Overview of Visualization

INFO 526 Data Analysis and Visualization

Adriana Picoral

This tutorial's aim is two-fold:

1. Provide an overview of visualization, including the language around encoding, color, visual perception etc. (We will be referring to the readings a lot in this tutorial)
2. Introduce you to the coding tools we will be using in this course (R, R Studio, R Markdown)

All tutorials for this class will be available in three formats on D2L: written tutorial as pdf file, a video tutorial with similar (but not always exactly the same) content, and the markdown file (.Rmd) with the source code for the pdf file.

The learning outcomes of this tutorial include:

- Use appropriate language/terminology for describing visualizations
- Identify and apply the basic principles of visualization, including color, expressiveness, effectiveness, and pre-attentive processing
- Identify visualization errors and pitfalls

Edges, contrasts, and colors

Our visual system (i.e., all the visual processing models from our eyes to our brain) re-constructs what we are looking at and it uses relative differences instead of absolute values for brightness and colors. This relativization of brightness is demonstrated in the image below, in which we perceive darker blobs where grid lines meet at the edge of our visual field (i.e., Herman grid effect). Please refer to Healy's chapter 1 (pages 14-15) for more information about this. Here's a recreation of Figure 1.13 found on page 14 (Healy, 2018) - we will be using the function `vi.grid.illusion()` from the `animation` library for this:

```
library(animation)

vi.grid.illusion(type = "h",
                 col = "white",
                 nrow = 6,
                 ncol = 6,
                 lwd = 15)
```

Another side effect of our visual system processing is perceiving the same shade as lighter or darker depending on the background behind shade. In the image below, the smaller squares are all the exact shade of gray, but we perceive the gray to be lighter in darker backgrounds and darker in lighter backgrounds. We will use the `grid` library to visualize this effect.

```
library(grid)
grid.newpage()
grid.rect(x = c(1,3,1,3)/4,
          y = c(3,3,1,1)/4,
          width = 1/2,
```

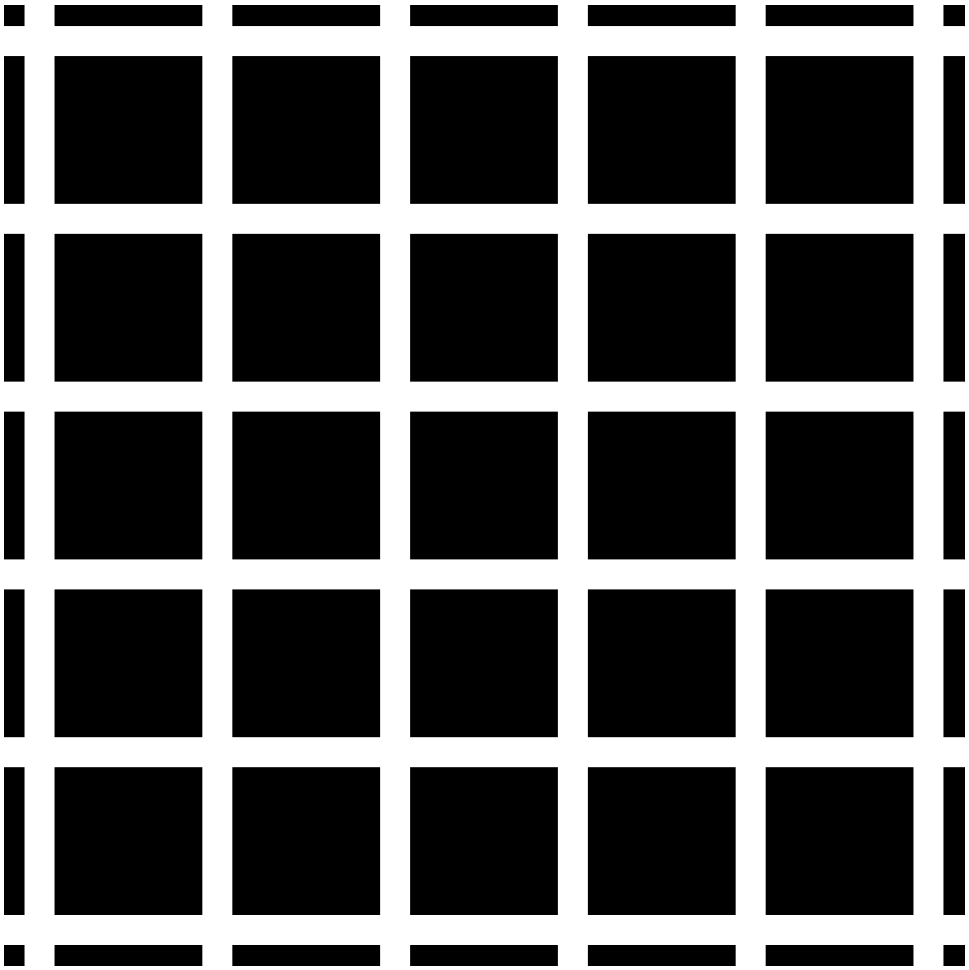

Figure 1: Hermann grid illusion. Grey blobs are perceived at the intersections of white grid lines.

```
          height = 1/2,
          gp = gpar(col = NA,
                    fill = gray(1:4/5)))

grid.rect(x = c(1,3,1,3)/4,
          y = c(3,3,1,1)/4,
          width = 1/6,
          height = 1/6,
          gp = gpar(col = NA,
                    fill = gray(0.5)))
```

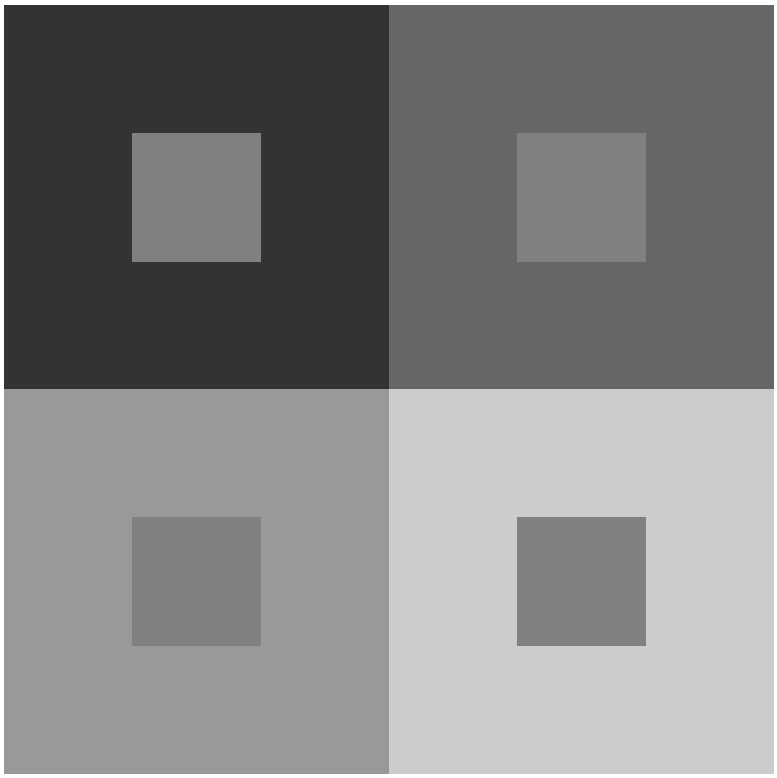

Figure 2: Same gray on different gray backgrounds looks darker in lighter backgrounds and lighter in darker backgrounds.

Things to keep in mind regarding edges, colors, and contrasts (from Healy, 2018):

- Our visual system is not a physical light meter, what we perceive is different from actual values for colors and brightness
- Humans are better at seeing edge contrasts for monochrome images
- We are also better at distinguishing dark shades (compared to light shades)

Color

Let's start with some definitions regarding color. "Color" refers to three components:

- *Hue* refers to the dominant wavelength of the color (e.g., red, blue, purple)
- *Chroma* refers to how intense of vivid the color is
- *Luminance* refers how bright a color is

What we need to consider when using color in our data visualizations is that color schemes need to represent accurately differences in our data. For example, **changes from one level to next need to be perceive as having the same magnitude**. We should be careful when picking colors for our variables in our data visualizations. Here are some points to consider:

- **Gradients** should be used as sequential scales from low to high to represent numeric continuous/ordered variables (e.g., population size). Light colors should represent low values, and dark colors high values.
- **Diverging scales** should be used where there is a neutral midpoint and then there is variance in ether direction from that neutral point (e.g., temperatures can be positive and negative).
- **Qualitative palettes** where each color has the same valence (i.e., no color dominates the other) should be used for unordered categorical variables (e.g., political parties, countries). Different colors in the palette should **not** imply differences in magnitude.

You can get more information about variable types (numeric, categorical, etc) on the reading for this module on data types: Chapter 1 of *Introduction to Modern Statistics* by Mine Çetinkaya-Rundel and Johanna Hardin (download pdf from D2L or access entire book at https://openintro-ims.netlify.app/)

In addition, we need to take into account people who are color-blind, and how they perceive colors.

In general, it is safe to use palettes that have been pre-built, such as the ones we find in the `RColorBrewer` library.

```
# load library
library(RColorBrewer)

# example of a gradient
display.brewer.pal(9,"Blues")
```

Blues (sequential)

Figure 3: An example of a gradient color scheme (use this type of color scheme to represent numeric continuous/ordered variables - e.g., population size)

```
# example of diverging scale
display.brewer.pal(9, "BrBG")
```

```
# example of a qualitative palette
display.brewer.pal(9,"Dark2")
```

```
## Warning in display.brewer.pal(9, "Dark2"): n too large, allowed maximum for palette Dark2 is 8
## Displaying the palette you asked for with that many colors
```

Visual Search and Pop-out Effect

In addition to color, we can use shape as a channel to encode information. The color channel, however, is considered more salient, with shape used as a secondary channel.

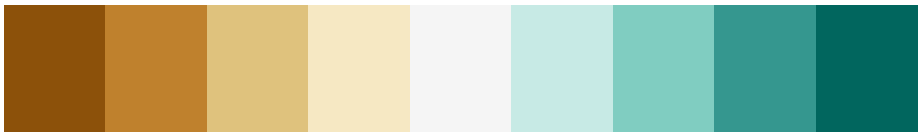

Figure 4: An example of a diverging color scheme (use this color scheme to represent data with a neutral midpoint and variance in ether direction from that neutral point - e.g., temperatures can be positive and negative)

Figure 5: An example of a qualitative color scheme (use this color scheme to represent unordered categorical variables - e.g., political parties, countries)

To demonstrate the differences between encoding information through color versus shape, let's first create some random data to visualize in a scatterplot. We will use a random uniform distribution function (i.e., `runif()`) to create 100 data points (x and y coordinates). We will label 99 of them as `A` and 1 of them as `B`.

```
# set seed to get reproducible results
set.seed(11)

# create random data set
my_random_data <- data.frame(y = runif(100),
                             x = runif(100),
                             type = c(rep("A", times = 99), "B"))
```

Try to find the `type B` data point in the three plots below. We will be creating new random data with a new `seed` for each plot so you have to look for point B at different positions. We will be using `ggplot` from the `tidverse` package for all of our plotting. We will also be using the `ggthemes` library to use the `scale_color_colorblind()` function in our scatterplots.

```
# load tidyverse
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
```

```
## x dplyr::lag()    masks stats::lag()
```

```
library(ggthemes)

# scatter plot
my_random_data %>%
  ggplot(aes(x = x,
             y = y,
             color = type)) +
  geom_point() +
  theme_linedraw() +
  scale_color_colorblind()
```

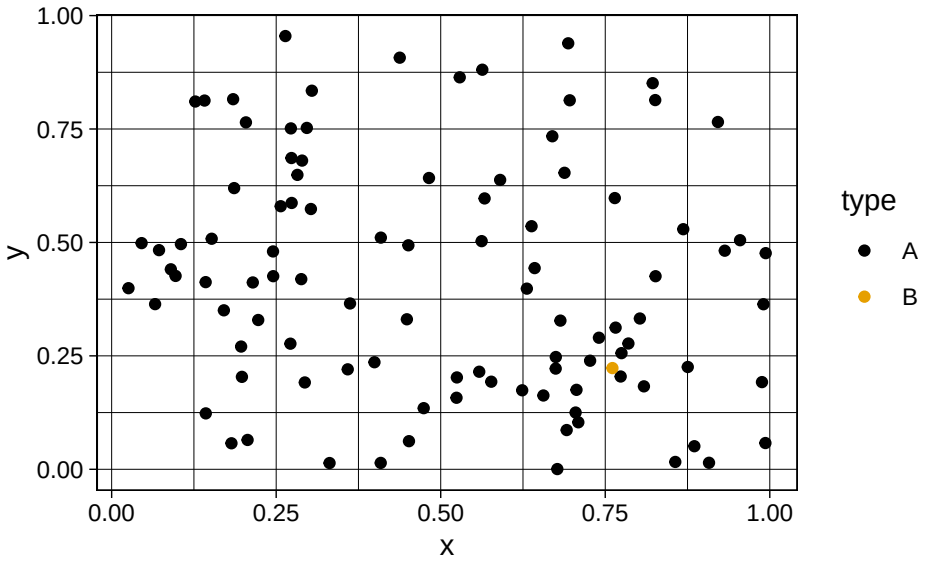

Figure 6: Find the yellow (i.e., type B) data point.

```
set.seed(13)

# create random data set
my_random_data <- data.frame(y = runif(100),
                             x = runif(100),
                             type = c(rep("A", times = 99), "B"))

# scatter plot
my_random_data %>%
  ggplot(aes(x = x,
             y = y,
             shape = type)) +
  geom_point() +
  theme_linedraw() +
  scale_color_colorblind()
```

```
set.seed(7)

# create random data set
my_random_data <- data.frame(y = runif(100),
                             x = runif(100),
```

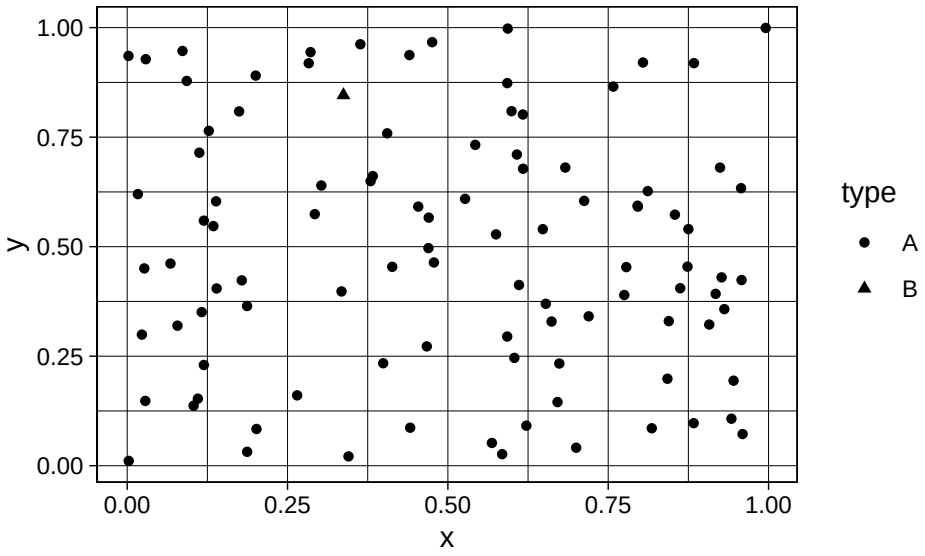

Figure 7: Find the triangular (i.e., type B) data point.

```
                        type = c(rep("A", times = 99), "B"))

# scatter plot
my_random_data %>%
  ggplot(aes(x = x,
             y = y,
             shape = type,
             color = type)) +
  geom_point() +
  theme_linedraw() +
  scale_color_colorblind()
```

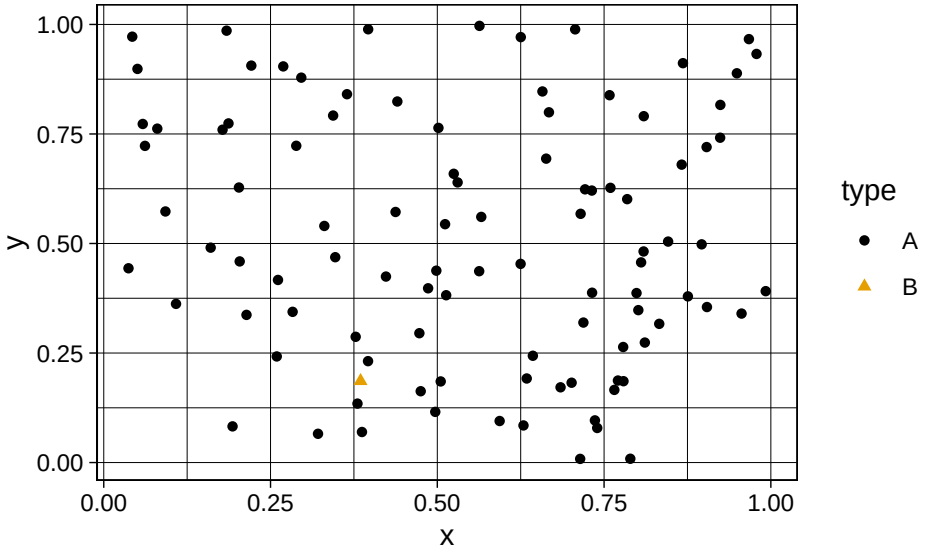

Figure 8: Find the yellow and triangular (i.e., type B) data point.

The `type B` data point should have been easier to find in the first and last plots, and harder in the middle

plot. That is because color is a channel that is easier to process than shape. Combining both shape and color is at times a good strategy. But not always! Encoding multiple channels in the same visualization can overtax the viewer (see the top plot below) when the multiple channels are not highlighting some structure in the data (see the bottom plot below).

We will be using the `patchwork` library to combine multiple plots into one for the next chunk of code.

```
#create type vectors
type1 = c(rep("A", times = 50),
          rep("B", times = 50))

type2 = c(rep("1", times = 50),
          rep("2", times = 50))

# create random data set
my_random_data <- data.frame(y = runif(100),
                             x = runif(100),
                             type1 = sample(type1),
                             type2 = sample(type2))

# create the first scatter plot with random data (save it as an object)
p1 <- my_random_data %>%
  ggplot(aes(x = x,
             y = y,
         color = type1,
         shape = type2)) +
  geom_point() +
  theme_linedraw() +
  scale_color_colorblind()


# create clusters of data points
my_group_data_1 <- data.frame(y = rnorm(25, mean =  0.8, sd = .15),
                              x = rnorm(25, mean =  0.8, sd = .15),
                              type1 = rep("A", times = 25),
                              type2 = rep("1", times = 25))

my_group_data_2 <- data.frame(y = rnorm(25, mean =  0.8, sd = .15),
                              x = rnorm(25, mean =  0.25, sd = .15),
                              type1 = rep("B", times = 25),
                              type2 = rep("1", times = 25))

my_group_data_3 <- data.frame(y = rnorm(25, mean =  0.25, sd = .15),
                              x = rnorm(25, mean =  0.25, sd = .15),
                              type1 = rep("A", times = 25),
                              type2 = rep("2", times = 25))

my_group_data_4 <- data.frame(y = rnorm(25, mean =  0.25, sd = .15),
                              x = rnorm(25, mean =  0.8, sd = .15),
                              type1 = rep("B", times = 25),
                              type2 = rep("2", times = 25))

# bind all data points together for visualization
my_group_data <- bind_rows(my_group_data_1,
                           my_group_data_2,
```

```
                          my_group_data_3,
                          my_group_data_4)

# create the second plot (save it as an object)
p2 <- my_group_data %>%
  ggplot(aes(x = x,
             y = y,
         color = type1,
         shape = type2)) +
  geom_point() +
  theme_linedraw() +
  scale_color_colorblind()
```

```
library(patchwork)
# combine two plots (need patchwork library for this to work)
p1 / p2
```

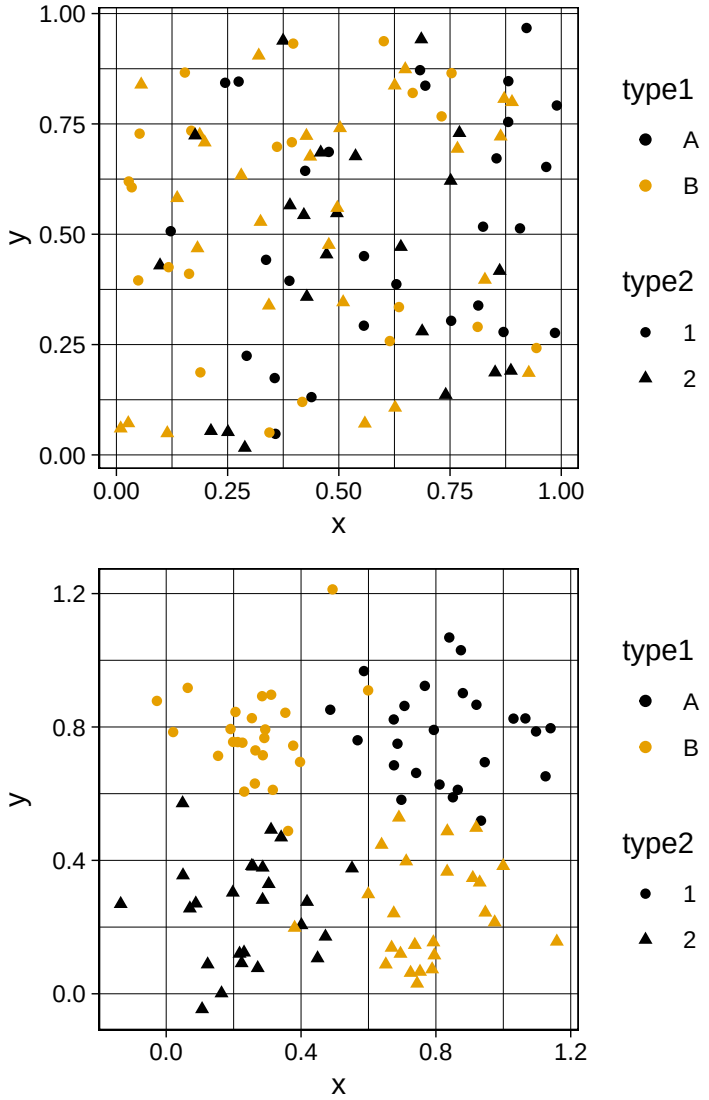

Figure 9: Encoding information in multiple channels is not always a good solution.

Other encoding channels that are even less easily processed than shape are size, elongation, and movement.

What makes a great visualization?

In general, a great visualization displays honest information in a beautiful manner that is easy to read and interpret. So a great visualization is the result of:

Honesty + Good Judgment

It's also important to keep in mind the amount of information displayed – a great visualization should not omit information, but it should not contain more information than the point you want to make (principle of expressiveness).

In the following sections, these two elements are broken down further. Most of this material is summarized from Kieran Healy's book "Data Visualization: A Practical Introduction" (Healy, 2019), Chapter 1.

Good aesthetics

Good aesthetics (as opposed to bad taste) is hard to operationalize, but here are some general guidelines (that might be broken at times):

- Maximize data-to-ink ratio
- Clean up typefaces (the fewer different typefaces, the better)
- Avoid extraneous colors and backgrounds
- Simplify, mute, or delete gridlines
- Remove superfluous axis marks and labels
- Avoid excessive decorative embellishments

You want to ensure that your visualizations are beautiful, easy to read and interpret, and memorable (i.e., visually distinctive) – these goals might conflict with each other at times and only practice will tell you how to balance all of these elements.

No substantive issues

Here are two main general issues to **avoid**:

- Bad data: ensure you are working with good quality data, and do your homework in terms of data checking and transforming (this process often takes a lot of time, but it is essential)
- Tricky scales: by simply choosing a line plot versus a bar plot, you scale for your axis might change, changing the perception of how big or small differences across years or groups are. When providing comparisons, make sure you are as transparent about your scales as possible. The easiest way is to be honest is to keep the same scale across different panels and/or plots and keep the full scale of the original data (e.g., if you are displaying percentages on the y scale, make sure your limits for the y axis are set for 0% to 100%).
- Perceptual issues: Healy (2019) argues that "visualizations encode numbers in lines, shapes, and colors" (p. 11), whose interpretations rely on how our visual system and how we perceive geometric shapes and relationships. As such, some charts are more difficult to interpret than others. As such, some visualizations are are more difficult than others. Due to human perception, 3D plots are often harder to read and interpret differences between bars, for example. Stacked bars make general trends easier to read, but differences between subcategories are harder to interpret.