# Working with Data Workshop Presentation

## NIH Library

Doug Joubert

2024-07-25

## Live Portion (RStudio)

### Outline for Demo

1. Open our Project
2. Load packages and import data
3. Explore options for time and date data in R
4. Build a line graph
5. Facet our data
6. Customize our basic plot by adding labels, adding a legend, and using color

## Opening the Project in RStudio

### Opening our Project

1. Open RStudio
2. Choose the Project that we created in the Intro to ggplot Class (nihl-r-classes)

### Creating a New Script

- New R script by clicking `File → New File → R Script`
- You can also click the green + button in the top left corner of RStudio
- You can also use the keyboard shortcut Shift+Cmd+N (Mac) or Shift+Ctrl+N (Windows)
- Script will be be unsaved, and called "Untitled1"
- Let us name our script `nihl-ggplot-custom`

## Installing and Loading Packages

### lubridate

- Lubridate package, makes it easier to work with dates and times in R
- Part of core tidyverse, but it also exists as a standalone package

### RColorBrewer

- Cynthia Brewer, a geographer and color specialist, created sets of colors for print and the web and they are available in the add-on package RColorBrewer

~~~
```{r}
# install.packages("RColorBrewer")
#| output: false
library(RColorBrewer)
```
~~~

### Tidyverse Package

- Collection of R packages designed for data science
- All packages share an underlying design philosophy, grammar, and data structures

~~~
```{r}
# install.packages("tidyverse")
#| output: false
library(tidyverse)
```
~~~

### viridis: Colorblind-Friendly Color Maps for R

- Color maps designed to improve graph readability for readers with common forms of color vision deficiency
- Color maps are also perceptually-uniform, both in regular form and also when converted to black-and-white for printing
- Package also contains 'ggplot2' bindings for discrete and continuous color and fill scales. Vignette has great examples

~~~
```{r}
# install.packages("viridis")
#| output: false
library(viridis)
```
~~~

## Importing Data

### About the Study (Raw Data)

- Using part of the data published by Blackmore *et al.* (2017), *The effect of upper-respiratory infection on transcriptomic changes in the CNS*
- Gender matched mice were inoculated with saline or with Influenza A by intranasal route
- Transcriptomic changes in the cerebellum and spinal cord tissues were evaluated by RNA-seq at days 0 (non-infected), 4 and 8
- Full column information can be found here

### Example 1: Importing SubRNA Data

- Used `mutate()` on a subset of the Blackmore data to create a `expression_log` variable
- Use `read_csv()` to import the sub-rna.csv data into a new object called `sub_rna`

### Example 1: Importing SubRNA Data (Solution)

~~~
```{r}
sub_rna <- read_csv("../lsc563-joubert/data-raw/sub-rna.csv")
```
~~~

### Example 2: Import Mean Expression by Time Data

- Using the `sub_rna` data, I used `group_by()` to group the data by `gene` and then by `time`
- Combined this with `summarize()` to calculate the mean of the `expression_log` variable
- Use `read_csv()` to import the mean-exp-by-time.csv data into a new object called `mean_exp_by_time`

### Example 2: Import Mean Expression by Time Data (Solution)

~~~
```{r}
mean_exp_by_time <- read_csv("../lsc563-joubert/data-raw/mean-exp-by-
time.csv")
```
~~~

## Pipes and Functional Programming

### Pipes

- What if you want to **select** and **filter** at the same time?
- There are three ways to do this: use intermediate steps, nested functions, or pipes
- We are only going to use **pipes** because they are more efficient
- **Pipes** let you take the output of one function and send it directly to the next function
- This is useful when you need to do many things to the same dataset

### Differences Between Base R and Magrittr pipes

- R 4.1.0 introduced a native pipe operator **>|**, which inserts the left-hand side as the first argument in the right-hand side call (Wickham, 2023)
- Native pipe behavior is mostly the same as that of the **`%>%`** pipe provided by the magrittr package
- Both let you “pipe” an object forward to a function or call expression, thereby allowing you to express a sequence of operations that transform an object (Wickham, 2023)

### Differences between the base R and magrittr pipes

- To learn more about the basic utility of pipes, see the pipe section of *R for Data Science*
- In RStudio, you can type the pipe with `Ctrl + Shift + M` if you have a PC or `Cmd + Shift + M` if you have a Mac
- In RStudio, you can type the pipe with `Ctrl + Shift + M` if you have a PC or `Cmd + Shift + M` if you have a Mac

### Example Using Pipes

1. Filter the data to only include male mice
2. Only keep data on the `gene`, `sample`, `tissue`, `expression` variables

### Example Using Pipes (Solution)

~~~
```{r}
sub_rna %>%
  filter(sex == "Male") %>%
  select(gene, sample, expression) %>%
  arrange(desc(expression))
```
~~~

### What is the Pipe Doing?

- Pipe sends the **sub_rna** dataset first through **filter()** to keep rows where sex is Male
- Pipe then uses **select()** to keep only the gene, sample, and expression columns
- Last, the pipe uses **arranger(desc())** to arrange the expression values from largest to smallest
- Since a **pipe** takes the object on its left and passes it as the first argument to the function on its right, we don't need to explicitly include the data frame as an argument to the **filter()**, **select()** and **arrange()** functions
- Some find it helpful to read the pipe like the word "then"
  - For example, we took the data frame **rna**, then we filtered for rows with **sex == "Male"**, then we selected columns **gene**, **sample**, and **expression**…

## Working with Dates or Time

### Working with Dates or Time

- Many different ways that times or dates can be represented
- Appropriate scale for times or dates can be continuous or discrete depending on how the values are stored (Wilke, 2019b):
  - May 31, 2024: 08:30AM
  - 2024-05-31 08:30:00
  - Generic dates such as July 4 or December 25, without any years

### Dates or Time in R

Three types of date/time data that refer to an instant in time (Wickham et al., 2023c):

- Date: tibbles print this as `<date>`
- Time within a day: tibbles print this as `<time>`
- Date-time: date plus a time
  - Uniquely identifies an instant in time (typically to the nearest second)
  - Tibbles print this as `<dttm>`
  - Base R calls these POSIXct. If you are working in base-R, this tutorial explores working with date and time field in R

### Dates in Imported Data

- If your CSV contains an ISO 8601 date or date-time, you don't need to do anything
- For other date-time formats, you'll need to use `col_types` plus `col_date()` or `col_datetime()` along with a date-time format
- Diving into these import options is beyond the scope of this class. However, this topic is covered in the latest version of R for Data Science (Wickham et al., 2023c)

## Building a Graph

### Example 3: Building our Line Graph I

Try this on your own:

1. Build a line plot (geom_line) with duration of the infection on the x-axis
2. Put mean expression on the y-axis
3. Did this work?

### Example 3: Building our Line Graph I: (Solution)

~~~
```{r}
mean_exp_by_time %>%
ggplot(mapping = aes(x = time,y = mean_exp)) +
    geom_line()
```
~~~

### Example 3: Building our Line Graph I: (Explainer)

- Unfortunately, this does not work because we plotted data for all the genes together
- We need to tell ggplot to draw a line for each gene by modifying the aesthetic function to include `group = gene`

### Example 4: Building our Line Graph II

1. Build a line plot with duration of the infection on the x-axis
2. Put mean expression on the y-axis

3. We need to tell ggplot to draw a line for each gene by modifying the aesthetic function to include `group = gene`

### Example 4: Building our Line Graph II (Solution)

~~~
```{r}
mean_exp_by_time %>%
ggplot(mapping = aes(x = time,y = mean_exp, group = gene)) +
    geom_line()
```
~~~

Figure 1: Mean expression values by number of days.

### Example 5: Building our Line Graph III

1. Look at the help documentation for the color attribute (geom_line)
2. See if you can figure out how we can use color with this geom
3. We would like each gene to have its own color

### Example 5: Building our Line Graph III (Solution)

~~~
```{r}
mean_exp_by_time %>%
ggplot(mapping = aes(x = time, y = mean_exp, color = gene)) +
  geom_line()
```
~~~

Figure 2: Mean expression values by number of days, by gene.

- Probably not the best way to divide the data
- Thankfully `ggplot` allows you to create small multiples of the data.

## Faceting

### Faceting

- `ggplot2` has a special technique called *faceting* that allows the user to split one plot into multiple (sub) plots based on a factor included in the dataset
- These different subplots inherit the same properties (axes limits, ticks, …) to facilitate their direct comparison
- We will use it to make a line plot across time for each gene

### Example 6: Facet on Gene

1. Use the `mean_exp_by_time` object
2. Map x to time, y to the mean expression values
3. Create a line graph
4. Facet on gene

## Example 6 (Solution)

~~~
```{r}
mean_exp_by_time %>%
ggplot(data = mean_exp_by_time,
       mapping = aes(x = time, y = mean_exp)) +
    geom_line() +
    facet_wrap(vars(gene))
```
~~~

Figure 3: Mean expression values by number of days, by gene (faceted)

## Modifying the Axis with Scales

- In the previous plot, both the x-axis and y-axis have the same scale for all the sub plots
- Scales are shared across all facets, and the default value is `"fixed"`
- You can change this default behavior by adding `scales =` and setting the value y-axis to `scales = "free_y"`
- You can also use `free_x`, the width will be proportional to the length of the x scale, or if `free` both height and width will vary

## Example 7: Facet with Free Y

1. Use the `mean_exp_by_time` object
2. Map x to time, y to the mean expression values
3. Create a line graph
4. Facet on gene
5. Set the scales value to `free_y`

## Example 7: Facet with Free Y (Solution)

~~~
```{r}
mean_exp_by_time %>%
ggplot(data = mean_exp_by_time,
       mapping = aes(x = time, y = mean_exp)) +
    geom_line() +
    facet_wrap(vars(gene), scales = "free_y")
```
~~~

Figure 4: Mean expression values by number of days, by gene (faceted).

## Example 8: Creating a New Variable with Mutate

1. Pull up the mean_exp_by_time data
2. What would we need to do if we would also like to split the line in each plot by the sex of the mice?
3. Grouped the data by `gene`, `time`, and `sex` and save this to a variable called `mean_exp`
4. Calculate the mean expression in the data frame, with a `mean_exp` variable
5. Save this to a new object called `mean_exp_by_time_sex`

### Example 8: Creating a New Variable with Mutate (Solution)

~~~
```{r}
mean_exp_by_time_sex <- sub_rna %>%
  group_by(gene, time, sex) %>%
  summarize(mean_exp = mean(expression_log))
```
~~~

### Example 9: Facet with a Grouping

We can now make the faceted plot by splitting further by sex using `color` (within a single plot).

1. Use the `mean_exp_by_time_sex` object
2. Map x to time, y to the mean expression values
3. Color by `sex`
4. Create a line graph
5. Facet on gene

### Example 9: Facet with a Grouping (Solution)

~~~
```{r}
mean_exp_by_time_sex %>%
ggplot(mapping = aes(x = time, y = mean_exp, color = sex)) +
    geom_line() +
    facet_wrap(vars(gene))
```
~~~

Figure 5: Mean expression values by number of days, by gene and my sex (faceted).

## Customizing our Plot

### What Are Our Options?

- You have probably noticed that the plots that we have been developing lack contextual elements like titles and annotations
- Default settings might be OK when you are analyzing data for yourself
- However, when it is time to publish, you will want to improve the readability of your plots by adding labels and annotations (Yau, 2024)
- I create my plots in R and then use different options for optimizing the plot
- However, depending on the publication you might need to refine your plot in a tool like Adobe Illustrator or Adobe InDesign (Yau, 2024)

### What Are Our Options in R?

- There are a number ways we can customize our last plot, `mean_exp_by_time_sex`
- We can change names of axes to something more informative than `time` and `mean_exp`, and add a title to the plot
- Before we do that, let us save the base plot to an object `mean_expression_plot`

### Annotation Options

- The easiest place to start when turning an exploratory graphic into an expository graphic is with good labels (Wickham et al., 2023a)
- You add labels with the `labs()` function, which allows you to add:
  - x and y labels
  - Label to the legend
  - A title, subtitle, and caption

### Example 10: Saving Our Plot Object

1. Create a new object called `mean_expression_plot`
2. Using the `mean_exp_by_time_sex` object that we previously create a ggplot statement with:
   1. Time on the x variable
   2. Mean expression on the y variable
   3. Group by the sex of the mouse, using color

### Example 10: Saving Our Plot Object (Solution)

~~~
```{r}
mean_expression_plot <- mean_exp_by_time_sex %>%
  ggplot(mapping = aes(x = time,
                     y = mean_exp,
                     color = sex))
```
~~~

## Adding Labels to Our Plot

### Example 11: Adding Labels to a Plot

1. Use the `mean_expression_plot` object
2. Create a line plot
3. Add a plot title: “Mean gene expression by duration of the infection”
4. Add a title for the x-axis: “Duration of the infection (in days)”
5. Add a title for the y-axis: “Mean gene expression”
6. Customize the legend title: “Sex”

### Example 11: Adding Labels to a Plot (Solution)

~~~
```{r}
mean_expression_plot +
      geom_line() +
    facet_wrap(vars(gene)) +
    labs(title = "Mean gene expression by duration of the infection",
         x = "Duration of the infection (in days)",
         y = "Mean gene expression",
         color = "Sex")
```
~~~

Figure 6: Mean expression values by number of days, by gene and my sex (faceted)

### A Note About Plot Titles

- Please note that the purpose of a plot title is to summarize the main finding
- Avoid titles that just describe what the plot is, e.g., "A scatterplot of engine displacement vs. fuel economy" (Wickham et al., 2023b)

## Using Color

### Colors in Visualization

Another scale that is frequently customized is color. There are three fundamental use cases for color in data visualizations (Wilke, 2019a):

- Distinguish discrete items or groups that do not have an intrinsic order, such as different countries on a map, or genes in a dataset
- Represent data values, such as income, temperature, or expression values
- Highlight specific categories or values in the dataset that carry key information about the story
- The types of colors we use and the way in which we use them are quite different for these three cases

### RColorBrewer

- ggplot package has a list of default colors that it uses for the elements in a plot depending on the number of total elements
- Default categorical scale picks colors that are evenly spaced around the color wheel
- RColorBrewer scales which have been hand tuned to work better for people with common types of color blindness
- efg's R Notes: RColorBrewer Package page has an overview of RColorBrewer palettes

### Example 12: Modifying Default Colors

1. Modify the `mean_expression_plot`, by using a color palette from the ColorBrewer package
2. Using efg's R Notes: RColorBrewer Package page, which type of palette will we use?
    1. Qualitative palettes are used to create the primary visual differences between classes
3. Modify the plot by using the **Set1** color palette from the ColorBrewer package
4. Data Novia has a useful help page that covers most of the basic function in this package
5. See if you can figure out which function to use
6. Two color scale functions are available in ggplot2 for using the RColorBrewer palettes:

- `scale_fill_brewer()` for box plot, bar plot, violin plot, dot plot, etc
- `scale_color_brewer()` for lines and points

### Example 12 (Solution)

~~~
```{r}
mean_expression_plot +
      geom_line() +
    facet_wrap(vars(gene)) +
    labs(title = "Mean gene expression by duration of the infection",
         x = "Duration of the infection (in days)",
         y = "Mean gene expression",
         color = "Sex") +
  scale_color_brewer(palette = "Set1")
```
~~~

Figure 7: Mean gene expression by duration in days, using RColorBrewer palette.

## Themes

### Five Mapping Components in ggplot

There are five mapping components that comprise the grammar of graphics in ggplot (Wickham et al., 2023d):

- Layer: collection of geometric elements and statistical transformations
- Scales: map values in the data space to values in the aesthetic space
- Coord, or coordinate system: describes how data coordinates are mapped to the plane of the graphic
- Facet specifies how to break up and display subsets of data as small multiples
- Theme controls the finer points of display, like the font size and background color

### ggplot Theme System

The ggplot theming system is composed of four main components (Wickham et al., 2022):

- `theme()` function: allows you to override the default theme elements by calling element functions
- Theme elements: specify the non-data elements that you can control. For example, the `plot.title` element controls the appearance of the plot title
- Theme element function: used with `theme_element()` describes the visual properties of the element. For example, `element_text()` sets the font size, color and face of text elements
- Complete themes: set all of the theme elements to values designed to work together harmoniously

### Legend Position

- Overall display of the plot legend is controlled through the theme system
- We are going to explore this in more detail in the next section
- For now we only need to know how to modify theme settings with the `theme()` function
  - These settings control the non-data parts of the plot
- Position and justification of legends are controlled by the theme setting `legend.position`, which takes values “right”, “left”, “top”, “bottom”, or “none” (no legend)

### Example 13: Legend Position

1. Use the `mean_expression_plot` that we created
2. Create a line plot
3. Facet on the gene variable
4. Use this same title and labels for the x and y axis that we used in the previous graph
5. Position the legend on the bottom of our graph

### Example 13: Legend Position (Solution)

~~~
```{r}
mean_expression_plot +
      geom_line() +
    facet_wrap(vars(gene)) +
    labs(title = "Mean gene expression by duration of the infection",
         x = "Duration of the infection (in days)",
         y = "Mean gene expression",
         color = "Sex") +
  theme(legend.position = "bottom")
```
~~~

Figure 8: Mean gene expression by duration in days, using legend position.

## Using the Guides Functions

### Guides

- Collectively axes and legends are called **guides**
- Axes are used for x and y aesthetics; legends are used for everything else (Wickham et al., 2023b)
- Breaks controls the position of the ticks, or the values associated with the keys
  - Most common use of `breaks()` is to override the default choice of a geom
  - For example using `scale_y_continuous(breaks = seq(15, 40, by = 5))` to override the default options for the labels on the y-axis
- Labels controls the text label associated with each tick/key

### Using the Guides to Control Legend Position

- One reason to change the legend appearance without changing the plot is to make the legend more readable
- Control the display of individual legends with `guides()` along with `guide_legend()` or `guide_colorbar()`
- `guide_legend()` settings `nrow` controls the number of rows the legend uses, and `override.aes` argument takes a list of aesthetic parameters that will *override* the default legend appearance

### Example 14: Modifying Legend Position with Guides

1. Copy the code that we used to created the previous plot
2. Use the `guides()` function with `guide_legend` to set the color legend to occupy 2 rows
3. Use the `guides()` function with `override.aes` to set the legend font size to 2

### Example 14: Modifying Legend Position with Guides (Solution)

~~~
```{r}
mean_expression_plot +
      geom_line() +
    facet_wrap(vars(gene)) +
    labs(title = "Mean gene expression by duration of the infection",
         x = "Duration of the infection (in days)",
         y = "Mean gene expression",
         color = "Sex") +
  theme(legend.position = "bottom") +
  guides(color = guide_legend(nrow = 2, override.aes = list(size = 2)))
```
~~~

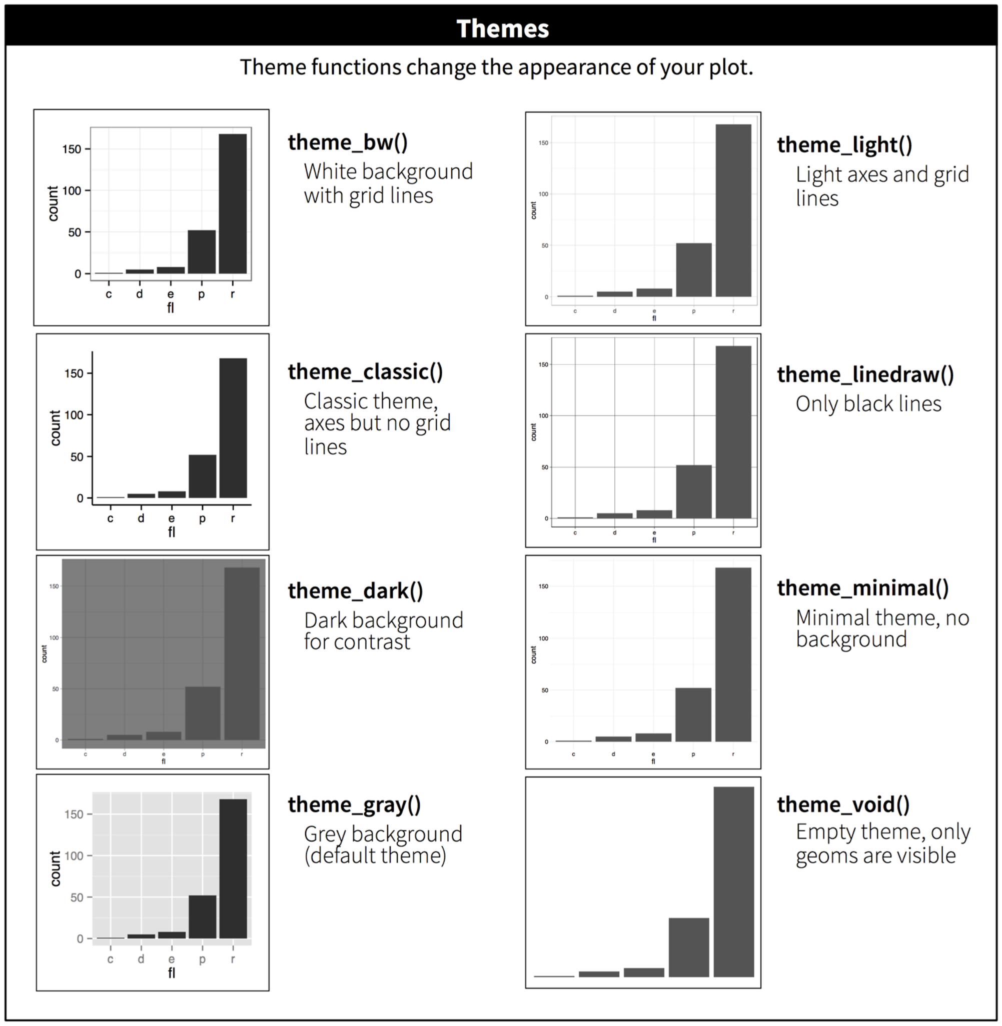
Figure 9: Mean gene expression by duration in days, using legend position.

## ggplot Themes

### ggplot Default Themes

- ggplot2 includes the eight themes (Figure 10)
- `theme_gray()` is the default
- Add-on packages like ggthemes include additional themes
- You can also create your own themes, if you are trying to match a particular corporate or journal style (Wickham et al., 2023b)
- Overview of all `theme()` components, using `?theme`
- ggplot2 book is also a great place to go for the full details on ggplot themes

## ggplot Default Themes



**Themes**

Theme functions change the appearance of your plot.

**theme_bw()**
White background with grid lines

**theme_light()**
Light axes and grid lines

**theme_classic()**
Classic theme, axes but no grid lines

**theme_linedraw()**
Only black lines

**theme_dark()**
Dark background for contrast

**theme_minimal()**
Minimal theme, no background

**theme_gray()**
Grey background (default theme)

**theme_void()**
Empty theme, only geoms are visible

*Figure 10: The eight themes built-in to ggplot2.*

## Example 15: Applying a ggplot default theme

1. Search RStudio Help for information about the built-in themes, by typing in `theme_`
2. Copy the code from the basic `mean_expression_plot` faceted line plot we created
3. Apply a theme you like to this plot
4. How does this compare with other plots that we created?

## Example 15: Applying a ggplot default theme (Solution)

~~~
```{r}
mean_expression_plot +
      geom_line() +
    facet_wrap(vars(gene)) +
    labs(title = "Mean gene expression by duration of the infection",
         x = "Duration of the infection (in days)",
         y = "Mean gene expression",
~~~

```
        color = "Sex") +
theme_light()
```

## References

Wickham, H. (2023). Differences between the base r and magrittr pipes. In *Tidyverse Blog*. https://www.tidyverse.org/blog/2023/04/base-vs-magrittr-pipe/

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023a). Communicate. In *R for data science : Import, tidy, transform, visualize, and model data* (2nd edition, pp. 505–530). O'Reilly Media, Inc.

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023b). Communication. In *R for data science : Import, tidy, transform, visualize, and model data* (2nd edition, pp. 169–201). O'Reilly Media, Inc.

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023c). Dates and Time. In *R for data science : Import, tidy, transform, visualize, and model data* (2nd edition, pp. 297–319). O'Reilly Media, Inc.

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023d). Layers. In *R for data science : Import, tidy, transform, visualize, and model data* (2nd edition, pp. 117–144). O'Reilly Media, Inc.

Wickham, H., Navarro, D., & Pedersen, T. L. (2022). *Themes* (3rd ed.). https://ggplot2-book.org/themes#introduction

Wilke, C. (2019a). Color Scales. In *Fundamentals of data visualization: A primer on making informative and compelling figures* (First edition., pp. 27–36). O'Reilly Media.

Wilke, C. (2019b). Visualizing data: Mapping data onto aesthetics. In *Fundamentals of data visualization: A primer on making informative and compelling figures* (First edition., pp. 7–12). O'Reilly Media.

Yau, N. (2024). Choosing Tools to Visualize Data. In *Visualize This: The flowingdata guide to design, visualization, and statistics second edition*. John Wiley and Sons.