BrailleR in Action

A. Jonathan R. Godfrey

2021-12-14

Contents

Preface

If blind students are to truly gain access to statistical analyses, they will need to be able to successfully complete a course in statistics at university level. To do this, they must learn how the graphical techniques used in the sighted world look and are used. Generation of tactile images can show blind students what a particular graph does in a general sense, but greater understanding will come from generating these graphs as part of an analysis — in the same way it does for sighted students.

While sighted students can make use of a number of graphical user interfaces (GUIs), blind students are restricted to use of the command-line mode of operation or typing out an R script in full. One key benefit of the GUI mode of working is the ability to quickly generate basic numerical and graphical analyses. Blind students need to gain the same information as their sighted peers without expending too much additional time and energy. The `BrailleR` package aims to bridge this gap by delivering the range of analyses commonly found in introductory courses via a reduced set of commands.

Once blind students have completed their first course in statistics, they may embark on research at a university, or head out into industry to apply their knowledge. Irrespective of the direction they choose, they will need certainty in being able to independently create graphs for the sighted readers of their work. Creation of tactile images that provide the same representation of the images to be placed in documents can provide a solution, but all too often blind people do not have access to the right software and hardware to generate tactile images for themselves with the immediacy that is required. The `BrailleR` package aims to provide textual information to the blind user in conjunction with the graph that would be placed in the final report. `BrailleR` does this by interpreting the object that is implicitly created whenever a graph is created in R. by creating an appropriate and concise text representation of the graph.

In summary, this book presents the work included in the `BrailleR` package that will assist blind students successfully complete an introductory course in statistics when other software options fail them. Many of the functions support workflows that improve the efficiency of blind users at all levels of experience.

I've tried a few ways to help get blind people using the BrailleR package and

needed a place to combine the efforts easily. I don't yet know if this e-book will turn into anything but a few webpages, but let's see shall we?

Jonathan Godfrey

Citation details

Please refer interested parties to the online edition of this work at https://R-Resources.massey.ac.nz/BrailleRInAction/

When citing this work, please use the title, author, and date information on this page. The online version has ISBN978-0-473-41495-5 and is preferred for citation over other formats. The epub version has ISBN 978-0-473-41493-1 and pdf version has ISBN 978-0-473-41494-8; these fixed formats were created in October 2017.

Copyright information

Chapter 1

Introduction

Access to information is crucial for a blind person's success in education, but transferring the knowledge about the existence of techniques into actually being able to perform those tasks is what will make that blind person employable. This e-book is based on the author's experiences as a blind academic in statistics who has often been called on for advice by blind students studying statistics and their sighted lecturers wanting to provide the best possible support. Unfortunately, there is little information available in academic journals or the public domain to offer the statistics lecturer who is about to teach a blind student for the first time. That is changing however.

At the 2009 Workshop on E-Inclusion in Mathematics and Sciences, I was able to meet other researchers and scientists who are also concerned about the low rate of blind people entering the sciences in a broad sense and the mathematical sciences in particular. (Godfrey, 2009) presented my first formalized summary of what I saw as the current state of affairs for blind people taking statistics courses; that work discussed what was possible in theory, but lacked evidence of what was happening in practice. A collaboration with the only other blind person to gain employment as a full-time Lecturer in Statistics led to a more comprehensive review of what we believe is the current state of affairs for blind students learning statistics. (Godfrey and Loots, 2015) present a wide range of issues that blind students and their teachers must overcome, often through use of the best software options available today. R (R Core Team, 2021) has featured as a key element in my recommendations for teaching statistics to blind students (Godfrey, 2009) because it offers the greatest opportunity for meeting the needs of blind users.

An invitation to attend the second "Summer University" event was taken up in mid 2011; the purpose of this event was to show blind students and disability support staff from universities what software and hardware is available to improve the success rates of blind students entering the sciences. This was my first direct

opportunity to show blind students how R could be used by them, and the relative merit that R has for them over alternative statistical software. I observed blind students working with R and struggling more than I thought was truly necessary. I knew I could do something about it and have spent a lot of time doing so ever since. An R Journal article (Godfrey, 2013b) followed which exposed the needs of blind users to the R community and also announced the initial development of the `BrailleR` package (Godfrey et al., 2021).

The `BrailleR` package is my attempt to help blind students and their lecturers. It is built on functions that I use to work as a lecturer in statistics. It aims to make up for the inability blind users have to work with the same extension tools their sighted peers are using, including the increasing number of graphical user interface (GUI) and integrated development environment (IDE) options.

1.1 Why will I use the BrailleR package as a novice?

Blind users will want to use the BrailleR package while they are novice R users, but may also want to continue using some of the tools as their skill levels increase. Each of the following reasons for using the BrailleR package are expanded on by way of examples that go into more detail in subsequent chapters of this text.

1.1.1 BrailleR improves the accessibility of graphical information

BrailleR converts standard graphs created by standard R commands into a textual form that can be interpreted by blind students who cannot access the graphs without printing the image to a tactile embosser, or who need the extra text to support any tactile images they do create.

At present this is limited to only a few graph types found in base R functionality. An example of a histogram is presented in Chapter 4.

1.1.2 BrailleR helps gain access to the content of the R console

BrailleR makes text output (that is visually appealing) more useful for a blind user who is reliant on synthesized speech or braille output to interpret the results. The first example of this kind presented in Chapter 4 shows how the summary statistics for a dataset can be made easier for a screen reader user.

1.1.3 BrailleR includes convenience functions

Many analyses get repeated over and over again with different variables. Some people like a graphical user interface (GUI) but none of the GUIs developed for R to date are accessible by screen reader users.

BrailleR includes some functions which generate pro forma analyses. When these functions are employed, they generate an HTML document that includes the analysis in an easy to use format. The R commands used to create the analysis are stored in an R script file so that a user can modify the commands if changes are necessary. These functions are introduced in Chapter 7.

1.2 Why will I use the BrailleR package if I am not a novice?

I think some of the reasons for using the package while you are a novice R user remain relevant to more-experienced users too, but perhaps the main reason for continuing to use BrailleR is that of efficiency. The convenience functions introduced in Chapter 7 give you a starting point for analyses. Behind those convenience functions was an R markdown file that generated the R script and the HTML document. Getting into markdown is a great idea and will not take you long to learn.

BrailleR also includes some tools for helping run your R jobs without running R. Experienced users do this all the time so these tools aren't really meant for blind users alone, but as blind people often find little inefficiencies tiresome, I've incorporated the tools I use for my own efficiency when I think they might prove useful to other blind people.

In summary, the `BrailleR` package is my attempt to help blind students and their lecturers. It is built on functions that I use to work as a lecturer in statistics. It aims to make up for the inability blind users have to work with the same extension tools their sighted peers are using, including the increasing reliance on GUIs and integrated development environment (IDE) options.

1.3 Why will I use this book?

This book has been compiled for a variety of reasons. The obvious reason is that the `BrailleR` package has many tools that cannot be demonstrated using the standard documentation options found in many packages. Some of the material found here did start in package vignettes, but the time taken to re-work them all was slowing down development of the package.

A key feature of this book is that it was written by a blind person, using only the tools the book suggests, to create a book that can be read by blind people. This end to end workflow is proof that the `BrailleR` package helps make some actions possible, and others easier for a blind person to undertake. All too often, blind people find themselves using substitute workflows that are painfully slow compared to the tools being used by our sighted peers. I'm not suggesting that the tools here will make a blind person more efficient than all of their peers, but being able to use the most efficient tools that sighted people could choose to use does matter. Using these tools will cut the disadvantage a blind person has to a minimum; I'd even go so far as to suggest that the gap between a blind person and a sighted person's efficiency is reduced if we use the best tools on offer as compared to the gap we'd suffer if we chose to use inferior tools.

1.4 What this book is not

This book is not a comprehensive guide to using R. There are now hundreds of books a blind (or sighted) person could choose to wade through, some are good, some are great, and some are truly awesome! There are also plenty of resources on the internet that are tired and really should be avoided. I am keen to promote those texts that can be easily worked with by a blind user. I'll mention them as we get to the right point in this book for doing so, but now is a good time to mention a few that have really made a difference to me and creation of this book.

I am truly reliant on the resources offered by help pages hosted by RStudio. I often include RStudio.com in my list of search terms. In adition, many of the individuals I hold in very high regard have close ties to RStudio.

Yihui Xie's work on the `knitr` package (Xie, 2021) and his books on writing documents using reproducible research techniques (Xie, 2015) are key to my success and in my opinion, have also led to making life much easier for blind users to read the work of others and to create new content for themselves.

I frequently use several books written by Hadley Wickham and his collaborators. In particular, I rate the R packages book (Wickham, 2015), the Advanced R book [Wickham (2014a)', and the R for Data Science book (Grolemund and Wickham, 2016).

Chapter 2

History of the BrailleR Project

I am one of only two blind people in the world today who gained employment as full-time lecturers of statistics, that is, teaching statistics classes and doing research in theoretical matters as against applying statistical techniques. For years, I tried to keep my blindness separate from my research but I took some opportunities that came my way and heeded the advice of some colleagues to put more energy into improving the ability of blind students around the world to have greater access to statistics courses and statistical understanding. This document shows you a bit more insight into how I (with the help of some useful collaborations) got the BrailleR package to where it is now.

2.1 My background

My adult life has been centred around Massey University, initially as an extra-mural student and then studying on campus. I have undergraduate degrees in Finance and Operations Research, a Master's degree in Operations Research and a PhD in Statistics. I was a Graduate Assistant from 1998 to 2002, and then Assistant Lecturer from January 2003 to June 2004 when I became a Lecturer in Statistics. I was promoted to Senior Lecturer in late 2014.

While I don't find it important, I do get asked about the condition that caused my blindness. It is Retinitis Pigmentosa. I do have some light perception, and can make use of it in familiar surroundings for orientation but it has no value to me for reading anything at all. I chose to work with screen reading software when I started university and obtained my first computer because my residual vision at the time was limiting my reading speed. I have therefore operated a computer as a totally blind user throughout my adult life.

I did not learn braille until after I completed my PhD. This might seem strange, but there was very little material in a suitable digital format for me to read throughout my student life. Things have changed and I now spend a lot more time reading material and doing programming where the accuracy of braille is absolutely necessary. Braille has now become a very important part of my working life and I have a braille display connected to my computer most of the time.

2.2 Getting the BrailleR Project started

I used to keep my research interests separate from my blindness, but I was regularly called upon to discuss how a blind person could study and teach Statistics by many people within New Zealand and occasionally from overseas. In 2009, I attended the Workshop on E-Inclusion in Mathematics and Science (WEIMS09) where I met other people interested in improving the success rates of blind students in the mathematical sciences. My paper was about accessibility of statistics courses, but I did point out the usefulness of R in preference to other tools I had used to that point in time (Godfrey, 2009).

I discovered that there is room for me to take a leading role in the development of ideas that can help other blind people learn about statistical concepts. I have been invited to all six Summer University events run by the organizers of the International Conference on Computers Helping People (ICCHP), but have been unable to attend twice due to the high cost of transporting me to Europe. I have delivered an introductory workshop on using R at four of these events (Godfrey, 2011; Godfrey, 2013c; Godfrey, 2014a; and Godfrey, 2016a).

Having observed the attendees at the 2011 Summer University as they came to grips with R, I knew there was more I could do to help them and other blind students. I started work on the BrailleR package (Godfrey et al., 2021) in the second half of 2011 and first proposed it could work for blind users at the Digitisation and E-Inclusion in Mathematics and Science (DEIMS12) workshop held in Tokyo during February 2012 (Godfrey, 2012a).

I wasn't to know the value of another talk I gave at DEIMS12 for another two years; this second talk and associated conference paper focused on how I was using Sweave to create accessible statistical reports for me and more beautifully formatted ones for my statistical consulting clients. (Godfrey, 2012b). I now know that the groundwork I had done contributed to my desire to present my workflow as a workshop at the 5^{th} Summer University in 2014 (Godfrey, 2014c). It also stood me in good stead for the work that followed on the `BrailleR` package as it developed in late 2014 and early 2015.

2.3 The starting point example

The basic graph that has been used for almost every presentation of the `BrailleR` package is a histogram. There is a more detailed example, but the following commands create a set of numbers that can be kept for further processing once the graph has been created. It is the re-processing of these numbers that leads to the text description that follows.

```
library(BrailleR)
```

```
## The BrailleR.View,  option is set to FALSE.
##
## Attaching package: 'BrailleR'
## The following objects are masked from 'package:graphics':
##
##     boxplot, hist
## The following object is masked from 'package:utils':
##
##     history
## The following objects are masked from 'package:base':
##
##     grep, gsub
```

```
x=rnorm(1000)
VI(hist(x))
```

```
## This is a histogram, with the title: Histogram of x
## "x" is marked on the x-axis.
## Tick marks for the x-axis are at: -3, -2, -1, 0, 1, 2, 3, and 4
## There are a total of 1000 elements for this variable.
## Tick marks for the y-axis are at: 0, 50, 100, 150, and 200
## It has 14 bins with equal widths, starting at -3 and ending at 4 .
## The mids and counts for the bins are:
## mid = -2.75  count = 5
## mid = -2.25  count = 17
## mid = -1.75  count = 41
## mid = -1.25  count = 79
## mid = -0.75  count = 150
## mid = -0.25  count = 206
## mid = 0.25  count = 192
## mid = 0.75  count = 153
## mid = 1.25  count = 95
## mid = 1.75  count = 32
## mid = 2.25  count = 21
## mid = 2.75  count = 7
```

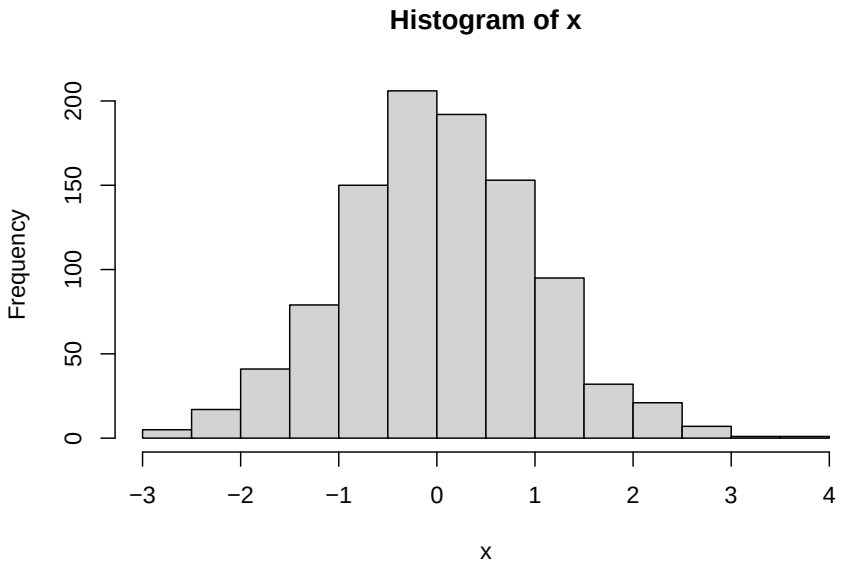


Figure 2.1: A histogram of 1000 random values from a normal distribution

```
## mid = 3.25  count = 1
## mid = 3.75  count = 1
```

This first example showed me what was possible if only I could get a few things sorted out. All histograms are created by a function that stores the results (both numeric and text details) and calls this stored set of values a "histogram". The main issue is that storing the set of details is not consistent in R, nor is the fact that the stored object gets given a "class" to tell me what type of object it is. This problem haunted me for quite some time because I was talking to the wrong people about the problem; it was time to find people that held the solution instead of talking to the people that would benefit if a solution was found.

2.4 Why just plain text?

The first initiative of `BrailleR` was to turn information presented in a visual medium into a medium that is simple to work with, efficient, and complementary to the skill set of as wide a group of blind people as possible. With this in mind, a plain text solution for gaining access to visual information was favoured above tactile images as there is no need for any additional adaptive technology (hardware or software) over that used for the blind person's other

work. Information presented in text is readable in braille or heard via synthetic speech, and is therefore only limited to the user's skill set.

Other research projects are progressing the creation of tactile images that replace the printed graphics sighted people deal with all the time in statistical work. Tactile images do have a role to play in educating blind people and providing them with access to information, but their greatest drawback remains their lack of immediacy. Rendering a graphic into a master file complete with braille labels, and then producing this for consumption are tasks receiving much attention, but the value for such efforts is probably greatest for situations where one master can be used to create multiple copies for wider consumption such as the conversion of textbooks into braille. Some preliminary investigations into the ability to create tactile image versions of graphs created in R have been made as part of the wider BrailleR Project.

Under a sighted paradigm, many graphs in statistical work are developed for one-time use and discarded almost immediately. It may be some time until the effectiveness of tactile image hardware and software is ready to deliver this outcome for all blind users of R; in the meantime, we must find ways of gaining access to the graphical information without reliance on another person's ability to translate the visual medium into something more useful for us. We must also acknowledge here that the number of blind people with their own tactile image hardware and software is regrettably low. Even though it is possible to directly export a graph created in R to an embosser, use of this practice is not yet commonplace. Perhaps we can look forward to the time when most blind people can have access to a refreshable tactile display that can display two-dimensional images that compare with the best that is possible in hard copy today.

Sonification is considered by many to be an alternative to production of tactile images for a blind audience. Work on sonifying statistical graphs is being considered, but is at present best described as experimental. Several concerns must be overcome before sonification can be claimed a truly useful method for interpretting data. There is an assumption made frequently by correspondents with this author that lack of vision implies a blind person has superior aural ability . This may be true for some blind people, but it is not universally true. We must be sure that the interpretation gained through sound is comparable to the interpretation that a graph in visual form would convey. Proponents of sonification often suggest that sound provides a different interpretation to a visual display. If they are correct, then sonification cannot be a true alternative to a visual graph. The same, of course, can be said for the difference between the interpretation obtained through touching tactile images. In both cases, the solution for blind users must be that the visual representation can be converted to a useful format when blind users need it, without the assistance of a sighted person to check the validity of the outcome.

2.5 Exposure of the BrailleR package outside the blind community

It was obvious to me that getting the word out to the masses about the usefulness of R for blind students and professionals was crucial. I started to compile my notes built up from various posts made to email groups and individuals over the years, as well as the lessons I learned from attendance at the 2nd Summer University event. This led to the eventual publication of my findings in (Godfrey, 2013b). I know that this was a worthwhile task because it was read by teachers of blind students who were already using R for their courses. One such person tested R and a screen reader and managed to find a solution to a problem posed in Godfrey (2013b) which led to an addendum (Godfrey and Erhardt, 2014).

I presented some of my work via a poster (Godfrey, 2013a) at the NZ Statistical Association conference in Hamilton during November 2013. This 'poster' presentation was developed as a multimedia presentation so that the audience could observe video footage, handle tactile images and be able to talk with me about the BrailleR Project. The plan to get talking with people instead of talking at them worked and I started a really useful collaboration with Paul Murrell from the University of Auckland. His major contributions didn't feature in the BrailleR package for some time, but we're making some really nice progress. Paul is an expert in graphics, especially their creation and manipulation in R. Our discussions about graphics has yielded a few titbits for my own work that have been tested for the package. We've been working on how to make scalable vector graphics that can be augmented to offer blind users greater interactivity and therefore hopefully greater understanding (see Godfrey and Murrell, 2016).

2.6 Reviewing statistical software

I have been asked about the use of R in preference to other statistical software by many blind students, their support staff, and their teachers. Eventually I joined forces with the only other blind person to gain employment as a lecturer of statistics (Theodor Loots, University of Pretoria) to compare the most commonly used statistical software for its accessibility (Godfrey and Loots, 2014). I summarised this paper at the 5th Summer University event (Godfrey, 2014d), and offered a similar presentation at the 6th Summer University event (Godfrey, 2016b) with a few updates. It is important to keep abreast of developments, because the statistical software changes, and so does the screen reading software that gives us access to the mainstream statistical software.

2.7 Attendance at UseR conferences

On my way to the 5th Summer University event, I managed to attend the principal conference for R users (UseR!2014) in Los Angeles where I presented my findings (Godfrey, 2014b). Perhaps the most valuable outcome of this conference was the ability to attend a tutorial on use of the `knitr` package (Xie, 2021) and then talk to its author, Yihui Xie. I'd already seen the `knitr` package before attending UseR!2014 and implemented it for some of my teaching material by updating the Sweave documents already in use.
The real value came in realising what I could probably do if I used R markdown to do a few things I had found very hard using the Sweave way of working. More specifically, generating an R markdown file (Rmd) from an R script was much easier than generating a Sweave file (Rnw). Writing the convenience functions for the `BrailleR` package started to look very achievable at this point, and so work began. I dug out some old work that wasn't fit for sharing and converted it to the markdown way of working. There has been sufficient progress in the BrailleR Project that I presented it at UseR!2015 (Godfrey, 2015). In 2016, I presented my findings on writing (and therefore reading) R markdown documents for (and by) blind users (Godfrey and Bilton, 2016).

2.8 The ongoing work

The introduction of R markdown to the BrailleR package made a huge difference. I've been able to write enough example code that once I found a friendly postgraduate student (Timothy Bilton) to put some time into it, we've managed to add more convenience functionality. Timothy improved some of my earlier work and tried a few things of his own. This left me with the time to add increased functionality for helping blind users get into markdown for themselves.

One of my irritations of working with markdown is that everyone else seems to write markdown and check their findings using RStudio (RStudio, 2018), which remains inaccessible for me and other screen reader users. I took an old experiment where I wrote an accessible text editor in wxPython, and with the help of a postgraduate student from Computer Science (James Curtis) we've modified it to process Rmd files. The `WriteR` application is now beyond experimental but there is still more to do on making it truly useful (Godfrey and Curtis, 2016). In 2018, I received a great deal of assistance from Marshall Flax who was able to help develop `WriteR` into a tool that could be very useful to blind people wishing to write and process R markdown files.

2.9 Acknowledgements

Contributions to the BrailleR Project are welcome from anyone who has an interest. I will acknowledge assistance in chronological order of the contributions I have received thus far.

Greg Snow was the first person to assist when he gave me copies of the original R code and help files for the R2txt functions that were part of his `TeachingDemos` package (Snow, 2020).

The Lions clubs of Karlsruhe supported my attendance at the 3rd Summer University event in 2013. This gave me the first opportunity to put the package in front of an audience that I hope will gain from the package's existence.

I've already mentioned the following contributors above:Paul Murrell, Yihui Xie, Timothy Bilton, James Curtis, and Marshall Flax.

I also need to acknowledge the value of attending the Summer University events. I gain so much from my interactions with the students who attend, the other workshop leaders who give me feedback, and the other professionals who assist blind students in their own countries.

Chapter 3

Getting started with BrailleR

The BrailleR package has been created for the benefit of blind people wishing to get more out of R than it already offers — which is actually quite a lot!

3.1 Accessibility of statistical software

A description of what makes statistical software accessible to the blind was given by (Godfrey and Loots, 2014). Many of the problems blind people face are a consequence of the failure of the specialist screen reading software we use to interact with graphical user interfaces. Many software applications are making use of graphics when once the information would be presented in text form with an accompanying graph. — SPSS (SPSS Inc., 2012) is an example of this, and to a lesser extent so is the standard edition of Minitab (Minitab Inc., 2012). The output from Minitab Express (Minitab Inc., 2014) is a prime example of the worst possible presentation of information that was originally text; this product generates graphics that include text which is not readable by a blind person for all of its statistical output. The speech output software used by blind people can only interpret information that is text. As a guide, if the individual text in a window cannot be highlighted using the keyboard and not the mouse, then it is likely that this text will not be read for the blind user.

It is true that some add-on packages for R also generate unreadable output, but as illustrated below, this is less of an issue than for software like SPSS or Minitab. (Godfrey and Loots, 2014) gave more detailed scrutiny of R, SAS (SAS Institute Inc., 2010) , SPSS, and Minitab. Of these four applications, R and SAS were clearly superior to SPSS and the standard edition of Minitab.

Some software applications have retained their historical links to the days when graphs and tables were rendered in well controlled monospace fonts — Minitab for example. In such applications the user may still be able to produce an ASCII graphic instead of the more commonly used high resolution graphs expected of today's software (and user). In contrast, SAS and SPSS can use HTML to present information in well-formatted tables. These output windows are preferred by blind users over the graph window displays but sometimes the amount of information is not easily understood. Presentation of output is often read by eye in a vertical direction, while speech output software will read line by line. Take for example, a multiple regression where the sighted reader may scan down the list of p values in the right hand column; the same information being read aloud in line by line style could prove quite difficult to interpret. Blind users can use combinations of keys to move around HTML tables to speed up this process and avoid reading the intervening columns of output. While SPSS and SAS can deliver formatted HTML as a matter of course, R users must resort to the add-on packages, many of which are available on CRAN.

3.2 Accessibility of R

(Godfrey, 2013b) documented the ability to use R almost immediately after installation; only one minor change is recommended and can be achieved in less than a minute even when explained via email or a telephone call. For users of the Windows operating system, up until Windows XP, I always recommended running R in a terminal window instead of the GUI; the shortcut placed on the Windows desktop would then need to be pointed to `Rterm.exe` instead of `Rgui.exe`. Macintosh and Linux users are operational with no special actions required. Users of Windows Vista or Windows 7 did have an additional challenge of what appeared to be the screen locking up, or more exactly, the screen reader software "losing focus" in the R terminal window. The solution for this problem, as documented in (Godfrey and Erhardt, 2014), was to hit the `Alt` key. Blind users now need to compare the combinations of screen reader and the terminal versus the GUI. The decision should be made by the individual user after some experimentation; their decision may depend on the skill level they have with their prefered screen reader and should be revisited at a later date.

Blind students attending the R workshop at the 2011 Summer University, held in the Czech Republic, were able to set up R for use in a classroom setting on their own machines. This included a variety of operating systems and adaptive technology (hardware and software for blind people). Similar events were held in 2013 and 2014 but most attendees used computers supplied by the host organizing committees. Many attendees have made contact when issues have arisen, but none of the issues relate to the installation of R or its interactions with the particular hardware or software being used. I am confident that anyone intending to undertake use of any statistical software will be able to get R

working with their screen reader.

3.3 Installing the BrailleR package

To use the functionality of the BrailleR package you need to have it installed. The package has several dependencies so installation from the CRAN repository is recommended. This would be done by issuing the following two commands in an R session:

```
chooseCRANmirror(ind=1)
install.packages("BrailleR")
```

Note that the first and last letters of `BrailleR` are capitalised. This is important in R, but is also useful for screen reading software which will then give audibly different feedback, as compared to what a screen reader user hears from the lower case text "brailler".

If for some reason you have difficulty with the above commands, you can install the BrailleR package using a zip file version available from a CRAN repository or the latest version on GitHub.

From time to time, you should check that you are using the most recent version of the BrailleR package. You can update all installed packages using the commands:

```
chooseCRANmirror(ind=1)
update.packages(ask=FALSE)
```

Once you've got the package installed, you still need to get it running in your current R session by issuing one last command. When you issue the first of the following lines, the package start messages will also appear.

```
library(BrailleR)
```

You're ready to go!

3.3.1 Some initial setting up instructions

When you first use the `library(BrailleR)` command, you will see some start up messages and a question. The rules of R packages include not writing to the user's hard drive without expressly asking them for permission to do so. If you do not want a folder for your `BrailleR` files then use the temporary folder which will be removed when you end your R session. This will mean you need to answer the question over the location of the `MyBrailleR` folder next time you issue the `library(BrailleR)` command though.

The welcome message from `BrailleR` suggests you issue the `GetGoing()` command. This will ask you a few questions that will help personalise your use of

the `BrailleR` package. We will see how to alter these settings in Chapter 8 later so don't panic if you don't do it all right the first time. You can re-issue the `GetGoing()` command again at any time.

The book you are reading now can be reached from your R session by issuing the command `BrailleRInAction()`. That might seem a bit much, but do remember you can use tab completion to avoid typing the whole command name out in full. You will probably need no more than `B`, `r`, `a`, then tab (which adds the rest of BrailleR), then `I` and one last tab; add the opening and closing parentheses and press the Enter key. This will open the front page of the book in your browser. A similar command, `BrailleRHome()`, will open the BrailleR Project home page. You will need to be careful with upper versus lower case when entering commands. Note that there are often capitalised letters in the middle of `BrailleR` commands. This is known as "camel case" and it works well for screen reader users.

It is all too easy to feel you're doing it on your own, which even the most accomplished people have experienced. I put the `ThankYou()` command in the BrailleR package so that it would be easy to send me a message to tell me about your experiences as a blind person using R or to ask for help; it starts an email message to me. I'm not the only blind person out there using R, and many of us are on an email list so that we can share ideas and solutions for problems, many of which are specific to blind users. The `JoinBlindRUG()` command will start the email needed to join the BlindRUG email list.

3.4 What else do you need?

You obviously have R installed or an intention to do so soon if you are reading this document. Aside from R and the add-on packages that BrailleR needs, there are no other software requirements. There are several optional software installations that could make life easier if they are installed before you need them. In order of necessity, they are:

3.4.1 The document converter — pandoc

BrailleR requires the very useful file converter called pandoc. Get it from the pandoc download page

3.4.2 The principal integrated development environment — RStudio

It is a good idea to install RStudio, even if you can't actually use it as a blind person using screen reading software. The reason is that RStudio installs a few

other useful tools that we will make use of by other means. Get it from the RStudio download page

3.4.3 One programming language — Python

WriteR is a simple text editor written in wxPython that needs Python and wxPython. Unfortunately, they require separate downloads. You do not need this editor so do not install Python unless you are really keen. Windows users can obtain an executable file by issuing `GetWriteR()` once the BrailleR package has been successfully installed. More on this in Chapter 11.

3.5 BrailleR commands used in this chapter

The only BrailleR command actually recommended in this chapter was `GetGoing()`. You might find it useful to use `BrailleRHome()` and `BrailleRInAction()` from time to time, but you're already reading the book that the second of these commands opens.

The `ThankYou()` and `JoinBlindRUG()` commands should be used when you want to connect with me, or other blind R users.

At this stage it is recommended that you install any additional software manually when it is required.

Chapter 4

Some basic examples

This chapter presents some examples of text output generated by the `VI()` command of the `BrailleR` package. These examples generate output that is displayed in the R session just like any output from standard R commands. Please note however that not all `VI()` commands behave in this fashion; some more advanced uses of `VI()` are discussed in later chapters.

You will need the `BrailleR` package to be ready for use to follow along with the examples in this chapter. Do this by issuing the command `library(BrailleR)` now.

4.1 Accessibility of graphics in statistical software

Access to graphical representations of information from mathematical or statistical software is quite limited, and therefore limits the blind user's capacity. To this author's knowledge, no mathematical or statistical software has the capability of directly linking to any hardware or software solutions that make the information presented in graphs immediately available. The scalable vector graphic (SVG) format can be used to present a graphic with text embedded into the file for creating access for a blind user (Bulatov and Gardner, 2004; Gardner and Bulatov, 2010). Only a small number of statistical software applications have the capacity to create SVG files, but this capacity does not in itself create access because the text that makes them accessible must be added somehow; generally this is a manual process. The World Wide Web Consortium has a recommendation on the use of SVGs in web content (Dengler et al., 2011) and a number of add-on packages for R make use of SVG because of the opportunity to enrich a graphic's interactivity in webpages; see (Murrell and Potter, 2014) for a more detailed discussion of these packages and their functionality. Of particular

note is that there are different ways to create an SVG and care must be taken if the maximum accessibility for blind users is ever to be achieved.

In R, a graph can be saved as an SVG using the `svg()` command, but this approach uses the Cairo SVG format; this has the unfortunate outcome that text is not always preserved as a string, and some shapes are represented by an unstructured set of straight lines (Gardner and Bulatov, 2010). Retention of text as strings is crucial if modifications such as changing the font of any text to the braille font of a user's choosing is to be managed easily. Use of the `gridSVG` package (Murrell and Potter, 2014) does lead to creation of SVG files that do keep strings of text intact and a hierarchy of graphical elements. These SVG files are more easily modified to create accessibility for blind users that have the technology to interpret them (Gardner and Bulatov, 2010).
Any graph created in R using functions from the `graphics` package will need to be converted to the `grid` package system for generating graphics through use of the `gridGraphics` (Murrell, 2015) package before they are exported in SVG format. Existence of the `gridSVG` and `gridGraphics` packages means that it should be possible to automate the creation and addition of the necessary text information to the SVG so that the need for human intervention is minimised.

Add-on scripts for screen reader software, such as JAWS (Freedom Scientific, 2018), that attempt to interpret the graphs created by common spreadsheet software has been tried in the past. One major problem that results from the creation of these add-on scripts is that of maintenance; unfortunately, the add-on scripts for screen reader software support of spreadsheet applications has not kept pace with the developments of those applications sufficiently enough to give blind people access to the full range of graphs. Sporadic effort has been made at providing access to statistical software through the creation of add-on scripts for screen readers, but little effort was ever given to creating access to the content of graphics. It is unfortunate that these have also not been sufficiently maintained. In this respect the use of R or SAS currently hold an advantage over other commonly used software (notably SPSS and Minitab) for the blind user because no additional scripts for the screen reading software are required. (Godfrey and Loots, 2014). R and SAS also work well with the open source screen reader called NVDA (NVDA Team, 2018).

In conclusion, it seems that until such time as adaptive technology for creating immediate access via tactile images is commonly available, other solutions will continue to be relevant. In fact, even once the immediacy issue is overcome, there will be a place for solutions such as the `BrailleR` package as a complementary solution rather than a substitute.

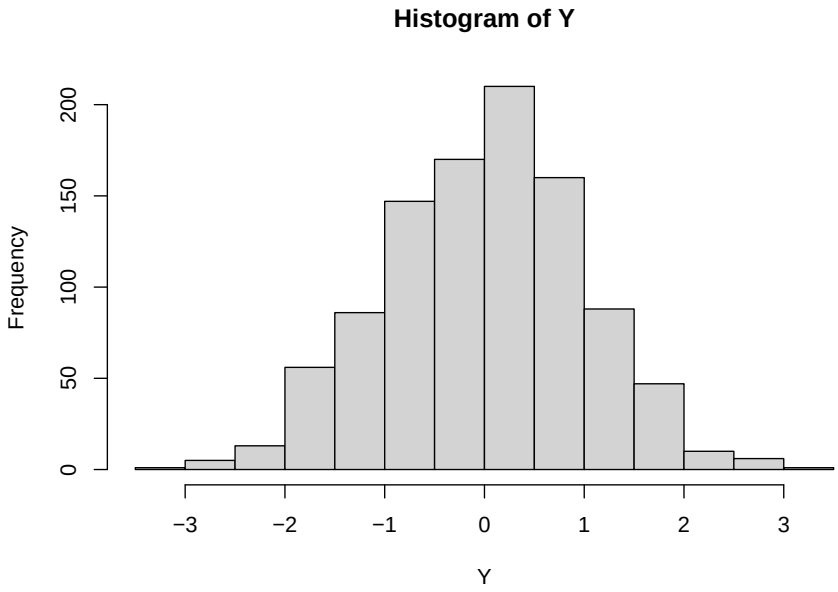
At each of the Summer University and Blind Science Conference events I have shown students that they could use R to gain access to graphed information in a manner almost unparalleled by other statistical software — the obvious exception to this rule is S-PLUS (Insightful Corp., 2003). This advantage comes from the fact that the S language, on which R is built, implicitly stores the data needed to create many graphs, and this data can be assigned to an object for further

manipulation. This is not true for graphs created using the ubiquitous `plot()` function however, and many other plotting functions that explicitly return a `NULL` object. We will first look at a graph object that can be explicitly stored.

4.2 Histograms

The first and most commonly used example demonstrating the value of the BrailleR package to a blind user is the creation of a histogram. For example, a sighted user wanting a histogram of 1000 randomly chosen values from a standard normal distribution would type

```
Y = rnorm(1000)
HIST = hist(Y)
```



A simple way for blind users to access the information used to create a graph is to ask R to print the object, using the `print()` command.

```
print(HIST)
```

```
$breaks
 [1] -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5

$counts
 [1]   1   5  13  56  86 147 170 210 160  88  47  10   6   1
```

```
$density
 [1] 0.002 0.010 0.026 0.112 0.172 0.294 0.340 0.420 0.320 0.176 0.094 0.020
[13] 0.012 0.002

$mids
 [1] -3.25 -2.75 -2.25 -1.75 -1.25 -0.75 -0.25  0.25  0.75  1.25  1.75  2.25
[13]  2.75  3.25

$xname
[1] "Y"

$equidist
[1] TRUE

$ExtraArgs
$ExtraArgs$main
[1] "Histogram of Y"

$ExtraArgs$xlab
[1] "Y"

$ExtraArgs$ylab
[1] "Frequency"

$ExtraArgs$sub
[1] ""


$NBars
[1] 14

$par
$par$xaxp
[1] -3  3  6

$par$yaxp
[1]   0 200   4


$xTicks
[1] -3 -2 -1  0  1  2  3

$yTicks
[1]   0  50 100 150 200

attr(,"class")
```

```
[1] "Augmented" "histogram"
```

The `print()` command can be wrapped around the `hist()` command which avoids explicit storage of the object, but still creates the histogram. This approach can be taken for many graph types where the option of suppressing the graph is not available. The `hist` command gives the user an explicit way of generating the information a blind user may want. Similar functionality exists using the `boxplot.stats()` command for example. Such workhorse functionality is useful but not well-documented.

Given the user has stored the output from the `hist()` function in an object, they can interpret the text to gain some idea of what appears in the histogram. As the object is designed for compatibility and flexibility, not all of the printed information is relevant to the end user however so some further education or perhaps further processing is still required. I found that some blind users had no appreciation for what a histogram actually looks like. In hindsight, this was reasonable because the way R presents a histogram (and many other graphs) does differ from presentations of the same data in other software.

This "print it out" approach is workable but is far from elegant. On some occasions the data structure listed is just a list of the actual data itself and is therefore not giving blind users the same information as sighted users get from a graph. Blind users need some sort of summarisation tool to replace the graph, and in many instances, are unlikely to have the skill to develop a sound statistical solution for themselves. It is also reliant on users knowing what the various elements printed out are, and how they link to the visual object created for the sighted world. The initial aim of the `BrailleR` package was to take the information created by various functions in R and create a text printout that reduces the amount of information that needs to be processed by blind users. In cases like that for the histogram example just introduced, this is a fairly simple task. For other situations, some further work needs to be done so that the textual information is a useful summary of the graphical information without crossing over the line of interpreting the information for the user.

So let's see what `BrailleR` is going to offer a blind user instead of the verbose printout seen above. First, the `Describe()` function describes a histogram in a general sense.

```
Describe(HIST)
```

```
A histogram created using the base graphics package.

General description: A histogram uses rectangles to represent the counts or relative frequencies
As with most graphs created by the base graphics package, the axes do not join at the bottom left
R normally plots a graph in a square window. This can be altered; the way this is done depends he

R hints: If you intend to make a tactile version of a histogram, you may find it useful to alter
```

Then, we can find out what is shown in the specific histogram using the `VI()`

function.

```
VI(HIST)
```

```
This is a histogram, with the title: Histogram of Y
"Y" is marked on the x-axis.
Tick marks for the x-axis are at: -3, -2, -1, 0, 1, 2, and 3
There are a total of 1000 elements for this variable.
Tick marks for the y-axis are at: 0, 50, 100, 150, and 200
It has 14 bins with equal widths, starting at -3.5 and ending at 3.5 .
The mids and counts for the bins are:
mid = -3.25  count = 1
mid = -2.75  count = 5
mid = -2.25  count = 13
mid = -1.75  count = 56
mid = -1.25  count = 86
mid = -0.75  count = 147
mid = -0.25  count = 170
mid = 0.25  count = 210
mid = 0.75  count = 160
mid = 1.25  count = 88
mid = 1.75  count = 47
mid = 2.25  count = 10
mid = 2.75  count = 6
mid = 3.25  count = 1
```

The `Describe()` and `VI()` commands actually call the `Describe.histogram()` and `VI.histogram()` commands because R knows it is a histogram that was generated by `hist()` earlier.

4.2.1 Important features

The commands used above explicitly stored the histogram. A blind user could use `VI(hist(y))` instead to get the same outcome. In that case, the `VI()` command would add to the impact of issuing the `hist()` command because the graphic is still generated for the sighted audience. The blind user can then read from the text description so that they can interpret the information that the histogram offers the sighted world.

4.2.2 Warning

The `VI()` function is partially reliant on the use of the `hist()` function that is included in the `BrailleR` package. If a histogram is created using a command that directly links to the original `hist()` command found in the `graphics` package, then the `VI()` command's output will not be as useful to the blind user.

This mainly affects the presentation of the title and axis labels; it should not affect the details of the counts etc. within the histogram itself.

This behaviour could arise if the histogram is sought indirectly. If for example, a function offers (as a side effect) to create a histogram, the author of the function may have explicitly stated use of the `hist()` function from the `graphics` package using `graphics::hist()` instead of `hist()`. Use of `graphics::hist()` will bypass the `BrailleR::hist()` function that the `VI()` command needs. This should not create error messages, but may result in some strange and possibly undesirable output.

4.3 Basic numerical summaries

The standard presentation of a summary of a data frame where each variable is given its own column is difficult for a screen reader user to read as the processing of information is done line by line. For example:

```
summary(airquality)
```

```
     Ozone           Solar.R          Wind             Temp      
 Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00  
 1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00  
 Median : 31.50   Median :205.0   Median : 9.700   Median :79.00  
 Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88  
 3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00  
 Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00  
 NA's   :37       NA's   :7                                       
     Month            Day      
 Min.   :5.000   Min.   : 1.0  
 1st Qu.:6.000   1st Qu.: 8.0  
 Median :7.000   Median :16.0  
 Mean   :6.993   Mean   :15.8  
 3rd Qu.:8.000   3rd Qu.:23.0  
 Max.   :9.000   Max.   :31.0  
```

The `VI()` command actually calls the `VI.data.frame()` command. It then processes each variable one by one so that the results are printed variable by variable instead of summary statistic by summary statistic. For example:

```
VI(airquality)
```

```
The summary of each variable is
Ozone: Min. 1   1st Qu. 18   Median 31.5   Mean 42.1293103448276   3rd Qu. 63.25   Max. 168   NA'
Solar.R: Min. 7   1st Qu. 115.75   Median 205   Mean 185.931506849315   3rd Qu. 258.75   Max. 334
Wind: Min. 1.7   1st Qu. 7.4   Median 9.7   Mean 9.95751633986928   3rd Qu. 11.5   Max. 20.7
```

```
Temp: Min. 56   1st Qu. 72   Median 79   Mean 77.8823529411765   3rd Qu. 85   Max. 97
Month: Min. 5   1st Qu. 6   Median 7   Mean 6.99346405228758   3rd Qu. 8   Max. 9
Day: Min. 1   1st Qu. 8   Median 16   Mean 15.8039215686275   3rd Qu. 23   Max. 31
```

4.3.1 Important features

Note that in this case, the blind user could choose to present the summary of each variable as generated by the `VI()` command, or the output from the standard `summary()` command. There is no difference in the information that is ultimately presented in this case.

4.4 BrailleR commands used in this chapter

The only explicit commands from the `BrailleR` package used in this chapter were the `Describe()` and `VI()` commands.

Chapter 5

New BrailleR commands for making and interpreting basic graphs

This chapter introduces two types of new commands found in the `BrailleR` package. There are several commands to help a blind user know what is included in a graph, starting with a tool to help "know" what is displayed in an otherwise inaccessible graphics device window. The other commands introduced in this chapter are substitutes for functions found in the base distribution of R. You can jump ahead to the examples, but there is some theory needed to explain how the `BrailleR` package does the extra work it does, and why we need to use these substitute commands.

You will need the `BrailleR` package to be ready for use to follow along with the examples in this chapter. Do this by issuing the command `library(BrailleR)` now.

5.1 What's in a graph?

A challenge for many blind people is to understand the way a scatter plot shows a relationship between two variables. In a theoretical sense they can learn that a scatter plot does show a relationship, outliers etc. but the ability to construct a scatter plot for themselves and then interpret it in a similar way to their sighted peers is not possible unless they have direct access to a tactile image embosser. Such access is not immediate as embossers are not portable. Although efforts to create tools that can be used by blind people who do not have access to an embosser are not new (Calder et al., 2006), more recent developments

have tended to focus on use of touch screen technologies. To this author's knowledge, no comprehensive solution exists to meet the need for blind users to independently create statistical graphs with confidence or to modify them without starting all over again.

Solutions for giving blind people access (in a general sense) are often aimed at providing an exact replica of what the sighted person can see. A statistical graph can exist on two levels: First, the exact detail of individual elements that are plotted in the graph, and second, the combined effect this collection of elements conveys.

As we look at the various graphs being produced in any analysis, we need to think about which of these activities is being done by the sighted world as they consume the content. If we do not understand what the consumer is doing, we cannot hope to provide an appropriate graph for them; to be a producer of a graph, you must also be a consumer of it, albeit temporarily. The solutions offered to blind users by `BrailleR` try to keep this ethos in mind.

5.2 Background

In Chapter 4, we saw creation of a histogram using the `hist()` command. The `hist()` command used for many years is found in the `graphics` package and has its own `plot()` command called `plot.histogram()` as well. This `plot()` command is actually a family of commands that all start with`plot.*()` where the star is replaced by the type of object that is being plotted. We use this `plot()` command all the time to give us plots for different reasons. When we fit a regression model, we need to create various plots of the residuals and it is done using `plot()` which actually employs `plot.lm()` in the background to do the work. The family of commands are referred to as "methods" and the types of objects being worked on are called "classes". We need a little more background before diving into the various new commands BrailleR offers.

5.2.1 Methods and classes

Development of the `BrailleR` Package and discussion of its opportunities is totally reliant on two of the structures used in the S language. These are "methods" and "classes". In brief, a set of commands that perform a similar task on a selection of different classes is a "method".

Methods and classes are important ideas because we can write a method function that says how we want an object with a stated class to be processed. Methods need classes, and we wouldn't need classes if we didn't hope to use methods.

When we create a histogram with the `hist()` command we can store an object of class "histogram", and when we create a regression model using `lm()` we

create an object of class "lm". The `hist()` command does create an object with the class attribute set to `histogram`, but only one specific function exists for this class, that being the `plot.histogram()` function to plot the histogram. Sighted users don't need an explicit print function for a histogram, nor does this summary graph need further summarisation.

Tasks that warrant a method being written for each of a variety of classes include:

- we might need a function to print the object out in an easy to use fashion. This happens all the time, but most R users just take it for granted that the output looks the way it does. In fact, the output is formatted behind the scenes.
- we may need to plot the results in a graph. A simple `plot()` command does all sorts of different things. Novice users just watch the magic without asking how it happens.
- we may need to create a different kind of object that summarizes the original object in some way. This is already done using the `summary()` function, but that simple `summary()` command is actually a set of functions for different classes.

These are just three tasks common to objects of many classes. There is nothing stopping any R user from writing a method to handle a variety of classes or adding to the methods that already exist.

A complete method will have a base function that informs the software that there is a family of functions written for different classes, and that a method has been written for the default action, which is applied if no specific method exists for a class. For example, the print method includes specific functions: `print()`, `print.default()`, `print.matrix()`, and many more. If we issue the command `print(x)`, and we know that `x` is a matrix, the `print()` method will employ the `print.matrix()` function to display the matrix. The example given in Chapterthat printed the results of the histogram object used the `print.default()` function as there is no `print.histogram()` function in existence.

Many R commands do lead to an object being created with a class attribute being explicitly stated, sometimes it is more implicit, but all too commonly no class attribute is established at all. The vast majority of statistical models set a class for the model being stored. There are some quite uninformative classes set as well. Assigning a class to an object means that we can write functions that relate to all objects of a particular class using a general approach. A class therefore needs to be defined for a set of objects that are going to be homogeneous in their structure.

Data is usually stored with a specified class attribute, such as a time series with class "ts" for example. We will generate different results from employing methods if we have correctly specified the data using a class attribute. We can also modify a data object's class, using commands like `as.tx()` to turn a vector of numbers into a time series if we need to do so to get the desired outcome.

A data.frame is itself of class "data.frame", a matrix is of class "matrix", but rather confusingly, a vector is not of class "vector". Vectors are assigned class attributes that depend on the type of data being stored, being "integer", "numeric", "logical", "character", etc.

As previously stated, the usefulness of methods is dependent on the use of classes being employed when objects are created. The original `hist()` command does specify the resulting histogram to have a class, but there is no explicit `print.histogram()` method at this time. In addition, not all objects are given a class so the default method must be constructed carefully. There are actually only a few basic data structures to work with, the easiest and most common of which is called a "list".

The results of applying the `print()` command to a histogram shown in Chapter 4, couldn't use the non-existent `print.histogram()` function so it used the `print.default()` method instead. The content of the object is stored as a "list", but note the last element of the list that states the class of the object. Adding this extra attribute to the data object is a minor matter that has very powerful consequences!

5.2.2 Who cares about classes anyway?

It is a reasonable question to ask. BrailleR cares because the functions written such as the `VI()` command used throughout Chapter 4 is actually a family of commands. The beginnings of the `BrailleR` Project were formed on the idea of writing a method that would provide the summarised text version of the graph object created. To this end, a method was started with the functions: `VI()`, `VI.default()`, and `VI.histogram()` in order to demonstrate the approach. The `VI()` commands called actually refered to the `VI.histogram()` and `VI.data.frame()` commands to generate output that is sensitive to the object of interest.

So for the `VI()` command to do the processing necessary to extract the information that is pushed into a graphic or textual output, we need to know what kind of object was being created. For the examples shown in Chapter 4, that was done with the standard R commands used when creating the histogram and the data.frame we used. Well that's almost true. The standard `hist()` command from the base distribution of R does assign the class "histogram" to the stored object, but it doesn't have all the necessary information in it to replicate a plotted histogram. The solution is to create a new `hist()` command in the `BrailleR` package that does all the work of the original function and does add the details we want to help describe the histogram being plotted.

5.3 Example: A histogram

One of the easiest ways to demonstrate code snippets is to include them in the help documentation of the function. Running these examples is then possible using the `example()` command. In this example, we see that use of the original `hist()` from the `graphics` package yields the same graph as the `BrailleR` package version, but that the additional text for such items as titles and axis labelling used in the text description are only added by `BrailleR::hist()`. Running the command, `example(hist)` command will give you the following:

```
> x = rnorm(1000)

> MyHist = graphics::hist(x, xlab = "random normal values",
+     main = "Example histogram (graphics package)")

> MyHist
$breaks
 [1] -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5

$counts
 [1]   1   5  24  48 102 147 186 208 120 102  36  13   6   2

$density
 [1] 0.002 0.010 0.048 0.096 0.204 0.294 0.372 0.416 0.240 0.204 0.072 0.026
[13] 0.012 0.004

$mids
 [1] -3.25 -2.75 -2.25 -1.75 -1.25 -0.75 -0.25  0.25  0.75  1.25  1.75  2.25
[13]  2.75  3.25

$xname
[1] "x"

$equidist
[1] TRUE

attr(,"class")
[1] "histogram"

> MyHist = hist(x, xlab = "random normal values", main = "Example histogram (BrailleR package)")

> MyHist
$breaks
 [1] -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5

$counts
 [1]   1   5  24  48 102 147 186 208 120 102  36  13   6   2
```

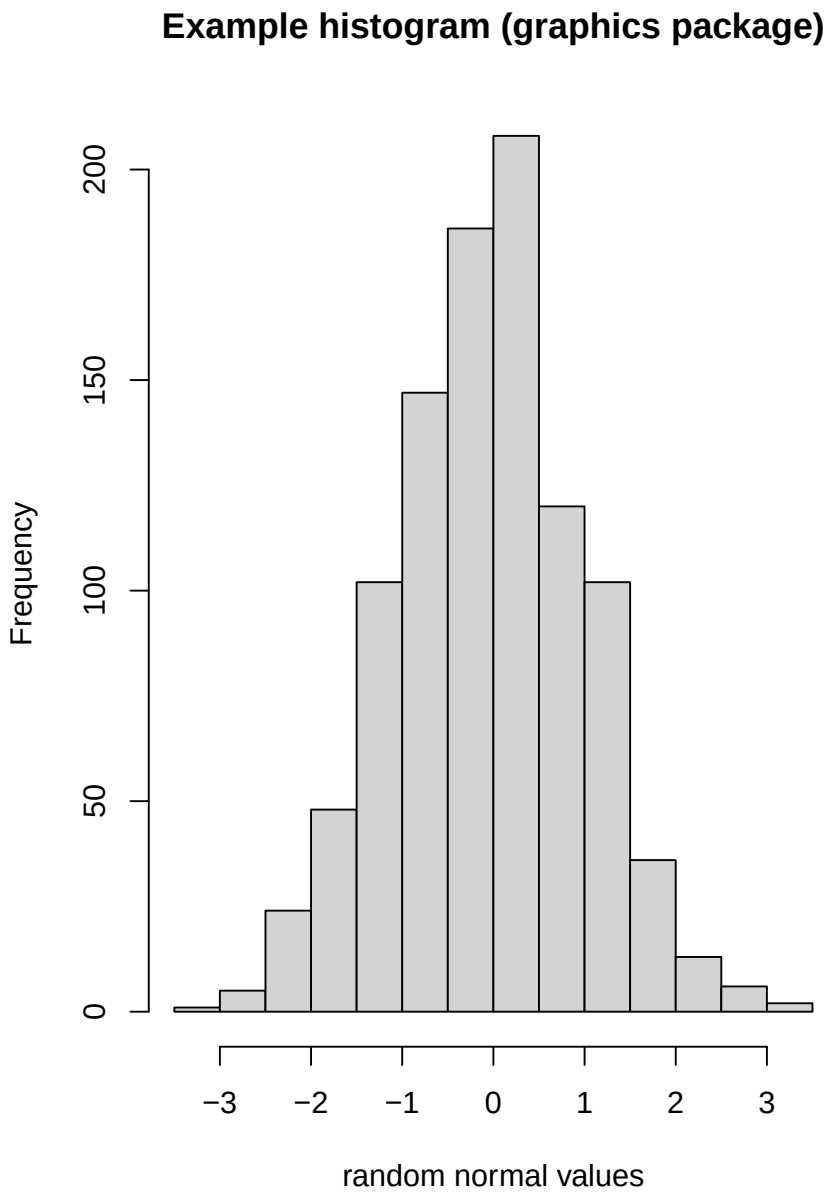


Figure 5.1: testing examples

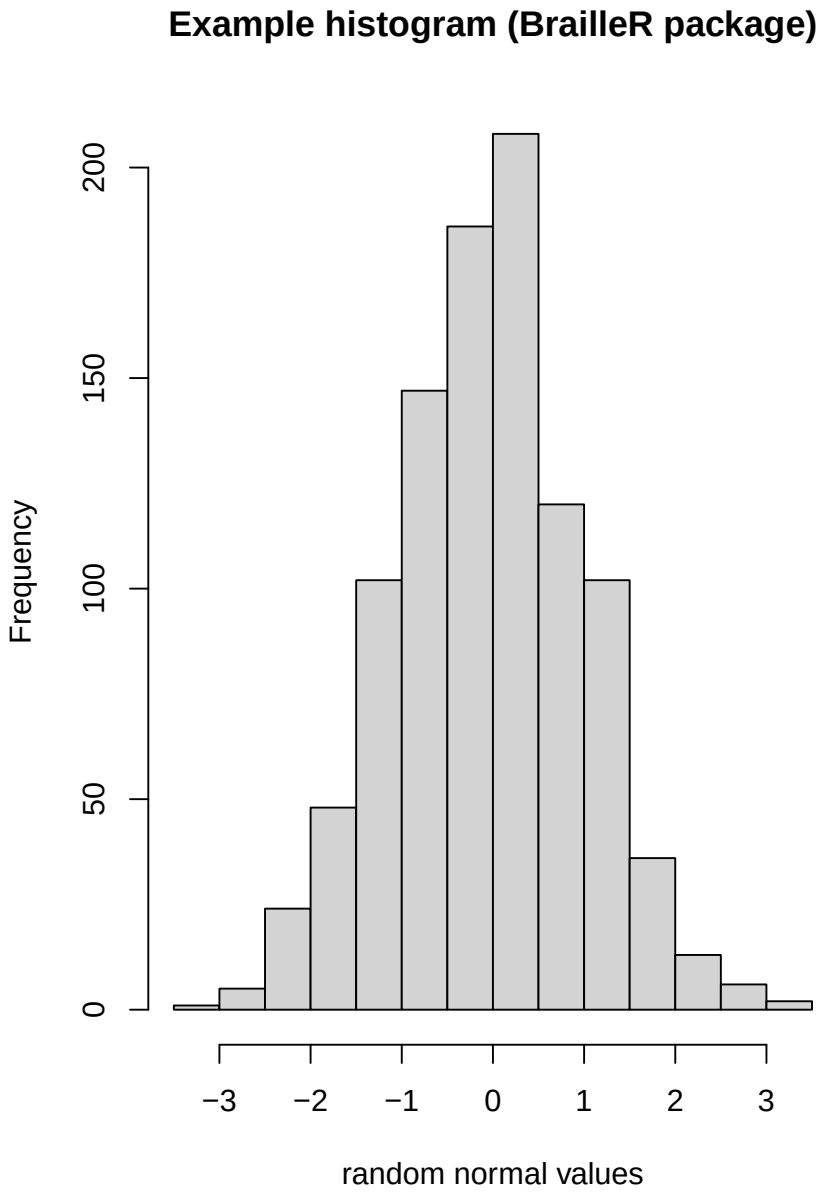


Figure 5.2: testing examples

```
$density
 [1] 0.002 0.010 0.048 0.096 0.204 0.294 0.372 0.416 0.240 0.204 0.072 0.026
[13] 0.012 0.004

$mids
 [1] -3.25 -2.75 -2.25 -1.75 -1.25 -0.75 -0.25  0.25  0.75  1.25  1.75  2.25
[13]  2.75  3.25

$xname
[1] "x"

$equidist
[1] TRUE

$main
[1] "Example histogram (BrailleR package)"

$xlab
[1] "random normal values"

$ExtraArgs
$ExtraArgs$main
[1] "Histogram of x"

$ExtraArgs$xlab
[1] "x"

$ExtraArgs$ylab
[1] "Frequency"

$ExtraArgs$sub
[1] ""


$NBars
[1] 14

$par
$par$xaxp
[1] -3  3  6

$par$yaxp
[1]   0 200   4
```

```
$xTicks
[1] -3 -2 -1  0  1  2  3

$yTicks
[1]   0  50 100 150 200

attr(,"class")
[1] "Augmented" "histogram"

> VI(MyHist)
This is a histogram, with the title: Histogram of x
"x" is marked on the x-axis.
Tick marks for the x-axis are at: -3, -2, -1, 0, 1, 2, and 3
There are a total of 1000 elements for this variable.
Tick marks for the y-axis are at: 0, 50, 100, 150, and 200
It has 14 bins with equal widths, starting at -3.5 and ending at 3.5 .
The mids and counts for the bins are:
mid = -3.25  count = 1
mid = -2.75  count = 5
mid = -2.25  count = 24
mid = -1.75  count = 48
mid = -1.25  count = 102
mid = -0.75  count = 147
mid = -0.25  count = 186
mid = 0.25  count = 208
mid = 0.75  count = 120
mid = 1.25  count = 102
mid = 1.75  count = 36
mid = 2.25  count = 13
mid = 2.75  count = 6
mid = 3.25  count = 2
```

When you first issued the `library(BrailleR)` command, there were several warnings printed out. One of them told you that the `hist()` function from the `graphics` package was masked by the `BrailleR` version. This means that when you use `hist()`, it is the `BrailleR` version being used.

the `BrailleR` package includes `hist()` and `boxplot()` functions that pass the details of the command on to the `graphics` package functions of the same name, and then add any additional content required that will improve the ability to describe the visual graphic produced in text. In most cases, the graph producing functions pass on arguments such as `main`, `xlab`, or `ylab` (for main title and axis labels) to the relevant plotting commands without storing these elements in the object that is created. These elements are stored as graphical parameters and can be recalled using `par()` commands.

5.4 Scatter plots

The description of the `hist()` function given above shows what is possible if a graph is created using a specific function. Many types of graphs are created using the `plot()` function which is actually a family of functions tailored to the type of object pushed into them. In addition, the `plot()` command is used to generate a simple scatter plot. This is slightly unfortunate in a theoretical sense, but useful in a practical sense. The use of `plot()` to generate a scatter plot cannot lead to a graph that the `VI()` functionality can work with. Unlike the `hist()` command which can be replaced by a function of the same name in the `BrailleR` package, the solution needs to be a new function of a new name. In addition to the new `ScatterPlot()` function, the `BrailleR` package has a `FittedLinePlot()` function that adds a fitted line to the scatter plot.

The example given on the help page for `ScatterPlot()` proves that the plots generated by `ScatterPlot()` and `FittedLinePlot()` are identical to those that would normally be created using `plot()` and the addition of the fitted line using `abline()`. Running the command, `example(ScatterPlot)` command will give you the following:

```
> attach(airquality)

> op = par(mfcol = c(3, 2))

> plot(x = Wind, y = Ozone, pch = 4)

> test1 = ScatterPlot(x = Wind, y = Ozone, pch = 4,
+     base = TRUE)

Warning in plot.window(...): "base" is not a graphical parameter

Warning in plot.xy(xy, type, ...): "base" is not a graphical parameter

Warning in axis(side = side, at = at, labels = labels, ...): "base" is not a
graphical parameter

Warning in axis(side = side, at = at, labels = labels, ...): "base" is not a
graphical parameter

Warning in box(...): "base" is not a graphical parameter

Warning in title(...): "base" is not a graphical parameter

> test1

> plot(x = Wind, y = Ozone)

> abline(coef(lm(Ozone ~ Wind)), col = 4)

> test2 = FittedLinePlot(x = Wind, y = Ozone, line.col = 4,
+     base = TRUE)
```

```
Warning in plot.window(...): "base" is not a graphical parameter

Warning in plot.xy(xy, type, ...): "base" is not a graphical parameter

Warning in axis(side = side, at = at, labels = labels, ...): "base" is not a
graphical parameter

Warning in axis(side = side, at = at, labels = labels, ...): "base" is not a
graphical parameter

Warning in box(...): "base" is not a graphical parameter

Warning in title(...): "base" is not a graphical parameter

> test2
```

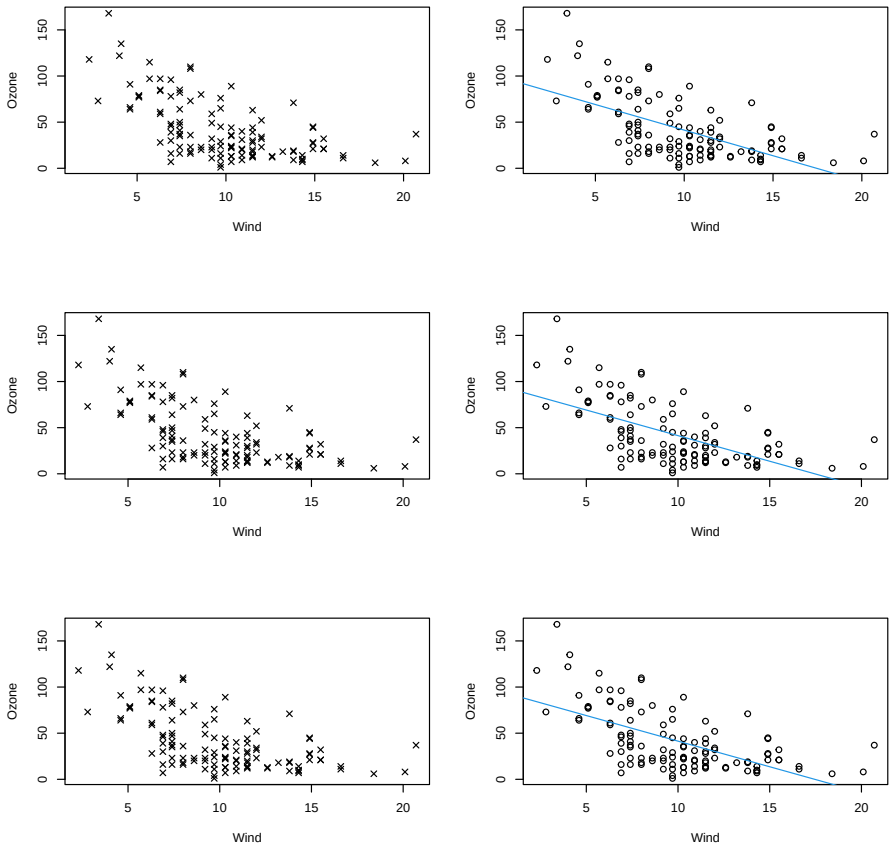


Figure 5.3: The six graphs generated by the `example(ScatterPlot)`

```
> par(op)
```

```
> detach(airquality)
```

5.4.1 What's in the scatter plot?

Well first we might ask what is in the graph window to be confident that a plot was actually made. The `WTF()` command was put in `BrailleR` to address this problem. For the record, WTF is the acronym for "What's this figure?" We'll see it's use in the next figure. It should tell us what appears in the graph window for things like axis lables and titles.

The current solution offered by the `BrailleR` package for helping describe the points plotted in a scatter plot, is to attempt to replicate the summarisation done by sight using a text construct. A sighted person looking at a scatter plot might look at the trend being displayed by a set of points, but they might as easily partition the plot area into a grid pattern and recognize the density of points in each region. For example,

```
attach(airquality)
plot(Wind, Ozone, pch = 4)
abline(v=min(Wind)+c(0.25,0.5,0.75)*(max(Wind)-min(Wind)), col=6)
abline(h = min(Ozone, na.rm=TRUE) +c(0.25,0.5,0.75)*(max(Ozone, na.rm=TRUE)-min(Ozone,
WTF()
```

```
This graph has no main title;  and o subtitle;
"Wind" as the x axis label;
"Ozone" as the y axis label;
There are 116 points marked on this graph.
```

```
detach(airquality)
```

Counting the number of points falling into each cell of the graphic and presenting the results as a table would give the reader an impression of the density of those points. The grid lines added in this last figure were spaced uniformly, and chosen to split the region into a 4×4 grid of sixteen cells. Refining the number of cells and the distributional assumptions for the grid lines should make it easier to understand the relationship between the two variables being plotted. Note that the general picture is what is sought, not the specific locations of every point. The number of points for the last scatter plot are counted by the `WhereXY()` command as follows:

```
attach(airquality)
WhereXY(Wind, Ozone, grid = c(4,4))
```

```
     1  2  3 4 Sum
4    2  0  0 0   2
3    7  5  0 0  12
2    9 15  6 0  30
```

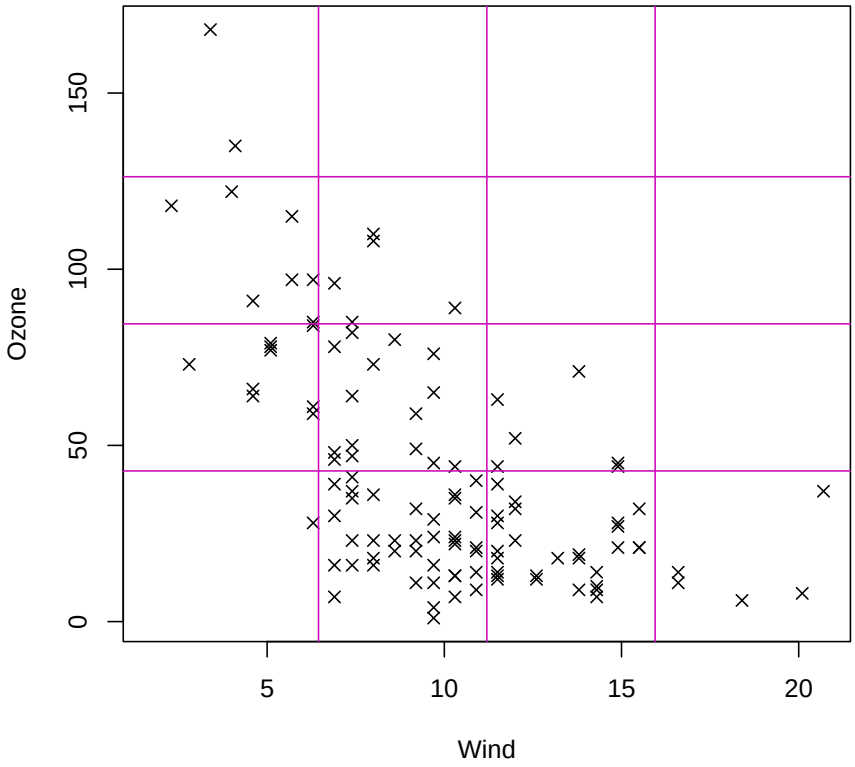


Figure 5.4: Scatter plot of Ozone versus Wind with grid lines added.

```
1    1 39 27 5  72
Sum 19 59 33 5 116
```

```
detach(airquality)
```

In situations where the user is confident that the marginal distributions of either or both of the variables being plotted are normal, the grid lines should be spaced accordingly.

5.5 BrailleR commands used in this chapter

The BrailleR versions of the `hist()` and `boxplot()` commands replace those found in the `graphics` package. The BrailleR commands `ScatterPlot()` and `FittedLinePlot()` are specific to BrailleR and replace the functionality usually obtained through use of `plot()` and `abline()`. The `WTF()` command helped prove a graph has appeared in the graph window, and `WhereXY()` has helped count the number of points falling into subregions of the plotting area.

Chapter 6

Ways of Working in R as a Blind User

This chapter presents some strategies to offer blind users options for producing and saving work from an R session. They complement the workflows used by sighted people, or replace the tools used by sighted people that are not able to be used by a blind person.

6.1 A little background

A major issue for blind users of any mathematical or statistical software is how the work that has been done will be transferred into a form that can be included in reports or assignments. Standard methods of working with R are possible as a blind user, but there is one crucial task that sighted users find very easy that is beyond the reach of the blind user. A sighted user can highlight a section of the output window (including commands and results) and copy the text into a document using the mouse. This task is done easily in word processing or text documents using keyboard commands by the blind user, but is often impractical or impossible within many software applications. A solution has been provided within the `BrailleR` package by adapting a tool developed by another R user for a completely different purpose. The `TeachingDemos` package (Snow, 2020), provided this tool. The original purpose was to quickly retain the output from an R session for distribution to students, and later in its development, to create output files (in MS Word or HTML) that would log the progress through an assignment question or project.

6.2 Using plain text files

The code for saving the basic text output and commands being generated during an interactive session into a plain text file was among the early developments of the `BrailleR` package. The functions `txtStart()`, `txtStop()`, `txtComment()`, and `txtSkip()` were copied (with permission) from the `TeachingDemos` package; in addition, a `txtOut()` function that simplifies use of `txtStart()` was added to the `BrailleR` package.

6.3 Use of R markdown

The general uptake of reproducible research ideas by many in the R community has vastly improved the opportunities for novice R users to create well-formatted HTML documents from markdown documents. This change in the general mindset has huge benefits for a blind user. An R markdown file is just plain text and is therefore accessible. Even more impressive, is that the HTML documents that result from these files are also very accessible to a blind reader. I discovered the potential for R markdown to make life so much easier for blind students by attending a workshop delivered by Yihui Xie at the 2014 UseR conference in Los Angeles. His book (Xie, 2015) serves as the guide by which I create a lot of R markdown content including this book.

The greatest challenge for blind R users who want to get into R markdown is that the majority of resources on the subject assume use of RStudio (RStudio, 2018) which is not accessible to a screen reader user. Use of RStudio is not essential. For example, a basic R markdown document might start like the following:

```
---
title: ""
author: ""
date: ""
output:
    html_document:
        toc: false
        number_sections: false
        fig_height: 7
        fig_width: 7
---

 ```{r ChunkName}
...
```
```

This text needs to be saved in a text file, usually with the file extension “Rmd”. It has two parts. The top section, starting and finishing with , is a YAML

header and establishes settings for the document. You will need to fill in the gaps between double quotes for the document's title, your name, and the date, but otherwise you can leave this section alone for the time-being.

The second section is an empty R chunk. The opening line starts with three accent grave symbols (sometimes called backticks) and then some details used only for this chunk. The details in this chunk header say to process the upcoming commands using R and that the chunk currently has the label "ChunkName" which you would normally change to something more meaningful. The second line of this section is where you would put an R command (instead of the meaningless dots put there now), and the last line is how we close a chunk (using those backticks again). You can put a whole series of R commands in the middle, but the chunk must start and finish with ``` if the document is to be processed properly.

To get this R markdown content to be converted to HTML, you must complete the following steps:

1. Save the above content into a text file and name it "FirstGo.Rmd". N.B. This must be plain text.
2. Edit the bits I suggested above. In fact, the bare minimum is to put a valid R command in the middle of the R chunk.
3. Put this file in the current working directory. Check what this is using `getwd()` if you need to be sure.
4. Type the command `rmarkdown::render("FirstGo.Rmd")`

You should see that this command generates a little output in the R session window and that a file called "FirstGo.html" has been created. Open that file now.

The basic steps followed by all users of R markdown files are:

1. Update the Rmd file.
2. "Knit" the R markdown file; The word "knit" comes from the R package that does the work here.
3. Review the impact on your HTML file. You might need to refresh the browser if the file is still open.

You can:

- add more R chunks; give them distinct labels though.
- Write plain text in between chunks;
- Add section headings; use number signs `#` at the start of a line that you want to be a heading. The level of heading is determined by the number of number signs you use.
- Add mathematical expressions using standard LaTeX notation; single or double dollar signs denote the start and end of mathematical content.
- ... and so much more.

Remember: This book is written entirely in R markdown. If you see something here, then you can have it in your own R markdown work!

We will see the power of R markdown for doing some tasks using the `BrailleR` package in Chapter 7 and start using WriteR as a substitute for RStudio in Chapter 11.

6.4 Running jobs offline

Blind users will benefit from switching to batch processing commands using `R CMD BATCH` at the command line, using the reproducible research functionality offered by the `knitr` package, or both strategies. Functions to help users (working under Windows operating systems) move to these ways of working have been included in the `BrailleR` package. In particular, a Windows user can use the `MakeBatch()` function to create a batch file in the current working directory, which creates the single command line that would be called to process a specified R script or R markdown file appropriately. The `BrailleR` package also shows a user how a test file would be processed using these batch files. Conversion of an R script or the history of the current workspace to an R markdown file are implemented using the `R2Rmd()` and `History2Rmd()` functions respectively. This suite of functions should assist the blind user migrate to the more efficient methods of working and ultimately become more proficient and efficient than their peers who are not yet making use of the reproducible research type of workflow.

Chapter 7

Use of R markdown to generate an analysis efficiently

In a general sense, R markdown has been used to create reports and package vignettes because it creates an analysis that is reproducible. The `BrailleR` package started to use R markdown in late 2014 as a method for generating simple analyses that might be needed by students taking introductory statistics courses. Since that time, the prevalence of R markdown as a teaching tool in these courses has increased. The functions described below are therefore also generating example R markdown files to help learn how to use R markdown. The workflow illustrated in this chapter is also the workflow used to create this book. It is heavily reliant on the work of Yihui Xie, much of which can be found in (Xie, 2015).

You will need the `BrailleR` package to be ready for use to follow along with the examples in this chapter. Do this by issuing the command `library(BrailleR)` now.

7.1 General information

Each command described in this chapter and other similar commands draft a new R markdown file and then compile it to an HTML file that is easily read by a screen reader user. They make extensive use of the `knitr` package (Xie, 2021) to process R commands and retrieve the R output and graphs, and then the `rmarkdown` package (Allaire et al., 2021) to process the markdown content of the raw input file into HTML (or other file types if we wanted them).

This HTML file is opened automatically if R is being used interactively, giving the blind user immediate access to the information. The content is automatically presented using sufficiently marked up HTML code including headings and tables so that the blind user can make best use of their screen reading software. All graphs are given an "alt tag" when they are included in the HTML file, and can be presented using a text description available from the `VI()` functionality of the `BrailleR` package.

In addition, the blind user may need one or more of the graphs in a variety of formats (png, pdf, eps, or svg), nicely formatted tables for insertion into documents (LaTeX or HTML), and access to the code that generated these graphs and tables (an R script). This is handled using add-on packages wherever possible so that blind users are completing tasks using the same tools as their sighted peers. For example, nicely tabulated results can be saved as individual text files for later inclusion in LaTeX documents using the `xtable` package (Dahl et al., 2019).

The commands shown in this chapter make use of R markdown, but they are not actually ready for direct use within other R markdown documents. If you wish to get the output you observe within the HTML documents that will be generated, you will need to extract the relevant parts of the R markdown script files that the commands create. That sounds harder than it is!

7.2 Replacing the Graphic User Interface (GUI)

One major criticism of R often heard from novice users is that a graphic user interface (GUI) is much easier to use than the command line mode of operation. Blind users are no different in this respect, but as has been reported previously (Godfrey, 2013b), none of the GUI-based front ends for R are accessible to the blind user. When considering the benefits of using a GUI mode of operation, we turn to the justification used for the creation of some of these interfaces; redacting all references to the GUI helps emphasize the reasons for their existence:

- R Commander aims "to support ... the statistical functionality required for a basic-statistics course; to make it relatively difficult to do unreasonable things; and to render visible the relationship between choices made ... and the R commands that they generate." (Fox, 2005).
- "Because R analyses must be called as text commands, the user is required to find out the name of the function that will accomplish their task, and then remember that name along with the names of the variables to feed it, and its argument options. Perhaps more fundamentally, many users have never dealt with a program that requires them to type in commands that manipulate objects in the program. For beginners, Deducer is designed to be an intuitive dialog based interface to common data manipulation and analysis tasks." (Fellows, 2012)

- "The scope of RKWard is deliberately broad, targeting both R novices and experts. For the first group, the aim is to allow any person with knowledge on statistical procedures to start using RKWard for their everyday work without having to learn anything about the R programming language, at least initially. At the same time, RKWard tries to support users who want to learn and exploit the full flexibility of the R language for automating or customizing an analysis." (Rödiger et al., 2012)
- RKWard "avoids wrapping complex sequences of data manipulation or analysis into custom highlevel R functions. The task of providing high-level functions is logically independent of the development of the GUI frontend, and should best be addressed in dedicated R packages, where necessary." (Rödiger et al., 2012)

Making any of these GUI tools accessible is difficult because they were not developed using interface development toolkits that automatically build accessibility into the interface. Creation of a GUI such as R Commander is not a small undertaking. Retrospectively building in the necessary accessibility features would be a massive undertaking that the blind community could not reasonably expect of a very small development team. Duplicating the work done to create such a GUI tool from scratch using a development toolkit that helps build in the necessary elements for accessibility and then maintaining it as operating systems develop over time is not a practical solution either. It may be possible to improve the toolkits that are used to create the GUI options like R Commander or Deducer so that their development is not hindered while their accessibility becomes automatic; unfortunately, this requires knowledge well beyond my skill set.

If blind users cannot make use of any of the GUI tools and we don't have the necessary skills to develop such a tool that does work for ourselves, what solutions exist for us? When I've discussed the merits of various statistical software with other blind users, they have questioned the use of R by saying something like, "When I use software x, I can do y and z. How do I do that in R?" With respect to SAS for example, a user can build up a set of templates for various commonly used analyses for themselves. The same is true as an R user, but SAS users can get the templates by using the menus in the standard GUI, and they can keep the procedures in files for later use. SPSS users can do this as well. (These seem to be the most commonly used software options for blind users who communicate with me.) In either case, the code can be reused after issuing a few find/replace searches. It seems that even though it is more efficient to make use of this ability to re-use code templates, that many of the blind users of these software options prefer to use key presses to simulate mouse clicks in order to use the menus and dialogue boxes. When questioning them on their motives, we quickly come to the point that the use of menus and dialogue boxes offers a degree of comfort that tasks have been done correctly. The next crucial aspect worth mentioning is that the GUI provides the user with more output than can be obtained from typing out numerous commands in R. Searching for the right functions and then ensuring the syntax of their arguments is correct limits a blind user's ability in

any command-line software, just as it does for sighted users.

The solution for a replacement of the GUI is to create convenience functions with simple names that require entry of a minimal number of arguments so that the user gets the maximum amount of useful information. This information might prove more than is needed for many users, but hopes to deliver what is needed by the widest possible range of blind users. Like RKWard, `BrailleR` functions generate R scripts that are somewhat verbose (Rödiger et al., 2012).

7.3 Description of a single numeric variable

There are many commands needed to get the numeric and graphic summary measures that might be required to collect all relevant information on a single numeric variable. The `UniDesc()` command has been written as a shortcut for a blind user who wishes to obtain:

- the counts of points in the sample that were observed and not observed,
- the mean and trimmed mean,
- the five number summary: minimum, lower quartile, median, upper quartile, and maximum,
- the interquartile range (IQR) and standard deviation,
- measures of skewness and kurtosis, relying on the `moments` package (Komsta and Novomestky, 2015),
- a histogram and/or a boxplot,
- a normality (quantile-quantile) plot,
- various tests for normality, courtesy of the `nortest` package (Gross and Ligges, 2015) , and
- tests on the significance of the skewness and kurtosis, also courtesy of the `moments` package (Komsta and Novomestky, 2015).

The `UniDesc()` function can deliver all of this with minimal effort from the user.

An example of the main output document (HTML) can be viewed by re-issuing the commands generated by calling

```
example(UniDesc)
```

while running R interactively. This issues the following commands.

```
DIR = getwd()
setwd(tempdir())
Ozone=airquality$Ozone
UniDesc(Ozone)
rm(Ozone)
# N.B. Various files and a folder were created in a temporary directory.
# Please investigate them to see how this function worked.
setwd(DIR)
```

As an alternative, and if you do have a current internet connection you can view the result of running the `UniDesc()` command on the Ozone data in your browser without having to re-enter the example commands. You can also view the R markdown script created by `UniDesc(Ozone)` which starts with the following lines:

```
# Univariate analysis for Ozone
#### Prepared by Jonathan Godfrey

```{r setup, purl=FALSE, include=FALSE}
library(BrailleR)
opts_chunk$set(dev=c("png", "pdf", "postscript", "svg"))
opts_chunk$set(comment="", echo=FALSE, fig.path="Ozone/Ozone-", fig.width=7)
```

Basic summary measures

```{r BasicSummaries}
Ozone.count = length(Ozone)
Ozone.unique = length(unique(Ozone))
Ozone.Nobs = sum(!is.na(Ozone))
Ozone.Nmiss = sum(is.na(Ozone))
Ozone.mean = mean(Ozone, na.rm = TRUE)
Ozone.tmean5 = mean(Ozone, trim =0.025, na.rm = TRUE)
Ozone.tmean10 = mean(Ozone, trim =0.05, na.rm = TRUE)
Ozone.IQR = IQR(Ozone, na.rm = TRUE)
Ozone.sd = sd(Ozone, na.rm = TRUE)
Ozone.var = var(Ozone, na.rm = TRUE)
Ozone.skew = moments::skewness(Ozone, na.rm = TRUE)
Ozone.kurt = moments::kurtosis(Ozone, na.rm = TRUE)
```
... ...
```

The full R markdown file can be opened in any text editor. A Windows user can open the file created by the `example(UniDesc)` command above, by issuing the command `Notepad("Ozone-UniDesc.Rmd")` at the R prompt. Opening the Windows Explorer file browser will help you select an alternative text editor; you can do this by typing `Explorer()`.

The document header includes a code chunk that sets the options for the `knit2html()` process used to convert this file to HTML. This processing allows for:

- multiple file types for the graphs,
- ensures a minimum of extra text in the output,
- hides the R code from the HTML file,
- sets the location of the saved graphs, and
- establishes the height of the figures.

The user that does not like these settings can edit the markdown file for themselves and re-process the file, but the intention is to deliver more than all users would want so that as many users as possible get what they need.

This R markdown script uses the `VI()` method for the graphs as well as the code that generates the HTML (via markdown) and LaTeX tables (using the `xtable` package). Take note of the arguments supplied to the code chunks for the graphs; these include a `fig.cap` which is used as an Alt tag in the resulting HTML files. The quoted string is the only text that is read aloud by a blind person's screen reading software as they move the cursor onto the graph while reading through the HTML document. The methods used for creating tables in markdown (either directly or using the `kable()` command from the `knitr` package) (Xie, 2021) both lead to a formatted HTML table that is easily navigated by a screen reader using keystrokes that help move between rows or columns. The result is that the HTML document is about as user-friendly as can be expected for a blind user. It is important to recognize that some of the text is arranged for the optimal use by a blind person; it is possible to alter the cosmetics of the HTML document without altering the experience for blind users. Given a blind user might be accompanied by sighted classmates, teaching staff, or colleagues, making the HTML pages presentable is desirable; this has been achieved using a custom style sheet (CSS) included in the package. This style includes some color, adequate spacing of content and other features that improve the clarity of the presented material which is important for a user with some residual vision that wishes to read through the results visually rather than using screen reading software.

If all of the optional arguments of the `UniDesc()` function are set at their defaults, the HTML file is automatically opened in a browser (courtesy of the `View` argument); it puts the R markdown file, the R script, and the HTML document in the current working directory, while the LaTeX and graph files are all placed in a subdirectory.

Several other convenience functions have been created that follow the same process as the `UniDesc()` function. The `OneFactor()` function compares one continuous response variable to a categorical variable, while `TwoFactors()` allows for two categorical variables and their possible interaction to help explain a single continuous response variable. These functions create group summary statistics and suitable graphs that a sighted audience might expect to see. Comparing a continuous response variable to a continuous predictor variable is achieved using the `OnePredictor()` function. Each of these functions has an example using data from the `datasets` package (R Core Team, 2021) so a user can see what can be expected from these functions.

7.4 Analysis of a single continuous variable with respect to a single grouping factor

There are many commands needed to get the numeric and graphic summary measures that might be required to collect all relevant information on a single numeric variable when it might depend on a grouping factor. The `OneFactor()` command has been written as a shortcut for a blind user who wishes to obtain:

- the counts of observations within each group,
- the mean, standard deviation and standard error for each group,
- comparative boxplots and/or dotplots,
- the one-way analysis of variance, and
- Tukey's Honestly Significant Difference (HSD) test on the significance of the between group differences.

There will be circumstances when the last, and perhaps evern second to last elements of this functionality are not justified by the outcome of issuing the earlier commands. In particular, generating the output from Tukey's HSD would not normally be sought unless the analysis of variance showed there was a significant difference between the group means. The choice of reading the output and subsequently using it are left to the user. In this respect, `BrailleR` has not become an expert system that decides to or not to provide something; the user is still expected to do the thinking about the relevance of each element of hte overall set of output provided.

An example of the main output document (HTML) can be viewed by re-issuing the commands generated by calling

```
example(OneFactor)
```

while running R interactively. This issues the following commands.

```
DIR = getwd()
setwd(tempdir())
data(airquality)
library(dplyr)
# the following line  returns an error:
## OneFactor("Ozone", "Month", airquality, View=FALSE)
# so we make a copy of the data.frame, and fix that:
if(require(dplyr)){
airquality2 = airquality %>% mutate(Month = as.factor(Month))
# and now all is good to try:
OneFactor("Ozone", "Month", airquality2)
# N.B. Various files and a folder were created in a temporary directory.
# Please investigate them to see how this function worked.
}
setwd(DIR)
```

As an alternative, and if you do have a current internet connection you can view the result of running the `OneFactor()` command on the Ozone data in your browser without having to re-enter the example commands.

The example here demonstrates the point that the grouping variable must be a factor. The month variable is not stored as a factor in the airquality data so its use would have created an error.

All elements of the analysis which incorporate graphical information, or text that is slow to read using a screen reader are processed using the `VI()` functionality.

7.5 Use of BrailleR for linear regression

Linear regression is almost always taught using graphical techniques, especially for the validation of the model being fitted. Of particular note is the way an instructor would teach sighted students about the sensibility of fitting any line to some data which cannot be easily judged even using the `WhereXY()` function described earlier. A blind student lacking an embosser to produce a tactile image that shows the fitted line and the data, will almost certainly need to fit the model and see how good or bad it is. These blind students are therefore even more reliant on the residual analysis than their sighted classmates

The `VI()` command can be applied to a linear model object. The specific function to do this is found in the `VI.lm()` function, but most users do not need to explicitly use `VI.lm()` because the call to `VI()` will know to use the `VI.lm()` function if it is the right one to use at the time.

The `VI.lm()` function generates so much text as a substitute for the graphs used by sighted users, that it is easier to put this text in an HTML document and have that new document opened in a browser instead of trying to use a screen reader within the R session.

Let's see an example using the `airquality` data. Please note: The example is chosen for reproducibility and its lack of statistical validity, as this is the best way to demonstrate the function's value to a blind user.

A simple linear regression model might be created and investigated using:

```
data(airquality)
MyModel = lm(Ozone~Wind, data=airquality)
summary(MyModel)
```

```
Call:
lm(formula = Ozone ~ Wind, data = airquality)

Residuals:
    Min      1Q  Median      3Q     Max
-51.572 -18.854  -4.868  15.234  90.000
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  96.8729     7.2387   13.38  < 2e-16 ***
Wind         -5.5509     0.6904   -8.04 9.27e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 26.47 on 114 degrees of freedom
  (37 observations deleted due to missingness)
Multiple R-squared:  0.3619,    Adjusted R-squared:  0.3563
F-statistic: 64.64 on 1 and 114 DF,  p-value: 9.272e-13
```

```
par(mfrow=c(2,2))
plot(MyModel)
```

The user now has a model stored as `MyModel` in the current workspace, has printed a summary of that model, and has plotted a set of four diagnostic plots in a 2×2 grid. The blind user will still need to issue those commands so that the output is created to meet the expectations of the sighted audience, but will also find value in issuing the two extra commands

```
VI(MyModel)
VI(summary(MyModel))
```

The use of the second of these commands will generate

```
The term which is significant to 1% is
Wind with an estimate of -5.550923 and P-Value of 9.271974e-13
```

which will be a much easier reading exercise for a screen reader user than would be the standard `summary()` output given earlier. Note that not all the information contained in the standard summary is contained in this output.

The output from use of the `VI()` command on the linear model can be viewed in your browser if you have a current internet connection. If you do not have a connection at this time, you will need to re-issue some of the above commands for yourself in an R session.

The HTML document created by `VI.lm()` is based on the results of the `UniDesc()` function applied to the Pearson residuals for the model and a number of other graphs and tables used to validate the model. The included graphs are of the residuals plotted against the fitted values, the order the data were collected (assumes data are presented in this order), the leverages, and the preceding residual. Each of these graphs is subjected to scrutiny using the `WhereXY()` function as described earlier. The marginal distribution of the residuals is assumed normal while the fitted values and leverages are categorized on the basis of a uniform distribution. The assumption of normality for residuals is immediately obvious, but the choice to explicitly use an incorrect assumption

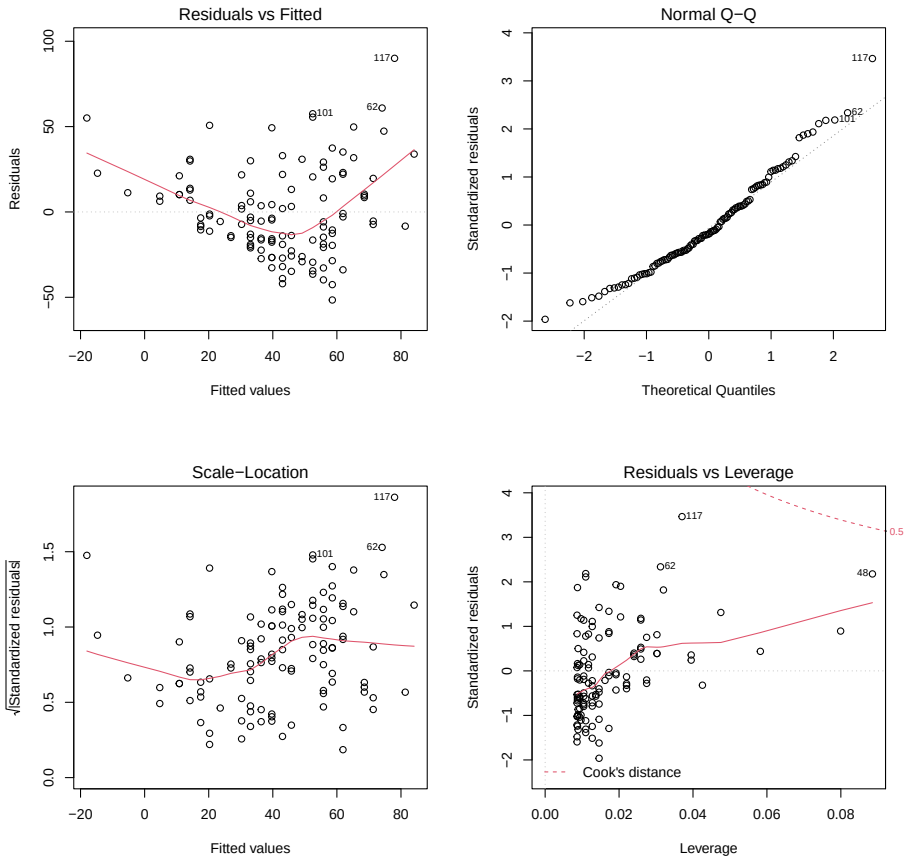


Figure 7.1: (#fig:AirQuality_lm)Diagnostic plots for the simple linear regression model.

of uniformity for the other values needs to be justified. While these quantities aare unlikely to be uniformly distributed, the intended audience needs to know about the pattern of their presentation on the graph; this should be easier to understand if the categorization is done using uniform spacing rather than another distribution that is unfamiliar to the intended audience.

A table of unusual observations is created that uses rules of thumb for magnitude of residuals, leverages, and Cook's distances. This table is presented in the HTML document and converted to a LaTeX file using the `xtable` package. The raw LaTeX for this table looks like:

```
% latex table generated in R 3.5.1 by xtable 1.8-2 package
% Thu Jan 24 21:36:59 2019
\begin{table}[ht]
\centering
\begin{tabular}{rrrrrrr}
  \hline
 & Ozone & Wind & Fit & St.residual & Leverage & Cooks.distance \\
  \hline
9 &     8 & 20.1000 & -14.7007 & 0.8934 & 0.0799 & 0.0347 \\
  18 &     6 & 18.4000 & -5.2641 & 0.4370 & 0.0582 & 0.0059 \\
  22 &    11 & 16.6000 & 4.7276 & 0.2408 & 0.0395 & 0.0012 \\
  48 &    37 & 20.7000 & -18.0312 & 2.2149 & 0.0885 & 0.2304 \\
  62 &   135 & 4.1000 & 74.1141 & 2.3847 & 0.0312 & 0.0880 \\
  86 &   108 & 8.0000 & 52.4655 & 2.1428 & 0.0110 & 0.0247 \\
  101 &   110 & 8.0000 & 52.4655 & 2.2233 & 0.0110 & 0.0265 \\
  117 &   168 & 3.4000 & 77.9998 & 3.6474 & 0.0370 & 0.2309 \\
  121 &   118 & 2.3000 & 84.1058 & 1.3164 & 0.0475 & 0.0430 \\
  126 &    73 & 2.8000 & 81.3303 & -0.3204 & 0.0426 & 0.0023 \\
  148 &    14 & 16.6000 & 4.7276 & 0.3561 & 0.0395 & 0.0026 \\
   \hline
\end{tabular}
\caption{Listing of suspected outliers and influential observations.}
\label{InflObsMyModel}
\end{table}
```

Note that the automatic formatting of this table as performed by the `xtable` package has not been altered to meet a specified publication style. Ultimatly, users will need to alter the presentation to meet publisher specifications for themselves.

7.6 Analysis of a single continuous variable with respect to another continuous variable

The `OnePredictor()` command is similar to the `OneFactor()` command describe earlier in this chapter and makes use of the `VI()` command as applied to the simple linear regression model fitted to a pair of continuous variables, one of which is determined to respond to the other. The `OnePredictor()` command has been written as a shortcut for a blind user who wishes to obtain:

- the counts of observations within each group,

An example of the main output document (HTML) can be viewed by re-issuing the commands generated by calling

```
example(OnePredictor)
```

while running R interactively. This issues the following commands.

```
DIR = getwd()
setwd(tempdir())
data(airquality)
OnePredictor("Ozone", "Wind", airquality)
# N.B. Various files and a folder were created in a temporary directory.
# Please investigate them to see how this function worked.
setwd(DIR)
```

As an alternative, and if you do have a current internet connection you can view the result of running the `OnePredictor()` command on the Ozone data in your browser without having to re-enter the example commands.

7.7 BrailleR commands used in this chapter

The first two `BrailleR` commands introduced in this chapter were the `UniDesc()` and `OneFactor()` commands; they used the `VI()` command in the R markdown files that they create, as was described back in Chapter 4, to give the text descriptions for graphs. We then saw a new use of the `VI()` command and several other commands designed to generate common analyses quickly. These included the `OnePredictor()` command which speeds up the presentation of a simple linear regression model. This function also creates an R markdown script which then makes an HTNML file for immediate use, and various files with tables and graphs commonly needed for simple linear regression. The specific command `VI.lm()` is an example of the `VI()` command tailored to linear models and their analysis. Consideration of the validity of linear models is generally done via graphs so the `WhereXY()` command introduced in Chapter 5 is applied in a variety of ways.

Windows users can use the `Notepad()` function to open the R markdown files created by the examples in this chapter, or any other text file for that matter. They can choose their own text editor by opening a Windows Explorer file browser that has focus in the current working directory using `Explorer()`. The initial letter of these commands can be in upper or lower case; that is, `Notepad()` is equivalent to `notepad()` and `explorer()` is equivalent to `Explorer()`.

Chapter 8

Personalising BrailleR

8.1 General

Once you've played with a few examples, you might want to settle on the way you want BrailleR to work for you. There are a wide range of options needed to get the best out of the `BrailleR` package specific to each user, and perhaps for each user who wants specific settings to be in play for different projects. All `BrailleR` settings are stored in a local file, and also in a global file. These files are both called `BrailleROptions`. The global settings file is located in a folder called `MyBrailleR` which is located where you let `BrailleR` choose when you first loaded the package using the `library(BrailleR)` command. You could have let this be a temporary location so you will be asked every time you start `BrailleR` until you let the standard location be used.

The `BrailleROptions` file in the `MyBrailleR` folder will be used unless a local version is found. This local file will be in the working directory being used when the package is loaded.

8.2 Settings that are about you

8.2.1 Who is the R markdown file being written by?

You might want your analyses to use your name instead of the default name `BrailleR`. Do this using the `SetAuthor()` function. e.g.

```
SetAuthor("Jonathan Godfrey")
```

OK, you ought to use your name not mine, but you get the point.

This will have an instant impact, even on the examples for `BrailleR` functions. Set the author and then try `example(UniDesc)` for example.

8.2.2 The use of the VI() command

The `BrailleR` package was intended for use by blind people, but we need to see more in text than do most people in our intended audiences. You may wish to turn off or on the use of the output generated by the `VI()` commands throughout the R markdown files written by such commands as `UniDesc()` etc. Do this using the functions `GoBlind()` to use the `VI()` command, and `GoSighted()` to turn it off.

I think a standard workflow might be to start `BrailleR`, do some analyses using `UniDesc()` or `OnePredictor()` and the like, and then having worked out what was working well, use `GoSighted()` and re-issue the commands that you want to share with others. Don't forget to `GoBlind()` again though so that you can get the text descriptions back when you need them.

8.3 BrailleR commands used in this chapter

We saw the use of the `SetAuthor()` function to personalise the work you are doing to use your name instead of a generic author. The `GoBlind()` and `GoSighted()` functions toggle between providing the additional output needed by blind users to interpret our work. Much of this additional output is not of interest to a sighted audience so changing back and forth should prove useful.

Chapter 9

BrailleR in the tidyverse

Hadley wickham is unquestionably a superstar in the R community, and is perhaps the first R celebrity. There can't be room for too many people to have had a suite of packages collectively named after them by numerous users, but history will show that the "Hadleyverse" existed until Hadley himself renamed it the "tidyverse". The tidyverse owes its prominence to the relative simplicity it offers people doing what should be simple tasks, but haven't been as easy as might have been. According to the tidyverse package (Wickham, 2021) documentation, "The 'tidyverse' is a set of packages that work in harmony because they share common data representations and 'API' design." The package is just a simple way to make sure these packages are all installed and available to the user. Many users will not use all of the packages in the tidyverse, but among my favourites are `lubridate` (Spinu et al., 2021) for handling dates in all manner of formats, `broom` (Robinson et al., 2021) for handling linear models more efficient, `magrittr` (Bache and Wickham, 2020) for giving me an alternative way of writing code, and of course `dplyr` (Wickham et al., 2021b) for making data manipulation and summarisation much easier to explain to others. The `ggplot2` package (Wickham et al., 2021a) is another tidyverse package but it deserves a separate chapter in order to show the way it works with `BrailleR`. For the purposes of showing how `BrailleR` works with the tidyverse packages, or more accurately, the tidyverse way of working, the examples in this chapter all make use of the `dplyr` package and any tools it calls on to support its functionality.

To replicate the examples in this chapter, you will need to have the tidyverse packages installed before running the following commands that prepare them and `BrailleR` for use.

```
library(BrailleR)
library(tidyverse)
```

```
-- Attaching packages ------------------------------------- tidyverse 1.3.1 --
```

```
v ggplot2 3.3.5     v purrr   0.3.4
v tibble  3.1.6     v stringr 1.4.0
v tidyr   1.1.4     v forcats 0.5.1
v readr   2.1.0
```

```
-- Conflicts ----------------------------------------- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

9.1 What is tidy and why do we care?

Wickham (2014b) describes tidy data as following three rules:

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

and then says that data not following these rules is "messy" (Wickham, 2014b) .

We care because tidy data is ready for an analysis, while messy data needs to be made tidy. We care because it is easier to use tools designed for tidy data, and this all means we should get the desired results effectively and efficiently. We care because it is more common for data to be messy than tidy, and we must be able to take messy data and tidy it up.

To further quote Wickham (2014b), the four most commonly used data manipulations (transformation, aggregation, filtering, and reordering) can be easily managed when we start with tidy data. The data manipulations are all performed by the `dplyr` package we use in this chapter. If we cannot work with tidy data successfully, then there is little hope for working with messy data.

A great resource for learning more about the tidyverse and its numerous tools is the R for Data Science book (Grolemund and Wickham, 2016).

9.2 What is the pipe operator, and why should we care?

The pipe operator `%>%`, found in the `magrittr` package, is used throughout the tidyverse because it makes code simpler to read. A series of pipes is referred to as a pipe chain, and it is when there are multiple pipe commands issued in conjunction that its simplicity becomes increasingly obvious.

Before the tidyverse, R users would follow a mixture of two general coding strategies. Either, we nested one command inside another, and perhaps another, and even worse, we'd nest and nest and so on; or, we can have each line of code

have a single function, with the outcome of each function being stored as an explicitly named object. Nesting commands inside one another makes code very hard to read, as we read from the inside outwards to get a handle on what is actually being achieved. Storing each and every element of our working could have memory management implications, but is also prone to having too many named objects floating around that must be kept track of.

In contrast, a pipe chain can be written so that each function is applied in order, left to right, top to bottom, with the answer being stored in a named object at the end if we want, or quite commonly, just printed out for us. Whether you put the individual commands in a single line or with each pipe on its own line is a matter of style and personal preference.

A simple example using the `dplyr` package and the `airquality` data could ask for the coldest and hottest temperatures for each of the five months in this dataset.

```
airquality %>% group_by(Month) %>% summarise(ColdestDay = min(Temp), HottestDay = max(Temp))
```

```
# A tibble: 5 x 3
  Month ColdestDay HottestDay
  <int>      <int>      <int>
1     5         56         81
2     6         65         93
3     7         73         92
4     8         72         97
5     9         63         93
```

In its most simple form, the outcome of everything done to the left of any pipe operator is used as the first argument of the first function to its right. This means that the first argument of the `group_by()` is the `airquality` data.frame we started with. Note that there are ways to use the left-hand-side of the pipe operator as a second, third or so on, argument and even more adventurous ways of piping, but these are not relevant to the presentation of the `BrailleR` tools in this chapter.

The next important note about functions used in pipe chains is that the type of object coming out of the function is the same as the object that was pushed in, although it has probably been modified on its path through the function. For example, the `group_by()` function didn't drastically alter the `airquality` data.frame, but it did add information that has an impact at the next step in the pipe chain; without it, the `summarise()` command would have operated on the entire dataset as a whole without splitting the data into months before applying the `min()` and `max()` functions.

The question is, how can we be sure that what is being passed on at each step is what we expected? In simple cases like that just seen, the answer justifies the work done to that point. Much longer pipe chains are possible, such as:

```
set.seed(123)
starwars %>% filter(!is.na(species)) %>% sample_n(50) %>% group_by(species) %>% summari
```

```
# A tibble: 6 x 3
  species      N MeanHeight
  <chr>    <int>      <dbl>
1 Droid        3       96.5
2 Human       20      175
3 Twi'lek      2      179
4 Gungan       2      201
5 Kaminoan     2      221
6 Wookiee      2      231
```

which takes a random sample of 50 Star Wars characters who each have a defined species, and evaluates the mean height for each species, but only then keeps the species with more than one character and then finally prints out the mean heights of the species in order from shortest to tallest.

This long chain is not likely to feature in anyone's real analysis, but for the purposes of demonstration it has the necessary random selection that could mean results differ from one application to the next. There is therefore uncertainty about what data reduction has taken place at each step in the chain. To prove this for yourself, you could alter the random seed forced with the `set.seed()` command.

9.3 Interrogation of data created within a pipe chain

The `BrailleR` commands `WhatIs()` and `CheckIt()` were designed to stop us from having to curtail our pipe chains. Both commands were designed to use the same syntax as the other `dplyr` commands in the pipe chain, and can be put in the middle or at the end of the chain.

```
set.seed(123)
starwars %>% filter(!is.na(species)) %>% sample_n(50) %>% group_by(species) %>% summari
```

```
tibble [25 x 3] (S3: tbl_df/tbl/data.frame)
 $ species   : chr [1:25] "Aleena" "Cerean" "Droid" "Dug" ...
 $ N         : int [1:25] 1 1 3 1 1 2 20 1 2 1 ...
 $ MeanHeight: num [1:25] 79 198 96.5 112 88 201 175 188 221 188 ...
```

```
tibble [6 x 3] (S3: tbl_df/tbl/data.frame)
```

```
 $ species   : chr [1:6] "Droid" "Gungan" "Human" "Kaminoan" ...
 $ N         : int [1:6] 3 2 20 2 2 2
 $ MeanHeight: num [1:6] 96.5 201 175 221 179 231


The summary of each variable is
species: Length 6   Class character   Mode character
N: Min. 2   1st Qu. 2   Median 2   Mean 5.16666666666667   3rd Qu. 2.75   Max. 20
MeanHeight: Min. 96.5   1st Qu. 176   Median 190   Mean 183.916666666667   3rd Qu. 216   Max. 231


# A tibble: 6 x 3
  species      N MeanHeight
  <chr>    <int>      <dbl>
1 Droid        3       96.5
2 Human       20      175
3 Twi'lek      2      179
4 Gungan       2      201
5 Kaminoan     2      221
6 Wookiee      2      231
```

These commands have been included in the above example using what we call "camel case" which has upper case letters for each word; to be specific it is "upper camel case" because the first word is also capitalised. The `dplyr` package uses "snake case" which replaces a space between words with an underscore, and uses only lower case letters. This choice is entirely up to the person who first develops the functions in a package. While `BrailleR` functions generally use upper camel case, it is a fairly simple exercise to add alternatives so `check_it()` and `what_is()` are also available if the user prefers to use snake case throughout the pipe chain.

The `CheckIt()` or `check_it()` command is actually just a substitute for the `str()` command from base R with the additional feature that it is compliant with the pipe operator and can therefore be used in a pipe chain. The `WhatIs()` or `what_is()` command is a substitute for the `VI()` command demonstrated in other chapters, but `VI()` is not pipe chain compliant.

Note that use of either `WhatIs()` or `CheckIt()` too early in the pipe chain may lead to verbose output. These functions are meant for checking that the output is what was sought, not for generating a final result for sharing.

9.4 BrailleR commands used in this chapter

The two main `BrailleR` functions relevant for use in the middle or at the end of a pipe chain are `WhatIs()` and `CheckIt()`. Alternative spelling of these

commands is also available; `check_it()` and `what_is()` are entirely equivalent versions.

Chapter 10

The ggplot world and BrailleR

The use of the ggplot style of graph production has increased markedly since its inception with the `ggplot2` package (Wickham et al., 2021a) now being one of the most used R packages. The "gg" is short for the "grammar of graphics" but the R code used to create an extremely wide range of graphs is seldom human-interpretable with ease. Creation of suitable support functionality via the `VI()` command is very definitely required. An initial attempt to extract any useful infromation from these graphs was contributed to the `BrailleR` package by Tony Hirst. Significant improvement has been made by Debra Warren as part of her postgraduate work under the supervision of Paul Murrell at the University of Auckland. Much of what is displayed in this chapter is only worth offering because of Debra's work and the interactions had between her, Paul and I in the second half of 2017.

In 2021, the `plot.ggplot()` command in the `ggplot2` package was updated to automatically call `BrailleR`'s `VI()` function if the user has specified they are blind. The `GoBlind()` and `GoSighted()` commands will toggle the automated use of `VI()` on and off respectively. The default starting mode is to get the automated `VI()` output.

N.B. the commands here are exact copies of the commands presented in Wickham (2009) or some minor alterations to them; any changes will be explicitly noted. All plots are created using the figure numbers from Wickham (2009) or the page numbers if no figure number was given. They are then automatically investigated using the `VI()` command from the `BrailleR` package which has not needed to be explicitly called since v0.32.0 of `BrailleR` and v3.3.5 of the `ggplot2` package..

You will need some additional packages to the `BrailleR` package to be loaded to follow along with the examples in this chapter. Do this by issuing the commands:

```
library(BrailleR)
library(ggplot2)
library(magrittr)
```

Note that one data set used for these examples is created by Wickham (2009) while the others are included in the `ggplot2` package.

```
set.seed(1410)
dsmall <- diamonds[sample(nrow(diamonds), 100),]
```

One important note for the graphs in this chapter is the difference in the way they appear here, as compared to the original figures of Wickham (2009) where differing height and width parameters have been set for each graph. For example, in the following graph, the points are smaller than in the original figure, and the aspect ratio is slightly different. The consequence is that this graph looks less cluttered than does the original.

```
p11a = qplot(carat, price, data = diamonds)
p11a
```

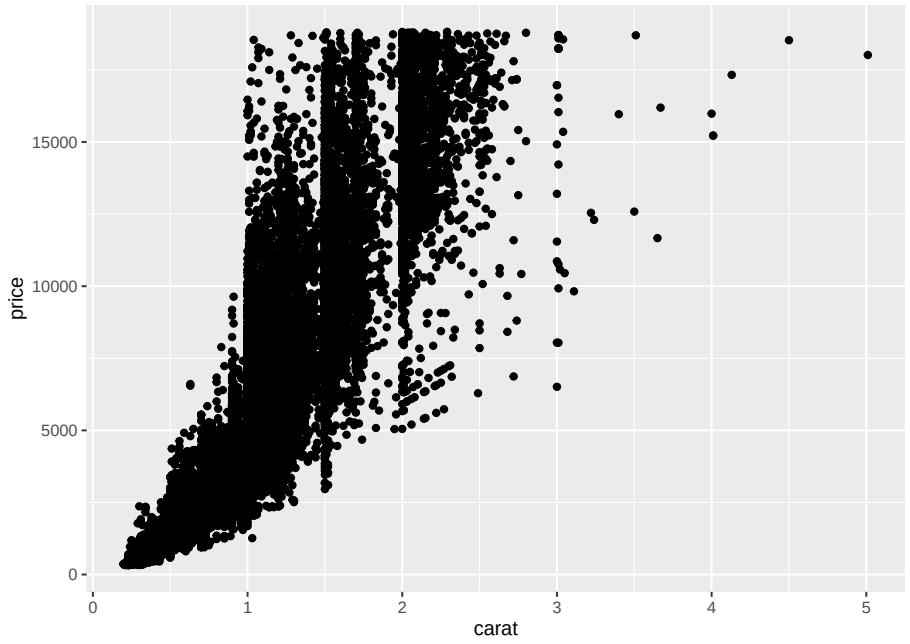


Figure 10.1: First graph on page 11 of Wickham (2009)

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 0, 1, 2, 3, 4 and 5.
It has y-axis 'price' with labels 0, 5000, 10000 and 15000.
```

```
The chart is a set of 53940 points.
```

Note that if you do not want to generate the graph but want to know what its text description has to offer, you can use the %>% pipe operator as follows:

```
fig2.2a = qplot(carat, price, data = dsmall, colour = color)
fig2.2a %>% VI()
```

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 1, 2, 3 and 4.
It has y-axis 'price' with labels 0, 5000, 10000 and 15000.
There is a legend indicating colour is used to show color, with 7 levels:
D shown as very deep purple colour,
E shown as vivid purplish blue colour,
F shown as moderate blue colour,
G shown as vivid bluish green colour,
H shown as brilliant green colour,
I shown as vivid yellow green colour and
J shown as vivid greenish yellow colour.
The chart is a set of 100 points.
```

We haven't been able to tell what exact colour was used in the Wickham (2009) rendering of this graph, but there has obviously been some minor alteration of the colour palette being used by the ggplot2 package over the years.

```
fig2.2b = qplot(carat, price, data = dsmall, shape = cut)
fig2.2b
```

```
Warning: Using shapes for an ordinal variable is not advised

Warning: Using shapes for an ordinal variable is not advised
```

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 1, 2, 3 and 4.
It has y-axis 'price' with labels 0, 5000, 10000 and 15000.
There is a legend indicating shape is used to show cut, with 5 levels:
Fair shown as solid circle shape,
Good shown as solid triangle shape,
Very Good shown as solid square shape,
Premium shown as plus shape and
Ideal shown as boxed X shape.
The chart is a set of 100 points.
```

To get semi-transparent points:

```
fig2.3b = qplot(carat, price, data = diamonds, alpha = I(1/100))
fig2.3b
```

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 0, 1, 2, 3, 4 and 5.
```

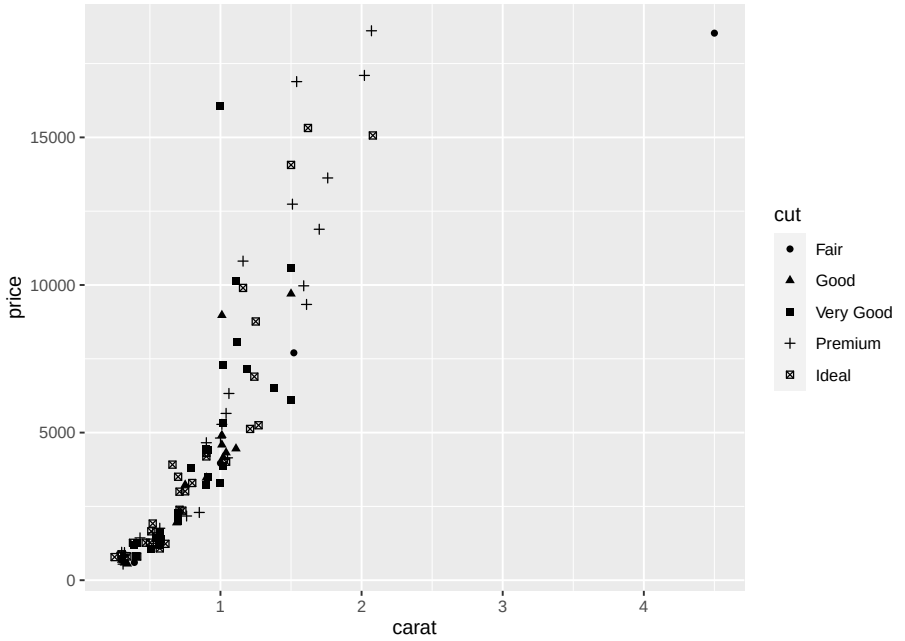


Figure 10.2: Right pane of Figure 2.2

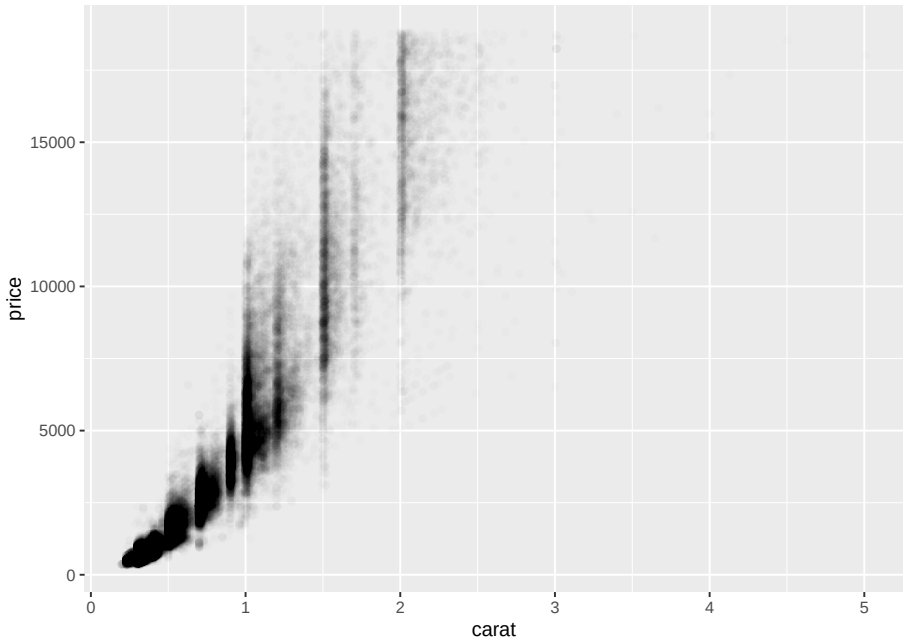


Figure 10.3: Middle pane from Figure 2.3

```
It has y-axis 'price' with labels 0, 5000, 10000 and 15000.
The chart is a set of 53940 points.
It has alpha set to 0.01.
```

To add a smoother (default is loess for n<1000):

```
fig2.4a = qplot(carat, price, data = dsmall, geom = c("point", "smooth"))
fig2.4a
```

```
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

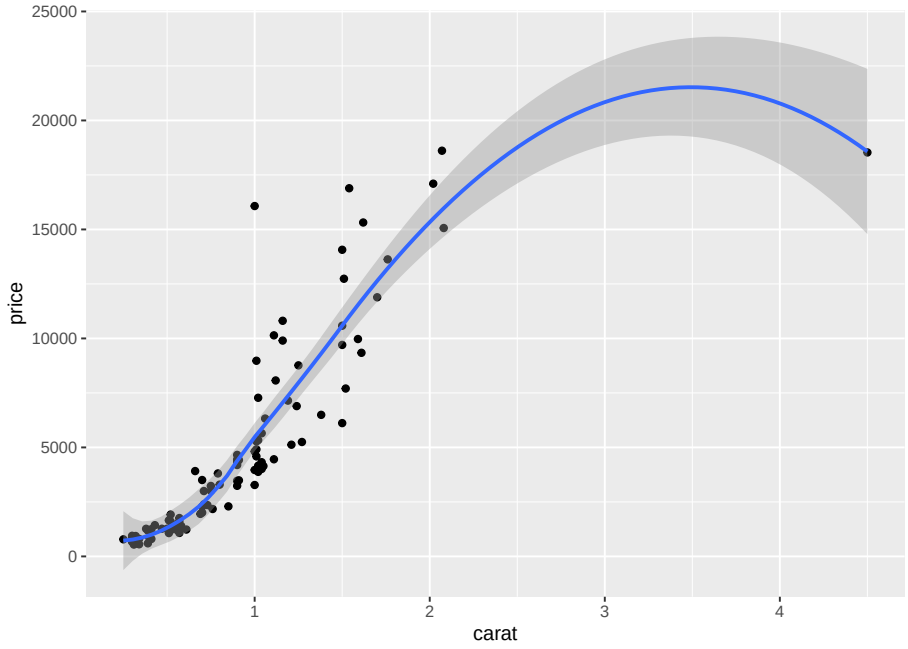


Figure 10.4: Left pane of Figure 2.4

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 1, 2, 3 and 4.
It has y-axis 'price' with labels 0, 5000, 10000, 15000, 20000 and 25000.
It has 2 layers.
Layer 1 is a set of 100 points.
Layer 2 is a 'lowess' smoothed curve with 95% confidence intervals.
```

10.1 Plotting a continuous variable against a categorical variable

```
fig2.8a  = qplot(color, price / carat, data = diamonds, geom = "jitter")
fig2.8a
```

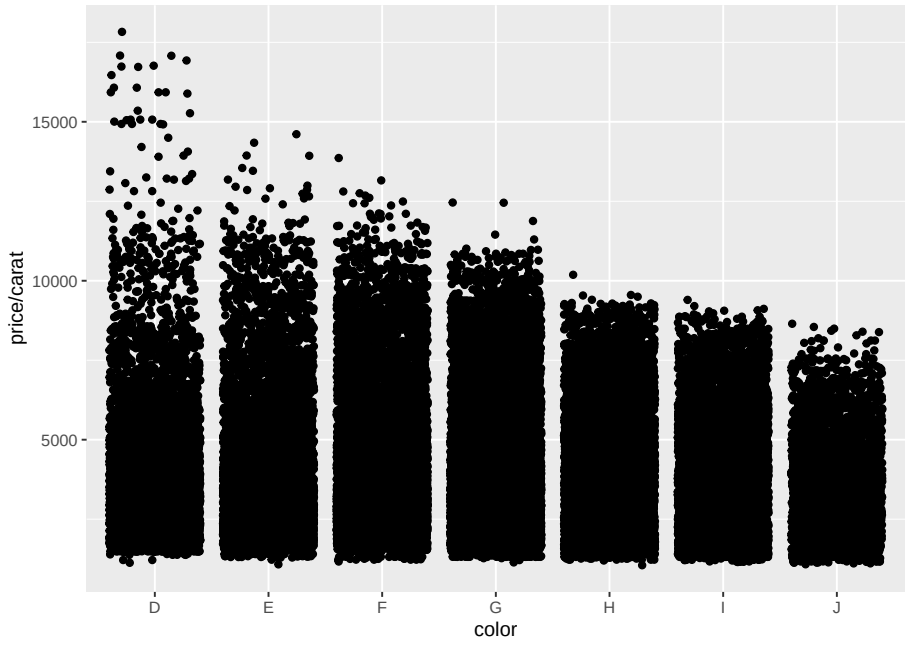
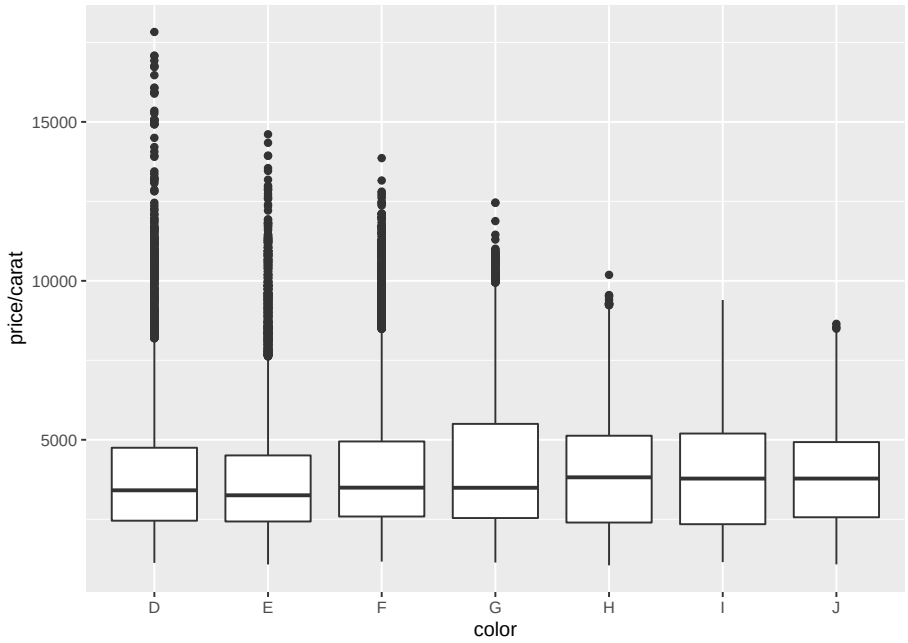


Figure 10.5: Left pane of Figure 2.8

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'color' with labels D, E, F, G, H, I and J.
It has y-axis 'price/carat' with labels 5000, 10000 and 15000.
The chart is a set of 53940 points.
```

```
fig2.8b = qplot(color, price / carat, data = diamonds, geom = "boxplot")
fig2.8b
```




```
This is an untitled chart with no subtitle or caption.
It has x-axis 'color' with labels D, E, F, G, H, I and J.
It has y-axis 'price/carat' with labels 5000, 10000 and 15000.
The chart is a boxplot comprised of 7 boxes with whiskers.
There is a box at x=D.
It has median 3410.53. The box goes from 2455 to 4749.31, and the whiskers extend to 1128.12 and
There are 338 outliers for this boxplot.
There is a box at x=E.
It has median 3253.66. The box goes from 2430.3 to 4508.41, and the whiskers extend to 1078.12 ar
There are 593 outliers for this boxplot.
There is a box at x=F.
It has median 3494.32. The box goes from 2587.1 to 4947.22, and the whiskers extend to 1168 and 8
There are 585 outliers for this boxplot.
There is a box at x=G.
It has median 3490.38. The box goes from 2538.24 to 5500, and the whiskers extend to 1139.02 and
There are 119 outliers for this boxplot.
There is a box at x=H.
It has median 3818.89. The box goes from 2396.88 to 5127.28, and the whiskers extend to 1051.16 a
There are 13 outliers for this boxplot.
There is a box at x=I.
It has median 3779.74. The box goes from 2344.65 to 5196.75, and the whiskers extend to 1151.72 a
There are 0 outliers for this boxplot.
There is a box at x=J.
It has median 3780. The box goes from 2562.87 to 4927.95, and the whiskers extend to 1080.65 and
```

```
There are 3 outliers for this boxplot.
```

When seeking to use shading or opaqueness to describe the density of the points, the fact the size of the points has an impact on the opaqueness is not currently realised by `BrailleR`.

```
fig2.9b = qplot(color, price / carat, data = diamonds, geom = "jitter", alpha = I(1 / 5
fig2.9b
```

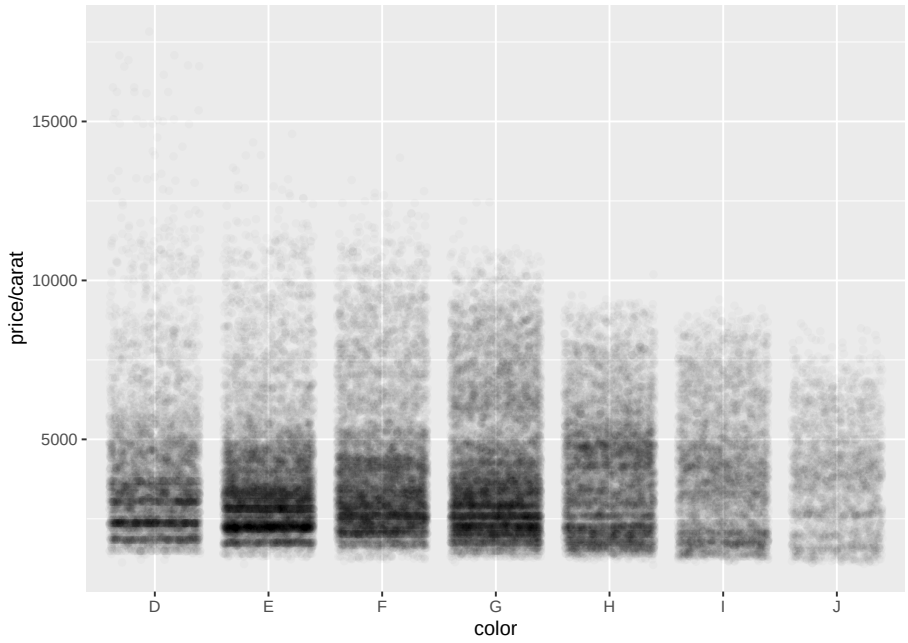


Figure 10.6: Middle pane of Figure 2.9

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'color' with labels D, E, F, G, H, I and J.
It has y-axis 'price/carat' with labels 5000, 10000 and 15000.
The chart is a set of 53940 points.
It has alpha set to 0.02.
```

10.1.1 Univariate plots

```
fig2.10a = qplot(carat, data = diamonds, geom = "histogram")
fig2.10a
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

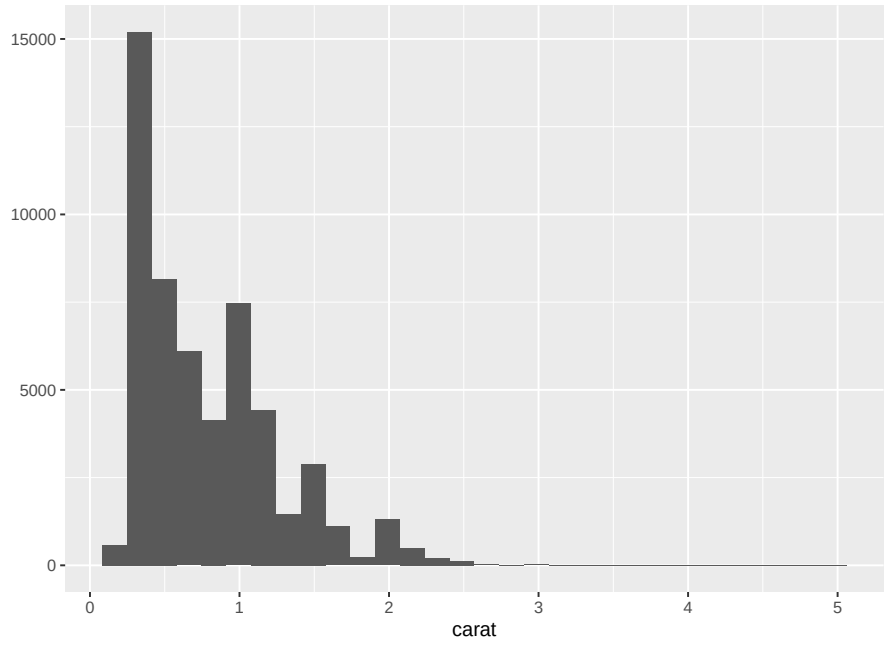


Figure 10.7: Left pane of Figure 2.10

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 0, 1, 2, 3, 4 and 5.
It has y-axis '' with labels 0, 5000, 10000 and 15000.
The chart is a bar chart with 30 vertical bars.
```

Warning: This figure does look different to the original in Wickham (2009) ins spite of using the same code and same data.

```
fig2.10b = qplot(carat, data = diamonds, geom = "density")
fig2.10b
```

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 0, 1, 2, 3, 4 and 5.
It has y-axis '' with labels 0.0, 0.5, 1.0 and 1.5.
The chart is a density graph that VI can not process.
```

```
fig2.11c = qplot(carat, data = diamonds, geom = "histogram", binwidth = 0.01, xlim = c(0,3))
fig2.11c
```

```
Warning: Removed 32 rows containing non-finite values (stat_bin).
```

```
Warning: Removed 2 rows containing missing values (geom_bar).
```

```
Warning: Removed 32 rows containing non-finite values (stat_bin).
```

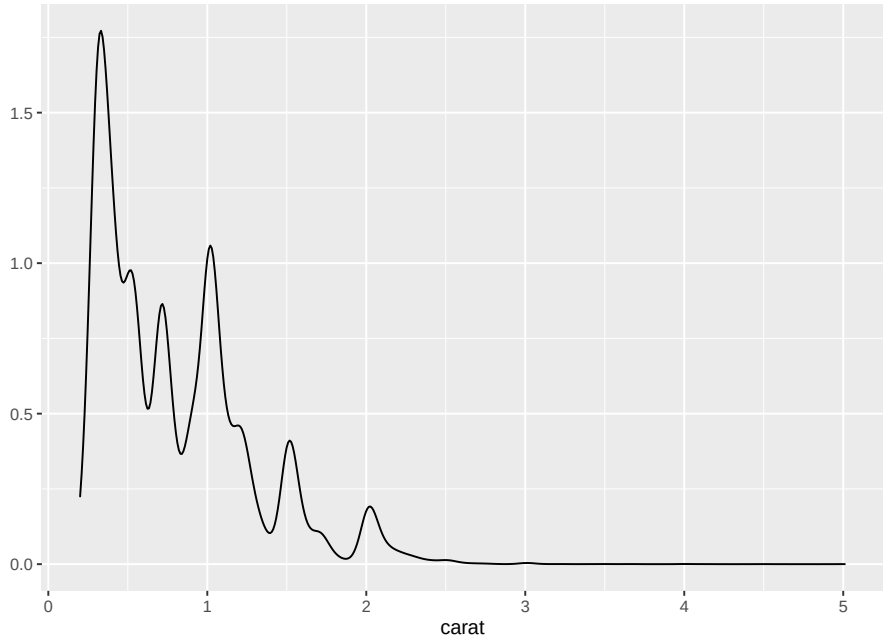


Figure 10.8: Right pane of Figure 2.10

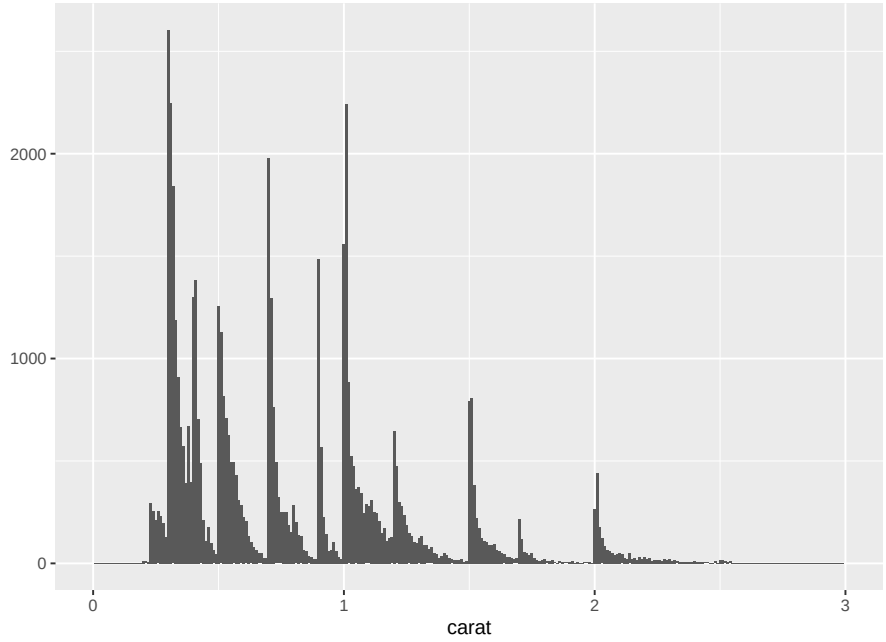


Figure 10.9: Right pane of Figure 2.11

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 0, 1, 2 and 3.
It has y-axis '' with labels 0, 1000 and 2000.
The chart is a bar chart with 299 vertical bars.
```

The data is separated by implication in the following graphs. The legend is automatically generated and has altered in appearance since the original was produced in Wickham (2009).

```
fig2.12a = qplot(carat, data = diamonds, geom = "density", colour = color)
fig2.12a
```

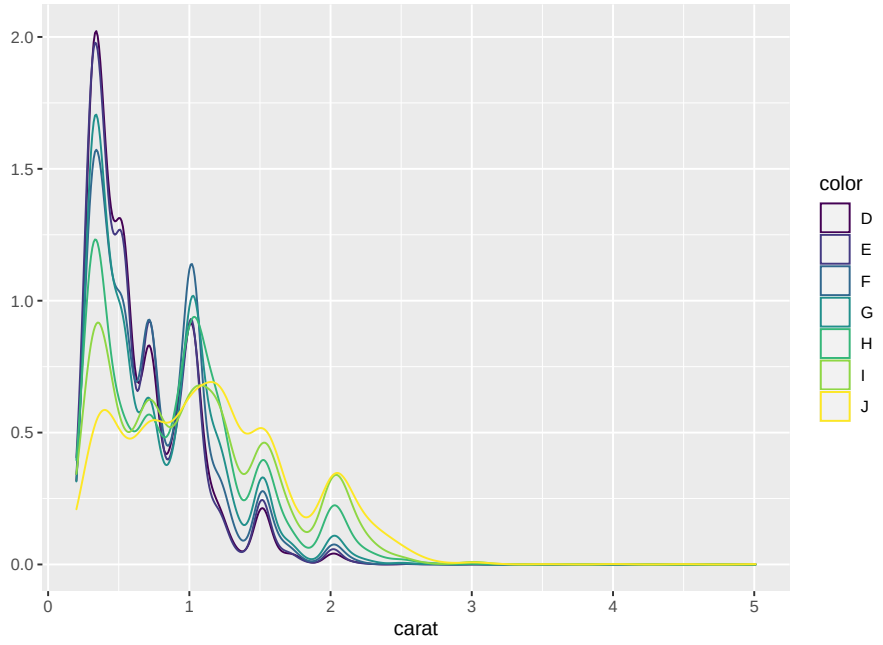


Figure 10.10: Left pane of Figure 2.12

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 0, 1, 2, 3, 4 and 5.
It has y-axis '' with labels 0.0, 0.5, 1.0, 1.5 and 2.0.
There is a legend indicating colour is used to show color, with 7 levels:
D shown as very deep purple colour,
E shown as vivid purplish blue colour,
F shown as moderate blue colour,
G shown as vivid bluish green colour,
H shown as brilliant green colour,
I shown as vivid yellow green colour and
J shown as vivid greenish yellow colour.
```

The chart is a density graph that VI can not process.

```
fig2.12b = qplot(carat, data = diamonds, geom = "histogram", fill = color)
fig2.12b
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

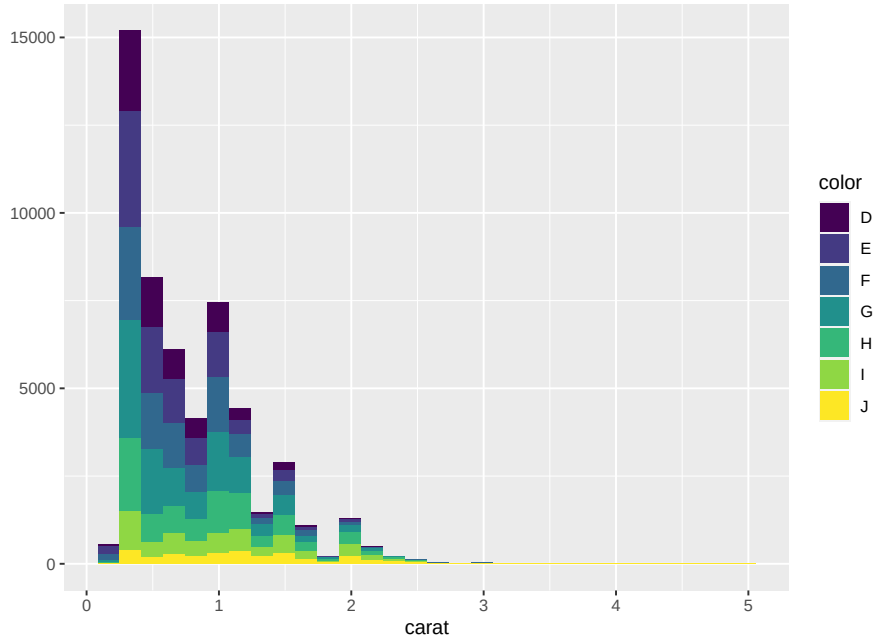


Figure 10.11: Right pane of Figure 2.12

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 0, 1, 2, 3, 4 and 5.
It has y-axis '' with labels 0, 5000, 10000 and 15000.
There is a legend indicating fill is used to show color, with 7 levels:
D shown as very deep purple fill,
E shown as vivid purplish blue fill,
F shown as moderate blue fill,
G shown as vivid bluish green fill,
H shown as brilliant green fill,
I shown as vivid yellow green fill and
J shown as vivid greenish yellow fill.
The chart is a bar chart with 210 vertical bars.
These are stacked, as sorted by color.
```

10.1.2 Bar charts for categorical variables

```
fig2.13a  = qplot(color, data = diamonds, geom = "bar") #geom="bar" is the default
fig2.13a
```

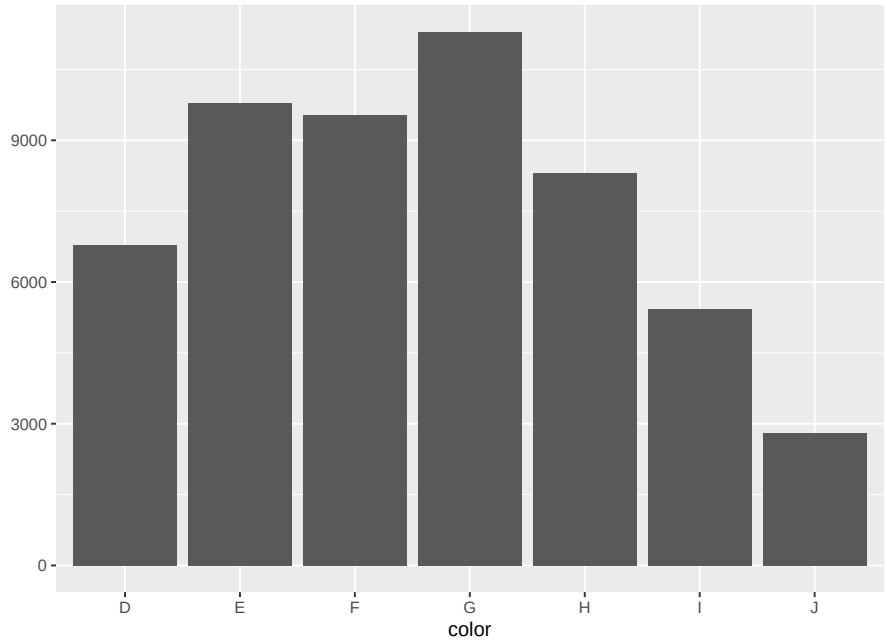


Figure 10.12: Left pane of Figure 2.13

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'color' with labels D, E, F, G, H, I and J.
It has y-axis '' with labels 0, 3000, 6000 and 9000.
The chart is a bar chart with 7 vertical bars.
Bar 1 is centered horizontally at D, and spans vertically from 0 to 6775.
Bar 2 is centered horizontally at E, and spans vertically from 0 to 9797.
Bar 3 is centered horizontally at F, and spans vertically from 0 to 9542.
Bar 4 is centered horizontally at G, and spans vertically from 0 to 11292.
Bar 5 is centered horizontally at H, and spans vertically from 0 to 8304.
Bar 6 is centered horizontally at I, and spans vertically from 0 to 5422.
Bar 7 is centered horizontally at J, and spans vertically from 0 to 2808.
```

need to check...

```
fig2.13b = qplot(color, data = diamonds, geom = "bar", weight = carat)
fig2.13b
```

```
This is an untitled chart with no subtitle or caption.
```

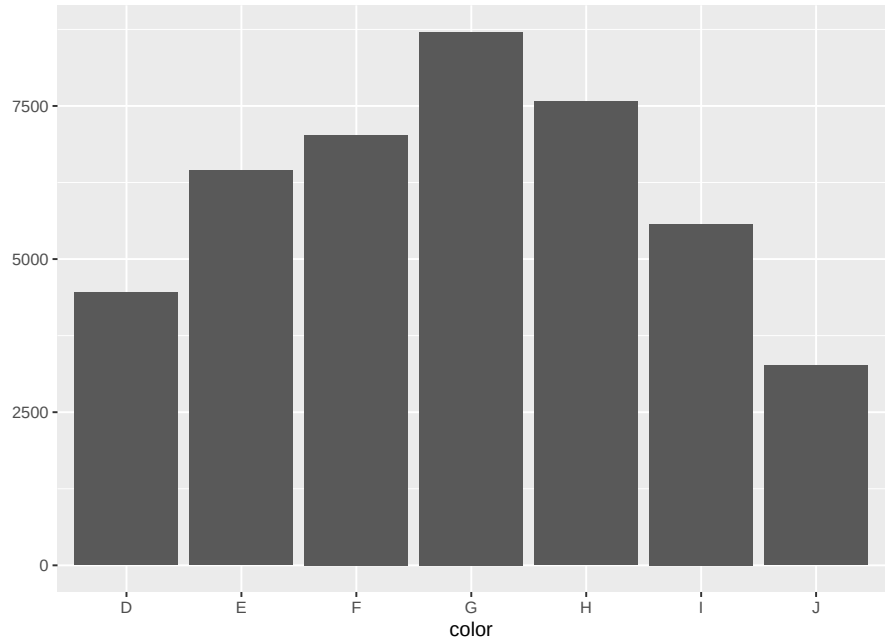


Figure 10.13: Right pane of Figure 2.13

```
It has x-axis 'color' with labels D, E, F, G, H, I and J.
It has y-axis '' with labels 0, 2500, 5000 and 7500.
The chart is a bar chart with 7 vertical bars.
Bar 1 is centered horizontally at D, and spans vertically from 0 to 4456.56.
Bar 2 is centered horizontally at E, and spans vertically from 0 to 6445.12.
Bar 3 is centered horizontally at F, and spans vertically from 0 to 7028.05.
Bar 4 is centered horizontally at G, and spans vertically from 0 to 8708.28.
Bar 5 is centered horizontally at H, and spans vertically from 0 to 7571.58.
Bar 6 is centered horizontally at I, and spans vertically from 0 to 5568.
Bar 7 is centered horizontally at J, and spans vertically from 0 to 3263.28.
```

```
fig2.13b = qplot(color, data = diamonds, geom = "bar", weight = carat) + scale_y_contin
fig2.13b
```

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'color' with labels D, E, F, G, H, I and J.
It has y-axis '' with labels 0, 2500, 5000 and 7500.
The chart is a bar chart with 7 vertical bars.
Bar 1 is centered horizontally at D, and spans vertically from 0 to 4456.56.
Bar 2 is centered horizontally at E, and spans vertically from 0 to 6445.12.
Bar 3 is centered horizontally at F, and spans vertically from 0 to 7028.05.
Bar 4 is centered horizontally at G, and spans vertically from 0 to 8708.28.
```

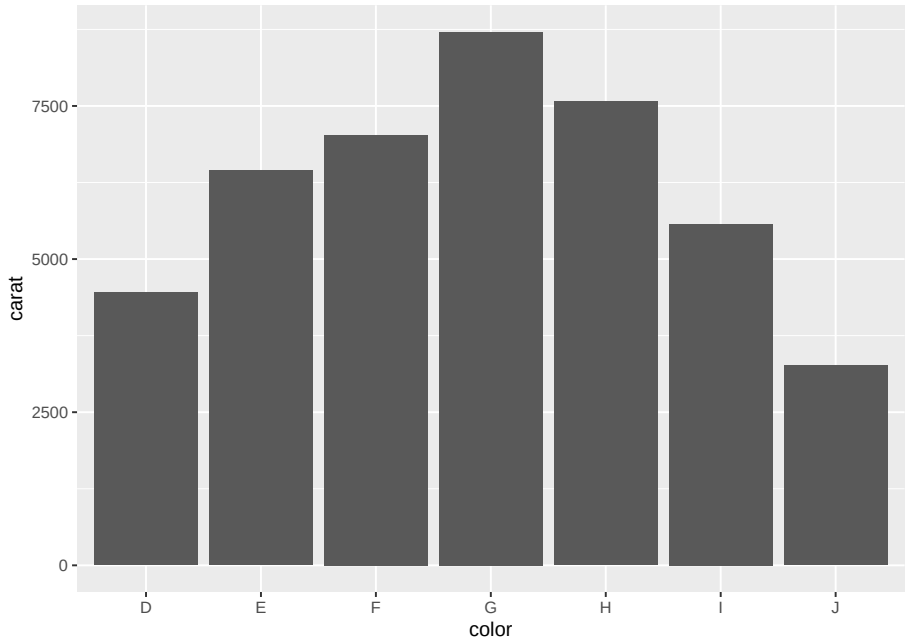


Figure 10.14: Right pane of Figure 2.13

```
Bar 5 is centered horizontally at H, and spans vertically from 0 to 7571.58.
Bar 6 is centered horizontally at I, and spans vertically from 0 to 5568.
Bar 7 is centered horizontally at J, and spans vertically from 0 to 3263.28.
```

10.2 Time series plots

It looks like the data used in the next graph has been updated since the publication of Wickham (2009)

```
fig2.14a = qplot(date, unemploy / pop, data = economics, geom = "line")
fig2.14a
```

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'date' with labels 1970, 1980, 1990, 2000 and 2010.
It has y-axis 'unemploy/pop' with labels 0.02, 0.03, 0.04 and 0.05.
The chart is a set of 1 line.
Line 1 connects 574 points.
```

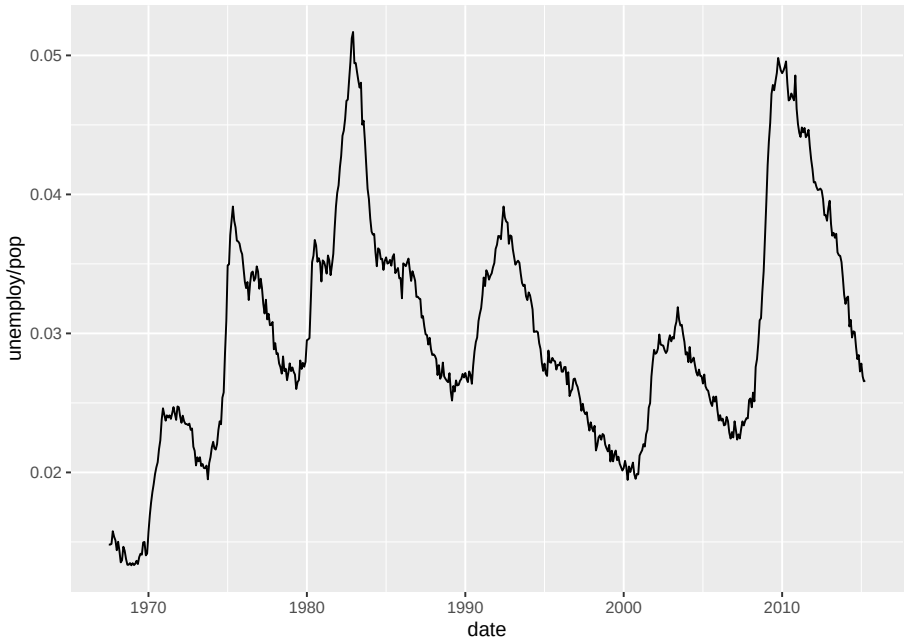


Figure 10.15: Left pane of Figure 2.14

10.3 Path plots

```
year <- function(x) as.POSIXlt(x)$year + 1900
fig2.15b = qplot(unemploy / pop, uempmed, data = economics, geom = "path", colour=year
#+ scale_area() # no longer works
fig2.15b
```

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'unemploy/pop' with labels 0.02, 0.03, 0.04 and 0.05.
It has y-axis 'uempmed' with labels 5, 10, 15, 20 and 25.
There is a legend indicating colour is used to show year(date), ranging from 1967 repre
The chart is a path graph that VI can not process.
```

10.4 Facets is the ggplot term for trellis' panels

The aspect ratio for the plot region is something that needs to be considered. I've manually adjusted the plotting window here so that the graph more closely matches that of Wickham (2009) but it is not an exact match.

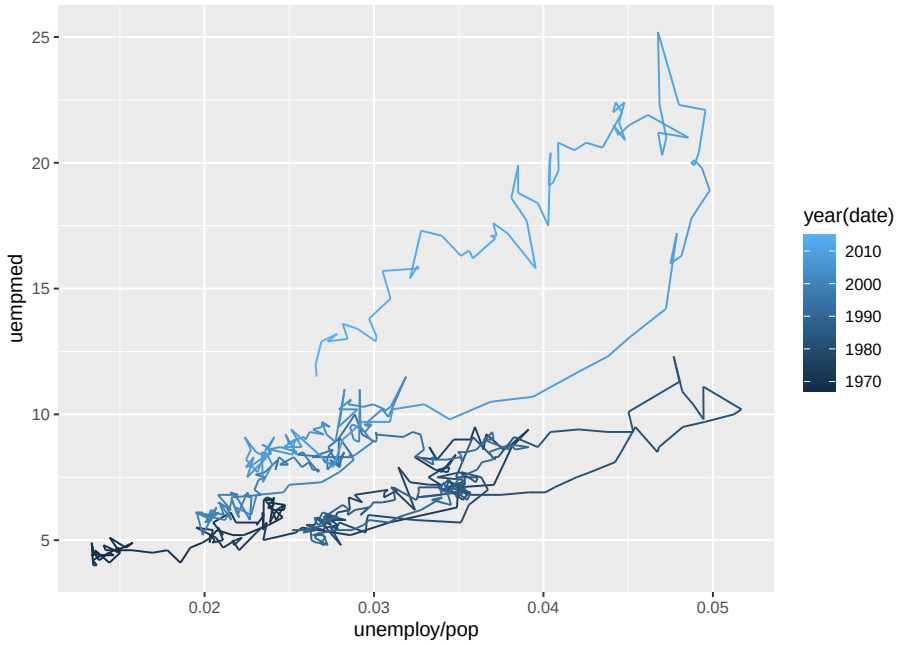


Figure 10.16: Right pane of Figure 2.15

```
fig2.16a = qplot(carat, data = diamonds, facets = color ~ ., geom = "histogram", binwidth = 0.1,
xlim = c(0, 3))
fig2.16a
```

```
Warning: Removed 32 rows containing non-finite values (stat_bin).
```

```
Warning: Removed 14 rows containing missing values (geom_bar).
```

```
Warning: Removed 32 rows containing non-finite values (stat_bin).
```

```
This is an untitled chart with no subtitle or caption.
The chart is comprised of 7 panels containing sub-charts, arranged vertically.
The panels represent different values of color.
Each sub-chart has x-axis 'carat' with labels 0, 1, 2 and 3.
Each sub-chart has y-axis '' with labels 0, 500, 1000, 1500, 2000 and 2500.
Panel 1 represents data for color = D.
Panel 1 is a bar chart with 29 vertical bars.
Panel 2 represents data for color = E.
Panel 2 is a bar chart with 29 vertical bars.
Panel 3 represents data for color = F.
Panel 3 is a bar chart with 29 vertical bars.
Panel 4 represents data for color = G.
Panel 4 is a bar chart with 29 vertical bars.
```

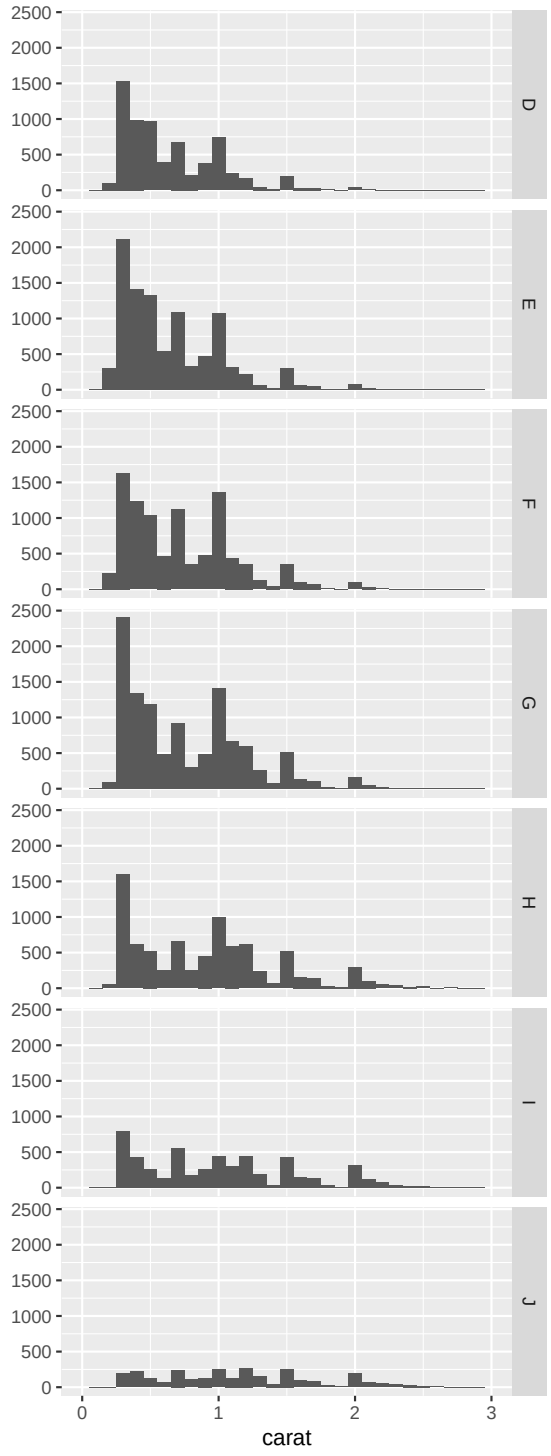


Figure 10.17: Left side of Figure 2.16

```
Panel 5 represents data for color = H.
Panel 5 is a bar chart with 29 vertical bars.
Panel 6 represents data for color = I.
Panel 6 is a bar chart with 29 vertical bars.
Panel 7 represents data for color = J.
Panel 7 is a bar chart with 29 vertical bars.
```

```
fig2.16b = qplot(carat, ..density.., data = diamonds, facets = color ~ ., geom = "histogram", bin
fig2.16b
```

```
Warning: Removed 32 rows containing non-finite values (stat_bin).

Warning: Removed 14 rows containing missing values (geom_bar).

Warning: Removed 32 rows containing non-finite values (stat_bin).

This is an untitled chart with no subtitle or caption.
The chart is comprised of 7 panels containing sub-charts, arranged vertically.
The panels represent different values of color.
Each sub-chart has x-axis 'carat' with labels 0, 1, 2 and 3.
Each sub-chart has y-axis '..density..' with labels 0.0, 0.5, 1.0, 1.5 and 2.0.
Panel 1 represents data for color = D.
Panel 1 is a bar chart with 29 vertical bars.
Panel 2 represents data for color = E.
Panel 2 is a bar chart with 29 vertical bars.
Panel 3 represents data for color = F.
Panel 3 is a bar chart with 29 vertical bars.
Panel 4 represents data for color = G.
Panel 4 is a bar chart with 29 vertical bars.
Panel 5 represents data for color = H.
Panel 5 is a bar chart with 29 vertical bars.
Panel 6 represents data for color = I.
Panel 6 is a bar chart with 29 vertical bars.
Panel 7 represents data for color = J.
Panel 7 is a bar chart with 29 vertical bars.
```

10.5 Rescaling of the axes

```
p26a = qplot(carat, price, data = dsmall, log = "xy")
p26a
```

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 0.3, 1.0 and 3.0.
It has y-axis 'price' with labels 1000, 3000 and 10000.
The chart is a set of 100 points.
```

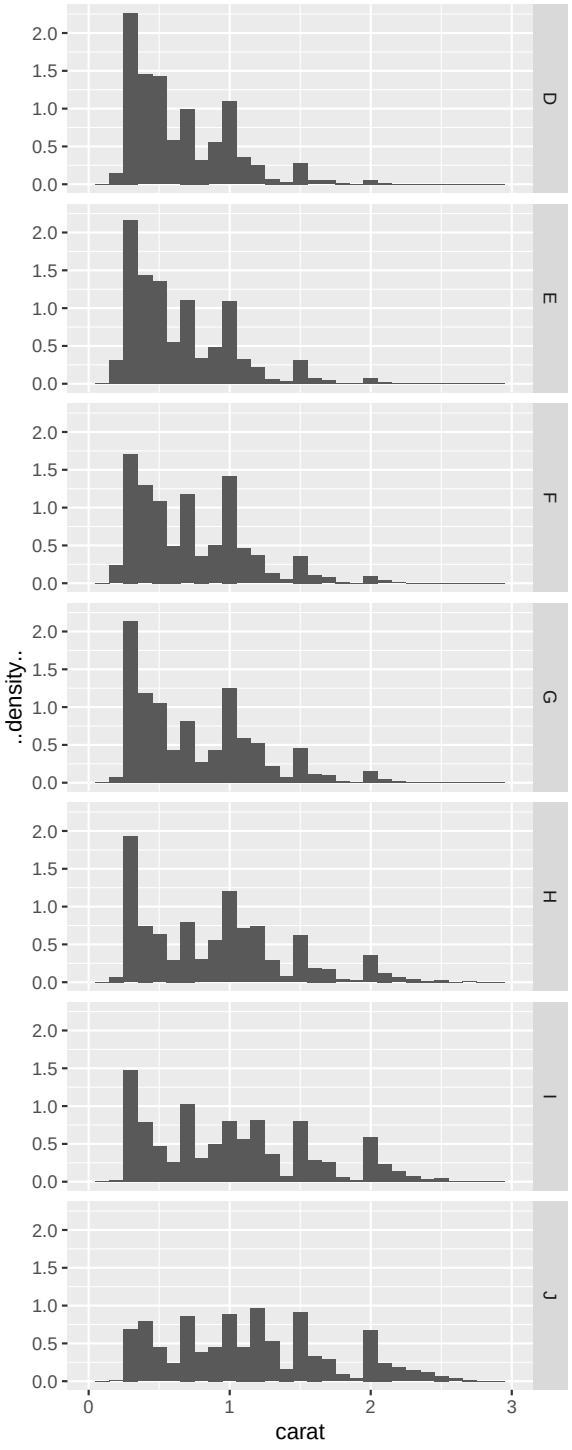


Figure 10.18: Right side of Figure 2.16

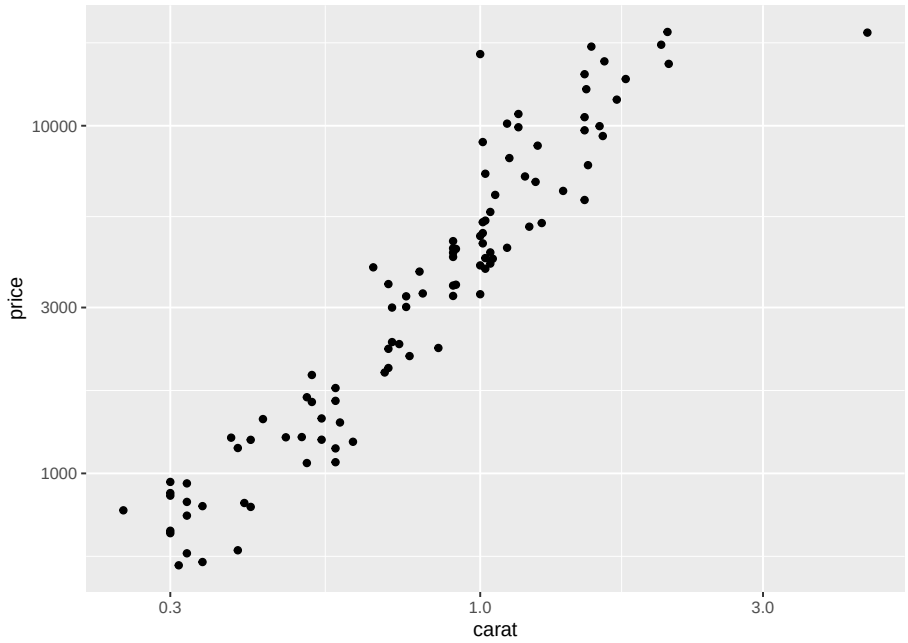


Figure 10.19: First graph on page 26 of

```
fig3.6 = qplot(displ, hwy, data=mpg, facets =~ year) + geom_smooth()
fig3.6
```

```
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

```
This is an untitled chart with no subtitle or caption.
The chart is comprised of 2 panels containing sub-charts, arranged horizontally.
The panels represent different values of year.
Each sub-chart has x-axis 'displ' with labels 2, 3, 4, 5, 6 and 7.
Each sub-chart has y-axis 'hwy' with labels 20, 30 and 40.
Each sub-chart has 2 layers.
Panel 1 represents data for year = 1999.
Layer 1 of panel 1 is a set of 117 points.
Layer 2 of panel 1 is a 'lowess' smoothed curve with 95% confidence intervals.Panel 2 represents
Layer 1 of panel 2 is a set of 117 points.
Layer 2 of panel 2 is a 'lowess' smoothed curve with 95% confidence intervals.
```

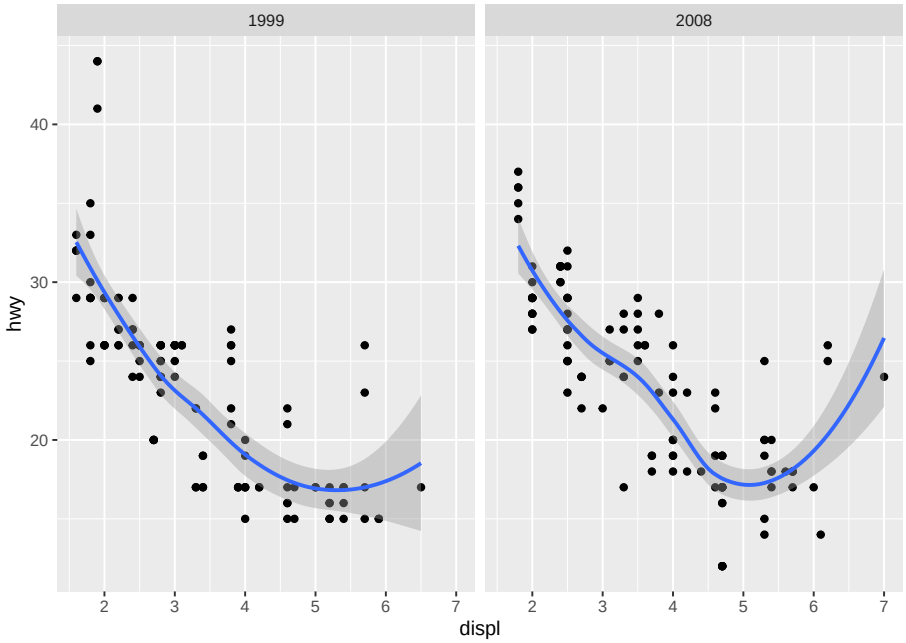


Figure 10.20: Figure 3.6 of

Chapter 11

Getting started with the WriteR application

The WriteR application was written to support use of R markdown and the `BrailleR` package. It is a Python script making use of wxPython to help build the graphic user interface (GUI) in such a way that it works for screen reader users. This book has been written in R markdown, and I have made extensive use of WriteR because it offers so many convenient tools for a blind user wanting to write R markdown files.

The script is in the `BrailleR` package, but it cannot run unless the user has both Python and wxPython installed. Commands have been included in the `BrailleR` package to help Windows users obtain installation files for them. Users of other operating systems currently have to install Pandoc, Python and wxPython independently, but these tools may well already be installed.

11.1 Getting the required software (Windows users only)

You can check if a version of Python is already installed on your computer using

```
Sys.which("Python")
```

```
                                Python
"C:\\PROGRA~1\\Python38\\Python.exe"
```

The output above shows you that I have Python 3.8 installed on my computer, and that it can be found in the folder I know is the default location. You can use this command later to check your progress, but there are other ways

to check your system that return more useful detail, but for the moment the `Sys.which()` command above issufficient. What it does not show you is whether the installation of Python is suited for a 64 bit machine, or is the more universal 32 bit installation. We really ought to have an exact match between the version of Python and the version of R being used.

The files downloaded as a consequence of running commands in this set of instructions will be saved in your `MyBrailleR` folder. You will need to follow the instructions and answer questions that arise whenever you install new software, but you should probably read all the way through this set of instructions before getting underway.

Ultimately, you will need a mainstream tool to process the markdown files you write into other formats. The WriteR application will end up using a tool called "pandoc" to do this so we need to get this installed. If you do not have an installation of Pandoc and Python 3 then you can use some functions from the `BrailleR` package to help make the setup smoother. Remember to load the package using:

```
library(BrailleR)
```

It doesn't matter if you install Python before or after Pandoc.

11.1.1 Installing Pandoc

Let's first install Pandoc using the command:

```
GetPandoc()
```

To check that you have Pandoc installed, and that R can therefore find it, use the command"

```
Sys.which("pandoc")
```

```
                                pandoc
"C:\\PROGRA~1\\Pandoc\\pandoc.exe"
```

which will show where Pandoc was installed.

11.1.2 Installing Python

The following instructions fetch the installation files from the reputable Python sites. Windows and any security software you might have should know that, but you can never tell! You may need to let Windows know it is OK to install the software in the default location. The pop-up might not appear as the window with focus so if things look like they're going slowly, look around for the pop-up window.

The following commands automatically download the installation files needed for Python 3.x, and start the installation process going. Issue them at the R prompt

```
GetPython3()
```

Make sure that the first thing you do as part of the installation is to choose to use a custom installation. This allows you to make sure Python will be availabel to all users, update the system's environment variables. You may need to make sure these options are selected.

You need to make sure that the second of these options definitely happens; this ensures that the Python folders are added to your system path which means R and any other software that wants Python can find it. This makes it possible to run Python scripts from any folder on your computer. The next few commands will fail if this is not done properly. If you missed that step during the installation, look for the installer file in your `MyBrailleR` folder and run it again manually before proceeding.

We can check the installation has worked properly at this point using:

```
Sys.which("python")
```

```
                              python
"C:\\PROGRA~1\\Python38\\python.exe"
```

```
shell("path", intern=TRUE)
```

```
[1] "PATH=C:\\Program Files\\Python38\\Scripts\\;C:\\Program Files\\Python38\\;C:\\Rtools\\bin;C:
```

This is a much stronger test than was done earlier. You can see the folders that Windows will search through to find the tools you are using all the time. The folder for Python needs to be listed there somewhere.

Now get the additional Python modules needed for WriteR using:

```
GetWxPython3()
```

Once you have completed these installations, you are ready to go. You shouldn't need to keep the installation files, but why not keep them just in case. You can now skip to the section which shows you how to check everything is ready for using WriteR.

11.2 Other operating systems

11.3 Checking your system is ready

All going to plan, you should now have Python on your system, and the additional wxPython module as well. You only need to check that R really can see the

right version of Python, and that one extra Python module is correctly installed and available to R to find out how well you've done, using:

```
TestWX()
```

```
Your system is using Python 3.8.0
```

```
Python can see the necessary wx module.
```

```
You are ready to use WriteR.
```

N.B. The command will return more output on your system if you test the function in an interactive setting. Do that now.

The outcome of this command tells you if your system is ready to run the WriteR application. It first checks that a version of Python is available, and if there is, then runs a short script that uses WxPython.

If you are in an interactive session, and Python is all working, then a small window will have opened on your system, and told you that you are ready to use WriteR.

If for some reason you are not getting it all going properly, then there are several commands that can be issued at a command prompt. Windows users get a command prompt by typing `cmd` at the `Run` (Found by `Windows+R`).

The following commands are not R commands. They should work on all operating systems. Do not proceed until each has worked without error messages.

1.Typing

```
python --version
```

will tell you if Python is installed by returning the version number. 2. Make sure you can use the tool that grabs extra Python modules from an official site by typing

```
pip --version
```

If this doesn't work you will need to seek out the correct strategy via the Python wiki pages available online. 3. Upgrade your Python modules using

```
python -m pip install --user --upgrade pip setuptools wheel
```

This should download a few files and automatically put them on your system. 4. Finnally, install the wxPython module using

```
pip install --user --upgrade wxPython
```

Remember, do not move on to the next step until each of the commands above has returned satisfactory feedback. Windows users close the window using `Alt+F4` or by typing `exit`.

I strongly recommend that you go back and check everything has worked using the commands presented earlier in this chapter.

11.4 Opening WriteR from BrailleR

Opening WriteR is as easy as typing WriteR! Well almost. You have the option of specifying a filename; if that file exists, it gets opened for you, and if it doesn't exist, then it gets created with a few lines already included at the top to help get you started. Try:

```
WriteR("MyFirst.Rmd")
```

This should open the WriteR application with the following lines already there for you to edit.

```
---
title: ""
author: ""
date: ""
output:
    html_document:
        toc: false
        number_sections: false
        fig_height: 5
        fig_width: 7
---
```

Some of these lines were explained back in Chapter 6. Fill in the gaps in the first few lines for title, author, and the date before continuing.

11.5 What can I do with WriteR?

The window you are in has a number of menus, a status bar at the bottom and a big space in the middle for your work. Take a quick look at those menus; some will look familiar because they are common to many Windows applications.

The file you have opened is a markdown file. It is just text which is why it is so easy to read. The file extension of `Rmd` means it is an R markdown file. There are several flavours of markdown in common use, but they are practically all the same except for some very minor differences.

A markdown file can be converted into many file formats for distribution. These include HTML, pdf, Microsoft Word, Open Office, and a number of different slide presentation formats. Let's make the HTML file now.

11.6 Our first HTML file

Making your first HTML file is as easy as hitting the `f5` key, or using one of the options in the `Build` menu. The variety of options are the commonly used ones in RStudio. Let's just stick to making an HTML file for the time being. You can investigate other formats later.

When the processing is underway, a second window will open which gives the same output that you would see printed in an R session window if you were processing R markdown files manually. If the last line says "done 0", then everything processed properly; if it says "done 1" then there was an error to fix, so look through this log to see what went wrong.

The processing of your R markdown document will use a suite of packages, primarily including the `knitr` (Xie, 2021) and `rmarkdown` (Allaire et al., 2021) packages. Press the `f4` key to switch back to the main document editing window in WriteR.

If the processing of the file "MyFirst.Rmd" was successful, then you will now have a file called "MyFirst.html" in your current working directory. You have several choices for finding the HTML file you have created:

- Navigate to the current working directory using your file browser. To find out where that is, type `getwd()` in the R window to see where the files really are located, or if you are a Windows user, issue the `BrailleR` function `Explorer()` to opent he folder automatically. You should see the file `MyFirst.Rmd` and once you have built it, the associated HTML file. Open that file in your browser.
- Use the `browseURL()` command in the R session. You will need to provide the filename, in quotes, for example `browseURL("MyFirst.html")`

Use one of these methods to open the HTML file. Read through it to see how the markdown has been rendered. You may need to switch back and forth between the WriteR window and your browser to compare the plain text and the beautiful HTML. If you didn't actually edit the R markdown file up to now then the output HTML file will be rather boring. Add in some text, or use the menus to see how to insert headings and other things.

N.B. Changes in your R markdown file are not automatically converted into the HTML file. You must re-build the HTML, and refresh your browser to see the impact of any changes you make, both actions use the `f5` key.

11.7 Some hints for writing Rmarkdown documents

The WriteR application really can make writing documents easier because it offers plenty of shortcut keystrokes for items in the pulldown menus. Formatting text in a markdown document which will be converted to HTML is done in a way that creates semantic structure at the same time as font changes, or to put more precisely, you can't get a larger font for a heading without using a proper heading style. My list of hints given here are to help make your final HTML document a pleasure to read as well as easy to write.

- Always use the standard markdown syntax for inserting headings, graphics, links, and equations. WriteR will insert placeholders for graphics and links and some mathematical structures.
- Concentrate on the material to be communicated; do not worry about the formatting of it. Keeping it simple is best.
- Make use of the lessons learned by others. If you like hte way something is presented, then re-use the approach taken.
- if you inset a graph created in an R chunk, then make sure you use the `fig.cap` properly. This text will be used as the "alt tag" for the resulting graph.
- One figure per chunk is recommended. This helps with the alt tags.
- The default presentation of R output is to have three hash signs at the left of each line of output. Inclusion of an R chunk at the start of the document can fix this. Use something like:

````
```r
library(knitr)
opts_chunk$set(comment="")
```
````

I found the flow of some documents was unnecessarily clunky when simple mathematical elements were entered as math mode elements. Sentences will read better if x and y are in italicised font and not math mode, and the sighted readers won't notice the difference. This won't work for Greek, some subscripts and superscripts, or symbols. If in doubt, use math mode everywhere. This means using a dollar on both sides of the mathematical content such as `$x$` for example.

The method used to convert your raw markdown to beautiful HTML will matter. WriteR has been configured to use `render()` from the `rmarkdown` package as it delivers the best outcome for screen reader users.

Finally, you should investigate which combination of screen reader and browser gives you the results you like best. I regularly make use of both JAWS and NVDA, and my preferred browser has changed over the years. I used to only ever use Firefox, but Chrome gets more use today; both Internet Explorer and

Opera have both been used at times.

11.8 BrailleR commands used in this chapter

We used `TestWX()` to check the necessary Python installations were successful. It automatically called the `TestPython()` command as its first step.

We needed to use `GetPython3()` and `GetWxPython3()` to install the necessary software to run a Python script like WriteR.

Finally, we opened a new file using `WriteR()`.

Chapter 12

Making Accessible Graphs

While the initial aim of the BrailleR Project was to create a text description of a graph, there are many ways to explore graphs created in R by other means. This chapter introduces some techniques for creating or modifying graphs so that they can be investigated by a blind person without the assistance of a sighted person.

Chapter 13

The Work Ahead

The `BrailleR` Project is likely to evolve over time. The work on the project will be dependent on interest being shown by blind users of R, and perhaps some willingness being shown by people who could make the work so much easier.

Developments that meet specific requests from blind users are being addressed. For example, the request from one user to have the `BrailleR` package loaded in every session led to creation of a `MakeRprofile()` function; this function simplifies the experience for the user (a novice) for creation of the `.Rprofile` file in the current working directory. This is easier to explain to a novice than the process of altering the `.Rprofile` or `.Rprofile.site` files.

A request for assistance getting braille labels onto graphs has brought forward the plans to address this issue within the `BrailleR` package. The experimental `BRLThis()` function has been included in the package, as well as a similar `SVGThis()` function to optimise the settings for creation of a version using a braille font and a structured SVG file for a graph. The `BRLThis()` function creates a pdf document that has been successfully embossed on two different models of embosser, while SVG files generated by the `SVGThis()` function have only the bare minimum of useful content at this stage.

It may prove necessary for the `BRLThis()` function to be converted to a method like the `SVGThis()` functions so that the best results can be created for different graph types. At present, the `SVGThis()` is reliant on the existence of the `hist()` and `boxplot()` commands within the `BrailleR` package that are wrappers to the `graphics` package functions of the same name. It is my hope that I can encourage the R development core team to create more classes. The additions are almost trivial from their perspective, but the workload for me as a programmer will reduce markedly if I can write simpler code to interpret the graphs. Knowing what type of graph has been created by an object rather than writing code that makes a reasonable attempt at guessing what type of graph was created is a key example. To this end, it will prove easier to work with the graphs created

by the `xyplot()` function from the `lattice` package than the standard `plot()` command.

I will also need feedback from students and users of R, and perhaps their lecturers, tutors, and teachers, to see which ideas are working well for blind users. At present, the attempts being made are based on my own experiences and desires; I wanted functions to convert an R script to an R markdown file and a similr function to convert the history of commands to an Rmd file. The `R2Rmd()` and `History2Rmd()` functions were duly created; they have proven very useful in my work. I can only hoep others find them as useful. I am, therefore, seeking opportunities to share my work with blind users all over the world via a specially created email list for those people interested in how blind users can work with R{}. I believe that adding perspectives is a crucial pathway for the project's development and all feedback is extremely welcome.

A key contribution is possible in the area of choosing appropriate default text for the `BrailleR` output. The text representation needs to be more efficient than using R in its vanilla form. To this end, I have put some time into thinking about which text will be most suitable for an audience that will often listen to the output using synthetic speech, perhaps in the user's second language, as well as the braille readers who may or may not be able to use contracted English braille. At first, I had thought this task would be made easier by reviewing resources created by transcribers who record books for the blind, but my investigations show that the descriptions used in spoken words do not always equate to efficient braille or synthetic speech and often provide more interpretation than I believe is appropriate in educational settings.

Collaboration from others in any role will feed my own enthusiasm for the project, especially those whose skills and knowledge complement my own. The work of willing and helpful contributors has been gratefully received. In some instances, other researchers in the R community have inadvertently contributed to `BrailleR` because I have been able to adapt their work to meet the needs of blind users.

Chapter 14

References

Bibliography

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2021). *rmarkdown: Dynamic Documents for R*. R package version 2.11.

Bache, S. M. and Wickham, H. (2020). *magrittr: A Forward-Pipe Operator for R*. R package version 2.0.1.

Bulatov, V. and Gardner, J. A. (2004). Making graphics accessible. In *SVG Open Conference*, Tokyo. September 7-10.

Calder, M., Cohen, R., Lanzoni, J., and Xu, Y. (2006). PLUMB: An interface for users who are blind to display, create and modify graphs. *ASSETS'06*, pages 263–264.

Dahl, D. B., Scott, D., Roosen, C., Magnusson, A., and Swinton, J. (2019). *xtable: Export Tables to LaTeX or HTML*. R package version 1.8-4.

Dengler, P., Jackson, D., Lilley, C., Fujisawa, J., McCormack, C., Dahlström, E., Grasso, A., Ferraiolo, J., Schepers, D., and Watt, J. (2011). *Scalable vector graphics (SVG) 1.1*. W3C recommendation, W3C, second edition.

Fellows, I. (2012). Deducer: A data analysis GUI for R. *Journal of Statistical Software*, 49(8):1–15.

Fox, J. (2005). The R Commander: A basic statistics graphical user interface to R. *Journal of Statistical Software*, 14(9):1–42.

Freedom Scientific (2018). *JAWS Version 2018.1805.33*. Freedom Scientific, St. Petersburg, FL.

Gardner, J. and Bulatov, V. (2010). Highly accessible scientific graphical information through daisy svg. In *Proceedings of the 2010 SVGOpen Conference, Paris, France*.

Godfrey, A. J. R. (2009). Are statistics courses accessible? In *Proceedings of the Workshop on E-Inclusion in Mathematics and Science 2009*, pages 72–80, Fukuoka, Japan.

Godfrey, A. J. R. (2011). R. Workshop was delivered three times on 1-2 August, 2011 at the 2^{nd} Summer university held in Telc, The Czech Republic.

Godfrey, A. J. R. (2012a). The BrailleR project. In Yamaguchi, K. and Suzuki, M., editors, *Proceedings of Digitization and E-Inclusion in Mathematics and Science*, pages 89–95, Tokyo, Japan.

Godfrey, A. J. R. (2012b). Putting it all together — a blind person's perspective on document preparation. In Yamaguchi, K. and Suzuki, M., editors, *Proceedings of Digitization and E-Inclusion in Mathematics and Science*, pages 135–141, Tokyo, Japan.

Godfrey, A. J. R. (2013a). Blindness in a visual discipline. Poster session held on 25 November, 2013 at the 64^{th} Annual Conference of the New Zealand Statistical Association held in Hamilton.

Godfrey, A. J. R. (2013b). Statistical software from a blind person's perspective: R is the best, but we can make it better. *The R Journal*, 5(1):73–79.

Godfrey, A. J. R. (2013c). Using R: the most accessible statistical software for blind students. Workshop was delivered on 5 September, 2013 at the 4^{th} Summer university held in Bad Herrenalb, Germany.

Godfrey, A. J. R. (2014a). Introduction to R: the most accessible statistical software for blind students. Workshop was delivered twice on 8 July, 2014 at the 5^{th} Summer university held in Paris, France.

Godfrey, A. J. R. (2014b). Practical use of R by blind people. Presentation delivered on 1 July 2014 at the useR!2014 conference held in Los Angeles.

Godfrey, A. J. R. (2014c). R and LaTeX — a powerful combination for assignment preparation. Workshop was delivered twice on 8 July, 2014 at the 5^{th} Summer university held in Paris, France.

Godfrey, A. J. R. (2014d). A review of statistical software for blind students. Plenary address delivered on 8 July, 2014 at the 5^{th} Summer university held in Paris, France.

Godfrey, A. J. R. (2015). While my base R gently weeps. Presentation delivered on xx July 2014 at the useR!2015 conference held in Aalborg, Denmark.

Godfrey, A. J. R. (2016a). Introduction to R: the most accessible statistical software for blind students. Workshop was delivered twice on 12 July, 2016 at the 6^{th} Summer University held in Linz, Austria.

Godfrey, A. J. R. (2016b). A review of statistical software for blind students. Plenary address delivered on 12 July, 2016 at the 6^{th} Summer University held in Linz, Austria.

Godfrey, A. J. R. and Bilton, T. P. (2016). R markdown: lifesaver or death trap? Presentation delivered on 29 June 2016 at the useR!2015 conference held at Stanford University, California.

Godfrey, A. J. R. and Curtis, J. M. P. (2016). Simple authoring of statistical analyses by and for blind people. In Yamaguchi, K. and Suzuki, M., editors,

Proceedings of The International Workshop on Digitization and E-Inclusion in Mathematics and Science 2016, pages 47–54, Kanegawa, Japan.

Godfrey, A. J. R. and Erhardt, R. (2014). Addendum to "statistical software from a blind person's perspective". *The R Journal*, 6(1):182.

Godfrey, A. J. R. and Loots, M. T. (2014). Statistical software (R, SAS, SPSS, and Minitab) for blind students and practitioners. *Journal of Statistical Software, Software Reviews*, 58(1):1–25.

Godfrey, A. J. R. and Loots, M. T. (2015). Advice from blind teachers on how to teach statistics to blind students. *Journal of Statistics Education*, 23(3):1–28.

Godfrey, A. J. R. and Murrell, P. (2016). Statistical graphs made tactile. In Yamaguchi, K. and Suzuki, M., editors, *Proceedings of The International Workshop on Digitization and E-Inclusion in Mathematics and Science 2016*, pages 69–74, Kanegawa, Japan.

Godfrey, A. J. R., Warren, D., Murrell, P., Bilton, T., and Sorge, V. (2021). *BrailleR: Improved Access for Blind Users*. R package version 0.33.0.

Grolemund, G. and Wickham, H. (2016). *R for Data Science*. O'Reilly Media.

Gross, J. and Ligges, U. (2015). *nortest: Tests for Normality*. R package version 1.0-4.

Insightful Corp. (2003). *S-PLUS Version 6.2*. Seattle, WA.

Komsta, L. and Novomestky, F. (2015). *moments: Moments, cumulants, skewness, kurtosis and related tests*. R package version 0.14.

Minitab Inc. (2012). *Minitab Statistical Software Version 16.2.3*. Minitab Inc., State College, PA.

Minitab Inc. (2014). *Minitab Express Statistical Software Version* . Minitab Inc., State College, PA.

Murrell, P. (2015). The gridGraphics package. *The R Journal*, 7(1):151–162.

Murrell, P. and Potter, S. (2014). The gridSVG package. *The R Journal*, 6(1):133–143.

NVDA Team (2018). *NVDA Version 2018.1.1*.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Robinson, D., Hayes, A., and Couch, S. (2021). *broom: Convert Statistical Objects into Tidy Tibbles*. R package version 0.7.10.

Rödiger, S., Friedrichsmeier, T., Kapat, P., and Michalke, M. (2012). RKWard: A comprehensive graphical user interface and integrated development environment for statistical analysis with R. *Journal of Statistical Software*, 49(9):1–34.

RStudio (2018). *RStudio: Integrated Development Environment for R, Version 1.1.453*. RStudio, Boston, MA.

SAS Institute Inc. (2010). *SAS/STAT Software, Version 9.3*. SAS Institute Inc., Cary, NC.

Snow, G. (2020). *TeachingDemos: Demonstrations for Teaching and Learning*. R package version 2.12.

Spinu, V., Grolemund, G., and Wickham, H. (2021). *lubridate: Make Dealing with Dates a Little Easier*. R package version 1.8.0.

SPSS Inc. (2012). *IBM SPSS Statistics 21*. SPSS Inc., Chicago, IL.

Wickham, H. (2009). *ggplot2: Elegant graphics for data analysis*. Springer, New York.

Wickham, H. (2014a). *Advanced R*. CRC Press.

Wickham, H. (2014b). Tidy data. *The Journal of Statistical Software*, 59.

Wickham, H. (2015). *R Packages*. O'Reilly Media.

Wickham, H. (2021). *tidyverse: Easily Install and Load the Tidyverse*. R package version 1.3.1.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., Yutani, H., and Dunnington, D. (2021a). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.3.5.

Wickham, H., François, R., Henry, L., and Müller, K. (2021b). *dplyr: A Grammar of Data Manipulation*. R package version 1.0.7.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2021). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.36.