Lecture 5: Data Visualization in R

James Sears*
AFRE 891/991 SS 25
Michigan State University

*Parts of these slides are adapted from **"Advanced Data Analytics"** by Nick Hagerty and **"Data Science for Economists"** by Grant McDermott, used under **CC BY-NC-SA 4.0**.

Table of Contents

Prologue

Prologue

Packages we'll use for today's examples:

```
pacman::p_load(dslabs, gapminder, lubridate, tidyverse)
```

Additional packages if you want to replicate the plots in the "Principles of Data Visualization" section:

```
pacman::p_load(ggrepel, readxl, scales)
```

Prologue

Additional packages if you want to replicate the color schemes later on:

```
pacman::p_load(RColorBrewer, viridis, Polychrome, broman)
```

and to replicate the theme/plot extensions:

```
pacman::p_load(ggthemes, showtext, gganimate, gifski, ggExtra)
```

Data Visualization

Being able to **visualize data well** is a critical skill for economists and data scientists.

- Effectively communicate a message
- Engage a reader/audience
- Discover patterns in the underlying data

To do this in R, we're going to use the incredibly powerful **ggplot2** package, but before we do, let's chat about...

Principles of Data Visualization

Principles of Data Visualization

We're going to focus on the **punchline** of the principles of data visualization today.

If you want more of the **setup**, there are excellent resources available:

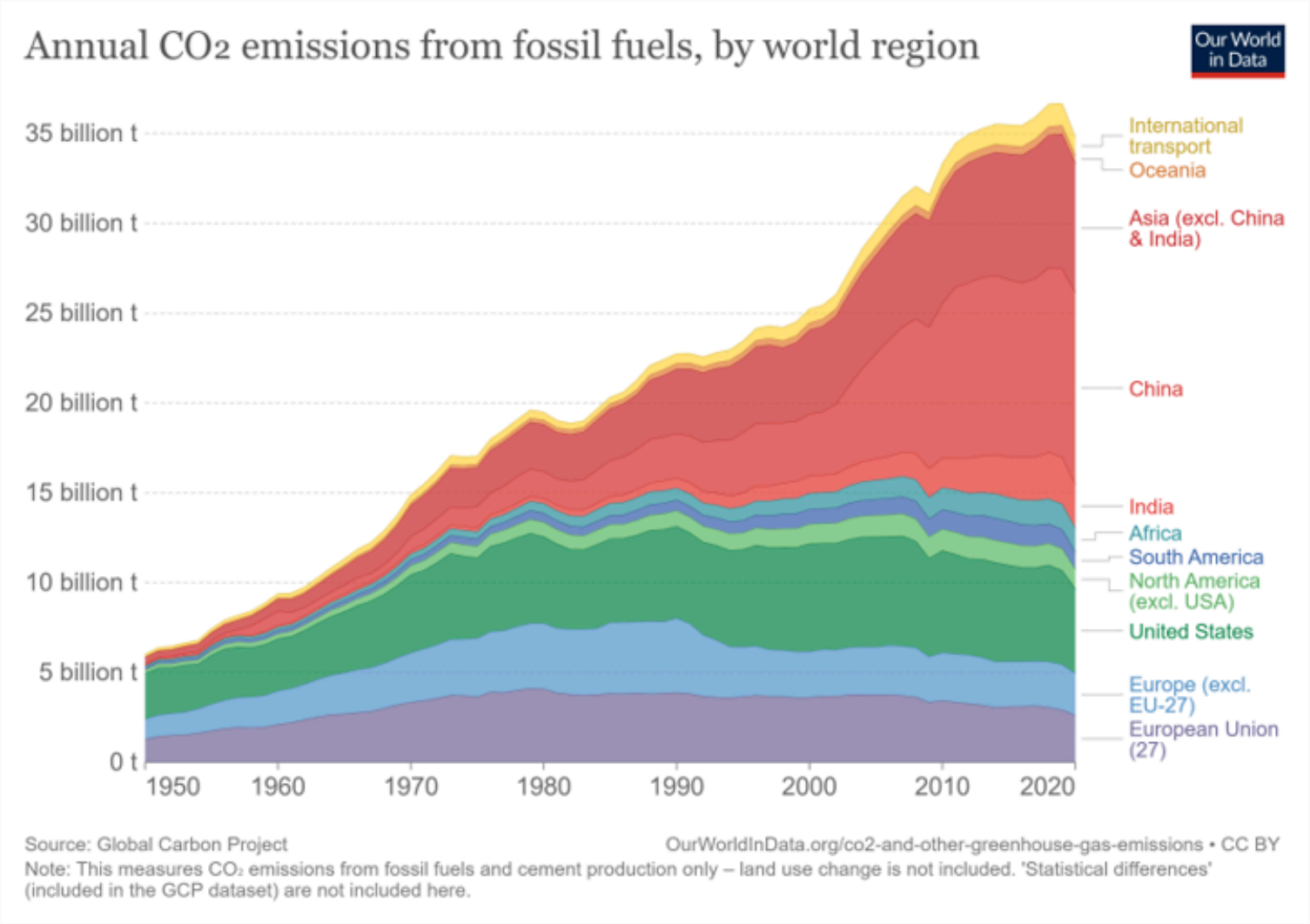
- **Introduction to Data Science** by Rafael A. Irizarry
- **Data Visualization a Practical Introduction** by Kieran Healy
- **"Creating Effective Figures and Tables" talk** by Karl Broman
- **An Economist's Guide to Visualizing Data** by Jonathan A. Schwabish (JPE 2014)
- **Modern Data Science with R, 2nd Ed.** by Baumer, Kaplan, and Horton
- **from Data to Viz (finding the appropriate graph)**
- **R Graph Gallery**

Effective Charts

An **effective graph or chart**

- Conveys information clearly
- Summarizes data quickly
- Helps identify salient features or patterns
- Conveys complex relationships in easy-to-communicate visuals

Effective Charts

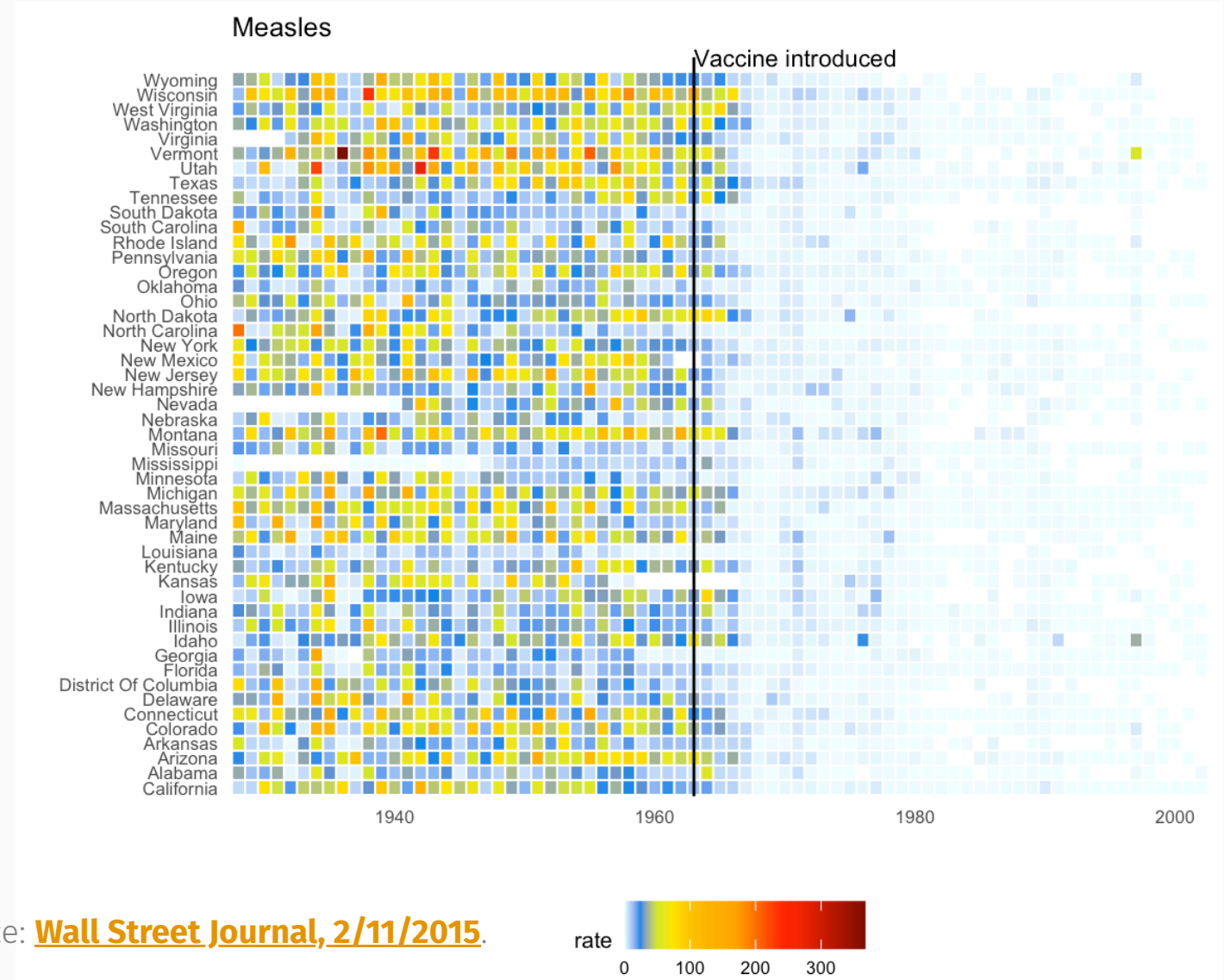


Poorly Constructed Charts

On the other hand, **poorly constructed charts**

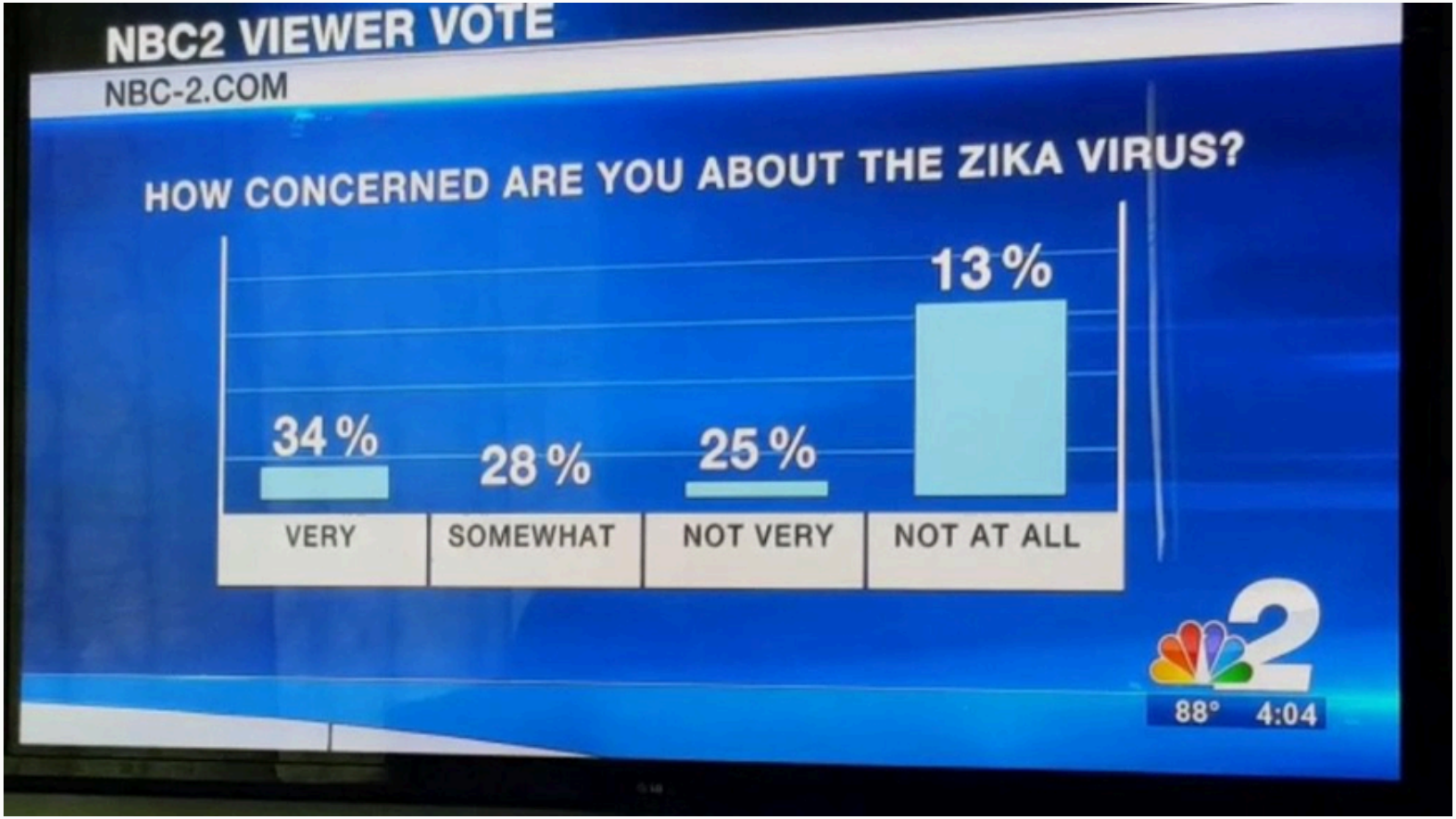
- Unintentionally obscure or purposefully misreport true relationships
- Convey too much information without the necessary orienting information easily accessible
- Are completely uninterpretable

Imperfect Charts



Source: **Wall Street Journal, 2/11/2015**.

Dishonest Charts



Source: **McGarry, 8/15/2016 (Mashable)**.

Principles of Data Visualization

When constructing charts, we need to keep in mind several **key principles:**

1. Show the Data

2. Reduce the Clutter

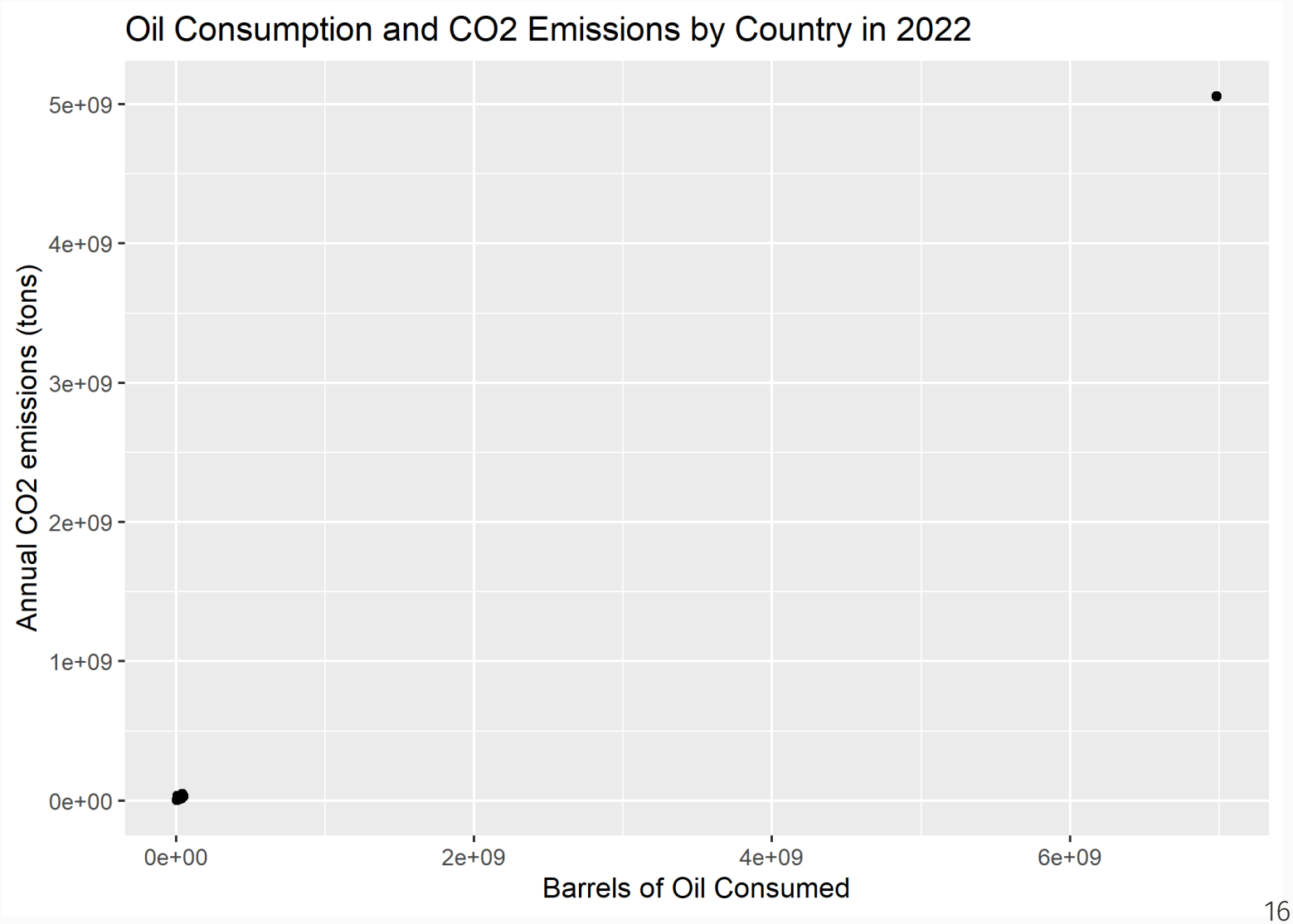
3. Integrate the Text and the Graph

Show the Data

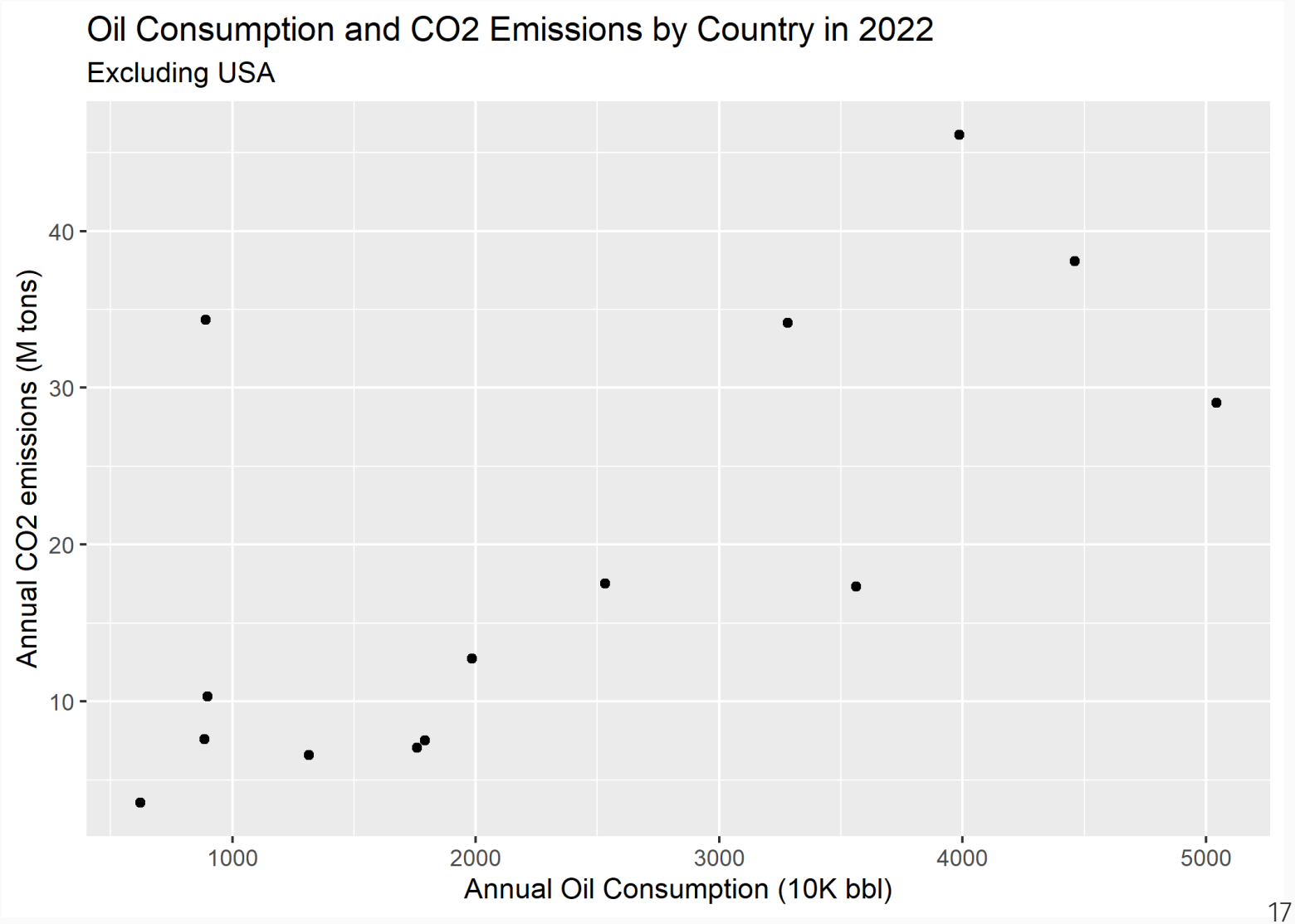
We use graphs to help readers/listeners understand a story. The data are the **most important part** of this.

- Present the data in the **clearest way possible**
- However, that doesn't mean we have to show **all the data**

Show the Data



Show the Data (Clearly)



Reduce the Clutter

Ask yourself: What is the **central message** you are trying to communicate?

Decide, then build your plot around that message.

- Make that message as **easy to see as you can**.
- **Remove the clutter**
  - Get rid of any features of the visualization that .hi-medgrn[do not contribute] to the .hi-medgrn[central message].

Example of A "Clutterplot"

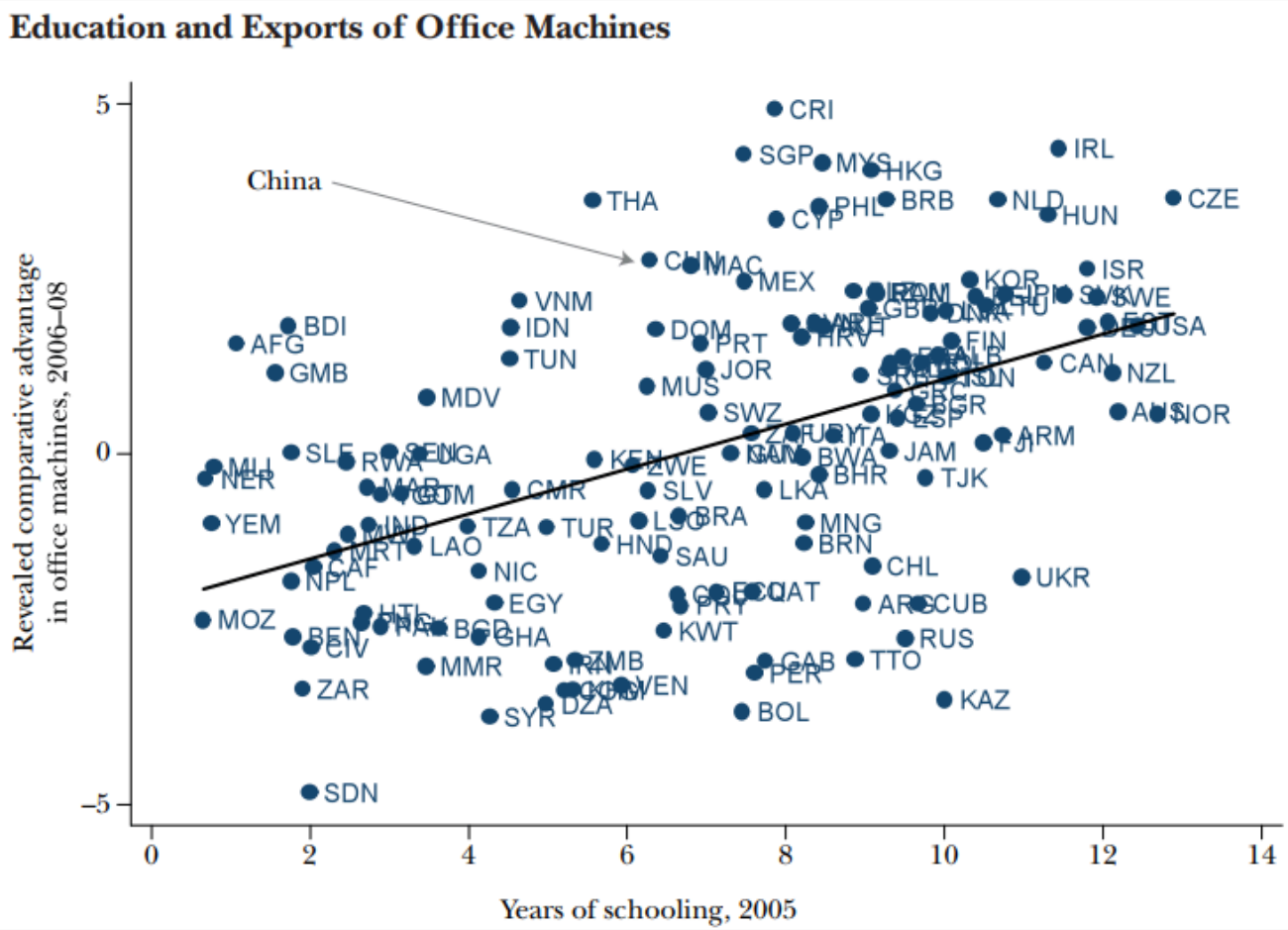


Image is from **"An Economist's Guide to Visualizing Data" by Jonathan Schwabish** and excluded from the overall CC license.

Reduce the Clutter

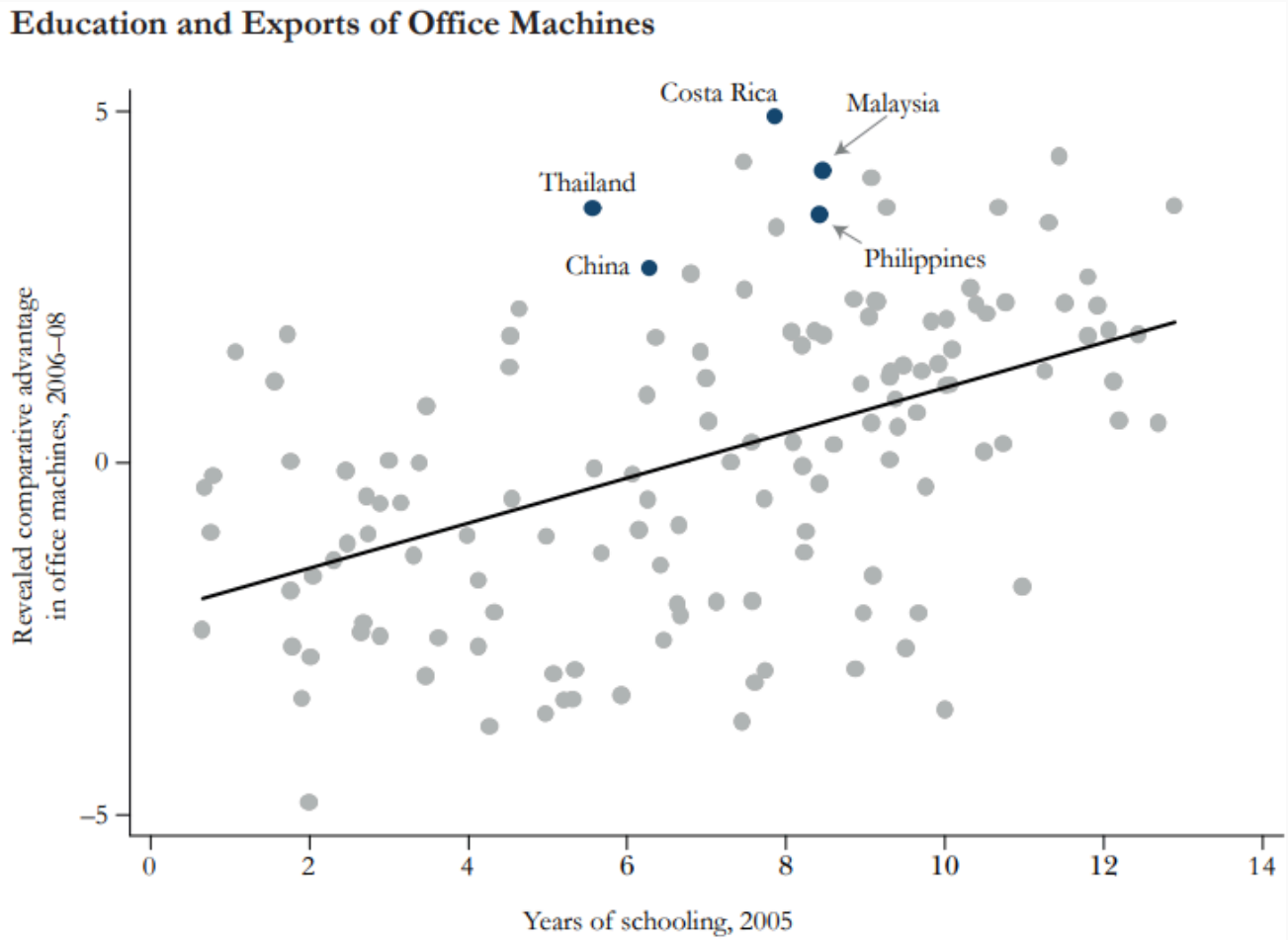


Image is from **"An Economist's Guide to Visualizing Data" by Jonathan Schwabish** and excluded from the overall CC license.

Integrate the Text and the Graph

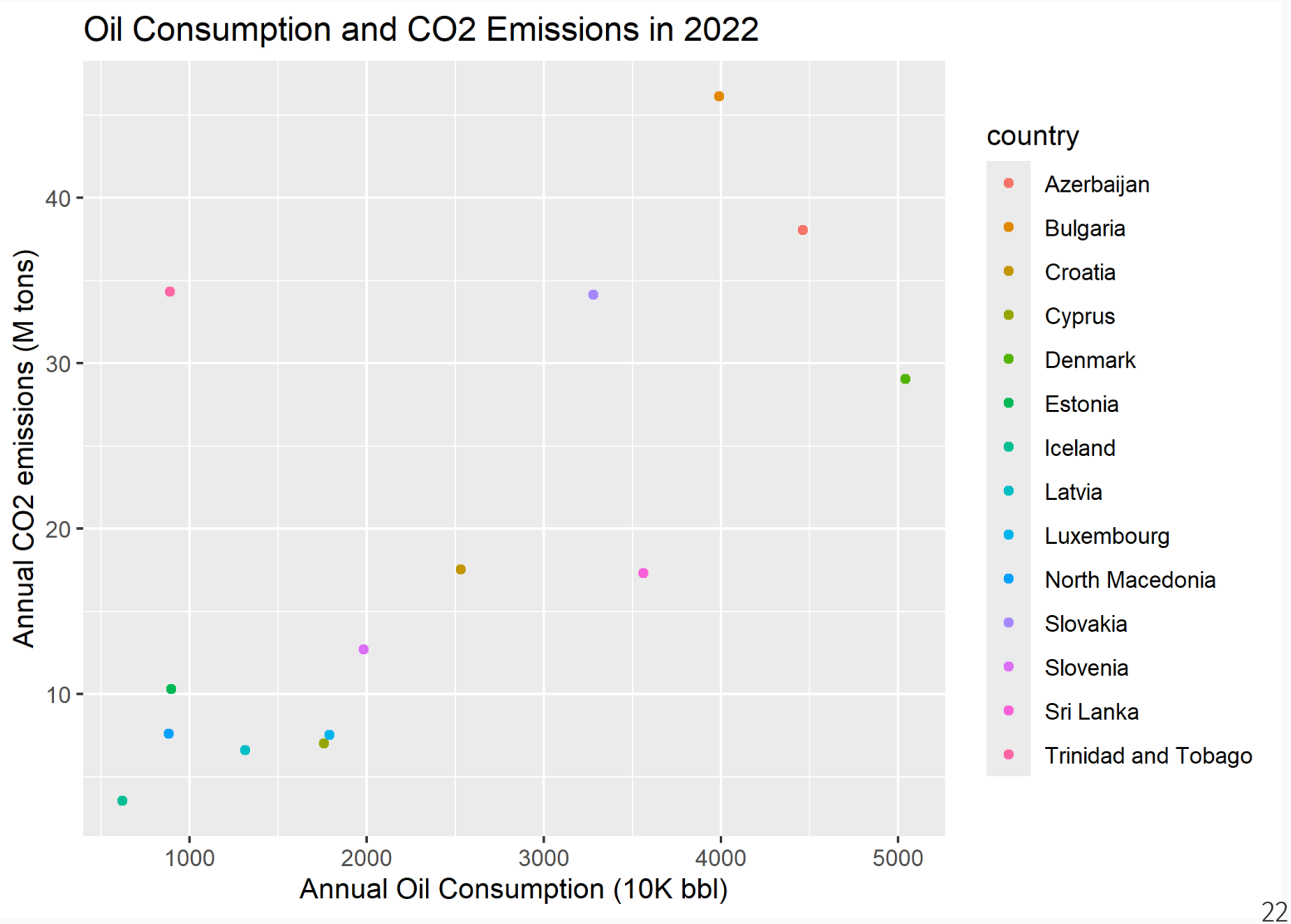
Research reports often suffer from the **slideshow effect**

- Writer narrates the graph's text elements

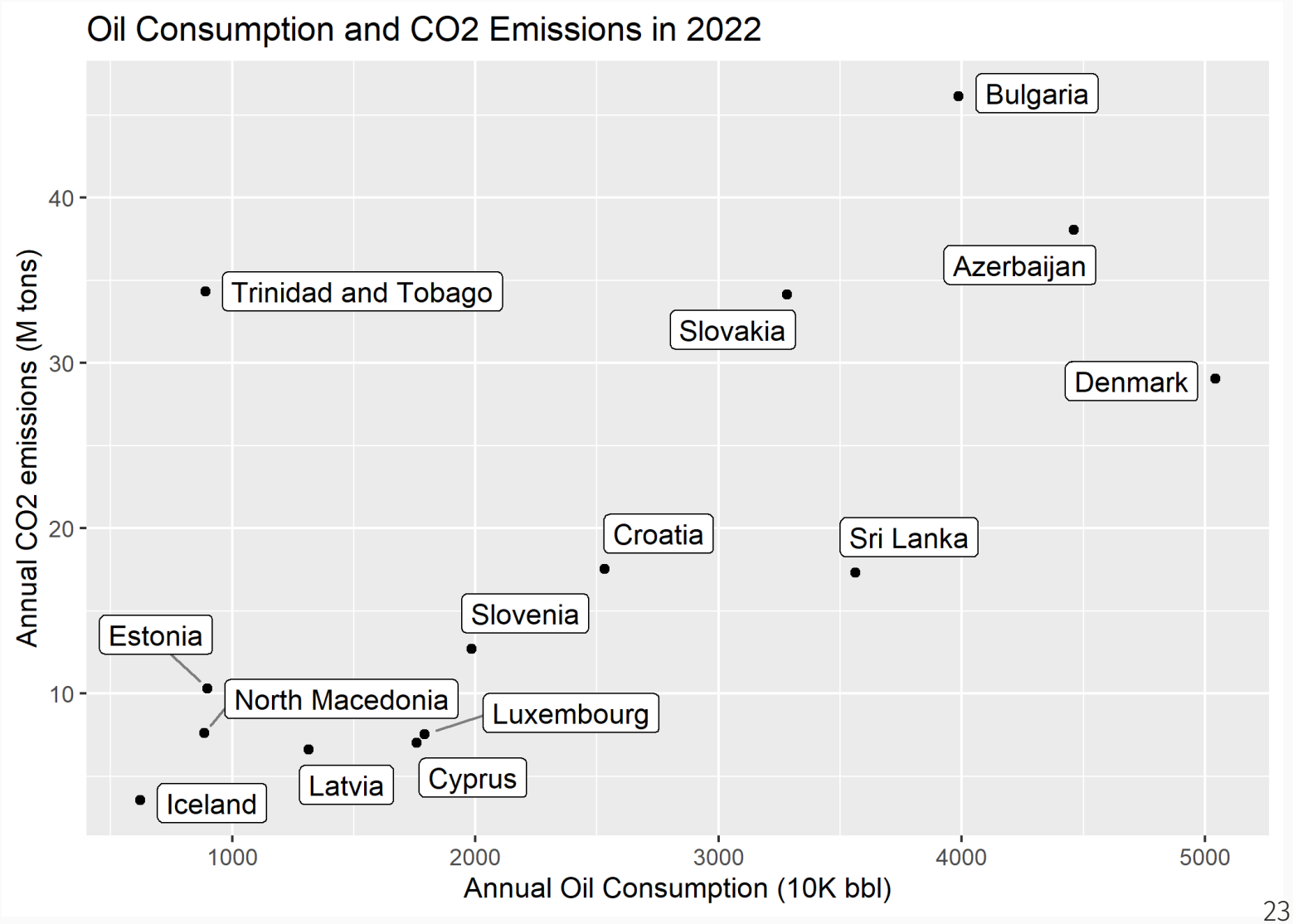
Better: integrate the text and the graph!

- Visuals are built to **complement the text**
- Charts contain enough information to **fully stand alone**
- Place labels/legend elements **close to the element its referencing**

Disconnected Text and Graph



Integrate the Text and the Graph



Principles of Data Visualization

Many of the charts that *don't* follow these principles often violate either

1. Area Principle: the area occupied by the chart element reflects its **full value**

- **Common violation:** bar/column chart axis **doesn't start at zero**

2. White Space Rule: if there's too much white space, **refocus the plot**

- Remove **unnecessary outliers** (i.e. that USA emissions observation)
- Split the chart into **multiple panels**
- **Change axis scales** (e.g. logarithmic - but know your audience!)

Violating the Area Principle

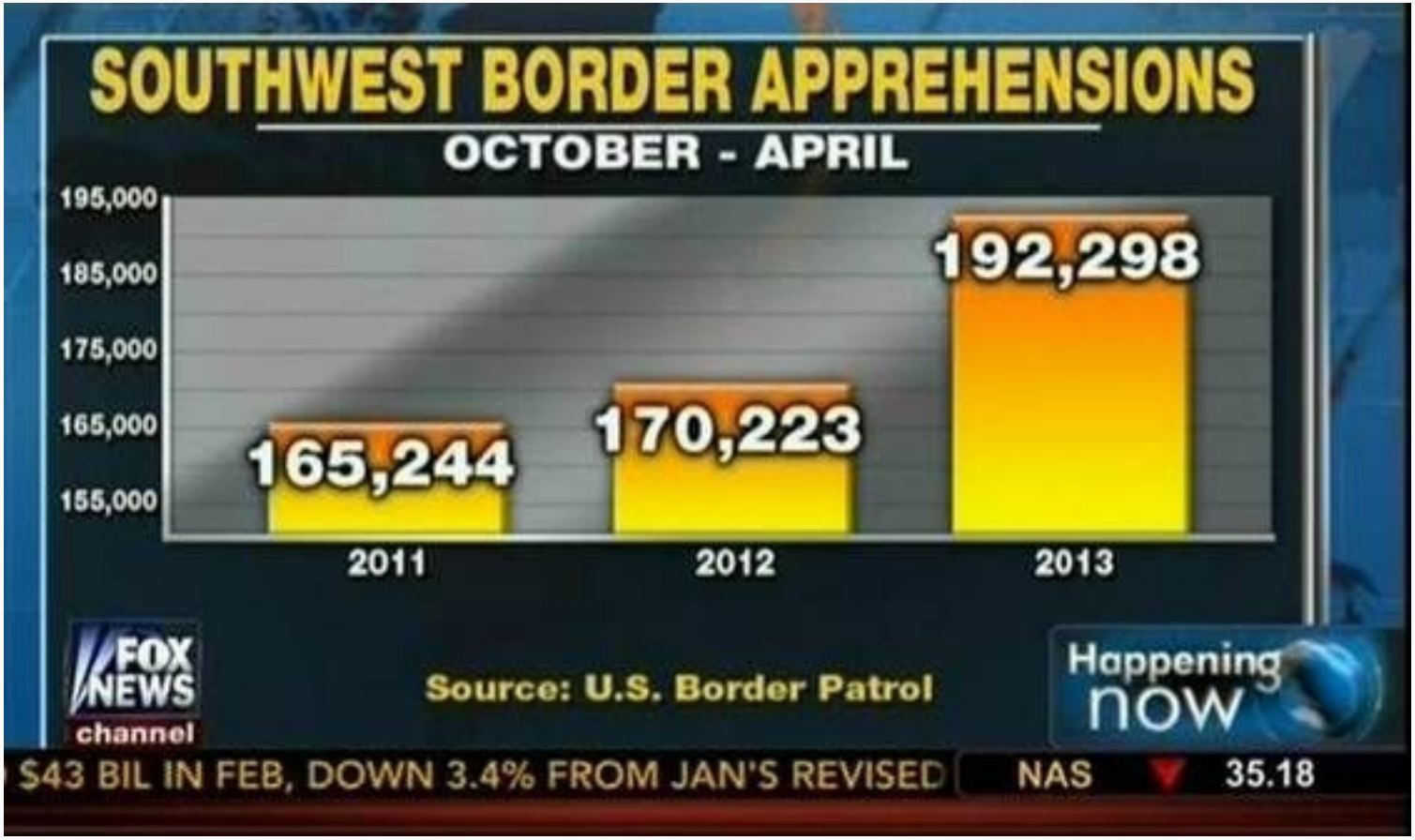


Image is from **"Introduction to Data Science" by Rafael A. Irizarry**

Violating the Area Principle

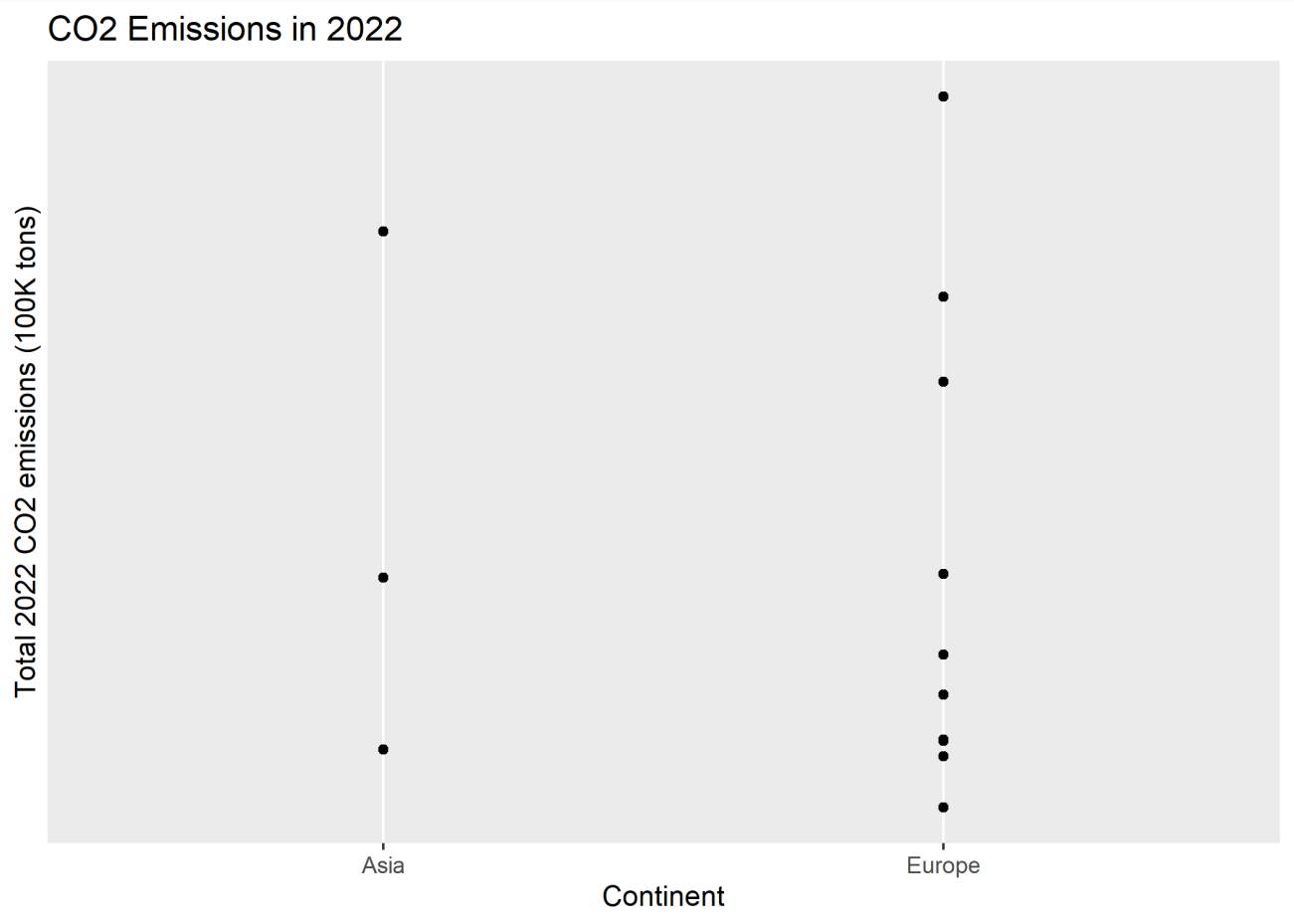
Image is from **"Introduction to Data Science" by Rafael A. Irizarry**

Violating the Area Principle (Again)

Image is from **"Introduction to Data Science" by Rafael A. Irizarry**

NOT an Area Principle Violation

Note that we *don't* need to include 0 if we use **position** rather than length:



Getting Started with ggplot2

Elements of ggplot2

ggplot2 is one of the most popular packages in the entire R canon.

- Built upon some deep visualization theory: i.e. **Leland Wilkinson's "The Grammar of Graphics"**.

There's a lot to say about **ggplot2**'s implementation of this "grammar of graphics" (gg) approach, but the three key elements are:

1. Your plot ("the visualization") is linked to your variables ("the data") through various **aesthetic mappings.**

2. Once the aesthetic mappings are defined, you can represent your data in different ways by choosing different **geoms** (i.e. "geometric objects" like points, lines or bars).

3. You build your plot in **layers**.

Elements of ggplot2

1. Link plots to variables through **aesthetic mappings**
2. Represent data using **geoms**
3. Build your plot in **layers**

That's kind of abstract. Let's review each element in turn with some actual plots, using panel data on life expectancy, population size, and GDP per capita for 142 countries since the 1950s from the **gapminder** package

Main Function: ggplot()

We'll start every plot with the core plot function, `ggplot()`.

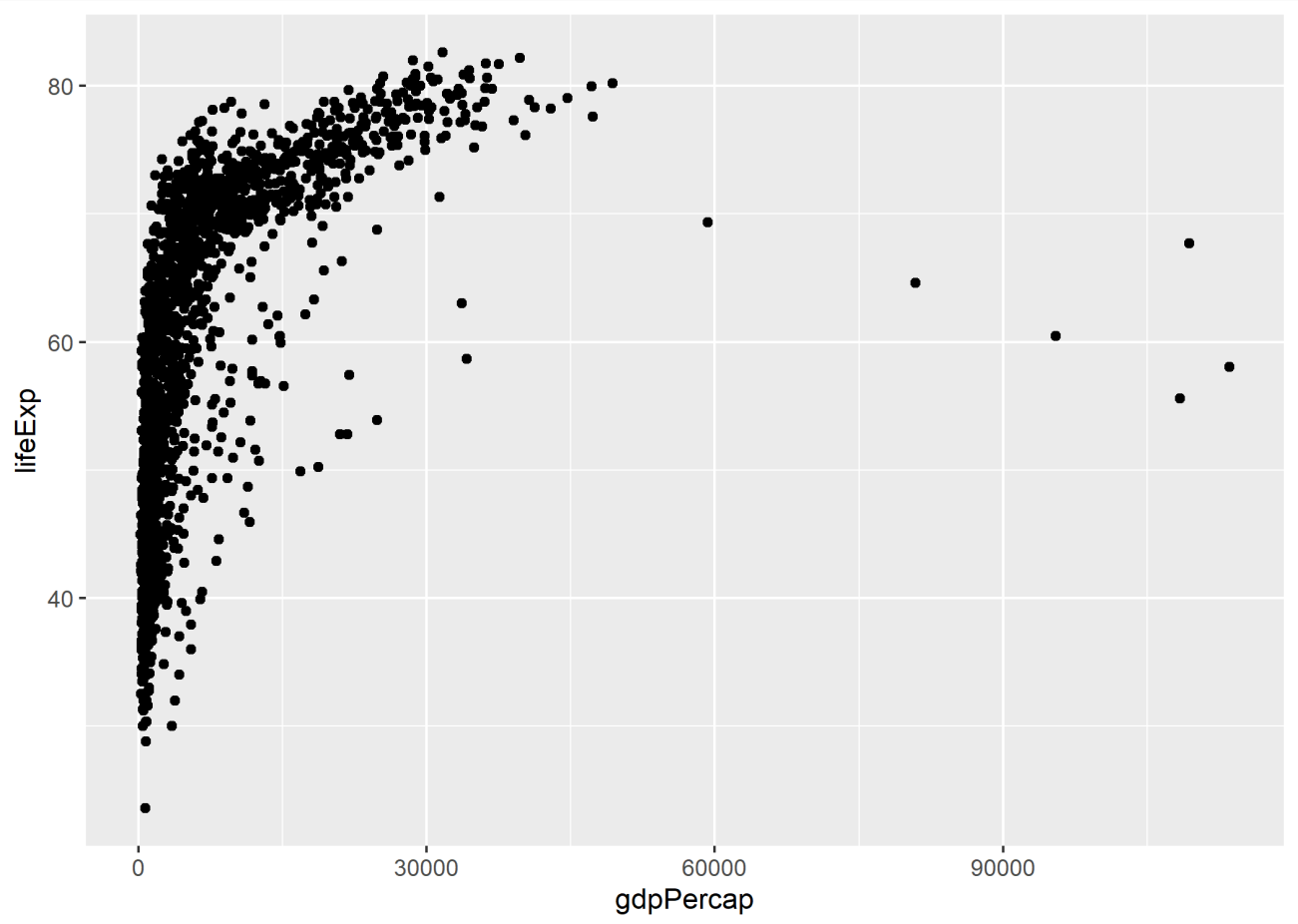
Note that on its own all `` `ggplot() `` does is initiate a plot. In order for that plot to actually **show us something useful**, we have to supply it with **aesthetic mappings**.

1. (No) Aesthetic Mappings

```
ggplot()
```

1. Aesthetic Mappings

```
ggplot(data = gapminder,
       mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point()
```



1. Aesthetic Mappings

```
ggplot(data = gapminder,
       mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point()
```

Focus on the top two lines, which contain the initializing `ggplot()` function call. This function accepts various arguments, including:

- Where the **data come from**
  - i.e. `data = gapminder`
- What the **aesthetic mappings** are
  - i.e. `mapping = aes(x = gdpPercap, y = lifeExp)`
- Here we're setting the data and mapping **globally**
  - The data source and mapping apply to **every subsequent geom layer automatically**

2. Layers

```
ggplot(data = gapminder,
       mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point()
```

The third line adds a **geom layer**

- Literally *added* to `ggplot()` with a `+`
- Second half of the **geom name** will determine what **type of geometry** it adds to the chart
- Here we're adding a scatterplot with `geom_point()`

1. Aesthetic Mappings

```
ggplot(data = gapminder,
       mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point()
```

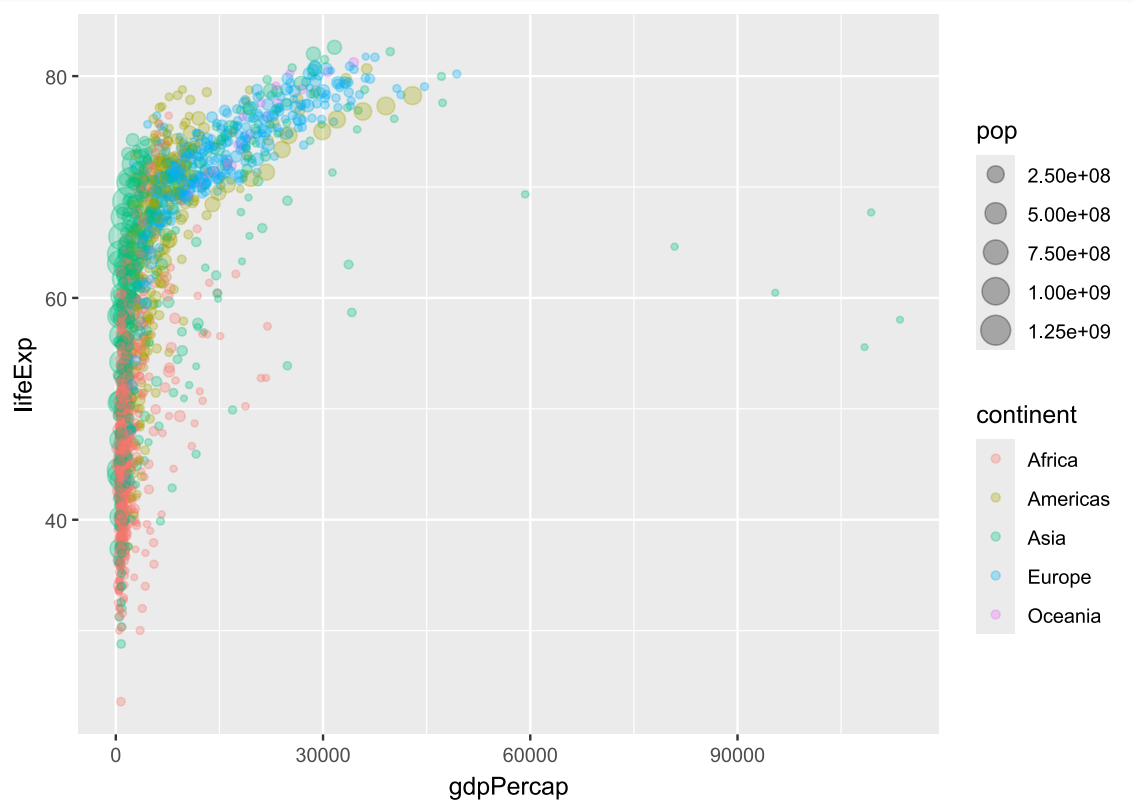
The aesthetic mappings here are pretty simple: They just define

- an x-axis (GDP per capita, `x`)
- a y-axis (life expectancy, `y`)

- To get a sense of the power and flexibility that comes with this approach, however, consider what happens if we add **more aesthetic mappings** to the plot call...

1. Aesthetic Mappings: size and col(or)

```
ggplot(data = gapminder,
       aes(x = gdpPercap, y = lifeExp,
           size = pop, col = continent)) +
  geom_point(alpha = 0.3) ## "alpha" controls transparency. Takes a value
```



1. Aesthetic Mappings

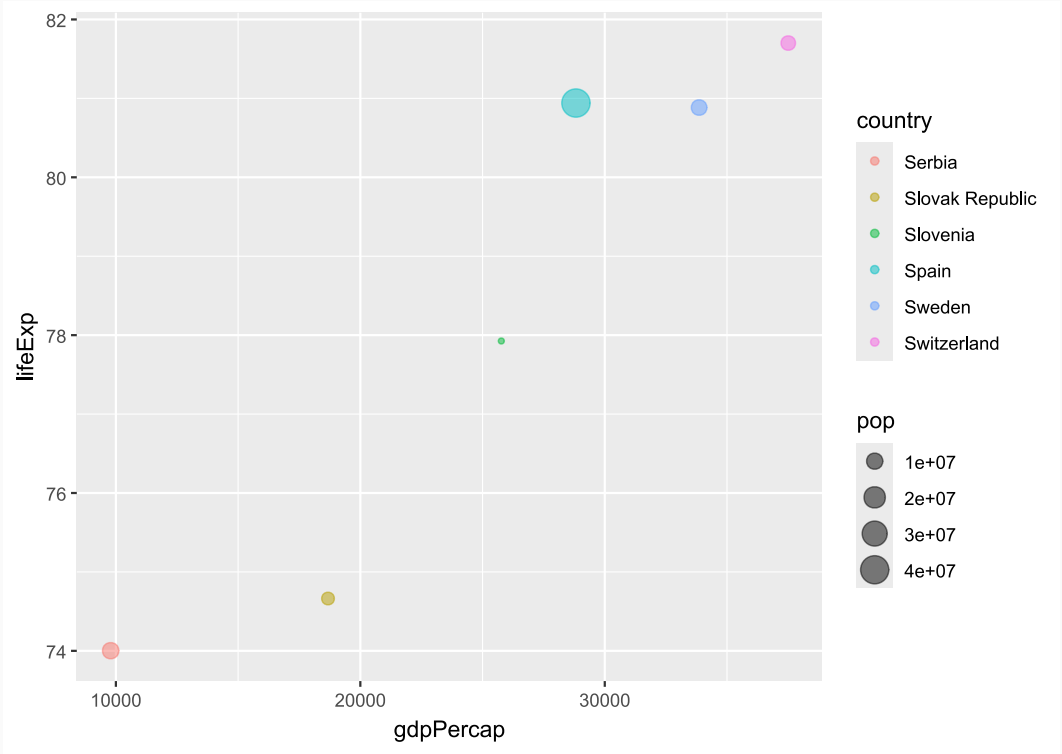
```
ggplot(data = gapminder,
       aes(x = gdpPercap,
           y = lifeExp, # set x/y variables
           size = pop, # scale point size with value of "pop"
           col = continent # change color on value of "continent"
           )
       ) +
  geom_point(alpha = 0.3) # "alpha" controls transparency in [0,1]
```

Note that I've dropped the `mapping =` part of the ggplot call. Most people just start with `aes( ... )`, since **ggplot2** knows the order of the arguments.

1. Aesthetic Mappings

We can also **pipe our data** into a ggplot object. For example, plotting just european countries starting with "S" for 2007:

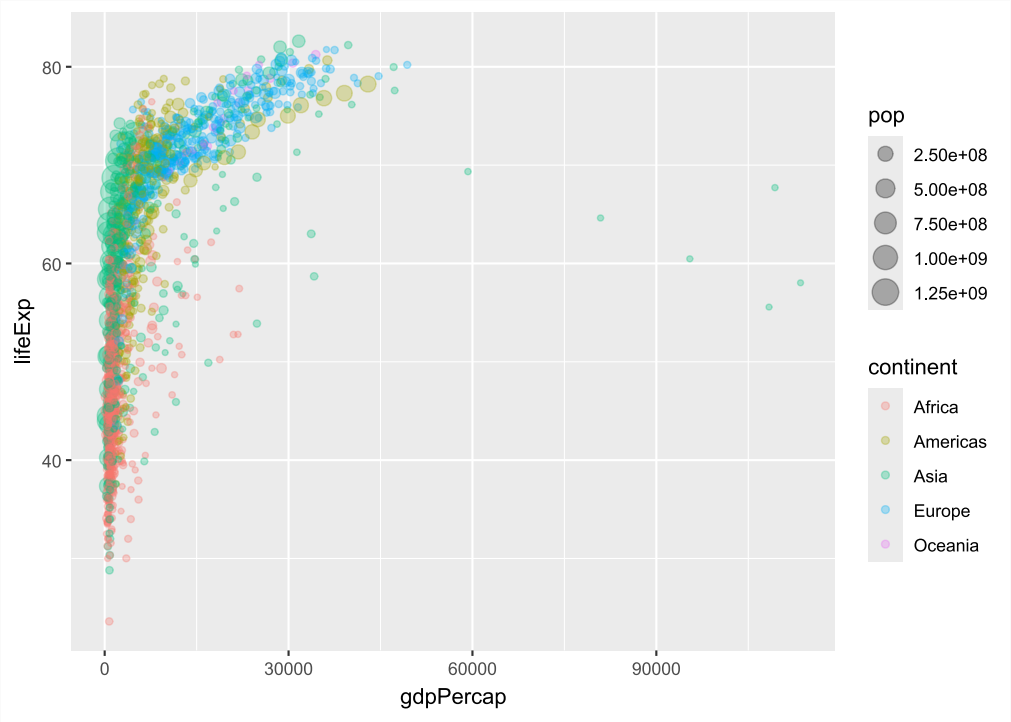
```
filter(gapminder, continent == "Europe", str_detect(country, "^S"), year == 2007) %>%
ggplot(aes(x = gdpPercap,  y = lifeExp,  size = pop, col = country)) +
  geom_point(alpha = 0.5) # "alpha" controls transparency in [0,1]
```



1. Aesthetic Mappings

We can specify aesthetic mappings **locally in the geom layer too**.

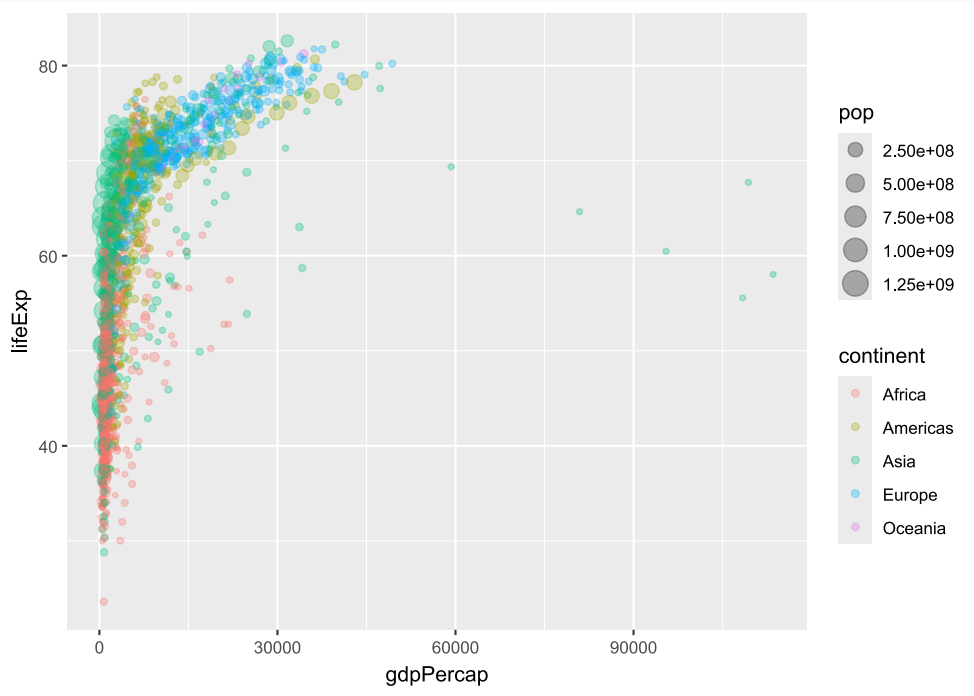
```
## First aes: applicable to all geoms
ggplot(data = gapminder, aes(x = gdpPercap, y = lifeExp)) +
## Next aes: specific to this geom only
  geom_point(aes(size = pop, col = continent), alpha = 0.3)
```



1. Aesthetic Mappings

Data can be declared **locally too**.

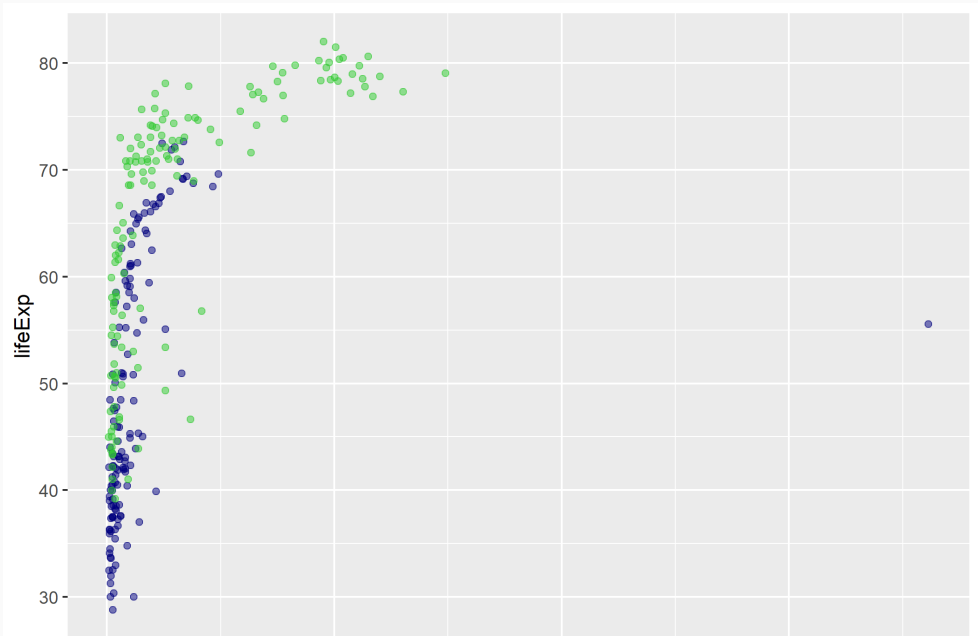
```
ggplot() +
  geom_point(data = gapminder,
             aes(x = gdpPercap, y = lifeExp, size = pop, col = continent)
```



1. Local Data Declaration

Useful when you want to plot **multiple datasets** on the **same plot**, but **more work** to get a **legend working properly** (foreshadowing)

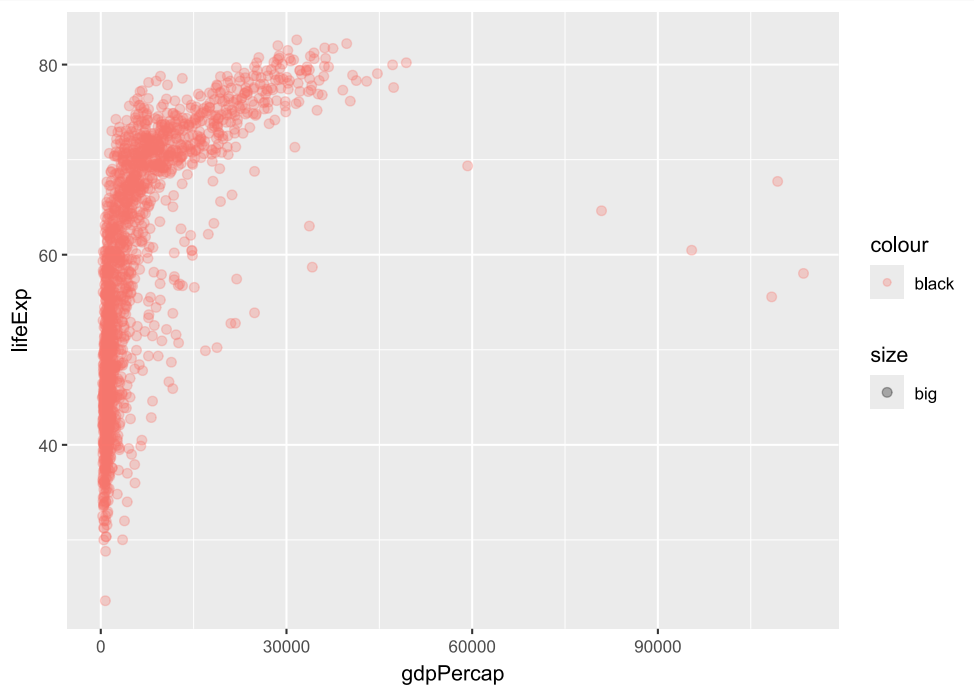
```
gap_52 ← filter(gapminder, year == 1952)
gap_02 ← filter(gapminder, year == 2002)
ggplot() +
  geom_point(data = gap_52, aes(x = gdpPercap, y = lifeExp), col = "navyblue", alp
  geom_point(data = gap_02, aes(x = gdpPercap, y = lifeExp), col = "limegreen", al
```



1. Aesthetic Mappings

Oops. What went wrong here?

```
ggplot(data = gapminder, aes(x = gdpPercap, y = lifeExp)) +
  geom_point(
    aes(size = "big", col="black"),
        alpha = 0.3)
```



1. Aesthetic Mappings

Oops. What went wrong here?

```
ggplot(data = gapminder, aes(x = gdpPercap, y = lifeExp)) +
  geom_point(
    aes(size = "big", col="black"),
    alpha = 0.3)
```

Answer: Aesthetics must be mapped to **variables**[2], not descriptions!

[2] This isn't *always* true; we'll see later how we can use this to customize legends/value labels

1. Aesthetic Mappings

Instead of repeating the same ggplot2 call every time, we can **store it in memory** as an intermediate **plot object** that we can re-use.[3]

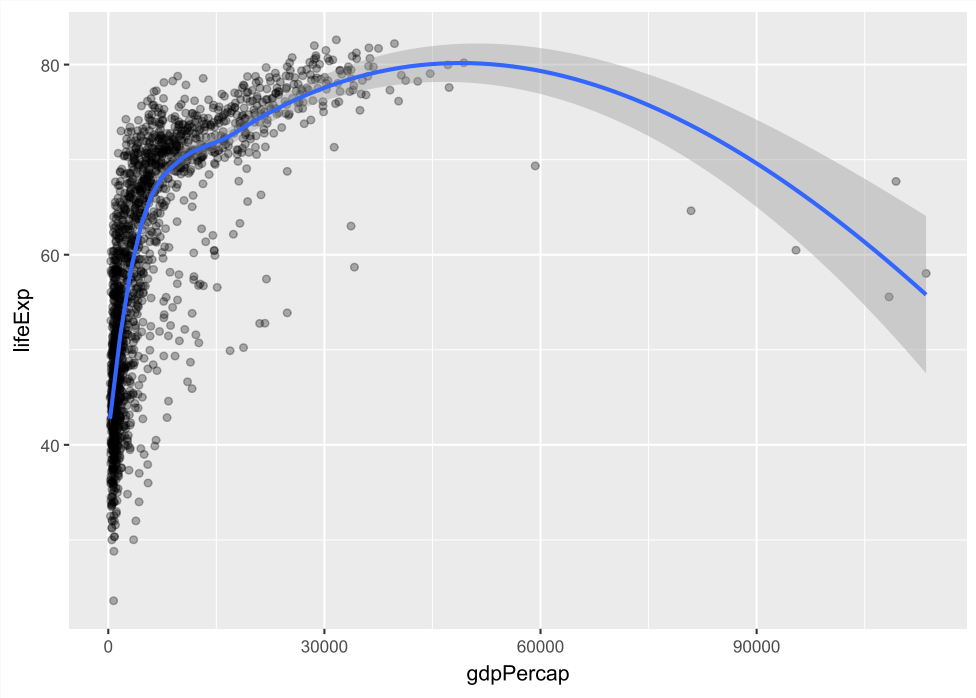
```
p ← ggplot(data = gapminder, aes(x = gdpPercap, y = lifeExp))
```

[3] We can store anything in memory! It's like R's version of the **Portlandia "put a bird on it" sketch**

2. Geoms

You can invoke and combine **different geoms** to generate **different visualizations**.

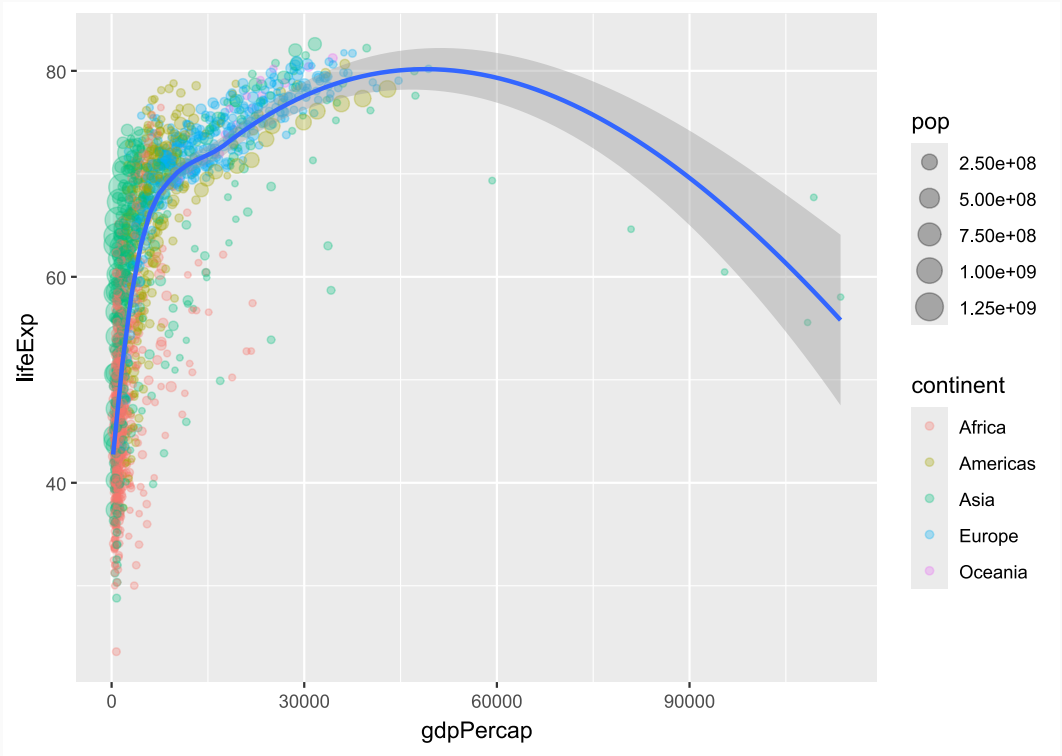
```
p +
  geom_point(alpha = 0.3)  +
  geom_smooth(method = "loess") # add local polynomial regression fit
```



2. Geoms

Aesthetics can be applied **differentially across geoms**, too.

```
p +
  geom_point(aes(size = pop, col = continent), alpha = 0.3)  +
  geom_smooth(method = "loess")
```



2. Geoms

The previous plot provides a good illustration of the power (or effect) that comes from assigning aesthetic mappings **"globally"** vs in the individual geom layers **"locally"**.

- **Compare:** What happens if you run the below code chunk?

```
ggplot(data = gapminder,
       aes(x = gdpPercap,
           y = lifeExp,
           size = pop,
           col = continent)) +
  geom_point(alpha = 0.3)  +
  geom_smooth(method = "loess")
```

2. Geoms (cont.)

Similarly, note that some geoms only accept **specific mappings**.

- E.g. `geom_density()` doesn't know what to do with the "y" aesthetic mapping.

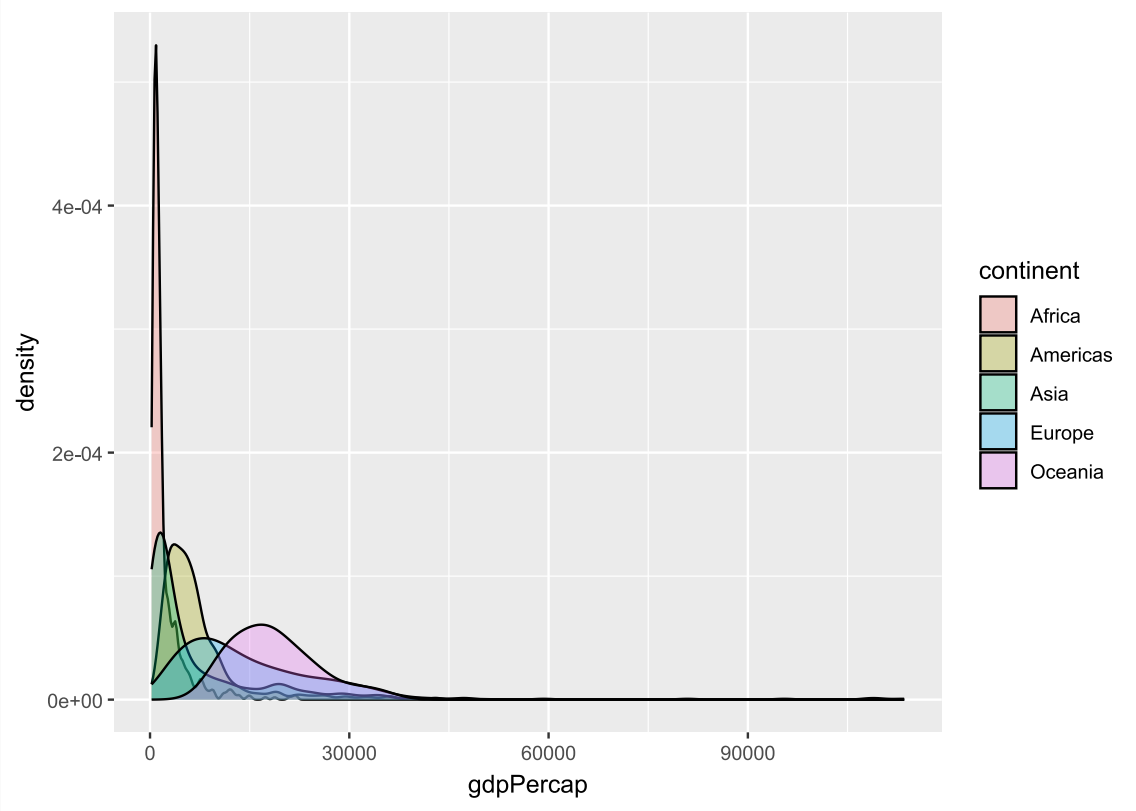
```
p + geom_density()
```

```
## Error in geom_density():
## ! Problem while setting up geom.
## ℹ Error occurred in the 1st layer.
## Caused by error in compute_geom_1():
## ! geom_density() requires the following missing aesthetics: y.
```

2. Geoms (cont.)

We can fix that by being more careful about how we build the plot.

```
ggplot(data = gapminder) + ## i.e. No "global" aesthetic mappings"
  geom_density(aes(x = gdpPercap, fill = continent), alpha=0.3)
```



3. Build your Plot in Layers

We've already seen how we can chain (or "layer") **consecutive plot elements** using the `+` connector.

- The fact that we can create and then re-use an intermediate plot object (e.g. `p`) is testament to this.

But it bears repeating: you can build out some **truly impressive complexity and transformation** of your visualization through this simple layering process.

- You don't have to transform your original data; **ggplot2** takes care of all of that.
- For example...

3. Build your Plot in Layers

1. Add points, size scaled to population and color by continent

```
p2 ← p +
  geom_point(aes(size = pop, col = continent), alpha = 0.3)
```

3. Build your Plot in Layers

2. Set a different color scale using RColorBrewer palettes with `scale_color_brewer()`

```
p2 ← p +
  geom_point(aes(size = pop, col = continent), alpha = 0.3) +
  scale_color_brewer(name = "Continent", palette = "Set1")
```

3. Build your Plot in Layers

3. Add a different legend scale for size with `scale_size()`

```
p2 ← p +
  geom_point(aes(size = pop, col = continent), alpha = 0.3) +
  scale_color_brewer(name = "Continent", palette = "Set1") +  ## Differen
  scale_size(name = "Population", labels = scales::comma)
```

3. Build your Plot in Layers

4. Change to a base-10 logarithm y-axis scale with `scale_x_log10()`

```
p2 ← p +
  geom_point(aes(size = pop, col = continent), alpha = 0.3) +
  scale_color_brewer(name = "Continent", palette = "Set1") +  ## Differen
  scale_size(name = "Population", labels = scales::comma) +  ## Different
  scale_x_log10(labels = scales::dollar)
```

3. Build your Plot in Layers

5. Add better axis titles with `labs()`

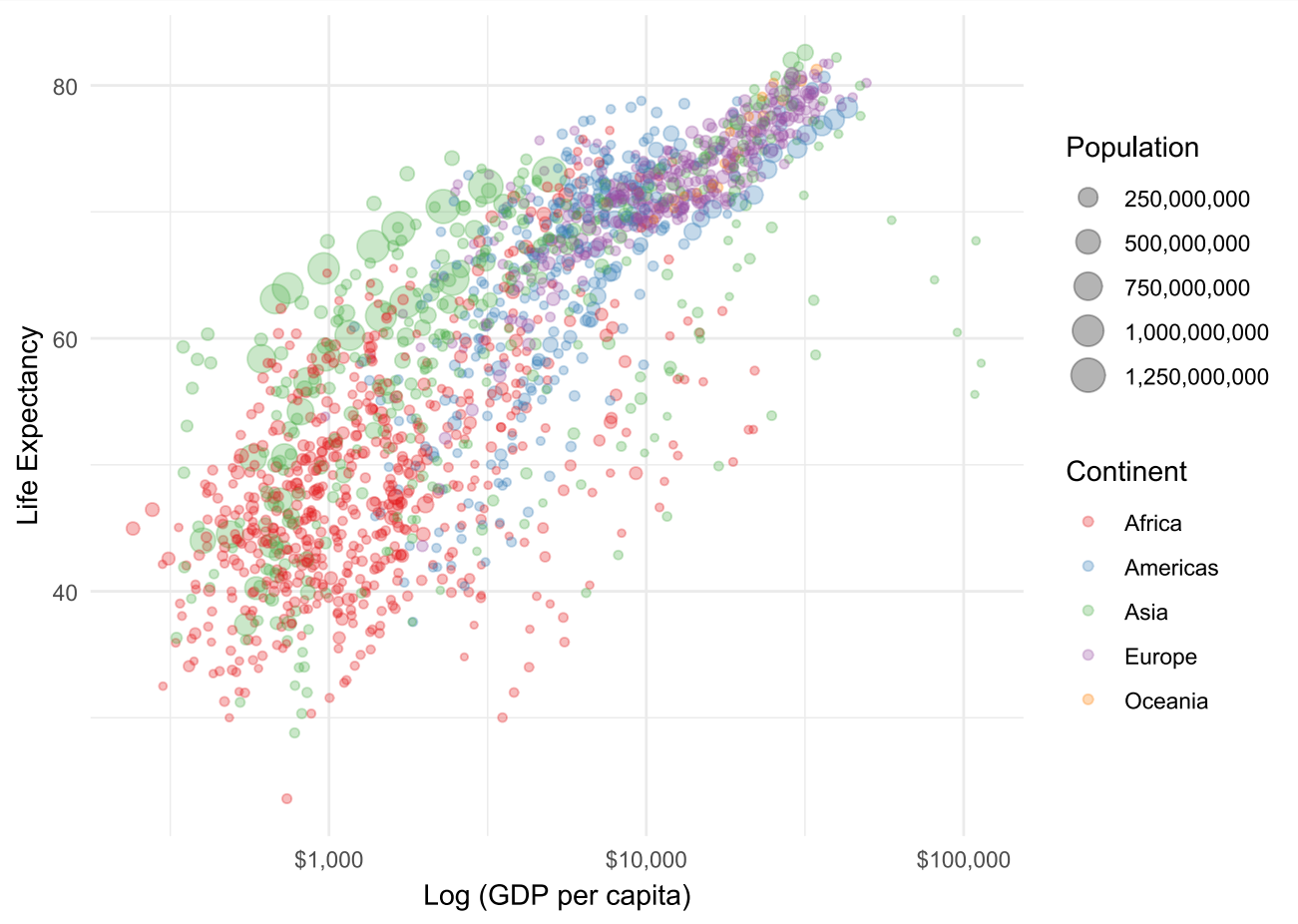
```
p2 ← p +
  geom_point(aes(size = pop, col = continent), alpha = 0.3) +
  scale_color_brewer(name = "Continent", palette = "Set1") +  ## Differen
  scale_size(name = "Population", labels = scales::comma) +  ## Different
  scale_x_log10(labels = scales::dollar) + ## Switch to logarithmic scale
  labs(x = "Log (GDP per capita)", y = "Life Expectancy")
```

3. Build your Plot in Layers

5. Add a clean theme with `theme_minimal()`

```
p2 ← p +
  geom_point(aes(size = pop, col = continent), alpha = 0.3) +
  scale_color_brewer(name = "Continent", palette = "Set1") +  ## Differen
  scale_size(name = "Population", labels = scales::comma) +  ## Different
  scale_x_log10(labels = scales::dollar) + ## Switch to logarithmic scale
  labs(x = "Log (GDP per capita)", y = "Life Expectancy") +  ## Better ax
  theme_minimal()
```

3. Build your Plot in Layers



Common Charts

Common Charts

Now that we've seen an example of the complex plots we can create with **ggplot2**, let's take a step back and talk through the **geoms** we'll need for common charts and some useful **customization settings**.

This Lecture

- Line
- Scatterplot
- Histogram
- Ridge Plots
- Kernel Densities

Later this Semester

- Ribbons
- Dot and Whisker Plots
- Event Study Plots
- Maps

Other Common Charts

Now that we're up to speed with the syntax of **ggplot2**, let's work through some more techniques with other chart types.

Let's use a *different* version[1] of the gapminder data from the **dslabs** package.

```
gap_ds ← dslabs::gapminder
```

[1] They're *both* called `gapminder` but have different variable names, so to prevent confusion we're assigning the **dslabs** to a named object.

Line Chart (*geom_line()*)

How has average life expectancy in the US evolved from 1960-2016?

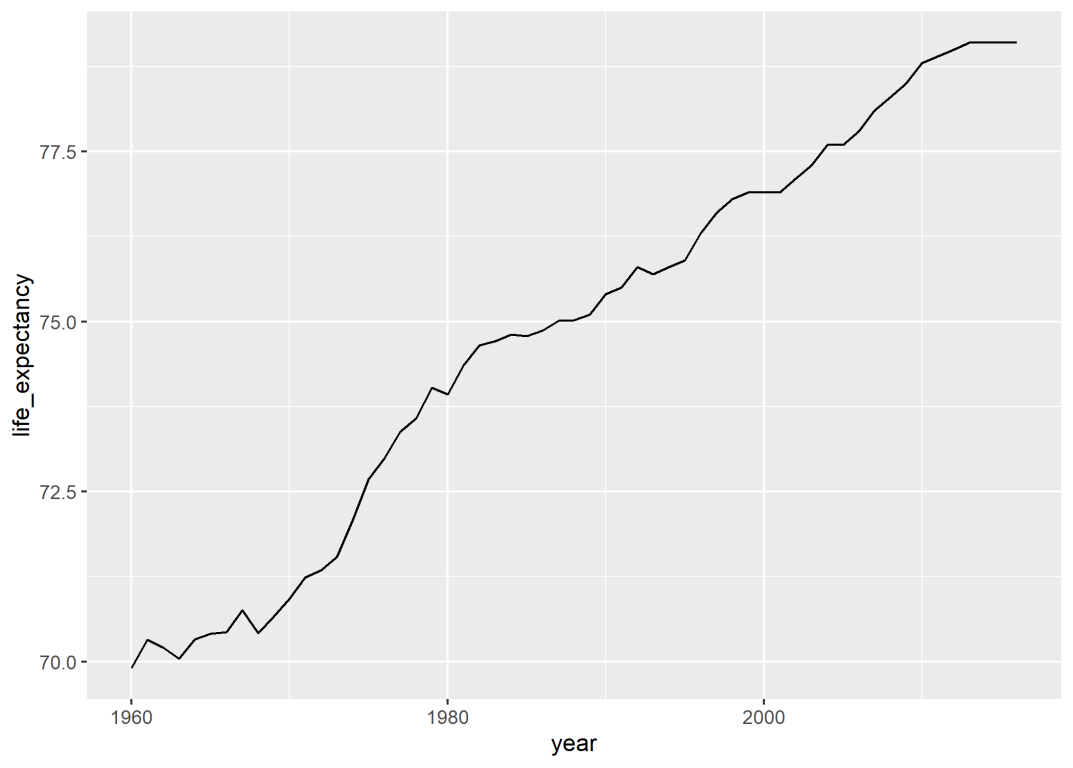
Workflow:

- filter the data to US observations
- initiate a plot
- add a line, plotting life expectancy over time

Line Chart (*geom_line()*)

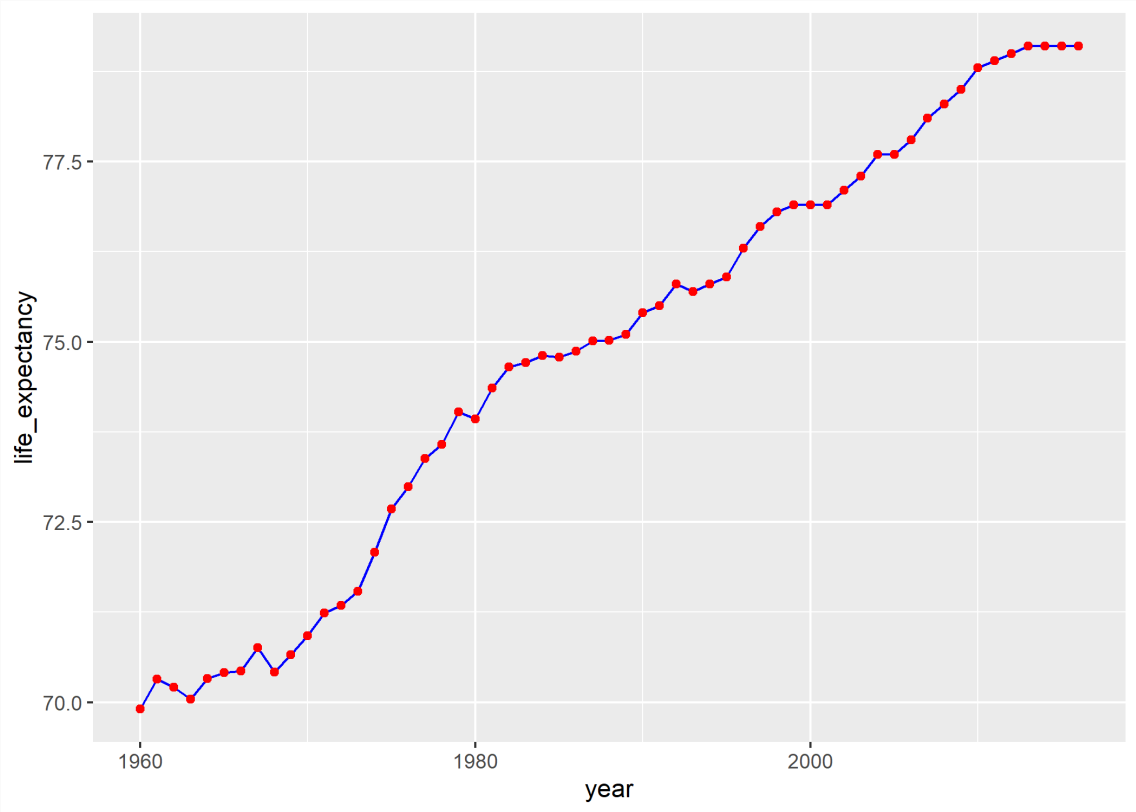
How has average life expectancy in the US evolved from 1960-2016?

```
filter(gap_ds, country == "United States") %>%
  ggplot()+
  geom_line(aes(x = year, y = life_expectancy))
```



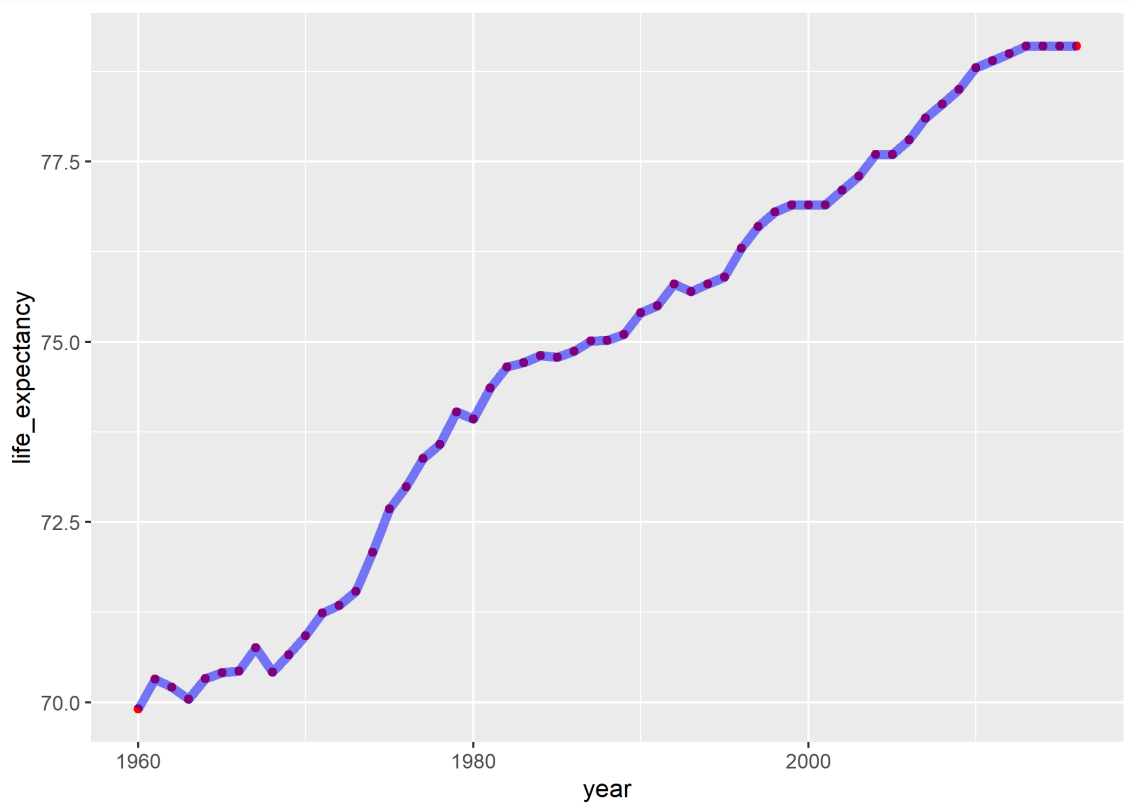
Points Connected with a Line

```
filter(gap_ds, country == "United States") %>%
  ggplot(aes(x = year, y = life_expectancy)) +
  geom_line(color = "blue") +
  geom_point(color = "red")
```



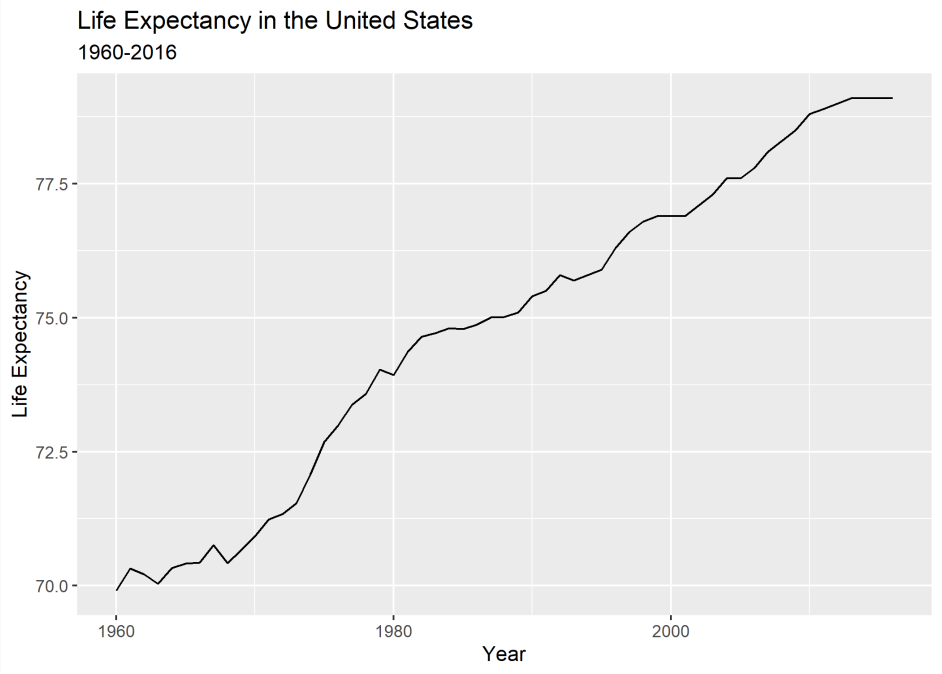
Layer Order: Lowest Layer on Top

```
filter(gap_ds, country == "United States") %>%
  ggplot(aes(x = year, y = life_expectancy)) +
  geom_point(color = "red") +
  geom_line(color = "blue", linewidth = 2, alpha = 0.5)
```



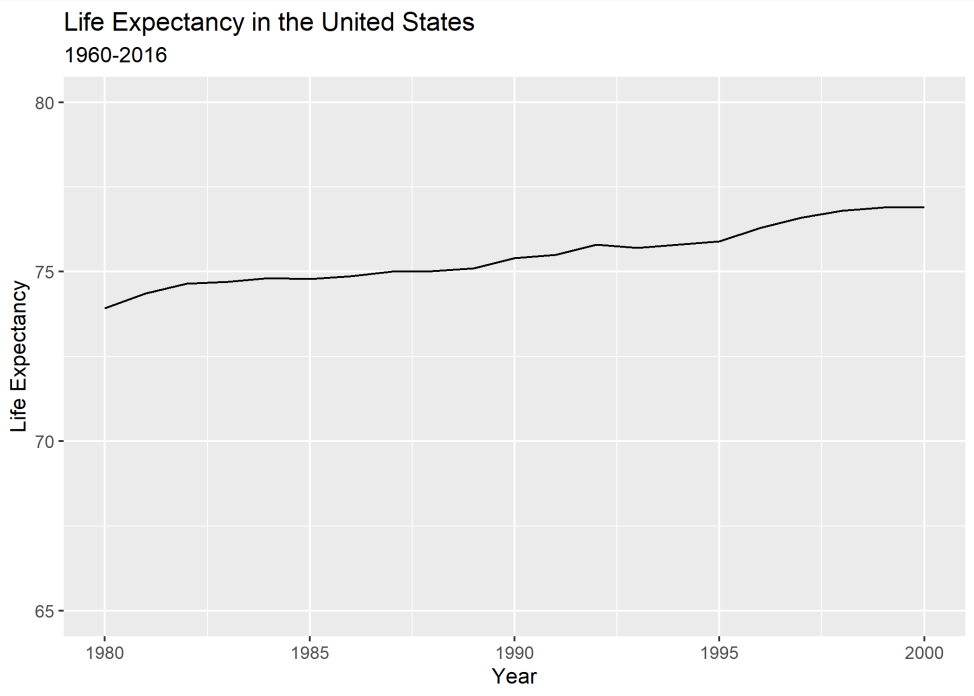
Modify Titles with *labs()*

```
filter(gap_ds, country == "United States") %>%
  ggplot() +
  geom_line(aes(x = year, y = life_expectancy)) +
  labs(title = "Life Expectancy in the United States",
       subtitle = "1960-2016",
       x = "Year",
       y = "Life Expectancy")
```



Change Axis Limits with *lims()*

```
filter(gap_ds, country == "United States") %>%
  ggplot() +
  geom_line(aes(x = year, y = life_expectancy)) +
  labs(title = "Life Expectancy in the United States",  subtitle = "1960-2
  lims(y = c(65, 80), x = c(1980, 2000))
```

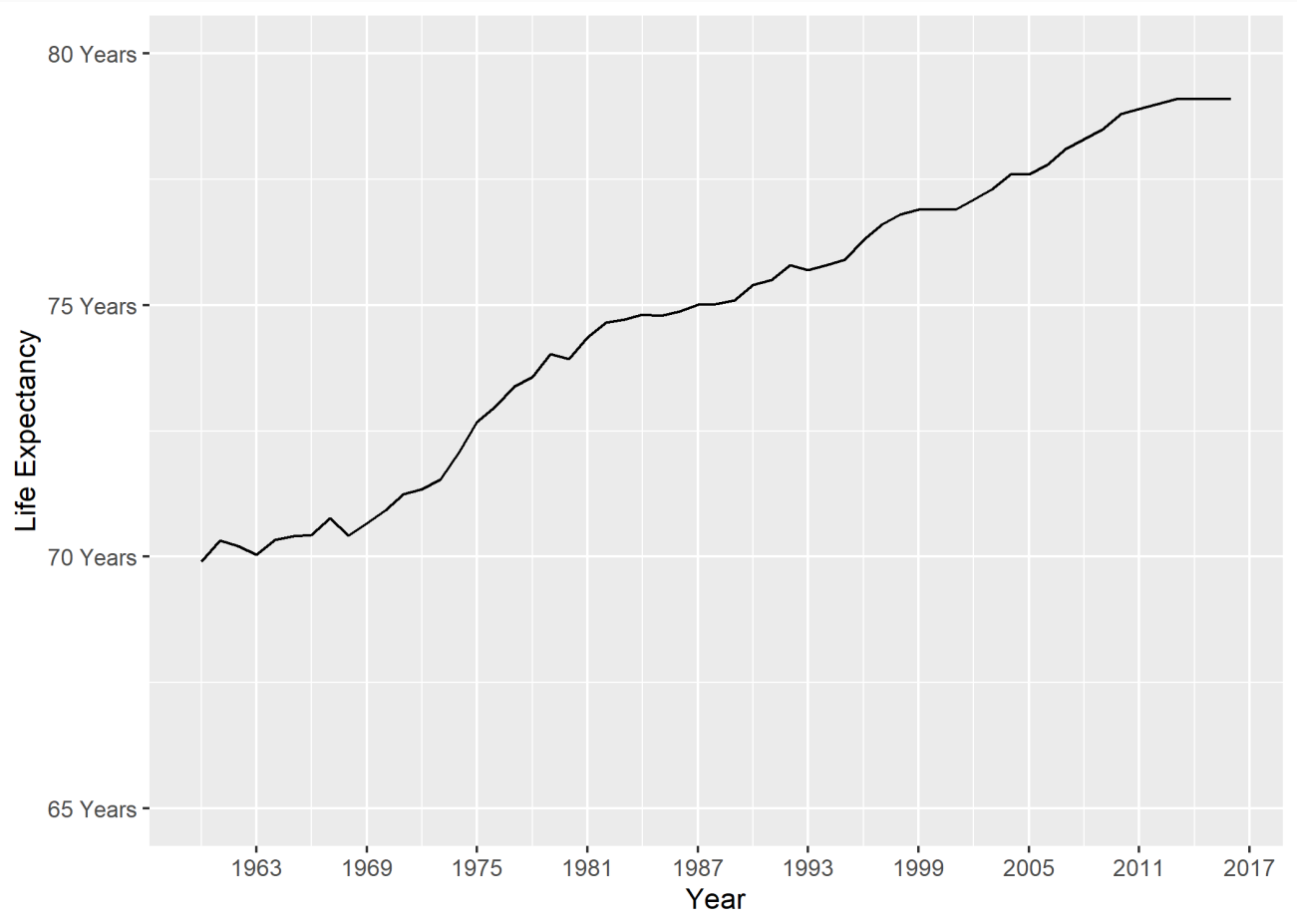


Customize Axis Scales

Change limits/breaks/labels for a **specific axis** with `scale_x/y_type()` layers (`type` one of "discrete", "continuous", "date", or "manual")

```
filter(gap_ds, country == "United States") %>%
  # add a date-formatted version of our year variable
  mutate(year_date = paste("01/01/",year) %>% mdy())  %>%
  ggplot() +
  geom_line(aes(x = year_date, y = life_expectancy)) +
  # Format continuous y axis
  scale_y_continuous(name = "Life Expectancy",
                     breaks = seq(65, 80, 5),
                     labels = paste0( seq(65, 80, 5), " Years"),
                     limits = c(65, 80)) +
  # format date x axis
  scale_x_date(name = "Year",
               date_breaks = "6 years",
               date_labels = "%Y")
```

Customize Axis Scales: The Chart



Mappings: Multiple Series

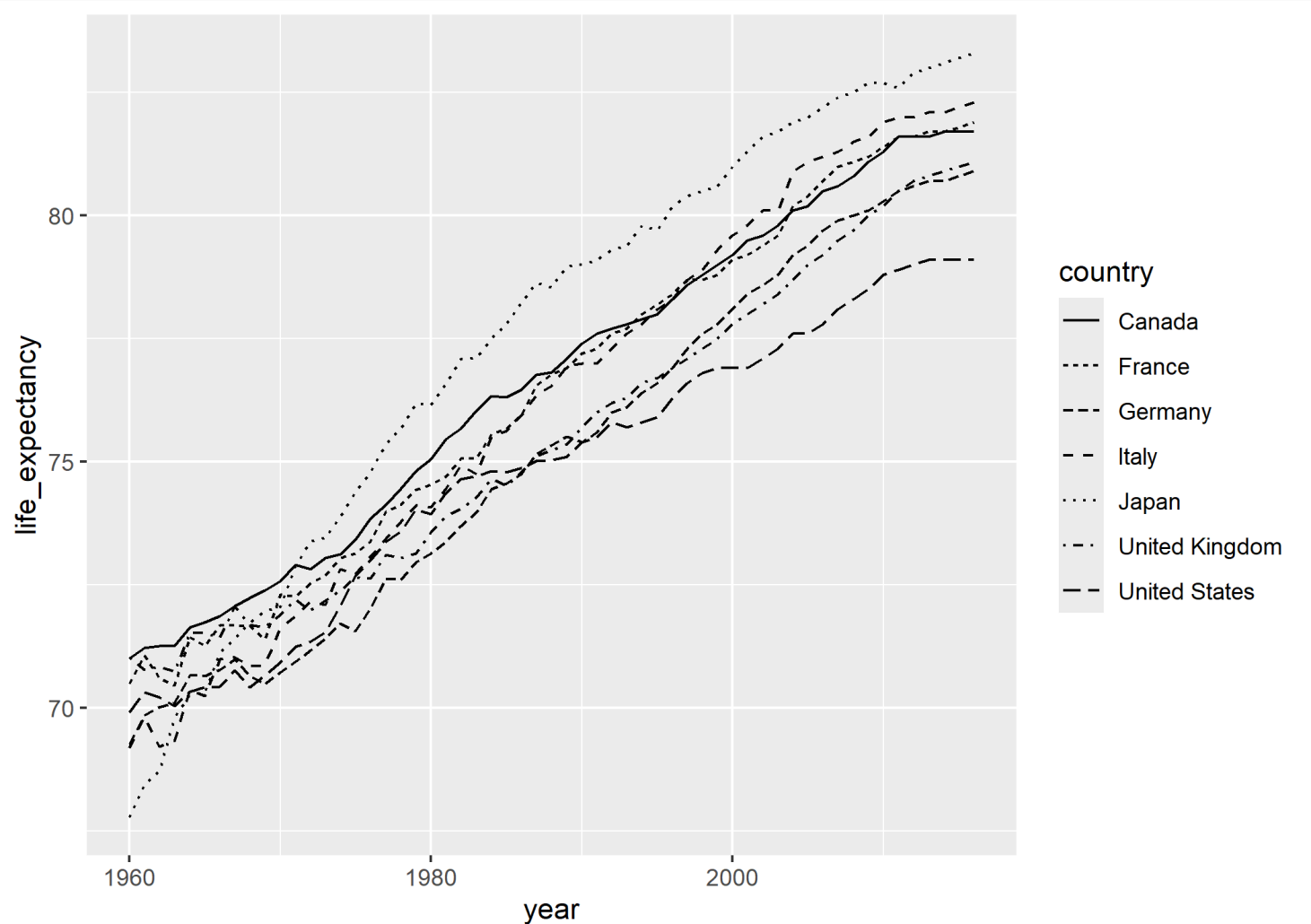
Add a `linetype` or `color` aesthetic within `aes()` to add a **separate line for each value of a categorical variable**.

What are trends in life expectancy for each of the G-7 countries?

```
g7 ← c("United States", "United Kingdom", "Germany", "Italy", "France",
#filter to G-7 countries
filter(gap_ds, country %in% g7) %>%
  ggplot() +
  geom_line(
    aes(x = year,
        y = life_expectancy,
        linetype = country)
  )
```

Mappings: Multiple Series (*linetype*)

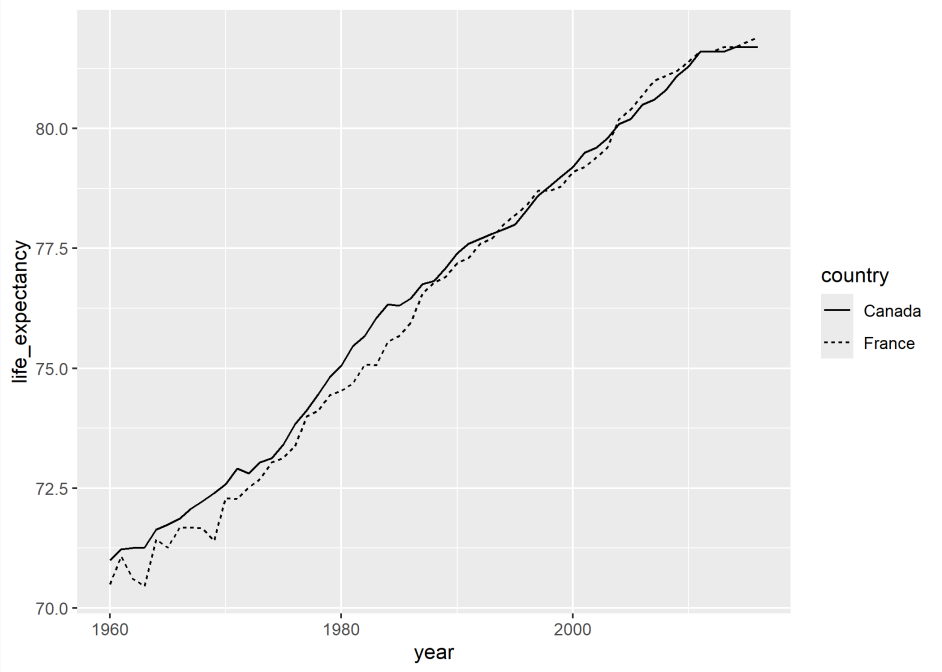
What are trends in life expectancy for the G-7 countries?



Mappings: Multiple Series (Manually)

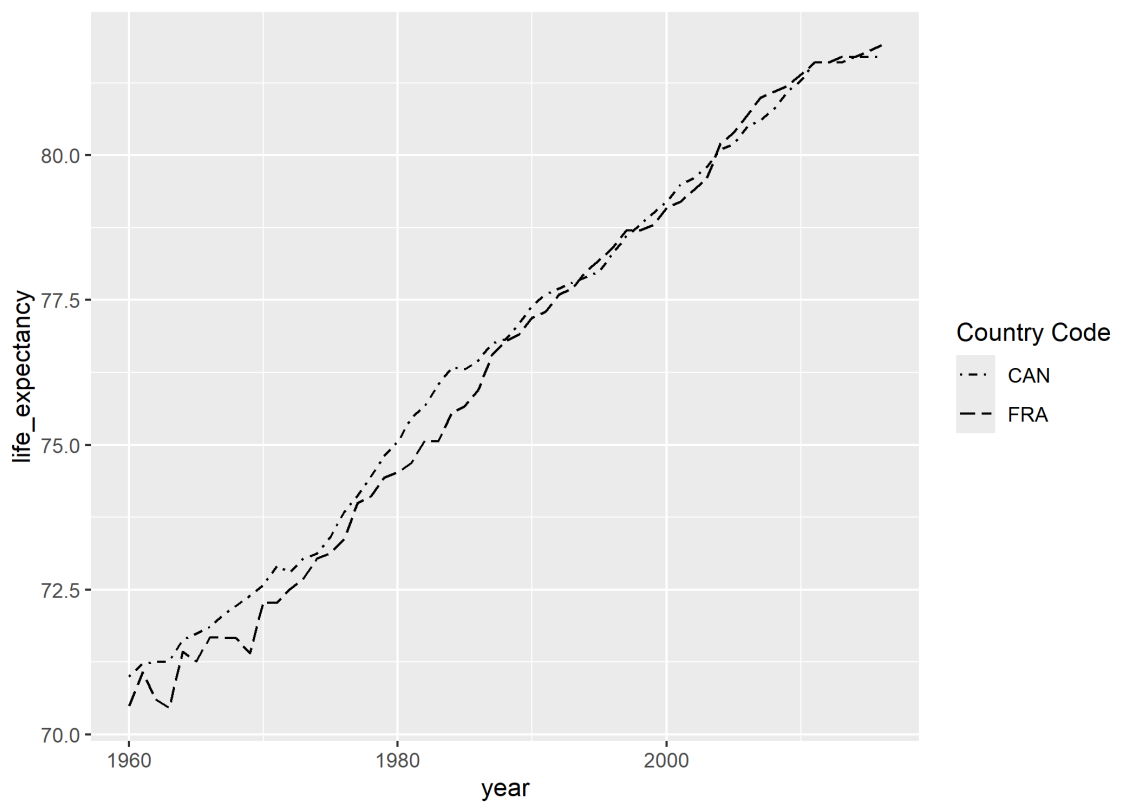
Use `scale_type_manual()` if you want to **customize the legend labels/values**:

```
ca_fr ← filter(gap_ds, country %in% c("Canada", "France")) %>%
  ggplot() +
  geom_line(aes(x = year, y = life_expectancy, linetype = country))
ca_fr
```



Mappings: Multiple Series (Manually)

```
ca_fr +
  scale_linetype_manual(
  labels = c("CAN", "FRA"),
  values = c("dotdash", "longdash"),
  name = "Country Code")
```



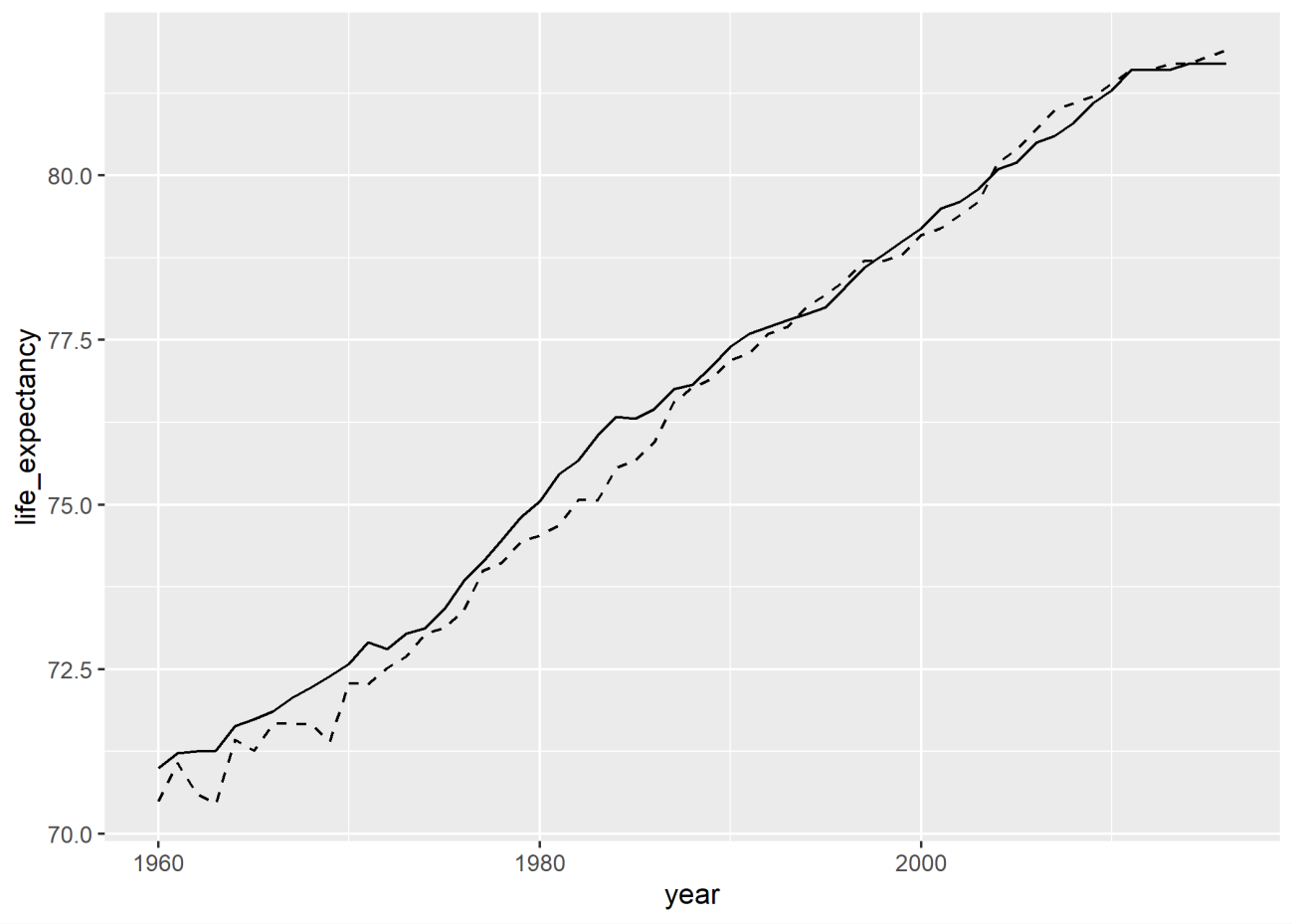
Mappings: Multiple Data Objects

When we have **multiple data objects**, we can set the line type (or color/fill/size) to specific values **outside of** `aes()`.

```
gap_can ← filter(gap_ds, country %in% c("Canada"))
gap_fra ← filter(gap_ds, country %in% c("France"))
ggplot() +
  geom_line(aes(x = year, y = life_expectancy),
            data = gap_can, linetype = "solid") +
  geom_line(aes(x = year, y = life_expectancy),
            data = gap_fra, linetype = "dashed")
```

Mappings: Multiple Data Objects

When we have **multiple data objects**, we can set the line type (or color/fill/size) to specific values **outside of** `aes()`.



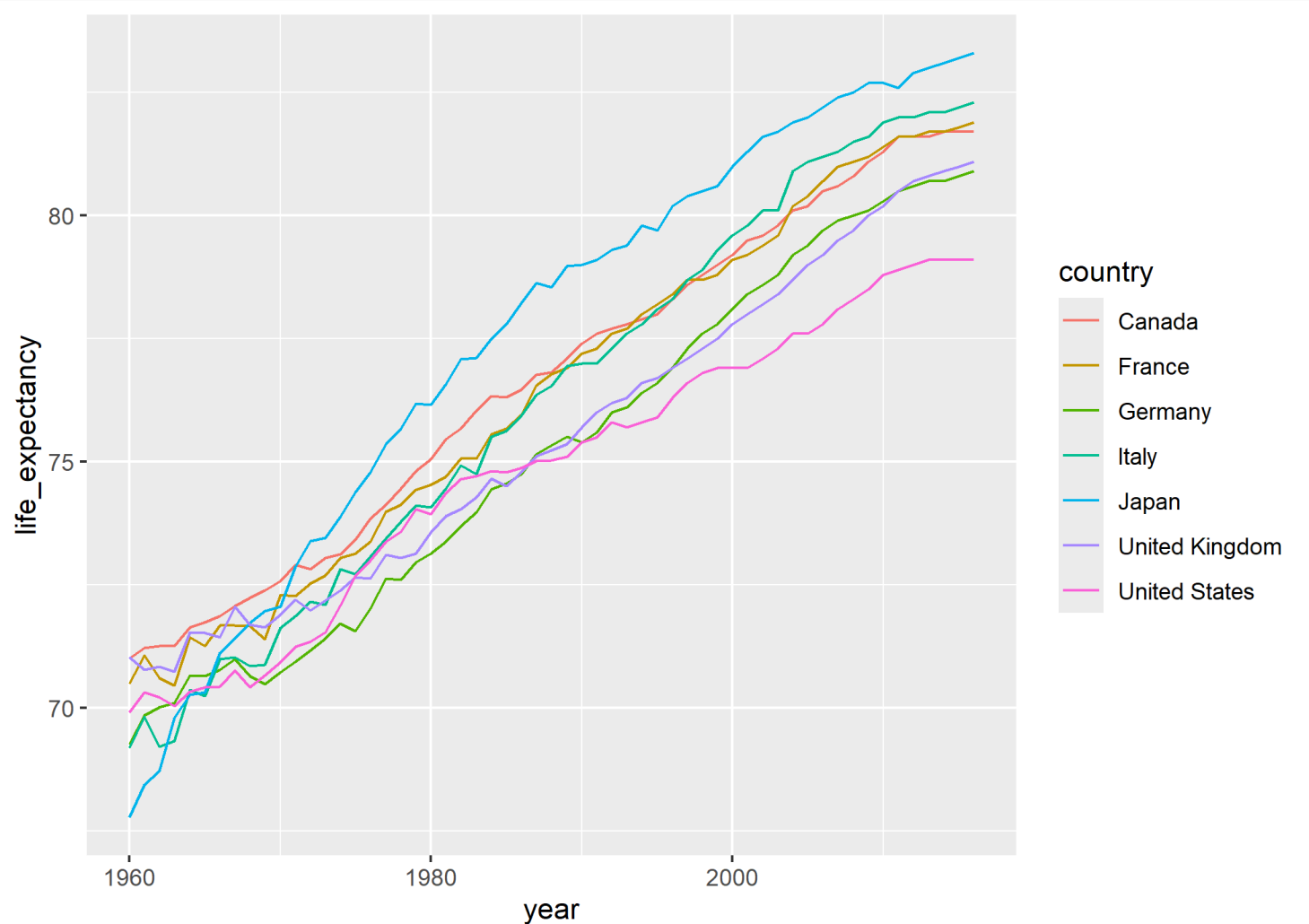
Mappings: Multiple Series (*color*)

We can similarly set a different color for each country:

```
# filter to G-7 countries
filter(gap_ds, country %in% g7) %>%
  ggplot() +
  geom_line(aes(x = year, y = life_expectancy,
                color = country
  ))
```

Mappings: Multiple Series (*color*)

We can similarly set a different color for each country:



Adding Text Labels

Text labels are generally more effective than legends

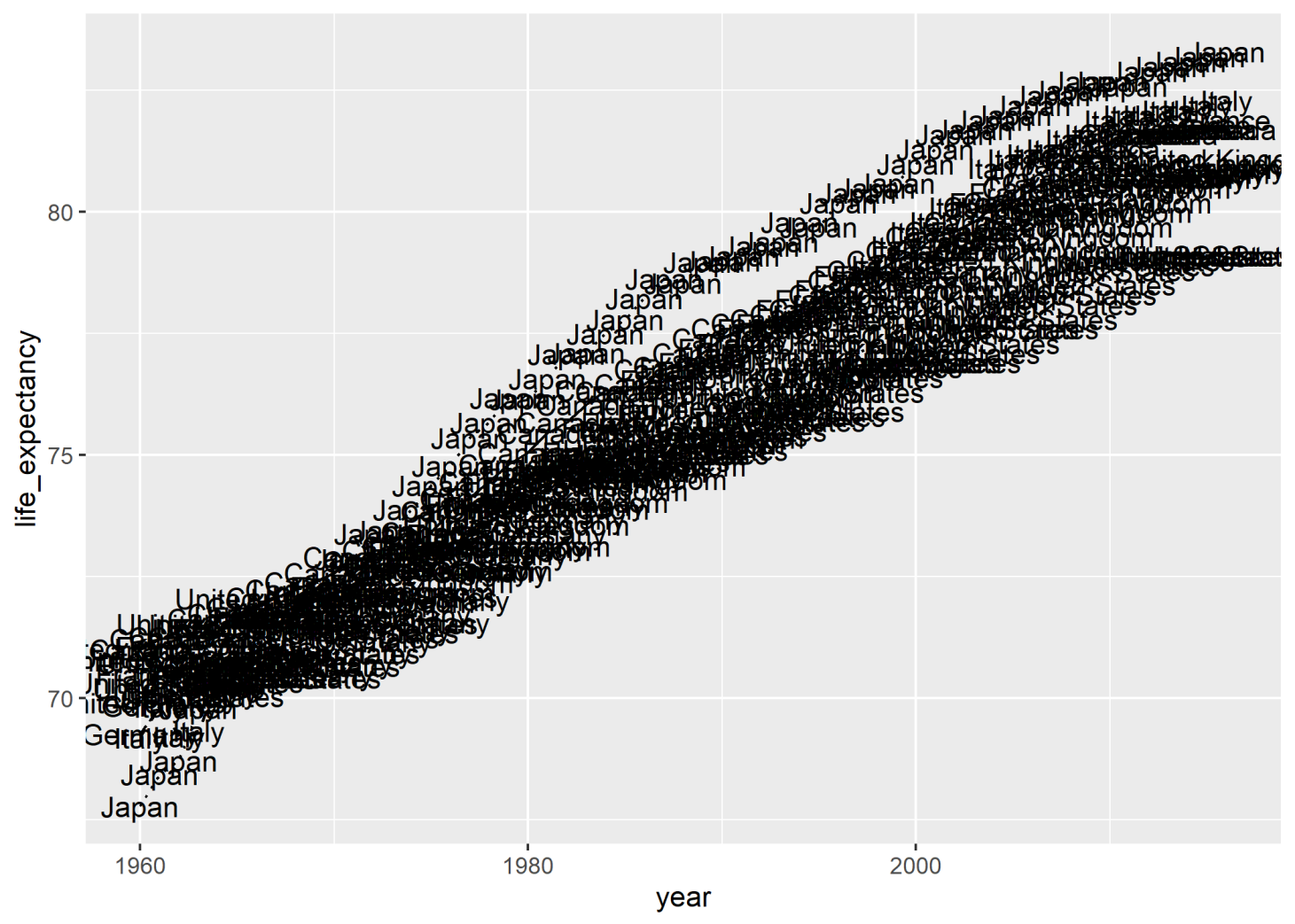
- `geom_text` from **ggplot** or `geom_text_repel` from **ggrepel** for text
- `geom_label_repel` from **ggrepel** for text + boxes

Try adding text labels to our chart:

```
# filter to G-7 countries
filter(gap_ds, country %in% g7) %>%
  ggplot(aes(x = year, y = life_expectancy)) +
  geom_line(aes(linetype = country)) +
  geom_text(aes(label = country)) +
  theme(legend.position = "none")
```

Adding Text Labels

Try adding text labels to our chart:



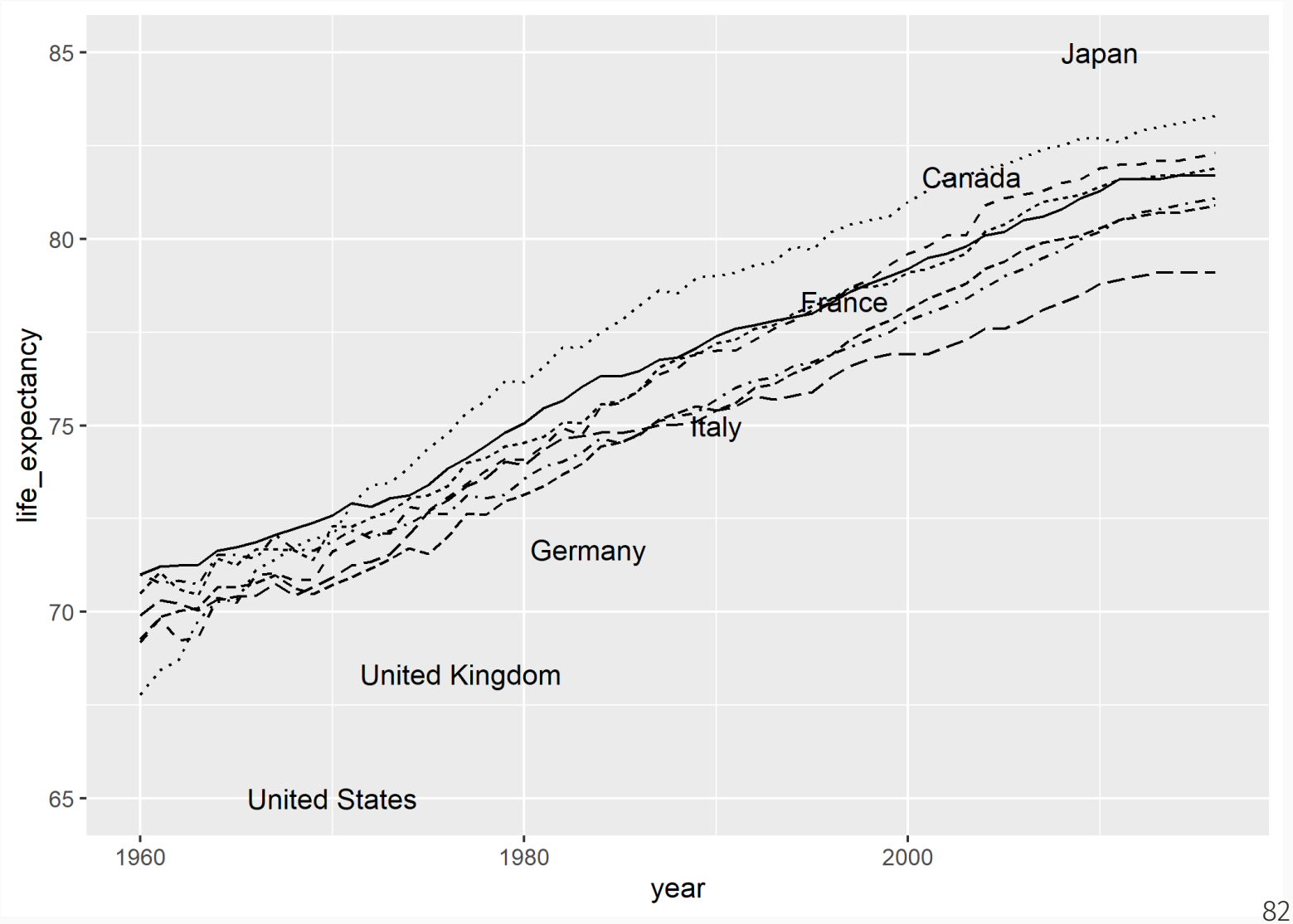
Adding Text Labels (At Specific Points)

Whoops! By default `geom_text()` adds text labels **at every data point.**

To add labels at **specific points**, we can **manually define the x/y locations** for `geom_text`

```
g7_labs ← data.frame(country = g7,
                     x = seq(1970, 2016, length.out = length(g7)),
                     y = seq(65, 85, length.out = length(g7)))

# filter to G-7 countries
filter(gap_ds, country %in% g7) %>%
  ggplot(aes(x = year, y = life_expectancy)) +
  geom_line(aes(linetype = country)) +
  geom_text(data = g7_labs, # switch data object
            aes(x=x, y=y, label = country)) +
  theme(legend.position = "none")
```

Adding Text Labels (At Specific Points)



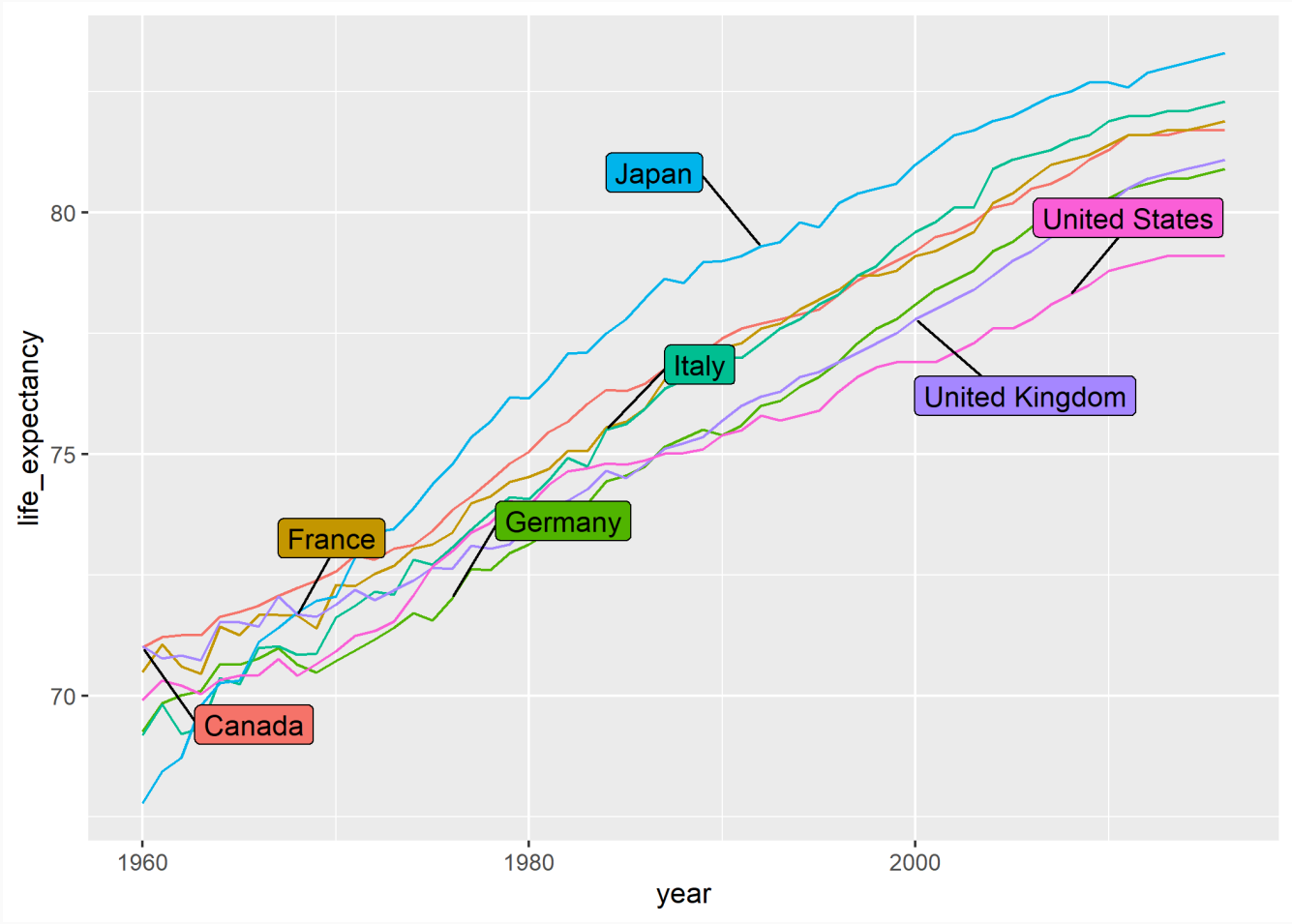
Adding Text Labels (At Specific Points)

Add text + boxes with customizable "repel" using `geom_label_repel()` from **ggrepel**

```
# get a label x/y point for each country
g7_labs ← filter(gap_ds, country %in% g7) %>%
  mutate(order = rep(seq(1:7), 57)) %>%
  filter(year == 1960+8*(order-1))

filter(gap_ds, country %in% g7) %>%
  ggplot(aes(x = year, y = life_expectancy)) +
  geom_line(aes(color = country)) +
  geom_label_repel(data = g7_labs, # switch data object
                   aes(label = country, fill = country),
                   box.padding = 1.5,
                   min.segment.length = unit(0, "lines")) +
  theme(legend.position = "none")
```

Adding Text Labels (At Specific Points)

Add text + boxes with customizable "repel" using `geom_label_repel()` from **ggrepel**



Mappings: Legend for Multiple Data

When we have **multiple data objects** and want to **add a legend**,

- Add desired label names as the mapping within `aes()`
- e.g. `linetype = "Label 1"`
- Using the `values` argument of the appropriate `scale_type_manual()` function, define a vector following the syntax

```
c("Label 1" = "value1", "Label 2" = "value2", ...)
```

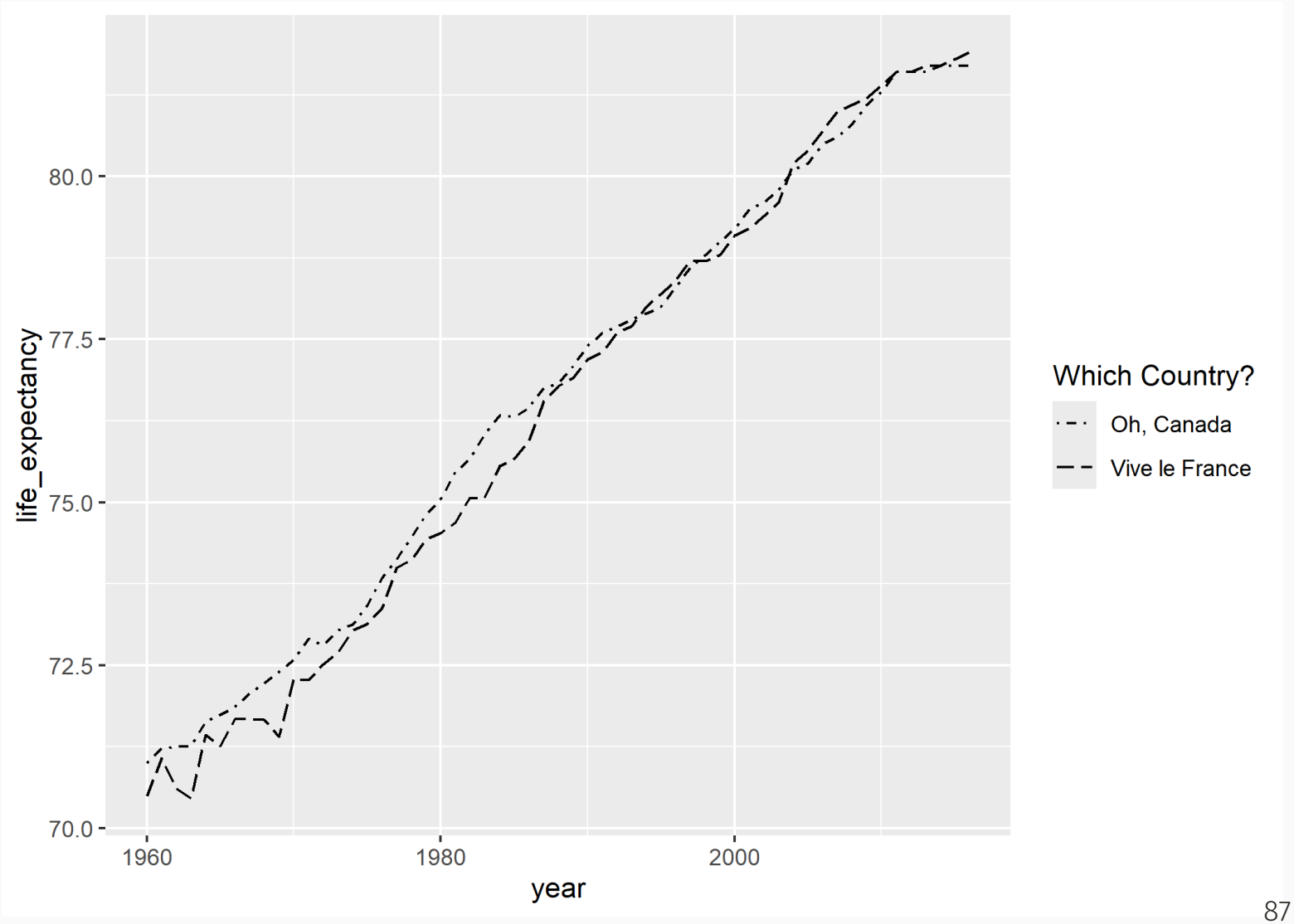
Legend for Multiple Objects

When we have **multiple data objects** and want to **add a legend**,

- Add desired label names as the mapping within `aes()`
- e.g. `linetype = "Label 1"`
- Using the `values` argument of the appropriate `scale_type_manual()` function, define a vector following the syntax

```
ggplot() +
  geom_line(aes(x = year, y = life_expectancy, linetype = "Oh, Canada"), c
  geom_line(aes(x = year, y = life_expectancy, linetype = "Vive le France"
  scale_linetype_manual(
    name = "Which Country?",
    values = c("Oh, Canada" = "dotdash", "Vive le France" = "longdash"))
```

Legend for Multiple Objects



Legend for Multiple Objects

Doing it for color:

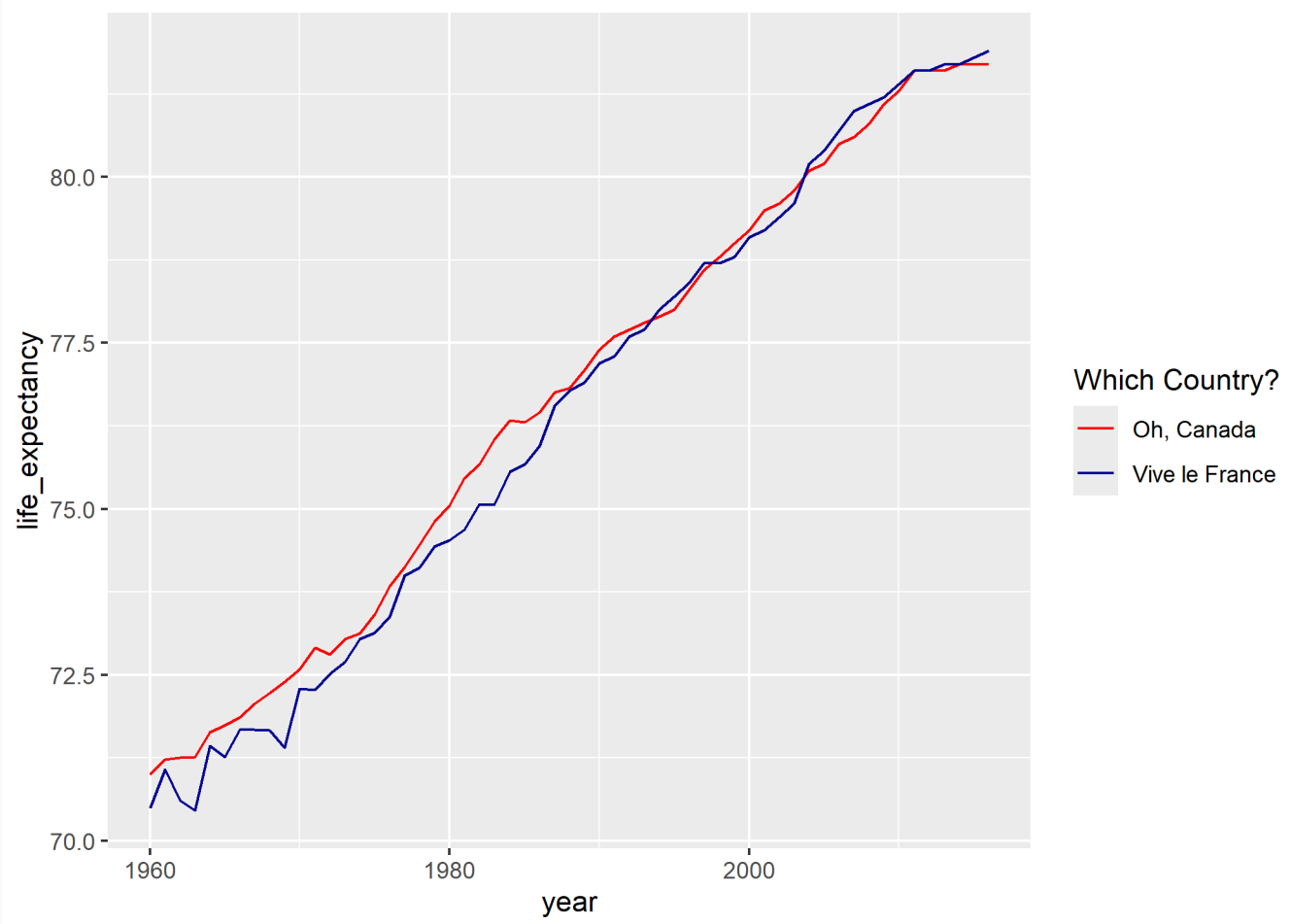
- Can specify colors by name for named ggplot colors, or by hex code

```
ggplot() +
  geom_line(aes(x = year, y = life_expectancy, color = "Oh, Canada"), data
  geom_line(aes(x = year, y = life_expectancy, color = "Vive le France"),
  scale_color_manual(name = "Which Country?",
                     values = c("Oh, Canada" = "#FF0000", "Vive le France"
```

Legend for Multiple Objects

Doing it for color:

- Can specify colors by name for named ggplot colors, or by hex code



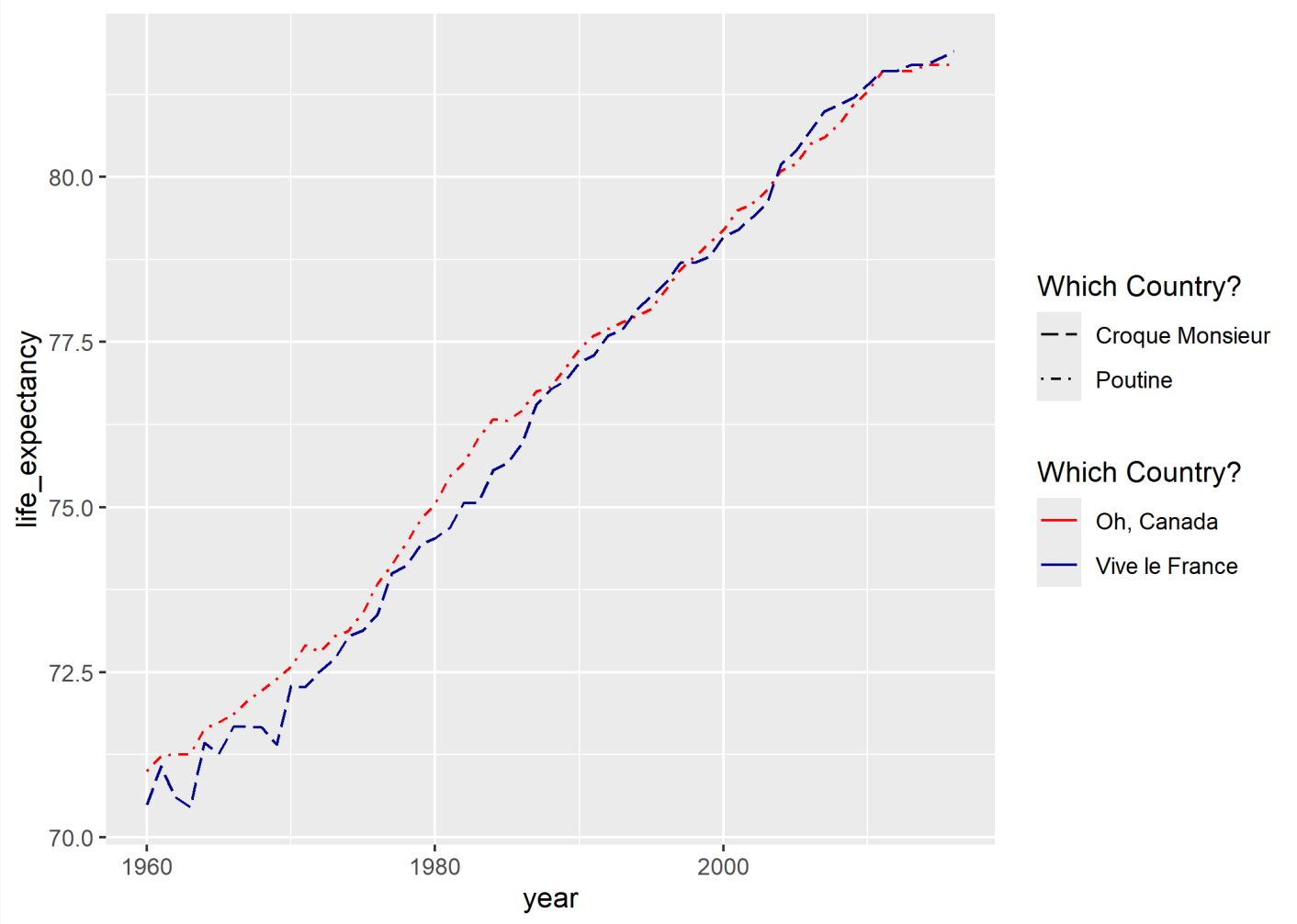
Mappings: Multiple Legends

Can we combine both? *of course* we can!

```
ggplot() +
  geom_line(aes(x = year, y = life_expectancy,
                color = "Oh, Canada", linetype = "Poutine"), data = gap_ca
  geom_line(aes(x = year, y = life_expectancy,
                color = "Vive le France", linetype = "Croque Monsieur"), d
  scale_color_manual(name = "Which Country?",
                     values = c("Oh, Canada" = "#FF0000", "Vive le France"
  scale_linetype_manual(
    name = "Which Country?",
    values = c("Poutine" = "dotdash", "Croque Monsieur" = "longdash"))
```

Mappings: Multiple Legends

Can we combine both? *of course* we can!



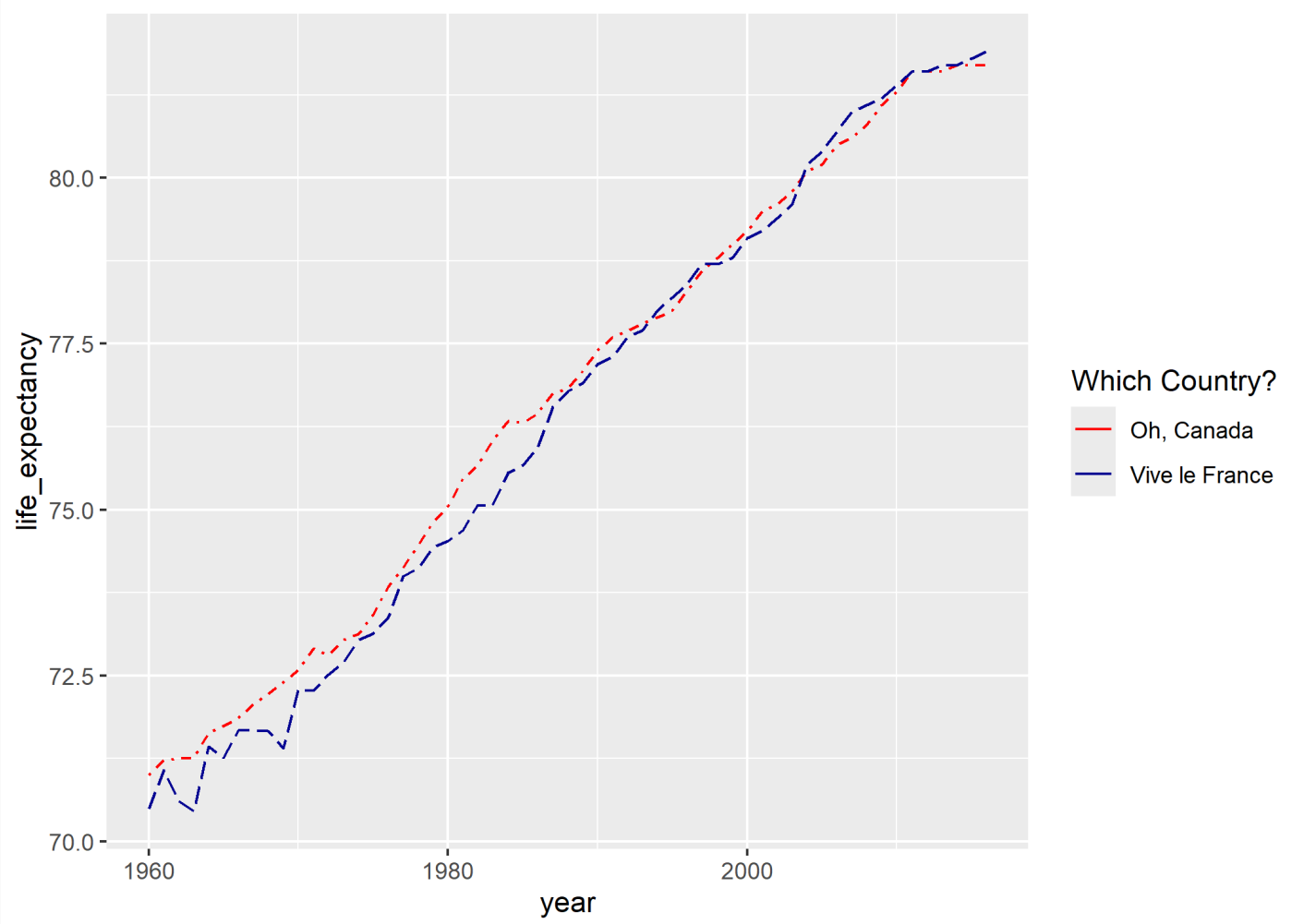
Mappings: Multiple Legends

The two legends are redundant, so **remove unwanted legends** with `guides(name = "none")`:

```
ggplot() +
  geom_line(aes(x = year, y = life_expectancy,
                color = "Oh, Canada", linetype = "Poutine"), data = gap_ca
  geom_line(aes(x = year, y = life_expectancy,
                color = "Vive le France", linetype = "Croque Monsieur"), d
  scale_color_manual(name = "Which Country?",
                     values = c("Oh, Canada" = "#FF0000", "Vive le France"
  scale_linetype_manual(
    name = "Which Country?",
    values = c("Poutine" = "dotdash", "Croque Monsieur" = "longdash")) +
  guides(linetype = "none")
```

Mappings: Multiple Legends

The two legends are redundant, so **remove unwanted legends** with `guides(name = "none")`:



Mappings: Combine Multiple Legends

Combine multiple legends within `guides()` using the syntax

```
AES_INC = guide_legend(override.aes = list(AES_EXC = c(VALUES)))
```

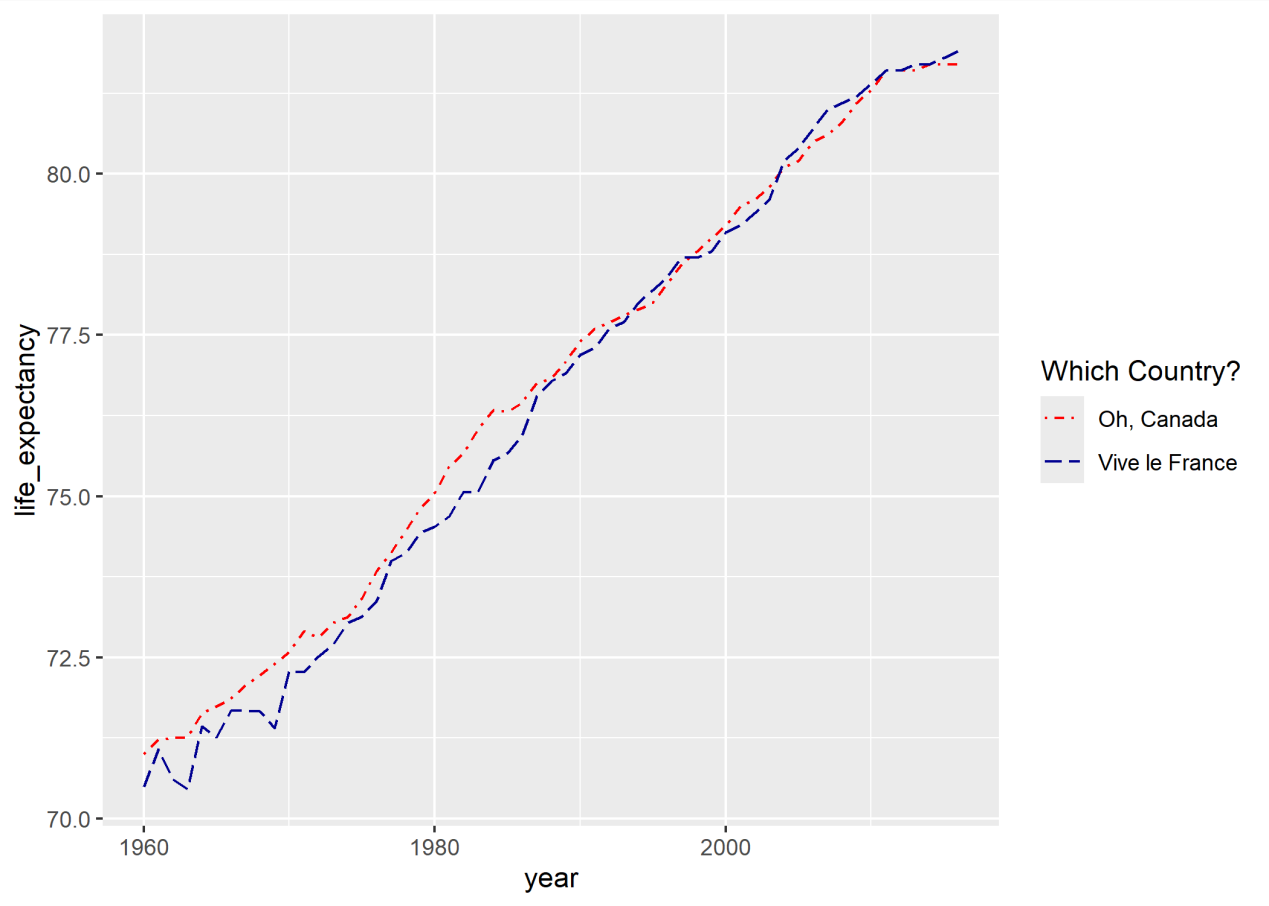
- `AES_INC` is the aesthetic whose legend you're **keeping**
- `AES_EXC` is the aesthetic whose legend you're **excluding**
- `VALUES` are the **aesthetic values** to change the included guide to

```
ggplot() + geom_line(aes(x = year, y = life_expectancy, color = "Oh, Cana
  guides(linetype = "none", # hide the linetype legend
         color = guide_legend(
           override.aes = list(linetype = c("dotdash", "longdash")))
         )
```

Mappings: Combine Multiple Legends

Combine multiple legends within `guides()` using the syntax

```
AES_INC = guide_legend(override.aes = list(AES_EXC = c(VALUES)))
```

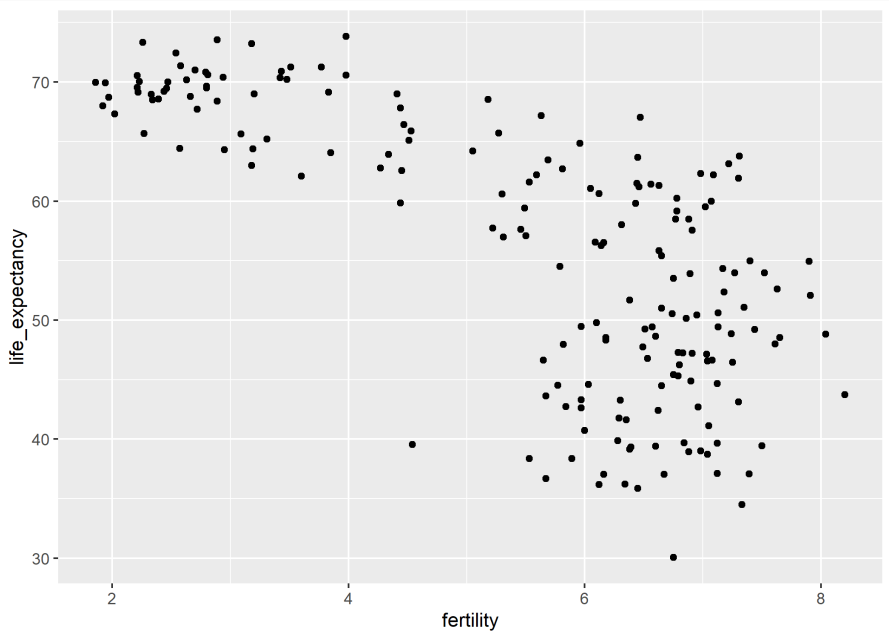


Scatterplot (*geom_point()*)

How have fertility rates and life expectancies co-evolved over time?

First, plotting the data from 1962:

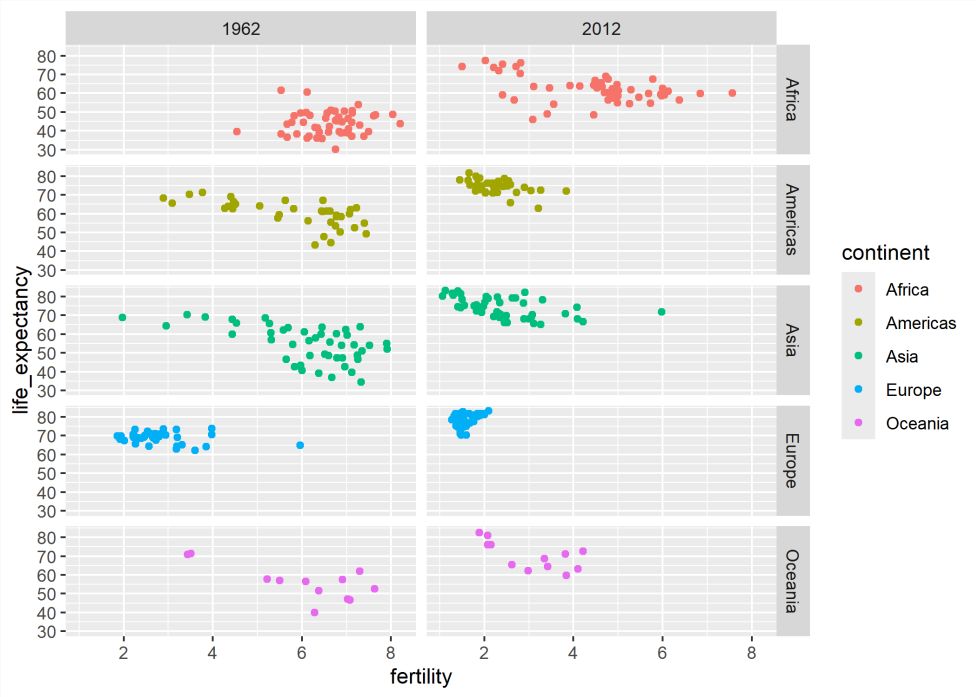
```
filter(gap_ds, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy)) + # omitting x/y = since in exp
  geom_point()
```



Faceting

Stratify (by continent, and compare 1962 to 2012) with `facet_grid()`:

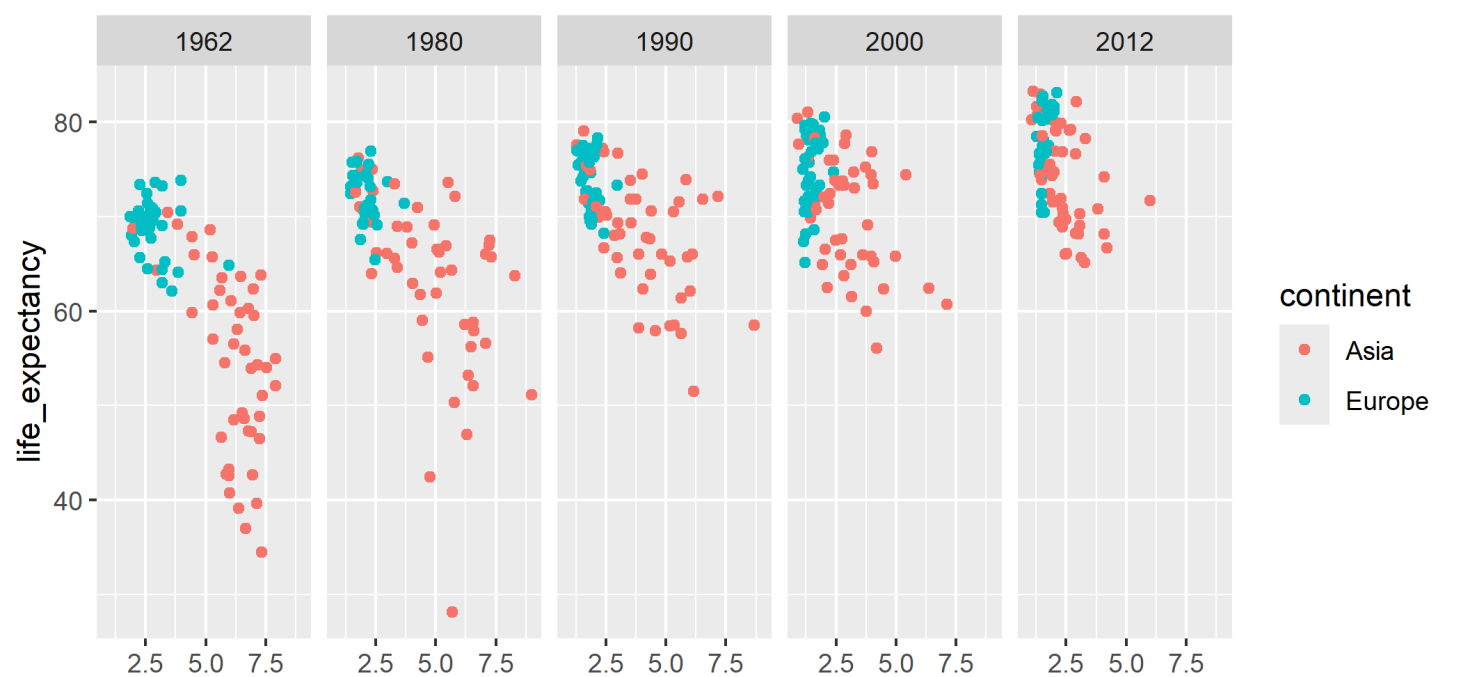
```
filter(gap_ds, year %in% c(1962, 2012)) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_grid(continent ~ year)  # "row ~ column"
```



Faceting

Show Europe vs. Asia for 5 different years:

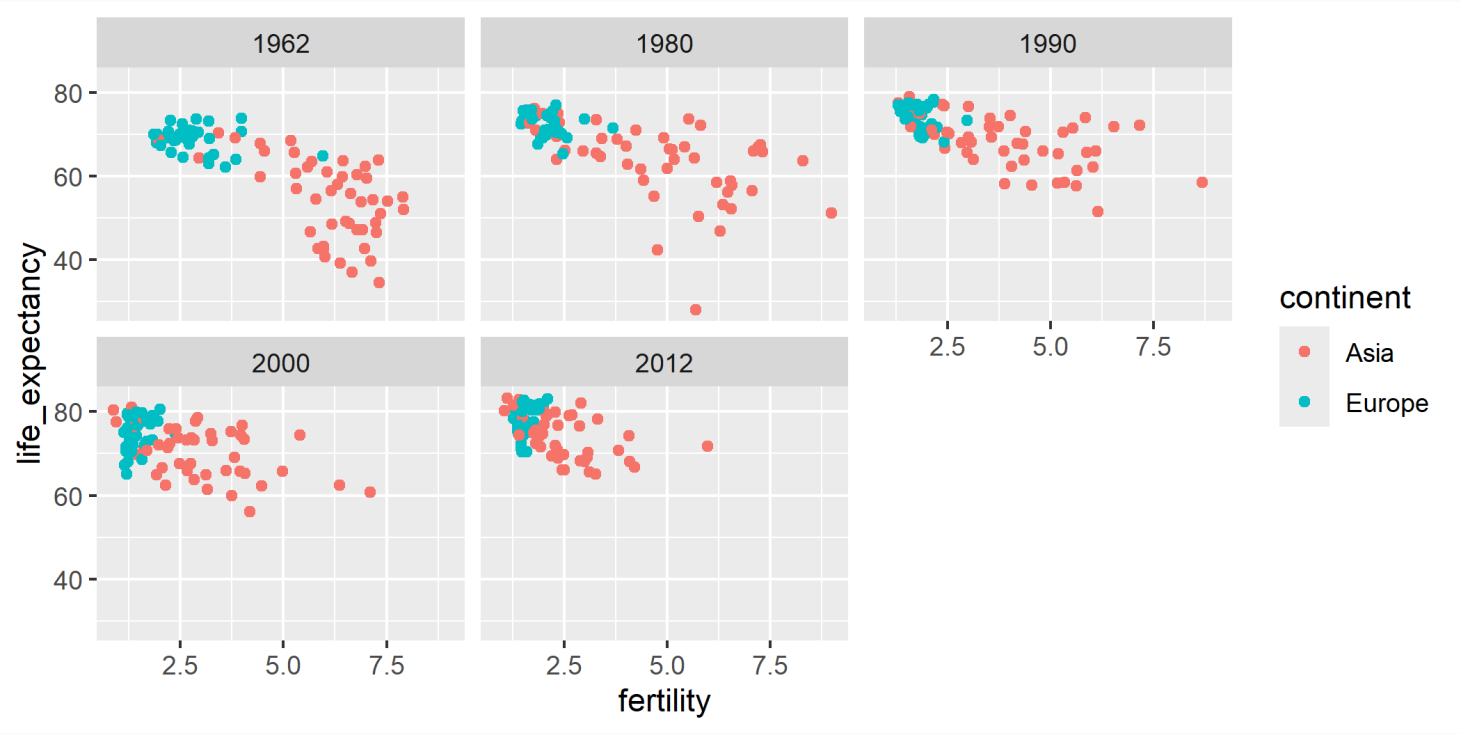
```
years ← c(1962, 1980, 1990, 2000, 2012)
continents ← c("Europe", "Asia")
gap_ds %>%
  filter(year %in% years & continent %in% continents) %>%
  ggplot( aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_grid(. ~ year)
```



Faceting

Too narrow? Wrap rows with `facet_wrap()`:

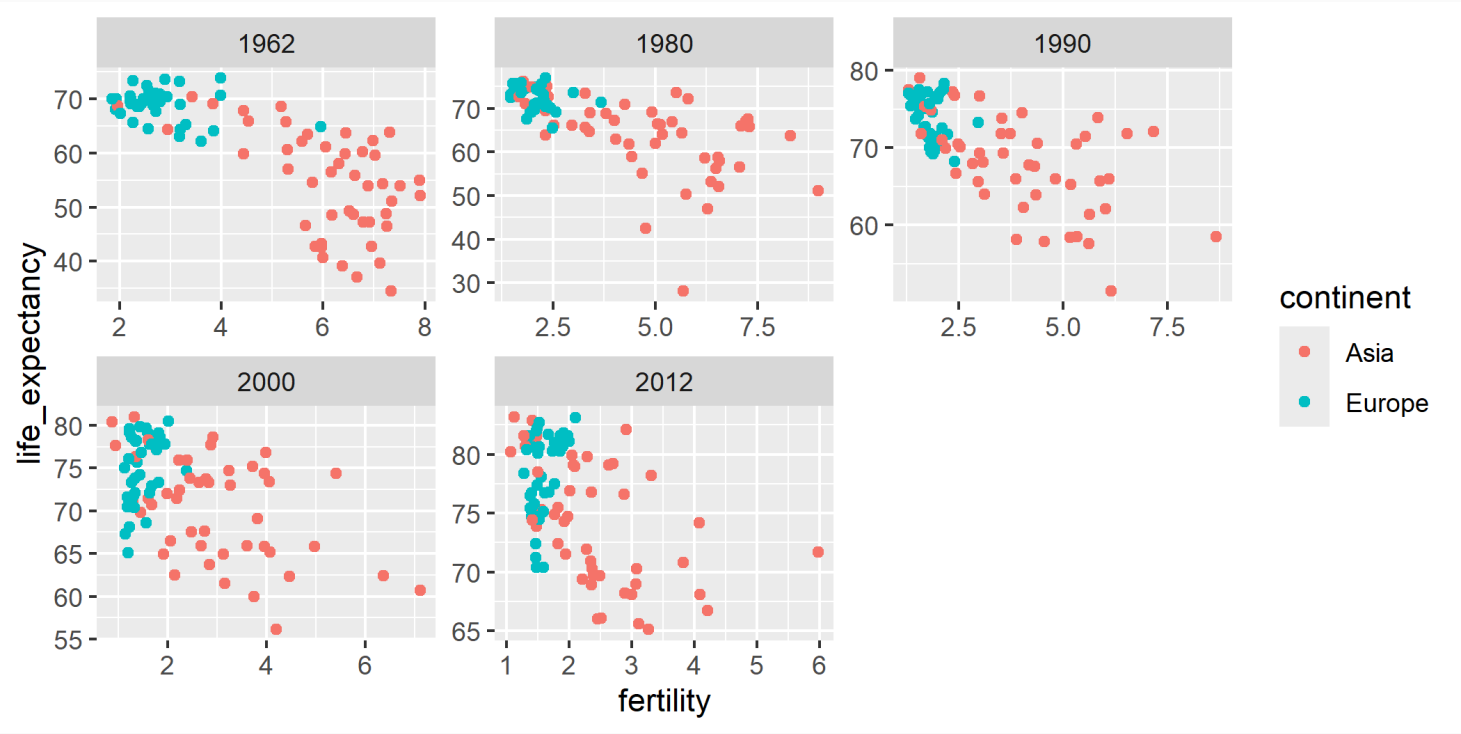
```
gap_ds %>%
  filter(year %in% years & continent %in% continents) %>%
  ggplot( aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_wrap(~year, nrow = 2) # can also use ncol
```



Faceting

An important thing `facet_` gives us is **common axis scales**. Otherwise graphs look like this:

```
gap_ds %>%
  filter(year %in% years & continent %in% continents) %>%
  ggplot( aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_wrap(~year, scales = "free")
```



Cleveland Dot Plots

Cleveland dot plots are uncluttered and can be **more effective** than **bar/column charts**.

- Especially when the x-intercept doesn't mean much.
- Or when plotting multiple values per category.

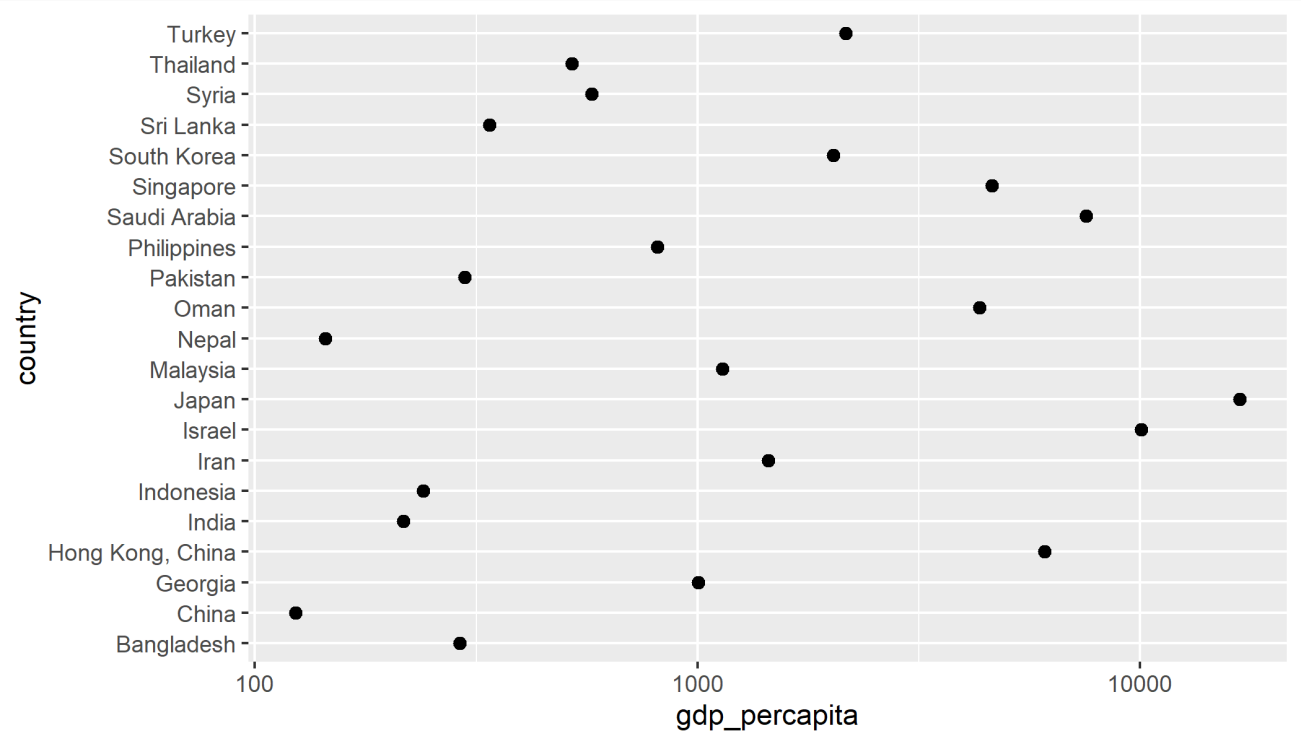
Structure:

- Categorical variable on y axis (easiest with factor)
- Points as values on x axis

```
gap_ds %>% mutate(gdp_percapita = gdp/population) %>%
  filter(year == 1970 & !is.na(gdp_percapita) & continent=="Asia") %>%
  ggplot(aes(gdp_percapita, country)) +
    geom_point(size=2) +
    scale_x_log10() # log 10 scale for ease of viewing
```

Cleveland Dot Plots

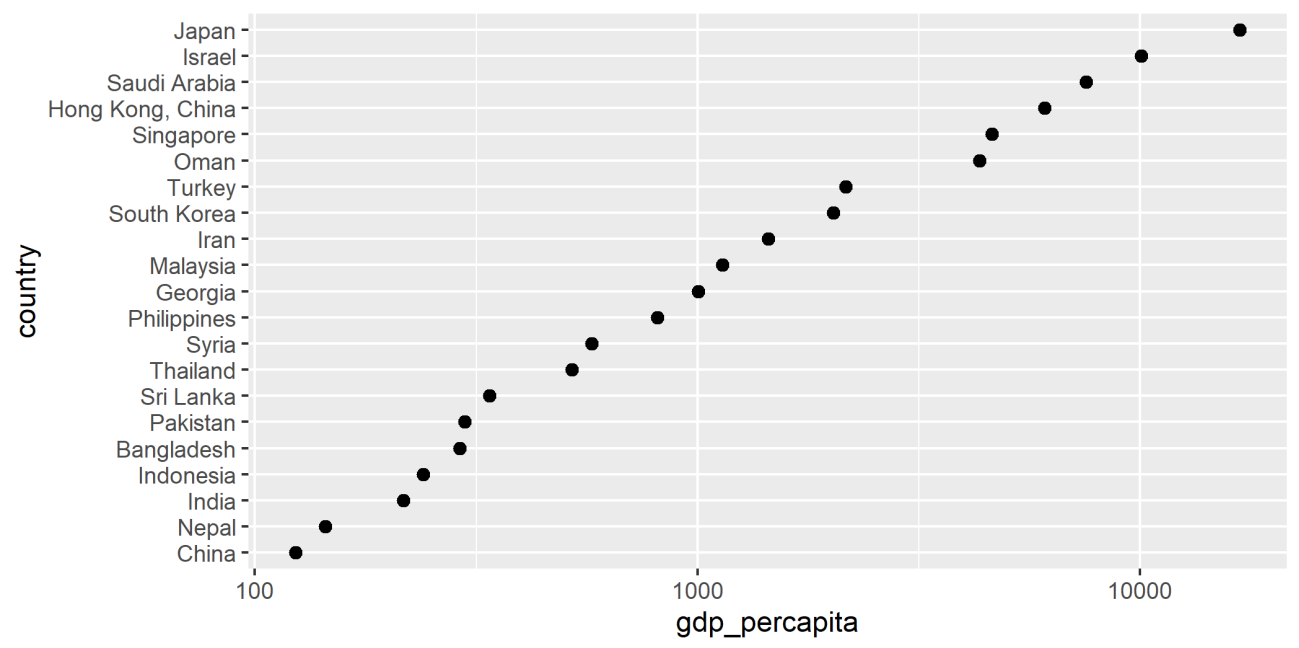
Cleveland dot plots are uncluttered and can be **more effective** than **bar/column charts**.



Cleveland Dot Plots

Use `reorder()` to conditionally reorder a factor (i.e. countries by GDP)

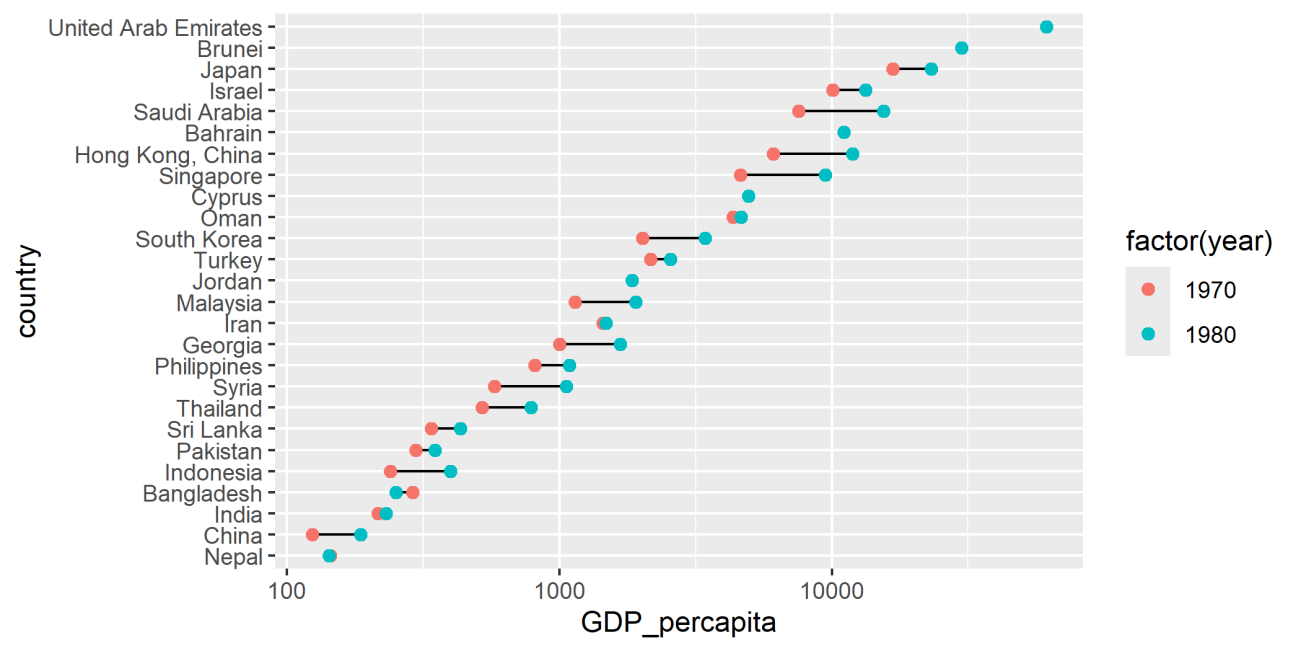
```
gap_ds %>% mutate(gdp_percapita = gdp/population) %>%
  filter(year == 1970 & !is.na(gdp_percapita) & continent=="Asia") %>%
  mutate(country = reorder(country, gdp_percapita)) %>%
  ggplot(aes(gdp_percapita, country)) +
    geom_point(size=2) +
    scale_x_log10()
```



Cleveland Dot Plots

Also useful when plotting **multiple values per category**.

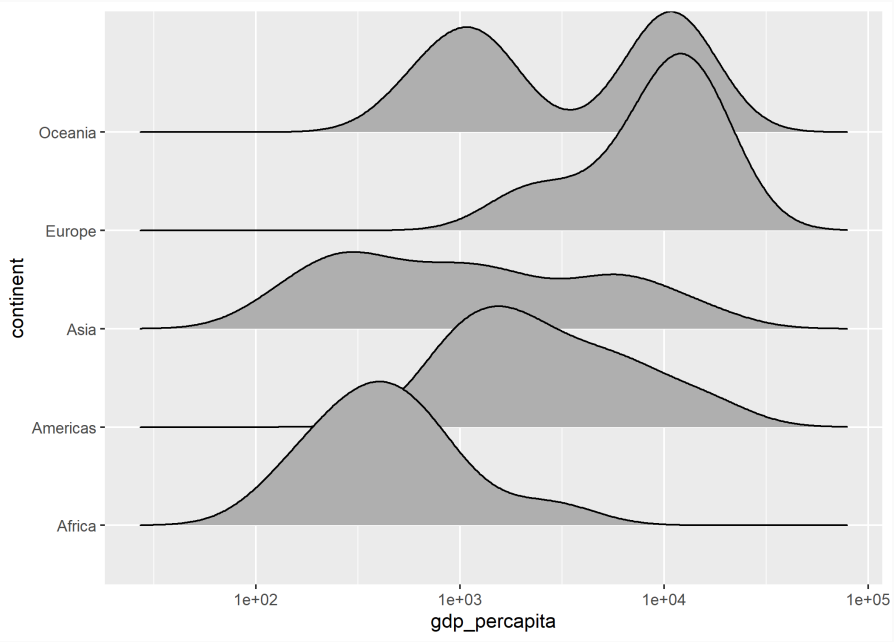
```
gap_ds %>% mutate(GDP_percapita = gdp/population) %>%
  filter(year %in% c(1970, 1980) & !is.na(GDP_percapita) & continent=="Asia") %>%
  mutate(country = reorder(country, GDP_percapita)) %>%
  ggplot(aes(GDP_percapita, country)) +
    geom_line(aes(group = country)) + # add line connecting gdp values per country
    geom_point(size=2, aes(color = factor(year))) + # add point for 1970 + 1980 values on top of l
    scale_x_log10()
```



Ridge Plots

Using **ggridges** for staggered densities (a la **Joy Division's "Unknown Pleasures"**)

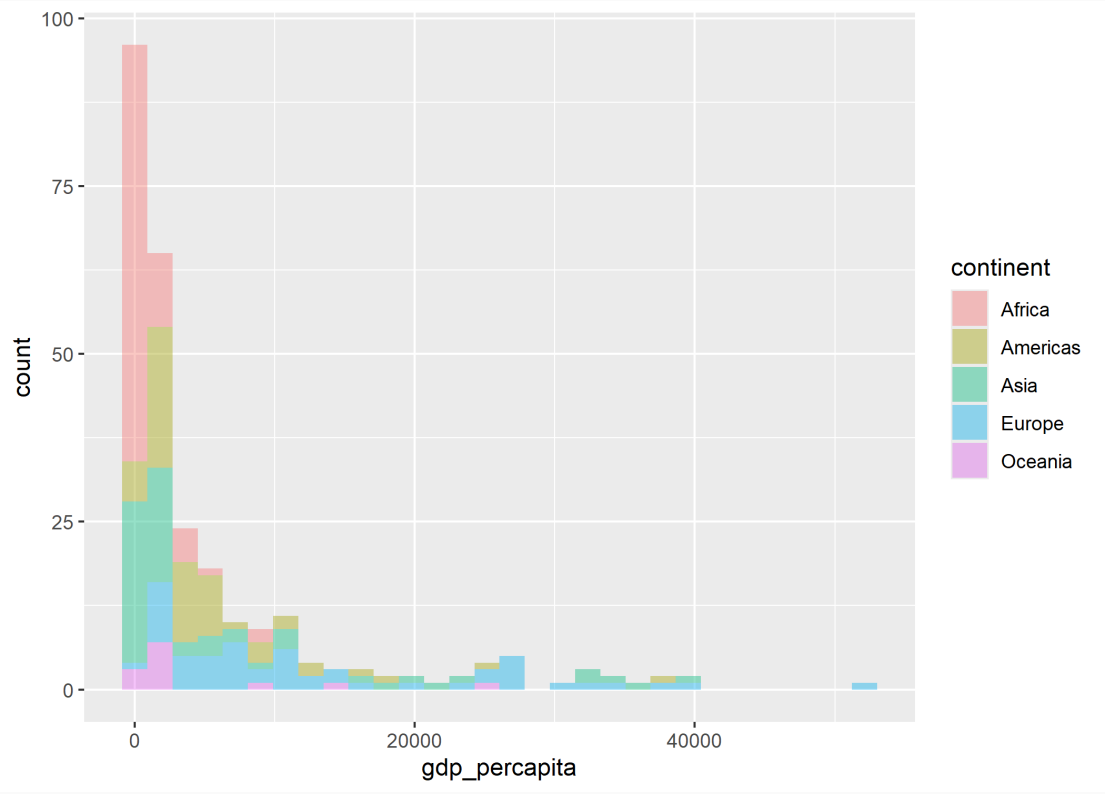
```
pacman::p_load(ggridges)
gap_ds %>% mutate(gdp_percapita = gdp/population) %>%
  filter(year == 1970 & !is.na(gdp_percapita)) %>%
  ggplot(aes(gdp_percapita, continent)) +
  geom_density_ridges() +
  scale_x_log10()
```



Histograms with *geom_histogram()*

In this case histograms are hard to view with so many continents

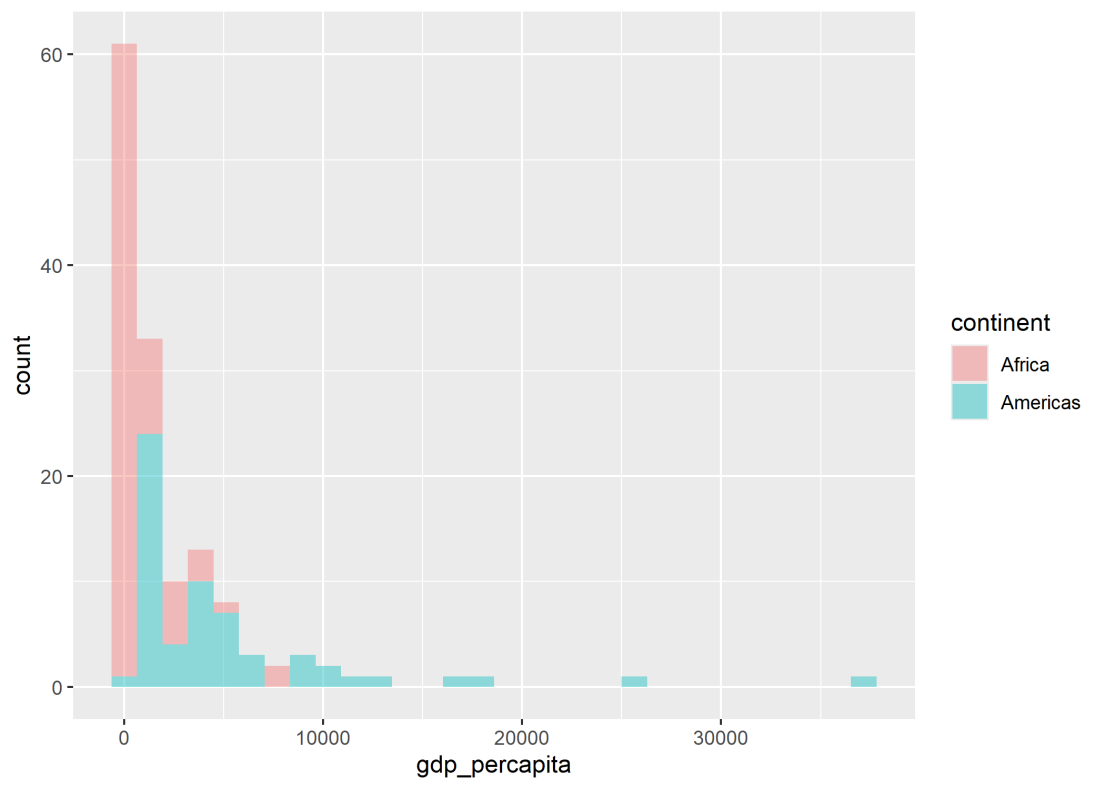
```
gap_ds %>% mutate(gdp_percapita = gdp/population) %>%
  filter(year %in% c(1960, 2010) & !is.na(gdp_percapita)) %>%
  ggplot(aes(gdp_percapita, fill = continent)) +
  geom_histogram(alpha=0.4)
```



Histograms with *geom_histogram()*

Note that `geom_histogram()` defaults to **stacking** for grouped aesthetics:

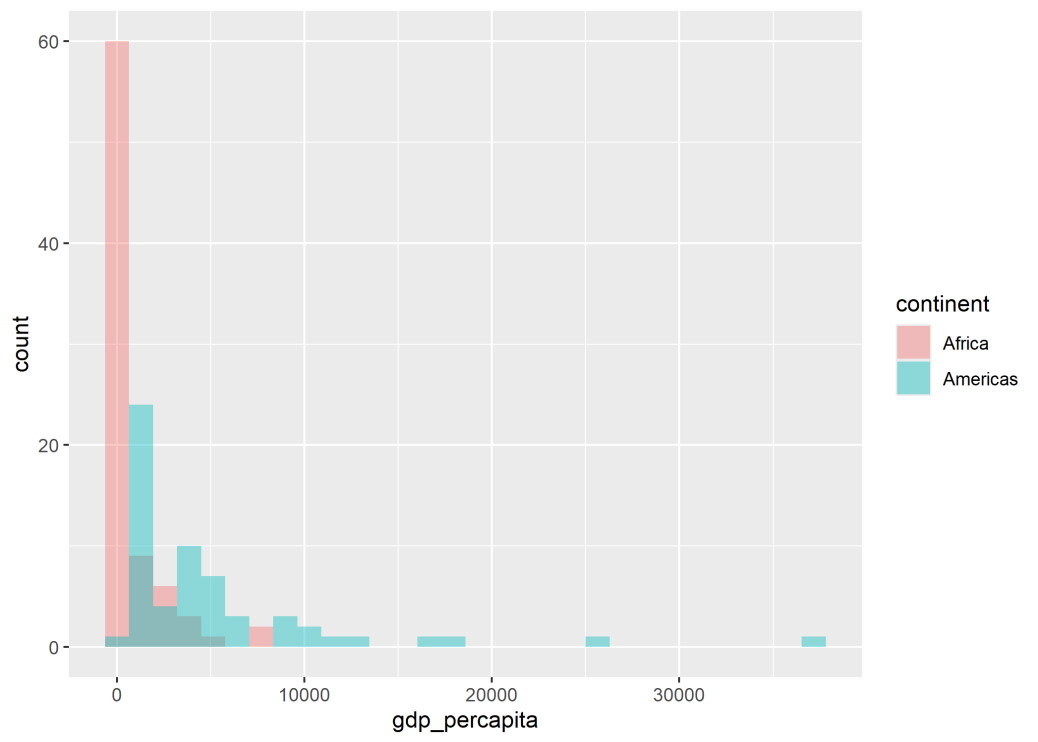
```
gap_ds %>% mutate(gdp_percapita = gdp/population) %>% filter(year %in% c(1960, 2010) & !is.n
  ggplot(aes(gdp_percapita, fill = continent)) +
  geom_histogram(alpha=0.4, position = "stack")
```



Histograms with *geom_histogram()*

To fix this to show the **correct totals**, set `position = "identity"`:

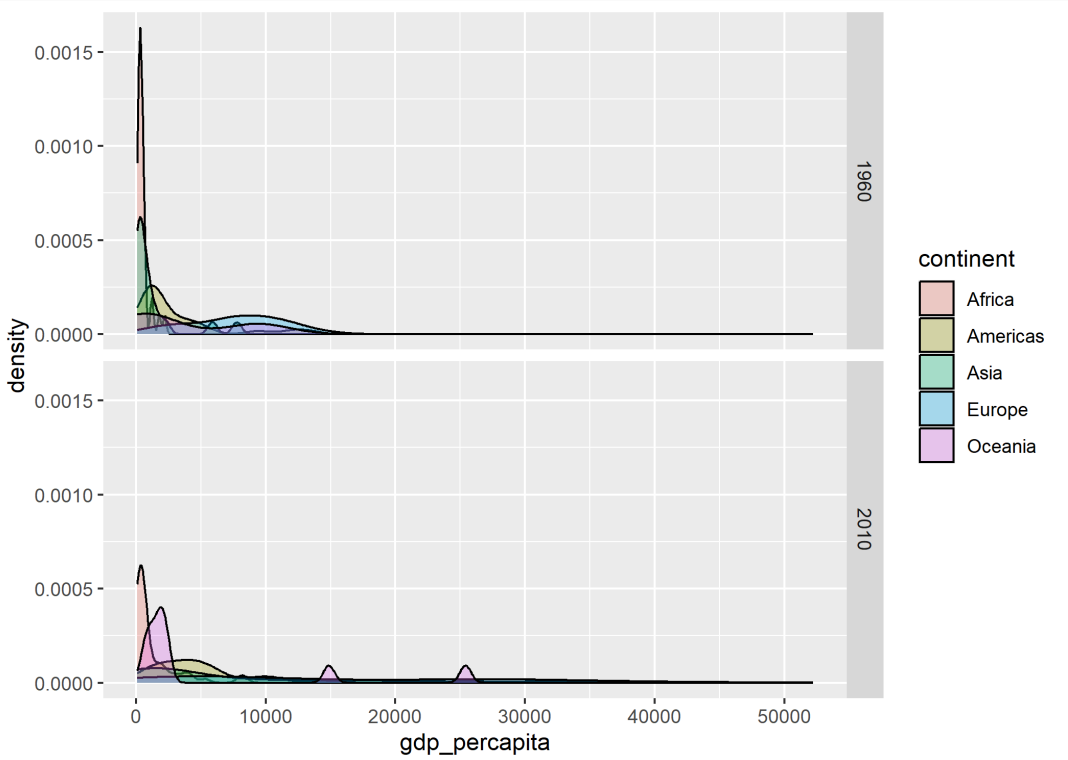
```
gap_ds %>% mutate(gdp_percapita = gdp/population) %>% filter(year %in% c(1960, 2010) & !is.n
  ggplot(aes(gdp_percapita, fill = continent)) +
  geom_histogram(alpha=0.4, position = "identity")
```



Density Plots

Densities with `geom_density()` default to `position = "identity"`:

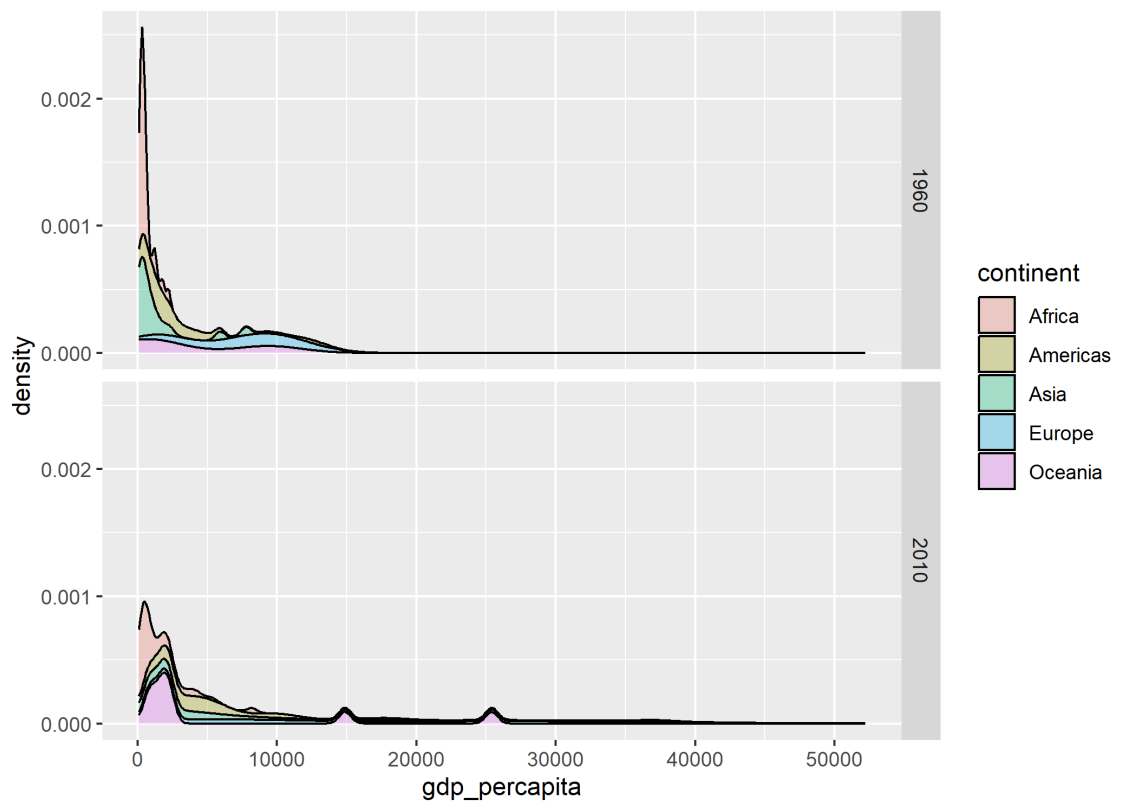
```
dens_df ← gap_ds %>% mutate(gdp_percapita = gdp/population) %>%
  filter(year %in% c(1960, 2010) & !is.na(gdp_percapita))
ggplot(dens_df, aes(gdp_percapita, fill = continent)) +
  geom_density(alpha=0.3, position = "identity") +
  facet_grid(year ~ .)
```



Stacked Density Plots

Stacked densities on [0,1] scale are possible, but hard to view

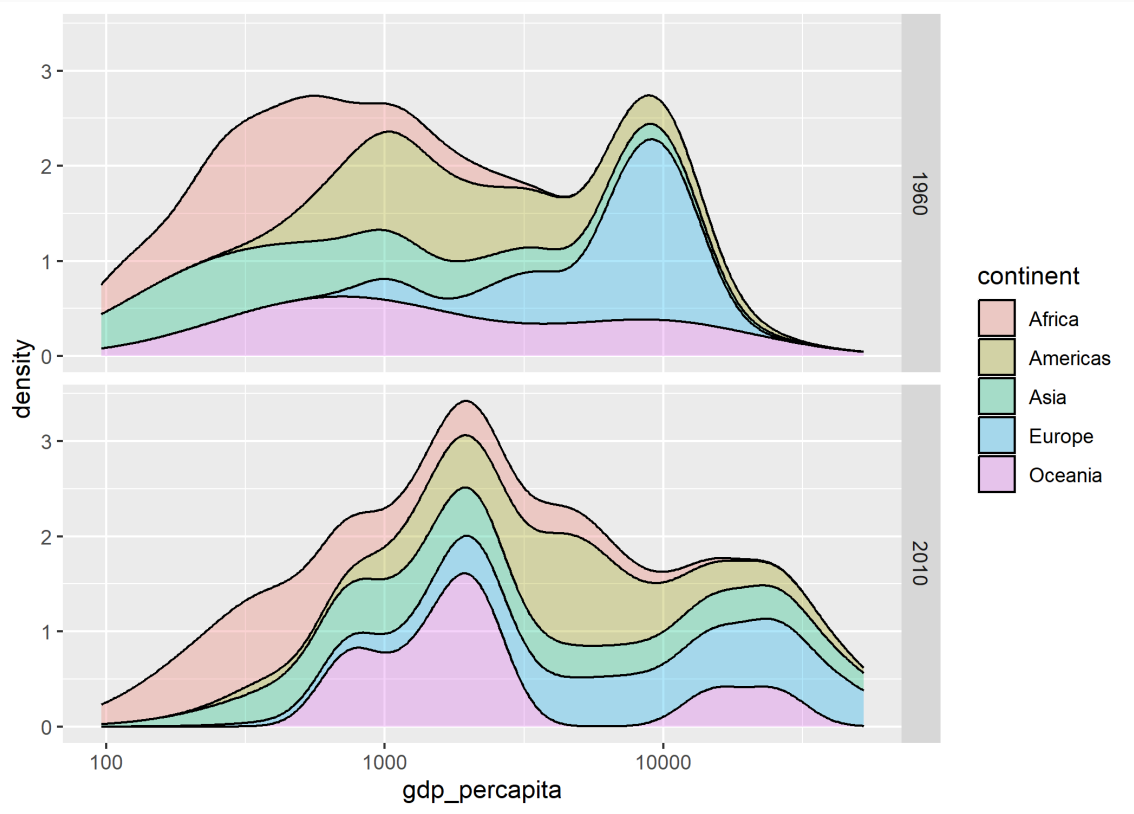
```
ggplot(dens_df, aes(gdp_percapita, fill = continent)) +
  geom_density(alpha=0.3, position = "stack") +
  facet_grid(year ~ .)
```



Stacked Density Plots

Easier to compare with log 10 transformation

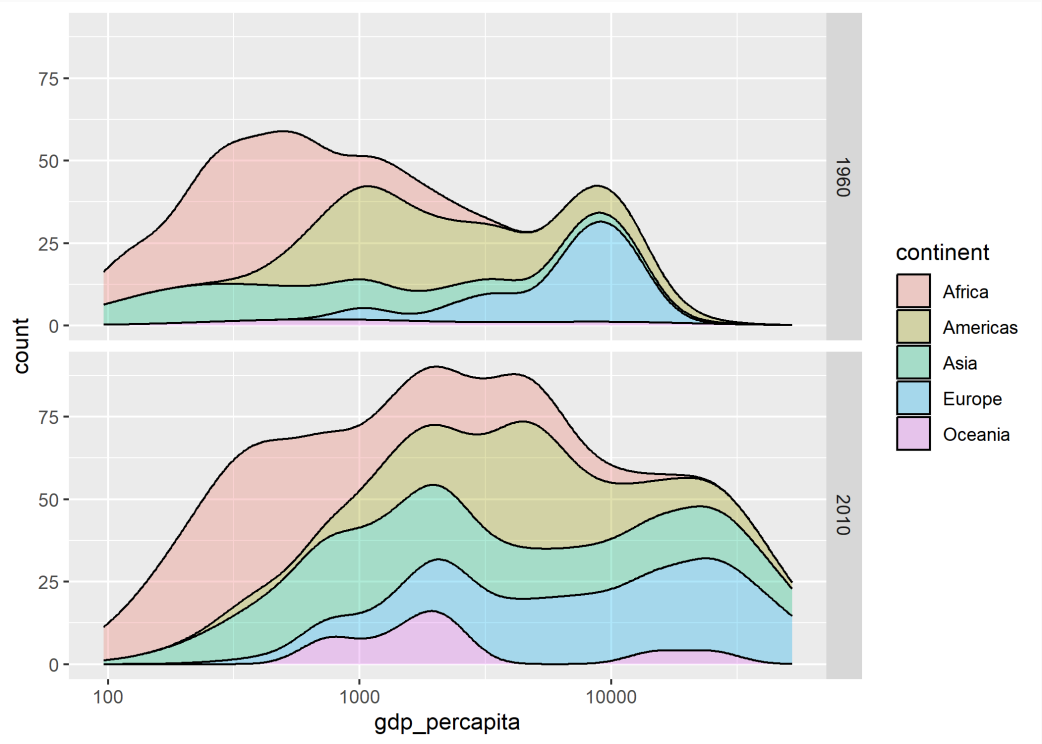
```
ggplot(dens_df, aes(gdp_percapita, fill = continent)) +
  geom_density(alpha=0.3, position = "stack") +
  facet_grid(year ~ .) + scale_x_log10()
```



Stacked Density Plots

Scale each continent by its number of countries using the **computed variable** `..count..`:

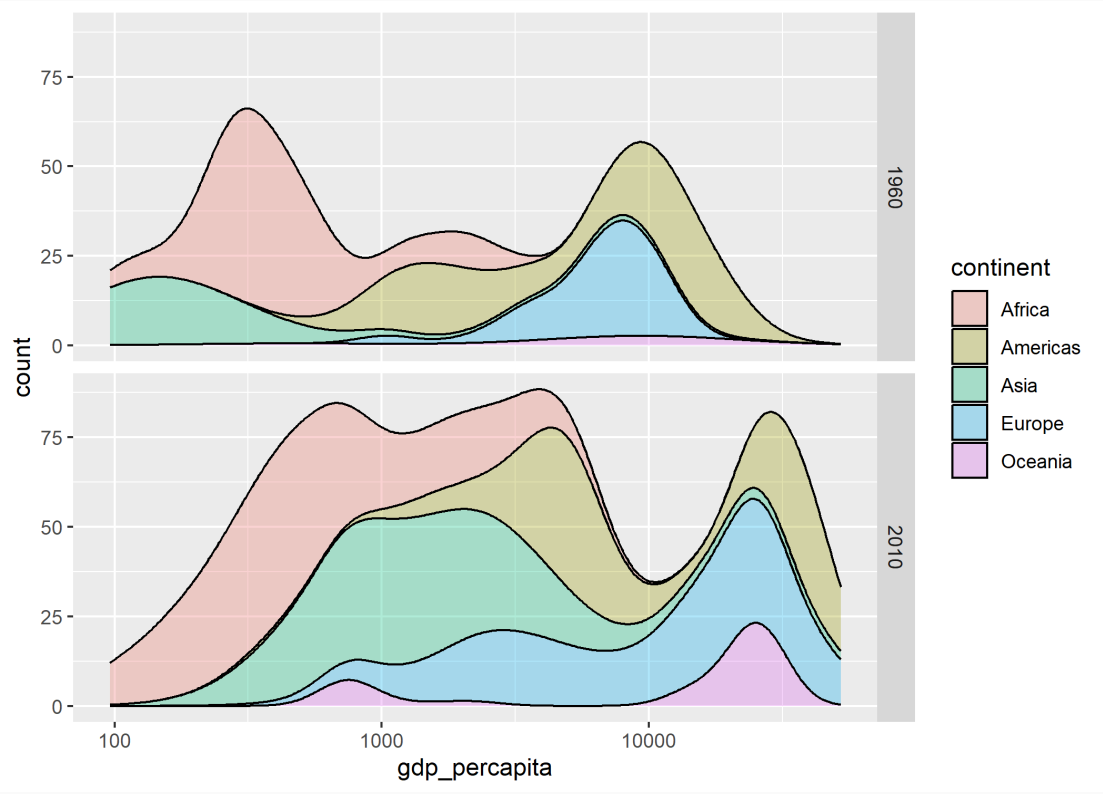
```
ggplot(dens_df, aes(gdp_percapita, y=..count.., fill = continent)) +
  geom_density(alpha=0.3, position = "stack") +
  facet_grid(year ~ .) + scale_x_log10()
```



Stacked Density Plots

And weight countries within each continent by its population:

```
ggplot(dens_df, aes(gdp_percapita, y=..count.., weight=population, fill = continent)) +
  geom_density(alpha=0.3, position = "stack") +
  facet_grid(year ~ .) +
  scale_x_log10()
```



Exporting Graphs

Exporting Graphs with *ggsave()*

Option 1: By default, `ggsave()` saves the last plot printed to the screen

- i.e. what's currently displayed in the "Plots" viewer
  - saves to the working directory

```
# Create a simple scatter plot
filter(gap_ds, year == 2000) %>%
ggplot(aes(x = fertility, y = life_expectancy)) +
geom_point()

# Save our simple scatter plot
ggsave(filename = "simple_scatter.pdf")
```

- This example creates a PDF. Change to `".png"` for PNG, etc.
  - Possible formats: "png", "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only)
- Optional arguments: `path`, `width`, `height`, `dpi`.

Exporting Graphs

For example, to save a print quality (300 dpi) 8in by 6in `".png"` file to the "output" subfolder, we can modify our `ggsave` call to

```
# Save our simple scatter plot
ggsave(filename = "simple_scatter.png",
       path = "output/",
       width = 8,
       height = 6,
       units = "in",
       dpi = "print"
       )
```

Exporting Graphs

Option 2: You can first assign your `ggplot()` objects to memory:

```
# Create a simple scatter plot named 'gg_points'
points_2010 ← filter(gap_ds, year == 2010) %>%
  ggplot(aes(x = fertility, y = life_expectancy)) +
  geom_point()
```

And then save this figure by name using the `plot` argument:

```
# Save our simple scatter plot name 'ggsave'
ggsave(
  # can add subfolder directly in filename
  filename = "output/simple_scatter.pdf",
  plot = points_2010
)
```

Exporting Graphs

In what format should you save your graphics?

Vector graphics are composed of **formulas or paths**.

- "Draw a straight line from (0, 0) to (13, 4)."
- Infinitely zoom-able. **Preserves all underlying information**.
- May be slow to load when complex.
- Fully modifiable in vector art software (i.e. Adobe Illustrator)
- `.pdf` or `.svg`.

Raster graphics are composed of **pixels** (a grid of squares with color information).

- Only an **approximation** to the underlying shapes or points.
- Work better with Microsoft Office and HTML.
- The original format of photographs.
- Usually best: `.png`. Also `.jpeg`, `.gif`.

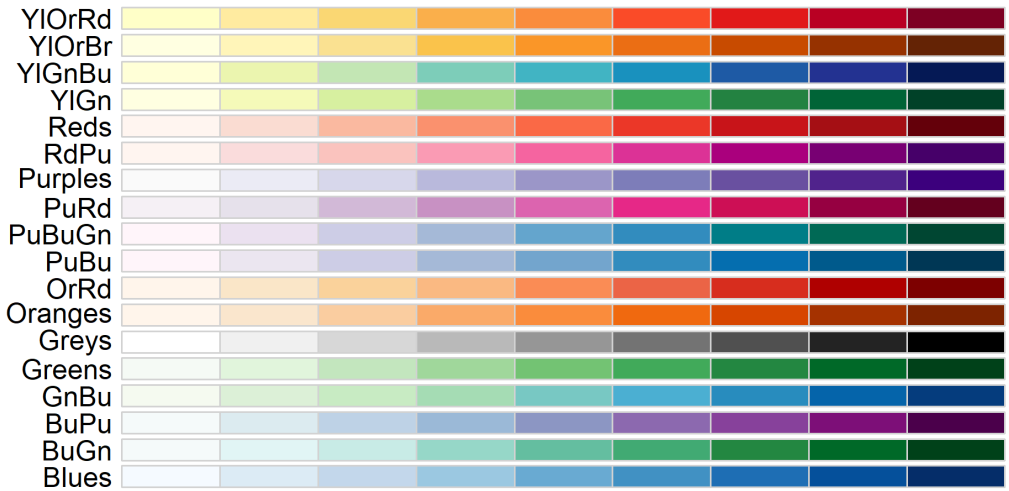
Colors Schemes

Color Schemes

Choose a **sequential** color scheme when your values are ordered in **only one direction**.

- Low to high; values are all positive; zero is defined arbitrarily.

```
pacman::p_load(RColorBrewer)
display.brewer.all(type="seq")
```

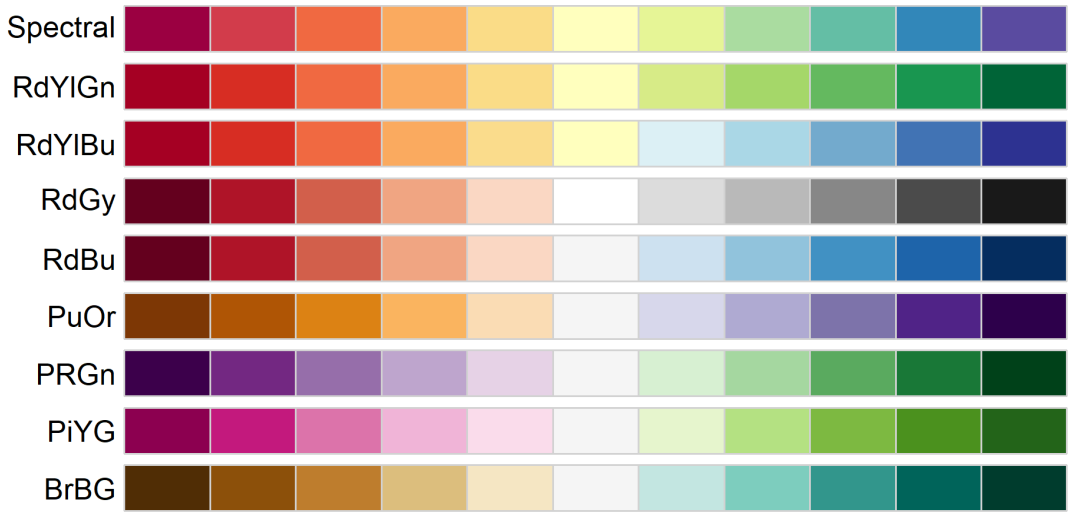


Color Schemes

Choose a **diverging** color scheme when your values are ordered in **two directions relative to a center**.

- Positive vs. negative; vote shares relative to 50%.

```
display.brewer.all(type="div")
```

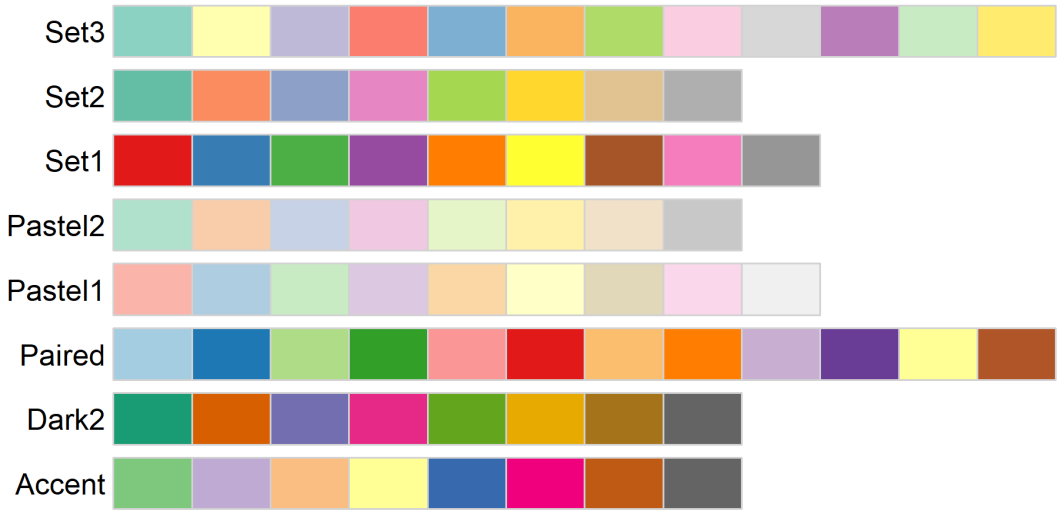


Color Schemes

Choose a **qualitative** color scheme when your values have **no ordering**.

- Only need to distinguish among categories.

```
display.brewer.all(type="qual")
```



Use Established Color Schemes

There are several great color schemes available in R created by professional visual designers.

- **RColorBrewer** is based on the research of cartographer Cynthia Brewer. Her **ColorBrewer** website lets you choose a color scheme by value ordering and whether you need it to be colorblind safe, printer friendly, or photocopy safe.

- **viridis** schemes are designed to span a large perceptual range while remaining perceptually uniform, robust to colorblindness, and pretty. (The next few slides show diagrams from the the package's **vignette**.)

Use Established Color Schemes

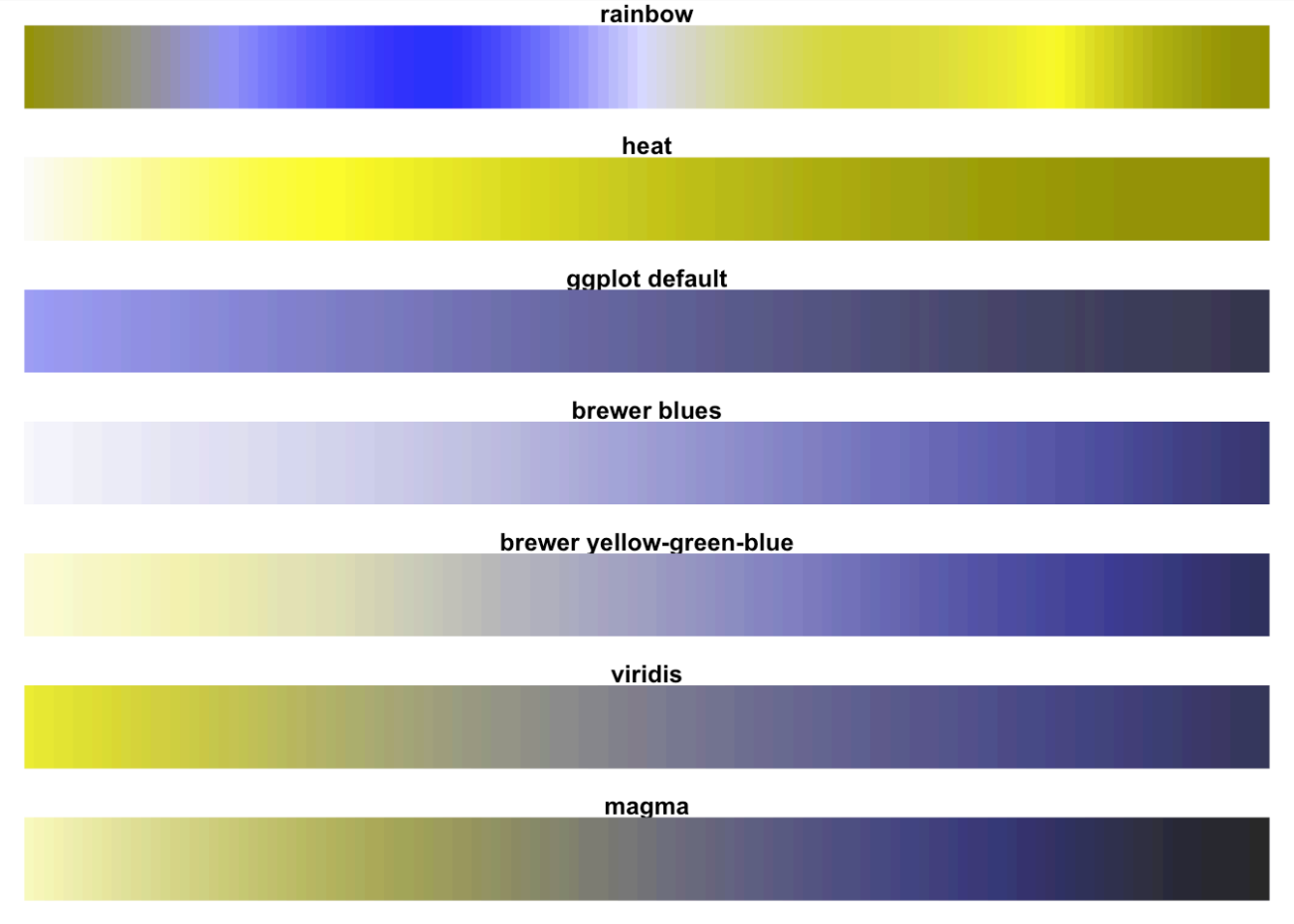
Palettes available in **viridis**:

Comparing Palettes

Compare `rainbow` and `heat` from base R, the default **ggplot2** palette, and palettes from **RColorBrewer** and **viridis**:

Consider Colorblindness

Green-Blind (Deuteranopia):

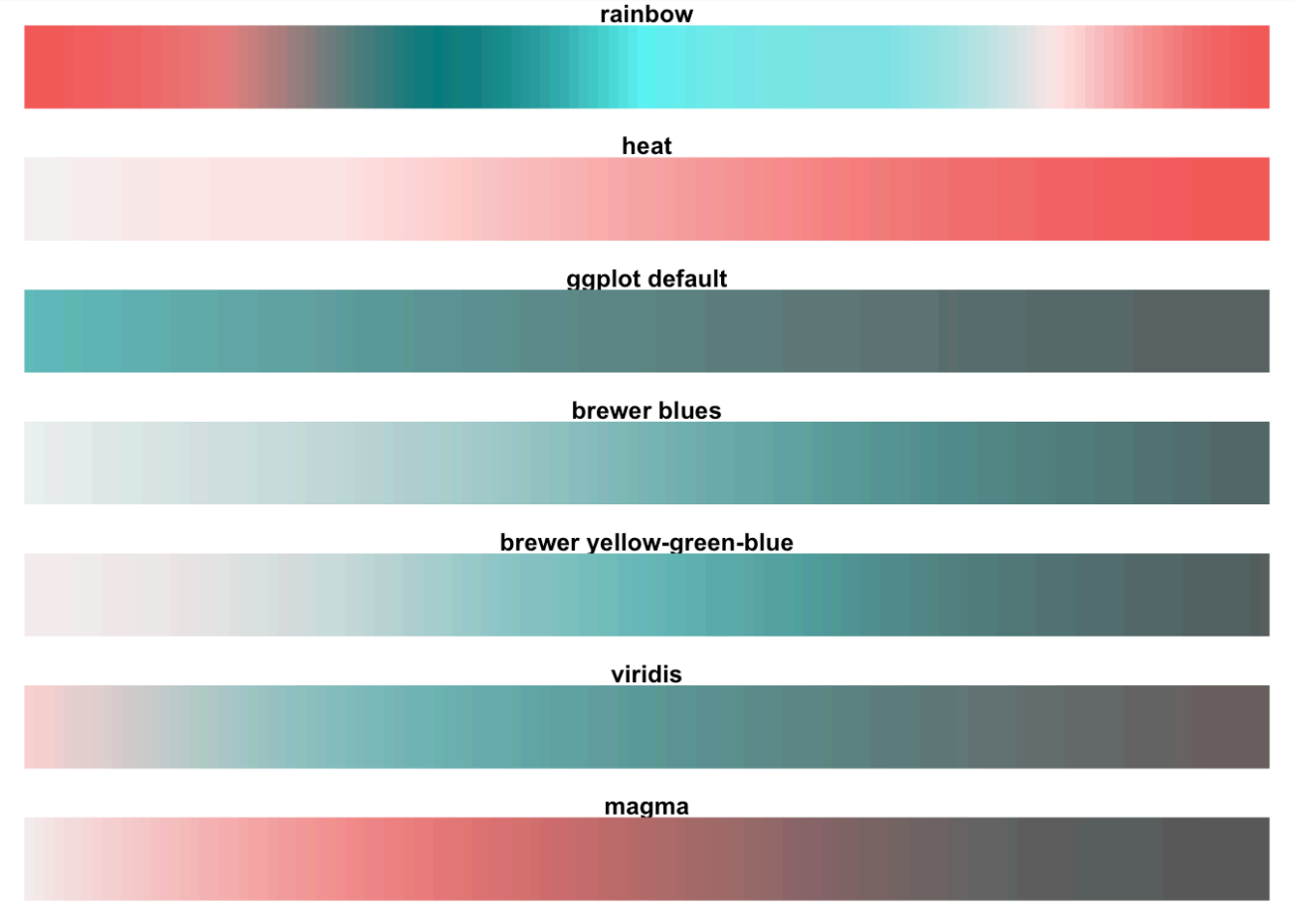


Consider Colorblindness

Red-Blind (Protanopia):

Consider Colorblindness

Blue-Blind (Tritanopia):



Consider Printer-Friendliness

Grayscale:

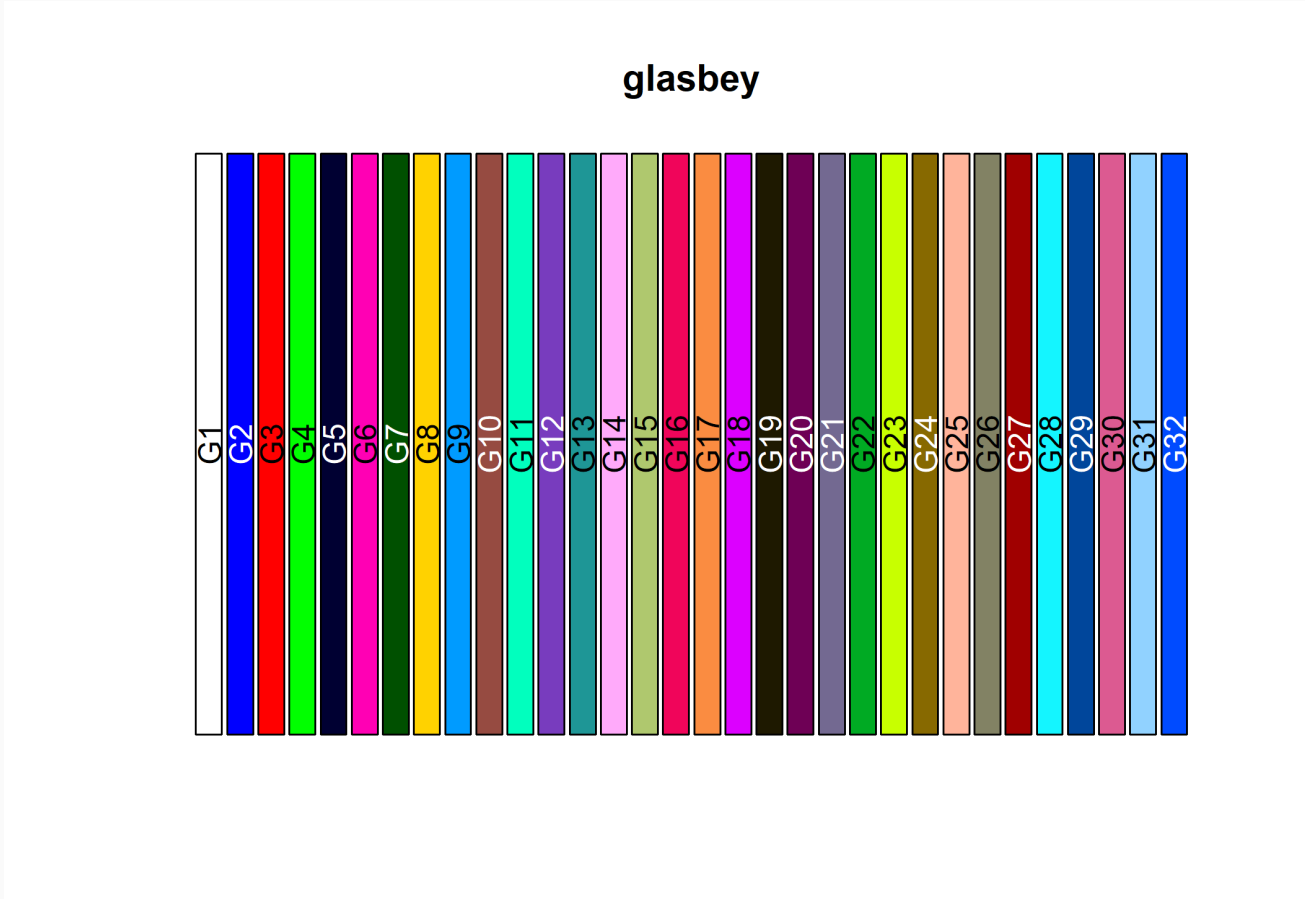
Other Established Color Schemes

Other packages for useful/fun color schemes:

- **Polychrome** has large **qualitative palettes** and functions for checking how palettes will look to a person with color deficit vision
- **broman** has Crayola crayon colors.

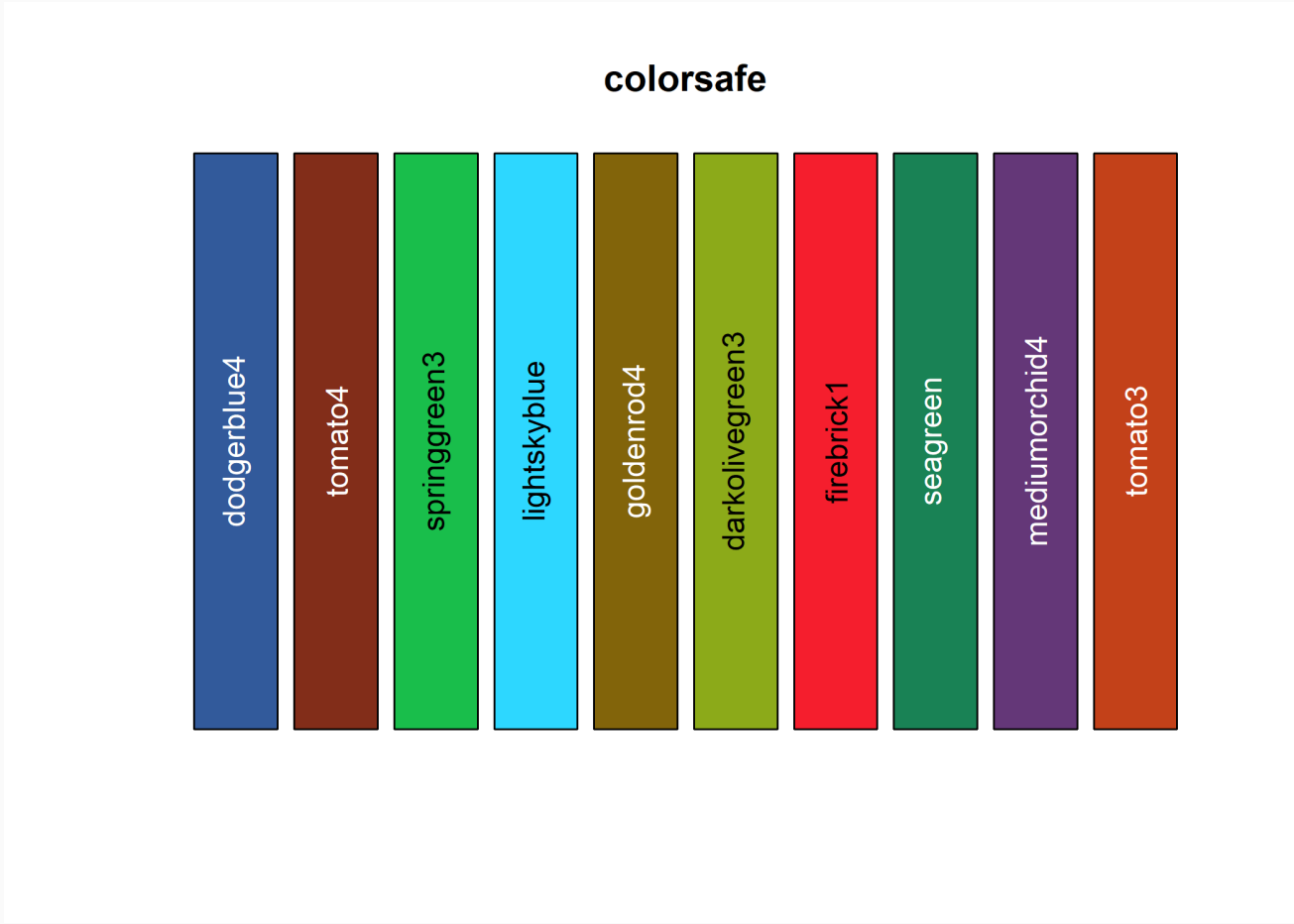
Polychrome: glasbey Palette

```
pacman::p_load(Polychrome)
data(glasbey)
swatch(glasbey)
```



Polychrome: colorsafe Palette

```
pacman::p_load(Polychrome)
data(colorsafe)
swatch(colorsafe)
```



Broman: brocolors

```
pacman::p_load(broman)
plot_crayons()
```

Themes

Themes

ggplot2's default theme is now fairly iconic - but that doesn't mean we can't do better.

Option 1: Pre-Existing Themes

- Default **ggplot2** choice is `theme_gray`
- Alternate **ggplot2** themes: `theme_bw`, `theme_linedraw`, `theme_light`, `theme_dark`, `theme_minimal`, `theme_classic`, `theme_void`
- Additional themes in **ggthemes**
  - i.e. want to replicate known appearances from **Excel, Stata, The Economist, FiveThirtyEight, or WSJ**

Themes

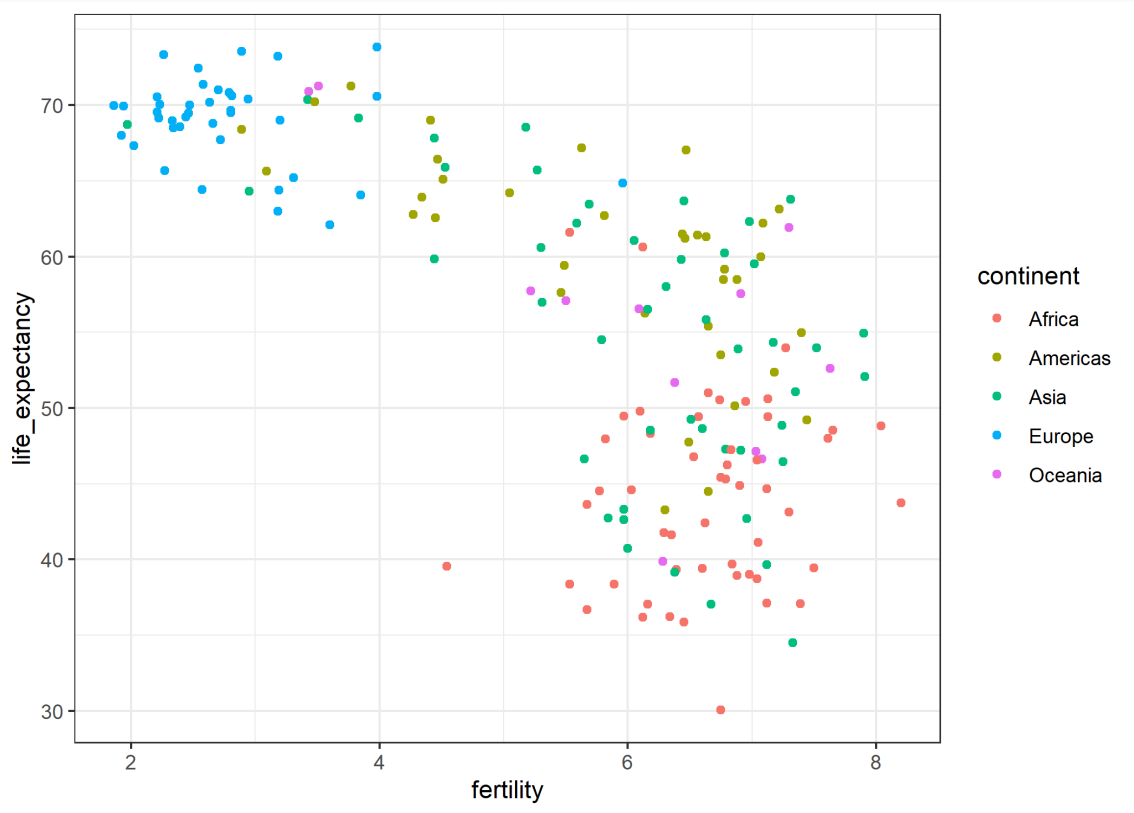
ggplot2's default theme is now fairly iconic - but that doesn't mean we can't do better.

Option 2: Create Your Own!

- Every element of the plot is customizable (gridlines, fonts, legends, margins, etc.)
- Use `theme` either on-the-fly for small tweaks or store a complete custom theme in memory for regular use

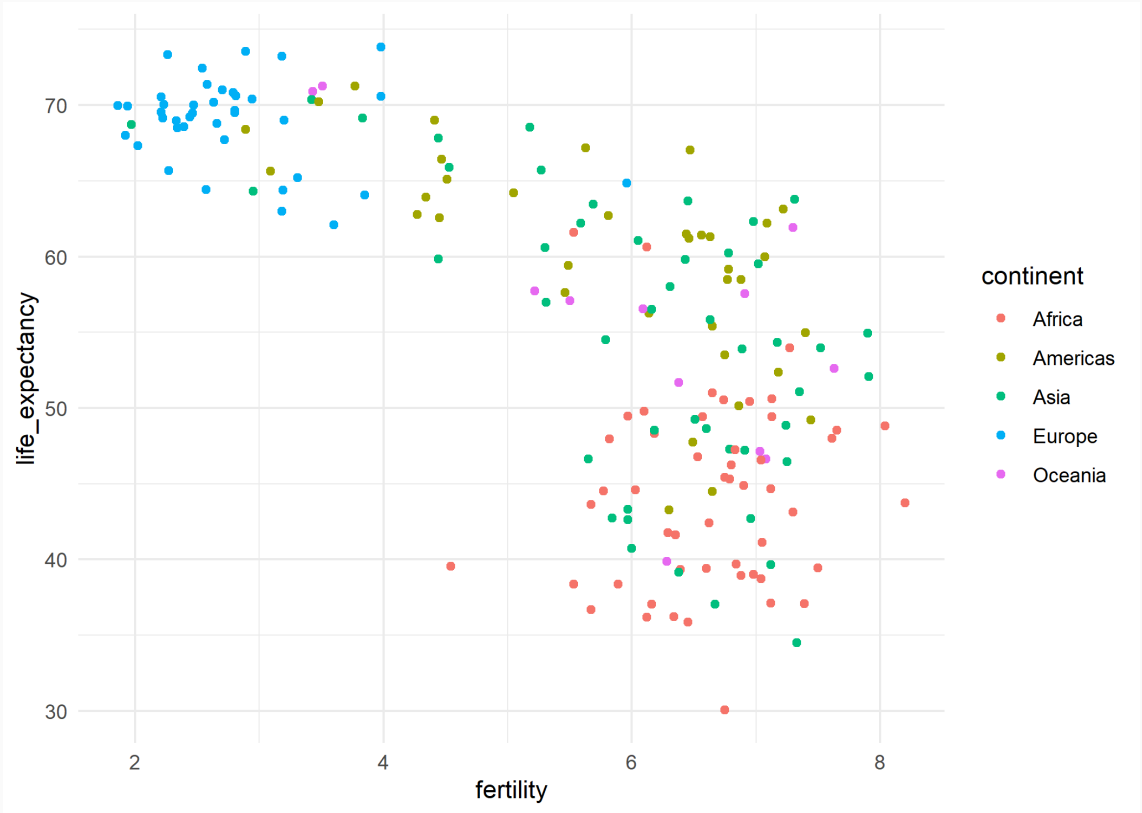
Other ggplot2 Themes: *theme_bw*

```
filter(gap_ds, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  theme_bw()
```



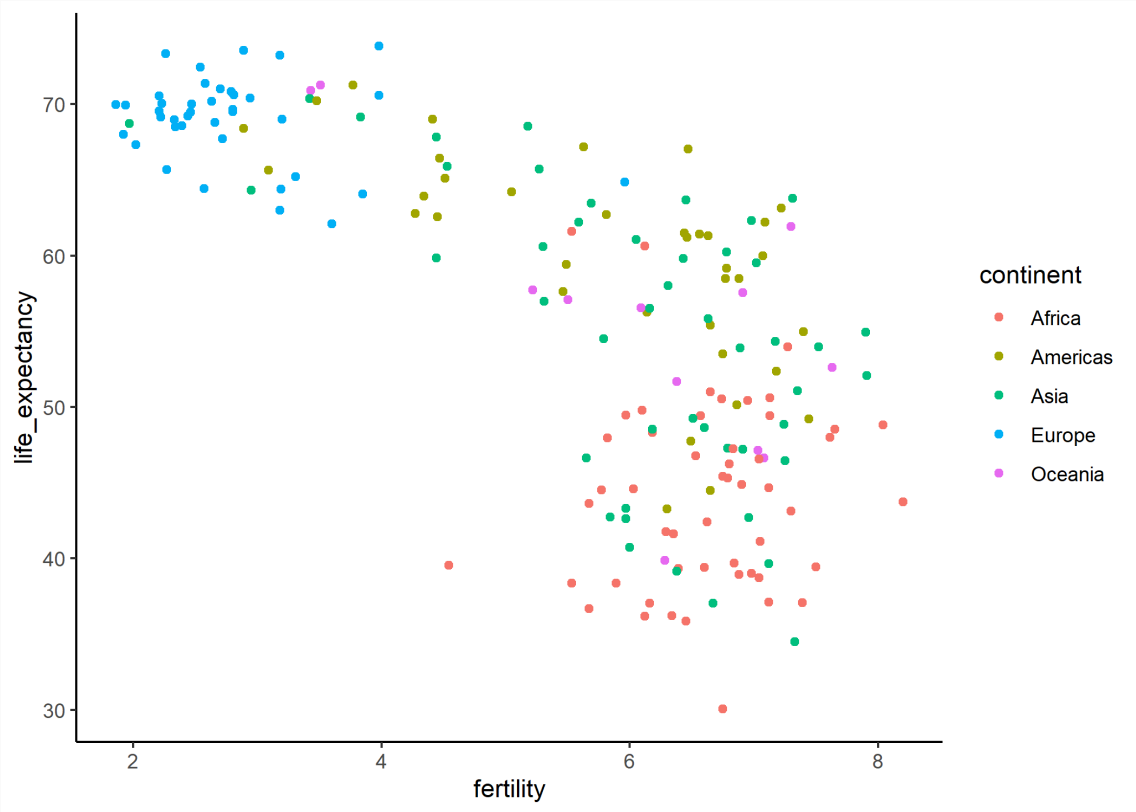
Other ggplot2 Themes: *theme_minimal*

```
filter(gap_ds, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  theme_minimal()
```



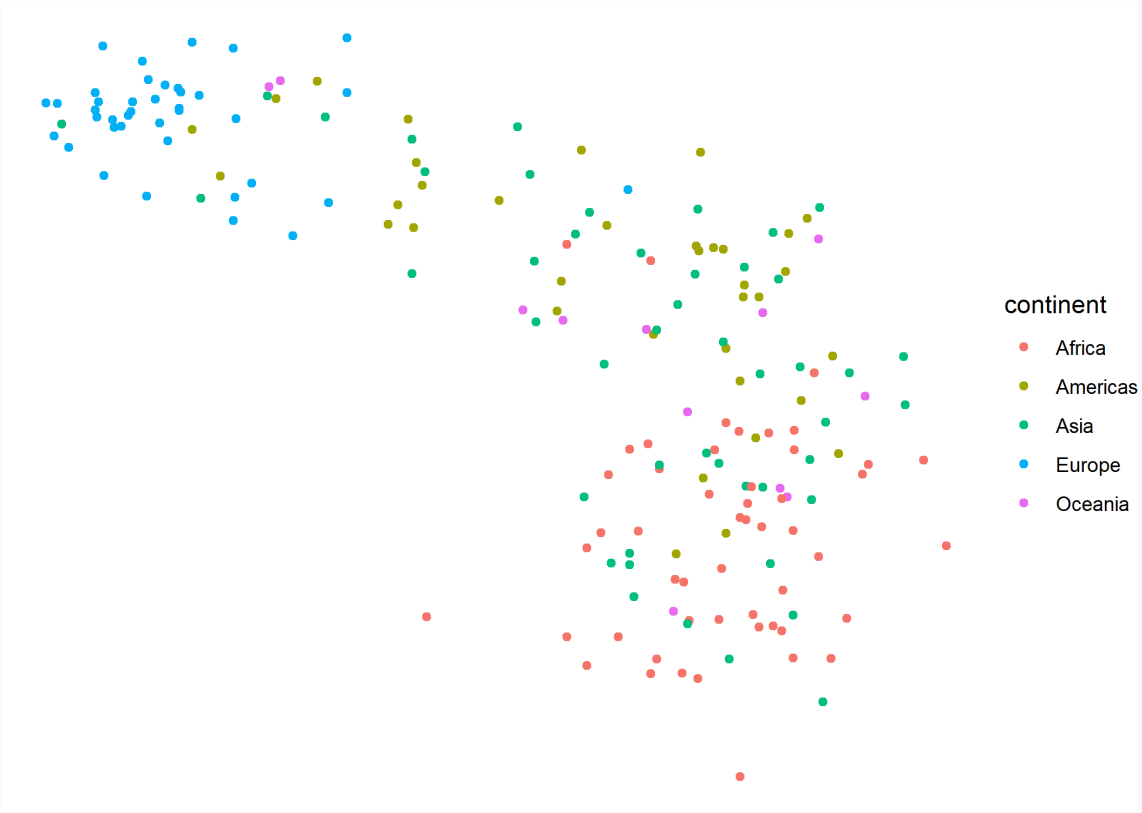
Other ggplot2 Themes: *theme_classic*

```
filter(gap_ds, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  theme_classic()
```



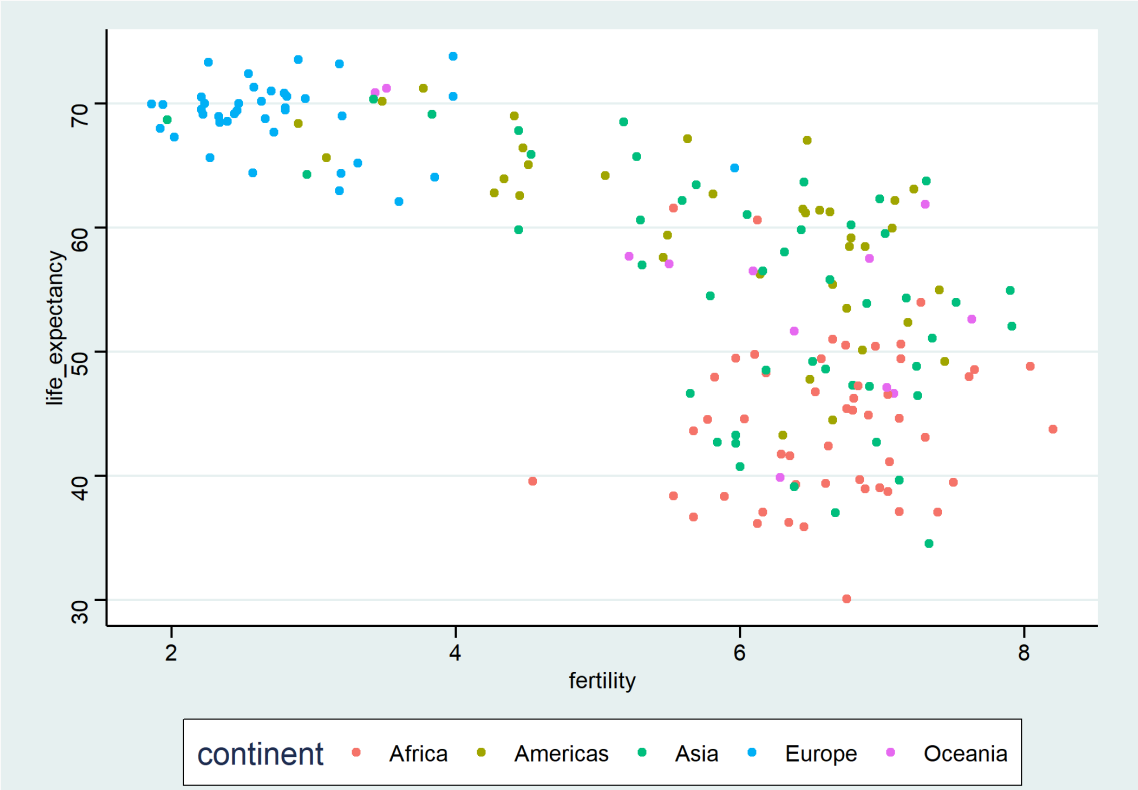
Other ggplot2 Themes: *theme_void*

```
filter(gap_ds, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  theme_void()
```



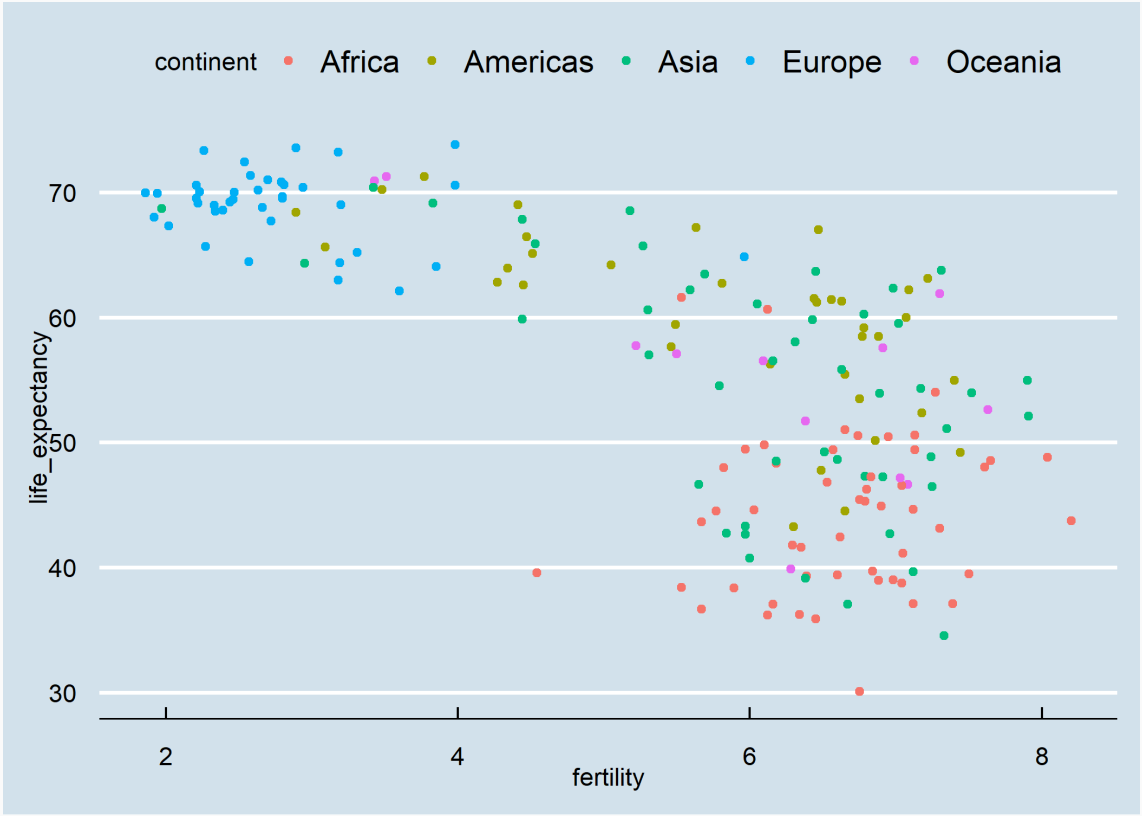
ggthemes Themes: *theme_stata*

```
pacman::p_load(ggthemes)
filter(gap_ds, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  theme_stata()
```



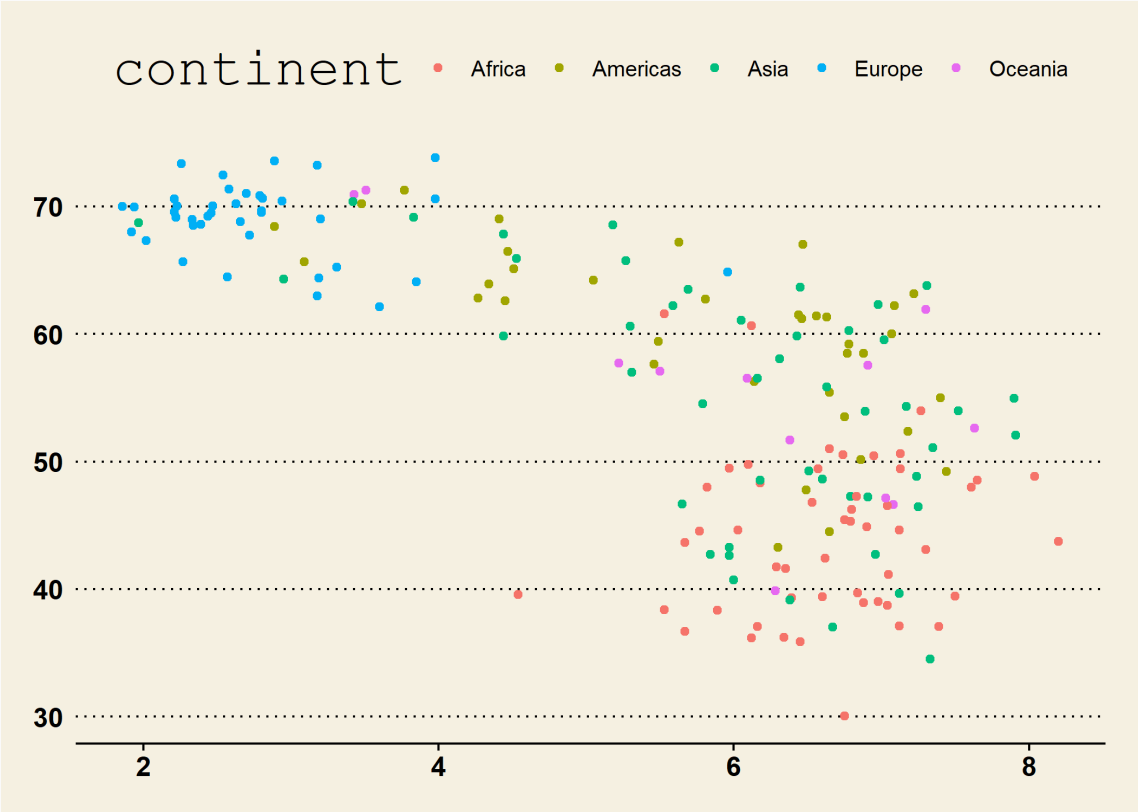
ggthemes Themes: *theme_economist*

```
filter(gap_ds, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  theme_economist()
```



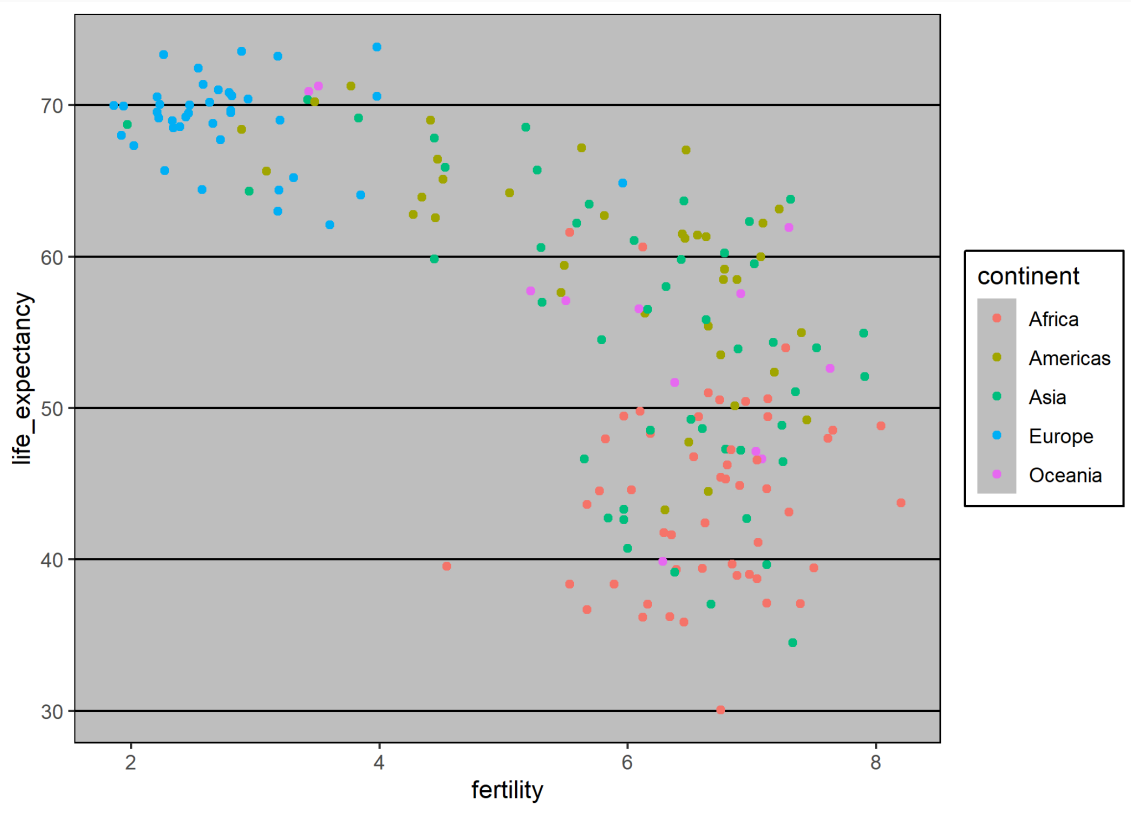
ggthemes Themes: *theme_wsj*

```
filter(gap_ds, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  theme_wsj()
```



ggthemes Themes: *theme_excel*

```
filter(gap_ds, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  theme_excel()
```



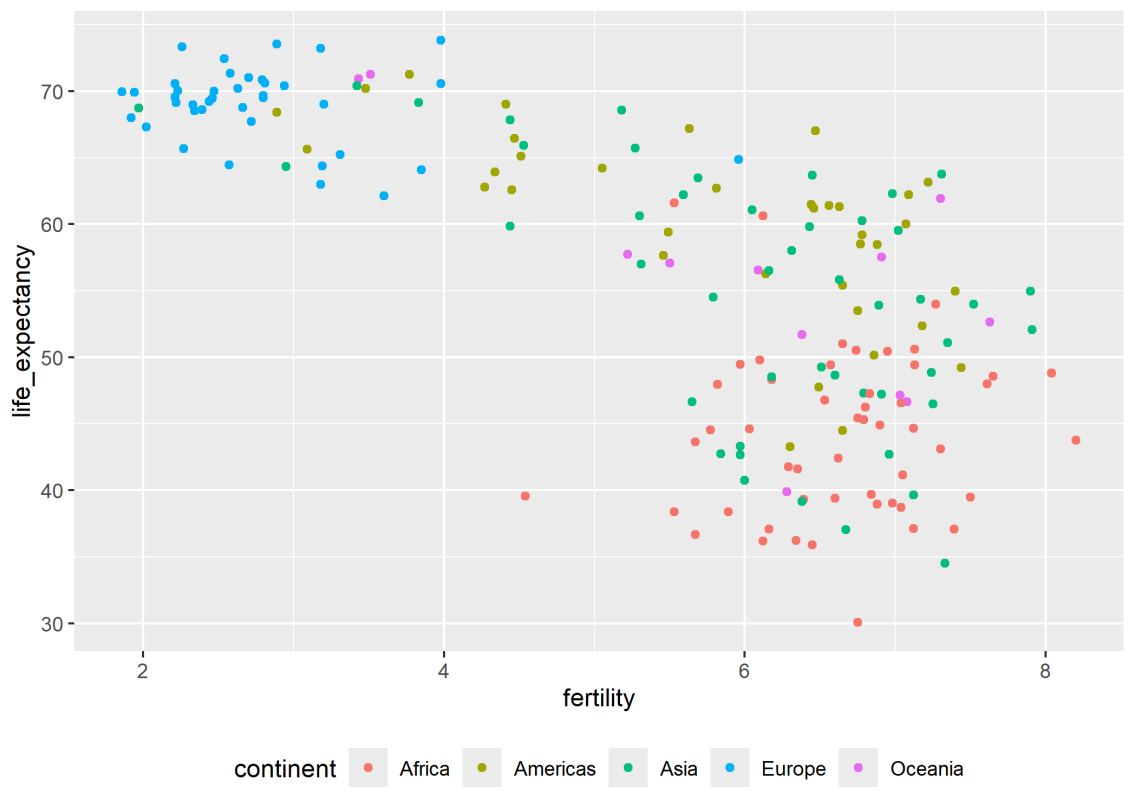
Creating Custom Themes

Alternatively, you can create a **custom theme** by tweaking **any and all plot elements**

- Specify elements and specifications within `theme`
- Either inline or as separate theme object in memory
- Full list of elements **here**

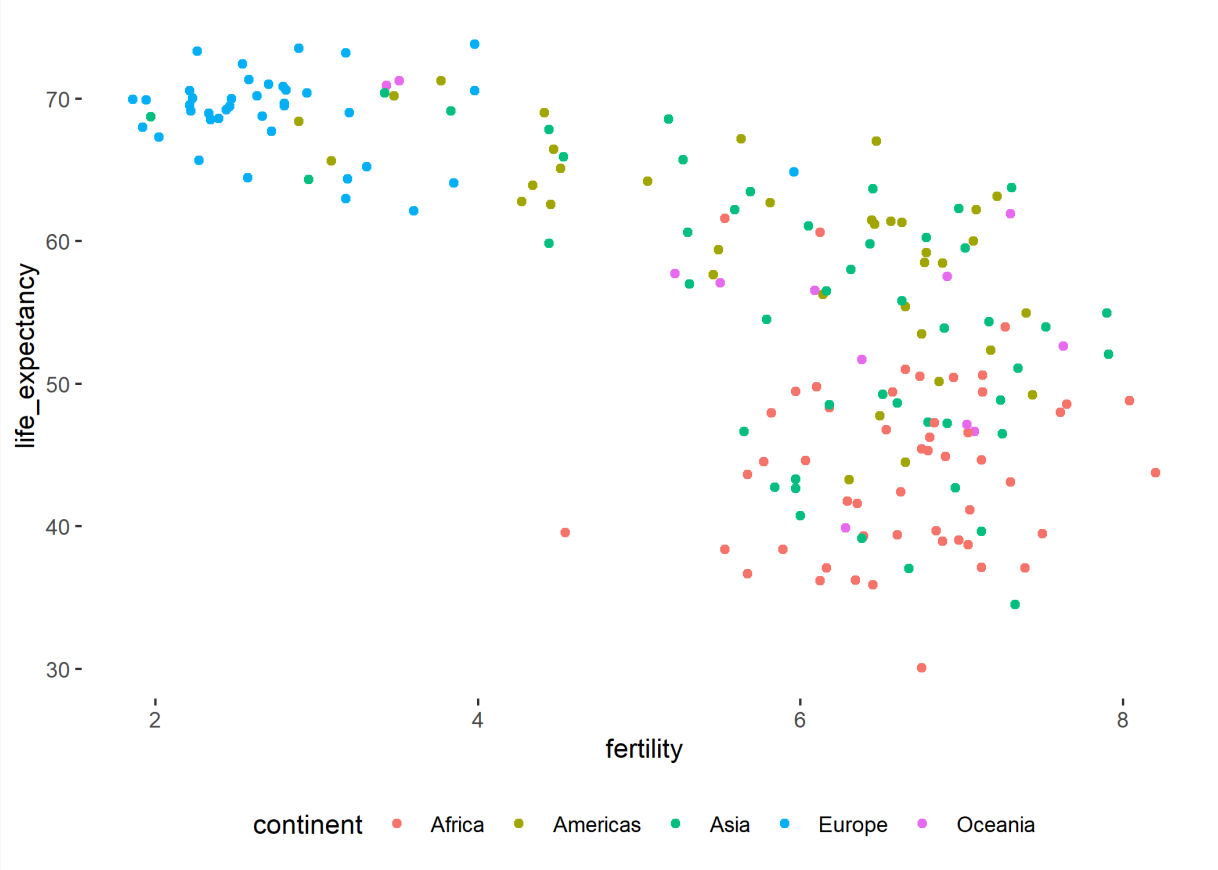
Custom Themes: Legend Position

```
theme_plot ← filter(gap_ds, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point()

theme_plot +
  theme(legend.position = "bottom")
```



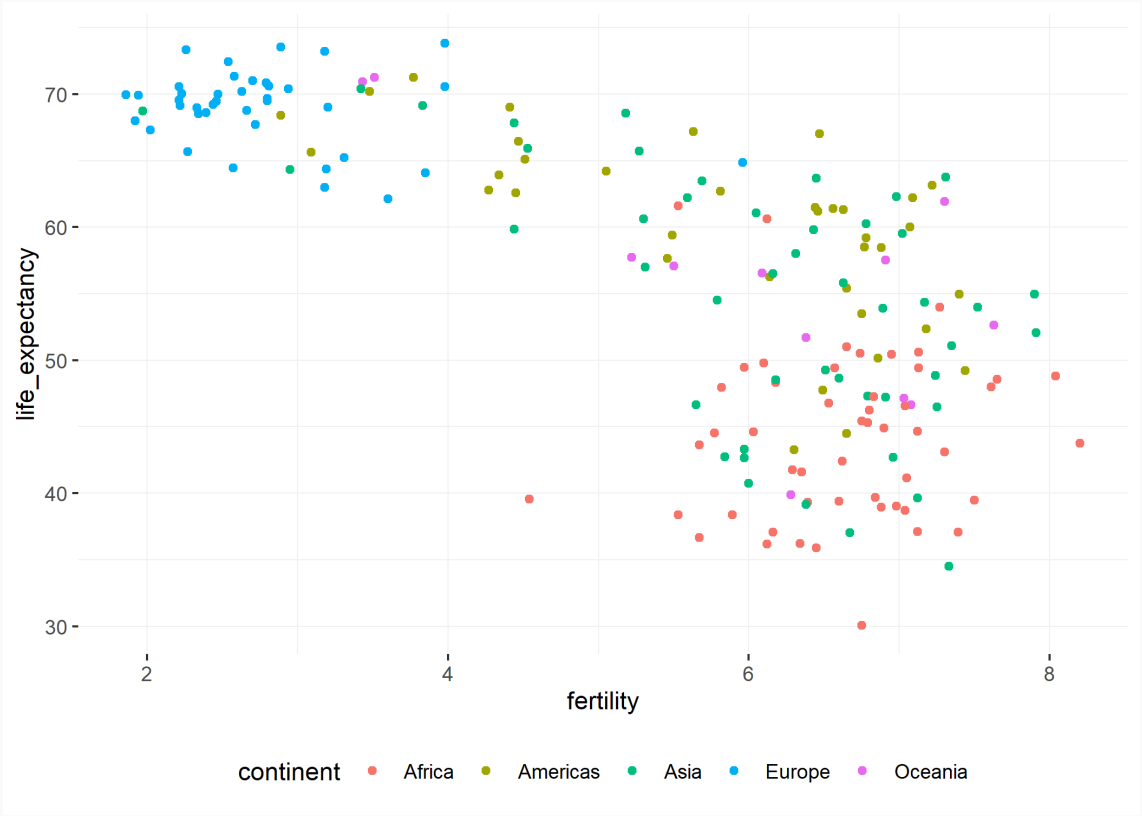
Custom Themes: Background Fill Color

```
theme_plot +
  theme(legend.position = "bottom",
        panel.background = element_rect(fill = NA)) # change to white
```



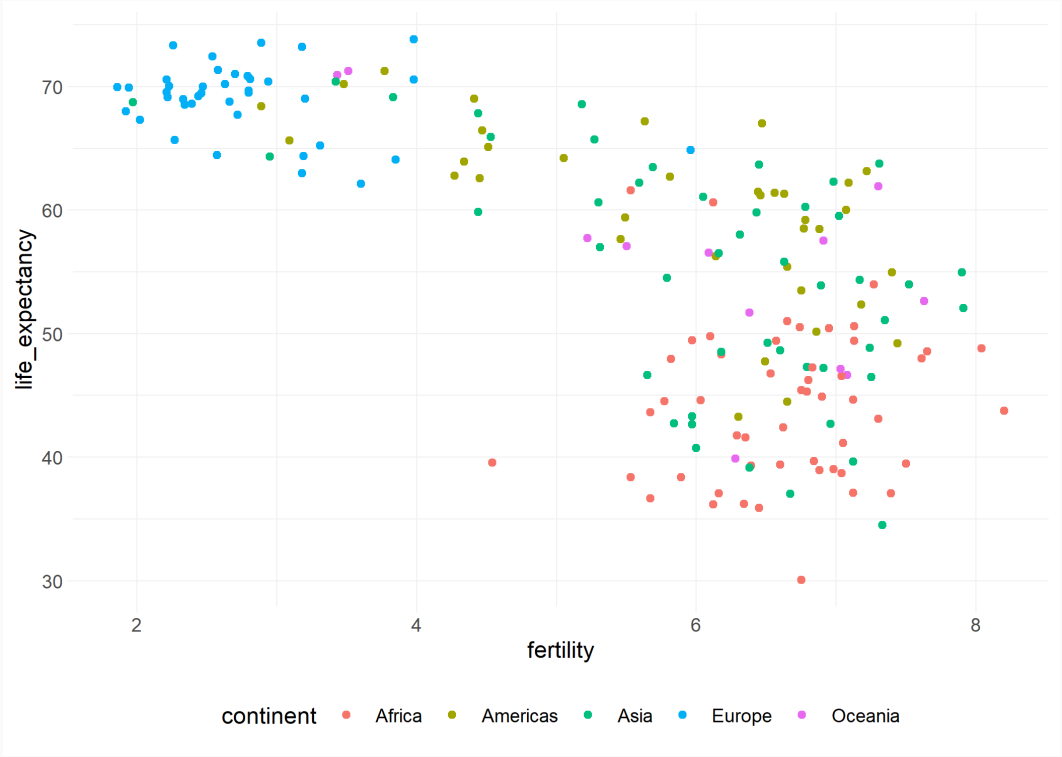
Custom Themes: Gridlines

```
theme_plot +
  theme(legend.position = "bottom",
        panel.background = element_rect(fill = NA), # change to white
        panel.grid.major = element_line(color = "grey95", linewidth = 0.3), # change color of majo
        panel.grid.minor = element_line(color = "grey95", linewidth = 0.3)) #change color of minor
```



Custom Themes: Tickmarks

```
theme_plot +
  theme(legend.position = "bottom",
        panel.background = element_rect(fill = NA), # change to white
        panel.grid.major = element_line(color = "grey95", linewidth = 0.3), # change color of majo
        panel.grid.minor = element_line(color = "grey95", linewidth = 0.3), #change color of minor
        axis.ticks = element_line(color = "grey95", linewidth = 0.3)) # make axis tick marks the s
```



Custom Themes: Fonts

Use **showtext** to add a custom font

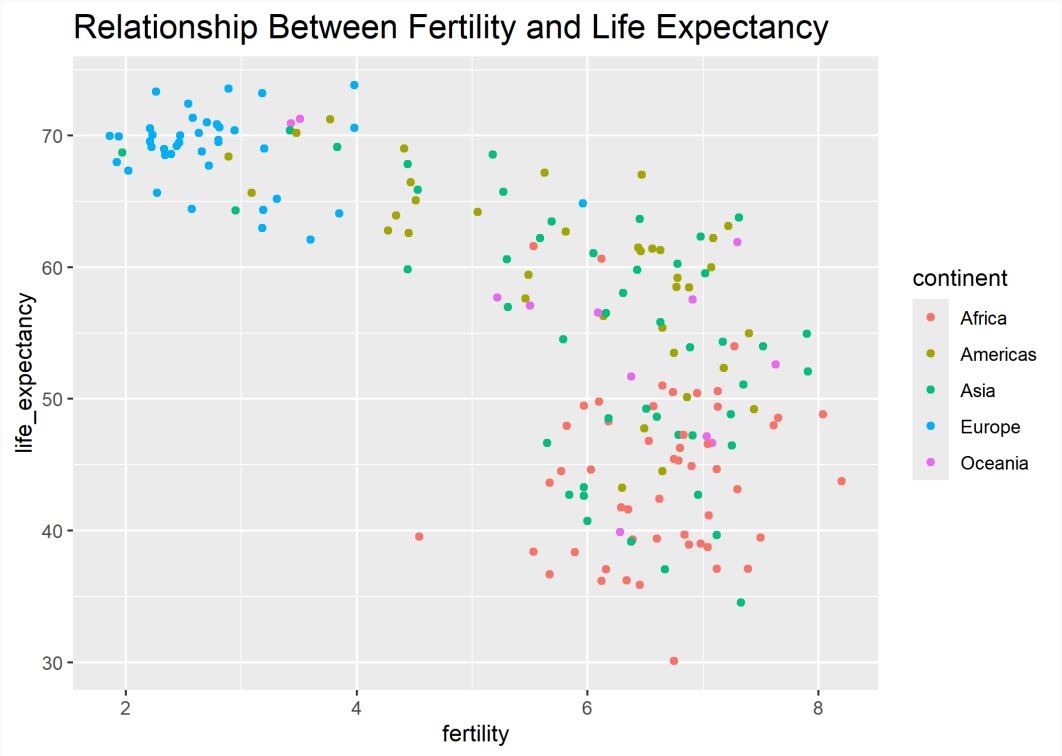
- i.e. add any **Google Font**
- I like Lato
- `showtext_auto(enable = TRUE)` to automatically use **showtext** in all plots
- `showtext_begin()/end()` to turn on and off when desired

```
# Name of Font Family, what we'll refer to it as in R
font_add_google("Schoolbell", "bell")
font_add_google("Lato", "lato")
```

Custom Themes: Fonts

With default font:

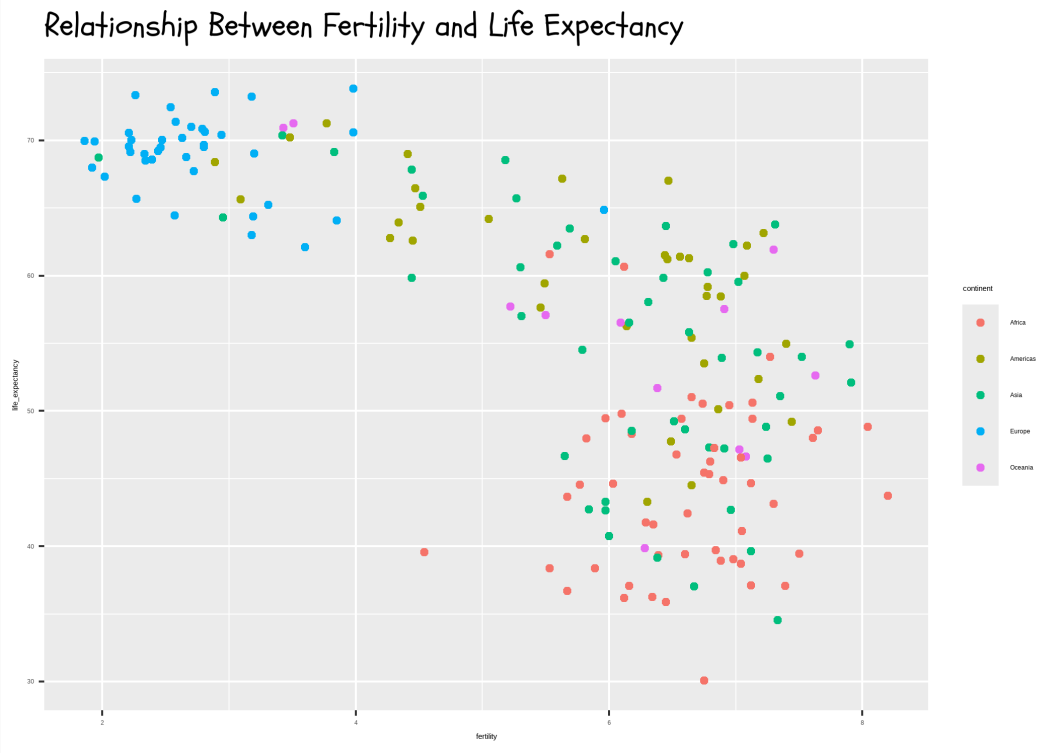
```
theme_plot +
  labs(title = "Relationship Between Fertility and Life Expectancy") +
  theme(plot.title=element_text(size=16))
```



Custom Themes: Fonts

With custom (dumb) font:

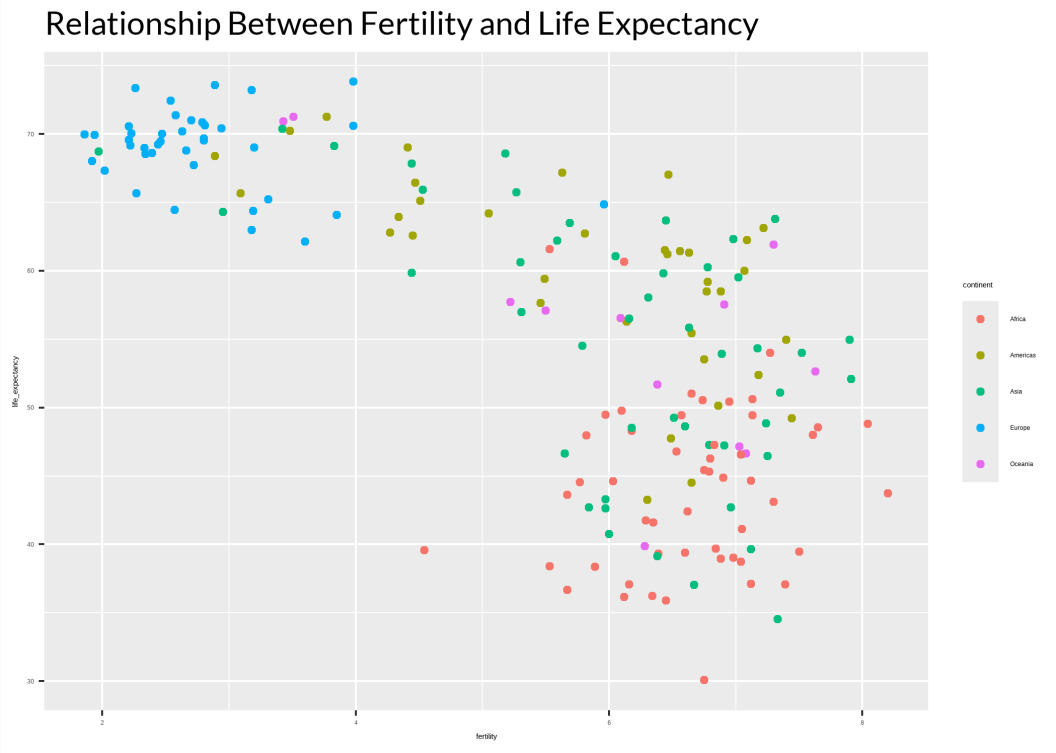
```
showtext_auto()

theme_plot +
  labs(title = "Relationship Between Fertility and Life Expectancy") +
  theme(plot.title=element_text(size=48, family = "bell"))
```



Custom Themes: Fonts

With custom (better) font:

```
showtext_auto()

theme_plot +
  labs(title = "Relationship Between Fertility and Life Expectancy") +
  theme(plot.title=element_text(size=48, family = "lato"))
```

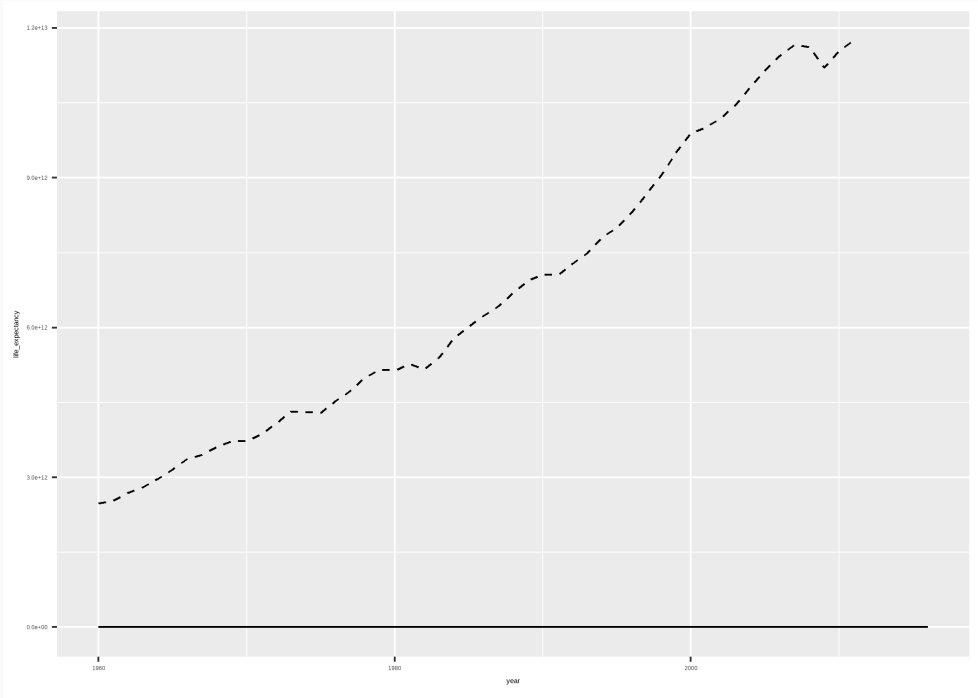


Extending ggplot2

Secondary Y Axes

Add a **secondary y axis** with `scale_y_continuous/discrete()` and `sec.axis = sec_axis()`

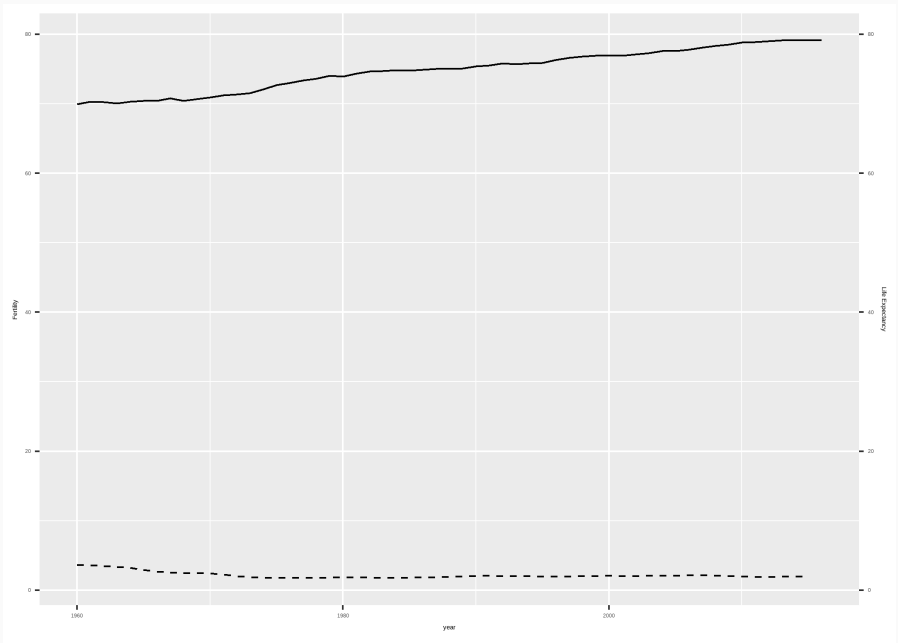
```
us_gap ← filter(gap_ds, country == "United States")

ggplot(us_gap, aes(x = year))+
  geom_line(aes(y = life_expectancy)) +
  geom_line(aes(y = gdp), linetype = "dashed")
```



Secondary Y Axes

Add a **secondary y axis** with `scale_y_continuous/discrete()` and `sec.axis = sec_axis()`

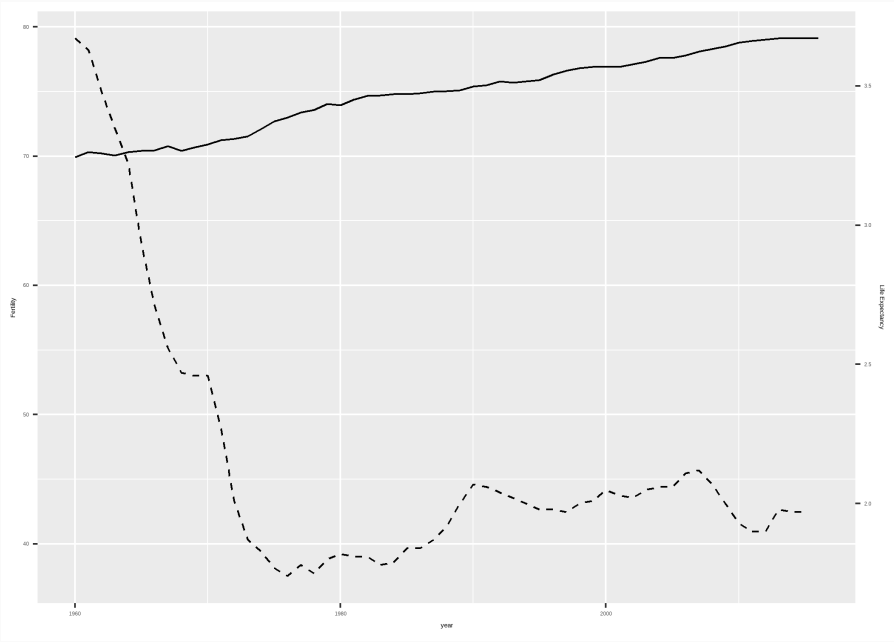
```
ggplot(us_gap, aes(x = year))+
  geom_line(aes(y = life_expectancy)) +
  geom_line(aes(y = fertility), linetype = "dashed") +
  scale_y_continuous(
    name = "Fertility", # name for first axis
    sec.axis = sec_axis(~., name = "Life Expectancy"))
```



Secondary Y Axes

Add a **scale factor adjustment** to rescale the second geom/right axis

```
scalefactor ← max(us_gap$life_expectancy, na.rm = T)/max(us_gap$fertility, na.rm = T)

ggplot(us_gap, aes(x = year)) +
  geom_line(aes(y = life_expectancy)) +
  geom_line(aes(y = fertility * scalefactor), linetype = "dashed") +
  scale_y_continuous(
    name = "Fertility", # name for first axis
    sec.axis = sec_axis(~./scalefactor, name = "Life Expectancy"))
```



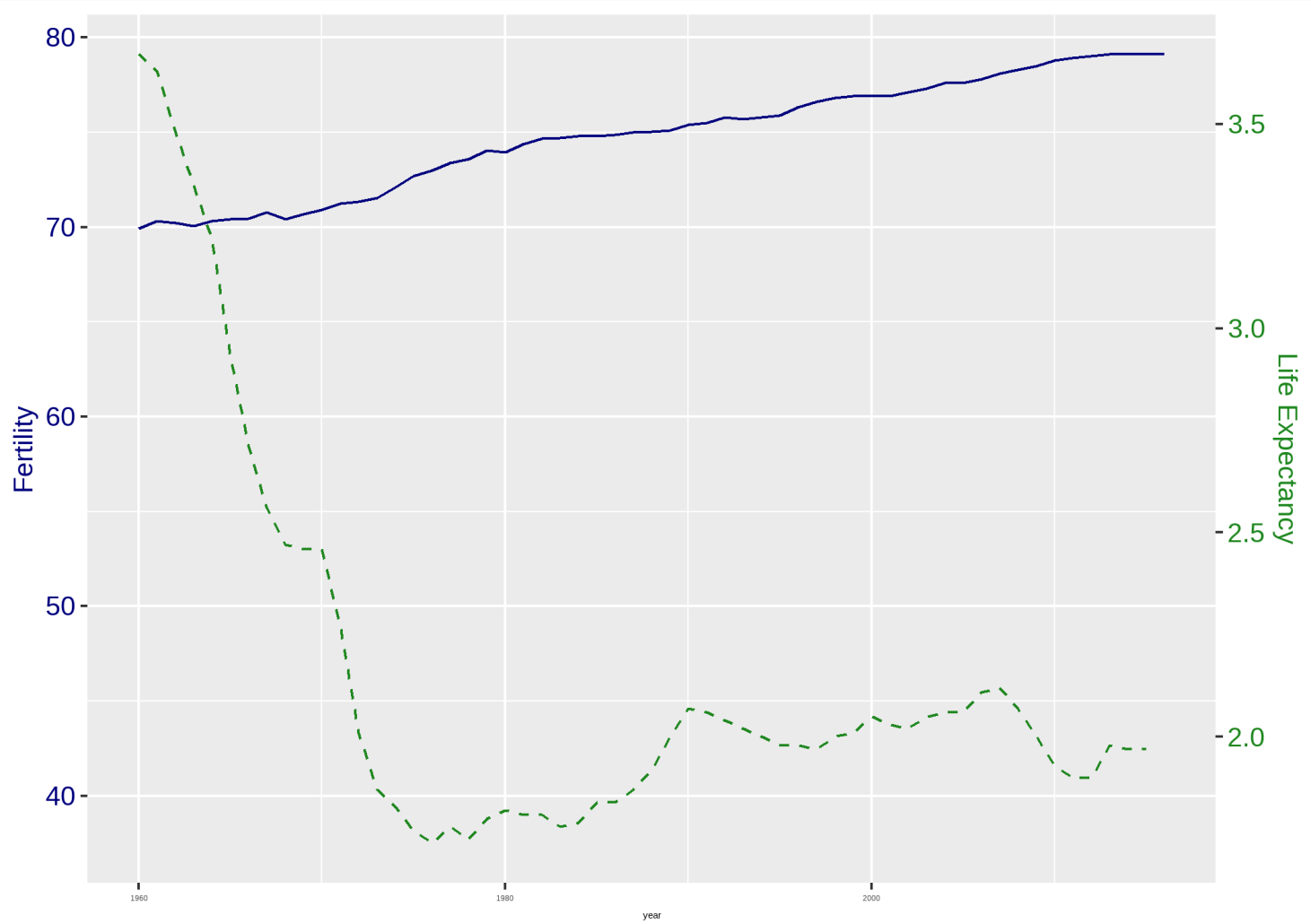
Secondary Y Axes

Add **color** to elements to emphasize which axis is for which series

```
ggplot(us_gap, aes(x = year))+
  geom_line(aes(y = life_expectancy), col = "navyblue") +
  geom_line(aes(y = fertility * scalefactor), linetype = "dashed", col = '
  scale_y_continuous(name = "Fertility", sec.axis = sec_axis(~./scalefact
  theme(axis.title.y.left=element_text(color="navyblue", size = 32),
        axis.text.y.left=element_text(color="navyblue", size = 32),
        axis.title.y.right=element_text(color="forestgreen", size =32),
        axis.text.y.right=element_text(color="forestgreen", size = 32))
```

Secondary Y Axes

Add **color** to elements to emphasize which axis is for which series



Animations with *gganimate*

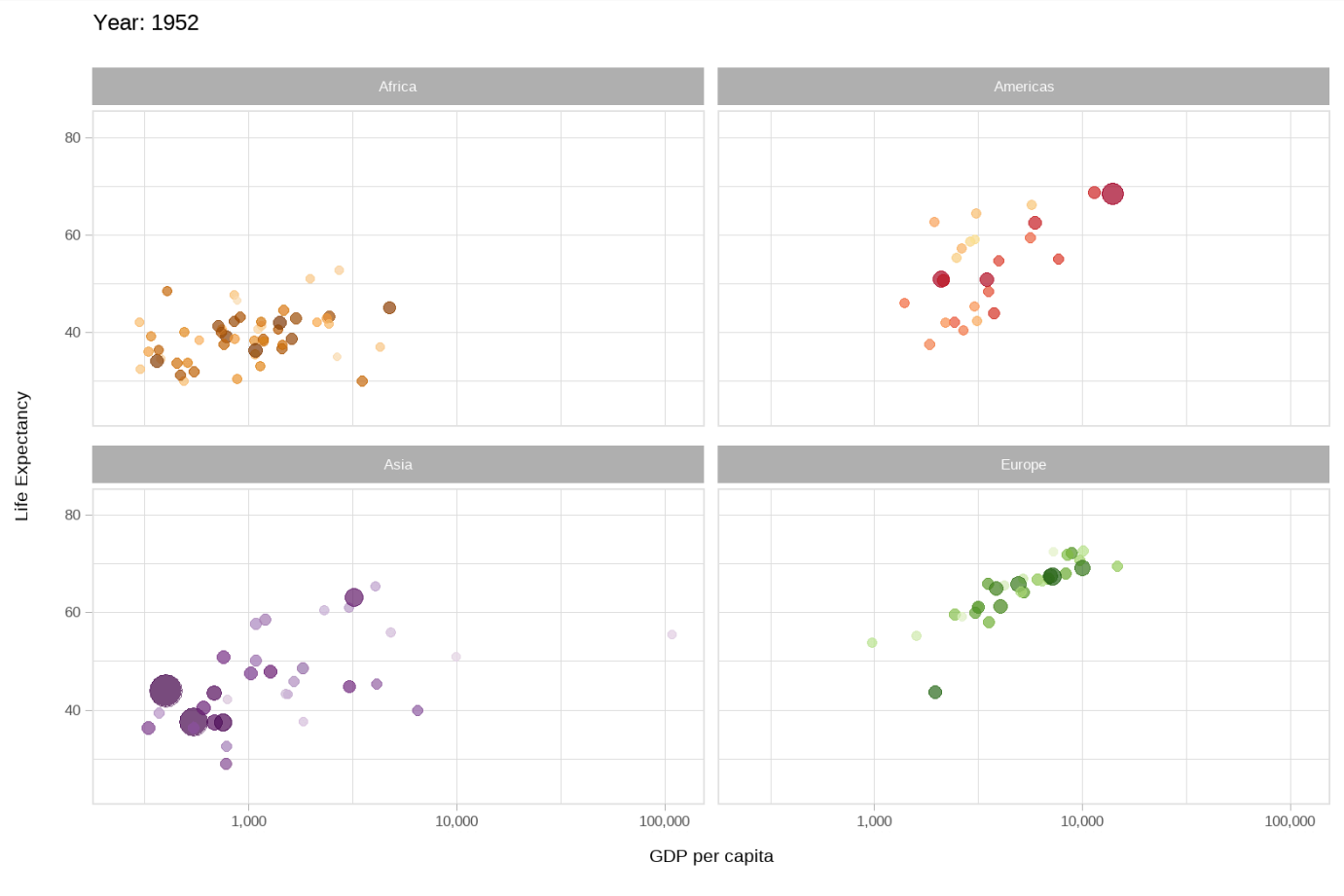


Image and code from **"Plotting in R"** by Edward Rubin, used with permission, and excluded from the overall CC license.

Marginal Distributions

Add marginal distributions to plot axes with **ggExtra**

```
# add marginal distribution to X axis
ggExtra::ggMarginal(theme_plot, margins = "x",
  type = "histogram", size = 2, fill = "aquamarine")
```

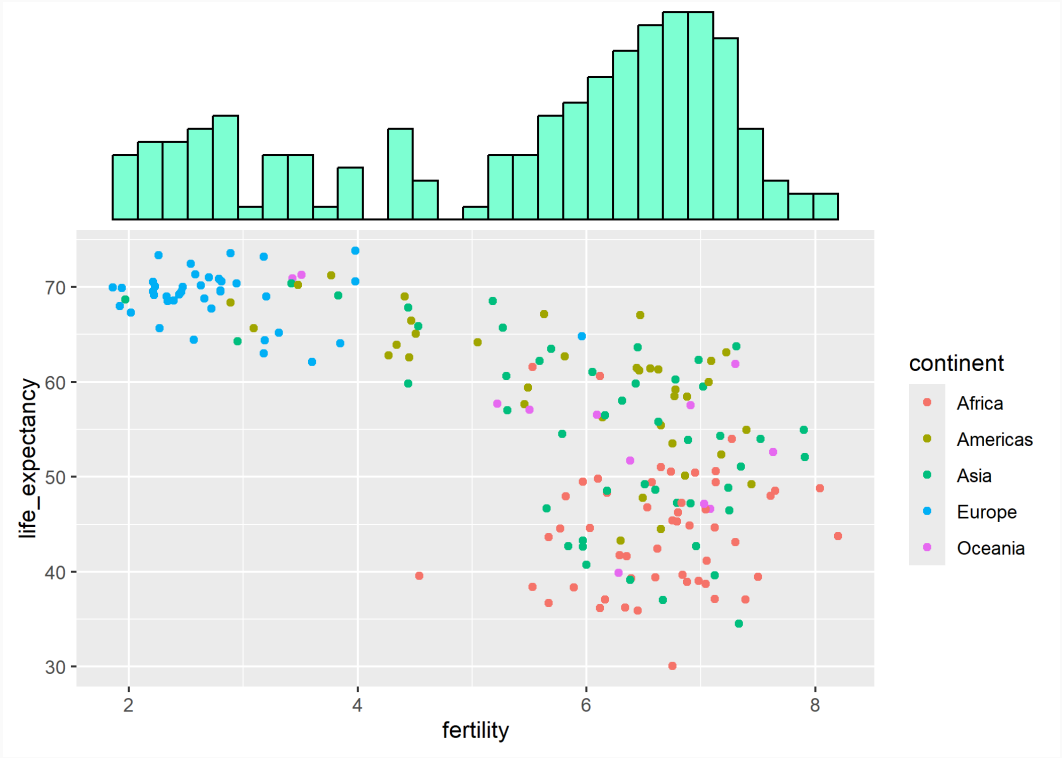


Table of Contents

This Lecture: