Appendix A

Translating between different syntaxes

A.1 Introduction

ggplot2 does not exist in isolation, but is part of a long history of graphical tools in R and elsewhere. This chapter describes how to convert between ggplot2 commands and other plotting systems:

- Within ggplot2, between the qplot() and ggplot() syntaxes, § A.2
- From base graphics, § A.3.
- From lattice graphics, § A.4.
- From GPL, § A.5.

Each section gives a general outline on how to convert between the difference types, followed by a number of examples.

A.2 Translating between qplot and ggplot

Within ggplot2, there are two basic methods to create plots, with qplot() and ggplot(). qplot() is designed primarily for interactive use: it makes a number of assumptions that speed most cases, but when designing multi-layered plots with different data sources it can get in the way. This section describes what those defaults are, and how they map to the fuller ggplot() syntax.

By default, qplot() assumes that you want a scatterplot, i.e., you want to use geom_point().

```
qplot(x, y, data = data)
ggplot(data, aes(x, y)) + geom_point()
```

A.2.1 Aesthetics

If you map additional aesthetics, these will be added to the defaults. With qplot() there is no way to use different aesthetic mappings (or data) in different layers.

```
qplot(x, y, data = data, shape = shape, colour = colour)
ggplot(data, aes(x, y, shape = shape, colour = colour)) +
  geom_point()
```

Aesthetic parameters in `qplot()` always try to map the aesthetic to a variable. If the argument is not a variable but a value, effectively a new column is added to the original dataset with that value. To set an aesthetic to a value and override the default appearance, you surround the value with `I()` in `qplot()`, or pass it as a parameter to the layer. Section 4.5.2 expands on the differences between setting and mapping.

```
qplot(x, y, data = data, colour = I("red"))
ggplot(data, aes(x, y)) + geom_point(colour = "red")
```

A.2.2 Layers

Changing the geom parameter changes the geom added to the plot:

```
qplot(x, y, data = data, geom = "line")
ggplot(data, aes(x, y)) + geom_line()
```

If a vector of multiple geom names is supplied to the geom argument, each geom will be added in turn:

```
qplot(x, y, data = data, geom = c("point", "smooth"))
ggplot(data, aes(x, y)) + geom_point() + geom_smooth()
```

Unlike the rest of `ggplot2`, stats and geoms are independent:

```
qplot(x, y, data = data, stat = "bin")
ggplot(data, aes(x, y)) + geom_point(stat = "bin")
```

Any layer parameters will be passed on to all layers. Most layers will ignore parameters that they don't need.

```
qplot(x, y, data = data, geom = c("point", "smooth"),
  method = "lm")
ggplot(data, aes(x, y)) +
  geom_point(method = "lm") + geom_smooth(method = "lm")
```

A.2.3 Scales and axes

You can control basic properties of the x and y scales with the `xlim`, `ylim`, `xlab` and `ylab` arguments:

```
qplot(x, y, data = data, xlim = c(1, 5), xlab = "my label")
ggplot(data, aes(x, y)) + geom_point() +
  scale_x_continuous("my label", limits = c(1, 5))

qplot(x, y, data = data, xlim = c(1, 5), ylim = c(10, 20))
ggplot(data, aes(x, y)) + geom_point() +
  scale_x_continuous(limits = c(1, 5))
  scale_y_continuous(limits = c(10, 20))
```

Like `plot()`, `qplot()` has a convenient way of log transforming the axes. There are many other possible transformations that are not accessible from within `qplot()` see Section 6.4.2 for more details.

```
qplot(x, y, data = data, log="xy")
ggplot(data, aes(x, y)) + geom_point() +
  scale_x_log10() + scale_y_log10()
```

A.2.4 Plot options

`qplot()` recognises the same options as plot does, and converts them to their `ggplot2` equivalents. Section 8.1.2 lists all possible plot options and their effects.

```
qplot(x, y, data = data, main="title", asp = 1)
ggplot(data, aes(x, y)) + geom_point() +
  opts(title = "title", aspect.ratio = 1)
```

A.3 Base graphics

There are two types of graphics functions in base graphics, those that draw complete graphics and those that add to existing graphics.

A.3.1 High-level plotting commands

`qplot()` has been designed to mimic `plot()`, and can do the job of all other high-level plotting commands. There are only two graph types from base graphics that cannot be replicated with `ggplot2`: `filled.contour()` and `persp()`

```
plot(x, y);  dotchart(x, y); stripchart(x, y)
qplot(x, y)

plot(x, y, type = "l")
qplot(x, y, geom = "line")
```

```
plot(x, y, type = "s")
qplot(x, y, geom = "step")

plot(x, y, type = "b")
qplot(x, y, geom = c("point", "line"))

boxplot(x, y)
qplot(x, y, geom = "boxplot")

hist(x)
qplot(x, geom = "histogram")

cdplot(x, y)
qplot(x, fill = y, geom = "density", position = "fill")

coplot(y ~ x | a + b)
qplot(x, y, facets = a ~ b)
```

Many of the geoms are parameterised differently than base graphics. For example, `hist()` is parameterised in terms of the number of bins, while `geom_histogram()` is parameterised in terms of the width of each bin.

```
hist(x, bins = 100)
qplot(x, geom = "histogram", binwidth = 1)
```

`qplot()` often requires data in a slightly different format to the base graphics functions. For example, the bar geom works with untabulated data, not tabulated data like `barplot()`; the tile and contour geoms expect data in a data frame, not a matrix like `image()` and `contour()`.

```
barplot(table(x))
qplot(x, geom = "bar")

barplot(x)
qplot(names(x), x, geom = "bar", stat = "identity")

image(x)
qplot(X1, X2, data = melt(x), geom = "tile", fill = value)

contour(x)
qplot(X1, X2, data = melt(x), geom = "contour", fill = value)
```

Generally, the base graphics functions work with individual vectors, not data frames like ggplot2. `qplot()` will try to construct a data frame if one is not specified, but it is not always possible. If you get strange errors, you may need to create the data frame yourself.

```
with(df, plot(x, y))
qplot(x, y, data = df)
```

By default, `qplot()` maps values to aesthetics with a scale. To override this behaviour and set aesthetics, overriding the defaults, you need to use `I()`.

```
plot(x, y, col = "red", cex = 1)
qplot(x, y, colour = I("red"), size = I(1))
```

A.3.2 Low-level drawing

The low-level drawing functions which add to an existing plot are equivalent to adding a new layer in `ggplot2`, described in Table A.1.

Base function	`ggplot2` layer
`curve()`	`geom_curve()`
`hline()`	`geom_hline()`
`lines()`	`geom_line()`
`points()`	`geom_point()`
`polygon()`	`geom_polygon()`
`rect()`	`geom_rect()`
`rug()`	`geom_rug()`
`segments()`	`geom_segment()`
`text()`	`geom_text()`
`vline()`	`geom_vline()`
`abline(lm(y ~ x))`	`geom_smooth(method = "lm")`
`lines(density(x))`	`geom_density()`
`lines(loess(x, y))`	`geom_smooth()`

Table A.1: Equivalence between base graphics methods that add on to an existing plot, and layers in `ggplot2`.

```
plot(x, y)
lines(x, y)

qplot(x, y) + geom_line()

# Or, building up piece-meal
qplot(x, y)
last_plot() + geom_line()
```

A.3.3 Legends, axes and grid lines

In ggplot2, the appearance of legends and axes is controlled by the scales. Axes are produced by the x and y scales, while all other scales produce legends. See plot themes, Section 8.1, to change the appearance of axes and legends, and, scales, Section 6.5, to change their contents. The appearance of grid lines is controlled by the `grid.major` and `grid.minor` theme options, and their position by the breaks of the x and y scales.

A.3.4 Colour palettes

Instead of global colour palettes, ggplot2 has scales for individual plots. Much of the time you can rely on the default colour scale (which has somewhat better perceptual properties), but if you want to reuse an existing colour palette, you can use `scale_colour_manual()`. You will need to make sure that the colour is a factor for this to work.

```
palette(rainbow(5))
plot(1:5, 1:5, col = 1:5, pch = 19, cex = 4)

qplot(1:5, 1:5, col = factor(1:5), size = I(4))
last_plot() + scale_colour_manual(values = rainbow(5))
```

In ggplot2, you can also use palettes with continuous values, with intermediate values being linearly interpolated.

```
qplot(0:100, 0:100, col = 0:100, size = I(4)) +
  scale_colour_gradientn(colours = rainbow(7))
last_plot() +
  scale_colour_gradientn(colours = terrain.colors(7))
```

A.3.5 Graphical parameters

The majority of `par` settings have some analogue within the theme system, or in the defaults of the geoms and scales. The appearance plot border drawn by `box()` can be controlled in a similar way by the `panel.background` and `plot.background` theme elements. Instead of using `title()`, the plot title is set with the `title` option.

A.4 Lattice graphics

The major difference between lattice and ggplot2 is that lattice uses a formula-based interface. ggplot2 does not because the formula does not generalise well to more complicated situations.

```
xyplot(rating ~ year, data=movies)
qplot(year, rating, data=movies)

xyplot(rating ~ year | Comedy + Action, data = movies)
qplot(year, rating, data = movies, facets = ~ Comedy + Action)
# Or maybe
qplot(year, rating, data = movies, facets = Comedy ~ Action)
```

While lattice has many different functions to produce different types of graphics (which are all basically equivalent to setting the panel argument), ggplot2 has `qplot()`.

```
stripplot(~ rating, data = movies, jitter.data = TRUE)
qplot(rating, 1, data = movies, geom = "jitter")

histogram(~ rating, data = movies)
qplot(rating, data = movies, geom = "histogram")

bwplot(Comedy ~ rating ,data = movies)
qplot(factor(Comedy), rating, data = movies, type = "boxplot")

xyplot(wt ~ mpg, mtcars, type = c("p","smooth"))
qplot(mpg, wt, data = mtcars, geom = c("point","smooth"))

xyplot(wt ~ mpg, mtcars, type = c("p","r"))
qplot(mpg, wt, data = mtcars, geom = c("point","smooth"),
  method = "lm")
```

The capabilities for scale manipulations are similar in both ggplot2 and lattice, although the syntax is a little different.

```
xyplot(wt ~ mpg | cyl, mtcars, scales = list(y = list(relation = "free")))
qplot(mpg, wt, data = mtcars) + facet_wrap(~ cyl, scales = "free")

xyplot(wt ~ mpg | cyl, mtcars, scales = list(log = 10))
qplot(mpg, wt, data = mtcars, log = "xy")

xyplot(wt ~ mpg | cyl, mtcars, scales = list(log = 2))
qplot(mpg, wt, data = mtcars) +
  scale_x_log2() + scale_y_log2()

xyplot(wt ~ mpg, mtcars, group = cyl, auto.key = TRUE)
# Map directly to an aesthetic like colour, size, or shape.
qplot(mpg, wt, data = mtcars, colour = cyl)

xyplot(wt ~ mpg, mtcars, xlim = c(20,30))
```

```
# Works like lattice, except you can't specify a different limit
# for each panel/facet
qplot(mpg, wt, data = mtcars, xlim = c(20,30))
```

Both lattice and ggplot2 have similar options for controlling labels on the plot.

```
xyplot(wt ~ mpg, mtcars,
  xlab = "Miles per gallon", ylab = "Weight",
  main = "Weight-efficiency tradeoff")
qplot(mpg, wt, data = mtcars,
  xlab = "Miles per gallon", ylab = "Weight",
  main = "Weight-efficiency tradeoff")

xyplot(wt ~ mpg, mtcars, aspect = 1)
qplot(mpg, wt, data = mtcars, asp = 1)
```

`par.settings()` is equivalent to `+ opts()` and `trellis.options.set()` and `trellis.par.get()` to `theme_set()` and `theme_get()`.

More complicated lattice formulas are equivalent to rearranging the data before using ggplot2.

A.5 GPL

The Grammar of Graphics uses two specifications. A concise format is used to caption figures, and a more detailed xml format stored on disk. The following example of the concise format is adapted from Wilkinson (2005, Figure 1.5, page 13).

```
DATA: source("demographics")
DATA: longitude, latitude = map(source("World"))
TRANS: bd = max(birth - death, 0)
COORD: project.mercator()
ELEMENT: point(position(lon * lat), size(bd), color(color.red))
ELEMENT: polygon(position(longitude * latitude))
```

This is relatively simple to adapt to the syntax of ggplot2:

- `ggplot()` is used to specify the default data and default aesthetic mappings.
- Data is provided as standard R data.frames existing in the global environment; it does not need to be explicitly loaded. We also use a slightly different world dataset, with columns lat and long. This lets us use the same aesthetic mappings for both datasets. Layers can override the default data and aesthetic mappings provided by the plot.
- We replace TRANS with an explicit transformation by R code.

- ELEMENTs are replaced with layers, which explicitly specify the data source. Each geom has a default statistic which is used to transform the data prior to plotting. For the geoms in this example, the default statistic is the identity function. Fixed aesthetics (the colour red in this example) are supplied as additional arguments to the layer, rather than as special constants.
- The SCALE component has been omitted from this example (so that the defaults are used). In both the `ggplot2` and GoG examples, scales are defined by default. In ggplot you can override the defaults by adding a scale object, e.g., `scale_colour` or `scale_size`.
- COORD uses a slightly different format. In general, most of the components specifications in ggplot are slightly different to those in GoG, in order to be more familiar to R users.
- Each component is added together with + to create the final plot.

All up the equivalent `ggplot2` code is:

```
demographics <- transform(demographics,
  bd = pmax(birth - death, 0))

ggplot(demographic, aes(lon, lat)) +
  geom_polyogon(data = world) +
  geom_point(aes(size = bd), colour = "red") +
  coord_map(projection = "mercator")
```

Appendix B

Aesthetic specifications

This appendix summarises the various formats that `grid` drawing functions take. Most of this information is available scattered throughout the R documentation. This appendix brings it all together in one place.

B.1 Colour

Colours can be specified with:

- A **name**, e.g., `"red"`. The colours are displayed in Figure B.1(a), and can be listed in more detail with `colours()`. The Stowers Institute provides a nice printable pdf that lists all colours: http://research.stowers-institute.org/efg/R/Color/Chart/.
- An **rgb specification**, with a string of the form `"#RRGGBB"` where each of the pairs `RR`, `GG`, `BB` consists of two hexadecimal digits giving a value in the range `00` to `FF`. Partially transparent can be made with `alpha()`, e.g., `alpha("red", 0.5)`.
- An **NA**, for a completely transparent colour.

The functions `rgb()`, `hsv()`, `hcl()` can be used to create colours specified in different colour spaces.

B.2 Line type

Line types can be specified with:

- An **integer** or **name**: 0=blank, 1=solid, 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash), illustrated in Figure B.1(b).
- The lengths of on/off stretches of line. This is done with a string of an even number (up to eight) of hexadecimal digits which give the lengths in consecutive positions in the string. For example, the string `"33"` specifies

three units on followed by three off and `"3313"` specifies three units on followed by three off followed by one on and finally three off.
The five standard dash-dot line types described above correspond to 44, 13, 134, 73 and 2262.

Note that `NA` is not a valid value for `lty`.

B.3 Shape

Shapes take four types of values:

- An **integer** in $[0, 25]$, illustrated in Figure B.1(c).
- A **single character**, to use that character as a plotting symbol.
- A `.` to draw the smallest rectangle that is visible (i.e., about one pixel).
- An **NA**, to draw nothing.

While all symbols have a foreground colour, symbols 19–25 also take a background colour (fill).

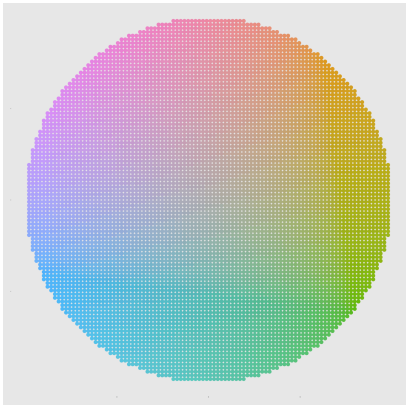
B.4 Size

Throughout `ggplot2`, for text height, point size and line width, size is specified in millimetres.

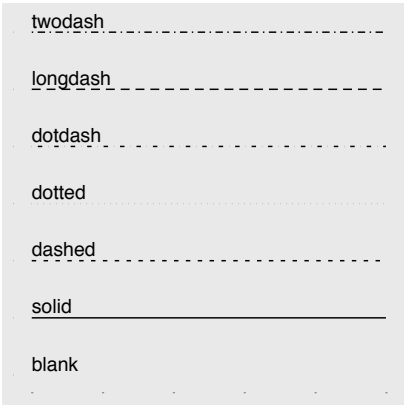
B.5 Justification

Justification of a string (or legend) defines the location within the string that is placed at the given position. There are two values for horizontal and vertical justification. The values can be:

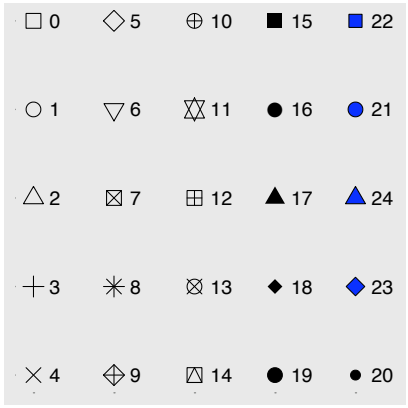
- A **string**: `"left"`, `"right"`, `"centre"`, `"center"`, `"bottom"`, and `"top"`.
- A **number** between 0 and 1, giving the position within the string (from bottom-left corner). These values are demonstrated in Figure B.1(d).



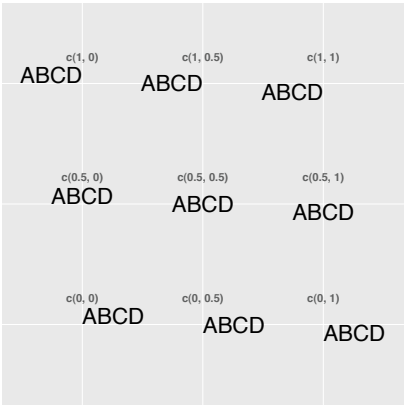
(a) All named colours in Luv space



(b) Built-in line types



(c) R plotting symbols. Colour is black, and fill is blue. Symbol 25 (not shown) is symbol 24 rotated 180 degrees.



(d) Horizontal and vertical justification settings.

Fig. B.1: Examples illustrating different aesthetic settings.

Appendix C

Manipulating plot rendering with `grid`

C.1 Introduction

Sometimes you may need to go beyond the theming system and directly modify the underlying grid graphics output. To do this, you will need a good understanding of grid, as described in "R Graphics" (Murrell, 2005). If you can't get the book, at least read Chapter 5, "The grid graphics model", which is available online for free at http://www.stat.auckland.ac.nz/~paul/RGraphics/chapter5.pdf. This appendix outlines the more important viewports and grobs used by `ggplot2` and should be helpful if you need to interact with the grobs produced by `ggplot2`.

C.2 Plot viewports

Viewports define the basic regions of the plot. The structure will vary slightly from plot to plot, depending on the type of faceting used, but the basics will remain the same.

The `panels` viewport contains the meat of the plot: strip labels, axes and faceted panels. The viewports are named according to both their job and their position on the plot. A prefix (listed below) describes the contents of the viewport, and is followed by integer x and y position (counting from bottom left) separated by "_". Figure C.1 illustrates this naming scheme for a 2×2 plot.

- `strip_h`: horizontal strip labels
- `strip_v`: vertical strip labels
- `axis_h`: horizontal axes
- `axis_v`: vertical axes
- `panel`: faceting panels

The `panels` viewport is contained inside the `background` viewport which also contains the following viewports:

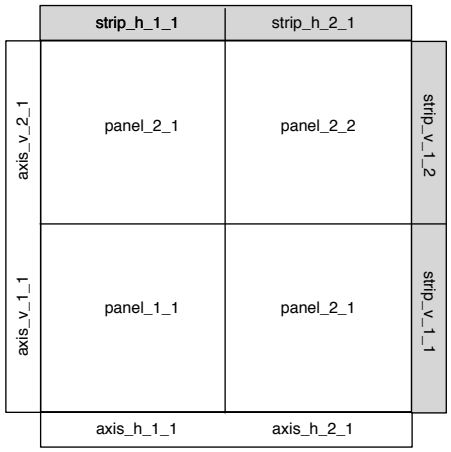


Fig. C.1: Naming scheming of the panel viewports.

- `title`, `xlabel` and `ylabel`: for the plot title, and x and y axis labels
- `legend_box`: for all of the legends for the plot

Figure C.2 labels a plot with a representative sample of these viewports. To get a list of all viewports on the current plot, run `current.vpTree(all=TRUE)` or `grid.ls(grobs = FALSE, viewports = TRUE)`.

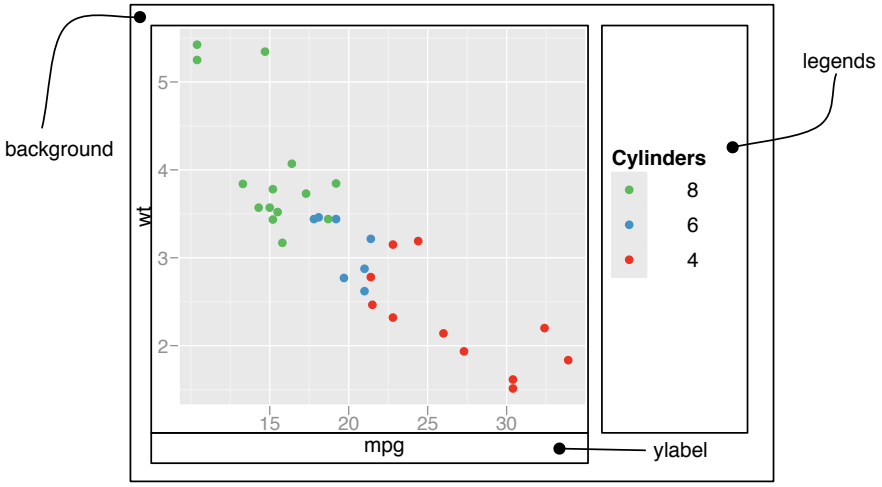


Fig. C.2: Diagram showing the structure and names of viewports.

C.3 Plot grobs

Grob names have three components: the name of the grob, the class of the grob and a unique numeric suffix. The three components are joined together with "." to give a name like `title.text.435` or `ticks.segments.15`. These three components ensure that all grob names are unique, and allow you to select multiple grobs with the same name at the same time. Figure C.3 labels some of these grobs. The grobs are arranged hierarchically, but it's hard to capture this in a diagram. You can see a list of all the grobs in the current plot with `grid.ls()`.

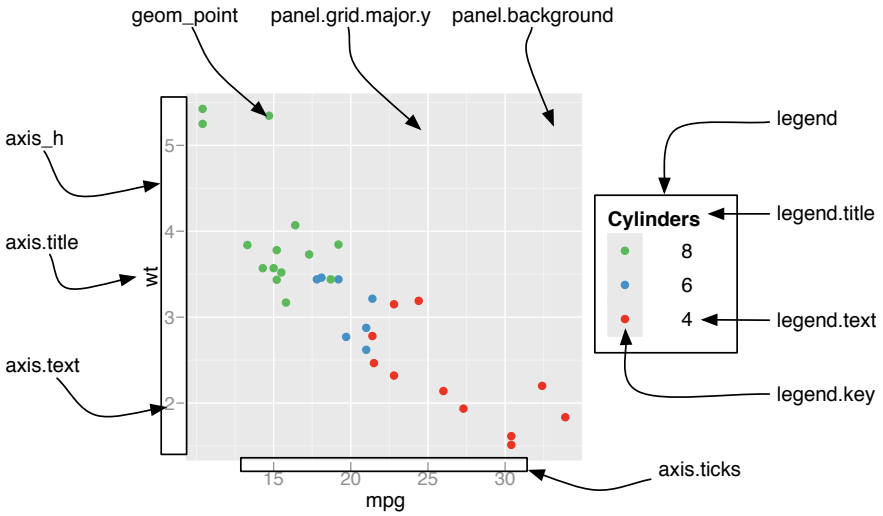


Fig. C.3: A selection of the most important grobs.

C.4 Saving your work

Using `grid.gedit()`, and similar functions, works fine if you are editing the plot on screen, but if you want to save it to disk you need to take some extra steps, or you will end up with multiple pages of output, each showing one change. The key is not to modify the plot on screen, but to modify the plot grob, and then draw it once you have made all the changes.

```
p <- qplot(wt, mpg, data=mtcars, colour=cyl)
# Get the plot grob
grob <- ggplotGrob(p)
# Modify in place
```

```
grob <- geditGrob(grob, gPath("strip","label"), gp=gpar(fontface="bold"))

# Draw it
grid.newpage()
grid.draw(grob)
```

An alternative is make all of the changes on screen, and then use `dev.copy2pdf()` to copy the final version to disk.