Exploring the Wide World of ggplot2 Extensions

Instructor notes

Eric R. Scott Kristina Riemer

```r
library(palmerpenguins)
library(ggplot2)
```

Intro

Go through introductory slides to start. Spend time looking through the ggplot extension gallery and ask if there are any particular packages that look useful to folks. Get people to share their discoveries.

Demonstrations

Go through vignette or other short demo of several extension packages.

`gganimate`

```r
library(gganimate)
```

- Animate plots and save as gifs
- Takes some getting used to lingo
- Can control: what changes over time with `transition_*`, how the view of the axes changes with `view_*`, how much "memory" of the change there is, with `shadow_*`, and how data appears and disappears with `enter_*` and `exit_*`
- Use `?` with functions to figure out what label name is

- These penguin plots don't really make sense because the data isn't same individuals over the years; should set group in `transition_states`
- Mention `animate()` for more control
- Resources:
  - Bat data slides
  - Penguins code source

`esquisse`

```r
library(esquisse)
library(dplyr)
```

- "Esquisse" means rough first sketch in French
- `esquisse` vignette
- Point and click to generate ggplot plots from data
- Good for exploring datasets (look at data too)
- Still have to modify dataset outside of tool, e.g., recoding factors

```r
penguins <- penguins %>% na.omit()
esquisser(penguins)
```

- Helpful things: color codes variables, suggests plot types
- Save plot as image file, or copy and paste code into console (uses `tidyverse`)
- Can add to Shiny app to let non-R users create ggplots of data

`plotly`

```r
library(plotly)
```

- `plotly` R package is interface to a JavaScript library for making interactive data visualizations
- Only important to know that because if you search for `plotly` help, sometimes you'll get JavaScript code examples.

- `plotly` *kind of* uses the grammar of graphics, but building a plot from scratch is tricky
- `plotly` functions mostly take lists are arguments, which makes it very difficult to figure out what default values are or even what arguments functions can take.
- `ggplotly()` **transforms** `ggplot` objects into interactive `plotly` plots. 90% of the time it gets you 90% of the way there

```r
p <-
  ggplot(penguins, aes(x = bill_length_mm, y = bill_depth_mm, color = species)) +
  geom_point() +
  labs(x = "Bill length (mm)", y = "Bill depth (mm)", color = "Penguin Species") +
  theme_bw()

ggplotly(p)
```

By default, you get:

- Drag a box to zoom, double-click to zoom out
- Tooltips on hover
- Ability to hide and show points by clicking on the legend
- Toolbar with tools including ability to download a PNG of the plot

This works in R Markdown or Quarto documents (i.e. there's not a way to make these plots "stand-alone"—they're rendered as HTML)

Types of plots

Works for most (all?) built-in geoms. Works for some plots made with ggplot2 extensions also. Demo a few different geoms like `geom_boxplot()`, `geom_hisotgram()`, `geom_smooth()`

Customize tooltip

You can customize the info displayed in the tooltip with `ggplotly` . Give it additional aesthetic `text` to include something *only* as a tooltip.

```r
p <-
  ggplot(penguins, aes(x = bill_length_mm, y = bill_depth_mm, color = species)) +
  #Ignore unknown aesthetics warning
  geom_point(aes(text = sex)) +
  labs(x = "Bill length (mm)", y = "Bill depth (mm)", color = "Penguin Species") +
  theme_bw()

ggplotly(p, tooltip = c("text", "bill_depth_mm", "bill_length_mm"))
```

Animations

You can use `ggplotly()` to make interactive animated plots by using the `frame` aesthetic

```r
library(gapminder) #dataset
head(gapminder)
p2 <- ggplot(gapminder, aes(gdpPercap, lifeExp, color = continent)) +
  geom_point(aes(size = pop, frame = year, ids = country)) +
  scale_x_log10()

ggplotly(p2)
```

`ggrepel`

The `ggrepel` package is helpful for directly labeling plots, especially when labels would otherwise overlap.

```r
#make rownames into columns
dat <- mtcars |> tibble::rownames_to_column(var = "car")
dat
p3 <-
  ggplot(dat,
  aes(wt, mpg, label = car, colour = factor(cyl))) +
  geom_point()
#without ggrepel
p3 + geom_text()
p3 + geom_label()
library(ggrepel)
p3 + geom_text_repel()
p3 + geom_label_repel()
```

Adjust appearance of labels

By default, little segments are drawn connecting labels to points only when labels are far enough away. This can be adjusted.

```
p3 + geom_label_repel(min.segment.length = 0)
```

Overlapping labels

You see a warning about labels being removed due to overlaps. Make text smaller, plot larger, adjust `max.overlaps` , force labels further apart, make it try harder to find positions that don't overlap.

```
#don't usually need to do all of these things, but they're some options
p4 <-
  p3 +
  geom_label_repel(
    min.segment.length = 0,
    #smaller labels
    size = 3,
    label.padding = 0.15,
    #draw labels even if overlapping with things
    max.overlaps = 15, #default is 10, increase to get more labels drawn
    #force labels further from point
    force = 5,
    #ask it to try harder
    max.time = 2,
    max.iter = 15000,
    show.legend = FALSE
  )
p4
```

Labels are random

Labels are positioned randomly, so they're different every time a plot is rendered

```
p4
p4
```

You can `set.seed()` at the top of your script to ensure reproducibility.

```r
set.seed(123)
p4
set.seed(123)
p4
```

Label just some points

You can label just select points

```r
cars <- c("Volvo 142E", "Merc 230")
ggplot(dat, aes(wt, mpg, label = ifelse(car %in% cars, car, ""))) +
  geom_point(color = "red") +
  geom_label_repel(min.segment.length = 0, nudge_x = 0.5, nudge_y = 3)
```

Wrap-up

Go through “where to find help” slides. Plug next workshop (our fall reproducibility series)

Ask if there are any other remaining `ggplot2`-related roadblocks people are experiencing and workshop it live.