6 Data Visualisation

R. Heller · Date: 2026-05-11 · Last changed: 2026-10-30

The `ggplot2` grammar, panelled and faceted layouts, statistical graphics for regression and survival, and visual diagnostics. Style choices are biased toward what reproduces in print and what survives a black-and-white photocopy.

This chapter contains **35 method pages** and **1 lab**. If you are not sure which method to read, return to Chapter 0 and follow the decision tree to the right node.

6.1 Method pages

Method	Source slug
Aesthetics and Geoms	`aesthetics-and-geoms`
Annotations and Labels	`annotations-and-labels`
Bar Charts	`bar-charts`
Bland-Altman Plots	`bland-altman-plots`
Boxplots	`boxplots`
Bubble Plots	`bubble-plots`
Colour Palettes	`colour-palettes`
Colour-Blind-Safe Plots	`colour-blind-safe-plots`
Contour Plots	`contour-plots`
Correlation Heatmaps	`correlation-heatmaps`
Density Plots	`density-plots`
Dot Plots	`dot-plots`
Facets and Panels	`facets-and-panels`
Forest Plots (Visualisation)	`forest-plots-viz`
Funnel Plots (Visualisation)	`funnel-plots-viz`
ggplot2 Themes	`ggplot-themes`
Heatmaps	`heatmaps`
Hexbin Plots	`hexbin-plots`
Histograms	`histograms`
Interactive Plots with ggiraph	`interactive-ggiraph`
Interactive Plots with plotly	`interactive-plotly`
Line Plots	`line-plots`
Pairs Plots	`pairs-plots`
Patchwork: Multi-Plot Composition	`patchwork-composition`
Raincloud Plots	`raincloud-plots`

Method	Source slug
Ridge Plots	`ridge-plots`
ROC Curves	`roc-curves-plot`
Saving and Exporting Figures	`ggsave-and-export`
Scales and Coordinates	`scales-and-coordinates`
Scatter Plots	`scatter-plots`
Stacked and Dodged Bars	`stacked-dodged-bars`
Survival Curves	`survival-curves-plot`
The Grammar of Graphics	`grammar-of-graphics`
Time Series Plots	`time-series-plots`
Violin Plots	`violin-plots`

6.2 Labs

Lab
ggplot2 grammar and multi-panel layouts

6.3 Aesthetics and Geoms

6.3.1 Introduction

`ggplot2` is built on Wilkinson's grammar of graphics, which separates two concerns that ad-hoc plotting languages tend to conflate. An **aesthetic** is a mapping from a column of data to a visual property: x-position, y-position, colour, fill, shape, size. A **geom** is the geometric object drawn using those aesthetics: a point, a line, a bar, a histogram, a smoother. Once you internalise the split — "what variable becomes which visual property" versus "what shape do I want drawn" — `ggplot2` becomes a Lego kit of swappable parts rather than a dialect to memorise.

6.3.2 Prerequisites

A working knowledge of basic `ggplot2` syntax (`ggplot()`, `aes()`, `+ geom_*`), and tidy data conventions where each row is an observation and each column a variable.

6.3.3 Theory

An **aesthetic mapping** lives inside `aes()` and binds a column to a visual property:

```
aes(x = flipper_length_mm, y = body_mass_g, colour = species)
```

The mapping triggers automatic scale construction, legend generation, and axis labelling — `ggplot2` knows that a continuous numeric column needs a continuous scale and a categorical column needs a discrete one.

A **geom** consumes the aesthetics and draws the corresponding marks. Common geoms include `geom_point()` for scatter, `geom_line()` for lines, `geom_bar()` and `geom_col()` for bars, `geom_histogram()` and `geom_density()` for distributions, `geom_smooth()` for trend lines.

Mapping vs setting is the most important distinction in `ggplot2`. Inside `aes()` binds a visual property to a variable, producing one value per row and a legend. Outside `aes()` sets a constant for all rows with no legend. `geom_point(colour = "blue")` makes every point blue; `geom_point(aes(colour = species))` colours points by species.

Multiple geoms in the same plot share base aesthetics declared in `ggplot(aes(...))` and may add their own. This layering is the heart of the grammar: a complex figure is just a stack of compatible geoms reading the same data.

6.3.4 Assumptions

Tidy data: one observation per row, one variable per column. Long-format frames are the standard input for `ggplot2`; wide-format data should be reshaped via `tidyr::pivot_longer()` before plotting.

6.3.5 R Implementation

```
library(ggplot2); library(palmerpenguins)
data(penguins, package = "palmerpenguins")

# Basic mapping: x, y, colour
ggplot(penguins, aes(flipper_length_mm, body_mass_g, colour = species)) +
  geom_point(alpha = 0.7) +
  theme_minimal()

# Adding a smoother geom on top
ggplot(penguins, aes(flipper_length_mm, body_mass_g, colour = species)) +
  geom_point(alpha = 0.7) +
  geom_smooth(method = "lm", se = FALSE) +
  theme_minimal()

# Aesthetics differ between geoms: shape is mapped only for points
ggplot(penguins, aes(flipper_length_mm, body_mass_g)) +
  geom_point(aes(colour = species, shape = sex), alpha = 0.7) +
  theme_minimal()

# Constant aesthetics outside aes()
ggplot(penguins, aes(flipper_length_mm, body_mass_g)) +
  geom_point(colour = "steelblue", size = 3, alpha = 0.6) +
  theme_minimal()
```

6.3.6 Output & Results

Four versions of the same scatter: a basic colour-coded plot, the same plot with per-species linear smoothers added, a four-aesthetic plot mapping species to colour and sex to shape simultaneously, and a single-colour version where every aesthetic except position is held constant. Each illustrates a different combination of mapping and setting.

6.3.7 Interpretation

A reporting sentence (figure caption): "Body mass against flipper length (n = 333 penguins, three species); colour encodes species and shape encodes sex. Per-species linear smoothers are overlaid in the right panel." Be explicit about which aesthetics carry information and which are constant; readers should not have to infer the mapping.

6.3.8 Practical Tips

- Put aesthetics shared across geoms in the base `ggplot()` call; put geom-specific aesthetics inside the geom. Repetition is fine for clarity but is rarely necessary.
- Reach for `alpha` to mitigate overplotting in dense scatter plots; values between 0.2 and 0.5 typically reveal density structure.
- `fill` controls the interior of geoms with area (bars, boxplots, polygons); `colour` controls edges and points. Mixing them up is one of the most common `ggplot2` errors.
- For categorical colour scales, prefer `viridis::scale_colour_viridis_d()` or `RColorBrewer::scale_colour_brewer()`; avoid the default rainbow.
- Use `shape` only for two to six groups; beyond that, shapes become indistinguishable and a facet or text label communicates better.
- When you want a constant value but `ggplot2` keeps drawing a legend, you forgot to move the assignment outside `aes()`; this is the single most common debugging step.

6.3.9 R Packages Used

`ggplot2` for the core grammar of graphics; `palmerpenguins` for a clean teaching dataset rich in aesthetic-mapping opportunities; `viridis` and `RColorBrewer` for accessible categorical and continuous scales; `ggrepel` for non-overlapping point labels that supplement aesthetic mappings with text.

6.3.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.3.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.4 Annotations and Labels

6.4.1 Introduction

A figure that shows the data without any annotation forces the reader to do the interpretive work alone. A figure with too many annotations buries the data under labels, arrows, and ornamental boxes. The middle ground — a few well-placed labels, an arrow that points at the one observation worth singling out, a rich-text title that names the comparison — is what separates publication-quality plots from informative-but-amateurish ones. The `ggplot2` ecosystem provides three families of annotation tools that cover virtually every requirement: `annotate()` for single hand-placed items, `geom_text` / `geom_label` (and the repelling `ggrepel` variants) for data-driven labels, and `ggtext` for rich Markdown and HTML inside titles, axes, and strip text.

6.4.2 Prerequisites

A working knowledge of `ggplot2`'s grammar of graphics, layers, and aesthetic mappings.

6.4.3 Theory

- `annotate()` adds a single geom (text, segment, rectangle, point) at hard-coded coordinates with no aesthetic mapping from the data; this is the right tool for a one-off label or arrow that does not depend on a row of the data frame.
- `geom_text()` and `geom_label()` map labels to rows of the data, drawing one label per row at the corresponding (x, y) coordinates.
- `ggrepel::geom_text_repel()` and `geom_label_repel()` solve the universal overlap problem by iteratively pushing labels away from each other and from the data points, with adjustable force, segment lines, and exclusion regions.
- `ggtext::element_markdown()` parses Markdown and a subset of HTML in axes, legends, titles, and strip text, enabling inline italics, bold, colour, and superscripts without dropping into expression syntax.

6.4.4 Assumptions

No statistical assumptions; the only requirement is that text strings be on the same scale as the surrounding data so labels appear in the right region of the plot.

6.4.5 R Implementation

```
library(ggplot2); library(ggrepel); library(ggtext)

p ← ggplot(mtcars, aes(wt, mpg)) +
  geom_point() +
  theme_minimal()

# Single annotation
p + annotate("text", x = 3, y = 30, label = "Lightweight, efficient", size = 5)

# Labelled points with repel
p + geom_text_repel(aes(label = rownames(mtcars)), size = 3)

# Arrow with annotation
p + annotate("segment", x = 5, xend = 4, y = 25, yend = 20,
             arrow = arrow(length = unit(0.3, "cm"))) +
  annotate("text", x = 5.1, y = 25.5, label = "Outlier")

# Markdown in titles
p + labs(title = "Weight vs <span style='color:#2A9D8F'>*mpg*</span>") +
  theme(plot.title = element_markdown())
```

6.4.6 Output & Results

Four variations on the same scatterplot: one with a hand-placed text annotation, one with each car labelled by name and laid out by `ggrepel` so labels do not collide, one with an arrow drawing attention to a single point, and one with a Markdown-formatted title. The point geom remains identical across all four; only the annotation layer changes.

6.4.7 Interpretation

A reporting sentence: “We labelled the four cars with the highest fuel efficiency using `ggrepel::geom_text_repel()` (force = 1, max.overlaps = 10), and emphasised the outlying Maserati Bora with a hand-placed arrow.” When annotations carry interpretive content, describe them in the figure caption so readers do not have to guess what is being highlighted.

6.4.8 Practical Tips

- Prefer `ggrepel::geom_text_repel()` over `geom_text()` whenever labels can overlap; the repelling variants always look more polished and are robust to data updates.
- `annotate()` is the right tool for a single fixed label; using `geom_text()` with a one-row tibble is unnecessarily verbose.
- For a small set of labels with the same style, build a small data frame and map all aesthetics through `geom_text()` rather than stacking many `annotate()` calls.
- Keep annotations away from data marks, legends, and figure margins; if you cannot, increase the plot size before re-styling the labels.
- Use `ggtext::element_markdown()` for inline italics in scientific names, bold for statistical thresholds, and colour for paired emphasis between text and data; it is far cleaner than `expression()`-based plotmath.

- Save the labelled and unlabelled versions of any figure separately; reviewers occasionally request a “clean” version for the supplement.

6.4.9 R Packages Used

`ggplot2` for the base plot system, `ggrepel` for collision-free labels, and `ggtext` for Markdown/HTML rendering inside themes; `cowplot` and `patchwork` complement the workflow when you need to compose annotated figures into multi-panel layouts.

6.4.10 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

6.4.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.5 Bar Charts

6.5.1 Introduction

Bar charts are the workhorse for categorical data and the geom most frequently misused. The confusion almost always stems from a single distinction: a bar can show how many observations fall in a category (a count) or a summary statistic computed for that category (a mean, a percentage, a proportion). `ggplot2` separates the two cases through `geom_bar()` for counts and `geom_col()` for pre-computed values, and getting the right geom for the data on hand is the first step to a sensible figure.

6.5.2 Prerequisites

A working knowledge of categorical variables, basic summary statistics (mean, standard error), and `ggplot2` ’s aesthetic mappings.

6.5.3 Theory

`geom_bar()` defaults to `stat = "count"` : it tabulates rows by the x-aesthetic and draws bars whose heights equal those counts. No y-aesthetic is required; supplying one along with `stat = "identity"` is equivalent to using `geom_col()` directly.

`geom_col()` plots an explicit y-aesthetic as the bar height — useful when the data are already summarised. For group means with uncertainty, pair `geom_col()` with `geom_errorbar()` whose `ymin` and `ymax` come from a confidence interval or standard error column.

The cognitive risk with bar-plus-error-bar plots is well documented: they hide the within-group distribution and overemphasise the mean. For showing distributional information, raincloud plots, dot plots, or boxplots are more transparent. Bars remain the right choice for genuine counts and for cumulative quantities (proportions, percentages, totals).

6.5.4 Assumptions

For count bars: the x-axis is categorical or discrete. For summary bars: the y-aesthetic is a meaningful summary of the data, ideally one that respects the bar's "starts at zero" implication.

6.5.5 R Implementation

```r
library(ggplot2); library(dplyr)

# Counts
ggplot(mpg, aes(class)) +
  geom_bar(fill = "#2A9D8F") +
  theme_minimal()

# Ordered bars by count
ggplot(mpg, aes(forcats::fct_infreq(class))) +
  geom_bar(fill = "#2A9D8F") +
  labs(x = "Class") +
  theme_minimal()

# Summarised values (e.g., mean mpg by class)
mpg_summary ← mpg ▷ group_by(class) ▷
  summarise(mean_hwy = mean(hwy), se = sd(hwy)/sqrt(n()))
ggplot(mpg_summary, aes(class, mean_hwy)) +
  geom_col(fill = "#2A9D8F") +
  geom_errorbar(aes(ymin = mean_hwy - se, ymax = mean_hwy + se),
                width = 0.2) +
  theme_minimal()
```

6.5.6 Output & Results

Three bar charts: a count of cars per class in alphabetical order, the same data with categories ordered by descending frequency, and a summary chart of mean highway mileage per class with ± 1 standard-error bars. Reordering and the explicit summary radically improve readability over the default.

6.5.7 Interpretation

A reporting sentence (figure caption): "Bar heights show the mean highway fuel efficiency per class (n varies, see Table 1); error bars are ± 1 standard error of the mean. Classes are ordered alphabetically." Always state the summary statistic and the uncertainty measure in the caption — bars are not self-explanatory.

6.5.8 Practical Tips

- Order categories by value (`forcats::fct_reorder()` or `fct_infreq()`) rather than alphabetically; sorted bars communicate ranking instantly.

- Use `geom_col()` when you already have summary statistics and `geom_bar()` (default `stat = "count"`) when you want raw frequencies.
- For two-category comparisons, a simple sentence often beats a bar chart; reserve bars for three or more groups.
- Add the underlying observations as jittered points (`geom_jitter(width = 0.2, alpha = 0.4)`) on top of summary bars to recover the distributional information bars hide.
- Always label the y-axis with units when bars summarise quantities ("Mean fuel economy (mpg)") rather than counts.
- Avoid 3D bars, gradient fills along the bar length, and other ornamental flourishes; they distort the perceptual mapping between bar length and value.

6.5.9 R Packages Used

`ggplot2` for `geom_bar()`, `geom_col()`, and `geom_errorbar()`; `dplyr` and `forcats` for the summary and category-ordering pipeline; `ggdist` if you want to overlay distributional summaries (slabintervals) on top of bar charts to communicate spread alongside the mean.

6.5.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.5.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.6 Bland-Altman Plots

6.6.1 Introduction

The Bland-Altman plot is the standard graphical tool for assessing agreement between two measurement methods, two raters, or two devices. Unlike a correlation coefficient or a 45-degree scatter, which conflate agreement with linear association, a Bland-Altman plot displays the *difference* between paired measurements against their *mean*, exposing systematic bias, proportional bias, and the limits within which 95 % of differences are expected to fall. Since Bland and Altman's 1986 *Lancet* paper, it has become the canonical method-comparison figure in clinical chemistry and laboratory medicine.

6.6.2 Prerequisites

A working understanding of paired measurements, mean and standard deviation, and the distinction between agreement (do two methods give the same value?) and correlation (do two methods rank observations the same way?).

6.6.3 Theory

For paired measurements (X_i, Y_i), compute the mean $M_i = (X_i + Y_i)/2$ and the difference $D_i = X_i - Y_i$. Plot M on the x-axis and D on the y-axis, with three horizontal reference lines:

- **Bias**: $\bar{D}$, the mean of the differences. Non-zero bias indicates systematic disagreement.
- **Upper limit of agreement (LoA)**: $\bar{D} + 1.96 \cdot \mathrm{SD}(D)$.
- **Lower LoA**: $\bar{D} - 1.96 \cdot \mathrm{SD}(D)$.

If the differences are approximately Normal, about 95 % of points fall within the LoA. Trends in D as a function of M — typically tested by regressing D on M — indicate proportional bias, where the disagreement scales with magnitude. Heteroscedasticity (a fan-shaped pattern) suggests the LoA should be expressed as a percentage of the mean rather than as an absolute interval.

6.6.4 Assumptions

Differences approximately Normal, no proportional bias (constant spread across the measurement range), and independent paired observations. For repeated measures within subjects, the modified Bland-Altman 1999 method partitions within- and between-subject variance.

6.6.5 R Implementation

```r
library(ggplot2); library(blandr)

set.seed(2026)
m1 <- rnorm(80, 100, 15)
m2 <- m1 + rnorm(80, 2, 5)

d <- data.frame(M = (m1 + m2)/2, D = m1 - m2)
bias <- mean(d$D)
loa <- bias + c(-1, 1) * 1.96 * sd(d$D)

ggplot(d, aes(M, D)) +
  geom_point(alpha = 0.6) +
  geom_hline(yintercept = bias, colour = "#2A9D8F", linewidth = 1) +
  geom_hline(yintercept = loa, colour = "#F4A261", linetype = "dashed") +
  labs(x = "Mean of methods", y = "Difference (M1 - M2)") +
  theme_minimal()

# With blandr
blandr.draw(m1, m2, plotter = "ggplot")
```

6.6.6 Output & Results

A scatter of differences vs means with three reference lines: a solid line at the bias of approximately 2 units and two dashed lines at the upper and lower limits of agreement of approximately -7.8 and 11.8 units. About 4 of the 80 points (5 %) should fall outside the LoA under the Normality assumption.

6.6.7 Interpretation

A reporting sentence: "Bland-Altman analysis showed a bias of $+2.0$ units (95 % CI 0.9 to 3.1) and 95 % limits of agreement of -7.8 to 11.8 units; clinical equivalence requires the LoA to fall within ± 5 units, so the methods are not interchangeable as currently calibrated." Always report the LoA with their 95 % confidence intervals — the LoA endpoints are themselves uncertain, especially for $n < 100$.

6.6.8 Practical Tips

- Check for proportional bias by regressing D on M; a significant slope indicates the disagreement scales with magnitude and the constant LoA misrepresents the data.
- Report the LoA with confidence intervals; the standard formulae give intervals of $\pm 1.71 \cdot \mathrm{SE}$ around each LoA.
- For repeated measures within subjects, use Bland-Altman's 1999 modified method that splits the variance into within- and between-subject components; ignoring the structure underestimates the LoA.
- A 45-degree scatter complements (but does not replace) the Bland-Altman plot; it reveals correlation but hides the level-dependent bias the Bland-Altman exposes.
- For heteroscedastic differences, log-transform both measurements first; the resulting Bland-Altman is on the log scale and the LoA become percentages.
- `blandr::blandr.statistics()` returns the full statistical summary including bias, LoA, their CIs, and a Shapiro-Wilk test for the Normality of differences.

6.6.9 R Packages Used

`blandr` for `blandr.statistics()` and `blandr.draw()`; `BlandAltmanLeh` for an alternative interface; `ggplot2` for hand-built versions; `epiR::epi.ccc()` for the concordance correlation coefficient as a complementary single-number agreement summary.

6.6.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.6.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.7 Boxplots

6.7.1 Introduction

A boxplot — also called a box-and-whisker plot — compresses a continuous distribution into a five-number summary: minimum, first quartile, median, third quartile, maximum, with Tukey's fences at $\pm 1.5 \times \mathrm{IQR}$ defining the whiskers and flagging anything beyond as an outlier. Boxplots are the workhorse for compact group-by-group comparisons because they put many groups side by side without ink overload. They are also the geom most often misused: they hide bimodality, hide sample size, and hide the raw observations behind a tidy rectangle, all of which can mislead readers who do not realise what the box actually summarises.

6.7.2 Prerequisites

A working understanding of quartiles, the interquartile range, and the standard $\pm 1.5 \cdot \mathrm{IQR}$ outlier convention.

6.7.3 Theory

The standard Tukey boxplot has the following components per group:

- A box spanning the first and third quartiles (the IQR).
- A horizontal line at the median inside the box.
- Whiskers extending to the most extreme observation within $\pm 1.5 \times \mathrm{IQR}$ of the box edges.
- Individual points beyond the whiskers drawn as outliers.

Variants:

- **Notched boxplot**: notches at $\pm 1.57 \cdot \mathrm{IQR}/\sqrt{n}$ give an approximate 95 % confidence interval for the median; non-overlapping notches between groups suggest a real difference in medians.
- **Variable-width boxplot**: box width proportional to $\sqrt{n}$, encoding group size visually.
- **Letter-value plot**: replaces the single box with nested boxes of decreasing depth, suitable for very large n where a single Tukey box hides tail structure.

6.7.4 Assumptions

For descriptive use, none. For the notched-boxplot CI interpretation, the approximation assumes approximately symmetric within-group distributions; under heavy skew the notches mislead.

6.7.5 R Implementation

```r
library(ggplot2)

# Standard
ggplot(mtcars, aes(factor(cyl), mpg, fill = factor(cyl))) +
  geom_boxplot() +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()

# Notched
ggplot(iris, aes(Species, Sepal.Length, fill = Species)) +
  geom_boxplot(notch = TRUE) +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()

# Boxplot + jittered points (show data and summary)
ggplot(mtcars, aes(factor(cyl), mpg)) +
  geom_boxplot(outlier.shape = NA, fill = "#2A9D8F", alpha = 0.5) +
  geom_jitter(width = 0.2, alpha = 0.5) +
  theme_minimal()
```

6.7.6 Output & Results

Three views: a standard side-by-side boxplot of mpg by cylinder count, a notched boxplot of iris sepal length per species (notches show approximate median CIs), and a hybrid plot that hides the default outlier dots and overlays jittered raw points so the boxplot's summary and the underlying data are both visible.

6.7.7 Interpretation

A reporting sentence (figure caption): "Boxplots of fuel efficiency by cylinder count (n = 11, 7, 14 for 4, 6, 8 cylinders); boxes span Q1 to Q3, the line marks the median, whiskers extend to the most extreme observation within $1.5 \cdot \mathrm{IQR}$, and dots are outliers." Always describe the whisker definition; some software uses 5 %/95 % quantiles or min/max instead of the Tukey rule.

6.7.8 Practical Tips

- Hide default outliers (`outlier.shape = NA`) whenever you overlay `geom_jitter()`; otherwise extreme points appear twice.
- Notches can overshoot the box for very small samples; R prints a warning when this happens, and the plot becomes uninterpretable.
- Use `varwidth = TRUE` to encode group sizes via box width when groups are unbalanced; the visual cue is subtle but informative.
- For very small samples ($n < 10$), prefer a dot plot or strip plot over a boxplot; quartile-based summaries are unstable for tiny groups.
- Boxplots hide multi-modality; always inspect a density or histogram alongside before concluding that a group is unimodal.
- For very large samples, a letter-value plot (`lvplot::geom_lv()`) preserves more of the tail structure than a single Tukey box.

6.7.9 R Packages Used

`ggplot2` for `geom_boxplot()` and the `varwidth` / `notch` parameters; `lvplot` for letter-value plots at very large n; `ggbeeswarm` and `ggdist` for richer alternatives that combine boxplot summaries with raw points and distributional shape.

6.7.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.7.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.8 Bubble Plots

6.8.1 Introduction

A bubble plot is a scatter plot enriched with a third continuous variable encoded by point size. It is a standard idiom in business analytics ("revenue × growth × market share") and in international development reporting (Hans Rosling's life-expectancy × income × population). The third dimension comes at a real perceptual cost — humans judge area less accurately than position — so a bubble plot's size aesthetic should be reserved for variables where rank order and rough magnitude matter more than precise comparison.

6.8.2 Prerequisites

A working knowledge of scatter plots, `ggplot2`'s aesthetic mappings, and the perceptual principles of position vs area encoding.

6.8.3 Theory

A bubble plot maps three continuous variables: x, y, and `size`. The size mapping must use an *area* scale — `scale_size_area()` — so that perceived magnitude (area) is proportional to the data value. The default `scale_size()` maps the data to point radius, which means a doubled data value occupies four times the visual area; this is a long-standing bug-as-feature in many bubble-chart implementations and produces misleading impressions of magnitude. Colour can encode a fourth variable, but four-variable bubble plots tend to overwhelm readers; alternatives include a facet on the categorical fourth variable or an animated time slider for longitudinal data.

6.8.4 Assumptions

All three (or four) variables are continuous (or treated as such). The size variable is non-negative and on a scale where zero means "absent"; mapping size to a centred or signed quantity is undefined.

6.8.5 R Implementation

```r
library(ggplot2)

ggplot(mtcars, aes(wt, mpg, size = hp, colour = factor(cyl))) +
  geom_point(alpha = 0.7) +
  scale_size_area(max_size = 12, name = "Horsepower") +
  scale_colour_brewer(palette = "Set2", name = "Cylinders") +
  theme_minimal()
```

6.8.6 Output & Results

A scatter of weight vs. miles-per-gallon with bubble area proportional to horsepower and colour encoding cylinder count. Heavy, high-horsepower cars cluster in the lower-right corner; light four-cylinder cars cluster in the upper-left. The combination of size and colour makes structural patterns visible that would require a four-panel facet to communicate without bubbles.

6.8.7 Interpretation

A reporting sentence (figure caption): "Bubble plot of vehicle weight vs fuel economy (n = 32); bubble area is proportional to horsepower (`scale_size_area` , max area 12), and colour encodes cylinder count." Always specify that area (not radius) is the encoding; readers familiar with poorly designed bubble charts will otherwise assume the wrong mapping.

6.8.8 Practical Tips

- Always use `scale_size_area()` so the visual area is proportional to the data; the default radius mapping double-counts magnitude.
- Set `max_size` deliberately (typically 8–15) to avoid enormous bubbles dominating the panel and obscuring smaller observations.
- For many overlapping bubbles, lower `alpha` to 0.4–0.6 and prefer a sequential fill so density still reads through the transparency.
- Do not map size to negative or signed values; area is undefined for negatives, and the visual mapping is ambiguous.
- For a fourth dimension, prefer a facet over a colour aesthetic on top of size; combining both fights for the reader's attention.
- For time series of bubbles ("Rosling-style"), use `gganimate::transition_time(year)` rather than a static panel; the temporal dimension is the variable that makes bubble plots compelling.

6.8.9 R Packages Used

`ggplot2` for the underlying scatter, `scale_size_area()` for correct area encoding, `gganimate` for time-animated bubble charts, and `ggrepel` for non-overlapping bubble labels when individual observations need to be named.

6.8.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.8.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.9 Colour-Blind-Safe Plots

6.9.1 Introduction

Roughly 8 % of men of European descent and about 0.5 % of women have some form of colour-vision deficiency. Red-green deficiency (deuteranopia and protanopia) is by far the most common; blue-yellow (tritanopia) is rarer but still present in any large readership. A figure that relies on red-versus-green as its only contrast is unreadable for a meaningful fraction of every audience, including journal reviewers. Designing for colour-blind safety is therefore not optional politeness — it is a basic requirement of accessible scientific communication, increasingly mandated by journal style guides and by web-accessibility standards (WCAG 2.x AA).

6.9.2 Prerequisites

A working knowledge of `ggplot2`'s colour and fill scales, basic familiarity with palette functions, and an understanding that "colour" includes both qualitative palettes (categorical groups) and sequential / diverging palettes (ordered or signed values).

6.9.3 Theory

Safe design strategies fall into three categories:

- **Perceptually uniform palettes**: `viridis`, `cividis`, `magma`, and `inferno` are designed so that equal steps in the colour space correspond to equal steps in perceived brightness, even under simulated colour-blindness. They handle the continuous case well.
- **Hand-curated qualitative palettes**: `RColorBrewer`'s `Set2`, `Dark2`, and `Paired` are explicitly tested for colour-blind safety up to a certain number of categories (six to

eight); beyond that, distinguishability degrades.
- **Redundant encoding**: pair colour with shape, linetype, line weight, or a direct text label so a reader who cannot distinguish the colours can still tell the groups apart from a second visual channel.

Simulation tools such as `colorBlindness::cvdPlot()` and `dichromat` re-render any `ggplot` under deuteranopia, protanopia, and tritanopia simulations; treat them as the colour equivalent of a spell-checker run before submission.

6.9.4 Assumptions

No statistical assumptions; the only constraint is that the figure's information content survives loss of one colour channel.

6.9.5 R Implementation

```
library(ggplot2); library(viridis); library(colorBlindness)

# Unsafe: default red-green
p_bad ← ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point(size = 3) +
  scale_colour_manual(values = c("red", "orange", "green")) +
  theme_minimal()

# Safe: viridis + shape redundancy
p_good ← ggplot(mtcars, aes(wt, mpg, colour = factor(cyl), shape = factor(cyl))) +
  geom_point(size = 3) +
  scale_colour_viridis_d(option = "D", end = 0.85) +
  theme_minimal()

cvdPlot(p_good)   # simulates deuteranopia, protanopia, tritanopia
```

6.9.6 Output & Results

`cvdPlot()` returns a four-panel composite showing the figure under normal vision, deuteranopia, protanopia, and tritanopia. A safely designed plot remains interpretable in all four panels; an unsafe one collapses to indistinguishable groups in at least one.

6.9.7 Interpretation

A reporting sentence (methods or figure-preparation note): "All figures use the `viridis` palette for ordered fills and `RColorBrewer::Set2` for qualitative colours; categorical groups are encoded redundantly with shape so colour-blind readers can distinguish them. Plots were simulated under deuteranopia and protanopia (`colorBlindness::cvdPlot`) before submission." Mentioning the accessibility check pre-empts reviewer requests to redo the figures.

6.9.8 Practical Tips

- Default to `viridis` or `cividis` for continuous scales; both are perceptually uniform and remain ordered under all common colour-blindness simulations.

- For categorical groups, pair colour with shape (`shape = group`) or with linetype (`linetype = group`); colour alone is insufficient.
- Avoid red-on-green for success/failure annotations; use blue-on-orange or a categorical palette like `Okabe-Ito`.
- Use `colorBlindness::cvdPlot()` (or the `Color Oracle` desktop tool) on every figure before submission; simulating colour-vision deficiencies takes seconds and catches problems that the human eye cannot.
- Beyond colour, ensure adequate contrast (WCAG AA minimum 4.5:1 for body text), avoid pure red or pure green for thin lines, and keep font sizes large enough to remain legible in greyscale photocopies.
- For network or flow diagrams, prefer position and topology to colour as the primary information channel; colour should be reinforcement, not the only signal.

6.9.9 R Packages Used

`viridis` for perceptually uniform sequential palettes, `RColorBrewer` for qualitative and diverging palettes (`brewer.pal`, `display.brewer.all(colorblindFriendly = TRUE)`), `colorBlindness` for `cvdPlot()` simulations, and `paletteer` for unified access to dozens of curated palettes from across the R ecosystem.

6.9.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.9.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.10 Colour Palettes

6.10.1 Introduction

Colour in a data visualisation is not decoration — it encodes information. The right palette makes ordered values read as ordered, signed values read as signed, and categorical values read as distinct without implying a ranking. The wrong palette (the infamous rainbow) creates false visual hotspots where the hue cycle bends and obscures structure that a perceptually uniform palette would reveal. Mastering the three palette families and choosing among them deliberately is one of the highest-payoff skills in figure design.

6.10.2 Prerequisites

A working knowledge of `ggplot2`'s colour and fill scales, and an awareness of colour-vision deficiencies that affect roughly 8 % of male readers.

6.10.3 Theory

Three palette families serve three different data types:

- **Sequential**: monotone hue or saturation gradient from low to high (`viridis`, `plasma`, `magma`, `YlOrRd`, `Blues`); for ordered data without a meaningful zero or centre.
- **Diverging**: two hues meeting at a neutral midpoint (`RdBu`, `PuOr`, `BrBG`); for signed data where positive and negative deviations from a centre are both meaningful.
- **Qualitative**: distinct, equally salient hues (`Set1`, `Set2`, `Dark2`, `Okabe-Ito`); for unordered categorical groups, ideally limited to eight or fewer.

`viridis` is the modern default for continuous scales because it is perceptually uniform (equal data steps look like equal visual steps), colour-blind safe under deuteranopia and protanopia, and degrades gracefully to greyscale. ColorBrewer palettes label colour-blind-friendly variants explicitly (`display.brewer.all(colorblindFriendly = TRUE)`), and the Okabe-Ito eight-colour palette was designed specifically for accessibility.

6.10.4 Assumptions

The data type matches the palette family — sequential for ordered values, diverging for signed, qualitative for nominal.

6.10.5 R Implementation

```
library(ggplot2); library(viridis); library(RColorBrewer)

# Continuous: viridis
ggplot(faithfuld, aes(waiting, eruptions, fill = density)) +
  geom_tile() +
  scale_fill_viridis_c(option = "plasma") +
  theme_minimal()

# Continuous: diverging
ggplot(mtcars, aes(wt, mpg, colour = hp - mean(hp))) +
  geom_point(size = 3) +
  scale_colour_gradient2(low = "blue", mid = "grey80", high = "red",
                         midpoint = 0) +
  theme_minimal()

# Qualitative: Brewer Set2
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point(size = 3) +
  scale_colour_brewer(palette = "Set2") +
  theme_minimal()

# Show colour-blind-safe palettes
display.brewer.all(colorblindFriendly = TRUE)
```

6.10.6 Output & Results

Four illustrative figures: a 2D density tile plot using `plasma` for sequential encoding, a centred-at-zero scatter using `gradient2` for signed deviations, a categorical scatter using Brewer `Set2` for cylinder counts, and a panel of all colour-blind-friendly Brewer palettes for reference.

6.10.7 Interpretation

A reporting sentence (figure caption): "Cell colour encodes 2D kernel density on the `viridis::plasma` perceptually uniform palette; positive and negative horsepower deviations from the mean use a red-blue diverging scale centred at zero." Always state the palette by name when it is non-default; readers should not have to guess.

6.10.8 Practical Tips

- Use `viridis` (or its variants `magma`, `plasma`, `inferno`, `cividis`) as the default for continuous scales; all are perceptually uniform and colour-blind safe.
- Limit qualitative palettes to 8 colours or fewer; beyond that, switch to shapes, linetypes, or facets to disambiguate groups.
- Use `scale_colour_gradient2()` (or `scale_fill_gradient2()`) for any signed quantity; the centred palette communicates symmetry that one-sided sequentials cannot.
- For greyscale fallback (photocopying, monochrome printing), `viridis` translates almost losslessly; rainbow hues collapse into similar greys and become unreadable.
- Verify colour-blind accessibility on every figure with `colorBlindness::cvdPlot()`; simulating deuteranopia and protanopia takes seconds and prevents costly resubmissions.
- For heatmaps, perceptual uniformity is non-negotiable — a non-uniform palette creates apparent hotspots wherever the hue cycle bends.

6.10.9 R Packages Used

`viridis` for `viridis`, `magma`, `plasma`, `inferno`, `cividis` continuous scales; `RColorBrewer` for the Brewer family of qualitative, sequential, and diverging palettes; `colorspace` for HCL-based custom palettes; `paletteer` for unified access to dozens of curated palettes from across the R ecosystem.

6.10.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.10.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.11 Contour Plots

6.11.1 Introduction

A contour plot draws isolines — curves along which a scalar field is constant — over a 2D domain. The same geom serves two related purposes: visualising the joint density of paired observations, where the contours are level sets of a kernel-density estimate, and visualising a deterministic function $z = f(x, y)$, where the contours are mathematical isolines. Both uses share a common visual grammar — peaks, ridges, saddles — and contour plots remain one of the most compact ways to communicate where probability mass concentrates or where a function attains specific values.

6.11.2 Prerequisites

A working knowledge of `ggplot2`, basic kernel-density estimation in two dimensions, and the convention that closer contour lines indicate steeper gradients.

6.11.3 Theory

Two distinct workflows feed contour layers in `ggplot2`:

- **Density contours** are computed from raw paired observations via `geom_density_2d()` or `stat_density_2d()`. Internally, a 2D kernel density estimate produces a smoothed surface and `MASS::kde2d()` or `ggplot2`'s built-in estimator returns the level sets.
- **Functional contours** require a complete grid of (x, y, z) values supplied by the user; `geom_contour()` then computes isolines of `z` exactly. The grid resolution determines smoothness; coarse grids produce jagged contours.

Filled variants — `geom_density_2d_filled()` and `geom_contour_filled()` — render closed bands rather than lines and demand a sequential or diverging colour palette so that ordering is preserved.

6.11.4 Assumptions

For density contours, the input is a sample large enough that a kernel density is meaningful (at least a few dozen points; ideally hundreds). For functional contours, the grid covers the domain of interest at adequate resolution.

6.11.5 R Implementation

```r
library(ggplot2); library(metR)

# Contour of 2D density
ggplot(faithful, aes(eruptions, waiting)) +
  geom_point(alpha = 0.3) +
  geom_density_2d(colour = "#F4A261") +
  theme_minimal()

# Filled contour
ggplot(faithful, aes(eruptions, waiting)) +
  geom_density_2d_filled(alpha = 0.7) +
  geom_point(alpha = 0.3, size = 0.5, colour = "white") +
  theme_minimal()

# Function surface: z = sin(x) * cos(y)
grid <- expand.grid(x = seq(-3, 3, 0.1), y = seq(-3, 3, 0.1))
grid$z <- sin(grid$x) * cos(grid$y)
ggplot(grid, aes(x, y, z = z)) +
  geom_contour_filled() +
  theme_minimal()
```

6.11.6 Output & Results

Three figures: density contours of the Old Faithful eruption / waiting joint distribution showing the two well-known modes, the same density rendered as filled bands with the underlying points overlaid, and a filled-contour view of $z = \sin(x)\cos(y)$ on a 61×61 grid showing alternating positive and negative regions.

6.11.7 Interpretation

A reporting sentence (figure caption): “Top: 2D kernel density of eruption duration vs waiting time for Old Faithful (n = 272), with raw observations as background points; the bimodality reflects two distinct eruption regimes. Bottom: deterministic surface $z = \sin(x)\cos(y)$ with filled contours at default levels.” Mention the kernel and bandwidth choice when density contours are used, since they shape the visible structure.

6.11.8 Practical Tips

- For sparse data, density contours can be misleading — the estimator extrapolates into low-mass regions and produces artefactual closed loops. Use `bins = 5` or `bins = 8` for sparse samples; let the default handle dense ones.
- Always overlay raw points (with low alpha) on density contour plots; the points anchor the contours to actual observations and let readers spot extrapolation.
- For functional surfaces, increase grid resolution until the contours are visibly smooth; under-resolved grids produce stair-stepped lines.
- `metR::geom_contour_fill()` and `metR::geom_text_contour()` add labelled contours that read more clearly on filled backgrounds.
- Use sequential palettes (`viridis`, `magma`) for ordered density and diverging palettes (`RdBu`, `BrBG`) for surfaces that cross zero; qualitative palettes destroy the level ordering and are wrong for filled contours.

- For published figures with multiple contour layers, label a few key isolines with their level values so readers can read off heights without consulting a separate legend.

6.11.9 R Packages Used

`ggplot2` for `geom_density_2d()`, `geom_density_2d_filled()`, `geom_contour()`, and `geom_contour_filled()`; `metR` for `geom_contour_fill()` and labelled contours; `MASS::kde2d()` as the underlying kernel density estimator when finer control is needed.

6.11.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.11.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.12 Correlation Heatmaps

6.12.1 Introduction

A correlation heatmap renders a correlation matrix as a grid of colour-encoded cells, with the cell colour mapping the correlation coefficient on a diverging palette centred at zero. It is the natural overview figure for a multivariate dataset: dozens of pairwise correlations that would be impossible to read as a numeric table become a single image where blocks of similarly correlated variables stand out by colour. Hierarchical reordering of rows and columns brings related variables together and reveals cluster structure, often suggesting candidate factors for downstream analysis.

6.12.2 Prerequisites

A working understanding of correlation coefficients (Pearson, Spearman, Kendall), basic familiarity with hierarchical clustering, and `ggplot2` colour scales.

6.12.3 Theory

A correlation heatmap displays r_{ij} in cell (i, j). Standard variants:

- Symmetric with numbers shown — useful when the matrix is small (≤ 8 variables).
- Upper or lower triangle only — avoids redundancy and saves visual space.
- Significance overlay — marks non-significant correlations as blank or with a cross, so the eye is not drawn to noise.

- Hierarchical reordering — uses correlation distance ($1 - |r|$) to cluster related variables and reveal block structure.

The colour scale must be diverging (typically red–white–blue or the Brewer `RdBu`) and centred at zero so positive and negative correlations are visually symmetric. Sequential or qualitative palettes destroy the sign information and are wrong for correlations.

6.12.4 Assumptions

A valid correlation matrix (any kind: Pearson for linear, Spearman for monotone, polychoric for ordinal). The matrix should be positive-semi-definite if it will feed downstream methods; this is automatic when computed from raw data with `cor()`.

6.12.5 R Implementation

```r
library(corrplot); library(ggcorrplot)

corr_mat ← cor(mtcars)

# corrplot
corrplot(corr_mat, method = "color", order = "hclust",
         addCoef.col = "black", tl.col = "black")

# ggcorrplot with p-value overlay
p_mat ← cor_pmat(mtcars)
ggcorrplot(corr_mat, hc.order = TRUE, type = "lower",
           p.mat = p_mat, insig = "blank",
           lab = TRUE, lab_size = 3)
```

6.12.6 Output & Results

Two correlation heatmaps for the 11-variable `mtcars` dataset: a full symmetric heatmap with hierarchical reordering and numeric cell labels, and a lower-triangle version with non-significant cells masked. The first emphasises the full structure; the second is a cleaner publication figure.

6.12.7 Interpretation

A reporting sentence (figure caption): “Pearson correlation matrix for 11 vehicle attributes (n = 32), reordered hierarchically using $1 - |r|$ as the distance; cells with $p > 0.05$ are blanked. The two visible blocks correspond to size-related variables (weight, displacement, horsepower) and efficiency-related variables (mpg, quarter-mile time).” Always state the correlation type and the reordering method.

6.12.8 Practical Tips

- Use a diverging palette centred at zero; the Brewer `RdBu` or `viridis::cividis` (with the centre stretched) work well in print.
- Hierarchical reordering (`hc.order = TRUE` in `ggcorrplot`, `order = "hclust"` in `corrplot`) brings related variables together and is almost always preferable to alphabetical order.

- Mask non-significant cells (`insig = "blank"`) to direct the reader's attention to robust signals; alternatively, mark with crosses to show the test was performed.
- For more than 30 variables, the figure becomes a coloured pixel grid without readable labels; consider summarising via principal components or factor analysis instead.
- State the correlation type (Pearson, Spearman, Kendall) and the test correction (if any) in the caption; readers should not have to guess.
- For partial correlations, use `ppcor::pcor()` and feed the resulting matrix into the same heatmap geom; partial correlations often reveal structure that pairwise correlations obscure.

6.12.9 R Packages Used

`corrplot` for the canonical correlation heatmap with reordering and significance overlays; `ggcorrplot` for a `ggplot2`-native interface; `Hmisc::rcorr()` for correlation matrices with associated p-values; `psych::corPlot()` for an alternative with rich cell formatting.

6.12.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.12.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.13 Density Plots

6.13.1 Introduction

A density plot estimates the underlying probability density of a continuous variable via kernel smoothing, producing a smooth curve instead of the binned bars of a histogram. Densities are easier to compare across groups (overlaid translucent fills are more legible than dodged or transparent bars) and they sidestep the histogram's bin-boundary arbitrariness — at the cost of introducing a different arbitrary parameter, the bandwidth. Choosing a sensible bandwidth and being explicit about it is the main craft of density-plot design.

6.13.2 Prerequisites

A working understanding of histograms, kernel density estimation basics, and the role of bandwidth as the smoothness parameter.

6.13.3 Theory

A kernel density estimate (KDE) is

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h} \right),$$

where K is a kernel (Gaussian by default) and h is the bandwidth. The bandwidth controls smoothness: small h undersmooths and produces a noisy curve that follows individual observations, large h oversmooths and hides modes or skew. R's default uses Silverman's rule of thumb (`bw = "nrd0"`), which works well for unimodal Normal-like data but can oversmooth multi-modal distributions; Sheather–Jones plug-in (`bw = "SJ"`) is a more flexible default that adapts to local variability.

The choice of kernel (K) matters far less than the bandwidth in practice — Gaussian, Epanechnikov, and rectangular kernels produce visually indistinguishable density curves at the same h.

6.13.4 Assumptions

Observations are independent and identically distributed; the underlying distribution has a density (not a discrete spike). For bounded variables (proportions, durations), the kernel density spills past the boundary and a reflection or truncation method is required to keep the estimate inside the support.

6.13.5 R Implementation

```
library(ggplot2)

# Basic density
ggplot(mtcars, aes(mpg)) +
  geom_density(fill = "#2A9D8F", alpha = 0.5) +
  theme_minimal()

# Density by group
ggplot(iris, aes(Sepal.Length, fill = Species)) +
  geom_density(alpha = 0.5) +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()

# Bandwidth control
ggplot(faithful, aes(eruptions)) +
  geom_density(bw = 0.05, colour = "red") +      # undersmoothed
  geom_density(bw = 0.5,  colour = "black") +    # default-ish
  geom_density(bw = 2.0,  colour = "blue") +     # oversmoothed
  theme_minimal()

# Histogram + density overlay
ggplot(mtcars, aes(mpg)) +
  geom_histogram(aes(y = after_stat(density)), bins = 15,
                 fill = "#2A9D8F", alpha = 0.4) +
  geom_density(linewidth = 1) +
  theme_minimal()
```

6.13.6 Output & Results

Four illustrative views: a single-group density, a three-group overlay for the iris dataset, a bandwidth comparison on the bimodal Old Faithful data showing how bandwidth choices alternately reveal or hide the second mode, and a histogram with a density overlay aligned on a common density y-axis.

6.13.7 Interpretation

A reporting sentence (figure caption): “Kernel density estimate of sepal length per species (n = 50 each), Gaussian kernel with Silverman bandwidth; setosa is well-separated from the other two species, which overlap considerably.” Always state the bandwidth method (or value) and the kernel; both shape the visible structure.

6.13.8 Practical Tips

- Show a histogram alongside the density during exploratory work; the density alone can hide binning artefacts that the histogram makes visible.
- For bounded data (proportions on $[0, 1]$, response times bounded below by zero), use `density(..., from = 0, to = 1)` or a reflection-based estimator; default kernels spill past the boundary and produce artificially low density at the edges.
- For small samples ($n < 30$), the density estimate is noisy and misleading; show a strip plot, dot plot, or histogram instead.
- Compare densities across groups via `fill` (with `alpha = 0.4`) for translucent overlays or `colour` for line-only renderings; the latter scales better to many groups.
- For 2D joint densities, `geom_density_2d()` draws contours and `stat_density_2d(aes(fill = after_stat(level)), geom = "polygon")` renders filled level sets.
- When in doubt, plot the same data with three bandwidths (Silverman, Sheather-Jones, and one tighter) and pick whichever respects the structure you already know exists; do not let bandwidth invent features.

6.13.9 R Packages Used

`ggplot2` for `geom_density()` and `stat_density()`; `KernSmooth` for advanced bandwidth selectors; `ks` for multivariate kernel densities; `ggdist` for slabinterval geoms that combine density and credible-interval annotation.

6.13.10 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

6.13.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.14 Dot Plots

6.14.1 Introduction

“Dot plot” refers to two distinct chart types that share a name and a geometry but answer different questions. **Cleveland dot plots** show one value per category as a single point, replacing bar charts for ranked comparisons; they were popularised by William Cleveland’s 1985 *Elements of Graphing Data* on the grounds that perceiving position is more accurate than perceiving length. **Wilkinson dot plots** show one point per observation, stacked at the relevant x-value, and act as a hand-drawn histogram that exposes individual observations rather than binned counts.

6.14.2 Prerequisites

A working knowledge of `ggplot2`, an idea of when to prefer position to length encoding, and basic familiarity with histograms and bar charts.

6.14.3 Theory

Cleveland dot plots (`geom_point()` with category on y and value on x) eliminate the visual noise of bar lengths and make small differences across many categories easy to compare. A faint horizontal segment from zero to the point — `geom_segment()` — preserves the bar-like sense of “distance from origin” without committing the visual heaviness of a filled bar.

Wilkinson dot plots (`geom_dotplot()`) bin observations along one axis and stack equal-sized dots in each bin. The `method = "histodot"` argument anchors bins to the data range; `method = "dotdensity"` adapts bin centres to local density. They are most useful for small samples (n < 100) where every observation should be visible; for larger samples, jittered strip plots or swarm plots scale better.

6.14.4 Assumptions

For Cleveland dot plots: one value per category (or a clearly defined summary). For Wilkinson dot plots: the binwidth makes sense for the data range — too small and dots stack into spikes; too large and they merge into bars.

6.14.5 R Implementation

```r
library(ggplot2)

# Cleveland dot plot: mean mpg by car class
mpg_summary <- aggregate(hwy ~ class, mpg, mean)
ggplot(mpg_summary, aes(hwy, reorder(class, hwy))) +
  geom_segment(aes(x = 0, xend = hwy, yend = class), colour = "grey80") +
  geom_point(size = 3, colour = "#2A9D8F") +
  labs(y = NULL) +
  theme_minimal()

# Wilkinson dot plot: distribution of a small sample
ggplot(mtcars, aes(mpg)) +
  geom_dotplot(method = "histodot", binwidth = 1, fill = "#2A9D8F") +
  theme_minimal()

# Grouped dot plot
ggplot(iris, aes(Species, Sepal.Length)) +
  geom_dotplot(binaxis = "y", stackdir = "center", binwidth = 0.1,
               fill = "#2A9D8F") +
  theme_minimal()
```

6.14.6 Output & Results

Three views: a Cleveland dot plot ranking car classes by mean highway mpg with faint baseline segments, a Wilkinson dot plot of the 32 `mtcars` mpg values stacked as a histogram, and a grouped Wilkinson plot showing sepal-length distributions for three iris species side by side.

6.14.7 Interpretation

A reporting sentence (figure caption): “Cleveland dot plot of mean highway fuel economy (mpg) by class for 234 vehicles; categories are ordered by descending mean. Wilkinson dot plot of mpg values for 32 vehicles, with each dot representing a single car.” When showing summary points, always describe which summary (mean, median, count) the dot represents.

6.14.8 Practical Tips

- Order Cleveland-dot categories by value with `forcats::fct_reorder()` or `reorder()`; alphabetic orderings rarely communicate ranking.
- Combine Cleveland dot plots with horizontal error bars (`geom_errorbarh()` or `geom_segment()` for confidence intervals) when uncertainty is important.
- Choose Wilkinson dot binwidth to match the data resolution; for integer data, set binwidth equal to the unit step.
- For grouped Wilkinson plots, set `stackdir = "center"` for symmetric stacks or `"up"` for one-sided; `binaxis = "y"` rotates the dotplot for vertical y-axis layouts.
- Dot plots in presentation slides work better than bar charts at small sizes; the bar’s length advantage becomes a length-perception headache when the bar is only a few pixels tall.

- Above a few hundred observations per category, switch from Wilkinson dot plots to strip plots, swarm plots (`ggbeeswarm`), or violin plots; stacked dots no longer scale.

6.14.9 R Packages Used

`ggplot2` for `geom_dotplot()`, `geom_point()`, and `geom_segment()`; `forcats` for category reordering; `ggbeeswarm` for swarm-style dot plots that scale to larger samples; `ggdist` for slabinterval geoms that combine the dot-plot and density idioms.

6.14.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.14.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.15 Facets and Panels

6.15.1 Introduction

Small multiples — a grid of identical plots, one per subgroup — are one of the most powerful visualisation idioms in modern data analysis. They let the reader compare a relationship across many conditions without overlaying everything on a single set of axes, which becomes unreadable beyond three or four series. `ggplot2` calls this faceting and provides two complementary functions: `facet_wrap()` for a one-dimensional set of panels arranged into rows and columns, and `facet_grid()` for a strict row-by-column matrix indexed by two grouping variables.

6.15.2 Prerequisites

A working knowledge of `ggplot2`'s grammar of graphics, layered geoms, and the distinction between aesthetic mappings and faceting variables.

6.15.3 Theory

`facet_wrap(~ var)` partitions the data by `var`, creates one panel per level, and arranges them into a single block whose dimensions can be specified with `nrow` or `ncol`. `facet_grid(rows ~ cols)` produces a matrix of panels indexed by two variables, leaving an empty cell for combinations that have no data. Both functions share a `scales` argument controlling axis behaviour:

- `"fixed"` (default): all panels share x and y axes — the right choice when the comparative question is the headline.
- `"free_x"`, `"free_y"`, `"free"`: each panel sets its own axis range — useful when scales differ enough that fixed axes hide structure within panels but at the cost of cross-panel comparability.

Strip labels (the panel headers) are themed via `labeller`; `label_both` combines variable name and level, `label_parsed` interprets the level as a math expression.

6.15.4 Assumptions

No statistical assumptions; faceting is a display convention. The only practical constraint is that the number of panels remains small enough to be readable at the final print size — usually no more than 9–12 panels per page.

6.15.5 R Implementation

```
library(ggplot2); library(palmerpenguins)
data(penguins)

# facet_wrap: one factor
ggplot(penguins, aes(flipper_length_mm, body_mass_g)) +
  geom_point(aes(colour = species), alpha = 0.7) +
  facet_wrap(~ sex) +
  theme_minimal()

# facet_grid: two factors (rows = island, cols = sex)
ggplot(penguins, aes(flipper_length_mm, body_mass_g)) +
  geom_point(aes(colour = species), alpha = 0.7) +
  facet_grid(island ~ sex) +
  theme_minimal()

# Free scales
ggplot(penguins, aes(flipper_length_mm, body_mass_g)) +
  geom_point(aes(colour = species), alpha = 0.7) +
  facet_wrap(~ species, scales = "free") +
  theme_minimal()
```

6.15.6 Output & Results

Three faceted views: a two-panel split by sex, a 3-by-3 grid by island and sex, and a three-panel split by species with free axes per panel. Each version answers a different question — “how does the relationship differ between sexes”, “across both island and sex”, or “what does it look like inside each species ignoring scale”.

6.15.7 Interpretation

A reporting sentence (figure caption): “Body mass against flipper length faceted by sex (`facet_wrap(~ sex)`); colour distinguishes the three penguin species.” Always describe the faceting variable in the caption — readers should not have to infer the grid axis from strip labels alone.

6.15.8 Practical Tips

- Reach for `facet_wrap()` when there is one grouping variable; use `facet_grid()` only when the row/column structure is meaningful and balanced.
- Keep `scales = "fixed"` whenever the cross-panel comparison is the inferential focus; switch to free scales only when within-panel magnitudes differ enough to obscure shape.
- Limit the panel count: 9–12 is the practical upper bound on a journal-sized figure; beyond that, faceting fragments rather than illuminates.
- Use `labeller = labeller(var = c("M" = "Male", "F" = "Female"))` to spell out short factor levels for the reader.
- For aligned axes across `facet_wrap()` panels with very different ranges, fix one axis (`scales = "free_y"`) rather than freeing both — this preserves comparability where it matters.
- `coord_cartesian(xlim = ..., ylim = ...)` zooms while leaving the underlying data unfiltered; combine with `facet_grid()` when you want fixed-scale comparisons but with a tighter view than the natural data range.

6.15.9 R Packages Used

`ggplot2` for `facet_wrap()` and `facet_grid()`; `ggh4x` for nested facets and per-panel scale control; `palmerpenguins` as a small dataset rich in faceting structure for examples.

6.15.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.15.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.16 Forest Plots (Visualisation)

6.16.1 Introduction

The forest plot is the signature visualisation of meta-analysis and of subgroup analyses in clinical trials. Each row shows one study (or one subgroup) as a point estimate and a horizontal confidence interval, with study name and sample size in a margin column and a pooled summary as a diamond at the bottom. The compact layout invites direct visual comparison: heterogeneity reads as horizontal scatter across rows, while consistency reads as alignment near the same vertical position. Done well, a forest plot communicates more about a meta-analysis than the accompanying numerical table.

6.16.2 Prerequisites

A working knowledge of point estimates, confidence intervals, the difference between fixed-effect and random-effects pooling, and basic `ggplot2`.

6.16.3 Theory

A forest plot encodes three quantities per study: the point estimate (a marker, with size optionally proportional to inverse-variance weight), the lower and upper confidence-interval limits (a horizontal segment), and a study label. Conventional additions:

- A vertical reference line at the null value (1 for ratios, 0 for differences).
- A diamond rather than a circle for pooled estimates, with the diamond width equal to the CI.
- Effect-size and CI text in a right-margin column for readers who need numbers as well as positions.

Ratios (HR, OR, RR) belong on a log axis so that the visual width of a CI corresponds to its multiplicative width and so that intervals symmetric on the log scale appear symmetric.

6.16.4 Assumptions

Effect estimates and confidence intervals are available and on a comparable scale across studies; weighting (if any) is computed appropriately for the meta-analytic model.

6.16.5 R Implementation

```r
library(ggplot2)

studies ← data.frame(
  study = c("Trial A", "Trial B", "Trial C", "Trial D", "Pooled"),
  estimate = c(0.85, 0.70, 0.95, 0.88, 0.85),
  lo = c(0.65, 0.50, 0.75, 0.66, 0.75),
  hi = c(1.10, 0.98, 1.20, 1.16, 0.96),
  is_pooled = c(FALSE, FALSE, FALSE, FALSE, TRUE)
)

ggplot(studies, aes(x = estimate, y = factor(study, levels = rev(study)))) +
  geom_vline(xintercept = 1, linetype = "dashed", colour = "grey60") +
  geom_errorbarh(aes(xmin = lo, xmax = hi), height = 0.2) +
  geom_point(aes(size = is_pooled, shape = is_pooled), colour = "#2A9D8F") +
  scale_size_manual(values = c(3, 5), guide = "none") +
  scale_shape_manual(values = c(16, 18), guide = "none") +
  scale_x_log10() +
  labs(x = "Hazard ratio (log scale)", y = NULL) +
  theme_minimal()
```

6.16.6 Output & Results

A five-row forest plot: four trials displayed as circles with horizontal error bars, a pooled summary as a larger diamond at the bottom, and a vertical null line at HR = 1 on a log-x axis. The dispersion of the four trial estimates around the pooled diamond communicates between-study heterogeneity at a glance.

6.16.7 Interpretation

A reporting sentence (figure caption): “Forest plot of hazard ratios from four trials and the random-effects pooled estimate (REML, $I^2 = 23\%$); horizontal error bars show 95 % confidence intervals on a log scale, the diamond marks the pooled estimate, and the dashed vertical line corresponds to no effect (HR = 1).” Always state the pooling model and a heterogeneity statistic.

6.16.8 Practical Tips

- Log-transform ratios on the x-axis so confidence-interval widths correspond to multiplicative width; otherwise small effects look disproportionately large.
- Make marker size proportional to inverse-variance weight in meta-analysis (`size = 1/SE^2`); this is the convention readers expect.
- Use a diamond for pooled estimates with width equal to the CI; `geom_polygon()` produces these cleanly.
- Order studies meaningfully — chronologically, by size, by point estimate, or by subgroup — and disclose the ordering choice in the caption.
- Add a right-margin column with numerical effect estimates and CIs (`geom_text()` at fixed x, mapped y); readers expect both visual and numeric information.
- For automated meta-analysis output, `metafor::forest.rma()` and `forestplot` produce publication-ready figures with weighted markers, columns, and heterogeneity annotations without hand-coding the `ggplot`.

6.16.9 R Packages Used

`ggplot2` for hand-built forest plots; `forestplot` for two-column publication-style forests; `metafor::forest.rma()` for meta-analysis output forests; `meta::forest()` for random-effects forests with prediction intervals; `ggforestplot` for tidy hazard-ratio displays from regression models.

6.16.10 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

6.16.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.17 Funnel Plots (Visualisation)

6.17.1 Introduction

A funnel plot is the canonical visualisation for diagnosing publication bias in meta-analysis. It plots each study's effect size against its precision (or its standard error), forming an inverted-funnel shape under the null of no bias: large precise studies cluster near the pooled estimate at the narrow top of the funnel, smaller imprecise studies scatter wider at the base. Asymmetry of the funnel — typically a missing wedge of small null or negative studies — suggests that studies have been suppressed for non-statistical reasons, often because the field's preferred direction is positive and significant.

6.17.2 Prerequisites

A working understanding of meta-analysis, the relationship between sample size and standard error, and the difference between heterogeneity and publication bias as competing explanations for asymmetry.

6.17.3 Theory

A funnel plot has axes:

- **x**: effect size (log odds ratio, log hazard ratio, standardised mean difference, etc.).
- **y**: standard error (with smaller SEs at the top, so the funnel narrows upward) or precision $1/\mathrm{SE}$.

Under the null of no publication bias and no small-study effects, studies fall symmetrically inside a triangular pseudo-confidence funnel centred on the pooled estimate. Asymmetry — most commonly an absence of small studies on one side — is the visual cue for bias. Formal tests quantify it: Egger's regression of the standardised effect on its precision is the most widely used; Begg–Mazumdar's rank correlation is more robust to sparse data. Trim-and-fill (`metafor::trimfill()`) imputes missing studies to estimate a bias-adjusted effect.

6.17.4 Assumptions

A meta-analysis with at least 10 studies (fewer makes asymmetry impossible to judge), study-level effects and standard errors on a comparable scale, and the working assumption that no other sources (e.g. true heterogeneity correlated with study size) explain the asymmetry.

6.17.5 R Implementation

```r
library(metafor)

# Meta-analysis of 10 hypothetical studies
dat ← data.frame(
  yi = c(-0.1, -0.3, -0.5, -0.4, -0.2, -0.6, -0.7, -0.3, -0.9, -1.2),
  vi = c(0.04, 0.03, 0.08, 0.05, 0.02, 0.09, 0.07, 0.06, 0.10, 0.12)
)
res ← rma(yi = yi, vi = vi, data = dat, method = "REML")
funnel(res, main = "Funnel plot of 10 studies")

# Egger test
regtest(res, model = "lm")
```

6.17.6 Output & Results

A funnel plot with each study marked at its (y_i, SE_i) position, a vertical line at the pooled estimate, and pseudo-95 % confidence funnel boundaries that widen as standard error increases. `regtest()` returns Egger's test slope and p-value; a significant slope is conventional evidence of asymmetry.

6.17.7 Interpretation

A reporting sentence: "The funnel plot of 10 studies showed visible asymmetry (Egger's regression slope -2.1, $p = 0.04$); trim-and-fill imputed three additional studies on the right and shifted the pooled estimate from -0.55 (95 % CI -0.78 to -0.32) to -0.42 (95 % CI -0.66 to -0.17), suggesting that publication bias inflates the apparent effect by roughly 25 %." Always pair the visual with a formal test; a single test rarely provides strong evidence on its own.

6.17.8 Practical Tips

- Require at least 10 studies before drawing conclusions from a funnel plot; fewer studies cannot support a meaningful asymmetry test.
- True between-study heterogeneity correlated with study size (smaller studies in different populations) can mimic publication-bias asymmetry; consider both explanations.
- `metafor::trimfill()` adjusts for detected asymmetry but is not a cure — it is one of several sensitivity analyses to report.
- Contour-enhanced funnel plots (`funnel(res, level = c(90, 95, 99))`) overlay significance contours, helping to distinguish bias from heterogeneity.
- For binary outcomes, plot on the log-OR or log-RR scale; raw OR axes distort the symmetry under the null.
- Custom `ggplot2` funnel plots are easy to build (`geom_point()` plus `geom_segment()` for the funnel boundaries) when journal style requires consistent typography across figures.

6.17.9 R Packages Used

`metafor` for `rma()`, `funnel()`, `regtest()`, and `trimfill()`; `meta` for `metabias()` and alternative funnel implementations; `ggplot2` for hand-built funnel plots when the default base-graphics output does not match a publication's style; `dmetar` for additional bias diagnostics.

6.17.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.17.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.18 ggplot2 Themes

6.18.1 Introduction

A `ggplot2` theme controls everything in a plot that is not the data itself — panel background, axis ticks, gridlines, font sizes, legend placement, plot margins. Two layers of control coexist: a **complete theme** (`theme_minimal()`, `theme_bw()`, `theme_classic()`, etc.) sets every non-data element at once and is usually the right starting point, while individual `theme()` calls override specific elements for publication polish. Mastering both layers is what turns a default plot into one that looks at home in a journal article or a polished report.

6.18.2 Prerequisites

A working knowledge of `ggplot2`'s grammar of graphics, layered geoms, and aesthetic mappings.

6.18.3 Theory

Complete themes are functions that return a fully populated `theme` object. Built-in choices include `theme_minimal()` (white background, light gridlines, the de facto modern default), `theme_bw()` (white panel with thin black border), `theme_classic()` (axes only, no gridlines, mimicking print journals), and `theme_void()` (no axes or gridlines, useful for maps and decorative composites). The `ggthemes` package adds journal-flavoured options (`theme_tufte()`, `theme_economist()`, `theme_few()`).

Element overrides come from `theme()`, which accepts named arguments for every theme element. Each element is built from one of four constructors: `element_text()` for text (size, family, face, colour, hjust, vjust, margin), `element_line()` for lines, `element_rect()` for rectangles, and `element_blank()` to hide an element entirely. Common overrides include `legend.position`, `axis.title`, `panel.grid.minor`, and `plot.title`.

`theme_set()` applies a theme globally to every subsequent plot in the session, useful for enforcing house style across a multi-figure report.

6.18.4 Assumptions

No statistical assumptions; the only practical constraint is that font sizes, line widths, and margins must remain legible at the final print size, which is best verified by saving the plot at the target dimensions before judging it on screen.

6.18.5 R Implementation

```
library(ggplot2); library(ggthemes)

p ← ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point(size = 3)

# Complete themes
p + theme_minimal()
p + theme_bw()
p + theme_classic()
p + theme_tufte()

# Custom theme
p +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 12),
    legend.position = "bottom",
    panel.grid.minor = element_blank()
  ) +
  labs(title = "MPG vs weight", x = "Weight (1000 lbs)", y = "MPG",
       colour = "Cylinders")

# Set a global theme
theme_set(theme_minimal(base_size = 12))
```

6.18.6 Output & Results

Side-by-side comparisons of the same scatterplot under four complete themes plus a fully customised version. The custom variant combines `theme_minimal()` as the base with overrides for title weight, axis title size, legend position, and gridline visibility — a typical recipe for a publication-ready figure.

6.18.7 Interpretation

A reporting sentence (figure caption): "All figures use `theme_minimal(base_size = 12)` with custom title and axis label sizes; legend position was moved to the bottom for horizontal compositions." Mention the theme only when it affects interpretation (e.g. removing gridlines deliberately to emphasise a contrast); otherwise it can be left to a methods section "figure preparation" line.

6.18.8 Practical Tips

- Set a global theme once at the top of your analysis script via `theme_set()`; the rest of the figures inherit it automatically and stay consistent.
- For biomedical publications, `theme_minimal(base_size = 12)` with `panel.grid.minor = element_blank()` is a sensible default; it reads well in print and on screen.
- Use `element_blank()` to hide an element entirely; setting `colour = NA` only makes it invisible while still occupying its space.
- Keep font choices simple: `theme(text = element_text(family = "Helvetica"))` (or the corresponding sans-serif default) avoids the rendering surprises that come with fancier typefaces.
- Always export with `ggsave()` at the final intended `width`, `height`, and `dpi` (300 for raster, 600+ for line art); judging a plot on a screen at the wrong dimensions hides legibility problems.
- For bespoke journal styles, build a wrapper function that returns a fully configured theme and use it instead of repeating the same `theme()` overrides in every script.

6.18.9 R Packages Used

`ggplot2` for the base theme system, `ggthemes` for journal- and Tufte-flavoured presets, `cowplot` for `theme_cowplot()` (a publication-oriented variant), and `extrafont` if you need access to non-default system fonts in PDF output.

6.18.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.18.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.19 Saving and Exporting Figures

6.19.1 Introduction

The final step of any figure pipeline is exporting the plot at the right dimensions and resolution. It is also the step where most figures go wrong: a plot designed and previewed at 1024 × 768 pixels on screen looks fine until it is dropped into a manuscript and reprinted at journal column width, where the fonts have shrunk to illegibility and the raster edges have blurred. `ggsave()` is `ggplot2`'s export function, and the discipline to invoke it with explicit dimensions and an explicit DPI early in the workflow is what separates publication-ready figures from ones that have to be re-made the night before submission.

6.19.2 Prerequisites

A working knowledge of `ggplot2`, an idea of what the final figure will be used for (print, web, slide), and basic familiarity with vector vs raster graphics.

6.19.3 Theory

File-format choice depends on the destination:

- **PDF** is vector and infinite-resolution; it is the standard for journal submissions and the safest default unless told otherwise.
- **SVG** is also vector and ideal when the figure will be edited in Illustrator or Inkscape before final use.
- **PNG** is raster with lossless compression; use 300 dpi or higher for print, 96 dpi for web.
- **TIFF** is raster and required by several biomedical journals; use lossless `compression = "lzw"` to avoid quality loss.
- **EPS** is vector and required by a shrinking minority of journals; PDF is generally a better choice when allowed.

Dimensions are specified through `width` and `height` with units (`"in"`, `"cm"`, `"mm"`); DPI applies only to raster formats. Font sizes scale with `base_size` in the theme; setting `base_size = 12` against a 6 × 4 inch figure produces fonts that print legibly at journal column width.

6.19.4 Assumptions

The graphics device for the chosen format is available (Cairo for PNG with anti-aliasing, `ragg` for high-quality raster output, `svglite` for compact SVG); modern R installations include all of these but verify if a specific format fails.

6.19.5 R Implementation

```r
library(ggplot2)

p ← ggplot(mtcars, aes(wt, mpg)) +
  geom_point(size = 3) +
  theme_minimal(base_size = 12)

# Standard PDF for journal submission
ggsave("figure1.pdf", p, width = 6, height = 4, units = "in")

# High-resolution PNG
ggsave("figure1.png", p, width = 6, height = 4, units = "in", dpi = 300)

# Journal-standard column width (typically 85 mm for single column)
ggsave("figure1.tiff", p, width = 85, height = 60, units = "mm",
       dpi = 300, compression = "lzw")

# Vector SVG for editing
ggsave("figure1.svg", p, width = 6, height = 4)
```

6.19.6 Output & Results

Four export variants of the same `ggplot` object: a PDF for journal submission, a 300-dpi PNG for web embedding, an LZW-compressed TIFF at 85 mm single-column width, and an SVG for downstream editing. Each file is identical in content but differs in format-specific properties.

6.19.7 Interpretation

A reporting sentence (methods or figure-preparation note): “All figures were rendered with `ggplot2` 3.5 (`base_size = 12`) and exported to PDF (vector) and TIFF (LZW-compressed, 300 dpi) at single-column width (85 mm); raster previews use the `ragg` device for anti-aliased text.” Documenting the export pipeline aids reproducibility and pre-empts reviewer questions about figure resolution.

6.19.8 Practical Tips

- Always export at the final publication size; resizing a saved figure in a word processor distorts fonts and produces blurry raster edges.
- For multi-panel figures composed with `patchwork`, export the composed object directly; do not save panels separately and stitch them in an editor.
- Use `ragg::agg_png()` (or set `options(ragg.background = "white")`) for anti-aliased PNG output that handles UTF-8 and emoji correctly.
- For TIFF submissions, `compression = "lzw"` produces lossless files at typically half the size of uncompressed; some journals also accept `"zip"`.
- Save a small reproducible script next to every published figure (`figure1.R`); regenerating a figure six months later is otherwise an exercise in archaeology.
- For vector figures with thousands of overlapping elements (large hexbins, dense scatter), file sizes balloon; rasterise the busy layer with `ggrastr::rasterise()` while keeping axes and text vector for a hybrid output that prints cleanly.

6.19.9 R Packages Used

`ggplot2` for `ggsave()`; `ragg` for fast, high-quality raster devices; `svglite` for compact SVG; `Cairo` for legacy anti-aliased rendering; `ggrastr` for selectively rasterising vector layers in otherwise vector exports.

6.19.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.19.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.20 The Grammar of Graphics

6.20.1 Introduction

`ggplot2` is the dominant plotting system in R because it is built on a principled grammar rather than an ad-hoc collection of plot types. Every plot is constructed from the same seven building blocks: data, aesthetic mappings, geometric objects (geoms), statistical transformations, scales, coordinate systems, and faceting. Once the grammar is internalised, the vast library of plot types reduces to a small set of decisions in a common vocabulary.

6.20.2 Prerequisites

The reader should know R vectors, data frames, and the `%>%` or `▷` pipe. No prior `ggplot2` knowledge is assumed.

6.20.3 Theory

The layered grammar expresses every plot as a stack of layers added to a base specification. Each layer may override or inherit data and mappings from the base. The seven components:

- **Data**: a data frame whose rows are observations and whose columns are variables.
- **Mappings**: assignments from variables in the data to visual aesthetics (x-position, y-position, colour, size, shape, alpha, etc.).
- **Geoms**: the geometric objects drawn on the plot – points, lines, bars, polygons, text.
- **Stats**: statistical transformations applied to the data before drawing (identity, binning for histograms, smoothing for trend lines, summary for means with error bars).

- **Scales**: the functions that translate data values into aesthetic values – a continuous scale for a numeric x-axis, a discrete ordinal scale for a factor colour mapping.
- **Coord**: the coordinate system, usually Cartesian but occasionally polar, map projections, or fixed aspect ratios.
- **Facets**: subplots defined by one or more categorical variables, laid out in rows, columns, or a grid.

A plot is rendered by evaluating the stats on the data, mapping the results through the scales to aesthetic values, drawing the geoms in the chosen coordinate system, and replicating the whole thing across facets.

6.20.4 R Implementation

The `palmerpenguins` package provides a tidy biological dataset for demonstration:

```
library(ggplot2)
library(palmerpenguins)

ggplot(data = penguins,
       mapping = aes(x = flipper_length_mm,
                     y = body_mass_g,
                     colour = species)) +
  geom_point(alpha = 0.7, size = 2) +
  geom_smooth(method = "lm", se = TRUE) +
  scale_colour_brewer(palette = "Set2") +
  facet_wrap(~ sex, nrow = 1) +
  labs(x = "Flipper length (mm)",
       y = "Body mass (g)",
       colour = "Species",
       title = "Body mass vs. flipper length by species and sex") +
  theme_minimal(base_size = 12) +
  theme(legend.position = "bottom")
```

This single plot uses all seven grammar components: the data is `penguins`, the mappings send `flipper_length_mm` to x, `body_mass_g` to y, and `species` to colour; two geoms (point and smooth) are layered; the `lm` stat fits a linear trend within each species; the Brewer scale converts the species factor to colours; the Cartesian coordinate system is implicit; faceting by sex produces a panel per level.

6.20.5 Output & Results

The plot shows three colour-coded point clouds in each of two panels (one per sex). Linear trends are drawn within each species, and the Brewer “Set2” palette is colour-blind-safe. Axis labels and a title provide context; the legend is placed at the bottom to give maximum plot area to the data.

6.20.6 Interpretation

The layered-grammar view turns plot design into a checklist:

1. What data?
2. Which variables go to which aesthetics?

3. What geoms best display the relationship?
4. Are any statistical transformations needed?
5. Which scales (colour palette, axis transformation) best carry the intended comparison?
6. Coordinate system – usually Cartesian, but check.
7. Faceting – does the story separate across a grouping variable?

Going through these questions in order produces plots that are intentional rather than stumbled into.

6.20.7 Practical Tips

- Start with `geom_point()` and add layers; do not try to build the final plot in one shot.
- Map aesthetics inside `aes()` when they depend on data; set them outside `aes()` (e.g. `colour = "steelblue"`) when they are constant.
- Use colour-blind-safe palettes by default: `scale_colour_viridis_d()`, `scale_colour_brewer()`, or the `colorblindr` package.
- Export figures with `ggsave(filename, width = 7, height = 5, dpi = 300)` to get publication-grade output. Most journals accept PDF or TIFF at 300+ DPI.
- Avoid dual-y-axis plots unless the two scales are genuinely comparable; they are almost always misleading.

6.20.8 For Reviewers

Use the shared reviewer checklist, together with this method's theory and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.20.9 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.21 Heatmaps

6.21.1 Introduction

A heatmap displays a matrix as a grid of coloured cells, with numerical values encoded by colour. The geom is a workhorse across genomics (gene-expression matrices), clinical research (cross-tabulations of conditions and measurements), classifier evaluation (confusion matrices), and exploratory data analysis (correlation matrices). Two design choices dominate readability: the colour palette (sequential, diverging, or qualitative) and the row / column order, which often comes from hierarchical clustering on rows, columns, or both.

6.21.2 Prerequisites

A working understanding of matrices, hierarchical clustering, and colour palette selection (sequential vs diverging).

6.21.3 Theory

Two ecosystems handle heatmaps in R:

- **`ggplot2` + `geom_tile()`**: flexible but requires long-format data and manual handling of clustering. Best when the heatmap is one panel in a larger composition.
- **`pheatmap` and `ComplexHeatmap`**: dedicated packages that handle clustering, dendrograms, row/column annotations, and z-scoring natively. Best when the heatmap is the figure and customisation focus is on annotation tracks.

The colour scale must match the data semantics: sequential (`viridis`, `Blues`) for ordered values without a meaningful zero, diverging (`RdBu`, `BrBG`) for z-scored data or any quantity centred on zero. Heatmaps with a one-sided palette applied to centred data are misleading because the visual midpoint does not match the numerical midpoint.

6.21.4 Assumptions

A matrix or matrix-like structure with rows and columns that admit a sensible reordering. For clustering, rows and columns must be on comparable scales (or pre-scaled per row).

6.21.5 R Implementation

```r
library(ggplot2)

# ggplot tile heatmap
d ← as.data.frame(scale(as.matrix(mtcars)))
d$car ← rownames(d)
d_long ← reshape2::melt(d, id.vars = "car")
ggplot(d_long, aes(variable, car, fill = value)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", mid = "white", high = "red",
                       midpoint = 0) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# pheatmap with hierarchical clustering
library(pheatmap)
pheatmap(scale(mtcars), clustering_method = "ward.D2",
         color = colorRampPalette(c("blue", "white", "red"))(100))
```

6.21.6 Output & Results

Two heatmaps of the z-scored `mtcars` matrix: a hand-built `ggplot2` tile heatmap (no clustering, alphabetical row order) and a `pheatmap` version with Ward's hierarchical clustering on both rows and columns plus accompanying dendrograms. The clustered version reveals two clear groups of cars distinguished by their efficiency-versus-power profile; the unclustered version requires the reader to find that structure unaided.

6.21.7 Interpretation

A reporting sentence (figure caption): "Heatmap of z-scored vehicle attributes (n = 32 cars × 11 variables); colour encodes z-score on a blue-white-red diverging scale centred at zero, and rows and columns are clustered with Ward's method on Euclidean distance." Always state the scaling, the distance metric, and the linkage method; results depend on all three.

6.21.8 Practical Tips

- Centre and scale rows (z-score per row) before clustering whenever variables are on different scales; otherwise, large-magnitude variables dominate the distance metric.
- Use a diverging palette with an explicit midpoint for z-scored data; one-sided sequential palettes collapse the visual midpoint and obscure structure.
- For large matrices (thousands of rows or columns), `ComplexHeatmap` scales better than `pheatmap` and supports complex annotation tracks (categorical, continuous, mark-driven).
- Add side annotations (group, condition, batch) to help readers spot whether clustering tracks biology or technical artefacts.
- Cluster only the dimensions that should be ordered; force the other dimension's order when biological or temporal meaning matters.
- Use `cluster_rows = FALSE` (or `cluster_cols = FALSE`) when an existing ordering carries information you want preserved (e.g. chromosome position).

6.21.9 R Packages Used

`ggplot2` for `geom_tile()` and flexible composition; `pheatmap` for one-call heatmaps with clustering; `ComplexHeatmap` for large-scale heatmaps with rich annotations; `superheat` for an alternative ggplot-flavoured heatmap with linked summary plots; `viridis` and `RColorBrewer` for palette selection.

6.21.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.21.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.22 Hexbin Plots

6.22.1 Introduction

A scatterplot of a few hundred points is informative; a scatterplot of a few hundred thousand points is mostly black ink. Once n exceeds a few thousand, individual points overplot to the point of hiding density structure: ridges, modes, and non-linear boundaries vanish under the cloud. Hexbin plots solve this by dividing the plane into hexagonal cells, counting points per cell, and colour-encoding the count. The result preserves the joint distribution at any sample size and renders quickly even for millions of observations.

6.22.2 Prerequisites

A working knowledge of scatterplots, basic 2D density estimation, and `ggplot2`'s aesthetic mappings.

6.22.3 Theory

Hexagonal bins tile the plane more efficiently than squares because every hexagon has six equidistant neighbours; this isotropy reduces the visual aliasing that affects rectangular bins along diagonal density boundaries. `ggplot2::geom_hex()` calls the `hexbin` package internally to construct the bins and aggregate counts. The `bins` argument sets the resolution along each axis; `binwidth` is an alternative that fixes the cell size in data units. The colour scale should respect the count distribution: linear scales for roughly uniform counts, logarithmic scales (`scale_fill_viridis_c(trans = "log")`) for highly skewed counts that span orders of magnitude.

6.22.4 Assumptions

Both x and y are continuous; categorical variables need a different geom (heatmap, mosaic, or jitter). The plot assumes the joint distribution is the inferential target, not individual points — for one-off outliers, overlay a labelled `geom_point()`.

6.22.5 R Implementation

```
library(ggplot2); library(hexbin)

# Diamonds dataset: 54 000 observations
ggplot(diamonds, aes(carat, price)) +
  geom_hex(bins = 40) +
  scale_fill_viridis_c() +
  theme_minimal()

# Log-scale for skewed data
ggplot(diamonds, aes(carat, price)) +
  geom_hex(bins = 40) +
  scale_x_log10() +
  scale_y_log10() +
  scale_fill_viridis_c(trans = "log") +
  theme_minimal()
```

6.22.6 Output & Results

A 40-by-40 hexbin grid showing the joint distribution of carat and price across 54,000 diamonds. The linear-scale version emphasises the dense middle of the distribution; the log-log version with log-scaled fill reveals the sparse tails — small low-priced diamonds and large high-priced ones — that are invisible at the original scale.

6.22.7 Interpretation

A reporting sentence (figure caption): "Hexbin density of diamond carat versus price (n = 53,940), 40 bins per axis, fill on a log-count scale; both axes are log-transformed to expose the price-vs-carat relationship across the full range." Always state the bin count and any transformations, since both shape the visual conclusion.

6.22.8 Practical Tips

- `bins = 30` to `bins = 60` is a sensible default for tens of thousands of observations; finer grids reveal more detail at the cost of noisier counts per cell.
- Log-transform the fill scale when counts span orders of magnitude; otherwise the dense centre saturates the palette and the tails vanish.
- For square bins, use `geom_bin2d()`; the choice between hex and square is largely aesthetic, but hex bins reduce diagonal-aliasing artefacts.
- Overlay a smoother (`geom_smooth(method = "gam")`) on top of the hexbin for a model-based summary; the bin colour shows where the smoother is well-supported.
- For extremely large n (millions of points), hexbin remains fast and effective where individual scatter would crash the renderer; the underlying counts are stored as a small array regardless of input size.
- Below a few thousand points, prefer alpha-blended scatter or 2D density contours; hexbin is overkill for small samples and reads as cluttered.

6.22.9 R Packages Used

`ggplot2` for `geom_hex()` and `geom_bin2d()`, `hexbin` for the underlying hexagonal binning, `viridis` for perceptually uniform fill scales, and `ggdensity` if you want to combine binning with model-based density contours.

6.22.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.22.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.23 Histograms

6.23.1 Introduction

A histogram is the simplest distributional display: bin the range of a continuous variable, count the observations in each bin, and draw bars proportional to those counts. It is the canonical first plot of any exploratory data analysis. It is also the geom most often misled by its own settings — the impression of "skewness" or "bimodality" can be conjured or hidden entirely by the choice of bin width, so a good histogram is one inspected at several resolutions before being committed to print.

6.23.2 Prerequisites

A working knowledge of continuous variables, the difference between counts and densities, and `ggplot2` aesthetic mappings.

6.23.3 Theory

Three choices shape every histogram:

- **Bin count** (or `binwidth`). Too few bins hide modes and skew; too many exaggerate sampling noise. Common rules of thumb: Sturges ($\log_2 n + 1$, optimal for Normal data of moderate size), Freedman–Diaconis (binwidth $= 2 \cdot \mathrm{IQR} / n^{1/3}$, robust to outliers), Scott ($3.5\sigma / n^{1/3}$, optimised for Normal data).
- **Frequency vs density y-axis**. Frequency (raw count) is the default; density (count divided by binwidth, integrating to 1) is essential for comparing groups of different sample sizes or overlaying with a density curve.
- **Bin boundaries**. Where the first bin starts affects the appearance, especially for small samples; this is the so-called "boundary effect", invisible in dense data and obvious in sparse.

The shape impression depends on all three; a histogram should always be reproduced with at least two bin counts before any conclusion is drawn.

6.23.4 Assumptions

A continuous (or discrete-but-numerous) variable. Categorical data should go to bar charts; small ordinal scales (Likert) often look better as bar charts even if technically histograms apply.

6.23.5 R Implementation

```r
library(ggplot2)

# Basic histogram
ggplot(mtcars, aes(mpg)) +
  geom_histogram(bins = 15, fill = "#2A9D8F", colour = "white") +
  theme_minimal()

# Density y-axis for group comparison
ggplot(iris, aes(Sepal.Length, fill = Species)) +
  geom_histogram(aes(y = after_stat(density)),
                 position = "identity", alpha = 0.5, bins = 20) +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()

# Automatic binwidth via Freedman-Diaconis
ggplot(diamonds, aes(price)) +
  geom_histogram(binwidth = 2 * IQR(diamonds$price) / nrow(diamonds)^(1/3),
                 fill = "#2A9D8F", colour = "white") +
  theme_minimal()
```

6.23.6 Output & Results

Three histograms: the 32-row `mtcars` mileage with 15 bins, a three-species iris overlay on a density y-axis (so the groups are comparable despite identical sample sizes), and a Freedman–Diaconis-binwidth view of diamond prices that handles the long right tail more gracefully than a default 30-bin choice.

6.23.7 Interpretation

A reporting sentence (figure caption): “Distribution of fuel efficiency (mpg) across 32 cars, 15 equal-width bins covering the range 10–35 mpg.” Always state the bin count or binwidth in the caption; a histogram without that information is impossible to reproduce.

6.23.8 Practical Tips

- Always inspect with several bin counts before committing to one; modes that survive a doubling of bins are real, modes that disappear were artefacts.
- Use `after_stat(density)` (formerly `..density..`) when overlaying multiple groups of different sizes; raw counts conflate distribution shape with sample-size differences.
- For long right tails (waiting times, prices, gene expression), log-transform the x-axis (`scale_x_log10()`) before binning; the resulting histogram reveals structure invisible on the linear scale.
- For small samples ($n < 30$) prefer a dot plot or strip plot; a histogram of 30 points binned into 10 bars is mostly empty.
- Overlay `geom_density()` (with `aes(y = after_stat(density))`) for a smooth summary that complements the bins; the two together communicate both the discrete bins and the smooth shape.
- Avoid mixing colour (group) with fill aesthetics when the same column drives both; pick one and let the legend speak for itself.

6.23.9 R Packages Used

`ggplot2` for `geom_histogram()`; `scales` for x-axis label formatters when the data are on currency or large-number scales; `ggdensity` for combined histogram-and-density displays; `ggdist` for modern slabinterval geoms that often replace histograms entirely in posterior visualisations.

6.23.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.23.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.24 Interactive Plots with ggiraph

6.24.1 Introduction

Where `plotly` translates `ggplot2` figures into a separate JavaScript engine, `ggiraph` keeps the original `ggplot2` rendering and adds SVG event handlers to individual elements — points, bars, polygons. The result is interactivity that preserves `ggplot2`'s native typography, themes, and palettes, with sharp vector output that scales cleanly to print and screen alike. Tooltips, click events, hover highlighting, and Shiny linkage come from a small set of `*_interactive` geoms that mirror their `ggplot2` counterparts.

6.24.2 Prerequisites

A working knowledge of `ggplot2` and an output environment that supports HTML — Quarto, RMarkdown, Shiny, or any browser-based viewer.

6.24.3 Theory

Each interactive geom (`geom_point_interactive()`, `geom_col_interactive()`, `geom_polygon_interactive()`, etc.) accepts additional aesthetics on top of its `ggplot2` base:

- `tooltip`: text (or HTML) shown on hover.
- `data_id`: identifier used for cross-element highlighting and selection.
- `onclick`: arbitrary JavaScript invoked when the element is clicked.

The `girafe()` function wraps the resulting plot into an HTML widget; `opts_*` helpers tune hover CSS, selection behaviour, zoom, toolbar visibility, and download options. Because the output is plain SVG, every interactive plot remains a standalone vector graphic that can be exported, embedded, or re-styled with external CSS.

6.24.4 Assumptions

The output target supports HTML and SVG rendering. For PDF or PNG output, `ggiraph` falls back to a static SVG snapshot — interactivity is lost but typography is preserved, unlike `plotly`'s rasterised exports.

6.24.5 R Implementation

```
library(ggplot2); library(ggiraph)

p ← ggplot(mtcars, aes(wt, mpg,
                       tooltip = rownames(mtcars),
                       data_id = rownames(mtcars))) +
  geom_point_interactive(size = 3, colour = "#2A9D8F") +
  theme_minimal()

girafe(ggobj = p,
       options = list(opts_hover(css = "fill:#F4A261;stroke:black"),
                      opts_selection(type = "multiple")))
```

6.24.6 Output & Results

An interactive SVG scatterplot in which hovering a point highlights it with the configured CSS rule, clicking selects it, and shift-clicking adds to the selection. Inside Shiny, the selection propagates to the `input$<plot_id>_selected` reactive, enabling brushed cross-filtering between charts and tables.

6.24.7 Interpretation

A reporting sentence: "Figure 2 (rendered with `ggiraph`): each point shows a vehicle; hover reveals the model name and selected points propagate to the linked summary table below." Mention the technology and any selection semantics in the caption — readers should know whether interactivity is decorative or part of the analytical workflow.

6.24.8 Practical Tips

- Each `geom_*_interactive()` accepts the same arguments as its non-interactive parent plus the new aesthetics, so refactoring an existing plot is a one-word swap per layer.
- Use `data_id` to group elements that should highlight together (all points from one country, all bars in one facet); the SVG event system propagates by `data_id` automatically.
- Inside Shiny, swap `plotOutput()` and `renderPlot()` for `girafeOutput()` and `renderGirafe()`; selection and hover events become reactive inputs without additional plumbing.

- Customise hover and selection CSS with `opts_hover(css = ...)` and `opts_selection(css = ...)`; the same plot can present very different interactive feedback depending on context.
- Avoid `ggiraph` for plots with many tens of thousands of elements; SVG renders every element as a DOM node, and browsers slow down beyond a few thousand. For very large data, hexbin or canvas-rendered alternatives (`scattermore`, `plotly` with `scattergl`) scale better.
- For static publication, the same `ggplot` object renders without modification through `ggsave()` — write your interactive code over a normal `ggplot` and you get both versions for free.

6.24.9 R Packages Used

`ggiraph` for the interactive geoms, `girafe()`, and the `opts_*` helpers; `ggplot2` for the underlying grammar; `htmlwidgets` for embedding standalone widgets; `shiny` when reactive selection events drive downstream computation.

6.24.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.24.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.25 Interactive Plots with plotly

6.25.1 Introduction

A static figure is what a journal prints; an interactive figure is what a colleague needs when exploring a result on a screen. `plotly::ggplotly()` converts an existing `ggplot` object into an HTML widget that supports hover tooltips, drag-to-zoom, panning, and selection brushing — without requiring any JavaScript. The result fits naturally into Quarto and RMarkdown reports, Shiny dashboards, and standalone HTML pages, and gives reviewers and collaborators a way to interrogate the data without exporting CSVs.

6.25.2 Prerequisites

A working knowledge of `ggplot2` (geoms, aesthetics, themes) and an output context that supports HTML rendering — Quarto, RMarkdown, Shiny, or RStudio's Viewer pane.

6.25.3 Theory

`ggplotly()` walks the structure of a `ggplot` object — its layers, scales, and aesthetic mappings — and translates each component into the corresponding `plotly.js` primitives. Hover tooltips default to whatever aesthetic mappings are in scope; custom tooltips come from mapping `aes(text = ...)` and passing `tooltip = "text"` to `ggplotly()`. The translation is faithful for most common geoms (point, line, bar, ribbon, density), partial for some statistical layers (`geom_smooth`, faceting with free scales), and weakest for non-standard extensions where you may need to drop down to native `plot_ly()` syntax.

6.25.4 Assumptions

The output environment renders HTML. Static targets (PDF, PNG, EPS) lose all interactivity, so for journal submission you should retain the original `ggplot` and re-render it with `ggsave()`; `plotly` is for screens, not paper.

6.25.5 R Implementation

```
library(ggplot2); library(plotly)

p ← ggplot(mtcars, aes(wt, mpg, colour = factor(cyl),
                       text = rownames(mtcars))) +
  geom_point(size = 3) +
  theme_minimal()

ggplotly(p, tooltip = "text")

# Link two plots
p2 ← ggplot(mtcars, aes(hp, mpg)) + geom_point()
subplot(ggplotly(p), ggplotly(p2))
```

6.25.6 Output & Results

An interactive HTML scatter where each marker shows the car name on hover, the legend is clickable to toggle cylinder groups, and drag-zoom narrows the axes. `subplot()` arranges multiple `ggplotly` objects side by side with synchronised axes — useful for cross-filtering between related views.

6.25.7 Interpretation

A reporting sentence: “Figure 2 is an interactive scatterplot rendered with `plotly::ggplotly()`; hovering over a marker shows the car name and cylinder count, and clicking a legend entry toggles the corresponding group.” Mention the technology in the figure caption when interactivity carries part of the figure’s meaning so static fallbacks (a screenshot for the print version) are not mistaken for the whole story.

6.25.8 Practical Tips

- Render the static `ggplot` and the `ggplotly` version from the same source; that way the printed and online versions stay in sync.

- For custom tooltips, include only the variables that matter: too many columns under the cursor become a wall of text. `aes(text = paste("Name:", rownames(data), "<br>HP:", hp))` is a common pattern.
- `subplot()` combines multiple `ggplotly` objects with optional shared axes; for many panels, build them via `lapply()` and combine with `do.call(subplot, ...)`.
- Performance degrades beyond a few thousand points; switch to `plot_ly(..., type = "scattergl")` for WebGL rendering when working with tens of thousands.
- Inside Shiny, swap `plotOutput()` / `renderPlot()` for `plotlyOutput()` / `renderPlotly()` and the rest of the app remains unchanged.
- Use `config(displayModeBar = FALSE)` to suppress the toolbar for embedded reports where the toolbar is more clutter than feature.

6.25.9 R Packages Used

`plotly` for `ggplotly()`, `plot_ly()`, `subplot()`, and the JavaScript bindings; `ggplot2` for the static plot underlying the conversion; `htmlwidgets` for embedding the result in standalone HTML pages.

6.25.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.25.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.26 Line Plots

6.26.1 Introduction

Line plots connect ordered observations and exploit a powerful perceptual cue: the human eye reads slope as rate of change effortlessly. They are the default geom for time-series, dose-response curves, individual-trajectory ("spaghetti") displays, cumulative distributions, and any context where the x-axis carries a meaningful sequence. The same simplicity that makes them effective also makes them dangerous when the x-axis lacks order — connecting unordered categories with a line implies a trajectory that does not exist and misleads readers into seeing trends in noise.

6.26.2 Prerequisites

A working knowledge of `ggplot2`'s aesthetic mappings, a clear distinction between continuous, ordered, and unordered variables, and an understanding of how `group` aesthetics control which points get joined.

6.26.3 Theory

`geom_line()` joins points in the order they appear within each group defined by the `group` aesthetic; default groups come from any discrete colour, fill, or linetype mapping. For paired or repeated-measures data, the `group` aesthetic must be set explicitly to the subject identifier so that individual trajectories are connected, not all observations across subjects. Variants include:

- `geom_step()` for cumulative quantities such as ECDFs or Kaplan–Meier survival curves, where the underlying process changes only at observed events.
- `geom_path()` for trajectories ordered by row position rather than the x-aesthetic, useful for state-space plots.
- `geom_ribbon()` paired with a line for confidence or prediction bands.

For many overlapping lines, alpha-blending or faceting prevents the "spaghetti pile" that hides individual trajectories.

6.26.4 Assumptions

The x-axis values are ordered meaningfully — by time, dose, position, or rank. Connecting nominal categories with a line is a presentation error, not a stylistic choice.

6.26.5 R Implementation

```r
library(ggplot2)

# Single time series
ggplot(economics, aes(date, unemploy)) +
  geom_line(colour = "#2A9D8F", linewidth = 1) +
  theme_minimal()

# Multiple groups
ggplot(airquality, aes(Day, Ozone, group = Month, colour = factor(Month))) +
  geom_line(linewidth = 1) +
  scale_colour_brewer(palette = "Set2") +
  theme_minimal()

# Paired trajectories (pre/post style)
df <- data.frame(
  subject = rep(1:20, each = 2),
  time    = rep(c("pre", "post"), 20),
  value   = c(rnorm(20, 50, 10), rnorm(20, 55, 10))
)
ggplot(df, aes(time, value, group = subject)) +
  geom_line(colour = "grey70") +
  geom_point(size = 2) +
  theme_minimal()

# Step line (cumulative)
ggplot(mtcars, aes(wt)) +
  stat_ecdf(geom = "step", colour = "#2A9D8F", linewidth = 1) +
  theme_minimal()
```

6.26.6 Output & Results

Four illustrative line plots: a long-running unemployment series, monthly ozone levels coloured by month, individual subject trajectories from pre to post, and the empirical cumulative distribution function of car weights as a step line. Each shows a different idiomatic use of the line geom.

6.26.7 Interpretation

A reporting sentence: “Individual subject responses from pre to post are connected by grey lines, with overlaid points indicating measurement times; group-mean changes (not shown) are reported in Table 1.” Always describe the unit of analysis a line represents (subject, country, replicate); a line plot without that context is ambiguous.

6.26.8 Practical Tips

- Set the `group` aesthetic explicitly for paired or repeated-measures data; the default group inferred from colour or linetype rarely matches the intended unit.
- Add `geom_point()` on top of `geom_line()` when x-values are sparse so individual observations remain visible; pure lines hide irregular sampling.
- For uncertainty bands, layer `geom_ribbon(aes(ymin = lo, ymax = hi), alpha = 0.3)` *before* the line so the line draws on top.

- Use `geom_step()` for ECDFs and survival curves — they are step functions, not smooth ones, and a smooth interpolation misrepresents the underlying mathematics.
- For dozens of overlapping trajectories, lower `alpha` (0.2–0.4) and add a single bold line for the group mean or median; pure spaghetti loses readability above 30–40 trajectories.
- When the y-axis spans orders of magnitude with multiplicative growth, use `scale_y_log10()` so constant-rate growth reads as a straight line.

6.26.9 R Packages Used

`ggplot2` for `geom_line()`, `geom_step()`, `geom_path()`, `geom_ribbon()`, and `stat_ecdf()`; `ggrepel` for non-overlapping line-end labels; `directlabels` for legend-free per-series labelling that scales better than colour-and-legend in print.

6.26.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.26.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.27 Pairs Plots

6.27.1 Introduction

A pairs plot — also called a scatterplot matrix — shows every pairwise scatter for a set of continuous variables in a single grid figure. Modern variants enrich the matrix with marginal histograms or densities along the diagonal and correlation coefficients (or other pairwise summaries) in the upper triangle. The result is the single most efficient exploratory diagnostic for a small-to-moderate set of variables: it surfaces non-linearity, outliers, subgroup separation, and multicollinearity in one pass.

6.27.2 Prerequisites

A working knowledge of scatter plots, correlation, and group-coloured aesthetics; familiarity with the `GGally` package's extensions to `ggplot2`.

6.27.3 Theory

For p variables the matrix has p^2 cells. The diagonal shows the marginal distribution of each variable (histogram, density, or a labelled bar for factors). The lower triangle shows pairwise scatter plots; the upper triangle shows summary statistics — most commonly Pearson, Spearman, or Kendall correlations, optionally annotated by significance. `GGally::ggpairs()` is the canonical implementation, with arguments `lower`, `upper`, and `diag` each accepting a list of cell-type specifications. Group structure is added via `aes(colour = group)` which propagates through every panel and provides immediate insight into between-group differences.

6.27.4 Assumptions

Each pairwise cell inherits the assumptions of the relevant geom: scatter plots assume continuous data; correlation cells assume meaningful linear (or rank) association. Categorical variables are displayed with bar charts and contingency tables in the off-diagonal cells.

6.27.5 R Implementation

```r
library(GGally)

# Basic pairs plot
ggpairs(iris, columns = 1:4, aes(colour = Species, alpha = 0.5)) +
  theme_minimal()

# Custom lower-triangle geoms
ggpairs(mtcars, columns = c("mpg", "wt", "hp", "qsec"),
        lower = list(continuous = "smooth"))
```

6.27.6 Output & Results

Two pairs plots: a four-variable matrix of the iris dataset coloured by species, and a four-variable matrix of `mtcars` numerical columns with LOESS smoothers in the lower triangle. The first reveals the species-driven separation across nearly every pairwise plot; the second emphasises the curvilinear weight–mpg and horsepower–quarter-mile relationships that a single scatter would obscure.

6.27.7 Interpretation

A reporting sentence (figure caption): “Pairs plot of four iris measurements (n = 150), coloured by species; diagonal shows kernel densities, lower triangle shows pairwise scatter, upper triangle shows Pearson correlations within each species (`GGally::ggpairs`).” Always state the correlation type used in the upper triangle and the geom chosen for the lower; defaults differ between versions.

6.27.8 Practical Tips

- Keep the matrix to 3–6 variables; cells shrink quadratically and become unreadable beyond that. For more variables, switch to a correlation heatmap or a focused PCA.

- Add `aes(colour = group)` to surface between-group structure; the same colour propagates through diagonal densities and lower-triangle scatters automatically.
- For large n, replace scatter cells with hexbin via `lower = list(continuous = "hex")`; a 50-by-50 grid scales to millions of points where individual scatters fail.
- `diag = list(continuous = "barDiag")` shows histograms instead of densities; useful for discrete-looking continuous variables (e.g. integer scores).
- Save pairs plots at large dimensions (`ggsave(..., width = 10, height = 10, dpi = 300)`); shrinking them to a single column makes the cells unreadable.
- For posterior or simulation diagnostics, `bayesplot::mcmc_pairs()` is the Bayesian-flavoured analogue and overlays divergent transitions on the off-diagonal cells.

6.27.9 R Packages Used

`GGally` for `ggpairs()`, `ggcorr()`, and the `wrap()` helper that lets you customise individual cell types; `ggplot2` for the underlying grammar; `bayesplot::mcmc_pairs()` for posterior-draw matrices in MCMC diagnostics.

6.27.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.27.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.28 Patchwork: Multi-Plot Composition

6.28.1 Introduction

Almost every journal-worthy figure is a composition: two or more panels arranged with shared legends, alphabetical tags, a unifying title, and aligned axes. Building such figures by hand with `gridExtra` or `cowplot` is verbose and brittle to layout changes. The `patchwork` package replaces that workflow with arithmetic-like operators that mirror the way the figure is read: `+` for "next to", `/` for "below", `|` for "in a row". Combined with `plot_layout()` for relative sizes and `plot_annotation()` for shared titles and tags, it is now the standard tool for assembling multi-panel `ggplot2` figures.

6.28.2 Prerequisites

A working knowledge of `ggplot2` — geoms, scales, themes — and an idea of which combinations of plots actually communicate (rather than merely fill space).

6.28.3 Theory

`patchwork` treats each `ggplot` as an object that supports a small algebra:

- `p1 + p2` places two plots side by side at equal width.
- `p1 / p2` stacks them vertically.
- Parentheses control nesting: `(p1 | p2) / p3` puts `p1` and `p2` horizontally with `p3` spanning below.
- `plot_layout(guides = "collect")` deduplicates shared legends, moving a single copy to the figure margin.
- `plot_annotation(title = ..., tag_levels = "A")` adds a global title and panel tags (A, B, C, ...) automatically.
- `& theme_minimal()` applies a theme to every panel in the composition simultaneously, useful when individual panels were built without a consistent theme.

The package handles axis alignment automatically when plots share axis ranges, which avoids the tiny horizontal offsets that bedevil hand-aligned multi-panel figures.

6.28.4 Assumptions

No statistical assumptions; the only practical constraint is that combined panels must remain legible at the final printed size, which usually means three to four panels per row at most.

6.28.5 R Implementation

```
library(ggplot2); library(patchwork)

p1 ← ggplot(mtcars, aes(wt, mpg))       + geom_point() + theme_minimal()
p2 ← ggplot(mtcars, aes(factor(cyl)))   + geom_bar()   + theme_minimal()
p3 ← ggplot(mtcars, aes(hp))             + geom_histogram(bins = 15) +
theme_minimal()

# Two side by side
p1 + p2

# Stacked
p1 / p2

# Complex layout
(p1 | p2) / p3 +
  plot_annotation(title = "mtcars overview", tag_levels = "A")

# Collect shared legends
p1 + p2 + plot_layout(guides = "collect")
```

6.28.6 Output & Results

Four progressively more complex layouts: two horizontal panels, two vertical panels, a 2-up-1-down arrangement with global title and panel tags A, B, C, and a side-by-side composition with one shared legend collected into the right margin. Saving any of these

with `ggsave()` produces a single file at the specified dimensions, ready for journal submission.

6.28.7 Interpretation

A reporting sentence (figure caption): “Figure 2 (assembled with `patchwork`): A) weight vs miles-per-gallon, B) cylinder distribution, C) horsepower distribution; panels share a common minimal theme and a single legend.” Caption-level credit to the composition tool is unnecessary, but tags and a one-line description of each panel are essential for any multi-panel figure.

6.28.8 Practical Tips

- Use `plot_layout(widths = c(2, 1))` to give one panel twice the horizontal space of another; `heights` does the same vertically.
- `tag_levels = "A"` produces A, B, C tags; `"1"` produces 1, 2, 3; `"i"` produces lowercase Roman numerals.
- `& theme_minimal()` (note the `&`) applies a theme globally to every panel; `+` only applies to the most recently added plot.
- For many small panels with the same structure, prefer `facet_wrap()` inside a single `ggplot` rather than composing dozens of plots; faceting is more efficient and produces aligned axes by default.
- Always save the composed plot as a single file via `ggsave()`; assembling separate exports in a vector editor produces fragile artwork.
- When a shared legend has many entries, place it on the right via `plot_layout(guides = "collect") & theme(legend.position = "right")` rather than letting it stack vertically inside one panel.

6.28.9 R Packages Used

`patchwork` for composition, `ggplot2` for the underlying plots, and `cowplot` for the alternative `plot_grid()` API and the `draw_image()` helper when embedding bitmap insets is required.

6.28.10 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

6.28.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.29 Raincloud Plots

6.29.1 Introduction

A raincloud plot is the modern alternative to the bar-with-error-bar chart for comparing continuous distributions across groups. Proposed by Allen and colleagues in 2019, it combines three pieces of information into one panel: a half-violin showing the kernel density (the "cloud"), jittered raw observations (the "rain"), and a small boxplot encoding the median and quartiles. The result preserves all the distributional information that bars throw away — modes, skew, outliers, sample sizes — while still presenting the central tendency that drives most reporting decisions.

6.29.2 Prerequisites

A working knowledge of boxplots, violin plots, kernel density estimation, and `ggplot2` aesthetic mappings.

6.29.3 Theory

A raincloud plot stacks three layers per group:

1. **Half-violin** drawn to one side (`stat_halfeye()` from `ggdist` or `geom_violinh()` from `gghalves`); displays the density shape.
2. **Jittered raw points** (`geom_point()` with `position_jitter()` or `ggbeeswarm::geom_beeswarm()`); shows individual observations.
3. **Narrow boxplot** with hidden outliers (`geom_boxplot(width = 0.12, outlier.shape = NA)`); communicates the median, IQR, and whisker range.

`coord_flip()` rotates the layout to horizontal, which makes long category names easier to read. The half-violin opens away from the data column so the cloud, the rain, and the boxplot sit side by side rather than stacked.

6.29.4 Assumptions

A continuous outcome and a categorical (or ordered) grouping variable. Each group should have enough observations (at least 20–30) for the kernel density to be meaningful; below that, drop the half-violin and show the points alone.

6.29.5 R Implementation

```
library(ggplot2); library(ggdist)

ggplot(iris, aes(Species, Sepal.Length, fill = Species)) +
  stat_halfeye(adjust = 0.5, justification = -0.2, .width = 0, point_colour = NA) +
  geom_boxplot(width = 0.12, outlier.shape = NA, alpha = 0.5) +
  geom_point(position = position_jitter(width = 0.05, seed = 1),
             size = 1.3, alpha = 0.5) +
  scale_fill_brewer(palette = "Set2") +
  coord_flip() +
  theme_minimal()
```

6.29.6 Output & Results

A horizontal raincloud plot for the iris dataset: each species occupies one row with its density rendered as a half-violin to the right, a small boxplot in the middle, and jittered raw points to the left. The full distribution, summary statistics, and individual observations are visible simultaneously.

6.29.7 Interpretation

A reporting sentence (figure caption): “Raincloud plots of sepal length by species (n = 50 per group): the half-violin shows kernel-density estimates (`adjust = 0.5`), the jittered points show individual observations, and the boxplot summarises the median and IQR. Setosa is well-separated from the other two species, which overlap considerably.” Mention the bandwidth (or `adjust`) used; it shapes the visible structure of the cloud.

6.29.8 Practical Tips

- Use `coord_flip()` for long group names; vertical raincloud plots crowd labels and waste space.
- Drop the boxplot layer when group sizes are small (< 30); it adds visual clutter without summarising much.
- For very large per-group n (thousands), replace jittered points with a single hexbin column or an alpha-blended strip; thousands of dots stack into noise.
- Set `outlier.shape = NA` in the boxplot so individual outliers do not appear twice (once in the rain, once flagged by the box whiskers).
- Use `ggbeeswarm::geom_beeswarm()` instead of `position_jitter()` when collisions matter; it produces a tidy spread along the category axis.
- `ggdist` is the modern implementation and integrates with Bayesian posterior summaries; `gghalves` is an older alternative still in active use.

6.29.9 R Packages Used

`ggdist` for `stat_halfeye()`, `stat_dots()`, and other slabinterval geoms; `ggplot2` for the underlying grammar; `gghalves` as an older alternative for half-violins; `ggbeeswarm` for collision-free jittered rain layers.

6.29.10 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

6.29.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.30 Ridge Plots

6.30.1 Introduction

A ridge plot stacks kernel-density curves for multiple groups vertically with a small overlap, evoking the cover of a 1979 Joy Division album (hence the alternative name "joy plot"). The idiom is particularly effective for comparing the shape of distributions across many ordered groups — months of the year, dose levels, time periods, ranked categories — where a panel of separate density plots would be too sparse and a violin plot would lose detail in the overlap.

6.30.2 Prerequisites

A working knowledge of kernel density estimation, grouped distributional comparisons, and `ggplot2`'s aesthetic mappings.

6.30.3 Theory

Each group's density curve is computed independently and then offset vertically by the group's position on the y-axis. The `scale` argument controls how much the curves overlap: a `scale` of 1 means the tallest curve in each group exactly reaches the next baseline, while values above 1 produce overlap and values below 1 produce gaps. The fill aesthetic can encode the group itself (categorical fill) or be mapped to the x-value via `after_stat(x)` to produce gradient fills that highlight quantiles or thresholds. The horizontal axis carries the variable of interest; the vertical axis carries group identity, ideally ordered.

6.30.4 Assumptions

A continuous outcome variable, a categorical (or ordered) grouping variable, and at least a few dozen observations per group so that the kernel density estimate is meaningful. With fewer than 20–30 observations per group, ridge plots can produce noisy, misleading curves.

6.30.5 R Implementation

```r
library(ggplot2); library(ggridges)

# Classic ridge plot
ggplot(iris, aes(Sepal.Length, Species, fill = Species)) +
  geom_density_ridges(alpha = 0.6) +
  scale_fill_brewer(palette = "Set2") +
  theme_ridges()

# With gradient fill encoding quantiles
ggplot(diamonds, aes(price, cut, fill = after_stat(x))) +
  stat_density_ridges(geom = "density_ridges_gradient", scale = 2) +
  scale_fill_viridis_c() +
  scale_x_continuous(trans = "log10") +
  theme_ridges()
```

6.30.6 Output & Results

Two ridge plots: the iris data showing three species along the sepal-length axis with translucent fills, and the diamonds data showing the price distribution per cut on a log scale with a gradient fill that encodes price itself. The second example is a particularly good use of ridges — five ordered groups, log-transformed x, and a gradient fill that emphasises the right tail.

6.30.7 Interpretation

A reporting sentence (figure caption): "Density of log-price per cut for 53,940 diamonds, computed via kernel density estimation with default bandwidth; fill encodes log price to highlight quantile structure across cuts." Mention bandwidth and any transformations because both shape the visible structure.

6.30.8 Practical Tips

- Choose `scale` deliberately: 1.5–2 is the typical sweet spot for 6–10 groups; tighter ranges work for fewer groups, wider ranges for many.
- Order groups meaningfully (`forcats::fct_reorder()`, `fct_relevel()`); alphabetical ordering rarely communicates the structure you want to show.
- For highly skewed distributions, log-transform the x-axis (`scale_x_continuous(trans = "log10")`); the ridges then capture multiplicative differences across groups.
- `theme_ridges()` removes axis lines and adjusts margins to suit stacked densities; pair with a sequential or qualitative palette.
- Ridge plots struggle with very few groups (two or three) where violin plots or paired density plots communicate better; reach for ridges when six or more groups need to be compared simultaneously.
- For groups with very different sample sizes, consider showing density on a comparable scale (`scale = 0.95`) so a small group does not dominate the figure visually.

6.30.9 R Packages Used

`ggridges` for `geom_density_ridges()`, `stat_density_ridges()`, and `theme_ridges()`; `ggplot2` for the underlying grammar; `viridis` and `RColorBrewer` for sequential and qualitative fill scales appropriate to ordered group comparisons.

6.30.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.30.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.31 ROC Curves

6.31.1 Introduction

A receiver operating characteristic (ROC) curve traces a binary classifier's sensitivity against $1-$ specificity as the decision threshold sweeps from one end of the score range to the other. It is the canonical diagnostic-performance figure in medicine and machine learning, summarising the trade-off between true-positive and false-positive rates without committing to a single threshold. The area under the curve (AUC) condenses the curve into one number, equal to the probability that the classifier ranks a randomly chosen positive case higher than a randomly chosen negative one.

6.31.2 Prerequisites

A working understanding of sensitivity, specificity, decision thresholds, and the difference between calibration and discrimination in classifier evaluation.

6.31.3 Theory

For a continuous score s and a binary outcome y, the ROC curve plots

$$\big(\mathrm{FPR}(t), \mathrm{TPR}(t)\big) = \big(P(s > t \mid y = 0), P(s > t \mid y = 1)\big)$$

for thresholds t. A perfect classifier passes through $(0, 1)$; a random one follows the diagonal. The AUC equals the Mann-Whitney U statistic divided by the sample sizes; it lies in $[0, 1]$ with 0.5 representing chance and 1 a perfect rank-ordering. Conventional benchmarks: AUC > 0.7 acceptable, > 0.8 good, > 0.9 excellent. DeLong's test compares two AUCs with correlated samples.

6.31.4 Assumptions

A binary outcome and a continuous (or ordinal) score. The ROC curve is invariant to monotone transformations of the score, so calibration is irrelevant to the curve itself; thresholding decisions, however, do depend on calibration.

6.31.5 R Implementation

```r
library(pROC); library(plotROC); library(ggplot2)

set.seed(2026)
n ← 200
disease ← rbinom(n, 1, 0.5)
score   ← rnorm(n, mean = disease, sd = 1)

# pROC
roc_obj ← roc(disease, score, quiet = TRUE)
auc(roc_obj); ci.auc(roc_obj)
plot(roc_obj, legacy.axes = TRUE, col = "#2A9D8F", lwd = 2)

# ggplot via plotROC
df ← data.frame(D = disease, score = score)
ggplot(df, aes(d = D, m = score)) +
  geom_roc(labelsize = 3, labelround = 2, colour = "#2A9D8F", size = 1) +
  style_roc() +
  labs(title = paste0("AUC = ", round(auc(roc_obj), 3)))
```

6.31.6 Output & Results

A monotone curve rising from $(0, 0)$ toward $(1, 1)$, with optional threshold labels at selected points and the AUC reported either in the title or via `ci.auc()` for its 95 % confidence interval. The `plotROC` package adds `ggplot2`-native styling; `pROC::plot()` produces a fast base-R version.

6.31.7 Interpretation

A reporting sentence: “The classifier achieved an AUC of 0.85 (95 % CI 0.79 to 0.91), indicating good discrimination; at the Youden-optimal threshold of 0.42, sensitivity was 0.78 and specificity was 0.81.” Always pair the AUC with its confidence interval and at least one operating point so readers can judge clinical utility.

6.31.8 Practical Tips

- Report the AUC with its 95 % CI, not just a point estimate; the bootstrap interval (`ci.auc(roc_obj)`) handles the typical correlated-sample structure.
- Compare two ROC curves with DeLong’s test (`pROC::roc.test`); pairwise AUC differences with p-values are the standard way to claim one classifier outperforms another.
- Avoid smoothed ROC curves (`smooth = TRUE`) for primary reporting; smoothing hides finite-sample irregularities. Show smoothed and raw side by side when smoothing aids interpretation.
- Under severe class imbalance, precision-recall curves often communicate operating-point trade-offs better than ROC; report both when positives are rare.
- Always draw the diagonal reference line ($y = x$); without it, readers cannot judge how far above chance the classifier is.
- For threshold selection, mark the Youden index, the equal-error-rate point, or a clinically motivated sensitivity / specificity target on the curve and quote its threshold.

6.31.9 R Packages Used

`pROC` for ROC objects, AUC, confidence intervals, and DeLong's test; `plotROC` for `ggplot2`-native ROC layers (`geom_roc()`, `style_roc()`); `ROCit` for an alternative API with bootstrap CIs; `yardstick` for tidy classifier metrics within the `tidymodels` ecosystem.

6.31.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.31.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.32 Scales and Coordinates

6.32.1 Introduction

Scales and coordinates are the two layers of `ggplot2` that turn raw data into visual marks on a page. **Scales** map data values to visual properties — numbers to positions, factor levels to colours, ordered values to sizes — while **coordinates** describe the geometry of the plot area itself — Cartesian, polar, log, fixed aspect ratio. Most figure customisation happens at the scale level; coordinates change less often, but understanding the difference between them is what lets you switch between zooming, transforming, and projecting without breaking the rest of the plot.

6.32.2 Prerequisites

A working knowledge of `ggplot2` aesthetics and geoms, and basic familiarity with how `aes()` mappings flow through to scale choices.

6.32.3 Theory

Every aesthetic has a family of scales:

- `scale_x_continuous`, `scale_x_discrete`, `scale_x_log10`, `scale_x_date`, `scale_x_datetime` for x; `scale_y_*` for y.
- `scale_colour_*` and `scale_fill_*` for colour and fill, with continuous, discrete, viridis, brewer, gradient, and manual variants.
- `scale_size_*`, `scale_alpha_*`, `scale_shape_*`, `scale_linetype_*` for the remaining aesthetics.

Each scale exposes `breaks`, `labels`, `limits`, `trans`, and (for continuous) `expand` arguments that control tick placement, label formatting, axis range, monotone transformations, and padding.

Coordinates change the geometry of the panel:

- `coord_cartesian()` is the default; supplying `xlim` / `ylim` zooms without dropping data.
- `coord_flip()` swaps x and y axes after scales are applied.
- `coord_polar()` produces pie charts and radial layouts.
- `coord_fixed(ratio = 1)` enforces equal aspect ratio, essential for maps and structural diagrams.
- `coord_sf()` and `coord_map()` apply geographic projections.

The crucial distinction: `scale_x_continuous(limits = c(a, b))` *filters* observations outside `[a, b]` and recomputes statistics on the remaining data; `coord_cartesian(xlim = c(a, b))` *zooms* without filtering, leaving statistics untouched.

6.32.4 Assumptions

The data supplied are valid for the chosen scale (no negative values for log scales, no fractional dates for date scales).

6.32.5 R Implementation

```
library(ggplot2); library(scales)

# Log-scale y with sensible breaks
ggplot(diamonds, aes(carat, price)) +
  geom_point(alpha = 0.1) +
  scale_y_log10(labels = scales::dollar_format()) +
  theme_minimal()

# Manual colour and custom axis formatting
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point(size = 3) +
  scale_colour_manual(values = c("4" = "#2A9D8F",
                                 "6" = "#F4A261",
                                 "8" = "#6A4C93")) +
  scale_x_continuous(breaks = 1:5, labels = paste0(1:5, "K lbs")) +
  theme_minimal()

# Coord flip: horizontal bar chart
ggplot(mpg, aes(class)) +
  geom_bar(fill = "#2A9D8F") +
  coord_flip() +
  theme_minimal()

# Zoom without filtering
ggplot(diamonds, aes(carat, price)) +
  geom_point(alpha = 0.1) +
  coord_cartesian(xlim = c(0, 2)) +
  theme_minimal()
```

6.32.6 Output & Results

Four illustrative plots: a log-scale price axis with currency formatting, a manual colour scale with custom x-axis labels, a horizontal bar chart via `coord_flip()`, and a zoomed-in scatter that preserves the full underlying dataset for accurate statistical layers.

6.32.7 Interpretation

A reporting sentence: "Price is shown on a base-10 log scale (`scale_y_log10`); the linear relationship between log-price and carat reflects an approximately power-law pricing structure." Note any non-default scale or coordinate in the figure caption — a log-scale axis is easy to overlook and can mislead readers expecting linear interpretation.

6.32.8 Practical Tips

- Reach for `scale_y_log10()` whenever the y-axis spans orders of magnitude; constant-rate growth then reads as a straight line.
- Use `coord_cartesian()` to zoom — `scale_x_continuous(limits = ...)` *removes* the filtered data and recomputes any smoother or summary, which can mislead.
- Always label axis units; "Weight" alone is ambiguous, "Weight (1000 lbs)" is informative.
- For date axes, use `scale_x_date(date_breaks = "1 year", date_labels = "%Y")` and pick break frequencies that match the publication width.
- Reverse a scale with `scale_x_reverse()` (continuous) or `scale_x_discrete(limits = rev)` (categorical); avoid hand-flipping data frames.
- For diverging numeric scales centred at zero, combine `scale_fill_gradient2()` with explicit `low`, `mid`, `high` colours; avoid one-sided palettes that misrepresent symmetry.

6.32.9 R Packages Used

`ggplot2` for the scale and coordinate functions; `scales` for label formatters (`dollar_format`, `percent_format`, `comma_format`, `label_log`); `ggh4x` for nested axis labels and per-panel scale control; `cowplot` for `theme_cowplot()` paired with these scales for a publication look.

6.32.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.32.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.33 Scatter Plots

6.33.1 Introduction

A scatter plot is the default visualisation for the relationship between two continuous variables: each observation becomes a point, position encodes the value pair, and the eye reads correlation, clustering, and outliers directly. It is the first graphic to make whenever a regression or correlation is contemplated. The simplicity is also the trap: at modest sample sizes a scatter is informative, but at large n overplotting hides density, and at small n a few points are easy to misread as a strong relationship that disappears under any reasonable significance test.

6.33.2 Prerequisites

A working knowledge of `ggplot2`'s aesthetic mappings and an idea of how the bivariate distribution differs from marginal univariate summaries.

6.33.3 Theory

The core geom is `geom_point()`, with point size, shape, alpha, and colour as the controllable aesthetics. Overplotting — multiple observations stacking on the same pixel — is the universal challenge once n exceeds a few hundred. Standard remedies:

- **Alpha blending**: lower the per-point opacity so density translates into colour intensity.
- **Jittering**: `position_jitter()` adds a small random offset; useful when one or both axes are discrete.
- **Hexagonal binning**: `geom_hex()` divides the plane into hexagons coloured by count; appropriate for $n \geq 5{,}000$.
- **Density colouring**: `ggpointdensity::geom_pointdensity()` keeps individual points but colours each by local density.
- **2D density contours**: `geom_density_2d()` and `stat_density_2d_filled()` overlay smooth level sets on top of a sparse scatter.

Trend layers come from `geom_smooth()`, with LOESS as the default for exploratory work and `method = "lm"` (or `"gam"`, `"glm"`) for parametric fits.

6.33.4 Assumptions

Both x and y are continuous; one categorical variable should be encoded with colour or shape rather than placed on an axis. No statistical assumptions until a smoother or fitted line is added.

6.33.5 R Implementation

```r
library(ggplot2); library(ggpointdensity)

# Simple scatter
ggplot(mtcars, aes(wt, mpg)) +
  geom_point(size = 3, colour = "#2A9D8F") +
  theme_minimal()

# Overplotted data with alpha
ggplot(diamonds, aes(carat, price)) +
  geom_point(alpha = 0.05) +
  theme_minimal()

# Point density colouring
ggplot(diamonds[sample(nrow(diamonds), 2000), ], aes(carat, price)) +
  geom_pointdensity() +
  scale_colour_viridis_c() +
  theme_minimal()

# Scatter with trend line
ggplot(mtcars, aes(wt, mpg)) +
  geom_point(size = 3) +
  geom_smooth(method = "lm", colour = "#F4A261") +
  theme_minimal()
```

6.33.6 Output & Results

Four scatter plots showing the same idea at different scales: a small clean scatter for the 32-row `mtcars`, a density-revealing alpha-blended scatter for the 53,940-row `diamonds`, a density-coloured point-by-point view for an intermediate sample, and a small scatter with an OLS smoother and confidence band overlay.

6.33.7 Interpretation

A reporting sentence: "Higher carat values are associated with higher price (Spearman $\rho = 0.93$, $n = 53{,}940$); the relationship is approximately linear on a log-log scale and the alpha-blended scatter reveals heteroscedasticity in the right tail." Always inspect a scatter before computing a correlation: non-linearity, subgroup structure, and outliers are visible here but hidden in a single summary number.

6.33.8 Practical Tips

- For $n > 1{,}000$ use alpha (0.1–0.3 typical); for $n > 10{,}000$ switch to hexbin or density colouring outright.
- Always label axes with units; "Weight" is ambiguous, "Weight (1000 lbs)" is informative.
- Add a smoother only when a relationship is plausible; over-eager LOESS curves through random scatter create the illusion of structure.
- For paired samples, connect points with `geom_line(aes(group = subject))` so within-subject change is visible alongside the cross-sectional spread.
- Use `stat_ellipse()` to draw 95 % data ellipses for group boundaries when colour-coded clusters need an extra visual hint.

- Avoid 3D scatter plots in print; rotation kills depth perception and the third dimension is better encoded as colour, size, or a faceting variable.

6.33.9 R Packages Used

`ggplot2` for `geom_point()`, `geom_smooth()`, `stat_ellipse()`; `ggpointdensity` for density-coloured scatter; `hexbin` for `geom_hex()` with large n; `ggrepel` for labelling individual points without overlap.

6.33.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.33.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.34 Stacked and Dodged Bars

6.34.1 Introduction

When a bar chart needs to display two categorical variables at once, three layout options compete: stacking the sub-bars within each main bar (showing totals and composition), placing them side by side (showing within-category comparison), or normalising to 100 % (showing proportions independent of total). Each layout answers a different question, and choosing the right one is more important than tweaking colours or fonts. The `position` argument of `geom_bar()` and `geom_col()` selects between the three.

6.34.2 Prerequisites

A working knowledge of bar charts, the distinction between `geom_bar()` (counts) and `geom_col()` (pre-computed values), and `ggplot2` aesthetic mappings.

6.34.3 Theory

`position = "stack"` is the default for bar geoms with a fill aesthetic: sub-bars stack vertically inside each main bar, the bar's total height equals the sum of the sub-counts, and segments encode composition.

`position = "dodge"` (or `position_dodge(width = ...)` for finer control) places sub-bars side by side, one per fill level, at the same x-tick. Each sub-bar's height equals its own count, making within-x comparison direct.

`position = "fill"` stacks but normalises every main bar to height 1 (or 100 %), hiding the absolute totals to expose proportional structure across the x-axis.

The choice maps directly onto the analytical question: “How big is each group?” → stack; “How does subgroup A compare to subgroup B within each main category?” → dodge; “How does the proportion of subgroup A shift across main categories?” → fill.

6.34.4 Assumptions

Two categorical variables (or one categorical and one ordered) and a count or summary that makes sense to add across sub-bars.

6.34.5 R Implementation

```r
library(ggplot2)

d ← data.frame(
  stage = factor(rep(c("I", "II", "III"), 2)),
  sex   = factor(rep(c("F", "M"), each = 3)),
  count = c(30, 45, 25, 25, 50, 30)
)

# Stacked
ggplot(d, aes(stage, count, fill = sex)) +
  geom_col(position = "stack") +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()

# Dodged
ggplot(d, aes(stage, count, fill = sex)) +
  geom_col(position = position_dodge(width = 0.8)) +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()

# Filled (proportions)
ggplot(d, aes(stage, count, fill = sex)) +
  geom_col(position = "fill") +
  scale_y_continuous(labels = scales::percent_format()) +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()
```

6.34.6 Output & Results

Three views of the same disease-stage-by-sex data: a stacked bar showing the total cases per stage decomposed by sex, a dodged bar enabling direct sex-vs-sex comparison within stage, and a filled bar showing the female-to-male proportion balance across stages with the y-axis as a percentage.

6.34.7 Interpretation

A reporting sentence (figure caption): “Stacked bars show total case counts per stage decomposed by sex; dodged bars display male and female counts side by side for direct comparison within each stage; filled bars normalise to 100 % to expose the sex proportion

across stages." Always describe which layout was chosen and why; it is the most common source of bar-chart misreading.

6.34.8 Practical Tips

- Stacked bars communicate totals cleanly but obscure within-stage comparisons because the segments above the first sub-bar do not start at zero — readers cannot judge their lengths accurately.
- Dodged bars need adequate bar width and stage spacing; with three or more fill levels, the figure widens quickly and benefits from `coord_flip()` for horizontal layouts.
- Filled bars hide absolute counts entirely; pair them with a text or table summary giving the per-stage n.
- For more than four fill levels, stacking and dodging both become hard to read; consider a heatmap, a faceted small-multiple, or a Sankey diagram instead.
- Order fill levels deliberately (`forcats::fct_relevel()`); the legend reads top-to-bottom and the bar reads bottom-to-top, so the visual flow depends on factor order.
- Use `geom_text(aes(label = count), position = position_stack(vjust = 0.5))` to label segments with their counts when the total alone is not enough.

6.34.9 R Packages Used

`ggplot2` for `geom_bar()` , `geom_col()` , and the `position_*` family; `forcats` for factor reordering of the fill levels; `scales` for percent and comma label formatters; `ggalluvial` for flow-style alternatives when stacked bars at three or more levels become confusing.

6.34.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.34.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.35 Survival Curves

6.35.1 Introduction

A Kaplan-Meier survival curve is the most recognisable figure in clinical biostatistics. It estimates the probability of surviving (or remaining event-free) past each time point and displays the result as a step function dropping at every observed event. A publication-grade KM plot is rarely just the curve, however — it carries a risk table beneath the panel

showing how many subjects remain at each landmark time, a log-rank p-value annotating the between-group comparison, and 95 % confidence bands shaded behind the curve. The `survminer` package combines all of these into a single `ggsurvplot()` call that works directly on `survfit` objects from the `survival` package.

6.35.2 Prerequisites

A working understanding of survival analysis, the Kaplan-Meier estimator, censoring, and the log-rank test. Familiarity with `ggplot2` helps when customising the resulting object.

6.35.3 Theory

The KM estimator is

$$\hat{S}(t) = \prod_{t_i \le t} \left(1 - \frac{d_i}{n_i}\right),$$

where d_i is the number of events and n_i the number at risk just before time t_i. The plot displays $\hat{S}(t)$ on the y-axis, time on the x-axis, with steps at each event and tick-marks (or shorter steps) at censoring times. Multiple groups are overlaid in different colours, and the risk-table below the panel shows n_i at chosen landmark times for each group. The log-rank p-value summarises the between-group difference under proportional hazards.

6.35.4 Assumptions

Right-censored survival data, non-informative censoring (the censoring mechanism does not depend on prognosis), and — for the log-rank annotation — approximately proportional hazards across groups. Severely non-proportional hazards make the log-rank test underpowered and a restricted-mean comparison may be more appropriate.

6.35.5 R Implementation

```
library(survival); library(survminer)

fit ← survfit(Surv(time, status) ~ sex, data = lung)

ggsurvplot(fit,
  data = lung,
  conf.int = TRUE,
  pval = TRUE,
  risk.table = TRUE,
  legend.labs = c("Male", "Female"),
  palette = c("#2A9D8F", "#F4A261"),
  xlab = "Days",
  ylab = "Survival probability",
  risk.table.y.text.col = TRUE
)

# Cumulative event curve (1 - S)
ggsurvplot(fit, data = lung, fun = "event")
```

6.35.6 Output & Results

A two-panel figure: the upper panel shows two survival curves (male and female) with shaded 95 % confidence bands and a log-rank p-value in the corner; the lower panel is a risk table tabulating the number at risk in each group at six evenly spaced time points. The cumulative-event variant (`fun = "event"`) inverts the y-axis to show $1 - \hat{S}(t)$, useful when the event rate is the more natural reporting scale.

6.35.7 Interpretation

A reporting sentence (figure caption): "Kaplan–Meier overall-survival curves by sex (`survival::survfit`); shaded bands are 95 % point-wise confidence intervals, the log-rank p-value compares the two groups, and the risk table reports the number of subjects at risk at each labelled time." Always state the time-zero anchor (date of randomisation, surgery, diagnosis) in the methods so the x-axis is unambiguous.

6.35.8 Practical Tips

- Always include a risk table; without it, the curve at late time points is misleading because the estimate is based on a tiny remaining cohort.
- Use `fun = "event"` for cumulative incidence framing when communicating to clinical audiences who think in event rates rather than survival probabilities.
- For competing risks, switch to a cumulative incidence function (CIF) plot via `cmprsk::cuminc()` and `ggcompetingrisks()`; KM curves overestimate the event of interest when competing events are non-trivial.
- Pair the log-rank p-value with a hazard-ratio estimate (e.g. from a univariate Cox model); the test alone gives no effect size.
- For multi-arm trials, lay out subgroups in panels (`facet.by = "subgroup"`) rather than crowding many curves into one panel.
- Export at 300 dpi (or higher for line art) and at the journal's column width so the risk-table text remains legible.

6.35.9 R Packages Used

`survival` for `survfit()` and `Surv()`; `survminer` for `ggsurvplot()`, `ggsurvtable()`, and risk-table support; `ggplot2` for downstream customisation of the returned object; `cmprsk` for competing-risks extensions when KM is not appropriate.

6.35.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.35.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.36 Time Series Plots

6.36.1 Introduction

Time-series plots are deceptively easy to make and surprisingly easy to ruin. The data are ordered, the x-axis is a date, and a single line is often the right geom — but the choice of tick spacing, transformation, and accompanying decomposition or forecast layer determines whether the plot communicates the underlying dynamics or hides them. Trend, seasonality, and forecast uncertainty are three distinct stories the same series can tell, and a good time-series figure usually shows at least two of them.

6.36.2 Prerequisites

A working knowledge of `ggplot2`, an understanding of R's date and date-time classes (`Date`, `POSIXct`), and basic familiarity with classical time-series concepts (trend, seasonality, forecast intervals).

6.36.3 Theory

Three structural elements appear in nearly every published time-series figure:

- A continuous date axis with carefully chosen `date_breaks` and `date_labels`. Defaults often produce overcrowded or sparse ticks; choose explicitly.
- A line geom for the observed series, optionally with a smooth or fitted-value overlay.
- An uncertainty layer — typically a `geom_ribbon()` for forecast prediction intervals or a faint shaded region for seasonal subsamples.

For multivariate decompositions, classical STL splits a series into trend, seasonal, and remainder components shown as a stacked panel. The `forecast::autoplot()` and `feasts::autoplot()` methods produce decomposition and forecast plots automatically; both are tidy-friendly and fit naturally into modern reporting workflows.

6.36.4 Assumptions

Time variable is monotonically ordered, on a regular grid (or explicitly handled if irregular), and on a class that `scale_x_date()` or `scale_x_datetime()` understands.

6.36.5 R Implementation

```r
library(ggplot2); library(forecast); library(ggfortify)

# Standard time series plot
ggplot(economics, aes(date, unemploy / 1000)) +
  geom_line(colour = "#2A9D8F", linewidth = 1) +
  scale_x_date(date_breaks = "10 years", date_labels = "%Y") +
  labs(x = NULL, y = "Unemployment (thousands)") +
  theme_minimal()

# Forecast with prediction intervals
fit ← auto.arima(AirPassengers)
fc  ← forecast(fit, h = 24)
autoplot(fc) +
  theme_minimal()

# STL decomposition
decomp ← stl(AirPassengers, s.window = "periodic")
autoplot(decomp) + theme_minimal()
```

6.36.6 Output & Results

Three figures: a long historical line plot of US unemployment with decade ticks, an ARIMA forecast with 80 % and 95 % prediction ribbons attached to the end of the historical series, and a four-panel STL decomposition of monthly air-passenger counts showing the rising trend, twelve-month seasonal cycle, and remaining residual variation.

6.36.7 Interpretation

A reporting sentence (figure caption): “Top: unemployment series 1967–2015 (thousands of persons). Middle: ARIMA(2,1,1)(0,1,0)

forecast for 24 months ahead with 80 % and 95 % prediction intervals. Bottom: STL decomposition (`s.window = "periodic"`) of monthly air-passenger counts.” Always show forecast intervals; a forecast line without uncertainty is misleading.

6.36.8 Practical Tips

- Convert text dates to `Date` or `POSIXct` before plotting; `lubridate::ymd()` , `dmy()` , and `as_datetime()` cover the common cases.
- Choose `date_breaks` to match the publication width: ten-year ticks for a multi-decade series, one-year ticks for a five-year window, monthly ticks for a single year.
- Log-transform the y-axis when the series exhibits multiplicative growth; the resulting plot reads percentage changes as constant slopes.
- For multiple series, prefer faceting (`facet_wrap(vars(metric), scales = "free_y")`) when scales differ; reach for colour only when all series share comparable scale.
- Use the modern `tsibble` + `feasts` + `fable` stack for tidy workflows; `forecast::autoplot()` remains useful for legacy `ts` objects and quick exploration.

- Always annotate forecast plots with the model class and forecast horizon in the caption; a ribbon without context can be mistaken for a confidence band on observed data.

6.36.9 R Packages Used

`ggplot2` for the underlying grammar, `forecast` for `auto.arima()` and `forecast()` plus `autoplot()` methods, `ggfortify` for `ts` and STL `autoplot()` extensions, and the modern `tsibble` / `feasts` / `fable` stack for tidy time-series workflows.

6.36.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.36.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.37 Violin Plots

6.37.1 Introduction

A violin plot displays a kernel-density estimate vertically and reflects it around the central axis, producing the symmetric outline that gives the geom its name. It combines the clarity of a density plot with the side-by-side grouping that boxplots provide, and — crucially — reveals multi-modality and skew that boxplots flatten into a single rectangle. For comparing the *shape* of distributions across categories, violin plots are often the right default; for comparing only the median and quartile structure, boxplots remain more compact.

6.37.2 Prerequisites

A working understanding of kernel density estimation, boxplots, and `ggplot2` aesthetic mappings.

6.37.3 Theory

For each group, `geom_violin()` computes a kernel density $\hat{f}(x)$ on the response variable, draws the density curve vertically, and mirrors it across the group's central axis. The resulting width at each y-value is proportional to $\hat{f}(y)$, so wide regions correspond to high density. The `scale` argument controls how groups are sized relative to each other: `"area"` makes all violins equal area, `"count"` scales width proportional to the group's

sample size, and `"width"` makes them all the same maximum width. The `trim` argument decides whether the density tails are truncated at the data extremes (`TRUE`, default) or extrapolated beyond them (`FALSE`).

6.37.4 Assumptions

A continuous outcome and a categorical (or ordered) grouping variable, with enough observations per group (at least 20–30) for the kernel density to be meaningful. Below that threshold the shape becomes unstable and a strip plot or jittered scatter is more honest.

6.37.5 R Implementation

```
library(ggplot2)

# Basic violin
ggplot(iris, aes(Species, Sepal.Length, fill = Species)) +
  geom_violin() +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()

# Violin + boxplot overlay
ggplot(iris, aes(Species, Sepal.Length, fill = Species)) +
  geom_violin(alpha = 0.5, trim = FALSE) +
  geom_boxplot(width = 0.2, fill = "white", outlier.shape = NA) +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()

# Violin + jittered points
ggplot(iris, aes(Species, Sepal.Length, fill = Species)) +
  geom_violin(alpha = 0.5) +
  geom_jitter(width = 0.1, alpha = 0.5) +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()
```

6.37.6 Output & Results

Three views of the iris sepal-length data: a plain violin per species, a violin with a narrow boxplot overlay showing median and quartiles, and a violin with jittered raw points overlaid for full transparency. The third version is the most informative when the audience is unfamiliar with kernel-density visualisations.

6.37.7 Interpretation

A reporting sentence (figure caption): “Sepal length by species (n = 50 per group); violin width is proportional to kernel-density estimate (`scale = "area"`), the inset boxplot summarises the median and IQR, and jittered points show individual observations.” Always describe what the violin’s width encodes; readers unfamiliar with the geom default to thinking it represents count.

6.37.8 Practical Tips

- For multi-modal distributions (e.g. Old Faithful eruption times), the violin reveals the second mode that a boxplot would hide entirely; this is the strongest single

argument for the geom.
- Set `trim = FALSE` only when the density tails are biologically meaningful; otherwise the default trim avoids implausible extrapolation.
- Combine with a narrow boxplot (`width = 0.15`) when readers expect numeric summaries; the inset preserves quartile information without dominating the visual.
- For paired or matched groups, `introdataviz::geom_split_violin()` produces left-right halves so two conditions share the same x-position.
- For large n, prefer violins over jittered points; violins scale to millions of observations whereas individual points stack into uninformative blobs.
- For small n (< 20 per group), a strip plot or a boxplot is more honest than a violin; the kernel density estimate is too noisy to trust.

6.37.9 R Packages Used

`ggplot2` for `geom_violin()`; `introdataviz` for split violins; `ggdist` for `stat_halfeye()` and other slabinterval geoms that combine violins with credible-interval annotations; `vioplot` for a base-R alternative when `ggplot2` is not available.

6.37.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

6.37.11 See also — labs in this chapter

- ggplot2 grammar and multi-panel layouts

6.38 Week 1, Session 5 — ggplot2 grammar and multi-panel layouts

Workflow labs use the variant template: Goal → Approach → Execution → Check → Report.

6.38.1 Learning objectives

- Build a ggplot by naming data, aesthetics, geoms, and scales rather than by calling a plot type.
- Use faceting to display the same relationship across subgroups.
- Combine several plots into a single figure with patchwork.

6.38.2 Prerequisites

Labs 1.2 through 1.4.

6.38.3 Background

ggplot2 is built on a grammar of graphics. Rather than call a function named “scatterplot” or “boxplot”, you declare what the plot contains: a dataset, mappings from variables to aesthetics (x, y, colour, size, shape), and one or more geoms that render those mappings. The result is that *any* plot you can picture can be built by the same small vocabulary. Once the vocabulary is second nature, you will spend less time searching for a plotting function and more time thinking about what the graph should show.

Multi-panel figures are the standard unit of publication. Faceting breaks one plot into small multiples — the same relationship repeated across a grouping variable — which is almost always a better answer than crowding more colours onto a single panel. The patchwork package takes independent ggplots and assembles them into a single figure using `+`, `|`, and `/` operators, so you can build a three-panel manuscript figure without leaving R.

A common beginner mistake is to treat ggplot as a collection of `geom_*` functions. It is cleaner to think of a plot as a data-plus- aesthetics object to which geoms are added. Changing the underlying data is then a substitution, not a rewrite.

6.38.4 Setup

```
library(tidyverse)
library(palmerpenguins)
library(patchwork)
set.seed(42)
theme_set(theme_minimal(base_size = 12))
```

6.38.5 1. Goal

Build a three-panel figure that describes bill morphology and body mass in the `penguins` dataset, using a scatter plot, a boxplot, and a faceted scatter plot, assembled with patchwork.

6.38.6 2. Approach

```
p ← penguins ▷
  drop_na(bill_length_mm, bill_depth_mm, body_mass_g, species, sex)
```

The plot object is built in three lines: data, aesthetics, geom.

```
p1 ← p ▷
  ggplot(aes(bill_length_mm, bill_depth_mm, colour = species)) +
  geom_point(alpha = 0.7) +
  labs(x = "Bill length (mm)",
       y = "Bill depth (mm)",
       colour = NULL)
p1
```

6.38.7 3. Execution

```
p2 ← p ▷
  ggplot(aes(species, body_mass_g, fill = species)) +
  geom_boxplot(alpha = 0.6, colour = "grey30") +
  labs(x = NULL, y = "Body mass (g)") +
  theme(legend.position = "none")
p2
```

```
p3 ← p ▷
  ggplot(aes(bill_length_mm, body_mass_g, colour = sex)) +
  geom_point(alpha = 0.6) +
  facet_wrap(~ species) +
  labs(x = "Bill length (mm)",
       y = "Body mass (g)",
       colour = NULL)
p3
```

Faceting shows the same relationship in three panels, one per species. The within-species slopes are visually separable from the between-species differences.

6.38.8 4. Check

Assemble the three panels with patchwork.

```
(p1 | p2) / p3 +
  plot_annotation(
    title = "Palmer penguins: bill morphology and body mass",
    tag_levels = "A"
  )
```

The layout operators: `|` places side-by-side, `/` stacks, `+` groups. `plot_annotation()` adds title and the familiar A/B/C labels used in manuscripts.

6.38.9 5. Report

> A three-panel figure was assembled from the `palmerpenguins` data (*n* = `nrow(p)` complete cases) using ggplot2 and patchwork. Panel A shows a clear separation of bill morphology among the three species. Panel B shows Gentoo as markedly heavier than the other two. Panel C, faceted by species, reveals a within-species positive association between bill length and body mass, with a sex offset visible in each panel.

Within-group and between-group variation are separate features of the data. Faceting is the simplest way to let a reader see both at once.

6.38.10 Common pitfalls

- Using colour to encode more than one thing at a time (group *and* magnitude — pick one).
- Mapping a continuous variable to shape.

- Overlaying more than four or five categories on a single scatter plot; use faceting instead.
- Defaulting to `theme_grey()`; the in-house convention is `theme_minimal(base_size = 12)`.

6.38.11 Further reading

- Wickham H. *ggplot2: Elegant Graphics for Data Analysis*, chapters on the grammar and on faceting.
- Pedersen T. *patchwork* package documentation.

6.38.12 Session info

```
sessionInfo()
```

6.38.13 See also — chapter index

- Methods in this chapter
- Find your method (Chapter 0)