Explore real-world data by using R

2023-02-28

# Explore real-world data by using R

In your last exercise, you looked at grades for your student data and investigated this visually by using histograms and box plots. Now you’ll look into more complex cases, describe the data more fully, and see how to make basic comparisons between data.

## Real-world data distributions

Last time, you looked at grades for your student data and estimated from this sample what the full population of grades might look like. Just to refresh, let’s take another look at this data.

# Load the required packages into 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

library(patchwork)  
library(statip)  
library(glue)

## Including Plots

Let’s take another look at the grades distribution:

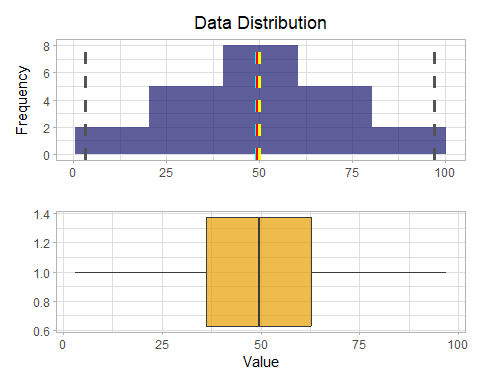
# Read a CSV file into a tibble  
df\_students <- read\_csv(file = "https://raw.githubusercontent.com/MicrosoftDocs/ml-basics/master/data/grades.csv")

## 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)  
  
# 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(  
 '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)}'  
 )  
   
 theme\_set(theme\_light())  
 # 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,  
 hist\_gram / bx\_plt)) # End of returned outputs  
   
} # End of function  
  
  
# Call the function  
show\_distribution(var\_data = select(df\_students, Grade), binwidth = 20)

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

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

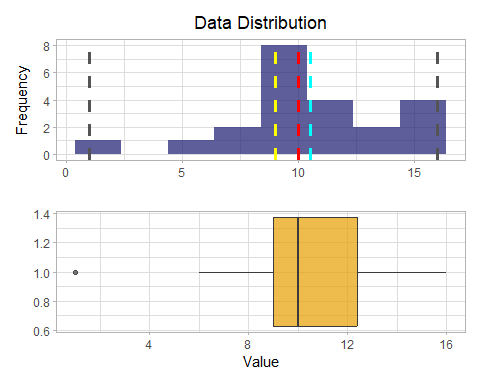


As you’ll recall, your data had the mean and mode at the center, with data spread symmetrically from there.

Now let’s take a look at the distribution of the study hours data.

# Get the variable to examine  
col <- df\_students %>%   
 select(StudyHours)  
  
# Call the function  
show\_distribution(var\_data = col, binwidth = 2)

## [[1]]  
## Minimum: 1.00  
## Mean: 10.52  
## Median: 10.00  
## Mode: 9.00  
## Maximum: 16.00  
##   
## [[2]]

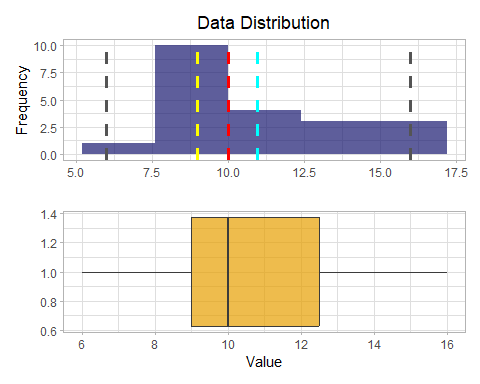
 The distributions of the study time data and the grades data differ significantly.

Note that the whiskers of the box plot begin at only about 6.0, indicating that the vast majority of the first quarter of the data is greater than this value. The minimum is marked with an o, indicating that it’s statistically an outlier, a value that lies significantly outside the range of the rest of the distribution.

Outliers can occur for many reasons. Maybe a student meant to record 10 hours of study time, but entered 1 and omitted the 0. Or maybe the student is abnormally lazy when it comes to putting in study time. Either way, the low number is a statistical anomaly that doesn’t represent a typical student. Let’s see what the distribution looks like without it.

# Get the variable to examine without outliers  
# You'll only get students who have studied more than one hour  
col <- df\_students %>%   
 select(StudyHours) %>%   
 filter(StudyHours > 1)  
  
  
# Call the function  
show\_distribution(var\_data = col, binwidth = 2.4)

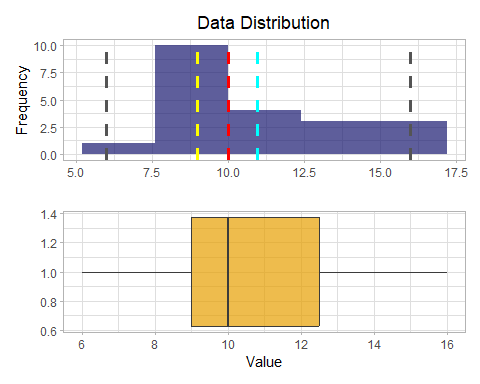
## [[1]]  
## Minimum: 6.00  
## Mean: 10.98  
## Median: 10.00  
## Mode: 9.00  
## Maximum: 16.00  
##   
## [[2]]

 For learning purposes, you’ve just treated the value 1 as a true outlier here and excluded it. In the real world, though, it would be unusual to exclude data at the extremes without more justification when your sample size is so small. This is because the smaller your sample size, the more likely it is that your sampling is a misrepresentation of the whole population (here, the population means grades for all students, not just the 22 in our sample). For example, if you were to sample study time for another 1000 students, you might find that it’s actually quite common to not study much.

When you have more data available, your sample becomes more reliable. This makes it easier to consider outliers as being values that fall below or above percentiles within which most of the data lie. For example, the following code uses the inbuilt *stats::quantile()* function to exclude observations below the 0.01th percentile (the value above which 99 percent of the data resides).

# Produce a quantile corresponding to 1%  
q01 <- df\_students %>%   
 pull(StudyHours) %>%   
 quantile(probs = 1/100, names = FALSE)  
  
# Get the variable to examine  
col <- df\_students %>%   
 select(StudyHours) %>%   
 filter(StudyHours > q01)  
   
# Call the function  
show\_distribution(var\_data = col, binwidth = 2.4)

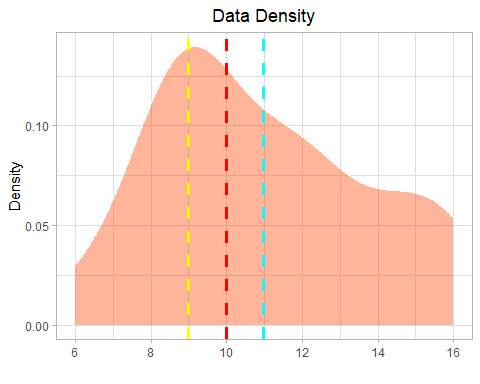
## [[1]]  
## Minimum: 6.00  
## Mean: 10.98  
## Median: 10.00  
## Mode: 9.00  
## Maximum: 16.00  
##   
## [[2]]



*Tip: You can also eliminate outliers at the upper end of the distribution by defining a threshold at a high percentile value. For example, you could use the stats::quantile() function to find the 0.99 percentile below which 99 percent of the data reside.*

With the outliers removed, the box plot shows all data within the four quartiles. Note that the distribution is not symmetric, as it is for the grade data. Some students have very high study times of around 16 hours, but the bulk of the data is between 7 and 13 hours, The few extremely high values pull the mean toward the higher end of the scale. Let’s look at the density for this distribution.

# 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 StudyHours  
show\_density(var\_data = col)



This kind of distribution is called right skewed. The mass of the data is on the left side of the distribution, creating a long tail to the right because of the values at the extreme high end, which pulls the mean to the right.

## Measures of variance

So now you have a good idea where the middle of the grade and study hours data distributions are. However, there’s another aspect of the distributions you should examine: how much variability is there in the data?

Typical statistics that measure variability in the data include:

* **Range**: The difference between the maximum and minimum. There’s no built-in function for this, but it’s easy to calculate by using the min and max functions. Another approach would be to use Base R’s base::range() which returns a vector containing the minimum and maximum of all the given arguments. Wrapping this in base::diff() will get you well on your way to finding the range.
* **Variance**: The average of the squared difference from the mean. You can use the built-in var function to find this.
* **Standard deviation**: The square root of the variance. You can use the built-in sd function to find this.

# Select columns to analyze the measure of variance  
cols <- df\_students %>%  
 select(c(Grade, StudyHours))  
  
# Apply a function to each column in cols  
map(cols, function(column){  
 rng <- diff(range(column)) # --> same as: max(column) - min(column)  
 var <- var(column)  
 std <- sd(column)  
 glue(  
 '  
 - Range: {format(round(rng, 2), nsmall = 2)}  
 - Variance : {format(round(var, 2), nsmall = 2)}  
 - Std.Dev : {format(round(std, 2), nsmall = 2)}',  
 .sep = '\n')  
})

## $Grade  
## - Range: 94.00  
## - Variance : 472.54  
## - Std.Dev : 21.74  
##   
## $StudyHours  
## - Range: 15.00  
## - Variance : 12.16  
## - Std.Dev : 3.49

By using [map()](https://purrr.tidyverse.org/reference/map.html) functions, you can replace many for loops with code that’s both more succinct and easier to read.

Of these statistics, the standard deviation is ordinarily the most useful. It provides a measure of variance in the data on the same scale as the data itself (grade points for the grades distribution and hours for the study hours distribution). The greater the standard deviation, the more variance there is when you compare values in the distribution with the distribution mean. That is, the data is more spread out.

When you’re working with a normal distribution, the standard deviation works with the particular characteristics of a normal distribution to provide even greater insight. This can be summarized by using the 68–95–99.7 rule, also known as the empirical rule, which is described as follows:

In any normal distribution:

* Approximately 68.26 percent of values fall within one standard deviation from the mean.
* Approximately 95.45 percent of values fall within two standard deviations from the mean.
* Approximately 99.73 percent of values fall within three standard deviations from the mean. As a quick detour, let’s verify that the distribution of grades follows a normal distribution.

# Get the variable to examine  
col <- df\_students %>%   
 select(Grade)  
  
# Get the mean grade  
mean\_grade <- mean(pull(col))  
  
# Get the standard deviation of grades  
std\_dev <- sd(pull(col))  
  
# Find proportion that will fall within 1 standard deviation  
one\_std\_dev <- pnorm((mean\_grade + std\_dev), mean = mean\_grade, sd = std\_dev) -  
 pnorm((mean\_grade - std\_dev), mean = mean\_grade, sd = std\_dev)  
  
# Find proportion that will fall within 2 standard deviations  
two\_std\_dev <- pnorm((mean\_grade + (2\*std\_dev)), mean = mean\_grade, sd = std\_dev) -   
 pnorm((mean\_grade - (2\*std\_dev)), mean = mean\_grade, sd = std\_dev)  
  
# Find proportion that will fall within 3 standard deviations  
three\_std\_dev <- pnorm((mean\_grade + (3\*std\_dev)), mean = mean\_grade, sd = std\_dev) -   
 pnorm((mean\_grade - (3\*std\_dev)), mean = mean\_grade, sd = std\_dev)  
  
glue(  
 '  
 {format(round(one\_std\_dev\*100, 2), nsmall = 2)}% of grades fall within one standard deviation from the mean.  
 {format(round(two\_std\_dev\*100, 2), nsmall = 2)}% of grades fall within one standard deviation from the mean.  
 {format(round(three\_std\_dev\*100, 2), nsmall = 2)}% of grades fall within one standard deviation from the mean.  
   
 '  
)

## 68.27% of grades fall within one standard deviation from the mean.  
## 95.45% of grades fall within one standard deviation from the mean.  
## 99.73% of grades fall within one standard deviation from the mean.

There’s no doubt that the grades distribution follows a normal distribution.

It wouldn’t hurt to show this graphically, right? Instead of using geom\_density, let’s manually calculate the density of the Grade column and then use the results to whip up a density plot. We’ll then fill the density plot depending on the number of standard deviations from the mean.

# Get the Grade column  
col <- df\_students %>%   
 select(Grade)  
  
# Compute kernel density estimates  
density <- density(pull(col), kernel = "gaussian")  
  
# Create a dataframe containing coordinates where density is estimated  
#and the corresponding estimated value  
density\_xy <- tibble(  
 x = density$x,  
 y = density$y  
)  
  
# Make density plots for various deviations of the mean  
  
plt\_sd\_1 <- density\_xy %>%  
 ggplot(mapping = aes(x = x, y = y)) +  
 geom\_line() +  
 geom\_area(mapping = aes(x = ifelse(x>(mean\_grade-std\_dev) & x<(mean\_grade+std\_dev), x, 0)), fill = "darkorange", alpha = 0.8) +  
 annotate("text", x = 50, y = 0.01, label = "1 std (68.26%)") +  
 xlim(1, max(col))+  
 geom\_vline(xintercept = mean\_grade, linetype = "dashed") +  
 labs(x = "", y = "Density")  
  
  
# 2 std deviations from the mean  
plt\_sd\_2 <- density\_xy %>%  
 ggplot(mapping = aes(x = x, y = y)) +  
 geom\_line() +  
 geom\_area(mapping = aes(x = ifelse(x>(mean\_grade-(2\*std\_dev)) & x<(mean\_grade+(2\*std\_dev)), x, 0)), fill = "purple", alpha = 0.7) +  
 annotate("text", x = 50, y = 0.01, label = "2 std (95.45%)") +  
 xlim(1, max(col)) +  
 geom\_vline(xintercept = mean\_grade, linetype = "dashed") +  
 labs(x = "", y = "")  
  
  
# 3 std deviations from the mean  
plt\_sd\_3 <- density\_xy %>%  
 ggplot(aes(x = x, y = y)) +  
 geom\_line() +  
 geom\_area(mapping = aes(x = ifelse(x>(mean\_grade-(3\*std\_dev)) & x<(mean\_grade+(3\*std\_dev)), x, 0)), fill = "cyan4", alpha = 0.5) +  
 annotate("text", x = 50, y = 0.01, label = "3 std (99.73%)") +  
 xlim(1, max(col)) +  
 geom\_vline(xintercept = mean\_grade, linetype = "dashed") +  
 labs(x = "Grade", y = "")  
  
  
  
# Patching things up  
plt\_sd\_1| (plt\_sd\_2/plt\_sd\_3)

## Warning: Removed 365 rows containing non-finite values (`stat\_align()`).

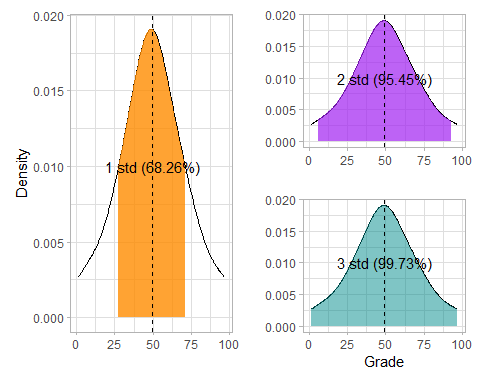
## Warning: Removed 189 rows containing missing values (`geom\_line()`).

## Warning: Removed 219 rows containing non-finite values (`stat\_align()`).

## Warning: Removed 189 rows containing missing values (`geom\_line()`).

## Warning: Removed 189 rows containing non-finite values (`stat\_align()`).

## Warning: Removed 189 rows containing missing values (`geom\_line()`).



Because you know that the mean grade is 49.18, the standard deviation is 21.74, and distribution of grades is normal, you can calculate that 68.26% of students should achieve a grade between 27.44 and 70.92, as shown in the first plot in the preceding code.

The descriptive statistics you’ve used to understand the distribution of the student data variables are the basis of statistical analysis. Because they’re such an important part of exploring your data, there’s a built-in Base R function, base::summary(), that returns the result summaries of the results of various objects that are passedto it.

# Get summary stats of the data frame  
summary(df\_students)

## Name StudyHours Grade Pass   
## Length:22 Min. : 1.00 Min. : 3.00 Mode :logical   
## Class :character 1st Qu.: 9.00 1st Qu.:36.25 FALSE:15   
## Mode :character Median :10.00 Median :49.50 TRUE :7   
## Mean :10.52 Mean :49.18   
## 3rd Qu.:12.38 3rd Qu.:62.75   
## Max. :16.00 Max. :97.00

Of course, because of the need for much more robust data exploration and reporting, many packages have been developed for summarizing data.

A good example would be the [summarytools](https://github.com/dcomtois/summarytools) package. summarytools::descr() does a remarkable job of finding statistics for numerical data.

# Get descriptive stats using summary tools package  
library(summarytools)

##   
## Attache Paket: 'summarytools'

## Das folgende Objekt ist maskiert 'package:tibble':  
##   
## view

descr(  
 df\_students,  
 stats = "common"  
)

## Non-numerical variable(s) ignored: Name, Pass

## Descriptive Statistics   
## df\_students   
## N: 22   
##   
## Grade StudyHours  
## --------------- -------- ------------  
## Mean 49.18 10.52  
## Std.Dev 21.74 3.49  
## Min 3.00 1.00  
## Median 49.50 10.00  
## Max 97.00 16.00  
## N.Valid 22.00 22.00  
## Pct.Valid 100.00 100.00

## Compare data

Now that you know something about the statistical distribution of the data in your dataset, you’re ready to examine it to identify any apparent relationships between variables.

First of all, remove any rows that contain outliers so that you have a sample that’s representative of a typical class of students. You identified that the StudyHours column contains some outliers with extremely low values, so you can remove those rows.

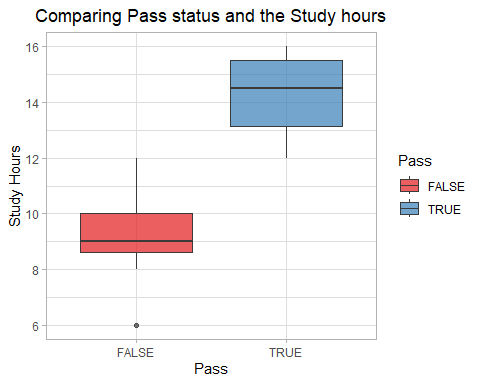
# Filter to remove outliers  
df\_sample <- df\_students %>%   
 filter(StudyHours > 1)

The data includes two numeric variables (StudyHours and Grade) and two *categorical variables* (Name and Pass). In R, categorical variables are usually saved as [factors](https://r4ds.had.co.nz/factors.html) or character vectors.

Let’s start by comparing the *numeric* StudyHours column with the *categorical* Pass column to see whether there’s an apparent relationship between the number of hours studied and a passing grade.

To make this comparison, you can create box plots that show the distribution of StudyHours for each possible Pass value (TRUE and FALSE).

# Plot a box plot comparing StudyHours and Pass  
df\_sample %>%   
 ggplot() +  
 geom\_boxplot(mapping = aes(x = Pass, y = StudyHours, fill = Pass), color = "gray23", alpha = 0.7) +  
 scale\_fill\_brewer(palette = "Set1") +  
   
# Add titles and labels  
 ggtitle("Comparing Pass status and the Study hours")+  
 xlab("Pass")+  
 ylab("Study Hours")+  
 theme(plot.title = element\_text(hjust = 0.5))

 When you compare the StudyHours distributions, it’s immediately apparent, if not particularly surprising, that students who passed the course tended to study for more hours than students who didn’t pass. If you want to predict whether students are likely to pass the course, the amount of time they spend studying might be a good predictive feature.

## Compare numeric variables

Now let’s compare two numeric variables. Start by creating a bar chart that shows both grades and study hours.

To do this, you can first transform your data to a long format by using tidyr::pivot\_longer() and then play around with the fill aesthetic and position\_dodge() to place overlapping objects directly beside one another.

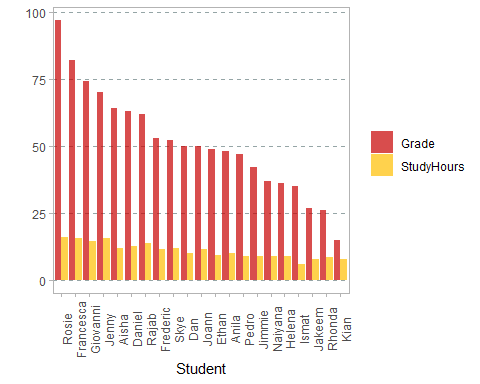
Also, we’ll try out the [paletter gallery](https://pmassicotte.github.io/paletteer_gallery/), a collection of color palettes in a single R package.

# Pivot data from wide to long  
df\_sample\_long <- df\_sample %>%  
 select(-Pass) %>%  
 mutate(Name = fct\_reorder(Name, Grade, .desc = TRUE)) %>%   
 pivot\_longer(!Name, names\_to = "Metrics", values\_to = "Values")  
  
# Show first 10 rows  
df\_sample\_long %>%   
 slice\_head(n = 10)

## # A tibble: 10 × 3  
## Name Metrics Values  
## <fct> <chr> <dbl>  
## 1 Dan StudyHours 10   
## 2 Dan Grade 50   
## 3 Joann StudyHours 11.5   
## 4 Joann Grade 50   
## 5 Pedro StudyHours 9   
## 6 Pedro Grade 47   
## 7 Rosie StudyHours 16   
## 8 Rosie Grade 97   
## 9 Ethan StudyHours 9.25  
## 10 Ethan Grade 49

# Try out some color palettes  
library(paletteer)  
  
# Compare numeric variables with bar plots  
ggplot(data = df\_sample\_long) +  
 geom\_bar(mapping = aes(x = Name, y = Values, fill = Metrics), alpha = 0.7, stat = "identity", position = position\_dodge(width = 0.9)) +  
 xlab('Student') +  
 ylab('') +  
 scale\_fill\_paletteer\_d("calecopal::kelp1") +  
 theme(  
 panel.grid = element\_blank(),  
 panel.grid.major.y = element\_line(color = '#95a5a6',  
 linetype = 'dashed',  
 size = 0.5),  
 axis.text.x = element\_text(angle = 90),  
 legend.title = element\_blank()  
   
 )

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



The chart shows bars for both the grade and study hours for each student. They’re not easy to compare, because the values are on different scales. Grades are measured in grade points, which range from 3 to 97, and study time is measured in hours, which range from 1 to 16.

A common technique when you’re dealing with numeric data in different scales is to normalize the data so that the values retain their proportional distribution, but are measured on the same scale. To do this, you use a technique called MinMax scaling, which distributes the values proportionally on a scale from 0 to 1. You could write the code to apply this transformation, but the [scales](https://scales.r-lib.org/) library provides a scaler, scales::rescale(), that does it for you.

The good news is that the scaler is installed when you install ggplot2 or the Tidyverse.

# Load the scales package  
library(scales)

##   
## Attache Paket: 'scales'

## Das folgende Objekt ist maskiert 'package:purrr':  
##   
## discard

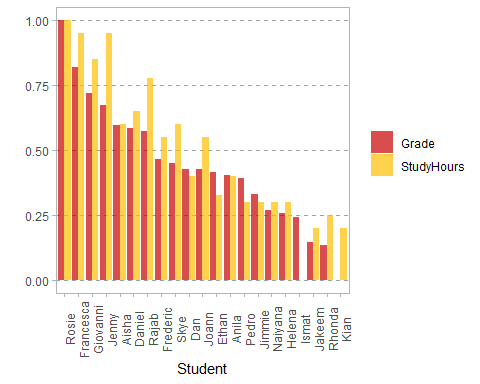
## Das folgende Objekt ist maskiert 'package:readr':  
##   
## col\_factor

# Normalize the numeric columns  
# group\_by ensures that StudyHours and Grade are normalized independently  
df\_normalized <- df\_sample\_long %>%   
 group\_by(Metrics) %>%   
 mutate(Values = rescale(Values, to = c(0, 1)))  
  
# Show some of the normalized data  
df\_normalized %>%   
 slice\_head(n = 5)

## # A tibble: 10 × 3  
## # Groups: Metrics [2]  
## Name Metrics Values  
## <fct> <chr> <dbl>  
## 1 Dan Grade 0.427  
## 2 Joann Grade 0.427  
## 3 Pedro Grade 0.390  
## 4 Rosie Grade 1   
## 5 Ethan Grade 0.415  
## 6 Dan StudyHours 0.4   
## 7 Joann StudyHours 0.55   
## 8 Pedro StudyHours 0.3   
## 9 Rosie StudyHours 1   
## 10 Ethan StudyHours 0.325

Great! Your grades and study hours are now rescaled.

# Compare numeric variables with bar plots  
ggplot(data = df\_normalized) +  
 geom\_bar(mapping = aes(x = Name, y = Values, fill = Metrics), alpha = 0.7, stat = "identity", position = position\_dodge(width = 0.9)) +  
 xlab('Student') +  
 ylab('') +  
 scale\_fill\_paletteer\_d("calecopal::kelp1") +  
 theme(  
 panel.grid = element\_blank(),  
 panel.grid.major.y = element\_line(color = '#95a5a6',  
 linetype = 'dashed',  
 size = 0.5),  
 axis.text.x = element\_text(angle = 90),  
 legend.title = element\_blank()  
   
 )

 With the data normalized, it’s easier to see an apparent relationship between grades and study time. It’s not an exact match, but it seems clear that students with higher grades tend to have studied more.

##Fitting a simple linear regression model

From the previous comparisons, there seems to be a correlation between study times and grades. In fact, there’s a statistical correlation measurement that you can use to quantify the relationship between these columns.

# Compute Pearson Product Moment correlation coefficient  
cor(df\_sample$StudyHours, df\_sample$Grade)

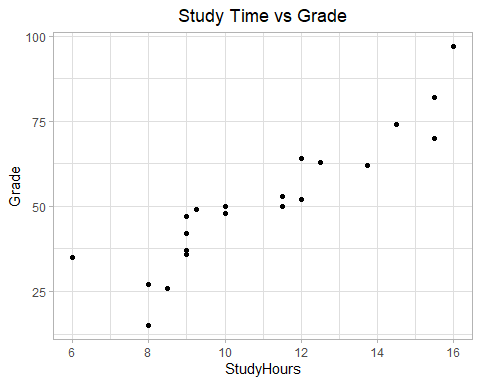
## [1] 0.9117666

The correlation statistic is a value from -1 to 1 that indicates the strength of a relationship. Values greater than 0 indicate a positive correlation (high values of one variable tend to coincide with high values of the other), while values less than 0 indicate a negative correlation (high values of one variable tend to coincide with low values of the other). A correlation equal to 0 indicates that there is no relationship between the two variables. In this case, the correlation value is close to 1, which shows a strong positive correlation between study times and grades.

**Note**: Data scientists often quote the maxim “correlation is not causation.” In other words, as tempting as it might be, you shouldn’t interpret the statistical correlation as explaining why one of the values is high. In the case of the student data, the statistic demonstrates that students with high grades tend to have studied a great number of hours.But this is not the same as saying that they achieved high grades because they studied a lot. The statistic could equally be used as evidence to support the nonsensical conclusion that the students studied a lot because their grades were going to be high.

Another way to visualize the apparent correlation between two numeric columns is to use a scatter plot. With geom\_point, you’ll be well on your way to whipping up a neat scatterplot.

# Create a scatter plot of study hours and grade  
df\_sample %>%   
 ggplot(aes(x = StudyHours, y = Grade)) +  
 geom\_point() +  
 ggtitle('Study Time vs Grade') +  
 theme(plot.title = element\_text(hjust = 0.5))



Again, it looks like there’s a discernible pattern in which the students who studied the most hours are also the students who got the highest grades.

You can see this more clearly by adding to the plot a regression line (or a line of best fit) that shows the general trend in the data. To do this, you can use a statistical technique called least squares regression.

**Warning: math ahead!** Cast your mind back to when you were learning how to solve linear equations in school, and recall that the slope-intercept form of a linear equation looks like this:y = mx + b. In this equation, y and x are the coordinate variables, m is the slope of the line, and b is the y-intercept (where the line goes through the y-axis). In the case of your scatter plot for our student data, you already have values for x (StudyHours) and y (Grade), so you need only calculate the intercept and slope of the straight line that lies closest to those points. Then you can form a linear equation that calculates a new y value on that line for each of your x values. To avoid confusion, let’s call this new y value f(x), because it’s the output from a linear equation function based on x. The difference between the original y (Grade) value and the f(x) value is the error between the regression line and the actual Grade achieved by the student. Your goal is to calculate the slope and intercept for a line with the lowest overall error. Specifically, you define the overall error by taking the error for each point, squaring it, and adding all the squared errors together. The line of best fit is the line that gives you the lowest value for the sum of the squared errors, hence the name least squares regression. Fortunately, you don’t need to code the regression calculation yourself. The inbuilt R Stats Package stats provides the lm() function to do the hard work for you. lm() takes a formula of the form:

outcome ~ predictor

This returns, among other things, the coefficients you need for the slope equation, slope (m) and intercept (b), based on a specified pair of variable samples you want to compare.

# Drop all columns except Grade and StudyHours  
df\_regression <- df\_sample %>%   
 select(c(Grade, StudyHours))  
  
# Fit a linear model  
lm\_df\_regression <- lm(Grade ~ StudyHours, data = df\_regression)  
  
# Get the regression slope and intercept  
intercept\_b <- lm\_df\_regression$coefficients[1]  
slope\_m <- lm\_df\_regression$coefficients[2]  
glue(  
 'slope: {format(round(slope\_m, 4), nsmall = 4)}  
 y-intercept: {format(round(intercept\_b, 4), nsmall = 4)}  
 so…  
 f(x) = {format(round(slope\_m, 4), nsmall = 4)}x + {format(round(intercept\_b, 4), nsmall = 4)}'  
)

## slope: 6.3134  
## y-intercept: -17.9164  
## so…  
## f(x) = 6.3134x + -17.9164

Now that you have the equation f(x) = 6.3134x + -17.9164 of the least squares regression line, you can use it to estimate the grade based on the amount of hours spent studying.

# Use the function (mx + b) to calculate f(x) for each x (StudyHours) value  
df\_regression <- df\_regression %>%   
 mutate(fx = (slope\_m \* StudyHours + intercept\_b),  
 error = fx - Grade)  
  
  
# Show data frame  
df\_regression

## # A tibble: 21 × 4  
## Grade StudyHours fx error  
## <dbl> <dbl> <dbl> <dbl>  
## 1 50 10 45.2 -4.78   
## 2 50 11.5 54.7 4.69   
## 3 47 9 38.9 -8.10   
## 4 97 16 83.1 -13.9   
## 5 49 9.25 40.5 -8.52   
## 6 53 11.5 54.7 1.69   
## 7 42 9 38.9 -3.10   
## 8 26 8.5 35.7 9.75   
## 9 74 14.5 73.6 -0.372  
## 10 82 15.5 79.9 -2.06   
## # … with 11 more rows

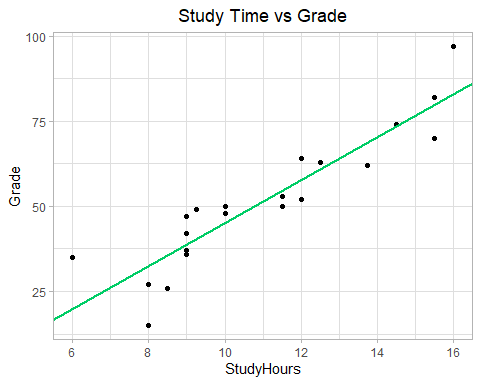
You now have a data frame that contains the following values:

* The StudyHours for each student
* The Grade achieved by each student
* The f(x) value, calculated by using the regression line coefficients
* The error between the calculated f(x) value and the actual Grade value.

Some of the errors, particularly at the extreme ends, are quite large (up to over 17.5 grade points). In general, though, the line is pretty close to the actual grades.

Now it’s time to show this visually. You’ll create the scatter plot of the sample study hours and grades, as before, and then add a line of best fit based on the least squares regression coefficients.

df\_sample %>%   
 ggplot() +  
 geom\_point(aes(x = StudyHours, y = Grade)) +  
 # Add a line based on the linear model  
 geom\_abline(intercept = intercept\_b, slope = slope\_m, color = "springgreen3", size = 1) +  
 ggtitle('Study Time vs Grade') +  
 theme(plot.title = element\_text(hjust = 0.5))



## Use the regression coefficients for prediction

Now that you have the regression coefficients for the study time and grade relationship, you can use them in a function to estimate the expected grade for a specified amount of study time.

# Define a function based on our regression coefficients  
f <- function(x) {  
 slope\_m <- 6.3134  
 y\_intercept\_b <- -17.9164  
 # y = mx + b  
 fx <- slope\_m \* x + y\_intercept\_b  
 return(fx)  
}  
  
# Assume the student studied for 14 hours per week  
study\_time <- 14  
  
# Get f(x) for study time  
prediction <- f(x = study\_time)  
  
# Grade can't be less than 0 or more than 100  
expected\_grade <- max(0, min(100, prediction))  
  
# Print the estimated grade  
glue(  
 'Studying for {study\_time} hours per week may result in a grade of {format(round(expected\_grade))}  
 ')

## Studying for 14 hours per week may result in a grade of 70

By applying statistics to sample data, you’ve determined a relationship between study time and grade. And you’ve encapsulated that relationship in a general function that can be used to predict a grade for a given amount of study time.

This technique is in fact the basic premise of machine learning. You can take a set of sample data that includes one or more features (in this case, the number of hours studied) and a known label value (in this case, the grade achieved) and use the sample data to derive a function that calculates predicted label values for any specified set of features.

## Summary

In this exercise, you’ve learned:

* What outliers are and how to remove them
* How data can be skewed
* How to look at the spread of data
* Basic ways to compare variables, such as grades and study time

## Further reading

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

[Tidyverse packages](https://www.tidyverse.org/packages/) [R for Data Science: Visualize, Model, Transform, Tidy, and Import Data](https://r4ds.had.co.nz/) by H. Wickham and G. Grolemund