Polishing your plots

Hadley Wickham

Assistant Professor / Dobelman Family Junior Chair

Department of Statistics / Rice University

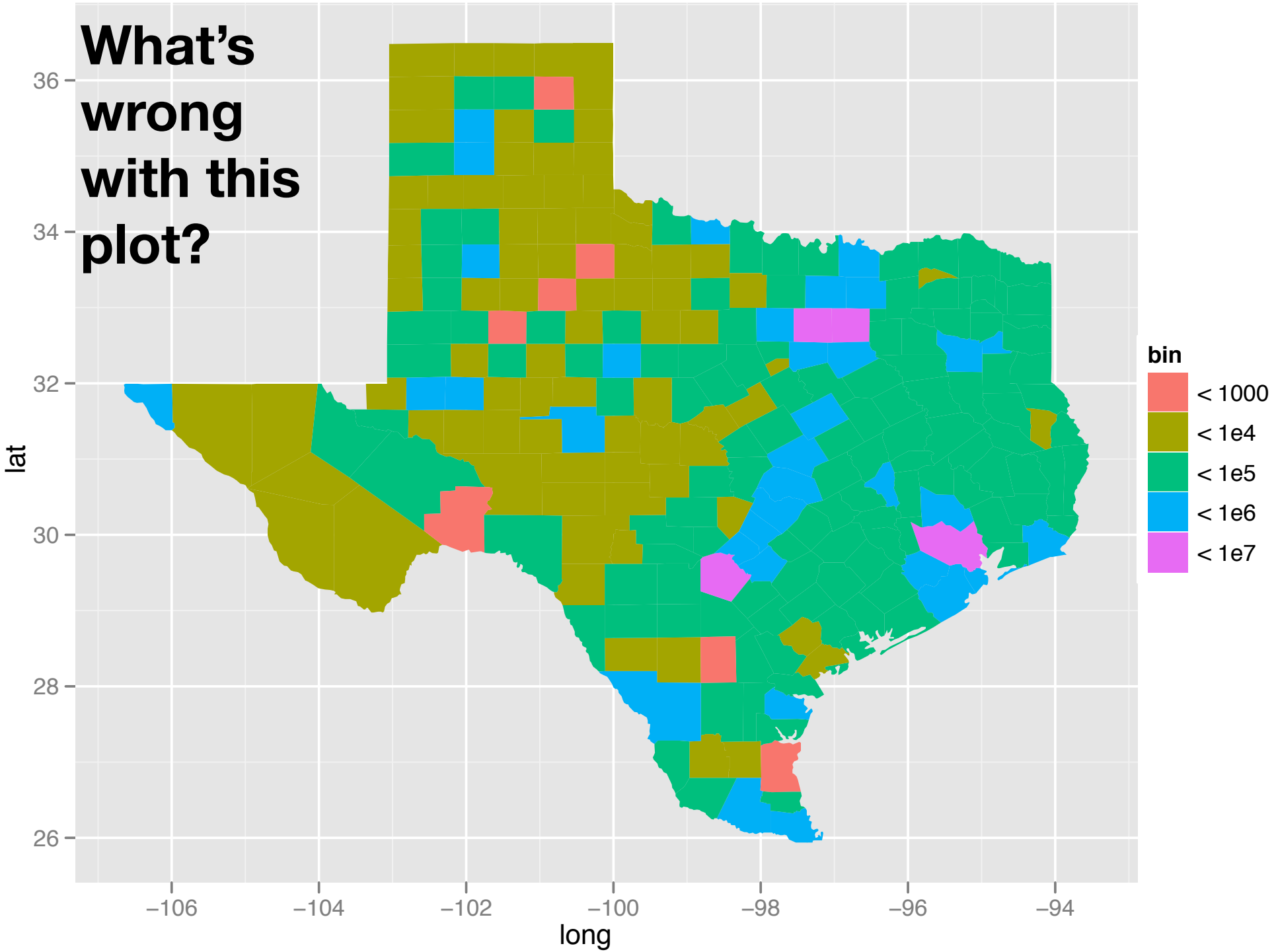
July 2010

cc BY NC

Communication graphics

When you need to **communicate** your findings, you need to spend a lot of time polishing your graphics to eliminate distractions and focus on the story.

Now it’s time to pay attention to the small stuff: labels, colour choices, tick marks...


What's wrong with this plot?
36
34
32
30
28
26
lat
−106
−104
−102
−100
−98
−96
−94
long
bin
< 1000
< 1e4
< 1e5
< 1e6
< 1e7

Some problems

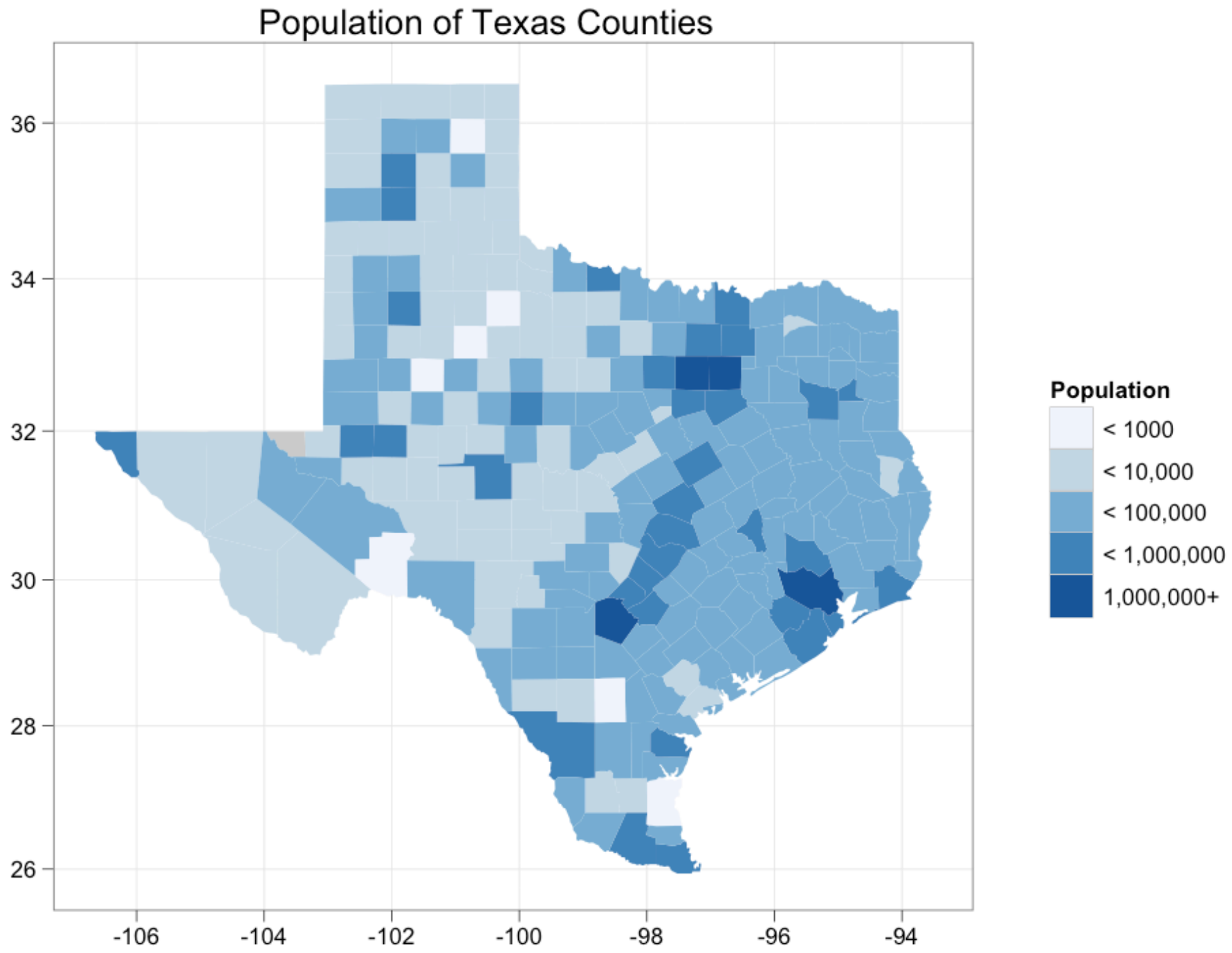
Incorrect coordinate system

Bad colour scheme

Unnecessary axis labels

Legend needs improvement: better title and better key labels

No title


Population of Texas Counties
36
34
32
30
28
26
-106
-104
-102
-100
-98
-96
-94
Population
< 1000
< 10,000
< 100,000
< 1,000,000
1,000,000+

1. **Scales**: used to override default perceptual mappings, and tune parameters of axes and legends.

2. **Themes**: control presentation of non-data elements.

3. **Saving your work**: to include in reports, presentations, etc.

Scales

Scales

Control how data is mapped to perceptual properties, and produce **guides** (axes and legends) which allow us to read the plot.

Important parameters: **name**, **breaks** & **labels**, **limits**.

Naming scheme: scale_*aesthetic*_*name*.
All default scales have name continuous or discrete.

```
# Default scales
scale_x_continuous()
scale_y_discrete()
scale_colour_discrete()

# Custom scales
scale_colour_hue()
scale_x_log10()
scale_fill_brewer()

# Scales with parameters
scale_x_continuous("X Label", limits = c(1, 10))
scale_colour_gradient(low = "blue", high = "red")
```

```
p <- qplot(cyl, displ, data = mpg)

# First argument (name) controls axis label
scale_y_continuous("Latitude")
scale_x_continuous("")

# Breaks and labels control tick marks
scale_x_continuous(breaks = -c(106,100,94))
scale_fill_discrete(labels = c("< 1000" = "< 1000",
  "< 1e4" = "< 10,000", "< 1e5" = "< 100,000",
  "< 1e6" = "< 1,000,000", "< 1e7" = "1,000,000+"))
scale_y_continuous(breaks = NA)

# Limits control range of data
scale_y_continuous(limits = c(26, 32))
# same as:
p + ylim(26, 32)
```

Your turn

Fix the axis and legend related problems that we have identified.

```
qplot(long, lat, data = choro, geom = "polygon", group = group, fill = bin) +
  scale_fill_discrete("Population", labels =
    c("< 1000" = "< 1000" , "< 1e4" = "< 10,000", "< 1e5" = "< 100,000",
    "< 1e6" = "< 1,000,000", "< 1e7" = "1,000,000+")) +
  scale_x_continuous("") +
  scale_y_continuous("") +
  coord_map()
```

Alternate scales

Can also override the default choice of scales. You are most likely to want to do this with **colour**, as it is the most important aesthetic after position.

Need a little background to be able to use colour effectively: colour **spaces** & colour **blindness**.

Colour spaces

Most familiar is **rgb**: defines colour as mixture of **red**, **green** and **blue**. Matches the physics of eye, but the brain does a lot of post-processing, so it's hard to directly perceive these components.

A more useful colour space is hcl: **hue**, **chroma** and **luminance**

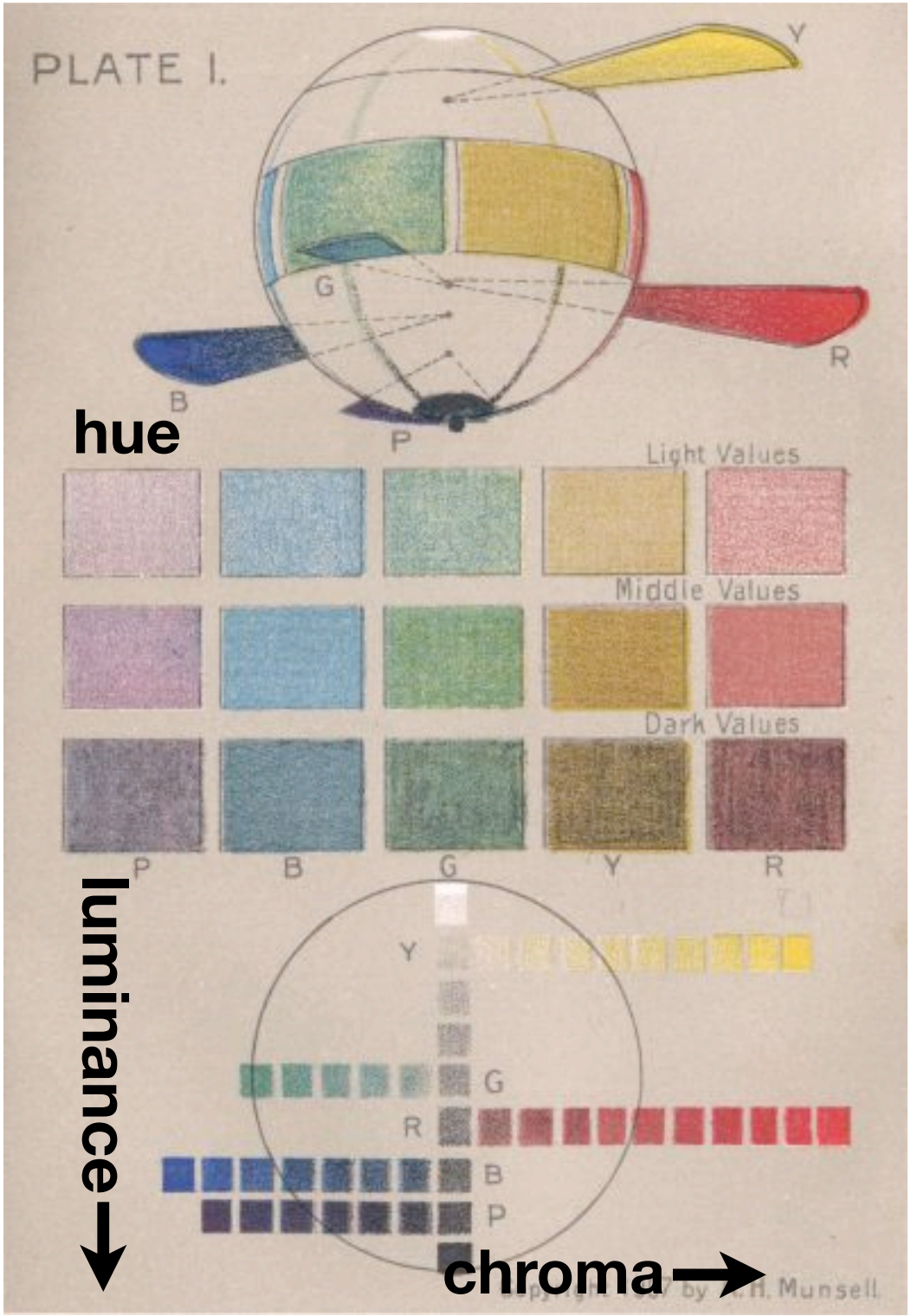


PLATE I.
Y
G
R
B
P
hue
Light Values
Middle Values
Dark Values
P
B
G
Y
R
luminance
Y
G
R
B
P
chroma
by A. H. Munsell.

Default colour scales

Discrete: evenly spaced hues of equal chroma and luminance. No colour appears more important than any other. Does not imply order.

Continuous: evenly spaced hues between two colours.

Colour blindness

7-10% of men are red-green colour “blind”. (Many other rarer types of colour blindness)

Solutions: avoid red-green contrasts; use redundant mappings; **test**. I like color oracle: http://colororacle.cartography.ch

Alternatives

Discrete: brewer, grey

Continuous: gradient2, gradientn

Your turn

Modify the fill scale to use a Brewer colour palette of your choice. (Hint: you will need to change the name of the scale)

Use `RColorBrewer::display.brewer.all` to list all palettes.

Themes

Visual appearance

So far have only discussed how to get the data displayed the way you want, focussing on the essence of the plot.

Themes give you a huge amount of control over the appearance of the plot, the choice of background colours, fonts and so on.

```
# Two built in themes.  The default:
qplot(carat, price, data = diamonds)

# And a theme with a white background:
qplot(carat, price, data = diamonds) + theme_bw()

# Use theme_set if you want it to apply to every
# future plot.
theme_set(theme_bw())

# This is the best way of seeing all the default
# options
theme_bw()
theme_grey()
```

Plot title

The plot theme also controls the plot title. You can change this for an individual plot by adding

```
opts(title = "My title")
```

Your turn

Add an informative title and see what the plot looks like with a white background.

Elements

You can also make your own theme, or modify and existing.

Themes are made up of elements which can be one of: `theme_line`, `theme_segment`, `theme_text`, `theme_rect`, `theme_blank`

Gives you a lot of control over plot appearance.

Elements

Axis: axis.line, axis.text.x, axis.text.y, axis.ticks, axis.title.x, axis.title.y

Legend: legend.background, legend.key, legend.text, legend.title

Panel: panel.background, panel.border, panel.grid.major, panel.grid.minor

Strip: strip.background, strip.text.x, strip.text.y

```
# To modify a plot
p + opts(plot.title =
  theme_text(size = 12, face = "bold"))
p + opts(plot.title = theme_text(colour = "red"))
p + opts(plot.title = theme_text(angle = 45))
p + opts(plot.title = theme_text(hjust = 1))
```

```
# If we want, we could also remove the axes:
last_plot() + opts(
  axis.text.x = theme_blank(),
  axis.text.y = theme_blank(),
  axis.title.x = theme_blank(),
  axis.title.y = theme_blank(),
  axis.ticks.length = unit(0, "cm"),
  axis.ticks.margin = unit(0, "cm"))
```

Saving your work

Raster	Vector
pixel-based	instruction-based
png	pdf
for plots with many points	for all other plots
ms office, web	latex

```
qplot(price, carat, data = diamonds)
ggsave("diamonds.png")

# Selects graphics device based on extension
ggsave("diamonds.png")
ggsave("diamonds.pdf")
```

```
# Uses on-screen device size, or override with
# width & height (to be reproducible)
ggsave("diamonds.png", width = 6, height = 6)

# Outputs last plot by default, override
# with plot:
dplot <- qplot(carat, price, data = diamonds)
ggsave("diamonds.png", plot = dplot)

# Defaults to 300 dpi for png
ggsave("diamonds.png", dpi = 72)
```

Your turn

Save a pdf of a scatterplot of price vs carat. Open it up in adobe acrobat.

Save a png of the same scatterplot and embed it into word.

This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 United States License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/us/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.