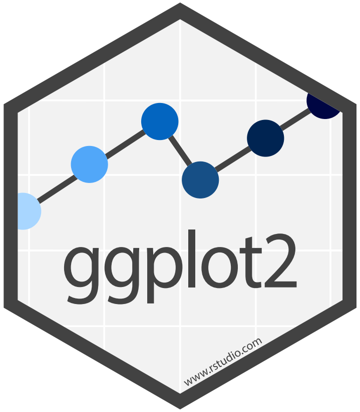
Data Visualization with ggplot2

Adam Kuczynski

UNIVERSITY OF WASHINGTON

`ggplot2()`

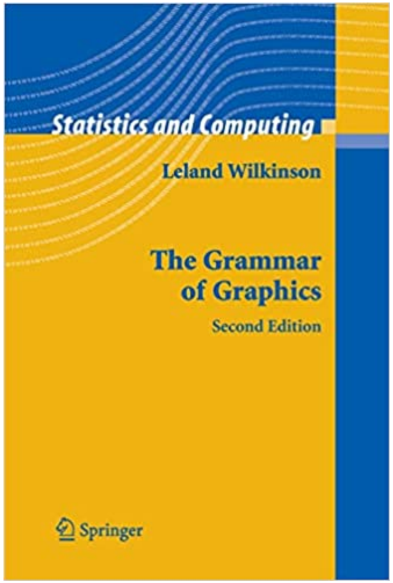


The `ggplot2` package is based on a philosophy outlined in *The Grammar of Graphics*

Understanding the philosophy is 90% of understanding how to create figures with `ggplot2`

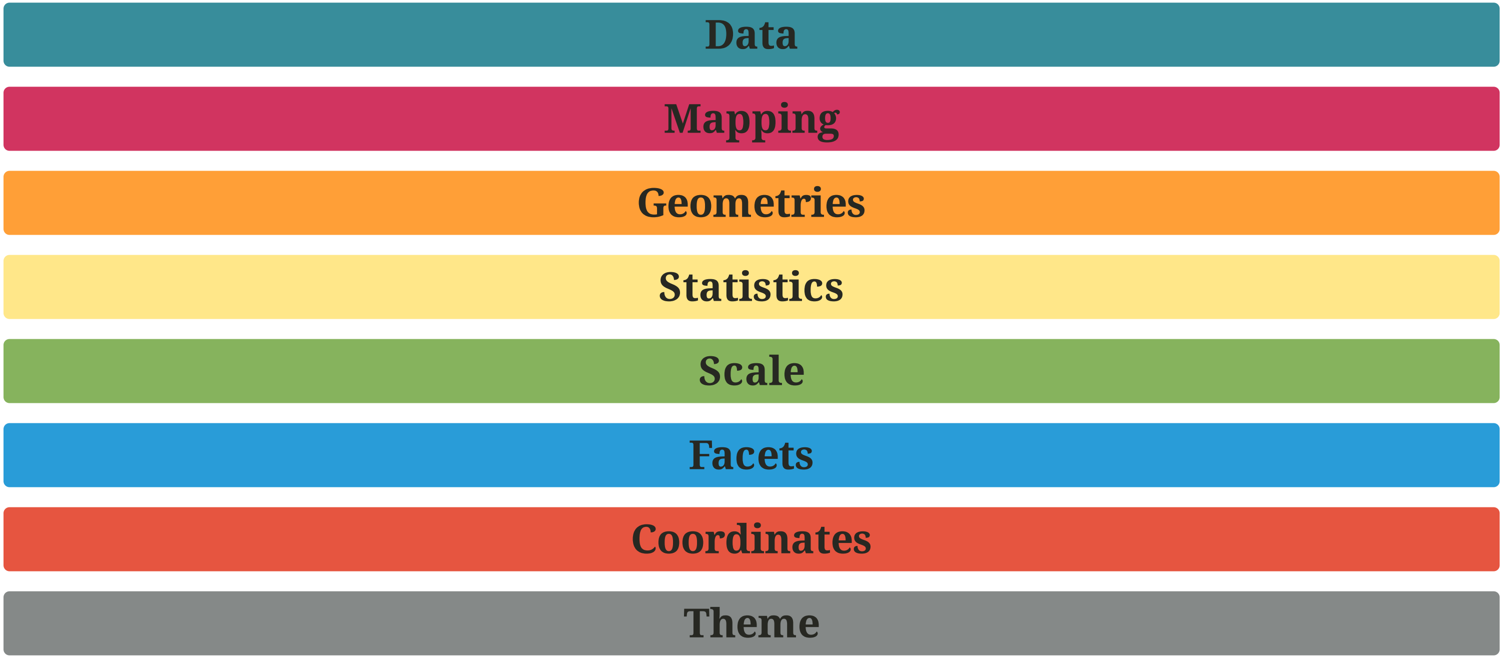
The remaining 10% is learning the various functions that correspond with each part of the philosophy

This lecture is focused on understanding that philosophy, but we will also play around with some example code



The Grammar of Graphics

Central Idea: Instead of creating a function for every single type of plot,[1] decompose graphics its its separate components/layers that can be used flexibly to create (almost) any type of plot you want



Data

Mapping

Geometries

Statistics

Scale

Facets

Coordinates

Theme

[1] New types of plots are being created nearly every day, so this would be impossible to accomplish

Example Data: `gapminder`

We will be using the `gapminder` data from the `gapminder` package for this lecture

```r
str(gapminder)
```

```
## tibble [1,704 × 6] (S3: tbl_df/tbl/data.frame)
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ year     : int [1:1704] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
##  $ lifeExp  : num [1:1704] 28.8 30.3 32 34 36.1 ...
##  $ pop      : int [1:1704] 8425333 9240934 10267083 11537966 13079460 14880372 1
##  $ gdpPercap: num [1:1704] 779 821 853 836 740 ...
```

- 142 countries (`country`)
- 5 continents (`continent`)
- 12 discrete years from 1952 to 2007 (`year`)
- life expectancy (`lifeExp`)
- population estimate (`pop`)
- GDP per capita (`gpsPercap`)

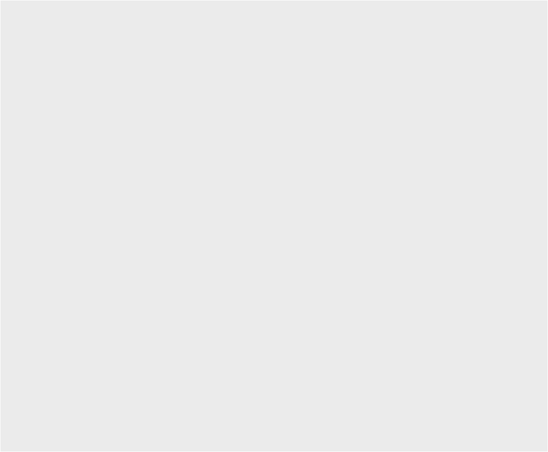
Constructing a `ggplot`

`ggplot` figures are created started with the `ggplot()` function

```
ggplot(data = NULL, mapping = aes(), ..., environment = parent.frame())
```

- `data` defined within a call to `ggplot()` are defined globally, which means each layer will used these data for plotting by default
- `mapping` defined within a call to `ggplot()` are also defined globally for each layer to use by default

```
# A blank ggplot template
ggplot()
```



The `+` sign is used to add layers

```
ggplot() +
  geom_layer() +
  another_geom_layer() +
  facet_layer() +
  theme_layer()
```

Layers are added on top of each other, so the order matters

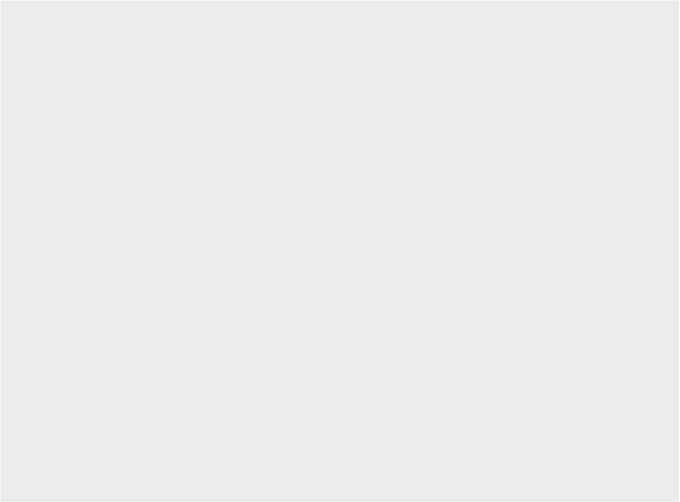
Unlike base R plots, `ggplots` can be saved as objects

```
p <- ggplot() + ...
```

Data

- This layer refers to the data that go into your figure
- Can be one dataframe (defined globally) or many dataframes (defined at each layer)
- Creating graphics is mostly about getting your data cleaned and in the format you need
    - Most of the time your data will need to be in long (aka "tidy") format
    - Sometimes you will need to supply data of summary statistics (e.g., for `geom_errorbar()`), but most of the time you will control the summary statistics within the `statistics` layer

```r
# Set global data as flights
# (from nycflights13 package)
ggplot(data = gapminder)
```



☝ Because we do not have any geometry, the plot is still blank

Mapping

Once you have your data, you need to inform the graphics function how those data fit into the plot you want to create

In other words, you need to tell the graphics which variable represents `x`, `y`, etc.

There are *tons* of different aesthetic specifications, which can be found in this documentation. Some of the most common are:

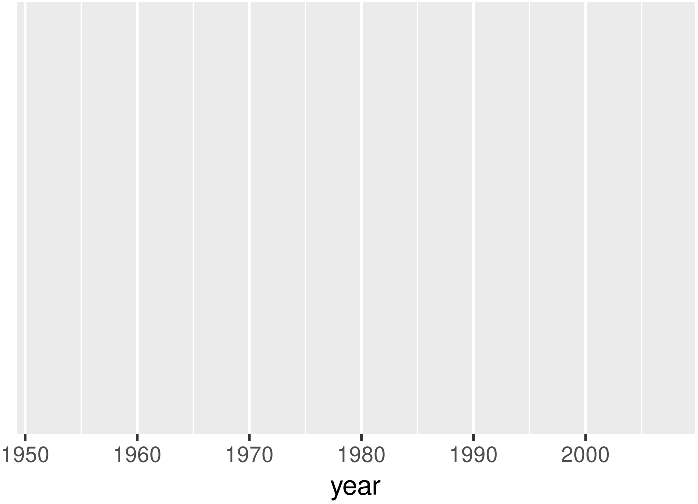
- `x`, `y` (x and y axis)
- `color`, `fill`
- `shape`
- `linetype`

Mapping occurs inside a function called `aes()`, which stands for for **aes**thetics

```
aes(x = my_x_var, y = my_y_var)
```

```
## Aesthetic mapping:
## * `x` -> `my_x_var`
## * `y` -> `my_y_var`
```

```
# Map the x-axis to `year`
# color to `continent`
ggplot(data = gapminder,
       mapping = aes(x = year, color = cont
```



☝ The plot is getting some shape, but still no data are plotted because we have not added a geometry layer

Mapping

Calls to `aes()` are *always* made within other `ggplot2` functions (i.e., they are attributes of a layer, not their own layer)

Most of the time mapping will take column names for your data that you want to map on to each aesthetic of a plot

However, `aes()` can also take expressions (i.e., R code) that determine the axes, color groups, etc.

For example:

```r
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = gdpPercap * pop,
                     color = continent == "Asia"))
```

☝ From the `gapminder` data:

- map `year` onto the x-axis
- total GDP (`gdpPercap * pop`) on the y-axis
- different colors where `contintent` `==` and `!=` "Asia"

Mapping vs. Setting

Arguments inside `aes()` (color, size, shape, etc.) **map** aesthetics to the data such that the colors, sizes, shapes, etc. *depend* on the data

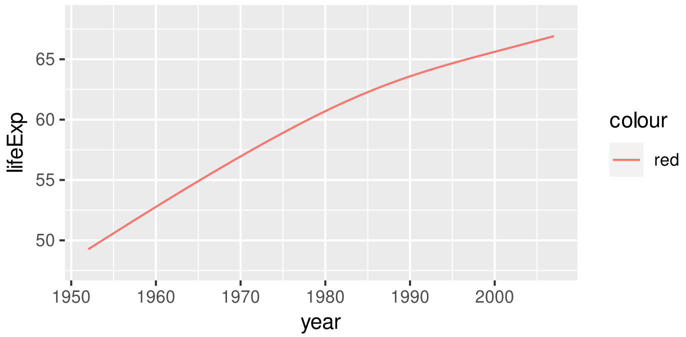
- Used to plot different colors, shapes, sizes, etc. based on groups/condition in your data

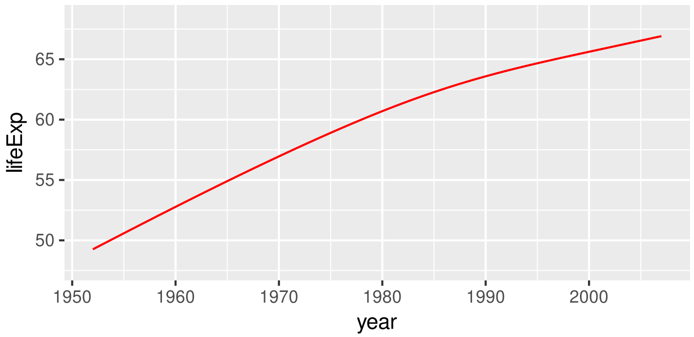
These same arguments placed outside `aes()` (e.g., within `geom_*()`) **set** aesthetics to the layer such that the colors *do not depend* on the data

- Used to change the colors, shapes, sizes, etc. of the entire plot/layer

```
# Mapping (wrong)
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp,
                     color = "red")) +
  ...
```

```
# Setting (right)
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  ...(color = "red")
#
```



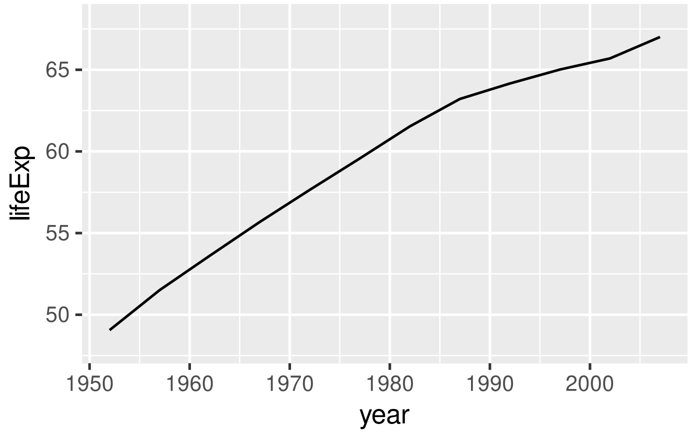


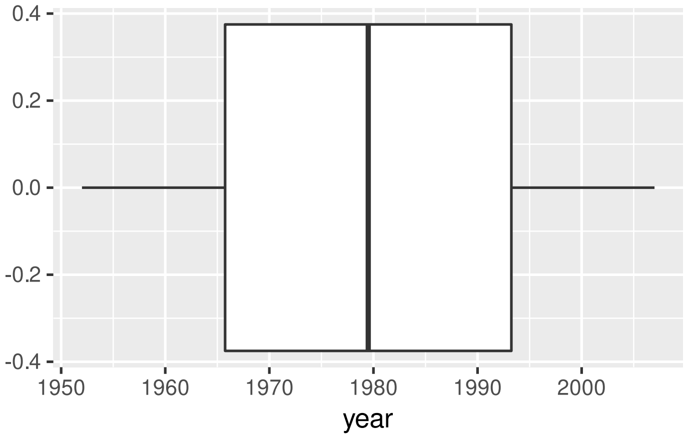
Geometries

- Mostly what you think about in `ggplot2`
- Take all the **scale** values from come from mapping and may have been transformed by **statistics** and interprets/plots them in some way
  - e.g., a line geometry (`geom_line()`) interprets data on way and creates lines on your figure while a boxplot geometry (`geom_boxplot()`) interprets the data another way

```r
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_line(stat = "summary")
```

```r
ggplot(data = gapminder,
       mapping = aes(x = year)) +
  # No `y` mapping needed for boxplot
  geom_boxplot()
```



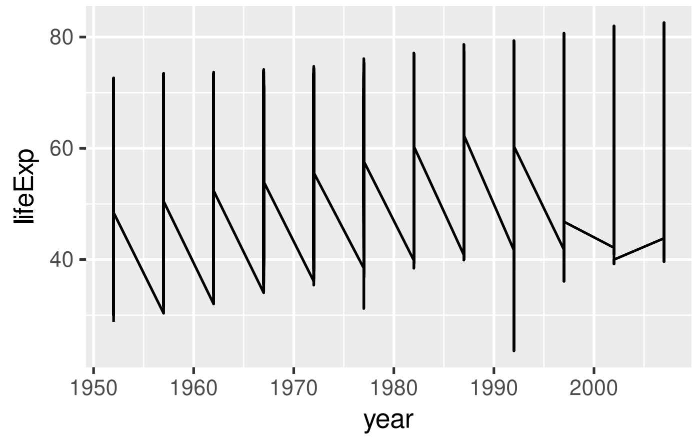


Geometries

- Geometries are intimately intertwined with **statistics**, and each `geom_*()` has a default statistic (`stat`) assigned
- The default statistic for `geom_line()` is identity, which means ("leave the data as is").

If we create a line geometry with the data "as is" we get the following plot:

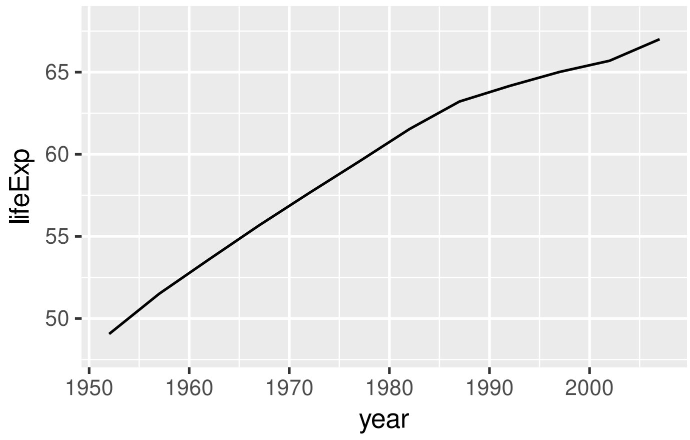
```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_line() # stat = "identity"
```



☝️ Plots every single point across every single year

When we change the default to summary, we get a plots of mean values (changed with `fun`)

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_line(stat = "summary")
```



☝️ Plots mean values for all observations within each year

Geometries

- Different geometries do not necessary share the same mapping. For example, `geom_point()` needs (at minimum) an `x` and `y` mapping, but `geom_histogram()` only needs an `x` mapping (the statistic determines the y-axis)

- There is an "Aesthetics" section in the help page for each `geom` that describes the required and optional mapping parameters

    - For example, `goem_linerange()` needs `x` *or* `y`, `ymin` *or* `xmin`, and `ymax` *or* `xmax` while `geom_histogram()` needs only `x` or `y`

Multiple Geometries

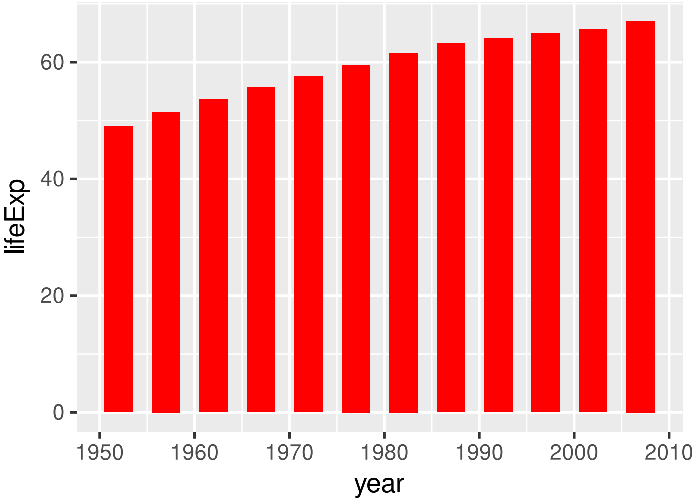
- You can (and often will) have multiple layers of geometries in the same figure

- The order or your geometries matter, because each later is plotted on top of the previous layers

Geometries

Add bar geometry

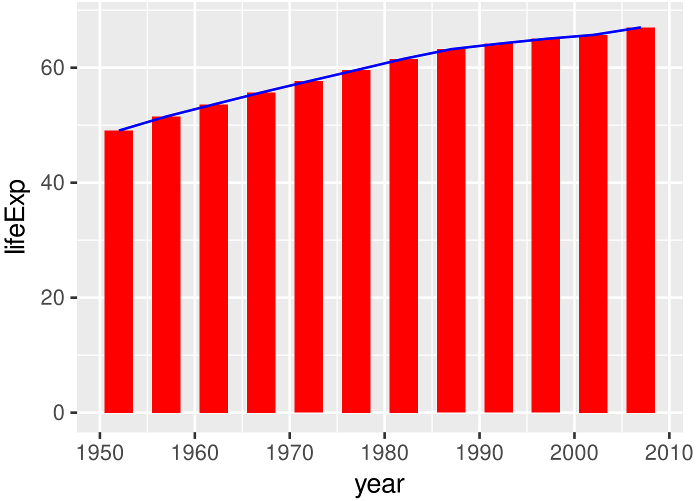
```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_bar(stat = "summary",
           width = 3,
           fill = "red")
```



Geometries

Add line geometry

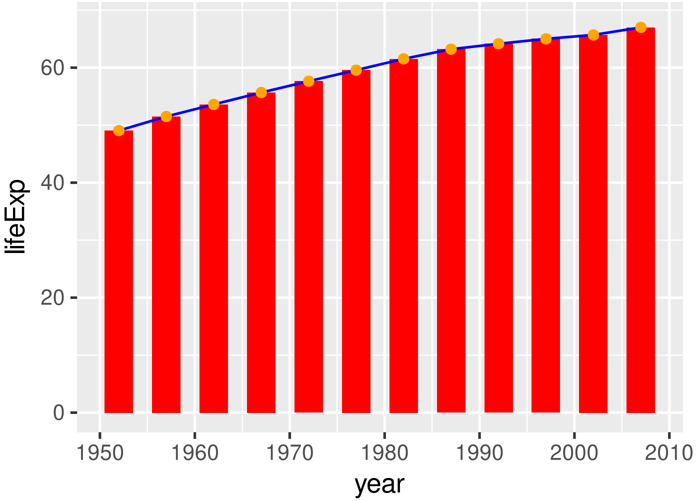
```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_bar(stat = "summary",
           width = 3,
           fill = "red") +
  geom_line(stat = "summary",
            color = "blue")
```



Geometries

Add point geometry

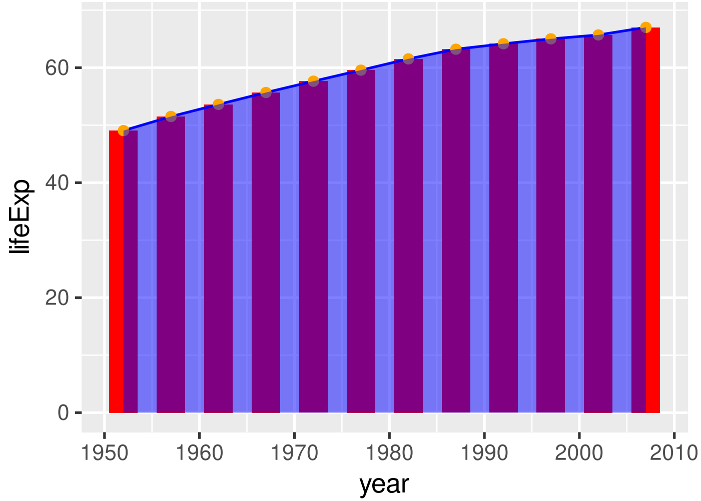
```r
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_bar(stat = "summary",
           width = 3,
           fill = "red") +
  geom_line(stat = "summary",
            color = "blue") +
  geom_point(stat = "summary",
             color = "orange")
```



Geometries

Add area geometry

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_bar(stat = "summary",
           width = 3,
           fill = "red") +
  geom_line(stat = "summary",
            color = "blue") +
  geom_point(stat = "summary",
             color = "orange") +
  geom_area(stat = "summary",
            alpha = .5,
            fill = "blue")
```



Statistics

- Your data do not always have the required statistics for each type of figure
    - For example, plotting a boxplot requires calculating the 25th, 50th, and 75th percentiles of your data and the interquartile range
    - Sometimes your data are exactly what is needed (e.g., creating a scatterplot), in which case you set your statistic to **identity** which just passes your data on to that layer
- Provides convenience and flexibility because you do not need to know how your data need to be manipulated for each type of figure
- However, sometimes you do need to manipulate your data to get the correct aesthetic mapping for a geom (e.g., creating errobars)

```r
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_bar(stat = "summary") +
  geom_errorbar()
```

```
Error: geom_errorbar
requires the following
missing aesthetics: ymin
and ymax or xmin and xmax
```

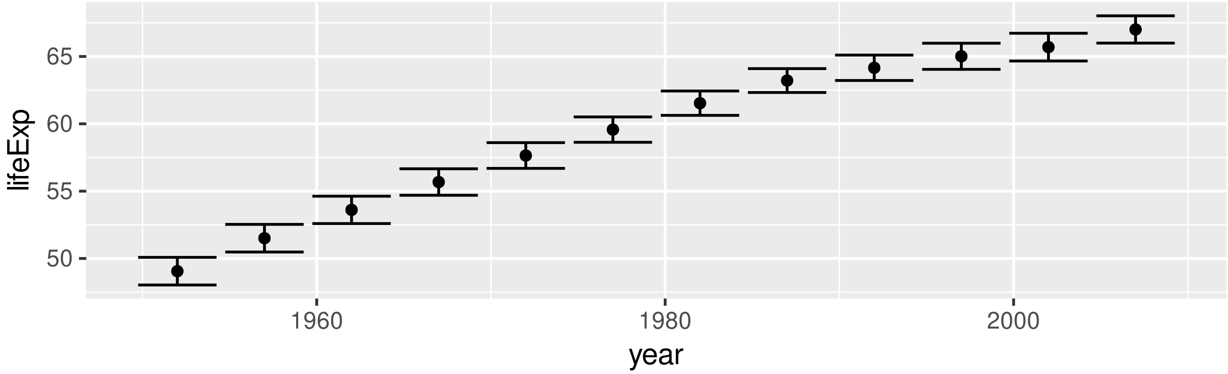
Statistics: Errorbars

Step 1: create summary statistics from our data (mean and standard error) :

```
gp_summary <- gapminder %>%
  group_by(year) %>%
  summarize(mean_lifeExp = mean(lifeExp, na.rm = T),
            se_lifeExp = sd(lifeExp, na.rm = T) / sqrt(n()))
```

Step 2: Supply these data to the `geom_errorbar()` layer to control the height of the errorbars

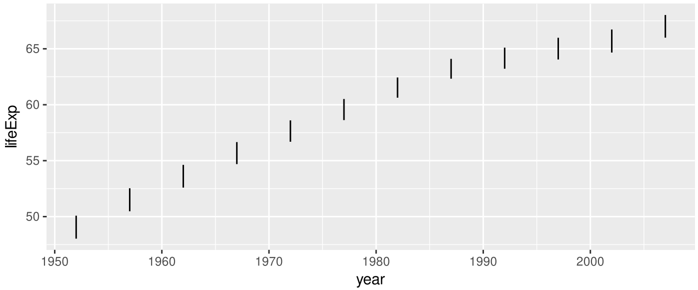
```
ggplot(data = gapminder, mapping = aes(x = year, y = lifeExp)) +
  geom_point(stat = "summary") +
  geom_errorbar(data = gp_summary,
                mapping = aes(x = year, y = mean_lifeExp,
                              ymin = mean_lifeExp - se_lifeExp,
                              ymax = mean_lifeExp + se_lifeExp))
```



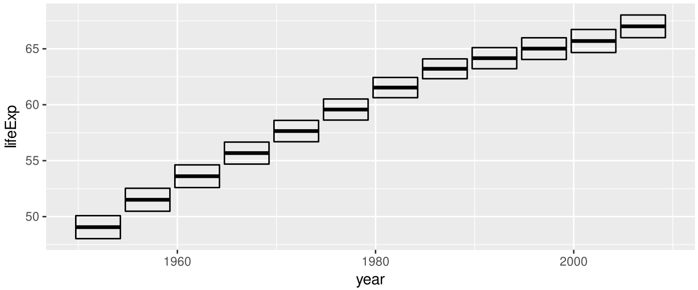
Statistics

Errorbar Types

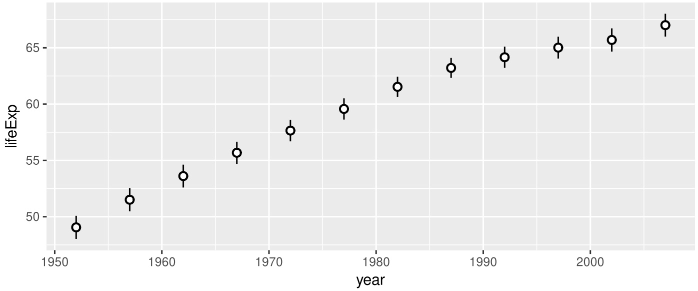
`geom_linerange()`



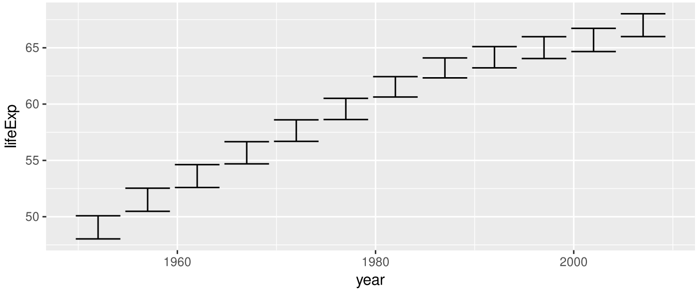
`geom_crossbar()`



`geom_pointrange()`



`geom_errorbar()`



Statistics

Statistics are linked to geometries such that each geometry requires a statistic (and vice versa: each statistic requires a geometry)

Thus, geometries have default statistics that try to guess what you want to plot but that can also be changed

Defaults for common geometries:

- `geom_point(stat = "identity"`
- `geom_count(stat = "sum")`
- `geom_jitter(stat = "identity")`
- `geom_bar(stat = "count")`
- `geom_density(stat = "density")`
- `geom_histrogram(stat = "bin")`
- `geom_boxplot(stat = "boxplot")`
- `geom_violin(stat = "ydensity")`
- `geom_rug(stat = "identity")`
- `geom_freqpoly(stat = "bin")`
- `geom_quantile(stat = "quantile")`
- `geom_smooth(stat = "smooth")`

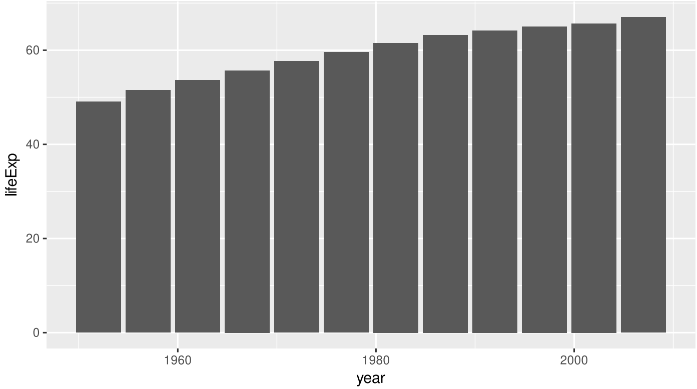
Statistics

Geometries can be created using `geom_*()` and passing in the statistic as an argument (as we have seen) *or* using `stat_*()` and passing the geometry in as an argument

There's no one right way to do this, although it is most common to create the geometry with `geom_*()` rather than `stat_*()`

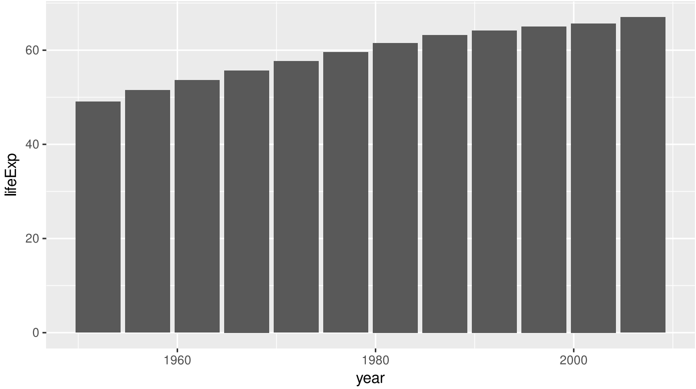
`geom_*()`

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_bar(stat = "summary")
```



`stat_*()`

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  stat_summary(geom = "bar")
```

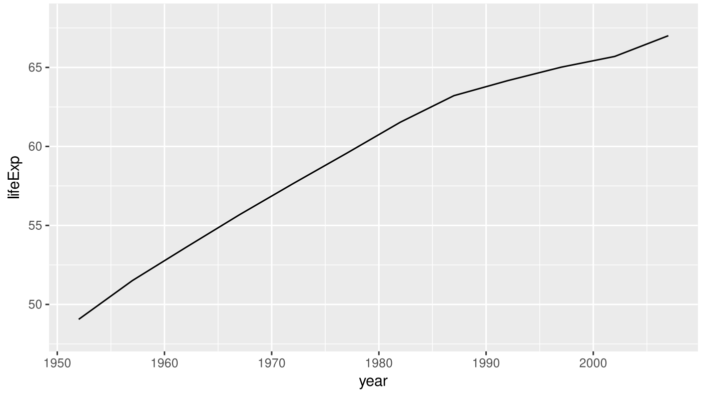


Mapping

Now that we know how to create geometry and statistics layers we can understand aesthetic mapping more completely

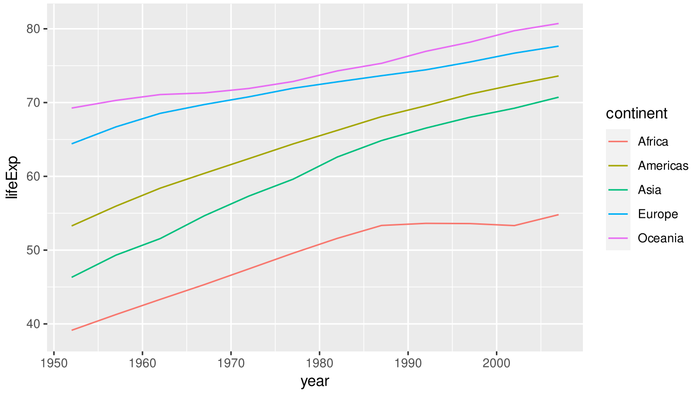
No color mapping

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  # No color mapping
  geom_line(stat = "summary")
```



Color mapped to continent

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp,
                     color = continent)) +
  geom_line(stat = "summary")
```

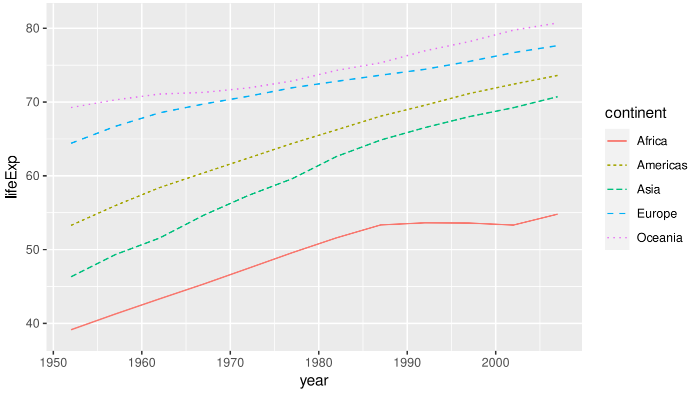


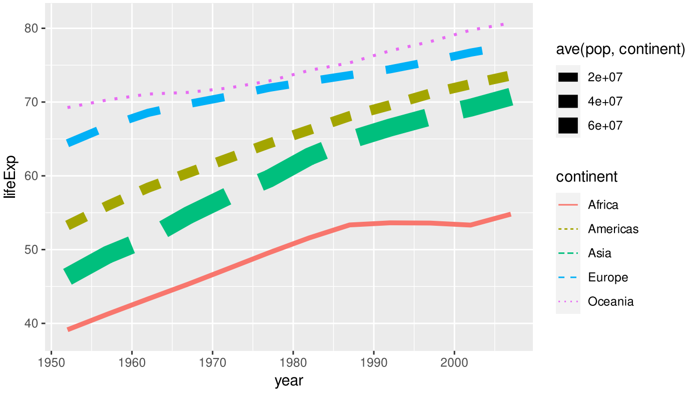
Mapping

You can map the same (or different) columns to multiple aesthetics within one call to `aes()`

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp,
                     color = continent,
                     linetype = continent)) +
  # Size equal among groups
  geom_line(stat = "summary")
```

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp,
                     color = continent,
                     linetype = continent,
                     size = ave(pop, contin
  geom_line(stat = "summary")
```

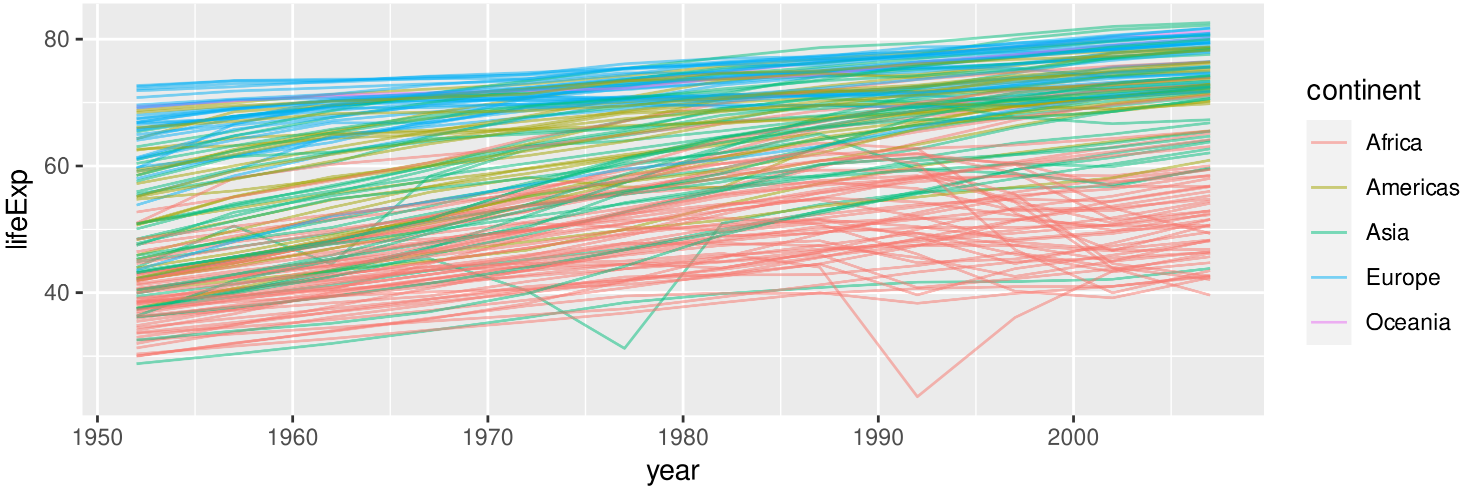




Mapping: Layer-level

Line geometry for each country, color by continent

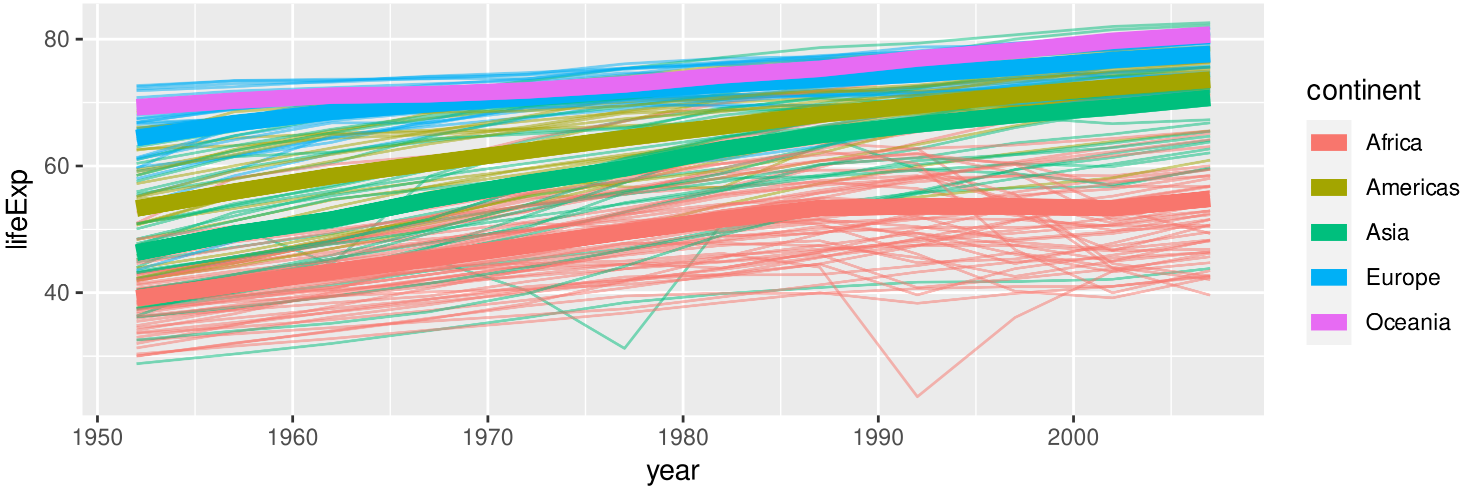
```
ggplot(data = gapminder,
       mapping = aes(x = year, y = lifeExp, color = continent)) +
  geom_line(mapping = aes(group = country),
            stat = "summary",
            alpha = .5)
#
#
```



Mapping: Layer-level

Line geometry for each continent, color by continent

```r
ggplot(data = gapminder,
       mapping = aes(x = year, y = lifeExp, color = continent)) +
  geom_line(mapping = aes(group = country),
            stat = "summary",
            alpha = .5) +
  geom_line(stat = "summary",
            size = 3)
```



Scales

How are properties of the axes, colors, and other aethetics determined? Scales!

Scales control the details of how data values are translated to visual properties (e.g., plot Africa with #F8766D, Americas with #B79F00, etc.)

All geometries are given default scales which you can override with the `scale_*()` function

To get the scales for each layer of your ggplot figure, use `build_ggplot()`

```r
p <- ggplot(data = gapminder,
       mapping = aes(x = year, y = lifeExp, color = continent)) +
  geom_line(stat = "summary")

ggplot_build(p)$data[[1]] %>%
  str()
```

```
## 'data.frame':    60 obs. of  11 variables:
##  $ colour     : chr  "#F8766D" "#F8766D" "#F8766D" "#F8766D" ...
##  $ x          : num  1952 1957 1962 1967 1972 ...
##  $ group      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ y          : num  39.1 41.3 43.3 45.3 47.5 ...
##  $ ymin       : num  38.4 40.5 42.5 44.5 46.6 ...
##  $ ymax       : num  39.8 42 44.1 46.2 48.3 ...
##  $ PANEL      : Factor w/ 1 level "1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ flipped_aes: logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ size       : num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
##  $ linetype   : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ alpha      : logi  NA NA NA NA NA NA ...
```

Scales

Scale functions have the syntax: `scale_<aesthetic>_<type>` where `<aesthetic>` refers to each aesthetic mapping (`x`, `y`, `color`, etc.) and `<type>` refers to the type of scale (continuous, discrete, log10, etc.)

Axes scales

- `scale_x_continous()`, `scale_y_continuous()` (transform with `trans` argument)
- `scale_x_log10()`, `scale_y_log10()`
- `scale_x_sqrt()`, `scale_y_sqrt()`
- `scale_x_reverse()`, `scale_y_reverse()`
- `scale_x_discrete()`, `scale_y_discrete()`
- `scale_x_binned()`, `scale_y_binned()`

Color, shape, size scales

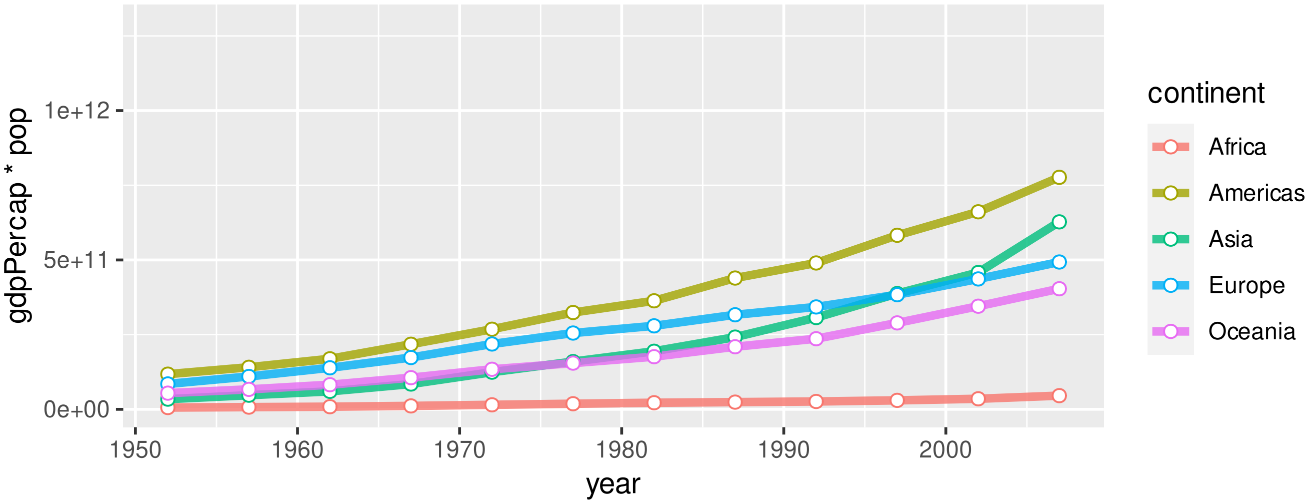
- `scale_color_continuous()`, `scale_shape_continuous()`, `scale_size_continuous()`
- `scale_color_discrete()`, `scale_shape_discrete()`, `scale_size_discrete()`
- `scale_color_binned()`, `scale_shape_binned()`, `scale_size_binned()`
- `scale_color_brewer()`, `scale_shape_brewer()`, `scale_size_brewer()`

There are dozens of different types of scales in `ggplot2`, all of which can be found in this documentation.

Also check out the scales package

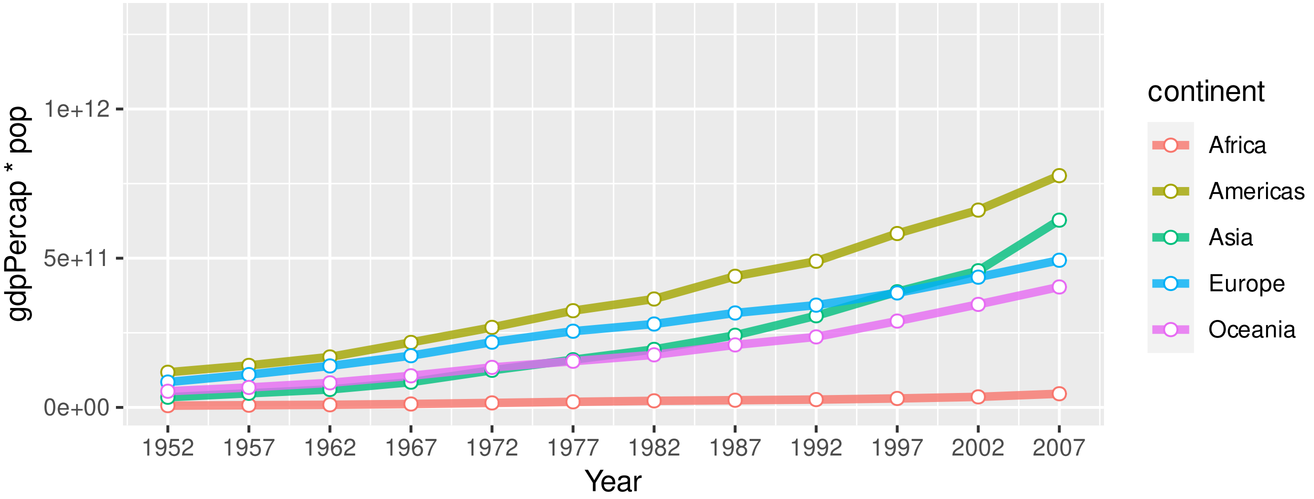
Scales: Example

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = gdpPercap * pop,
                     color = continent)) +
  geom_line(stat = "summary", size = 1.5) +
  geom_point(stat = "summary", shape = 21, fill = "white", size = 2)
  #
  #
  #
  #
  #
```



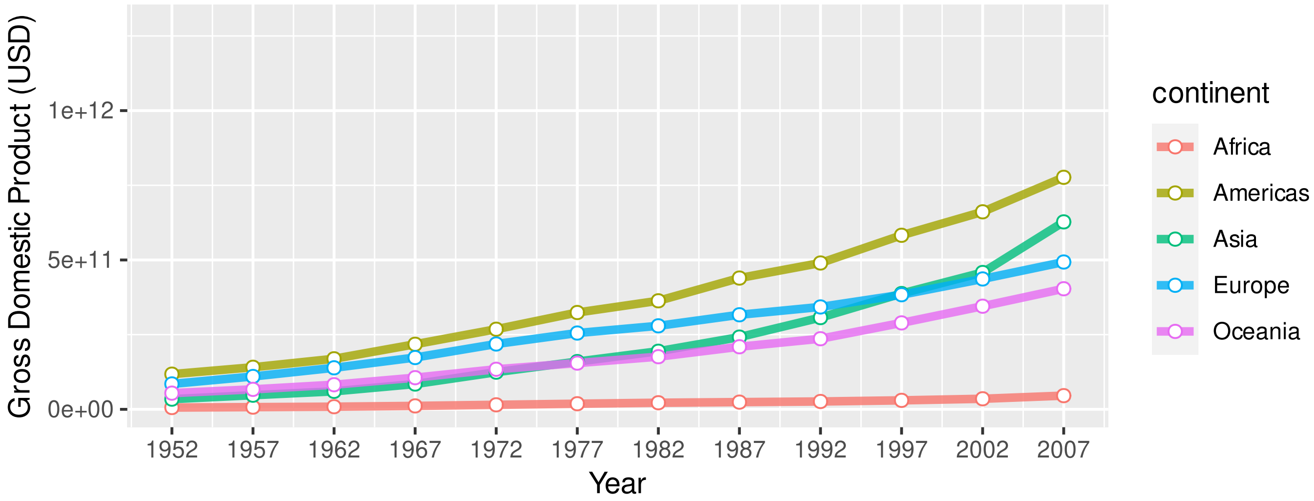
Scales: Example

```r
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = gdpPercap * pop,
                     color = continent)) +
  geom_line(stat = "summary", size = 1.5) +
  geom_point(stat = "summary", shape = 21, fill = "white", size = 2) +
  scale_x_continuous(name = "Year",
                     breaks = unique(gapminder$year))
  #
  #
  #
```



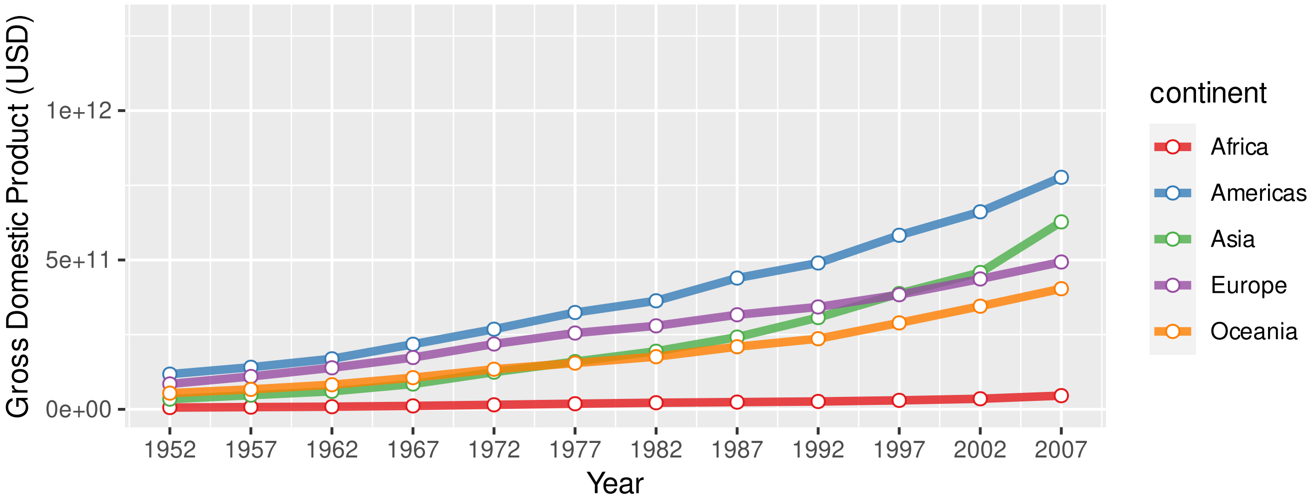
Scales: Example

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = gdpPercap * pop,
                     color = continent)) +
  geom_line(stat = "summary", size = 1.5) +
  geom_point(stat = "summary", shape = 21, fill = "white", size = 2) +
  scale_x_continuous(name = "Year",
                     breaks = unique(gapminder$year)) +
  scale_y_continuous(name = "Gross Domestic Product (USD)")
  #
  #
```



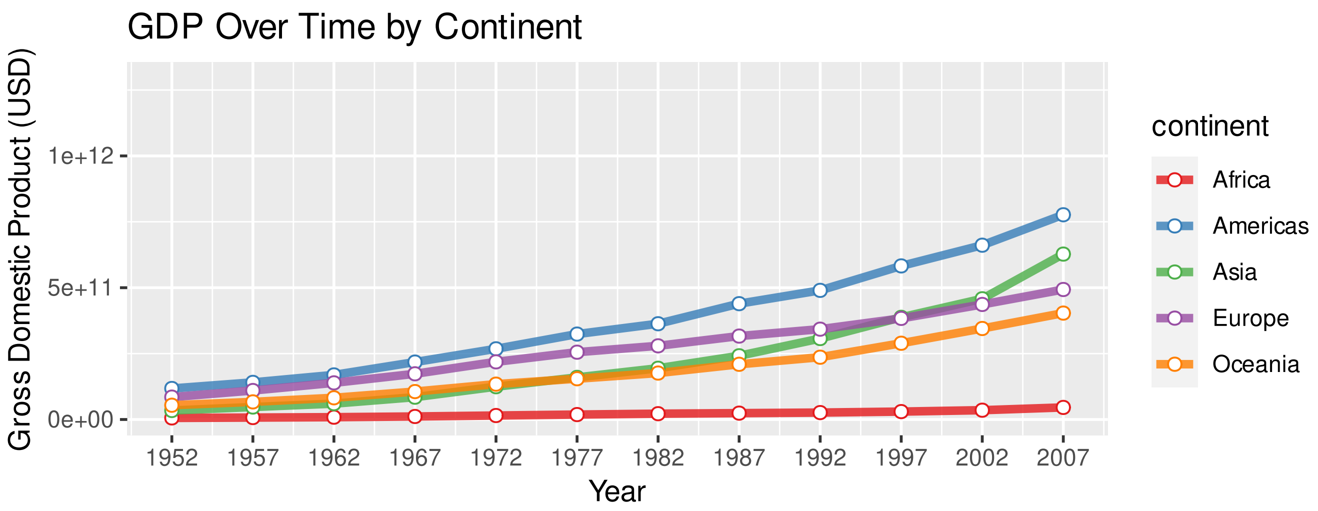
Scales: Example

```r
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = gdpPercap * pop,
                     color = continent)) +
  geom_line(stat = "summary", size = 1.5) +
  geom_point(stat = "summary", shape = 21, fill = "white", size = 2) +
  scale_x_continuous(name = "Year",
                     breaks = unique(gapminder$year)) +
  scale_y_continuous(name = "Gross Domestic Product (USD)") +
  scale_color_brewer(palette = "Set1")
  #
```



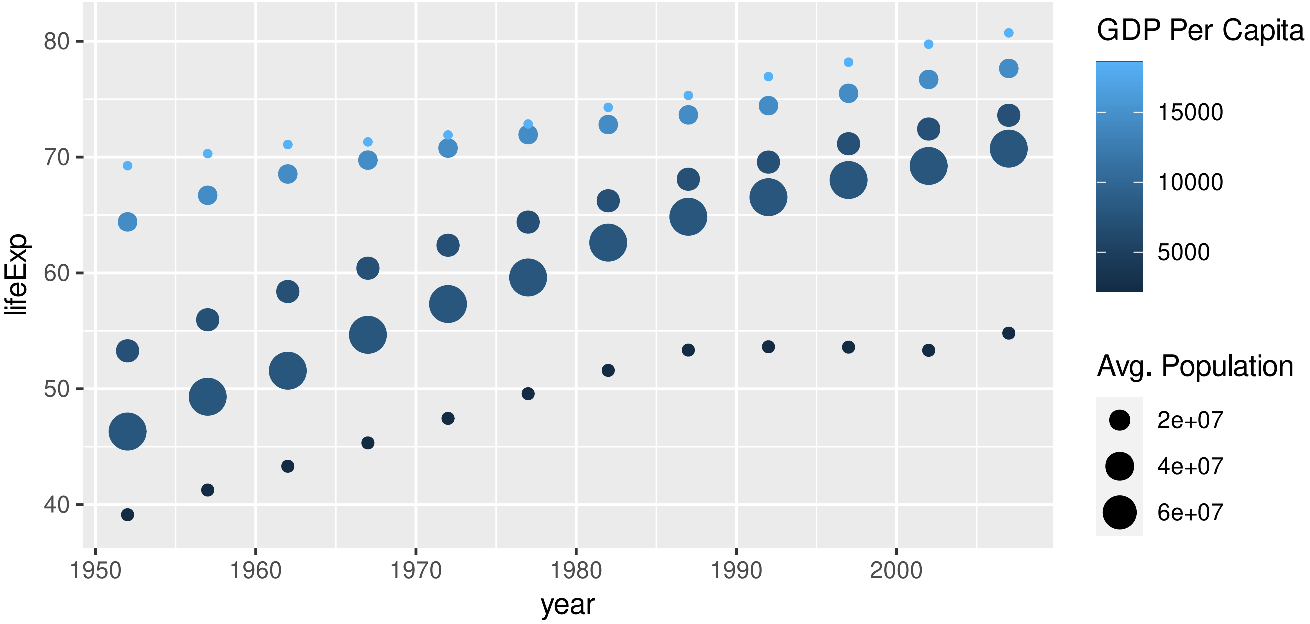
Scales: Example

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = gdpPercap * pop,
                     color = continent)) +
  geom_line(stat = "summary", size = 1.5) +
  geom_point(stat = "summary", shape = 21, fill = "white", size = 2) +
  scale_x_continuous(name = "Year",
                     breaks = unique(gapminder$year)) +
  scale_y_continuous(name = "Gross Domestic Product (USD)") +
  scale_color_brewer(palette = "Set1") +
  ggtitle("Life Expectancy Over Time by Continent")
```



Scales: Continuous Color

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp,
                     group = continent,
                     size = ave(pop, continent),
                     color = ave(gdpPercap, continent))) +
  geom_point(stat = "summary") +
  scale_color_continuous(name = "GDP Per Capita") +
  scale_size_continuous(name = "Avg. Population")
```

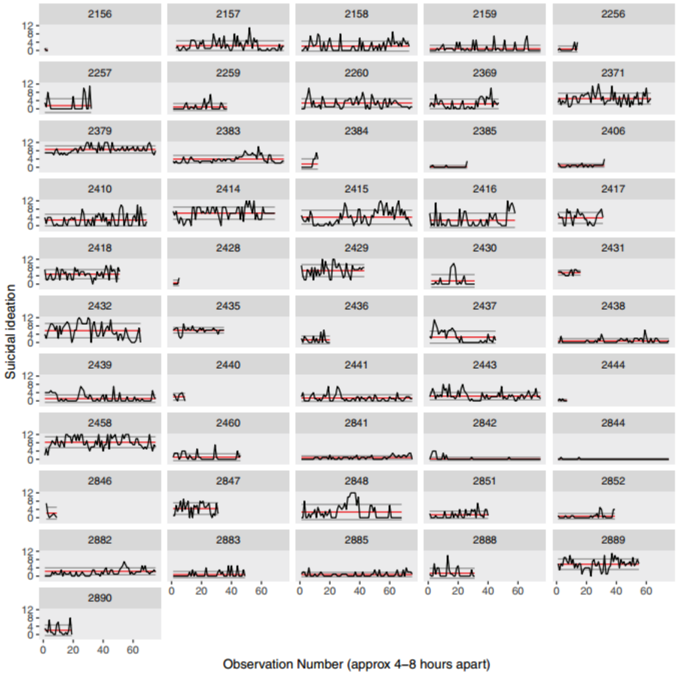


Facets

- Often we are focused on creating one figure per plotting area, but we are not constrained to this and may want to create multiple subplots when looking at our data

- **Facets** are multiple panels of plots, with the same plotting logic, on different groups of your data

- Facets are most helpful when you are investigating your data, but they may help you create figures for publication as in Kleiman et al. (2017)



- Use facets to prevent overplotting (plotting too much data in one figure)

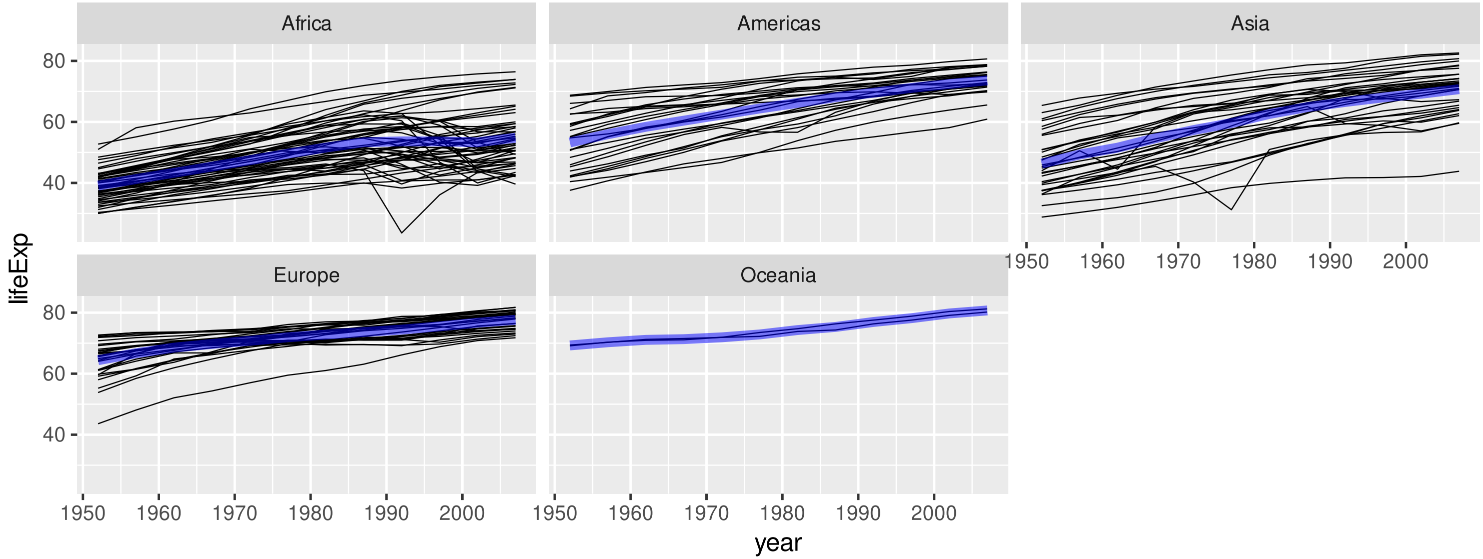
- Two different kinds of facets: `facet_wrap()` and `facet_grid()`

- Because facets are extendable, there are packages (e.g., ggh4x) with additional types of facets

Facets: `facet_wrap()`

`facet_wrap()` takes a column from your data with a grouping structure and creates several subplots for each group (`facet_wrap(~ groupvar)`)

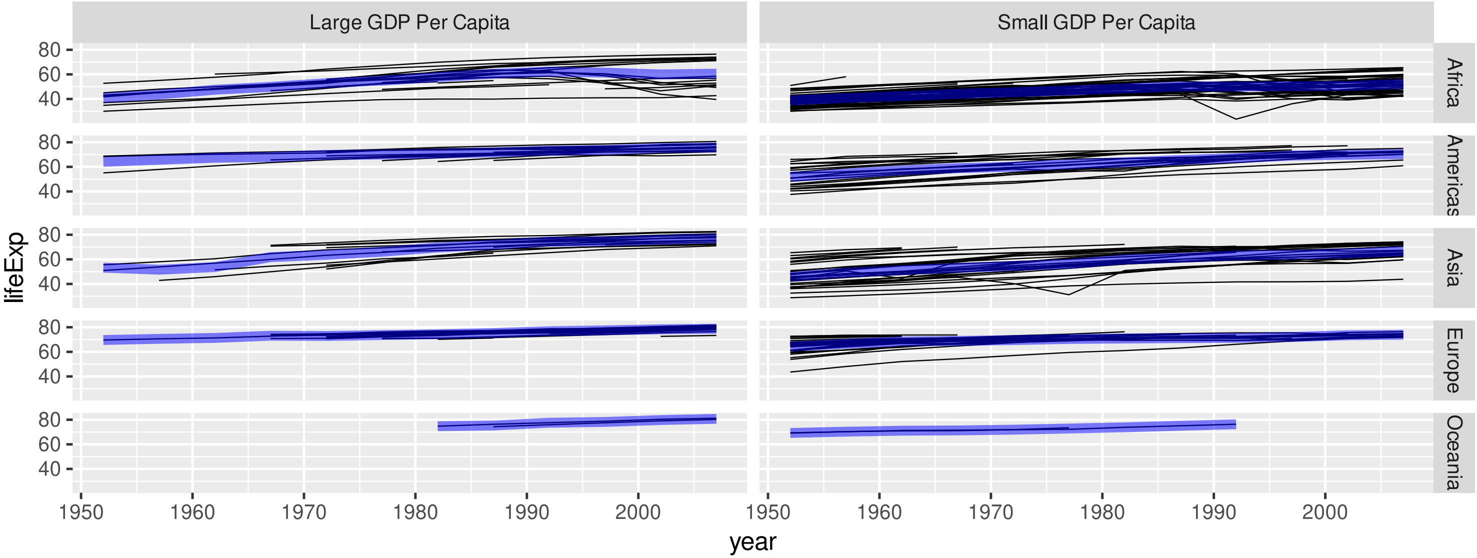
```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_line(mapping = aes(group = country), stat = "summary", size = .25) +
  geom_line(stat = "summary", size = 2, alpha = .5, color = "blue") +
  facet_wrap(~ continent)
```



Facets: `facet_grid()`

`facet_grid()` takes two grouping variables and creates plots that show the intersection between them (`facet_grid(group1 ~ group2)`)

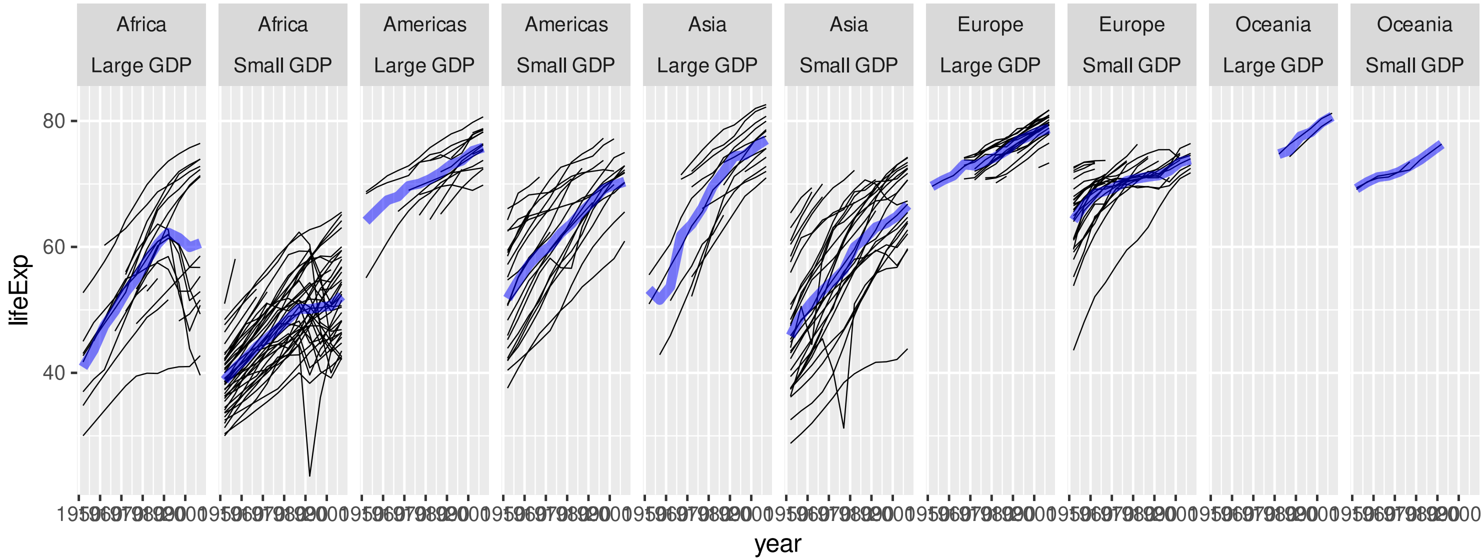
```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_line(mapping = aes(group = country), stat = "summary", size = .25) +
  geom_line(stat = "summary", size = 2, alpha = .5, color = "blue") +
  facet_grid(continent ~ gdprelative) # GDP larger/smaller to continent mean
```



Facets: `facet_grid()`

`facet_grid()` can also be arranged with n_{groups} panels with `~ group1 + group2`

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_line(mapping = aes(group = country), stat = "summary", size = .25) +
  geom_line(stat = "summary", size = 2, alpha = .5, color = "blue") +
  facet_grid(~ continent + gdprelative)
```

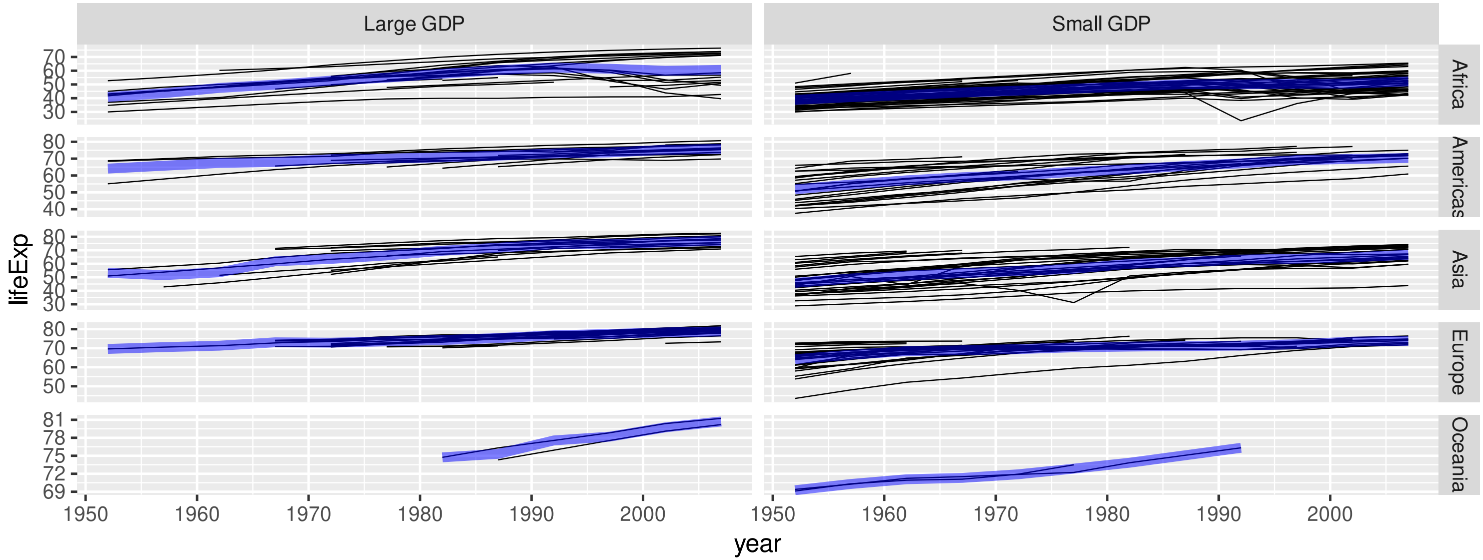


UNIVERSITY of WASHINGTON

Facets + Scales

- By default, facets fix the `x` and `y` scales across all plots
- Often this makes sense, because you want to compare the same data across different groups, but sometimes you may want to free either/both of the axis scales. You can do this with the `scales` argument (`"free"`, `"free_x"`, `"free_y"`)

```
ggplot(...) +
  facet_grid(continent ~ gdprelative,
             scales = "free")
```



Coordinates

The **coordinate** system represents a physical mapping of the plot's aesthetics onto the screen

Many types of coordinate systems, but we are most used to the Cartesian system (`x`, `y` value pairs)

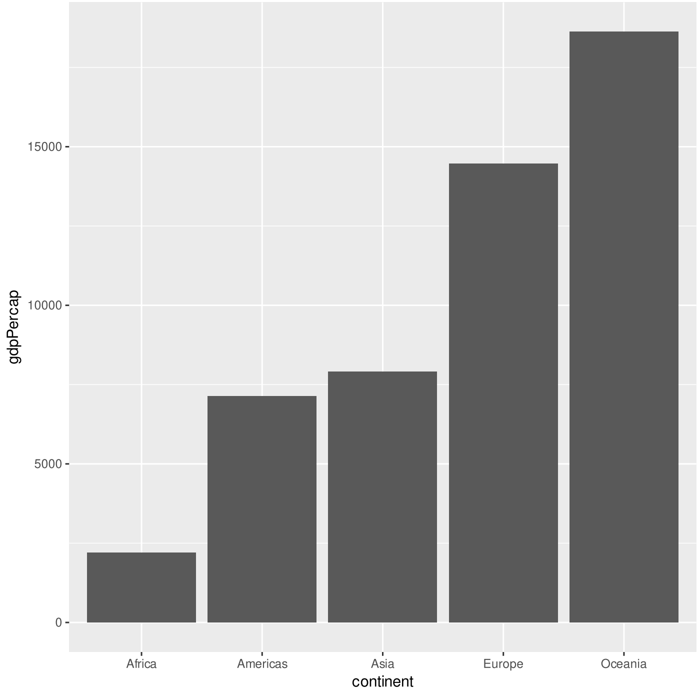
Types of coordinate systems in ggplot2:

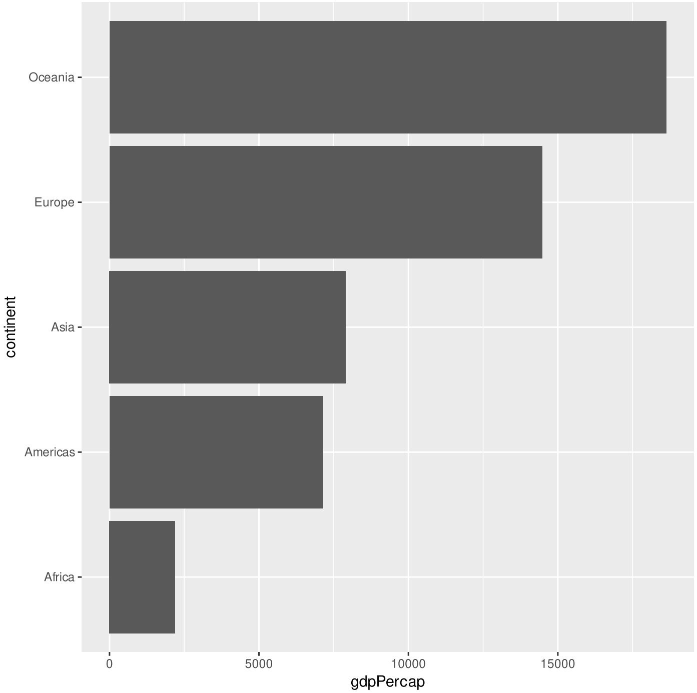
- `coord_cartesian()`: Cartesian coordinates
- `coord_trans()`: Transformed Cartesian coordinate system
- `coord_fixed()`: Cartesian coordinates with a fixed aspect ratio
- `coord_flip()`: Cartesian coordinates with `x` and `y` flipped
- `coord_polar()` polar coordinates
- `coord_map()`, `coord_quickmap()`: map projections (latitude, longitude)

Coordinates: `coord_flip()`

```
ggplot(data = gapminder,
       mapping = aes(x = continent,
                     y = gdpPercap)) +
  geom_bar(stat = "summary")
  #
```

```
ggplot(data = gapminder,
       mapping = aes(x = continent,
                     y = gdpPercap)) +
  geom_bar(stat = "summary") +
  coord_flip()
```



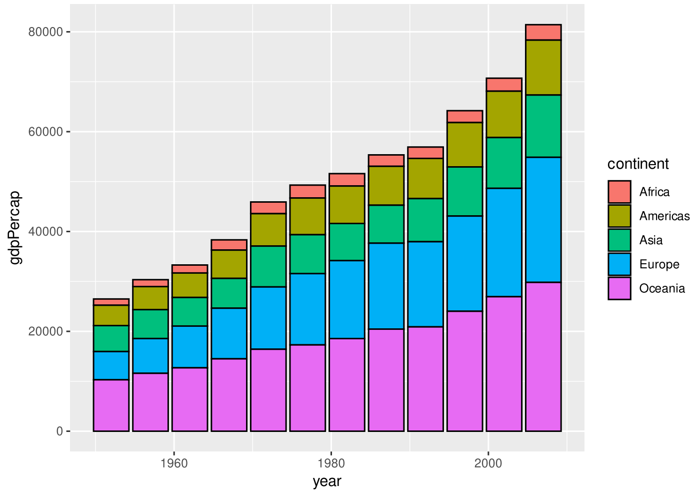


Coordinates: `coord_polar()`

`coord_polar()` interprets `x` and `y` as the radius and angle, respecitvely

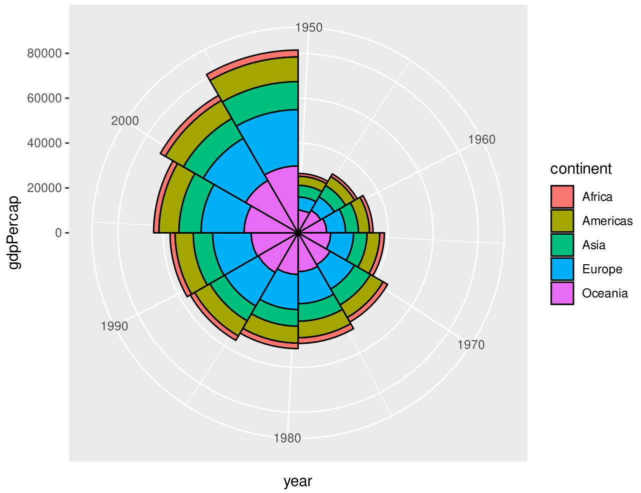
Bar Chart

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = gdpPercap,
                     fill = continent)) +
  geom_bar(stat = "summary", color = "black
  #
```



Coxcomb Plot

```
ggplot(data = gapminder,
       mapping = aes(x = year,
                     y = gdpPercap,
                     fill = continent)) +
  geom_bar(stat = "summary", width = 5, col
  coord_polar()
```

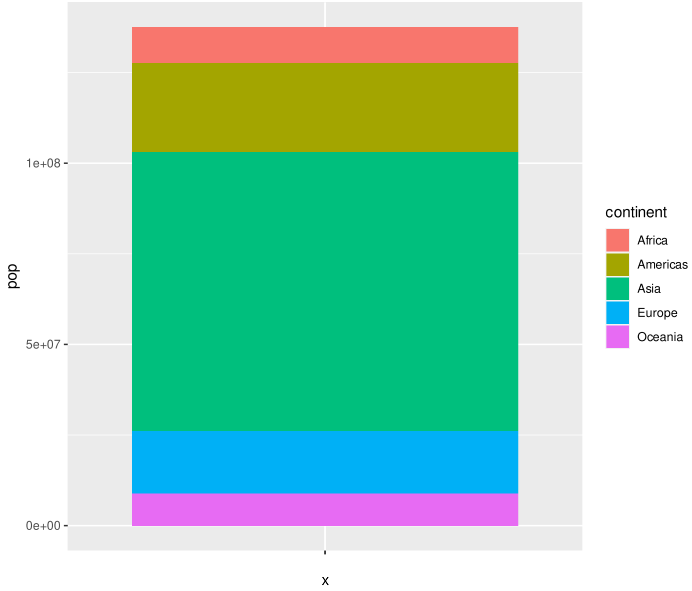


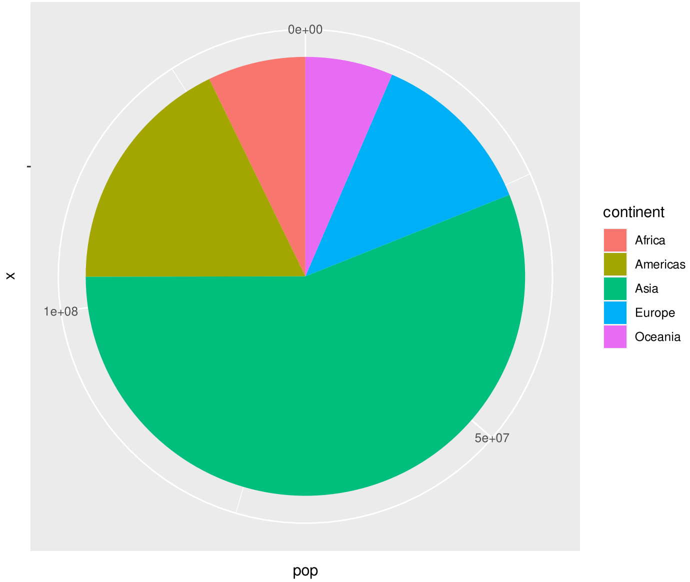
Coordinates: `coord_polar()`

Pie Chart

```r
ggplot(data = gapminder,
       mapping = aes(x = "", y = pop, fill
  geom_bar(stat = "summary") +
  #
```

```r
ggplot(data = gapminder,
       mapping = aes(x = "", y = pop, fill
  geom_bar(stat = "summary") +
  coord_polar(theta = "y") # map angle to y
```

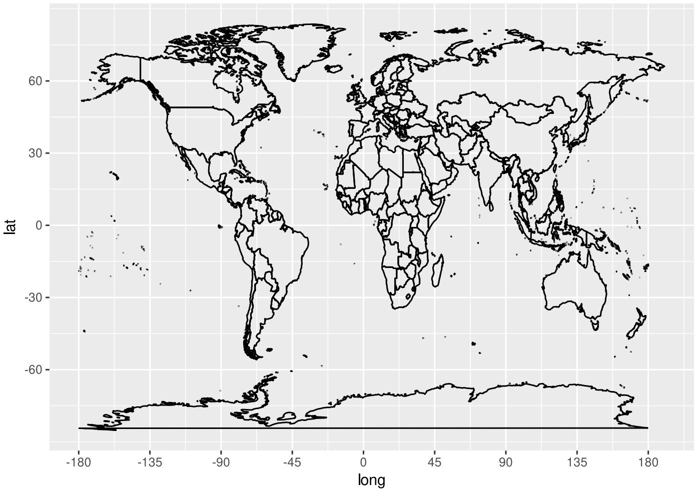




Coordinates: `coord_map()`

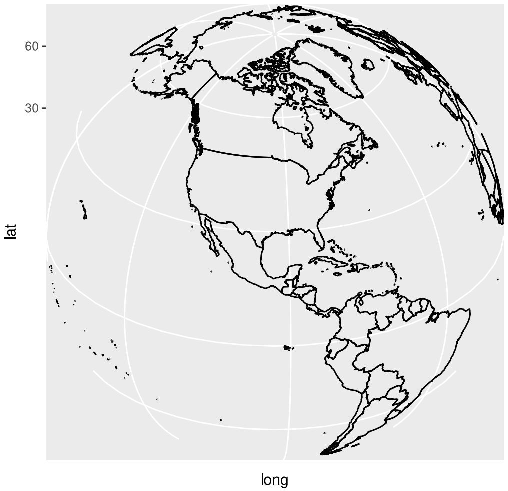
Cartesian System

```
ggplot(data = map_data("world"),
       mapping = aes(x = long,
                     y = lat,
                     group = group)) +
  geom_path() +
  scale_y_continuous(breaks = (-2:2) * 30)
  scale_x_continuous(breaks = (-4:4) * 45)
  #
  #
```



Azimuthal (orthographic) Projection

```
ggplot(data = map_data("world"),
       mapping = aes(x = long,
                     y = lat,
                     group = group)) +
  geom_path() +
  scale_y_continuous(breaks = (-2:2) * 30)
  scale_x_continuous(breaks = (-4:4) * 45)
  coord_map(projection = "ortho",
            orientation = c(30, -94, 0))
```

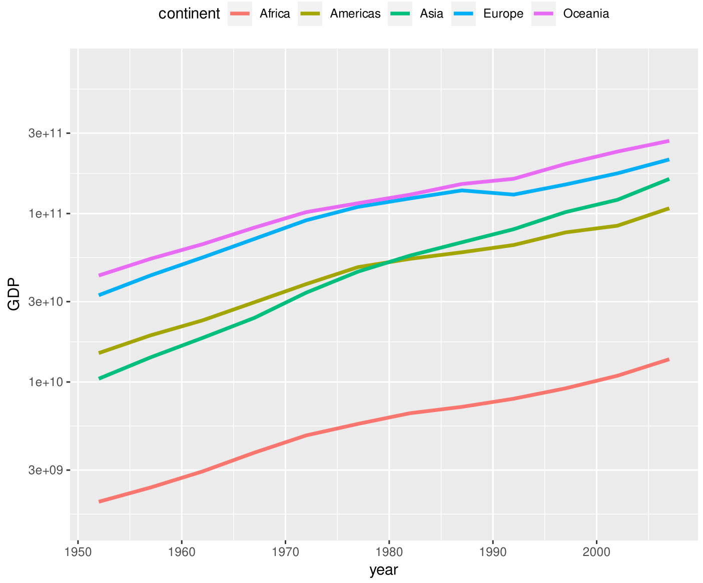


Scales vs. Coordinates

Scales

1. Transform data
2. Estimate statistic

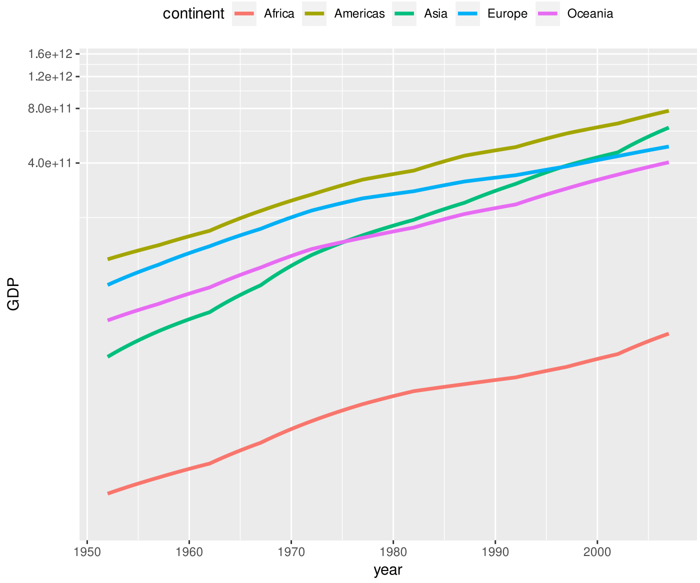
```
# log10(GDP) %>% mean()
ggplot(...) +
  scale_y_log10(name = "GDP")
```



Coordinates

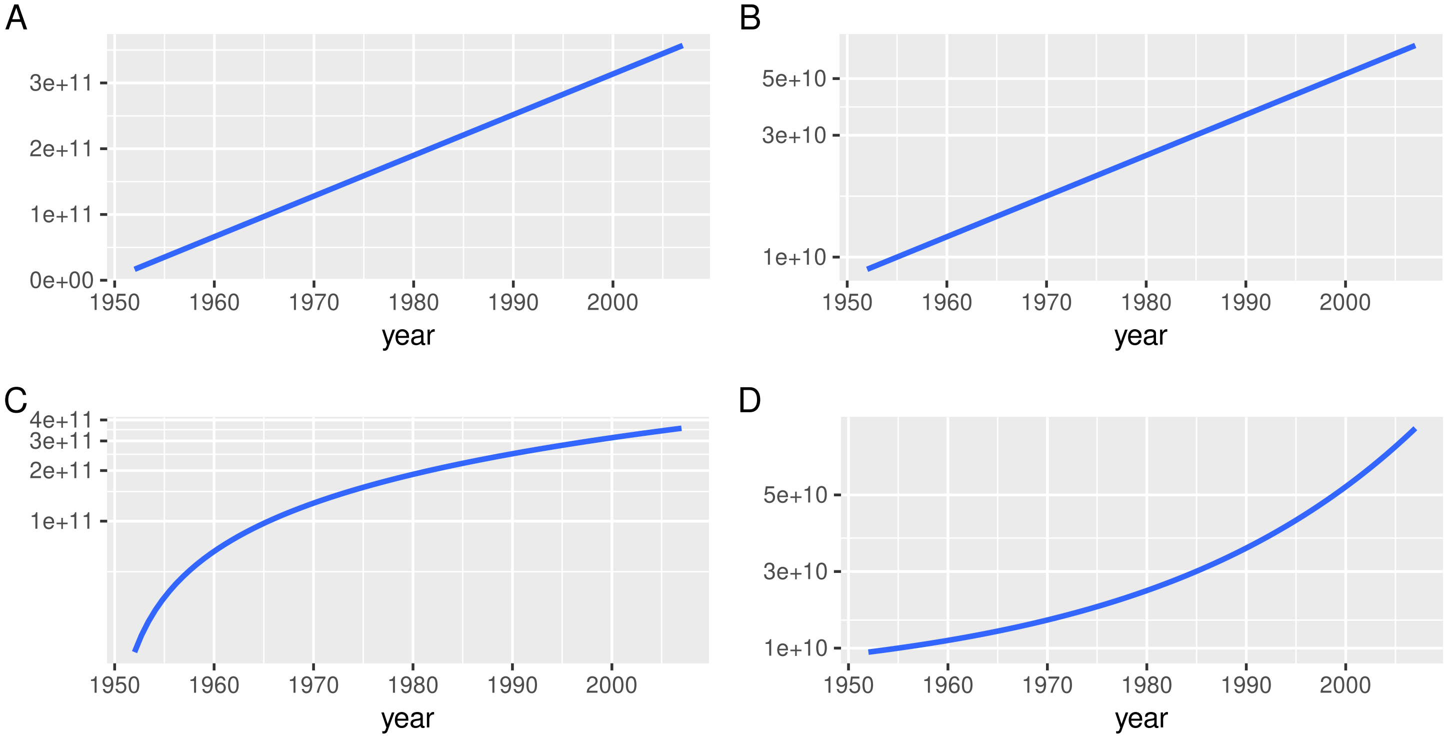
1. Estimate statistic
2. Transform data

```
# mean(GDP) %>% log10
ggplot(...) +
  coord_trans(y = "log10")
```



Scales vs. Coordinates

You can use both **scales** and **coordinates** together to estimate your model on the transformed data and transform is back for interpretation



(A) Linear model on original data - does not fit well; **(B)** linear model on log10 transformed GDP - fits well; **(C)** Linear model on original data, then log10 transform y-axis (not in original scale); **(D)** log10 transform GDP, estimate model, backtransform axes to get original scale

Theme

The **theme** encompasses every part of the graphic that is not part of the data (i.e., has no mapping to the data)

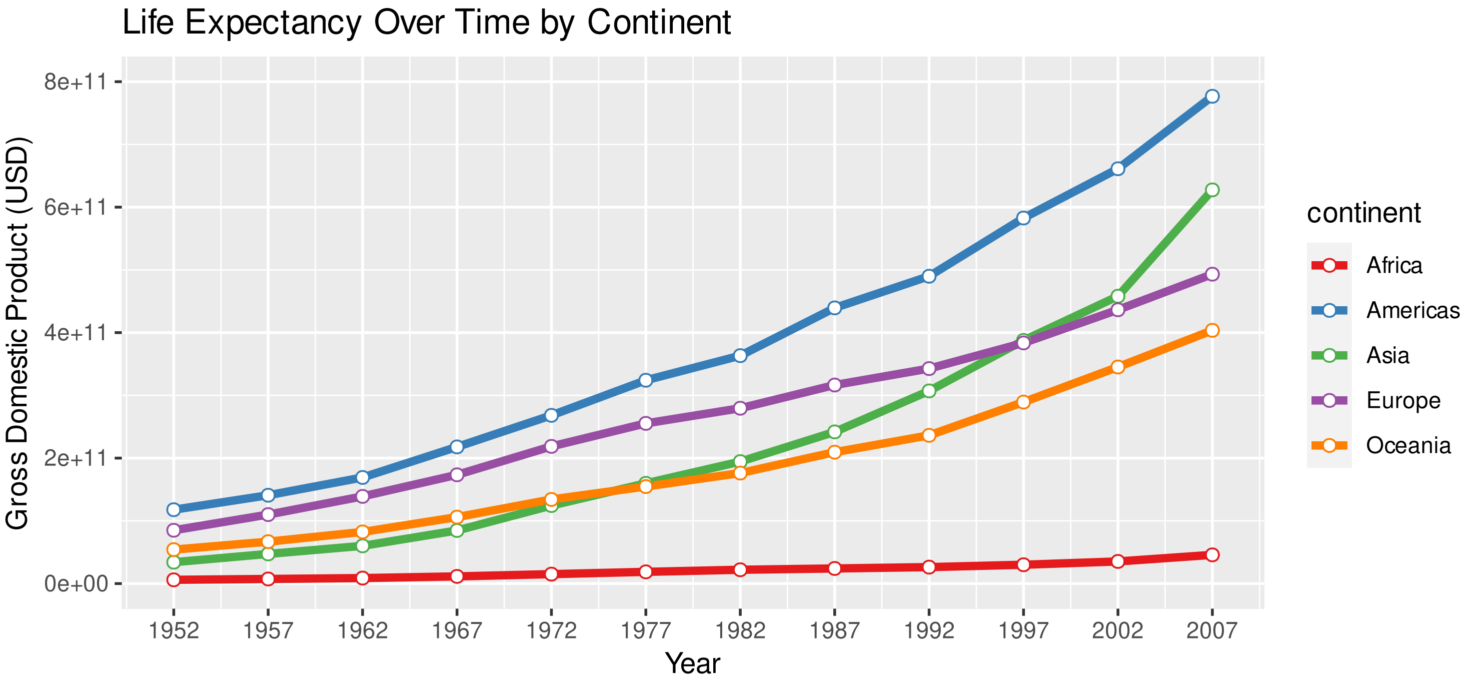
There are several pre-made themes that come with ggplot2:

- `theme_grey()` 👈 default
- `theme_bw()`
- `theme_linedraw()`
- `theme_light()`
- `theme_dark()`
- `theme_minimal()`
- `theme_classic()`
- `theme_void()`
- `theme_test()`

Theme

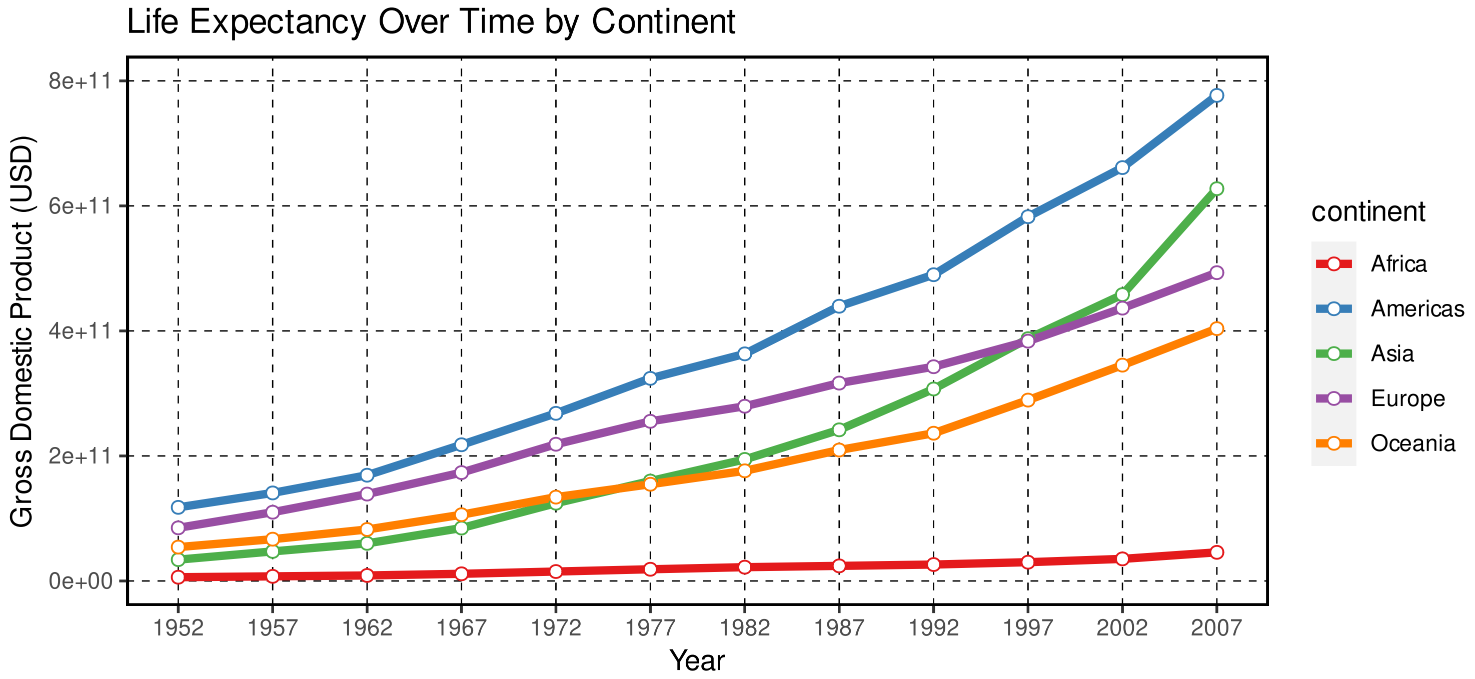
To tweak other aspects of your plots them you can use the `theme()` function, which has 94 arguments to give you complete control over all elements of your plot

To demonstrate, we'll use the following base plot from Slide 32:



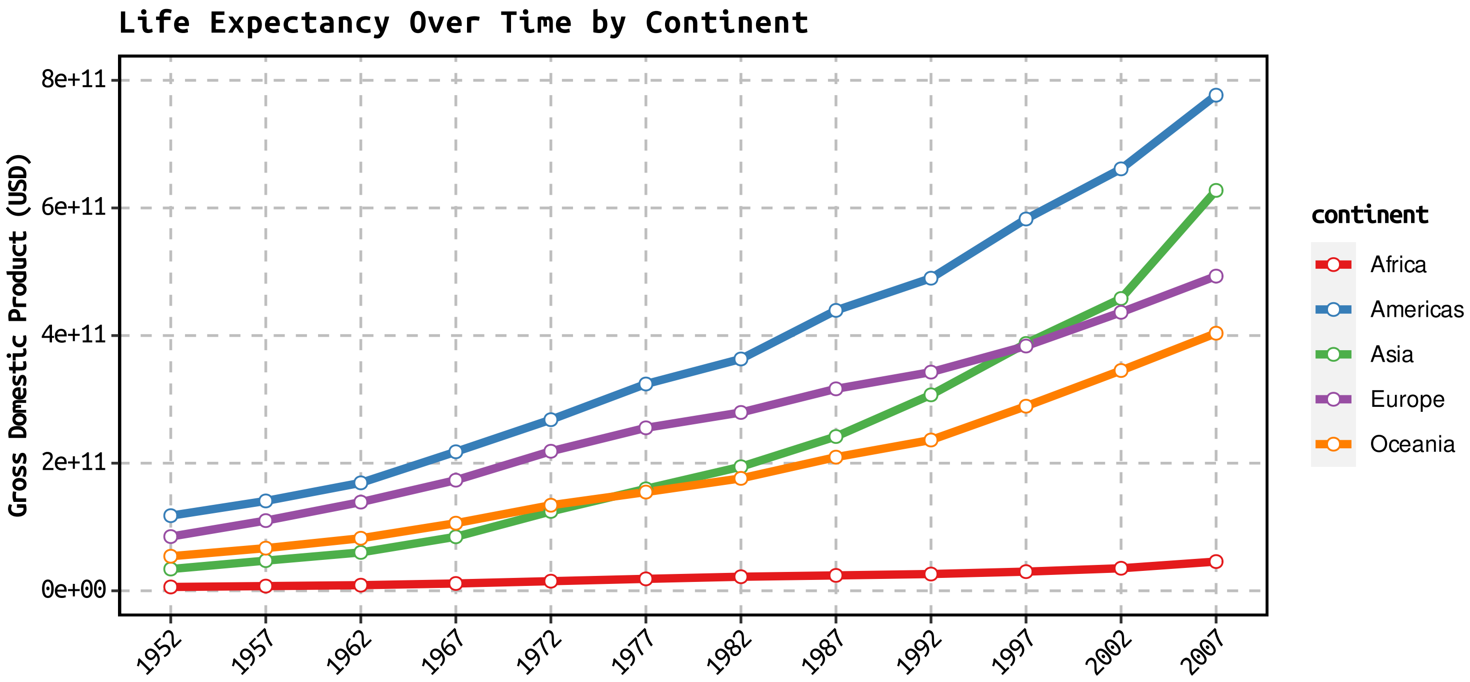
Theme: `panel`

```
p +
  theme(panel.grid.major = element_line(color = "black", linetype = 2, size = 0.25),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))
```



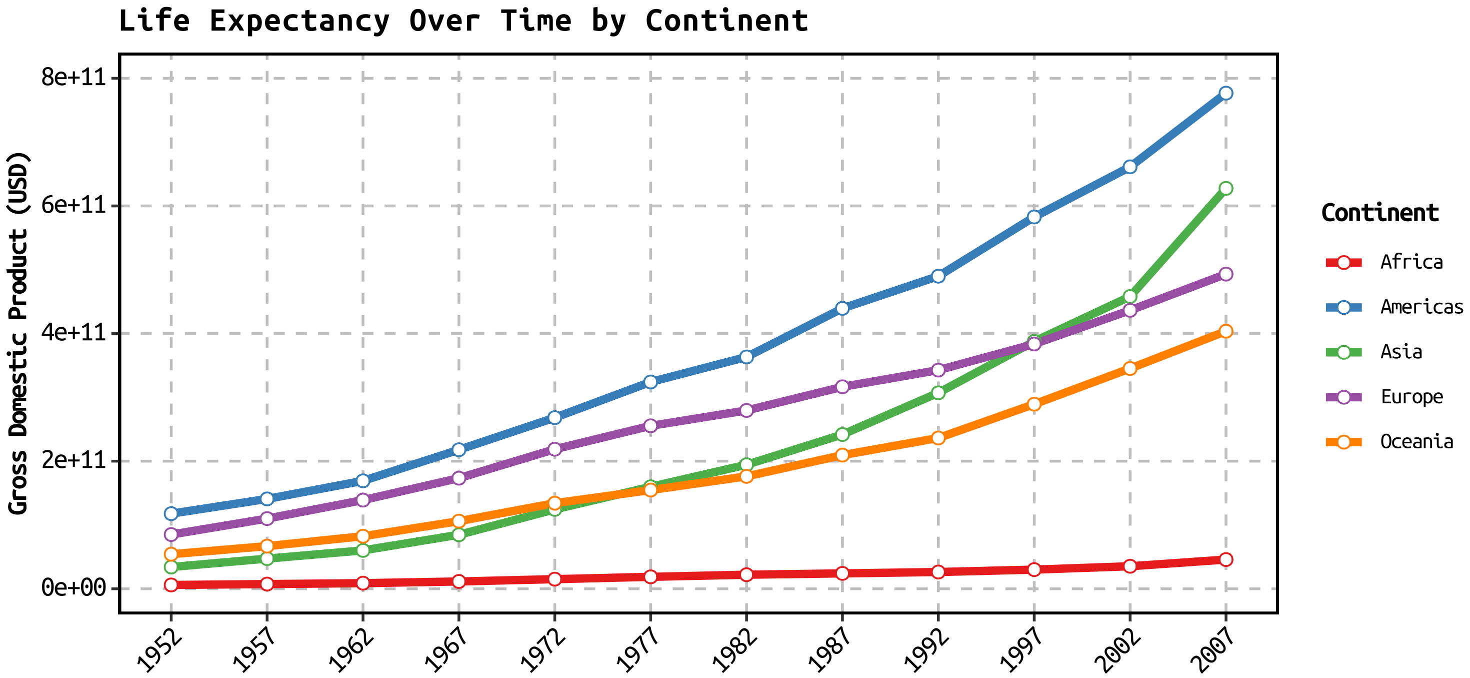
Theme: `axes`

```
p +
  theme(title = element_text(family = "Ubuntu Mono", face = "bold"),
        axis.title.y = element_text(family = "Ubuntu Mono"),
        axis.title.x = element_blank(),
        axis.text = element_text(family = "Ubuntu Mono", color = "black", size = 11),
        axis.text.x = element_text(angle = 45, hjust = 1))
```



Theme: `legend`

```
p +
  theme(legend.key = element_blank(),
        legend.text = element_text(family = "Ubuntu Mono")) +
  guides(color = guide_legend(title = "Continent"))
```

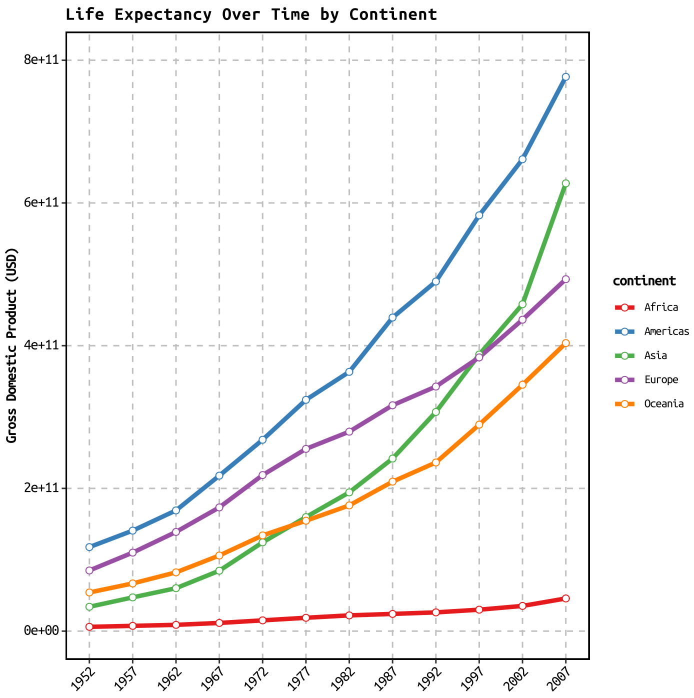


Theme

Just like `ggplot2` comes with pre-made themes, you can create your own themes to use repeatedly throughout your data visualizations

```r
theme_monotype <- theme(
  # Grid theme
  panel.grid.major = element_line(color = "
  panel.grid.minor = element_blank(),
  panel.background = element_rect(fill = "w
  panel.border = element_rect(color = "blac

  # Axis theme
  title = element_text(family = "Ubuntu Mon
  axis.title.y = element_text(family = "Ubu
  axis.title.x = element_blank(),
  axis.text = element_text(family = "Ubuntu
  axis.text.x = element_text(angle = 45, hj

  # Legend theme
  legend.key = element_blank(),
  legend.title = element_text(family = "Ubu
  legend.text = element_text(family = "Ubun
)
```

```r
p +
  theme_monotype
```

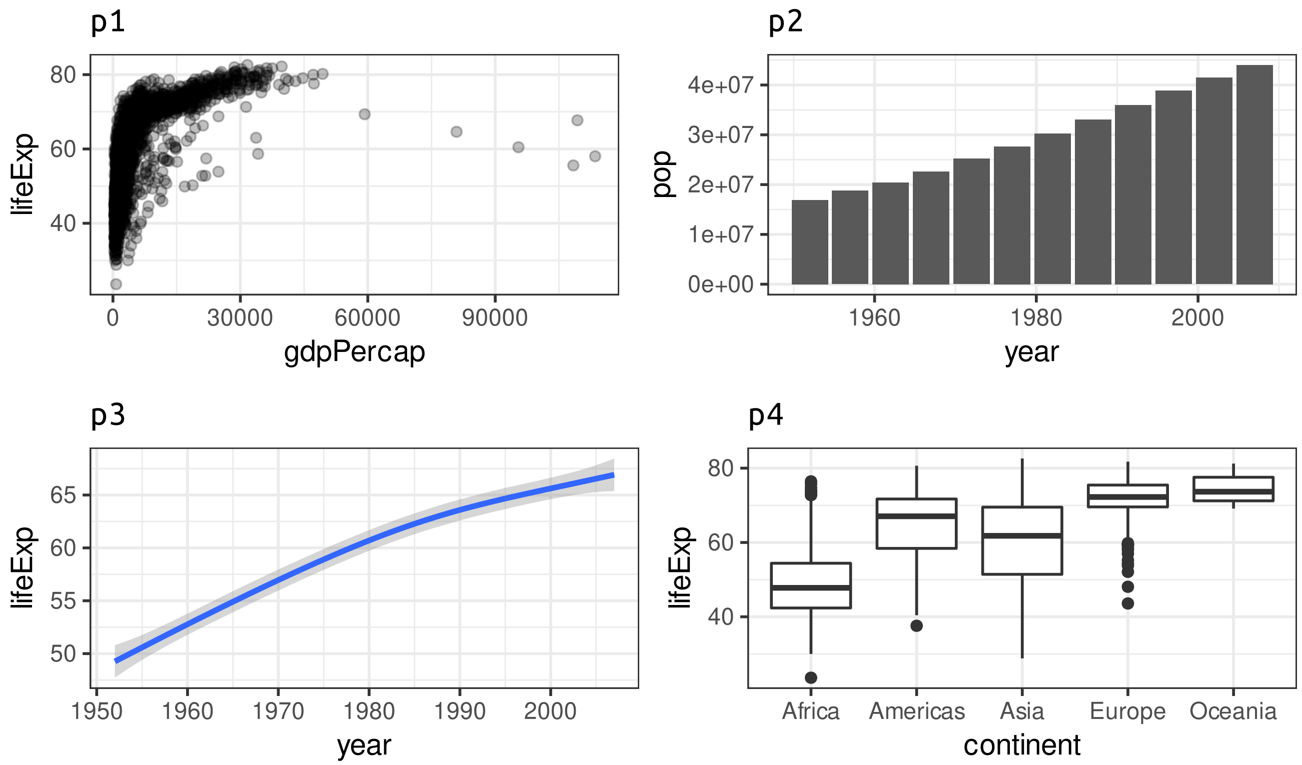


patchwork

What is patchwork?

Patchwork is a package created by Thomas Lin Pedersen (also the maintainer of `ggplot2`) to help you easily and flexibly combine several ggplots into the same graphic

Consider the following four plots (`p1`, `p2`, `p3`, `p4`):


p1

p2

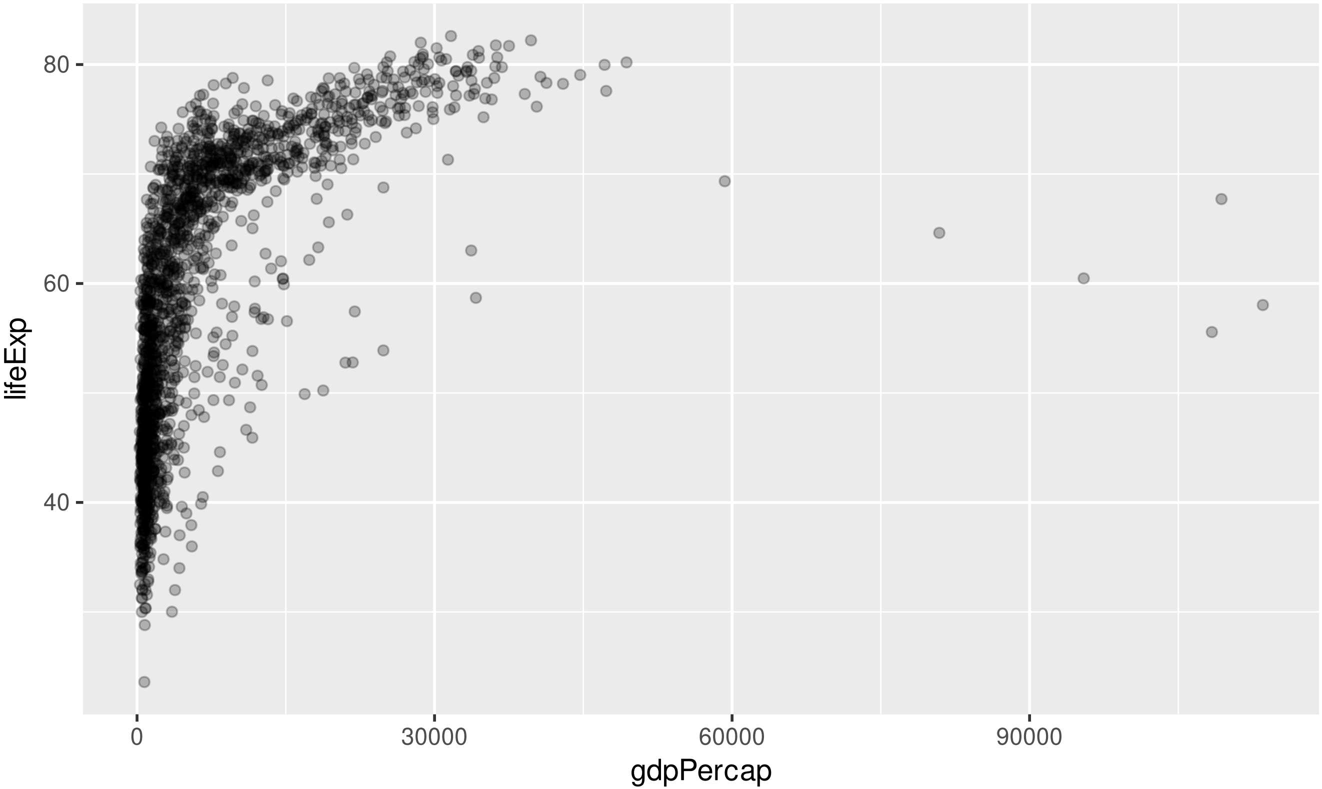
p3

p4

Patchwork arithmetic

Patchwork uses arithmetic (`+`, `-`, `*`, `/`) and logical (`|`, `&`) operators to control the layout of your figure

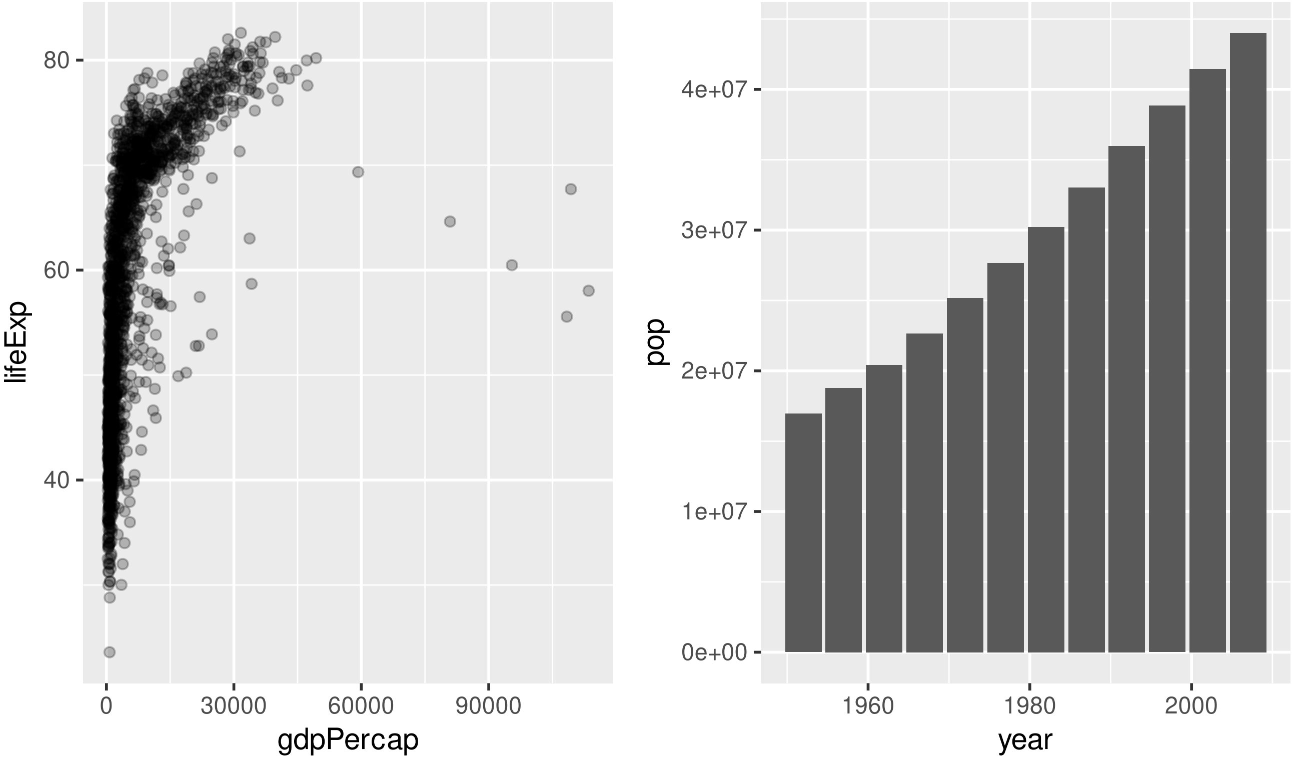
```
(p1)
```



Patchwork arithmetic

Add `p2`

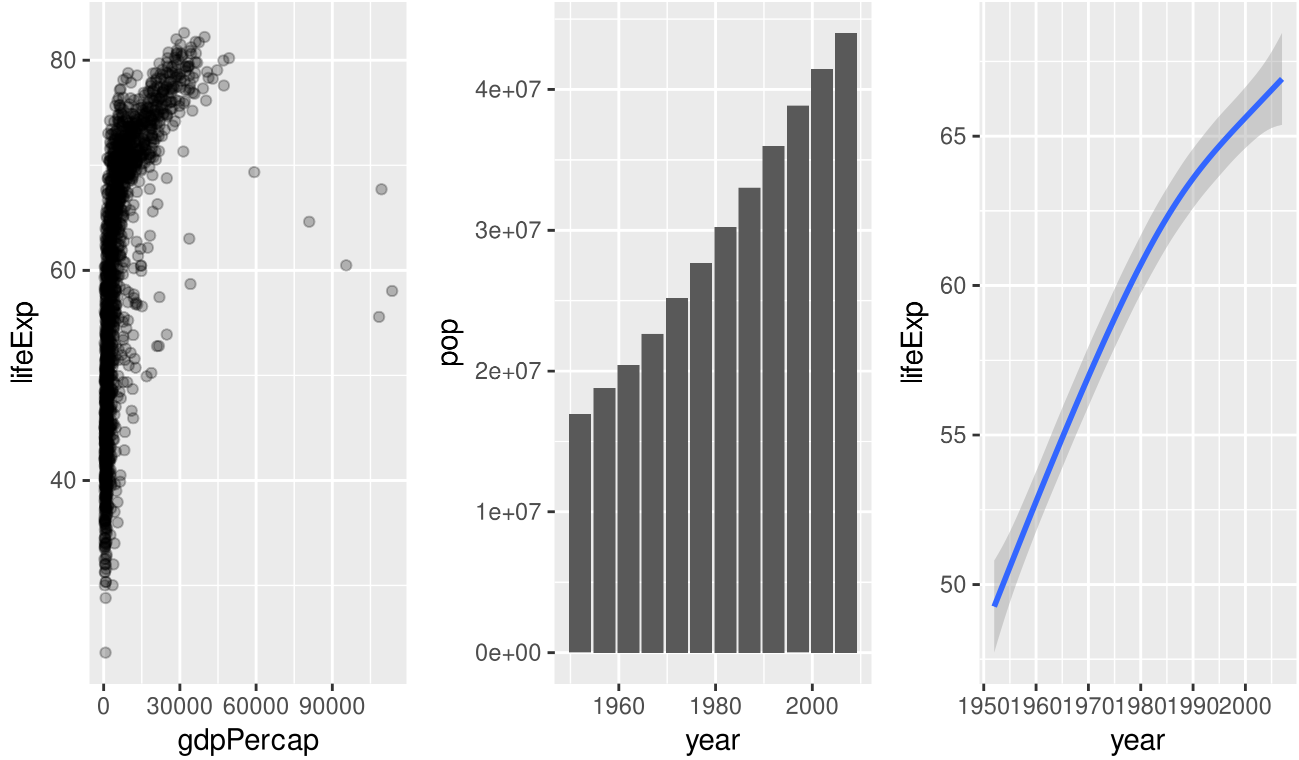
```
(p1 + p2)
```



Patchwork arithmetic

Add `p3`

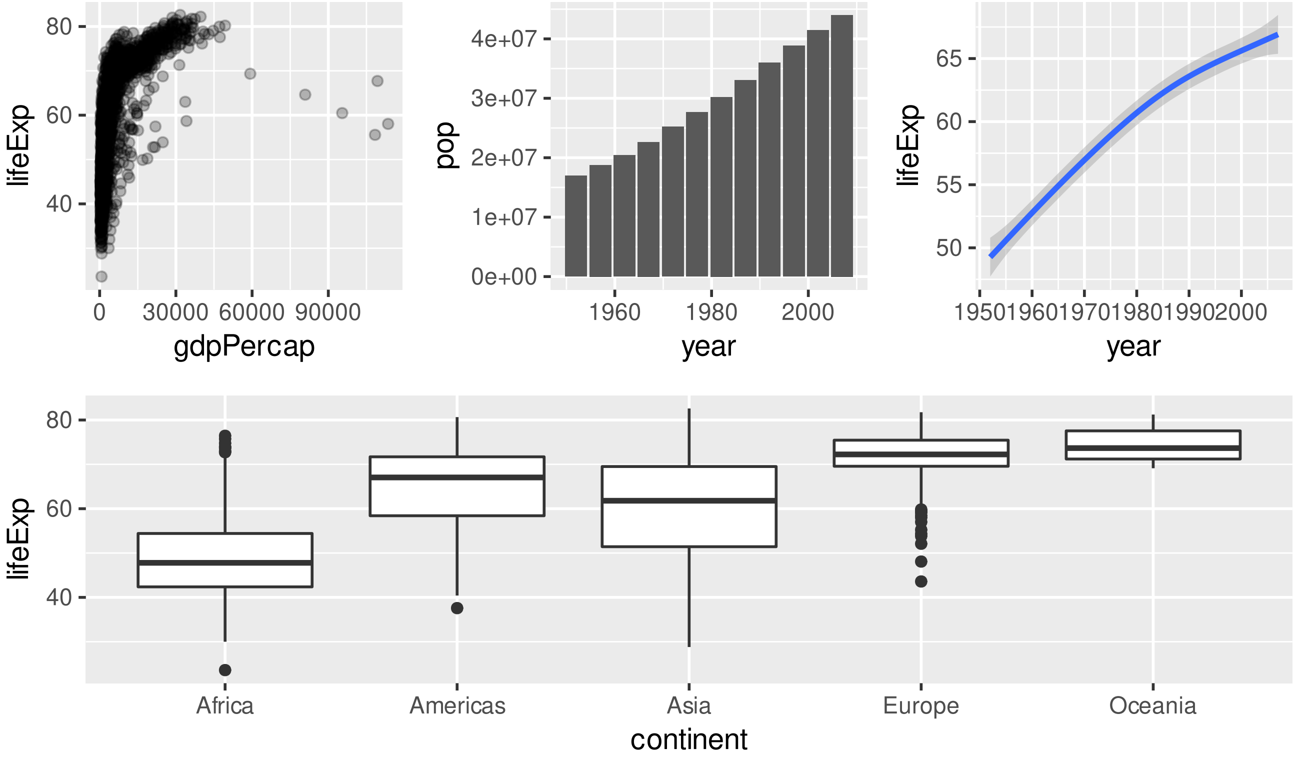
```
(p1 + p2 + p3)
```



Patchwork arithmetic

Add `p4`

```
(p1 + p2 + p3) / p4
```

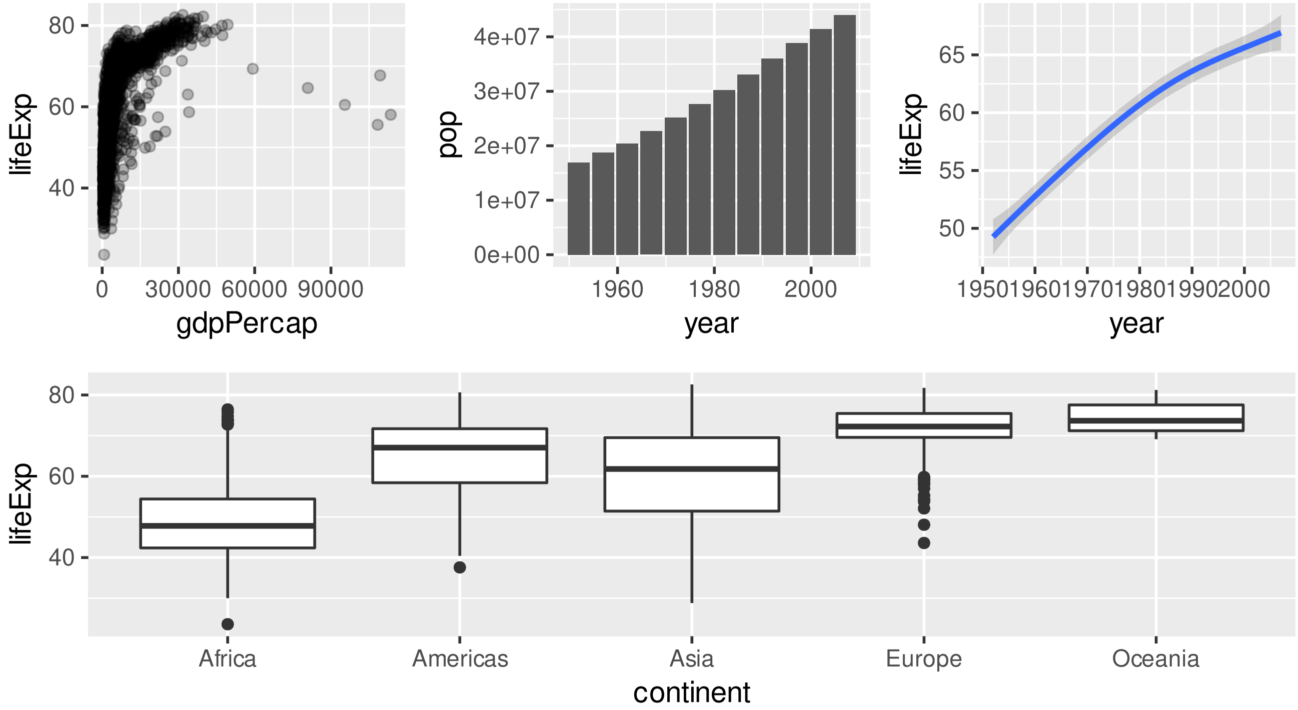


`plot_layout()`

You can also use `plot_layout()` to control the layout of your plots

The `-` sign ensures that all the patchwork on the LHS (`p1 + p2 + p3`) and on the RHS (`p4`) are grouped (i.e., nested) together

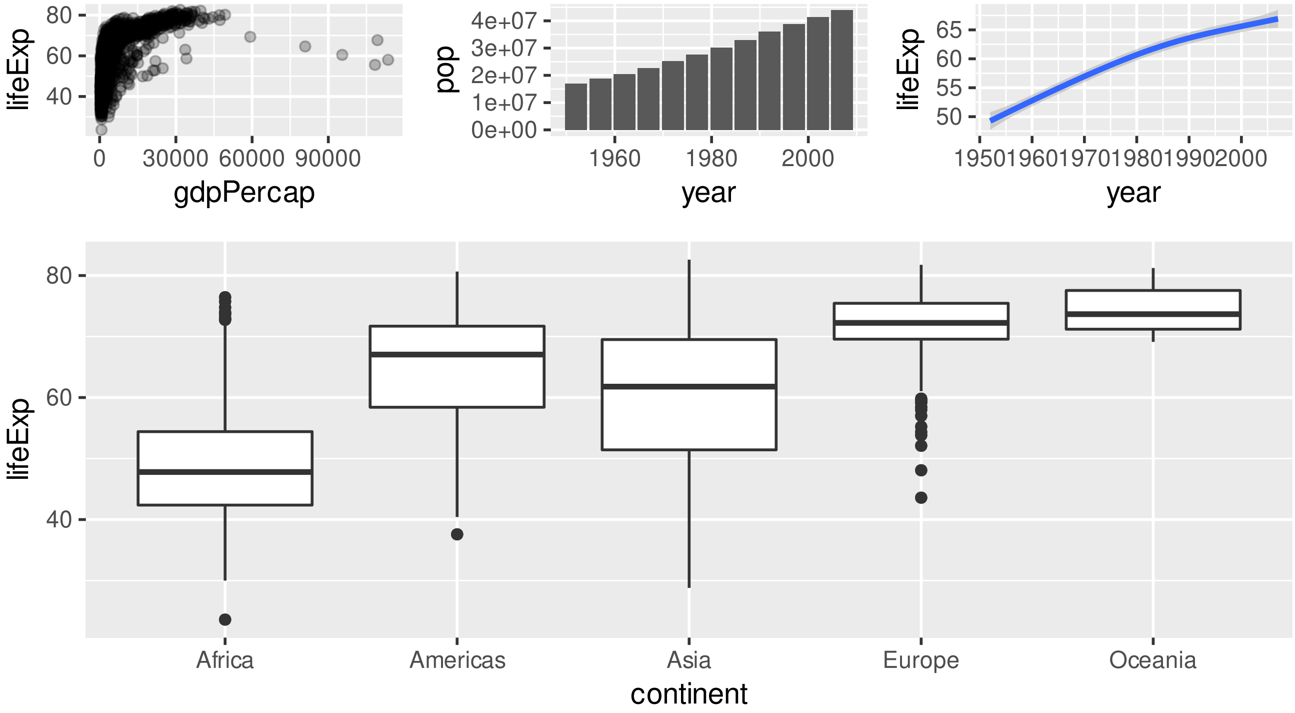
```
p1 + p2 + p3 - p4 + plot_layout(nrow = 2)
```



`plot_layout()`

Use `plot_layout()` to control the heights and widths of your patchwork elements

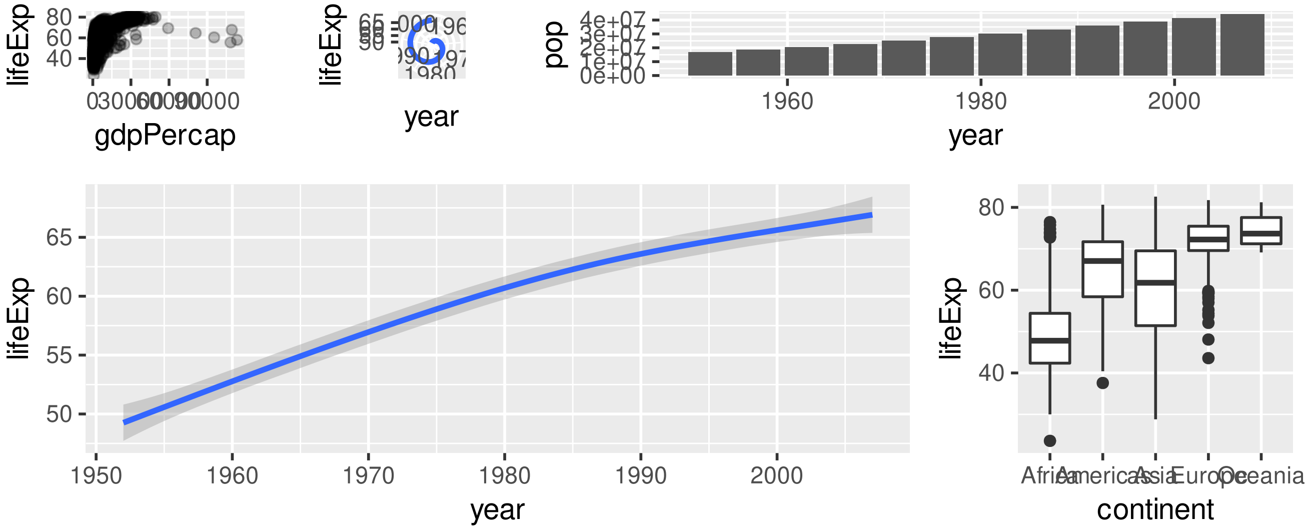
```
(p1 + p2 + p3) / p4 + plot_layout(heights = c(1, 2))
```



Combine multiple patchworks

To keep your code organized in a complex patchwork, you can save individual patchworks and combine them at the end

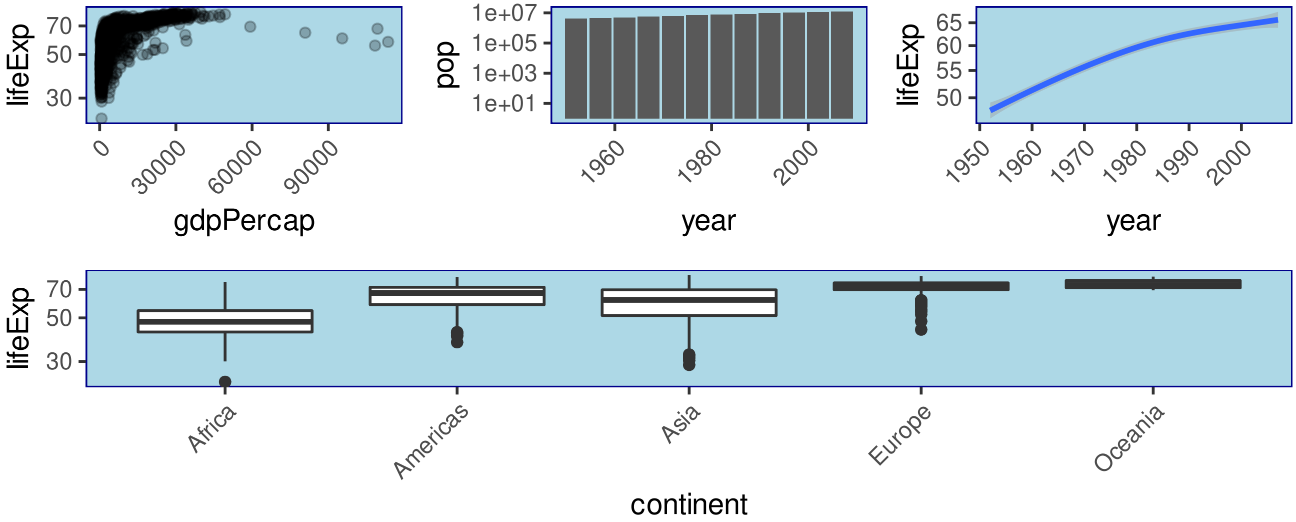
```
patchwork1 <- (p1 + p3 + coord_polar() + p2) + plot_layout(widths = c(1, 1, 4))
patchwork2 <- (p3 + p4) + plot_layout(widths = c(3, 1))

patchwork1 / patchwork2 + plot_layout(height = c(1, 4))
```



Add layers to all ggplots

With `patchwork`, you can control the layers (e.g., geoms, theme) of your ggplots all at once using the `&` operator. This is useful, for example, when you have several plots with the same theme or want to add a layer to every plot without adding this code to each and every plot

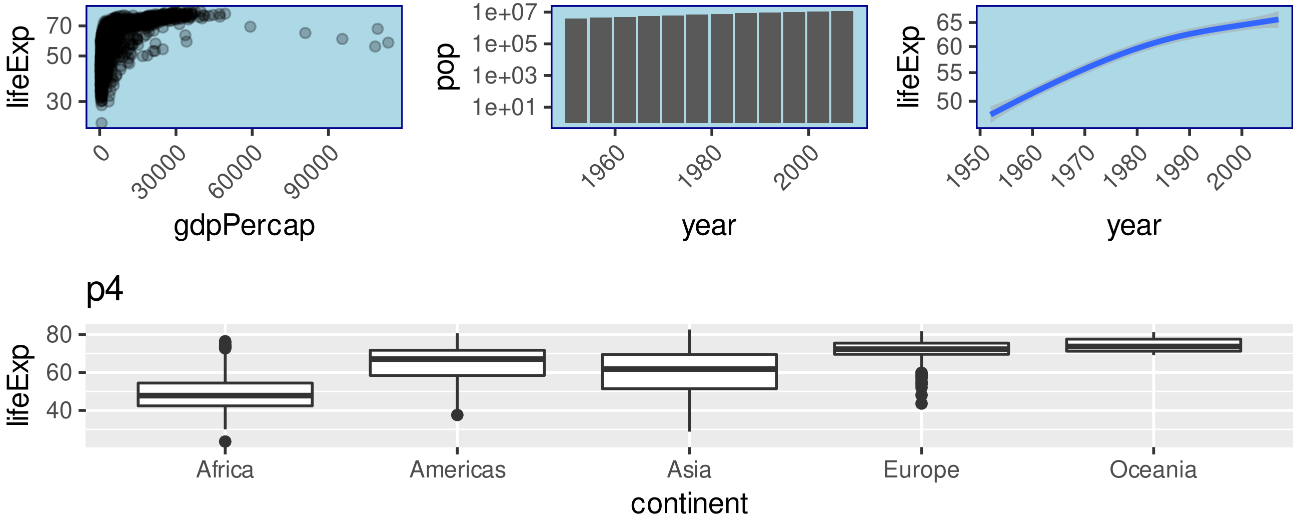
```
(p1 + p2 + p3) / p4 &
  scale_y_continuous(trans = "log10") &
  theme(plot.title = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1),
        panel.grid = element_blank(),
        panel.background = element_rect(color = "darkblue",
                                        fill = "lightblue"))
```



Add layers to some ggplots

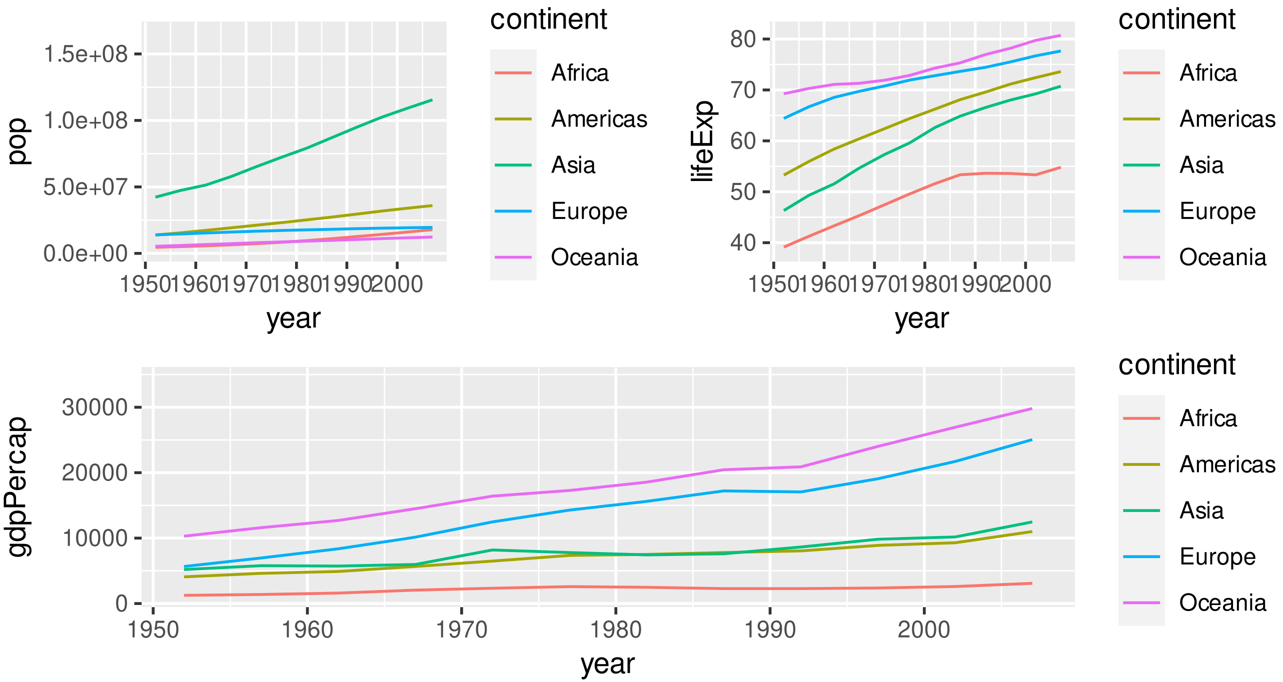
While the `&` operator adds layers to all ggplots in a patchwork, the `*` operator adds layers only to the current nesting level

```
(p1 + p2 + p3) *
  scale_y_continuous(trans = "log10") *
  theme(plot.title = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1),
        panel.grid = element_blank(),
        panel.background = element_rect(color = "darkblue",
                                        fill = "lightblue")) / p4
```



Controlling Legends

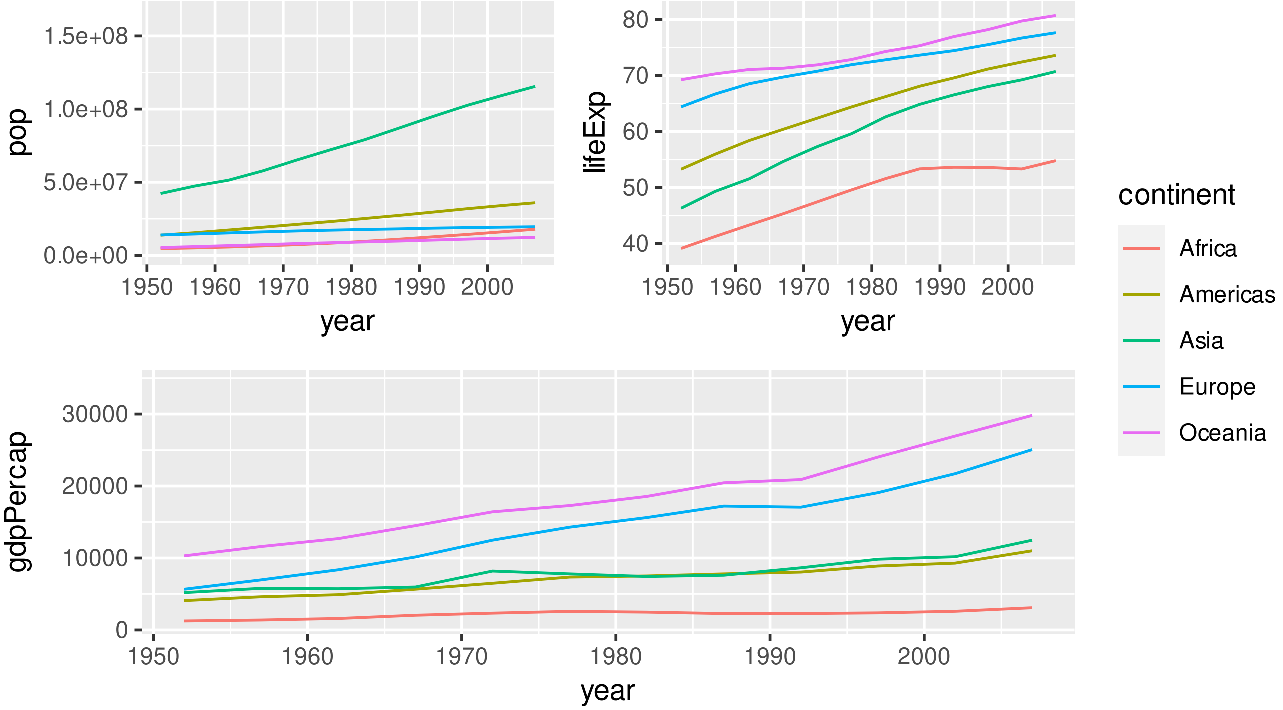
Imagine you have the following three plots configured using patchwork (`(p1 + p2) / p3`). The legends are identical and it is only necessary to have one of them for the entire figure.



Controlling Legends

Combine identical legends with `guides = "collect"`

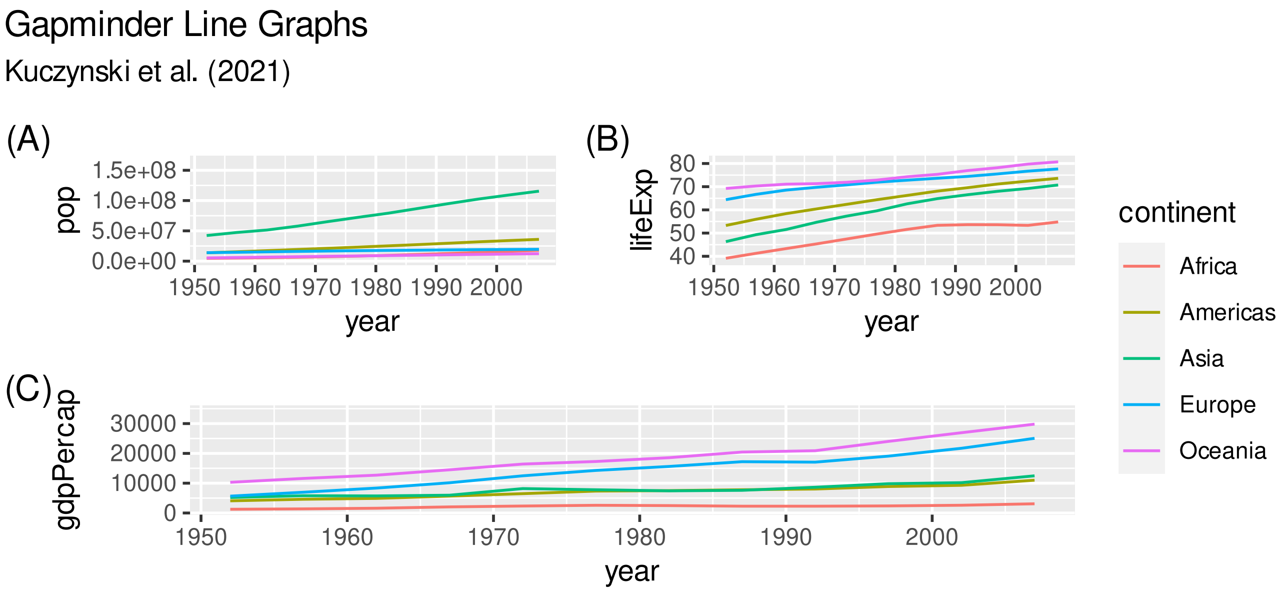
```
(p1 + p2) / p3 +
  plot_layout(guides = "collect")
```



Titles & Tags

When you have a multi-plot figure such as that in the previous slide, it is common to label each plot to refer to in your manuscript. To apply titles and tags to the entire plotting window, you can use `plot_annotation()`

```
(p1 + p2) / p3 +
  plot_layout(guides = "collect") +
  plot_annotation(title = "Gapminder Line Graphs",
                  subtitle = "Kuczynski et al. (2021)",
                  tag_levels = "A", tag_prefix = "(", tag_suffix = ")")
```


Gapminder Line Graphs
Kuczynski et al. (2021)
(A)
pop
1.5e+08
1.0e+08
5.0e+07
0.0e+00
year
(B)
lifeExp
year
(C)
gdpPercap
year
continent
Africa
Americas
Asia
Europe
Oceania