February 3, 2023

The results below are generated from an R script.

```
---
title: Data visualization with ggplot2
author: Data Carpentry contributors
minutes: 60
editor_options:
  chunk_output_type: console
---


```{r setup, echo=FALSE, message = FALSE, warning = FALSE, purl=FALSE}
source("setup.R")
surveys_complete <- read_csv(file = "data_raw/portal_data_joined.csv", col_types = cols()) %>%
  drop_na(weight, hindfoot_length, sex) %>%
  group_by(species_id) %>%
  filter(n() >= 50) %>%
  ungroup()
```

```{r, echo=FALSE, purl=TRUE}
### Data Visualization with ggplot2
```

## Error:  <text>:2:13:  unexpected symbol
## 1:  --
## 2:  title:  Data visualization
##                ^
```

```
> ### Learning Objectives
>
> * Produce scatter plots, boxplots, and time series plots using ggplot.
> * Set universal plot settings.
> * Describe what faceting is and apply faceting in ggplot.
> * Modify the aesthetics of an existing ggplot plot (including axis labels and color).
> * Build complex and customized plots from data in a data frame.

## Error:  <text>:2:1:  unexpected '>'
## 1:
## 2:  >
##     ^
```

We start by loading the required packages. **‘ggplot2‘** is included in the **‘tidyverse‘** package.

```{r load-package, message=FALSE, purl=FALSE}
library(tidyverse)
```

If not still in the workspace, load the data we saved in the previous lesson.

```{r load-data, eval = FALSE, purl = FALSE}
surveys_complete <- read_csv("data/surveys_complete.csv")
```

## Plotting with **‘ggplot2‘**

**‘ggplot2‘** is a plotting package that provides helpful commands to create complex plots from data in a data frame. It provides a more programmatic interface for specifying what variables to plot, how they are displayed, and general visual properties. Therefore, we only need minimal changes if the underlying data change or if we decide to change from a bar plot to a scatterplot. This helps in creating publication quality plots with minimal amounts of adjustments and tweaking.

**‘ggplot2‘** refers to the name of the package itself. When using the package we use the function **‘ggplot()‘** to generate the plots, and so references to using the function will be referred to as **‘ggplot()‘** and the package as a whole as **‘ggplot2‘**

**‘ggplot2‘** plots work best with data in the ’long’ format, i.e., a column for every variable, and a row for every observation. Well-structured data will save you lots of time when making figures with **‘ggplot2‘**

ggplot graphics are built layer by layer by adding new elements. Adding layers in this fashion allows for extensive flexibility and customization of plots.

To build a ggplot, we will use the following basic template that can be used for different types of plot

```
ggplot(data = <DATA>, mapping = aes(<MAPPINGS>)) +  <GEOM_FUNCTION>()
```

- use the ‘ggplot()‘ function and bind the plot to a specific data frame using the ‘data‘ argument

```{r, eval = FALSE, purl = FALSE}
ggplot(data = surveys_complete)
```

- define an aesthetic mapping (using the aesthetic (‘aes‘) function), by selecting the variables to be plotted and specifying how to present them in the graph, e.g., as x/y positions or characteristics such as size, shape, color, etc.

```{r, eval = FALSE, purl = FALSE}
ggplot(data = surveys_complete, mapping = aes(x = weight, y = hindfoot_length))
```

```
```

- add 'geoms' - graphical representations of the data in the plot (points,
  lines, bars). **`ggplot2`** offers many different geoms; we will use some
  common ones today, including:

  * `geom_point()` for scatter plots, dot plots, etc.
  * `geom_boxplot()` for, well, boxplots!
  * `geom_line()` for trend lines, time series, etc.

To add a geom to the plot use `+` operator. Because we have two continuous
variables, let's use `geom_point()` first:

```{r first-ggplot, purl = FALSE}
ggplot(data = surveys_complete, aes(x = weight, y = hindfoot_length)) +
  geom_point()
```

The `+` in the **`ggplot2`** package is particularly useful because it allows
you to modify existing `ggplot` objects. This means you can easily set up plot
"templates" and conveniently explore different types of plots, so the above
plot can also be generated with code like this:

```{r, first-ggplot-with-plus, eval = FALSE, purl = FALSE}
# Assign plot to a variable
surveys_plot <- ggplot(data = surveys_complete,
                       mapping = aes(x = weight, y = hindfoot_length))

# Draw the plot
surveys_plot +
    geom_point()
```

```{r, eval = FALSE, purl = TRUE, echo = FALSE, purl = FALSE}
## Create a ggplot and draw it.
surveys_plot <- ggplot(data = surveys_complete,
                       aes(x = weight, y = hindfoot_length))

surveys_plot +
  geom_point()
```

**Notes**

- Anything you put in the `ggplot()` function can be seen by any geom layers
  that you add (i.e., these are universal plot settings). This includes the x-
  and y-axis you set up in `aes()`.
- You can also specify aesthetics for a given geom independently of the
  aesthetics defined globally in the `ggplot()` function.
- The `+` sign used to add layers must be placed at the end of each line
  containing a layer. If, instead, the `+` sign is added in the line before the
  other layer, **`ggplot2`** will not add the new layer and will return an error
  message.

```
- You may notice that we sometimes reference 'ggplot2' and sometimes 'ggplot'.
  To clarify, 'ggplot2' is the name of the most recent version of the package.
  However, any time we call the function itself, it's just called 'ggplot'.
- The previous version of the **`ggplot2`** package, called **`ggplot`**,
  which also contained the `ggplot()` function is now unsupported and has
  been removed from CRAN in order to reduce accidental installations
  and further confusion.

```{r, ggplot-with-plus-position, eval=FALSE, purl=FALSE}
# This is the correct syntax for adding layers
surveys_plot +
  geom_point()

# This will not add the new layer and will return an error message
surveys_plot
  + geom_point()
```

> ### Challenge (optional)
>
> Scatter plots can be useful exploratory tools for small datasets. For data
> sets with large numbers of observations, such as the `surveys_complete` data
> set, overplotting of points can be a limitation of scatter plots. One strategy
> for handling such settings is to use hexagonal binning of observations. The
> plot space is tessellated into hexagons. Each hexagon is assigned a color
> based on the number of observations that fall within its boundaries.  To use
> hexagonal binning with **`ggplot2`**, first install the R package `hexbin`
> from CRAN:
>
>
> ```{r, eval = FALSE}
> install.packages("hexbin")
> library(hexbin)
> ```
>
> Then use the `geom_hex()` function:
>
> ```{r, eval = FALSE}
> surveys_plot +
>  geom_hex()
> ```
>
> - What are the relative strengths and weaknesses of a hexagonal bin plot
>   compared to a scatter plot? Examine the above scatter plot and compare it
>   with the hexagonal bin plot that you created.

```{r hexbin-challenge, echo = FALSE, eval = FALSE, purl = TRUE}
### Challenge with hexbin
##
## To use the hexagonal binning with **`ggplot2`**, first install the `hexbin`
## package from CRAN:

install.packages("hexbin")
library(hexbin)
```
```

```
## Then use the `geom_hex()` function:

surveys_plot +
    geom_hex()

## What are the relative strengths and weaknesses of a hexagonal bin
## plot compared to a scatter plot?
```

## Building your plots iteratively

Building plots with **`ggplot2`** is typically an iterative process. We start by defining the dataset we'll use, lay out the axes, and choose a geom:

```{r create-ggplot-object, purl = FALSE}
ggplot(data = surveys_complete, aes(x = weight, y = hindfoot_length)) +
    geom_point()
```

Then, we start modifying this plot to extract more information from it. For instance, we can add transparency (`alpha`) to avoid overplotting:

```{r adding-transparency, purl = FALSE}
ggplot(data = surveys_complete, aes(x = weight, y = hindfoot_length)) +
    geom_point(alpha = 0.1)
```

We can also add colors for all the points:

```{r adding-colors, purl=FALSE}
ggplot(data = surveys_complete, mapping = aes(x = weight, y = hindfoot_length)) +
    geom_point(alpha = 0.1, color = "blue")
```

Or to color each species in the plot differently, you could use a vector as an input to the argument **c

```{r color-by-species-1, purl=FALSE}
ggplot(data = surveys_complete, mapping = aes(x = weight, y = hindfoot_length)) +
    geom_point(alpha = 0.1, aes(color = species_id))
```

> ### Challenge
>
> Use what you just learned to create a scatter plot of `weight` over
> `species_id` with the plot types showing in different colors.
> Is this a good way to show this type of data?
>
> ```{r scatter-challenge-answer, answer=TRUE, purl=FALSE}
> ggplot(data = surveys_complete,
>        mapping = aes(x = species_id, y = weight)) +
>    geom_point(aes(color = plot_type))
> ```

```{r scatter-challenge, echo = FALSE, eval = FALSE, purl = TRUE}
### Challenge with scatter plot:
##
##  Use what you just learned to create a scatter plot of `weight`
## over `species_id` with the plot types showing in different colors.
## Is this a good way to show this type of data?
```

## Boxplot

We can use boxplots to visualize the distribution of weight within each species:

```{r boxplot, purl=FALSE}
ggplot(data = surveys_complete, mapping = aes(x = species_id, y = weight)) +
    geom_boxplot()
```

By adding points to the boxplot, we can have a better idea of the number of
measurements and of their distribution:

```{r boxplot-with-points, purl=FALSE}
ggplot(data = surveys_complete, mapping = aes(x = species_id, y = weight)) +
    geom_boxplot(alpha = 0) +
    geom_jitter(alpha = 0.3, color = "tomato")
```

Notice how the boxplot layer is behind the jitter layer? What do you need to
change in the code to put the boxplot in front of the points such that it's not
hidden?

> ### Challenges
>
> Boxplots are useful summaries, but hide the *shape* of the distribution. For
> example, if there is a bimodal distribution, it would not be observed with a
> boxplot. An alternative to the boxplot is the violin plot (sometimes known as
> a beanplot), where the shape (of the density of points) is drawn.
>
> - Replace the box plot with a violin plot; see `geom_violin()`.
>
> In many types of data, it is important to consider the *scale* of the
> observations.  For example, it may be worth changing the scale of the axis to
> better distribute the observations in the space of the plot.  Changing the scale
> of the axes is done similarly to adding/modifying other components (i.e., by
> incrementally adding commands). Try making these modifications:
>
> - Represent weight on the log~10~ scale; see `scale_y_log10()`.
>
> So far, we've looked at the distribution of weight within species. Try making
> a new plot to explore the distribution of another variable within each species.
>
> - Create boxplot for `hindfoot_length`.  Overlay the boxplot layer on a jitter
>   layer to show actual measurements.

```
>
> - Add color to the data points on your boxplot according to the plot from which
>   the sample was taken (`plot_id`).

> Hint: Check the class for `plot_id`. Consider changing the class of `plot_id`
> from integer to factor. Why does this change how R makes the graph?

```{r boxplot-challenge, eval = FALSE, purl = TRUE, echo = FALSE}
## Challenge with boxplots:
##  Start with the boxplot we created:
ggplot(data = surveys_complete, mapping = aes(x = species_id, y = weight)) +
  geom_boxplot(alpha = 0) +
  geom_jitter(alpha = 0.3, color = "tomato")

##  1. Replace the box plot with a violin plot; see `geom_violin()`.

##  2. Represent weight on the log10 scale; see `scale_y_log10()`.

##  3. Create boxplot for `hindfoot_length` overlaid on a jitter layer.

##  4. Add color to the data points on your boxplot according to the
##  plot from which the sample was taken (`plot_id`).
##  *Hint:* Check the class for `plot_id`. Consider changing the class
##  of `plot_id` from integer to factor. Why does this change how R
##  makes the graph?
```


## Plotting time series data

Let's calculate number of counts per year for each genus. First we need
to group the data and count records within each group:

```{r, purl=FALSE}
yearly_counts <- surveys_complete %>%
  count(year, genus)
```

Timelapse data can be visualized as a line plot with years on the x-axis and
counts on the y-axis:

```{r first-time-series, purl = FALSE}
ggplot(data = yearly_counts, aes(x = year, y = n)) +
     geom_line()
```

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

```{r time-series-by-species, purl = FALSE}
ggplot(data = yearly_counts, aes(x = year, y = n, group = genus)) +
    geom_line()
```
```

```

We will be able to distinguish genera in the plot if we add colors (using
`color` also automatically groups the data):

```{r time-series-with-colors, purl = FALSE}
ggplot(data = yearly_counts, aes(x = year, y = n, color = genus)) +
    geom_line()
```


## Integrating the pipe operator with ggplot2

In the previous lesson, we saw how to use the pipe operator `%>%` to use
different functions in a sequence and create a coherent workflow.
We can also use the pipe operator to pass the `data` argument to the
`ggplot()` function. The hard part is to remember that to build your ggplot,
you need to use `+` and not `%>%`.

```{r integrating-the-pipe, purl=FALSE}
yearly_counts %>%
    ggplot(mapping = aes(x = year, y = n, color = genus)) +
    geom_line()
```

The pipe operator can also be used to link data manipulation with consequent data visualization.

```{r pipes-and-manipulation, purl=FALSE}
yearly_counts_graph <- surveys_complete %>%
    count(year, genus) %>%
    ggplot(mapping = aes(x = year, y = n, color = genus)) +
    geom_line()

yearly_counts_graph
```


## Faceting

`ggplot` has a special technique called *faceting* that allows the user to split
one plot into multiple plots based on a factor included in the dataset. We will
use it to make a time series plot for each genus:

```{r first-facet, purl = FALSE}
ggplot(data = yearly_counts, aes(x = year, y = n)) +
    geom_line() +
    facet_wrap(facets = vars(genus))
```

Now we would like to split the line in each plot by the sex of each individual
measured. To do that we need to make counts in the data frame grouped by `year`,
`genus`, and `sex`:
```

```{r, purl = FALSE}
 yearly_sex_counts <- surveys_complete %>%
                      count(year, genus, sex)
```

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

```{r facet-by-genus-and-sex, purl=FALSE}
ggplot(data = yearly_sex_counts, mapping = aes(x = year, y = n, color = sex)) +
  geom_line() +
  facet_wrap(facets =  vars(genus))
```

We can also facet both by sex and genus:

```{r average-weight-time-facet-both, purl=FALSE, fig.width=9.5}
ggplot(data = yearly_sex_counts,
       mapping = aes(x = year, y = n, color = sex)) +
  geom_line() +
  facet_grid(rows = vars(sex), cols =  vars(genus))
```

You can also organise the panels only by rows (or only by columns):

```{r average-weight-time-facet-sex-rows, purl=FALSE, fig.height=8.5, fig.width=8}
# One column, facet by rows
ggplot(data = yearly_sex_counts,
       mapping = aes(x = year, y = n, color = sex)) +
  geom_line() +
  facet_grid(rows = vars(genus))
```

```{r average-weight-time-facet-sex-columns, purl=FALSE, fig.width=9.5, fig.height=5}
# One row, facet by column
ggplot(data = yearly_sex_counts,
       mapping = aes(x = year, y = n, color = sex)) +
  geom_line() +
  facet_grid(cols = vars(genus))
```

**Note:**
`ggplot2` before version 3.0.0 used formulas to specify how plots are faceted.
If you encounter `facet_grid`/`wrap(...)` code containing `~`, please read
<https://ggplot2.tidyverse.org/news/#tidy-evaluation>.

## **`ggplot2`** themes

Usually plots with white background look more readable when printed.
Every single component of a `ggplot` graph can be customized using the generic
`theme()` function, as we will see below. However, there are pre-loaded themes

````
available that change the overall appearance of the graph without much effort.

For example, we can change our previous graph to have a simpler white background
using the `theme_bw()` function:

```{r facet-by-species-and-sex-white-bg, purl=FALSE}
 ggplot(data = yearly_sex_counts,
        mapping = aes(x = year, y = n, color = sex)) +
     geom_line() +
     facet_wrap(vars(genus)) +
     theme_bw()
```

In addition to `theme_bw()`, which changes the plot background to white, **`ggplot2`**
comes with several other themes which can be useful to quickly change the look
of your visualization. The complete list of themes is available
at <https://ggplot2.tidyverse.org/reference/ggtheme.html>. `theme_minimal()` and
`theme_light()` are popular, and `theme_void()` can be useful as a starting
point to create a new hand-crafted theme.

The
[ggthemes](https://jrnold.github.io/ggthemes/reference/index.html) package
provides a wide variety of options.

> ### Challenge
>
> Use what you just learned to create a plot that depicts how the average weight
> of each species changes through the years.
>
> ```{r average-weight-time-series, answer=TRUE, purl=FALSE}
> yearly_weight <- surveys_complete %>%
>                 group_by(year, species_id) %>%
>                  summarize(avg_weight = mean(weight))
> ggplot(data = yearly_weight, mapping = aes(x=year, y=avg_weight)) +
>    geom_line() +
>    facet_wrap(vars(species_id)) +
>    theme_bw()
> ```


```{r, eval = FALSE, purl = TRUE, echo = FALSE}
### Plotting time series challenge:
##
##  Use what you just learned to create a plot that depicts how the
##  average weight of each species changes through the years.

```


## Customization

Take a look at the [**`ggplot2`** cheat sheet](https://raw.githubusercontent.com/rstudio/cheatsheets/mai
think of ways you could improve the plot.
````

````
Now, let's change names of axes to something more informative than 'year'
and 'n' and add a title to the figure:

```{r number-species-year-with-right-labels, purl = FALSE}
ggplot(data = yearly_sex_counts, aes(x = year, y = n, color = sex)) +
    geom_line() +
    facet_wrap(vars(genus)) +
    labs(title = "Observed genera through time",
         x = "Year of observation",
         y = "Number of individuals") +
    theme_bw()
```

The axes have more informative names, but their readability can be improved by
increasing the font size. This can be done with the generic `theme()` function:

```{r number-species-year-with-right-labels-xfont-size, purl=FALSE}
ggplot(data = yearly_sex_counts, mapping = aes(x = year, y = n, color = sex)) +
    geom_line() +
    facet_wrap(vars(genus)) +
    labs(title = "Observed genera through time",
        x = "Year of observation",
        y = "Number of individuals") +
    theme_bw() +
    theme(text=element_text(size = 16))
```

Note that it is also possible to change the fonts of your plots. If you are on
Windows, you may have to install
the [**`extrafont`** package](https://github.com/wch/extrafont), and follow the
instructions included in the README for this package.

After our manipulations, you may notice that the values on the x-axis are still
not properly readable. Let's change the orientation of the labels and adjust
them vertically and horizontally so they don't overlap. You can use a 90 degree
angle, or experiment to find the appropriate angle for diagonally oriented
labels. We can also modify the facet label text (`strip.text`) to italicize the genus
names:

```{r number-species-year-with-theme, purl=FALSE}
ggplot(data = yearly_sex_counts, mapping = aes(x = year, y = n, color = sex)) +
    geom_line() +
    facet_wrap(vars(genus)) +
    labs(title = "Observed genera through time",
        x = "Year of observation",
        y = "Number of individuals") +
    theme_bw() +
    theme(axis.text.x = element_text(colour = "grey20", size = 12, angle = 90, hjust = 0.5, vjust = 0.5)
                        axis.text.y = element_text(colour = "grey20", size = 12),
                        strip.text = element_text(face = "italic"),
                        text = element_text(size = 16))
```

If you like the changes you created better than the default theme, you can save
````

````
them as an object to be able to easily apply them to other plots you may create:


```{r number-species-year-with-right-labels-xfont-orientation, purl = FALSE}
grey_theme <- theme(axis.text.x = element_text(colour="grey20", size = 12,
                                               angle = 90, hjust = 0.5,
                                               vjust = 0.5),
                    axis.text.y = element_text(colour = "grey20", size = 12),
                    text=element_text(size = 16))

ggplot(surveys_complete, aes(x = species_id, y = hindfoot_length)) +
    geom_boxplot() +
    grey_theme
```

> ### Challenge
>
> With all of this information in hand, please take another five minutes to either
> improve one of the plots generated in this exercise or create a beautiful graph
> of your own. Use the RStudio [**ggplot2** cheat sheet](https://www.rstudio.com/wp-content/uploads/20
> for inspiration.
>
> Here are some ideas:
>
> * See if you can change the thickness of the lines.
> * Can you find a way to change the name of the legend? What about its labels?
> * Try using a different color palette (see
>   <https://www.cookbook-r.com/Graphs/Colors_(ggplot2)/>).

## Arranging plots

Faceting is a great tool for splitting one plot into multiple plots, but
sometimes you may want to produce a single figure that contains multiple plots
using different variables or even different data frames. The **patchwork**
package allows us to combine separate ggplots into a single figure while keeping
everything aligned properly. Like most R packages, we can install `patchwork`
from CRAN, the R package repository:

```{r patchwork-install, eval = FALSE}
install.packages("patchwork")
```

After you have loaded the `patchwork` package you can use `+` to place plots
next to each other, `/` to arrange them vertically, and `plot_layout()` to
determine how much space each plot uses:

```{r patchwork-example, message = FALSE, purl = FALSE, fig.width = 10}
library(patchwork)

plot_weight <- ggplot(data = surveys_complete, aes(x = species_id, y = weight)) +
  geom_boxplot() +
  labs(x = "Species", y = expression(log[10](Weight))) +
  scale_y_log10()
````
````

```
plot_count <- ggplot(data = yearly_counts, aes(x = year, y = n, color = genus)) +
  geom_line() +
  labs(x = "Year", y = "Abundance")

plot_weight / plot_count + plot_layout(heights = c(3, 2))
```

You can also use parentheses `()` to create more complex layouts. There are
many useful examples on the [patchwork website](https://patchwork.data-imaginist.com/)

*## Exporting plots*

After creating your plot, you can save it to a file in your favorite format. The
Export tab in the **Plot** pane in RStudio will save your plots at low
resolution, which will not be accepted by many journals and will not scale well
for posters. The [**`ggplot2`** extensions website](https://exts.ggplot2.tidyverse.org/) provides a list
of packages that extend the capabilities of **`ggplot2`**, including additional
themes.

Instead, use the `ggsave()` function, which allows you to easily change the
dimension and resolution of your plot by adjusting the appropriate arguments
(`width`, `height` and `dpi`):

```{r ggsave-example, eval = FALSE, purl = FALSE}
my_plot <- ggplot(data = yearly_sex_counts,
                  aes(x = year, y = n, color = sex)) +
    geom_line() +
    facet_wrap(vars(genus)) +
    labs(title = "Observed genera through time",
        x = "Year of observation",
        y = "Number of individuals") +
    theme_bw() +
    theme(axis.text.x = element_text(colour = "grey20", size = 12, angle = 90,
                                     hjust = 0.5, vjust = 0.5),
          axis.text.y = element_text(colour = "grey20", size = 12),
          text = element_text(size = 16))

ggsave("name_of_file.png", my_plot, width = 15, height = 10)

*## This also works for plots combined with patchwork*
plot_combined <- plot_weight / plot_count + plot_layout(heights = c(3, 2))
ggsave("plot_combined.png", plot_combined, width = 10, dpi = 300)
```

Note: The parameters `width` and `height` also determine the font size in the
saved plot.

```{r final-challenge, eval = FALSE, purl = TRUE, echo = FALSE}
*### Final plotting challenge:*
*##  With all of this information in hand, please take another five*
*##  minutes to either improve one of the plots generated in this*
*##  exercise or create a beautiful graph of your own. Use the RStudio*
```

```
##  ggplot2 cheat sheet for inspiration:
##  https://www.rstudio.com/wp-content/uploads/2015/08/ggplot2-cheatsheet.pdf
```

```{r, child="_page_built_on.Rmd"}
```

```
## Error:  <text>:2:4:  unexpected symbol
## 1:
## 2:  We start
##       ^
```

The R session information (including the OS info, R version and all packages used):

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19045)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_New Zealand.utf8  LC_CTYPE=English_New Zealand.utf8
## [3] LC_MONETARY=English_New Zealand.utf8 LC_NUMERIC=C
## [5] LC_TIME=English_New Zealand.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] ggrepel_0.9.2      ggthemes_4.2.4     sf_1.0-9           leaflet_2.1.1
##  [5] fpp_0.5            tseries_0.10-53    lmtest_0.9-40      zoo_1.8-11
##  [9] expsmooth_2.3      fma_2.4            forecast_8.20      seasonal_1.9.0
## [13] lubridate_1.9.1    timetk_2.8.2       yardstick_1.1.0    workflowsets_1.0.0
## [17] workflows_1.1.2    tune_1.0.1         rsample_1.1.1      recipes_1.0.4
## [21] parsnip_1.0.3      modeldata_1.1.0    infer_1.0.4        dials_1.1.0
## [25] scales_1.2.1       broom_1.0.3        tidymodels_1.0.0   modeltime_1.2.4
## [29] flextable_0.8.5    XKCDdata_0.1.0     forcats_1.0.0      stringr_1.5.0
## [33] dplyr_1.1.0        purrr_1.0.1        readr_2.1.3        tidyr_1.3.0
## [37] tibble_3.1.8       ggplot2_3.4.0      tidyverse_1.3.2    knitr_1.42
##
## loaded via a namespace (and not attached):
##   [1] utf8_1.2.3           tidyselect_1.2.0     htmlwidgets_1.6.1    grid_4.2.1
##   [5] munsell_0.5.0        units_0.8-1          codetools_0.2-18     xgboost_1.7.3.1
##   [9] future_1.31.0        withr_2.5.0          colorspace_2.1-0     highr_0.10
##  [13] uuid_1.1-0           rstudioapi_0.14      stats4_4.2.1         wk_0.7.1
##  [17] officer_0.5.2        TTR_0.24.3           listenv_0.9.0        labeling_0.4.2
##  [21] rstan_2.21.8         DiceDesign_1.9       farver_2.1.1         parallelly_1.34.0
##  [25] vctrs_0.5.2          generics_0.1.3       ipred_0.9-13         xfun_0.37
##  [29] timechange_0.2.0     R6_2.5.1             lhs_1.1.6            cachem_1.0.6
##  [33] assertthat_0.2.1     promises_1.2.0.1     nnet_7.3-17          googlesheets4_1.0.1
##  [37] gtable_0.3.1         globals_0.16.2       processx_3.8.0       timeDate_4022.108
##  [41] rlang_1.0.6          systemfonts_1.0.4    splines_4.2.1        lazyeval_0.2.2
##  [45] gargle_1.3.0         inline_0.3.19        s2_1.1.2             yaml_2.3.7
```

```
##  [49] modelr_0.1.10        crosstalk_1.2.0      backports_1.4.1      httpuv_1.6.8
##  [53] quantmod_0.4.20      tools_4.2.1          lava_1.7.1           ellipsis_0.3.2
##  [57] proxy_0.4-27         Rcpp_1.0.10          base64enc_0.1-3      classInt_0.4-8
##  [61] ps_1.7.2             prettyunits_1.1.1    rpart_4.1.16         openssl_2.0.5
##  [65] fracdiff_1.5-2       haven_2.5.1          fs_1.6.0             tinytex_0.44
##  [69] furrr_0.3.1          crul_1.3             magrittr_2.0.3       data.table_1.14.6
##  [73] reprex_2.0.2         GPfit_1.0-8          googledrive_2.0.0    x13binary_1.1.57-3
##  [77] matrixStats_0.63.0   hms_1.1.2            mime_0.12            evaluate_0.20
##  [81] xtable_1.8-4         readxl_1.4.1         gridExtra_2.3        compiler_4.2.1
##  [85] KernSmooth_2.23-20   crayon_1.5.2         StanHeaders_2.21.0-7 htmltools_0.5.4
##  [89] later_1.3.0          tzdb_0.3.0           RcppParallel_5.1.6   DBI_1.1.3
##  [93] dbplyr_2.3.0         MASS_7.3-57          Matrix_1.5-3         cli_3.6.0
##  [97] quadprog_1.5-8       parallel_4.2.1       gower_1.0.1          pkgconfig_2.0.3
## [101] plotly_4.10.1        xml2_1.3.3           foreach_1.5.2        hardhat_1.2.0
## [105] prodlim_2019.11.13   rvest_1.0.3          snakecase_0.11.0     callr_3.7.3
## [109] digest_0.6.31        janitor_2.2.0        httpcode_0.3.0       rmarkdown_2.20
## [113] cellranger_1.1.0     gdtools_0.3.0        curl_5.0.0           shiny_1.7.4
## [117] urca_1.3-3           lifecycle_1.0.3      nlme_3.1-157         jsonlite_1.8.4
## [121] viridisLite_0.4.1    askpass_1.1          fansi_1.0.4          pillar_1.8.1
## [125] lattice_0.20-45      loo_2.5.1            fastmap_1.1.0        httr_1.4.4
## [129] pkgbuild_1.4.0       survival_3.3-1       glue_1.6.2           xts_0.12.2
## [133] zip_2.2.2            iterators_1.0.14     class_7.3-20         stringi_1.7.12
## [137] prophet_1.0          gfonts_0.2.0         memoise_2.0.1        e1071_1.7-13
## [141] future.apply_1.10.0
```

```
Sys.time()
```

```
## [1] "2023-02-03 13:27:05 NZDT"
```