**Principles for effective visualizations**

* Order matters
* Put long categories on the y-axis
* Keep scales consistent
* Select meaningful colors
* Use meaningful and nonredundant labels

X-axis label:

\* alphabetical order

ggplot(brexit, aes(x = opinion)) +

geom\_bar()

\* fct\_infreq: Reorder factors' levels by frequency

ggplot(brexit, aes(x = fct\_infreq(opinion))) +

geom\_bar()

\* clean up labels

ggplot(brexit, aes(x = opinion)) +

geom\_bar() +

labs(

x = "Opinion",

y = "Count"

)

\* fct\_relevel: Reorder factor levels using a custom order

brexit <- brexit %>%

mutate(

region = fct\_relevel(

region,

"london", "rest\_of\_south", "midlands\_wales", "north", "scot"

)

)

\* fct\_recode: Change factor levels by hand

brexit <- brexit %>%

mutate(

region = fct\_recode(

region,

London = "london",

`Rest of South` = "rest\_of\_south",

`Midlands / Wales` = "midlands\_wales",

North = "north",

Scotland = "scot"

)

)

ES: From what i can tell, this just gives better labels than the ones that are already there

Mason says to move long categories to the y-axis. By “categories” she means category labels. This makes frequency go on the x-axis. I don’t like this approach, but can see that others might disagree.

ggplot(brexit, aes(y = region)) +

geom\_bar()

ggplot(brexit, aes(x = region)) +

geom\_bar() +

coord\_flip()

(I believe the above two commands do the same thing.)

\* fct\_rev: Reverse order of factor levels

ggplot(brexit, aes(y = fct\_rev(region))) +

geom\_bar()

She has this under “clean up labels” – again, I think this is just labeling. (It seems to be necessary, because for some reason the “fact\_rev” included fct\_rev as part of the variable name.)

ggplot(brexit, aes(y = fct\_rev(region))) +

geom\_bar() +

labs(

x = "Count",

y = "Region"

)

*Segmented bar charts*

ggplot(brexit, aes(y = region, fill = opinion)) +

geom\_bar()

Can use facets:

ggplot(brexit, aes(y = opinion, fill = region)) +

geom\_bar() +

facet\_wrap(~region, nrow = 1)

\* es – this looks horrible!

Mason has: Avoid redundancy?

ggplot(brexit, aes(y = opinion)) +

geom\_bar() +

facet\_wrap(~region, nrow = 1)

Then she has: “redundancy can help tell a story” (this makes sense to me…)

ggplot(brexit, aes(y = opinion, fill = opinion)) +

geom\_bar() +

facet\_wrap(~region, nrow = 1)

If the legend is taking up too much space, can take it out….

ggplot(brexit, aes(y = opinion, fill = opinion)) +

geom\_bar() +

facet\_wrap(~region, nrow = 1) +

guides(fill = FALSE)

Note in the slides there’s:

## Warning: `guides(<scale> = FALSE)` is deprecated. Please use

## `guides(<scale> = "none")` instead.

For some reason, this isn’t in the video.

*Adding a title*:

ggplot(brexit, aes(y = opinion, fill = opinion)) +

geom\_bar() +

facet\_wrap(~region, nrow = 1) +

guides(fill = FALSE) +

labs(

title = "Was Britain right/wrong to vote to leave EU?",

x = NULL, y = NULL

)

*Can add subtitles and captions*:

ggplot(brexit, aes(y = opinion, fill = opinion)) +

geom\_bar() +

facet\_wrap(~region, nrow = 1) +

guides(fill = FALSE) +

labs(

title = "Was Britain right/wrong to vote to leave EU?",

subtitle = "YouGov Survey Results, 2-3 September 2019",

caption = "Source: https://d25d2506sfb94s.cloudfront.net/cumulus\_uploads/document/x0msmggx08/YouGov%20-%20Brexit%20and%202019%20election.pdf",

x = NULL, y = NULL

)

*can adjust facet labels*:

ggplot(brexit, aes(y = opinion, fill = opinion)) +

geom\_bar() +

facet\_wrap(~region,

nrow = 1,

labeller = label\_wrap\_gen(width = 12)

) +

guides(fill = FALSE) +

labs(

title = "Was Britain right/wrong to vote to leave EU?",

subtitle = "YouGov Survey Results, 2-3 September 2019",

caption = "Source: bit.ly/2lCJZVg",

x = NULL, y = NULL

)

*can manually choose colors*

ggplot(brexit, aes(y = opinion, fill = opinion)) +

geom\_bar() +

facet\_wrap(~region, nrow = 1, labeller = label\_wrap\_gen(width = 12)) +

guides(fill = FALSE) +

labs(title = "Was Britain right/wrong to vote to leave EU?",

subtitle = "YouGov Survey Results, 2-3 September 2019",

caption = "Source: bit.ly/2lCJZVg",

x = NULL, y = NULL) +

scale\_fill\_manual(values = c(

"Wrong" = "red",

"Right" = "green",

"Don't know" = "gray"

))

To get choice of colors:

There’s a website that lets you specify the exact color:

<https://colorbrewer2.org/>

This lets you specify the precise color, by its alphanumeric designation!

ggplot(brexit, aes(y = opinion, fill = opinion)) +

geom\_bar() +

facet\_wrap(~region, nrow = 1, labeller = label\_wrap\_gen(width = 12)) +

guides(fill = FALSE) +

labs(title = "Was Britain right/wrong to vote to leave EU?",

subtitle = "YouGov Survey Results, 2-3 September 2019",

caption = "Source: bit.ly/2lCJZVg",

x = NULL, y = NULL) +

scale\_fill\_manual(values = c(

"Wrong" = "#ef8a62",

"Right" = "#67a9cf",

"Don't know" = "gray"

))

*Can select a theme*:

ggplot(brexit, aes(y = opinion, fill = opinion)) +

geom\_bar() +

facet\_wrap(~region, nrow = 1, labeller = label\_wrap\_gen(width = 12)) +

guides(fill = FALSE) +

labs(title = "Was Britain right/wrong to vote to leave EU?",

subtitle = "YouGov Survey Results, 2-3 September 2019",

caption = "Source: bit.ly/2lCJZVg",

x = NULL, y = NULL) +

scale\_fill\_manual(values = c("Wrong" = "#ef8a62",

"Right" = "#67a9cf",

"Don't know" = "gray")) +

theme\_minimal()

Can see some themes at:

https://www.datanovia.com/en/blog/ggplot-themes-gallery/

https://ggplot2.tidyverse.org/reference/ggtheme.html

*Viridis*

The viridis R package contain palettes that represent good choices for color-blind friendly palettes and printing in gray scale. The developers provide [additional information](https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html) about each of the eight color palettes available, including how it is visualized with different types of color-blindness and in gray scale.

Usage of the viridis palettes is straight-forward, and the package provides ggplot2 scale functions to easily switch to these palettes. Let’s use the default color palette, viridis.

Mason says viridis is good for ordinal data

ggplot(brexit, aes(y = region, fill = opinion)) +

geom\_bar(position = "fill") +

scale\_fill\_viridis\_d()

**Grammar of graphics**

The power of the grammar of graphics is that it is modular:

* different aspects of the plot can be specified independently of each other.

Can specify an element, which can then be used in later analyses. In this case, I believe “bar\_plot” is an element that is being called in later commands (?)

bar\_plot = ggplot(diamonds) +

aes(x ="", fill = clarity)+

geom\_bar(width = 1, position = "stack")

The following 3 commands produce very different coordinates.

bar\_plot

bar\_plot + coord\_polar()

bar\_plot + coord\_polar(theta = "y")

Now adding in “dodge” (Mason seems to think we know what this does already … )

Google says: A dodged bar plot is **used to compare a grouping variable, where the groups are plotted side by side**.

Again, the point here is to show how different it looks with different coordinates:

dodged\_bar\_plot = ggplot(diamonds) +

geom\_bar(aes(x = "", fill = clarity), width = 1, position = "dodge")

dodged\_bar\_plot

dodged\_bar\_plot + coord\_polar()

dodged\_bar\_plot + coord\_polar(theta = "y")

**Components of a plot**

* A default dataset and set of mappings from variables to aesthetics.
* One or more *layers*, each of which contains
  + one geometric (geom\_\*) object,
  + one statistical transformation (stat\_\*),
  + one position adjustment (position\_\*),
* and one dataset and set of aesthetic mappings.
* One scale for each aesthetic mapping.
* A coordinate system (coord\_\*).
* A facet specification (facet\_\*).

**What is a layer?**

* data and aesthetic mapping
* statistical transformation (stat)
* geometric object (geom)
* position adjustment

p = ggplot(diamonds, aes(x = color, y = clarity)) + geom\_point()

p$layers

## [[1]]

## geom\_point: na.rm = FALSE

## stat\_identity: na.rm = FALSE

## position\_identity

Aesthetic mapping

* Describes how variables in the dataset are mapping
  + to "aesthetic" attributes of the plot.
* "Aesthetic" here means perceivable:
  + something you can observe on the plot.

**Aesthetic/perceivable attributes**

Examples include:

 position along the x-axis,

 color,

 shape,

 position along the y-axis,

 opacity,

 linetype

The aesthetic mapping takes variables in your data and maps them to aesthetics/perceivable parts of the plot and is therefore specific to a dataset.

Statistical transformations:

A black and white text on a white background

Description automatically generated

**Geometric Objects**

Geometric objects (geom\_\*) control the type of plot you create.

Examples are:

* Points, text
  + (zero-dimensional geometric objects)
* Line, path
  + (one-dimensional geometric objects)
* Polygon, interval
  + (two-dimensional geometric objects)
* More complicated: boxplot
* Every statistic has a default geometric object,
  + and every geometric object has a default statistic.
* Stats and geoms are not completely orthogonal:
  + not every combination is valid
  + (although many are).
* stat\_bin and geom\_bar
  + is valid and
  + standard for a histogram.
* stat\_bin and geom\_point or stat\_bin + geom\_line
  + are valid
  + but non-standard combinations.
  + They give a plot that is resembles a histogram
    - (interpretable that same way, too)
* stat\_identity and geom\_boxplot
  + is invalid, because
  + boxplot needs certain computed quantities from the data.

**Position adjustment**

Used to avoid "collisions" in the plot objects:

* In bar plots where one of the aesthetics is height,
  + the bars would often be plotted over each other.
  + In this case we use the "dodge" or "stack" position adjustments.
* If we have issues with overplotting (multiple points in exactly the same place),
  + we can use the "jitter" position adjustment
  + to randomly move the points a small amount.

(p = ggplot(diamonds, aes(x = color, y = price)) +

geom\_boxplot())

ggplot(diamonds,aes(x = color, y = price, color = clarity)) +

geom\_boxplot(position = "identity")

ggplot(diamonds,aes(x = color, y = price, color = clarity)) +

geom\_boxplot(position = "dodge")

(Note: "dodge" is the default for boxplots, so you don't need to specify it.)

ggplot(diamonds,aes(x = color, y = price, color = clarity)) +

geom\_boxplot(position = position\_dodge(width = 1))

**Scales**

So far, we’ve defined aesthetic mappings that specify which perceived aspects of the plot correspond to which variables. However, perceivable aspects of the plot can be mapped to variables many other ways.

For example, if we have a categorical variable that takes values “A” and “B” to the color aesthetic, that means that color is going to represent whether variable took value “A” or “B.” But, we could do that in practically an infinite number of ways.

*ggplot* has good default mappings from values into aesthetic space, but you will sometimes want to set them by hand. You can use the *scale* function to do that.

The most recent version of ggplot uses viridis by default for both continuous values and ordered factors.

**Coordinate Systems**

* coord\_cartesian is the default,
  + and is almost always what you want.
* coord\_flip is sometimes useful:
  + for example, boxplots require the explanatory variable to be mapped to x,
  + so if you want a horizontal boxplot you need to use coord\_flip.+ coord\_polar will often make your plots look cooler
  + and more difficult to read.
  + Not usually recommended.

**Faceting**

Allows you to make small multiples of plots.

* Other grammars/plotting systems implement faceting with a fancy coordinate system,
* but it turns out that it's easier to use if you think about it separately.
* Each facet plots a subset of the data,
  + and it takes as input
  + what variable(s) to use to make the subsets and
  + how to lay out the subsets.

*Facet Options*

* facet\_wrap:
  + Lays out the plots for each subset sequentially.
* facet\_grid:
  + Subsets the data according to two separate variables.

The facet position along the x-axis represents levels of one variable, and the facet position along the y-axis represents levels of the other variable.

**Implementing in ggplot**

One way to specify a ggplot is to specify all of the components we've seen. If you understand all the parts, this way is probably the least confusing method to specify a ggplot.

\* The problem is that it's verbose.

* Suppose we want to make a plot with points and a smoother
  + from the diamonds dataset.
* We can specify data, mapping, geom, stat, and positions for each layer,
  + along with scales and the coordinate system as follows:

ggplot() +

layer(

data = diamonds, mapping = aes(x = carat, y = price),

geom = "point", stat = "identity", position = "identity") +

layer(

data = diamonds, mapping = aes(x = carat, y = price),

geom = "smooth", position = "identity", stat = "smooth", params = list(method = "lm")) +

scale\_x\_log10() + scale\_y\_log10() + coord\_cartesian()

The more standard way of generating the same plot would be:

p = ggplot(data = diamonds, aes(x = carat, y = price)) +

geom\_point() +

stat\_smooth(method = "lm") +

scale\_x\_log10() +

scale\_y\_log10()

This code is still fairly long, but we don't have to specify...

* position: Default for both geom\_point and stat\_smooth is position = "identity".
* stat, for geom\_point: The default stat for geom\_point is stat = "identity".
* geom, for stat\_smooth: The default geom for stat\_smooth is geom\_smooth.
* coordinate system: coord\_cartesian is always the default.

If you don't know what the default values for some of the aspects of the plot, examine p$layers

Remember that a histogram is a plot with stat\_bin and geom\_bar.

One issue is to make sure everything is in a data frame. This is simple in ggplot, but can cause problems with specifying in other commands:

Sadly, not all functions offer a data = argument. Take cor(), for example, which computes correlation. This does **not** work:

cor(year, lifeExp, data = gapminder)

#> Error in cor(year, lifeExp, data = gapminder): unused argument (data = gapminder)

A solution is to use “with” :

The with() function is a better workaround. Provide the data frame as the first argument. The second argument is an expression that will be evaluated in a special environment. It could be a single command or a multi-line snippet of code. What’s special is that you can refer to variables in the data frame by name.

with(

gapminder,

cor(year, lifeExp)

)

#> [1] 0.436

If you use the magrittr package, another option is to use the %$% operator to expose the variables inside a data frame for further computation:

library(magrittr)

gapminder %$%

cor(year, lifeExp)

#> [1] 0.436

Can restructure datasets, for example, combining multiple variables into one variable. Here’s an example:

Here’s the minimal code to produce our Japan example.

japan\_tidy <- gapminder %>%

filter(country == "Japan") %>%

gather(key = var, value = value, pop, lifeExp, gdpPercap)

ggplot(japan\_tidy, aes(x = year, y = value)) +

facet\_wrap(~var, scales = "free\_y") +

geom\_point() +

geom\_line() +

scale\_x\_continuous(breaks = seq(1950, 2011, 15))

This snippet demonstrates the payoffs from the rules we laid out at the start:

* We isolate the Japan data into its own **data frame**.
* We **reshape** the data. We gather three columns into one, because we want to depict them via position along the y-axis in the plot.
* We use a **factor** to distinguish the observations that belong in each mini-plot, which then becomes a simple application of faceting.
* This is an example of expedient data reshaping. I don’t actually believe that gdpPercap, lifeExp, and pop naturally belong together in one variable. But gathering them was by far the easiest way to get this plot.

**ggsave**

If you are using ggplot2, write figures to file with [ggsave()](https://rdrr.io/cran/ggplot2/man/ggsave.html).

If you are staring at a plot you just made on your screen, you can call ggsave(), specifying only a filename:

ggsave("my-awesome-graph.png")

It makes a sensible decision about everything else. In particular, as long as you use a conventional extension, it will guess what type of graphics file you want. If you need control over, e.g., width, height, or dpi, roll up your sleeves and [use the arguments](https://rdrr.io/cran/ggplot2/man/ggsave.html).

p <- ggplot(gapminder, aes(x = year, y = lifeExp)) +

geom\_jitter()

# during development, you will uncomment next line to print p to screen

# p

ggsave("fig-io-practice.png", p)

**Scaling**

Figures need to be prepared differently for a presentation versus a poster versus a manuscript. You need to fiddle with the size of text, such as the title and axis labels, relative to the entire plot area. There are at least two ways to do this, with slightly different effects and workflows.

**Via the scale = argument to ggsave()**: This actually changes the physical size of the plot, but as an interesting side effect, it changes the relative size of the title and axis labels. Therefore, tweaking this can be a quick-and-dirty way to get different versions of a figure appropriate for a presentation versus a poster versus a manuscript. You can still insert the figure downstream with a different physical size, though you may need to adjust the dpi accordingly on the front end. When scale < 1, various plot elements will be bigger relative to the plotting area; when scale > 1, these elements will be smaller. YMMV but scale = 0.8 often works well for posters and slides. Figure [39.1](https://datascience4psych.github.io/DataScience4Psych/save-figs.html#fig:exaggerated-scale) shows two versions of a figure, with exaggerated values of scale, to illustrate its effect.

library(ggplot2)

library(gapminder)

p <- ggplot(gapminder, aes(

x = year,

y = lifeExp

)) +

geom\_jitter()

p1 <- p + ggtitle("scale = 0.6")

p2 <- p + ggtitle("scale = 2")

ggsave("img/fig-io-practice-scale-0.6.png", p1, scale = 0.6)

#> Saving 4.2 x 3 in image

ggsave("img/fig-io-practice-scale-2.png", p2, scale = 2)

#> Saving 14 x 10 in image

**Via the base\_size of the active theme**: The base\_size of the [theme](https://ggplot2.tidyverse.org/reference/ggtheme.html#arguments) refers to the base font size. This is NOT a theme element that can be modified via ggplot(...) + theme(...). Rather, it’s an argument to various functions that set theme elements. Therefore, to get the desired effect you need to create a complete theme, specifying the desired base\_size. By setting base size < 12, the default value, you shrink text elements and by setting base\_size > 12, you make them larger. Figure [39.2](https://datascience4psych.github.io/DataScience4Psych/save-figs.html#fig:exaggerated-base-size) shows two versions of a figure, with exaggerated values of base\_size, to illustrate its effect.

p3 <- p + ggtitle("base\_size = 20") + theme\_grey(base\_size = 20)

p4 <- p + ggtitle("base\_size = 3") + theme\_grey(base\_size = 3)

ggsave("img/fig-io-practice-base-size-20.png", p3)

#> Saving 7 x 5 in image

ggsave("img/fig-io-practice-base-size-3.png", p4)

#> Saving 7 x 5 in image