Level up your labels

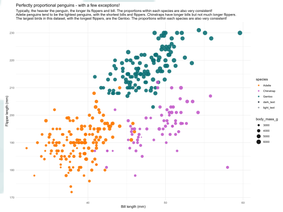
Tips and tricks for annotating plots

Cara Thompson, PhD | @cararthompson

Polished annotations can make all the difference between a good plot that contains all the necessary information, and a great plot that engages readers with a clear story. Whether we're using annotations to highlight different groups, to tell stories about an outlier data point, to add detail about key values or to explain how a predictive model works, applying a few simple tricks allows them to shine as integral parts of our data visualisations.

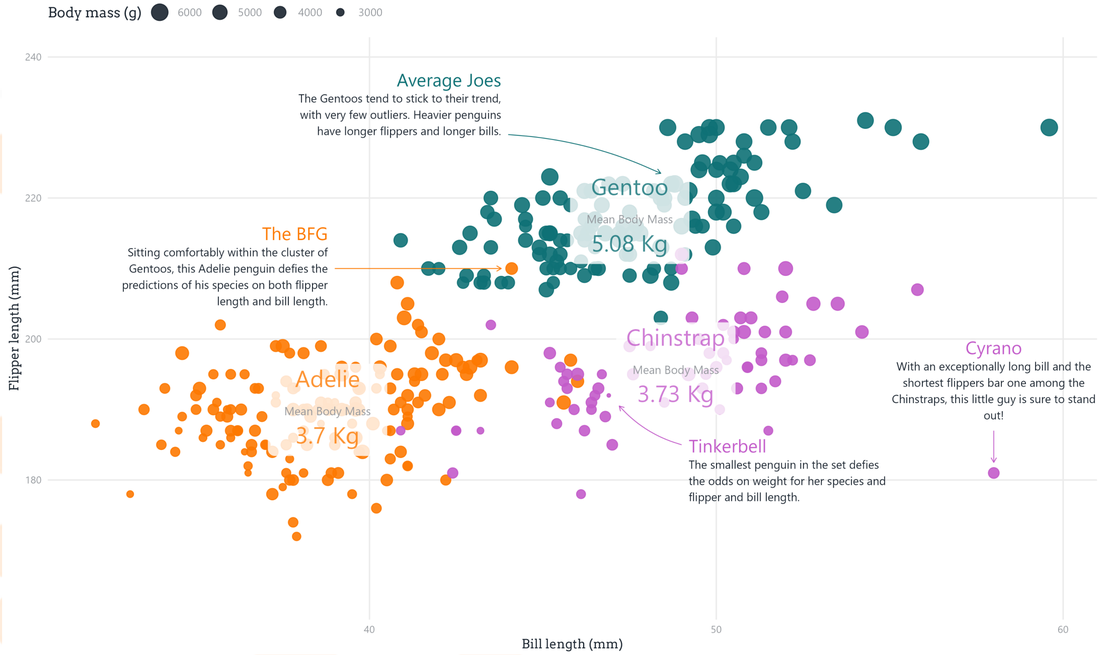
Here, I'm using the penguins dataset from 📦 `{palmerpenguins}` and have created two extra tibbles: p_summary (containing summary info by species) and p_exceptions (containing data, nicknames, and descriptions for the penguins I want to highlight).

Applying these five tips and the coding tricks to implement them gets us from this → to this ↓



Perfectly proportional penguins - with a few exceptions!

Typically, the heavier the penguin, the longer its flippers and bill. The proportions within each species are also very consistent. **Adelie** penguins tend to be the smallest penguins, with the shortest bills and flippers. **Chinstraps** have longer bills but not much longer flippers. The largest birds in this dataset, with the longest flippers, are the **Gentoo**.



#1 Make it easy for your readers

Use colour to help readers orient themselves as they read the explanatory text, using 📦 `{ggtext}` and a bit of CSS formatting. Set up your palette, then add the colours into the text using 📦 `{glue}`:

```
penguin_palette ← list("Adelie" = "#fd7901",
                       "Chinstrap" = "#c35bca",
                       "Gentoo" = "#0e7175")


plot +
  labs(subtitle = glue(" ... very consistent.<br>
<span style='color: {penguin_palette$Adelie}'>
**Adelie**</span> penguins tend to be the smallest
penguins, with the shortest bills and flippers.
<span style='color:{penguin_palette$Chinstrap}'>
**Chinstraps**</span> have longer bills .. ")) +
  theme(plot.subtitle = element_markdown())
```

#3 Use alignments to direct the reader's gaze

Work out which data points you want to highlight, and where the boxes will sit relative to them. Then apply text alignments programmatically, depending on whether the box will be to the left, right, top or bottom of the data point.

```
p_exceptions ← p_exceptions %>%
  mutate(hjust = case_when(label_x < bill_length_mm ~ 1,
                           label_x == bill_length_mm ~ 0.5,
                           TRUE ~ 0),
         vjust = case_when(label_y < flipper_length_mm ~ 0.85,
                           label_y < flipper_length_mm & label_x == bill_length_mm ~ 1,
                           label_y == flipper_length_mm ~ 0.5,
                           label_y > flipper_length_mm & label_x == bill_length_mm ~ 0,
                           TRUE ~ 0.15))

plot +
  geom_textbox(data = p_exceptions,
               aes(x = label_x, y = label_y,
                   colour = species,
                   label = glue("{nickname}<br><span style='color:
{penguin_palette$dark_text};font-size:9pt'>{description}</span>"),
                   vjust = vjust, valign = vjust,
                   hjust = hjust, halign = hjust),
               box.colour = NA)
```

#5 Give everything space to breathe

Make the most of theme options such as `lineheight` and `margin`, remove unnecessary legends, and expand your scales as required.

```
plot +
  scale_y_continuous(expansion = expand(c(0.2, 0.2))) +
  guides(colour = "none")
```

#2 Keep the main thing the main thing

Use a secondary text colour for explanatory text, such as the subtitle and the "mean body mass" label in the three big boxes. Make use of more CSS formatting to alter font size on the fly, and of the alpha property of `{ggtext}`'s `geom_textbox()` to place a box over data while keeping the main story visible.

```
penguin_palette ← list_modify(penguin_palette,
                              "dark_text" = "#1A242F",
                              "light_text" = "#94989D")


plot +
  geom_textbox(data = p_summaries,
               aes(x = mean_x, y = mean_y,
                   colour = species,
                   label = glue(
"{species}<br><span style='color:{penguin_palette$light_text};
font-size:9pt'>Mean Body Mass<br></span>{mean_mass_kg} Kg")),
               box.color = NA,
               alpha = 0.8) +
  theme(text = element_text(colour = penguin_palette$light_text))
```

#4 Add arrows sparingly

`annotate()` can take vectors of `x/end` and `y/end` coordinates, but not of curvatures. Build the data programmatically within a tibble, and loop through the unique curvatures in that tibble to add arrows with a single `annotate()` call.

```
p_exceptions ← p_exceptions %>%
  mutate(arrow_end_x = case_when(label_x < bill_length_mm ~
                                   bill_length_mm - 0.3,
                                 label_x == bill_length_mm ~
                                   bill_length_mm,
                                 TRUE ~ bill_length_mm + 0.3),
         arrow_end_y = # ... apply same principle ... %>%
  mutate(curvature = c(0, -0.15, -0.1, 0))

for(curv in unique(p_exceptions$curvature)) {
  filtered_data ← filter(p_exceptions,
                         curvature == curv)
  plot ← plot +
    annotate(geom = "curve",
             x = filtered_data$label_x,
             y = filtered_data$label_y,
             xend = filtered_data$arrow_end_x,
             yend = filtered_data$arrow_end_y
             curvature = curv,
             arrow = arrow(length = unit(1.5, "mm")))
}
```

CARA THOMPSON
BUILDING STORIES WITH DATA

Full plot code: cararthompson.com/talks/useR2022