a common kind of experiment that produces measurements likely to violate this assumption, i.e., an experiment that measures the same subject before and after the intervention.

The analysis of samples containing repeated measurements of the same subject is known as *within-subject*, while analysis of samples containing a single measurement of each subject is known as *between-subjects*.

Samples may be compared to check whether they are the same/different, in some sense, or by specifically testing whether one sample is greater or less than the other:

- in a *two-sided* test (also known as a *two-tailed* or *non-directional* test) the samples are checked for being the same or different, where an increase or decrease in some attribute is considered a difference. The percentage on each side, see Figure 8.5, is half the chosen p-value,
- in a *one-sided* test (also known as a *one-tailed* or *directional* test) the samples are checked for only one case, either an increase or a decrease in the measured attribute. The percentage on the one side, see Figure 8.5, is the chosen p-value,

A commonly encountered null hypothesis, when comparing two samples, is that there is no difference between them. In many practical situations a difference is expected to exist, otherwise no effort would have been invested in obtaining the data needed to perform the analysis.

Experiments are often performed because a difference in one direction is of commercial interest. However, expecting or wanting a result that shows a difference in one direction is not sufficient justification for using a one-sided statistical test.

A one-sided test should only be used when the direction is already known or when an effect in the non-predicted direction would be ignored. If an effect in a particular direction is expected, but an effect in the opposite direction would not be ignored (i.e., would be considered significant) a two-sided test should be used.

Some of the kinds of sample comparisons commonly made include:

- a level of confidence that sample values have been drawn from the same/different distribution,
- the difference, d_m, in the mean of two samples,
- the difference, d_v, in the variance of two samples,

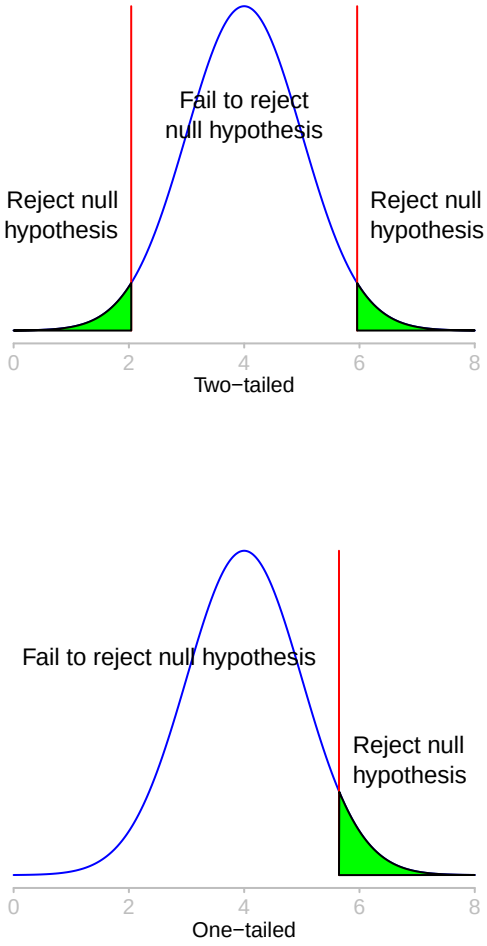


Figure 8.5: One and two-sided significance testing. code

[iii] These techniques come with requirements on the characteristics of the data, e.g., the before/after sample measurements share a common distribution, often the Normal distribution, or that samples have the same variance.

[iv] Other books tend to cover the manual techniques: such as the t-test, which is a special case of multiple regression using an explanatory variable indicating group membership, and the Wilcoxon-Mann-Whitney test, which is essentially proportional odds ordinal logistic regression.

- the correlation, C, between values paired from two samples.

Correlated measurements When measurements are made at specific points in time, while an experiment is running, it is possible that later measurements will be affected by earlier events that are not part of the benchmark. Perhaps running one program of a multi-program benchmark causes the system to enter a state that changes its performance characteristics.

A correlation between successive measurements, where none should exist, either needs to be removed or taken into account during analysis. The Durban Watson test can be used to check for a correlation between successive measurements within each run. The `durbinWatsonTest` function, in the `car` package implements this test. This issue is discussed elsewhere, see Figure 6.22.

8.3.1 Regression modeling

Regression modeling is used to analyse many of the datasets in this book. Using regression modeling to analyse experimental data might appear to be over-kill. But when a computer is available to do the calculations, it makes sense to use the most powerful analysis techniques available; why waste developer time learning to apply one of a variety of less powerful techniques (practical manual techniques are available for the different kinds of experimental data).

A study by Potanin, Damitio and Noble[457] refactored the Java Development Kit collection so that it nolonger made use of incoming aliases (e.g., following the owner-as-dominator or owner-as-accessor encapsulation discipline). The performance of the original and refactored versions were compared using the DaCapo benchmark,[63] which contains 14 separate programs, each of which iterates 30 times, with measurements made during each of the last five iterations; this process is repeated five times, generating 25 measurements for each program for a total of 350 measurements.

Potanin et al claimed that their changes to the aliasing properties of the original code did not degrade performance. If the claim is true, the explanatory variable kind-of-refactoring will have a trivial impact on the quality of the fitted regression model. The simplest model possible is based just on the name of the program and explains 99.9% of the variance (in this case the intercept is an unnecessary degree of freedom):

```
prog_mod=glm(performance ~ progname-1, data=dacapo_bench)
```

The fitted equation essentially just contains the mean value of the runtime of each separate program, for all programs in the sample. The `summary` output is: code

```
Call:
glm(formula = performance ~ progname - 1, data = dacapo)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4174.6   -205.0     -9.0    116.6   3946.5

Coefficients:
                  Estimate Std. Error  t value Pr(>|t|)
prognameavrora     22881.32      48.03  476.439  < 2e-16 ***
prognamebatik       2519.87      48.03   52.469  < 2e-16 ***
prognameeclipse    53660.53      48.03 1117.330  < 2e-16 ***
prognamefop          395.89      48.03    8.243 2.92e-16 ***
prognameh2         24100.39      48.03  501.823  < 2e-16 ***
prognamejython     15808.13      48.03  329.160  < 2e-16 ***
prognameluindex      708.00      48.03   14.742  < 2e-16 ***
prognamelusearch    7239.52      48.03  150.743  < 2e-16 ***
prognamepmd         4017.61      48.03   83.656  < 2e-16 ***
prognamesunflow    22788.81      48.03  474.513  < 2e-16 ***
prognametomcat      7672.11      48.03  159.750  < 2e-16 ***
prognametradebeans 27987.82      48.03  582.768  < 2e-16 ***
prognametradesoap  64888.58      48.03 1351.122  < 2e-16 ***
prognamexalan      26381.35      48.03  549.318  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
(Dispersion parameter for gaussian family taken to be 345970)

    Null deviance: 1.5873e+12  on 2100  degrees of freedom
Residual deviance: 7.2169e+08  on 2086  degrees of freedom
AIC: 32759

Number of Fisher Scoring iterations: 2
```

Apart from improving the fit for one program there is not much left to explain. Adding kind-of-refactoring as an explanatory variable (see rexample[dacapo_progname.R] for details) shows that it is not significant on its own, but some interaction exists between a few programs (primarily `sunflow`) and some refactorings. The slightly more complicated model:

```
prog_refact_mod=glm(performance ~ progname+progname:refact_kind,
                                                    data=dacapo_bench)
```

explains 99.92% of the variance. There are 12 program/refactoring interactions with p-values less than 0.05 (out of 84 possible interactions), with most of these changing the estimated mean performance by around 1% and one making 8% difference (`sunflow`, see [rexample[dacapo_progname_refact.R]).

Building a regression model has enabled us to confirm that apart from a few, small, interactions the various refactorings of the JDK did not change the DaCapo benchmark performance.

8.3.2 Factorial designs

A variety of techniques have been created to help visualise the results obtain from experiments using a factorial design. The analysis is the same for full and partial fractional designs, the only difference is the number of interactions between factors that will be available for analysis.

The study by Citron and Feitelson[112] discussed earlier used the SPEC CPU 2000 benchmark, and the results measured were integer floating point performance, power consumption, processor timing and die area of the chip.

For simplicity consider three of the factors (ignoring, for the time being, the replacement method), the results can be visualised as a cube with each of the eight vertices representing one combination of factor values. In Figure 8.6 the values at each vertice are the SPEC benchmark cint performance figures.

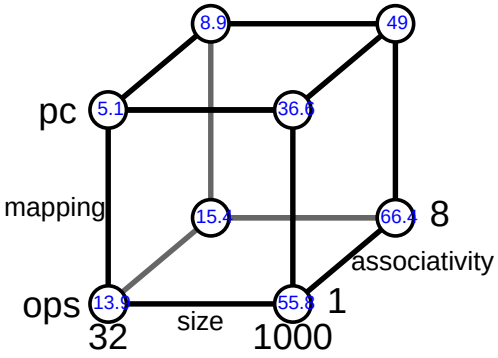


Figure 8.6: A cube plot of three configuration factors and corresponding benchmark results (blue) from Memory table experiment. Data from Citron et al.[112] code

Imagine taking two opposite faces of the above cube, say the two for `size` on the left and right going into the page, and finding the mean `cint` value for both faces, the difference between these two values is known as the *main effect* for `size`; this calculation can be repeated to find the main effect for the other factors.

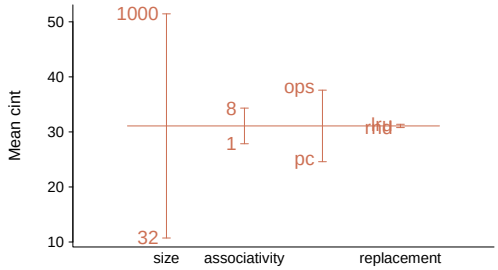


Figure 8.7: Design plot showing the impact of each configuration factor on the performance of Memo table on benchmark performance. Data from Citron et al.[112] code

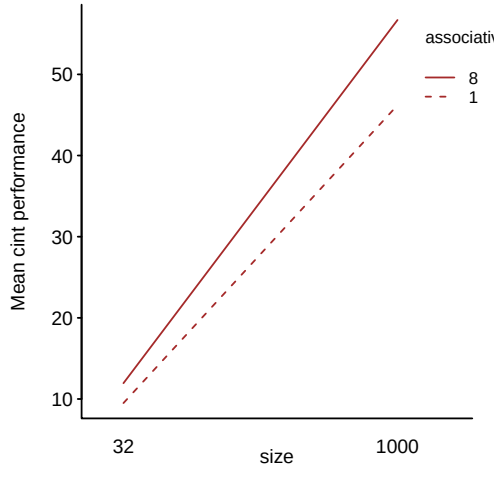
A design plot is a visualisation of these main effects, with a central horizontal line showing the overall mean value of the response variable. Figure 8.7 was created using the `plot.design` function and shows the impact that each factor can have on the value of `cint`, offset from the mean value.[v]

For an example involving more changeable parameters see rexample[lee.R].

At a finer level of granularity, an interaction plot shows the interaction between pairs of factors. The upper plot in Figure 8.8 shows how the mean value of `cint` varies as `size` is changed, for a given value of `associativity` (see legend); the lower plot shows the variation for given values of `mapping`.

For an equation based analysis, a regression model could be used to investigate the interactions between factors. For instance, the following code specifies interactions between all variable pairs (see rexample[MemoPower03.R] for details):

```
Memo_glm=glm(cint ~ (size+associativity+mapping)^2, data=Memo)
```



A study by Pallister, Hollis and Bennett[432] investigated the power consumed by various embedded programs when compiled with gcc using various command lines parameters... Thirty six rexample[gcc-power.R]...

[v] `plot.design` makes some unexpected display decisions when the explanatory variables are not factors.

8.3.3 Comparing sample means

Comparing two samples to check for any difference in their mean values is probably the most common statistical test of experimental data.

In the past the appropriate statistical technique to use has depended on the design of the experiment performed (e.g., within-subjects or between-subjects), the kind of question being asked and the characteristics of the data. Now that computers are available to do the calculation, the bootstrap has become the hammer used to answer sample comparison questions.

The bootstrap procedure often starts by assuming there is no difference, in some characteristic, between samples; it then calculates the likelihood of two samples having characteristic that they are measured to have. The assumption of no difference requires that the items in both samples be *exchangeable*. Deciding which items, if any, in a sample are exchangeable is a crucial aspect of using the bootstrap to answer questions about samples.

The nVidia GTX 970 is a very popular graphics card and many variations on the reference design have been produced (during August 2016 there were 51 variants included in the 64,392 results for this card in the UserBenchmark.com database). Figure 8.9 shows the number of Reflection benchmark results reported for GTX 970 cards from three third-party manufacturers.

The mean score of these Asus, MSI and Gigabyte cards are 176.2, 179 and 186.8 respectively. Are these differences most likely caused by random variation or by some real difference?

The following bootstrap technique provides a way of answering this question.

Assume there is no difference in the mean performance of, say, MSI and Gigabyte on the Reflection benchmark. In this case the benchmark results (255 from MSI and 73 from Gigabyte) can be merged to form a sample of 328 results. Using this combined empirical sample perform the following:

- randomly select, with replacement, 328 items from the empirical sample,
- divide this new sample into two subsamples, one containing 255 items and the other 73 items,
- find the mean of the two subsamples, subtract the two mean values and record the result,
- repeat this process R times,
- count how many bootstrapped differences in the mean are greater than the differences in the means of the two cards; no assumption is made about the direction of the difference, i.e., this is a two-sided test.

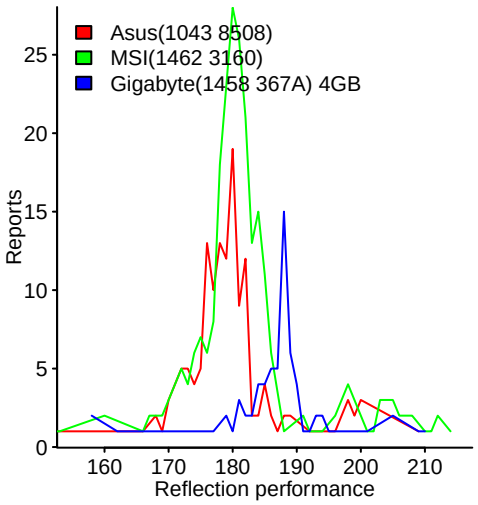


Figure 8.9: Number of Reflection benchmark results achieving a given score, reported for GTX 970 cards from three third-party manufacturers. Data extracted from UserBenchmark.com. code

The following code uses the `boot` function, from the `boot` package, to implement the above algorithm, with the user provided function (`mean_diff` in this case) that is called for each randomly generated sample (see rexample[UserBenchmark_compare.R]):

```
library("boot")

mean_diff=function(res, indices)
{
t=res[indices]
return(mean(t[1:num_MSI])-mean(t[(num_MSI+1):total_reps]))
}

MSI_refl=MSI_1462_3160$Reflection
Giga_refl=Gigabyte_1458_367A$Reflection

num_MSI=length(MSI_refl)
num_Giga=length(Giga_refl)
total_reps=num_MSI+num_Giga

GTX_boot=boot(c(MSI_refl, Giga_refl), mean_diff, R = 4999)

refl_mean_diff=mean(MSI_refl)-mean(Giga_refl)
# Two-sided test
length(GTX_boot$t[abs(GTX_boot$t) >= abs(refl_mean_diff)]) # E
```

The argument R specifies the number of resamples and boot returns the result of calling mean_diff for each of these samples.

The likelihood of encountering a difference in mean values as large as that seen in the MSI and Gigabyte performance (i.e., the p-value) is given by the following equation:

$$\frac{E+1}{R+1}$$

where E is the number of cases where the bootstrap sample had a larger mean difference. The result varies around: $\frac{34+1}{4999+1} \rightarrow 0.007$ (the MSI/Asus comparison the value is: $\frac{840+1}{4999+1} \rightarrow 0.17$).

If there were no difference in performance, a difference in mean value as large as that seen for MSI/Gigabyte is likely to occur 0.7% of the time; we might reasonably claim that this percentage is so small that there is likely to be a real difference in performance. A mean difference at least as large as the MSI/Asus mean difference is likely to occur 17% of the time if there was no real difference in performance; a large enough percentage to infer that there is unlikely to be any difference in performance.

This test answers the question of whether the difference in mean values is likely to be a chance effect.

If a difference is now thought likely to exist, the next question is the likely size of the difference and the confidence intervals on this difference.

A bootstrap procedure can be used to answer these questions.

Once the two samples are considered to be different, their contents can only be treated as exchangeable within each sample, not between the two samples. The two subsample now have to be generated from their respective empirical samples. The following code implements the functionality (see rexample[UserBenchmark_mdiff.R]):

```
library("boot")

mean_diff=function(res, indices)
{
t=res[indices, ]
return(mean(t$refl[t$vendor == "Gigabyte"])- mean(t$refl[t$vendor == "MSI"]))
}

# Create dataframe identifying vendor used for each measurement.
MSI_refl=data.frame(vendor="MSI", refl=MSI_1462_3160$Reflection)
Giga_refl=data.frame(vendor="Gigabyte", refl=Gigabyte_1458_367A$Reflection)

MSI_Giga=rbind(MSI_refl, Giga_refl)

# Pass combined dataframe and specify identifying column
GTX_boot=boot(MSI_Giga, mean_diff, R = 4999, strata=MSI_Giga$vendor)
```

The boot.ci function calculates confidence intervals from the value returned by boot: code

```
> boot.ci(GTX_boot)
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 4999 bootstrap replicates

CALL :
boot.ci(boot.out = GTX_boot)

Intervals :
Level      Normal              Basic
95%   ( 4.462, 11.218 )   ( 4.348, 11.068 )

Level     Percentile            BCa
95%   ( 4.561, 11.281 )   ( 4.879, 11.769 )
Calculations and Intervals on Original Scale
> mean(GTX_boot$t)
[1] 7.788764
> sd(GTX_boot$t)
[1] 1.723573
```

Deciding if and when items in a sample are exchangeable can be non-trivial and require an understanding of the problem domain.

A study by Gandomani, Wei and Binhamid[197] investigated the accuracy of software cost estimates made using both expert judgement and Planning Poker on 15 projects in one company and both expert judgement and Wideband Delphi in 17 projects in another company; Table 8.1 shows a subset.

Is there a difference in the estimates made using expert judgement and either of the other two techniques?

Project	Expert judgement	Planning Poker	Difference
P1	41	40	1
P4	60	56	4
P7	33	45	-12
P12	18	20	-2

Table 8.1: Effort estimates made using expert judgement and Planning Poker for several projects. Data from Gandomani et al.[197]

Each estimate is specific to one project and it makes no sense to include estimates from other projects in the random selection process; estimates from different projects are not exchangeable. Possible ways of handing this include:

- treating each project as being exchangeable; the resampling could occur at the project level with both estimates for each selected project being used.
- randomly selecting from the set of estimates for each project. But there are only two estimates...

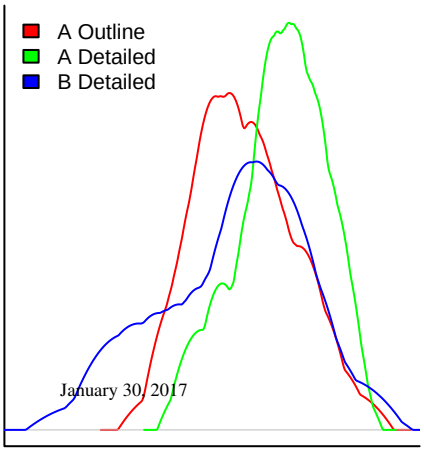
The following code randomly samples estimates for each project only within the sample of estimates for that project (see rexample[16.R]):

```
mean_diff=function()
{
s_ind=rnorm(len_est_2) # random numbers centered on zero
# Randomly assign estimates to each group
expert=c(est_2$expert[s_ind < 0], est_2$planning.poker[s_ind >= 0])
# Sampling with replacement, so two sets of random numbers needed
s_ind=rnorm(len_est_2) # random numbers centered on zero
poker=c(est_2$expert[s_ind < 0], est_2$planning.poker[s_ind >= 0])
# The code for sampling without replacement
# poker=c(est_2$expert[s_ind >= 0], est_2$planning.poker[s_ind < 0])
return(mean(expert)-mean(poker))
}

est_mean_diff=abs(mean(est_2$expert)-mean(est_2$planning.poker))
len_est_2=nrow(est_2)

t=replicate(4999, mean_diff()) # Implement our own bootstrap

length(t[abs(t) > est_mean_diff]) # How many this extreme?
```

The p-value for Expert vs. Planning Poker is 0.02 (see rexample[16.R])....

A study by Jørgensen and Carelius[308] asked companies to bid on a software development project.[vi] In the first round of bidding 17 companies were given a one page description of user needs and asked to supply a non-binding bid; in the second round the original 17 companies plus an additional 18 companies (who had not participated in the first round) were given an 11 page specification (developed based on feedback from the first round) and asked to submit firm-price bids.

What difference, if any, did participating in the first round make to the second bids submitted by the initial 17 companies (call them the A companies) and how did these bids compare to those submitted by the second sample of 18 companies bidding for the first time (call them the B companies)?



[vi] Four of the companies that submitted a bid were selected to independently implement the project.

Figure 8.10 shows density plots of the submitted bids. The mean values were: mean of kr183,051[vii] for initial bid from A companies, mean of kr277,730 for final bid from A companies and mean of kr166,131 for only bid from B companies.

Are the items in each sample (the companies asked to submit a bid) exchangeable? Small companies have lower operating costs than large companies; it is unrealistic to consider bids from small/large companies to be exchangeable. The size of companies involved in bidding were classified as small (five or fewer developers), medium (between 6 and 49 developers) and large (50 or more developers).

The call to `boot` now has to include information on how the data is stratified (i.e., split into different levels). The `strata` is used to pass a vector of integer values specifying the strata membership of the values in the first argument. Everything else stays the same, with `boot` treating member of each strata as exchangeable generating new samples (see rexample["compare-bid.R]):

```
bid_boot=boot(comp_bid$Bid, mean_diff, R = 4999,
                                  strata=as.factor(comp_bid$CompSize))
```

The p-value, for the hypothesis that the mean values are the same, is: $\frac{52+1}{4999+1} \to 0.01$, i.e., a difference this large is surprising.

Jørgensen and Carelius proposed the hypothesis that the main factor controlling the size of the bids was the information contained in the project specification. I think this is rather idealistic and more practical considerations are discussed in the section on project bidding.

Within subjects experiments using bootstrap... Given two samples, S_1 where X was not performed and S_2 where X was performed, the mean performance time of the samples, for instance, is likely to be different. How surprised should we be at this difference?

Permutation tests Sometimes the two samples are small enough that it is practical to generate and test all possible item permutations.

A study by Grant and Sackman[224] measured the time taken for subjects to write a program using either an online or offline computer interface (this experiment was run in the mainframe era of the 1960s). Given 12 subjects split into two groups of six, how likely is the difference in mean time between the online/offline cases?

This question is about the population of people that took part in the experiment, not a wider population. For this population there are `choose(12, 6) ==924` possible subject combinations. The following is an excerpt of an implementation of a two-sided test (see rexample[GS-perm-diff.R]):

```
subj_time=c(online$time, offline$time)
subj_mean_diff=mean(online$time)-mean(offline$time)

# Exact permutation test
subj_nums =seq(1:total_subj)
# Generate all possible subject combinations
subj_perms=combn(subj_nums, subj_online)

mean_diff = function(x)
{
# Difference in mean of one combination of subjects
mean(subj_time[x]) - mean(subj_time[!(subj_nums %in% x)])
}

# Indexing by column iterates through every permutation
perm_res=apply(subj_perms, 2, mean_diff)

# p-value of two-sided test
sum(abs(perm_res) >= abs(subj_mean_diff)) / length(perm_res)
```

For the Algebra program, 272 of the possible groups, of subject combinations, had a difference in mean time greater than, or equal, to that of the empirical sample. Because all possibilities have been calculated, the p-value is exact: $\frac{272}{924} \to 0.2934....$

[vii] The exchange rate is approximately 10 Norwegian Krone to one Euro.

The coin package provides this kind of exact calculation for many of the traditional group comparison tests, e.g., the wilcoxsign_test function is permutation test equivalent of the wilcox.test function in the base library.

Clustering by mean value using Scott-Knott test... ScottKnott

The bootstrap techniques applied to question about differences in the mean of two samples can be generalised to a wide variety of comparison tests. A new comparison test can be implemented by replacing the mean_diff function used above (the requirement of exchangeability is integral to the process).

8.3.4 Comparing standard deviation

A study by Jørgensen and Moløkken[311] asked 19 professional developers to estimate the effort required to implement a task, along with an uncertainty estimate (i.e., minimum and maximum about the most likely value). Nine of the developers were explicitly instructed to compare the current task with similar projects they had worked on (they were also given a table that asked them to assess similarity within various percentage bands).

The visual appearance of the density plots in Figure 8.11 suggests that there is a difference in the standard deviation of the estimates in the two samples. A bootstrap test of the difference in the standard deviations of the two samples can be implemented by replacing the mean_diff used in the previous section by the function sd_diff below (see rexample[simula_04sd.R]):

```
sd_diff=function(est, indices)
{
t=est[indices]
return(sd(t[1:num_A_est])-sd(t[(num_A_est+1):total_est]))
}
```

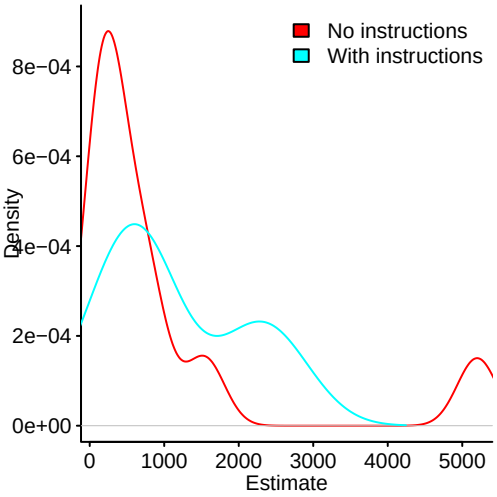


Figure 8.11: Density plot of task implementation estimates: with no instructions (red) and with instruction on what to do (blue). Data from Jørgensen el al.[311] code

The p-value, for the hypothesis that the standard deviations are the same, is: $\frac{2170+1}{4999+1} \rightarrow 0.43$, i.e., the difference is not that surprising.

The ansari_test function of the coin package [viii] implements a permutation test of the *Ansari-Bradley Test* (a two-sample test for a difference in variance); see rexample[simula_04_var.R].

Median Absolute Deviation mad for robust estimation of variance... Winsorized variance...

8.3.5 Correlation

Correlation is a measure of the closeness of linear association between variables, e.g., if one variable always increases/decreases when another variable increases/decreases. The range of correlation values is -1 (the variables change together, but in opposite directions) to 1 (the variables always change together), with zero denoting no correlation.

Correlation is related to regression, except that: it treats all variables equally (i.e., there are no response or explanatory variables), the correlation value is dimensionless and correlation is a linear relationship (e.g., in the $y = x^2$ relationship y can be predicted x, but there is zero correlation between them).

[viii] The ansari.test function is included in R's base system.

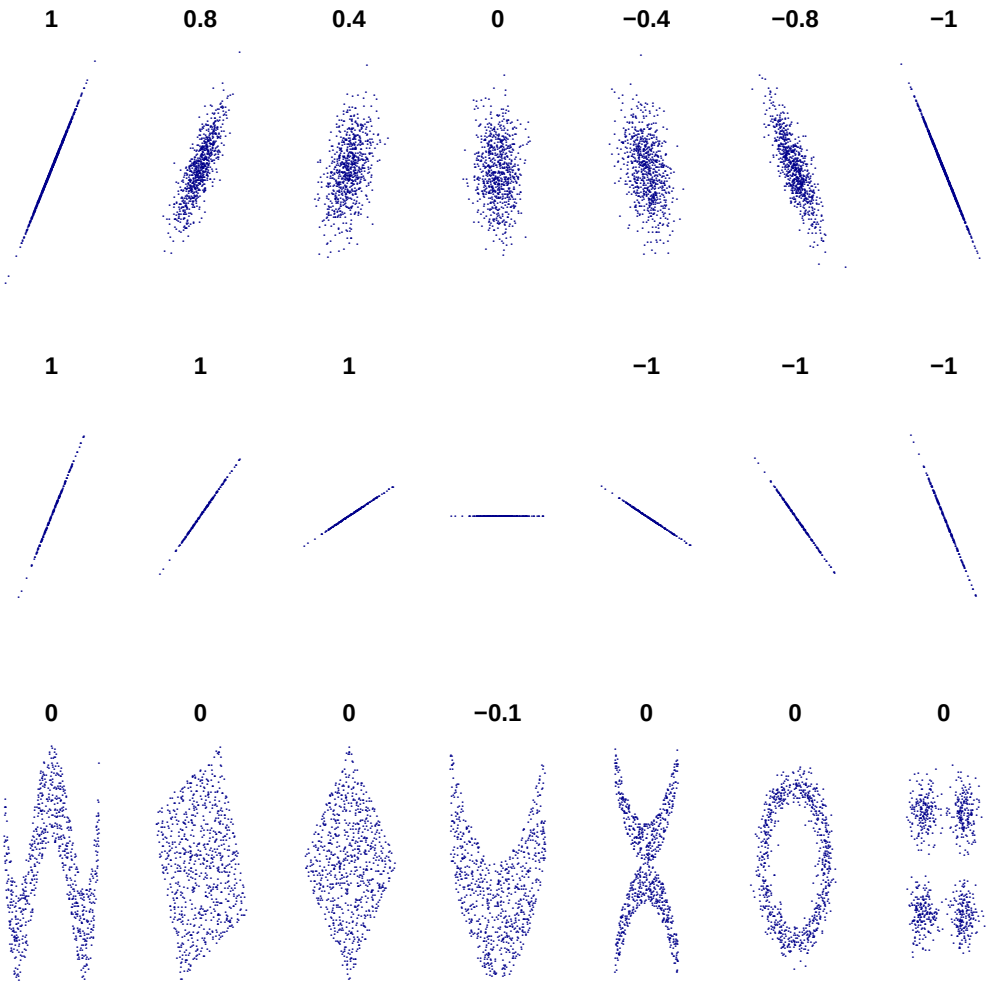


Figure 8.12: Examples of correlation between samples of two value pairs, plotted on x and y axis. code

Three commonly encountered metrics for correlation are:

- *Pearson product-moment correlation coefficient*, (also known as Pearson's R or Pearson's r), which applies to continuous variables,
- Spearman's rho, ρ (a lowercase Greek letter), is identical to Pearson's coefficient except the correlation is calculated from the ranked values, i.e., the sorted order (which makes it immune to extreme values),
- Kendall's tau, τ (a lowercase Greek letter), is like Spearman's rho in that it is based on ranked values, but the calculation is based on the number of items sharing the same rank (i.e., relative difference in rank plays is not included in the calculation; Spearman's rho does include relative differences).

Because they involve ranking items both Spearman's rho and Kendall's tau have problems handling samples containing items having the same value…

The `cor.test` function, included in the base system, supports all three coefficients and provides confidence interval.

8.3.5.1 Dichotomous variables

When the result of a measurement can only have one of two values, the standard techniques for calculating correlation, which require that most if not all values be unique, cannot be used. It is possible to recast the problem in terms of probabilities, which means that the approach taken for every problem could be different.

The following is an example of one approach to a particular binary problem.

If we are interested in being able to access files with very high reliability, then hosting the files on two or more websites would mean that if one site cannot be accessed the file could

be obtained from another site. The naive analysis suggests that if the average reliability of the websites is 95% then the reliability of two paired sites would be 99.75%; however this assumes that the unavailability of each website is independent of its paired site.

A study by Bakkaloglu, Wylie, Wang and Ganger[39] had a client program read a file from over 120 websites every 10 minutes, between September 2001 and April 2002. They recorded whether the file was successfully accessed or not.

Most of the websites are available most of the time, and this makes it harder to detect if other patterns are present. Bakkaloglu et al proposed various techniques for calculating correlated failures, based on the probability that site X is unavailable when site Y is unavailable, i.e., $P(Xunavailable|Yunavailable)$. The following example takes the mean value over all pairs of sites:

$$= mean(P(Xunavailable|Yunavailable))$$

$$= mean\left(\frac{P(X\&Yunavailable)}{P(Yunavailable)}\right)$$

The following calculates the average unavailability probability for one site paired with every other site (see rexample[web-avail.R]):

```
given=web_down[ , ind]
others=web_down[ , -ind]

both_down=(others & given)

av_prob=mean(colSums(both_down)/sum(given))
```

Averaged over all pairs of sites the probability of one site being unavailable when its pair is also unavailable is 0.3 (at the 10 minute measurement point). Given that all accessed originated from the same client it is not surprising that this probability is much higher than the average probability of one site being unavailable (0.1); all accesses start off going through the same internet infrastructure and problems with this infrastructure will affect access to all sites.

8.3.6 Contingency tables

Count data with categorical explanatory variables has a natural visual representation as a table of numbers; these tables are known as *contingency tables*. Table 8.2 shows a 2-dimensional table, with each entry containing a count of the items in the sample having both of the listed row and column attributes.

It is surprisingly common to encounter situations where lots of data has been reduced to a contingency table, i.e., potentially useful information has been thrown away. The only reason for reducing lots of data to the form of a contingency table is to hide information from readers. Analysis the whole sample is likely to reveal more about it than an analysis of a simplified form, i.e, analysis of a contingency table.

Sometimes the available measurements are only sufficient to build a contingency table.

A study by Nightingale, Douceur and Orgovan[411] investigated the characteristics of hardware failures over a very large number of consumer PCs. Table 8.2 shows a contingency table containing the number of system crashes believed to have been caused by hardware problems involving the system DRAM or CPU.

	DRAM failure	**no DRAM failure**
CPU failure	5	2,091
no CPU failure	250	971,191

Table 8.2: Number of system crashes of consumer PCs traced to CPU or DRAM failures. Data from Nightingale et al.[411]

The traditional, manual friendly, technique for analyzing this kind of data is the chi-squared test (χ is the Greek letter), which provides a yes/no answer.[ix]

[ix] The `chisq.test` function is part of the base system if your readership demands a chi-squared test and the `chisq_test` function in the `coin` package can be used to bootstrap confidence intervals.

A regression model can be fitted to this data (even though there is not a lot of it) and can extract a more information than the manual method. code

```
Call:
glm(formula = failures ~ CPU * DRAM, family = poisson, data = PC_crash)

Deviance Residuals:
[1]  0  0  0  0

Coefficients:
                  Estimate Std. Error   z value Pr(>|z|)
(Intercept)      13.786278   0.001015 13586.243  < 2e-16 ***
CPUTRUE          -6.140881   0.021892  -280.505  < 2e-16 ***
DRAMTRUE         -8.264818   0.063254  -130.661  < 2e-16 ***
CPUTRUE:DRAMTRUE  2.228858   0.452194     4.929 8.27e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance:  2.6646e+06  on 3  degrees of freedom
Residual deviance: -7.7825e-11  on 0  degrees of freedom
AIC: 43.948

Number of Fisher Scoring iterations: 3
```

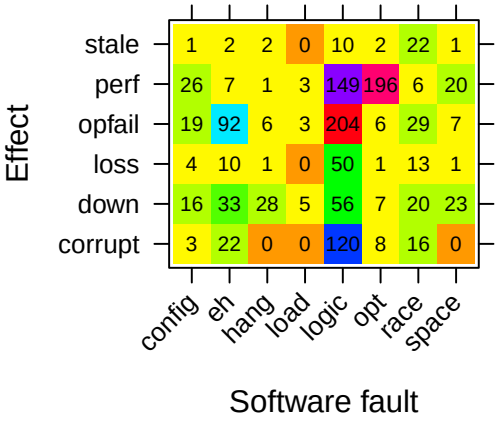


Figure 8.13: Number of software faults having a given consequence, based on an analysis of faults in Cassandra. Data from Gunawi et al.[238] code

The information in the fitted model that is not immediately obvious from the numbers in the table is that the crash rate is higher when both the CPU and DRAM fail.

The regression approach makes it trivial to handle more rows and columns, as well as non-straight line fits. As the number of values in the table increases it becomes more difficult to visually extract any patterns that may be present; a heatmap is one way of highlighting details, see Figure 8.13.

There are a wide variety of techniques for comparing multiple contingency tables. Different pair comparison algorithms can give very different results.[561]

8.3.7 Agreement between raters

A measurement may be based on human judgement, e.g., making a product rating. Different people may make different judgements of the same characteristic/entity and a way of evaluating the agreement between the different judgements is needed.

Cohen's Kappa is a measure of inter-rater agreement between two raters, it varies from zero (no agreement) to one (perfect agreement). When comparing categorical variables, all differences are usually given equal weight (i.e., considered equally different). When comparing ordinal values, where the degree of difference between raters is easier to quantify, a weighting based on the square of the difference is often recommended.

Fleiss's Kappa is a measure of inter-rater agreement between three or more raters.

The `kappa2` function in the `irr` package supports Cohen's Kappa (weighting is supported by passing the argument `weight="squared"`). The `kappam.fleiss` function supports Fleiss's Kappa.

A study by Schach, Jin, Yu, Heller and Offutt[499] categorised the kinds of maintenance activity performed on various systems. Table 8.3 lists the categories assigned by two raters to 215 maintenance categories involving the first 20 versions of Linux. The value of Cohen's Kappa for these two raters is 0.805 (see rexample[agreement.R]).

Rater agreement as a means of measuring ability of rules to be accurately followed[591]...

8.3.8 ANOVA

Readers are likely to encounter the acronym ANOVA (*Analysis of variance*), an analysis technique which developed independently of linear regression and having its own specialized terminology. This technique was designed for manual implementation.

	Adaptive	Corrective	Perfective	Other	Total
Adaptive	2	0	0	0	2
Corrective	0	82	16	0	98
Perfective	0	5	99	2	106
Other	0	0	0	9	9
Total	2	87	115	11	215

Table 8.3: Maintenance categories assigned by two raters for the first 20 versions of the Linux kernel at the change-log level. Data from Schach et al.[499]

Functionally ANOVA and least squares are both special cases of the general linear model (ANOVA is a special case of multiple linear regression with orthogonal, categorical predictors; ANCOVA adds covariates to mix). A one-way analysis of variance can be thought of as a regression model having a single categorical predictor that has at least two (usually more) categories.

Treating the various kinds of ANOVA models as special cases of the family of regression models makes it possible to use the more flexible options available in regression modeling (e.g., easier handling of unequal group sizes, adjusting for covariates and methods for checking models).

The `anova` function generates ANOVA style output when passed a model built using `glm` and some other regression model building functions. The `Anova` function in the `car` package...

One-way ANOVA focuses on testing for differences among a group of means; it evaluates the hypothesis that $\alpha_i = 0$ in the following equation:

$$Y_i = \mu + \alpha_i + \varepsilon_i$$

where μ is the group mean, α_i is the effect of the response variable on the i'th group and ε_i is the corresponding error.

The bootstrap is a more flexible and powerful technique.

Anova can also be used to compare linear models...

The `glht` function provides a comprehensive set of methods for multiple mean comparisons, works for GLM...

8.4 Benchmarking

Benchmarking is the process of running an experiment to obtain information about some aspect of hardware and/or software performance. The performance measure of interest may be as simple as checking that the system completes a specified task within a given amount of time or may involve complex measurements of resource usage.

Common reasons for benchmarking in software engineering include comparing before/after performance and obtaining numbers to put in a report. For a more general audience benchmarking information is often used as input to a selection process and as such often has a marketing orientation. Accurate measurements are not always necessary, showing that a system is good enough may be good enough.

The time taken for operations such as add and multiply was used to compare the performance of early computers.[610,611] Studies by Knight[327,328] calculated performance based on a weighted average of instruction times, based on the kinds of instructions executed by commercial and scientific programs. Based on a study of over 300 computers available between 1944 and 1967 the rental cost for performing an operation decreased with increasing computer performance; a power law with an exponent between two and three (see Figure 8.14 and also rexample[EvolvingCompPerf_1963-1967.R]).

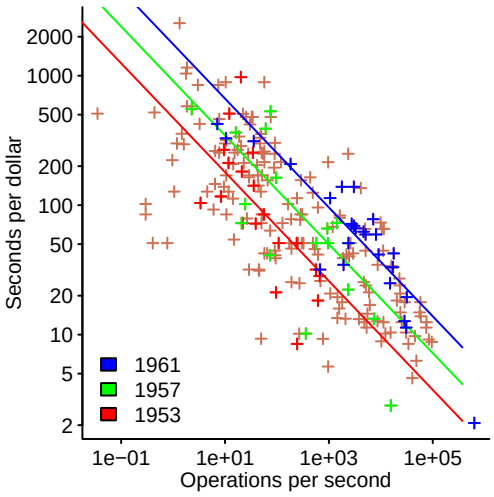


Figure 8.14: Performance and rental cost of early computers, with straight line fits for a few years. Data from Knight.[327] code

On modern computing platforms, obtaining accurate benchmark data for many kinds of question is likely to be economically infeasible.[x] A consequence of the continual reduction in the size of components within microprocessors, see Figure 8.15 and Figure **??**, is that individual components are now so small that variations in the fabrication process (e.g., differences in the number of atoms added or removed during the fabrication process) can noticeably change

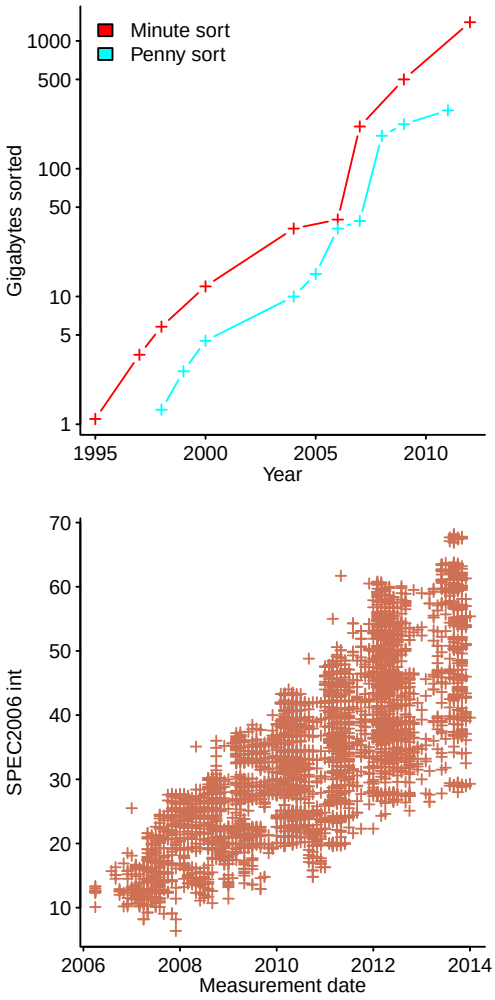


Figure 8.16: Maximum number of records sorted in 1 minute and using 1 penny's worth of system time (upper). SPEC2006 integer benchmark results (lower). Data from Gray et al[227] and SPEC.[534] code

their geometry, leading to large variations in the runtime electrical characteristics of supposedly identical devices.[56]

Manufacturers offer computing systems having a range of different performance characteristics; the lower plot in Figure 8.16 shows all published results for the integer SPEC2006 benchmark. While hardware performance has now improved to the point where for many uses it appears to be good enough, it is still possible to buy hardware that is under-powered for the job it is expected to perform (the upper plot in Figure 8.16 shows the orders of magnitude improvements in cost and performance of sorting over 15 years).

Benchmark results published for general consumption should be treated with extreme caution, they are often little more than marketing information. The author(s) may have reasons for wanting to create a favourable impression for one system in preference to others or may just have done a sloppy job (because of inexperience, incompetence or lack of resources). One class action suite alleged that:[200] `Intel used its enormous resources and influence in the computing industry to, in Intel's own words, "falsely improve" the Pentium 4's performance scores. It secretly wrote benchmark tests that would give the Pentium 4 higher scores, then released and marketed these "new" benchmarks to performance reviewers as "independent third-party" benchmarks. It paid software companies to make covert programming changes to inflate the Pentium 4's performance scores and even disabled features on the Pentium III so that the Pentium 4's scores would look better by comparison.'[xi]

In some consumer goods markets, product benchmark results attract a lot of publicity, with potential customers being influenced by the results achieved by similar products. A study by Shimpi and Klug[19] of Android benchmarks found that some mobile phone vendors detected when a particular benchmark was being run and raised the devices thermal limits (allowing the system clock rate to run faster for longer; a 4.4% performance improvement was measured).

In published benchmark results the Devil is in the detail, or more often in the lack of detail, as illustrated by the following:

- Bailey[38] lists twelve ways in which parallel supercomputer benchmarks have been written in a way likely to mislead readers, including: quoting 32-bit, not 64-bit results, quoting figures for the inner kernel of the computation as if they applied to the complete application and comparing sequential code against parallelized code.
- Citron[111] analysed the ways in which many research papers using the SPEC CPU2000 suite have produced misleading results by only using a subset of the benchmark programs (only 23 out of 115 papers surveyed used the whole suite). In one case a reported speed up of 1.42 is reduced to 1.16 when the whole suite is included in the analysis (reduction from 1.43 to 1.13 in another and from 1.76 to 1.15 in a third).

The primary purpose of this section is to highlight the many sources of variability present in modern computing systems. The available evidence suggests that large variations in benchmark results are now the norm. Large variations in measured performance do not prevent accurate results being obtained, they increase the time and money needed (i.e., it is simply a case of making enough measurements). Advice on how to perform benchmarks is available elsewhere.[174,240]

8.4.1 Following the herd

When choosing a benchmark, there is a lot to be said for doing what everybody else does, advantages include:

- can significantly reduce the cost and time needed to obtain benchmark data,
- an established benchmark is likely to be usable out-of-the-box. It takes time for a benchmark to become established; an analysis[444] of one Java source code corpora was only able to build 86 of the 106 Java systems in the corpus (and 56 of these had to be patched to get them to build),

[x] Obtaining accurate benchmark results has always been an expensive and time consuming process, but at least it used to be possible to count on devices sharing the same part number having the same performance characteristics.

[xi] The class action was settled[554] with Intel agreeing to pay $15 to Pentium 4 purchasers, $4 million to a non-profit entity and an amount not to exceed $16,450,000 to the lawyers who brought the suit.

- it is an easier sell to the result to audiences when the benchmark used is known to them.

The problem with following the herd is that there may be fitness-for-purpose issues associated with using the benchmark, i.e., herd behavior is adapted to environments that may be substantially different from the environment in which the system will operate. For instance, the SPEC benchmark is often used for comparing compiler performance, while SPEC is designed for benchmarking processor performance not compiler performance.

Commonly used benchmarks suffer from vendors tuning their products to perform well on the known characteristics of the benchmark. The SPEC benchmark has been used over many years for compiler benchmarking and compiler vendors often use it in-house for performance regression testing.

8.4.2 Variability in today's computing systems

In the good old days computer performance tended to be relatively consistent across identical, but physically different, components, i.e., the same model of cpu or memory chip. Also, software often had relatively few options that could significantly alter its performance characteristics.

A consequence of the components in modern hardware being fabricated using handfuls of atoms is that process variations of an atom or two here and there can produce devices that are apparently identical, but have surprisingly different performance characteristics. Further reductions in the number of atoms used to fabricate devices will lead to greater variations in the final product. Today's consumer of benchmark results has to chose between:

- accepting a wide margin of error,
- requiring that a benchmark be executed very many times to ensure there are enough measurements to obtain the desired statistical confidence interval for the results; this approach also involves checking that a wide range of, possible unknown, factors are controlled for by those running the benchmark.

Intrinsic variability in system performance has implications for companies that regularly monitor the performance of their product during ongoing development. For instance, Mozilla regularly measures the performance of the latest checked-in version of source code of Firefox, if an update results in a decrease in performance that exceeds a predefined limit, it update is rolled back. Successful implementation of such a policy requires that the impact of external factors on performance are carefully controlled.

Performance variation has to be addressed from a system wide perspective,[xii] hardware/-software interaction can have a significant performance impact and there are often multiple, independent, sources of variation.

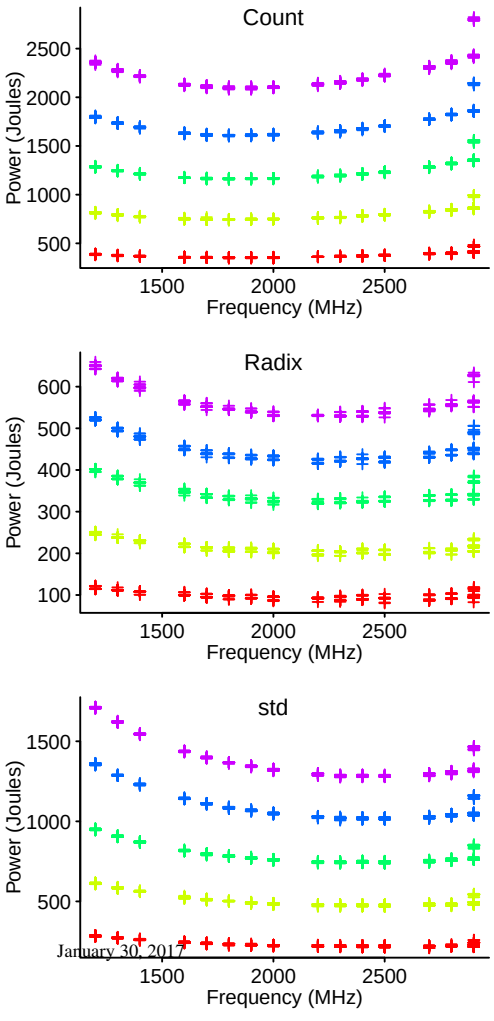
At the systems level differences in component characteristics (e.g., differences in system clock frequency drift in multiprocessor systems[280]) can interact to produce emergent effects.

DVFS (Dynamic Voltage and Frequency Scaling) provides an example of how the complexities of system component interactions make it difficult to reliably predict performance. As its name suggests DVFS allows processor voltage and frequency to be changed while programs are running and an analysis of system power consumption[622] concludes that total power consumed, executing a program from start to finish, is minimised by running the processor as fast as possible (assuming there is no waiting for user input).

A study by Götz, Ilsche, Cardoso, Josef Spillner, Aßmann, Nagel and Schill[220] investigated how the total system power consumed by implementations of various algorithms varied with cpu clock frequency, with the intent of finding the frequency which minimised power consumption.

Figure 8.17 shows that total power consumption does not decline with frequency, there is a frequency below the maximum that minimises power consumed.[596] The power minimisation frequency depends on the implementation of the sorting algorithm and the difference between minimum and maximum depends on the number of items being sorted. Predicting the power consumed[60] by a program is obviously a non-trivial problem.

Programming languages are starting to support constructs that provide developers with options for dealing with power consumption issues.[497]



[xii] The following two sections separately discuss performance variation whose root cause in hardware or software; this is for simplicity of presentation.

8.4.2.1 Hardware variation

This section outlines the evidence for large variations in hardware performance. Much of the detailed data in the following analysis was obtained using components manufactured five to ten years before this book was published. The major computing hardware components include:

- CPU: power consumption and instruction counts,
- Main storage: hard disc performance and power consumption,
- Memory: performance and power consumption,
- Network performance,
- Aging of components.

The Intel Haswell processor is the latest iteration (at the time of writing) of the growth in processor variability.[242] This processor contains integrated power regulators per core, allowing each core to operate at a difference voltage and frequency. The temperature of each core is monitored to ensure it keeps within thermal constraints, with voltage/frequency reductions when necessary. Use of the AVX instructions requires higher voltages and the processor automatically reduces clock frequency to keep within thermal limits; return to non-AVX operating mode takes 1 ms and I'm sure that in the coming years research will uncover some surprising consequences of this delay.

When computing devices obtain their electricity from the mains power supply, there is rarely a need to be interested in the supply characteristics. Batteries have characteristics that can affect the performance of devices connected to them, such as level of power delivery depending on current charge state and power draw frequency characteristics. In a mobile computing environment power consumption can be just as important, if not more so, than performance. Peltonen et al[446] is a public dataset of power consumption on 149,788 mobile devices made up of 2535 different Android models; Jongerden[306] analyses various models of battery powered systems and Buchmann[83] covers rechargeable batteries in detail.

In mobile devices a large percentage of power is consumed by the display; optimization display intensity and choice of color,[539] while an app is running, is not discussed here.

CMOS (complementary metal-oxide-semiconductor) is the dominant technology used to fabricate computing devices, and until a so-called beyond-CMOS device[413] technology becomes commercially viable, only the characteristics of CMOS devices are considered.

CPU Developers have generally considered the processor executing their code to be interchangeable with any of the other mass-produced etched slices of silicon stamped with the same model number; while never exactly true, deviations from this interchangeability assumption were once small enough to only be of interest within specialised niches, such as hardware modders interested in running systems beyond rated limits e.g., higher clock rates.

The micro-architecture of modern processors has become so complicated that apparently minor changes to an instruction sequence can have a major impact on performance;[279] a trivial change to the source code or the use of a different compiler flag is enough.

Power consumption and clock frequency are intimately connected processor characteristics. Increasing clock frequency increases power consumption (a good approximation for processor power consumption is $P = \alpha FV^2 + I_0V$, where F is the clock frequency, V is voltage supplied to the cpu and I_0 is leakage current). Processors clocked at the same frequency execute instructions at the same rate. However, variations in the number of atoms implementing internal circuitry produces variations in power consumption. Some processors reach maximum operating temperature more quickly than others; to prevent overheating destroying the device, power consumption is reduced by reducing the clock rate. Different processors have different sustained performance rates because of differences in their power consumption characteristics.

Vogeleer[137] is a readable technical discussion of modeling low level temperature/power relationships for the kind of processors used to run applications.

rexample[data-rbook/Exynos-7420/]...

A study by Wanner, Apte, Balani, Gupta and Srivastava[602] measured the power consumed by 10 separate Amtel SAM3U microcontrollers (derived from an ARM's Cortex M3 processor core) at various ambient temperatures. Figure 8.18 shows a 5-to-1 difference, between supposedly identical processors, in power consumption when in sleep-mode and around 5% difference when operating at 4MHz.

Another example of power variations, involving the Intel Core processor, is discussed in the section on building mixed-effect models.

Any power related benchmark made using a single instance of a processor is a sample drawn from a population that could vary by 10% or more when executing code and several hundred percent when idling. The extent to which results based on this minimum sample is of practical use will depend on the questions being asked. If the power consumption characteristics of the population of a particular CPU is required, then it is necessary to benchmark a sample containing an appropriate number of *identical* processors.

Power consumption measurements are often implemented by periodically sampling the voltage across a known resistance. A study by Saborido, Arnaoudova, Beltrame, Khomh and Antoniol[496] investigated the measurement error introduced by different sampling rates. Figure 8.19 shows the power spectrum of the ??? App sampled at 500K per second. This very high sampling rate makes it possible to see the noticeable peak in power consumed by very short lived events, something that low frequency sampling would not detect (the paper lists error estimates for lower sampling rates).

In theory counting the instruction executed by a program is a means of obtaining, power independent, answers to questions about comparative program performance. Many processors include hardware containing counts of operations performed, e.g., instruction opcodes executed and cache misses; however, the purpose of these counters is to help manufacturers debug their processors, not provide end-user functionality. Consequently, counter values are not guaranteed to be consistent across variants of processors within the same family[606] and fixing faults in the counting hardware does not have a high priority (e.g., counting some instructions twice or not at all; Weaver and Dongarra[606] found that in most cases the differences were a fraction of a percent of the total count, but for some kinds of instructions, such as floating-point, the counts were substantially different).

A study by Melhus and Jensen[384] showed that address aliasing, of objects in memory, could have a huge impact on the relative values of some hardware performance counters.

Counting the number of instructions executed by a program begs the question of how calls into operating system routines, which may execute at higher privilege levels, are counted. Also, the execution time of some instructions will depend on other instructions executed at around the same time (modern processors have multiple functional units and allocate resources based on the instructions currently in the pipeline), the time taken to execute other instructions can be very unpredictable (e.g., time taken to load a value from memory depends on the current contents of the cache and other outstanding load requests). A study by Weaver and McKee[607] found that it was possible to adjust for the known faults in the hardware counters (see rexample[iiswc2008-i686.R]).

Hardware counters need not be immune to *observer effects*; in particular, the values returned can depend on the number of different hardware counters being collected.[404]

Main storage Traditionally main storage has meant hard disks (sometimes backed-up with tape), but solid state devices (SSD) are rapidly growing in capacity[556] and importance; for extremely large capacity magnetic tape is used: this niche use is not discussed here.

Data is read/written to a hard disk by moving a magnetic sensor across a rotating surface. These spatial movements create a correlation between successive operations, e.g., the time taken to perform the second read will depend on its location on the disk relative to the first. Disk access issues have been studied for around 50 years and techniques such as data buffering, by the operating system, and reordering requests to optimise overall throughput, by the device driver or firmware, are well established.[?]

Figure 8.20 shows that data near the outside of one (not unusual) disk family is read approximately twice as fast as data located near the center. This offset dependent performance has always existed in some form and its impact on benchmark performance is very difficult to estimate, depending on the history of the data that is added and deleted.

Storage farms organise files so that those likely to be most likely to be accessed are stored on the outer tracks while files less likely to be accessed are stored on the inner tracks.

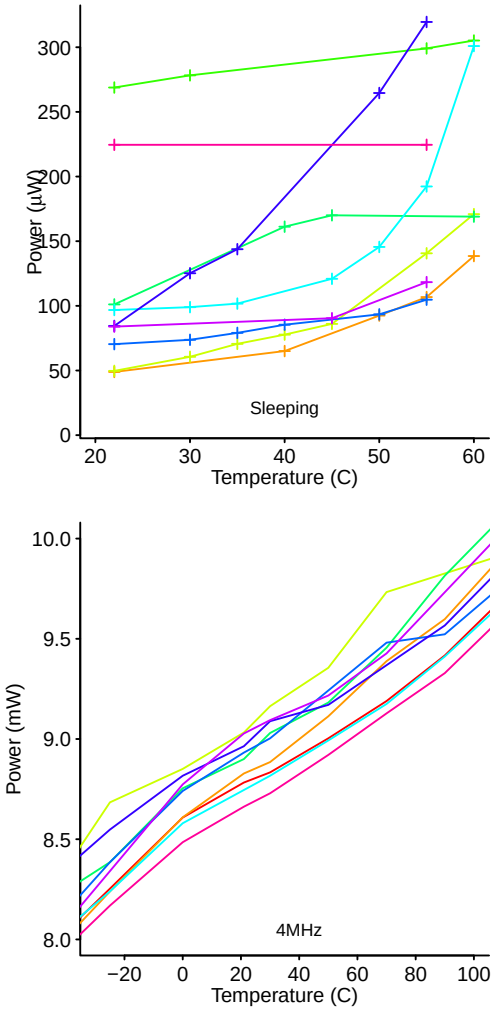


Figure 8.18: Power consumed by 10 Amtel SAM3U microcontrollers at various temperatures when sleeping or running. Data from Wanner et al.[602] code

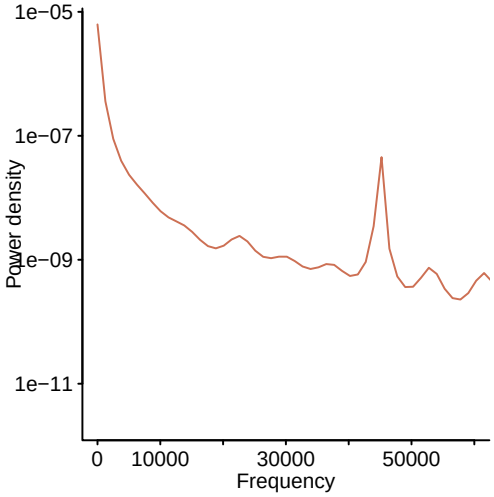


Figure 8.19: Power spectrum of electrical power consumed by an app running on a ???. Data from Saborido et al.[496] code

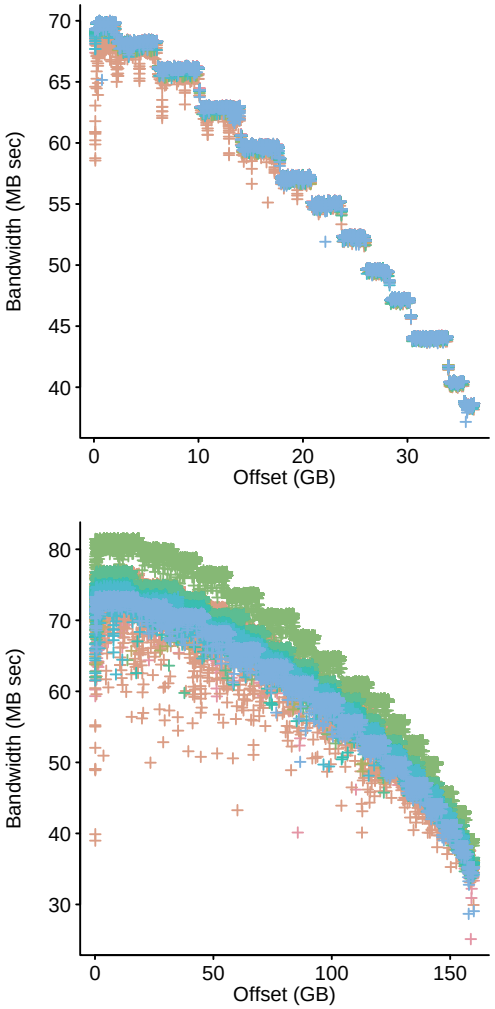
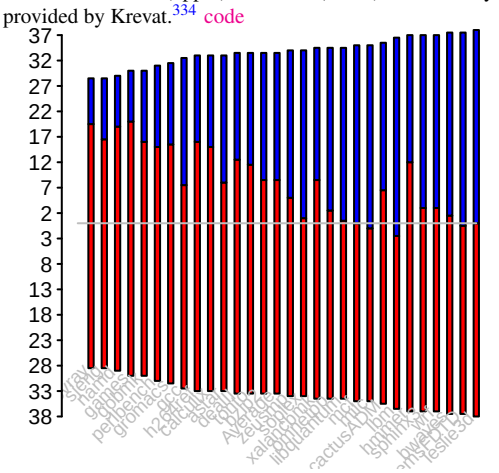
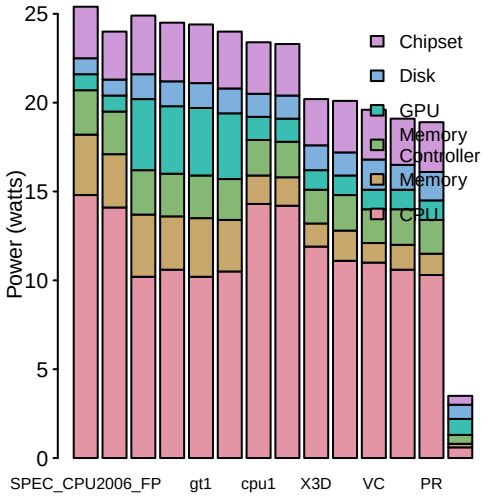


Figure 8.20: Read bandwidth at various offsets for new disks sold in 2002 (upper) and 2006 (lower). Data kindly provided by Krevat.[334] code





The continuing increase in the number of bits that can be stored within the same area of rotating rust has been achieved by reducing the size of the magnetic domain used to store a bit. Like silicon wafer production, variations in the fabrication process of disc platters can now result in large differences in the performance of supposedly identical drives. A growing percentage of disks are used in data centers and at some point manufacturers may decide to concentrate on designing drives for this market.[77]

A study by Krevat, Tucek and Ganger[334] measured the performance of disk drives originally sold in 2002, 2006, 2008 and 2009. In Figure 8.20 the staircase effect is a result of zoning? (disks spin at a constant rate and in the same time interval more data can be read from the area swept out near the outer edge of a platter than from one near the inner edge).

The upper plot shows the performance of nine disks from 2002, each displayed using a different color and there is little variation between different disks (fitting a regression models shows that disk identity is not a significant, p-values around 0.2, predictor of performance).

The lower plot is nine disks from 2006, each displayed using a different color and the visibility of different colors shows there is a noticeable variation between different disks (fitting a regression models shows that disk identity is a significant, p-values around 10^{-16}, component of performance prediction).

To increase recording density, drive manufacturers are now using Shingled Magnetic Recording (SMR), where tracks overlap like rows of shingles on a roof. Singled discs have very different performance characteristics,[4] but little data is publicly available at the time of writing.

SSDs are sufficiently new that little performance data is publicly available at the time of writing.

A study by Kim[322] ran eight different benchmarks on SSD cards from nine different vendors. The range of performance values was different for both vendors and benchmark, meaning that in their original form neither of these variables are of any use in modeling performance. However, if the performance on each benchmark is normalised across all vendors (e.g., by transforming into the range zero to one) it might be possible to fit a model using vendor as the single explanatory variable (see rexample[hyojun.R]). The results show that only one vendor has a sufficiently consistent performance (that appears to be significantly better than the other vendors) to be included in a model, with all other vendors appearing to have the same performance.

Memory Memory chips tend to be thought about in terms of their capacity and not their performance (e.g., read/write delays or power consumption). Performance is governed by access rate and by the number of bytes transferred per access, with accesses usually made via some form of memory control chip (the capabilities of this controller have a significant impact on performance). Many motherboards provide options to select memory chip timing characteristics.

The upper plot in Figure 8.21 breaks down average power by CPU and memory when running the SPEC CPU2006 benchmark, while the lower plot breaks the power down by the major subcomponents of a server running various programs.

The storage hierarchy...

The following are examples of memory chip characteristics that have been found to noticeably variation:

- a study by Gottscho, Kagalwalla and Gupta[219] measured power consumption variability of 13 DIMMs of the same model of 1G DRAM from four vendors. The variation about the mean, at one standard deviation, was 5% for read operations, 9% for write and 7% for idling (see rexample[J20_paper.R]),
- a study by Schöne, Hackenberg and Molka[504] found that memory bandwidth was reduced by up to 60% as the frequency of the cpu was reduced and that memory performance characteristics varied between consecutive generations of Intel processors and even between server and desktop parts,
- a study by Gottscho[218] measured the power consumption of 22 DDR3 DRAMs, manufactured in 2010 and 2011, from four vendors. Read operations consumed around 60% of the power needed for write operations, with idle consuming around 40%; the standard deviation varied from 10% to 20%. The power consumed also varied with value being read-/written, e.g., writing 1 to storage containing a 0 required 25% more power than writing a 0 over a 1 (see rexample[MSTR10-DIMM.R] for data).

The variability of memory chip performance is likely to increase as vendors start to reduce power consumption and improve performance by lengthening DRAM refresh times, optimising each computer by tuning it to the unique characteristics of the particular chips present in each system.[350]

Chandrasekar[99] provides a detailed discussion of DRAM power issues and includes code for a tool to obtain detailed information about the memory chips installed on a system.

Network performance A study by Schad, Dittrich and Quiané-Ruiz[500] submitted various benchmarks as jobs to Amazon's Elastic Computing...

Aging emailed for data...

8.4.2.2 Software variation

This section outlines the evidence for large variations in software performance. The following software characteristics are briefly covered:

- The environment: interaction with the environment, file system, support libraries and aging,
- Configurations,
- Creating an executable: compiler optimization and link order,
- Tools.

The environment The environment within which programs execute often contains a complicated ensemble of interconnecting processes and services that cannot be treated as independent standalone components. One consequence of this interconnectedness is that the order in which processes are initiated during system startup can have a noticeable impact on system performance.

The impact of prior history on program performance is seen in a study by Kalibera, Bulej and Tůma[316] who measured the execution time of various programs. Figure 8.22 shows 10 iterations of the procedure: reboot computer and make 2,048 performance measurements. The results show performance variation after each reboot is around 0.1%, but a reboot can cause a shift of 3% in the average performance (the ordering of processes executed during system startup varies across reboots, due to small changes in the time taken to execute the many small scripts that are invoked during startup).

A later study[271] found that the non-determinism of initial program execution, in this case, could be reduced by having the operating system use cache-aware page allocation.

Environmental interactions are not always obvious. A study by Mytkowicz, Diwan, Hauswirth and Sweeney[404] increased the number of bytes occupied by a Linux environment variable between runs of the perlbench program; the results from each of 15 executions were recorded, an environment variable increased in size by one character and this process repeated 100 times.

Figure 8.23 shows the percentage change in performance, relative to the environment variable containing zero characters, at each size of environment variable, along with 95% confidence intervals of the mean of each 15 runs.

Incremental operating system updates can change program performance. A study by Flater[184] compared the performance of cpu intensive and I/O bound programs on two different versions of Slackware running on the same hardware (versions 14.0 and 14.1, using Linux kernels 3.12.6 and 3.14.3 respectively). The results show consistent differences in performances of up to 1.5% (rebooting did not have any significant impact on performance).

Many systems allow multiple programs to be executing at the same time, sharing system resources. Sharing becomes a performance bottleneck when one program cannot immediately access resources when it requests them; access to memory is a common resource contention issue on multi-processing systems.

Figure 8.24 shows changes in SPEC CPU2006 benchmark performance caused by cache and memory bus resource contention on a dual processor Intel Xeon E5345 system (from a study by Babka[31]).

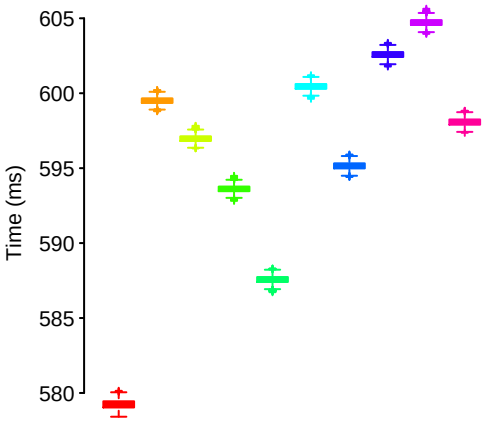


Figure 8.22: FFT benchmark executed 2,048 times followed by system reboot, repeated 10 times. Data kindly provided by from.[316] code

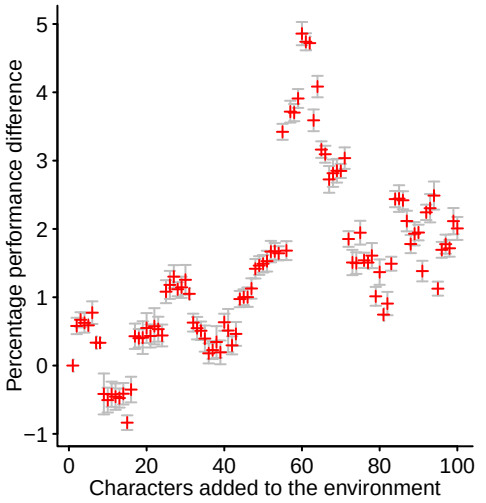


Figure 8.23: Percentage change, relative to no environment variables, in perlbench performance as characters are added to the environment. Data extracted from Mytkowicz et al.[404] code

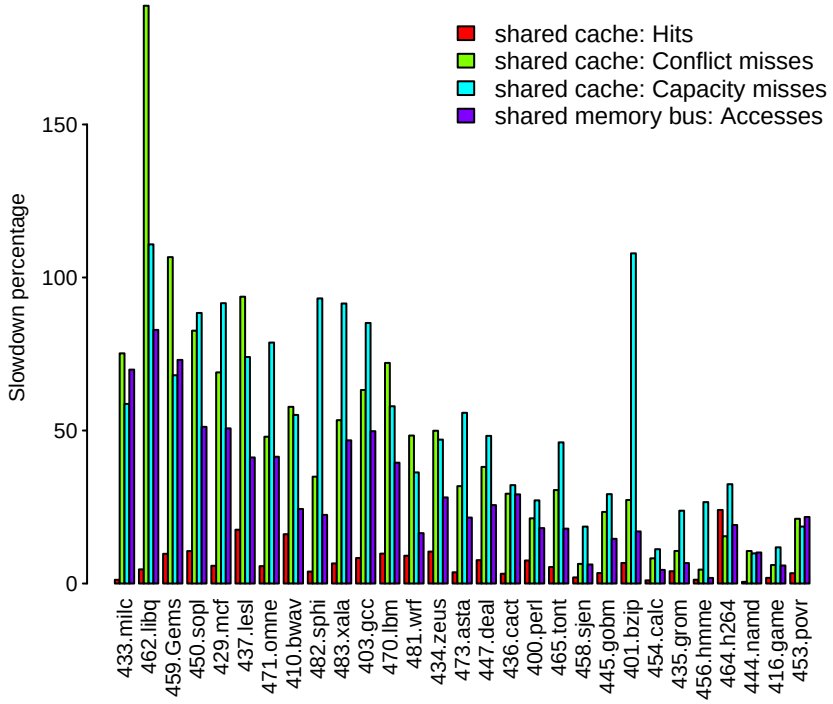


Figure 8.24: Changes in SPEC CPU2006 benchmark performance c on for one dual pr kindly provided by Babka.[31] code

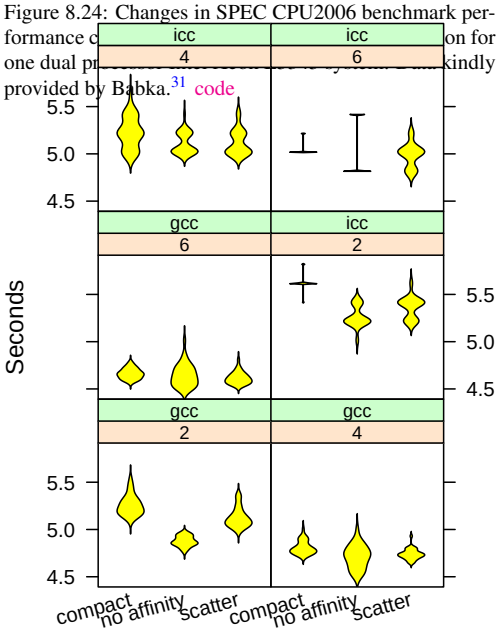


Figure 8.25: Execution time of `330.art_m`, an OpenMP benchmark program, using different compilers, number of threads and setting of thread affinity. Data kindly provided by Mazouz.[374] code

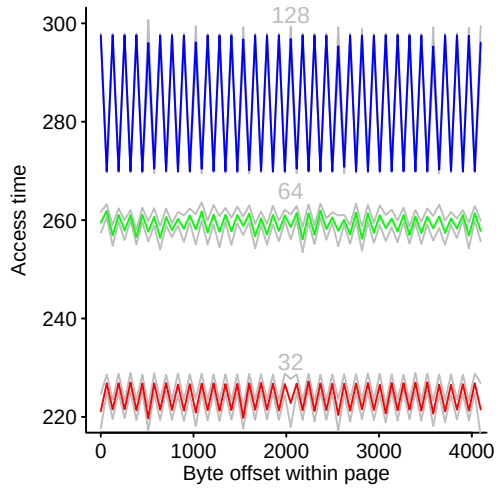
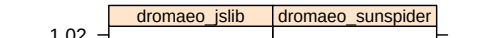


Figure 8.26: Access times when walking through memory using three fixed stride patterns (i.e., 32, 64 and 128 bytes) on a quad-core Intel Xeon E5345; grey lines at one standard deviation. Data kindly provided by Babka.[32] code



A study by Mazouz[374] investigated the performance of the SPEC OpenMP 2001 benchmark programs, compiled using gcc 4.3.2 and icc 11.0, running on multicore devices. It is possible for a program's code to execute on a different core after every context switch. Allowing the operating system to select the core to continue program execution is good for system level load balancing, but can reduce the performance of individual programs because recently accessed data is less likely to be present in the cache of the newly selected core. *Thread affinity* is the process of assigning each thread to a subset of cores, with the intent of improving data locality i.e., recently accessed data is more likely to available in accessible caches.

Figure 8.25 shows the time taken to execute one program in 2, 4, and 6 threads, with thread affinity set to scatter (distribute the threads evenly over all cores), no affinity (allow the OS to assign threads to cores) and compact (threads share an L2 cache), each repeated 35 times.

Configuring the system being benchmarked to only run one program at a time solves some, but not all, cache contention issues. Walking through memory, in a loop, may result in a small subset of the available cache storage being used (main memory is mapped to a much smaller cache memory, which means that many main memory addresses are mapped to the same cache address). Figure 8.26, from a study by Babka and Tåma,[32] shows the effect of walking through memory using three different fixed width strides; for 32 and 64 byte strides accesses to even cache lines is faster than odd lines, with the pattern reversed for a 128 byte stride.

Operating systems generally have background processes that spend most of the time idling, but wake up every now and again. A background processes that wakes up will consume system resources and can have an impact on the performance reported by a benchmark, they are a source of variation.

A study by Larres[347] investigated how the performance of one version of Firefox changed as various operating system features were disabled (the intent being to reduce the likelihood that external factors added noise to the result). The operating system features modified were: 1) every process that was not necessary was terminated, 2) address-space randomization was disabled, 3) the Firefox process was bound exclusively to one cpu, and 4) the Firefox binary was copied to and executed from a RAMDISK.

The Talos benchmark was used (the performance testing framework used by Mozilla) and every program was run 30 times. Figure 8.27 shows the performance of various programs running in original and stabilised (i.e., low-noise) configurations.

Those users interested in consistent performance will want to minimise the variation in benchmark results (which did occur for some programs), while users interested in actual benchmark performance will be interested that significant changes in the mean occurred for some programs.

File systems File systems are a way of organising information on a storage device. The traditional view of a file being just a leaf in a directory tree has become blurred, with many file system managers now treating compressed archived files (e.g., zip files) as-if they had a directory structure that can be traversed and Microsoft's .doc format containing a FAT (File Allocation Table, just like a mounted Windows file system) that can refer to contents that may be distributed outside of the *file*.

A study by Zhou, Huang, Li and Wang[629] looked at the performance interplay between file systems and Solid State Disks (SSD) by running a file-server benchmark on a Kingston MLC 60GB SSD. Four commonly used Linux filesystems (ext2, ext3, reiserfs and xfs) were mounted in turn using various options, e.g., various block sizes, noatime, etc.

Figure 8.28 shows the number of operations per second for a file-server benchmark (see paper and data for other benchmarks). The data is poorly fitted by a linear regression model (i.e., not significant on the filesystem or mount options; see rexample[filesystem-SSD.R]).

A study by Sehgal, Tarasov and Zadok[511] compared the power used when four commonly used filesystems were mounted in various ways, e.g., fixed vs. variable sector size, different journal modes, etc. Various server workloads running on Linux were measured; web server power consumption varied by a factor of eight, mail server by a factor of six and file and database by a factor of two.

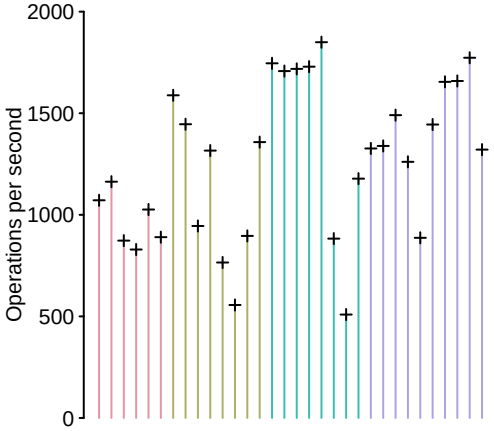


Figure 8.28: Operations per second of a file-sever mounted on one of ext2, ext3, rfs and xfs filesystems (same color for each filesystem) using various options. Data kindly supplied by Huang.[629] code

Creating an executable Many applications are built by translating source code to an executable binary, with the translation tools often supporting many options, e.g., gcc supports over 160 different options for controlling machine independent optimization behavior. Compiler writers strive to improve the quality of generated code and it is to be expected that the performance of each release of a compiler will be different from the previous one; there have been around 150 released versions of gcc in its 30 year history.

A study by Makarow[370] measured the performance of nine releases of gcc made between 2003 and 2010, on the same computer using the same benchmark suite (SPEC2000), at optimization levels 02 and 03.

Figure 8.29 shows the percentage change in SPEC number, relative to version 4.0.4, for the 12 integer benchmark programs compiled using six different versions of gcc. SPEC has a long history if being used for compiler benchmarking and it is possible that all the versions of gcc used for this comparison have already been tuned to do well on this benchmark, meaning there is little, benchmark specific, improvement to be had in the successive versions used in this study.

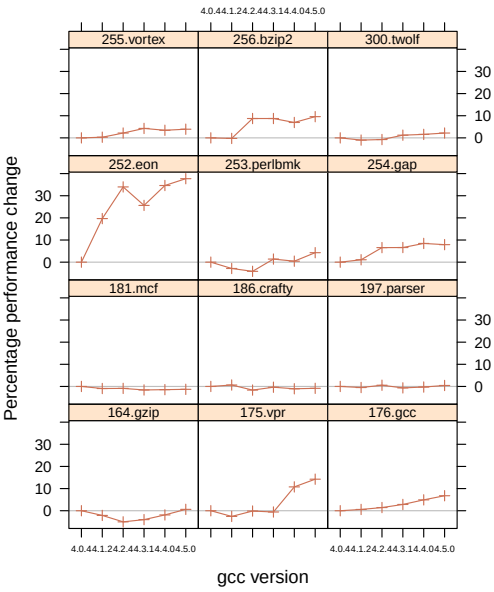


Figure 8.29: Percentage change in SPEC number, relative to version 4.0.4, for 12 programs compiled using six different versions of gcc (compiling to 64-bits with the 03 option). Data from Makarow.[370] code

The following `summary` output is from a mixed-effect model with the random effect on the intercept and slope: code

```
Linear mixed model fit by REML ['lmerMod']
Formula: value ~ gcc_version + (gcc_version | Name)
   Data: lme_02

REML criterion at convergence: 400.6

Scaled residuals:
    Min      1Q  Median      3Q     Max
-2.7256 -0.2748 -0.0683  0.3039  4.3372

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 Name     (Intercept) 1192.795 34.537
          gcc_version    3.155  1.776   -1.00
 Residual                8.632  2.938
Number of obs: 72, groups:  Name, 12

Fixed effects:
            Estimate Std. Error t value
(Intercept) -29.7469    11.0553  -2.691
gcc_version   1.4126     0.5513   2.562

Correlation of Fixed Effects:
            (Intr)
gcc_version -0.997
```

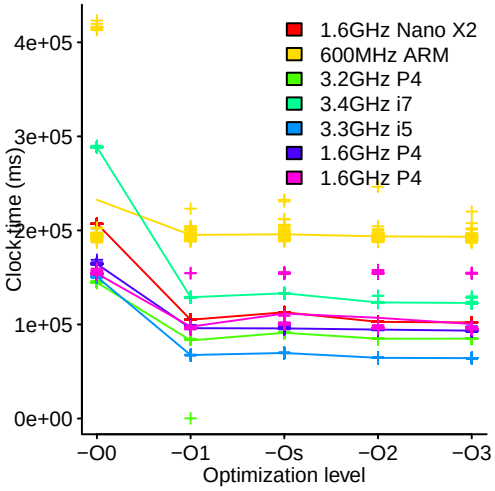


Figure 8.30: Execution time of xy file compressor, compiled using gcc using various optimization options, running on various systems (lines are mean execution time when compiled using each option). Data kindly supplied by Petkovich.[135] code

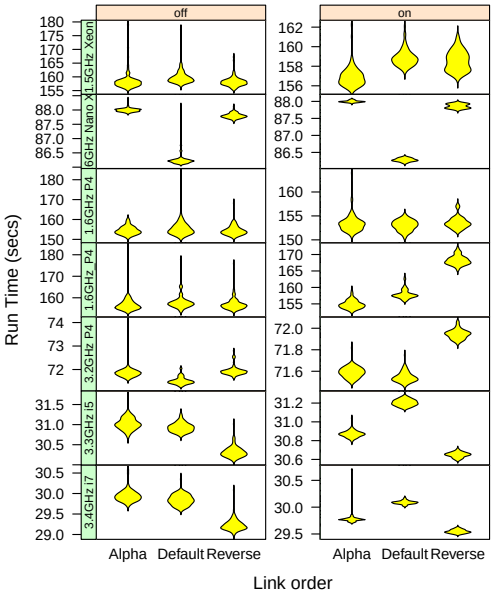
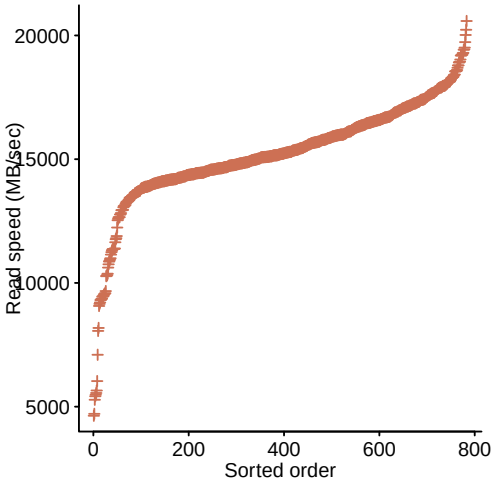
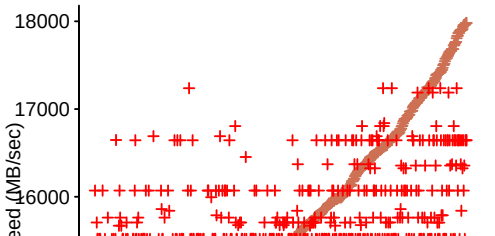


Figure 8.31: Execution time of Perlbench, from SPEC benchmark, on six systems, when linked in three different orders and address randomization on/off. Data kindly supplied by Reidemeister.[135] code





The general picture paint by the model results is a small improvement with each gcc release is swamped by the size of the random effects, while the picture pained by Figure 8.29 is of some releases having a large impact on some programs.

A study by de Oliveira, Petkovich, Reidemeister and Fischmeister[135] investigated the impact of compiler optimization and object module link order on program performance.

Figure 8.30 shows the time taken by the xy file compression program, compiled by gcc using various optimization options, to process the Maximum Compression test set on various systems. The results show that different optimization levels have a different performance impact on different systems (the lines would be parallel if optimization level had the same impact for each system).

Compiling is the first step in the chain of introducing variability into program performance, the next step is linking. Figure 8.31 shows execution time of Perlbench (one of the SPEC benchmark programs), on six systems, when the object files used to build the executable are linked in three different orders and with address randomization on/off.

Some systems share a consistent performance pattern across link orderings and some systems are not affected by address randomization. But there is plenty of variation across all the variables measured.

Tools Dynamic profiling tools such a `grpof` work by interrupting a program at regular intervals during execution (e.g., once every 0.01 seconds) and recording the current code location (often at the granularity of a complete function). The results obtained can depend on interrupt frequency and the likelihood of being in the process of calling/returning from the profiled function.[183]

8.4.2.3 End user systems

Benchmark data derived from end user systems is likely to be subject to numerous known and unknown unknowns.

PassMark Software specializes in benchmark solutions[620] and over the years has collected 800,000+ benchmark results from Microsoft Windows based computers; David Wren kindly supplied the 10,000 memory benchmark results used in the following analysis, as well as insights into possible reasons for the performance characteristics seen.

The results obtained from end users running a benchmark on one or more of their computers will depend on the characteristics of the hardware and the software executing at the time the benchmark is run. Background processes that may be running on a Windows machine include Internet based toolbars, anti-virus systems and general OS housekeeping processes.

The upper plot in Figure 8.32 shows the results (in ascending order) for 783 systems containing an Intel Core i7-3770K processor (whose official clock speed is 3.5GHz, some users may be overclocking); the lower plot has had the values at each end trimmed by around 10% and the red dots are predictions from a regression model built using information on the characteristics of the memory chips in each computer as explanatory variables.

The scattering of red dots around the regression line illustrates how poor the predictions can be from a regression model that explains about 30% of the variance in the data. This data is another example (see Figure 8.9) of the wide range of performance reported for apparently very similar end user systems.

8.4.2.4 The cloud

Cloud computing is becoming a popular platform for running applications that require non-trivial compute resources. The service level agreements offered by cloud providers specify minimum levels of service, e.g., Amazon's June 2013 EC2 terms specify 99.95% monthly uptime.[16] Cloud services general run virtualized instances, which means access to the real hardware may sometimes be shared. Shared hardware access results in user visible performance varying from one run to the next; what are the characteristics of this variation?

A study by Schad, Dittrich and Quiané-Ruiz[500] submitted various benchmarks as jobs to Amazon's Elastic Computing Cloud (EC2) twice an hour over a 31 day period; a variety of resource usage measurements were recorded.

Figure 8.33 shows one set of resource usage measurements from the study, the Unix benchmark utility (Ubench; a cpu benchmark) running on small (upper) and large (lower) EC2 instances located both in Europe (red) and the US (green).

Both plots show more than one distinct ranges of performance. This data is an example of the variation experienced in Amazon's EC2 performance over one particular time period and there is no reason to believe that any subsequent benchmarking will exhibit one, two, three or more distinct performance ranges.

8.5 User interface testing

When working on systems being built by a small group of people, software developers are called on to solve a wide range of computer related issues, including the testing and tuning of the user interface.

The System Usability Scale (SUS)[79,80] is a straight-forward and widely used usability questionnaire that produces a single number for usability. One study[574] comparing five methods of evaluating website usability found that SUS produced the most consistent results for smaller sample sizes.

User interface design...?

8.6 Surveys

A lot of software engineering information only exists in the heads' of the people who build software systems. Obtaining this information requires asking these people questions and analysing their answers. This is the subject of survey analysis.

A survey is essentially an experiment where the questions are used to control the explanatory variables.

The choice of the population to sample, for a survey, is driven by where accurate answers to the questions are most likely to be obtained.

The `survey` package ...

The techniques used to analyse survey results are the same as those used to analyse other kinds of data. However, there are some characteristics that are often encountered in survey data, including the following:

- missing data: people don't answer all the questions or stop answering after some point,
- misleading answers: giving answers that show those involved in a better light, such as job adverts listing trendy topics and languages to attract more applicants,
- spatial information: how subjects are distributed geographically,
- stratification: to increase the number of responses the survey is sent to people with a wider range of characteristics than is strictly applicable, potentially resulting in samples containing clusters having their own distinct characteristics,

Like most experiments, finding the appropriate questions to ask is an iterative process. It is worthwhile running a small survey to obtain feedback on whether the questions produce the desired kind of response, updating the questions based on this feedback and repeating as many often as time and resources allow.

Studies have found[153] that self-assessment of skills and character have a tenuous to modest relationship with actual performance and behavior. The correlation between self-ratings of skill and actual performance in many domains is moderate to meager....

Several studies by your author[300,301,302] included a component that asked developers about how many lines they had read and written during their professional career.

This question requires a lot of thought to answer and there are many ways of adding up the numbers. Does reading the same line twice count as two lines, or one line unless the developer

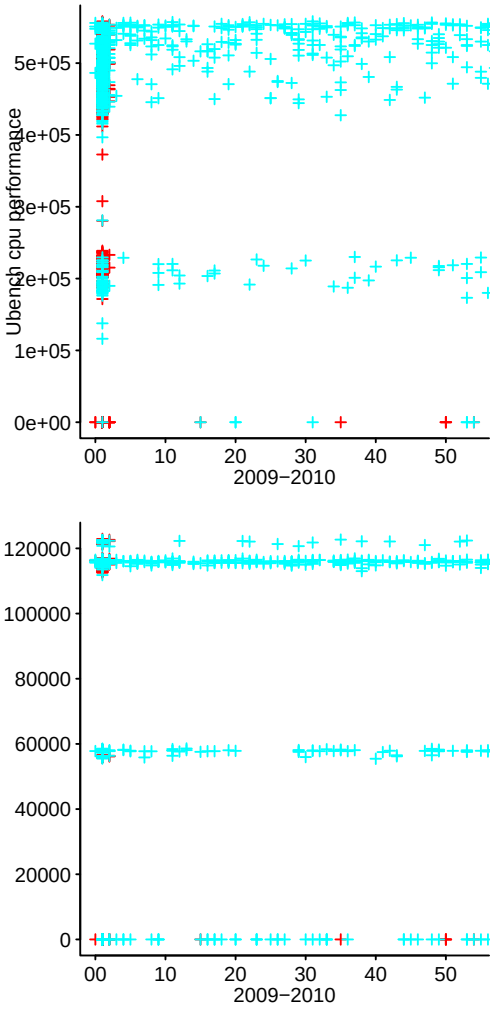


Figure 8.33: Ubench cpu performance on small (upper) and large (lower) EC2 instances, Europe in red and US in green. Data kindly provided by Dittrich.[500] code

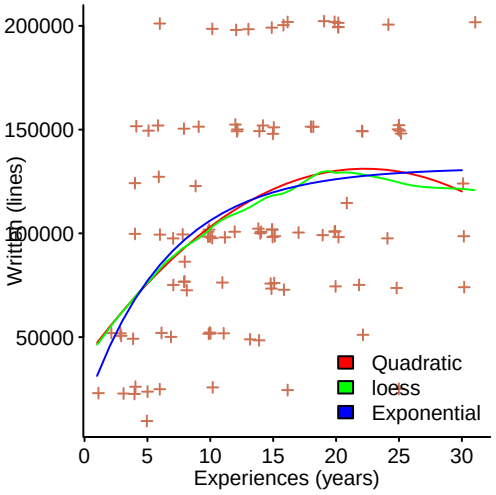


Figure 8.34: Lines of code that 101 professional developers, with a given number of years experience, estimate they have written. Data from Jones[300][301].[302] code

involved had forgotten reading it? How much does visually searching a screen of code for a particular identifier count towards lines read? Counting the number of lines in the programs written by a developer is likely to be underestimate the number of lines they have written; a line of code may be written and then deleted, an existing line may be modified slightly.

The plot below shows the number of lines of code that 101 professional developers estimate they have written. While an exponential model may fit the data, the variance explained is small...

How well do developers' know a particular language? Answering this question first requires a definition of what it means to know a language. A study by Dietrich, Jezek and Brada[148] investigated one aspect of knowing a language: knowledge of an important component of the language semantics, in this case Java type compatibility.

They started with a thorough set of questions, but quickly found that subjects were not answering many questions; they switched to a shorter list of questions.

As a self administered survey, they had no control over how many questions were answered...

The questions required detailed technical knowledge... breaking down what was asked into subcomponents might have been better than an all or nothing approach to the correct answer...

Allowing subjects to provide more specific values for experience...

REXAMPLE[java-type-quest.R]

Some surveys are made to gain general information about the characteristics of a particular population of interest. The questions are designed to obtain learn about characteristics of this population, e.g., characteristics of open source developers,[486] such as the age started contributing to open source projects.

145 Questions for data scientists in software engineering[49]

How often hardware and software is updated, a basic survey[568]...

1. *What questions to ask?*
   ?,?,?

Questionnaire data on training and related issues... Difference in survey answers between years...

2008 answers:?

2009 answers:?

rexample[embedded_survey/]

8.6.1 Checking survey reports

Does mean age make sense given the sample size..

Solving an underspecified set of equations... Survey with only totals given rexample[Success2007.pdf]...

Chapter 9

Overview of R

This chapter gives a brief overview of R for developers who are fluent in at least one other computer language. The discussion pays attention to language features are very different from languages the reader is likely to be familiar with and concentrate on a few language features that can be used to solve most problems.

The R language is defined by its one implementation; available from the R core team.[469] A language definition[470] is gradually being written.

R programs tend to be very short, compared to programs in languages such as C++ and Java; 100 lines is a long R program. It is assumed that most readers will be casual users of R whose programs generally follow the pattern:

```
d=read_data()
clean_d=clean_and_format(d)
d_result=applicable_statistical_routine(clean_d)
display_results(d_result)
```

If your problem cannot be solved using this algorithm, then perhaps the most efficient solution may be for you, dear reader, to use the languages and tools you are already familiar with to preprocess the data so that it can be analysed and processed using R.

R is a domain specific language whose designers have done an excellent job of creating a tool suited to the tasks frequently performed when analysing the kinds of datasets encountered in statistical analysis. Yes, R is Turing complete, so any algorithm that can be implemented in other programming languages can be implemented in R, but it is designed to do certain things very well with no regard to making it suitable for general programming tasks.

As a language the syntax and semantics of R is a lot smaller than many other languages. However, it has a very large base library, containing over 1,000 functions. Most of the investment needed to become a proficient R user needs to go into learning to use and apply these functions. There are over 6,000 add-on packages available from the CRAN (Comprehensive R Archive Network).

Help on a specific identifier, if any is available, can be obtained using the `?` (question mark) unary operator followed by an identifier. The `??` unary operator, followed by an identifier, returns a list of names associated with that identifier for which a help page is available.

The call `library(help=circular)` lists the functions and objects provided by the package named in the argument.

9.1 Your first R program

Much like in Python, Perl and many other interpreted implementations, R is often run in an interactive mode, where code can be typed and immediately executed (with `"Hello world"` producing the obvious output).

Your first R program has to read some data and plot it, not just print `"Hello World"`. The following program reads a file containing a single column of values and plots them:

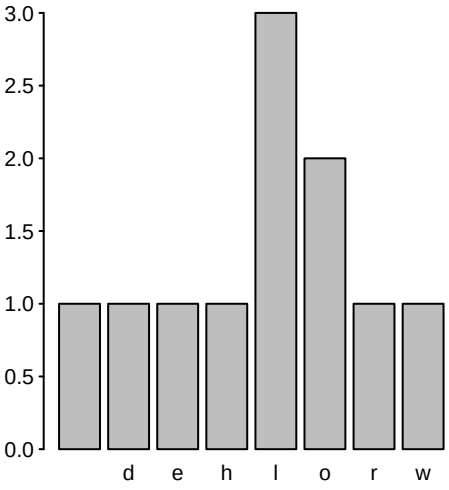


Figure 9.1: Plot produced by hello_world.R program. code

```
the_data=read.csv("hello_world.csv")

plot(the_data)
```

to produce Figure 9.1.

The `read.csv` function is included in the library that comes bundled with the base system (functions not included in this library have to be loaded using the `library` function before they can be referenced; they may also need to be installed via the `install.packages` function) and has a variety of optional arguments (arguments can be omitted if the function definition includes default values for the corresponding parameter). Perhaps the most commonly used optional arguments are `sep` (the character used to separate values on a line, defaults to comma) and `header` (whether the contents of the first line should be treated as column names, default `TRUE`).

The value returned by `read.csv` has class `data.frame`, which might be thought of as a C struct type (it contains data only, there are no member functions as such).

The `plot` function attempts to produce a reasonable looking graphic of whatever data is passed, which for character data is a histogram of the number of occurrences. R is not intended for manipulating low level details and some work is needed to get at the numeric values of the characters which appear in the second plot.

There are a wide variety of options to change the appearance of plot output; these are can be applied on each call to `plot` or globally for every call (using the `par` function).

All objects in the current environment can be saved to a file using the `save` function and a previously saved environment can be restored using the `load` function. When quiting an R session the user is given the option of saving the current environment to a file named `.RData`; if a file of this name exists when R is started, its contents are automatically loaded.

9.2 Language overview

R is a language and an environment. Like Perl, it is defined by how its dominant implementation behaves (i.e., the software maintained by the R project[469]).

R was designed, in the mid-1990s, to be largely compatible with S (a language, which like C, started life in the mid-1970s at Bell Labs). When S was created, Fortran was the dominant engineering language and the Fortran way of doing things had a strong impact on early design decisions, compared to the C way.

The designers of R have called it a functional language and it does support a way of doing things that is most strongly associated with functional program languages (including making life cumbersome for developers wanting to assign to global variables).

The language also contains constructs that are said to make it an object oriented language, and it certainly contains some features found in object oriented languages. OO constructs were first added in the third iteration of the S language and are more of an additional to the functional flavor of the language than a complete make-over. The primary OO feature usage is function overloading when accessing functions from library packages.

Lateral thinking is often required to code a problem in R, using knowledge of functions contained in the base system, e.g., calling `order` to map a vector of strings to a unique vector of numbers.

9.2.1 Differences between R and widely used languages

The following list describes language behaviors that are different from that encountered in other commonly used languages:

- There are no scalars, e.g., `2` is a vector containing one element and is equivalent to writing `c(2)`. The unary and binary operators operate on complete vectors (among other things).

  Many operations that involve iterating over scalar values in other languages, e.g., adding two arrays, can be performed without explicit iteration in R, e.g., `c(1, 2) + c(3, 4)` has the value `c(4, 6)`,

- arrays start at one, not zero,
- matrices and data frames are indexed in row-column order (C-like languages use column-row order),
- case is significant in identifiers, e.g., `some_data` and `Some_data` are considered to refer to different objects,
- some language constructs implemented via specific language syntax in other languages are implemented as function calls in R, e.g., the functionality of `return` and `switch` is provided by function calls,
- there is a special operator, `<<-` to assign to a global variable from within a function,
- vectors/arrays/data.frames can be sliced to return a subset of the original,
- explicit support for `NA` (Not Available). This value denotes a number that exists, but whose value is unknown. Operations involving `NA` return `NA` when the result value is not known because the value of `NA` is unknown, but will return a value when the result is independent of the value of `NA`, e.g., `NA || TRUE`,
- type conversion behavior may be driven by semantics rather than the underlying representation, e.g., `as.numeric("1") ==1` and `as.numeric("a")` returns `NA`.

The following R language features are found in commonly used languages:

- objects and functions come into existence during program execution when they appear on the left-hand-side of an assignment, function parameter, or in more obscure ways, and a value is assigned (there is no mechanism for declaring any kind of identifier),
- the type of an object is the type of the value last assigned to it,
- decimal and hexadecimal literals have type numeric (literals starting with zero are not treated as octal literals; the zero is ignored) even if they look like integers, because they do not contain a decimal point). Some input functions, e.g., `read.csv`, will consider a column to have integer type if all its values can be represented as an integer,
- what most other languages considered to be a statement (i.e., something that does not return a value) R treats as an expression (e.g., if/for statements return a value).

9.2.2 Objects

Operations in R operate on objects, sometimes known as variables. Objects are characterized by their names and their contents; with the contents in turn being characterized by attributes specifying the kind of data contained in the object.

The R type system has evolved over time and includes the terms *mode* (a higher level view of the value representation, at least sometimes, than *typeof*, e.g., integer and double have mode numeric), *storage.mode* (a concept going back to the S language) and *typeof* (the underlying representation used by the C implementation of the language).

The `mode`, `storage.mode` and `typeof` functions return a string containing the respective information, e.g., `numeric`, `integer` or `function`).

The length of an object is the number of elements it contains. The `length` function returns the number of elements contained in its argument.

The assignment operator creates an object, with the object name being the left operand and its value and type being that of the right operand.

9.3 Operations on vectors

9.3.1 Creating a vector/array/matrix

An R vector can be thought of as a one-dimensional array. Vectors are indexed starting at 1 (not zero) and it is possible to added additional elements to a vector, but not remove an existing element.

```
x = 2                   # new vector containing one value
x = c(2, 4, 6, 8, 10)   # new vector containing five values
# new vector containing the contents of x and two values
x = c(x, 12, 14)
y = vector(length=5)    # new vector created by function call
y = 3:8                 # same as c(3, 4, 5, 6, 7, 8)
z = seq(from=3, to=13, by=3)      # create a sequence of values
# All elements converted to a common type
z = c(1, 2, "3")        # String has the greater conversion precedence
```

Multidimensional arrays are created using the `array` function, with the common case of 2-dimensional arrays supported by a specific function, i.e., the `matrix` function.

```
> # create 3-dimensional array of 2 by 4 by 6, initialized to 0
> a3=array(0, c(2, 4, 6))
> matrix(c(1, 2, 3, 4, 5, 6), ncol=2) # default, populate in column order
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> # specify the number of rows and populate by row order
> matrix(c(1, 2, 3, 4, 5, 6), nrow=2, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> x = matrix(nrow=2, ncol=4) # create a new matrix
> y = c(1, 2, 3)
> z = as.matrix(y)  # convert a vector to a matrix
> str(y)
 num [1:3] 1 2 3
> str(z)
 num [1:3, 1] 1 2 3
```

9.3.2 Indexing

One or more elements or a vector/array/matrix can be accessed using indexing. Accesses to elements that do not exist return `NA`. Negative index values specify elements that should not be included in the returned value.

The zeroth element returns an empty vector.

```
> x = 10:19
> x[2]
[1] 11
> x[-1]                         # exclude element 1
[1] 11 12 13 14 15 16 17 18 19
> x[12]                         # there is no 12'th element
[1] NA
> x[12]=100                     # there is now
> x
 [1]  10  11  12  13  14  15  16  17  18  19  NA 100
```

Multiple elements can be returned by an indexing operation.

```
> x = 20:29
> x[c(2,5)]                     # elements 2 and 5
[1] 21 24
> y = x[x > 25]                 # all elements greater than 25
> y
[1] 26 27 28 29
> # The expression x > 25 returns a vector of boolean values
> i = x > 25
> i
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
> # an element of x is returned if the corresponding index is TRUE
> x[i]
[1] 26 27 28 29
```

Matrix indexing differs from vector indexing in that an out-of-bounds access generates an error.

```
> x = matrix(c(1, 2, 3, 4, 5, 6), ncol=2)
> x[2, 1]
[1] 2
> # x[2, 3] need to be able to handle out-of-bounds subscripts in Sweave...
> x[, 1]
[1] 1 2 3
> x[1, ]
[1] 1 4
> x[3, ]=c(0, 9)
> x
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    0    9
> x=cbind(x, c(10, 11, 12))  # add a new column
> x
     [,1] [,2] [,3]
[1,]    1    4   10
[2,]    2    5   11
[3,]    0    9   12
> x=rbind(x, c(5, 10, 20))   # add a new row
> x
     [,1] [,2] [,3]
[1,]    1    4   10
[2,]    2    5   11
[3,]    0    9   12
[4,]    5   10   20
```

9.3.3 Lists

The difference between a list and a vector is that different elements in a list may have different modes (types) and existing elements can be removed.

```
> x = list(name="Bill", age=25, developer=TRUE)
> x
$name
[1] "Bill"

$age
[1] 25

$developer
[1] TRUE
> x$name
[1] "Bill"
> x[[2]]
[1] 25
> x = list("Bill", 25, TRUE)
> x
[[1]]
[1] "Bill"

[[2]]
[1] 25

[[3]]
[1] TRUE
> y = unlist(x) # convert x to a vector, all elements are converted to strings
> y
[1] "Bill" "25"   "TRUE"
> x = list(name="Bill", age=25, developer=TRUE)
> x$sex="M"  # add a new element
> x
$name
```

```
[1] "Bill"

$age
[1] 25

$developer
[1] TRUE

$sex
[1] "M"
> x$age = NULL # remove an existing element
> x
$name
[1] "Bill"

$developer
[1] TRUE

$sex
[1] "M"
```

The **[[ ]]** operator returns a value while **[ ]** returns a sublist (which has mode list).

9.3.4 Data frames

From the perspective of a programmer coming from another language, it seems obvious to think of a data frame as behaving like a matrix, and in some cases it can be treated in this way (e.g., when all columns have the same appropriate type functions expecting a matrix argument may work). However, a better analogy is to treat it like an indexable structure type (where different members can have different types).

The `read.csv` functions reads a file containing columns of different values and returns a data frame.

When indexing a data frame like a matrix, elements are accessed in row-column order (not the column-row order found in C-like languages). The following code selects all rows for which the `num` column is greater than 2.

```
> x = data.frame(num=c(1, 2, 3, 4), name=c("a", "b", "c", "d"))
> x
  num name
1   1    a
2   2    b
3   3    c
4   4    d
> # Have to remember that rows are indexed first and also specify x twice
> x[x$num > 2, ]
  num name
3   3    c
4   4    d
> # No need to remember row/column order and only specify x once
> subset(x, num > 2)
  num name
3   3    c
4   4    d
```

If one or more columns contain character mode values, `read.csv` will create a factor rather than a vector.

9.3.5 Symbolic forms

```
exp = expression(x/(y+z))
eval(expr) # evaluate expression using the current values of x, y and z
```

Uses of expression objects include specifying which vectors in a table to plot in a graph and including equations in graphs, for instance:

```
text(x, y, expression(p == over(1, 1+e^(alpha*x+beta))))
```

will cause the following equation to be displayed at the point (x, y).

$$p = \frac{1}{1 + e^{\alpha x + \beta}}$$

The D function takes an expression as its first argument and based on the second argument returns its derivative:

```
> D(expression(x/(y + z)^2), "z")
-(x * (2 * (y + z))/((y + z)^2)^2)
```

9.3.6 Factors and levels

Statisticians found that when manipulating non-numeric values (e.g., names) it can be convenient to map them to integer values and manipulate these integers. In programming terminology a variable used to represent one or more of these integer values could be said to have a *factor* type, with the actual numeric values known as *levels* (a parallel can be drawn with the enumeration types found in C++ and C, except these assign names to integer values).

```
> factor(c("win", "win", "lose", "win", "lose", "lose"))
[1] win  win  lose win  lose lose
Levels: lose win
```

Some operations implicitly convert a sequence of values to a factor. For instance, `read.csv` will, by default, convert any column of string values to a factor; this conversion is a simplistic form of hashing and (when a megabyte was considered a lot of memory) was once driven by the rationale to save storage space. These days the R implementation uses more sophisticated hashing and we have to live with the consequences of historical baggage.

Operations of objects holding values represented as factors sometimes has surprising effects for those unaware of how things used to be.

9.4 Operators

Operators in R follow the same precedence rules as Fortran, which in some cases differ from the C precedence rules (which most commonly languages now use). An example of this difference is: `!x ==y` which is equivalent to `!(x ==y)` in R, but equivalent to `(!x) ==y` in C-like languages (if x and y have type boolean, there is no effective difference, but expressions such as: `!1 ==2` produce a different result).

A list of operators and their precedence can be obtained by typing `?Syntax` at the R command line.

Within an expression operand evaluation is left to right, except assignment which evaluates the right operand and then the left.

In most cases, all elements of a vector are operated on by operators:

```
> c(5, 6) + 1
[1] 6 7
> c(1, 2) + c(3, 4)
[1] 4 6
> c(7, 8, 9, 10) + c(11, 12)
[1] 18 20 20 22
> c(0, 1) < c(1, 0)
[1]  TRUE FALSE
```

in the last two examples *recycling* occurs, that is the elements of the shorter vector are reused until all the elements of the longer vector have been operated on.

The operators **&&** and || differ from **&** and | in that they operate on just the first element of their operands and return a vector containing one element, e.g., `c(0,1) && c(1,1)` returns the vector `FALSE`.

Operator	Description
:: :::	access variables in a name space
$ @	component / slot extraction (member selection has lower precedence than subscripting in C-influenced languages)
[[[	indexing
^	exponentiation (associates right to left)
- +	unary minus and plus
:	sequence operator
%any%	special operators (%% and %/% has the same precedence as * and / in C-influenced languages)
* /	multiply and divide
+ -	(binary) add subtract
< > <= >= == !=	relational and equality (non-associative; equality has lower precedence in C-influenced languages)
!	negation (greater precedence than any binary operator in C-influenced languages)
& &&	and
| ||	or
~	as in formulae
-> ->>	rightwards assignment
=	assignment (associates right to left)
<- <<-	assignment (associates right to left)
?	help (unary and binary)

Table 9.1: Operators listed in precedence order.

The base system includes a set of rfunc[bitw???] functions that perform bitwise operations on their integer arguments and there is the `bitops` package.

The character used for exclusive-or in C-influenced languages, ^, is used for exponentiation in R; the `xor` function performs an exclusive-or of its operands.

The `[` and `[[` operators differ by more than being array and list indexing. The result of the index `x[1]` has the same type as `x` (i.e., the operation preserves the type), while the result of `x[[1]]` is a simplified version of the type of `x` (if simplification is possible).[i]

```
> x = c(a = 1, b = 2)
> x[1]
a
1
> x[[1]]
[1] 1
> x = list(a = 1, b = 2)
> str(x[1])
List of 1
 $ a: num 1
> str(x[[1]])
 num 1
> x = matrix(1:4, nrow = 2)
> x[1, ]
[1] 1 3
> x[1, , drop = FALSE]
     [,1] [,2]
[1,]    1    3
> # x[[1, ]] is not allowed
>
> df = data.frame(a = 1:2, b = 1:2)
> str(df[1])
'data.frame':   2 obs. of  1 variable:
 $ a: int  1 2
> str(df[[1]])
 int [1:2] 1 2
> str(df[, "a", drop = FALSE])
'data.frame':   2 obs. of  1 variable:
 $ a: int  1 2
```

[i] Out of bounds handling is also different, but I'm sure readers' don't do that sort of thing.

```
> str(df[, "a"])
 int [1:2] 1 2
```

9.4.1 Testing for equality

In addition to the equality operators the base system includes two equality related functions, `identical` and `all.equal`.

```
> x = 1:5  ;  y = 1:5
> x == y  # Return the result of equality test for each element
[1] TRUE TRUE TRUE TRUE TRUE
> identical(x, y) # Return a single value denoting exact equality
[1] TRUE
> 1L == 1   # 1L is stored internally as an integer, 1 is stored as a double
[1] TRUE
> identical(1L, 1)  # identical requires the stored type be the same
[1] FALSE
> 0.9 == (1.1 - 0.2)  # could be affected by lack of precision
[1] FALSE
> all.equal(0.9, 1.1 - 0.2)  # do a fuzzy compare
[1] TRUE
> all.equal(0.9, 1.1 - 0.2, tolerance=0)  # find out much fuzz there is
[1] "Mean relative difference: 1.233581e-16"
```

The default tolerance used by the `all.equal` function is `.Machine$double.eps ^ 0.5`.

Comparisons against `NA` always returns `NA` and the `is.na` function has to be used to check for this quantity; the `anyNA` function returns `TRUE`/`FALSE` if its argument contains an `NA`.

9.4.2 Assignment

The following are four ways of the ways of assigning a value to a variable in R:

```
x <- 3  # Operator used by people who follow the herd
x <<- 3 # Assigns to the x at global scope

3 -> x  # Not often encountered outside descriptions of the language

x = 3   # Supported since R version 1.4
```

Many R books and articles use the token **<-**. Developers are used to seeing the **=** token and with nothing other than conformity to existing R usage to recommend the alternative, the token that developers are already very familiar with is used in this book.

There is one context where **=** does not behave like normal assignment. R supports the use of parameter names in arguments to function calls to explicitly specify that a named parameter is to be assigned a given value. In the context of a function argument list the left operand of **=** is treated as the name of a parameter and the right operand as the value to be assigned. An error is flagged if the function definition does not have a parameter having the specified name.

```
func = function (a, b, c) a + b * c

func(2, 3, 9)
func(c=9, b=3, a=2)

func(d=3, 4, 5)  # no parameter named d, an error is raised

# use <- if the intent is to assign to d and pass this value as an argument
func(d<-3, 4, 5)
```

9.5 The R type (mode) system

R supports values having the following basic types (R also has the concept of *mode*, which is based on semantics rather than underlying representation, e.g., the `mode` function returns `numeric` where `typeof` returns either `integer` or `double`):

- `NULL`:
- raw: essentially uninterpreted byte values,
- logical: holds one of the values: `TRUE`, `FALSE`, `T` or `F`. The conversion `as.logical(any_non_zero_value)` returns `TRUE`,
- character: what many other languages call a string type,
- integer: the only integer type uses 32 bits (`NA` is represented using the most negative value, so this value is not available as an integer; trying to generate this or any other value outside of the representable value of a 32-bit integer will result in a value having a real type),
- double: the only floating-point type contains 64 bits,
- complex: contains a real and imaginary double type,

An object may be reported to have one of these basic types, but it may actually be a vector or array of this type.

More complicated types may be created, such as lists, data frames, etc.

9.5.1 Converting the type (mode) of a value

It is often possible to convert the mode (type) of a value by calling the `as.some_mode` function, where `some_more` is the name of a mode, e.g., `integer`. If a conversion fails, `NA` is returned.

Conversion precedence

`NULL < raw < logical < integer < real < complex < character < list < expression`

9.6 Statements

R contains the usual language constructs that look like statements, but can behave like expressions:

- **function**: defines a function, whose value has to be assigned to an object:

  ```
  f=function(p1, p2) {return(p1+p2)}
  ```

- blocks of code are bracketed using the punctuation pair: **{** and **}**,
- **;** (semicolon) is required to delimit multiple expressions on the same line, but is otherwise optional,
- **if**: which takes an optional **else** arm (there no **then** keyword, but there is an `ifthenelse` function),
- **for**: which has the form `for (i in x)`, where `x` is a vector (such as `1:10`),
- **while** and **repeat** loops are available,
- loops may be terminated using **break** keyword or the `break` function, and may be continued at the next iteration using the **next** keyword or `next` function,
- `return` is a function: `return(1+return(1))` returns the value `1`,
- `switch` is a function.

9.7 Defining a function

```
> g=1  # a global variable
> f = function(p1, p2) # define a function and assign it to f
+ {
+ l=g  # Value access, check lexical and dynamic scope for g
+ g=2  # Assignment: only check local scope, if no variable exists, create one
+
+ m=h  # h is dynamically in scope
+
+ return(return(1)+1) # return is a function call
+ }
> h=2  # another global variable
> f(1, 2)
[1] 1
> g
[1] 1
> h=3 # At global scope, so must be global variable
```

Argument evaluation is lazy, that is they are evaluated the first time their value is required.

The `...` token specifies that a variable number of unknown arguments may be passed.

```
unk_args=function(...)
{
a=list(...) # Convert any arguments passed to a list of values
# Access the list  of values in a
}
```

9.8 Commonly used functions

Technically every operation is a function call (so `'+'(1, 2)` and `1+2` are equivalent), but not all function calls have equivalent operator tokens.

```
> x = 1:10
> if (any(x > 7)) print("At least one value greater than 7")
[1] "At least one value greater than 7"
> if (all(x > 0)) print("All values greater than zero")
[1] "All values greater than zero"
> rep(1:2, 3)
[1] 1 2 1 2 1 2
```

`head`/`tail` mimics the behavior of the unix `head`/`tail` program,

`length` returns the number of elements in its vector argument,

`nrow`/`ncol` return the number of rows/columns in the data frame argument (`NROW`/`NCOL` gracefully handle vector arguments),

`order` returns a vector containing an index in to the argument in the order needed to sort the argument values,

`str` lists the columns in a variable, along with their type and the first few values in each row; it provides a quick way of verifying that columns have the expected type.

`which` returns a vector of values containing the index of the argument values that are true,

`methods`: list functions overloaded on the argument name

`installed.packages`: list all installed packages

`getwd`, `setwd`:

`list.files`, `list.dirs`:

`ls` lists variables that exist in the current environment,

`system.time`, `proc.time`:

9.9 Input/Output

Functions are available for reading data having a variety of formats (e.g., comma separated values) from all the common data sources (e.g., files, databases, web pages). In some cases the contents of a compressed file will be automatically uncompressed before reading. The data read is often returned as a single object.

Many functions try to automatically deduce the datatype of the data read, e.g., whether it is integer, real, sequence of characters, etc. Sometimes the datatype selected is not correct and work has to be done to ensure the data is treated as having the desired type; `read.csv` bases its decision on the type of the columns by analysing the first 6 or so lines of the file.

Some functions in the base system, e.g., `read.csv`, convert columns containing string values to factors; the original intent was presumably to reduce the storage needed to hold the data. A column of factors does not always behave the same as a column of strings and this default conversion behavior is now a liability. Passing the argument `as.is=TRUE` stops values being converted to factors (it is used in all the example code).

```
data=read.csv("measurements.csv.gz", as.is=TRUE) # file will be uncompressed

data=read.csv("measurements.csv", sep="|", as.is=TRUE) # change separator

data=read.csv("https://github.com/Derek-Jones/ESEUR-code-data/blob/master/benchmark/MST
```

The first line of the input file is assumed to denote the column names, and `header=FALSE` has to be specified to switch off this behavior.

All characters on an input line after, and including, the comment character, #, are ignored (various options interact with this behavior, including the `comment.char` option which can be used to change the character used).

If data is not already in a form that can be easily processed by R, it may be simpler to convert it by using a language or tool that you are already familiar with, rather than using R.

Output is supported by a wide range of functions: `print` performs relatively simple formatted output (the `format` function can be used to create more sophisticated formatting that can then be output), the `cat` function performs relatively little formatting but is more flexible and in particular does not terminate its output with a newline, the `sink` function can be used to specify an alternative location to write console output, and there are often `write` equivalents of the `read` functions such as `write.csv`.

The R environment includes a simple spreadsheet like editor for manual data entry and patching existing data.

```
scores = edit(scores) # use built in spread-sheet like editor
```

9.9.1 Graphical output

There may be more functions supporting graphical output in R than textual output. Perhaps the most commonly used graphical output function is `plot`. This function often does a surprisingly good job of producing a reasonable graphical representation of the data. Overloaded versions of this function are sometimes provided by packages, to plot data having a particular class created by the package.

By default, graphical output is sent to the console device; this behavior can be overridden to produce a file having a particular format, e.g., `pdf`, `jpeg`, `png` and `pictex`. The list of supported output devices varies across the operating systems on which R runs.

The behavior of the `plot` function can be influenced by previous calls to the `par` function, which set configurable options.

Various packages proving graphical output are available, with the `ggplot` package probably being the most commonly used by frequent R users.

9.10 Other uses for R

While the target of R's domain specialised functionality is statistical data analysis, there are other application domains where this functionality could be useful (but do not warrant effort needed to learn R).

A variety of functions designed for manipulating the rows and columns of delimited data files are available; see rexample[Top500.R].

A useful technique for spotting whether a file contains compressed data, e.g., a virus hidden in a script by compressing it to look like a jumble of numbers. Compressed data contains a uniform distribution of byte values (after all, compression is achieved by reducing apparent information content), your mileage may vary between compression techniques.

The following code reads a complete file, applies a sliding window to the data and then plots it.

```
window_width=256 # if less than 256, divisor has to change in plot call

plot_unique=function(filename)
{
t=readBin(filename, what="raw", n=1e7)

# Sliding the window over every point is too much overhead
cnt_points=seq(1, length(t)-window_width, 5)

u=sapply(cnt_points, function(X) length(unique(t[X:(X+window_width)])))
plot(u/256, type="l", xlab="Offset", ylab="Fraction Unique", las=1)

return(u)
}

dummy=plot_unique("http://www.coding-guidelines.com/R_code/requirements.tgz")
```

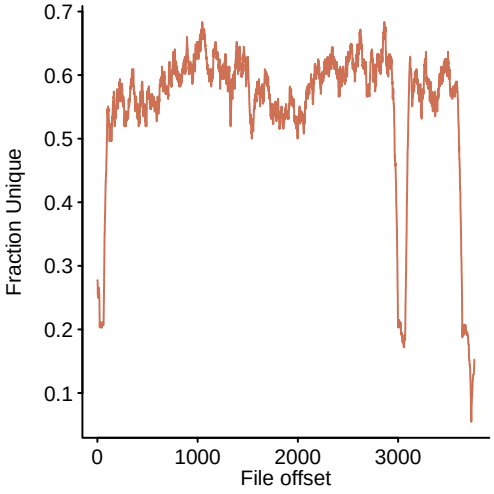


Figure 9.2: The unique bytes per window (256 bytes wide) of a pdf file. code

9.11 Very large datasets

While most existing software engineering datasets tend to be relatively small, exceptions may occur from time to time. A variety of techniques are available for handling large datasets, including the following:

- the `bigmemory` package provides software defined memory management (i.e., swapping data between memory and main storage). The `bigtabulate` package and other `big` packages contains functions that perform commonly used operate on this data.
- the `data.table` package extends data.frames to support up to 100G of storage,

9.12 Debugging R code

The `traceback` function...

Packages to help debug R programs include: `RUnit` for unit testing, `covr` for code coverage,...

Chapter 10

Data preparation

10.1 Introduction

The most important question to keep asking yourself while examining, preparing and analyzing any data is: Do I believe this data?

Books and presentations on data analysis rarely mention that a significant amount of effort often has to be invested in preparing the data (perhaps 80% or more), getting it ready for analysis.[i]

Perhaps the largest task within data preparation is data cleaning, an often overlooked[358] aspect of data analysis that is an essential part of the workflow needed to avoid falling foul of the adage: Garbage in garbage out.

Domain knowledge is essential for data cleaning; values have to be understood in the context in which they occur. The fact that many data cleaning activities are generic does not detract from the importance of domain knowledge. For instance, software knowledge tells us that 1.1 is not a sensible measurement for lines of code (in the NASA MDP dataset), talking to developers at a company that they don't work at weekends (e.g., the dates in the 7digital data) and knowing that system support staff used the Unix `pwd` command to check that the system was operational (an analysis of job characteristics for a NASA supercomputer[175] has to first remove 56.8% of all logged jobs).

Data cleaning is often talked about as-if it is something that happens before data analysis, in practice, the two activities intermingle; the time spent checking and cleaning the data provides insights that lead to a better understanding of the kinds of analysis that might be applicable, also, results from a preliminary analysis can highlight data needing to be cleaned in some way. Data preparation is discussed here in its own chapter, as-if it was performed as a stand-alone activity, in order to simplify the discussion of the material in other chapters.

Once cleaned the data may need to be restructured, e.g., rows/columns contained in different files merged into a single data table, or the row/columns in an existing table reorganised in some way. The required structure of the data is driven by the operations that will be performed on it (e.g., finding the median value of some attribute) or the requirements of the library function used to perform the analysis.

It may be necessary to remove confidential information from the data or to remove information that might be used to identify individuals. Datasets do not exist in isolation and it may be possible to combine apparently anonymous datasets to reveal information;[ii] k-anonymity and l-diversity are popular techniques for handling this situation (in a k-anonymized dataset each record is indistinguishable from at least $k-1$ other records and l-diversity requires at least l distinct values for each sensitive attribute). Techniques for anonymizing data are not covered here; Fung et al[194] survey techniques for privacy-preserving data publishing, Templ et al[563] provide an introduction to statistical disclosure control and the `sdcMicro` package.

[i] This chapter appeared immediately after the Introduction in early versions of this book, but in response to customer demand was moved here (the last chapter); people want to read about the glamorous stuff, data analysis, not the grunt work, data cleaning.

[ii] 63% of the US population can be uniquely identified using only gender, ZIP code and full date of birth.[211]

While a lot of software engineering data comes from measurements made by programs, some is still obtained from human written records (which can contain a substantial number of small mistakes[297]).

Data cleaning involves a lot of grunt work that often requires making messy trade-offs and having to make do. Tools are available to reduce the amount of manual work involved, but these may require placing some trust in the tool doing the right thing. The following are a few of the available tools:

- OpenRefine[426] (was Google Refine) reads data into a spreadsheet like form and supports sophisticated search/replace and data transformations editing,
- the `editrules` package checks that data is consistent with user specified consistency rules about the column values (e.g., `total_fruit ==total_apples+total_oranges`),
- the `deducorrect` package performs automatic value transformation based on user specified consistency rules about the column values that must be met (e.g., failure to meet the condition `total_x > 0` will result in any value in that column having a negative sign removed); this package can also impute missing values,
- there are a variety of special purpose packages that handle domain specific data, e.g., the `CopyDetect` package detects copying of exam answers in multi-choice questions.

?

10.1.1 Data cleaning must be documented

The changes made to the original data during the cleaning process need to be documented; this change log serves a variety of purposes, including:

- enabling third-parties to check that the changes are reasonable and don't substantially alter the results of the analysis,
- enabling a potential source of uncertainty to be checked when multiple outlets publish results based on the same dataset, i.e., if there are differences in the results it is possible to check that these differences are not primarily the result of differences in cleaning operations,
- providing confidence to users of the final results of the analysis that the researcher doing the work is competent, i.e., that cleaning was performed.

Ideally the operations performed on the original data to transform it into what is considered a clean state are collected together as a script for ease of replication.

Some cleaning activities are trivial and yet need to be performed to prevent the analysis being overwhelmed by what appears to be lots of special cases. For instance, an analysis[27] of different company's response to vulnerabilities reported in their projects, started with data that sometimes used slightly different ways of naming the same company. The company names data had to be cleaned to ensure that one name was consistently used to denote each organization (see rexample[patch-behav.R]).

The publicly available NASA Metrics Data Program (MDP) dataset contains fault data on 13 projects and has been widely used in academic research (a literature survey for the period 2000 to 2010[244] found that 58 out of 208 papers used it). This dataset contains many problems[226] that need to sorted out, e.g., columns with all entries having the same value (suggesting a measurement or conversion error has occurred), duplicate rows, missing values (many occurred in rows calculated from other rows and involved a divide by zero), inconsistent values (e.g., number of function calls being greater than the number of operators) and nonsensical values (e.g., lines of code having fractional values).

Despite the extensive work needed to clean the MDP dataset to get it into a reliable state, the authors of many of the published papers using this dataset have either not cleaned it or have only given a cursory summary of their cleaning activities[68] ("... removes duplicate tuples ... along with tuples that have questionable values...", does not specify what values were questionable). Consequently, even although the original dataset is publicly available

it is difficult to compare the results published in different papers because no information is available on what, if any, data cleaning operations were performed; so much of the data is in need of cleaning that any results based on an uncleaned version of this dataset must be treated with suspicion.

See rexample[NASA_MDP-data_check.R] for examples of various integrity checks performed on the MDP dataset.

A survey of 682,000 unique Android devices in use during 2015, by OpenSignal,[427] included the screen height and width reported by the device (see left plot in Figure 10.1). Many devices appear to have greater width than height, particularly those with smaller screens. Perhaps the device owners are viewing the OpenSignal website with their phones in landscape mode (the right plot in Figure 10.1 switches the dimensions so that height has the larger value; switched values in red).

Like their source code, the fault repositories of open source projects are publicly available and the repositories of larger projects are a frequent source of data for fault analysis/prediction researchers.

A study by Herzig, Just and Zeller[261] manually classified over 7,000 issue reports extracted from the fault repositories of seven large Java projects. They found that on average 42.6% of reports had been misclassified, with 39% of files marked as defective not actually containing any reported fault (the implication being that any fault prediction models built using this uncleaned data are likely to be misleading at best and possibly very wrong). An earlier analysis[262] had found that between 6 and 15% of bug fixing changes addressed more than one issue.

Possible reasons for the misclassification include: the status of an issue not being specified when the initial report is filed, resulting in the default setting of *Bug* being used; issue submitters having the opinion that a missing feature is a bug (request for enhancement was the most commonly reclassified status); and bug reporting systems only supporting a limited number of different issue statuses (forcing the submitter to use an inappropriate status).

This study shows, at least for the fault repositories of seven large Java projects, that data cleaning is essential for any analysis based on this data to be reliable. The study also highlighted how much effort data cleaning consumes; the work was performed independently by two people and took a total of 725 hours (90 working days).

Sometimes the measured values from one or more subjects (e.g., people or programs) are remarkably different from the measured values of other subjects. It can be very tempting to clean the data by removing the measured value for these subjects from the dataset. A study by Müller and Höfer[402] removed data on seven out of 18 subjects because they considered the performance of these subjects was so poor that they constituted a threat to the validity of the experiment (whose purpose was to compare the performance of students and professional developers). This kind of activity might be classified as outlier removal or manipulating data to obtain a desired result, either way, documenting the cleaning activity allows the audience to decide.

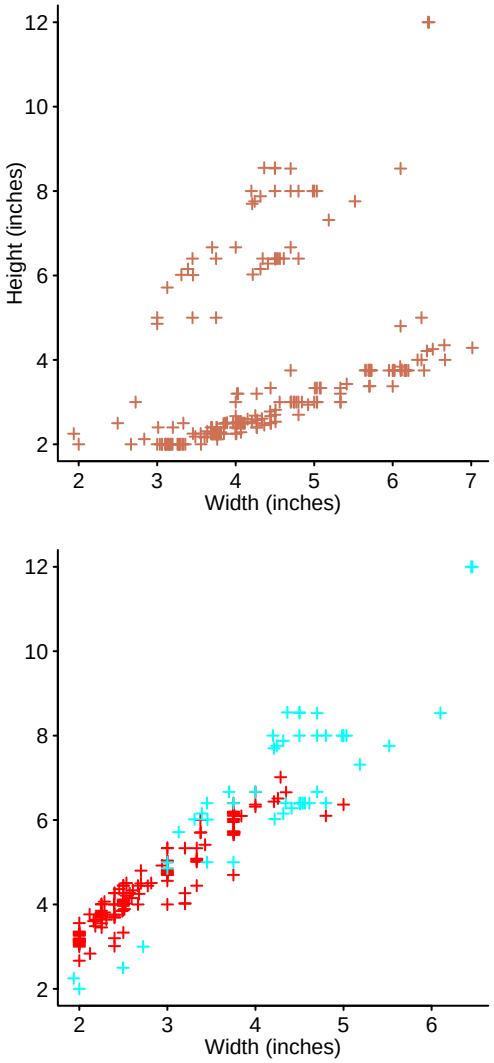


Figure 10.1: Screen height and width reported by 682,000 unique devices that downloaded an App from OpenSignal in 2015 (upper), values switched to ensure height is always the largest value (lower). Data from OpenSignal.[427] code

10.1.2 Outliers

"An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism".[250]

Without some knowledge of the mechanism that produced a given sample it is not possible to label any observation as suspicious, although it might be possible to claim that the observations were generated by more than one mechanism. If the percentage of deviate observations is small, the mechanism that produced them might be labelled as unimportant and those observations excluded from the subsequent analysis; either way, document the situation and let your audience decide.

In some applications the deviate observations are the ones of interest,[98] e.g., intrusion detection and credit card fraud. This subsection covers the case where deviate observations are unwanted; some later subsections cover software engineering situations where deviate observations are themselves the subject of study.

Two commonly used ways of handling outliers are:

- using a statistical technique that does not assign too much weight to observations that deviate from the common pattern followed by other observations. Techniques capable of performing the desired statistical analysis are not always available, but when R functions implementing them are available they are discussed in the appropriate section,
- detecting and excluding outliers from the subsequent analysis. Traditionally outliers have been manually selected and excluded from subsequent analysis. This approach can work well when the sample contains a small amount of data and the person doing the detection has sufficient domain knowledge. There are a variety of functions that automate the process of outlier selection and handling, some of these are discussed below.

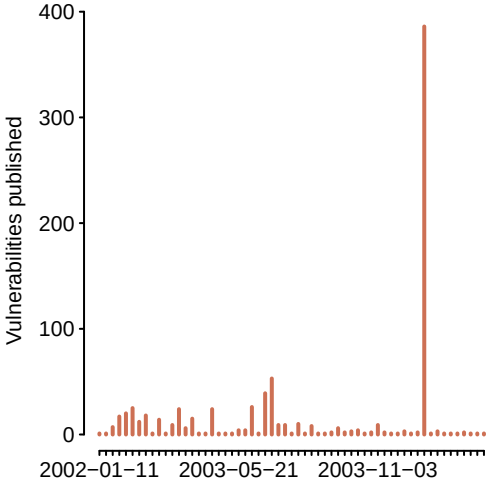


Figure 10.2: Number of reported vulnerabilities, per day, in the US National Vulnerability Database for 2003. Data from the National Vulnerability Database.[415] code

Another definition of outlier detection[98] is "... the problem of finding patterns in data that do not conform to the expected normal behavior." This definition requires that an expected normal behavior be known, along with a method of comparing values for *outlyingness*.

Figure 10.2 shows a very suspicious spike in the number of daily reported vulnerabilities recorded in the US National Vulnerability Database for 2003. Perhaps all vulnerabilities that had been reported, but not yet fully processed, were simply published for the public to see at the end of the year? In this sample the analysis is highlighting a subset of the data reported on one day as a suspected outlier.

Should outliers be removed from a sample?

While removing outliers may improve the quality of the fit to an equation, does it improve the quality of the fit of the model to reality?

Without understanding the process that generated the data there is no justification for removing any value.

The real issue with outliers is the impact they have on the final result. In a large sample a few unusual values are unlikely to have any real impact data.

However, outliers are handled, the decision needs to be documented, giving either the reason for excluding observations from the analysis or the reason for keeping them and their impact on the final result.

10.2 Malformed file contents

The first sign that data or its organization in a file is malformed in some way, usually occurs when the variable into which the file contents have been read does not have the expected contents (e.g., incorrect number of columns or the type of data in one or more columns is not as expected, such as strings where a number should have occurred).

The `str` function provides a quick way of verifying that columns in a data.frame have the expected type. For instance, unexpected characters in the input data can result in some column values being treated as strings when a numeric type is expected.

File contents formatting issues to watch out for include:

- functions for reading data in R (e.g., `read.csv` and `read.table`) often use the first few lines of file being read as the format to when reading the rest of the file, i.e., the number of columns contained in each row and the datatype of the values in each column. If there are one or more rows that do not follow the format selected at the start of the file (e.g., different number of column delimiters, perhaps the result of non-delimited strings such as a missing pair of quote characters) then subsequent values may not appear in the correct column or be converted to a different type,
- termination delimiter missing from a string value, which can result in the following row being treated as part of the current row (because the newline is treated as part of the string),
- cut-and-pasting of data introducing conversion errors, e.g., the digit zero treated as the letter D or G.

A variety of ad-hoc techniques are available for locating the cause of problems. For instance, the following code will convert all values that do not have the format of a number to `NA`, which are easily located using built-in support for processing `NA`s and their row index listed using `which`.

```
which(is.na(as.number(as.character(data_frame$column_name))))
```

The `complete.cases` function returns a vector specifying which of the rows in its data.frame argument are complete, i.e., do not contain any `NA` values; the `na.omit` function returns a copy of its argument with any rows containing `NA` omitted.

10.3 Missing data

Missing data (for instance, a survey where the entry for a person's age is empty) is often the rule rather than the exception and whole books have been written on the subject. Missing data may be disguised in that it appears as a reasonable looking value,[443] e.g., zero when the range of possible legitimate values includes zero.

The starting point for handling missing data is to normalise how it is denoted to the representation used by R, i.e., `NA` (Not Available). Normalisation ensures that all missing values are treated consistently; special case handling of `NA` is built into R and many functions include options for handling `NA`.

A wide variety of different representations for missingness are likely to be encountered (e.g., special values that cannot occur as legitimate data values such as 9999, "#N/A", "missing" or no value appearing between two commas in a comma separated list), and it is not uncommon for different columns within a dataset to use different representations (because they originate from different sources of measurement).

Missing values are sometimes *disguised* in that they look reasonable, e.g., a value of zero (where domain knowledge does not rule out zero as a realistic measurement value) or when data input only provides a two item choice, such as male/female, with one item being the default and thus appearing as the missing value when no explicit choice is made. Disguised missing values can have a significant impact on the results.[443]

The following shows one way of changing a known representation of missing value to `NA` (the second form would be necessary if 9999 could appear as a legitimate value in a column other than `size`):

```
data[ data == 9999 ] = NA  # set all elements having value 999 to NA
# set all elements of column size having value 999 to NA
data$size[ data$size == 9999 ] = NA
```

Once missing values have been explicitly identified it is possible to move on to deciding whether to ignore these cases or to replace `NA` with some numeric value. Some algorithms can handle missing values while others cannot; R functions vary in their ability to handle missing values. A few techniques for selecting the replacement value are discussed below.

The R base I/O functions, such as `read.csv`, have conventions for handling the case of zero characters appearing between the delimiters on each line of a file. The behavior depends on what type the values in a particular column are assumed to have. For columns considered to have a numeric type the zero character case is treated as if `NA` occurred between the delimiters, while for columns considered to have a string type the zero character case is treated as the empty string rather than `NA` (i.e., treated the same as the string ""). These functions support a variety of options for changing the default handling of leading/trailing white-space between delimiters and the handling of zero characters.

For instance, reading a file containing the columns appearing below left has the same effect as reading a file containing the columns appearing below right:

```
X,Y_str,Z          X,Y_str,Z
1,"abc",2.2        1,"abc",2.2
2,,3.1             2,"",3.1
,NA,2              NA,NA,2
```

The `table` function is commonly used to count occurrences of values, but by default does not include `NA` in its count; the `useNA` needs to be used to explicitly specify that they be included:

```
table(data$some_column, useNA="ifany") # limit the count to one column
```

This one column use can be expanded to cover every column in `data`. If the output is too voluminous, the number of columns processed will need to be reduced or the call to `table` replaced by a call to `tabulate` which provides more options to control behavior:

```
sapply(colnames(data), function(x) table(data[ , x], useNA="ifany"))
```

While there may be documentation specifying how missing values are represented, such details are often not written down. An analysis of a dataset using the above code may show a suspiciously large number of values such as 9999 or -1 (for an attribute that can never be negative), something that warrants further investigation.

10.3.1 Handling missing values

When deciding what to do about missing values it is important to try to understand why the values are missing. The following categories are commonly encountered in the analysis of missing data:

- Missing completely at random (MCAR): As the name suggests the selection of missing values occurred completely at random. Statistically this is the most desirable kind of missingness because it means there is no bias in the missing values.
- Missing at random (MAR): This sounds exactly like MCAR, it is not completely random in the sense that the choice of which values are missing is influenced by other values in the sample. For instance, the level of seniority may correlate with the likelihood that survey questions about salary are answered,
- Missing not at random (MNAR): This could be just as random as MAR, with the only difference being that the choice of missing values is influenced by values not in the sample. For instance, the name of the developer who originally wrote the code referenced in a fault report may be missing if that developer is a friend of the person reporting the fault, with friendship not being a recorded in the sample.

The following code can be used to get a rough estimate of the correlation between the rows of a data.frame that contain missing values (Figure 3.7 illustrates a way of visualizing this information):

```
x=is.na(some_data_frame)
# highlight rows having some, but not all, missing values
cor(subset(x, sd(x) > 0))
```

Many analysis techniques handle missing values by ignoring the rows or columns that contain them; if the sample has many rows and a low percentage of missing values, then this behavior may not be a problem. However, if the sample contains a large percentage of missing values, any analysis will either have to make do with a smaller number of measurements, be limited to using techniques that can gracefully adapt to missing data (i.e., don't ignore rows containing one or more missing values) or be forced to used estimated values for missing data.

The ideal approach is to use an algorithm capable of handling samples that include missing data.

If missing data is to be replaced by values calculated from values that are present, other data cleaning operations should be performed first to ensure that substitute values are calculated from what are considered to be acceptable values. The issues involved in replacement values are discussed here to concentrate the discussion of this issue in one place.

A quick and dirty method that can be surprisingly effective is to replace missing values by the mean of the values in the corresponding column containing each missing value; alternatively if the data is ordered in some way, the last value appearing before the missing value might be used.

A more sophisticated approach to imputing values is to fill the missing value entries from other values in the dataset. `Amelia` and `VIM` are two packages that provide a variety of functions for visualizing datasets containing missing values and imputing values for these entries (the `VIMGUI` package provides a GUI to `VIM`).

The `aggr` and `marginplot` functions...

As Figure 6.65 shows, data that is missing over a complete range of values, rather than spread over the sample, can have a dramatic impact on fitted models.

When a single object attribute is counted, e.g., number of times the functions in a particular library are called, there is no missing data because there are no other measured attributes that can be missing. However, data may be missing because the sample may not contain any instances of particular cases (which would be seen in a larger sample). Good-Turing smoothing[196] is a technique for adding non-zero counts for unseen items and is a technique that is usually part of an approach to handling missing items, rather than something to use standalone.

10.3.2 `NA` handling by library functions

R functions vary in their ability to handle data.frames containing `NA` and exhibit the following behaviors:

- behaving in unpredictable ways when `NA` is encountered,
- behaving in predictable, but perhaps surprising to the unknowledgeable, ways, e.g., the value of `NA ==NA` is `NA` as is `NA !=NA`,

  functions that operate on complete rows or columns have a variety of behaviors when they counter one or more `NA`s, including:

  - supporting a parameter, often called `na.rm`, which can be used to specify that `NA` should be taken into account/ignored,
  - ignoring rows containing one or more `NA`, e.g., `glm` ignores these rows by default, but this behavior can be changed using the `na.action` option,
  - making use of information present in rows containing one or more `NA`, e.g., the `rpart` function,

Some regression model building functions return information associated with individual data points, such as residuals. If the function removes rows containing any `NA` before building the regression model, then the number of data rows included in the returned model may be less than originally passed in, unless rows containing `NA` is reinserted (e.g., by using the `naresid` function).

10.4 Restructuring data

When the data of interest is contained in several files it may be necessary to read two or more files and merge their contents into a single data.frame.

If two datasets contain share columns (i.e., column names, column ordering and contents are the same) the `rbind` function joins rows together, returning a single data.frame; the `cbind` function performs the same operation for columns.

When two data.frames share common columns the `merge` function can be used to join the two data.frames based on one or more shared column names; there are a variety of options for selecting how merging is performed.

10.4.1 Reorganizing rows/columns

The organization of rows and columns in a data.frame may not be appropriate for that used by the library functions used to perform the analysis.

The values in a dataset are sometimes held in a wide format (i.e., a few rows and many columns) and a long format (i.e., many rows and a few columns) is required, or vice versa.

An example of wide format data is that used in Figure 2.4; the IQ test scores have in the following form:

```
test,gender,1,2,3,4,5,6,7,8,9
verbal,Boy,8455,14171,17596,29308,30490,27544,16037,9857,4635
verbal,Girl,5448,10570,15312,28591,32385,30830,18557,11443,5321
quantitative,Boy,3138,19634,18258,29037,23255,30376,16504,12565,5095
quantitative,Girl,2313,16905,19002,32707,26438,32413,15215,10007,3406
non-verbal,Boy,1390,18144,20713,29245,25720,27077,18095,11369,6077
non-verbal,Girl,1165,14370,18564,30488,29342,30458,18387,10450,5075
CAT3,Boy,2505,14505,19556,29917,29607,30327,17960,9392,2787
CAT3,Girl,1813,10927,17872,31059,32867,33269,18016,9041,2394
```

The `melt` function in the `reshape2` package transforms data.frames to a long format, such as the following (only the first 11 lines are shown):

```
           test gender stanine count
1        verbal    Boy      X1  8455
2        verbal   Girl      X1  5448
3  quantitative    Boy      X1  3138
4  quantitative   Girl      X1  2313
5    non-verbal    Boy      X1  1390
6    non-verbal   Girl      X1  1165
7          CAT3    Boy      X1  2505
8          CAT3   Girl      X1  1813
9        verbal    Boy      X2 14171
10       verbal   Girl      X2 10570
11 quantitative    Boy      X2 19634
```

which was reorganized using the call (where `b_g_IQ` contains the data):

```
b_g=melt(b_g_IQ, id.vars=c("test", "gender"),
                 variable.name="stanine", value.name="count")
```

It is also possible to convert from long to wide format.

10.5 Miscellaneous issues

10.5.1 Application specific cleaning

The analysis of some kinds of data has acquired established preprocessing procedures; the data is not wrong, but transforming it in some way improves the quality of the subsequent analysis. For instance, before analyzing text, common low interest words (such as 'the' and 'of', known as *stop words*) are removed; also words may be stemmed[69] (a process that removes suffixes with the intent of uncovering the root word, e.g., kicked and kicking become kick).

10.5.2 Different name, same meaning

Typos in character based data may be easy to detect because of constraints on what can appear in any sequence (e.g., the spelling of words). Harder to detect problems include different people using different terminology for the same concept or the same terminology for different concepts.

The SPEC 2006 benchmark results often include a description of the characteristics of the memory used by the computer under test. For historical marketing reasons two scales are commonly used to specify memory performance; the DDR scale is based on peak bandwidth while the PC scale uses clock rate. The SPEC result descriptions are not consistent in their choice of scale and so before any analysis can be performed the values need to be converted to either all DDR form or all PC form. Also, for marketing reasons the numbers have been rounded to reduce the number of non-zero digits; any analysis interested in high accuracy would map all the *marketing* values to their actual values (see rexample[SPEC-memory.awk]).

An email address is sometimes the only unique identifying information available, e.g., the list of developers who have contributed to an open source project. The same person may have

used more than one email address over the period of their involvement in a project and it is necessary to detect which addresses belong to the same person. The `find.aliases` function in the `tm.plugin.mail` package attempts to detect which email addresses refer to the same person.

10.5.3 Multiple sources of signals

Sometimes a value appearing in a sample could have come from multiple sources, only one of which is of interest. An example of this is the question: when did hexadecimal literals first appear as such in print?

One way of answering this question is to analyze the word n-grams (and associated year of book publication) Google have made available from their English book scanning project.[214]

The regular expression `^[0oO[xX][0-9a-fA-FoOl]]` (ohh, Ohh and ell were treated as the corresponding digits) returned 89 thousand matches.

OCR mistakes have resulted in some words being treated as hexadecimal literals, e.g., Oxford scanned as 0xf0fd. The character sequence oxo is surprisingly common and looking at some of the contexts in which this occurs suggests that the usage is mainly related to chemical formula (some are also likely to be references to a product of that name).

Assuming that hexadecimal notation did not start appearing in books before electronic computers were invented, books prior to say 1945, the end of World War II, can be ignored; let's also ignore oxo.

It seems to me that if any hexadecimal literal appears in a book at least one more is likely to occur; applying this final filter reduced the number of matches to 7,292; with 319 unique character sequences.

Comparing the use of hexadecimal literals in C source with those extracted from Google books words we get:

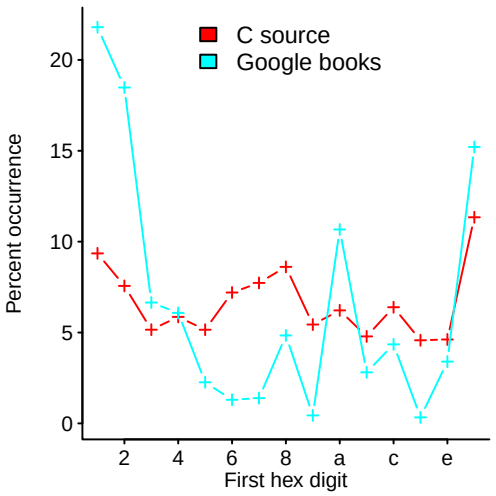


Figure 10.3: Percentage occurrence of the first digit of hexadecimal numbers in C source and estimated from Google book data. Data from Jones[299] and Michel et al.[388] code

10.5.4 Duplicate data

Duplicate data can occur for various reasons, including:

- collation of data from various sources, before it reaches the person performing the analysis, duplication of a particular list of values is always a possibility, e.g., all the values in one row/column match the values in another row/column or differ by a constant factor. This duplication may result in 100% correlation, e.g., one row contains temperature in Celsius while another uses Fahrenheit.
- repeated output from the measuring process. For instance, logging of computer faults where a single root cause can produce duplicate messages at sporadic times after the fault occurs[560] and spatially or functionally adjacent units to generate messages[356] (ee rexample[Blue-Gene.log] directory).

The `duplicated` function returns information about exact duplicate values appearing in rows.

When the data is numeric, *close* duplicates can be highlighted using correlation.

Some R functions handle duplicate row/columns gracefully (e.g., the `glm` function) while others give unpredictable results (e.g., the `solve` function which inverts a matrix), the behavior depends on the algorithm used and what if any consistency checks were added to the implementation by the person who wrote the code.

10.5.5 Default values

Sometimes a measurement process returns what is considered to be a reasonable value if it cannot return the actual value. For instance, IP geolocation services are always able to associate a country with an IP address, but when they are not able to further refine the location within a country they return a location near the center of the country; for the USA this is close to the town of Potwin in Kansas (population 449) which appears to experience orders of magnitude more Internet related events for its population size than other towns in the US.[291]

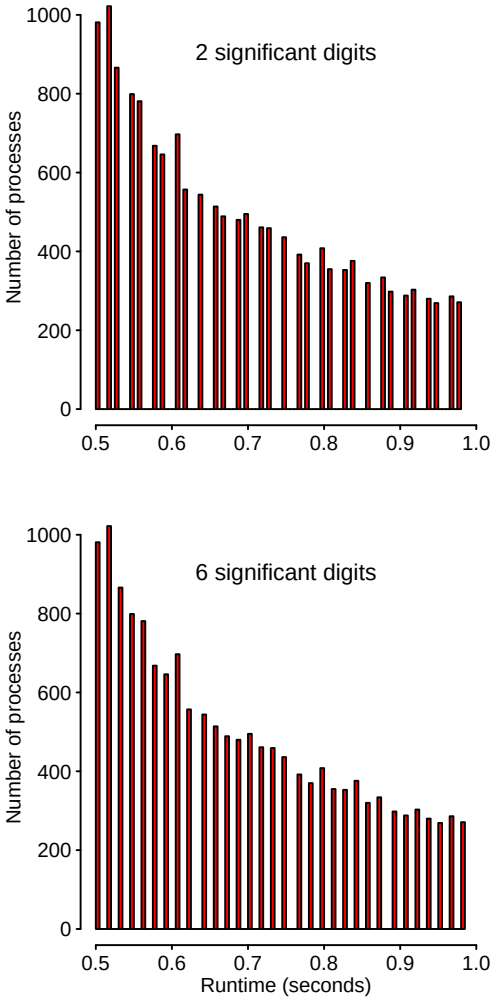


Figure 10.4: Number of processes executing for a given amount of time, with measurements expressed using two and six significant digits. Data from Feitelson.[174] code

10.5.6 Resolution limit of measurements

Some kinds of measurement are inherently inexact, e.g., time. When working close to the resolution limit of the measuring process care needs to be taken to ensure that false signals are not generated by an interaction between the measurement resolution and the analysis procedure.

A study by Feitelson[174] measured the runtime of processes executed on a Unix based system to an accuracy of two decimal digits. Subsequent analysis of the number of processes whose execution fell within a given time interval showed a surprising result, there were many time intervals that did not contain any processes (left plot in Figure 10.4). Subsequent analysis found that the timer resolution was 1/64 second and the gaps were an artefact of the number of digits recorded, recording more digits (left plot in Figure 10.4) resulted in fewer intervals containing no measurement points.

10.6 Detecting fabricated data

A data set is not always derived from accurate measurements, the accuracy failures may be accidental or intentional, or might not involve any actual measurements (i.e., it has been fabricated).

Like all data analysis detection of fabricated data is based on finding known patterns in the data and as always the interpretation of why the data contains these patterns is the responsibility of the audience of the results; it is always worth repeating that domain knowledge is key.

One pattern of behavior that has been observed to occur with some regularity is that the first digit of sample values follows Benford's law to a reasonable degree of approximation (while a figure of 30% of all datasets has been quoted, the actual figure is likely to be much smaller[508]). Benford's law has been used to detect accounting and election[491] fraud, identification of fake survey interviews[506] and scientific research.[147]

While references to Benford's law usually refer to the first digit of numeric values, there is a form that applies to the second and perhaps other significant digits.[412] There has also been work[47] suggesting that the digit at the opposite end of numeric literals, the least significant digit, sometimes has a uniform distribution.

Benford's law says the probability of the first digit having value d is given by:

$$P(d) = \log_{10}(1 + \frac{1}{d})$$

Figure **??** investigates counts of the first digit of numeric literals in C source code.

If a set of independent and identically distributed random variables are sorted into order the distribution of digits of the differences between adjacent sorted values is close to Benford's law.[392] A test based on this fact can detect rounded data, data generated by linear regression and data generated by using the inverse function of a known distribution.[412]

The `BenfordTests` package contains a variety of function for evaluating the conformity of a dataset to Benford's law.

When generating fabricated data, it is sometimes necessary to produce a random sequence of items. People hold incorrect beliefs[66] about the properties of random sequences and when asked to generate them produce sequences that contain predictable patterns (i.e., they are not random).

One study[507] was able to build a model that predicted repeated patterns in an individual's *randomly* selected sequence with around 25% success rate, but the model built for one person's behavior was used to make predictions about another persons the success rate dropped to around 18%.

Detecting divergence from or agreement with these patterns of behavior is dependent on the authors of the data set being lazy (i.e., being unwilling to spend the time making sure that the data they generate has the expected properties; the creators of the fictitious accounts publicly published by Madoff's companies, before his fraud was uncovered, made the effort to ensure that they followed Benford's law[510]) or unfamiliar with the expected patterns of behavior.

References

1. J. T. Abbott, J. L. Austerweil, and T. L. Griffiths. Random walks on semantic networks can resemble optimal foraging. *Psychological Review*, 122(3):558–569, July 2015.
2. J. Adams. *Risk and Freedom: The record of road safety regulation*. Transport Publishing Projects, 1985.
3. P. J. Ågerfalk. Insufficient theoretical contribution: a conclusive rationale for rejection? *European Journal of Information Systems*, 23(6):593–599, Nov. 2014.
4. A. Aghayev and P. Desnoyers. Skylight–A window on shingled disk operation. In *13th USENIX Conference on File and Storage Technologies (FAST'15)*, pages 135–149, Feb. 2015.
5. N. Agrawal. *Representative, reproducible, and practical benchmarking of file and storage systems*. PhD thesis, University of Wisconsin-Madison, 2009.
6. N. Agrawal, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Generating realistic impressions for file-system benchmarking. *ACM Transactions on Storage*, 5(4):125–138, Dec. 2009.
7. N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch. A five-year study of file-system metadata. *ACM Transactions on Storage*, 3(3):31–45, Oct. 2007.
8. H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman. Analysis of safety-critical computer failures in medical devices. *IEEE Security & Privacy*, 11(4):14–26, July 2013.
9. N. Ali, Z. Sharafi, Y.-G. Guéhéneuc, and G. Antoniol. An empirical study on requirements traceability using eye-tracking. In *28th IEEE International Conference on Software Maintenance (ICSM), 2012*, pages 191–200, Sept. 2012.
10. M. G. Almiron, E. S. Almeida, and M. N. Miranda. The reliability of statistical functions in four software packages freely used in numerical computation. *Brazilian Journal of Probability and Statistics*, 23(2):107–119, 2009.
11. M. G. Almiron, B. Lopes, A. L. C. Oliveira, A. C. Medeiros, and A. C. Frery. On the numerical accuracy of spreadsheets. *Journal of Statistics*, 34(4):1–29, Apr. 2010.
12. L. E. Alteneder. The learning curve in solving a jig-saw puzzle: A teaching device. *Journal of Educational Psychology*, 26(3):231–232, Mar. 1935.
13. E. M. Altmann. *Episodic Memory for External Information*. PhD thesis, Carnegie Mellon University, Aug. 1996.
14. E. M. Altmann. Functional decay of memory for tasks. *Psychological Research*, 66(4):287–297, 2002.
15. E. M. Altmann, J. G. Trafton, and D. Z. Hambrick. Effects of interruption length on procedural errors. *Journal of Experimental Psychology: Applied*, ???(???):???, Dec. 2016.
16. Amazon, Inc. Amazon ec2 service level agreement. ???, June 2013.
17. J. M. Amiri and V. V. K. Padmanabhuni. A comprehensive evaluation of conversion approaches for different function points. Thesis (m.s.), Blekinge Institute of Technology, Sweden, Sept. 2011.
18. L. V. B. An T. Oskarsson, G. H. McClelland, and R. Hastie. What's next? Judging sequences of binary events. *Psychological Bulletin*, 135(2):262–285, 2009.
19. They're (almost) all dirty: The state of cheating in Android benchmarks. AnandTech website, Oct. 2013. webpage???
20. J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, Apr. 2007.
21. J. R. Anderson and R. Milson. Human memory: An adaptive perspective. *Psychological Review*, 96(4):703–719, 1989.
22. M. L. Anderson. Neural reuse: A fundamental organizational principle of the brain. *Behavioral and Brain Sciences*, 33(4):245–313, Apr. 2010.
23. M. H. and Katerina Goseva-Popstojanova. Exploring the missing link: an empirical study of software fixes. *Software Testing, Verification and Reliability*, 24(8):684–705, Dec. 2014.
24. F. J. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, Feb. 1973.
25. H. R. Arkes, R. M. Dawes, and C. Christensen. Factors influencing the use of a decision rule in a probabilistic task. *Organizational Behavior and Human Decision Processes*, 37:93–110, 1986.
26. T. B. Arnold and J. W. Emerson. Nonparametric goodness-of-fit tests for discrete null distributions. *The R Journal*, 3(2):34–39, Dec. 2011.
27. A. Arora, R. Krishnan, R. Telang, and Y. Yang. An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. *Information Systems Research*, 21(1):115–132, Mar. 2010.
28. S. E. Asch. Studies of independence and conformity: A minority of one against a unanimous majority. *Psychological Monographs: General and Applied*, 70(9):1–70, 1956.
29. T. A. Åstebro, S. A. Jeffrey, and G. K. Adomdza. Inventor perseverance after being told to quit: The role of cognitive biases. *Journal of Behavioral Decision Making*, 20(3):253–272, Apr. 2007.
30. P. Azoulay, C. Fons-Rosen, and J. S. G. Zivin. Does science advance one funeral at a time? Technical Report NBER Working Paper No. 21788, National Bureau of Economic Research, Dec. 2015.
31. V. Babka. *Improving Accuracy of Software Performance Models on Multicore Platforms with Shared Caches*. PhD thesis, Faculty of Mathematics and Physics, Charles University in Prague, Oct. 2012.
32. V. Babka and P. Tůma. Investigating cache parameters of x86 family processors. In *Proceedings of the 2009 SPEC Benchmark Workshop on Computer Performance Evaluation and Benchmarking*, pages 77–96, Jan. 2009.
33. A. Bacon, S. Handley, and S. Newstead. Individual differences in strategies for syllogistic reasoning. *Thinking & Reasoning*, 9(2):133–168, 2003.
34. A. Baddeley. Working memory. In A. Baddeley, M. W. Eysenck, and M. Anderson, editors, *Memory*, chapter 3, pages 41–69. Psychology Press, Feb. 2009.
35. A. Baddeley. Working memory: Theories, models, and controversies. *Annual Review of Psychology*, 63:1–29, Sept. 2012.
36. A. D. Baddeley, N. Thomson, and M. Buchanan. Word length and the structure of short-term memory. *Journal of Verbal Learning and Verbal Behavior*, 14:575–589, 1975.
37. J. N. Bailenson, M. S. Shum, S. Atran, D. L. Medin, and J. D. Coley. A bird's eye view: biological categorization and reasoning within and across cultures. *Cognition*, 84:1–53, 2002.
38. D. H. Bailey. Misleading performance reporting in the supercomputer field. Technical Report RNR-92-005, Numerical Aerodynamic Simulation Division, NASA Ames Research Center, Dec. 1992.
39. M. Bakkaloglu, J. J. Wylie, C. Wang, and G. R. Ganger. On correlated failures in survivable storage systems. Technical Report CMU-CS- 02-129, Carnegie Mellon University, May 2002.
40. B. Balaji, J. McCullough, R. K. Gupta, and Y. Agarwal. Accurate characterization of the variability in power consumption in modern mobile processors. In *HotPower'12 Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, page ???, Oct. 2012.
41. P. Banyard and N. Hunt. Something missing? *The Psychologist*, 13(2):68–71, 2000.
42. J. H. Barkow, L. Cosmides, and J. Tooby. *The Adapted Mind: Evolutionary Psychology and the Generation of Culture*. Oxford University Press, 1992.
43. A. Baronchelli, V. Loreto, and A. Puglisi. Individual biases, cultural evolution, and the statistical nature of language universals: The case of colour naming systems. *PLoS ONE*, 10(5):e0125019, May 2015.
44. L. Barrett, R. Dunbar, and J. Lycett. *Human Evolutionary Psychology*. Palgrave Macmillan, 2002.
45. V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sørumgård, and M. V. Zelkowitz. The empirical investigation of perspective-based reading. In *Proceedings of the Twentieth Annual Software Engineering Workshop*, pages 21–69, Dec. 1995.
46. R. F. Baumeister. *Is There Anything Good About Men?* Oxford University Press, 2010.
47. B. Beber and A. Scacco. What the numbers say: A digit-based test for election fraud. *Political Analysis*, 20(2):211–234, Apr. 2012.
48. R. A. Becker and W. S. Cleveland. *Trellis Graphics User's Manual*. AT&T Bell Laboratories, Murray Hill, Dec. 1995.
49. A. Begel and T. Zimmermann. Analyze this! 145 questions for data scientists in software engineering. Technical Report MSR-TR-2013-111, Microsoft Research, Oct. 2013.
50. M. Bekoff, C. Allen, and G. M. Burghardt. *The Cognitive Animal: Empirical and Theoretical Perspectives on Animal Cognition*. MIT Press, 2002.

51. V. A. Bell and P. N. Johnson-Laird. A model theory of modal reasoning. *Cognitive Science*, 22(1):25–51, 1998.
52. J. Bennet B. Murdoch. The serial position effect of free recall. *Journal of Experimental Psychology*, 64(5):482–488, 1962.
53. T. Berger, S. She, K. Czarnecki, and A. Wąsowski. Feature-to-code mapping in two large product lines. In J. Bosch and J. Lee, editors, *Software Product Lines: Going Beyond*, volume 6287 of *Lecture Notes in Computer Science*, chapter ???, pages 498–499. Springer Berlin Heidelberg, 2010.
54. T. Berger, S. She, R. Lotufo, A. Wąsowski, and K. Czarnecki. Variability modeling in the systems software domain. Technical Report GSDLAB-TR 2012-07-06, Generative Software Development Laboratory, University of Waterloo, July 2012.
55. B. Berlin and P. Kay. *Basic Color Terms: Their Universality and Evolution*. Berkeley: University of California Press, 1969.
56. K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance CMOS variability in the 65-nm regime and beyond. *IBM Journal of Research and Development*, 50(4/5):433–449, July 2006.
57. K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In C. Beeri and P. Buneman, editors, *Database Theory –ICDT'99: 7th International Conference*, pages 217–235. Springer-Verlag, Jan. 1999.
58. D. Biber, S. Johansson, G. Leech, S. Conrad, and E. Finegan. *Longman Grammar of Spoken and Written English*. Pearson Education, 1999.
59. S. Bikhchandani, D. Hirshleifer, and I. Welch. A theory of fads, fashion, custom, and cultural change as informational cascades. *Journal of Political Economy*, 100(5):992–1026, Oct. 1992.
60. W. L. Bircher. *Predictive Power Management for Multi-Core Processors*. PhD thesis, The University of Texas at Austin, Dec. 2010.
61. S. Bird. Software knows best: A case for hardware transparency and measurability. Thesis (m.s.), Department of Electrical Engineering and Computer Science, University of California at Berkeley, May 2010.
62. T. F. Bissyandé, F. Thung, D. Lo, L. Jiang, and L. Réveillère. Popularity, interoperability, and impact of programming languages in 100,000 open source projects. In *37th Annual International Computer Software & Applications Conference (COMPSAC 2013)*, pages 303–312, July 2013.
63. S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khan, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann. The DaCapo benchmarks: Java benchmarking development and analysis (extended version). Technical Report TR-CS-06-01, Department of Computer Science, Australian National University, Aug. 2006.
64. A.-R. Blais and E. U. Weber. A domain-specific risk-taking (DOSPERT) scale for adult populations. *Judgment and Decision Making*, 1(1):33–47, Apr. 2006.
65. M. S. Blaubergs and M. D. S. Braine. Short-term memory limitations on decoding self-embedded sentences. *Journal of Experimental Psychology*, 102(4):745–748, 1974.
66. D. S. Blinder and D. M. Oppenheimer. Beliefs about what types of mechanisms produce random sequences. *Journal of Behavioral Decision Making*, 21(4):414–427, Oct. 2008.
67. B. W. Boehm. *Software Engineering Economics*. Prentice-Hall, Inc, 1981.
68. G. D. Boetticher. Improving credibility of machine learner models in software engineering. In D. Zhang and J. J. P. Tsai, editors, *Advances in Machine Learning Applications in Software Engineering*, chapter 3, pages 52–73. Idea Group Publishing, Oct. 2006.
69. A. Bohn, I. Feinerer, K. Hornik, and P. Mair. Content-based social network analysis of mailing lists. *The R Journal*, 3(1):11–18, June 2011.
70. A. Börsch-Supan and M. Weiss. Productivity and age: Evidence from work teams at the assembly line. Technical Report 148-2007, Manheim Research Institute for the Economics of Aging, 2007.
71. N. Bostrom and A. Sandberg. The wisdom of nature: An evolutionary heuristic for human enhancement. In J. Savulescu and N. Bostrom, editors, *Human Enhancement*, chapter 18, pages 375–416. Oxford University Press, Jan. 2011.
72. T. F. Brady, T. Konkle, G. A. Alvarez, and A. Oliva. Visual long-term memory has a massive storage capacity for object details. *PNAS*, 105(38):14325–14329, Sept. 2008.
73. D. W. Braithwaite and R. L. Goldstone. Flexibility in data interpretation: effects of representational format. *Frontiers in Psychology*, 4(980):1–16, Dec. 2013.
74. B. Brembs, K. Button, and M. Munafò. Deep impact: Unintended consequences of journal rank. *Frontiers in Human Neuroscience*, 7(291), June 2013.
75. S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: Improving cooperation between developers and users. In *CSCW '10 Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 301–310, Feb. 2010.
76. C. A. Brewer. Color use guidelines for mapping and visualization. In A. M. Maceachren and D. R. F. Taylor, editors, *Visualization in Modern Cartography*, chapter 7, pages 123–147. Pergamon, Nov. 1994.
77. E. Brewer, L. Ying, L. Greenfield, R. Cypher, and T. Ts'o. Disks for data centers. Technical report, Google, Inc, Feb. 2016.
78. S. Broadbent. Font requirements for next generation air traffic management systems. Technical Report HRS/HSP-006-REP-01, European Organisation for the Safety of Air Navigation, 2000.
79. J. Brooke. SUS: A 'quick' and 'dirty' usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, and I. L. McClelland, editors, *Usability Evaluation in Industry*, chapter 21, pages 189–194. Taylor and Francis, June 1996.
80. J. Brooke. SUS: A retrospective. *Journal of Usability Studies*, 8(2):29–40, Feb. 2013.
81. F. P. Brooks, Jr. *The Mythical Man-Month*. Addison–Wesley, anniversary edition, 1995.
82. J. Brunner and P. C. Austin. Inflation of Type I error rate in multiple regression when independent variables are measured with error. *The Canadian Journal of Statistics*, 37(1):33–46, Mar. 2009.
83. I. Buchmann. *Batteries in a Portable World: A Handbook on rechargeable Batteries for Non-engineers*. Cadex Electronix Inc, third edition, 2011.
84. J. B. Buckheit and D. L. Donoho. WaveLab and reproducible research. In A. Antoniadis and G. Oppenheim, editors, *Wavelets and Statistics*, chapter 5, pages 55–81. Springer-Verlag, 1995.
85. M. Budden, P. Hadavas, L. Hoffman, and C. Pretz. Generating valid 4×4 correlation matrices. *Applied Mathematics E-Notes*, 7:53–59, 2007.
86. D. J. Buettner. *Designing an Optimal Software Intensive System Acquisition: A Game Theoretic Approach*. PhD thesis, University of Southern California, Sept. 2008.
87. K. P. Burnham and D. R. Anderson. Multimodel inference: Understanding AIC and BIC in model selection. *Sociological Methods and Research*, 33(2):261–304, Nov. 2004.
88. Q. L. Burrell. A note on ageing in a library circulation model. *Journal of Documentation*, 41(2):100–115, 1985.
89. J. Businge. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. PhD thesis, Eindhoven University of Technology, Sept. 2013.
90. J. Businge, A. Serebrenik, and M. van den Brand. Survival of Eclipse third-party plug-ins. In *ICSM '12 Proceedings of the 2012 IEEE International Conference on Software Maintenance*, pages 368–377, Sept. 2012.
91. J. Calhoun, C. Savoie, M. Randolph-Gips, and I. Bozkurt. Human reliability analysis in spaceflight applications. *Quality and Reliability Engineering International*, 29(6):869–882, Aug. 2013.
92. J. I. D. Campbell. On the relation between skilled performance of simple division and multiplication. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 23(5):1140–1159, 1997.
93. G. Canfora, L. Cerulo, M. Cimitile, and M. Di Penta. Social interactions around cross-system bug fixings: the case of FreeBSD and OpenBSD. In *MSR'11 Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 143–152, May 2011.
94. S. Carter-Thomas and E. Rowley-Jolivet. *If*-conditionals in medical discourse: From theory to disciplinary practice. *Journal of English for Academic Purposes*, 7(3):191–205, July 2008.
95. J. P. Cavanagh. Relation between the immediate memory span and the memory search rate. *Psychological Review*, 79(6):525–530, 1972.
96. F. Chandler, I. A. Heard, M. Presley, A. Burg, E. Midden, and P. Mongan. NASA human error analysis. Technical report, NASA Office of Safety and Mission Assurance, Sept. 2010.
97. F. T. Chandler, Y. H. J. Chang, A. Mosleh, J. L. Marble, R. L. Boring, and D. I. Gertman. Human reliability analysis methods: Selection guidance for NASA. Technical Report ???, NASA/OSMA, July 2006.

98. V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection - A survey. *ACM Computing Surveys*, 41(3):1–58, July 2009.
99. K. Chandrasekar. *High-Level Power Estimation and Optimization of DRAMs*. PhD thesis, Technische Universiteit Delft, Oct. 2014.
100. A. C. Chang and P. Li. Is economics research replicable? Sixty published papers from thirteen journals say "usually not". Technical Report Finance and Economics Discussion Series 2015-083, Washington: Board of Governors of the Federal Reserve System, Sept. 2015.
101. W. Chang. *R Graphics Cookbook*. O'Reilly, 2012.
102. G. Charness and U. Gneezy. Strong evidence for gender differences in risk taking. *Journal of Economic Behavior & Organization*, 83(1):50–58, June 2012.
103. W. G. Chase and K. A. Ericsson. Skill and working memory. In G. H. Bower, editor, *The Psychology of Learning and Motivation*, pages 1–58. Academic, 1982.
104. T. Chen, Y. Chen, Q. Guo, O. Temam, T. Wu, and W. Hu. Statistical performance comparisons of computers. In *18th International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12, Feb. 2012.
105. Y. Chen, A. Groce, X. Fern, C. Zhang, W.-K. Wong, E. Eide, and J. Regehr. Taming compiler fuzzers. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'13*, pages 197–208, June 2013.
106. P. W. Cheng, K. J. Holyoak, R. E. Nisbett, and L. M. Oliver. Pragmatic versus syntactic approaches to training deductive reasoning. *Cognitive Psychology*, 18:293–328, 1986.
107. R. C. Cheung. A user-oriented software reliability model. *IEEE Transactions on Software Engineering*, 6(2):118–125, 1980.
108. J. Y. Chiao, A. R. Bordeaux, and N. Ambady. Mental representations of social status. *Cognition*, 93(???):B49–B57, Apr. 2004.
109. CRAN task view: Probability distributions. website, June 2016. http://CRAN.R-project.org/view=Distributions.
110. A. CIA. Analytic thinking and presentation for intelligence producers: Analysis training handbook. Technical report, Office of Training and Education, Central Intelligence Agency, Aug. 199?
111. D. Citron. MisSPECulation: Partial and misleading use of SPEC CPU2000 in computer architecture conferences. In *ISCA '03 Proceedings of the 30th annual international symposium on Computer architecture*, pages 52–61, June 2003.
112. D. Citron and D. G. Feitelson. "look it up" or "do the math": An energy, area, and timing analysis of instruction resuse and memoization. Technical Report H-0196, IBM, Oct. 2003.
113. A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009.
114. W. S. Cleveland. *The Elements of Graphing Data*. Wadsworth Advanced Book Program, 1985.
115. W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, Sept. 1984.
116. J. Cohen. *Statistical Power Analysis for the Behavioural Sciences*. Routledge, second edition, 1988.
117. J. Cohen. The Earth is round ($p < 0.05$). *American Psychologist*, 49(12):997–1003, 1994.
118. M. Cokol, I. Iossifov, R. Rodriguez-Esteban, and A. Rzhetsky. How many scientific papers should be retracted? *European Molecular Biology Organization*, 8(5):422–423, Apr. 2007.
119. Numbers every programmer should know. website, Oct. 2016. https://github.com/colin-scott/interactive_latencies.
120. C. Collberg, T. Proebsting, and A. M. Warren. Repeatability and benefaction in computer systems research –A study and a modest proposal. Technical Report TR 14-014, Department of Computer Science, University of Arizona, Feb. 2015.
121. B. Conrad and M. Mitzenmacher. Power laws for monkeys typing randomly: The case of unequal probabilities. *IEEE Transactions on Information Theory*, 50(7):1403–1414, July 2004.
122. J. J. Cook and C. Zilles. A characterization of instruction-level error derating and its implications for error detection. In *DSN 2008. IEEE International Conference on Dependable Systems and Networks With FTCS and DCC*, pages 482–491, June 2008.
123. J. Corbet, G. Kroah-Hartman, and A. McPherson. Linux kernel development: How fast it is going, who is doing it, what they are doing, and who is sponsoring it. Technical report, The Linux Foundation, Mar. 2012.
124. M. Correll and M. Gleicher. Error bars considered harmful: Exploring alternate encodings for mean and error. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2142–2151, Dec. 2014.
125. g, a statistical myth. Three-Toed Sloth, blog, Oct. 2007. http://bactra.org/weblog/523.html.
126. L. Cosmides and J. Tooby. Evolutionary psychology: A primer. Technical report, Center for Evolutionary Psychology, University of California, Santa Barbara, 1998.
127. N. Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1):87–185, 2001.
128. F. E. Croxton and R. E. Stryker. Bar charts versus circle diagrams. *Journal of the American Statistical Association*, 22(160):473–482, Dec. 1927.
129. G. Cumming and R. Maillardet. Confidence intervals and replication: Where will the next mean fall? *Psychological Methods*, 11(3):217–227, 2006.
130. A. Damasio. *Self Comes to Mind: Constructing the Conscious Brain*. Vintage books, 2012.
131. A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz. CPU DB: Recording microprocessor history. *Communications of the ACM*, 55(4):55–63, Apr. 2012.
132. J. Darley and C. D. Batson. From Jerusalem to Jericho: A study of situational and dispositional variables in helping behavior. *Journal of Personality and Social Psychology*, 27(1):100–108, 1973.
133. S. Dayal. Characterizing HEC storage systems at rest. Technical Report CMU-PDL-08-109, Parallel Data Laboratory, Carnegie Mellon University, July 2008.
134. A. D. de Groot. *Thought and Choice in Chess*. Amsterdam University Press, 2008.
135. A. B. de Oliveira, J.-C. Petkovich, T. Reidemeister, and S. Fischmeister. DataMill: Rigorous performance evaluation made easy. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering, ICPE'13*, pages 137–148, Apr. 2013.
136. C. B. De Soto, M. London, and S. Handel. Social reasoning and spatial paralogic. *Journal of Personality and Social Psychology*, 2(4):513–521, 1965.
137. K. De Vogeleer. *La loi de convexité énergie-fréquence de la consommation des programmes : modélisation, thermosensibilité et applications*. PhD thesis, Informatique [cs] Telecom ParisTech, Sept. 2015.
138. I. J. Deary. *Intelligence: A Very Short Introduction*. Oxford University Press, 2001.
139. S. Dehaene. Symbols and quantities in parietal cortex: elements of a mathematical theory of number representation and manipulation. In P. Haggard, Y. Rossetti, and M. Kawato, editors, *Sensorimotor Foundations of Higher Cognition (Attention and Performance) XXII*, chapter 24, pages 527–574. Oxford University Press, Nov. 2007.
140. S. Dehaene. *Reading in the Brain: The Science and evolution of a human invention*. Viking, 2009.
141. S. Dehaene. *The Number Sense*. Oxford University Press, revised and updated edition, 2011.
142. S. Dehaene, E. Dupoux, and J. Mehler. Is numerical comparison digits? Analogical and symbolic effects in two-digit number comparisons. *Journal of Experimental Psychology: Human Perception and Performance*, 16(3):626–641, 1990.
143. S. Dehaene, V. Izard, E. Spelke, and P. Pica. Log or linear? Distinct intuitions of the number scale in Western and Amazonian indigene cultures. *Science*, 320(5880):1217–1220, May 2008.
144. S. DellaVigna. Psychology and economics: Evidence from the field. Technical Report 13420, National Bureau of Economic Research, USA, Sept. 2007.
145. M. Di Penta, L. Cerulo, and L. Aversano. The life and death of statically detected vulnerabilities: an empirical study. *Information and Software Technology*, 51(10):1469–1484, Oct. 2009.
146. L. S. Dickstein. The effect of figure on syllogistic reasoning. *Memory & Cognition*, 6(1):76–83, 1978.
147. A. Diekmann. Not the first digit! using Benford's law to detect fraudulent scientific data. *Journal of Applied Statistics*, 34(3):321–329, Oct. 2007.
148. J. Dietrich, K. Jezek, and P. Brada. What Java developers know about compatibility, and why this matters. Technical report, School of Engineering and Advanced Technology, Massey University, Aug. 2014.

149. C. DiMarco, G. Hirst, and M. Stede. The semantic and stylistic differentiation of synonyms and near-synonyms. In *AAAI Spring Symposium on Building Lexicons for Machine Translation*, pages 114–121, Mar. 1993.

150. D. K. Dirlam. Most efficient chunk sizes. *Cognitive Psychology*, 3:355–359, 1972.

151. J. R. Douceur and W. J. Bolosky. A large-scale study of file-system contents. In *SIGMETRICS '99 Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 59–70, July 1999.

152. J. R. Dunham and J. L. Pierce. An experiment in software reliability. Technical Report NASA Contractor report 172553, NASA Langley Research Center, Mar. 1986.

153. D. Dunning, C. Heath, and J. M. Suls. Flawed self-assessment: Implications for health, education, and the workplace. *Psychology of Science in the Public Interest*, 5(3):69–106, Apr. 2004.

154. T. Dybå, V. B. Kampenes, and D. I. K. Sjøberg. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, 48(8):745–755, Aug. 2006.

155. The changing US technology sector. Daily chart for April 21 2015 on The Economist webpage, Apr. 2015. As of Q1 2015, Sources: Thomson Reuters; awk scripts+R converted the data embedded in Javascript.

156. A. Edmundson, B. Holtkamp, E. Rivera, M. Finifter, A. Mettler, and D. Wagner. An empirical study on the effectiveness of security code review. In *ESSoS'13 Proceedings of the 5th international conference on Engineering Secure Software and Systems*, pages 197–212, Feb. 2013.

157. K. El Emam, S. Benlarbi, N. Goel, W. Melo, H. Lounis, and S. N. Rai. The optimal class size for object-oriented software. *IEEE Transactions on Software Engineering*, 28(5):494–509, Mar. 2002.

158. N. C. Ellis and R. A. Hennelly. A bilingual word-length effect: Implications for intelligence testing and the relative ease of mental calculation in Welsh and English. *British Journal of Psychology*, 71:43–51, 1980.

159. P. D. Ellis. *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*. Cambridge University Press, 2010.

160. R. Engbert, A. Nuthmann, E. M. Richter, and R. Kliegl. SWIFT: A dynamical model of saccade generation during reading. *Psychological Review*, 112(4):777–813, Apr. 2005.

161. J. Engblom. Why SpecInt95 should not be used to benchmark embedded systems tools. *ACM SIGPLAN Notices*, 34(7):96–103, July 1999.

162. D. M. Erceg-Hurn and V. M. Mirosevich. Modern robust statistical methods. *American Psychologist*, 63(7):591–601, Oct. 2008.

163. K. A. Ericsson and N. Charness. Expert performance. *American Psychologist*, 49(8):725–747, Aug. 1994.

164. K. A. Ericsson, R. T. Krampe, and C. Tesch-Römer. The role of deliberate practice in the acquisition of expert performance. *Psychological Review*, 100(3):363–406, 1993. also University of Colorado, Technical Report #91-06.

165. K. A. Ericsson and A. C. Lehmann. Expert and exceptional performance: Evidence of maximal adaption to task constraints. *Annual Review of Psychology*, 47:273–305, 1996.

166. H. Esmaeilzadeh, T. Cao, X. Yang, S. M. Blackburn, and K. S. McKinley. Looking back on the language and hardware revolutions: Measured power, performance, and scaling. In *ASPLOS XVI Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, pages 319–332, Mar. 2011.

167. W. K. Estes. *Classification and Cognition*. Oxford University Press, 1994.

168. J. S. B. T. Evans, J. L. Barston, and P. Pollard. On the conflict between logic and belief in syllogistic reasoning. *Memory & Cognition*, 11(3):295–306, 1983.

169. J. Eyolfson, L. Tan, and P. Lam. Do time of day and developer experience affect commit bugginess? In *MSR'11 Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 153–162, May 2011.

170. J. Eyolfson, L. Tan, and P. Lam. Correlations between bugginess and time-based commit characteristics. *Empirical Software Engineering*, 19(4):1009–1039, Aug. 2014.

171. D. Fanelli. How many scientists fabricate and falsify research? A systematic review and meta-analysis of survey data. *PLoS ONE*, 4(5):e5738, May 2009.

172. D. Fanelli. "Positive" results increase down the hierarchy of the sciences. *PLoS ONE*, 5(4):e10068, Apr. 2010.

173. F. C. Fang, R. G. Steen, and A. Casadevall. Misconduct accounts for the majority of retracted scientific papers. *PNAS*, 109(42):17028–17033, Oct. 2012.

174. D. G. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, 2014.

175. D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing: Lect. Notes Comput. Sci. vol. 949*, chapter 19, pages 337–360. Springer-Verlag, June 1995.

176. J. Feldman. Minimization of boolean complexity in human concept learning. *Nature*, 407:630–633, Oct. 2000.

177. J. Feldman. An algebra of human concept learning. *Journal of Mathematical Psychology*, 50(4):339–368, Aug. 2006.

178. N. Fenton, M. Neil, W. Marsh, P. Hearty, Ł. Radliński, and P. Krause. On the effectiveness of early life cycle defect prediction with Bayesian nets. *Empirical Software Engineering*, 13(5):499–537, Oct. 2008.

179. A. Filippin and P. Crosetto. A reconsideration of gender differences in risk attitudes. Technical Report IZA DP No. 8184, The Institute for the Study of Labor, Bonn, May 2014.

180. C. J. Fillmore. Topics in lexical semantics. In R. W. Cole, editor, *Current Issues in Linguistic Theory*, pages 76–138. Indiana University Press, 1977.

181. K. Flamm. *Targeting the Computer*. The Brookings Institution, Washington, D.C., 1987.

182. K. Flamm. *Creating the Computer*. The Brookings Institution, Washington, D.C., 1988.

183. D. Flater. Estimation of uncertainty in application profiles. Technical Report NIST.TN.1826, National Institute of Standards and Technology, Apr. 2014.

184. D. Flater. Screening for factors affecting application performance in profiling measurements. Technical Report NIST Technical Note 1855, National Institute of Standards and Technology, Oct. 2014.

185. D. Flater and W. F. Guthrie. A case study of performance degradation attributable to run-time bounds checks on C++ vector access. *Journal of Research of the National Institute of Standards and Technology*, 118(012):260–279, May 2013.

186. J. I. Flombaum, J. A. Junge, and M. D. Hauser. Rhesus monkeys (*macaca mulatta*) spontaneously compute addition operations over large numbers. *Cognition*, 97(3):315–325, Oct. 2005.

187. J. Fodor. *The Modularity of Mind: An Essay on Faculty Psychology*. MIT Press, 1983.

188. R. A. Foley. An evolutionary and chronological framework for human social behaviour. *Proceedings of the British Academy*, 88:95–117, 1996.

189. C. E. Ford and S. A. Thompson. Conditionals in discourse: A text-based study from English. In E. C. Traugott, A. T. Meulen, J. S. Reilly, and C. A. Furguson, editors, *On Conditionals*, chapter 18, pages 353–372. Cambridge University Press, 1986.

190. J. Förster, E. Higgins, and A. T. Bianco. Speed/accuracy decisions in task performance: Built-in trade-off or separate strategic concerns? *Organizational Behavior and Human Decision Processes*, 90(1):148–164, Jan. 2003.

191. M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison–Wesley, 1999.

192. P. A. Freund and N. Kasten. How smart do you think you are? A meta-analysis on the validity of self-estimates of cognitive ability. *Psychological Bulletin*, 138(2):296–321, Mar. 2011.

193. W.-T. Fu and W. D. Gray. Memory versus perceptual-motor tradeoffs in a blocks world task. In *Proceedings of the Twenty-second Annual Conference of the Cognitive Science Society*, pages 154–159, Hillsdale, NJ, 2000. Erlbaum.

194. B. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys*, 42(4):1–53, June 2010.

195. M. T. Gailliot and R. F. Baumeister. The physiology of willpower: Linking blood glucose to self-control. *Personality and Social Psychology Review*, 11(4):303–327, Nov. 2007.

196. W. A. Gale. Good-Turing smoothing without tears. Technical Report 94.5, AT&T Bell Laboratories, Aug. 1994.

197. T. J. Gandomani, K. T. Wei, and A. K. Binhamid. A case study research on software cost estimation using experts' estimates, Wideband Delphi, and Planning Poker technique. *International Journal of Software Engineering and Its Applications*, 8(11):173–182, Apr. 2014.

198. S. A. Gelman and E. M. Markman. Categories and induction in young children. *Cognition*, 23:183–209, 1986.

199. D. Gentner and S. Goldin-Meadow. *Language In Mind: Advances in the Study of Language and Thought*. MIT Press, 2003.

200. E. H. Gibbs, G. A. Munroe, A. M. Zeman, and C. T. Cottingham. *JANET SKOLD and DAVID DOSSANTOS, on behalf of themselves and all others similarly situated and the general public, v. INTEL CORPORATION, HEWLETT PACKARD COMPANY and DOES 1-50, Case No. 1-05-CV-039231, Filing #G-43414*. Superior court of the state of California for the county of Santa Clara, May 2012.

201. G. Gigerenzer. *Rationality for Mortals –How People cope with Uncertainty*. Oxford University Press, 2008.

202. G. Gigerenzer, W. Gaissmaier, E. Kurz-Milcke, L. M. Schwartz, and S. Woloshin. Helping doctors and patients make sense of health statistics. *Psychological Science in the Public Interest*, 8(2):53–96, Apr. 2008.

203. G. Gigerenzer, S. Krauss, and O. Vitouch. The null ritual: What you always wanted to know about significance testing but were afraid to ask. In D. Kaplan, editor, *The Sage handbook of quantitative methodology for the social sciences*, chapter 21, pages 391–408. SAGE Publications, Inc, 2004.

204. G. Gigerenzer, P. M. Todd, and The ABC Research Group. *Simple Heuristics That Make Us Smart*. Oxford University Press, 1999.

205. V. Girotto, A. Mazzocco, and A. Tasso. The effect of premise order on conditional reasoning: a test of the mental model theory. *Cognition*, 63:1–28, 1997.

206. F. Gobet. *Understanding Expertise: A Multi-disciplinary Approach*. Palgrave, 2016.

207. D. R. Godden and A. D. Baddeley. Context-dependent memory in two natural environments: On land and underwater. *British Journal of Psychology*, 66(3):325–331, 1975.

208. M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In 16^{th} *International Conference on Software Maintenance*, pages 131–142, Oct. 2000.

209. L. R. Goldberg, J. A. Johnson, H. W. Eber, R. Hogan, M. C. Ashton, C. R. Cloninger, and H. G. Gough. The international personality item pool and the future of public-domain personality measures. *Journal of Research in Personality*, 40(1):84–96, 2006.

210. E. Goldvarg and P. Johnson-Laird. Naive causality: a mental model theory of causal meaning and reasoning. *Cognitive Science*, 25(4):565–610, July 2001.

211. P. Golle. Revisiting the uniqueness of simple demographics in the US population. In *WPES'06 Proceedings of the 5th ACM workshop on Privacy in electronic society*, pages 77–80, Oct. 2006.

212. J. M. González-Barahona, G. Robles, I. Herraiz, and F. Ortega. Studying the laws of software evolution in a long-lived FLOSS project. *Journal of Software Evolution and Process*, 26(7):589–612, July 2014.

213. B. H. Good, Y.-A. de Montjoye, and A. Clauset. The performance of modularity maximization in practical contexts. Technical Report arXiv:0910.0165v2, ???, Apr. 2010.

214. Google books ngram dataset. website, 2015. http://storage.googleapis.com/books/ngrams/books/datasetsv2.html.

215. R. Gopinath, A. Alipour, I. Ahmed, C. Jensen, and A. Groce. How hard does mutation analysis have to be, anyway? In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, pages 216–227, Nov. 2015.

216. R. Gopinath, C. Jensen, and A. Groce. Code coverage for suite evaluation by developers. In *ICSE'14 Proceedings of the 36th International Conference on Software Engineering*, pages 72–82, June 2014.

217. R. J. Gordon. The postwar evolution of computer prices. Technical Report NBER Working Paper No. 2227, National Bureau of Economic Research, Apr. 1987.

218. M. Gottscho. ViPZonE: Exploiting DRAM power variability for energy savings in Linux x86-64. Thesis (m.s.), Electrical Engineering, UCLA, Mar. 2014.

219. M. Gottscho, A. A. Kagalwalla, and P. Gupta. Power variability in contemporary DRAMs. *IEEE Embedded Systems Letters*, 4(12):37–40, June 2012.

220. S. Götz, T. Ilsche, J. Cardoso, J. Spillner, U. Aßmann, W. Nagel, and A. Schill. Energy-efficient data processing at sweet spot frequencies. In *OTM 2014 Workshops*, pages 154–171, Apr. 2014.

221. G. Gousios and A. Zaidman. A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, pages 368–371, May 2014.

222. G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering*, pages 358–368, May 2015.

223. K. Goševa-Popstojanova, M. Hamill, and R. Perugupalli. Large empirical case study of architecture-based software reliability. In *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)*, pages 43–52, Nov. 2005.

224. E. E. Grant and H. Sackman. An exploratory investigation of programmer performance under on-line and off-line conditions. *IEEE Transactions on Human Factors in Electronics*, 8(1):33–48, Mar. 1967.

225. Graphviz –graph visualization software. website, 2015. http://www.graphviz.org.

226. D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. The misuse of the NASA metrics data program data sets for automated software defect prediction. In *15th Annual Conference on Evaluation & Assessment in Software Engineering 2011 (EASE 2011)*, pages 96–103, Apr. 2011.

227. J. Gray, C. Nyberg, M. Shah, and N. Govindaraju. Sort benchmark. http://sortbenchmark.org, July 2014.

228. W. D. Gray, C. R. Sims, W.-T. Fu, and M. J. Schoelles. The soft constraints hypothesis: A rational analysis approach to resource allocation for interactive behavior. *Psychological Review*, 113(3):461–482, 2006.

229. J. H. Greenberg. Some universals of grammar with particular reference to the order of meaningful elements. In J. H. Greenberg, editor, *Universals of Language*, chapter 5, pages 58–90. MIT Press, 1963.

230. Linux kernel statistics. website, June 2016. https://www.github.com/gregkh/kernel-history.

231. C. Gregg and K. Hazelwood. Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 134–144, Apr. 2011.

232. D. A. Grier. The ENIAC, the verb "to program" and the emergence of digital computers. *IEEE Annals of the History of Computing*, 18(I):51–55, 1996.

233. E. Grochowski and R. E. Fontana, Jr. Future technology challenges for NAND flash and HDD products. Flash Memory Summit 2012, Santa Clara, CA, July 2012.

234. U. Grömping. Relative importance for linear regression in R: The package relaimpo. *Journal of Statistical Software*, 17(1):1–27, Sept. 2006.

235. A. Grübler and N. Nakićenović. Long waves, technology diffusion, and substitution. Technical Report RP-91-17, International Institute for Applied Systems Analysis Laxenburg, Austria, Oct. 1991.

236. W. Gruhl. Lessons learned cost/schedule assessment guide. Slides of talk, July 199x.

237. L. Guerrouj, M. Di Penta, Y.-G. Guéhéneuc, and G. Antoniol. An experimental investigation on the effects of context on source code identifiers splitting and expansion. *Empirical Software Engineering*, 19(6):1706–1753, Dec. 2014.

238. H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and A. D. Satria. What bugs live in the cloud? A study of 3000+ issues in cloud systems. In *Proceedings of the 5th ACM Symposium on Cloud Computing (SOCC'14)*, pages 1–14, Nov. 2014.

239. N. J. Gunther. A simple capacity model of massively parallel transaction systems. In *Proceedings of 19th International CMG Conference*, pages 1035–1044, Dec. 1993.

240. N. J. Gunther. *Analysing Computer System Performance with Perl::PDQ*. Springer-Verlag, 2005.

241. J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wąsowski. Variability-aware performance prediction: A statistical learning approach. In *IEEE/ACM 28th International Conference on Automated Software Engineering (ASE 13)*, pages 301–311, Nov. 2013.

242. D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer. An energy efficiency feature survey of the Intel Haswell processor. In *International Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015*, pages 896–904, May 2015.

243. J. Haidt. *The Righteous Mind*. Vintage books, 2012.

244. T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, Nov. 2012.

245. D. Z. Hambrick, F. L. Oswald, E. M. Altmann, E. J. Meinz, F. Gobet, and G. Campitelli. Deliberate practice: Is that all it takes to become an expert? *Intelligence*, 45(???):34–45, Apr. 2014.

246. J. E. Hannay, D. I. K. Sjøberg, and T. Dybå. A systematic review of theory use in software engineering experiments. *IEEE Transactions on Software Engineering*, 33(2):87–107, Feb. 2007.

247. L. Hatton. Reexamining the fault density-component size connection. *IEEE Software*, 14(2):89–97, Mar. 1997.

248. L. Hatton. How accurately do engineers predict software maintenance tasks? *Computer*, 40(2):64–69, Feb. 2007.

249. M. D. Hauser, S. Carey, and L. B. Hauser. Spontaneous number representation in semi-free-ranging rhesus monkeys. *Proceedings of the Royal Society B*, 267(1445):829–833, Apr. 2000.

250. D. M. Hawkins. *Identification of Outliers*. Springer, 1980.

251. G. Hawkins, S. D. Brown, M. Steyvers, and E.-J. Wagenmakers. Context effects in multi-alternative decision making: Empirical data and a bayesian model. *Cognitive Science*, 36(3):498–516, 2012.

252. G. E. Hawkins, S. D. Brown, M. Steyvers, and E.-J. Wagenmakers. An optimal adjustment procedure to minimize experiment time in decisions with multiple alternatives. *Psychonomic Bulletin & Review*, 19(2):339–348, 2012.

253. B. Hayes. Third base. *American Scientist*, 89(6):490–494, 2001.

254. S. Hazelhurst. Truth in advertising: Reporting performance of computer programs, algorithms and the impact of architecture and systems environment. *South African Computer Journal*, 46:24–37, Dec. 2010.

255. A. Heathcote, S. Brown, and D. J. K. Mewhort. The power law repealed: The case for an exponential law of practice. *Psychonomic Bulletin & Review*, 7(2):185–207, Apr. 2000.

256. D. R. Helsel. *Statistics for Censored Environmental Data using Minitab and R*. John Wiley & Sons, second edition, 2012.

257. A. Henik and J. Tzelgov. Is three greater than five: The relation between physical and semantic size in comparison tasks. *Memory & Cognition*, 10(4):389–395, 1982.

258. J. Henrich, S. J. Heine, and A. Norenzayan. The weirdest people in the world? Technical Report Working Paper No. 139, German Data Forum (RatSWD), Apr. 2010.

259. J. Henrich and R. Mcelreath. Are peasants risk-averse decision makers? *Current Anthropology*, 43(1):172–181, Feb. 2002.

260. E. Herrmann, J. Call, M. Victoria, Hernández-Lloreda, B. Hare, and M. Tomasello. Humans have evolved specialized skills of social cognition: The cultural intelligence hypothesis. *Science*, 317(5843):1360–1366, Sept. 2007.

261. K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *ICSE '13 Proceedings of the 2013 International Conference on Software Engineering*, pages 392–401, May 2013.

262. K. Herzig and A. Zeller. Untangling changes. Submitted to MSR 2013, 2013.

263. T. Hesterberg. What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. Technical report, Google, Nov. 2014.

264. M. Hicks, C. O'Malley, S. Nichols, and B. Anderson. Comparison of 2D and 3D representations for visualising telecommunication usage. *Behaviour & Information technology*, 22(3):185–201, May 2003.

265. E. T. Higgins. Value from regulatory fit. *Current Directions in Psychological Science*, 14(4):209–213, 2005.

266. M. Hilbert and P. López. Supporting online material for: The world's technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, Apr. 2011.

267. T. T. Hills, P. M. Todd, and M. N. Jones. Foraging in semantic fields: How we search through memory. *Topics in Cognitive Science*, 7(3):513–534, July 2015.

268. D. J. Hilton. The social context of reasoning: Conversational inference and rational judgment. *Psychological Bulletin*, 118(2):248–271, 1995.

269. A. Hindle, M. W. Godfrey, and R. C. Holt. Reading beside the lines: Indentation as a proxy for complexity metrics. In *ICPC 2008. The 16th IEEE International Conference on Program Comprehension*, pages 133–142, June 2008.

270. S. C. Hirtle and J. Jonides. Evidence for hierarchies in cognitive maps. *Memory & Cognition*, 13(3):208–217, 1985.

271. M. Hocko and T. Kalibera. Reducing performance non-determinism via cache-aware page allocation strategies. In *WOSP/SIPEW'10 Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, pages 223–234, Jan. 2010.

272. A. Höfer. Exploratory comparison of expert and novice pair programmers. *Computing and Informatics*, 29(1):73–91, 2010.

273. D. D. Hoffman. *Visual Intelligence: How We Create What We See*. W. W. Norton, 2000.

274. R. M. Hogarth and H. J. Einhorn. Order effects in belief updating: The belief-adjustment model. *Cognitive Psychology*, 24:1–55, 1992.

275. R. M. Hogarth, C. R. M. McKenzie, B. J. Gibbs, and M. A. Marquis. Learning from feedback: Exactness and incentives. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 17(4):734–752, 1991.

276. R. Hoosain. Correlation between pronunciation speed and digit span size. *Perception and Motor Skills*, 55:1128–1128, 1982.

277. R. Hoosain and F. Salili. Language differences, working memory, and mathematical ability. In M. M. Grunberg, P. E. Morris, and R. N. Sykes, editors, *Practical aspects of memory: Current research and issues*, volume 2, pages 512–517. John Wiley & Sons, Inc, 1988.

278. M. W. Howard and M. J. Kahana. Context variability and serial position effects in free recall. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 25(4):923–941, 1999.

279. R. Hundt, E. Raman, M. Thuresson, and N. Vachharajani. MAO - an extensible micro-architectural optimizer. In *CGO '11 Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, pages 1–10, Apr. 2011.

280. S. Hunold and A. Carpen-Amarie. MPI benchmarking revisited: Experimental design and reproducibility. Technical Report arXiv:1505.07734v3 [cs.DC], Faculty of Informatics, Vienna University of Technology, Sept. 2015.

281. S. Hunold, A. Carpen-Amarie, and J. L. Träff. Reproducible MPI micro-benchmarking isn't as easy as you think. In *EuroMPI/ASIA'14 Proceedings of the 21st European MPI Users' Group Meeting*, pages 69–76, Sept. 2014.

282. M. J. Hurlstone, G. J. Hitch, and A. D. Baddeley. Memory for serial order across domains: An overview of the literature and directions for future research. *Psychonomic Bulletin & Review*, 140(2):229–373, Mar. 2014.

283. J. Iivonen. Identifying and characterizing highly performing testers –A case study in three software product companies. Thesis (m.s.), Helsinki University of Technology, Department of Computer Science and Engineering, Oct. 2009.

284. I. Imbo and J.-A. LeFevre. Cultural differences in complex addition: Efficient Chinese versus adaptive Belgians and Canadians. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 35(6):1465–1476, Nov. 2009.

285. I. Imbo, A. Vandierendonck, and E. Vergauwe. The role of working memory in carrying and borrowing. *Psychological Research*, 71(4):467–483, July 2007.

286. J. P. A. Ioannidis. Contradicted and initially stronger effects in highly cited clinical research. *JAMA*, 294(2):218–228, July 2005.

287. J. P. A. Ioannidis. Why most published research findings are false. *PLoS Medicine*, 2(8):e124, Aug. 2005.

288. G. Irlam. Unix file size survey - 1993. http://www.base.com/gordoni/ufs93.html, Sept. 1993.

289. A. Israeli and D. G. Feitelson. The Linux kernel as a case study in software evolution. *Journal of Systems and Software*, 83(3):485–501, Mar. 2010.

290. M. Y. Jaber. Learning and forgetting models and their applications. In A. B. Badiru, editor, *Handbook of Industrial and Systems Engineering*, chapter 30. CRC Press –Taylor & Francis Group, Dec. 2005.

291. J. Jacobs and B. Rudis. *Data-Driven Security*. John Wiley & Sons, Inc, 2014.

292. L. R. Jager and J. T. Leek. Empirical estimates suggest most published research is true. *Biostatistics*, 15(1):1–12, 2014.

293. A. R. Jansen. *Encoding and Parsing of Algebraic Expressions by Experienced Users of Mathematics*. PhD thesis, School of Computer Science and Software Engineering, Monash University, Jan. 2002.

294. A. R. Jansen, K. Marriott, and G. W. Yelland. Parsing of algebraic expressions by experienced users of mathematics. *European Journal of Cognitive Psychology*, 19(2):286–320, 2007.

295. The cpushack museum. website, Feb. 2010. http://www.cpushack.com/life-cycle-of-cpu.html.

296. D. D. P. Johnson, N. B. Weidmann, and L.-E. Cederman. Fortune favours the bold: An agent-based model reveals adaptive advantages of overconfidence in war. *PLoS ONE*, 6(6):e20851, Apr. 2011.

297. P. M. Johnson and A. M. Disney. A critical analysis of PSP data quality: Results from a case study. *Empirical Software Engineering*, 4(4):317–349, Dec. 1999.

298. D. M. Jones. The 7±2 urban legend. MISRA C 2002 conference http://www.knosof.co.uk/cbook/misart.pdf, Oct. 2002.

299. D. M. Jones. The new C Standard: An economic and cultural commentary. Knowledge Software, Ltd, 2005.

300. D. M. Jones. Developer beliefs about binary operator precedence. *C Vu*, 18(4):14–21, Aug. 2006.

301. D. M. Jones. Operand names influence operator precedence decisions. *C Vu*, 20(1):5–11, Feb. 2008.

302. D. M. Jones. Developer characterization of data structure fields decisions. *C Vu*, 20(6):14–18, Jan. 2009.

303. D. M. Jones. Effects of risk attitude on recall of assignment statements. *C Vu*, 23(6):19–22, Jan. 2012.

304. D. M. Jones. Code & data for Empirical software engineering using R. http://www.github.com/Derek-Jones/ESEUR, 2017.

305. R. Jongeling, S. Datta, and A. Serebrenik. Choosing your weapons: On sentiment analysis tools for software engineering research. In *31st International Conference on Software Maintenance and Evolution: ICSME 2015*, pages 531–535, Sept. 2015.

306. M. R. Jongerden. *Model-based energy analysis of battery powered systems*. PhD thesis, Centre for Telematics and Information Technology, University of Twente, Dec. 2010.

307. M. Jørgensen. An empirical study of software maintenance tasks. *Software Maintenance: Research and Practice*, 7(1):27–48, Jan. 1995.

308. M. Jørgensen and G. J. Carelius. An empirical study of software project bidding. *IEEE Transactions on Software Engineering*, 30(12):953–969, Dec. 2004.

309. M. Jørgensen, T. Dybå, K. Liestøl, and D. I. K. Sjøberg. Incorrect results in software engineering experiments: How to improve research practices. *Journal of Systems and Software*, 116(???):133–145, June 2016.

310. M. Jørgensen and S. Grimstad. Software development estimation biases: The role of interdependence. *IEEE Transactions on Software Engineering*, 38(3):677–693, May 2012.

311. M. Jørgensen and K. Moløkken. Eliminating over-confidence in software development effort estimates. In F. Bomarius and H. Iida, editors, *Product Focused Software Process Improvement*, volume 3009 of *Lecture Notes in Computer Science*, pages 174–184. Springer Berlin Heidelberg, Apr. 2004.

312. M. Jørgensen and K. Moløkken. Understanding reasons for errors in software effort estimates. *IEEE Transactions on Software Engineering*, 30(12):993–1007, Dec. 2004.

313. M. Jørgensen and D. I. K. Sjøberg. The impact of customer expectation on software development effort estimates. *International Journal of Project Management*, 22(4):317–325, May 2004.

314. M. Jørgensen and D. I. K. Sjøberg. Learning from experience in a software maintenance environment. *Journal of Computer Science*, 1(4):538–542, Apr. 2005.

315. J. W. Kalat. *Biological Psychology*. Wadsworth, seventh edition, 2001.

316. T. Kalibera, L. Bulej, and P. Tůma. Benchmark precision and random initial state. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005)*, pages 853–862. Society for Modeling and Simulation (SCS), July 2005.

317. A. Kaltenbrunner, V. Gómez, A. Moghnieh, R. Meza, J. Blat, and V. López. Homogeneous temporal activity patterns in a large online communication space. Technical Report arXiv:0708.1579 [cs.NI], Departament de les Tecnologies de la Informació i les comunicacions, Aug. 2007.

318. V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K. Sjøberg. A systematic review of effect size in software engineering experiments. *Information and Software Technology*, 49(11-12):1073–1086, Apr. 2007.

319. P. Kampstra and C. Verhoef. Benchmarking the expected loss of a federal IT portfolio. ???, July 2009.

320. D. Karlis and E. Xekalaki. Mixed poisson distributions. *International Statistical Review*, 73(1):35–58, Apr. 2005.

321. D. O. Kennedy and A. B. Scholey. Glucose administration, heart rate and cognitive performance: effects of increasing mental effort. *Psychopharmacology*, 149(1):63–71, May 2000.

322. H. Kim. *Informed Storage Management for Mobile Platforms*. PhD thesis, College of Computing, Georgia Institute of Technology, Dec. 2012.

323. J. King and M. A. Just. Individual differences in syntactic processing: The role of working memory. *Journal of Memory and Language*, 30:580–602, 1991.

324. D. Kirsh and P. Maglio. On distinguishing epistemic from pragmatic action. *Cognitive Science*, 18(4):513–549, Oct. 1994.

325. D. Klahr, W. G. Chase, and E. A. Lovelace. Structure and process in alphabetic retrieval. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 9(3):462–477, 1983.

326. S. B. Klein, L. Cosmides, J. Tooby, and S. Chance. Decisions and the evolution of memory: Multiple systems, multiple functions. *Psychological Review*, 109(2):306–329, 2002.

327. K. E. Knight. Changes in computer performance. *Datamation*, 12(9):40–54, Sept. 1966.

328. K. E. Knight. Evolving computer performance 1963-1967. *Datamation*, 14(1):31–35, Jan. 1968.

329. R. Kohavi and R. Longbotham. Unexpected results in online controlled experiments. *ACM SIGKDD Explorations Newsletter*, 12(2):31–35, Dec. 2010.

330. A. Koriat. How do we know that we know? The accessibility model of the feeling of knowing. *Psychological Review*, 100(4):609–639, 1993.

331. A. G. Koru, K. El Emam, D. Zhang, H. Liu, and D. Mathew. Theory of relative defect proneness: Replicated studies on the functional form of the size-defect relationship. *Empirical Software Engineering*, 13(5):473–498, Oct. 2008.

332. S. M. Kosslyn. *Graph Design for the Eye and Mind*. Oxford University Press, 2006.

333. S. M. Kosslyn and S. P. Shwartz. Empirical constraints on theories of visual imagery. In J. Long and A. D. Baddeley, editors, *Attention and Performance IX*, pages 241–260. Lawrence Erlbaum Associates, 1981.

334. E. Krevat, J. Tucek, and G. R. Ganger. Disks are like snowflakes: No two are alike. In *HotOS'13 Proceedings of the 13th USENIX conference on Hot topics in operating systems*, page ???, May 2013.

335. M. Kubovy and M. van den Berg. The whole is equal to the sum of its parts: A probabilistic model of grouping by proximity and similarity in regular patterns. *Psychological Review*, 115(1):131–154, 2008.

336. M. Kuhrmann, C. Konopka, P. Nellemann, P. Diebold, and J. Münch. Software process improvement: Where is the evidence? In *Proceedings of the 2015 International Conference on Software and System Process*, pages 107–116, Aug. 2015.

337. P. Küngas, S. Vakulenko, M. Dumas, C. Parra, and F. Casati. Reverse-engineering conference rankings: What does it take to make a reputable conference? *Scientometrics*, 96(2):651–665, Aug. 2013.

338. G. Kunst. Language popularity. http://langpop.corger.nl/results, 2013.

339. R. Kurzban, A. Duckworth, J. W. Kable, and J. Myers. An opportunity cost model of subjective effort and task performance. *Behavioral and Brain Sciences*, 36(6):661–679, Dec. 2013.

340. W. Labov. The boundaries of words and their meaning. In C.-J. N. Bailey and R. W. Shuy, editors, *New ways of analyzing variation of English*, pages 340–373. Georgetown Press, 1973.

341. G. Lakoff and M. Johnson. *Metaphors We Live By*. The University of Chicago Press, 1980.

342. T. K. Landauer. How much do people remember? Some estimates of the quantity of learned information in long-term memory. *Cognitive Science*, 10:477–493, 1986.

343. D. Landy, D. Brookes, and R. Smout. Abstract numeric relations and the visual structure of algebra. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 40(5):1404–1418, Sept. 2014.

344. D. Landy, A. Charlesworth, and E. Ottmar. Categories of large numbers in line estimation. *Cognitive Science*, ???(???):1–28, Feb. 2016.

345. D. Landy and R. L. Goldstone. Proximity and precedence in arithmetic. *The Quarterly Journal of Experimental Psychology*, 63(10):1953–1968, Oct. 2010.

346. E. J. Langer. The illusion of control. *Journal of Personality and Social Psychology*, 32(2):311–328, 1975.

347. J. Larres. Performance variance evaluation on Mozilla Firefox. Thesis (m.s.), Victoria University of Wellington, May 2012.

348. C. Lebiere. *The Dynamics of Cognition: An ACT-R Model of Cognitive Arithmetic*. PhD thesis, Carnegie Mellon University, Nov. 1998.

349. B. C. Lee and D. M. Brooks. Regression modeling strategies for microarchitecture performance and power prediction. Technical Report TR-08-06, Division of Engineering and Applied Sciences, Harvard University, Mar. 2006.

350. D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu. Adaptive-latency DRAM: Optimizing DRAM timing for the common-case. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 489–501, Feb. 2015.

351. G. Leech, P. Rayson, and A. Wilson. *Word Frequencies in Written and Spoken English*. Pearson Education, 2001.

352. G. E. Legge, T. A. Hooven, T. S. Klitz, J. S. Mansfield, and B. S. Tjan. Mr. Chips 2002: New insights from an ideal-observer model of reading. *Vision Research*, 42(18):2219–2234, Aug. 2002.

353. D. R. Lehman, R. O. Lempert, and R. E. Nisbett. The effects of graduate training on reasoning. *American Psychologist*, 43(6):431–442, 1988.

354. P. Lemaire and M. Fayol. When plausibility judgments supersede fact retrieval: The example of the odd-even effect on product verification. *Memory & Cognition*, 23(1):34–48, Feb. 1995.

355. P. Lennie. The cost of cortical computation. *Current Biology*, 13(6):493–497, Mar. 2003.

356. Y. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. Moreira, and M. Gupta. Filtering failure logs for a BlueGene/L prototype. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 476–485, June 2005.

357. S. Lichtenstein and B. Fishhoff. Do those who know more also know more about how much they know? *Organizational Behavior and Human Performance*, 20:159–183, 1977.

358. G. A. Liebchen and M. Shepperd. Data sets and data quality in software engineering. In *PROMISE'08 Proceedings of the 4th international workshop on Predictor models in software engineering*, pages 39–44, May 2008.

359. R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, 36(4):119–150, Dec. 2004.

360. P. Louridas, D. Spinellis, and V. Vlachos. Power laws in software. *ACM Transaction on Software Engineering and Methodology*, 18(1):1–26, Sept. 2008.

361. L. Lu, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and S. Lu. A study of Linux file system evolution. In *11th USENIX Conference on File and Storage Technologies (FAST '13)*, pages 31–44, Feb. 2013.

362. K. M. Lui and K. C. C. Chan. Pair programming productivity: Novice–novice vs. expert–expert. *International Journal of Human-Computer Studies*, 64(9):915–925, Sept. 2006.

363. P. Lukowicz, E. A. Heinz, L. Prechelt, and W. F. Tichy. Experimental evaluation in computer science: A quantitative study. Technical Report 17/94, University of Karlsruhe, Germany, Aug. 1994.

364. A. K. Luria. Towards the problem of the historical nature of psychological processes. *International Journal of Psychology*, 6(4):259–272, 1971.

365. A. R. Luria. *The Mind of a Mnemonist*. Penguin Education, 1975.

366. B. Luthiger and C. Jungwirth. Pervasive fun. *First Monday*, 12(1), Jan. 2007.

367. J. N. MacGregor. Short-term memory capacity: Limitation or optimization? *Psychological Review*, 94(1):107–108, 1987.

368. A. Machiry, R. Tahiliani, and M. Naik. Dynodroid: An input generation system for Android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering ESEC/FSE '13*, pages 224–234, Aug. 2013.

369. W. T. Maddox and C. J. Bohil. Costs and benefits in perceptual categorization. *Memory & Cognition*, 28:597–615, 2000.

370. V. N. Makarow. SPEC benchmark page. http://vmakarov.fedorapeople.org/spec/index.html, July 2014.

371. M. Mangel and F. J. Samaniego. Abraham Wald's work on aircraft survivability. *Journal of the American Statistical Association*, 79(386):259–267, June 1984.

372. J. N. Marewski and L. J. Schooler. Cognitive niches: An ecological model of strategy selection. *Psychological Review*, 118(3):292–437, 2011.

373. E. Matias, I. S. MacKenzie, and W. Buxton. One-handed touch-typing on a QWERTY keyboard. *Human-Computer Interaction*, 11:1–27, 1996.

374. A. Mazouz. *An Empirical Study of Program Performance of OpenMP Applications on Multicore Platforms*. PhD thesis, Université de Versailles-Saint Quentin en Yvelines, Dec. 2013.

375. J. C. McCallum. Historical cost of computer memory and storage. http://www.jcmit.com, July 2016.

376. S. McCloud. *Understanding Comics*. HarperPerennial, 1993.

377. M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, 33(4):125–180, June 2001.

378. R. R. McCrae and J. Paul T. Costa. Reinterpreting the Myers-Briggs type indicator from the perspective of the five-factor model of personality. *Journal of Personality*, 57(1):17–40, Mar. 1989.

379. B. D. McCullough and D. A. Heiser. On the accuracy of statistical procedures in Microsoft Excel 2007. *Computational Statistics and Data Analysis*, 52:4570–4578, 2008.

380. D. McFadden. Rationality for economists? *Journal of Risk and Uncertainty*, 19:73–105, 1999.

381. K. B. McKeithen, J. S. Reitman, H. H. Ruster, and S. C. Hirtle. Knowledge organization and skill differences in computer programmers. *Cognitive Psychology*, 13:307–325, 1981.

382. T. P. McNamara, J. K. Hardy, and S. C. Hirtle. Subjective hierarchies in spatial memory. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 15(2):211–227, 1989.

383. F. Mechner. Probability relations within response sequences under ratio reinforcement. *Journal of the Experimental Analysis of Behavior*, 1(2):109–121, Apr. 1958.

384. L. K. Melhus and R. E. Jensen. Measurement bias from address aliasing. In *iWAPT 2016 Eleventh International Workshop on Automatic Performance Tuning*, page ???, May 2016.

385. H. Mercier and D. Sperber. Why do humans reason? Arguments for an argumentative theory. *Behavioral and Brain Sciences*, 34(2):57–111, Apr. 2011.

386. R. C. Merkle. Energy limits to the computational power of the human brain. *Foresight Update*, 6, Aug. 1989.

387. T. Micceri. The unicorn, the normal curve, and other improbable creatures. *Psychological Bulletin*, 105(1):156–166, Apr. 1989.

388. J.-B. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, The Google Books Team, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. A. Nowak, and E. L. Aiden. Quantitative analysis of culture using millions of digitized books. *Science*, 14(6014):176–182, Jan. 2011.

389. S. Milgram. *Obedience to Authority*. McGraw–Hill, 1974.

390. G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, 1956.

391. G. A. Miller and S. Isard. Free recall of self-embedded English sentences. *Information and Control*, 7:292–303, 1964.

392. S. J. Miller and M. J. Nigrini. Order statistics and Benford's law. *International Journal of Mathematics and Mathematical Sciences*, 2008, 2008.

393. W. R. Miller and M. Sanchez-Craig. How to have a high success rate in treatment: advice for evaluators of alcoholism programs. *Addiction*, 91(6):779–785, Apr. 1996.

394. M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(2):226–251, 2003.

395. M. Mitzenmacher. Dynamic models for file sizes and double Pareto distributions. *Internet Mathematics*, 1(3):305–333, Apr. 2004.

396. A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2), Apr.-June 2000.

397. T. Moher and G. M. Schneider. Methods for improving controlled experimentation in software engineering. In *Proceedings of the 5th International Conference on Software Engineering*, pages 224–233. IEEE Computer Society, Mar. 1981.

398. K. Moløkken-Østvold and K. M. Furulund. The relationship between customer collaboration and software project overruns. In *2007 Agile Conference (AGILE)*, pages 72–83, Aug. 2007.
399. J. Montgomery Phister. *Data Processing Technology and Economics*. Santa Monica Publishing Company and Digital Press, second edition, 1979.
400. S. T. Mueller and A. Krawitz. Reconsidering the two-second decay hypothesis in verbal working memory. *Journal of Mathematical Psychology*, 53(1):14–25, Feb. 2009.
401. S. T. Mueller and C. T. Weidemann. Alphabetic letter identification: Effects of perceivability, similarity, and bias. *Acta Psychologica*, 139(1):19–37, Jan. 2012.
402. M. M. Müller and A. Höfer. The effect of experience on the test-driven development process. *Empirical Software Engineering*, 12(6):593–615, 2007.
403. P. Murrell. *R Graphics*. Chapman & Hall/CRC, 1st edition, 2006.
404. T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney. We have it easy, but do we have it right? In *IPDPS 2008 International Symposium on Parallel and Distributed Processing*, pages 1–7, Apr. 2008.
405. M. Nagappan, T. Zimmermann, and C. Bird. Diversity in software engineering research. In *ESEC/FSE 2013 Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 466–476, Aug. 2013.
406. N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. Change bursts as defect predictors. In *IEEE 21st International Symposium on Software Reliability Engineering (ISSRE)*, pages 309–318, Nov. 2010.
407. P. Naur and B. Randell. Software engineering report on a conference sponsored by the NATO science committee. Technical report, NATO, Jan. 1969.
408. R. E. NeSmith II. A study of software maintenance costs of Air Force large scale computer systems. Thesis (m.s.), School of Systems and Logistics, Air Force Institute of Technology, Air University, Sept. 1986.
409. A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1991.
410. A. Newell and P. S. Rosenbloom. Mechanisms of skill acquisition and the power law of practice. Technical report, Carnegie Mellon University, Aug. 1982.
411. E. B. Nightingale, J. R. Douceur, and V. Orgovan. Cycles, cells and platters: An empirical analysis of hardware failures on a million consumer PCs. In *Proceedings of the sixth conference on Computer systems, EuroSys'11*, pages 343–356, Apr. 2011.
412. M. J. Nigrini and S. J. Miller. Data diagnostics using second order tests of Benford's law. *Auditing: A Journal of Practice and Theory*, 28(2):305–324, June 2009.
413. D. E. Nikonov and I. A. Young. Overview of beyond-CMOS devices and a uniform methodology for their benchmarking. Technical report, Intel Corp, Feb. 2013.
414. R. E. Nisbett, D. H. Krantz, C. Jepson, and Z. Kunda. The use of statistical heuristics in everyday inductive reasoning. *Psychological Review*, 90(4):339–363, 1983.
415. NIST??? National vulnerability database. http://nvd.nist.gov, Dec. 2014.
416. G. P.-C. no and D. M. German. Software patents: A replication study. In *OpenSym '15 Proceedings of the 11th International Symposium on Open Collaboration*, pages 5:1–5:4, Aug. 2015.
417. S. Nørby. Why forget? On the adaptive value of memory loss. *Perspectives on Psychological Science*, 10(5):551–578, 2015.
418. W. D. Nordhaus. The progress of computing. Technical Report Cowles Foundation Discussion Paper No. 1324, Yale University, Sept. 2001.
419. R. E. Núñez. No innate number line in the human brain. *Journal of Cross-Cultural Psychology*, 42(4):651–668, 2011.
420. NVIDIA. *CUDA CUBLAS Library*. NVIDIA Corporation, CA, USA, 3.1 edition, Aug. 2010.
421. M. Oaksford and N. Chater. A rational analysis of the selection task as optimal data selection. *Psychological Review*, 101(4):608–631, 1994.
422. K. Oberauer, H.-M. Süß, O. Wilhelm, and W. W. Wittmann. The multiple faces of working memory: Storage, processing, supervision, and coordination. *Intelligence*, 31:167–193, 2003.
423. O. O. Odeh, A. M. Featherstone, and J. S. Bergtold. Reliability of statistical software. *American Journal of Agricultural Economics*, 92(5):1472–1489, Sept. 2010.
424. P. Oladimeji. Devices, errors and improving interaction design - A case study using an infusion pump. Thesis (m.res.), Department of Computer Science, Swansea University, Oct. 2008.
425. Open Science Collaboration. Estimating the reproducibility of psychological science. *Science*, 349(6251):aac4716, Aug. 2015.
426. OpenRefine. website, Oct. 2014. http://openrefine.org.
427. OpenSignal. Android fragmentation visualized (august 2015). Technical Report ???, OpenSignal, Aug. 2015.
428. N. Osaka. Eye fixation and saccade during kana and kanji text reading: Comparison of English and Japanese text processing. *Bulletin of the Psychonomic Society*, 27(6):548–550, 1989.
429. G. L. Ourada. Software cost estimating models: A calibration, validation, and comparison. Thesis (m.s.), Air Force Institute of Technology, Air University, USA, Dec. 1991.
430. S. Owsowitz and A. Sweetland. Factors affecting coding errors. Technical Report RM-4346-PR, The RAND Corporation, Apr. 1965.
431. P. Padfield. *Battleship*. Thistle Publishing, 2015.
432. J. Pallister, S. Hollis, and J. Bennett. Identifying compiler options to minimise energy consumption for embedded platforms. Technical report, Bristol University, Aug. 2013.
433. S. E. Palmer. *Vision Science: Photons to Phenomenology*. The MIT Press, 1999.
434. J. M. Parkman. Temporal aspects of simple multiplication and comparison. *Journal of Experimental Psychology*, 95(2):437–444, 1972.
435. J. M. Parkman and G. J. Groen. Temporal aspects of simple addition and comparison. *Journal of Experimental Psychology*, 89(2):335–342, 1971.
436. H. E. Pashler. *The Psychology of Attention*. The MIT Press, 1999.
437. A. Patel. Auditors' belief revision: Recency effects of contrary and supporting audit evidence and source reliability. Technical Report 2001-1, University of South Pacific, Dept. of AFM/SSE, June 2001.
438. F. M. Paulus, L. Rademacher, T. A. J. Schäfer, L. Müller-Pinzler, and S. Krach. Journal impact factor shapes scientists' reward signal in the prospect of publication. *PLoS ONE*, 10(11):e0142537, Nov. 2015.
439. A. Pavese and C. Umiltà. Symbolic distance between numerosity and identity moulates stroop-like interference. *Journal of Experimental Psychology: Human Perception and Performance*, 24(5):1535–1545, 1998.
440. J. W. Payne, J. R. Bettman, and E. J. Bettman. *The Adaptive Decision Maker*. Cambridge University Press, 1993.
441. G. Paz-y-Miño C, A. B. Bond, A. C. Kamil, and R. P. Balda. Pinyon jays use transitive inference to predict social dominance. *Nature*, 430:778–781, Aug. 2004.
442. J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.
443. R. K. Pearson. The problem of disguised missing data. *ACM SIGKDD Explorations Newsletter*, 8(1):83–92, June 2006.
444. E. Pek, V. Klebanov, and R. Lämmel. Re-engineering software corpora for simplified adoption. ???, July 201?
445. D. G. Pelli, C. W. Burns, B. Farell, and D. C. Moore. Identifying letters. *Vision Research*, 46(28):4646–4674, 2006.
446. E. Peltonen, E. Lagerspetz, and P. N. S. Tarkoma. Energy modeling of system settings: A crowdsourced approach. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 37–45, Mar. 2015.
447. N. Pennington. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19:295–341, 1987.
448. D. E. Perry and W. M. Evangelist. An empirical study of software interface faults – An update. In *Proceedings of the Twentieth Annual Hawaii International Conference on Systems Sciences, Vol II*, pages 113–126, Jan. 1987.
449. R. Perugupalli. Empirical assessment of architecture-based reliability of open-source software. Thesis (m.s.), Department of Computer Science and Electrical Engineering, West Virginia University, May 2004.
450. A. Pewsey, M. Neuhäuser, and G. D. Ruxton. *Circular Statistics in R*. Oxford University Press, 2013.
451. C. Phillips. *Order and Structure*. PhD thesis, M.I.T., Aug. 1996.
452. R. Pieters and L. Warlop. Visual attention during brand choice: The impact of time pressure and task motivation. *International Journal of Research in Marketing*, 16:1–16, 1999.

453. A. M. Pires and C. ao Amado. Interval estimators for a binomial proportion: Comparison of twenty methods. *REVSTAT –Statistical Journal*, 6(2):165–197, June 2008.

454. D. J. Pittenger. Measuring the MBTI... And coming up short. *Journal of Career Planning and Employment*, 54(1):48–52, Nov. 1993.

455. A. Pollatsek, E. D. Reichle, and K. Rayner. Tests of the E-Z reader model: Exploring the interface between cognition and eye-movement control. *Cognitive Psychology*, 52(1):1–56, Feb. 2006.

456. A. Porter, H. Siy, A. Mockus, and L. Votta. Understanding the sources of variation in software inspections. *ACM Transactions on Software Engineering Methodology*, 7(1):41–79, Jan. 1998.

457. A. Potanin, M. Damitio, and J. Noble. Are your incoming aliases really necessary? Counting the cost of object ownership. In *ICSE '13 Proceedings of the 2013 International Conference on Software*, pages 742–751, May 2013.

458. E. M. Pothos and N. Chater. Rational categories. In *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society*, pages 848–853, 1998.

459. J. Potts, J. Hartley, L. Montgomery, C. Neylon, and E. Rennie. A journal is a club: A new economic model for scholarly publishing. Technical Report SSRN Working Paper n. 2763975, Apr. 2016.

460. A. L. Powell. *Right on Time: Measuring, Modelling and Managing Time-Constrained Software Development*. PhD thesis, Department of Computer Science, University of York, Aug. 2001.

461. L. Prechelt. The 28:1 Grant/Sackman legend is misleading, or: How large is interpersonal variation really? Technical Report iratr-1999-18, Universität Karlsruhe, 1999.

462. L. Prechelt. Plat_Forms 2007: The web development platform comparison –evaluation and results. Technical Report B-07-10, Institut für Informatik, Freie Universität Berlin, June 2007.

463. L. Prechelt, F. Zieris, and H. Schmeisky. Difficulty factors of obtaining access for empirical studies in industry. In *CESI '15 Proceedings of the Third International Workshop on Conducting Empirical Studies in Industry*, pages 19–25, May 2015.

464. L. S. Premo and S. L. Kuhn. Modeling effects of local population extinctions on cultural change and diversity in the paleolithic. *PLoS One*, 5(12):e15582, Dec. 2010.

465. C. C. Presson and D. R. Montello. Updating after rotational and translational body movements: coordinate structure of perspective space. *Perception*, 23:1447–1455, 1994.

466. L. H. Putnam and W. Myers. *Measures for Excellence: Reliable software on time, within budget*. Prentice-Hall, Inc., 1992.

467. Z. Pylyshyn. Is vision continuous with cognition? The case for cognitive impenetrability of visual perception. *Behavioral and Brain Sciences*, 22(3):341–423, 1999.

468. R. Queiroz, L. Passos, M. T. Valente, C. Hunsen, S. Apel, and K. Czarnecki. The shape of feature code: an analysis of twenty C-preprocessor-based systems. *Journal on Software and Systems Modeling*, ???(???):1–20, July 2015.

469. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. ISBN 3-900051-07-0.

470. R Core Team. R language definition. Technical Report 3.3.1, R Foundation for Statistical Computing, June 2016.

471. F. Rahman, C. Bird, and P. Devanbu. Clones: What is that smell? In *MSR'10 Proceedings of the 7th International Workshop on Mining Software Repositories*, pages 72–81, May 2010.

472. K. Rayner. Eye movements and attention in reading, scene perception, and visual search. *The Quarterly Journal of Experimental Psychology*, 62(8):1457–1506, 2009.

473. K. Rayner. Eye movements in reading: Models and data. *Journal of Eye Movement Research*, 2(5):1–10, Apr. 2009.

474. N. M. Razali and Y. B. Wah. Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. *Journal of Statistical Modeling and Analytics*, 2(1):21–33, 2011.

475. J. Reason. *Human Error*. Cambridge University Press, 1990.

476. A. S. Reber and S. M. Kassin. On the relationship between implicit and explicit modes in the learning of a complex rule structure. *Journal of Experimental Psychology: Human Learning and Memory*, 6(5):492–502, 1980.

477. B. Regnell, M. Höst, J. N. och Dag, P. Beremark, and T. Hjelm. An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software. *Requirements Engineering*, 6(1):51–62, Apr. 2001.

478. E. D. Reichle, T. Warren, and K. McConnell. Using E-Z reader to model the effects of higher-level language processing on eye movements during reading. *Psychonomic Bulletin & Review*, 16(1):1–21, Feb. 2009.

479. G. Remillard. Implicit learning of second-, third-, and fourth-order adjacent and nonadjacent sequential dependencies. *The Quarterly Journal of Experimental Psychology*, 61(3):400–424, Apr. 2008.

480. R. W. Remington, H. W. H. Yuen, and H. Pashler. With practice, keyboard shortcuts become faster than menu selection: A crossover interaction. *Journal of Experimental Psychology: Applied*, 22(1):95–106, 2016.

481. Research Councils UK. RCUK policy on open access and supporting guidance. Technical Report ???, RCUK, Apr. 2013.

482. 7digital, ltd??? website, July 2012. http://www.7digital.com.

483. M. J. Roberts, D. J. Gilmore, and D. J. Wood. Individual differences and strategy selection in reasoning. *British Journal of Psychology*, 88:473–492, 1997.

484. S. Roberts and J. Winters. Linguistic diversity and traffic accidents: Lessons from statistical studies of cultural traits. *PLoS ONE*, 8(8):e70902, Aug. 2013.

485. D. E. Robinson. Fashions in shaving and trimming of the beard: The men of the Illustrated London News, 1842-1972. *American Journal of Sociology*, 81(5):1133–1141, Mar. 1976.

486. G. Robles, L. A. Reina, A. Serebrenik, B. Vasilescu, and J. M. González-Barahona. FLOSS 2013: A survey dataset about free software contributors: Challenges for curating, sharing, and combining. In *MSR'14*, pages 396–399, May 2014.

487. R. D. Rogers and S. Monsell. Costs of a predictable switch between simple cognitive tasks. *Journal of Experimental Psychology: General*, 124(2):207–231, 1995.

488. E. Rosch, C. B. Mervis, W. D. Gray, D. M. Johnson, and P. Boyes-Braem. Basic objects in natural categories. *Cognitive Psychology*, 8(3):382–439, July 1976.

489. B. H. Ross and G. L. Murphy. Food for thought: Cross-classification and category organization in a complex real-world domain. *Cognitive Psychology*, 38:495–552, 1999.

490. L. Ross, M. R. Lepper, and M. Hubbard. Perseverance in self-perception and social perception: Biased attributional processes in the debeliefing paradigm. *Journal of Personality and Social Psychology*, 32(5):880–892, 1975.

491. B. F. Roukema. A first-digit anomaly in the 2009 Iranian presidential election. Technical report, ???, June 2013.

492. M. M. Roy, N. J. S. Christenfeld, and C. R. M. McKenzie. Underestimating the duration of future events: Memory incorrectly used or memory bias? *Psychological Bulletin*, 131(5):738–756, 2005.

493. P. Royston, D. G. Altman, and W. Sauerbrei. Dichotomizing continuous predictors in multiple regression: a bad idea. *Statistics in Medicine*, 25(1):127–141, Jan. 2006.

494. D. C. Rubin and A. E. Wenzel. One hundred years of forgetting: A quantitative description of retention. *Psychological Review*, 103(4):734–760, 1996.

495. R. Sabherwal, A. Jeyaraj, and C. Chowa. Information systems success: Individual and organizational determinants. *Management Science*, 52(12):1849–1864, Dec. 2006.

496. R. Saborido, V. Arnaoudova, G. Beltrame, F. Khomh, and G. Antoniol. On the impact of sampling frequency on software energy measurements. *Peerj PrePrints*, 3:e1219, July 2015.

497. A. Sampson, W. Dietl, E. Fortuna, and D. Gnanapragasam. EnerJ: Approximate data types for safe and general low-power computation. In *PLDI'11 Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, pages 164–174, June 2011.

498. D. Sarkar. *Lattice Multivariate Data Visualization with R*. Springer Science+Business Media, 2008.

499. S. R. Schach, B. Jin, L. Yu, G. Z. Heller, and J. Offutt. Determining the distribution of maintenance categories: Survey versus measurement. *Empirical Software Engineering*, 8(4):351–363, Dec. 2003.

500. J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. In *Proceedings of the VLDB Endowment*, pages 460–471, Sept. 2010.

501. K. W. Schaie. *Developmental Influences on Adult Intelligence: The Seattle Longitudinal Study*. Oxford University Press, second edition, 2013.

502. E. Schneider, M. Maruyama, S. Dehaene, and M. Sigman. Eye gaze reveals a fast, parallel extraction of the syntax of arithmetic formulas. *Cognition*, 125(3):475–490, Dec. 2012.
503. A. Scholey and L. Owen. Effects of chocolate on cognitive function and mood: a systematic review. *Nutrition Reviews*, 71(10):665–681, Apr. 2013.
504. R. Schöne, D. Hackenberg, and D. Molka. Memory performance at reduced CPU clock speeds: An analysis of current x86_64 processor. In *HotPower'12 Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, page ???, Oct. 2012.
505. E. R. Schotter, B. Angele, and K. Rayner. Parafoveal processing in reading. *Attention, Perception & Psychophysics*, 74(1):5–35, Jan. 2012.
506. J.-P. Schraepler and G. G. Wagner. Identification of faked interviews in surveys by means of Benford's law?: An analysis by means of genuine fakes in the raw data of SOEP. Technical report, Technische Universiät Berlin, Aug. 2004.
507. M.-A. Schulz, B. Schmalbach, P. Brugger, and K. Witt. Analysing humanly generated random number sequences: A pattern-based approach. *PLoS ONE*, 7(7):e41531, July 2012.
508. P. D. Scott and M. Fasli. Benford's law: An empirical investigation and a novel explanation. Technical Report CSM Technical Report 349, Department of Computer Science, University of Essex, Aug. 2001.
509. S. Scribner. Modes of thinking and ways of speaking: culture and logic reconsidered. In P. N. Johnson-Laird and P. C. Wason, editors, *Thinking: Readings in Cognitive Science*, chapter 29, pages 483–500. Cambridge University Press, 1977.
510. The world's largest hedge fund is a fraud. SEC MADOFF EXHIBITS-04451, Nov. 2005. November 7, 2005 Submission to the SEC, Madoff Investment Securities, LLC.
511. P. Sehgal, V. Tarasov, and E. Zadok. Evaluating performance and energy in file system server workloads. In *FAST'10 Proceedings of the 8th USENIX conference on File and storage technologies*, page ???, Feb. 2010.
512. B. R. Shear and B. D. Zumbo. False positives in multiple regression: Unanticipated consequences of measurement error in the predictor variables. *Educational and Psychological Measurement*, 73(5):733–756, Oct. 2013.
513. B. A. Sheil. The psychological study of programming. *ACM Computing Surveys*, 13(1):101–120, Mar. 1981.
514. A. Shenhav, S. Musslick, F. Lieder, W. Kool, T. L. Griffiths, J. D. Cohen, and M. M. Botvinick. Toward a rational and mechanistic account of mental effort. *Annual Review of Neuroscience*, 40(???):???, Apr. 2017.
515. R. N. Shepard, C. I. Hovland, and H. M. Jenkins. Learning and memorization of classifications. *Psychological Monographs: General and Applied*, 75(15):1–39, 1961.
516. R. N. Shepard and J. Metzler. Mental rotation of three-dimensional objects. *Science*, 171:701–703, Feb. 1971.
517. E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K. ichi Matsumoto. Predicting re-opened bugs: A case study on the Eclipse project. In *17th Working Conference on Reverse Engineering WCRE*, pages 249–258, Oct. 2010.
518. T. C. Shrum. Calibration and validation of the checkpoint model to the air force electronic systems center software database. Thesis (m.s.), Graduate School of Logistics and Acquisition Management or the Air Force Institute of Technology, Air University, Sept. 1997.
519. A. Shtub, N. Levin, and S. Globerson. Learning and forgetting industrial skills: An experimental model. *The International Journal of Human Factors in Manufacturing*, 3(3):293–305, July 1993.
520. K. M. Simmons and D. Sutter. False alarms, tornado warnings, and tornado casualties. *Weather, Climate, and Society*, 1(1):38–53, Oct. 2009.
521. H. A. Simon. *Models of Bounded Rationality: Behavioral Economics and Business Organization*. The MIT Press, 1982.
522. H. A. Simon. Making management decisions: the role of intuition and emotion. *The Academy of Management Executive (1987-1989)*, 1(1):57–64, Feb. 1987.
523. I. Simonson. Choice based on reasons: The case of attraction and compromise effects. *Journal of Consumer Research*, 16:158–173, Sept. 1989.
524. I. C. Simpson, P. Mousikou, J. M. Montoya, and S. Defior. A letter visual-similarity matrix for Latin-based alphabets. *Behavior and Research Methods*, 45(2):431–439, June 2013.
525. D. I. K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanović, E. F. Koren, and M. Vokác. Conducting realistic experiments in software engineering. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE'02)*, pages 17–26, Oct. 2002.
526. D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanović, N.-K. Liborg, and A. C. Rekdal. A survey of controlled experiments in software engineering. Technical Report 2004-4, SIMULA Research Laboratory, 2004.
527. G. Slade. *Made to Break*. Harvard University Press, 2007.
528. S. Sloman, A. K. Barbey, and J. M. Hotaling. A causal model theory of the meaning of cause, enable, and prevent. *Cognitive Science*, 33(1):21–50, Jan. 2009.
529. S. A. Sloman. The empirical case for two systems of reasoning. *Psychological Bulletin*, 119(1):3–22, 1996.
530. S. A. Sloman. Categorical inference is not a tree: The myth of inheritance hierarchies. *Cognitive Psychology*, 35(1):1–33, Feb. 1998.
531. S. A. Sloman and D. Lagnado. Causality in thought. *Annual Review of Psychology*, 66:223–247, 2015.
532. S. A. Sloman and D. A. Lagnado. Do we "do"? *Cognitive Science*, 29(1):5–39, Jan.-Feb. 2005.
533. R. W. Soukoreff. *Quantifying Text Entry Performance*. PhD thesis, York University, Toronto, Canada, Apr. 2010.
534. SPEC. Standard performance evaluation corporation. http://spec.org, July 2014.
535. SPEC. SPEC power_ssj 2008. http://spec.org/power_ssj2008, June 2016.
536. I. Spence and S. Lewandowsky. Displaying proportions and percentages. *Applied Cognitive Psychology*, 5(1):61–77, Apr. 1991.
537. D. Sperber and D. Wilson. *Relevance: Communication and Cognition*. Blackwell Publishers, second edition, 1995.
538. L. R. Squire and A. J. O. Dede. Conscious and unconscious memory systems. *Perspectives in Biology*, 7(3):a021667, Mar. 2015.
539. P. Stanley-Marbell, V. Estellers, and M. Rinard. Crayon: Saving power through shape and color approximation on next-generation displays. In *Proceedings of the Eleventh European Conference on Computer Systems, EuroSys '16*, page ???, Apr. 2016.
540. K. E. Stanovich. *Who Is Rational? Studies of Individual Differences in Reasoning*. Lawrence Erlbaum Associates, 1999.
541. K. E. Stanovich and R. F. West. Individual differences in reasoning: Implications for the rationality debate? *Behavioral and Brain Sciences*, 23(5):645–726, Oct. 2000.
542. M. Staples, R. Kolanski, G. Klein, C. Lewis, J. Andronick, T. Murray, R. Jeffery, and L. Bass. Formal specifications better than function points for code sizing. In *International Conference on Software Engineering, ICSE 2013*, pages 1257–1260, May 2013.
543. J. Starek. A large-scale analysis of Java API usage. Thesis (m.s.), Institut für Informatik, Universität Koblenz-Landau, Mar. 2010.
544. M. Steele and J. Chaseling. Powers of discrete goodness-of-fit test statistics for a uniform null against a selection of alternative distributions. *Communications in Statistics –Simulation and Computation*, 35(4):1067–1075, Apr. 2006.
545. R. G. Steen, A. Casadevall, and F. C. Fang. Why has the number of scientific retractions increased? *PLOS ONE*, 8(7), Apr. 2013.
546. K. Stenning, , and M. van Lambalgen. Semantics as a foundation for psychology: A case study of Wason's selection task. *Journal of Logic, Language and Information*, 10(3):273–317, June 2001.
547. K. Stenning, , and M. van Lambalgen. *Human Reasoning and Cognitive Science*. MIT Press, 2008.
548. K. Stenning and M. van Lambalgen. A little logic goes a long way: basing experiment on semantic theory in the cognitive science of conditional reasoning. *Cognitive Science*, 28(4):481–530, July-Aug. 2004.
549. M. A. Stephens. EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, Sept. 1974.
550. R. J. Sternberg and E. M. Weil. An aptitude-strategy interaction in linear syllogistic reasoning. Technical Report 15, Department of Psychology, Yale University, Apr. 1979.
551. S. Sternberg. Memory-scanning: Mental processes revealed by reaction-time experiments. *American Scientist*, 57(4):421–457, 1969.
552. A. Stevens and P. Coupe. Distortions in judged spatial relations. *Cognitive Psychology*, 10(4):422–437, Oct. 1978.

553. V. Stodden, P. Guo, and Z. Ma. Toward reproducible computational research: An empirical analysis of data and code policy adoption by journals. *PLoS ONE*, 8(6):e13636, June 2013.

554. G. P. Stone, D. B. Levin, H. Hwang, M. Kim, and C. Mckay. *JANET SKOLD and DAVID DOSSANTOS, on behalf of themselves and all others similarly situated, v. INTEL CORPORATION, HEWLETT PACKARD COMPANY and DOES 1-50, Case No. 1-05-CV-039231, Filing #G-64475*. Superior court of the state of California for the county of Santa Clara, 2014.

555. S. Strand, I. J. Deary, and P. Smith. Sex differences in cognitive abilities test scores: A UK national picture. *British Journal of Educational Psychology*, 76(3):463–480, Apr. 2006.

556. K. Suzuki and S. Swanson. A survey of trends in non-volatile memory technologies: 2000–2014. In *IEEE International Memory Workshop (IMW)*, pages 1–4, May 2015.

557. T. N. Suzuki, D. Wheatcroft, and M. Griesser. Experimental evidence for compositional syntax in bird calls. *Nature Communications*, 7(10986), Mar. 2016.

558. R. A. Syed, B. Robinson, and L. Williams. Does hardware configuration and processor load impact software fault observability? In *Third International Conference on Software Testing, Verification and Validation (ICST)*, pages 285–294, Apr. 2010.

559. I. H. Tabernero. *A statistical examination of the properties and evolution of libre software*. PhD thesis, Universidad Rey Juan Carlos, Oct. 2008.

560. N. Taerat, N. Naksinehaboon, C. Chandler, J. Elliot, C. B. Leangsuksun, G. Ostrouchov, S. L. Scott, and C. Englemann. Blue Gene/L log analysis and time to interrupt estimation. In *ARES '09. International Conference on Availability, Reliability and Security*, pages 173–180, Oct. 2009.

561. P.-N. Tan and V. K. J. Srivastava. Selecting the right objective measure for association analysis. *Information Systems*, 29(4):293–313, June 2004.

562. V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok. Generating realistic datasets for deduplication analysis. In *Proceedings of the 2012 USENIX Annual Technical Conference, ATC'12*, page ???, June 2012.

563. M. Templ, B. Meindl, and A. Kowarik. Introduction to statistical disclosure control (SDC). Technical report, International Household Survey Network, Oct. 2015.

564. P. E. Tetlock. Accountability: The neglected social context of judgment and choice. *Research in Organizational Behavior*, 7:297–332, 1985.

565. P. E. Tetlock. An alternative metaphor in the study of judgment and choice: People as politicians. *Theory and Psychology*, 1(4):451–475, 1991.

566. P. E. Tetlock, O. V. Kristel, S. B. Elson, M. C. Green, and J. S. Lerner. The psychology of the unthinkable: Taboo trade-offs, forbidden base rates, and heretical counterfactuals. *Journal of Personality and Social Psychology*, 78:853–870, 2000.

567. T. A. Thayer, M. Lipow, and E. C. Nelson. *Software Reliability*. North-Holland Publishing Company, 1978.

568. E. Thereska, B. Doebel, A. X. Zheng, and P. Nobel. Practical performance models for complex, popular applications. In *SIGMETRICS'10 Performance Evaluation Review*, pages 1–12, June 2010.

569. S. Thummalapenta, L. Cerulo, L. Aversano, and M. D. Penta. An empirical study on the maintenance of source code clones. *Empirical Software Engineering*, 15(1):1–4, Feb. 2009.

570. J. T. Townsend. Theoretical analysis of an alphabetic confusion matrix. *Perception & Psychophysics*, 9(1A):40–50, 1971.

571. A. Treisman and J. Souther. Search asymmetry: A diagnostic for preattentive processing of separable features. *Journal of Experimental Psychology: General*, 114(3):285–310, 1985.

572. J. E. Triplett. Performance measures for computers. In *Deconstructing the Computer*, pages 99–139, Feb. 2003.

573. K. S. Trivedi. *Probability & Statistics with Reliability, Queuing and Computer Science Applications*. Wiley, second edition, 2002.

574. T. S. Tullis and J. N. Stetson. A comparison of questionnaires for assessing website usability. In *Proceedings of Usability Professionals Association*, pages 1–12, June 2004.

575. J. Turley. Embedded processors. http://www.extremetech.com, Jan. 2002.

576. H. Turner and D. Firth. *Generalized nonlinear models in R: An overview of the gnm package*. University of Warwick, UK, 1.0-8 edition, Apr. 2015.

577. J. Tzelgov, V. Yehene, L. Kotler, and A. Alon. Automatic comparisons of artificial digits never compared: Learning linear ordering relations. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 26(1):103–120, 2000.

578. The ultimate Debian database. website, 2014. http://wiki.debian.org/UltimateDebianDatabase/.

579. Defence technical information center. http://dsearch.dtic.mil, July 2016. Search page for DTIC reports.

580. I. Utting, D. Bouvier, M. Caspersen, A. E. Tew, R. Frye, Y. B.-D. Kolikant, M. McCracken, J. Paterson, J. Sorva, L. Thomas, and T. Wilusz. A fresh look at novice programmers' performance and their teachers' expectations. In *ITiCSE-WGR'13*, pages 15–32, June 2013.

581. J. v. Kistowski, H. Block, J. Beckett, K.-D. Lange, J. A. Arnold, and S. Kounev. Analysis of the influences on server power consumption and energy efficiency for CPU-intensive workloads. In *ICPE '15 Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 223–234, Jan. 2015.

582. K. G. van den Boogaart and R. Tolosana-Deldago. *Analyzing Compositional Data with R*. Springer, 2013.

583. M. J. P. van der Meulen. *The Effectiveness of Software Diversity*. PhD thesis, Centre for Software Reliability, City University, Nov. 2007.

584. M. P. van Oeffelen and P. G. Vos. A probabilistic model for the discrimination of visual number. *Perception & Psychophysics*, 32(2):163–170, 1982.

585. H. VanLehn. *Mind Bugs: The Origins of Procedural Misconceptions*. The MIT Press, 1990.

586. R. Vasa. *Growth and Change Dynamics in Open Source Software Systems*. PhD thesis, Faculty of Information and Communication Technology, Swinburne University of Technology, Melbourne, Oct. 2010.

587. B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens. On the variation and specialisation of workload - A case study of the Gnome ecosystem community. *Empirical Software Engineering*, 19(4):955–1008, Aug. 2012.

588. W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, fourth edition, 2002.

589. P. Verghese and D. G. Pelli. The information capacity of visual attention. *Vision Research*, 32(5):983–995, 1992.

590. I. Vessey. Cognitive fit: A theory-based analysis of the graphs versus tables literature. *Decision Sciences*, 22(2):219–240, Mar. 1991.

591. A. Vetro, N. Zazworka, C. Seaman, and F. Shull. Using the ISO/IEC 9126 product quality model to classify defects: a controlled experiment. In *16th International Conference on Evaluation & Assessment in Software Engineering, EASE 2012*, pages 187–196, May 2012.

592. B. Veytsman and L. Akhmadeeva. Towards evidence-based typography: First results. *TUGboat*, 33(2):156–156, Apr. 2012.

593. F. Viénot, H. Brettel, and J. D. Mollon. Digital video colourmaps for checking the legibility of displays by dichromats. *COLOR research and application*, 24(4):243–252, Aug. 1999.

594. V. Villard. Android version distribution history. http://www.bidouille.org/misc/androidcharts, 2015.

595. T. H. Vines, A. Y. K. Albert, R. L. Andrew, F. Débarre, D. G. Bock, M. T. Franklin, K. J. Gilbert, J.-S. Moore, S. Renaut, and D. J. Rennison. The availability of research data declines rapidly with article age. Technical report, ???, Dec. 2013.

596. K. D. Vogeleer, G. Memmi, and P. Jouvelot. Parameter sensitivity analysis of the energy/frequency convexity rule for nanometer-scale application processors. Technical report, ???, Aug. 2015.

597. K. von Fintel and L. Matthewson. Universals in semantics. *The Linguistic Review*, 25(1-2):139–201, 2008.

598. J. Wagemans, J. H. Elder, M. Kubovy, M. A. Peterson, S. E. Palmer, M. Singh, and R. von der Heydt. A century of Gestalt psychology in visual perception: I. Perceptual grouping and figure–ground organization. *Psychological Bulletin*, 138(6):1172–1217, 2012.

599. J. Wainer, C. G. N. Barsottini, D. Lacerda, and L. R. M. de Marco. Empirical evaluation in computer science research published by ACM. *Information and Software Technology*, 51(6):1081–1085, June 2009.

600. S. Waligora, J. Bailey, and M. Stark. Impact of Ada and object-oriented design in the flight dynamics division at Goddard space flight center. Technical Report SEL-95-001, Goddard Space Flight Center, Mar. 1995.

601. P. Wang. Chasing the hottest IT: Effects of information technology fashion on organizations. *MIS Quarterly*, 34(1):63–85, Mar. 2010.

602. L. Wanner, C. Apte, R. Balani, P. Gupta, and M. Srivastava. A case for opportunistic embedded sensing in presence of hardware power variability. In *HotPower'10 Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8, Oct. 2010.

603. C. Ware. *Information Visualization Perception for Design*. Morgan Kaufmann Publishers, 2000.

604. W. H. Ware, S. N. Alexander, P. Armer, M. M. Astrahan, L. Bers, H. H. Goode, H. D. Huskey, and M. Rubinoff. Soviet computer technology—1959. Technical Report RM-2541, The RAND Corporation, Mar. 1960.

605. P. C. Wason. Reasoning about a rule. *The Quarterly Journal of Experimental Psychology*, 20(3):273–281, 1968.

606. V. M. Weaver and J. Dongarra. Can hardware performance counters produce expected, deterministic results? In *3rd Workshop on Functionality of Hardware Performance Monitoring*, pages 1–11, Dec. 2010.

607. V. M. Weaver and S. A. McKee. Can hardware performance counters be trusted? In *IISWC'08 IEEE International Symposium on Workload Characterization*, pages 141–150, Sept. 2008.

608. E. U. Weber, A.-R. Blais, and N. E. Betz. A domain-specific risk-attitude scale: Measuring risk perceptions and risk behaviors. *Journal of Behavior and Decision Making*, 15(4):263–290, Apr. 2002.

609. D. M. Wegner. *The Illusion of Conscious Will*. MIT Press, 2002.

610. M. H. Weik. A survey of domestic electronic digital computing systems. Technical Report 971, Ballistic Research Laboratories, Maryland, Dec. 1955.

611. M. H. Weik. A third survey of domestic electronic digital computing systems. Technical Report 1115, Ballistic Research Laboratories, Maryland, Mar. 1961.

612. J. A. White. Grapher pics. http://www.talljerome.com/mathnerd.html, Oct. 2012.

613. J. M. Wicherts, M. Bakker, and D. Molenaar. Willingness to share research data is related to the strength of the evidence and the quality of reporting of statistical results. *PLoS ONE*, 6(11):e26828, Nov. 2011.

614. W. A. Wickelgren. Size of rehearsal group and short-term memory. *Journal of Experimental Psychology*, 68(4):413–419, 1964.

615. A. Wierzbicka. *Semantics: Primes and Universals*. Oxford University Press, 1996.

616. R. Wilcox. *Introduction to Robust Estimation & Hypothesis Testing*. Elsevier, 3rd edition, 2012.

617. J. Wiley. Expertise as mental set: The effects of domain knowledge in creative problem solving. *Memory & Cognition*, 26(4):716–730, 1998.

618. L. Wilkinson. *The Grammar of Graphics*. Springer, second edition, 2005.

619. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition, 2005.

620. D. Wren. Passmark website??? http://www.passmark.com, July 2014.

621. Microsoft server protocol documentation. website, 2015. http://www.microsoft.com.

622. T. Yuki and S. Rajopadhye. Folklore confirmed: Compiling for speed = compiling for energy. Technical Report CS13-107, Computer Science Department, Colorado State University, Aug. 2013.

623. S. F. Zeigler. Comparing development costs of C and Ada. Technical report, Rational Software Corporation, Mar. 1995.

624. A. Zeileis, K. Hornik, and P. Murrell. Escaping RGBland: Selecting colors for statistical graphics. *Computational Statistics & Data Analysis*, 53(9):3259–3270, July 2009.

625. A. Zeller, T. Zimmermann, and C. Bird. Failure is a four-letter word –A parody in empirical research –. In *Promise '11 Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, pages 5:1–5:7, Sept. 2011.

626. J. Zhang and H. Wang. The effect of external representations on numeric tasks. *Quarterly Journal of Experimental Psychology*, 58(5):817–838, Oct. 2005.

627. X. Zhang. *An Analysis of the Effect of Environmental and Systems Complexity on Information Systems Failures*. PhD thesis, University of North Texas, Aug. 2001.

628. H. Zhou and A. Fishbach. The pitfall of experimenting on the web: How unattended selective attrition leads to surprising (yet false) research conclusions. *Journal of Personality and Social Psychology*, 111(4):493–504, Oct. 2016.

629. K. Zhou, P. Huang, C. Li, and H. Wang. An empirical study on the interplay between filesystems and SSD. In *7th International Conference on Networking, Architecture and Storage (NAS)*, pages 124–133, June 2012.

630. X. Zhu, E. J. W. Jr., C. Sadowski, and Q. Song. An analysis of programming language statement frequency in C, C++, and Java source code. *Software–Practice and Experience*, 15(11):1479–1495, Nov. 2015.

631. P. M. Zislis. An experiment in algorithm implementation. Technical Report CSD-TR-96, Purdue University, Indiana, USA, June 1973.