Layered grammar of graphics

Data visualization 2024/2025

Matteo Ceccarello

2025-01-20

> `grammar`:
> The principles or rules of an art, science, or technique
> – Merriam-Webster dictionary

This note focuses on describing the *Layered Grammar of Graphics* introduced by Wickham (2010), which forms the basis of the `ggplot2` library. At a more fundamental level, the *Layered Grammar of Graphics* allows us to *think* about graphics and plots in a principled way.

Wickham, Hadley. 2010. "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics* 19 (1): 3–28.

It is by no means the only effort to provide a formalism for data visualization. Two notable examples are the book by Wilkinson (2012) and the grammar for interactive graphics proposed by Satyanarayan et al. (2016).

Wilkinson, Leland. 2012. *The Grammar of Graphics*. Springer.

Satyanarayan, Arvind, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. "Vega-Lite: A Grammar of Interactive Graphics." *IEEE Transactions on Visualization and Computer Graphics* 23 (1): 341–50.

Most notably, the grammar of graphics:

- Allows to gain insight into complicated graphics produced by other people;
- Allows more flexibility and expressiveness when creating our graphics;
- Provides a consistent framework to think about graphics;
- Constrains what can be expressed in a plot with principled rules rather than by the API that is provided by a library designer.

The *Layered Grammar of Graphics* is comprised of the following eight fundamental components:

- data
- aesthetic mapping
- geometric objects
- scales
- statistical transformations
- position adjustments
- facet specification
- coordinate system

Data

This is the most fundamental element of the grammar: without data, there can be no data visualization, of course. In the following, we will assume that we are dealing with *tidy data*. Recall from the previous lecture that a table is tidy if:

- Each column is a *variable*
- Each row is an *observation*
- Each cell is a single *value*

Aesthetic mappings

In the context of the *Layered Grammar of Graphics*, an *aesthetic* is a visual property of the objects in the plot. For instance, the thickness of a line is an aesthetic, the color of a point is an aesthetic, and the length of a bar is one as well. In general, anything in the plot whose variation can be perceived by readers of the plot can be thought of as an aesthetic.

An *aesthetic mapping* associates data variables with aesthetics. For instance, associating country names to the position on the x axis and the gross domestic product to the position on the y is an example of an aesthetic mapping.

Geometric objects

In the previous paragraph, we mentioned several marks that can appear in a plot: lines, points, and bars. These are called *geometric objects* in the grammar of graphics.

In general, *geometric objects* are how a chart designer represents data.

Notably, an *aesthetic mappint* associates variables in the data with visual properties of *geometric objects*.

Scales

A scale controls the mapping from *data values* to *aesthetic values*. The simplest example is the scale on the axes of a plot. The ticks and labels provide a mapping between data values and the positions of dots in the plot, i.e. the aesthetic values of the point geometric objects.

Another example is a color scale, providing a mapping between data values (for instance countries) and color values.

The most important feature is that a scale should be *invertible*: we need to be able to go from data values to aesthetic values to *build* the plot, and we need to go from aesthetic values back to data values (with some approximation) in order to *read* the plot.

Statistical transformations

The role of a statistical transformation is to, well… transform the data. Typically by summarizing it. For instance, the computation of the mean (or median, or quantiles) is an example of statistical transformation. But so is binning, or computing a density estimation from the data.

Position adjustments

Sometimes the position of geometric objects in a plot, based on the aesthetic mapping and mediated by the scale, needs to be adjusted.

It might be for a design choice: in a stacked bar chart, several bars share the same x position and their y position must be adjusted so that they stand on top of each other.

It might be to reduce overplotting: often in plots using the point geometric object there are areas in which points are very dense. Adding a small amount of random noise to the position of the points allows to make the picture clearer without losing too much accuracy.

Facet specification

Faceting allows to create multiple plots with the same combination of aesthetic mapping, geometric objects, scales, position adjustments and statistical transformation. The only difference is the data they represent. In particular, each plot shows a subset of the data based on the values of a combination of variables: we can have for instance a subplot showing data for countries in different continents in the same dataset.

Coordinate system

The coordinate system maps the position of objects (as specified by the x and y aesthetic mappings, mediated by the respective scales) onto the plane of the plot.

Most commonly we use the *cartesian* coordinate system, but other coordinate systems are possible, for instance the *polar* coordinate system.

Layering

One of the defining features of the Layered Grammar of Graphics is to be *layered*. In particular, a layer is defined as the combination of:

- *data*
- *aesthetic mapping*
- *geometric object*
- *statistical transformation*
- *position adjustments*

Layers can be stacked on top of one another, with lower layers being (partially) hidden by upper layers. This allows the creation of rather complex graphics by overlaying several simple layers.

Observe what is missing from the layer specification:

- *scales*
- *faceting*
- *coordinate system*

These components work *across* layers to make the plot readable. As such they should be shared by all layers. The implication is that some graphics that are commonly seen *cannot be expressed* in the grammar of graphics.

For instance, consider Figure Figure 1 which shows the correlation between the number of grand slam finals played by Roger Federer and the number of electronic engineers in New Mexico.

Of course, there is no relationship between the two, but using two different scales for the two different layers (one for the tennis player, and the other for the New Mexico engineers) misleads readers into seeing an association.

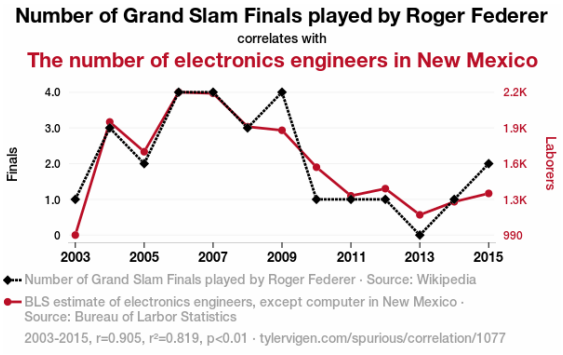


Figure 1: Image from http://www.tylervigen.com/spurious-correlations

The `ggplot2` implementation of the Layered Grammar of Graphics

There is a close correspondence between the functions provided by `ggplot` and the elements of the grammar.

In particular, a plot specification in `ggplot` looks like the following.

```
ggplot() +
  <GEOM_FUNCTION>(
    data = <DATA>,
    mapping = aes(<MAPPINGS>),
    stat = <STAT>,
    position = <POSITION>
  ) +
  <SCALE_FUNCTIONS> +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

The first line, `ggplot()`, defines the entry point and creates an empty graph to which several layers can then be added. The functions used to build this specification follow a strict naming convention:

- the `aes` function is to specify aesthetic mappings
- `geom_*` functions are for geometric objects
- `stat_*` functions are for statistical transformations
- `scale_*` functions are for scales
- `position_*` functions are for position adjustments
- `facet_*` functions are for faceting
- `coord_*` functions are for the coordinate system

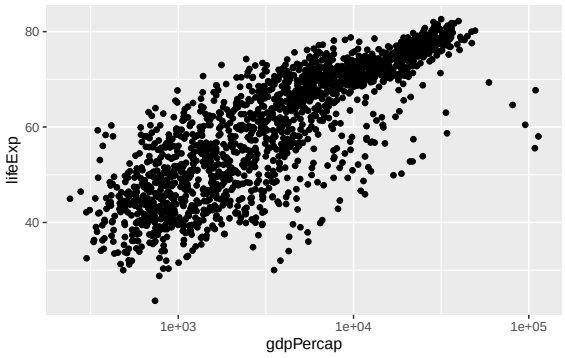
To define a layer consisting of the point geometric object with the identity statistical transformation and position adjustment we can use the following code:

```
ggplot() +
  layer(
    data = gapminder,
    mapping = aes(x=gdpPercap, y=lifeExp),
    geom = 'point',
    stat = 'identity',
    position = 'identity'
  ) +
  scale_x_log10()
```

In the following examples we use data from the `gapminder` package, which you can install with `install.packages("gapminder")`

Multiple layers can be added to show different subsets of the data, for instance to show different years:

```
ggplot() +
  layer(
    data = filter(gapminder, year == 1952),
    mapping = aes(x=gdpPercap, y=lifeExp,
                  color=factor(year)),
    geom = 'point',
    stat = 'identity',
    position = 'identity'
  ) +
  layer(
    data = filter(gapminder, year == 2007),
    mapping = aes(x=gdpPercap, y=lifeExp,
                  color=factor(year)),
    geom = 'point',
    stat = 'identity',
    position = 'identity'
  ) +
  scale_x_log10()
```



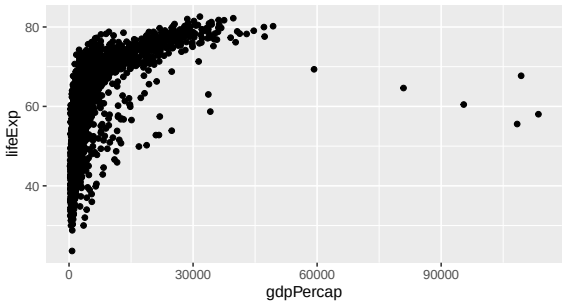
Default values

Oftentimes data and aesthetic mapping are shared across all layers. In such cases, we can provide the default in the `ggplot` function. Furthermore, each geometric object has a default statistical transformation, and each statistical transformation has a default geometric object. The specifics can be found in the documentation of each object.

Therefore, we can write a scatterplot as follows:

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

In the following we will showcase a few of the features of the ggplot implementation of the grammar of graphics by conducting a few case studies on the `gapminder` dataset.



Case study: Gapminder

As a first step, we will build a *bubble chart* showing the relation between the GDP of a country and the life expectancy in 2007.

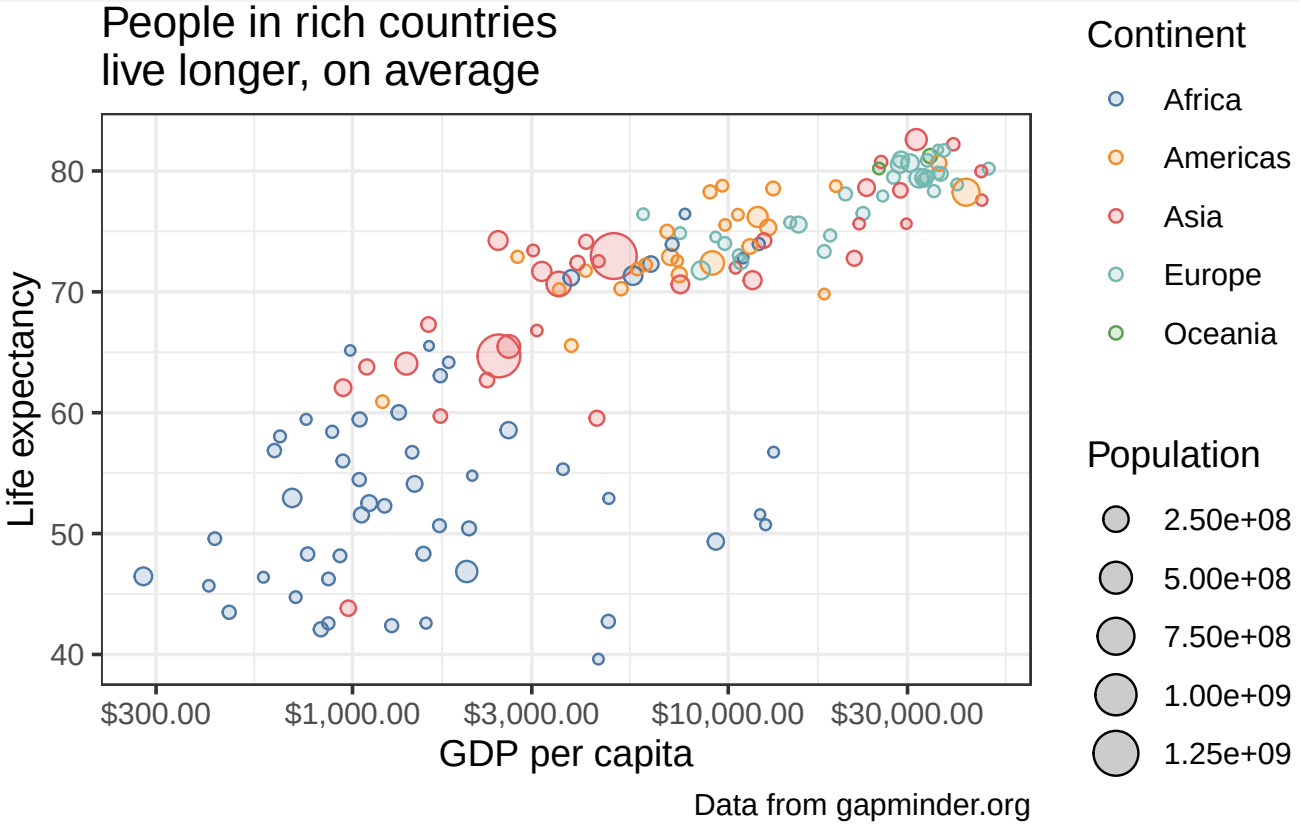
Key points:

- Note how using several aesthetic variables allows to encode quite some information
- Using two layers of points allows to create points that are semi-transparent while having a solid border

Hints:

- Use transparency to show overlapping points
- A log scale helps to spread out values that are very skewed
- The `shape` aesthetic is configured through numeric values. The documentation lists all of them.

```
library(tidyverse)
library(gapminder)
library(ggthemes) # For the scale_color_tableau function

gapminder |>
  filter(year == 2007) |>
  ggplot(aes(gdpPercap, lifeExp, size=pop, color=continent)) +
  geom_point(alpha=0.2) +
  geom_point(shape=21) +
  scale_color_tableau() +
  scale_x_log10(labels=scales::label_dollar()) +
  labs(
    title="People in rich countries\nlive longer, on average",
    y = "Life expectancy", x = "GDP per capita",
    size = "Population", color = "Continent",
    caption = "Data from gapminder.org"
  ) +
  theme_bw()
```



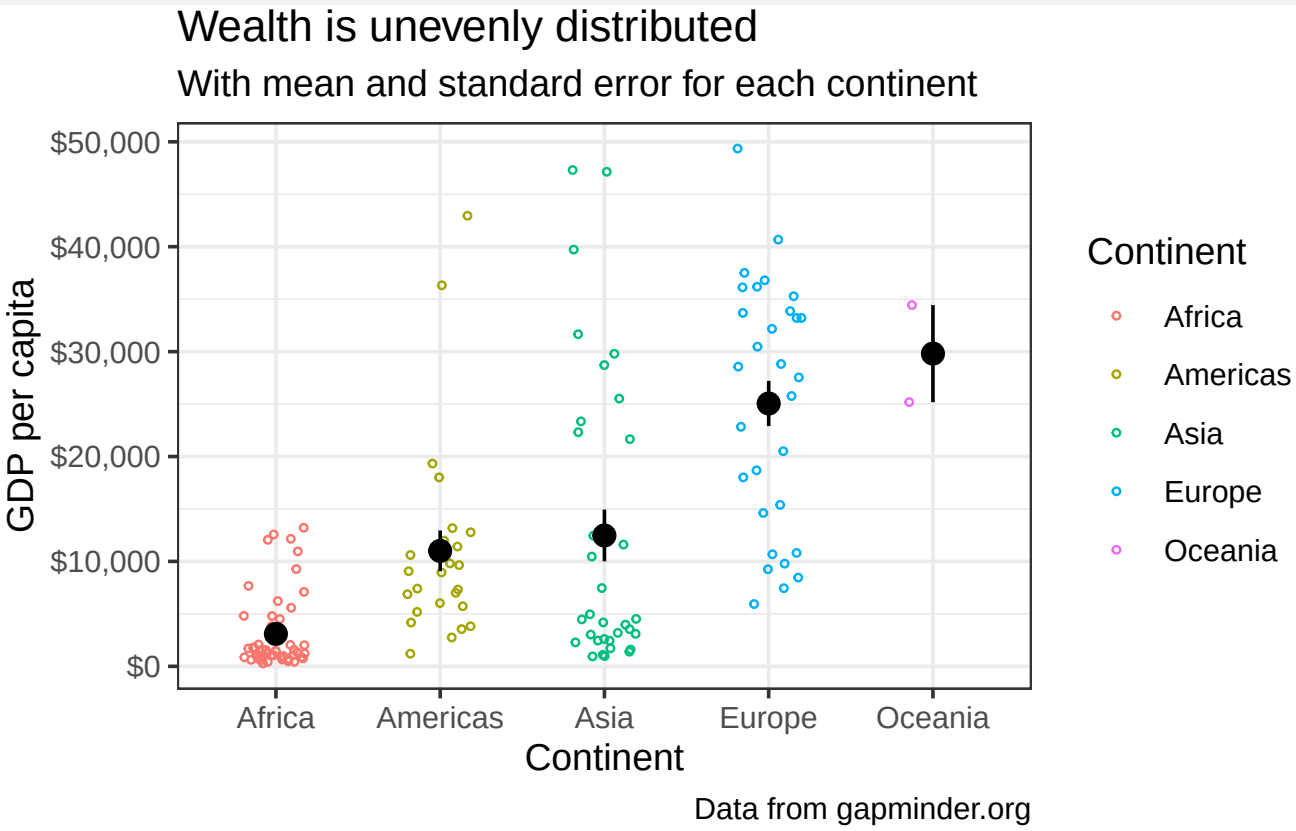
Case study: statistical summaries and position adjustments

This second case study focuses on the use of statistical summaries. In particular, the `stat_summary` function accepts the `fun.data` argument which should be a function returning three values, which will then be used as the top, bottom, and middle point of the `pointrange` geometric object, which by default is used to represent summarizations. The geometric object can be tweaked by changing the `geom` argument: using `point` will use just the middle value computed by the `fun.data` function, `linerange` will just use the two extremes.

Hints

- use `position_jitter` to reduce overplotting
- transofmrations can be done either in the preprocessing phase or in the grammar itself

```
library(tidyverse)
library(gapminder)

gapminder |>
  filter(year==2007) |>
  ggplot(aes(x=continent, y=gdpPercap)) +
  geom_point(
    aes(color=continent),
    size=0.7, shape=21,
    position=position_jitter(width=0.2)
  ) +
  stat_summary(
    fun.data=mean_se,
    geom = "pointrange"
  ) +
  scale_y_continuous(labels=scales::label_dollar()) +
  labs(
    title="Wealth is unevenly distributed",
    subtitle="With mean and standard error for each continent",
    x = "Continent", y = "GDP per capita",
    color = "Continent", caption = "Data from gapminder.org"
  ) +
  theme_bw()
```



Case study: secondary axis

Sometimes we need to use two different y scales in the same plot. While this is not advisable in general, in some scientific fields it is a conventional representation. In `ggplot` this is forbidden by the grammar with one exception: when the secondary axis is a linear transformation of the primary axis.

We can abuse this mechanism to produce a plot with two

different y scales. The basic idea is that, upon plotting, the data referring to the secondary axis will be rescaled to the primary axis range. When building the secondary axis, the range of the primary axis will be rescaled to the range of the secondary axis in order to get the correct labels.

In this example we will build a line plot for Italy with a line referring to the GDP (primary axis) and a second line referring to the life expectancy (secondary axis).

The first thing to do is to get the ranges of the two variables[1].

[1] The `pull` function transforms a column of a table into a vector.

```
library(tidyverse)
library(gapminder)

italy <- gapminder |> filter(country == "Italy")

# Pick the range of the variables thare are going to be plotted along the y axis
range_primary <- range(pull(italy, gdpPercap))
range_sec <- range(pull(italy, lifeExp))
```

Then we start building the plot. We will use different colors to allow the reader to infer the scale associated to each line. First, we create a line with the data for the primary axis, as usual.

```
# This is the color that we are going to assign to the secondary axis
color_sec <- "#b62900"
p <- ggplot(italy, mapping=aes(x=year)) +
  # First layer: primary axis
  geom_line(
    mapping = aes(y = gdpPercap),
    linetype = "dashed"
  )
```

Then we add the second layer. Note that in the aesthetic mapping we rescale the `lifeExp` column so that it fits the range of the primary axis.

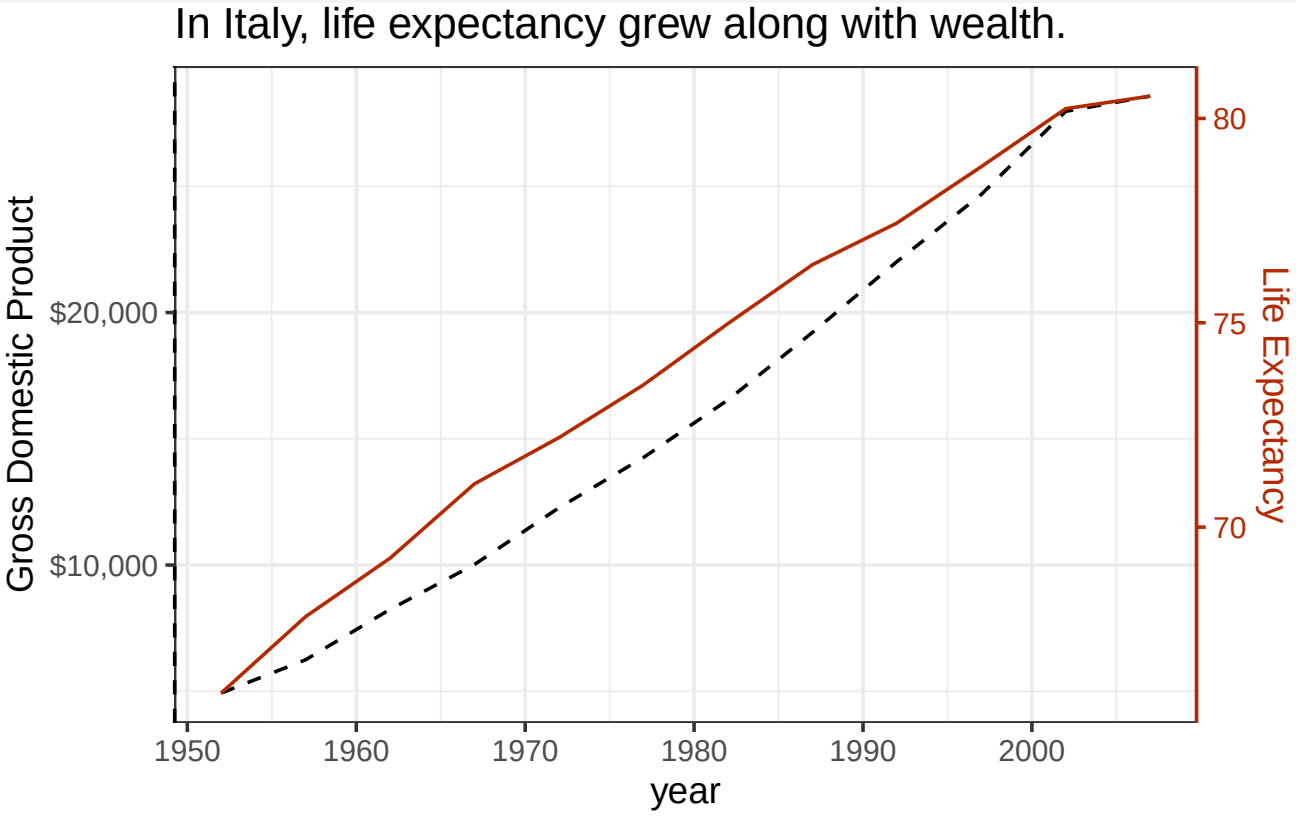
```
# Second layer. We have to rescale the data to fit the
# range of the primary axis.
p <- p + geom_line(
  mapping = aes(y = scales::rescale(lifeExp, from=range_sec, to=range_primary)),
  color = color_sec
)
```

Now that we have both lines on the same plot, we need to add the secondary axis. We do it using the `sec.axis` argument of `scale_y_continuous`, which requires a function transforming ticks on the primary scale to ticks on the secondary scale. In our case this function will rescale from the primary range to the secondary range.

```
# Here we fix both scales.
p <- p + scale_y_continuous(
  # The primary scale is for the GDP
  name = "Gross Domestic Product",
  labels = scales::label_dollar(),
  # For the secondary scale we need to scale back from the primary
  # range to the secondary range.
  sec.axis = sec_axis(
    function(ys) { scales::rescale(ys, from=range_primary, to=range_sec) },
    name = "Life Expectancy"
  )
)
```

Finally, we make some fundamental cosmetic changes using the `theme` function: both axes must use the same color and linetype as the respective lines. This helps readers associate lines to the correct scale.

```
p + labs(title = "In Italy, life expectancy grew along with wealth.") +
  theme_bw() +
  theme(
    # Tweaking the theme a bit is fundamental here: we have to match
    # the appearance of the axes with the color and dashing of the lines.
    axis.line.y.left = element_line(linetype = "dashed"),
    axis.line.y.right = element_line(color = color_sec),
    axis.ticks.y.right = element_line(color = color_sec),
    axis.text.y.right = element_text(color = color_sec),
    axis.title.y.right = element_text(color = color_sec)
  )
```



Case study: faceting and line charts

Line charts require special care when we want to plot several lines with the same color. In the following example, where we plot the change of life expectancy throughout the years, we have to use the `group` aesthetic to make sure that there is a single group per country.

We use `facet_wrap` to get a subplot for each continent in the dataset. To make the plot a bit more readable and highlight some patterns, we make all the country lines light gray, except for countries that at some point saw a decline in life expectancy.

To this end, we first identify these countries.

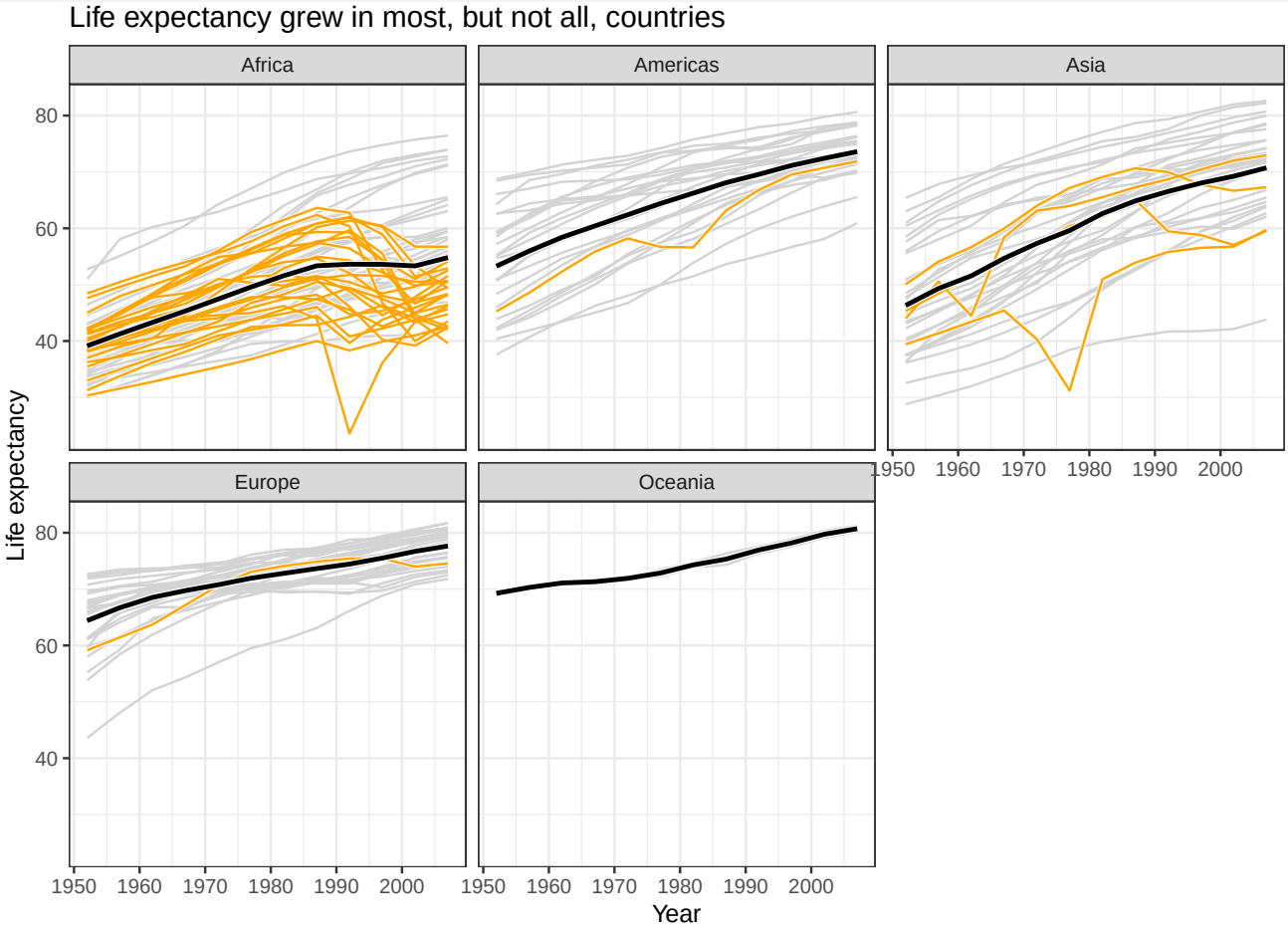
```
library(tidyverse)
library(gapminder)

declining <- gapminder |>
  group_by(country) |>
  arrange(country, year) |>
  mutate(diff = lifeExp - lag(lifeExp)) |>
  ungroup() |>
  filter(diff < -1.0) |>
  distinct(country)
```

Then we use `semi_join` and `anti_join` to partition the original dataset in countries that saw a decline in life expectancy (to be highlighted in orange) and countries that should remain in the background.

```
highlight_countries <- semi_join(gapminder, declining)
no_highlight_countries <- anti_join(gapminder, declining)
```

We use the two datasets above in two different layers, with different colors. Furthermore, we add a line reporting the average for each continent.

```
gapminder |>
  ggplot(aes(x=year, y=lifeExp)) +
  geom_line(aes(group=country), data=no_highlight_countries, color="lightgray", linewidth=0.5) +
  geom_line(aes(group=country), data=highlight_countries, color="white", linewidth=0.9) +
  geom_line(aes(group=country), data=highlight_countries, color="orange", linewidth=0.5) +
  # The following layer is a bit of a trick: adding a slightly thicker line with the
  # same color as the background allows for the main line to stand out more from
  # the surrounding context
  geom_line(stat="summary", linewidth=1.5, color="white") +
  geom_line(stat="summary", linewidth=1) +
  facet_wrap(vars(continent)) +
  labs(
    title="Life expectancy grew in most, but not all, countries",
    x = "Year",
    y = "Life expectancy",
    color = "Continent",
    caption = "Data from gapminder.org"
  ) +
  theme_bw()
```



Case study: pie charts

Pie charts are very popular, even though they are not a very accurate way of representing information. In the grammar of graphics a pie chart is a bar chart in polar coordinates. The following snippet gives an example.

Hints:

- Pie charts are best used when there are few slices.
- Use `color="white"` in order to have a nice white line separating the slices of the pie.

```
# This function is used to format labels.
num_fmt <- scales::label_number(scale=1/1000000, suffix=" M")

gapminder |>
  filter(year == 2007) |>
  group_by(continent) |>
  summarise(pop = sum(pop)) |>
  ungroup() |>
  arrange(desc(continent)) |>
  # The following mutate is needed to compute the position of labels.
  mutate(labelpos = cumsum(pop) - pop/2) |>
  ggplot(aes(
    x=0,
    y=pop,
    fill=continent
  )) +
  geom_col(color="white") +
  geom_text(aes(label=num_fmt(pop), x=0.6, y=labelpos), color="black") +
  scale_fill_tableau() +
  coord_polar(theta="y") +
  theme_void()
```