BrailleR in Action

A. Jonathan R. Godfrey

22 December 2017

Contents

Preface

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 18 September 2017

0.0.1 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.

0.0.2 Copyright information

Chapter 1

Introduction

The BrailleR Project started back in 2011 when 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.

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 6.

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 6 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.

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 extramural 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 (**?**).

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 [**?**; **?**; **?**, and **?**.

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 (**?**) 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 (**?**).

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. (**?**). I now know that the groundwork I had done contributed to my desire to present my workflow as a workshop at the 5th Summer University in 2014 (**?**). It also stood me in good stead for the work that followed in the way the package 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: -2, 0, 2, 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 15 bins with equal widths, starting at -3.5 and ending at 4 .
## The mids and counts for the bins are:
## mid = -3.25  count = 2
## mid = -2.75  count = 5
## mid = -2.25  count = 16
```

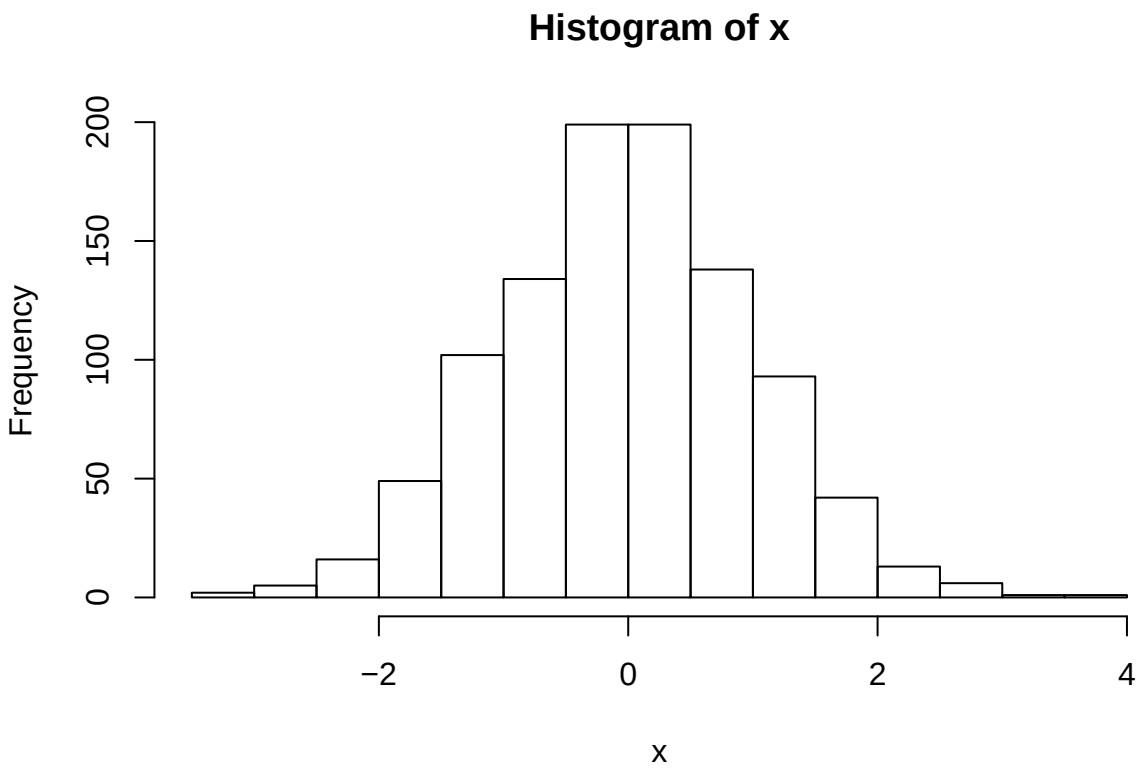


Figure 2.1: A histogram of 1000 random values from a normal distribution

```
## mid = -1.75  count = 49
## mid = -1.25  count = 102
## mid = -0.75  count = 134
## mid = -0.25  count = 199
## mid = 0.25  count = 199
## mid = 0.75  count = 138
## mid = 1.25  count = 93
## mid = 1.75  count = 42
## mid = 2.25  count = 13
## mid = 2.75  count = 6
## 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 called the stored set of numbers 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 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 (**?**). 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 **?** which led to an addendum (**?**).

I presented some of my work via a poster (**?**) 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 **?**).

2.5 Review of 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 lecturer of statistics (Theodor Loots, University of Pretoria) to compare the most commonly used statistical software for its accessibility (**?**). I summarised this paper at the 5th Summer University event (**?**), and offered a similar presentation at the 6th Summer University event (**?**) with a few updates.

2.6 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 (**?**). Perhaps the most valuable outcome of this conference was the ability to attend a tutorial on use of the knitr package 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 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 (**?**). In 2016, I presented the associated work on writing R markdown documents for blind users could be done, and how blind authors might also therefore write their own documents (**?**).

2.7 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 has improved some of my earlier work and tried a few things of his own. This has 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, 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. This is now beyond experimental but there is still more to do on making this application truly useful (**?**).

2.8 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 (**?**).

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, and James Curtis.

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.

2.9 References

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 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")
```

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!

3.1.1 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 blah 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.

Itis 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.

3.2 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:

3.2.1 The document converter — pandoc

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

3.2.2 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

3.2.3 One programming language — Python

WriteR is a simple text editor written in wxPython that needs Python27 and wxPython. Unfortunately, they are two separate downloads at present. 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 #WriteR

3.3 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 a later chapter.

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 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.

```
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, and 3
There are a total of 1000 elements for this variable.
Tick marks for the y-axis are at: 0, 50, 100, and 150
It has 13 bins with equal widths, starting at -3.5 and ending at 3 .
The mids and counts for the bins are:
mid = -3.25  count = 1
mid = -2.75  count = 4
mid = -2.25  count = 14
mid = -1.75  count = 50
mid = -1.25  count = 81
mid = -0.75  count = 163
mid = -0.25  count = 176
mid = 0.25  count = 185
mid = 0.75  count = 158
mid = 1.25  count = 105
mid = 1.75  count = 38
mid = 2.25  count = 20
mid = 2.75  count = 5
```

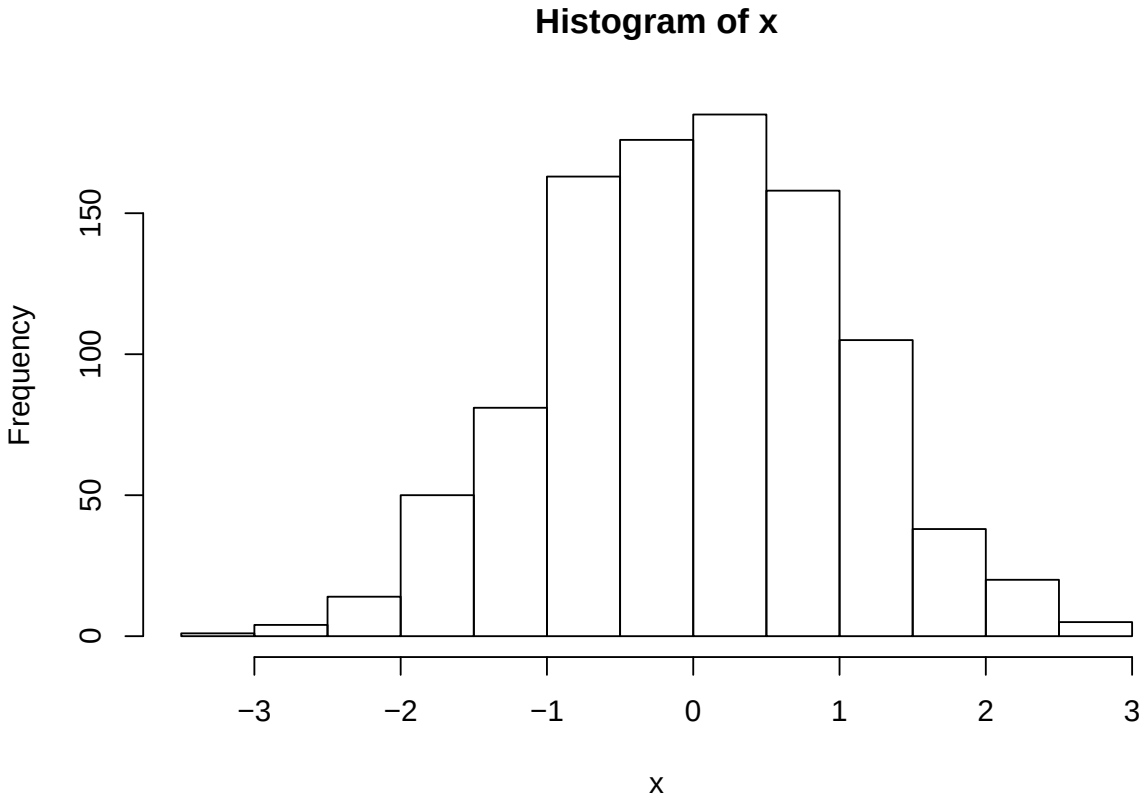


Figure 4.1: A histogram of 1000 random values from a normal distribution

The `VI()` command actually calls the `VI.histogram()` command as the `hist()` command creates an object of class "histogram". This means we can tailor the output to the information needed for any histogram created using the `hist()` command.

4.1.1 Important features

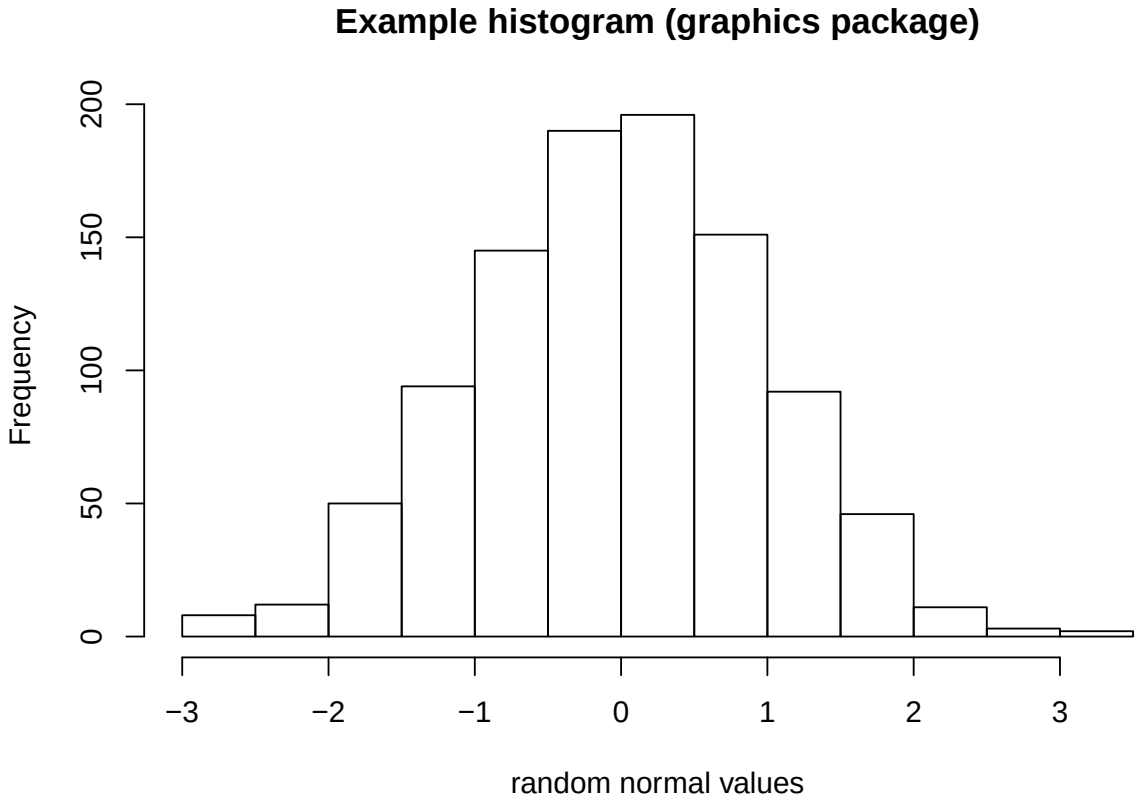
The `VI()` command has added to the impact of issuing the `hist()` command as the actual graphic is generated for the sighted audience. The blind user can read from the text description so that they can interpret the information that the histogram offers the sighted world.

The above example showed the standard implementation of the `hist()` function. The `hist()` function of the `graphics` package does not store the additional arguments that improve the visual attractiveness. The solution (perhaps temporary) is to mask the original function with one included in the `BrailleR` package that calls the `graphics` package function, and then adds extra detail for any added plotting arguments.

This is best illustrated using the example included in the `BrailleR::hist()` function.

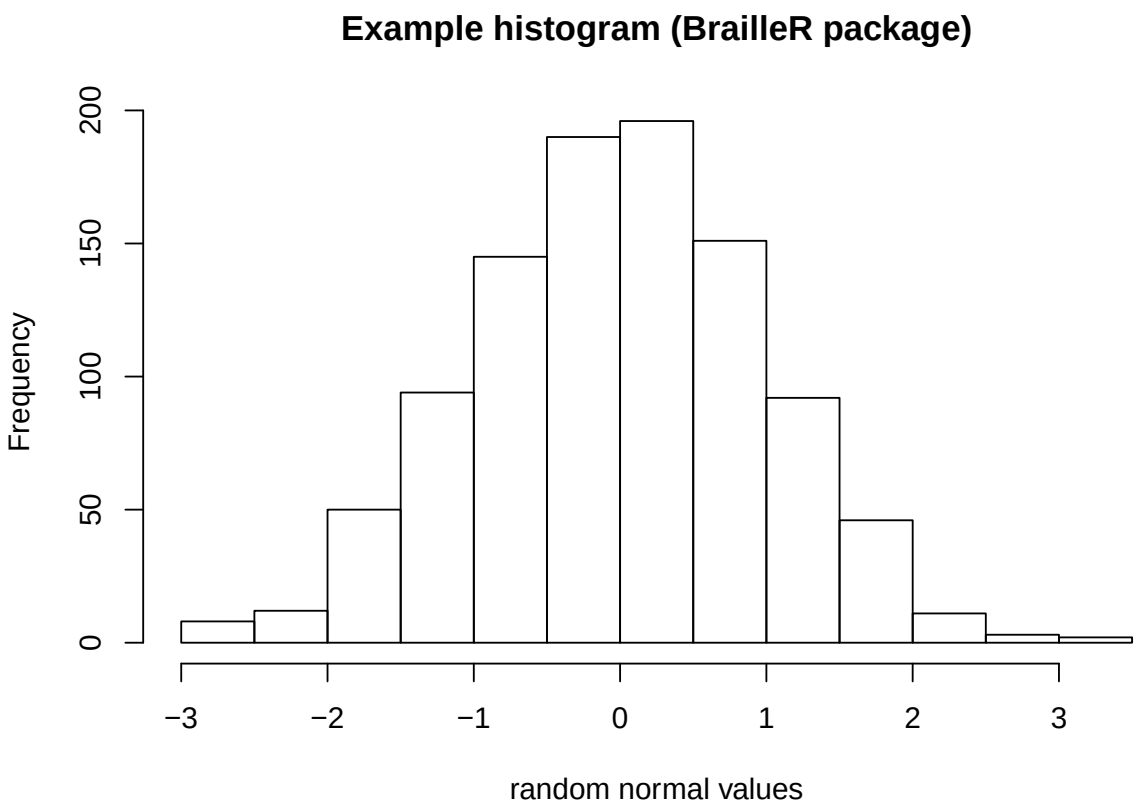
```
example(hist)
```

```
hist> x=rnorm(1000)

hist> # the stamdard hist function returns
hist> MyHist=graphics::hist(x, xlab="random normal values", main="Example histogram (graphics package)"
```




```
hist> #dev.off()
```

```
hist> MyHist
$breaks
 [1] -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]   8  12  50  94 145 190 196 151  92  46  11   3   2

$density
 [1] 0.016 0.024 0.100 0.188 0.290 0.380 0.392 0.302 0.184 0.092 0.022
[12] 0.006 0.004

$mids
 [1] -2.75 -2.25 -1.75 -1.25 -0.75 -0.25  0.25  0.75  1.25  1.75  2.25
[12]  2.75  3.25

$xname
[1] "x"

$equidist
[1] TRUE

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

hist> # while this version returns
hist> MyHist=hist(x, xlab="random normal values", main="Example histogram (BrailleR package)")
```




```
hist> #dev.off()
hist> MyHist
$breaks
 [1] -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]   8  12  50  94 145 190 196 151  92  46  11   3   2

$density
 [1] 0.016 0.024 0.100 0.188 0.290 0.380 0.392 0.302 0.184 0.092 0.022
[12] 0.006 0.004

$mids
 [1] -2.75 -2.25 -1.75 -1.25 -0.75 -0.25  0.25  0.75  1.25  1.75  2.25
[12]  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] 13

$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"

hist> # The VI() method then uses the extra information stored
hist> 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 13 bins with equal widths, starting at -3 and ending at 3.5 .
The mids and counts for the bins are:
mid = -2.75  count = 8
mid = -2.25  count = 12
mid = -1.75  count = 50
mid = -1.25  count = 94
mid = -0.75  count = 145
mid = -0.25  count = 190
mid = 0.25  count = 196
```

```
mid = 0.75  count = 151
mid = 1.25  count = 92
mid = 1.75  count = 46
mid = 2.25  count = 11
mid = 2.75  count = 3
mid = 3.25  count = 2
```

4.1.2 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.

4.2 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's 37
```

```
Solar.R: Min. 7   1st Qu. 115.75   Median 205   Mean 185.931506849315   3rd Qu. 258.75   Max. 334   NA's
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
```

4.2.1 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.

4.3 BrailleR commands used in this chapter

The only explicit command from the `BrailleR` package used in this chapter was the `VI()` command.

Chapter 5

New BrailleR commands for making basic graphs

This chapter introduces some of the new commands found in the `BrailleR` package that are substitutes for other 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 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.

5.1.1 Methods and classes

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. 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". This class attribute is stored when many objects are created by R commands and many classes have methods written for them such as `print()`, `summary()`, and `plot()`.

If data is stored with a specified class attribute, such as a time series with class "ts", we will generate different results from employing the methods. 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.

5.1.2 Who cares about classes anyway?

It is a reasonable question to ask. BrailleR cares because the functions written such as the `VI()` command used in Chapter 4 is actually a family of commands. 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.

5.2 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 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
[15]  3.5

$counts
 [1]   1   6  21  40  89 167 175 203 144  85  49  14   4   2

$density
 [1] 0.002 0.012 0.042 0.080 0.178 0.334 0.350 0.406 0.288 0.170 0.098
[12] 0.028 0.008 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
[12]  2.25  2.75  3.25

$xname
[1] "x"

$equidist
[1] TRUE
```

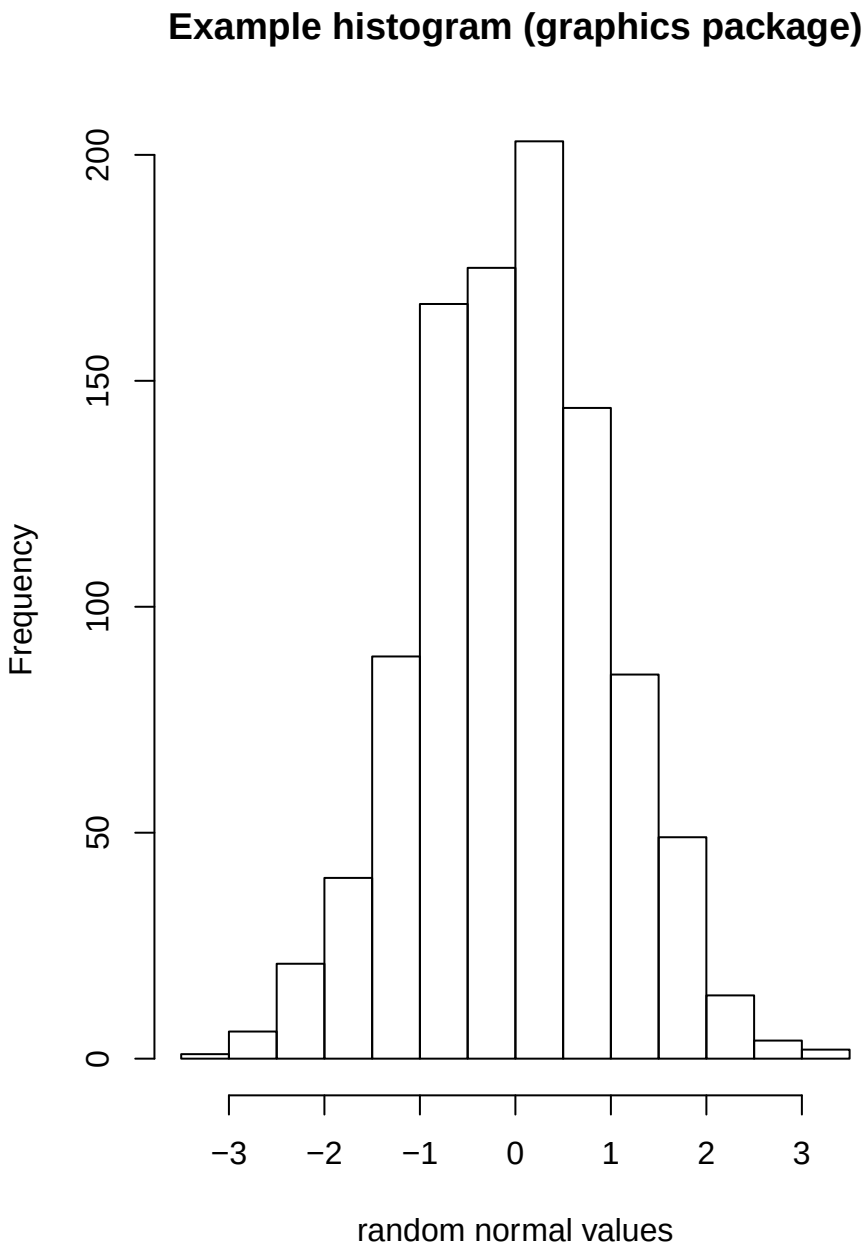


Figure 5.1: testing examples

```
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
[15]  3.5

$counts
 [1]   1   6  21  40  89 167 175 203 144  85  49  14   4   2

$density
 [1] 0.002 0.012 0.042 0.080 0.178 0.334 0.350 0.406 0.288 0.170 0.098
[12] 0.028 0.008 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
[12]  2.25  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
```

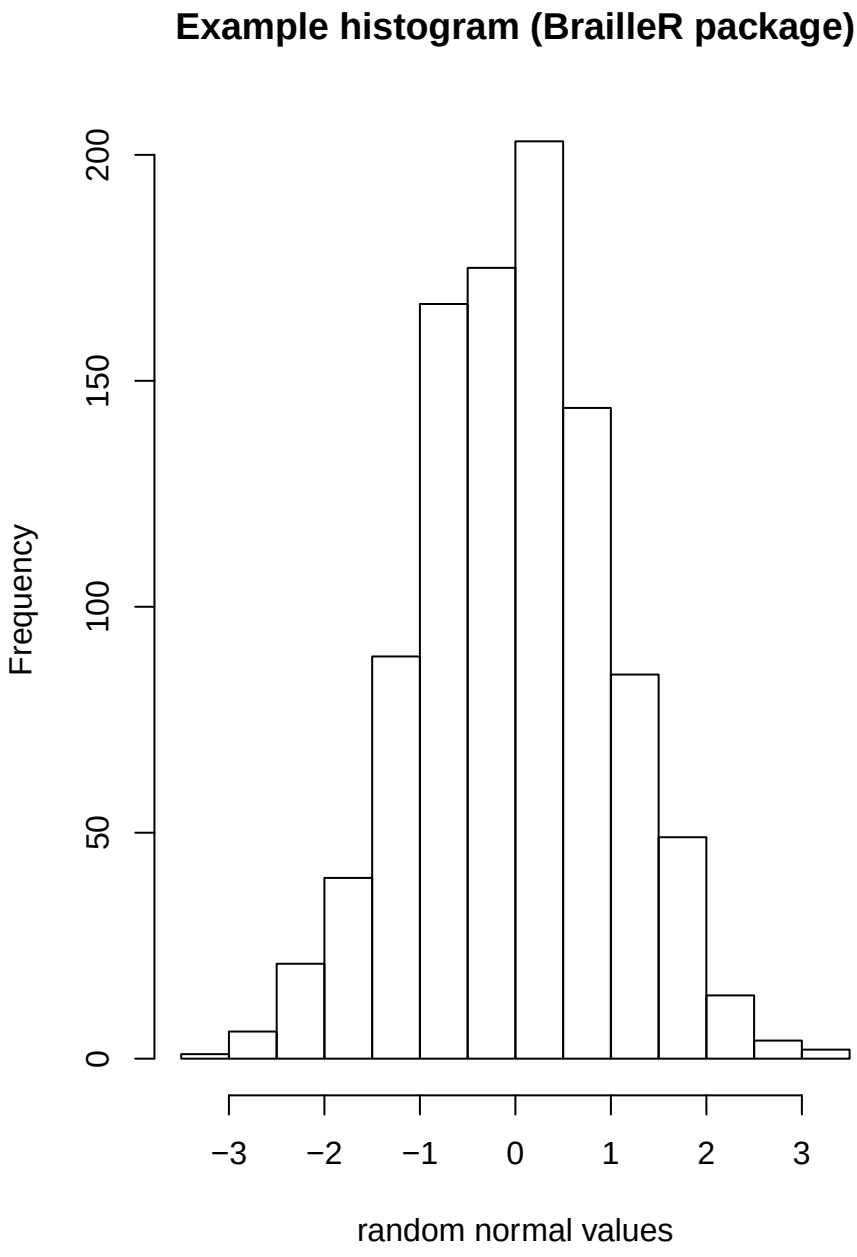


Figure 5.2: testing examples

```
[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 = 6
mid = -2.25  count = 21
mid = -1.75  count = 40
mid = -1.25  count = 89
mid = -0.75  count = 167
mid = -0.25  count = 175
mid = 0.25  count = 203
mid = 0.75  count = 144
mid = 1.25  count = 85
mid = 1.75  count = 49
mid = 2.25  count = 14
mid = 2.75  count = 4
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.

5.3 Scatter plots

The description of the `hist()` function given above shows what is possible if a graph is created using a specific function. Many plots 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 slightyly 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(Wind, Ozone, pch = 4)

> test1 = ScatterPlot(Wind, Ozone, pch = 4)

> test1

> plot(Wind, Ozone)

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

> test2 = FittedLinePlot(Wind, Ozone, line.col = 4)

> test2

> par(op)

> detach(airquality)

> rm(test1)

> rm(test2)

> rm(op)
```

5.4 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()`.

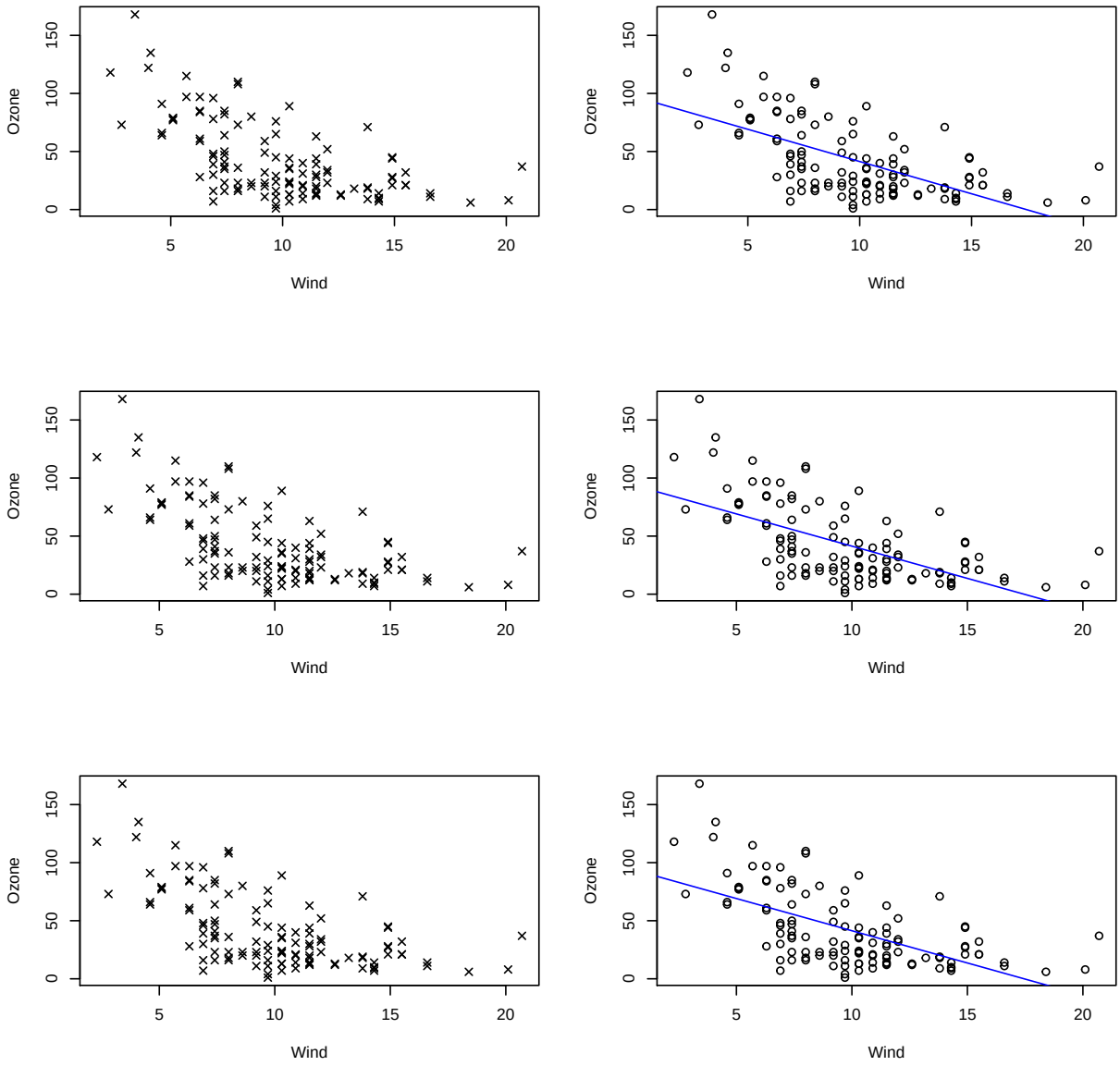


Figure 5.3: The six graphs generated by the `example(ScatterPlot)`

Chapter 6

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.

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.

6.1 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.

This HTML file is opened automatically if R is being used interactively, giving the blind user immediate access to the information. The content is 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, which are given an "alt tag" when they are included in the HTML file, can be presented using a text description available from the `VI()` functionality of the `BrailleR` package.

In addition, the blind user may need any/all 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).

While these commands show the use of R markdown in action, they cannot actually be used within a user's R markdown document. 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.

6.2 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 (thanks to the `moments` package),
- a histogram and/or a boxplot,
- a normality (quantile-quantile) plot,
- various tests for normality (thanks to the `nortest` package), and
- tests on the significance of the skewness and kurtosis (again, thanks to the `moments` package).

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.

```
Ozone=airquality$Ozone
UniDesc(Ozone, View=FALSE)
rm(Ozone)
# N.B. Various files and a folder were created in the working directory.
# Please investigate them to see how this function worked.
```

There is one small change to make To get the desired outcome. The argument `View=FALSE` stops the default action which is to open the HTML document automatically. This avoids problems while the BrailleR package is being created. The `UniDesc()` function was designed for interactive use so do not include this argument if you do want this function to open the HTML file automatically.

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.

6.3 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.

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.

```
data(airquality)
```

```
# the following line  returns an error:
## OneFactor("Ozone", "Month", airquality, View=FALSE)
# so we make a copy of the data.frame, and fix that:

airquality2 = airquality
airquality2$Month = as.factor(airquality$Month)
# and now all is good to try:
OneFactor("Ozone", "Month", airquality2)
# N.B. Various files and a folder were created in the working directory.
# Please investigate them to see how this function worked.
```

As before, there is one small change to make To get the desired outcome. The argument `View=FALSE` which stops the HTML document opening automatically needs to be removed.

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.

6.4 Use of BrailleR for linear regression

It is common for sighted users to create a handful of graphs that help them determine the validity of a linear model, even the most basic simple linear regressions. 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. A simple linear regression model might be created and investigated using:

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

```

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

Residuals:
    Min      1Q  Median      3Q     Max
-40.729 -17.409  -0.587  11.306 118.271

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -146.9955    18.2872  -8.038 9.37e-13 ***
Temp           2.4287     0.2331  10.418  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 23.71 on 114 degrees of freedom
  (37 observations deleted due to missingness)
```

```
Multiple R-squared:  0.4877,    Adjusted R-squared:  0.4832
F-statistic: 108.5 on 1 and 114 DF,  p-value: < 2.2e-16
```

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

```
cat(paste("MyModel = lm(Ozone~Temp, data=airquality, echo=FALSE)",
    "VI(MyModel)",
    sep="\n"), file="RunLater.R", append=TRUE)
```

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
Temp with an estimate of 2.428703 and P-Value of 2.931897e-18
```

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.

6.5 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.

```
data(airquality)
OnePredictor("Ozone", "Wind", airquality, View=FALSE)
# N.B. Various files and a folder were created in the working directory.
# Please investigate them to see how this function worked.
```

As before, there is one small change to make To get the desired outcome. The argument `View=FALSE` which stops the HTML document opening automatically needs to be removed.

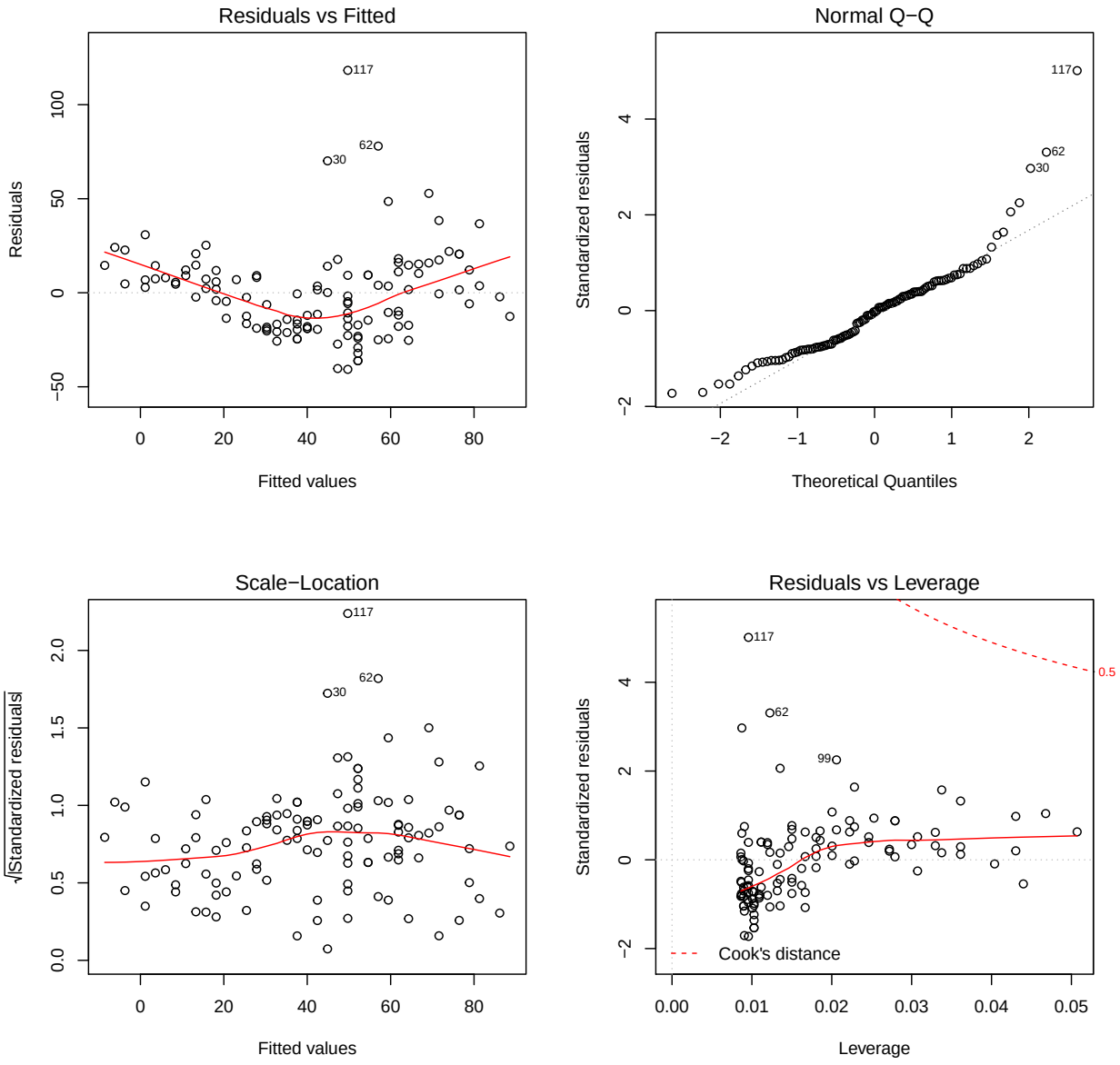


Figure 6.1: (#fig:AirQuality.lm)Diagnostic plots for the simple linear regression model.

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.

6.6 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()`, etc.

Chapter 7

Personalising BrailleR

7.1 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.

7.2 Settings that are about you

7.2.1 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.

7.2.2 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.

7.3 Settings for saving

7.4 BrailleR commands used in this chapter

`SetAuthor()`, `GoBlind()` and `GoSighted()`,

Chapter 8

The ggplot world and BrailleR

The use of the ggplot style of graph production has increased markedly since its inception. The grammar of graphics as seen in the R code used to create the 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.

N.B. the commands here are exact copies of the commands presented in **?** or some minor alterations to them; any changes will be explicitly noted. All plots are created using the figure numbers from **?** or the page numbers if no figure number was given. They are then investigated using the `VI()` command from the `BrailleR` package.

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

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

```
Attaching package: 'ggplot2'
```

```
The following objects are masked from 'package:BrailleR':

    xlab, ylab
```

```
library(magrittr)
```

Note that one data set used for these examples is created by **?** 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 **?** 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
```

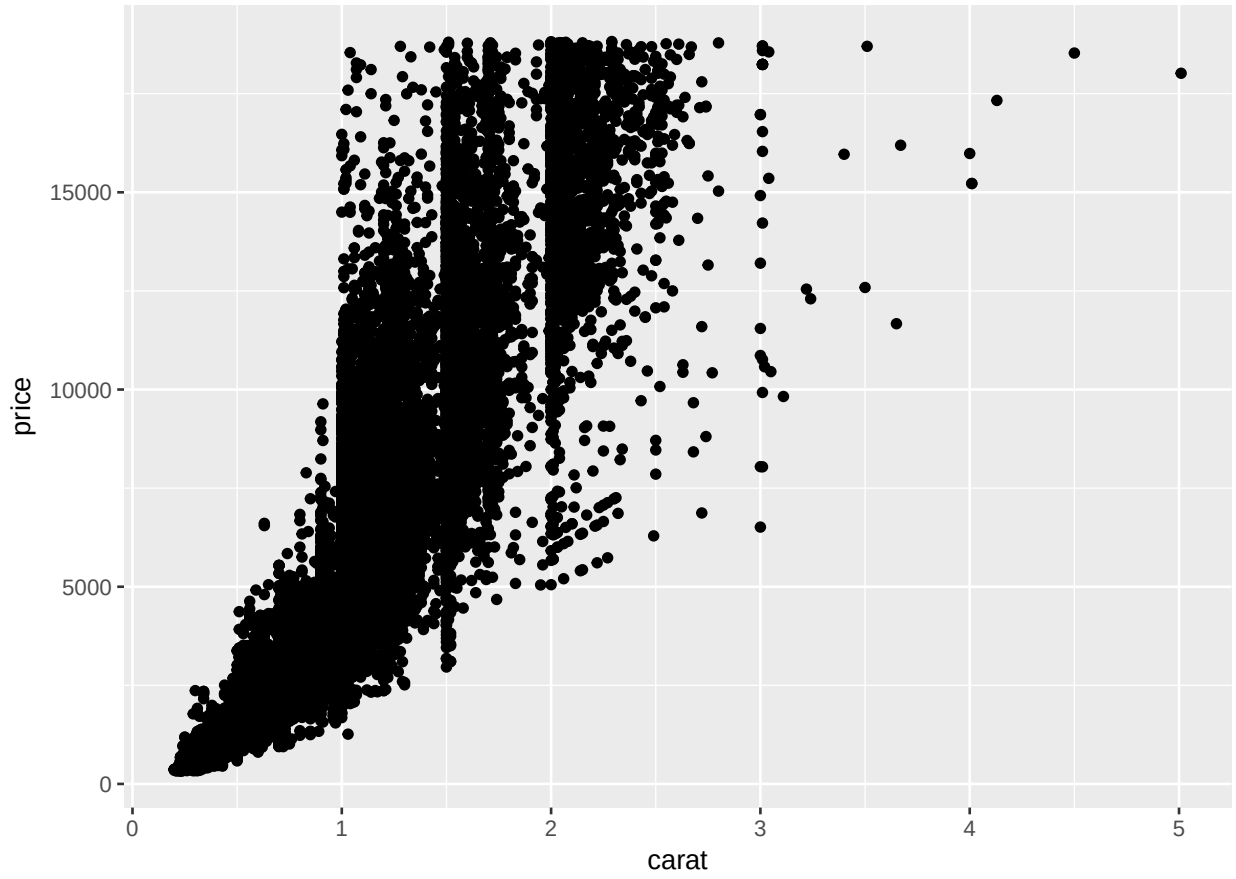


Figure 8.1: First graph on page 11 of **?**

```
VI(p11a)
```

```
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 unlike some other ways the `VI()` command has worked, the graph was not created by the nesting of the call to render the graph when nested inside the `VI()` command. In all the examples that follow, I use the pipe `%>%` operator from the `magrittr` package to push the graph into the `VI()` function.

```
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 0.5, 1.0, 1.5, 2.0 and 2.5.
It has y-axis 'price' with labels 0, 5000, 10000 and 15000.
There is a legend indicating that colour is used to represent color, with 7 levels:
D represented by colour #F8766D,
E represented by colour #C49A00,
F represented by colour #53B400,
G represented by colour #00C094,
H represented by colour #00B6EB,
I represented by colour #A58AFF and
J represented by colour #FB61D7.
The chart is a set of 100 points.
```

```
fig2.2a
```

We haven't been able to tell what exact colour was used in the **?** rendering of this graph, but there has obviously been some minor alteration of the colour palette being used by the `ggplot2` package.

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

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 0.5, 1.0, 1.5, 2.0 and 2.5.
It has y-axis 'price' with labels 0, 5000, 10000 and 15000.
There is a legend indicating that shape is used to represent cut, with 5 levels:
Fair represented by shape solid circle,
Good represented by shape solid triangle,
Very Good represented by shape solid square,
Premium represented by shape plus and
Ideal represented by shape boxed X.
The chart is a set of 100 points.
```

```
fig2.2b
```

To get semi-transparent points:

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

```
fig2.3b %>% VI()
```

```
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.
```

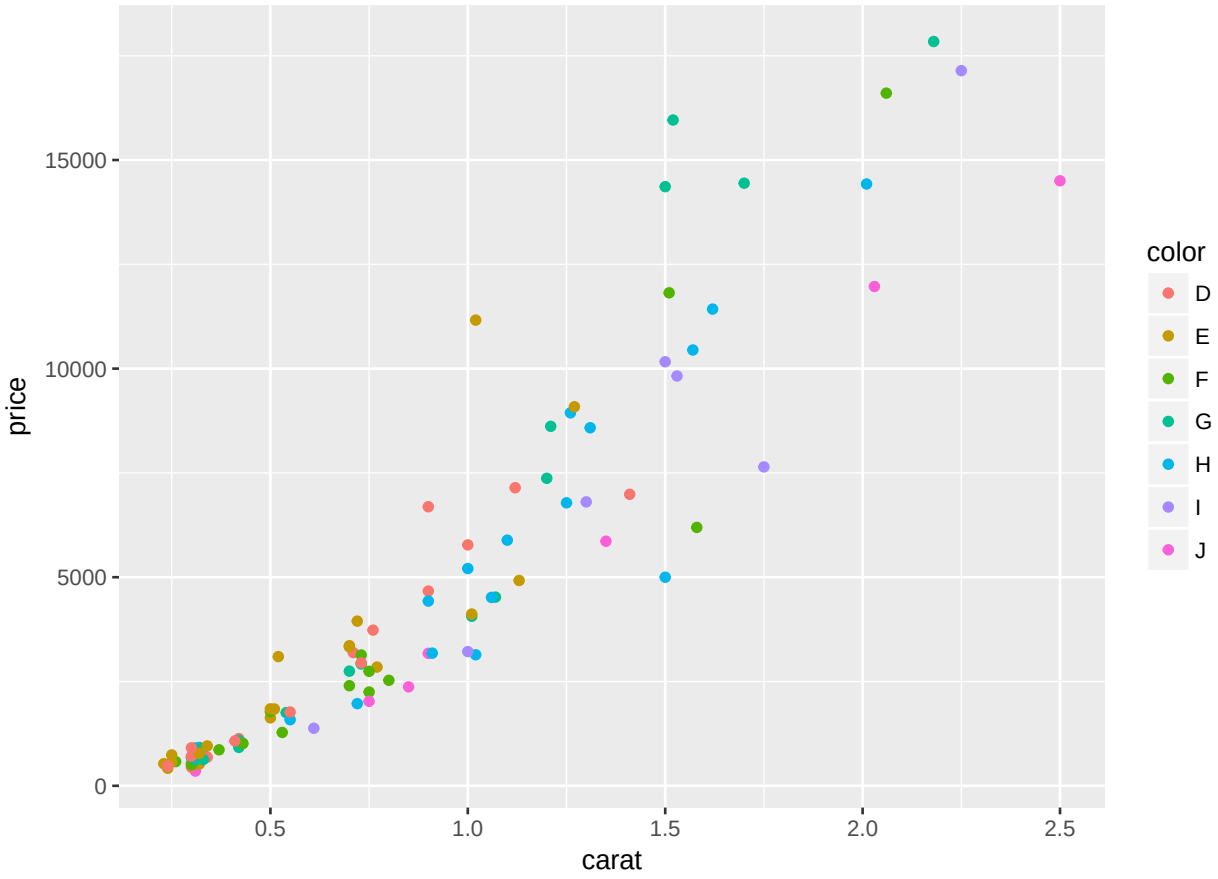


Figure 8.2: Left pane of Figure 2.2 of **?**

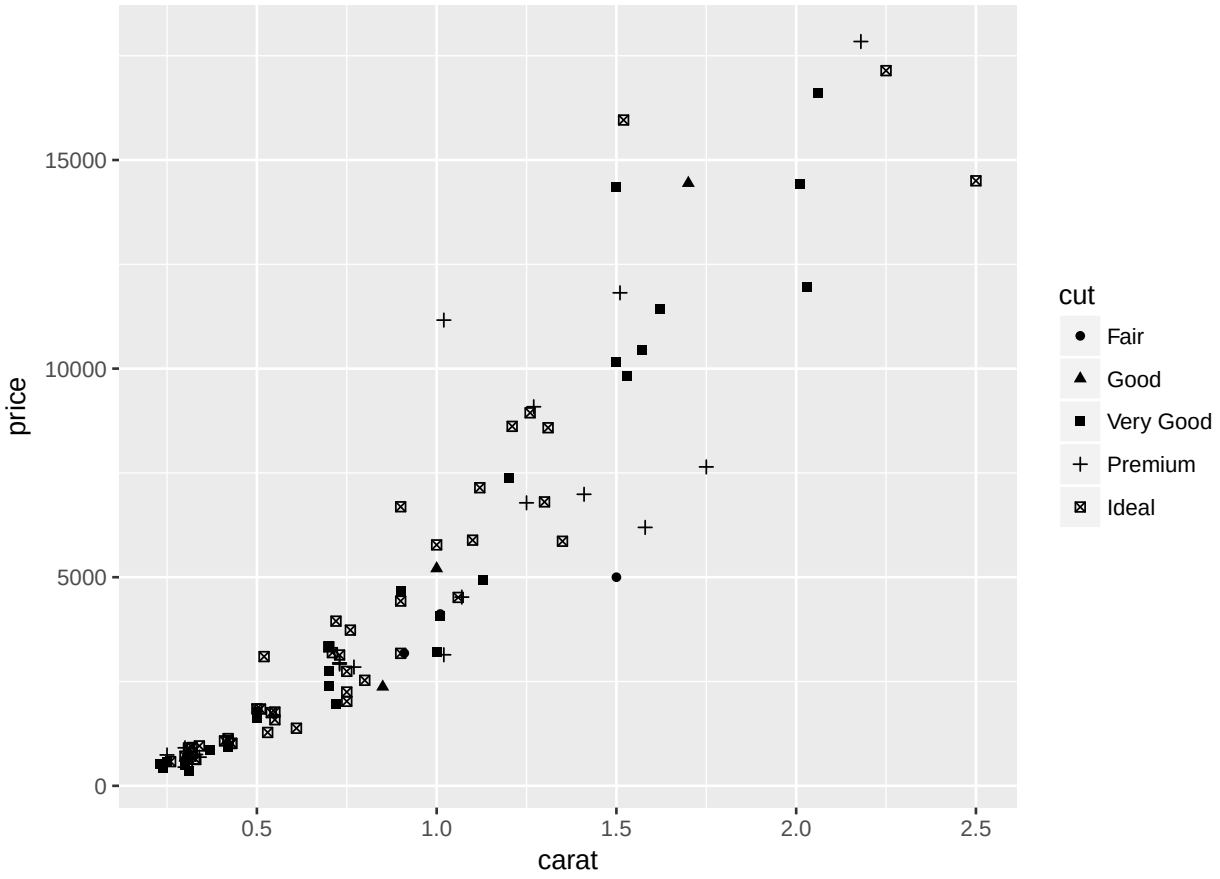


Figure 8.3: Right pane of Figure 2.2

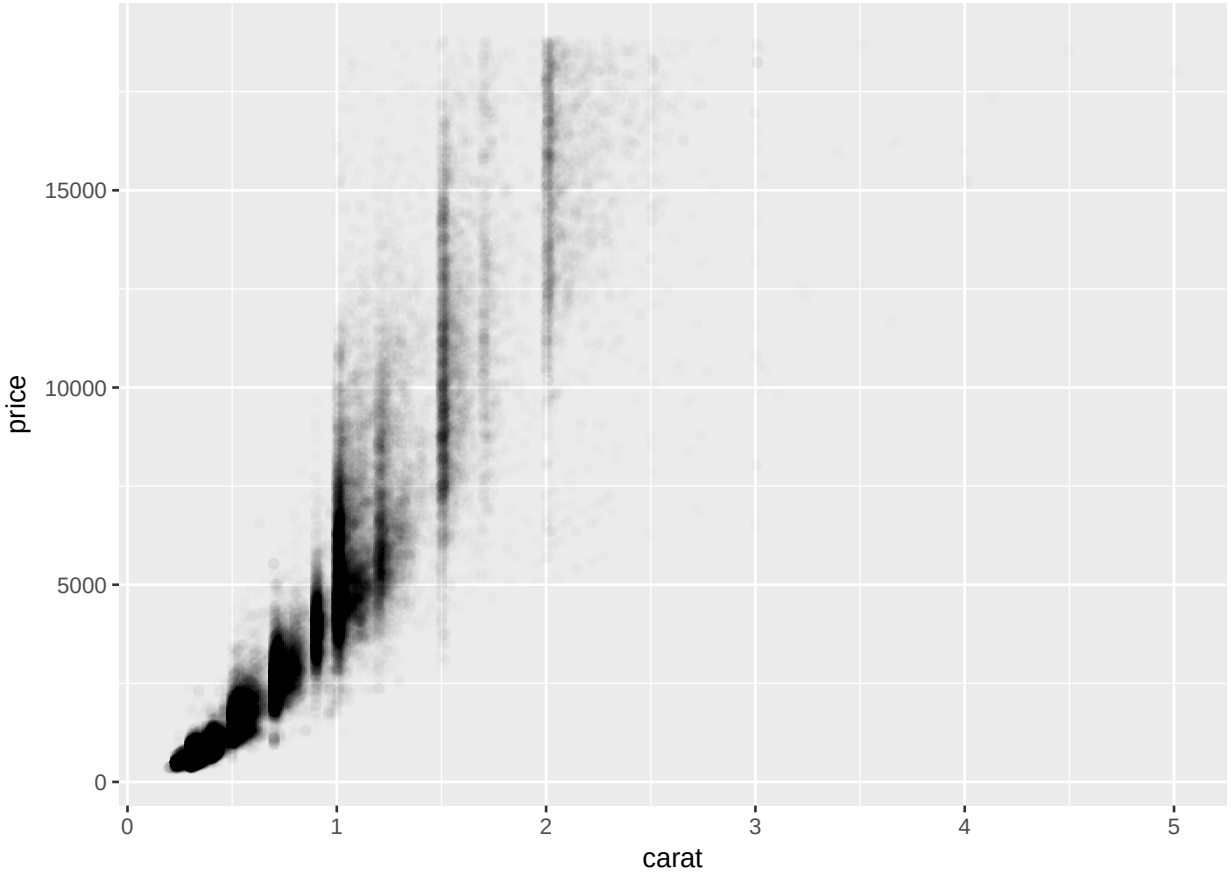


Figure 8.4: Middle pane from Figure 2.3

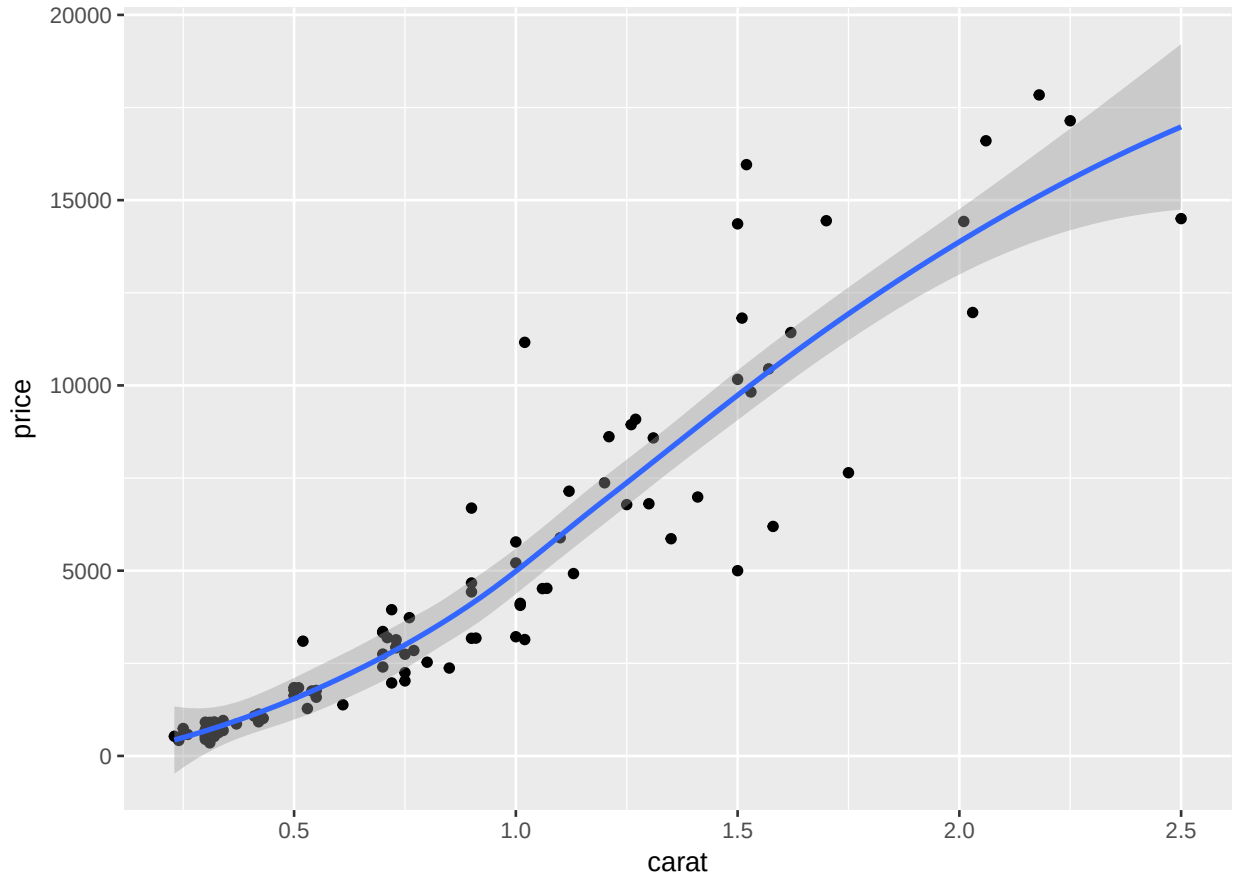


Figure 8.5: Left pane of Figure 2.4

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

```
fig2.4a %>% VI()
```

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 0.5, 1.0, 1.5, 2.0 and 2.5.
It has y-axis 'price' with labels 0, 5000, 10000, 15000 and 20000.
It has 2 layers.
Layer 1 is a set of 100 points.
Layer 2 is a smoothed curve using method 'auto' with confidence intervals.
```

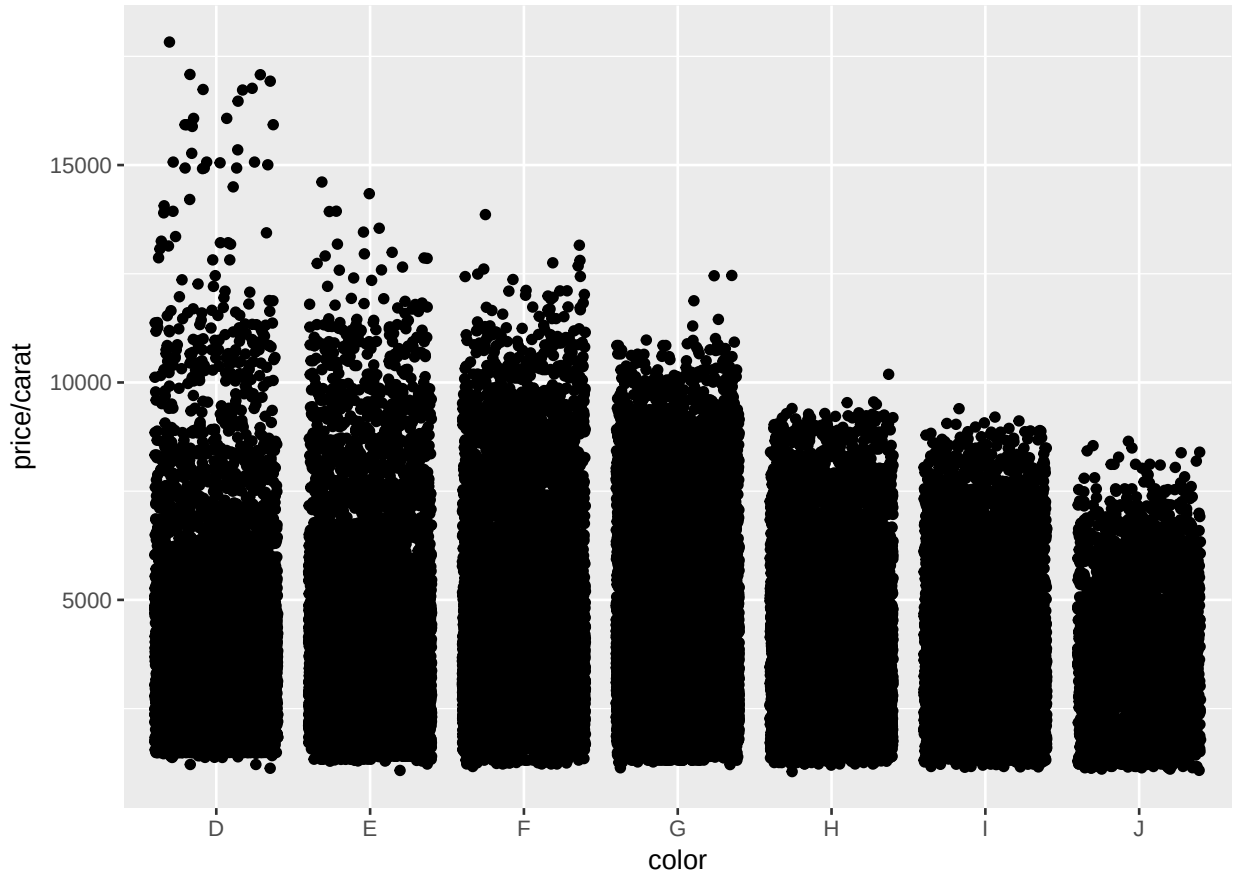


Figure 8.6: Left pane of Figure 2.8

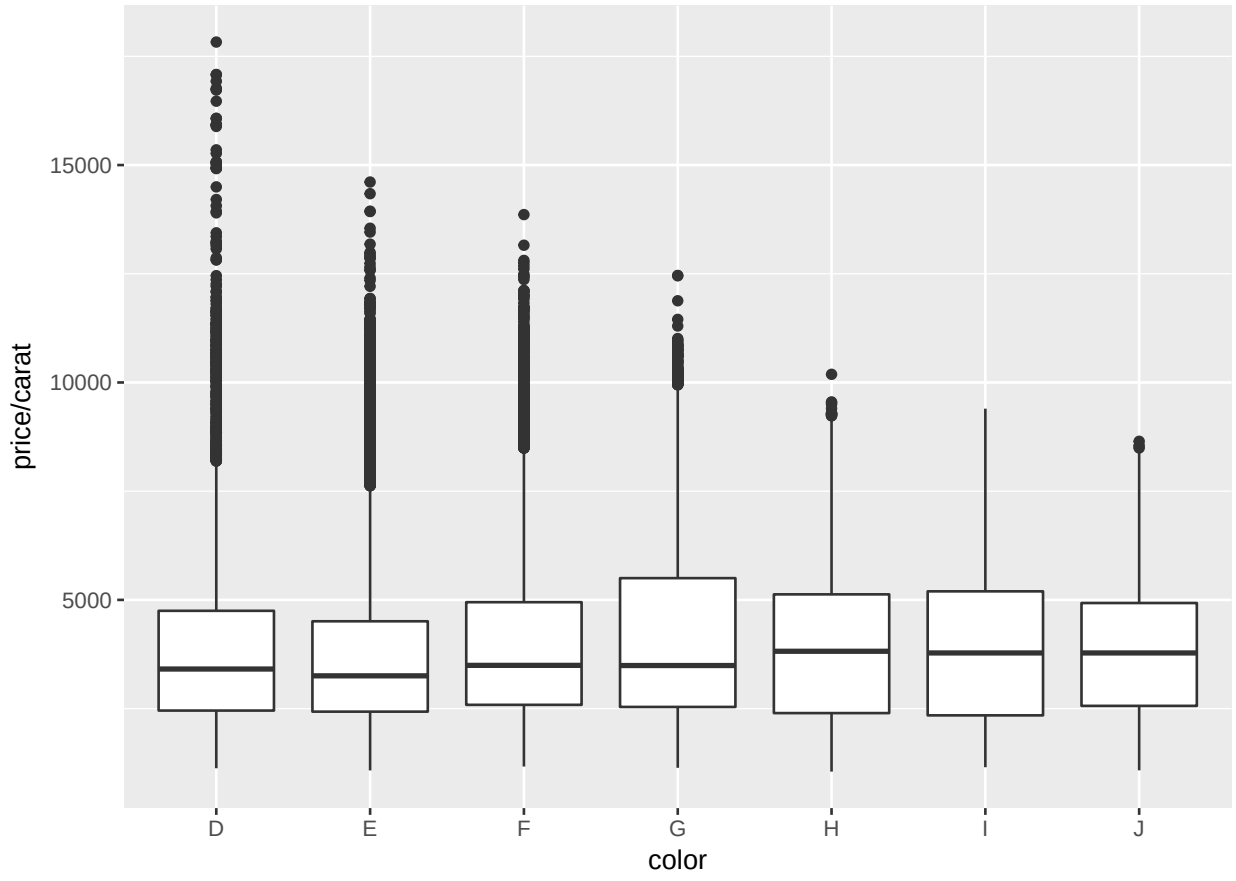
8.1 Plotting a continuous variable against a categorical variable

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

```
fig2.8a %>% VI()
```

```
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
```




```
fig2.8b %>% VI()
```

```
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 8183.3
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 and 7616
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 8477.5.
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 9937.2
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 and 922
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 and 939
There are 0 outliers for this boxplot.
There is a box at x=J.
```

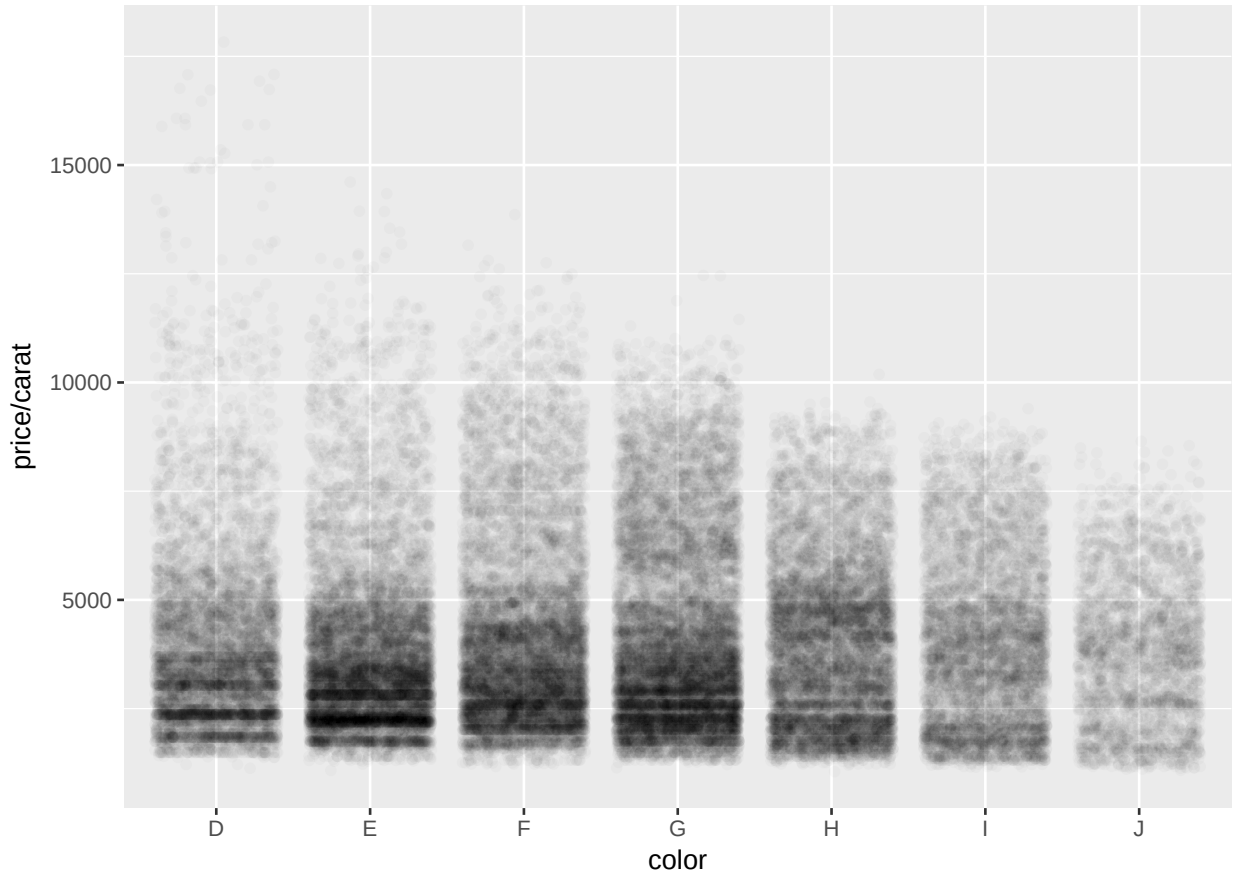


Figure 8.7: Middle pane of Figure 2.9

```
It has median 3780. The box goes from 2562.87 to 4927.95, and the whiskers extend to 1080.65 and 8426.13
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 realised by `BrailleR`.

```
fig2.9b = qplot(color, price / carat, data = diamonds, geom = "jitter", alpha = I(1 / 50))
fig2.9b
```

```
fig2.9b %>% VI()
```

```
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.
The chart has alpha set to 0.02.
```

8.1.1 univariate plots

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

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

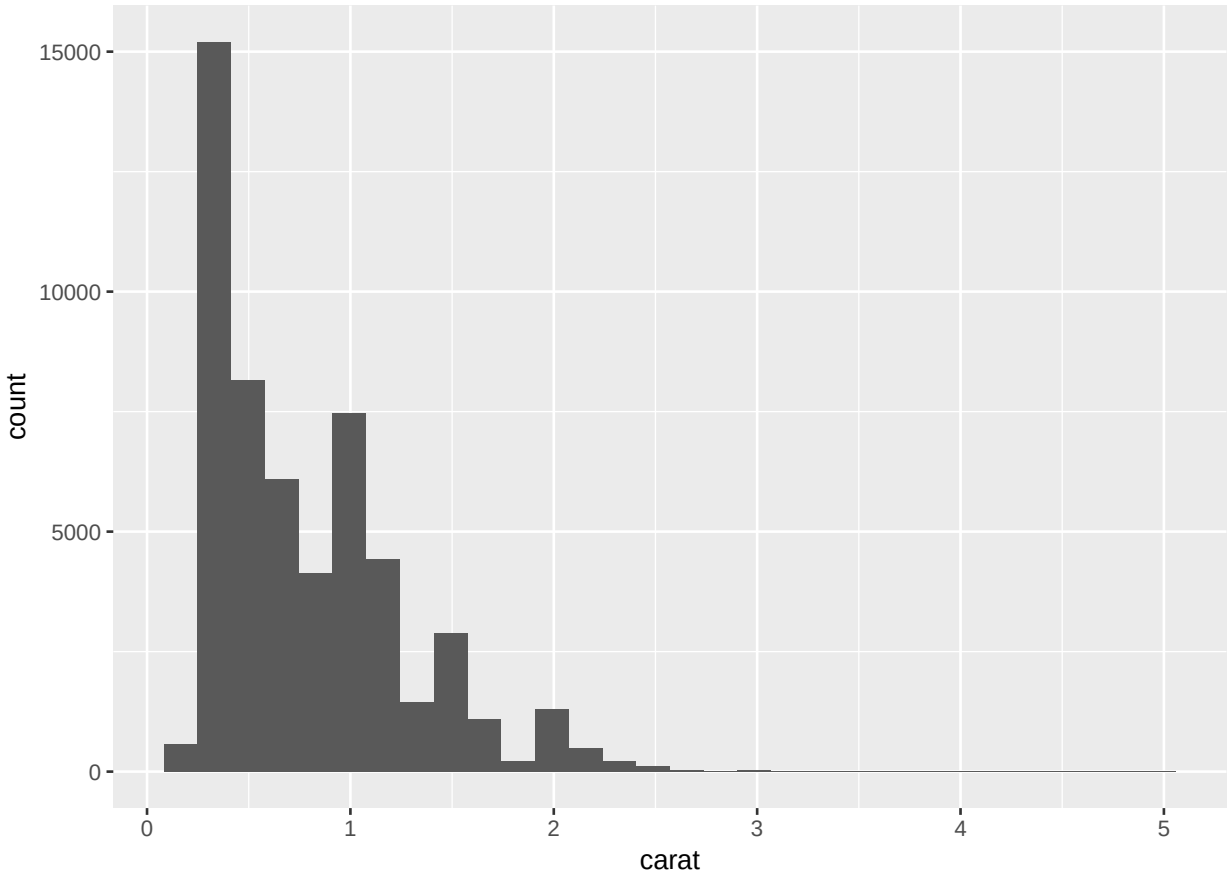


Figure 8.8: Left pane of Figure 2.10

```
fig2.10a %>% VI()
```

```
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 'count' with labels 0, 5000, 10000 and 15000.
The chart is a bar chart containing 30 vertical bars.
```

Warning: This figure does look different to the original in **?** ins spite of using the same code and same data.

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

```
fig2.10b %>% VI()
```

```
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 'density' with labels 0.0, 0.5, 1.0 and 1.5.
The chart is a type that VI isn't able to 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).
```

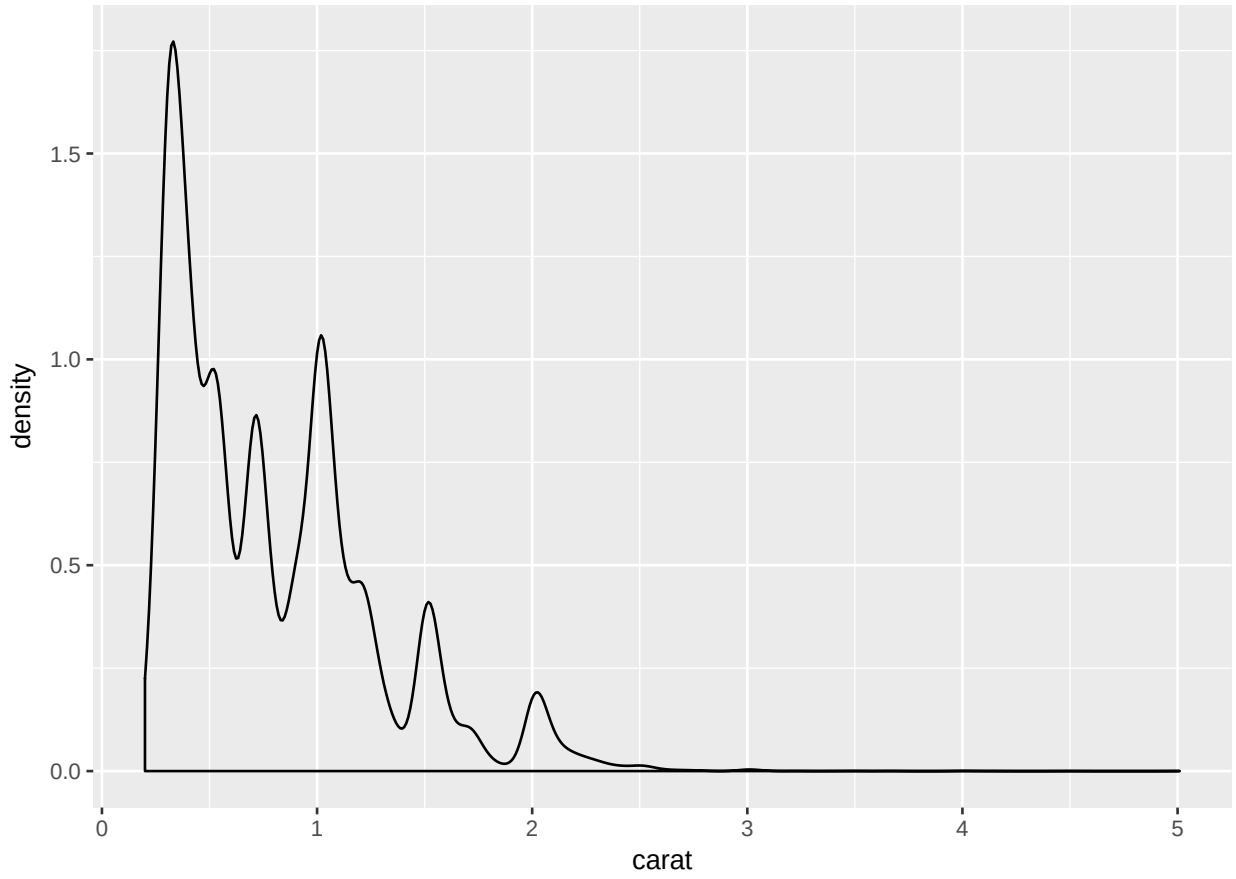


Figure 8.9: Right pane of Figure 2.10

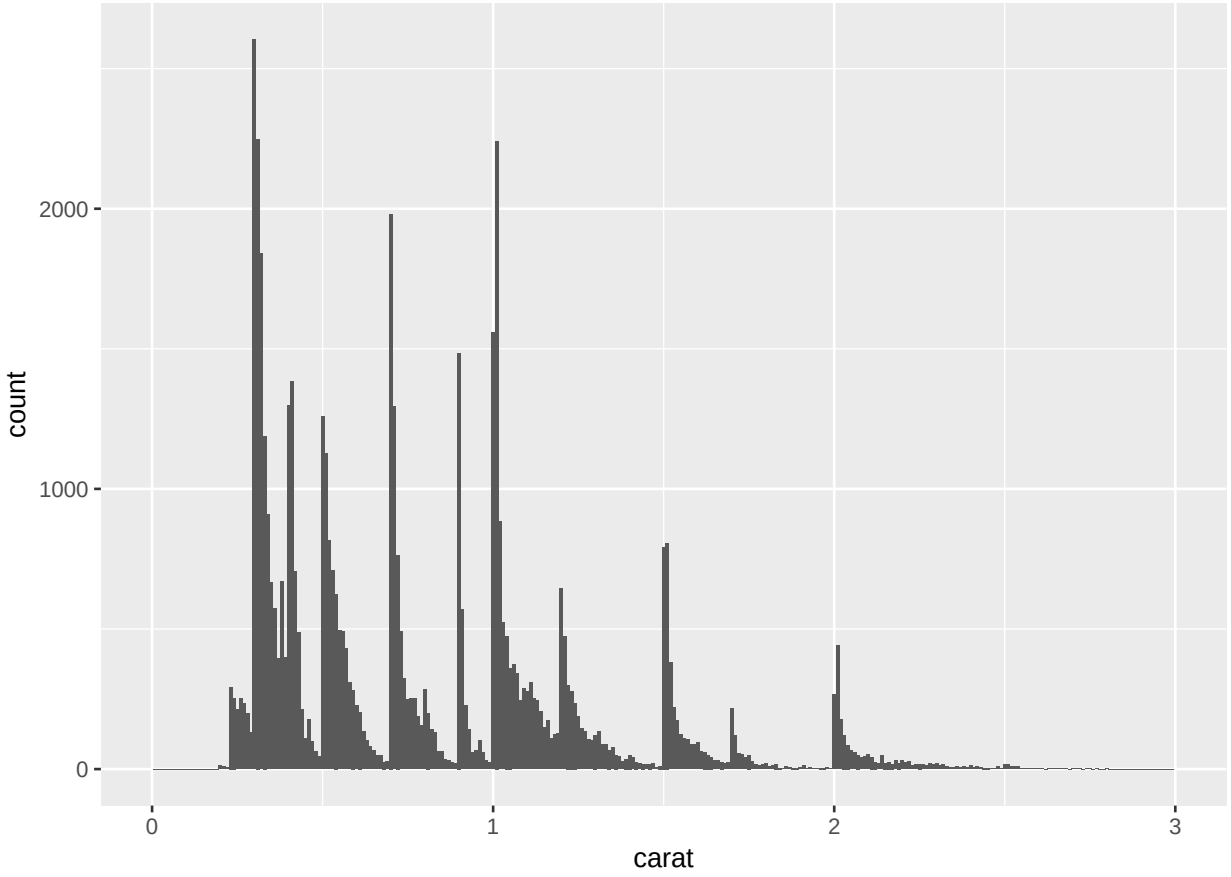


Figure 8.10: Right pane of Figure 2.11

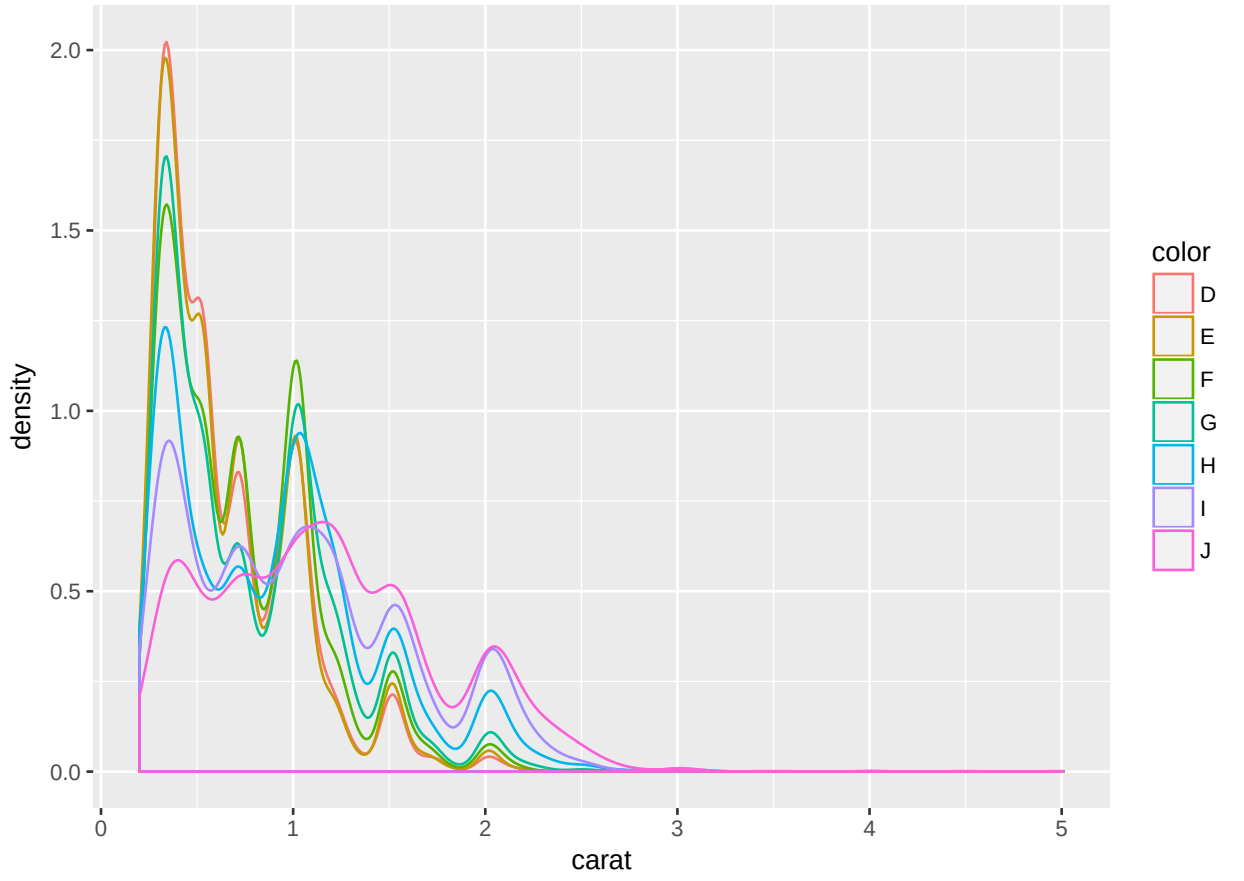


Figure 8.11: Left pane of Figure 2.12

```
fig2.11c %>% VI()
```

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

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 'count' with labels 0, 1000 and 2000.
The chart is a bar chart containing 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 prodcued in **?**.

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

```
fig2.12a %>% VI()
```

```
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 'density' with labels 0.0, 0.5, 1.0, 1.5 and 2.0.
There is a legend indicating that colour is used to represent color, with 7 levels:
D represented by colour #F8766D,
E represented by colour #C49A00,
F represented by colour #53B400,
```

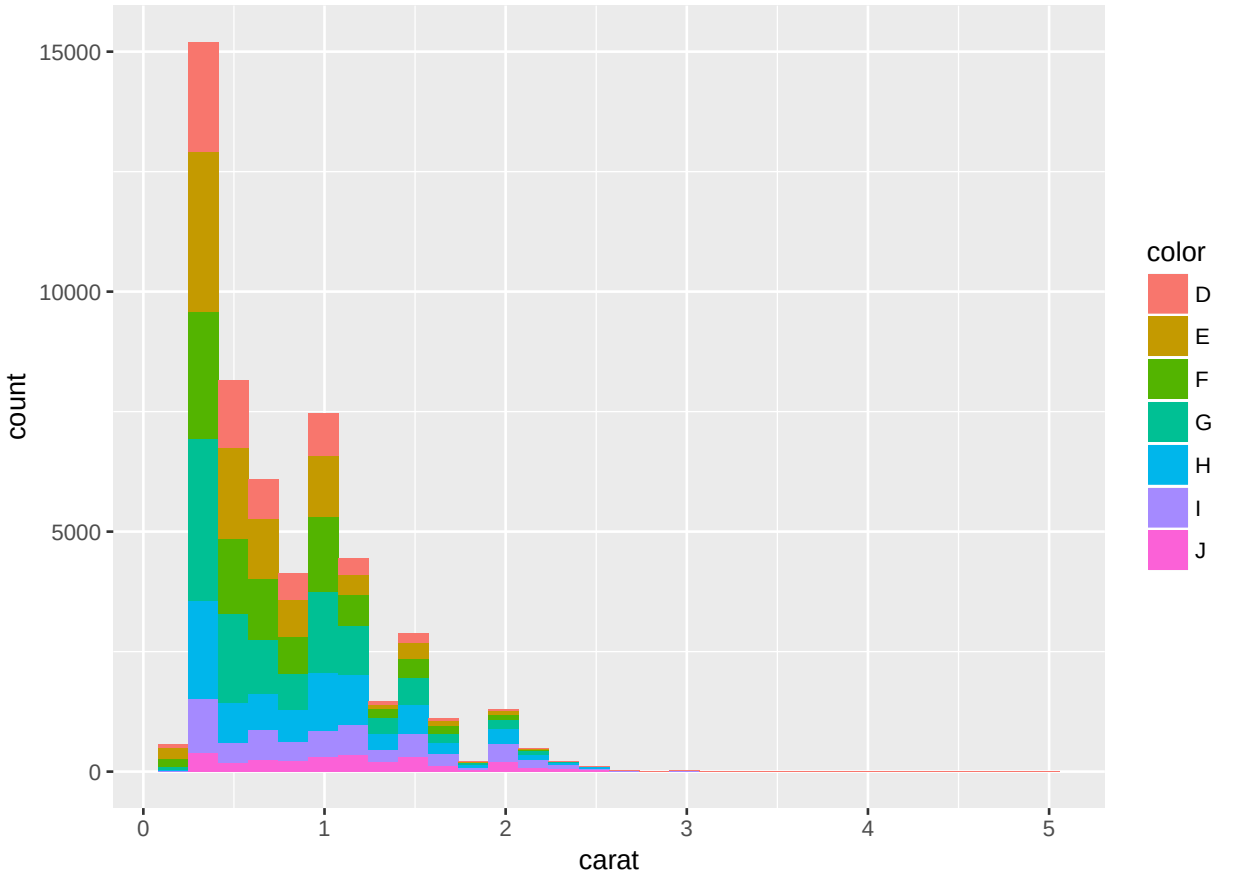


Figure 8.12: Right pane of Figure 2.12

```
G represented by colour #00C094,
H represented by colour #00B6EB,
I represented by colour #A58AFF and
J represented by colour #FB61D7.
The chart is a type that VI isn't able to process.
```

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

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

```
fig2.12b %>% VI()
```

```
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 'count' with labels 0, 5000, 10000 and 15000.
There is a legend indicating that fill is used to represent color, with 7 levels:
D represented by fill #F8766D,
E represented by fill #C49A00,
F represented by fill #53B400,
G represented by fill #00C094,
H represented by fill #00B6EB,
I represented by fill #A58AFF and
J represented by fill #FB61D7.
```

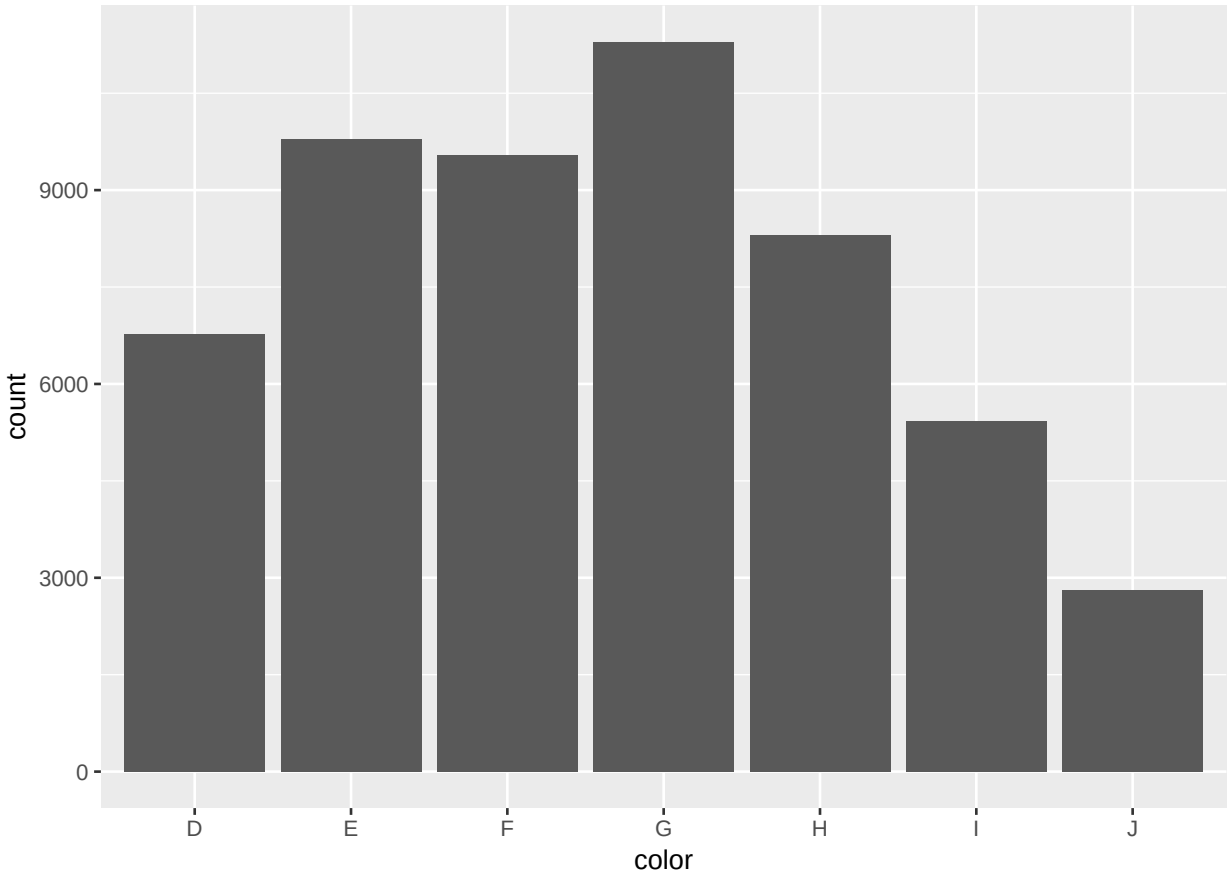


Figure 8.13: Left pane of Figure 2.13

```
The chart is a bar chart containing 210 vertical bars.
```

8.1.2 bar charts for categorical variables

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

```
fig2.13a %>% VI()
```

```
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 'count' with labels 0, 3000, 6000 and 9000.
The chart is a bar chart containing 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...

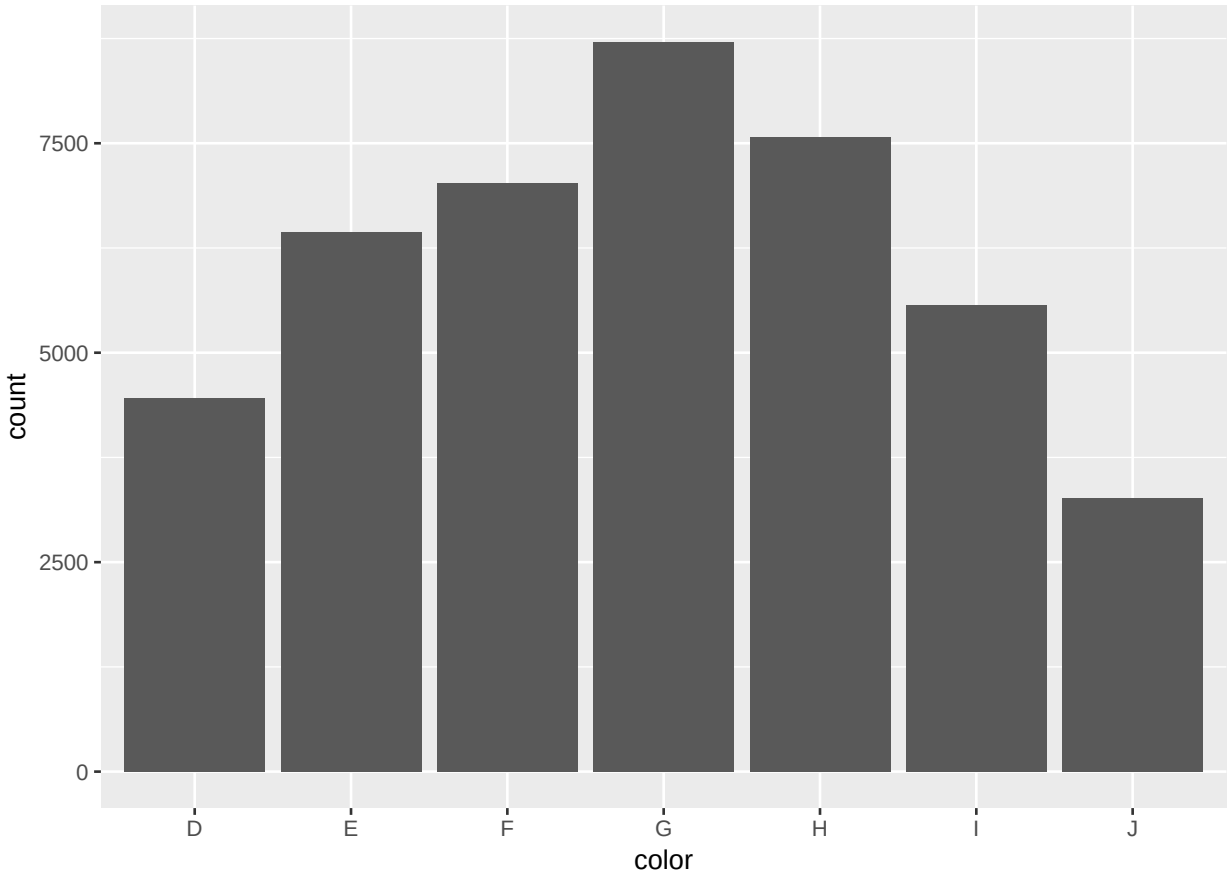


Figure 8.14: Right pane of Figure 2.13

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

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

```
fig2.13b %>% VI()
```

```
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 'count' with labels 0, 2500, 5000 and 7500.
The chart is a bar chart containing 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.
```

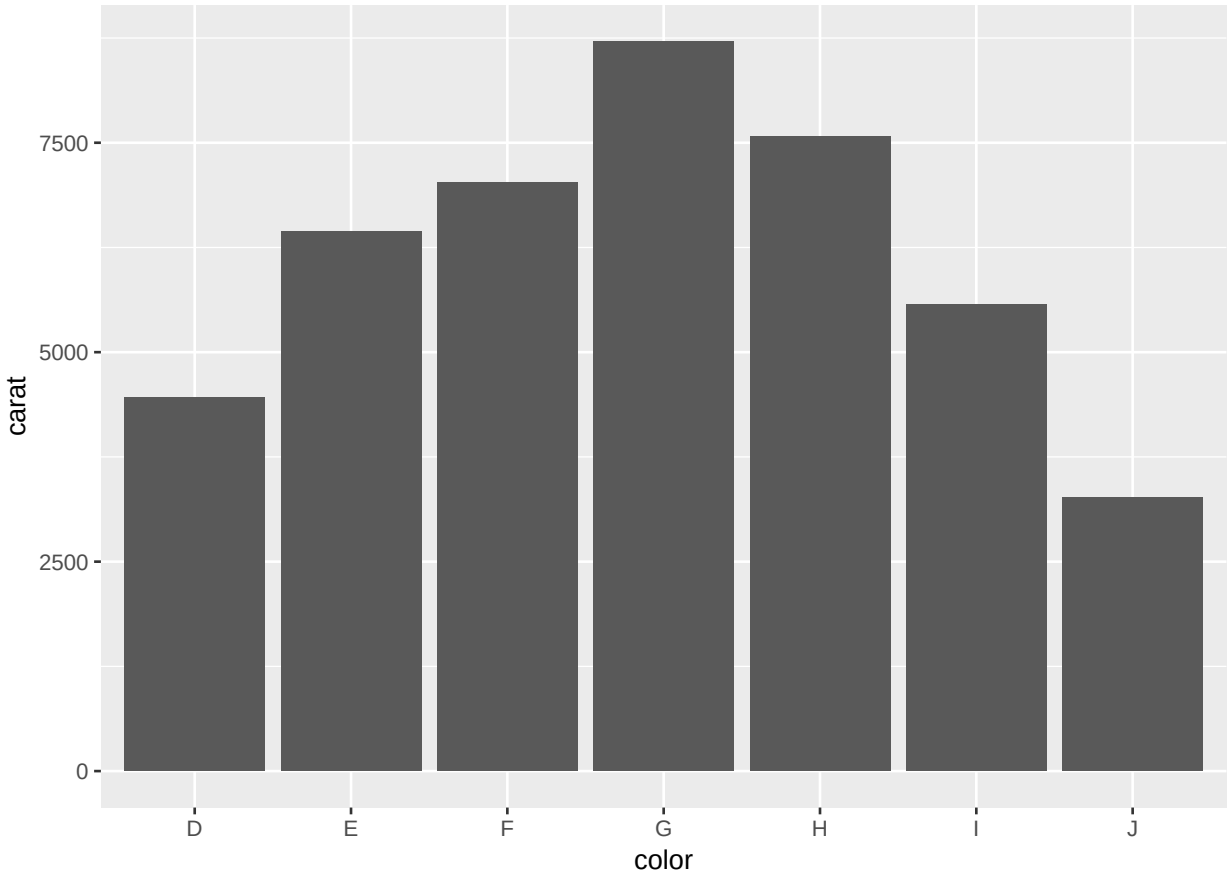


Figure 8.15: Right pane of Figure 2.13

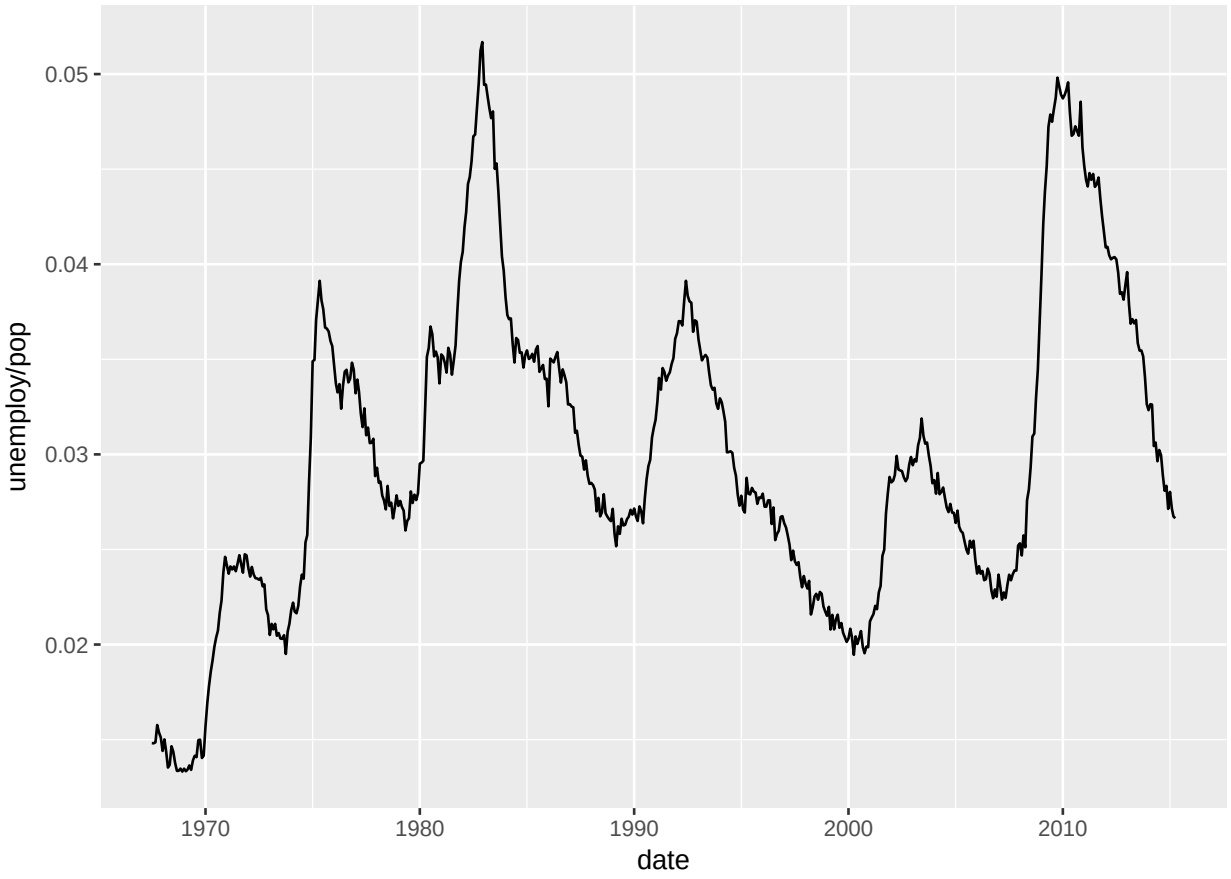


Figure 8.16: Left pane of Figure 2.14

8.2 time series plots

It looks like the data used in the next graph has been updated since the publication of **?**

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

```
fig2.14a %>% VI()
```

```
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.
```

8.3 path plots

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

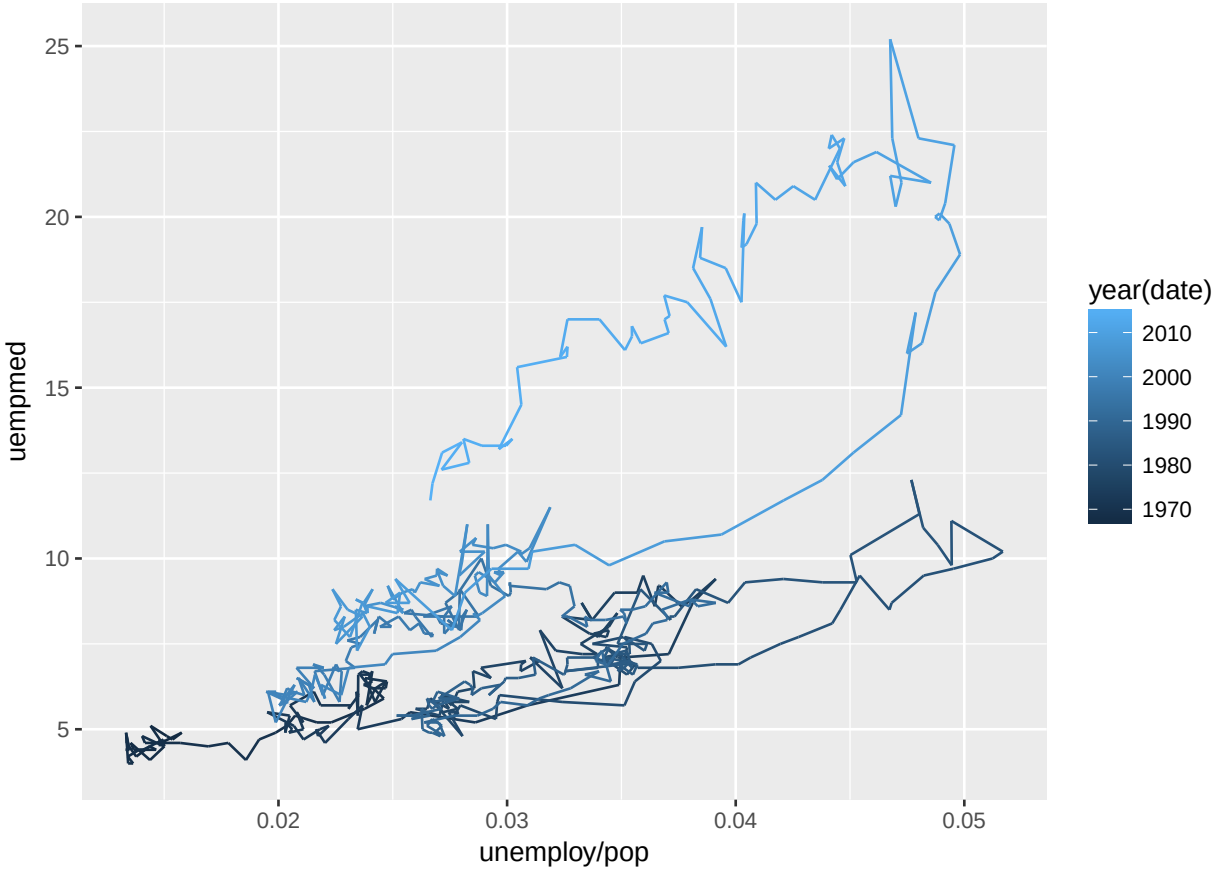


Figure 8.17: Right pane of Figure 2.15

```
fig2.15b %>% VI()
```

```
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 that colour is used to represent year(date), ranging from 1967 represented
The chart is a type that VI isn't able to process.
```

8.4 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 he graph more closely matches that of **?** but it is not an exact match.

```
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).
```

```
fig2.16a %>% VI()
```

```
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 'count' with labels 0, 500, 1000, 1500, 2000 and 2500.
Panel 1 represents data for color = D.
Panel 1 is a bar chart containing 29 vertical bars.
Panel 2 represents data for color = E.
Panel 2 is a bar chart containing 29 vertical bars.
Panel 3 represents data for color = F.
Panel 3 is a bar chart containing 29 vertical bars.
Panel 4 represents data for color = G.
Panel 4 is a bar chart containing 29 vertical bars.
Panel 5 represents data for color = H.
Panel 5 is a bar chart containing 29 vertical bars.
Panel 6 represents data for color = I.
Panel 6 is a bar chart containing 29 vertical bars.
Panel 7 represents data for color = J.
Panel 7 is a bar chart containing 29 vertical bars.
```

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

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

```
fig2.16b %>% VI()
```

```
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.
```

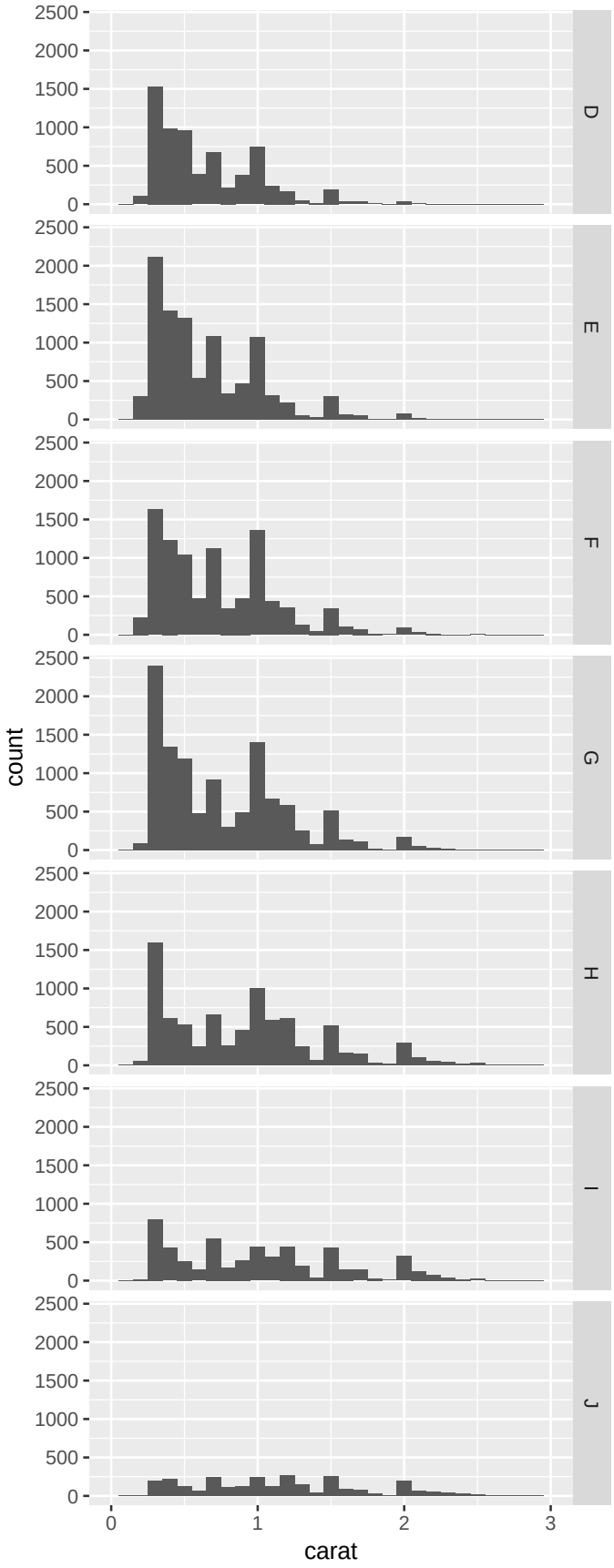


Figure 8.18: Left side of Figure 2.16

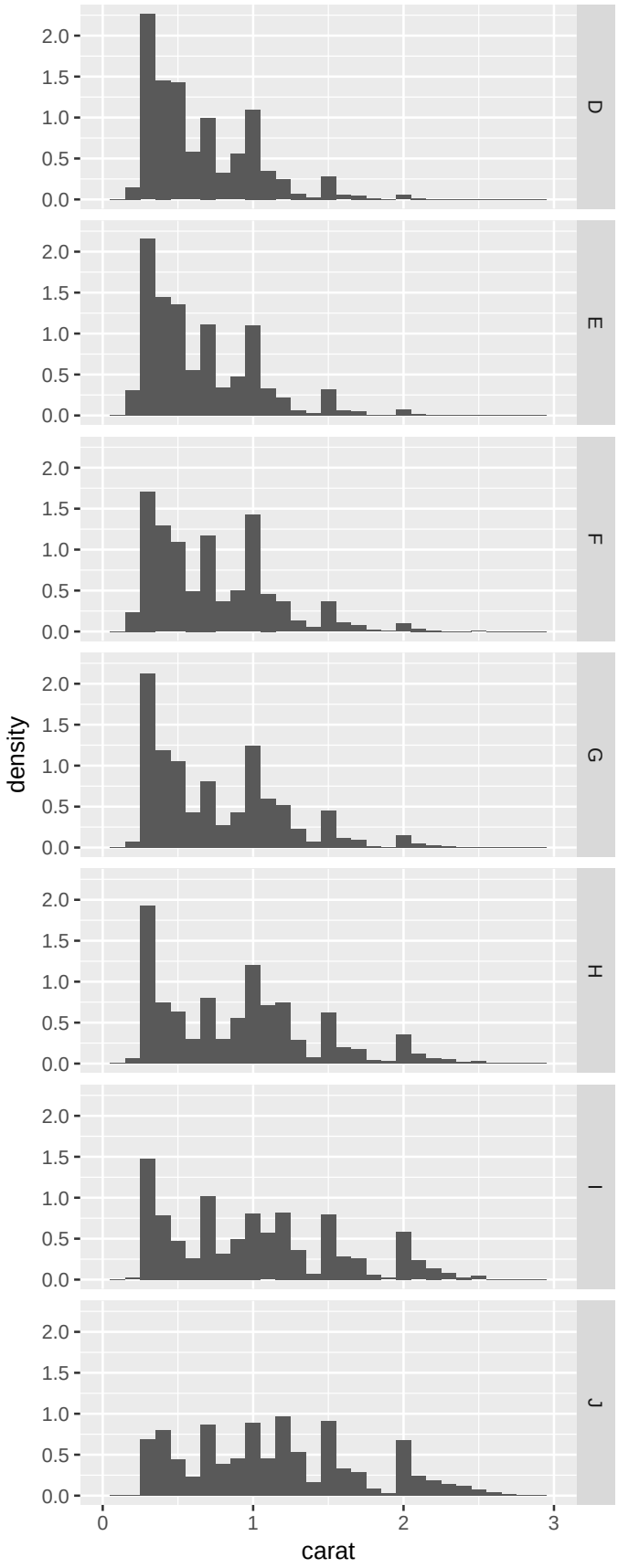


Figure 8.19: Right side of Figure 2.16

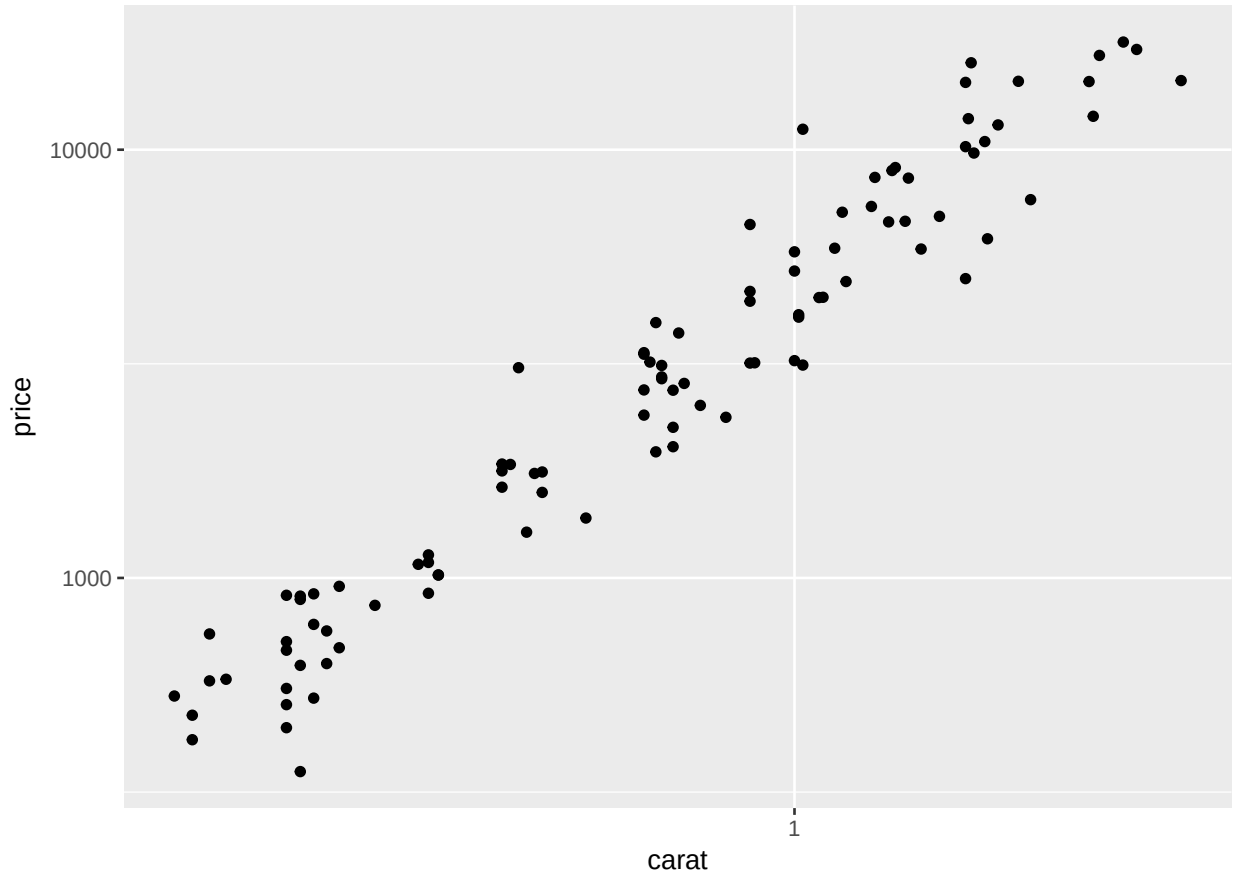


Figure 8.20: First graph on page 26 of

```
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 containing 29 vertical bars.
Panel 2 represents data for color = E.
Panel 2 is a bar chart containing 29 vertical bars.
Panel 3 represents data for color = F.
Panel 3 is a bar chart containing 29 vertical bars.
Panel 4 represents data for color = G.
Panel 4 is a bar chart containing 29 vertical bars.
Panel 5 represents data for color = H.
Panel 5 is a bar chart containing 29 vertical bars.
Panel 6 represents data for color = I.
Panel 6 is a bar chart containing 29 vertical bars.
Panel 7 represents data for color = J.
Panel 7 is a bar chart containing 29 vertical bars.
```

8.5 rescaling of the axes

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

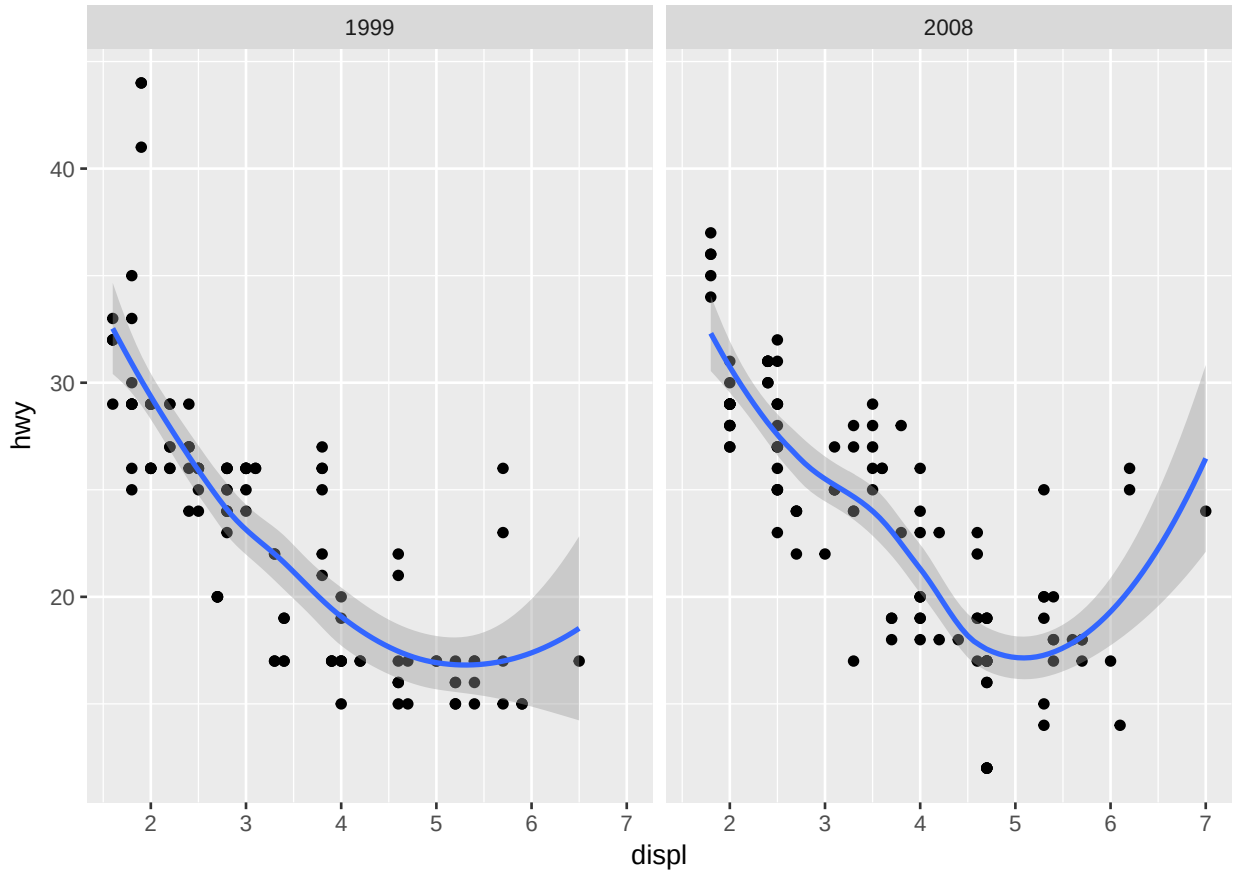


Figure 8.21: Figure 3.6 of

```
p26a %>% VI()
```

```
This is an untitled chart with no subtitle or caption.
It has x-axis 'carat' with labels 1.
It has y-axis 'price' with labels 1000 and 10000.
The chart is a set of 100 points.
```

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

```
`geom_smooth()` using method = 'loess'
```

```
fig3.6 %>% VI()
```

```
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 smoothed curve using method 'auto' with confidence intervals.
```

```
Panel 2 represents data for year = 2008.
Layer 1 of panel 2 is a set of 117 points.
Layer 2 of panel 2 is a smoothed curve using method 'auto' with confidence intervals.
```

Chapter 9

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 the author has 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. Two 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 Python and WxPython independently.

9.1 Getting Python and wxPython (Windows users only)

Issue the following commands at the R prompt

```
library(BrailleR)
```

```
GetPython27()
```

```
GetWxPython27()
```

These commands automatically download the installation files and start the installation process going. The downloaded files will be saved in your MyBrailleR folder. You will need to follow the instructions and answer questions that arise whenever you install new software. These are reputable installation files from the primary sites for Python and wxPython. Windows and any security software you might have should know that, but you can never tell! You will probably need to let Windows know it is OK to install the software in the default location. That 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.

Once you have completed both installations, you are ready to go. You shouldn't need those installation files again, but keep them just in case.

9.2 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")
```

9.3 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 open 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.

9.4 Our first HTML file

Making your first HTML file is as easy as hitting a single key, or using one of the options in the `Build` menu. The variety of options are the commonly used ones in RStudio.

Navigate to the current working directory using your file browser. To find out where that is, type `getwd()` back in the R window. You should see the file `MyFirst.Rmd` and once you have built it, the associated HTML file.

Open the HTML file and 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.

9.5 BrailleR commands used in this chapter

We needed to use `GetPython27()` and `GetWxPython27()` to install the necessary software to allow us to run a Python script like WriteR. We then opened a new file using `WriteR()`.

Chapter 10

References