Visualize your data by using ggplot2

2023-03-01

## Visualize your data by using R

There’s a wise saying that goes something like this: a picture is worth a thousand rows and columns.

(Actually, we made that up. There’s no such wise saying, but you get the gist, right?)

In this notebook, you’ll apply basic techniques to analyze data with basic statistics and visualize them by using graphs with ggplot2, a core member of the Tidyverse.

## Load your data

Before you begin, load the same data about study hours that you analyzed in the last notebook. You’ll also recalculate who passed your class, just as you did before. To see the data, run the code in the following cell by selecting the **► Run** button.

# Load the tidyverse and make it available in the current R session  
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.0 ✔ readr 2.1.4  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ ggplot2 3.4.1 ✔ tibble 3.1.8  
## ✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
## ✔ purrr 1.0.1   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force all conflicts to become errors

Now, let’s get the data ready for further analysis.

# Read a CSV file into a tibble  
df\_students <- read\_csv(file = "https://raw.githubusercontent.com/MicrosoftDocs/ml-basics/master/data/grades.csv")

## Rows: 24 Columns: 3  
## ── Column specification ────────────────────────────────────────────────────────  
## Delimiter: ","  
## chr (1): Name  
## dbl (2): StudyHours, Grade  
##   
## ℹ Use `spec()` to retrieve the full column specification for this data.  
## ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

# Remove any rows with missing data  
df\_students <- df\_students %>%   
 drop\_na()  
  
# Add a column "Pass" that specifies whether a student passed or failed  
# Assuming '60' is the grade needed to pass  
df\_students <- df\_students %>%   
 mutate(Pass = Grade >= 60)  
  
# Print the results  
df\_students

## # A tibble: 22 × 4  
## Name StudyHours Grade Pass   
## <chr> <dbl> <dbl> <lgl>  
## 1 Dan 10 50 FALSE  
## 2 Joann 11.5 50 FALSE  
## 3 Pedro 9 47 FALSE  
## 4 Rosie 16 97 TRUE   
## 5 Ethan 9.25 49 FALSE  
## 6 Vicky 1 3 FALSE  
## 7 Frederic 11.5 53 FALSE  
## 8 Jimmie 9 42 FALSE  
## 9 Rhonda 8.5 26 FALSE  
## 10 Giovanni 14.5 74 TRUE   
## # … with 12 more rows

Good job!

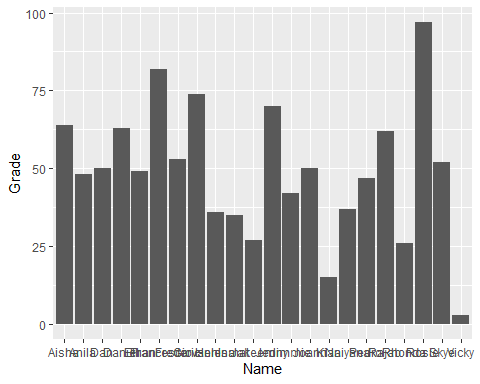
## Visualize data by using ggplot2

Data frames provide a great way to explore and analyze rectangular data, but sometimes plotting the data can greatly improve your ability to analyze the data, see underlying trends, and raise new questions.

Ggplot2 is a package for creating elegant graphics for data analysis in R. Compared with other graphing systems, ggplot2 provides a flexible and intuitive way of creating graphs by combining independent components of a graphic in a series of iterative steps. This allows you to create visualizations that match your needs rather than being limited to sets of predefined graphics.

Now let’s see this in action. Start with a simple bar chart that shows each student’s grade:

ggplot(data = df\_students) +  
 geom\_col(mapping = aes(x = Name, y = Grade))



Well, that worked. But the chart could use some improvements to help clarify what you’re looking at. We’ll get to that, but let’s first walk through the process of creating graphics in ggplot2.

You initialize a graphic by using the function ggplot() and the data frame to use for the plot. ggplot(data = df\_students) basically creates an empty graph to which you can add layers by using a plus sign (+).

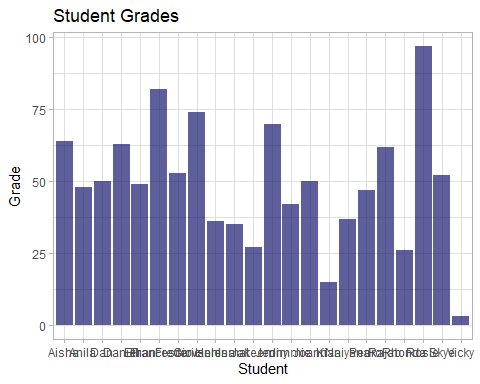
geom\_col() then adds a layer of bars whose height corresponds to the variables that are specified by the mapping argument. The mapping argument is always paired with aes(), which specifies how variables in the data are mapped. What goes into aes() are variables found in the data. In this case, you specified that you want to map Name to the x-axis and Grade to the y-axis.

And that’s it! You’ll follow and extend this blueprint to make different types of graphs.

Now, let’s improve the visual elements of the plot. For example, the following code:

* Specifies the color of the bar chart.
* Adds a title to the chart (so you know what it represents)
* Adds labels to the x-axis and y-axis (so you know which axis shows which data)

# Change the default grey background  
theme\_set(theme\_light())  
  
  
ggplot(data = df\_students) +  
 geom\_col(mapping = aes(x = Name, y = Grade),  
 # Specifiy color and transparency of the bars  
 fill = "midnightblue", alpha = 0.7) +  
 # Add a title to the chart  
 ggtitle("Student Grades") +  
 # Add labels to axes  
 xlab("Student") +  
 ylab("Grade")



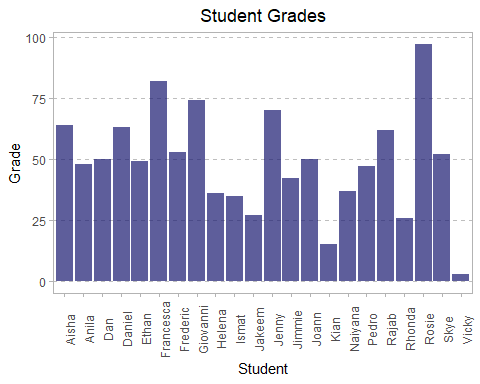
Whoa! That’s a step in the right direction. You can improve this even further using ggplot2’s comprehesive theming system. Themes are a powerful way to customize the non-data components of your plots. That is, you can add titles, labels, fonts, background, gridlines, and legends. You can learn more about modifying components of a theme by running ?theme.

For example, you can:

* Center the title
* Add a grid (to make it easier to determine the values for the bars)
* Rotate the x-axis markers (so you can read them)

ggplot(data = df\_students) +  
 geom\_col(mapping = aes(x = Name, y = Grade),  
 fill = "midnightblue", alpha = 0.7) +  
 ggtitle("Student Grades") +  
 xlab("Student") +  
 ylab("Grade") +  
 theme(  
 # Center the title  
 plot.title = element\_text(hjust = 0.5),  
   
 # Add a grid (to make it easier to determine the bar values  
 panel.grid = element\_blank(),  
 panel.grid.major.y = element\_line(color = "gray", linetype = "dashed", size = 0.5),  
   
 # Rotate the x-axis markers so you can read them  
 axis.text.x = element\_text(angle = 90)  
   
 )

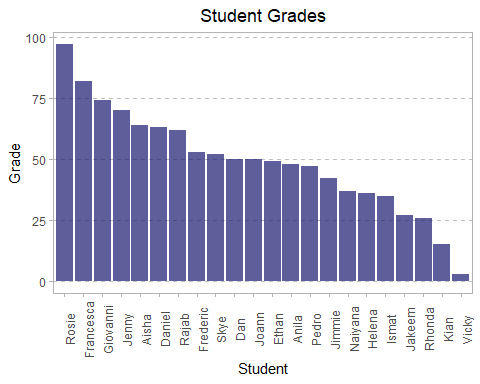
## Warning: The `size` argument of `element\_line()` is deprecated as of ggplot2 3.4.0.  
## ℹ Please use the `linewidth` argument instead.



Good job! Perhaps the bar chart would be more informative if the student names were in a certain order, right? This is a good chance to showcase how to use dplyr and ggplot2 to derive insights from your data.

So, let’s reorder the levels of the Name column in descending order, based on the Grade column, and then plot this.

df\_students %>%   
 mutate(Name = fct\_reorder(Name, Grade, .desc = TRUE)) %>%   
 ggplot() +  
 geom\_col(mapping = aes(x = Name, y = Grade),  
 fill = "midnightblue", alpha = 0.7) +  
 ggtitle("Student Grades") +  
 xlab("Student") +  
 ylab("Grade") +  
 theme(  
 plot.title = element\_text(hjust = 0.5),  
   
 panel.grid = element\_blank(),  
 panel.grid.major.y = element\_line(color = "gray", linetype = "dashed", size = 0.5),  
   
 axis.text.x = element\_text(angle = 90)  
   
 )



That’s a much better plot, both aesthetically and informationally. For example, you can quickly and easily discern how each student performed.

## Get started with statistical analysis

Now that you know how to use R to manipulate and visualize data, you can start analyzing it.

A lot of data science is rooted in *statistics*, so you’ll explore some basic statistical techniques.

**Note**: This exercise isn’t intended to teach you statistics. That’s much too big a topic for this notebook. But it will introduce you to some statistical concepts and techniques that data scientists use as they explore data in preparation for machine-learning modeling.

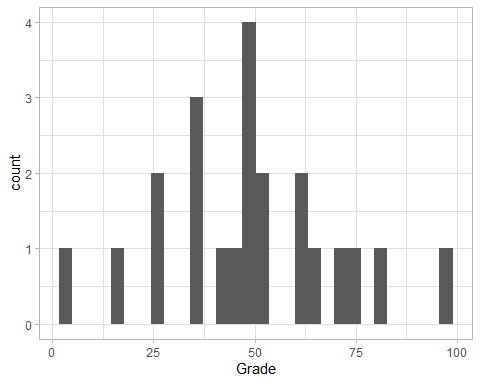
### Descriptive statistics and data distribution

When data scientists examine a *variable* (for example, a sample of student grades), they’re particularly interested in the variable’s *distribution*. That is, they want to know how the various grade values are spread across the sample. The starting point for this exploration is often to visualize the data as a histogram, and to see how frequently each variable value occurs.

So what geom are you going to use? You’ll say, ” geom\_histogram“, because you are already getting the hang of this.

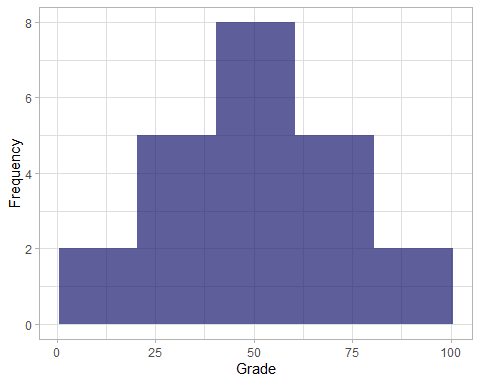
# Visualize distribution of the grades in a histogram  
ggplot(data = df\_students) +  
 geom\_histogram(mapping = aes(x = Grade))

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



All right, this certainly tells you something about your data. For example, most of the grades seem to be about 50. However, ggplot2 is extremely flexible and allows you to experiment with different function arguments to better reveal the story behind your data. By looking at ?geom\_histogram, you can experiment with arguments such as binwidth and boundary, as shown here:

# Visualize distribution of the grades in a histogram  
ggplot(data = df\_students) +  
 geom\_histogram(mapping = aes(x = Grade), , binwidth = 20, boundary = 0.5, fill = "midnightblue", alpha = 0.7) +  
 xlab('Grade') +  
 ylab('Frequency') +  
 theme(plot.title = element\_text(hjust = 0.5))



Much better! The histogram for grades is a symmetric shape, where the most frequently occurring grades tend to be in the middle of the range (about 50), with the least frequently occurring grades displayed at the extreme ends of the scale.

### Measures of central tendency

To understand the distribution better, you can examine so-called *measures of central tendency*, which is a fancy way of describing statistics that represent the *middle* of the data. The goal of this is to try to find a *typical* value. Common ways to define the middle of the data include:

* The *mean*: A simple average that’s calculated by adding all the values in the sample set, and then dividing the total by the number of samples.
* The *median*: The value in the middle of the range of all of the sample values.
* The *mode*: The most commonly occurring value in the sample set.
* **Note**: Of course, in some sample sets, there might be a tie for the most common value. When this occurs, the dataset is described as *bimodal* or even *multimodal*.
* Base R doesn’t provide a *function* for finding the *mode*. But worry not, statip::mfv returns the most frequent values or modes found in a vector. For other excellent workarounds, see [stackoverflow thread](https://stackoverflow.com/questions/2547402/how-to-find-the-statistical-mode).

Let’s calculate these values, along with the minimum and maximum values for comparison, and show them on the histogram.

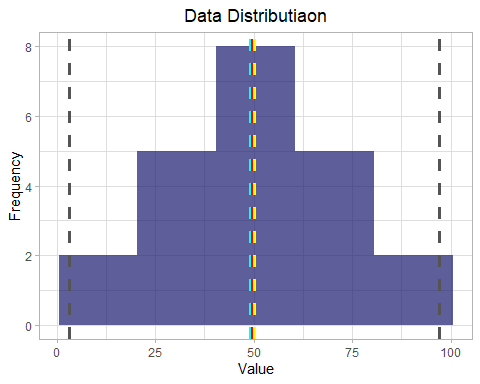
# Load statip into the current R sesssion  
library(statip)  
  
# Get summary statistics  
min\_val <- min(df\_students$Grade)  
max\_val <- max(df\_students$Grade)  
mean\_val <- mean(df\_students$Grade)  
med\_val <- median(df\_students$Grade)  
mod\_val <- mfv(df\_students$Grade)  
  
# Print the stats  
cat(  
 "Minimum: ", round(min\_val, 2),  
 "\nMean: ", round(mean\_val, 2),  
 "\nMedian: ", round(med\_val, 2),  
 "\nMode: ", round(mod\_val, 2),  
 "\nMaximum: ", round(max\_val, 2)  
)

## Minimum: 3   
## Mean: 49.18   
## Median: 49.5   
## Mode: 50   
## Maximum: 97

Now let’s incorporate these statistics in your graph:

# Plot a histogram  
ggplot(data = df\_students) +  
 geom\_histogram(mapping = aes(x = Grade), binwidth = 20, fill = "midnightblue", alpha = 0.7, boundary = 0.5) +  
   
# Add lines for the statistics  
 geom\_vline(xintercept = min\_val, color = 'gray33', linetype = "dashed", size = 1.3) +  
 geom\_vline(xintercept = mean\_val, color = 'cyan', linetype = "dashed", size = 1.3) +  
 geom\_vline(xintercept = med\_val, color = 'red', linetype = "dashed", size = 1.3 ) +  
 geom\_vline(xintercept = mod\_val, color = 'yellow', linetype = "dashed", size = 1.3 ) +  
 geom\_vline(xintercept = max\_val, color = 'gray33', linetype = "dashed", size = 1.3 ) +  
   
# Add titles and labels  
 ggtitle('Data Distributiaon')+  
 xlab('Value')+  
 ylab('Frequency')+  
 theme(plot.title = element\_text(hjust = 0.5))

## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
## ℹ Please use `linewidth` instead.



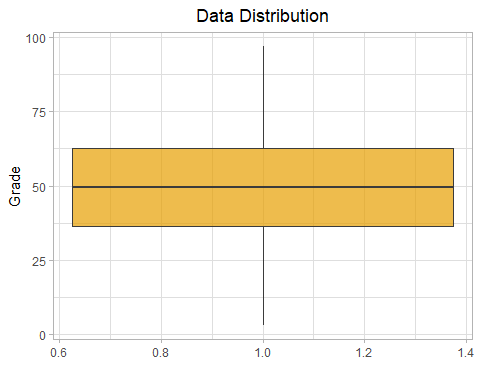
geom\_vline() adds a vertical reference line to a plot.

Good job!

For the grade data, the mean, median, and mode all seem to be more or less in the middle of the minimum and maximum, at around 50.

Another way to visualize the distribution of a variable is to use a *box* plot (sometimes called a *box-and-whiskers* plot). Let’s create one for the grade data.

# Plot a box plot  
ggplot(data = df\_students) +  
 geom\_boxplot(mapping = aes(x = 1, y = Grade), fill = "#E69F00", color = "gray23", alpha = 0.7) +  
   
 # Add titles and labels  
 ggtitle("Data Distribution") +  
 xlab("") +  
 ylab("Grade") +  
 theme(plot.title = element\_text(hjust = 0.5))



The box plot shows the distribution of the grade values in a different format than the histogram shows it. The *box* part of the plot shows where the inner two *quartiles* of the data reside. So, in this case, half of the grades are between approximately 36 and 63. The *whiskers* extending from the box show the outer two quartiles, so the other half of the grades in this case are between 0 and 36 or 63 and 100. The line in the box indicates the *median* value.

It’s often useful to combine histograms and box plots, with the box plot’s orientation changed to align it with the histogram. In some ways, it can be helpful to think of the histogram as a “front elevation” view of the distribution, and the box plot as a “plan” view of the distribution from above. Because you might need to plot histograms and box plots for different variables, it might also be convenient to write a function. With a function, you can automate common tasks in a more powerful and general way than by copying and pasting.

Let’s get right to it. Functions in R are generally defined in this fashion:

name <- function(variables) {return(value)}

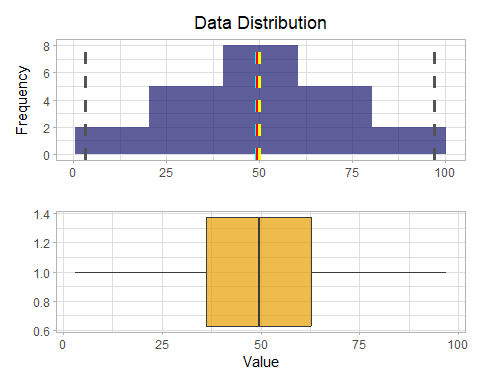
[patchwork](https://patchwork.data-imaginist.com/) extends the ggplot API by providing mathematical operators (such as + or /) for combining multiple plots. Yes, it’s as easy as that.

library(patchwork)  
# Create a function that you can reuse  
show\_distribution <- function(var\_data, binwidth) {  
   
 # Get summary statistics by first extracting values from the column  
 min\_val <- min(pull(var\_data))  
 max\_val <- max(pull(var\_data))  
 mean\_val <- mean(pull(var\_data))  
 med\_val <- median(pull(var\_data))  
 mod\_val <- statip::mfv(pull(var\_data))  
  
 # Print the stats  
 stats <- glue::glue(  
 'Minimum: {format(round(min\_val, 2), nsmall = 2)}  
 Mean: {format(round(mean\_val, 2), nsmall = 2)}  
 Median: {format(round(med\_val, 2), nsmall = 2)}  
 Mode: {format(round(mod\_val, 2), nsmall = 2)}  
 Maximum: {format(round(max\_val, 2), nsmall = 2)}'  
 )  
   
 # Plot the histogram  
 hist\_gram <- ggplot(var\_data) +  
 geom\_histogram(aes(x = pull(var\_data)), binwidth = binwidth,  
 fill = "midnightblue", alpha = 0.7, boundary = 0.4) +  
   
 # Add lines for the statistics  
 geom\_vline(xintercept = min\_val, color = 'gray33', linetype = "dashed", size = 1.3) +  
 geom\_vline(xintercept = mean\_val, color = 'cyan', linetype = "dashed", size = 1.3) +  
 geom\_vline(xintercept = med\_val, color = 'red', linetype = "dashed", size = 1.3 ) +  
 geom\_vline(xintercept = mod\_val, color = 'yellow', linetype = "dashed", size = 1.3 ) +  
 geom\_vline(xintercept = max\_val, color = 'gray33', linetype = "dashed", size = 1.3 ) +  
   
 # Add titles and labels  
 ggtitle('Data Distribution') +  
 xlab('')+  
 ylab('Frequency') +  
 theme(plot.title = element\_text(hjust = 0.5))  
   
 # Plot the box plot  
 bx\_plt <- ggplot(data = var\_data) +  
 geom\_boxplot(mapping = aes(x = pull(var\_data), y = 1),  
 fill = "#E69F00", color = "gray23", alpha = 0.7) +  
   
 # Add titles and labels  
 xlab("Value") +  
 ylab("") +  
 theme(plot.title = element\_text(hjust = 0.5))  
   
   
 # To return multiple outputs, use a list  
 return(  
   
 list(stats,  
 # Combine histogram and box plot using library patchwork  
 hist\_gram / bx\_plt)  
   
 ) # End of returned outputs  
   
} # End of function

Now that the show\_distribution() function is done, let’s get a variable column to examine and then call the function.

# Get the variable to examine  
col <- df\_students %>%   
 select(Grade)  
  
# Call the function  
show\_distribution(var\_data = col, binwidth = 20)

## [[1]]  
## Minimum: 3.00  
## Mean: 49.18  
## Median: 49.50  
## Mode: 50.00  
## Maximum: 97.00  
##   
## [[2]]



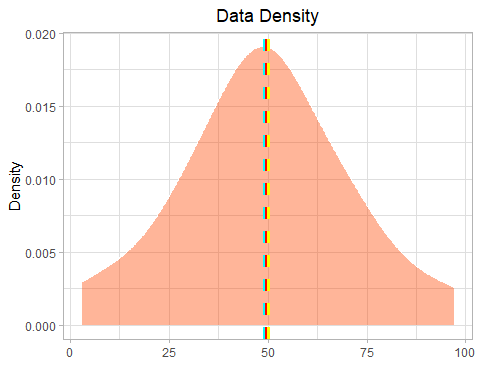
All the measurements of central tendency are right in the middle of the data distribution, which is symmetric with values becoming progressively lower in both directions from the middle.

To explore this distribution in more detail, you need to understand that statistics is fundamentally about taking *samples* of data and using probability functions to extrapolate information about the full *population* of data. For example, the student data consists of 22 samples, and for each sample there’s a grade value. You can think of each sample grade as a variable that’s been randomly selected from the set of all grades awarded for this course. With enough of these random variables, you can calculate something called a *probability density function*, which estimates the distribution of grades for the full population.

A density plot is a representation of the distribution of a numeric variable. It is a smoothed version of the histogram and is often used in the same kind of situation.

geom\_density() computes and draws a kernel density estimate, which is a smoothed version of the histogram.

# Create a function that returns a density plot  
show\_density <- function(var\_data) {  
   
 # Get statistics  
 mean\_val <- mean(pull(var\_data))  
 med\_val <- median(pull(var\_data))  
 mod\_val <- statip::mfv(pull(var\_data))  
   
   
 # Plot the density plot  
 density\_plot <- ggplot(data = var\_data) +  
 geom\_density(aes(x = pull(var\_data)), fill="orangered", color="white", alpha=0.4) +  
   
 # Add lines for the statistics  
 geom\_vline(xintercept = mean\_val, color = 'cyan', linetype = "dashed", size = 1.3) +  
 geom\_vline(xintercept = med\_val, color = 'red', linetype = "dashed", size = 1.3 ) +  
 geom\_vline(xintercept = mod\_val, color = 'yellow', linetype = "dashed", size = 1.3 ) +  
   
 # Add titles and labels  
 ggtitle('Data Density') +  
 xlab('') +  
 ylab('Density') +  
 theme(plot.title = element\_text(hjust = 0.5))  
   
   
   
 return(density\_plot) # End of returned outputs  
   
} # End of function  
  
  
# Get the density of Grade  
col <- df\_students %>% select(Grade)  
show\_density(var\_data = col)



As expected from the histogram of the sample, the density shows the characteristic “bell curve” of what statisticians call a *normal* distribution with the mean and mode at the center and symmetric tails.

## Summary

We’ve introduced quite a few new concepts here. In this exercise, you’ve:

* Made graphs with ggplot2.
* Seen how to customize these graphs.
* Calculated basic statistics, such as medians.
* Looked at the spread of data by using box plots and histograms.
* Learned about samples versus populations.
* Estimated what the population of graphs might look like from a sample of grades.

In your next notebook, you’ll look at spotting unusual data and finding relationships between and among your data.

## Further reading

To learn more about the R packages and concepts you explored in this notebook, see:

* [Tidyverse packages](https://www.tidyverse.org/packages/)
* [Patchwork](https://patchwork.data-imaginist.com/)
* [Functions with R](https://skirmer.github.io/presentations/functions_with_r.html#1)