Polishing plots for communication

`06-Polishing-plots.Rmd`

Solution for yesterday's problem

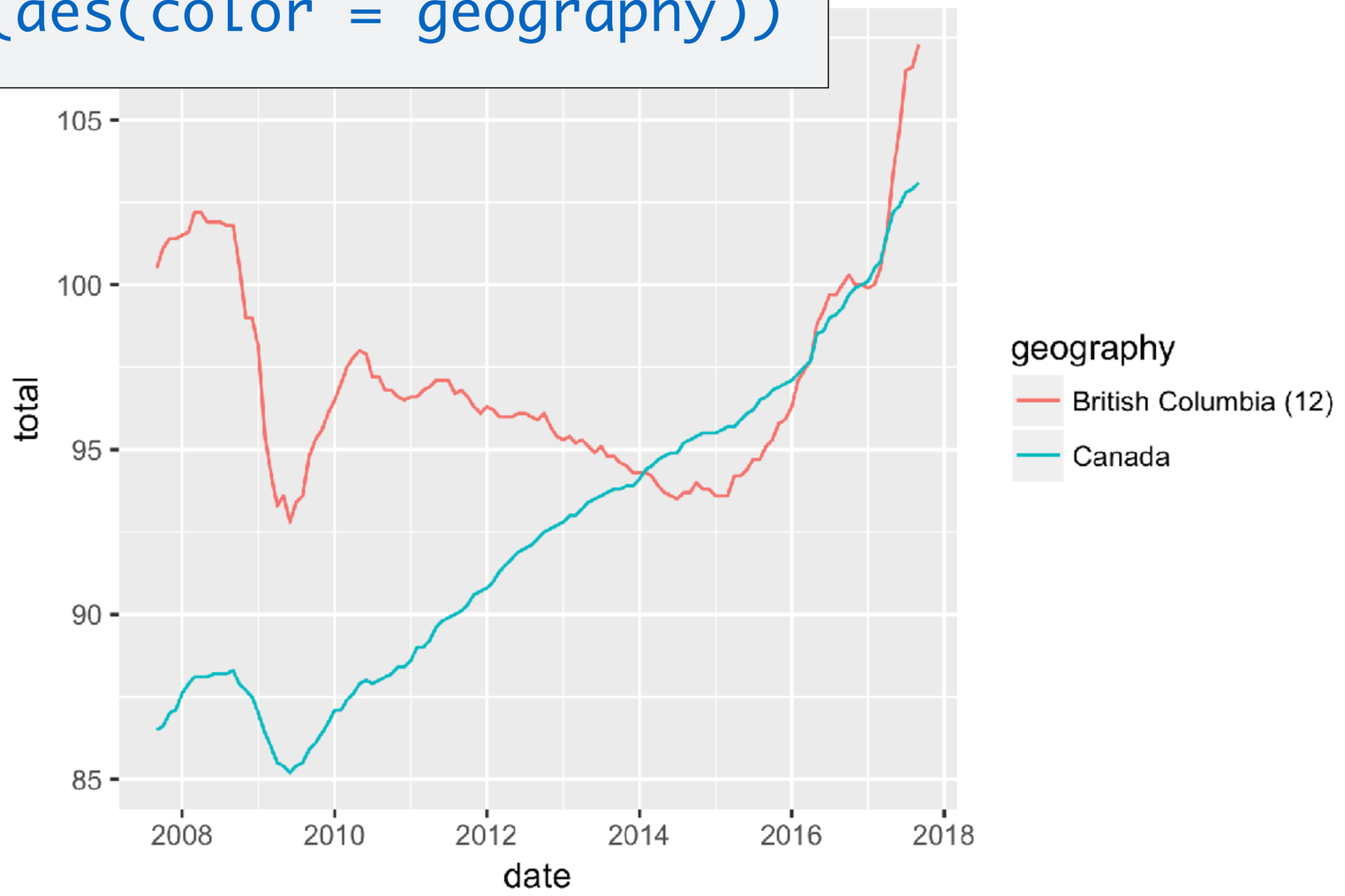
```
housing_reshaped <- housing_raw %>%
  gather(key = "time", value = "hpi",
    -Geography, -`New housing price indexes`) %>%
  spread(`New housing price indexes`, hpi)

housing_bc <- housing_reshaped %>%
  mutate(date = lubridate::parse_date_time(time, order = "my")) %>%
  filter(Geography %in% c("Canada", "British Columbia (12)")) %>%
  rename(total = `Total (house and land)`,
         geography = Geography)
```

To export the data:

```
write_csv(housing_bc, "housing.csv")
```

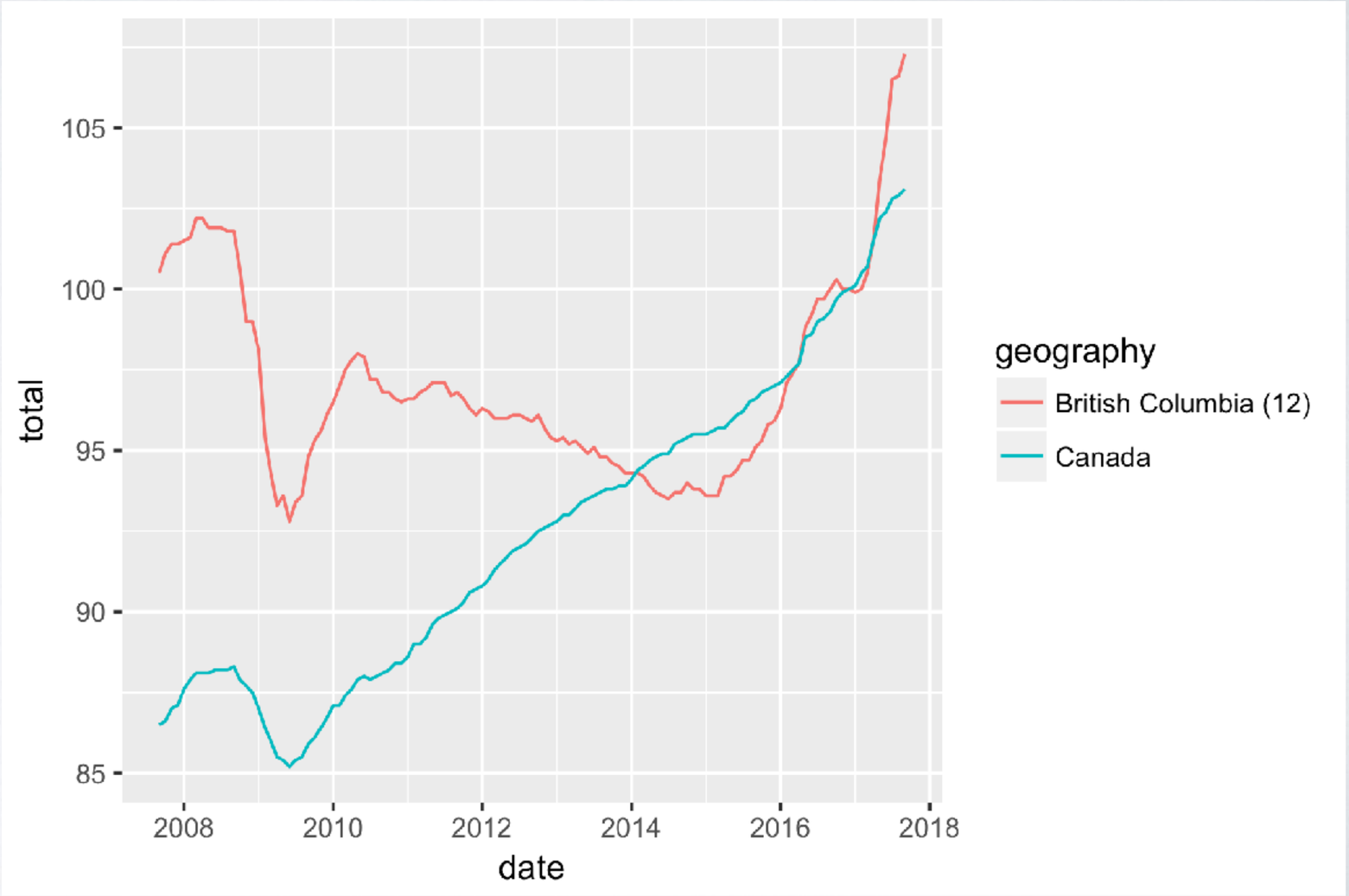
```
housing_bc %>%
  ggplot(aes(date, total)) +
    geom_line(aes(color = geography))
```



Your Turn 1

Discuss with your neighbours:

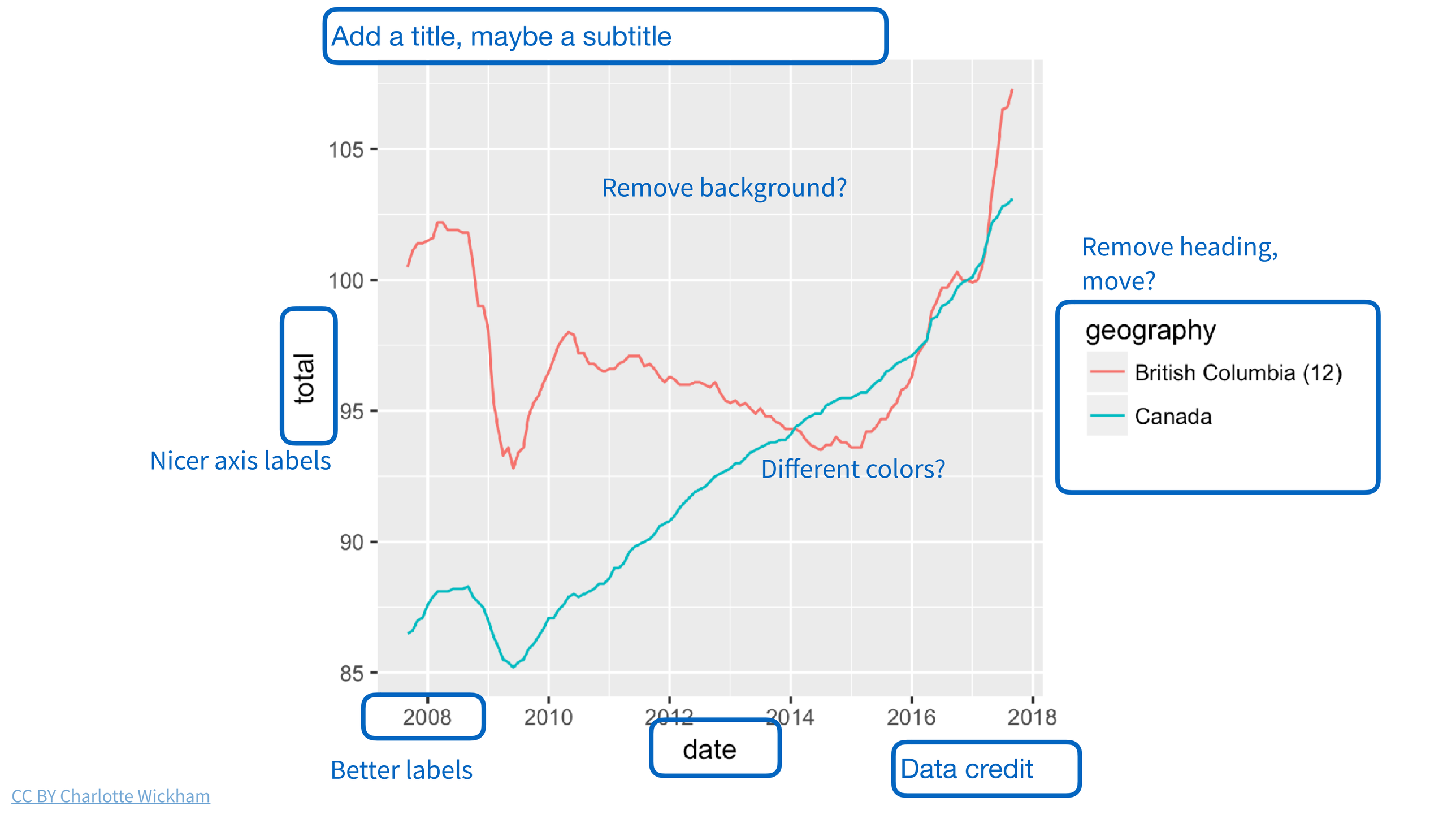
What would you want to change before publishing this plot?



Assigning the result of the plot to a variable

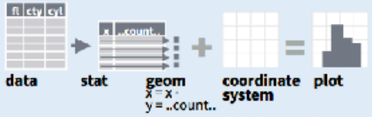
```
basic_plot <- housing %>%
  filter(geography %in% c("Canada", "British Columbia (12)")) %>%
  ggplot(mapping = aes(x = date, y = total)) +
    geom_line(aes(color = geography))
basic_plot
```

Asking for the variable, displays the plot



Stats An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

geom to use | stat function | geommappings

i + stat_density2d(aes(fill = ..level..), geom = "polygon")

variable created by stat

c + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y,** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins=30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5) **x, y** | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area") **x, y** | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se=T, level=0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) **sample, x, y** | ..sample.., ..theoretical..

e + stat_sum() **x, y, size** | ..n.., ..prop..

e + stat_summary(fun.data = "mean_cl_boot")

h + stat_summary_bin(fun.y = "mean", geom = "bar")

e + stat_unique()

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

(n <- d + geom_bar(aes(fill = fl)))

scale | aesthetic to adjust | prepackaged scale to use | scale-specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

GENERAL PURPOSE SCALES

Use with most aesthetics

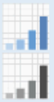
scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d", date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

COLOR AND FILL SCALES (DISCRETE)



n <- d + geom_bar(aes(fill = fl))

n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

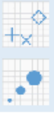
o + scale_fill_distiller(palette = "Blues")

o + scale_fill_gradient(low="red", high="yellow")

o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)

o + scale_fill_gradientn(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES



p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7))

p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartes an coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

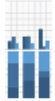
r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation=c(41, -74, 0))projection, brienztation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.



s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height

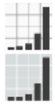
e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

e + geom_label(position = "nudge")
Nudge labels away from points

s + geom_bar(position = "stack")
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments
s + geom_bar(position = position_dodge(width = 1))

Themes



r + theme_bw()
White background with grid lines

r + theme_gray()
Grey background (default theme)

r + theme_dark()
dark for contrast

r + theme_classic()
r + theme_light()
r + theme_linedraw()
r + theme_minimal()
Minimal themes
r + theme_void()
Empty theme

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(. ~ fl)
facet into columns based on fl

t + facet_grid(year ~ .)
facet into rows based on year

t + facet_grid(year ~ fl)
facet into both rows and columns

t + facet_wrap(~ fl)
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(drv ~ fl, scales = "free")
x and y axis limits adjust to individual facets
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(. ~ fl, labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(fl ~ ., labeller = label_bquote(alpha ^ .(fl)))

α^{c}	α^{d}	α^{e}	α^{p}	α^{r}

t + facet_grid(. ~ fl, labeller = label_parsed)

c	d	e	p	r

Labels

t + labs(**x** = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
<AES> = "New <AES> legend title")

Use scale functions to update legend labels

t + annotate(geom = "text", x = 8, y = 9, label = "A")

geom to place | manual values for geom's aesthetics

Legends

n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"

n + guides(fill = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

n + scale_fill_discrete(name = "Title",
labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming

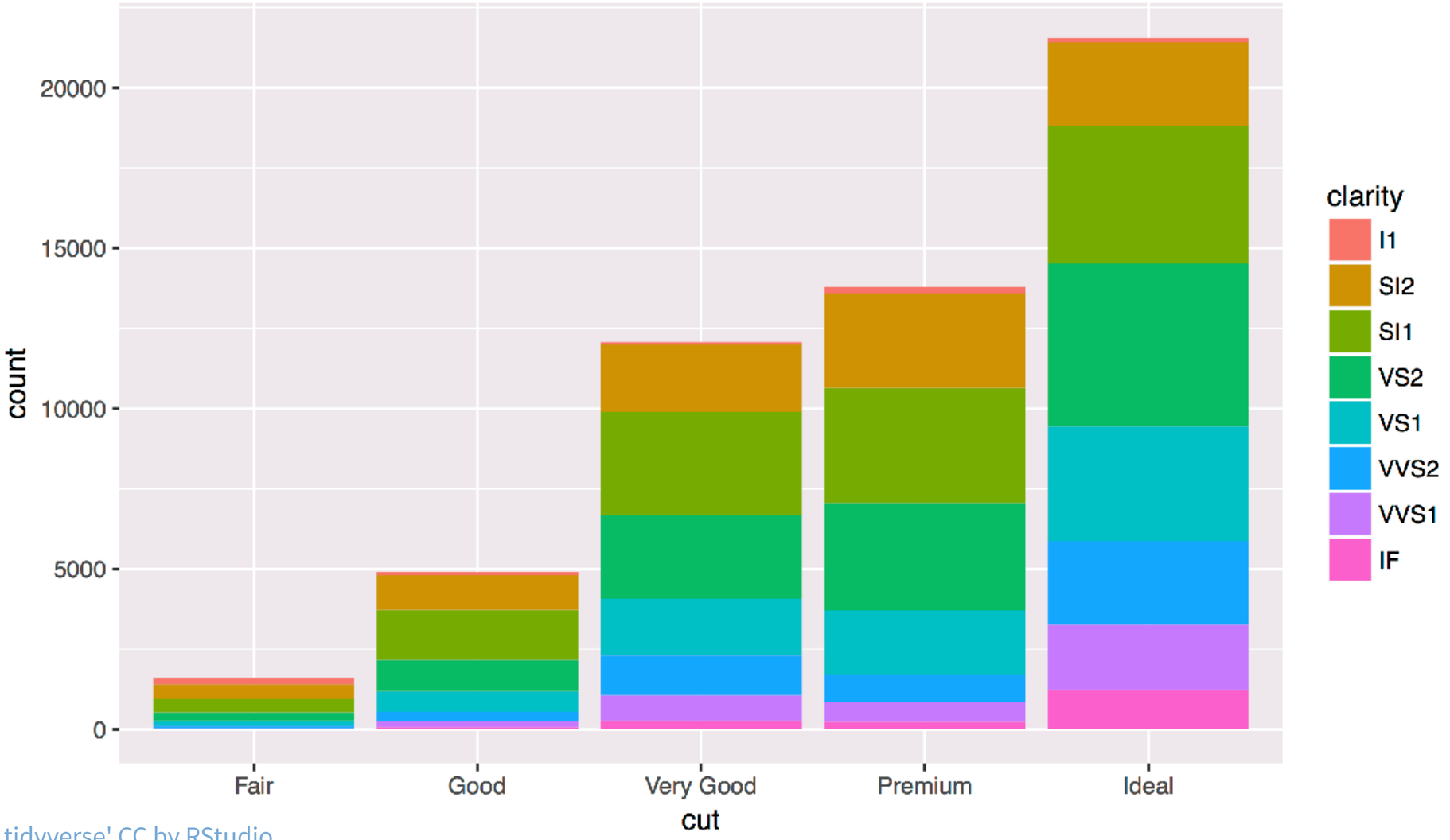
Without clipping (preferred)
t + coord_cartesian(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)
t + xlim(0, 100) + **ylim**(10, 20)
t + scale_x_continuous(limits = c(0, 100)) +
scale_y_continuous(limits = c(0, 100))

RStudio

Labels

```
diamonds_plot <- ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity))
```



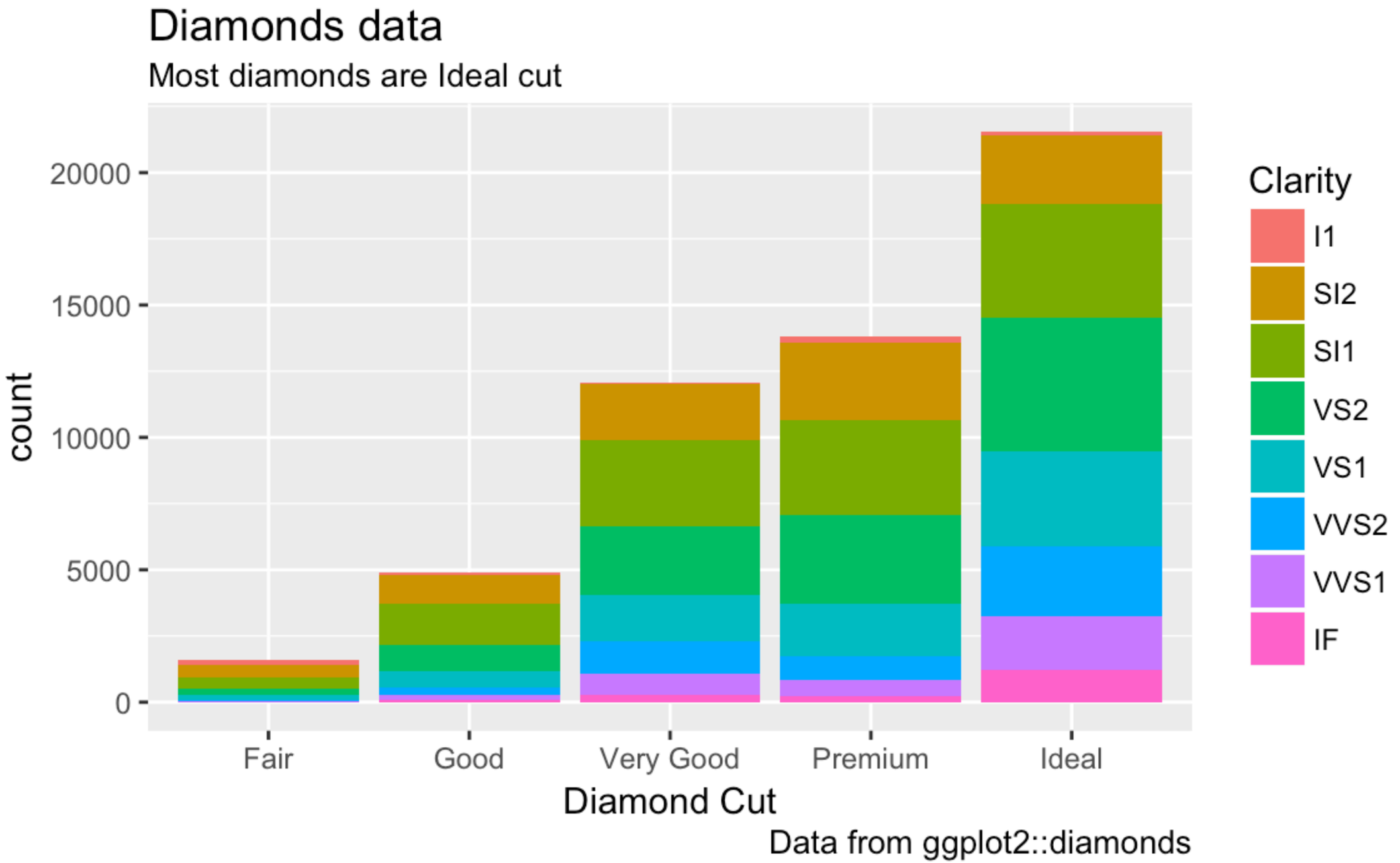
+ labs()

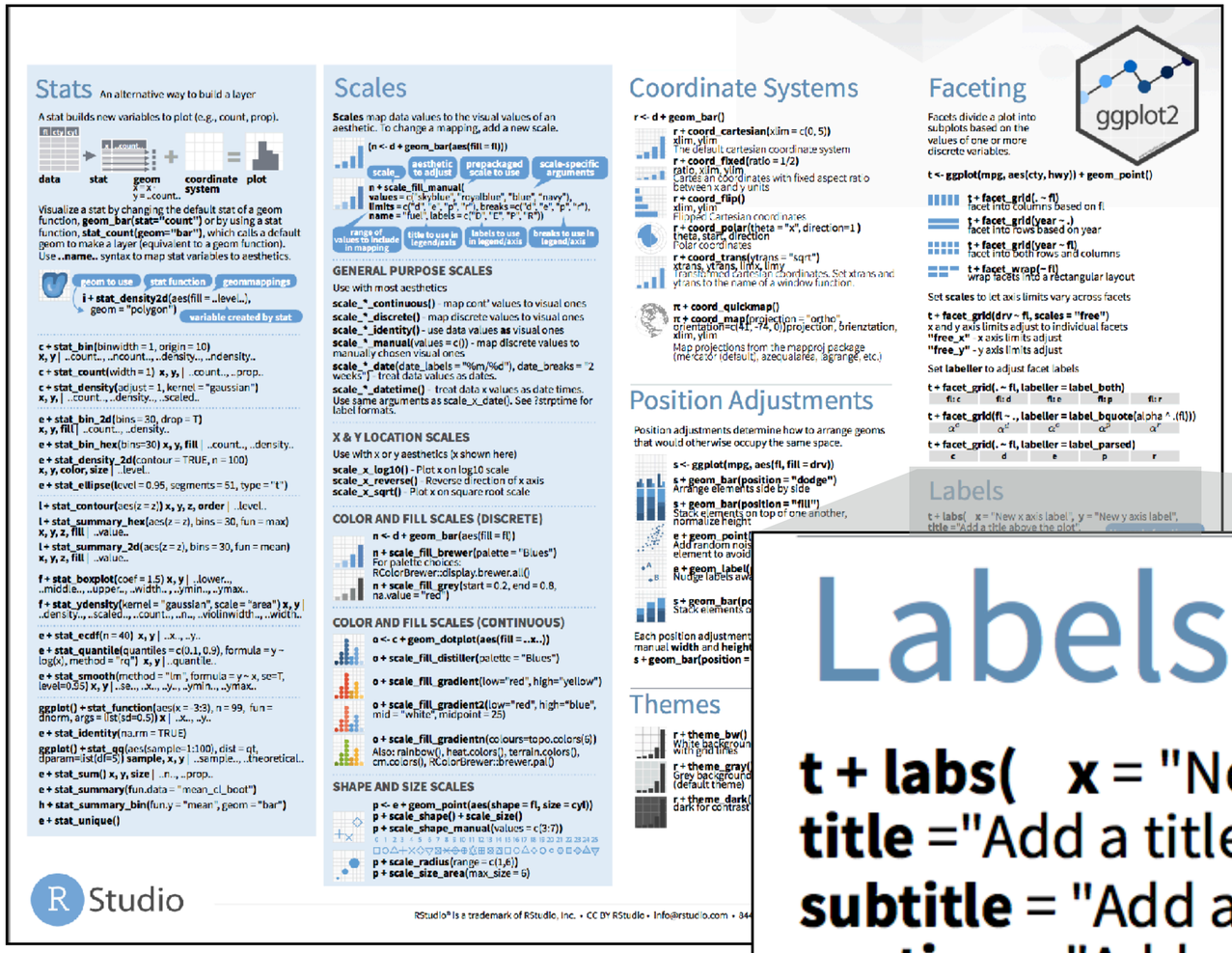
Our original plot

```
diamonds_plot +
  labs(title = "Diamonds data",
       subtitle = "Most diamonds are Ideal cut",
       caption = "Data from ggplot2::diamonds",
       x = "Diamond Cut",
       fill = "Clarity"
  )
```

title, subtitle, caption

Labels for other scales





Labels

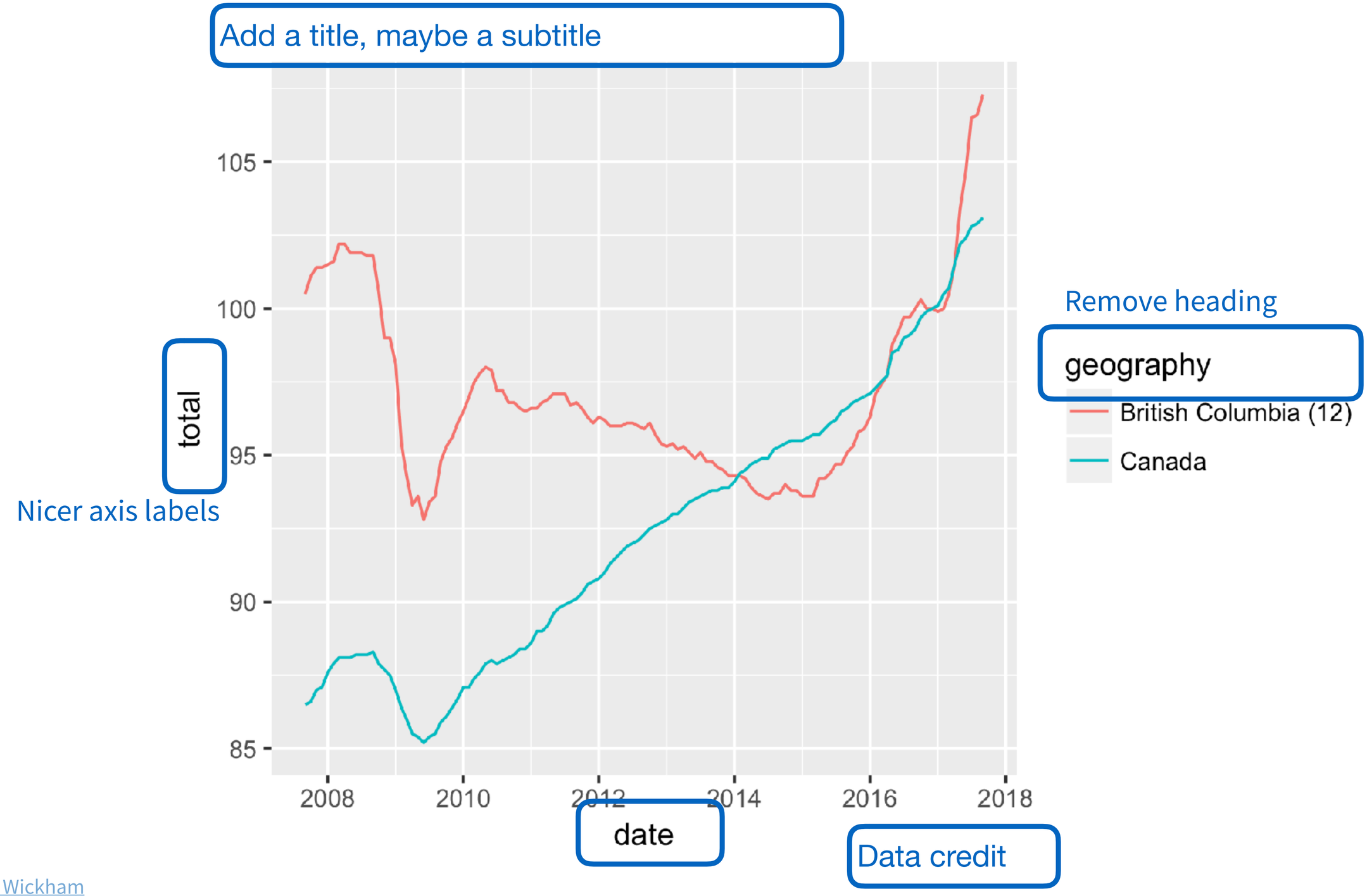
t + labs(**x** = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
<AES> = "New **<AES>** legend title")

Use scale functions to update legend labels

t + annotate(geom = "text", x = 8, y = 9, label = "A")

geom to place

manual values for geom's aesthetics



Your Turn 2

Edit the code to add appropriate labelling to the house price plot.

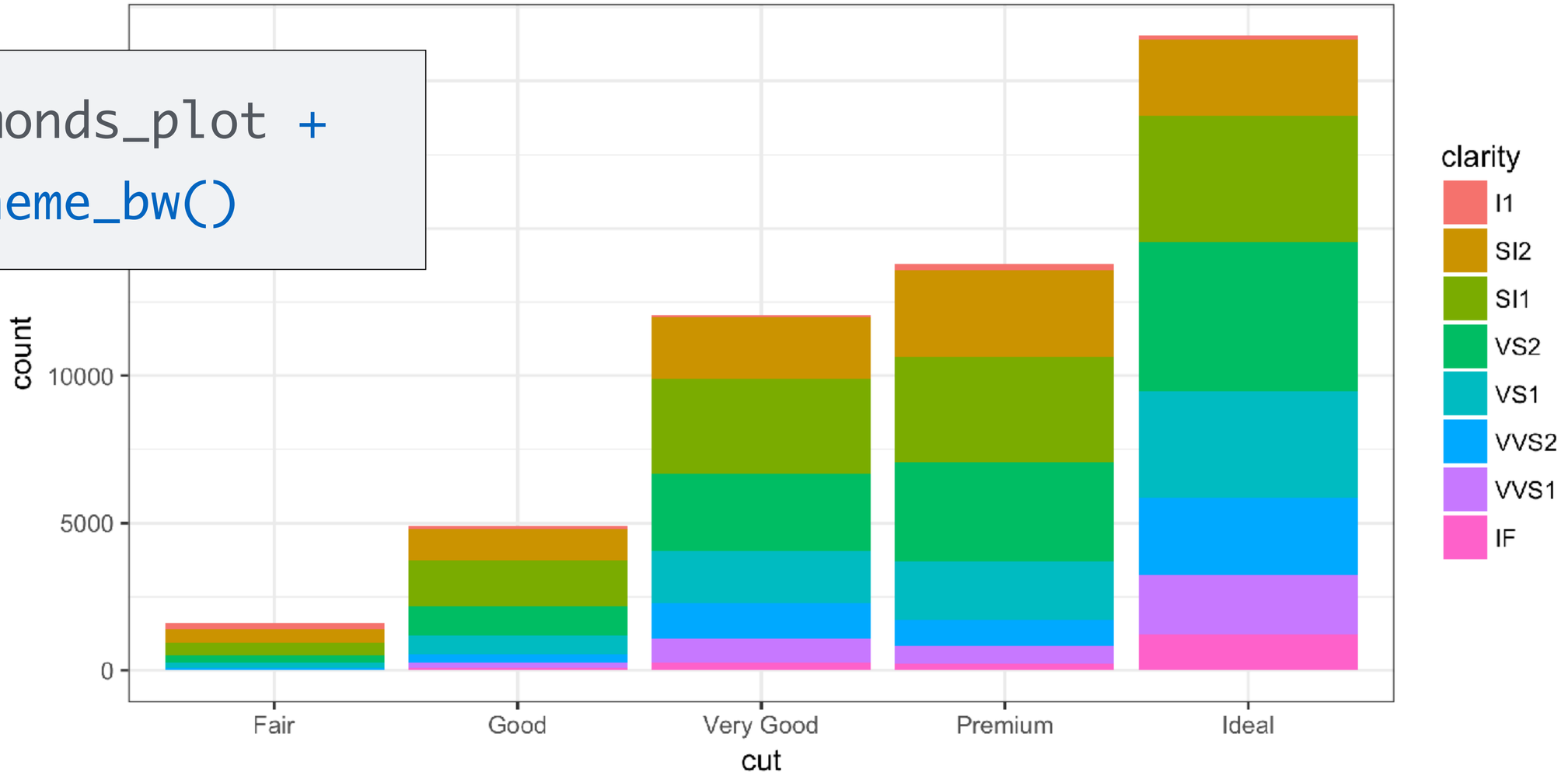
```
basic_plot +
  labs(title = "New Housing Price Index, Canada and B.C.",
       subtitle = "Total (house and land)",
       x = "",
       y = "NHPI (December 2016 = 100)",
       color = "",
       caption = "Source: Statistics Canada CANSIM table
327-0056"
  )
```

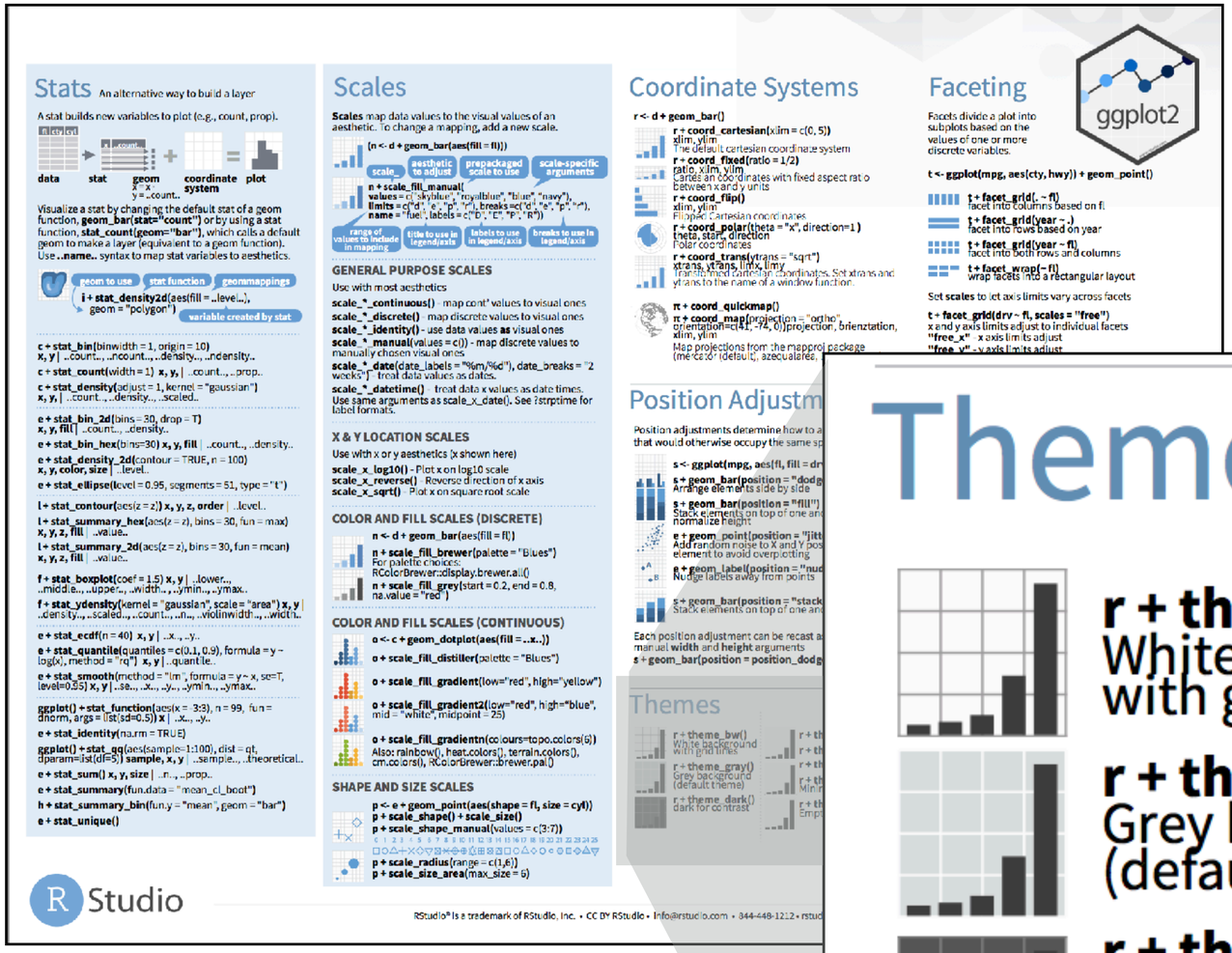
Themes

Themes

Visual appearance of non-data elements

```
diamonds_plot +
  theme_bw()
```

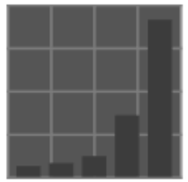




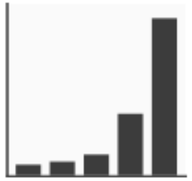
Themes

r + theme_bw()
White background with grid lines

r + theme_gray()
Grey background (default theme)

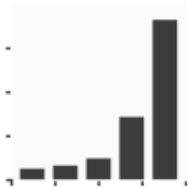


r + theme_dark()
dark for contrast



r + theme_classic()

r + theme_light()



r + theme_linedraw()

r + theme_minimal()
Minimal themes

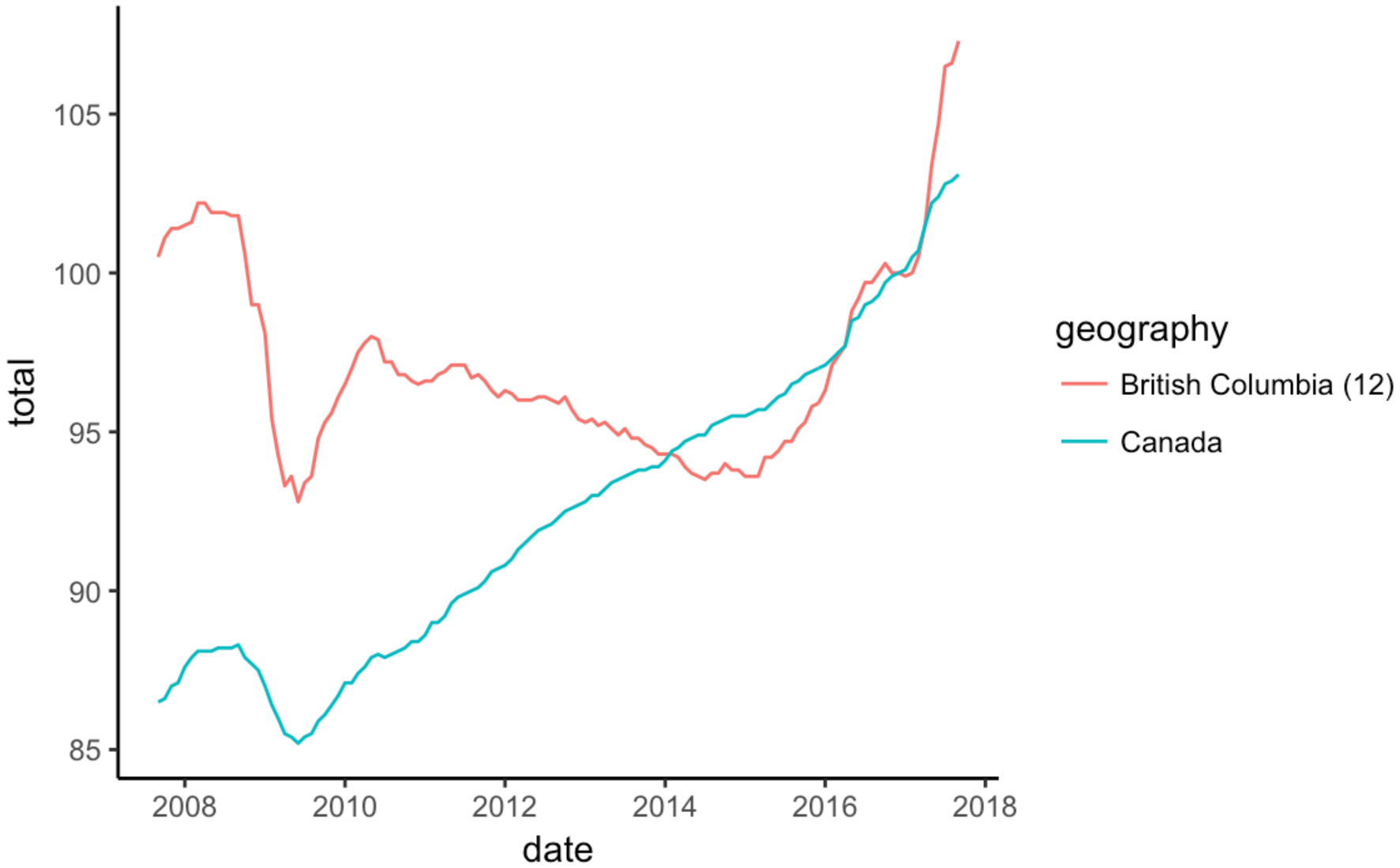
r + theme_void()
Empty theme

[Adapted from 'Master the tidyverse' CC by RStudio]

Your Turn 3

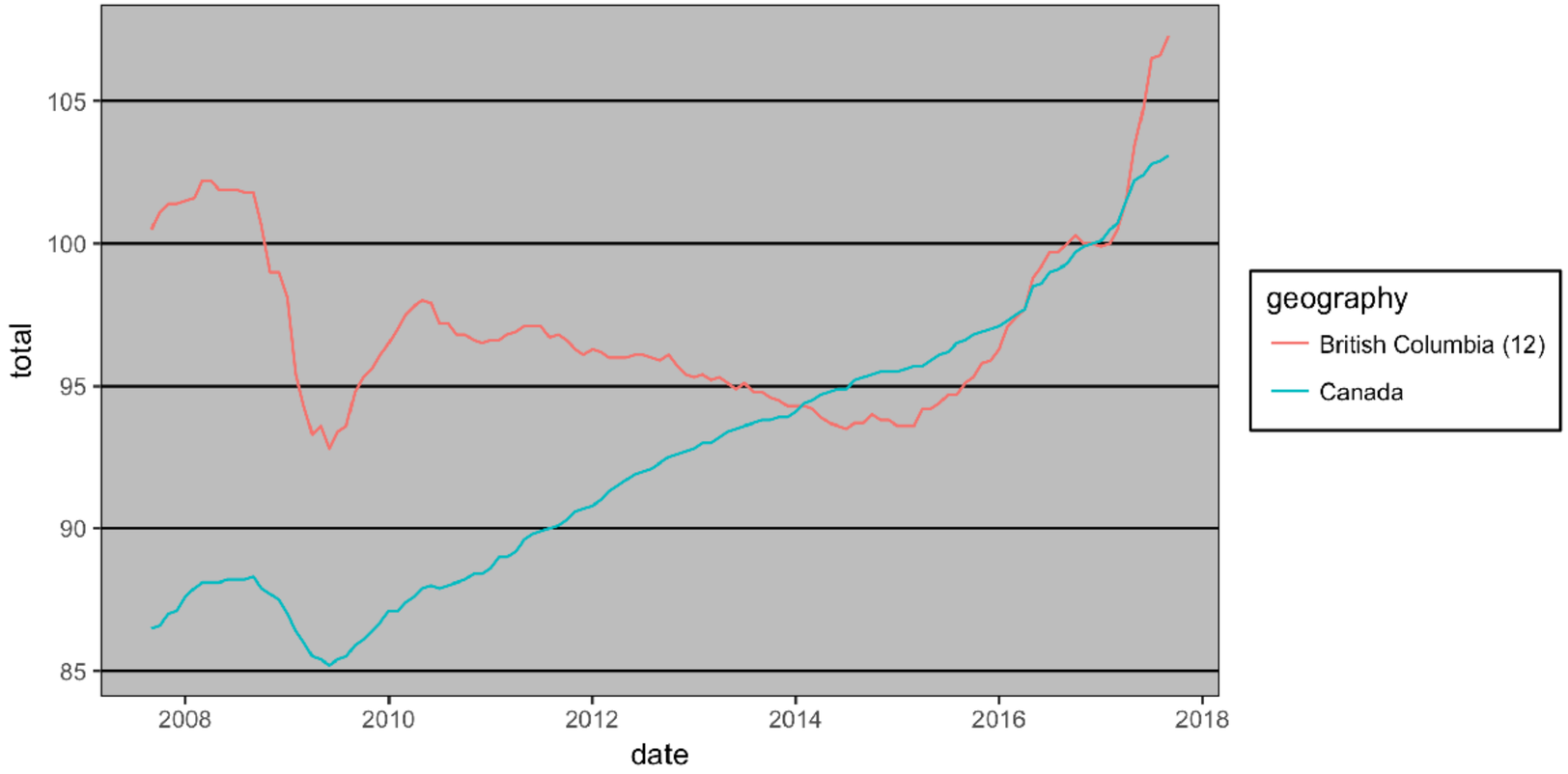
Add a theme to `basic_plot`. Try a few and pick one you like.

```
basic_plot + theme_classic()
```

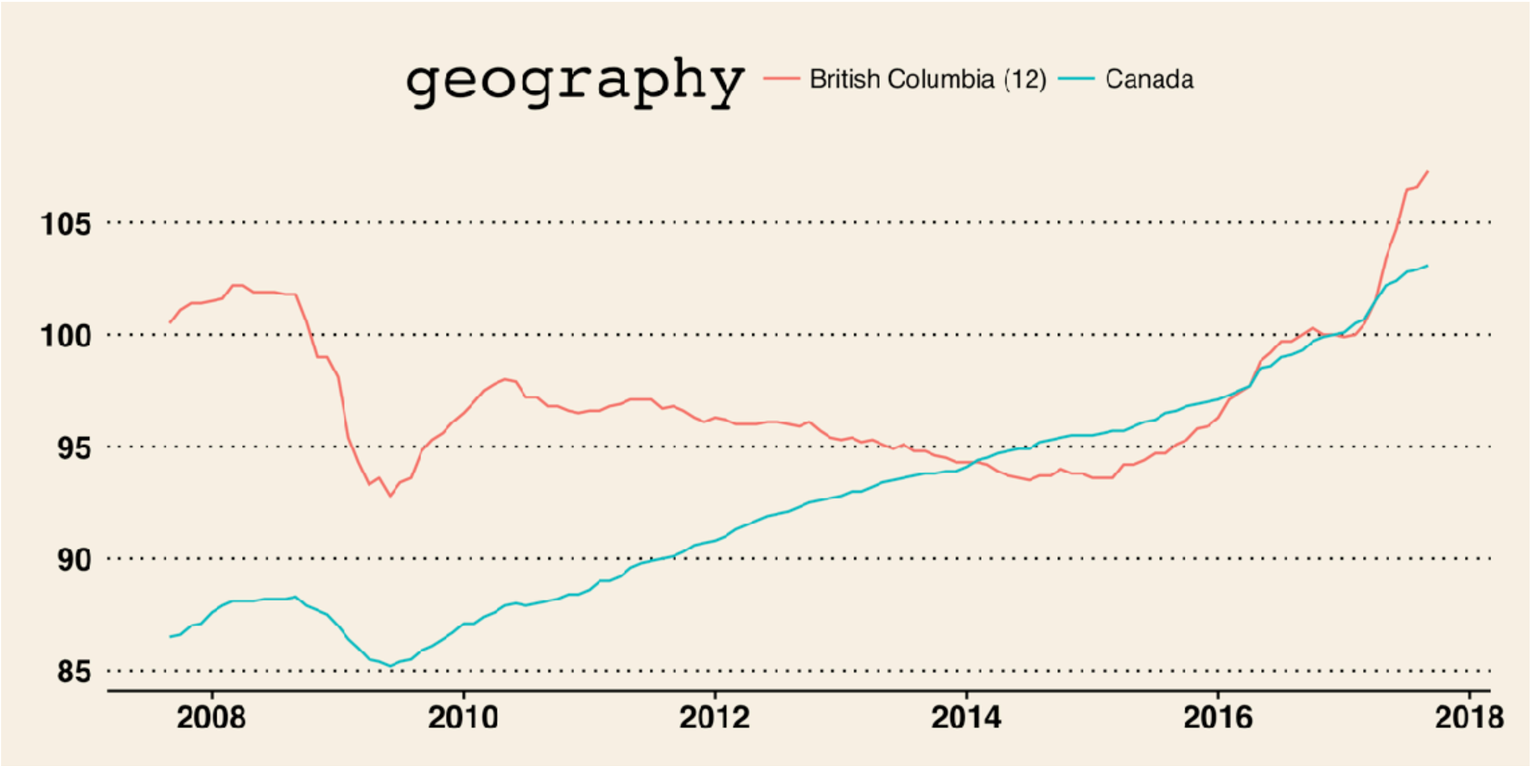


```
# install.packages("ggthemes")
library(ggthemes)
basic_plot +
  theme_excel()
```

Even more themes



```
# install.packages("ggthemes")
library(ggthemes)
basic_plot +
  theme_wsj()
```

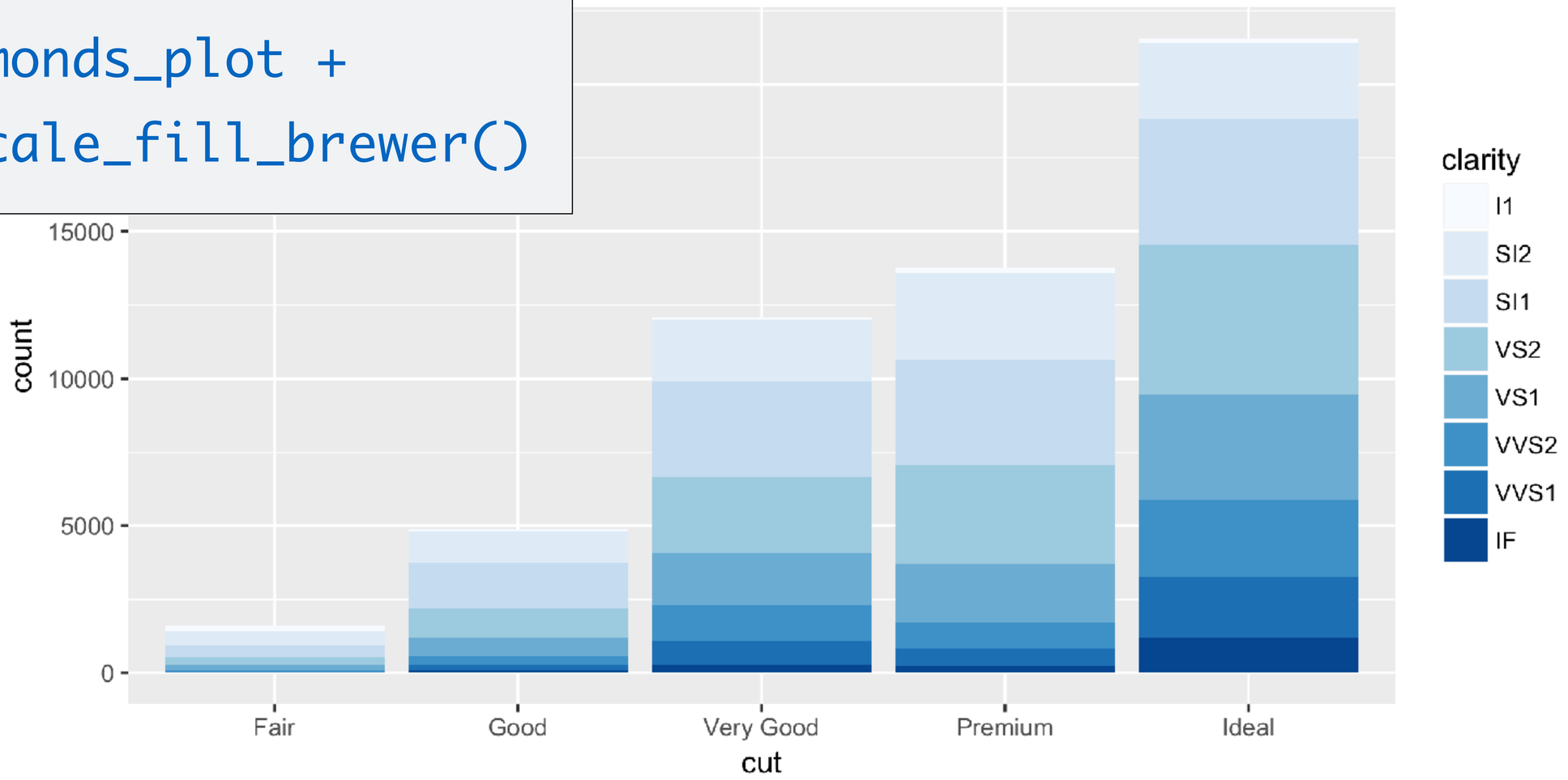


Scales

Scales

Customize color scales, other mappings

```
diamonds_plot +
  scale_fill_brewer()
```



Scale functions

```
diamonds_plot +
   scale_fill_brewer()
```

All start with `scale_`

Aesthetic the scale applies to

A specific kind of scale

Other arguments to the scale

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

(n <- d + geom_bar(aes(fill = fl)))

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

scale_ — aesthetic to adjust — prepackaged scale to use — scale-specific arguments

range of values to include in mapping — title to use in legend/axis — labels to use in legend/axis — breaks to use in legend/axis

GENERAL PURPOSE SCALES

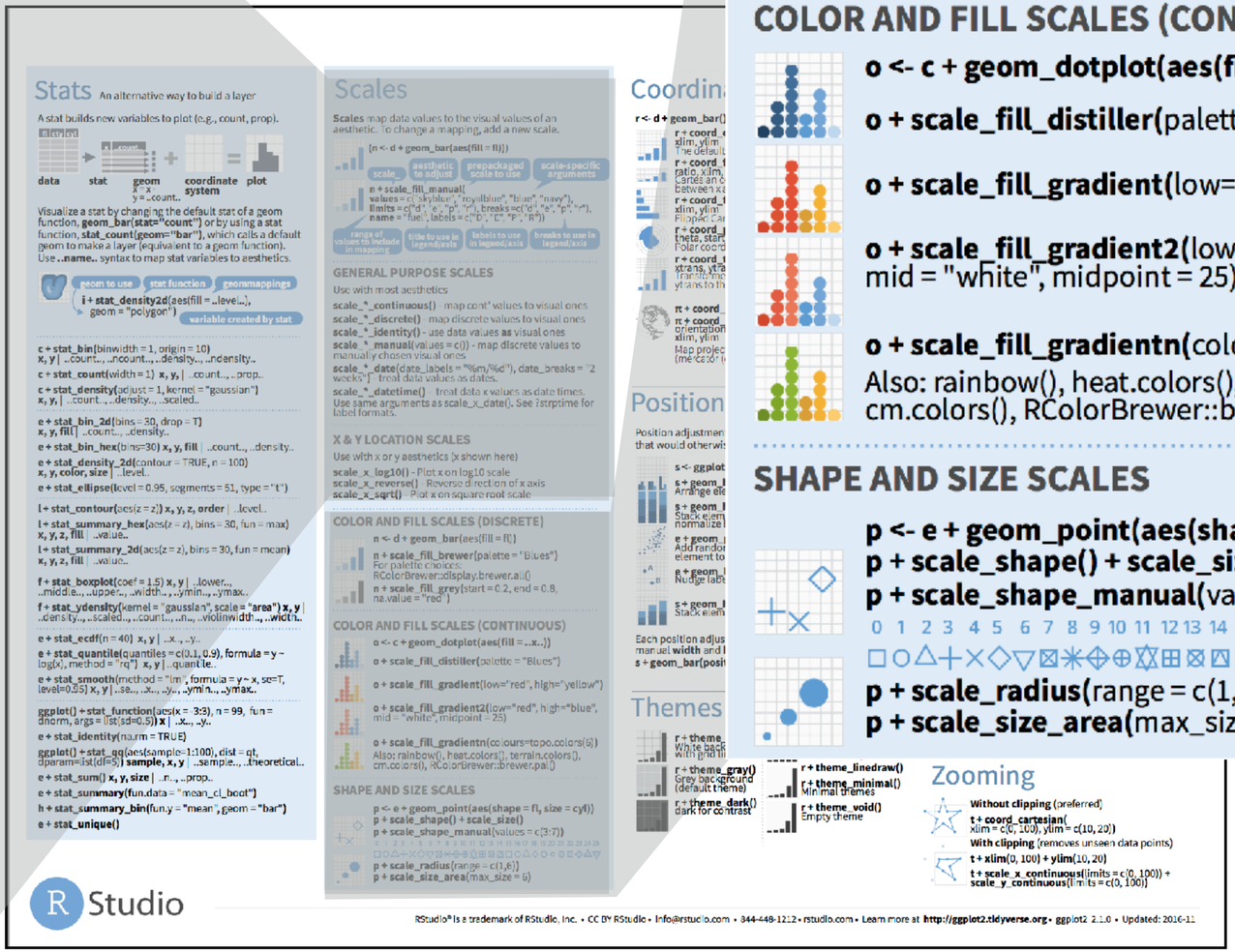
Use with most aesthetics

scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale



COLOR AND FILL SCALES (DISCRETE)

n <- d + geom_bar(aes(fill = fl))

n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

o + scale_fill_distiller(palette = "Blues")

o + scale_fill_gradient(low="red", high="yellow")

o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)

o + scale_fill_gradientn(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7))

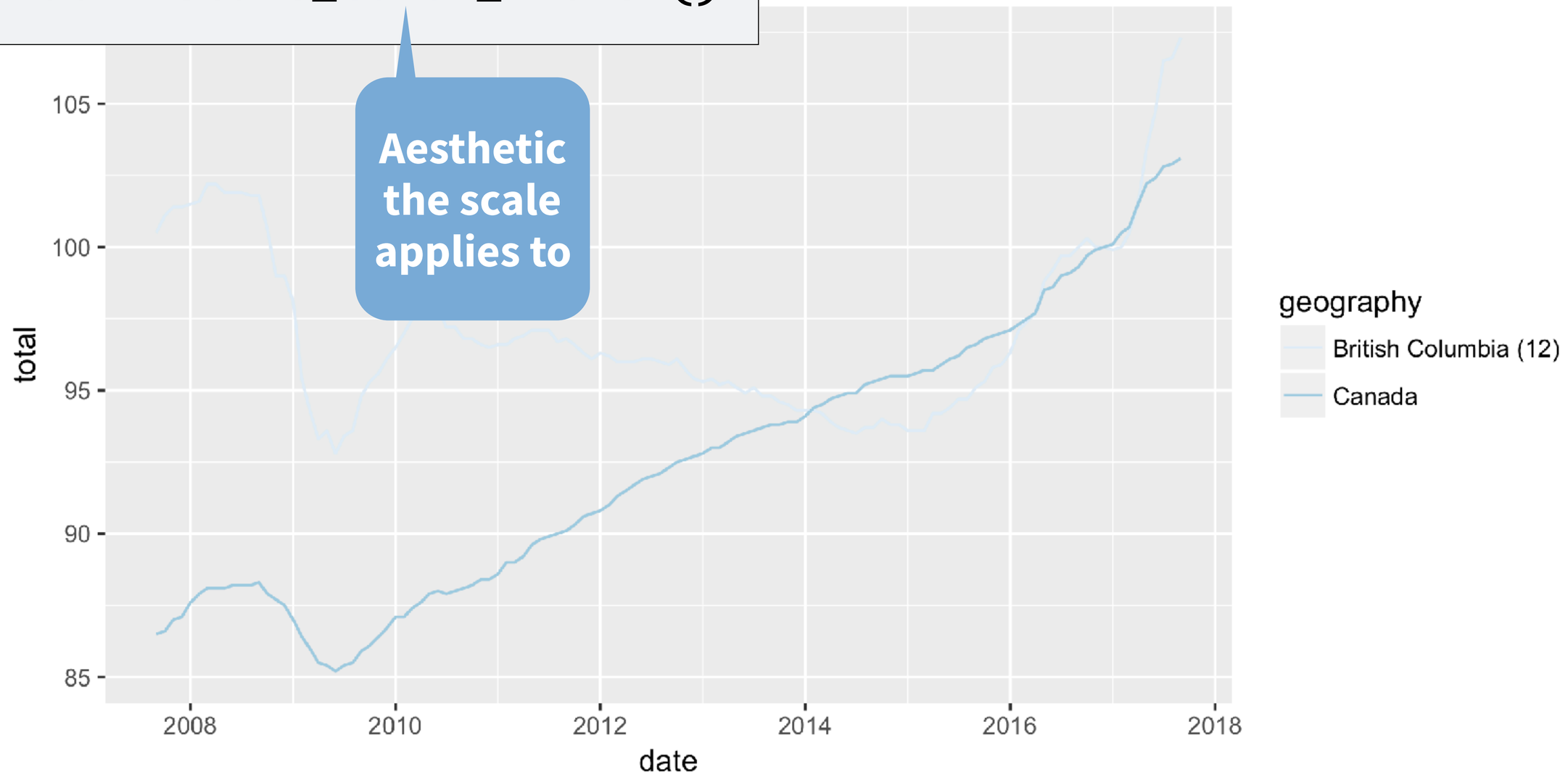
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

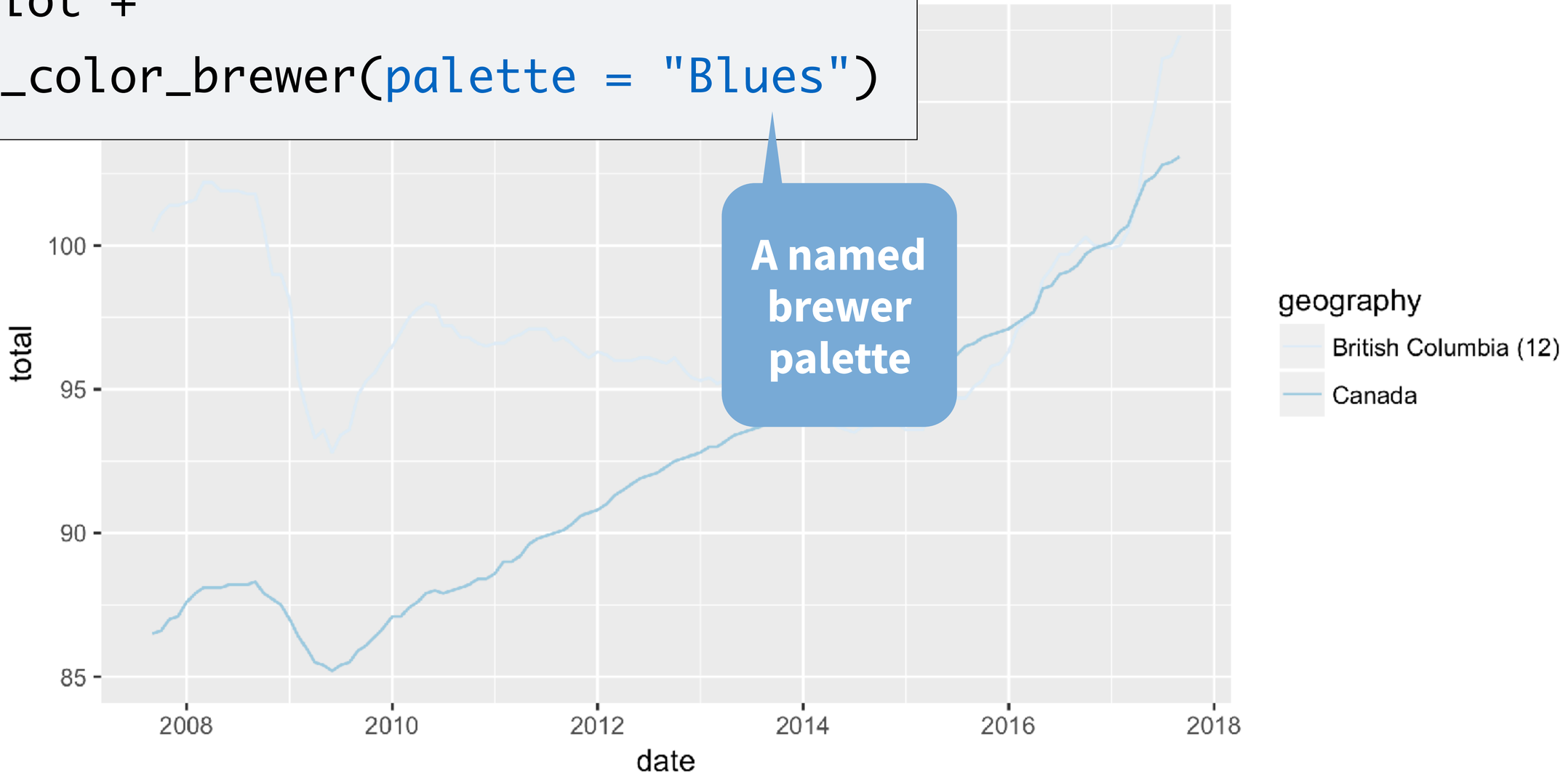
Your Turn 4

Add a brewer scale to the basic_plot.

```
basic_plot + scale_color_brewer()
```

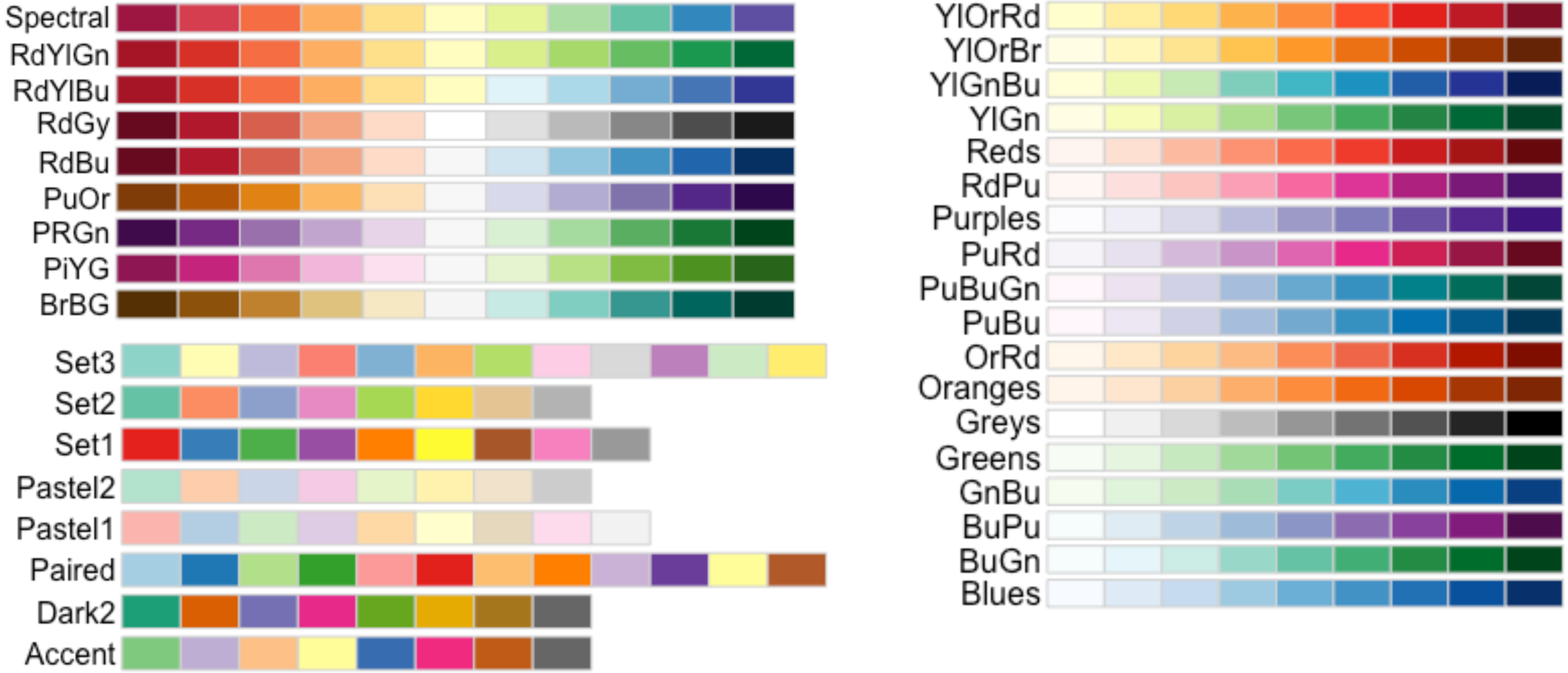


```
basic_plot +
  scale_color_brewer(palette = "Blues")
```

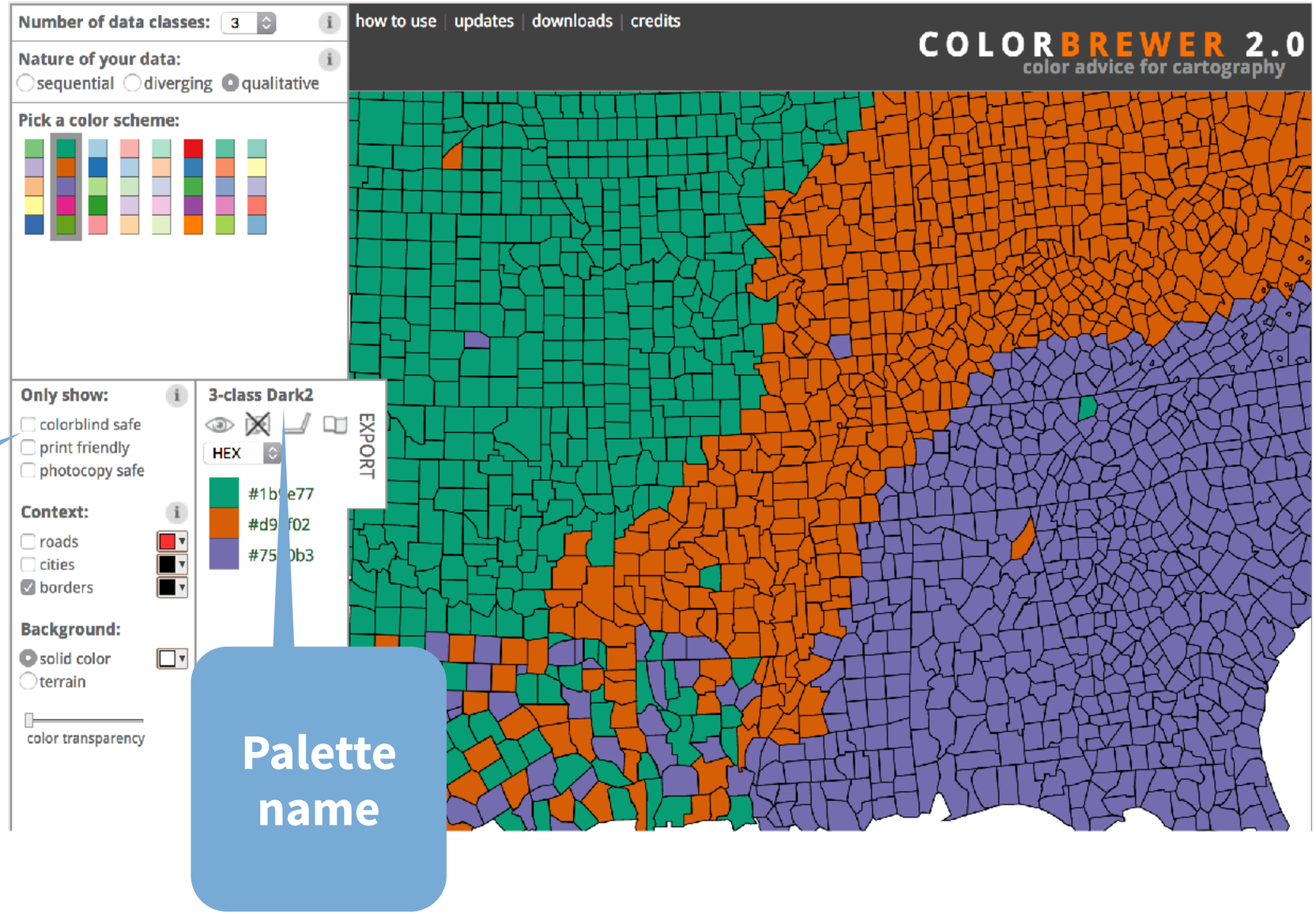


To see available brewer palettes

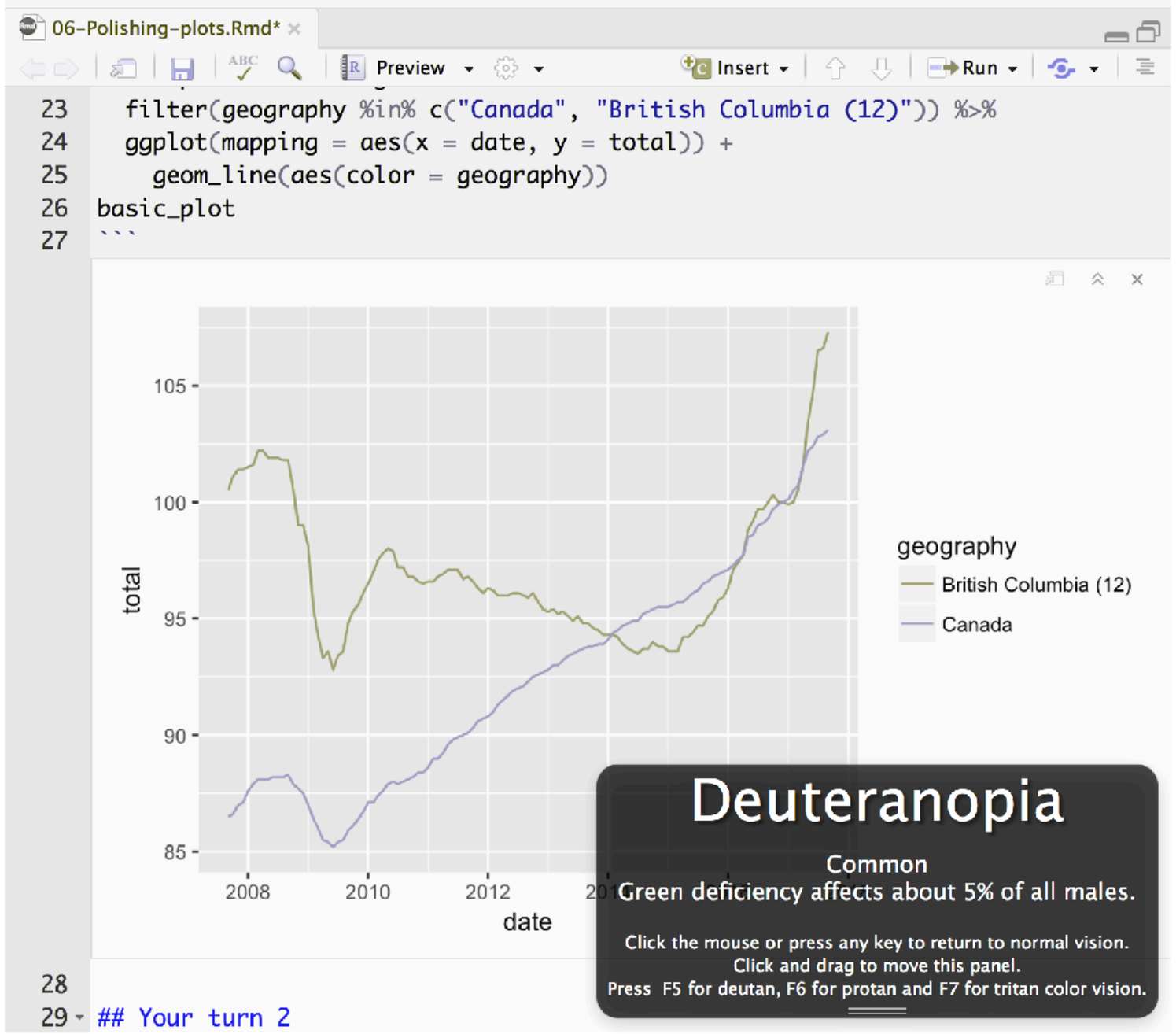
```
RColorBrewer::display.brewer.all()
```



http://colorbrewer2.org/



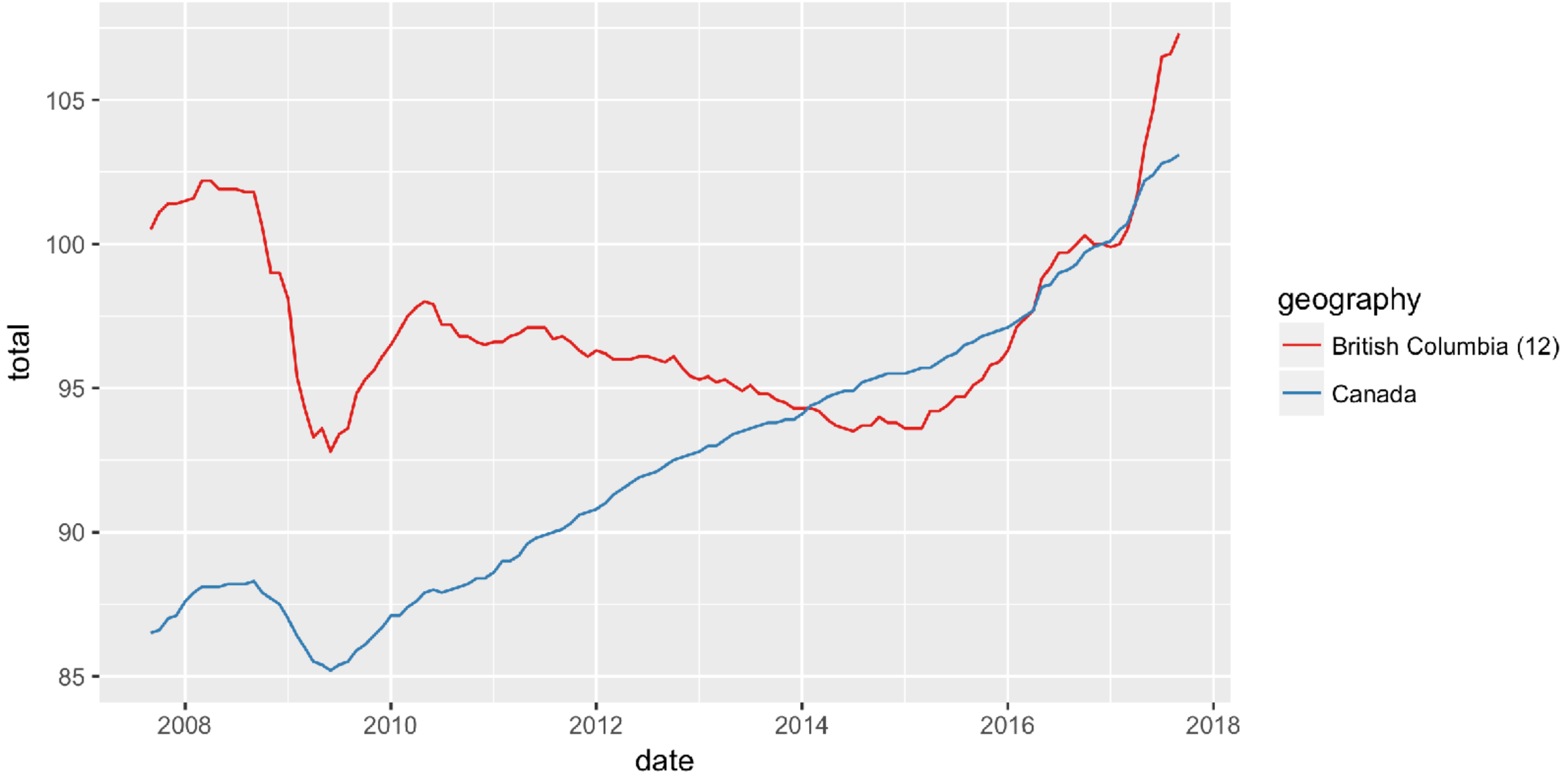
http://colororacle.org/



Your Turn 5

Choose a better color palette for the brewer scale.

```
basic_plot + scale_color_brewer(palette = "Set1")
```



scale_color_manual()

An way to completely control mapping

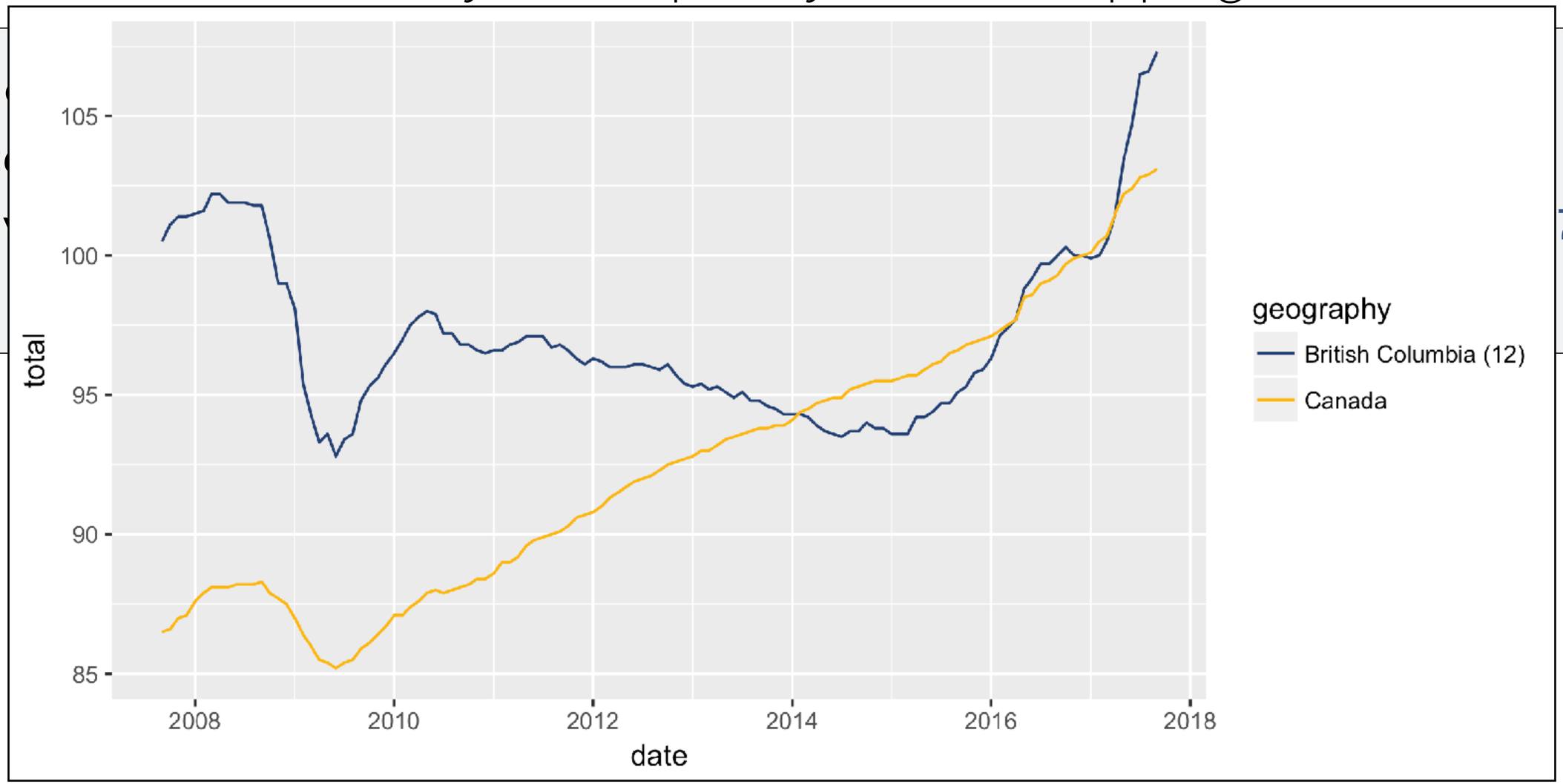
```
basic_plot +
  scale_color_manual(
    labels = c("Canada", "British Columbia"),
    values = c("Canada" = "#fdb913", "British Columbia (12)" = "#234075")
  )
```

scale_color_manual()

An way to completely control mapping

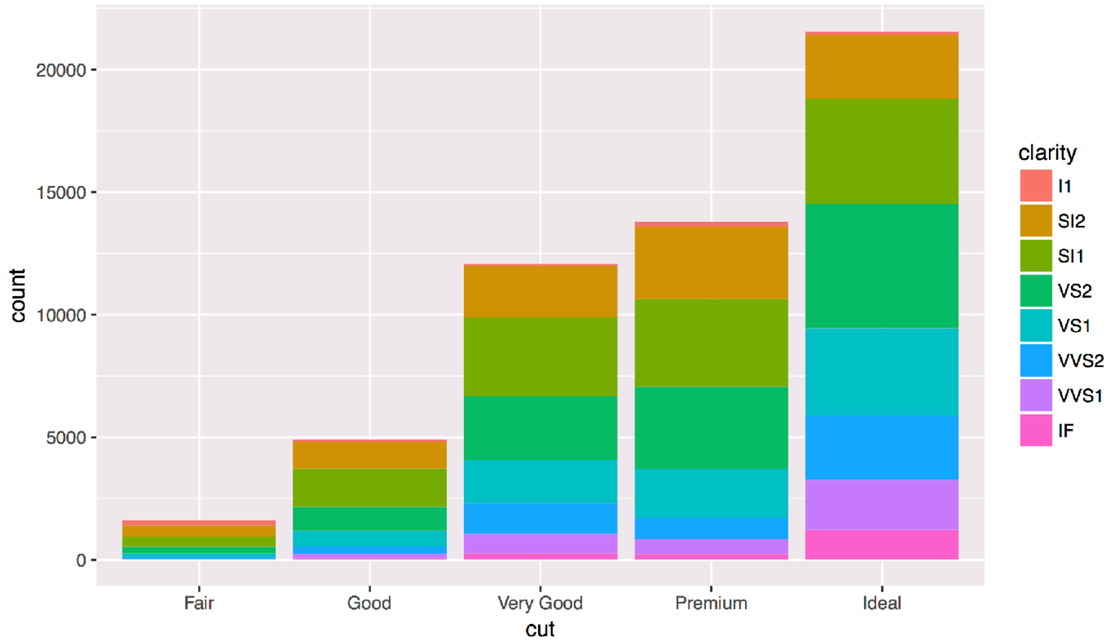
```
basi
  sc
    )
```

```
75")
```

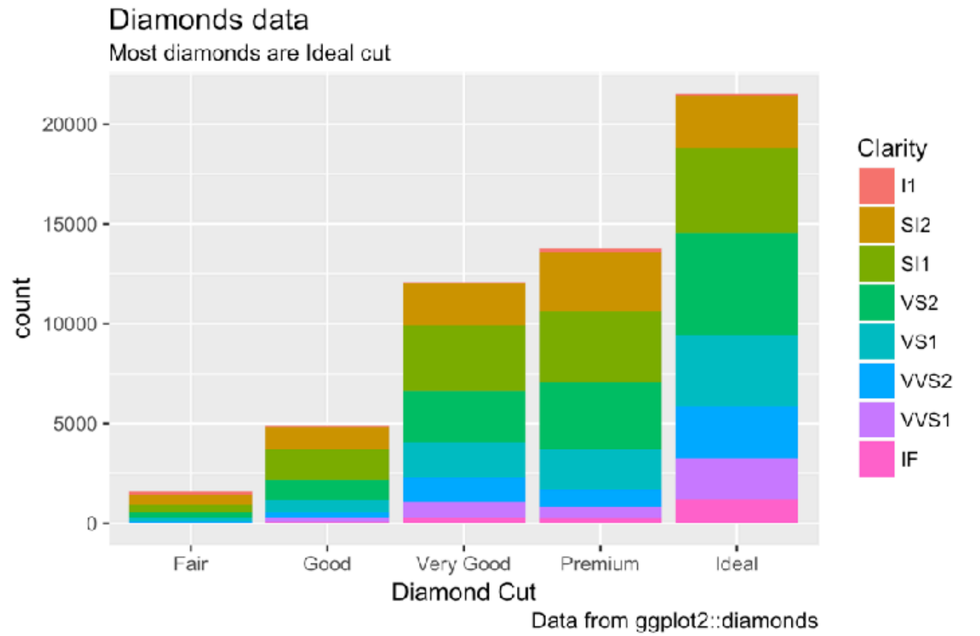


Putting it together

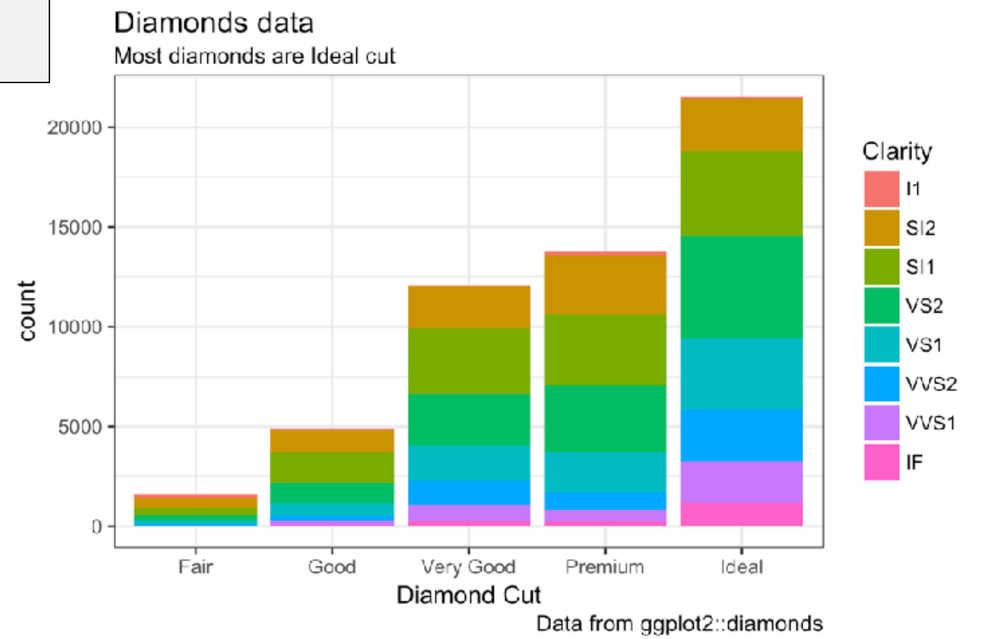
```
diamonds_plot
```



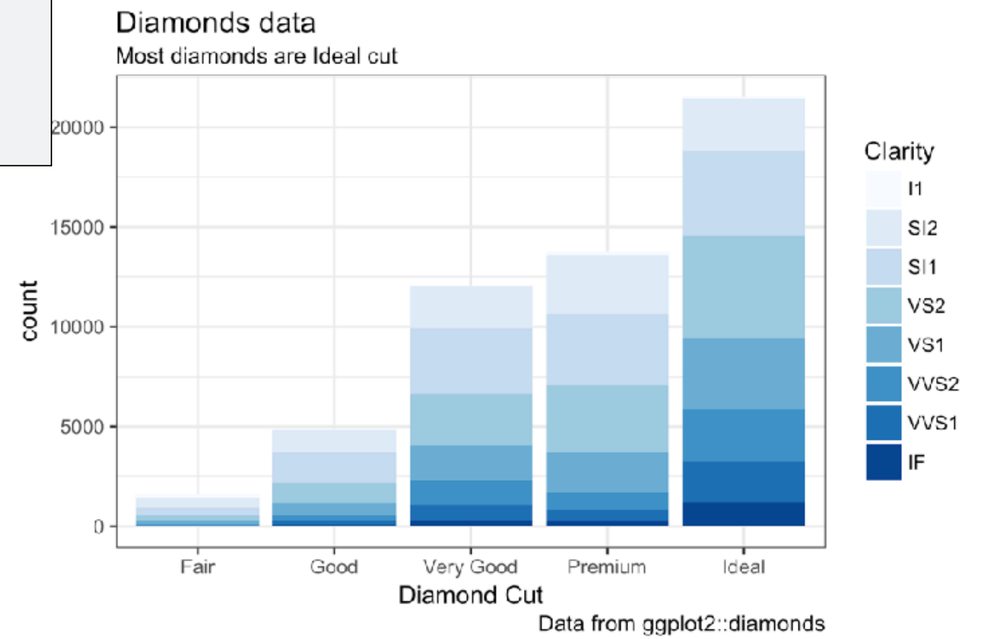
```
diamonds_plot  +
   labs(title = "Diamonds data",
        subtitle = "Most diamonds are Ideal cut",
        caption = "Data from ggplot2::diamonds",
        x = "Diamond Cut",
        fill = "Clarity"
  )
```



```
diamonds_plot  +
   labs(title = "Diamonds data",
        subtitle = "Most diamonds are Ideal cut",
        caption = "Data from ggplot2::diamonds",
        x = "Diamond Cut",
        fill = "Clarity"
  ) +
  theme_bw()
```



```
diamonds_plot  +
   labs(title = "Diamonds data",
        subtitle = "Most diamonds are Ideal cut",
        caption = "Data from ggplot2::diamonds",
        x = "Diamond Cut",
        fill = "Clarity"
  ) +
  theme_bw() +
  scale_fill_brewer()
```

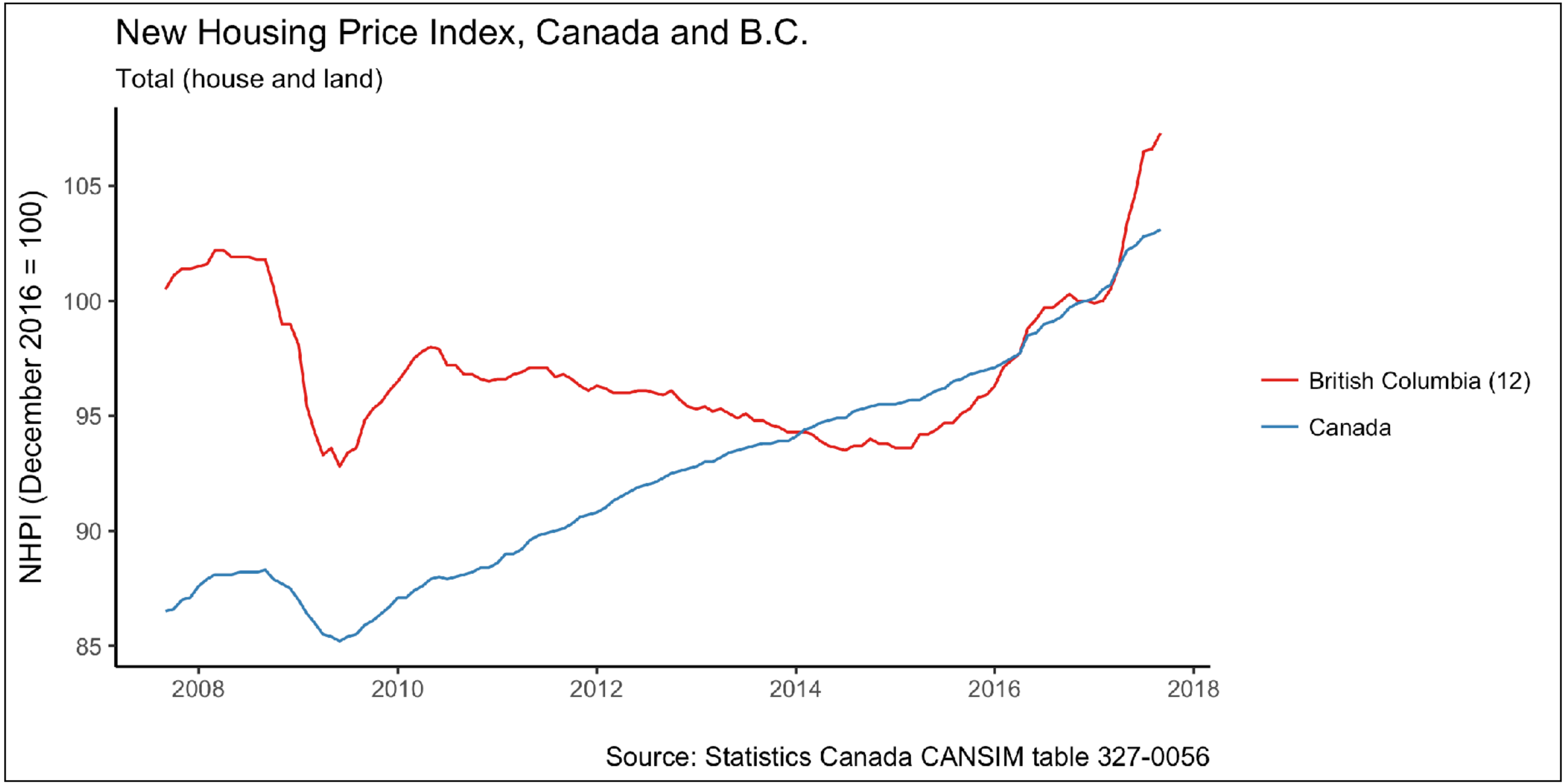


Your Turn 6

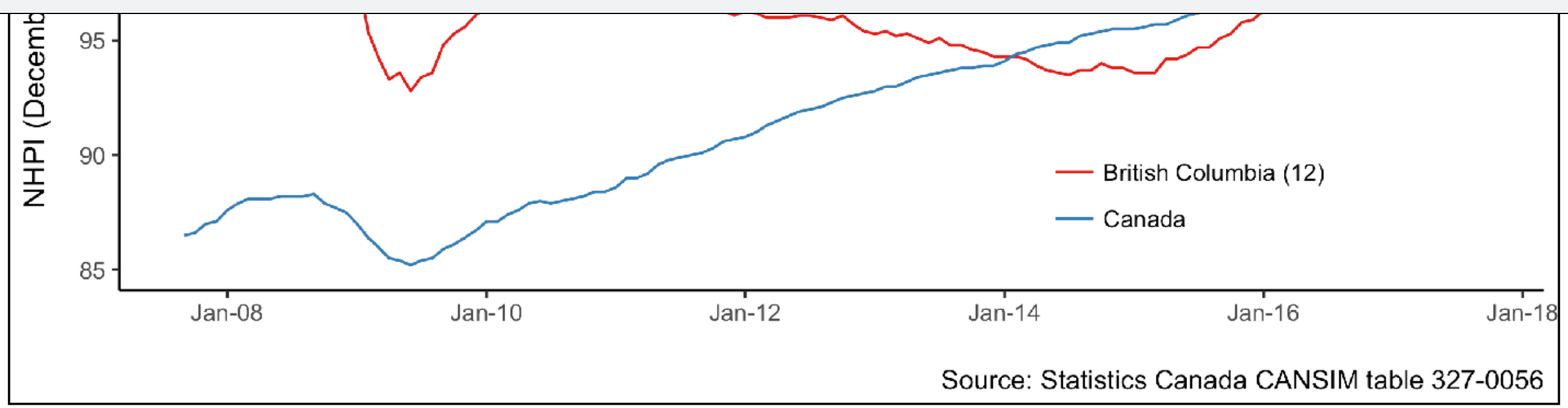
Put the labels, theme and scale changes together for basic_plot.

What is left to change?

```
basic_plot +
  labs(title = "New Housing Price Index, Canada and B.C.",
       subtitle = "Total (house and land)",
       x = "",
       y = "NHPI (December 2016 = 100)",
       color = "",
       caption = "Source: Statistics Canada CANSIM table 327-0056") +
  theme_classic() +
  scale_color_brewer(palette = "Set1")
```



```
last_plot() +
  scale_color_manual(
    labels = c("Canada" = "Canada",
               "British Columbia (12)" = "British Columbia"),
    values = c("Canada" = "#fdb913", "British Columbia (12)" = "#234075")
  )
  scale_x_datetime(date_labels = "%b-%y") +
  theme(legend.position = c(0.75, 0.2))
```



Summary

Everything visual on the plot is customizable

Labels - easiest thing to change to increase readability

Themes - change all non-data visual elements

Scales - control mappings and the legends that go with them