Pedro J. Aphalo

Learn R: As a Language

Supplementary Chapters and Appendixes

Contents

Foreword to supplementary chapters

This file contains additional on-line only open-access chapters for the book *Learn R: As a Language* (Aphalo 2020) with specialized and advanced content. Chapters numbered 9 and higher are in this file, while chapters 1 to 8 are in the book published in The R Series. Chapter 9 starts on page 351. This document includes cross references by chapter, section and page numbers to chapters and pages both in this PDF as in the book.

In the book as a whole I have de-emphasized the various ways of changing the visual design of ggplots, focusing more on the different ways in which data can be displayed. In part this is because to keep a reasonable number of printed pages I had to restrict the number of illustrations. It also has to do with myself needing most of the time to produce illustrations for scientific publications where a formal and simple style is the norm. The on-line tutorial *A ggplot2 Tutorial for Beautiful Plotting in R* (Scherer 2019) covers these aspects thoroughly.

This file will be updated more frequently than the book, and includes errata to the book as an Appendix.

Preface (Reproduced from book)

> "Suppose that you want to teach the 'cat' concept to a very young child. Do you explain that a cat is a relatively small, primarily carnivorous mammal with retractible claws, a distinctive sonic output, etc.? I'll bet not. You probably show the kid a lot of different cats, saying 'kitty' each time, until it gets the idea. To put it more generally, generalizations are best made by abstraction from experience."
>
> R. P. Boas
> *Can we make mathematics intelligible?*, 1981

This book covers different aspects of the use of the R language. Chapters 1 to 5 describe the R language itself. Later chapters describe extensions to the R language available through contributed *packages*, the *grammar of data* and the *grammar of graphics*. In this book, explanations are concise but contain pointers to additional sources of information, so as to encourage the development of a routine of independent exploration. This is not an arbitrary decision, this is the normal *modus operandi* of most of us who use R regularly for a variety of different problems. Some have called approaches like the one used here "learning the hard way," but I would call it "learning to be independent."

I do not discuss statistics or data analysis methods in this book; I describe R as a language for data manipulation and display. The idea is for you to learn the R language in a way comparable to how children learn a language: they work out what the rules are, simply by listening to people speak and trying to utter what they want to tell their parents. Of course, small children receive some guidance, but they are not taught a prescriptive set of rules like when learning a second language at school. Instead of listening, you will read code, and instead of speaking, you will try to execute R code statements on a computer—i.e., you will try your hand at using R to tell a computer what you want it to compute. I do provide explanations and guidance, but the idea of this book is for you to use the numerous examples to find out by yourself the overall patterns and coding philosophy behind the R language. Instead of parents being the sound board for your first utterances in R, the computer will play this role. You will *play* by modifying the examples, see how the computer responds: does R understand you or not? Using a language actively is the most efficient way of learning it. By using it, I mean actually reading, writing, and running scripts or programs (copying and pasting, or typing ready-made examples from books or the internet, does not qualify as using a language).

I have been using R since around 1998 or 1999, but I am still constantly learning new things about R itself and R packages. With time, it has replaced in my work as a researcher and teacher several other pieces of software: SPSS, Systat, Origin, MS-Excel, and it has become a central piece of the tool set I use for producing lecture slides, notes, books, and even web pages. This is to say that it is the most useful piece of software and programming language I have ever learned to use. Of course, in time it will be replaced by something better, but at the moment it is a key language to learn for anybody with a need to analyze and display data.

What is a language? A language is a system of communication. R as a language allows us to communicate with other members of the R community, and with computers. As with all languages in active use, R evolves. New "words" and new "constructs" are incorporated into the language, and some earlier frequently used ones are relegated to the fringes of the corpus. I describe current usage and "modisms" of the R language in a way accessible to a readership unfamiliar with computer science but with some background in data analysis as used in biology, engineering, or the humanities.

When teaching, I tend to lean toward challenging students, rather than telling an over-simplified story. There are two reasons for this. First, I prefer as a student, and I learn best myself, if the going is not too easy. Second, if I would hide the tricky bits of the R language, it would make the reader's life much more difficult later on. You will not remember all the details; nobody could. However, you most likely will remember or develop a sense of when you need to be careful or should check the details. So, I will expose you not only to the usual cases, but also to several exceptions and counterintuitive features of the language, which I have highlighted with icons. Reading this book will be about exploring a new world; this book aims to be a travel guide, but neither a traveler's account, nor a cookbook of R recipes.

Keep in mind that it is impossible to remember everything about R! The R language, in a broad sense, is vast because its capabilities can be expanded with independently developed packages. Learning to use R consists of learning the basics plus developing the skill of finding your way in R and its documentation. In early 2020, the number of packages available in the Comprehensive R Archive Network (CRAN) broke the 15,000 barrier. CRAN is the most important, but not only, public repository for R packages. How good a command of the R language and packages a user needs depends on the type of activities to be carried out. This book attempts to train you in the use of the R language itself, and of popular R language extensions for data manipulation and graphical display. Given the availability of numerous books on statistical analysis with R, in the present book I will cover only the bare minimum of this subject. The same is true for package development in R. This book is somewhere in-between, aiming at teaching programming in the small: the use of R to automate the drudgery of data manipulation, including the different steps spanning from data input and exploration to the production of publication-quality illustrations.

As with all "rich" languages, there are many different ways of doing things in R. In almost all cases there is no one-size-fits-all solution to a problem. There is always a compromise involved, usually between time spent by the user and processing time required in the computer. Many of the packages that are most popular nowadays did not exist when I started using R, and many of these packages make

new approaches available. One could write many different R books with a given aim using substantially different ways of achieving the same results. In this book, I limit myself to packages that are currently popular and/or that I consider elegantly designed. I have in particular tried to limit myself to packages with similar design philosophies, especially in relation to their interfaces. What is elegant design, and in particular what is a friendly user interface, depends strongly on each user's preferences and previous experience. Consequently, the contents of the book are strongly biased by my own preferences. I have tried to write examples in ways that execute fast without compromising readability. I encourage readers to take this book as a starting point for exploring the very many packages, styles, and approaches which I have not described.

I will appreciate suggestions for further examples, and notification of errors and unclear sections. Because the examples here have been collected from diverse sources over many years, not all sources are acknowledged. If you recognize any example as yours or someone else's, please let me know so that I can add a proper acknowledgement. I warmly thank the students who have asked the questions and posed the problems that have helped me write this text and correct the mistakes and voids of previous versions. I have also received help on online forums and in person from numerous people, learned from archived e-mail list messages, blog posts, books, articles, tutorials, webinars, and by struggling to solve some new problems on my own. In many ways this text owes much more to people who are not authors than to myself. However, as I am the one who has written this version and decided what to include and exclude, as author, I take full responsibility for any errors and inaccuracies.

Why have I chosen the title "*Learn R: As a Language*"? This book is based on exploration and practice that aims at teaching to express various generic operations on data using the R language. It focuses on the language, rather than on specific types of data analysis, and exposes the reader to current usage and does not spare the quirks of the language. When we use our native language in everyday life, we do not think about grammar rules or sentence structure, except for the trickier or unfamiliar situations. My aim is for this book to help you grow to use R in this same way, to become fluent in R. The book is structured around the elements of languages with chapter titles that highlight the parallels between natural languages like English and the R language.

I encourage you to approach R like a child approaches his or her mother tongue when first learning to speak: do not struggle, just play, and fool around with R! If the going gets difficult and frustrating, take a break! If you get a new insight, take a break to enjoy the victory!

Acknowledgements

First I thank Jaakko Heinonen for introducing me to the then new R. Along the way many well known and not so famous experts have answered my questions in usenet and more recently in Stackoverflow. As time went by, answering other people's questions, both in the internet and in person, became the driving force

for me to delve into the depths of the R language. Of course, I still get stuck from time to time and ask for help. I wish to warmly thank all the people I have interacted with in relation to R, including members of my own research group, students participating in the courses I have taught, colleagues I have collaborated with, authors of the books I have read and people I have only met online or at conferences. All of them have made it possible for me to write this book. This has been a time consuming endeavour which has kept me too many hours away from my family, so I specially thank Tarja, Rosa and Tomás for their understanding. I am indebted to Tarja Lehto, Titta Kotilainen, Tautvydas Zalnierius, Fang Wang, Yan Yan, Neha Rai, Markus Laurel, other colleagues, students and anonymous reviewers for many very helpful comments on different versions of the book manuscript, Rob Calver, as editor, for his encouragement and patience during the whole duration of this book writing project, Lara Spieker, Vaishali Singh, and Paul Boyd for their help with different aspects of this project.

Icons used to mark different content

Text boxes are used throughout the book to highlight content that plays specific roles in the learning process or that require special attention from the reader. Each box contains one of five different icons that indicate the type of its contents as described below.

 Signals *playground* boxes which contain open-ended exercises—ideas and pieces of R code to play with at the R console.

 Signals *advanced playground* boxes which will require more time to play with before grasping concepts than regular *playground* boxes.

 Signals important bits of information that must be remembered when using R—i.e., explain some unusual feature of the language.

 Signals in-depth explanations of specific points that may require you to spend time thinking, which in general can be skipped on first reading, but to which you should return at a later peaceful time, preferably with a cup of coffee or tea.

 Signals text boxes providing general information not directly related to the R language.

Part I

More on Grammar of Graphics

9

Grammar of Graphics: Fonts

> Typographic clarity consists of **legibility** and **readability**. Good legibility means that individual words are easily and quickly recognizable, and that the letters are distinct.
>
> Juuso Koponen and Jonatan Hildén
> *Data visualization handbook*, 2019

9.1 Aims of this chapter

Even though English is widely understood, I expect a considerable number of readers of *Learn R: As a Language* to use R to create plots annotated in languages other than English. In many cases these languages are written using non-Latin alphabets. A simpler but related problem is the use of fonts and font-faces different to the default ones.

When using 'ggplot2' we can rely on two different approaches to the use of other alphabets: making system fonts available within R and relying on R expressions or character strings containing *special characters* using an specific or extended font encoding (such as UTF-8 or UTF-16) or using an "external" text formatting language like LaTeX, Markdown or HTML. We describe both approaches.

9.2 Packages used in this chapter

The list of packages used in the current chapter is very long. However, there are only few incompatibilities among them. Even if one is not attempting to use functions from different packages within the same plot, one may want to use them to create different figures within the same document when using the literate approach to programming and scripting (see 3.2.4 on page 90), with incompatibilities requiring additional coding to work around them.

If the packages used in this chapter are not yet installed in your computer, you

can install them with the code shown below, as long as package 'learnrbook' (>= 1.0.4) is already installed.

```
packages_at_cran <-
  setdiff(learnrbook::pkgs_ch_ggplot_extra, learnrbook::pkgs_at_github)
install.packages(packages_at_cran)
for (p in learnrbook::pkgs_at_github) {
  devtools::install_github(p)
}
```

For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(learnrbook)
library(tibble)

## Warning: package 'tibble' was built under R version 4.1.1

library(magrittr)
library(ggplot2)
library(ggpmisc)
library(ggtext)
library(ragg)
library(showtext)
library(tikzDevice)
```

We set a font of larger size than the default

```
theme_set(theme_grey(14))
```

9.3 System fonts

Access to system fonts is not native to the R language, and requires the use of extension packages. Recent versions of R support UTF encoded characters, making it possible to use of non-Latin and large alphabets. See section 7.4.7 for the use of text natively in 'ggplot2' using character strings and *plotmath* expressions.

P

System fonts are those fonts available through the operating system. These fonts are installed so that, at least in principle, they are accessible to any program run locally. There is usually a basic set of fonts distributed with the operating system but additional ones, both free and commercial, can be installed. Installers of software like word processors or graphic editors frequently install additional system fonts. The installed system fonts will also depend on the localization, i.e., on the language in use by the operating system. Operating systems support browsing, installation and uninstallation of system fonts while additional free (e.g., NexusFont, FontBase) and commercial (e.g., FontBase Awesome) programs for these tasks also exist.

In this section we give examples of the use of non-Latin characters from default fonts and of package 'showtext' to make available locally installed fonts known to the operating system and of Google fonts available through the internet. Enabling such fonts not only makes available glyphs used in other than the default system language but also to collections of icons such as those used in this book.

A font with Chinese characters is included in package 'showtext'. This example is borrowed from the package vignette, but modified to use default fonts, of which `"wqy-microhei"` is a Chinese font.

 When using 'knitr' as for this book, rendering using system fonts made available with 'showtext', the chunk option `fig.showtext=TRUE` needs to be set. Otherwise, instead of the expected characters a faint square will be used as place holder for characters in the output.

```
ggplot(NULL, aes(x = 1, y = 1)) + ylim(0.8, 1.2) +
    theme(axis.title = element_blank(), axis.ticks = element_blank(),
          axis.text = element_blank()) +
    annotate("text", 1, 1.1, family = "wqy-microhei", size = 8,
             label = "\u4F60\u597D\uFF0C\u4E16\u754C") +
    annotate("text", 1, 0.9, label = 'Chinese for "Hello, world!"',
             family = "sans", fontface = "italic", size = 6)
```

Next we load some system fonts, from the same family used for the text of this book. Within code chunks when using 'knitr' we can enable `showtext` with chunk option `fig.showtext = TRUE` as done here (but not visible). In a script or at the console we can use `showtext.auto()`, or `showtext.begin()` and `showtext.end()`. As explained in the package vignette, using `showtext` can increase the size of the PDF files created, but on the other hand, it ensures by embedding the fonts that the PDF is self-contained and portable.

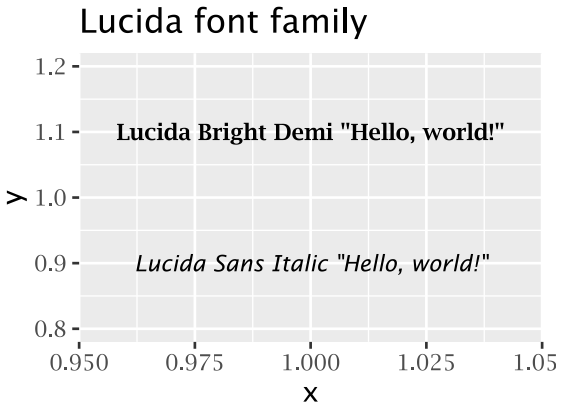
Function `font.families()` lists the fonts known to R, and function `font.add()` can be used to make *system fonts* visible to R. We set `family`, and indicate the font names for each `face`.

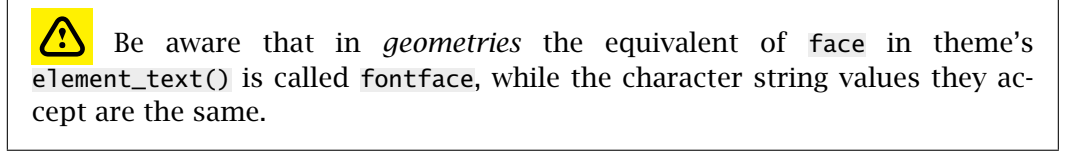

The fonts shown in this example are likely to be unavailable in your own computer, so you may need to replace the font names by those of available fonts. Each combination of `family` and `fontface` corresponds to a font file (in this case Open Type fonts, hence the `.otf` ending of the files names).

```
font_families()
## [1] "sans"         "serif"        "mono"         "wqy-microhei"

font_add(family = "Lucida.Sans",
         regular = "LucidaSansOT.otf",
         italic = "LucidaSansOT-Italic.otf",
         bold = "LucidaSansOT-Demi.otf",
         bolditalic = "LucidaSansOT-DemiItalic.otf")

font_add(family = "Lucida.Bright",
         regular = "LucidaBrightOT.otf",
         italic = "LucidaBrightOT-Italic.otf",
         bold = "LucidaBrightOT-Demi.otf",
         bolditalic = "LucidaBrightOT-DemiItalic.otf")

font_families()
## [1] "sans"          "serif"         "mono"          "wqy-microhei"
## [5] "Lucida.Sans"   "Lucida.Bright"
```

We can then select these fonts similarly as we would for R's `"serif"`, `"sans"`, `"mono"` and `"symbol"`. Please, see section 7.10 on page 275 for examples of how to set the `family` for individual elements of the plot. We use the same `family` names as used above to add the fonts. We show the effect passing family and fontface constants to `geom_text()` and by overriding the default for the theme.

```
ggplot(NULL, aes(x = 1, y = 1)) +
    ylim(0.8, 1.2) +
    annotate("text", 1, 1.1, label = 'Lucida Bright Demi "Hello, world!"',
             family = "Lucida.Bright", fontface = "bold", size = 4) +
    annotate("text", 1, 0.9, label = 'Lucida Sans Italic "Hello, world!"',
             family = "Lucida.Sans", fontface = "italic", size = 4) +
    labs(title = "Lucida font family") +
    theme(text = element_text(family = "Lucida.Bright"),
          title = element_text(family = "Lucida.Sans"))
```

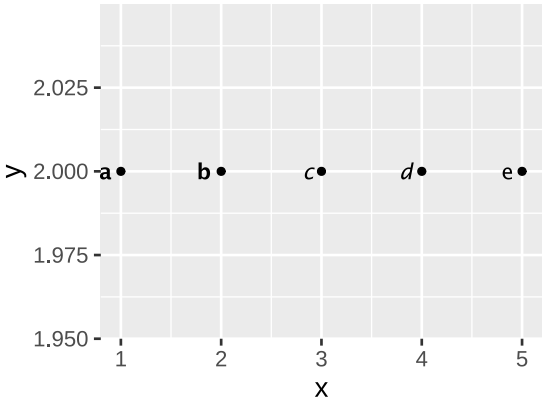




Be aware that in *geometries* the equivalent of `face` in theme's `element_text()` is called `fontface`, while the character string values they accept are the same.

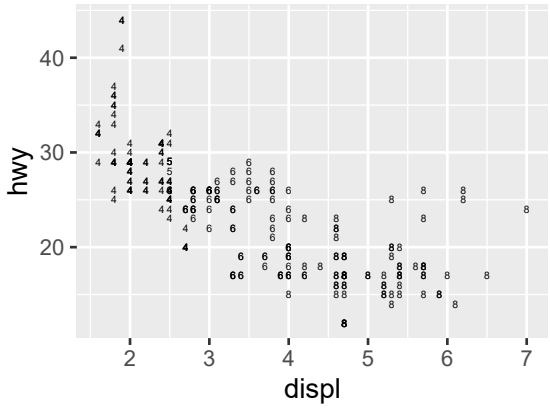
In geometries `family` and `fontface` can be passed constants as arguments or variables mapped to the `family` and `fontface` aesthetics.

```
my.data <-
  data.frame(x = 1:5, y = rep(2, 5),
             label = c("a", "b", "c", "d", "e"),
             face = c("bold", "bold", "italic", "italic", "plain"))

ggplot(my.data, aes(x, y, label = label, fontface = face)) +
  geom_text(hjust = 1.7, family = "Lucida.Sans") +
  geom_point()
```



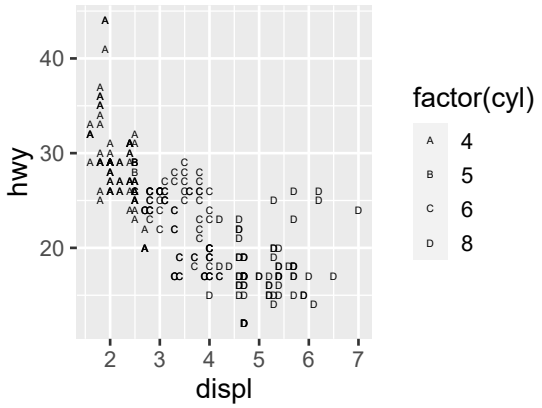
R uses by default special characters from the Symbol font as *points* in plots. However, any character such as letters or numbers are also accepted. A vector containing character strings, each a single character in length, can be mapped to the 'shape' aesthetic can be mapped. These characters do need to be alphanumeric.

```
ggplot(mpg, aes(displ, hwy, shape = as.character(cyl))) +
  geom_point() +
  scale_shape_identity()
```



The intuitive approach of using `scale_shape_manual()` does not allow shapes from system fonts, as 'Grid' always uses the Symbol font for point shapes. Code as shown here let us map characters, but not to switch fonts or access all characters in this font.

```
ggplot(mpg, aes(displ, hwy, shape = factor(cyl))) +
  geom_point() +
  scale_shape_manual(values = LETTERS[1:length(unique(mpg$cyl))])
```

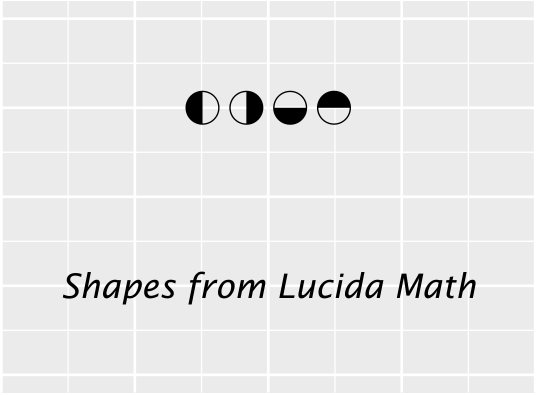


Only the code in one the last two chunks above would have worked if the data set had included cars with 12 cylinders engines. Which one and why?

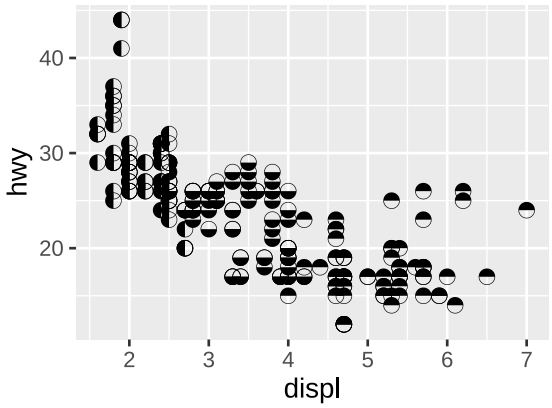
To use shapes from other fonts we need to use single-character text labels. We make available the Math font corresponding to Lucida Bright. This font is very comprehensive and uses a mapping compatible with LaTeX.

```
font_add(family = "Lucida.Bright.Math",
         regular = "LucidaBrightMathOT.otf",
         italic = "LucidaBrightMathOT.otf",
         bold = "LucidaBrightMathOT-Demi.otf",
         bolditalic = "LucidaBrightMathOT-Demi.otf")

font_families()
## [1] "sans"                "serif"               "mono"
## [4] "wqy-microhei"        "Lucida.Sans"         "Lucida.Bright"
## [7] "Lucida.Bright.Math"
```

```
ggplot(NULL, aes(x = 1, y = 1)) +
    ylim(0.8, 1.2) +
    theme(axis.title = element_blank(), axis.ticks = element_blank(),
          axis.text = element_blank()) +
    annotate("text", 1, 1.1, family = "Lucida.Bright.Math", size = 8,
             label = "\u25d0\u25d1\u25d2\u25d3") +
    annotate("text", 1, 0.9, label = 'Shapes from Lucida Math',
             family = "Lucida.Sans", fontface = "italic", size = 6)
```




```
my.mpg <- mpg
# we manually map factor lebels to labels stored in variable shapes
my.mpg$cyl.f <- factor(mpg$cyl)
shapes.map <- c("\u25d0", "\u25d1", "\u25d2", "\u25d3")
names(shapes.map) <- levels(my.mpg$cyl.f)
my.mpg$shapes <- unname(shapes.map[my.mpg$cyl.f])
head(my.mpg$shapes)
## [1] "<U+25D0>" "<U+25D0>" "<U+25D0>" "<U+25D0>" "<U+25D2>" "<U+25D2>"

ggplot(my.mpg, aes(displ, hwy, label = shapes)) +
    geom_text(family = "Lucida.Bright.Math")
```



The examples above will work only if the fonts are locally available in the computer. This compromises portability but such flexibility is important from a graphics design perspective.

New R (virtual/software) graphic devices based on the AGG library support system fonts natively. These graphic devices are implemented in R package 'ragg' and support rendering to different *bitmap* formats: PNG, JPEG, TIFF and PPM. We can use device `agg_png()` in place of R's own `png()` device, or when using 'knitr' add the chunk option `dev = "ragg_png"` as we do here. AGG in addition to supporting system fonts provides better output faster. When using system fonts, one needs to implement a fall back mechanism as there is no guarantee of availability of the requested fonts. This is in contrast to the use of R's generic names like `"sans"` and `"serif"` which are always available.

9.4 Google fonts

Next we use function `font.add.google()`, also from package 'showtext', to make Google fonts available. As long as internet access is available, Google fonts can be downloaded if not available locally. You can browse the available fonts at `https://fonts.google.com/` and access them through the names listed in this page.

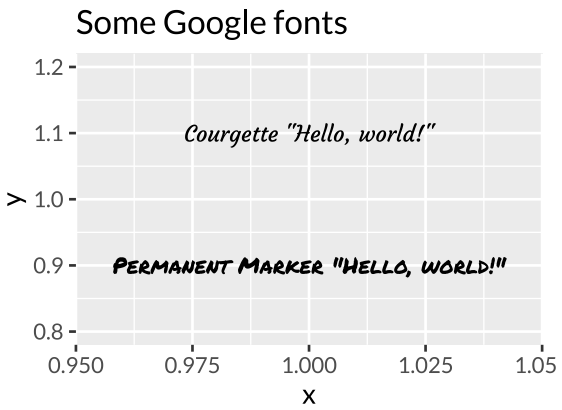
The concept of a font `family` in R is not exactly consistent with the graphic designers' use of the concept of a *font family* as being a set of fonts designed to be used together. In R, with 'showtext' we can create a `family` with an arbitrary choice of fonts for the different `faces`, however, it is in general best to use fonts designed to work well together as we did above when using Lucida fonts. When visiting `https://fonts.google.com/` once you select a font, you can press the rightmost (at this time) icon to display the whole family of related fonts. You will notice that some modern font families are available at many different weights between the lightest (*hairline*) and the boldest (*black*) as well as in multiple character widths between *condensed* and *wide*. From a design

perspective, the recomendation is not to mix many weights in the same typeset text, but rather choose a suitable pair of weights for `"regular"` and `"bold"` font faces.

Here we use two decorative fonts, `"Permanent Marker"` and `"Courgette"`, and `"Lato"`, a more formal font very popular for web pages. All fonts in the Lato family support for more than 100 Latin-based languages, more than 50 Cyrillic-based languages as well as Greek and IPA phonetics.

```
## Loading Google fonts (http://www.google.com/fonts)
font_add_google(name = "Permanent Marker", family = "Marker")
font_add_google(name = "Courgette")
font_add_google(name = "Lato")
```

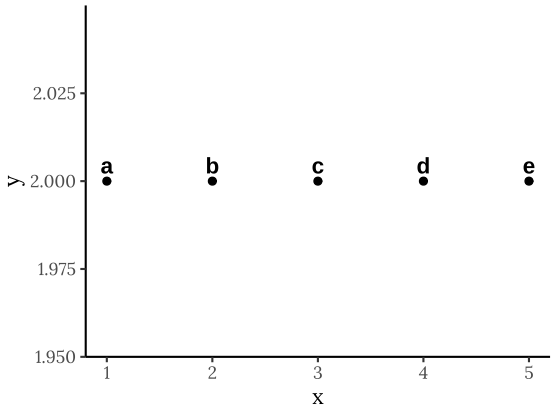
```
ggplot(NULL, aes(x = 1, y = 1)) +
ylim(0.8, 1.2) +
    annotate("text", 1, 1.1, label = 'Courgette "Hello, world!"',
             family = "Courgette", size = 4) +
    annotate("text", 1, 0.9, label = 'Permanent Marker "Hello, world!"',
             family = "Marker", size = 4) +
    labs(title = "Some Google fonts") +
    theme(text = element_text(family = "Lato"))
```



Some modern fonts are parameterized so that the weight can be set numerically. Here we show for font Lora how weights can be assigned to font faces, in this case resulting in a large (actually excessive) difference in weight between `"regular"` and `"bold"` font faces in the plot.

```
font_add_google(name = "Lora", regular.wt = 400, bold.wt = 700)
font_families()
##  [1] "sans"               "serif"              "mono"
##  [4] "wqy-microhei"       "Lucida.Sans"        "Lucida.Bright"
##  [7] "Lucida.Bright.Math" "Marker"             "Courgette"
## [10] "Lato"               "Lora"

ggplot(my.data, aes(x, y, label = label)) +
  geom_text(vjust = -0.5,
            fontface = "bold",
```

```
            size = 4) +
  geom_point() +
  theme_classic(base_size = 11, base_family = "Lora")
```



In all the examples above we used `geom_text()`. However, `geom_label()` can be used instead, as well as geometries from packages like 'ggrepel' and 'ggpmic' that render text using the same approach as 'ggplot2'.

Practice using system and Google fonts, adding new families and their corresponding font faces, and using them in plots created with 'ggplot2'. Consider both the R coding and graphic design related aspects. Be aware that depending on the font or the operating system in use, the non-ASCII characters available can vary.

9.5 Markdown and HTML

Package 'ggtext' implements support for Markdown and a small subset of HTML. It provides new geometries as well as an element that can replace `element_text()` in theme definitions. This means that geometries from 'ggplot2' and other extension packages will remain with their functionality unchanged, although still usable. At this time only some statistics from package 'ggpmisc' can on request generate Markdown-encoded character strings for use as text labels with the new geometries. Using Markdown-encoded titles and axis labels is controlled the theme settings and will not interfere with any geometry or statistic.

Package 'ggtext' is new and changes can be expected as well as support for additional HTML features to be added. Package 'ggtext' provides an alternative to the use of *plotmath* expressions, described in section 7.12).

Markdown was designed as a very simple language to encode formatting

of text to be displayed in web pages. It is normally translated into HTML before deployment and it supports embedding of HTML markup. Embedded HTML is passed through unchanged during translation into HTML. Markdown has become popular as it is both terser and simpler than HTML markup.

Package 'ggtext' implements only a subset of the HTML by means of package 'gridtext' resulting in a text encoding suitable for 'grid', one of R's graphics systems.

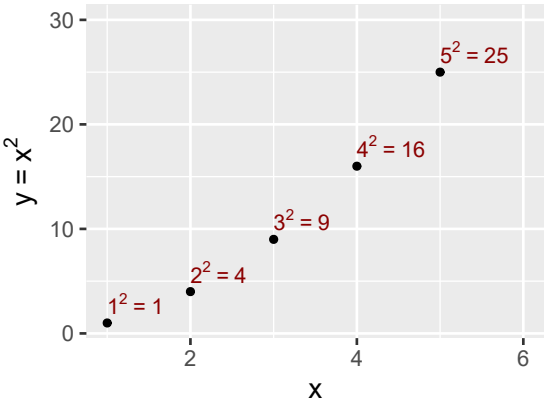
By using 'ggtext' we gain the possibility of using a simpler encoding of text formatting with which R users may be already familiar. We also gain some features that are missing from *plotmath* like highlighting with colour individual words in a title or label, wrapping of long text lines, and the possibility of embedding bitmaps and graphic elements in place of text. On the other hand the ability to nicely format complex mathematical equations available with *plotmath* and LaTeX is missing. So, LaTeX remains the most powerful approach, supporting best and flexible formatting of both text and maths, but requires the use of a specific graphic driver (see section 9.6).

We create some data. To generate character strings to use as text labels we use `sprintf()` taking advantage that it supports vectors as arguments. We define `x` and `y` in the user environment (before calling `data.frame()`) so that `x` and `y` are available when for calling `sprintf()`.

```
x <- 1:5
y <- x^2
my.labels.data <-
  data.frame(x,
             y,
             mkdwn.labs = sprintf("%.0f<sup>2</sup> = %.0f", x, y),
             plotmath.labs = sprintf("%.0f^{2}~`=`~%.0f", x, y),
             latex.labs = sprintf("$%.0f^{2} = %.0f$", x, y))
```

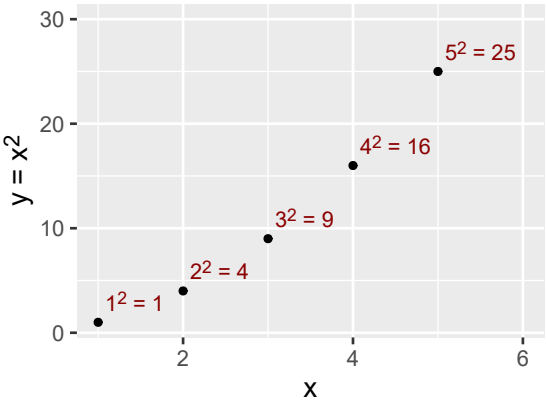
A plot using *plotmath* syntax as described in section 7.12 on page 282.

```
ggplot(my.labels.data,
       aes(x, y, label = plotmath.labs)) +
  geom_text(parse = TRUE, vjust = -0.4, hjust = 0, colour = "darkred") +
  geom_point() +
  expand_limits(x = 6, y = 30) +
  labs(y = expression(y~`=`~x^{2}))
```



The same plot using Markdown syntax. There are some differences: while rendering of R expressions drops trailing zeros, rendering of Markdown does not. In addition, `geom_richtext()` is similar in to `geom_label` in having a box, which we need to make invisible by passing `NA` as arguments. To use Markdown in the axis title we need to override the default in of the theme definition in use.

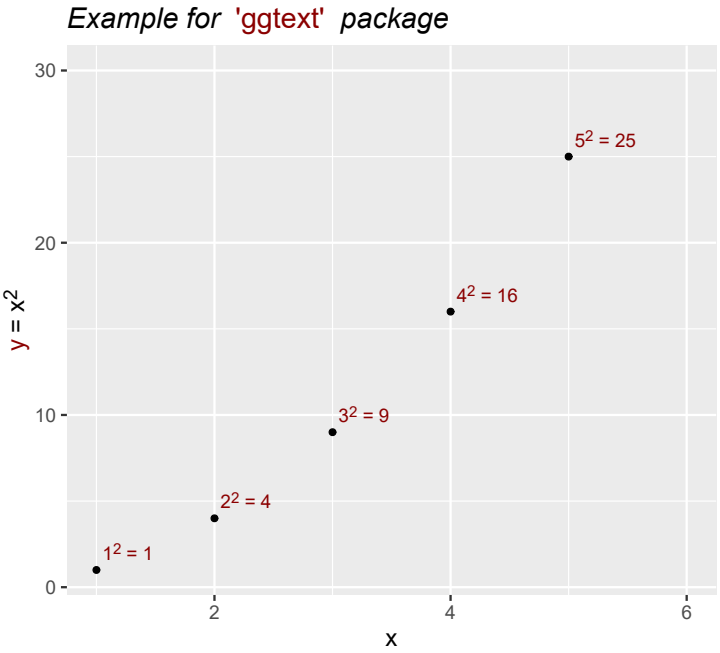
```
ggplot(my.labels.data,
       aes(x, y, label = mkdwn.labs)) +
  geom_richtext(vjust = 0, hjust = 0, colour = "darkred",
                fill = NA, label.size = NA) +
  geom_point() +
  expand_limits(x = 6, y = 30) +
  labs(y = "y = x<sup>2</sup>") +
  theme(axis.title.y = element_markdown())
```



The next example shows some new features introduced by 'ggtext'.

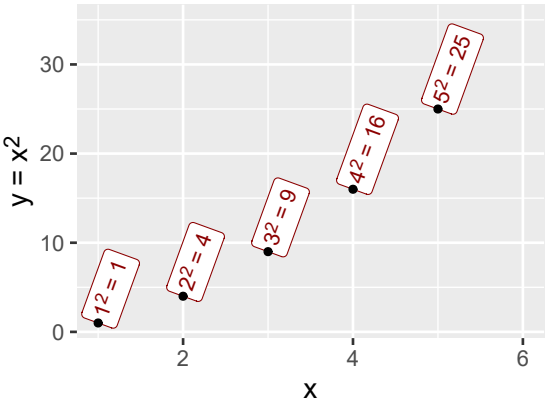
```
ggplot(my.labels.data,
       aes(x, y, label = mkdwn.labs)) +
  geom_richtext(vjust = 0, hjust = 0, colour = "#8B0000",
                fill = NA, label.size = NA) +
  geom_point() +
  expand_limits(x = 6, y = 30) +
  labs(y = "<span style = 'color:#8B0000;'>y</span> = x<sup>2</sup>",
       title = "_Example for_ <span style = 'color:#8B0000;'>'ggtext'</span> _package_",
```

```
         caption = "This is a _caption_. It is encoded using **Markdown** syn-
tax so that we can add some <span style = 'color:#8B0000;'>red colour</span> to it.") +
  theme(axis.title.y = element_markdown(),
        plot.title = element_markdown(),
        plot.caption =
          element_textbox_simple(
            padding = margin(5.5, 5.5, 5.5, 5.5),
            margin = margin(0, 0, 5.5, 0)))
```



While `geom_label()` does not yet support rotation, `geom_richtext()` does.

```
ggplot(my.labels.data,
       aes(x, y, label = mkdwn.labs)) +
  geom_richtext(vjust = 0.5, hjust = 0, colour = "darkred",
                angle = 70) +
  geom_point() +
  expand_limits(x = 6, y = 35) +
  labs(y = "y = x<sup>2</sup>") +
  theme(axis.title.y = element_markdown())
```

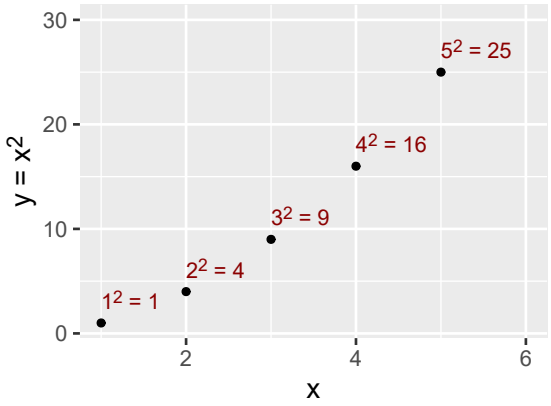


To avoid having to override the defaults repeatedly, we can define a wrapper on `geom_richtext()` with new defaults. In this example we try to match `geom_text()`, so in addition to setting `fill = NA` and `label.size = NA` as above, we set the padding to zero so that justification is based on the boundary of the text like in `geom_text()`. We name the new geometry as `geom_mkdwntext()`.

```
geom_mkdwntext <- function(fill = NA,
                           label.size = NA,
                           label.padding = grid::unit(rep(0, 4), "lines"),
                           ...) {
  geom_richtext(fill = fill,
                label.size = label.size,
                label.padding = label.padding,
                ...)
}
```

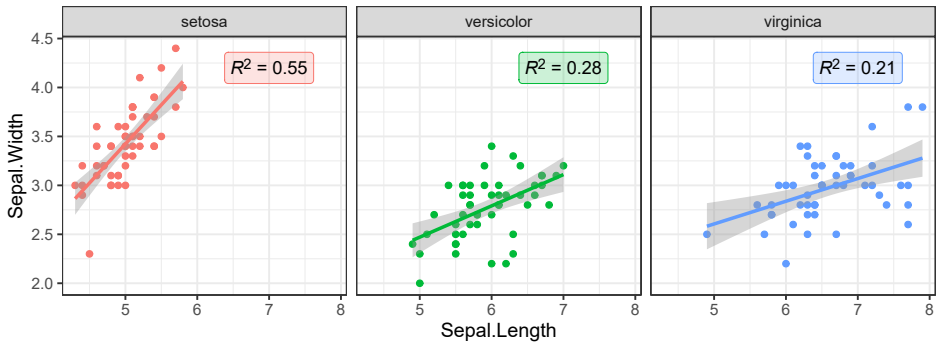
We can now rewrite the first example of the use of Markdown from page 364 as follows. We can use the same justification as with `geom_text()` because we have removed the padding.

```
ggplot(my.labels.data,
       aes(x, y, label = mkdwn.labs)) +
  geom_mkdwntext(vjust = -0.4, hjust = 0, colour = "darkred") +
  geom_point() +
  expand_limits(x = 6, y = 30) +
  labs(y = "y = x<sup>2</sup>") +
  theme(axis.title.y = element_markdown())
```



Package 'ggpmisc' as (described in section 7.5.3, page 244) makes it possible to automate model annotations. We here show an alternative version of an example from the vignette of package 'ggtext' avoiding the need to separately fit the models by using `geom_poly_eq()` from 'ggpmisc'. The geometries `geom_poly_eq()` and `geom_quant_eq()` can automatically generate text labels with model equations and other estimates encoded as *plotmath*, Markdown, LaTeX or plain text. Here we use Markdown and `geom_richtext()`.

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, colour = Species)) +
    geom_point() +
    stat_poly_line() +
    stat_poly_eq(aes(fill = after_scale(alpha(colour, .2))),
                 label.x = 7, label.y = 4.2, text.colour = "black",
                 geom = "richtext") +
    facet_wrap(~Species) +
    theme_bw(12) +
    theme(legend.position = "none")
```



We have only provided some examples of the use of Markdown in ggplots. Current versions of Markdown and HTML markup are described at `https://daringfireball.net/projects/markdown/syntax` and `https://html.spec.whatwg.org/multipage/`, however none of them are fully supported by 'ggtext'. See `https://wilkelab.org/ggtext/` for up to date documentation and further examples.

9.6 LaTeX

LaTeX is the most powerful and stable typesetting engine currently available. I used it to typeset the body text of *Learn R: As a Language* but not for plot labels. 'Tikz' is a LaTeX extension package implementing a language for graphical output. A graphic driver is available for R that renders plots into TikZ commands instead of a bitmap, Postscript or PDF files. The final rendering is done with TeX or one of its derivatives LuaTeX or XeTeX, in most cases after automatically including it into the LaTeX-encoded source file for a manuscript.

The examples in this section use the 'knitr' option `driver="Tikz"` and require R package 'TikzDevice' to be installed. For R users already familiar with LaTeX or needing one of the many alphabets only supported by LaTeX this is the best alternative. Markdown is easy to write but unsuitable for all but the simplest math expressions while R expressions lack the font choices provided by LaTeX.

If a book or article is typeset using LaTeX, this approach not only allows access to fonts suitable for most currently used languages and even some historical ones, but also allows using the use of fonts from the same family in the body of the text and in illustrations enhancing the aesthetics of the typeset document.

```
try(detach(package:tikzDevice))
try(detach(package:ggtext))
try(detach(package:showtext))
try(detach(package:ggpmisc))
try(detach(package:ggpp))
try(detach(package:ggplot2))
try(detach(package:magrittr))
try(detach(package:tibble))
try(detach(package:learnrbook))
```

10

Grammar of Graphics: Color palettes

> Show data variation, not design variation.
>
> Edward Tufte
> *The Visual Display of Quantitative Information*, 1983

10.1 Aims of this chapter

In this chapter I describe some advanced features of package 'ggplot2' and a selection packages that extend it following the same grammar of graphics. Some extensions provide additional *graphical designs*, others *extend the grammar* to additional types of plots and others *wrap frequently used features* into easy to use functions. The update of 'ggplot2' to version 2.0.0 made it straightforward to write these extensions. To keep up-to-date with the release of new extensions I recommend to regularly check the site 'ggplot2 Extensions' (maintained by Daniel Emaasit and colaborators) at `https://exts.ggplot2.tidyverse.org/`.

10.2 Packages used in this chapter

The list of packages used in the current chapter is very long. However, there are only few incompatibilities among them. Even if one is not attempting to use functions from different packages within the same plot, one may want to use them to create different figures within the same document when using the literate approach to programming and scripting (see 3.2.4 on page 90), with incompatibilities requiring additional coding to work around them.

If the packages used in this chapter are not yet installed in your computer, you can install them with the code shown below, as long as package 'learnrbook' (>= 1.0.3) is already installed.

```
packages_at_cran <-
  setdiff(learnrbook::pkgs_ch_ggplot_extra, learnrbook::pkgs_at_github)
```

```
install.packages(packages_at_cran)
for (p in learnrbook::pkgs_at_github) {
  devtools::install_github(p)
}
```

For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(learnrbook)
library(tibble)

## Warning: package 'tibble' was built under R version 4.1.1

library(magrittr)
library(lubridate)
library(ggplot2)
library(viridis)
library(pals)
library(ggsci)
```

We set a font of larger size than the default

```
theme_set(theme_grey(14))
```

10.3 Colours and colour palettes

In addition to the scales in 'ggplot2' (see section **??** on page **??** for the basic concepts), there are packages that implement additional colour and fill scales. Package 'viridis' provides an older implementation of the *Viridis* palettes not available in 'ggplot2' 2, with additional flexibility and a different interface through `scale_colour_viridis()` and `scale_fill_viridis()`.

There are two key aspects to choosing a colour palette for a plot: aesthetics and readability. Both aspects concern the design of information graphics. The *Viridis* palettes are special in that they have been designed to be meaningfully readable under various types of colour blindness as well as when printed or displayed as a grey scale. They are also visually uniform, which is crucial for objectivity: no colours overwhelm others becoming accidentally emphasized. Recent versions of 'ggplot2' include an implementation of discrete and continuous scales based on *Viridis* palettes in `scale_colour_viridis_d()`, `scale_colour_viridis_c()`, `scale_fill_viridis_d()`, and `scale_fill_viridis_c()`. Other colour and fill scales in 'ggplot2', frequently used by default, are not as good in these respects as the *Viridis* ones. The higher the number of distinct colours used, the more difficult it becomes to ensure that they can be distinguished.

Packages exist that include colour palettes, e.g., 'ggsci' that reproduces palettes used by some scientific journals, data analysis software and even TV series, 'hrbrthemes' that defines a few colour and fill scales used in the new themes included in it and 'pals'. Modifying and subsetting palettes needs to be done with

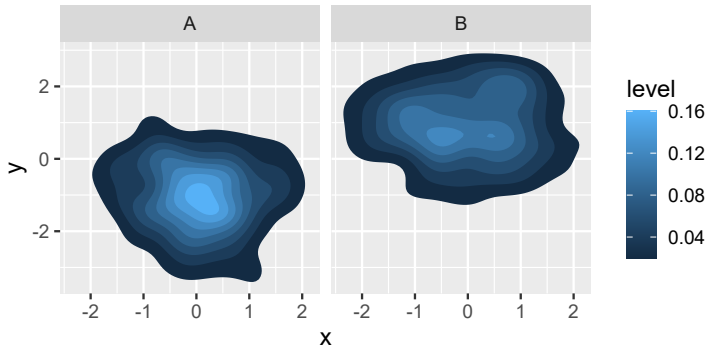
care, and package 'pals' provides in addition to palette definitions, the tools to assess the performance of palettes.

Some of the continuous scales from 'ggplot2' such as `scale_fill_gradient()` and `scale_fill_gradientn()` allow to easily use of colours chosen by users. We use these scales in the examples below. When a factor is mapped to `color` or `fill` aesthetics, we need to use discrete scales such `scale_color_brewer()` or `scale_fill_brewer()` instead of continuous ones.

```
set.seed(56231)
my.data <-
  tibble(x = rnorm(500),
         y = c(rnorm(250, -1, 1), rnorm(250, 1, 1)),
         group = factor(rep(c("A", "B"), c(250, 250))) )
```

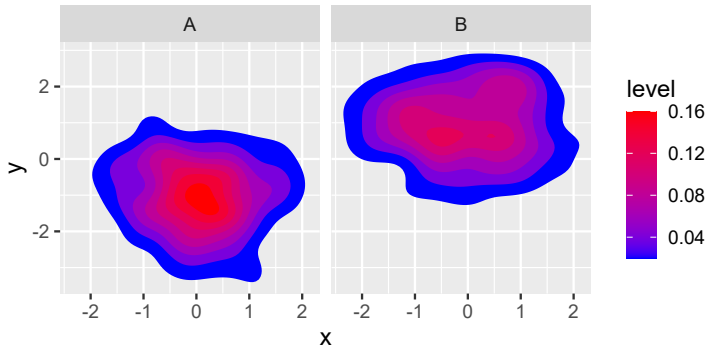
First we use 'ggplot2''s default, based on luminance and saturation of the same color.

```
ggplot(my.data, aes(x, y)) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  facet_wrap(~group) +
  scale_fill_gradient()
```



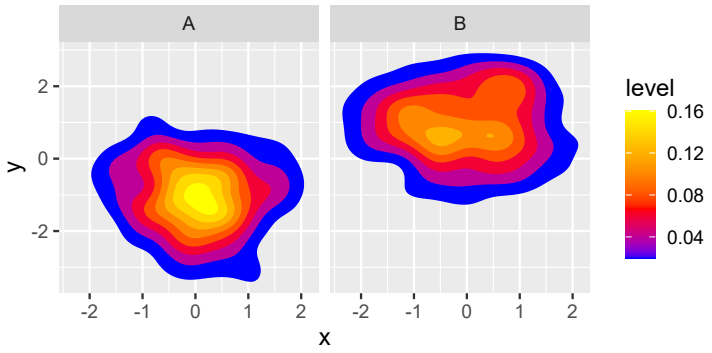
A gradient between two arbitrary colours is not necessarily good, as shown here.

```
ggplot(my.data, aes(x, y)) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  facet_wrap(~group) +
  scale_fill_gradient(low = "blue", high = "red")
```



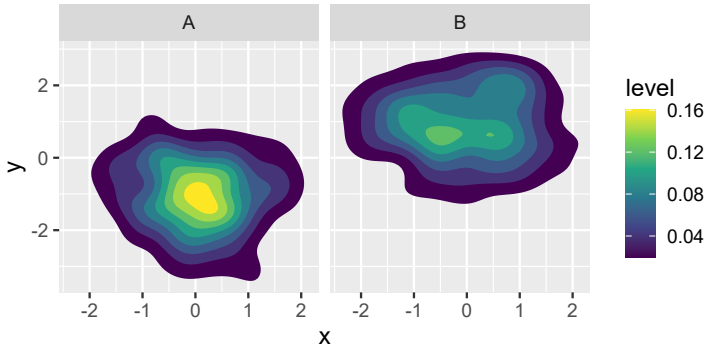
We can also define a gradient based on more than two colour definitions.

```
ggplot(my.data, aes(x, y)) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  facet_wrap(~group) +
  scale_fill_gradientn(colors = c("blue", "red", "orange", "yellow"))
```



Or use a scale based on an specific palette out of a set, here the default *Viridis* palette.

```
ggplot(my.data, aes(x, y)) +
    stat_density_2d(aes(fill = ..level..), geom = "polygon") +
    facet_wrap(~group) +
    scale_fill_viridis_c()
```



As shown in the next section, instead of passing a manually created vector to `scale_fill_gradientn()`, we can generate it using a palette builder.

Modify the example based on the `iris` data from page 367 playing with discrete color and fill scales. Add `scale_fill_discrete()` and `scale_color_discrete()` and adjust the colours and fill in a way that enhances the appearance and/or the readability of the plot.

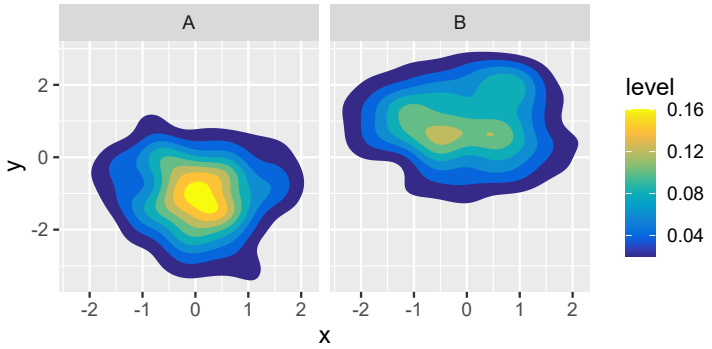
10.3.1 'pals'

```
citation(package = "pals")
```

Package 'pals' provides definitions for palettes and color maps, and also palette evaluation tools. Being a specialized package, we describe it briefly and recommend readers to read the vignette and other documentation included with the package.

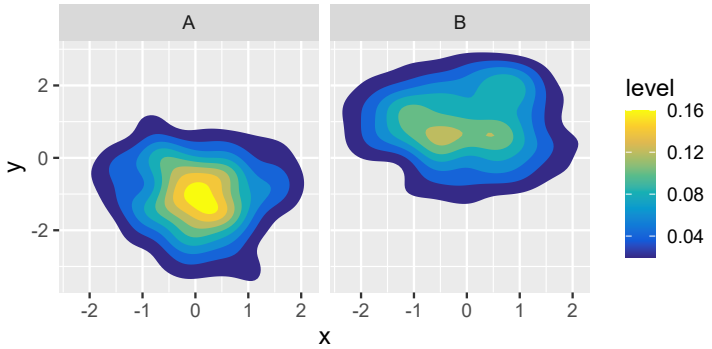
First we reproduce the last example above using palette .

```
ggplot(my.data, aes(x, y)) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  facet_wrap(~group) +
  scale_fill_gradientn(colors = parula(100), guide = "colourbar")
```



The biggest advantage is that we can in the same way use any color map and palette and, in addition, choose how smooth a color map we use. We here use a much shorter vector as argument passed to `colors`. The plot remains almost identical because interpolation is applied.

```
ggplot(my.data, aes(x, y)) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  facet_wrap(~group) +
  scale_fill_gradientn(colours = parula(10), guide = "colourbar")
```

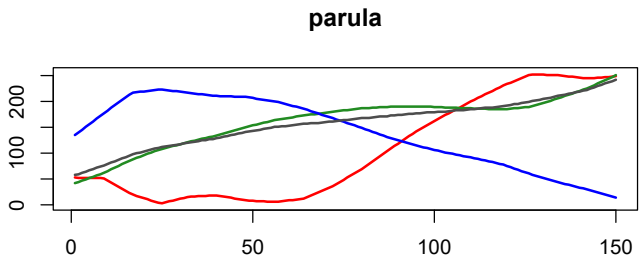


We can compare different color maps with `pal.bands()`, here those used in the last two examples above.

```
pal.bands(parula(100), parula(10))
```

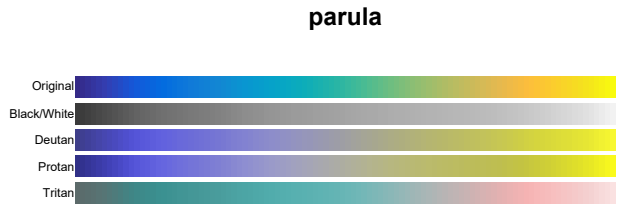
How does the luminance of the red, green and blue colour channels vary along the palette or color map gradient? We can see this with `pal.channels()`.

```
pal.channels(parula, main = "parula")
```



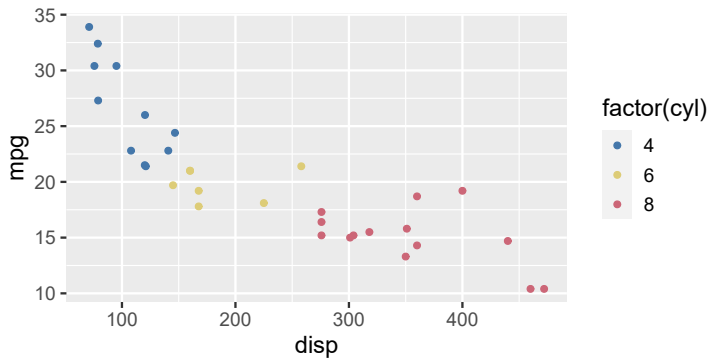
How would `viridis` look in monochrome, and to persons with different kinds of color blindness? We can see this with `pal.safe()`.

```
pal.safe(parula, main = "parula")
```



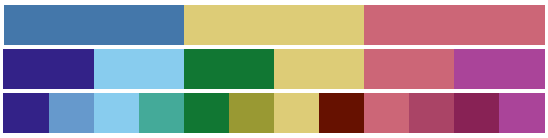
A brief example with using the discrete palette `tol()` from package 'pals'.

```
ggplot(data = mtcars,
    aes(x = disp, y = mpg, color = factor(cyl))) +
    geom_point() +
    scale_color_manual(values = tol(n = 3))
```



Parameter `n` gives the number of discrete values in the palette. Discrete palettes have a maximum value for `n`, in the case of tol, 12 discrete steps.

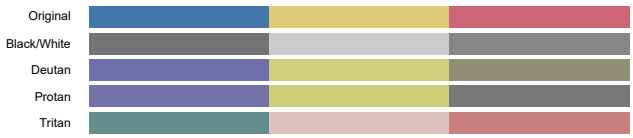
```
pal.bands(tol(n = 3), tol(n = 6), tol())
```




Play with the argument passed to `n` to test what happens when the number of values in the scale is smaller or larger than the number of levels of the factor mapped to the color *aesthetic*.

Is this palette safe?

```
pal.safe(tol(n = 3))
```



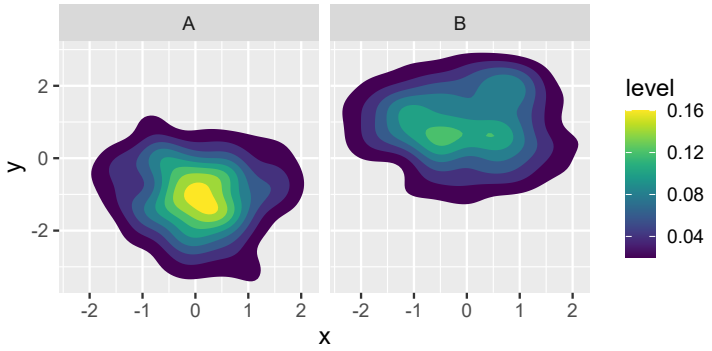
Explore the available palettes until you find a nice one that is also safe with three steps. Be aware that color maps, like `viridis()` can be used to define a discrete color scale using `scale_color_manual()` in exactly the same way as palettes like `tol()`. Colormaps, however, may be perceived as gradients, rather than un-ordered discrete categories, so care is needed.

10.3.2 'viridis'

```
citation(package = "viridis")
```

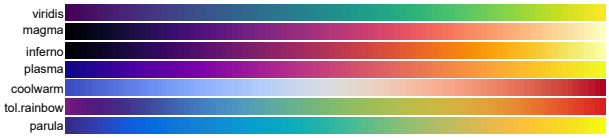
First we simply reproduce the example from page **??**, obtaining the same plot as with `scale_fill_viridis_c()` but using a palette builder function to create a vector of color definitions of length 100.

```
ggplot(my.data, aes(x, y)) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  facet_wrap(~group) +
  scale_fill_gradientn(colors = viridis(100), guide = "colourbar")
```



Next we compare palettes defined in package 'viridis' with some of the palettes defined in package 'pals'.

```
pal.bands(viridis, magma, inferno, plasma, coolwarm, tol.rainbow, parula)
```



10.3.3 'ggsci'

```
citation(package = "ggsci")
##
## To cite package 'ggsci' in publications use:
##
##   Nan Xiao (2018). ggsci: Scientific Journal and Sci-Fi Themed Color
##   Palettes for 'ggplot2'. R package version 2.9.
##   https://CRAN.R-project.org/package=ggsci
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {ggsci: Scientific Journal and Sci-Fi Themed Color Palettes for
## 'ggplot2'},
##     author = {Nan Xiao},
##     year = {2018},
##     note = {R package version 2.9},
##     url = {https://CRAN.R-project.org/package=ggsci},
##   }
```

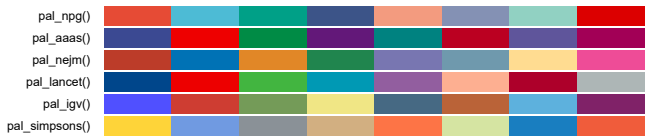
I list here package 'ggsci' as it provides several *color palettes* (and color maps) that some users may like or find useful. They attempt to reproduce the those used by several publications, films, etc. Although visually attractive, several of them are not safe, in the sense discussed in section 10.3.1 on page 374. For each palette, the package exports a corresponding *scale* for use with package 'ggplot2'.

Here is one example, using package 'pals', to test if it is "safe".

```
pal.safe(pal_uchicago(), n = 9)
```

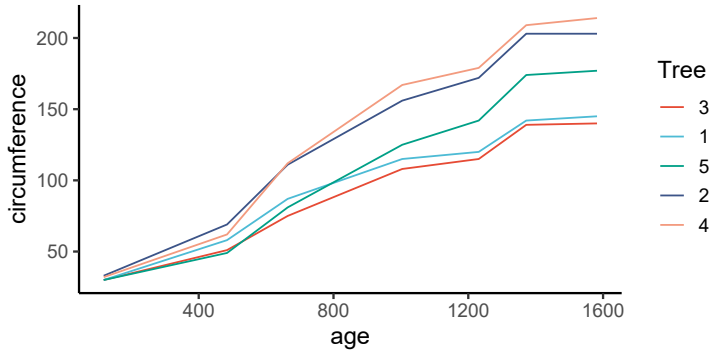
A few of the discrete palettes as bands, setting `n` to 8, which is the largest value supported by the smallest of these palettes.

```
pal.bands(pal_npg(),
          pal_aaas(),
          pal_nejm(),
          pal_lancet(),
          pal_igv(),
          pal_simpsons(),
          n = 8)
```



And a plot using a palette mimicking the one used by Nature Publishing Group (NPG).

```
ggplot(data = Orange,
       aes(x = age, y = circumference, color = Tree)) +
  geom_line() +
  scale_color_npg() +
  theme_classic(14)
```



```
try(detach(package:ggsci))
try(detach(package:pals))
try(detach(package:viridis))
try(detach(package:ggplot2))
try(detach(package:magrittr))
try(detach(package:tibble))
try(detach(package:learnrbook))
```

Part II

Example Cases

A

Case: Timeline plots

> What this means is that we shouldn't abbreviate the truth but rather get a new method of presentation.
>
> Edward Tufte
> *Q+A - Edward Tufte*, 2007

A.1 Aims of this chapter

The chapters in this appendix exemplify specific use cases. The present chapter shows how to produce nice and accurate timeline plots as ggplots. It is based on an article written by the author currently *in press* in the *UV4Plants Bulletin* (`https://bulletin.uv4plants.org`) issue 2021:1.

A.2 Packages used in this chapter

If the packages used in this chapter are not yet installed in your computer, you can install them with the code shown below, as long as package 'learnrbook' (>= 1.0.4) is already installed.

```
packages_at_cran <-
  setdiff(learnrbook::pkgs_ch_ggplot_extra, learnrbook::pkgs_at_github)
install.packages(packages_at_cran)
for (p in learnrbook::pkgs_at_github) {
  devtools::install_github(p)
}
```

For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(dplyr)
library(ggplot2)
```

```
library(ggpp)
library(ggrepel)
library(lubridate)
library(photobiology)
```

We set a font of larger size than the default.

```
theme_set(theme_grey(14))
```

A.3 What is a timeline plot?

The earliest examples of timeline plots date from the 1 700's (Koponen and Hildén 2019). Initially used to depict historical events, timeline plots are useful to describe any sequence of events and periods along a time axis. To be useful the events should be positioned proportionally, i.e., the distance along the timeline at which events are marked, should be proportional to the time interval separating these events (Koponen and Hildén 2019).

Timelines provide a very clear, unambiguous and concise way of describing a sequence of historical events, the time course of scientific experiments, the timing of steps in laboratory protocols and even the progress of people's careers in science, sports, politics, etc. I suspect they are used less frequently than they could because drafting them can seem like a lot of work. But is it necessarily so?

Let's think what a time-line really is: it is in essence a one-dimensional data plot, where a single axis represents time and events or periods are marked and usually labelled. So the answer on how to easily and accurately draw a timeline plot is to use data plotting software instead of "free-hand" drafting software. I will show examples using R package 'ggplot2' and some extensions to it. The beauty of this approach is that there is little manual fiddling, the time-line plot is created from data, and the code can be reused with few changes for other timelines by plotting different data. As this is the only question for this issue, I provide an extended answer with multiple examples.

A.4 A simple timeline

The first example is a timeline plot showing when each past issue of the UV4Plants Bulletin was published. We use package 'ggplot2' for the plotting, and package 'lubridate' to operate on dates.

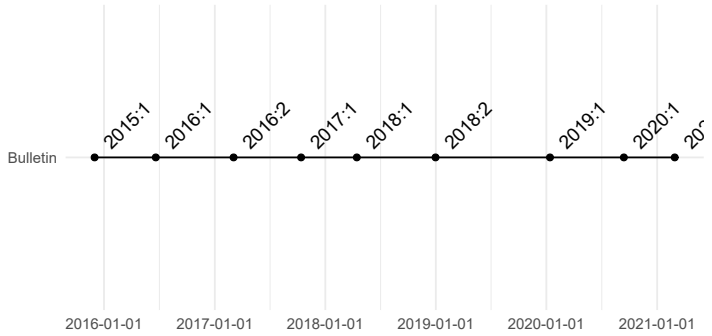
We write the data into a data frame.

```
issues.tb <-
  data.frame(what = c("2015:1", "2016:1", "2016:2", "2017:1", "2018:1",
                      "2018:2", "2019:1", "2020:1", "2020:2"),
             when = ymd(c("2015-12-01", "2016-06-20", "2017-03-04",
```

```
                          "2017-10-13", "2018-04-15", "2018-12-31",
                          "2020-01-13", "2020-09-13", "2021-02-28")),
            event.type = "Bulletin")
```

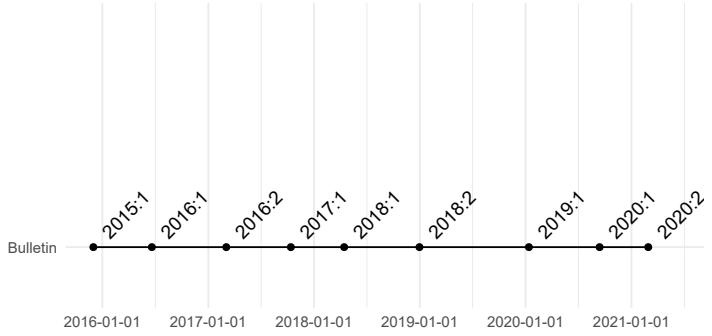
A timeline with sparse events is easy to plot once we have the data. The plots are displayed in this chaapter as narrow horizontal strips. The width and height of the plots is decided when they are printed or exported (code not visible here).

```
ggplot(issues.tb, aes(x = when, y = event.type, label = what)) +
  geom_line() +
  geom_point() +
  geom_text(hjust = -0.3, angle = 45) +
  scale_x_date(name = "", date_breaks = "1 years") +
  scale_y_discrete(name = "") +
  theme_minimal()
```



In the plot above, using defaults, there is too much white space below the timeline and the rightmost text label is not shown in full. We expand the x axis on the right and remove some white space from the y axis by adjusting the scales' expansion (the default is `mult = 0.05`).

```
ggplot(issues.tb, aes(x = when, y = event.type, label = what)) +
  geom_line() +
  geom_point() +
  geom_text(hjust = -0.3, angle = 45) +
  scale_x_date(name = "", date_breaks = "1 years",
               expand = expansion(mult = c(0.05, 0.1))) +
  scale_y_discrete(name = "",
                   expand = expansion(mult = c(0.01, 0.04))) +
  theme_minimal()
```



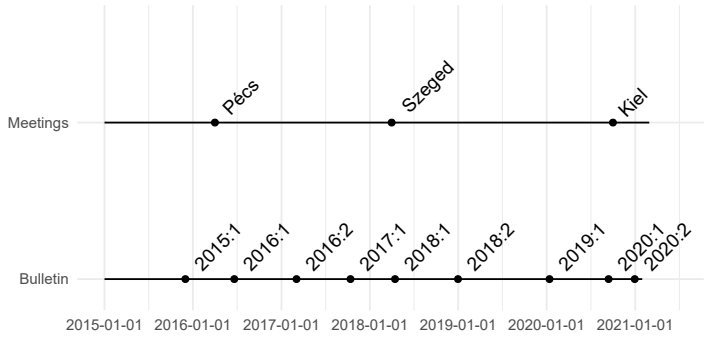
A.5 Two parallel timelines

Let's add a second parallel timeline with the UV4Plants Network Meetings and extend both ends of the lines. We use package 'dplyr' to filter part of the data on the fly so that no points are plotted at the ends of the lines.

```
uv4plants.tb <-
  data.frame(what = c("", "2015:1", "2016:1", "2016:2", "2017:1", "2018:1",
                      "2018:2", "2019:1", "2020:1", "2020:2", "",
                      "", "Pécs  ", "Szeged", "Kiel  ", ""),
             when = ymd(c("2015-01-01", "2015-12-01", "2016-06-20",
                          "2017-03-04", "2017-10-13", "2018-04-15",
                          "2018-12-31", "2020-01-13", "2020-09-13",
                          "2020-12-30", "2021-01-30",
                          "2015-01-01", "2016-04-01", "2018-04-01",
                          "2020-10-01", "2021-02-28")),
             event.type = c(rep("Bulletin", 11), rep("Meetings", 5)))
```

Compared to a single timeline the main change is in the data. The code remains very similar. We do need to adjust the expansion of the y-axis (by trial and error).

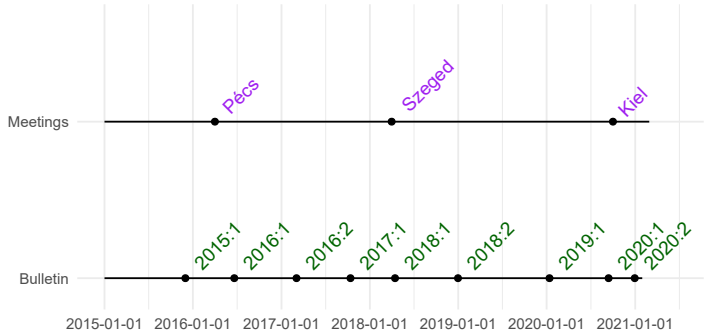
```
ggplot(uv4plants.tb, aes(x = when, y = event.type, label = what)) +
  geom_line() +
  geom_point(data = . %>% filter(what != "")) +
  geom_text(hjust = -0.3, angle = 45) +
  scale_x_date(name = "", date_breaks = "1 years",
               expand = expansion(mult = c(0.05, 0.1))) +
  scale_y_discrete(name = "",
                   expand = expansion(mult = c(0.2, 0.75))) +
  theme_minimal()
```



We add colours by adding `aes(colour = event.type)` to the call to `geom_text()`. To override the default colours we use `scale_colour_manual()` and as we do not need a key to indicate the meaning of the colours, we add to the call `guide = "none"`. In this example the colour is only applied to the text labels, but we can similarly add colour to the lines and points.

```
ggplot(uv4plants.tb, aes(x = when, y = event.type, label = what)) +
  geom_line() +
  geom_point(data = . %>% filter(what != "")) +
  geom_text(aes(colour = event.type), hjust = -0.3, angle = 45) +
```

```
  scale_x_date(name = "", date_breaks = "1 years",
               expand = expansion(mult = c(0.05, 0.1))) +
  scale_y_discrete(name = "",
                   expand = expansion(mult = c(0.2, 0.75))) +
  scale_colour_manual(values = c(Bulletin = "darkgreen", Meetings = "purple"),
                      guide = "none") +
  theme_minimal()
```



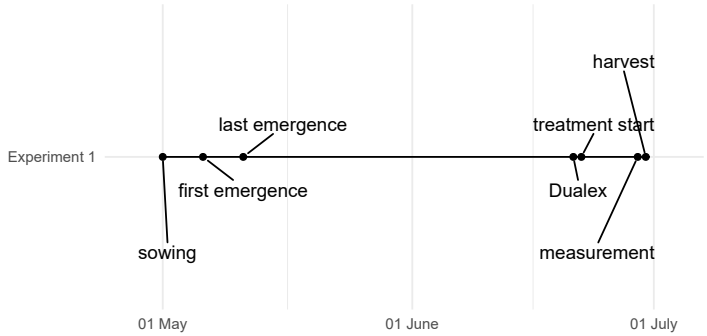
A.6 Crowded timeline

Let's assume an experiment with plants, and create some data. As the labels are rather long and we want to keep the text horizontal, we will use package 'ggrepel' which provides geoms that implement repulsion of labels to automatically avoid overlaps.

```
plants.tb <-
  data.frame(what = c("sowing", "first emergence", "last emergence", "Dualex",
                      "treatment start", "measurement", "harvest"),
             when = ymd(c("2020-05-01", "2020-05-06", "2020-05-11", "2020-06-21",
                          "2020-06-22", "2020-06-29", "2020-06-30")),
             series = "Experiment 1")
```

Now the labels would overlap, so we let R find a place for them using `geom_text_repel()` instead of `geom_text()`.

```
ggplot(plants.tb, aes(x = when, y = series, label = what)) +
    geom_line() +
    geom_point() +
    geom_text_repel(direction = "y",
                    point.padding = 0.5,
                    hjust = 0,
                    box.padding = 1,
                    seed = 123) +
    scale_x_date(name = "", date_breaks = "1 months", date_labels = "%d %B",
                 expand = expansion(mult = c(0.12, 0.12))) +
    scale_y_discrete(name = "") +
    theme_minimal()
```

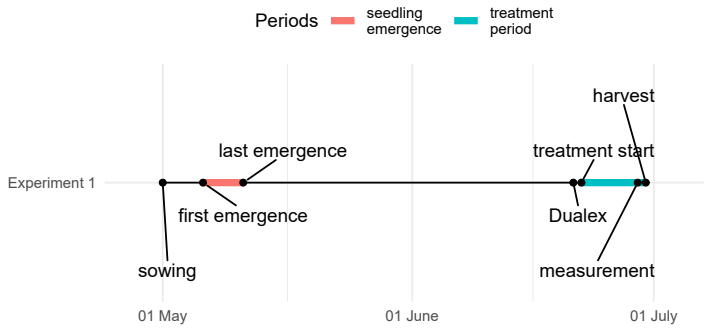


As germination and treatments are periods, we can highlight them more elegantly. For this we will create a second data frame with data for the periods.

```
plants_periods.tb <-
  data.frame(Periods = c("seedling\nemergence",
                         "treatment\nperiod"),
             start = ymd(c("2020-05-06", "2020-06-22")),
             end = ymd(c("2020-05-11", "2020-06-30")),
             series = "Experiment 1")
```

We highlight two periods using colours, and move the corresponding key to the top.

```
ggplot(plants.tb, aes(x = when, y = series)) +
  geom_line() +
  geom_segment(data = plants_periods.tb,
               mapping = aes(x = start, xend = end,
                             y = series, yend = series,
                             colour = Periods),
               size = 2) +
  geom_point() +
  geom_text_repel(aes(label = what),
                  direction = "y",
                  point.padding = 0.5,
                  hjust = 0,
                  box.padding = 1,
                  seed = 123) +
  scale_x_date(name = "", date_breaks = "1 months", date_labels = "%d %B",
               expand = expansion(mult = c(0.12, 0.12))) +
  scale_y_discrete(name = "") +
  theme_minimal() +
  theme(legend.position = "top")
```



A.7 Timelines of graphic elements

Finally an example where the "labels" are inset plots. The aim is to have a timeline of the course of the year with plots showing the course of solar elevation through specific days of the year. This is a more complex example where we customize the plot theme.

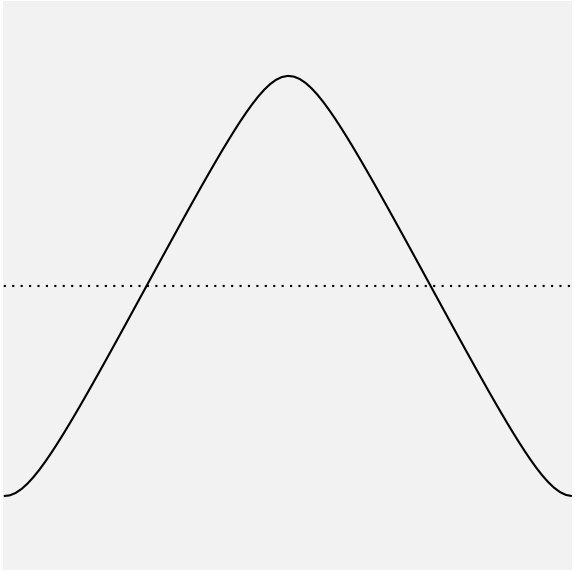
We will use package 'ggpp' that provides a geom for easily insetting plots into a larger plot and package 'photobiology' to compute the solar elevation.

As we will need to make several, very similar plots, we first define a function that returns a plot and takes as arguments a date and a geocode. The intention is to create plots that are almost like icons, extremely simple but still comparable and conveying useful information. To achieve these aims we need to make sure that irrespective of the actual range of solar elevations the limits of the y-axis are -90 and +90 degrees. We use a pale gray background instead of axes to show this range, but making sure that the default expansion of the axis limits is not applied.

```
make_sun_elevation_plot <- function(date, geocode) {
  # 97 points in time from midnight to midnight
  t <- rep(date, 24 * 4 + 1) +
         hours(c(rep(0:23, each = 4), 24)) +
         minutes(c(rep(c(0, 15, 30, 45), times = 24), 0))
  e <- sun_elevation(time = t, geocode = geocode)
  ggplot(data.frame(t, e), aes(t, e)) +
    geom_hline(yintercept = 0, linetype = "dotted") +
    geom_line() +
    expand_limits(y = c(-90, 90)) +
    scale_x_datetime(date_labels = "%H:%m", expand = expansion()) +
    scale_y_continuous(expand = expansion()) +
    theme_void() +
    theme(panel.background = element_rect(fill = "grey95",
                                          color = NA),
          panel.border = element_blank())
}
```

```
make_sun_elevation_plot(date = ymd("2020-06-21"),
                        geocode = tibble(lon = 0,
```

```
                                    lat = 0,
                                    address = "Equator"))
```

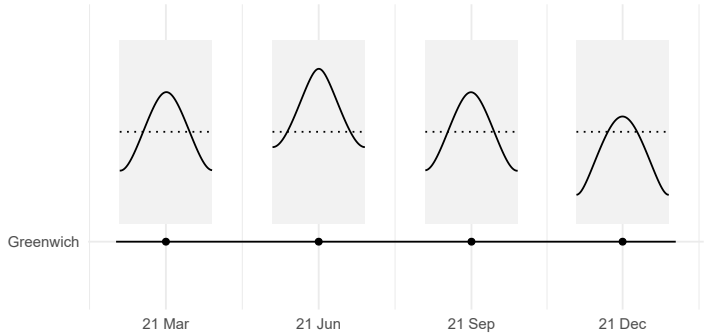


Now that we have a working function, assembling the data for the timeline is not much more complex than in earlier examples. We add two dates, one at each

```
geocode <- tibble(lon = 0, lat = 51.5, address = "Greenwich")
dates <- ymd(c("2020-02-20", "2020-03-21", "2020-6-21",
               "2020-09-21", "2020-12-21", "2021-01-22"))
date.ticks <- dates[2:5]
date.ends <- dates[c(1, 6)]
sun_elevation.tb <-
  tibble(when = dates,
         where = geocode$address,
         plots = lapply(dates, make_sun_elevation_plot, geocode = geocode))
```

The code used to plot the timeline follows the same pattern as in the examples above, except that we replace `geom_text()` with `geom_plot()`. This also entails overriding the default size (`vp.height` and `vp.width`) and justification (`vjust` and `hjust`) of the insets .

```
ggplot(sun_elevation.tb, aes(x = when, y = where, label = plots)) +
    geom_line() +
    geom_point(data = . %>% filter(day(when) == 21)) +
    geom_plot(data = . %>% filter(day(when) == 21),
              inherit.aes = TRUE,
              vp.width = 0.15, vp.height = 0.6, vjust = -0.1, hjust = 0.5) +
    scale_x_date(name = "", breaks = date.ticks, date_labels = "%d %b") +
    scale_y_discrete(name = "",
                     expand = expansion(mult = c(0.1, 0.35))) +
    theme_minimal()
```

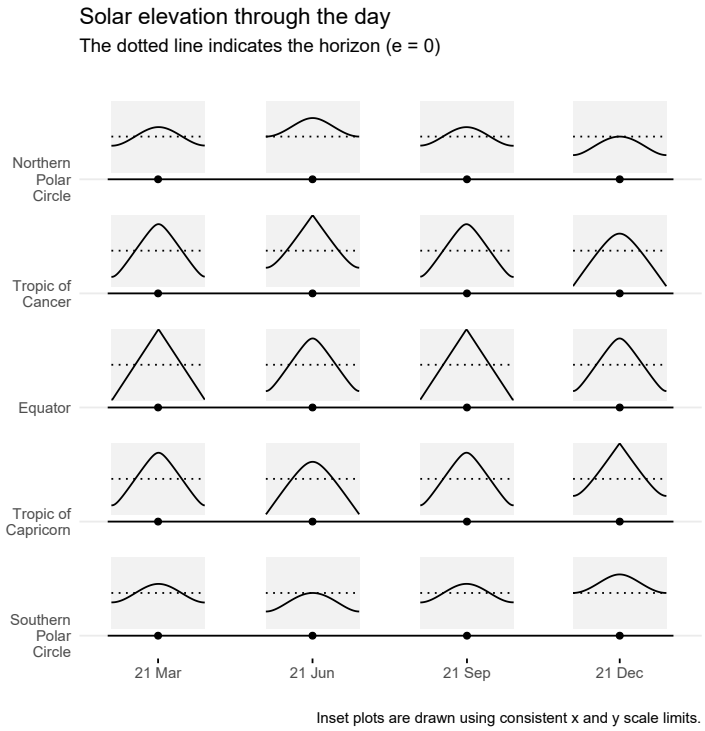


And to finalize, we plot three parallel timelines, each for a different latitude. For this we can reuse the function defined above, passing as argument geocodes for three different locations. We reuse the dates defined above, but use `rep()` to repeat this sequence for each location.

```
geocodes <-
  tibble(lon = c(0, 0, 0, 0, 0),
         lat = c(66.5634, 23.4394, 0, -23.4394, -66.5634),
         address = c("Northern\nPolar\nCircle", "Tropic of\nCancer", "Equator",
                     "Tropic of\nCapricorn", "Southern\nPolar\nCircle"))
sun_elevation.tb <-
  tibble(when = rep(dates, nrow(geocodes)),
         where = rep(geocodes$address, each = length(dates)),
         plots = c(lapply(dates, make_sun_elevation_plot, geocode = geocodes[1, ]),
                   lapply(dates, make_sun_elevation_plot, geocode = geocodes[2, ]),
                   lapply(dates, make_sun_elevation_plot, geocode = geocodes[3, ]),
                   lapply(dates, make_sun_elevation_plot, geocode = geocodes[4, ]),
                  lapply(dates, make_sun_elevation_plot, geocode = geocodes[5, ])))
sun_elevation.tb$where <-
  factor(sun_elevation.tb$where, levels = rev(geocodes$address))
```

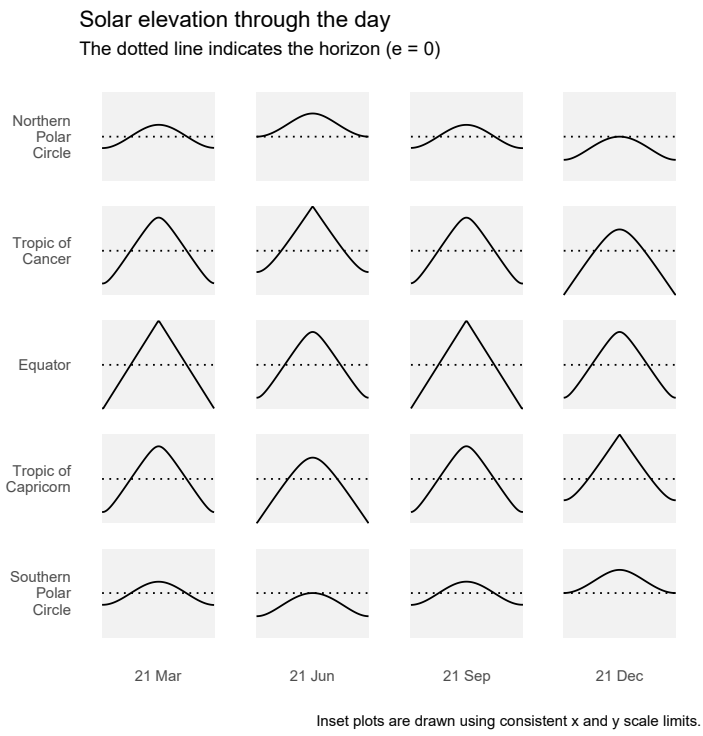
The only change from the code used above to plot a single timeline is related to the vertical size of the inset plots as it is expressed relative to the size of the whole plot. We also add a title, a subtitle and a caption, and tweak the theme of the main plot.

```
ggplot(sun_elevation.tb, aes(x = when, y = where, label = plots)) +
  geom_line() +
  geom_point(data = . %>% filter(day(when) == 21)) +
  geom_plot(data = . %>% filter(day(when) == 21),
            inherit.aes = TRUE,
            vp.width = 0.15, vp.height = 0.12, vjust = -0.1, hjust = 0.5) +
  scale_x_date(name = "", breaks = date.ticks, date_labels = "%d %b") +
  scale_y_discrete(name = "", expand = expansion(mult = c(0.05, 0.25))) +
  labs(title = "Solar elevation through the day",
       subtitle = "The dotted line indicates the horizon (e = 0)",
       caption = "Inset plots are drawn using consistent x and y scale limits.") +
  theme_minimal() +
  theme(panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        axis.ticks.x.bottom = element_line())
```



The five parallel timelines become rather crowded so an option is to build a matrix of plots. We achieve this by editing the code for the previous plot.

```
ggplot(sun_elevation.tb, aes(x = when, y = where, label = plots)) +
  geom_plot(data = . %>% filter(day(when) == 21),
            inherit.aes = TRUE,
            vp.width = 0.18, vp.height = 0.15, vjust = 0.5, hjust = 0.5) +
  scale_x_date(name = "", breaks = date.ticks,
               limits = date.ends,
               date_labels = "%d %b") +
  scale_y_discrete(name = "") +
  labs(title = "Solar elevation through the day",
       subtitle = "The dotted line indicates the horizon (e = 0)",
       caption = "Inset plots are drawn using consistent x and y scale limits.") +
  theme_minimal() +
  theme(panel.grid.minor = element_blank(),
        panel.grid.major = element_blank())
```



A.8 Related plots

Package 'ggpp' provides in addition to `geom_plot()`, `geom_table()` and `geom_grob()`. The first of them makes it possible to inset a data frame as a table, and the second any graphic object supported by package 'grid' (`grob` for short), which is part of R itself. These graphic objects can be vector graphics or bitmaps (or raster images) converted into `grob`s. Examples of how to convert bitmaps and vector graphics read from files of various formats is described in the documentation of packages 'grid', 'magick' and more briefly in package 'ggpp'. See the book *R Graphics* (Murrell 2011) for details.

Using photographs converted to grobs one can, for example, create phenological time lines. Another variation could to be to use a similar approach to represent geographic or topographic transects. In this last case instead of only using the x-axis to map time, one could map distance to the x-axis and elevation to the y-axis. Furthermore, one can use package 'patchwork' to assemble a multi-panel figure in which one panel is a timeline plot and other panels display other types of plots or even tables.

```
try(detach(package:photobiology))
try(detach(package:lubridate))
try(detach(package:ggrepel))
try(detach(package:ggpp))
try(detach(package:ggplot2))
try(detach(package:dplyr))
```

Bibliography

Aphalo, P. J. (2020). *Learn R: As a Language*. The R Series. CRC/Taylor & Francis Ltd. 350 pp. ISBN: 036718253X (cit. on p. ix).

Boas, R. P. (1981). "Can we make mathematics intelligible?" In: *The American Mathematical Monthly* 88.10, pp. 727–731.

Koponen, J. and J. Hildén (2019). *Data visualization handbook*. Espoo, Finland: Aalto University. ISBN: 9789526074498 (cit. on p. 386).

Murrell, P. (2011). *R Graphics*. 2nd ed. CRC Press, p. 546. ISBN: 1439831769 (cit. on p. 395).

Scherer, C. (2019). *A ggplot2 Tutorial for Beautiful Plotting in R*. URL: `https://www.cedricscherer.com/2019/08/05/a-ggplot2-tutorial-for-beautiful-plotting-in-r/` (visited on 07/21/2021) (cit. on p. ix).

Tufte, E. R. (1983). *The Visual Display of Quantitative Information*. Cheshire, CT: Graphics Press. 197 pp. ISBN: 0-9613921-0-X.

General index

Index of R names by category

Alphabetic index of R names