INF2167: R For Data Science – In Class Coding Demonstrations

Contents

About this Book

This repository contains code written during in-class coding sessions for the class INF2167: R For Data Science in Fall 2025.

All code was written by **Jennet Baumbach, Ph.D**. Questions about this code can be directed to jennetbaumbach@gmail.com.

Week 1: September 2 2025

Intro to Markdown

Rmarkdown is an end-to-end document creation tool. The main advantage of using this workflow is the ability to generate **reproducible output files**. The key benefit of working in this fashion is that you can integrate code, statistical analyses, charts, and written interpretations into a single document.

You can organize an Rmarkdown document using headings and subheadings, which are written using hashtags:

Two Hashtags Makes a Level 2 header

Three Hashtags Makes a Level 3 header

You can also add emphasis to written sections by bolding and / or italicizing key words.

- Write single asterisks on each side of text to make *italics*.
- Wrap text with double asterisks to make text **bold**.
- Triple asterisks on each side will make the text ***both italic and bold***.

Intro to Using R Code

Using markdown code (as described above) goes *outside* of code blocks. All R code must be written ***inside*** of code blocks. You can generate a code block by pressing `control` + `shift` + `i` on your keyboard (or `command` + `option` + `i` if you're using a mac) to generate a code block.

When we enter a math equation with no assignment it will print out the result:

```
12 * 4
```

```
## [1] 48
```

- So R can be used as a badass calculator.

Object Oriented Programming

Although computing calculations can be useful, we often want to save the results of computational processes so that they can be used in subsequent analysis steps. Instead of just writing code "in line" (as shown above), we can **assign** the results of a computational process onto an **object** in the working environment. Use the arrow (`<-`) to assign a new object.

```
a <- 12 * 4
```

- The results of this process are now assigned onto the object `a`.
- We can then call back the object `a` to use it in subsequent computational steps.

```
a * 3
```

```
## [1] 144
```

- Here, R will print out 48 * 3

We could also assign the results of `a` * 3 onto another object in the environment:

```
b <- a * 3
```

One thing to watch out for when working with object-oriented programming is that objects can be overwritten. R will not give any warning to let you know when something is overwritten.

```
# R will overwrite with no warning! Watch out!
a <- b * a
```

- Now the object `a` will be the value 6912.

Lists of Values (Vectors)

We often work with ordered lists of values, not just single values as I showed above. In R, we can assign lists of values onto objects in the environment as well. In order to write an ordered list of values, we an use the function `c()` which means "concatenate" (a.k.a. "combine"). We then **pass** the list of values separated by commas inside the parentheses.

```
c <- c(1, 2, 3, 4   , 5, 6,7,8,9,0)
```

- R language does not care about spaces or line breaks.
- You should use spacing to help organize your code and to increase readability.

Functions in R

Most of the time when using R, you will want to structure your work using **functions**, rather than just writing out mathematical equations. Functions in R follow a simple syntax:

$$do\ this(to\ that,\ with\ specifications)$$

- The "verb" goes before the parentheses and specifies a mathematical process that should be carried out.
- The "noun" goes inside the brackets, telling R what the verb should be applied to.

Example: we can use the `data.frame` function to assign multiple lists of values onto an object in the environment. This allows us to store data in a 2-dimensional space where there are multiple rows and multiple columns.

```
d <- data.frame(
  ID = c(1,2,3,4,5,6),
  Score = c(100, 120, 110, 98, 75, 65)
)
```

- Ideally, data should be organized as ***one subject one row***.
- Most of the time when working in analytics / data science, you will work with data frames.

We could also pass our data frame to other functions. For example, we could pass our object `d` to the `tail()` function, which would print out the last six rows of data.

```
tail(d, 3)
```

```
##   ID Score
## 4  4    98
## 5  5    75
## 6  6    65
```

- Since I added ,3 inside the brackets, we only see the last three rows of data printed out.

Week 2: September 11 2025

Packages in R

Many useful functions come built in when you download Rstudio. Any functionality that belongs to the program in this nascent state is called **base R**.

Packages can be downloaded separately, and allow us to access custom functions that are not available in base R. Many people independently develop and maintain specialized packages that contain custom functions to simplify comoplex tasks.

Accessing a package is a two-step process. First, we must download the contents of a given package onto our computer:

```
# Run once per computer:
# install.packages("tidyverse")
# install.packages("faux")
```

- You only need to run the `install.packages()` command once per computer.
- I have the `install.packages()` lines "noted out" in the code above, because they prevent the document from knitting.
- If you encounter issues while knitting documents, check that you do not have any `install.packages()` commands written in the code.

Each time you re-open R, you need to call all the packages that you want to use to the current working session using the `library()` command.

```
# Run every time you re-open R:
library(tidyverse)
library(faux)
```

- It is good practice to load up all required packages at the top of an Rmarkdown file.

Example: Generating Ficticous Data

It is often quite useful to be able to generate quick fictitious datsets to test out analytic pipelines. Since we have been talking about correlation and simple linear regression today, I will demonstrate this by creating a dataframe of 100 observations that have a strong linear relationship between them.

The `rnorm_multi` function from the `faux` package allows us to generate multiple columns of simulated data with a specified correlation between them.

```
set.seed(994) # For a reproducible example

ex <- rnorm_multi(n = 100, vars = 2,
                  varnames = c("Height", "Foot_length"),
                  mu = c(65,12), sd = c(10,4), r = .7)
head(ex)
```

```
##     Height Foot_length
## 1 68.17282    9.053081
## 2 66.07463   11.923344
## 3 70.57495   13.669080
## 4 76.48643   18.544183
## 5 55.35918   14.376149
## 6 76.52625   16.950847
```

- The `set.seed()` function makes the example reproducible. If you use the same value in the `set.seed()` command as me, you will get the exact same values as me.
- I have created two columns of data named **Height** and **Foot_length**.
- **Height** has a mean of 65 and a standard deviation of 10.
- **Foot_length** has a mean of 12 and a standard deviation of 4.
- I specified that I would like the correlation between the two columns to be around 0.7 (which is a strong correlation).

Compute Correlations

We don't need any specialized packages to compute correlations because these functions come built into Base R. In fact, there are two functions from base R that we can use to compute correlation.

Option #1: `cor.test`

```
options(scipen = 999) # turn off scientific notation
cor.test(ex$Height, ex$Foot_length)
```

```
##
##  Pearson's product-moment correlation
##
## data:  ex$Height and ex$Foot_length
## t = 9.0028, df = 98, p-value = 0.00000000000001764
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.5489307 0.7677638
## sample estimates:
##       cor
## 0.6728068
```

- The correlation between foot length and height is 0.67, which would be incredibly unlikely to be obtained by chance ($p < 0.001$)
- The `cor.test()` function shows us all statistical information including the t-statistic, the degrees of freedom, the 95% confidence interval and the point estimate for the correlation.

Option #2: `cor()`

```
cor(ex$Height, ex$Foot_length)
```

```
## [1] 0.6728068
```

- Only prints out the single value representing the correlation.

A Picture is Worth 1000 Words

In addition to seeing these values printed out, we might want to generate a chart to showcase the linear relationship. It is often easier to correctly interpret relationships from charts, and they are a good tool to ensure that your interpretation of the values makes sense given the data.

We can make charts using Base R, but they are not especially nice looking. A better, faster, more flexible way of generating charts is to use the `ggplot2()`

package. This package allows us to create publication-ready charts that can be endlessly customized. Most any chart that you can image can be generated using `ggplot2()`. For this reason, I recommend getting very comfortable using `ggplot2()` if you are going to be generating any visualizations through R. The `ggplot2()` package is part of the `tidyverse()`. If you have already loaded the conglomerate `tidyverse()` package, then you will already have access to `ggplot2()`.

```
a <- ggplot(data = ex, aes(x = Height, y = Foot_length)) +
  geom_point(size = 2, alpha = 0.5, colour = "#800020") +
  geom_smooth(method = "lm", colour = "#800020", alpha = 0.01) +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  labs(
    x = "Height (in)",
    y = "Foot Length (in)"
  )
```

- The code above generates the chart and saves it onto the object `a` in my environment.

- We *could* print the chart out in-line, but it would not appear very high-quality

To generate a high-quality chart, it is best to save it off as a .png file, then to call that .png file back to the current document. In the codde below, I specify a file name for the chart, dimensions, and ensure that it will appear high-quality by specifying `dpi = 300`. Then, I use `knitr::include_graphics()` to call back the chart that I saved.

```
ggsave("ex_chart.png", a, height = 3, width = 3, dpi = 300)
knitr::include_graphics("ex_chart.png")
```

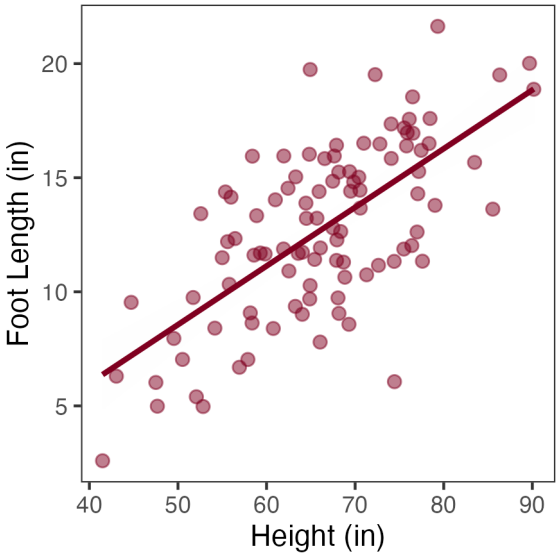


Figure 1. Linear relationship between height and foot length, both measured in inches.

- There is a clear linear relationship between these two quantities.
- Taller people tend to have larger feet.
- We know that the relationship is statistically signficiant based on the correlation anaylsis above.
- But the correlation analysis does not provide information about the slope of the line of best fit. In order to understand the slope, we must construct a linear model.

Construct a Linear Model

We can use the `lm()` command to request a linear model from R. The output looks different, but mathematically this is the same process as the correlation analysis shown above.

```
res <- lm(Foot_length ~ Height, data = ex)
summary(res)
```

```
##
## Call:
## lm(formula = Foot_length ~ Height, data = ex)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.7832 -2.1480  0.4244  2.0276  7.3441
##
## Coefficients:
##             Estimate Std. Error t value           Pr(>|t|)
## (Intercept) -4.29949    1.91489  -2.245              0.027 *
## Height       0.25717    0.02857   9.003 0.0000000000000176 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.844 on 98 degrees of freedom
## Multiple R-squared:  0.4527, Adjusted R-squared:  0.4471
## F-statistic: 81.05 on 1 and 98 DF,  p-value: 0.00000000000001764
```

- The R-squared value tells us the proportion of variance in the dependent variable that is explained by variance in the independent variable.
- Variance in height accounts for 45% of the variance in foot length.
- The R-squared value is simply the correlation coefficient squared.

```
sqrt(0.4527)
```

```
## [1] 0.6728298
```

- If we square root the R-squared value, we get the correlation coefficient that we computed above.
- The Estimate for the predictor tells us the slope of the line of best fit. This gives us information about ***the predicted change in y given a 1-unit change in x***.
- A 1-inch increase in height is associated with a 0.25-inch increase in foot length.

Tidy Outputs

The results from the linear model shown above print out a bit messy. We can use the `tidy()` function from the `broom` package to generate better looking outputs for statistical reporting:

```
library(broom)
knitr::kable(tidy(res), digits = 3)
```

term	estimate	std.error	statistic	p.value
(Intercept)	-4.299	1.915	-2.245	0.027
Height	0.257	0.029	9.003	0.000

Week 3: September 18 2025

```
library(tidyverse)
```

- Anytime I will be running exploratory data analysis I start by loading up the `tidyverse` package.
- The `tidyverse` is a conglomerate package that contains `ggplot2` and assorted other packages that assist with data wrangling.

Today's Topic: Intro to Multiple Regression

In order to explore multiple regression modelling, we need to acquire some data to work with. Today, I'll demonstrate this analysis approach using a famous dataset that comes with Rstudio.

The `mtcars` Dataset

`mtcars` is a dataset that comes pre-loaded in Rstudio. It's a great resource to practice analyses because there are many robust effects present in this dataset. The `mtcars` dataset is what we would refer to as a "toy dataset"; it was sepcifically designed to showcase statistical effects. Real world datasets do not always contain such robust relationships.

You can use the `?` in Rstudio to get information about the mtcars dataset:

```
?mtcars
```

- Running this line of code will open up the help panel in the lower right-hand quadrent of the screen.
- The help panel will provide information about what kind of information is contianed in each column of the dataset.

Assigning the mtcars dataset onto an object:

```
dat <- mtcars
```

- `mtcars` is always present in the working environment, but it is hidden (you won't see it in the environment tab)
- I prefer to assign the datset onto a visible object as I find this makes it easier to work with.
- The line of code above assigns the `mtcars` dataset onto an object in the environment named `dat`.

Recall: Simple Regression

Any regression model that contains only a single predictor is consdiered to be a simple regression model. Simple regression modeling is very useful for understanding relationships between pairs of variables.

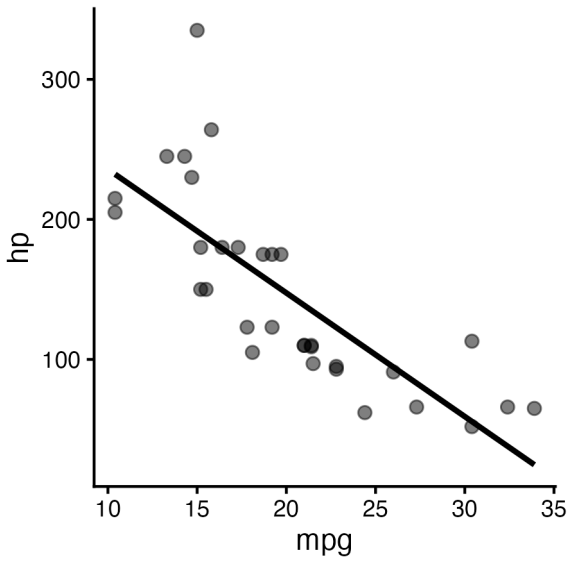
For example, if we wanted to understand the relationship between a car's horsepower and it's fuel efficiency, we could run a simple regression:

```
options(scipen = 999)
res <- lm(mpg ~ hp, data = dat)
summary(res)
```

```
##
## Call:
## lm(formula = mpg ~ hp, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.7121 -2.1122 -0.8854  1.5819  8.2360
##
## Coefficients:
##             Estimate Std. Error t value             Pr(>|t|)
## (Intercept) 30.09886    1.63392  18.421 < 0.0000000000000002 ***
## hp          -0.06823    0.01012  -6.742          0.000000179 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.863 on 30 degrees of freedom
## Multiple R-squared:  0.6024, Adjusted R-squared:  0.5892
## F-statistic: 45.46 on 1 and 30 DF,  p-value: 0.0000001788
```

- There is a strong and significant relationship between cars' horsepower and their mileage.
- Variance in fuel efficiency accounts for 60% of variance in horsepower (R^2 = 0.60).
- A 1-unit increase in horsepower is associated with a 0.06-unit decrease in the number of miles the car can travel per gallon of fuel.

```
# Create a scatterplot
a <- dat %>%
  ggplot(aes(x = mpg, y = hp)) +
  geom_point(size = 2, alpha = 0.5) +
  geom_smooth(method = "lm", colour = "black", se = FALSE)+
  theme_classic()

# Save the chart off as a high quality .png image
ggsave("chart_1.png", a, height = 3, width = 3, dpi = 300)

# Call the high quality .png image back
knitr::include_graphics("chart_1.png")
```



Multiple Regression

In contrast to simple regression, any analysis that inovlves entering multiple predictors into the model is considered to be **multiple regression**. There are

two basic ways that we can use multiple regresssion in analysis: (1) to control for other variables and (2) to test the influence of moderator variables. Each approach is explained and demonstrated in the following sections.

1. Control for Other Variables

When multiple predictors are simultaneously entered into a regression model, the output will show the ***unique*** varaince. Variance that is shared between the predictors is omitted from the coefficients table. Entering multiple variables in this way can provide information about whether two (or more) predictors are unique or overlapping.

```
res_2 <- lm(mpg ~ hp + wt, data = dat)
summary(res_2)
```

```
##
## Call:
## lm(formula = mpg ~ hp + wt, data = dat)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.941 -1.600 -0.182  1.050  5.854
##
## Coefficients:
##             Estimate Std. Error t value             Pr(>|t|)
## (Intercept) 37.22727    1.59879  23.285 < 0.0000000000000002 ***
## hp          -0.03177    0.00903  -3.519              0.00145 **
## wt          -3.87783    0.63273  -6.129           0.00000112 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.593 on 29 degrees of freedom
## Multiple R-squared:  0.8268, Adjusted R-squared:  0.8148
## F-statistic: 69.21 on 2 and 29 DF,  p-value: 0.000000000009109
```

- This analysis shows us that the weight of a car and the car's horsepower are unique predictors of its fuel efficiency.
- We could says that the car's horsepower is a significant predictor of the fuel efficiency even when we account for vehicle weight.
- The combination of weight and horsepower account for 81.5% of variance in fuel efficiency.

2. Investigate the Influence of Moderators

In many cases, the relationship between a predictor and a dependent variable differs based on the level of some third variable. The third variable in this instance is referred to as a **moderator**.

```
# Recode the column am to a factor
dat$am <- factor(dat$am, levels = c(0,1),
                    labels = c("Automatic", "Manual"))
```

```
# Include an interaction term
res_4 <- lm(mpg ~ hp * am, data = dat)
summary(res_4)
```

```
##
## Call:
## lm(formula = mpg ~ hp * am, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.3818 -2.2696  0.1344  1.7058  5.8752
##
## Coefficients:
##               Estimate Std. Error t value         Pr(>|t|)
## (Intercept) 26.6248479  2.1829432  12.197 0.00000000000101 ***
## hp          -0.0591370  0.0129449  -4.568 0.00009018507863 ***
## amManual     5.2176534  2.6650931   1.958           0.0603 .
## hp:amManual  0.0004029  0.0164602   0.024           0.9806
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.961 on 28 degrees of freedom
## Multiple R-squared:  0.782,  Adjusted R-squared:  0.7587
## F-statistic: 33.49 on 3 and 28 DF,  p-value: 0.000000002112
```

- Horsepower is a significant predictor of fuel efficiency for automatic cars ($p < 0.001$).
- For automatic cars, a 1-unit increase in horsepower is associated with a 0.059-unit decrease in fuel efficiency. We could also transform the estimate to make it more interpenetrate in the context of our data. For example, we could say that an increase in horsepower of 100 is associated with around a 6-mile decrease in mpg.
- The interaction between horsepower and transmission type is non-significant ($p = 0.98$), indicating that the slopes of the lines of best fit for

the relationship between horsepower and fuel efficiency do not differ for automatic and manual cars.

Note that we only get to see the estimate for the slope of the line of best fit for the *reference group*. When working with a binary variable, the reference group will be whichever group is coded as 0. If we want to see the slope of the line of best fit for manual transmission cars, we could re-code the column that contains this data so that Maunal cars are coded as 0.

```
dat$am <- relevel(dat$am, ref = "Manual")
```

```
res_5 <- lm(mpg ~ hp * am, data = dat)
summary(res_5)
```

```
##
## Call:
## lm(formula = mpg ~ hp * am, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.3818 -2.2696  0.1344  1.7058  5.8752
##
## Coefficients:
##                   Estimate Std. Error t value             Pr(>|t|)
## (Intercept)    31.8425012  1.5288820  20.827 < 0.0000000000000002 ***
## hp             -0.0587341  0.0101671  -5.777           0.00000334 ***
## amAutomatic    -5.2176534  2.6650931  -1.958               0.0603 .
## hp:amAutomatic -0.0004029  0.0164602  -0.024               0.9806
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.961 on 28 degrees of freedom
## Multiple R-squared:  0.782,  Adjusted R-squared:  0.7587
## F-statistic: 33.49 on 3 and 28 DF,  p-value: 0.000000002112
```

Things that have changed

- We now see the slope of the line of best fit in the **hp** row for Manual cars. The slope is not exactly the same as in the previous output, the it is very similar.
- The value of the intercept for the am line is the same value, but has been reversed. (-5.21 here, +5.21 above).

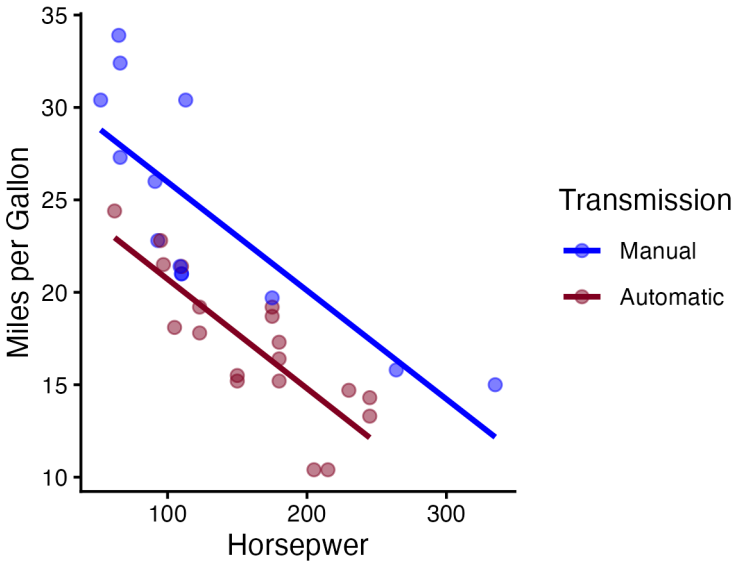
Things that remain the same

- The interaction term is exactly the same as above.
- The R-squared value (which represents the total variance accounted for) has not changed.

A picture's worth 1000 words. This is true all the time, and is an important heuristic to remember when conducting data analysis.

- It is often much easier to extract important information about the relationships that are present from figures rather than focusing only on statistical output.

```
b <- dat %>%
  ggplot(aes(x = hp, y = mpg, colour = am, group = am)) +
  geom_point(size = 2, alpha = 0.5) +
  geom_smooth(method = "lm", se = FALSE) +
  scale_colour_manual(values = c("blue", "#800020")) +
  theme_classic() +
  labs(
    y = "Miles per Gallon",
    x = "Horsepwer",
    colour = "Transmission",
    group = "Transmission"
  )

ggsave("chart_2.png", b, height = 3, width = 4, dpi = 300)
knitr::include_graphics("chart_2.png")
```



- The chart shows that there is a strong negative relationship between horsepower and fuel efficiency.
- As horsepower goes up, fuel efficiency goes down.
- The slopes of the lines of best fit for automatic and manual cars look very similar, which maps on to the non-significant interaction term in the regression model.

Week 4: September 25 2025

```
library(tidyverse)
```

```
dat <- mtcars
?mtcars
```

Good Predictors

- mpg –> becomes n.s. when wt. added
- drat –> Is sig over and above mpg; predictive value of mpg can be explained by drat (but the reverse is not true)
- wt –> predictive value of drat is ooverlapping with predictive value of wt.
- gear –> both gear and mpg are n.s. when modelled together

Bad Predictors

- cyl
- disp
- hp
- qsec
- vs
- carb

```
m <- glm(am ~ drat, data = dat, family = binomial)
summary(m)
```

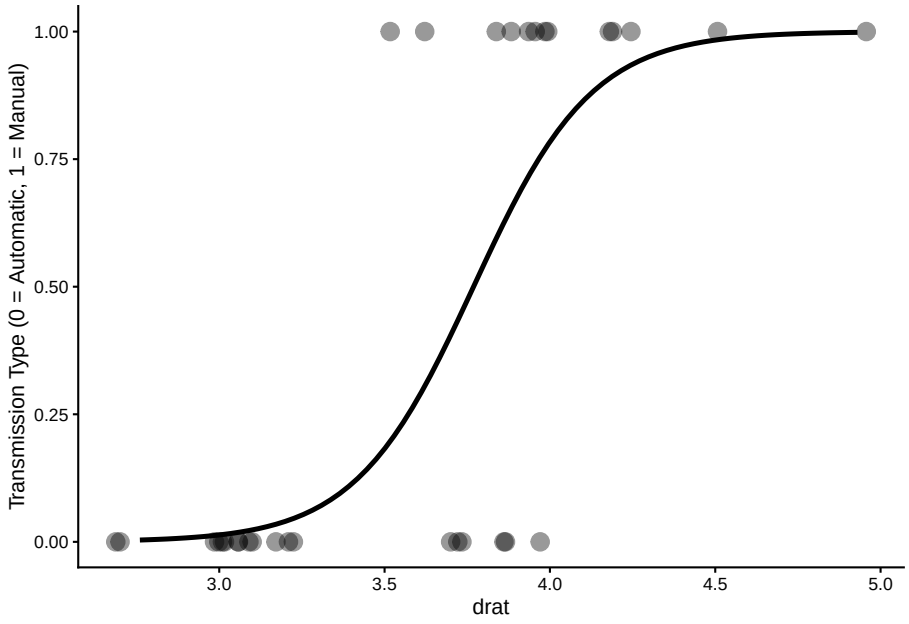
```
##
## Call:
## glm(formula = am ~ drat, family = binomial, data = dat)
##
## Coefficients:
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -21.021      7.838  -2.682  0.00732 **
## drat           5.577      2.063   2.704  0.00685 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 43.23  on 31  degrees of freedom
## Residual deviance: 21.65  on 30  degrees of freedom
## AIC: 25.65
##
## Number of Fisher Scoring iterations: 6
```

```
dat %>%
ggplot(aes(x = drat, y = am)) +
  geom_jitter(height = 0, width = 0.1, size = 4, alpha = 0.4) +
  stat_smooth(
    aes(x = drat, y = am, group = 1),
    method = "glm",
    method.args = list(family = binomial),
    se = FALSE,
    colour = "black",
    size = 1.2,
    inherit.aes = FALSE
  ) +
  theme_classic() +
  labs(
    y = "Transmission Type (0 = Automatic, 1 = Manual)"
  )
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




```
options(scipen = 999)

knitr::kable(round(exp(coef(m)), digits = 5))
```

	x
(Intercept)	0.0000
drat	264.3723

```
head(dat)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
library(mosaic)
```

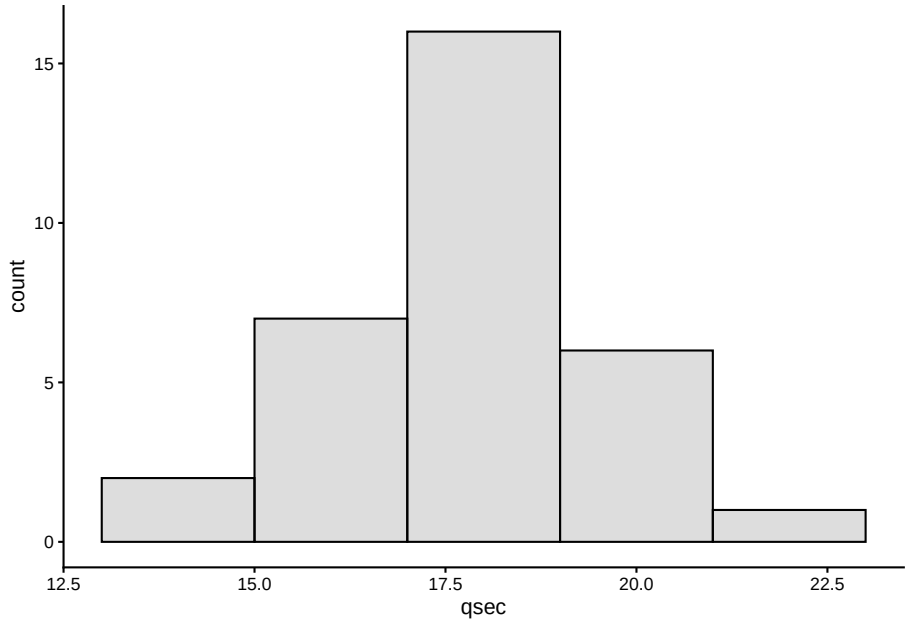
```
## Registered S3 method overwritten by 'mosaic':
##   method                           from
##   fortify.SpatialPolygonsDataFrame ggplot2
```

```
##
## The 'mosaic' package masks several functions from core packages in order to add
## additional features.  The original behavior of these functions should not be affecte


##
## Attaching package: 'mosaic'


## The following object is masked from 'package:Matrix':
##
##     mean


## The following objects are masked from 'package:dplyr':
##
##     count, do, tally


## The following object is masked from 'package:purrr':
##
##     cross


## The following object is masked from 'package:ggplot2':
##
##     stat


## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cor.test, cov, fivenum, IQR, median, prop.test,
##     quantile, sd, t.test, var


## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum
```

```
favstats(dat$qsec)
```

```
##   min      Q1 median   Q3  max     mean       sd  n missing
##  14.5 16.8925  17.71 18.9 22.9 17.84875 1.786943 32       0
```

```
ggplot(data = dat, aes(x = qsec)) +
  geom_histogram(binwidth = 2, alpha = 0.2, colour = "black") +
  theme_classic()
```



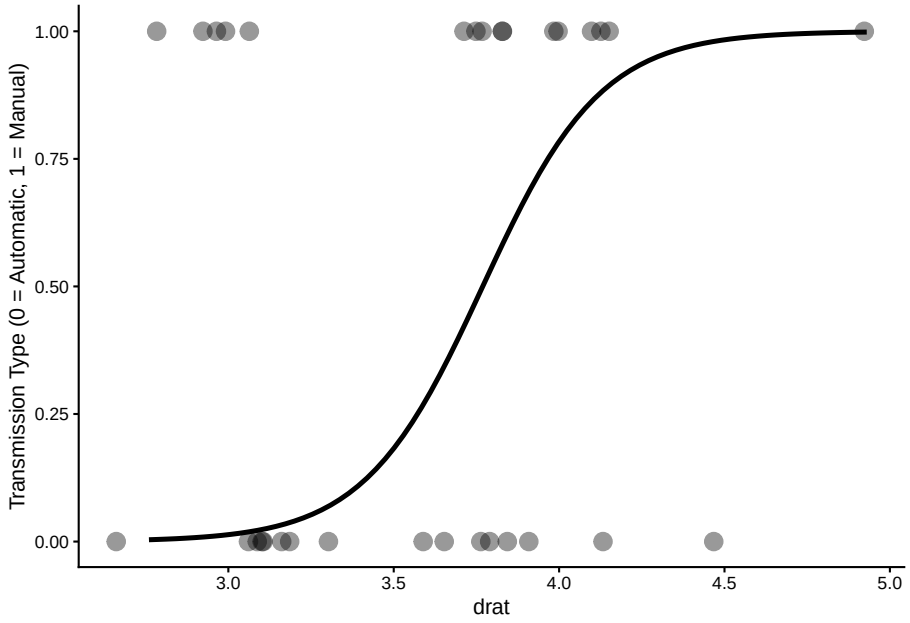

```
dat <- dat %>%
  mutate(qsec_cat = case_when(
    qsec > 17.71  ~ 1,
    qsec < 17.71  ~ 0
  ))

m2 <- glm(qsec_cat ~ drat, data = dat, family = binomial)
summary(m2)
```

```
##
## Call:
## glm(formula = qsec_cat ~ drat, family = binomial, data = dat)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.8792     2.5636  -1.123    0.261
## drat          0.8010     0.7067   1.133    0.257
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 44.361  on 31  degrees of freedom
## Residual deviance: 43.011  on 30  degrees of freedom
## AIC: 47.011
##
```

```
## Number of Fisher Scoring iterations: 4
```

```
dat %>%
ggplot(aes(x = drat, y = qsec_cat)) +
  geom_jitter(height = 0, width = 0.1, size = 4, alpha = 0.4) +
  stat_smooth(
    aes(x = drat, y = am, group = 1),
    method = "glm",
    method.args = list(family = binomial),
    se = FALSE,
    colour = "black",
    size = 1.2,
    inherit.aes = FALSE
  ) +
  theme_classic() +
  labs(
    y = "Transmission Type (0 = Automatic, 1 = Manual)"
  )
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




```
knitr::kable(round(exp(coef(m2)), digits = 5))
```

	x
(Intercept)	0.05618
drat	2.22772

```
head(dat)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb qsec_cat
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4        0
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4        0
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1        1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1        1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2        0
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1        1
```

```
?cut
```

Week 5: October 2 2025

Today's Topic: Bootstraping

Bootstrapping is a resampling technique that allows us to test the ***robustness*** of a point estimate (such as a mean or a regression slope). The idea is that the dataset is "resampled", usually around 5000 times **with** replacement. In this way, we treat the dataset as a population and repeatedly sample from it. The bootstrapped estimates. can then be treated as their own distribution. Bootstraping can either reinforce or refute the original point estimate.

Example #1: Bootstrapping a Mean

To demonstrate, I will simulate skewed data, look at the mean, then see the bootstrap distribution. Suppose that these data represent time to deliver pizzas from a pizza shop.

```
set.seed(1) # For a reproducible example
# Simulate skewed data
x <- rlnorm(60, 3, 0.6)
```

- This code generates a dataframe `x` that is right-skewed.
- Outliers can have a heavy influence on descriptive statistics such as the mean value.

Today I will generate charts using **Base R** instead of ggplot. Base R charts are typically less visually pleasing than ggplots, and they are not as customizeable. However, base R charts are quick to produce and are useful to know about. Especially if you are exploring a dataset privately (i.e., not preparing a report for a stakeholder), it is sometimes preferable to be able to ***quickly*** create different visualizations.

```
hist(x, main = "Raw Data", xlab = "Minutes to Deliver")
abline(v = mean(x), col = "red", lwd = 2)
```

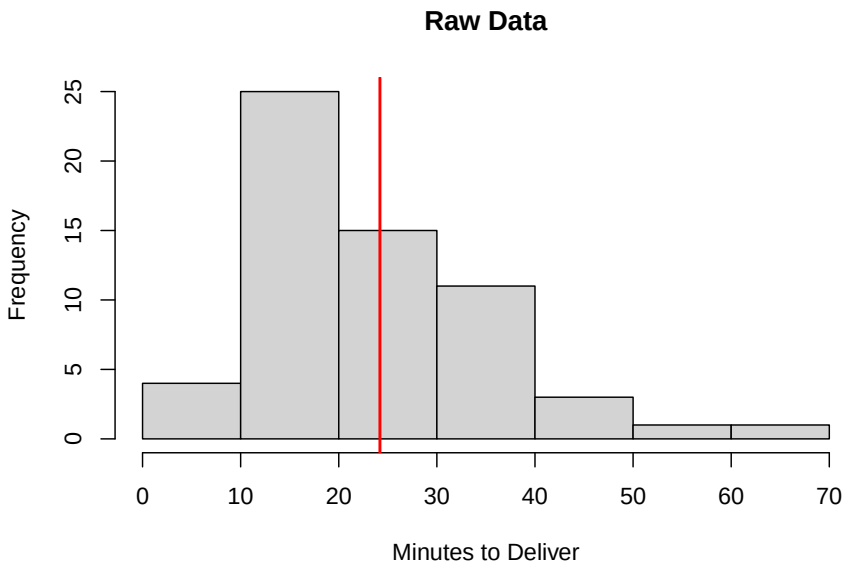


Figure 1. The distribution of delivery timees is right-skewed. Most deliveries take under 40 minutes, but a few take longer – up to 70 minutes – to complete. The vertical red line represents the mean delivery time of ~ 24 minutes.

We can also ask R to calculate and print out the exact mean value:

```
mean(x)
```

```
## [1] 24.19566
```

- The average delivery takes 24.19 minutes to complete.

Especially because the distribution of delivery times is right-skewed, we cannot comment on how ***robust*** the mean estimate is. Is it a good representation of the average delivery time? Or are the outlying times creating a skewed estimate? We can use bootstrapping to calculate a more robust estimate:

```
boot_means <- replicate(2000, mean(sample(x, replace = TRUE)))
```

- This code resamples the data `x` 2000 times with replacement.

- Each time the dataset is resampled, the mean delivery time is calculated.
- The bootstrapping process generates 2000 mean estimates based on the 2000 samples that were taken.

We can then plot a histogram of the bootstrapped means to understand the shape of the bootstrapped distribution of estimates:

```
hist(boot_means, main = "Bootstrap Distribution of the Mean Delivery Time",
     xlab = "Bootstrapped Means")
abline(v = mean(x), col = "red", lwd = 2)
```

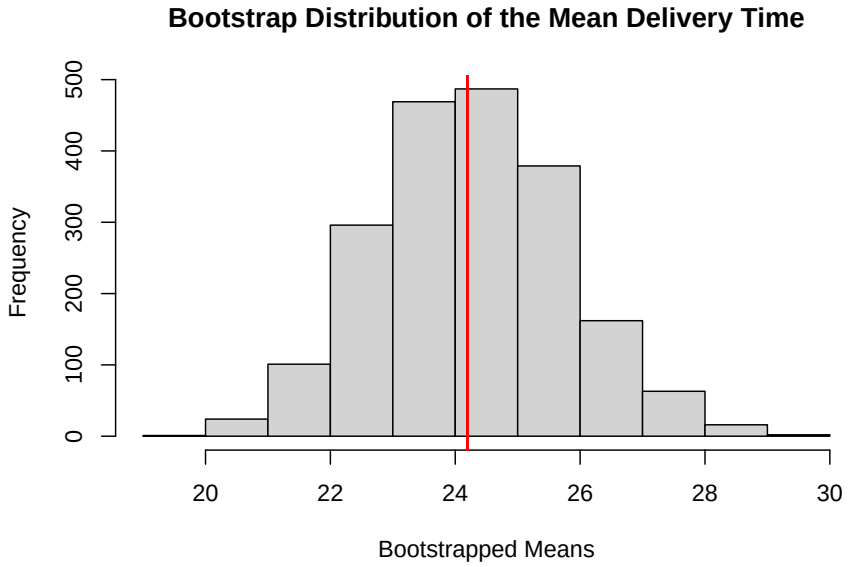


Figure 2. Estimates of mean delivery time taken from 2000 samples. The red vertical line represents the mean delivery time from the original dataset.

- In this case, the distribution of bootstrapped means is centered around the original point estimate.
- Therefore, the bootstrapping procedure increases our confidence in the original point estimate.

Example #2: Bootstrapping a Regression Slope

To demonstrate bootstrapping a regression coefficient, I will simulate a new set of fictitious data. The code to generate the data is set up to create a scenario

where there is a positive linear relationship between x and y, but that linear relationship depends on a few outlying datapoints. The goal here is to showcase a scenario where bootstrapping does not increase our confidence in the originial point estimate.

```
# For reproducibility
set.seed(123)

# Generate two vectors: x and y to model
x <- c(rexp(40, rate = 0.1), 50, 60)      #
y <- 5 + 0.5*x + rnorm(length(x), 0, sd = 5 + 0.2*x)

# Fit regression
fit <- lm(y ~ x)

# Print regression results
summary(fit)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -20.4208  -3.9460  -0.5488   3.8842  20.9524
##
## Coefficients:
##             Estimate Std. Error t value   Pr(>|t|)
## (Intercept)  5.15729    1.66402   3.099    0.00354 **
## x            0.49185    0.09637   5.104 0.00000851 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.882 on 40 degrees of freedom
## Multiple R-squared:  0.3944, Adjusted R-squared:  0.3792
## F-statistic: 26.05 on 1 and 40 DF,  p-value: 0.000008507
```

- There is a significant relationship between x and y in this example ($p < 0.001$).
- A 1-unit increase in x is associated with a 0.49-unit increase in y.
- Variance in x accounts for 39% of variance in y ($R^2 = 0.3944$)

As I've mentioned previously, a picture is always worth 1000 words in analytics, so let's have a look at the scatterplot:

```r
# Make a Scatterplot
plot(x, y, main = "Skewed data with leverage points")

# Add the Regression line
abline(fit, col = "red", lwd = 2)
```

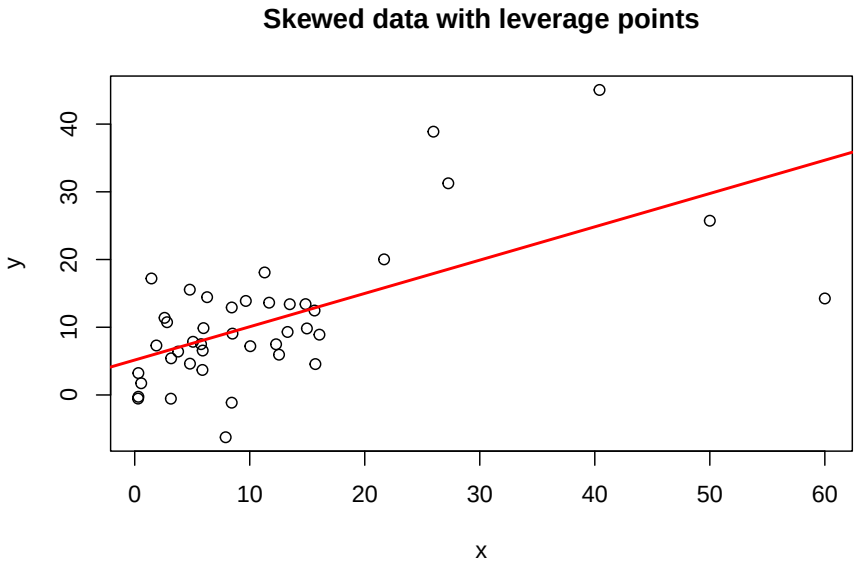


Figure 3. Linear relationship between `x` and `y`. Note that there are a few outlying datapoints towards the right of the x-axis that seem to be influencing the upward slope of the regression line.

```r
confint(fit)["x", ]
```

```
##     2.5 %    97.5 %
## 0.2970743 0.6866348
```

- The 95% confidence interval around the slope estimate is [0.30, 0.69]. Based on these data, we would expect the slope of the line of best fit to fall between 0.3 and 0.69 95% of the time, if the sampling procedure were to be repeated.

We might want to bootstrap the estimate to get a good idea of how ***robust*** the point estimate for the slope is.

```
boot_slopes <- replicate(2000, coef(lm(y[sample(length(x), replace=TRUE)] ~ x[sample(le
```

- This code resamples the dataset 2000 times (with replacement) and computes the coefficient of the line of best fit between `x` and `y` each time.
- The result is a dataframe `boot_slopes` with 2000 estimates of the slope of the relationship between `x` and `y`.

We can then visualize the distribution of bootstrapped slopes to see how closely they map on to the original estimate of the slope that we computed above.

```
hist(boot_slopes, breaks = 30, main = "Bootstrap slopes", xlab = "Slope")
abline(v = coef(fit)[2], col = "red", lwd = 2)
```

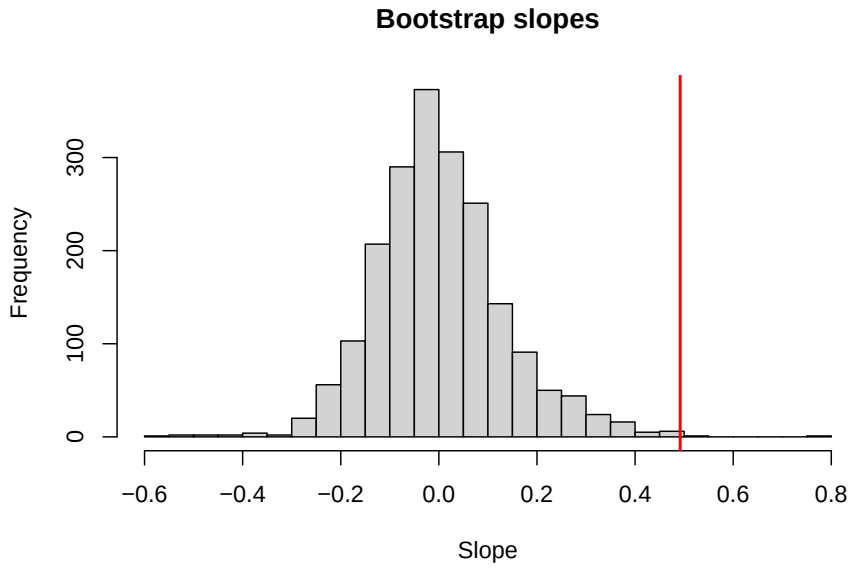


Figure 4. Distribution of bootstrapped slopes taken from 2000 samples. The red vertical line represents the original slope that we computed from the data above ($\beta = 0.49$).

- The bootstrapped distribution of slopes is centered around 0.
- A slope of zero in a regression model indicates a flat horizontal line of best fit (i.e., no linear relationship between `x` and `y`).
- Here, the bootstrapping procedure has indicated that our original point estimate ***is not*** robust.

We could also use the `quantile` command from base R to find the boundaries of where 95% of the bootstrapped slopes fall.

```
quantile(boot_slopes, c(.025, .975))
```

```
##       2.5%      97.5%
## -0.2309303  0.3038000
```

- 95% of the bootstrapped slopes fall between -0.23 and 0.30.
- This further emphasizes that the relationship between `x` and `y` is not robust in our dataset.

Week 6: October 9 2025

Get Data

Read in the datafile that I posted on Quercus under the "Lecture_6" slides. These are ficticious data about how one specific product listed on Amazon sells. I've also included a seperate "codebook" file for your convenience, which provides descriptions of each column in the data.csv file.

```
dat <- read_csv("data.csv")
```

Run Descrptives

Before jumping in to model generation, it may be a good idea to run a few quick commands to check out the dataset:

Print Data Preview

```
# Show me the first 6 rows of data
knitr::kable(head(dat))
```

sales	price	marketing	competitor_price	traffic	season
703	10.25	7.945	9.70	2898	Q1
547	11.00	10.396	10.51	2320	Q1
678	11.41	9.528	11.19	2597	Q1
483	11.38	9.621	11.00	2341	Q1
721	11.05	5.441	9.21	2476	Q2
733	11.77	11.562	11.88	2748	Q4

- Each row is a week in this dataset

- We have the number of sales, the price of the item, how much we spent on marketing that week (in 1000's of $), what a competator has a similar product listed for, traffic (the number of people that clicked on the listing) and the season (Q1 - Q4)
- All the variables except for season are numeric in nature.

Summary Statistics

```
# Show me summaries of each column
summary(dat)
```

```
##      sales          price          marketing      competitor_price
##  Min.   : 307   Min.   : 8.48   Min.   : 4.527   Min.   : 5.95
##  1st Qu.: 642   1st Qu.:10.25   1st Qu.: 9.752   1st Qu.: 9.13
##  Median : 742   Median :10.76   Median :12.207   Median :10.02
##  Mean   : 754   Mean   :10.74   Mean   :12.970   Mean   : 9.96
##  3rd Qu.: 872   3rd Qu.:11.29   3rd Qu.:15.549   3rd Qu.:10.79
##  Max.   :1370   Max.   :12.91   Max.   :41.051   Max.   :14.19
##     traffic        season
##  Min.   :1610   Length:600
##  1st Qu.:2354   Class :character
##  Median :2589   Mode  :character
##  Mean   :2571
##  3rd Qu.:2775
##  Max.   :3448
```

- We can glance through these summaries to see descriptive information about each column
- Take now of how spread out the scores in each column are to get a feel for the data.

Data Structure

```
# Show me the structure of the dataset
str(dat)
```

```
## spc_tbl_ [600 x 6] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ sales           : num [1:600] 703 547 678 483 721 ...
##  $ price           : num [1:600] 10.2 11 11.4 11.4 11.1 ...
```

```
##  $ marketing       : num [1:600] 7.95 10.4 9.53 9.62 5.44 ...
##  $ competitor_price: num [1:600] 9.7 10.51 11.19 11 9.21 ...
##  $ traffic         : num [1:600] 2898 2320 2597 2341 2476 ...
##  $ season          : chr [1:600] "Q1" "Q1" "Q1" "Q1" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   sales = col_double(),
##   ..   price = col_double(),
##   ..   marketing = col_double(),
##   ..   competitor_price = col_double(),
##   ..   traffic = col_double(),
##   ..   season = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

- Similar to what we see in the working environment
- Provides each column header, the type of data contained, and a few example values.

Histograms

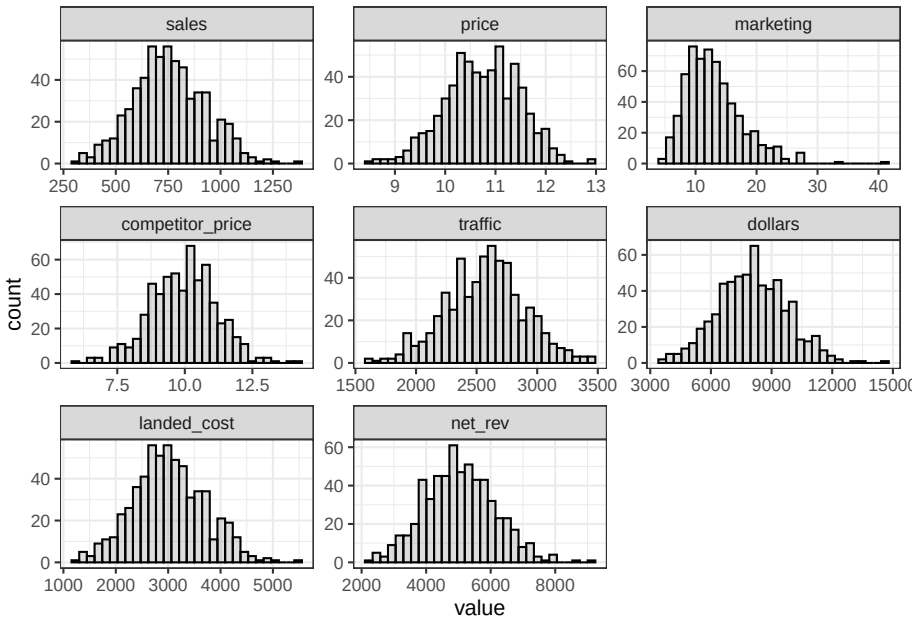
Histograms convey the same information as the summary tables above. We can estimate the mean, the range, and understand the spread of the scores by looking at the shapes of the distributions.

- I find that the visual pictures of the histograms are easier to understand than the values in the summary stats.
- e.g., we sell an average of about 750 units per week. Or the amount we spend on marketing is right-skewed.
- Looking at these charts also makes me wonder what we would see if we computed $ earned per week. This would make sense to do, because we really care about how much we earn, not how many units we sell.

We can compute a new column called `dollars`, which will contain information about how much we earned each week.

```
dat$dollars <- dat$sales * dat$price
dat$landed_cost <- 4 * dat$sales
dat$net_rev <- dat$dollars - dat$landed_cost
```

```
dat %>%
  select(-"season") %>%
  mutate(week = c(1:600)) %>%
  melt(id.vars = "week") %>%
  ggplot(aes(x = value)) +
  geom_histogram(alpha = 0.2, colour = "Black") +
  theme_bw() +
  facet_wrap(~variable, scales = "free")
```



Check Correlations

First, we need to remove the season column because it is non-numeric, and we can't compute correlations involving it.

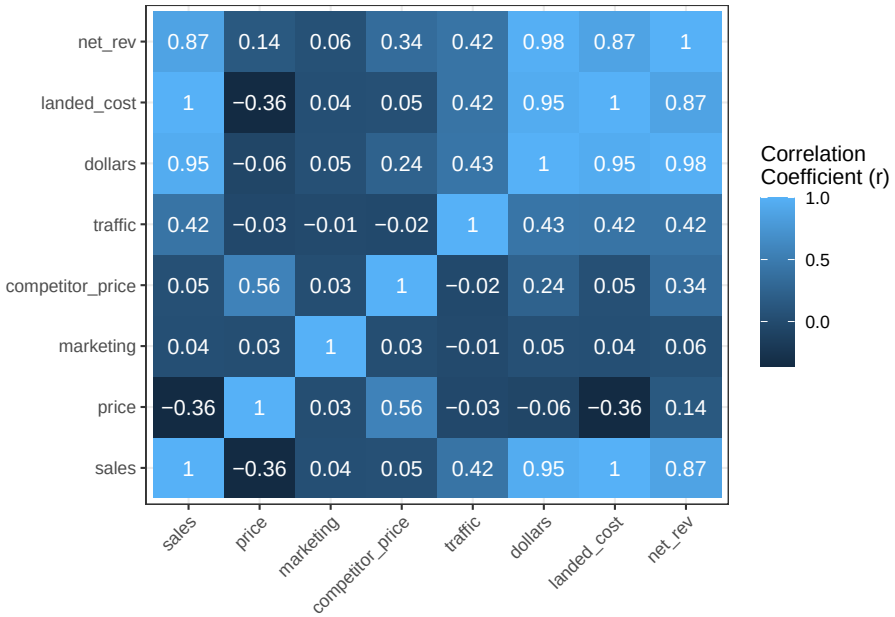
- I am assigning the data onto a new object, `a` to preserve the complete data stored in the object `dat`.
- This way, if I want to use that season column in some subsequent analysis, I won't have to re-load the data from the .csv file.

```
a <- dat %>%
  select(c(-"season"))
```

Sales will be our outcome variable of interest. From a business perspective, we generally care about predicting sales more than anything else! Have a good understanding of how many sales we could expect under different conditions might inform us of how we want to strategically price our item, how much inventory to have on hand at any given time, etc.

A nice way to view all the pairwise correlations between the variables together is to create a correlation matrix, then plot it as a heatmap. I will show you how to accomplish this in a single chunk of code using `ggplot()` below.

```
cor(a) %>%
  melt() %>%
  mutate(value = round(value, 2)) %>%
  ggplot(aes(x = Var1, y = Var2, fill = value, label = value)) +
  geom_tile() +
  geom_text(colour = "white") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        axis.title = element_blank()) +
  labs(
    fill = "Correlation \nCoefficient (r)"
  )
```



- There is a negative correlation between **sales** and **price** (when the price goes down, the sales go up).

- There is a positive correlation between **sales** and **traffic** (the more people click on the listing, the more sales).
- There is also a positive correlation between our item's price and the competitor's price (this is less useful to our modelling efforts than the two points written above).

Split the Data

First, we must split the existing data into a **training set** and a **test set**. We will randomly select 70% of the rows from the data to assign to the training set and the rest will be our test data.

```
set.seed(994)
train_index <- sample(1:nrow(dat), 0.7 * nrow(dat))
head(train_index)
```

```
## [1] 550 595 454 538 294 413
```

```
train <- dat[train_index, ]
test <- dat[-train_index, ]
```

Model Training

We will start with a "base model", then test whether adding additional variables to it further improves its ability to model the data. In this scenario, it makes sense to use `price ~ sales` as the base model.

```
options(scipen = 999)

model <- lm(sales ~ price, data = train)
summary(model)
```

```
##
## Call:
## lm(formula = sales ~ price, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -426.56 -111.95    1.38  107.05  615.48
##
```

```
## Coefficients:
##             Estimate Std. Error t value             Pr(>|t|)
## (Intercept)  1690.05     116.49  14.507 < 0.0000000000000002 ***
## price         -87.76      10.82  -8.111  0.00000000000000563 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 162.2 on 418 degrees of freedom
## Multiple R-squared:  0.136,  Adjusted R-squared:  0.1339
## F-statistic: 65.79 on 1 and 418 DF,  p-value: 0.00000000000000563
```

- A one-dollar increase in price is associated with a reduction in sales per week of 88 unites.
- The effect of price on sales is statistically significant with $p < 0.001$.
- The price of our item accounts for 13.6% of the variability in the number of sales we see across weeks ($R^2 = 0.136$)

```
model_2 <- lm(sales ~ price + traffic, data = train)
summary(model_2)
```

```
##
## Call:
## lm(formula = sales ~ price + traffic, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -379.90 -110.04   -7.81   99.18  464.21
##
## Coefficients:
##               Estimate Std. Error t value             Pr(>|t|)
## (Intercept) 1068.95025  121.62641   8.789 < 0.0000000000000002 ***
## price        -82.22217    9.73864  -8.443 0.000000000000000518 ***
## traffic        0.21895    0.02183  10.031 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 145.8 on 417 degrees of freedom
## Multiple R-squared:  0.3039, Adjusted R-squared:  0.3006
## F-statistic: 91.04 on 2 and 417 DF,  p-value: < 0.00000000000000022
```

- The price of the item and the number of visits to the item's page are unique (independent) predictors of the number of sales (both $p < 0.001$)

- Together, the item's price and the number of clicks to the page account for 30% of the variability in sales per week.

We can formilly test whether the more complex model (involving both traffic and price) is **statistically** better than our base model (where price is the only predictor)

```
anova(model, model_2)
```

```
## Analysis of Variance Table
##
## Model 1: sales ~ price
## Model 2: sales ~ price + traffic
##   Res.Df      RSS Df Sum of Sq      F                Pr(>F)
## 1    418 11000454
## 2    417  8862079  1   2138375 100.62 < 0.00000000000000022 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- The analysis of variance (ANOVA) indicates that `model_2` performed significantly better than the base model ($p < 0.001$)

Let's try add another predictor from the data:

```
model_3 <- lm(sales ~ price + traffic + competitor_price, data = train)
summary(model_3)
```

```
##
## Call:
## lm(formula = sales ~ price + traffic + competitor_price, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -385.10  -99.98   -8.45   95.69  484.35
##
## Coefficients:
##                   Estimate Std. Error t value             Pr(>|t|)
## (Intercept)      1067.18656  115.10904   9.271 < 0.0000000000000002 ***
## price            -128.21046   11.29706 -11.349 < 0.0000000000000002 ***
## traffic             0.22437    0.02067  10.854 < 0.0000000000000002 ***
## competitor_price   48.36781    6.87059   7.040       0.000000000008 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 138 on 416 degrees of freedom
## Multiple R-squared:  0.378,  Adjusted R-squared:  0.3736
## F-statistic: 84.28 on 3 and 416 DF,  p-value: < 0.00000000000000022
```

- The item's price, the number of hits on our Amazon page, and the competitor's price are all significant unique predictors of the number of weekly sales (all $p < 0.001$)
- Combined, variability in these three predictors' levels account for 37.8% of the variability in number of sales.

Again, we can use the `anova` command to test whether adding the new predictor significantly improved model fit over our current best model:

```
anova(model_2, model_3)
```

```
## Analysis of Variance Table
##
## Model 1: sales ~ price + traffic
## Model 2: sales ~ price + traffic + competitor_price
##   Res.Df     RSS Df Sum of Sq      F            Pr(>F)
## 1    417 8862079
## 2    416 7918702  1    943378 49.559 0.000000000007996 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- Since $p < 0.001$, `model_3` does a significantly better job at fitting the data than does `model 2`

Ok, so let's make one more model with all four predictors entered:

```
model_4 <- lm(sales ~ price + traffic + competitor_price + marketing, data = train)
summary(model_4)
```

```
##
## Call:
## lm(formula = sales ~ price + traffic + competitor_price + marketing,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -373.43 -103.27   -6.93   99.14  481.66
##
## Coefficients:
```

```
##                     Estimate Std. Error t value             Pr(>|t|)
## (Intercept)      1047.59456  116.75866   8.972 < 0.0000000000000002 ***
## price            -127.78228   11.30510 -11.303 < 0.0000000000000002 ***
## traffic             0.22433    0.02067  10.852 < 0.0000000000000002 ***
## competitor_price   48.07517    6.87677   6.991       0.000000000011 ***
## marketing           1.39738    1.39510   1.002                0.317
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 138 on 415 degrees of freedom
## Multiple R-squared:  0.3795, Adjusted R-squared:  0.3736
## F-statistic: 63.46 on 4 and 415 DF,  p-value: < 0.00000000000000022
```

- The item's price, the number of visits to the page, and the competator's price are all significant unique predictors, over and above the influence of marketing $ spent (all $p < 0.001$)
- Marketing spend is not a significant unique predictor of sales.
- Combined, these predictors account for 37.95% of the variability in the number of weekly sales.

Compare `model_4` to `model_3`:

```
anova(model_3, model_4)
```

```
## Analysis of Variance Table
##
## Model 1: sales ~ price + traffic + competitor_price
## Model 2: sales ~ price + traffic + competitor_price + marketing
##   Res.Df     RSS Df Sum of Sq      F Pr(>F)
## 1    416 7918702
## 2    415 7899604  1     19097 1.0033 0.3171
```

- `model_4` is not a significantly better fit to the data than `model_3`
- Therefore, `model_3` performed the best out of the options that we tested.

Apply `model_3` to the Test Data

In order to investigate how well `model_3` will perform on data that it has never seen before, we can superimpose it over the test data and investigate how well the line fits:

```
predictions <- predict(model_3, newdata = test)
```

The key metric of model fit is the $RMSE$ (root mean squared error) which tells us the amount of *unexplained* variability:

```
sqrt(mean((test$sales - predictions)^2))
```

```
## [1] 127.142
```

- Here, $RMSE = 127.14$.
- To interpret the RMSE, we compare to the residual error that was leftover when that model was generated on the training dataset.

```
sqrt(mean(model_3$residuals^2))
```

```
## [1] 137.3101
```

- In this case, the model actually performed a little better on the test data than it did during training!
- This is very strong evidence that the current model will perform well predicting future sales.

Remake the models with net_rev as the DV

```
head(dat)
```

```
## # A tibble: 6 x 9
##   sales price marketing competitor_price traffic season dollars landed_cost
##   <dbl> <dbl>     <dbl>            <dbl>   <dbl> <chr>    <dbl>       <dbl>
## 1   703  10.2      7.94             9.7     2898 Q1       7206.        2812
## 2   547  11       10.4             10.5     2320 Q1       6017         2188
## 3   678  11.4      9.53            11.2     2597 Q1       7736.        2712
## 4   483  11.4      9.62            11       2341 Q1       5497.        1932
## 5   721  11.0      5.44             9.21    2476 Q2       7967.        2884
## 6   733  11.8     11.6             11.9     2748 Q4       8627.        2932
## # i 1 more variable: net_rev <dbl>
```

```r
cost_model <- lm(net_rev ~ traffic, data = dat)
summary(cost_model)
```

```
##
## Call:
## lm(formula = net_rev ~ traffic, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2517.2  -689.5   -50.6   631.3  3739.6
##
## Coefficients:
##              Estimate Std. Error t value             Pr(>|t|)
## (Intercept) 1455.5291   321.1477   4.532          0.00000705 ***
## traffic        1.3930     0.1239  11.242 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 988.4 on 598 degrees of freedom
## Multiple R-squared:  0.1745, Adjusted R-squared:  0.1731
## F-statistic: 126.4 on 1 and 598 DF,  p-value: < 0.00000000000000022
```

```r
cost_model_2 <- lm(net_rev ~ price + traffic + competitor_price, data = dat)
summary(cost_model_2)
```

```
##
## Call:
## lm(formula = net_rev ~ price + traffic + competitor_price, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2492.8  -615.9   -38.3   577.4  3306.2
##
## Coefficients:
##                  Estimate Std. Error t value            Pr(>|t|)
## (Intercept)      -953.6049   631.8273  -1.509               0.132
## price             -99.4490    61.6952  -1.612               0.108
## traffic             1.4113     0.1144  12.336 <0.0000000000000002 ***
## competitor_price  344.3674    36.8791   9.338 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 912.2 on 596 degrees of freedom
## Multiple R-squared:  0.2992, Adjusted R-squared:  0.2957
## F-statistic: 84.81 on 3 and 596 DF,  p-value: < 0.00000000000000022
```

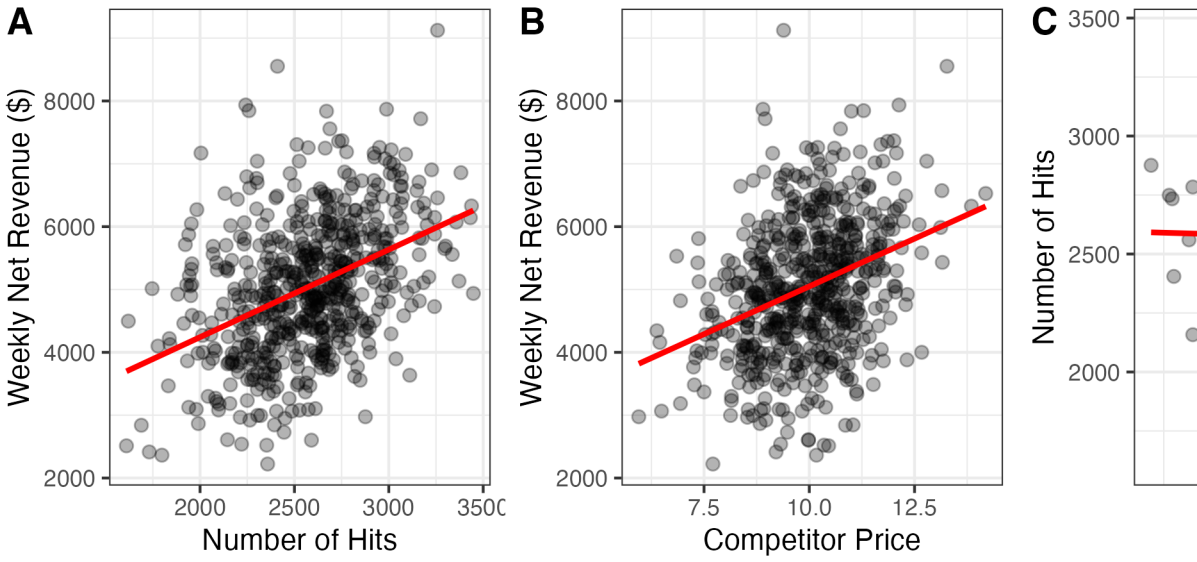
```r
cost_model_2 <- lm(net_rev ~ traffic + competitor_price + price, data = dat)
summary(cost_model_2)
```

```
##
## Call:
## lm(formula = net_rev ~ traffic + competitor_price + price, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2492.8  -615.9   -38.3   577.4  3306.2
##
## Coefficients:
##                   Estimate Std. Error t value            Pr(>|t|)
## (Intercept)      -953.6049   631.8273  -1.509               0.132
## traffic             1.4113     0.1144  12.336 <0.0000000000000002 ***
## competitor_price  344.3674    36.8791   9.338 <0.0000000000000002 ***
## price             -99.4490    61.6952  -1.612               0.108
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 912.2 on 596 degrees of freedom
## Multiple R-squared:  0.2992, Adjusted R-squared:  0.2957
## F-statistic: 84.81 on 3 and 596 DF,  p-value: < 0.00000000000000022
```

- The effects of traffic and the other guy's price remain significant even after controlling for our price point
- Our price point is not a significant unique predictor of revenue when controlling for the other factors in the model

```r
a <- dat %>%
  ggplot(aes(x = traffic, y = net_rev)) +
  geom_point(size = 2, alpha = 0.3) +
  geom_smooth(method = "lm", se = F, colour = "red") +
  theme_bw()+
  labs(
    x = "Number of Hits",
    y = "Weekly Net Revenue ($)"
  )

b <- dat %>%
  ggplot(aes(x = competitor_price, y = net_rev)) +
  geom_point(size = 2, alpha = 0.3) +
  geom_smooth(method = "lm", se = F, colour = "red") +
```

```
  theme_bw()+
  labs(
    x = "Competitor Price",
    y = "Weekly Net Revenue ($)"
  )

c <- dat %>%
  ggplot(aes(x = competitor_price, y = traffic)) +
  geom_point(size = 2, alpha = 0.3) +
  geom_smooth(method = "lm", se = F, colour = "red") +
  theme_bw()+
  labs(
    x = "Competitor Price",
    y = "Number of Hits"
  )

panel <- ggarrange(a,b,c,
          nrow = 1,
          labels = c("A","B","C"))

ggsave("Figs/panel.png", height = 3, width = 8, dpi = 300)
knitr::include_graphics("Figs/panel.png")
```



Week 7: October 16 2025

Week 8: October 23 2025

Penguins PCA Analysis

Data come from the `palmerpenguins` package

```
library(palmerpenguins)
library(tidyverse)
library(reshape2)
a <- penguins
```

The penguins dataset involves 344 observations, where each row is a penguin. For each penguin, we have some demographic information (e.g., the island where it was measured). We also have four columns of numeric data that represent each penguins' measurements:

1. `bill_length_mm`: Length of the penguin's bill
2. `bill_depth_mm`: "height" of the penguin's bill
3. `flipper_length_mm`: Length of the penguin's flippers
4. `body_mass_g`: Weight of the penguin in grams

We will use these four numeric columns to construct a PCA.

Exploratory Data Analysis

It is always a good idea to explore a new dataset a bit before beginning analysis.

Have a look at the data:

```
knitr::kable(head(a))
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	se
Adelie	Torgersen	39.1	18.7	181	3750	m
Adelie	Torgersen	39.5	17.4	186	3800	fe
Adelie	Torgersen	40.3	18.0	195	3250	fe
Adelie	Torgersen	NA	NA	NA	NA	N
Adelie	Torgersen	36.7	19.3	193	3450	fe
Adelie	Torgersen	39.3	20.6	190	3650	m

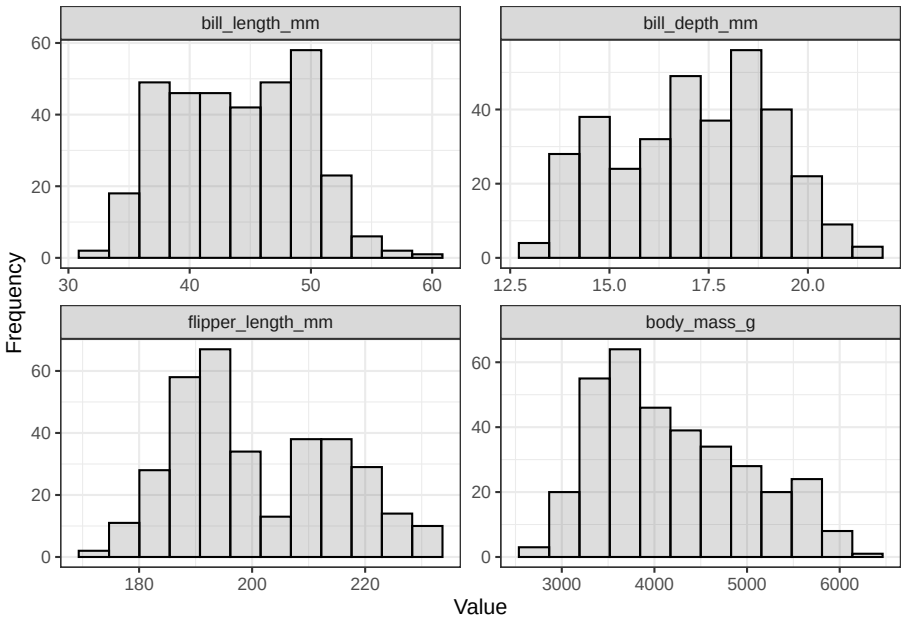
- Notice that there are some missing values in this dataset.

- Seeing values of `NA` is common in real-world analysis settings.

- We will have to do something about this during our analysis as we cannot compute correlations for columns that contain `NA` values

Histograms of the variables (raw values):

```
a %>%
  melt(id.vars = c("species","island","sex","year"))%>%
  ggplot(aes(x = value)) +
  geom_histogram(alpha = 0.2, colour = "black", bins = 12) +
  facet_wrap(~variable, scales = "free") +
  theme_bw() +
  labs(
    x = "Value",
    y = "Frequency"
  )
```



- The histograms show us the range of each variable and the distribution of data.
- Notice that the measurements for weight are much larger numeric values than the other variables.
- For PCA, we wouldn't want a column that happens to have larger values to dominate the analysis, so we need to standardize each column to have a mean value of 0 and a standard deviation of 1.

Standardize Variables

We can use the built in `scale` function to standardize variables. However, we cannot pass the entire dataframe to `scale` because some columns contain character strings. Only numeric columns can be transformed using `scale`. In the code below, I overwrote the numeric columns one at a time instead.

```
b <- a %>%
  na.omit()

b$bill_length_mm <- scale(b$bill_length_mm)
b$bill_depth_mm <- scale(b$bill_depth_mm)
b$flipper_length_mm <- scale(b$flipper_length_mm)
```

```
b$body_mass_g <- scale(b$body_mass_g)

knitr::kable(head(b))
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	se
Adelie	Torgersen	-0.8946955	0.7795590	-1.4246077	-0.5676206	m
Adelie	Torgersen	-0.8215515	0.1194043	-1.0678666	-0.5055254	fe
Adelie	Torgersen	-0.6752636	0.4240910	-0.4257325	-1.1885721	fe
Adelie	Torgersen	-1.3335592	1.0842457	-0.5684290	-0.9401915	fe
Adelie	Torgersen	-0.8581235	1.7444004	-0.7824736	-0.6918109	m
Adelie	Torgersen	-0.9312674	0.3225288	-1.4246077	-0.7228585	fe

- The values in the numeric columns have been adjusted.

We could also run summary statistics to ensure that the transformed variables are standardized in the way that we want them:

```
b %>%
  melt(id.vars = c("species","island","sex","year")) %>%
  group_by(variable) %>%
  summarise(
    n = n(),
    mean = mean(value),
    sd = sd(value),
    min = round(min(value),2),
    max = round(max(value),2)
  ) %>%
  knitr::kable()
```

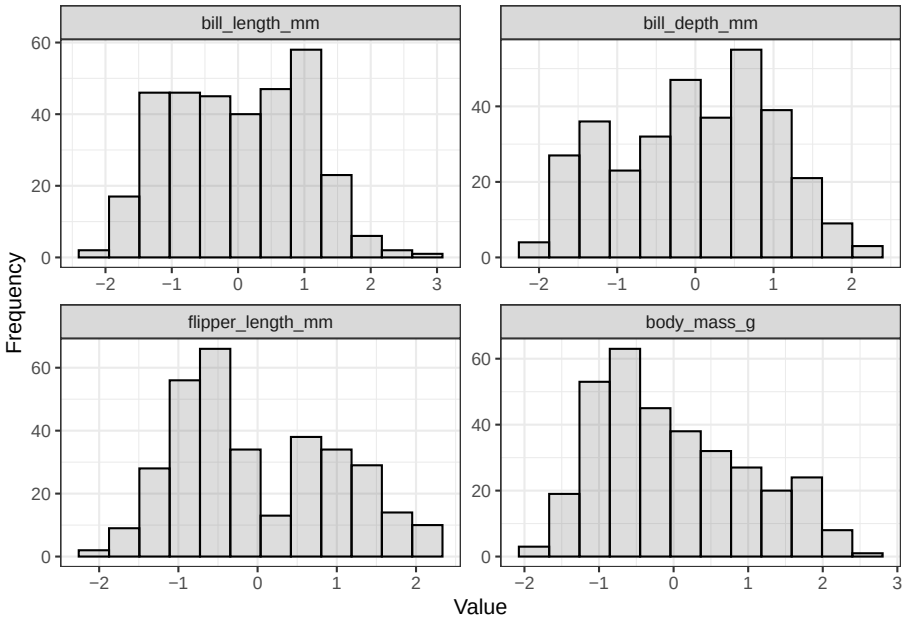
variable	n	mean	sd	min	max
bill_length_mm	333	0	1	-2.17	2.85
bill_depth_mm	333	0	1	-2.06	2.20
flipper_length_mm	333	0	1	-2.07	2.14
body_mass_g	333	0	1	-1.87	2.60

- Looks good! Each of the numeric columns now has a mean of 0 and a standard deviation of 1.

Make histograms of scaled variables:

```
b %>%
  melt(id.vars = c("species","island","sex","year"))%>%
  ggplot(aes(x = value)) +
```

```
  geom_histogram(alpha = 0.2, colour = "black", bins = 12) +
  facet_wrap(~variable, scales = "free") +
  theme_bw() +
  labs(
    x = "Value",
    y = "Frequency"
  )
```



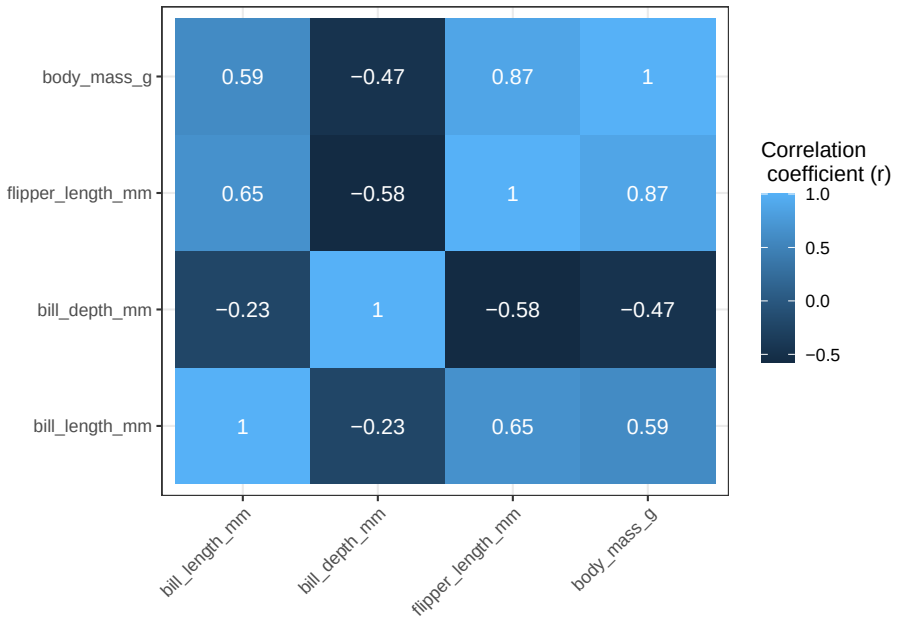
- Notice how the shapes of the distributions are the same as above when I plotted the raw data
- Transforming a variable will not change the shape of the distribution! Only the x-axis is re-scaled.
- Notice that now, the distributions for each varaible center around 0 and have a range of approximately -2 to +2.
- Now that all of the variables are on the same scale, they are ready to be used in PCA.

Make correlation matrix:

Before computing the PCA, it is useful to generate a correlation matrix to get a feel for which variables covary. This is especially useful when working with

many columns of data.

```
b %>%
  select(c("bill_length_mm", "bill_depth_mm", "flipper_length_mm", "body_mass_g")) %>%
  cor() %>%
  melt() %>%
  ggplot(aes(x = Var1, y = Var2, fill = value, label = round(value,2))) +
  geom_tile() +
  geom_text(colour = "white")+
  theme_bw() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    axis.title = element_blank()) +
  labs(
    fill = "Correlation \n coefficient (r)"
  )
```

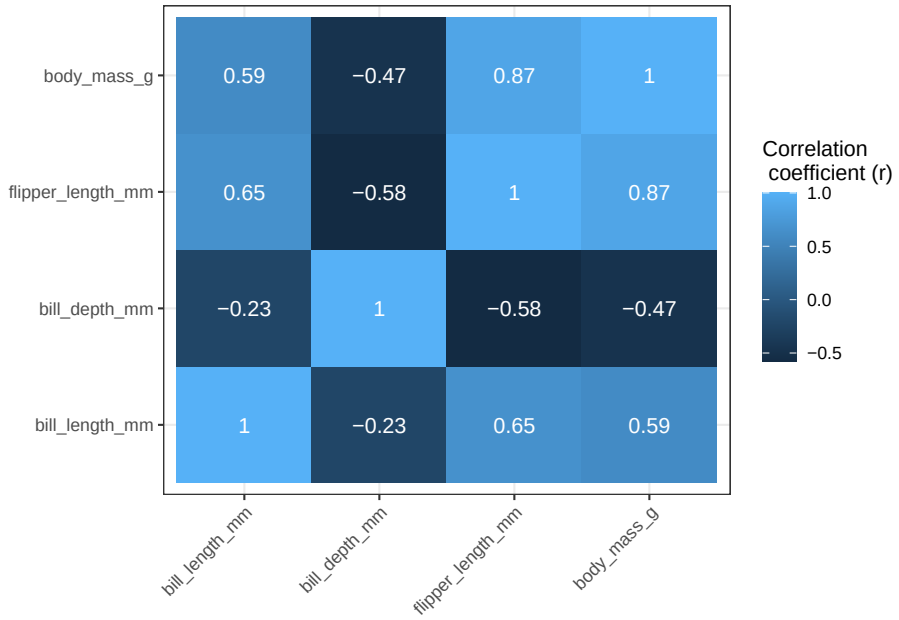


- There are strong positive correlations between body mass and bill length (r = 0.59), and between body mass and flipper length (r = 0.87).
- There is a strong positive correlation between flipper length and bill length (r = 0.65)
- Bill depth is negatively correlated with body mass and flipper length (r = -0.47, -0.58, respectively)

- There is a moderate negative correlation between bill depth and bill length (r = -0.23)

For illustration, I will compute the same correlation matrix on the raw (unstandardized) data as well:

```
a %>%
  na.omit() %>%
  select(c("bill_length_mm", "bill_depth_mm", "flipper_length_mm", "body_mass_g")) %>%
  cor() %>%
  melt() %>%
  mutate(value = round(value,2)) %>%
  ggplot(aes(x = Var1, y = Var2, fill = value, label = value)) +
  geom_tile() +
  geom_text(colour = "white")+
  theme_bw() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    axis.title = element_blank()) +
  labs(
    fill = "Correlation \n coefficient (r)"
  )
```



- Notice that the correlations are exactly the same as above

- Transforming the variables does not alter the nature of the relationships between them (just as it did not alter the nature of the distributions when we made histograms)

Run PCA

In order to compute the PCA we must first select only the numeric columns that will be used. I will assign a new object `c` that drops the other four columns in the data that contain demographic information.

```
c <- b[ ,c(3:6)]
pca_res <- prcomp(c)
summary(pca_res)
```

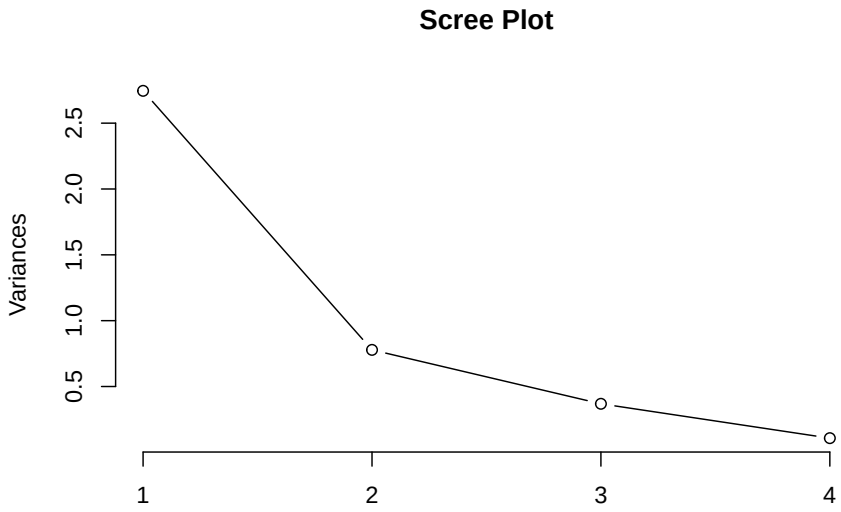
```
## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.6569 0.8821 0.60716 0.32846
## Proportion of Variance 0.6863 0.1945 0.09216 0.02697
## Cumulative Proportion  0.6863 0.8809 0.97303 1.00000
```

- The first principal component accounts for 68% of variance in the data
- The second principal component accounts for an additional 19% of variance (so the combination of the first and second PCs account for 88% of variance)

Choose How Many PCs to Retain

We typically don't want to look at all of the principal components that the model generates. Instead, we want to select to top few for further investigation. Selecting the first 2-3 is often a good way to balance simplicity and explanatory power. One way that we can visualzie how much explanatory power each PC adds is to generate the Scree plot:

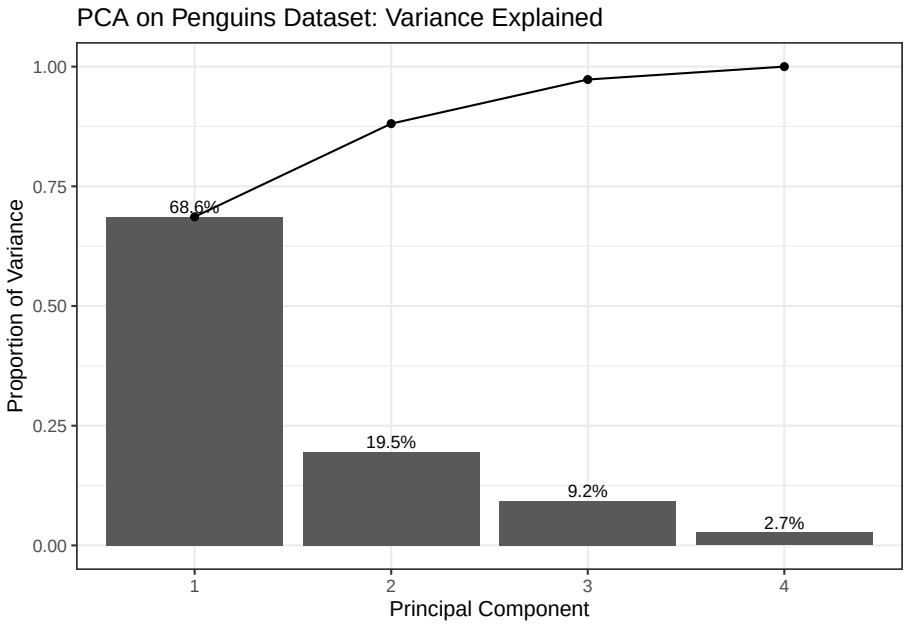
```
plot(pca_res, type = "l", main = "Scree Plot")
```



- The quick n dirty base R scree plot shows that the first principal component explains the most variance, the second also seems to have reasonable explanatory power, and that the 3rd and 4th PCs don't add much more.

I also showed you some more complex code in lecture that can be used to better visualize the variance explained by each PC:

```
var_expl <- pca_res$sdev^2
prop_var <- var_expl / sum(var_expl)
scree_df <- tibble(PC = factor(seq_along(prop_var)),
                   prop_var = prop_var,
                   cum_prop = cumsum(prop_var))
ggplot(scree_df, aes(PC, prop_var)) +
  geom_col() +
  geom_text(aes(label = scales::percent(prop_var, accuracy = 0.1)),
            vjust = -0.3, size = 3) +
  geom_line(aes(y = cum_prop, group = 1)) +
  geom_point(aes(y = cum_prop)) +
  labs(title = "PCA on Penguins Dataset: Variance Explained",
       x = "Principal Component", y = "Proportion of Variance") +
  theme_bw()
```



- The line on the chart shows cumulative proportion of variance explained
- I will select only the first and second PCs for additional analysis based on the scree plot

Assess Loadings

Next, we can look into how the original variables map onto the first and second principal components.

```
loadings <- as_tibble(pca_res$rotation, rownames = "variable")
```

Assess Loadings onto the principal components:

```
loadings %>%
  select(variable, PC1) %>%
  arrange(desc(abs(PC1))) %>%
  knitr::kable(digits = 2)
```

variable	PC1
flipper_length_mm	0.58
body_mass_g	0.55
bill_length_mm	0.45
bill_depth_mm	-0.40

- Length of the flippers, body mass, and bill length all load positively onto the first PC
- Bill depth loads negatively onto the first PC
- It seems like PC1 might be capturing something related to **overall penguin size**

```
loadings %>%
  select(variable, PC2) %>%
  arrange(desc(abs(PC2))) %>%
  knitr::kable(digits = 2)
```

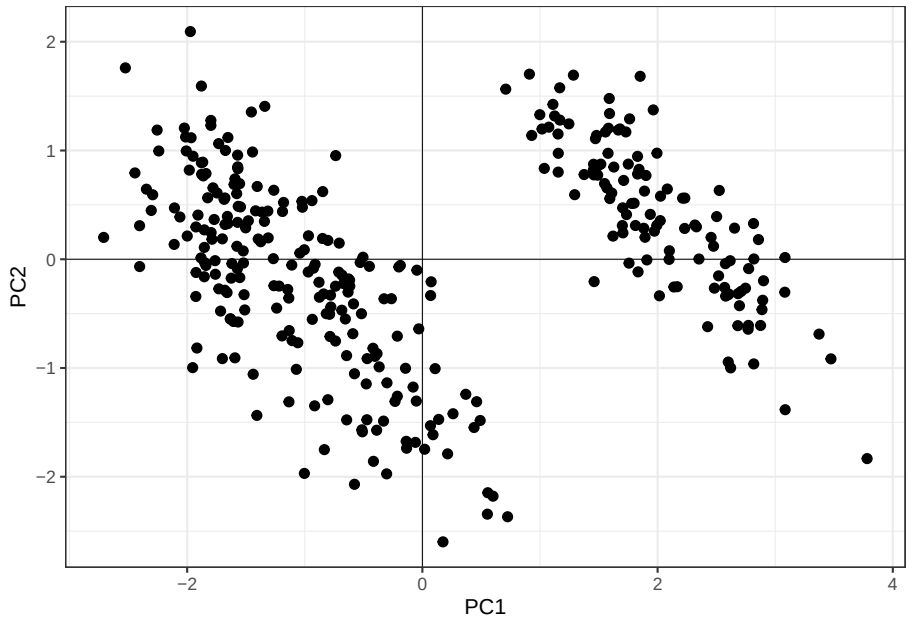
variable	PC2
bill_depth_mm	-0.80
bill_length_mm	-0.60
body_mass_g	-0.08
flipper_length_mm	-0.01

- The two variables related to bill measurements (length and depth) have strong negative relationships to the second PC
- The other to variables (weight and flipper length) have very small negative relationships to PC2 as well.
- PC2 seems to capture **bill characteristics**

Plot the Raw Data in PC Space

The two principal components that I selected for further investigation represent new "axes" that capture a summary of the original variables entered. To visualize how the raw data map to this space, we can generate a two dimensional plot where PC1 is mapped to the x-axis and PC2 is plotted on the y-axis. We can then add datapoints for all the individual penguins and see how they map to our two-dimensional PCA.

```
scores <- as_tibble(pca_res$x)

ggplot(scores, aes(PC1, PC2)) +
  geom_hline(yintercept = 0, linewidth = 0.2) +
  geom_vline(xintercept = 0, linewidth = 0.2) +
  geom_point(size = 2) +
  theme_bw()
```

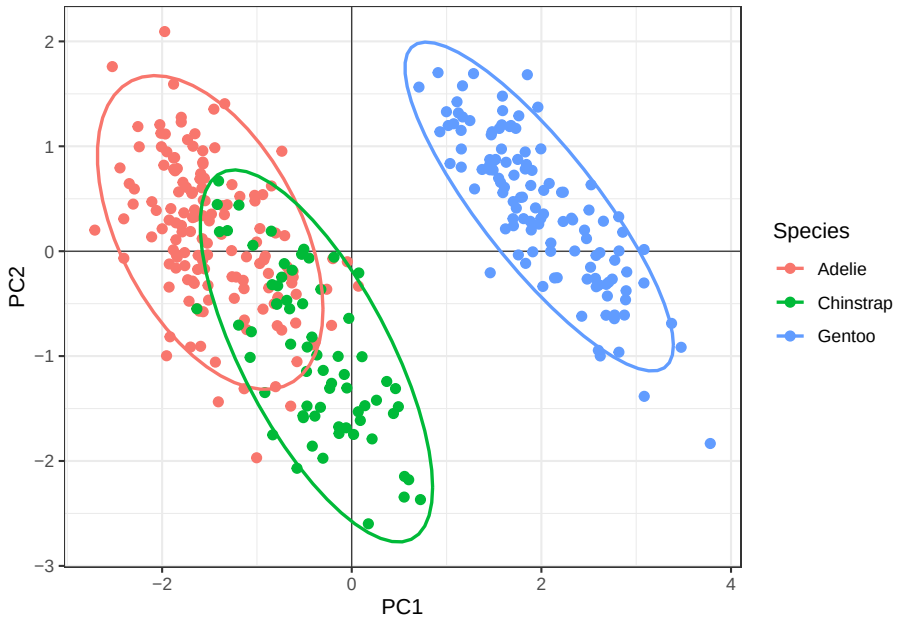


- There is a clear clustering of datapoints towards the right side of the x-axis.

- Remember, the `penguins` dataset also came with a column of data that represented their species.

Let's re-attach that column, colour the points by species, and see whether the clustering is related to their species!

```
scores$Species <- b$species
ggplot(scores, aes(PC1, PC2, colour = Species)) +
  geom_hline(yintercept = 0, linewidth = 0.2) +
  geom_vline(xintercept = 0, linewidth = 0.2) +
  geom_point(size = 2) +
  stat_ellipse(level = 0.95, linewidth = 0.8) +
  theme_bw()
```



- This looks great! It seems that the variance capture by PC1 separates Gentoo penguins from the other two species in the dataset.

- PC2 then separates the Adelie and Chinstrap penguins, although this separation is not as clean as we see in PC1 (i.e., there is still lots of overlap between the pink and green points.)

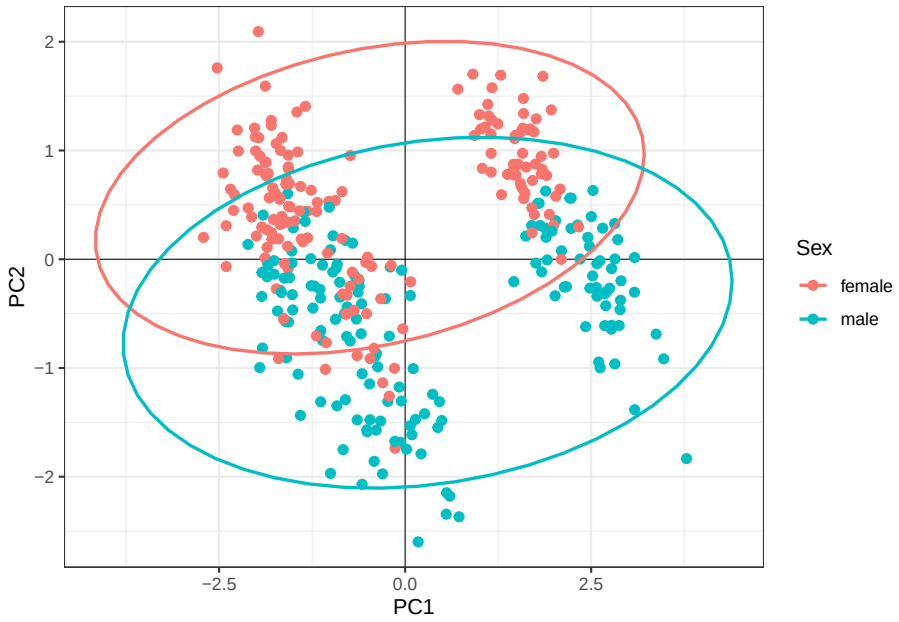
When I look up a photo of the three species of penguins, the Gentoo species does indeed seem larger than the other two types:

- When I look up a photo of the three species of penguins, the Gentoo species does indeed seem larger than the other two types

- This confirms my interpretation of what PC1 represents: It seems to be related to overall penguin size.

We can also test whether the other demographic factors contained in the dataset might be related to the second PC. I might next map the sex variable to colour to see whether male and female penguins differ on either PC axis.

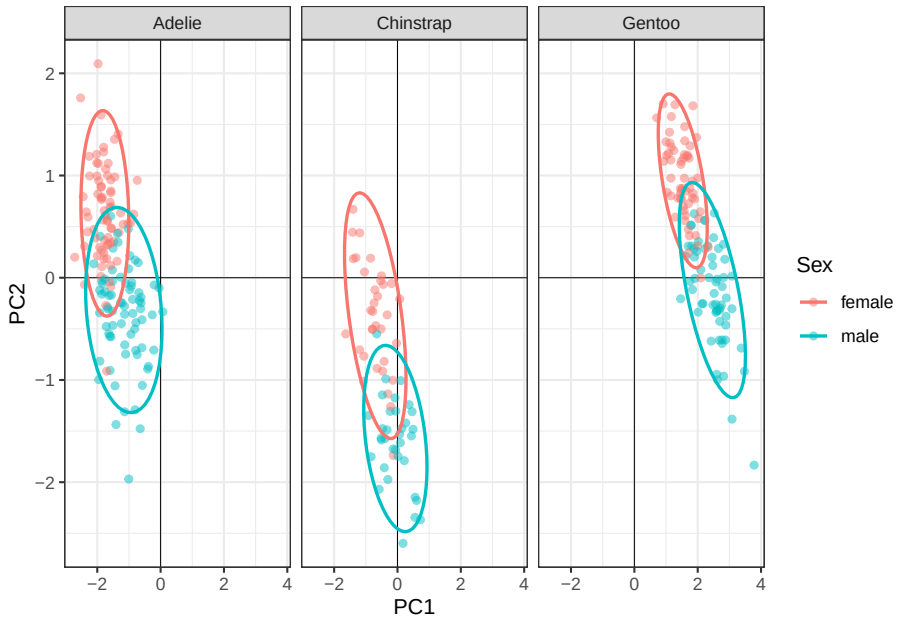
```
scores$Sex <- b$sex
ggplot(scores, aes(PC1, PC2, colour = Sex)) +
  geom_hline(yintercept = 0, linewidth = 0.2) +
  geom_vline(xintercept = 0, linewidth = 0.2) +
  geom_point(size = 2) +
  stat_ellipse(level = 0.95, linewidth = 0.8) +
  theme_bw()
```



- This looks interesting! It seems that our second PC helps to separate penguin sex in the chart.
- It could be that PC2 is capturing sex-specific patterns of bill shape in our penguin population
- This is a point in the analysis pipeline where domain expertise is very valuable!
- A quick google search indicates that male penguins tend to have larger bills than females.

I could also show the plot subdivided by species to visualize how sex relates to the second PC for each penguin species:

```
ggplot(scores, aes(PC1, PC2, colour = Sex)) +
  geom_hline(yintercept = 0, linewidth = 0.2) +
  geom_vline(xintercept = 0, linewidth = 0.2) +
  geom_point(size = 1.5, alpha = 0.5) +
  stat_ellipse(level = 0.95, linewidth = 0.8) +
  theme_bw() +
  facet_wrap(~Species)
```



- This plot nicely shows that within each species, PC2 seems to be separating male and female penguins based on sex differences in bill characteristics.

Summary of Analyses

The PCA uncovered latent structure in the data: Gentoo penguins tend to be larger than the other two species, and this variance was captured by PC1. For all species of penguins, there are sex-specific bill characteristics, which were captured by PC2.

Note: I know that many of you are interested in putting together personal portfolios to support your resumes. This practice is common in data science, and it is a good idea to build up / maintain a personal portfolio that showcases your analysis skills. Something like what I've written here would constitute a perfectly respectable portfolio project. I note this to highlight that especially when you're developing your portfolio from scratch, a project like this one (which could be put together in about 2 hours total) is great. You could also apply this same process to any other dataset that interests you as a portfolio project! :)

Week 9: November 6 2025

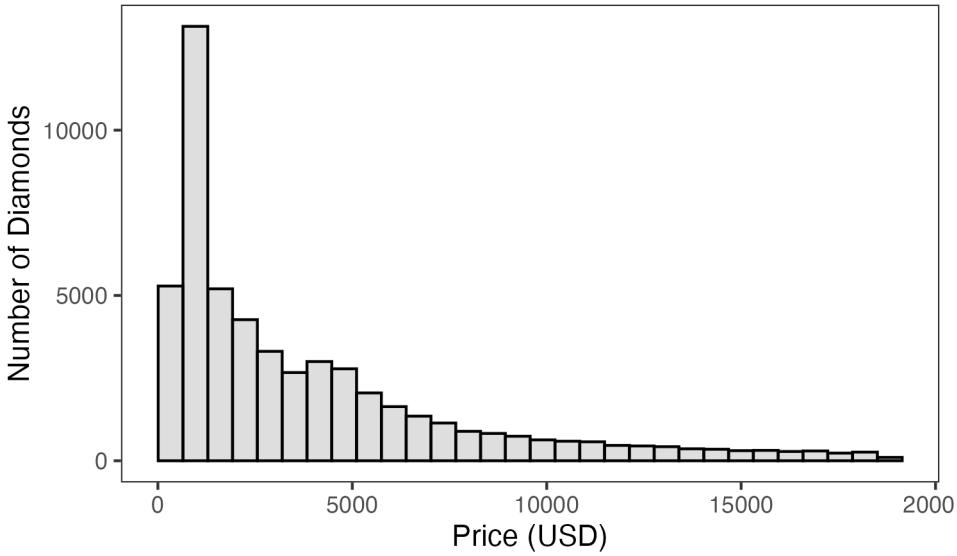
Exploratory Data Analysis

The `diamonds` dataset comes with ggplot 2. it contains measurements of ~54000 diamonds.

```
library(tidyverse)
library(reshape2)
data <- diamonds
```

We have a column for **price**. This will be our key dependent variable that we want to understand.

```
a <- data %>%
  ggplot(aes(x = price)) +
  geom_histogram(alpha = 0.2, colour = "black") +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  labs(
    x = "Price (USD)",
    y = "Number of Diamonds"
  )

ggsave("Figs/Price_hist.png", height = 3, width = 5, dpi = 300)
knitr::include_graphics("Figs/Price_hist.png")
```

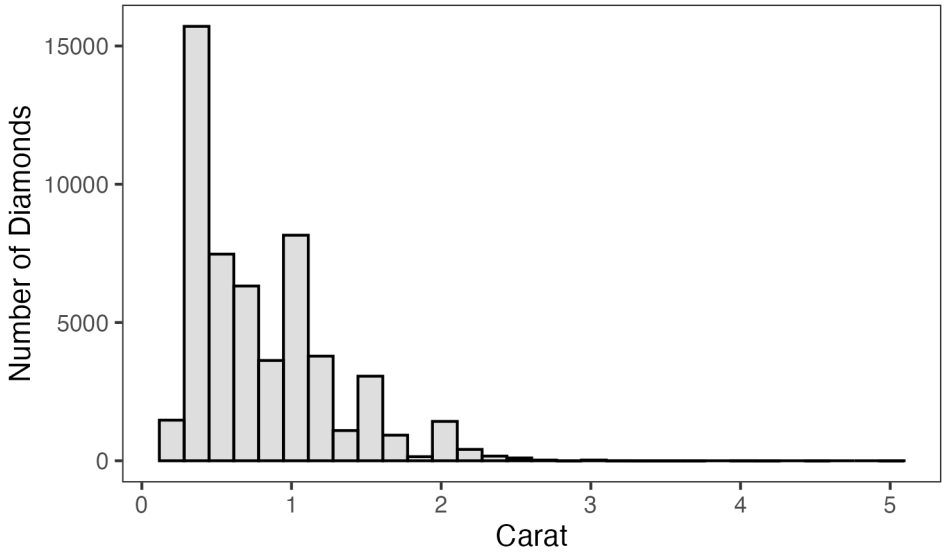


- Right-skewed distribution

- Most diamonds cost < 5000 USD, and some diamonds cost a lot more, up to about 20000 USD.

Explore Predictors

```
b <- diamonds %>%
  ggplot(aes(x = carat)) +
  geom_histogram(alpha = 0.2, colour = "black") +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  labs(
    x = "Carat",
    y = "Number of Diamonds"
  )

ggsave("Figs/Carat_hist.png", height = 3, width = 5, dpi = 300)
knitr::include_graphics("Figs/Carat_hist.png")
```

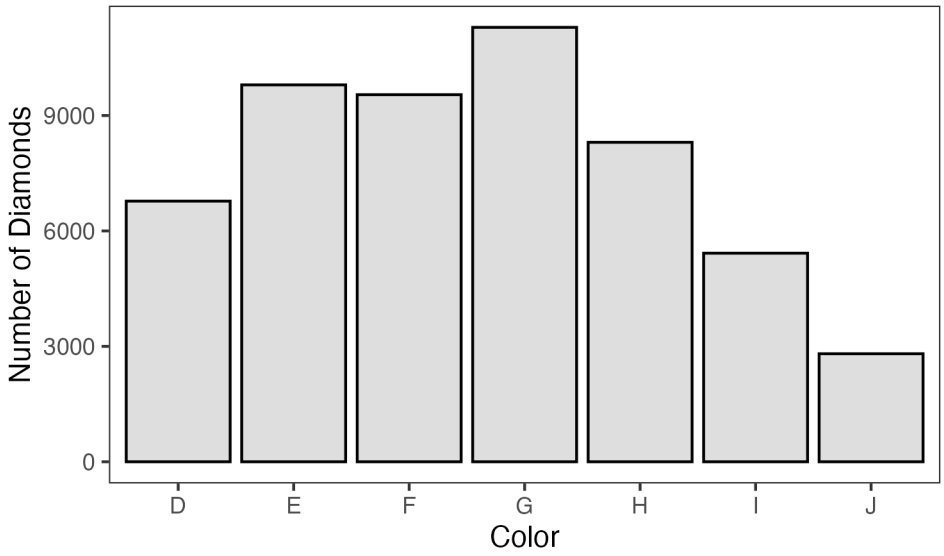



```
data %>%
  group_by(color) %>%
  summarise(
    n = n()
  )
```

```
## # A tibble: 7 x 2
##   color     n
##   <ord> <int>
## 1 D      6775
## 2 E      9797
## 3 F      9542
## 4 G     11292
## 5 H      8304
## 6 I      5422
## 7 J      2808
```

```
c <- data %>%
  group_by(color) %>%
  summarise(
    n = n()
  ) %>%
  ggplot(aes(x = color, y = n)) +
  geom_bar(stat = "identity", alpha = 0.2, colour = "black") +
  theme_bw() +
  theme(panel.grid = element_blank()) +
```

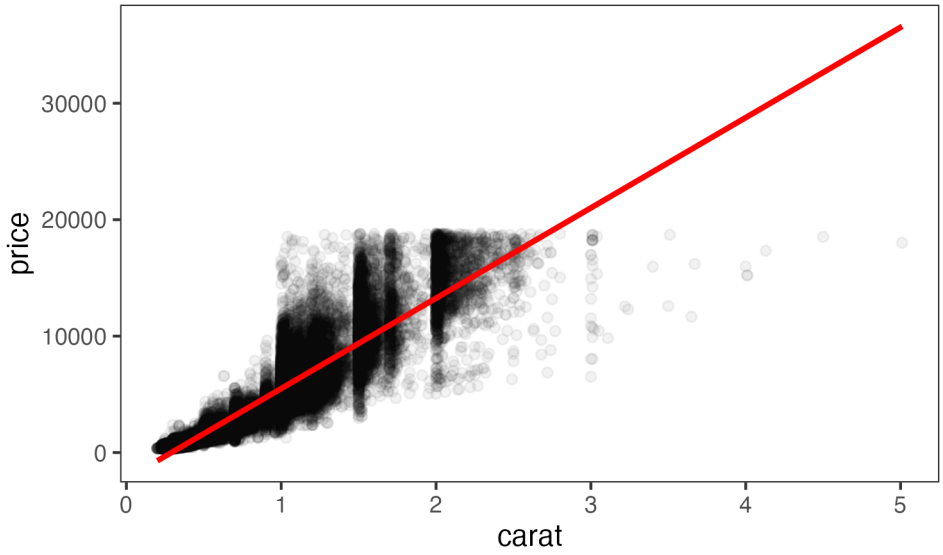
```
  labs(
    x = "Color",
    y = "Number of Diamonds"
  )

ggsave("Figs/Color_hist.png", height = 3, width = 5, dpi = 300)
knitr::include_graphics("Figs/Color_hist.png")
```



Simple Scatterplot

I'm going to choose `carat` as my predictor to start with, because I suspect that this will be closely related to the price.

```
d <- data %>%
  ggplot(aes(x = carat, y = price)) +
  geom_point(alpha = 0.05) +
  geom_smooth(method = "lm", se = F, colour = "red") +
  theme_bw() +
  theme(panel.grid = element_blank())

ggsave("Figs/carat_price.png", height = 3, width = 5, dpi = 300)
knitr::include_graphics("Figs/carat_price.png")
```



Compute Simple Regression

There is a clear relationship between carat and price: larger diamonds are more expensive. Let's start by making a simple regression modeling the relationship between size and price.

```
options(scipen = 999)
res <- lm(price ~ carat, data = data)
summary(res)
```

```
##
## Call:
## lm(formula = price ~ carat, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18585.3   -804.8    -18.9    537.4  12731.7
##
## Coefficients:
##             Estimate Std. Error t value            Pr(>|t|)
## (Intercept) -2256.36      13.06  -172.8 <0.0000000000000002 ***
## carat        7756.43      14.07   551.4 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 1549 on 53938 degrees of freedom
## Multiple R-squared:  0.8493, Adjusted R-squared:  0.8493
## F-statistic: 3.041e+05 on 1 and 53938 DF,  p-value: < 0.00000000000000022
```

- Carat accounts for 85% of variability in price ($R^2 = 0.85$).
- A 1-carat increase in diamond weight is associated with a $7756-dollar increase in price.

Multiple Regression

Try adding additional predictors in with `carat` to investigate whether they provide additional explantory power.

```
head(data)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

```
res_2 <- lm(price ~ carat + clarity + cut + color, data = data)
summary(res_2)
```

```
##
## Call:
## lm(formula = price ~ carat + clarity + cut + color, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16813.5   -680.4   -197.6    466.4  10394.9
##
## Coefficients:
##              Estimate Std. Error  t value             Pr(>|t|)
## (Intercept) -3710.603     13.980 -265.414 < 0.0000000000000002 ***
## carat        8886.129     12.034  738.437 < 0.0000000000000002 ***
## clarity.L    4217.535     30.831  136.794 < 0.0000000000000002 ***
```

```
## clarity.Q   -1832.406     28.827  -63.565 < 0.0000000000000002 ***
## clarity.C     923.273     24.679   37.411 < 0.0000000000000002 ***
## clarity^4    -361.995     19.739  -18.339 < 0.0000000000000002 ***
## clarity^5     216.616     16.109   13.447 < 0.0000000000000002 ***
## clarity^6       2.105     14.037    0.150                0.881
## clarity^7     110.340     12.383    8.910 < 0.0000000000000002 ***
## cut.L         698.907     20.335   34.369 < 0.0000000000000002 ***
## cut.Q        -327.686     17.911  -18.295 < 0.0000000000000002 ***
## cut.C         180.565     15.557   11.607 < 0.0000000000000002 ***
## cut^4          -1.207     12.458   -0.097                0.923
## color.L     -1910.288     17.712 -107.853 < 0.0000000000000002 ***
## color.Q      -627.954     16.121  -38.952 < 0.0000000000000002 ***
## color.C      -171.960     15.070  -11.410 < 0.0000000000000002 ***
## color^4        21.678     13.840    1.566                0.117
## color^5       -85.943     13.076   -6.572        0.00000000005 ***
## color^6       -49.986     11.889   -4.205        0.00002620629 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1157 on 53921 degrees of freedom
## Multiple R-squared:  0.9159, Adjusted R-squared:  0.9159
## F-statistic: 3.264e+04 on 18 and 53921 DF,  p-value: < 0.00000000000000022
```

- Adding `clarity`, `color`, and `cut` into the model increases the R^2 value to 0.91.
- This is 6% more variance accounted for than when we modeled only carat as the predictor.

  But is that 6% improvement a significant improvement???????

Let's formally test:

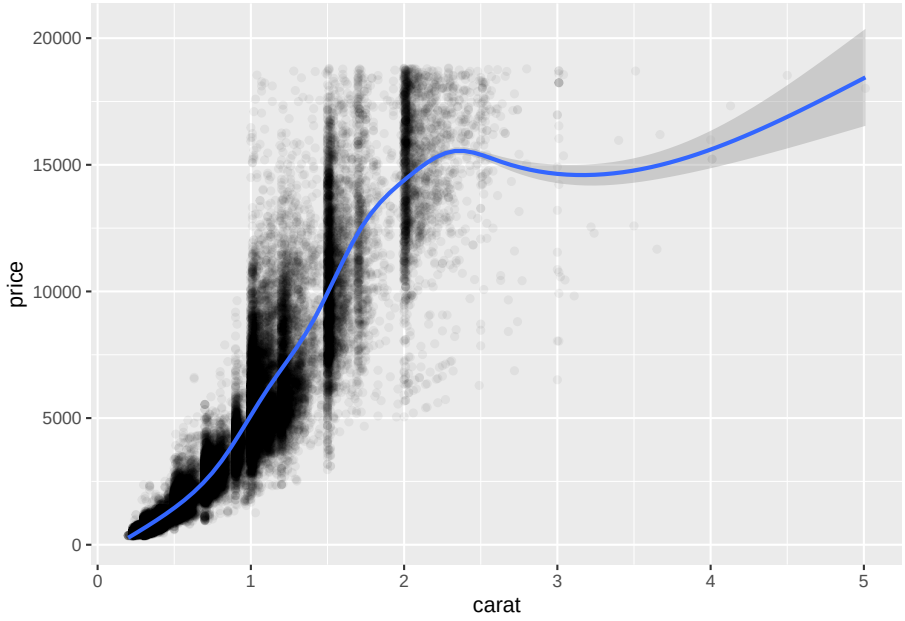
```
anova(res, res_2)
```

```
## Analysis of Variance Table
##
## Model 1: price ~ carat
## Model 2: price ~ carat + clarity + cut + color
##   Res.Df          RSS Df   Sum of Sq      F                Pr(>F)
## 1  53938 129345695398
## 2  53921  72162776162 17 57182919236 2513.4 < 0.00000000000000022 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- My second model is significantly better at capturing variance in `price` than the first model.

Compare Non-Linear Regression

```
data %>%
  ggplot(aes(x = carat, y = price)) +
  geom_point(alpha = 0.05) +
  geom_smooth(span = 0.5)
```



- Appears to be a non-linear relationship between carat and price.

Quadratic Model

```
res_3 <- lm(price ~ poly(carat, 2), data = data)
summary(res_3)
```

```
##
## Call:
## lm(formula = price ~ poly(carat, 2), data = data)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -26350.0   -724.2    -35.9    445.8  12881.1
##
## Coefficients:
##                   Estimate Std. Error t value            Pr(>|t|)
## (Intercept)       3932.800      6.631   593.1 <0.0000000000000002 ***
## poly(carat, 2)1 853889.595   1540.103   554.4 <0.0000000000000002 ***
## poly(carat, 2)2  37572.214   1540.103    24.4 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1540 on 53937 degrees of freedom
## Multiple R-squared:  0.851,  Adjusted R-squared:  0.851
## F-statistic: 1.54e+05 on 2 and 53937 DF,  p-value: < 0.00000000000000022
```

- Similar to the linear model above, the polynomial base model can account for 85% of variance in diamond price.

Let's compare the simple polynomial model to the simple regression model:

```
anova(res, res_3)
```

```
## Analysis of Variance Table
##
## Model 1: price ~ carat
## Model 2: price ~ poly(carat, 2)
##   Res.Df          RSS Df  Sum of Sq      F                Pr(>F)
## 1  53938 129345695398
## 2  53937 127934024108  1 1411671290 595.16 < 0.00000000000000022 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- The model that involved a polynomial was significantly better at capturing variance in the price of the diamonds.

Multiple Regression with a Polynomial

```
res_4 <- lm(price ~ poly(carat, 2) + cut + color + clarity, data = data)
summary(res_4)
```

```
##
```

```
## Call:
## lm(formula = price ~ poly(carat, 2) + cut + color + clarity,
##     data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -22942.8   -640.0   -187.3    416.4  10566.1
##
## Coefficients:
##                    Estimate Std. Error  t value             Pr(>|t|)
## (Intercept)        3362.045      9.488  354.345 < 0.0000000000000002 ***
## poly(carat, 2)1 977442.847   1317.069  742.135 < 0.0000000000000002 ***
## poly(carat, 2)2  30198.755   1171.539   25.777 < 0.0000000000000002 ***
## cut.L              680.759     20.223   33.662 < 0.0000000000000002 ***
## cut.Q             -329.741     17.802  -18.523 < 0.0000000000000002 ***
## cut.C              184.461     15.462   11.930 < 0.0000000000000002 ***
## cut^4               -1.062     12.382   -0.086               0.9317
## color.L          -1932.034     17.624 -109.624 < 0.0000000000000002 ***
## color.Q           -657.281     16.063  -40.919 < 0.0000000000000002 ***
## color.C           -172.049     14.979  -11.486 < 0.0000000000000002 ***
## color^4             31.405     13.761    2.282               0.0225 *
## color^5            -90.364     12.998   -6.952     0.00000000000364 ***
## color^6            -47.807     11.816   -4.046     0.00005222452368 ***
## clarity.L         4191.370     30.660  136.704 < 0.0000000000000002 ***
## clarity.Q        -1897.378     28.762  -65.968 < 0.0000000000000002 ***
## clarity.C          966.831     24.587   39.323 < 0.0000000000000002 ***
## clarity^4         -374.152     19.624  -19.066 < 0.0000000000000002 ***
## clarity^5          231.114     16.021   14.426 < 0.0000000000000002 ***
## clarity^6            2.080     13.951    0.149               0.8815
## clarity^7          103.122     12.311    8.376 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1150 on 53920 degrees of freedom
## Multiple R-squared:  0.917,  Adjusted R-squared:  0.9169
## F-statistic: 3.134e+04 on 19 and 53920 DF,  p-value: < 0.00000000000000022
```

```
anova(res_2, res_4)
```

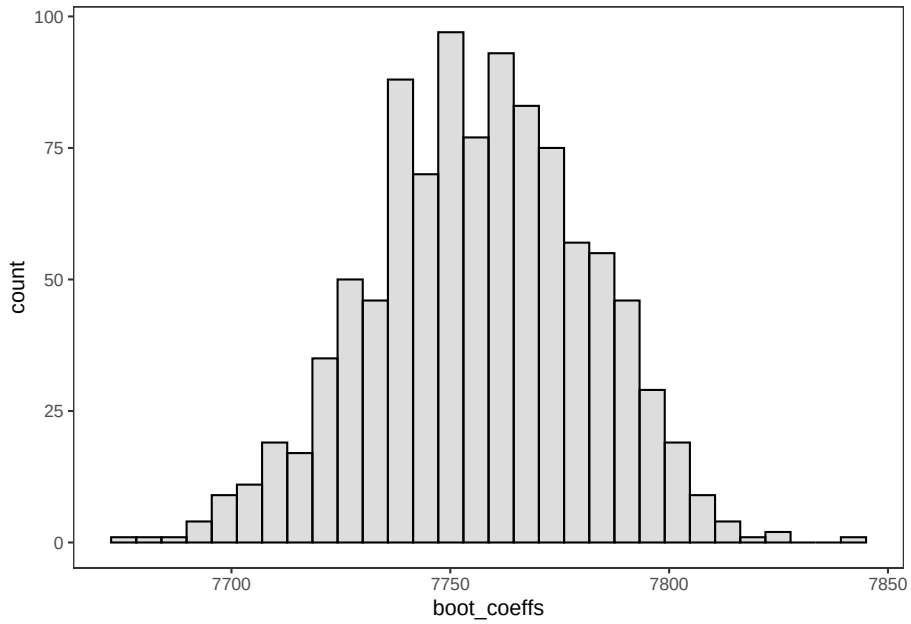
```
## Analysis of Variance Table
##
## Model 1: price ~ carat + clarity + cut + color
## Model 2: price ~ poly(carat, 2) + cut + color + clarity
##   Res.Df         RSS Df Sum of Sq      F               Pr(>F)
## 1  53921 72162776162
```

```
## 2  53920 71284343343  1 878432819 664.45 < 0.00000000000000022 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- The multiple regression including the polynomial term for `carat` is significantly better at capturing variance in price than the strictly linear multiple regression model from above.

Bootstrap the Estimates

```
set.seed(994)

B <- 1000
boot_coeffs <- replicate(B, {
  idx <- sample(nrow(data), replace = T)
  coef(lm(price ~ carat, data = data[idx, ]))[2]
})

ggplot(data.frame(boot_coeffs), aes(boot_coeffs)) +
  geom_histogram(alpha = 0.2, colour = "black") +
  theme_bw() +
  theme(panel.grid = element_blank())
```



- The distribution of slopes is tightly centered around 7750, which is close to the original slope estimate.
- There is not much range in the estimated slopes (min ~ 7500, max ~ 7800)
- This reinforces the conclusion from the simple regression: `carat` is a strong, robust predictor of `price`.

```
quantile(boot_coeffs, probs = c(0.025,0.975))
```

```
##     2.5%    97.5%
## 7705.957 7801.569
```

- 95% of the bootstrapped slopes are between 7705 and 7801.

Principal Component Analysis

I will follow the same steps that we used for the `penguins` dataset analysis on Oct 23rd.

Explore via Histograms

```
summary(data)
```

```
##      carat               cut        color        clarity          depth
##  Min.   :0.2000   Fair      : 1610   D: 6775   SI1    :13065   Min.   :43.00
##  1st Qu.:0.4000   Good      : 4906   E: 9797   VS2    :12258   1st Qu.:61.00
##  Median :0.7000   Very Good:12082   F: 9542   SI2    : 9194   Median :61.80
##  Mean   :0.7979   Premium   :13791   G:11292   VS1    : 8171   Mean   :61.75
##  3rd Qu.:1.0400   Ideal     :21551   H: 8304   VVS2   : 5066   3rd Qu.:62.50
##  Max.   :5.0100                     I: 5422   VVS1   : 3655   Max.   :79.00
##                                     J: 2808   (Other): 2531
##      table           price             x                y
##  Min.   :43.00   Min.   :  326   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720
##  Median :57.00   Median : 2401   Median : 5.700   Median : 5.710
##  Mean   :57.46   Mean   : 3933   Mean   : 5.731   Mean   : 5.735
##  3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540
##  Max.   :95.00   Max.   :18823   Max.   :10.740   Max.   :58.900
##
```

```
##        z
##  Min.   : 0.000
##  1st Qu.: 2.910
##  Median : 3.530
##  Mean   : 3.539
##  3rd Qu.: 4.040
##  Max.   :31.800
##
```

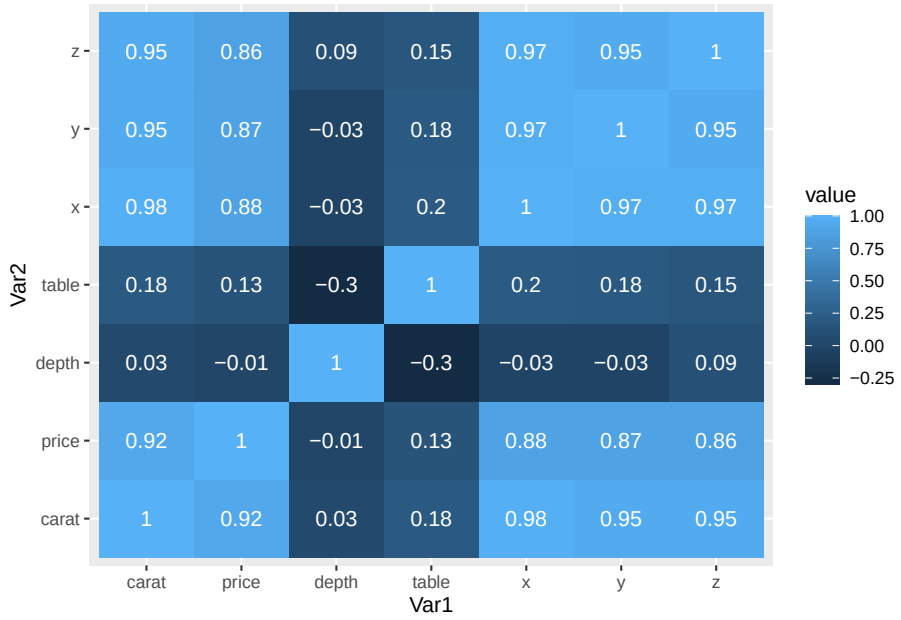
- The varaibles are measured on different scales, we need to match them in order to run PCA.

```
# Select only the numberic columns
a <- data %>%
  select(c(carat, price, depth, table, x, y, z))
```

```
a <- scale(a)
```

Correlation Matrix

```
cor(a) %>%
  melt() %>%
  mutate(value = round(value,2)) %>%
  ggplot(aes(x = Var1, y = Var2, fill = value, label = value)) +
  geom_tile() +
  geom_text(colour = "white")
```

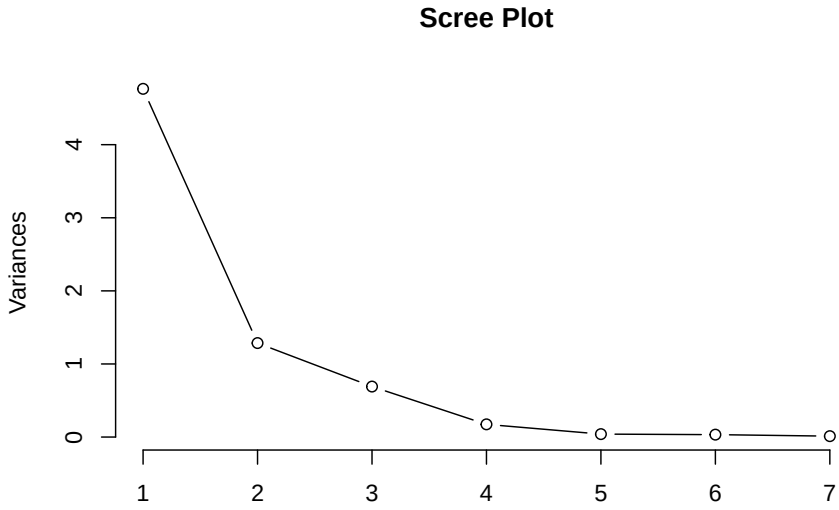


Compute PCA

```
pca_res <- prcomp(a)
summary(pca_res)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.1826 1.1340 0.83115 0.41684 0.20077 0.18151 0.11135
## Proportion of Variance 0.6806 0.1837 0.09869 0.02482 0.00576 0.00471 0.00177
## Cumulative Proportion  0.6806 0.8642 0.96294 0.98776 0.99352 0.99823 1.00000
```

```
plot(pca_res, type = "l", main = "Scree Plot")
```




```
loadings <- as_tibble(pca_res$rotation, rownames = "variable")
```

```
loadings %>%
  select(variable, PC1) %>%
  arrange(desc(abs(PC1))) %>%
  slice_head(n = 8)
```

```
## # A tibble: 7 x 2
##   variable       PC1
##   <chr>        <dbl>
## 1 x          0.453
## 2 carat      0.452
## 3 y          0.447
## 4 z          0.446
## 5 price      0.426
## 6 table      0.0995
## 7 depth     -0.000916
```

```
loadings %>%
  select(variable, PC2) %>%
  arrange(desc(abs(PC2))) %>%
  slice_head(n = 8)
```

```
## # A tibble: 7 x 2
```

```
##   variable      PC2
##   <chr>       <dbl>
## 1 depth    -0.731
## 2 table     0.675
## 3 z        -0.0890
## 4 price    -0.0353
## 5 carat    -0.0347
## 6 x         0.00351
## 7 y         0.00216
```

```
scores <- as_tibble(pca_res$x)

a <- ggplot(scores, aes(PC1, PC2)) +
geom_hline(yintercept = 0, linewidth = 0.2) + geom_vline(xintercept = 0, linewidth = 0

ggsave("Figs/large_ggplot.png", height = 9, width = 9, dpi = 300)
```

Week 10: November 13 2025

Overview

The purpose of this demonstration is to show some of my favorite dplyr / tidyverse functions. I have chosen some of the functions that I use regularly during data preparation in my own work.

Load Data

I'll demonstrate the functions using the `palmerpenguins` dataset. The penguins dataset is good to practice data wrangling because it contains some `NA` values.

```
library(palmerpenguins)
library(tidyverse)
data <- penguins
```

Base R vs. Tidyverse

In `tidyverse` syntax, multiple commands are strung together using "pipes": `%>%`. The pipe takes the result of what was generated before it, and passes that the the next function. Most things that I am showing you using `tidyverse` syntax could also be accomplished through `base R`, but that approach would involve assigning off intermediary objects. `tidyverse` is elegant in the way that it allows us to compute complex multi-step computational processes in a single step.

Favorite Tidyverse Functions

Select Some Rows

You may want to choose only some rows of a given dataset to work with.

Select rows that are Gentoo penguins who have a bill length over 40mm:

```
data %>%
  filter(species == "Gentoo", bill_length_mm > 40)
```

```
## # A tibble: 123 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>             <int>       <int>
##  1 Gentoo  Biscoe           46.1          13.2               211        4500
##  2 Gentoo  Biscoe           50            16.3               230        5700
##  3 Gentoo  Biscoe           48.7          14.1               210        4450
##  4 Gentoo  Biscoe           50            15.2               218        5700
##  5 Gentoo  Biscoe           47.6          14.5               215        5400
##  6 Gentoo  Biscoe           46.5          13.5               210        4550
##  7 Gentoo  Biscoe           45.4          14.6               211        4800
##  8 Gentoo  Biscoe           46.7          15.3               219        5200
##  9 Gentoo  Biscoe           43.3          13.4               209        4400
## 10 Gentoo  Biscoe           46.8          15.4               215        5150
## # i 113 more rows
## # i 2 more variables: sex <fct>, year <int>
```

- We see a preview of the first few rows of the data
- There are 113 additional rows and 2 additional columns that are not shown.

Group the Data

Count the number of penguins from each species:

```
data %>%
  group_by(species) %>%
  summarise(
    n = n()
  )
```

```
## # A tibble: 3 x 2
##   species       n
##   <fct>     <int>
## 1 Adelie      152
## 2 Chinstrap    68
## 3 Gentoo      124
```

- There are 152 Adelie penguins, 68 Chinstrap penguins and 124 Gentoo penguins in the dataset.

Choose Rows Based on a Factor

Select the Gentoo and Chinstrap penguins, group the data by species, and count the number of penguins from each species:

```
data %>%
  filter(species %in% c("Gentoo","Chinstrap")) %>%
  group_by(species) %>%
  summarise(
    n = n()
  )
```

```
## # A tibble: 2 x 2
##   species       n
##   <fct>     <int>
## 1 Chinstrap    68
## 2 Gentoo      124
```

- Now we only see two rows, which correspond to the two penguin species that we asked for.
- Can be useful if you have many levels of a variable, and you are interested in seeing some specific categories.

Choose the rows that do not have "Adelie" in the species column, then group the data by species, then summarise the number of penguins from each species:

```
data %>%
  filter(species != "Adelie") %>%
  group_by(species) %>%
  summarise(
    n = n()
  )
```

```
## # A tibble: 2 x 2
##   species       n
##   <fct>     <int>
## 1 Chinstrap    68
## 2 Gentoo      124
```

- Same values as the ouptut above
- Selecting all the Gentoo and Chinstrap rows is the same as selecting all the rows that are not Adelie penguins!

Arrange the Data

Order the rows based on the values in a column:

```
data %>%
  arrange(body_mass_g)
```

```
## # A tibble: 344 x 8
##    species   island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>     <fct>             <dbl>         <dbl>             <int>       <int>
##  1 Chinstrap Dream              46.9          16.6               192        2700
##  2 Adelie    Biscoe             36.5          16.6               181        2850
##  3 Adelie    Biscoe             36.4          17.1               184        2850
##  4 Adelie    Biscoe             34.5          18.1               187        2900
##  5 Adelie    Dream              33.1          16.1               178        2900
##  6 Adelie    Torgers~           38.6          17                 188        2900
##  7 Chinstrap Dream              43.2          16.6               187        2900
##  8 Adelie    Biscoe             37.9          18.6               193        2925
##  9 Adelie    Dream              37.5          18.9               179        2975
## 10 Adelie    Dream              37            16.9               185        3000
## # i 334 more rows
## # i 2 more variables: sex <fct>, year <int>
```

- Rows arranged by penguin weights: The lightest penguin's row is shown first.

```
data %>%
  arrange(desc(body_mass_g))
```

```
## # A tibble: 344 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>             <int>       <int>
```

```
##  1 Gentoo  Biscoe           49.2          15.2               221        6300
##  2 Gentoo  Biscoe           59.6          17                 230        6050
##  3 Gentoo  Biscoe           51.1          16.3               220        6000
##  4 Gentoo  Biscoe           48.8          16.2               222        6000
##  5 Gentoo  Biscoe           45.2          16.4               223        5950
##  6 Gentoo  Biscoe           49.8          15.9               229        5950
##  7 Gentoo  Biscoe           48.4          14.6               213        5850
##  8 Gentoo  Biscoe           49.3          15.7               217        5850
##  9 Gentoo  Biscoe           55.1          16                 230        5850
## 10 Gentoo  Biscoe           49.5          16.2               229        5800
## # i 334 more rows
## # i 2 more variables: sex <fct>, year <int>
```

- Rows arranged by body mass
- Heaviest penguin is first down to the lightest penguin.

Selecting Some Columns

Sometimes datasets have many columns that are irrelevant to a given analysis. Sometimes it works best to select the columns that will be involved in making each analysis or chart to ensure that extraneous problems don't create issues.

```
data %>%
  select(c("body_mass_g", "species", "sex"))
```

```
## # A tibble: 344 x 3
##    body_mass_g species sex
##          <int> <fct>   <fct>
##  1        3750 Adelie  male
##  2        3800 Adelie  female
##  3        3250 Adelie  female
##  4          NA Adelie  <NA>
##  5        3450 Adelie  female
##  6        3650 Adelie  male
##  7        3625 Adelie  female
##  8        4675 Adelie  male
##  9        3475 Adelie  <NA>
## 10        4250 Adelie  <NA>
## # i 334 more rows
```

- Only the columns that we asked for are the in the preview

Generating New columns

The block of code below performs the following functions:

- Calculate a new column called `body_mass_kg` that is the value in the `body_mass_g` column * 1000.
- Select the columns `species` and `body_mass_kg`
- Group the Data by the `species column`
- Remove rows that contain an `NA` value
- Calculate the number of penguins of each species and the mean and standard deviation of the body_mass_kg column for each penguin species
- Print the results out in a professional looking table

```
data %>%
  mutate(body_mass_kg = body_mass_g / 1000) %>%
  select(c(species, body_mass_kg)) %>%
  group_by(species) %>%
  na.omit() %>%
  summarise(
    count = n(),
    mean = mean(body_mass_kg),
    sd = sd(body_mass_kg)
  ) %>%
  mutate(
    mean = round(mean, digits = 2),
    sd = round(sd, digits = 2)
  ) %>%
  knitr::kable()
```

species	count	mean	sd
Adelie	151	3.70	0.46
Chinstrap	68	3.73	0.38
Gentoo	123	5.08	0.50

- The mean body mass for Gentoo penguins is higher than the mean body weights for the other two species.

Select Distinct instances

Selecting distinct instances is useful when exploring a dataset to get familiar with the levels of the variables (i.e., the names of the categories).

```
data %>%
  distinct(island)
```

```
## # A tibble: 3 x 1
##   island
##   <fct>
## 1 Torgersen
## 2 Biscoe
## 3 Dream
```

- There are three islands in the dataset: Torgersen, Biscoe, and Dream.

Count the number of pengins from each species measured on each island:

```
data %>%
  group_by(island, species) %>%
  summarise(
    n = n()
  ) %>%
  knitr::kable()
```

island	species	n
Biscoe	Adelie	44
Biscoe	Gentoo	124
Dream	Adelie	56
Dream	Chinstrap	68
Torgersen	Adelie	52

- Adelie penguins were measured on all three islands
- Gentoo penguins were only measured on Biscoe island
- Chinstrap penguins were only measured on Dream island

Rename Columns

Sometimes datasets come with long, similar, or otherwise annoying column headers. It is a good idea for the column names to be systematic, distinct, and as short as possible.

```
data <- data %>%
  rename(
    b_length = bill_length_mm,
    b_depth = bill_depth_mm,
    flip_length = flipper_length_mm,
    body_mass = body_mass_g
  )
```

- In this case we are overwriting the data object `data`.
- We are overwriting `data` because we would want to save the shortened column header names for subsequent analyses.

Rearranging columns

It is my preference to see identifying information first in a dataset followed by continuous measurements. The code below moves the two columns of identifying information that by default appear at the far right before the numeric variables.

```
data <- data %>%
  relocate(sex, year, .before = b_length)
```

- Just like above, here, I am overwriting the data object so that my preferred organization is saved in the working environment.

Write Off a .csv File

Sometimes it is useful to save off a hard copy of a dataset or a summary that you've created in R. The cow below generates summary statistics split by all the demographic variabless, then saves off a .csv file.

```
data %>%
  group_by(year, island, species, sex) %>%
  summarise(
    n = n()
  ) %>%
  write_csv("summary.csv")
```

See More of Less of the Data Preview

Much of the time that you work with tidyverse codeblocks, you may want to see more or less of the output in the data preview. The code below alters the length of the data preview.

Show me the first 15 rows:

```
data %>%
  slice_head(n = 15)
```

```
## # A tibble: 15 x 8
##    species island    sex     year b_length b_depth flip_length body_mass
##    <fct>   <fct>     <fct>  <int>    <dbl>   <dbl>       <int>     <int>
##  1 Adelie  Torgersen male    2007     39.1    18.7         181      3750
##  2 Adelie  Torgersen female  2007     39.5    17.4         186      3800
##  3 Adelie  Torgersen female  2007     40.3    18           195      3250
##  4 Adelie  Torgersen <NA>    2007     NA      NA            NA        NA
##  5 Adelie  Torgersen female  2007     36.7    19.3         193      3450
##  6 Adelie  Torgersen male    2007     39.3    20.6         190      3650
##  7 Adelie  Torgersen female  2007     38.9    17.8         181      3625
##  8 Adelie  Torgersen male    2007     39.2    19.6         195      4675
##  9 Adelie  Torgersen <NA>    2007     34.1    18.1         193      3475
## 10 Adelie  Torgersen <NA>    2007     42      20.2         190      4250
## 11 Adelie  Torgersen <NA>    2007     37.8    17.1         186      3300
## 12 Adelie  Torgersen <NA>    2007     37.8    17.3         180      3700
## 13 Adelie  Torgersen female  2007     41.1    17.6         182      3200
## 14 Adelie  Torgersen male    2007     38.6    21.2         191      3800
## 15 Adelie  Torgersen male    2007     34.6    21.1         198      4400
```

Show me the lightest penguin in the dataset:

```
data %>%
  na.omit() %>%
  arrange(body_mass) %>%
  slice_tail(n = 1)
```

```
## # A tibble: 1 x 8
##   species island sex    year b_length b_depth flip_length body_mass
##   <fct>   <fct>  <fct> <int>    <dbl>   <dbl>       <int>     <int>
## 1 Gentoo  Biscoe male   2007     49.2    15.2         221      6300
```

Show me the 10 lightest penguins in the dataset:

```
data %>%
  slice_min(body_mass, n = 10)
```

```
## # A tibble: 11 x 8
##    species   island    sex     year b_length b_depth flip_length body_mass
```

```
##    <fct>     <fct>     <fct>  <int>    <dbl>   <dbl>       <int>     <int>
##  1 Chinstrap Dream     female  2008     46.9    16.6         192      2700
##  2 Adelie    Biscoe    female  2008     36.5    16.6         181      2850
##  3 Adelie    Biscoe    female  2008     36.4    17.1         184      2850
##  4 Adelie    Biscoe    female  2008     34.5    18.1         187      2900
##  5 Adelie    Dream     female  2008     33.1    16.1         178      2900
##  6 Adelie    Torgersen female  2009     38.6    17           188      2900
##  7 Chinstrap Dream     female  2007     43.2    16.6         187      2900
##  8 Adelie    Biscoe    female  2009     37.9    18.6         193      2925
##  9 Adelie    Dream     <NA>    2007     37.5    18.9         179      2975
## 10 Adelie    Dream     female  2007     37      16.9         185      3000
## 11 Adelie    Dream     female  2009     37.3    16.8         192      3000
```

Show me the 10 heaviest penguins:

```
data %>%
  slice_max(body_mass, n = 10)
```

```
## # A tibble: 11 x 8
##    species island sex    year b_length b_depth flip_length body_mass
##    <fct>   <fct>  <fct> <int>    <dbl>   <dbl>       <int>     <int>
##  1 Gentoo  Biscoe male   2007     49.2    15.2         221      6300
##  2 Gentoo  Biscoe male   2007     59.6    17           230      6050
##  3 Gentoo  Biscoe male   2008     51.1    16.3         220      6000
##  4 Gentoo  Biscoe male   2009     48.8    16.2         222      6000
##  5 Gentoo  Biscoe male   2008     45.2    16.4         223      5950
##  6 Gentoo  Biscoe male   2009     49.8    15.9         229      5950
##  7 Gentoo  Biscoe male   2007     48.4    14.6         213      5850
##  8 Gentoo  Biscoe male   2007     49.3    15.7         217      5850
##  9 Gentoo  Biscoe male   2009     55.1    16           230      5850
## 10 Gentoo  Biscoe male   2008     49.5    16.2         229      5800
## 11 Gentoo  Biscoe male   2008     48.6    16           230      5800
```

Show me the 5 heaviest penguins from each species:

```
data %>%
  group_by(species) %>%
  slice_max(body_mass, n = 5)
```

```
## # A tibble: 16 x 8
## # Groups:   species [3]
##    species   island    sex    year b_length b_depth flip_length body_mass
##    <fct>     <fct>     <fct> <int>    <dbl>   <dbl>       <int>     <int>
##  1 Adelie    Biscoe    male   2009     43.2    19           197      4775
```

```
##  2 Adelie    Biscoe    male   2009     41      20           203      4725
##  3 Adelie    Torgersen male   2008     42.9    17.6         196      4700
##  4 Adelie    Torgersen male   2007     39.2    19.6         195      4675
##  5 Adelie    Dream     male   2007     39.8    19.1         184      4650
##  6 Chinstrap Dream     male   2008     52      20.7         210      4800
##  7 Chinstrap Dream     male   2008     52.8    20           205      4550
##  8 Chinstrap Dream     male   2008     53.5    19.9         205      4500
##  9 Chinstrap Dream     male   2009     50.8    18.5         201      4450
## 10 Chinstrap Dream     male   2007     49.2    18.2         195      4400
## 11 Gentoo    Biscoe    male   2007     49.2    15.2         221      6300
## 12 Gentoo    Biscoe    male   2007     59.6    17           230      6050
## 13 Gentoo    Biscoe    male   2008     51.1    16.3         220      6000
## 14 Gentoo    Biscoe    male   2009     48.8    16.2         222      6000
## 15 Gentoo    Biscoe    male   2008     45.2    16.4         223      5950
## 16 Gentoo    Biscoe    male   2009     49.8    15.9         229      5950
```

Randomly Sample From The Data

There are instances where it is helpful to select a random sample of rows. This is especially useful to check other computational processes.

Sample 20 random rows from the data:

```
data %>%
  sample_n(20)
```

```
## # A tibble: 20 x 8
##    species   island    sex     year b_length b_depth flip_length body_mass
##    <fct>     <fct>     <fct>  <int>    <dbl>   <dbl>       <int>     <int>
##  1 Adelie    Dream     male    2009     40.7    17           190      3725
##  2 Chinstrap Dream     male    2007     50.5    19.6         201      4050
##  3 Gentoo    Biscoe    male    2009     48.8    16.2         222      6000
##  4 Adelie    Dream     female  2009     37      16.5         185      3400
##  5 Gentoo    Biscoe    female  2008     45.7    13.9         214      4400
##  6 Gentoo    Biscoe    male    2008     45      15.4         220      5050
##  7 Adelie    Biscoe    female  2008     39      17.5         186      3550
##  8 Gentoo    Biscoe    male    2009     50.4    15.7         222      5750
##  9 Adelie    Torgersen male    2007     39.3    20.6         190      3650
## 10 Chinstrap Dream     female  2009     50.9    17.9         196      3675
## 11 Gentoo    Biscoe    female  2008     45.5    13.9         210      4200
## 12 Adelie    Dream     female  2007     39.5    16.7         178      3250
## 13 Adelie    Dream     female  2008     36.9    18.6         189      3500
## 14 Adelie    Biscoe    male    2008     41.6    18           192      3950
## 15 Chinstrap Dream     female  2007     45.9    17.1         190      3575
## 16 Adelie    Torgersen male    2008     42.1    19.1         195      4000
```

```
## 17 Chinstrap Dream     male    2009     50.8    18.5         201      4450
## 18 Adelie    Biscoe    female  2009     38.1    17           181      3175
## 19 Gentoo    Biscoe    male    2008     46.4    15.6         221      5000
## 20 Gentoo    Biscoe    female  2008     45.3    13.8         208      4200
```

Samples a random 10% of the rows:

```
data %>%
  sample_frac(0.01)
```

```
## # A tibble: 3 x 8
##   species   island sex    year b_length b_depth flip_length body_mass
##   <fct>     <fct>  <fct> <int>    <dbl>   <dbl>       <int>     <int>
## 1 Adelie    Dream  male   2009     39.2    18.6         190      4250
## 2 Chinstrap Dream  male   2009     49.3    19.9         203      4050
## 3 Adelie    Dream  male   2009     40.6    17.2         187      3475
```

- There are 3 rows of data in the preview because 1% of 334 is ~3.

Compute Quick Descriptive Stats

If there are `NA` values in quantitative columns that you want to summarize, you must first remove he NA values.

```
data %>%
  na.omit() %>%
  summarise(median = median(body_mass))
```

```
## # A tibble: 1 x 1
##   median
##    <int>
## 1   4050
```

- The median body mass in the whole dataset is 4050g.

Generate New Categorical Columns

It is sometimes useful to calculate new variables that combine multiple levels from another variable. The code below makes a new column named "heavy". Penguins that are under 4050g will be assigned the label "light" and penguins that are over the median weight of 4050g will be labeled as "heavy". Any rows that have missing data in the `body_mass` column will be labeled as "unknown".

```
data %>%
  mutate(heavy = if_else(body_mass > 4050, "heavy", "light", missing = "unknown")) %>%
  group_by(species, heavy) %>%
  summarise(n = n()) %>%
  knitr::kable()
```

species	heavy	n
Adelie	heavy	33
Adelie	light	118
Adelie	unknown	1
Chinstrap	heavy	11
Chinstrap	light	57
Gentoo	heavy	122
Gentoo	light	1
Gentoo	unknown	1

Generate a new column called "heavy_3", with 3 levels based on body mass:

```
data %>%
  mutate(heavy_3 = case_when(
    body_mass < 3500 ~ "light",
    body_mass >= 3500 & body_mass < 4500 ~ "medium",
    body_mass > 4500 ~ "hefty"
  )) %>%
  group_by(species, heavy_3) %>%
  summarise(count = n()) %>%
  na.omit() %>%
  knitr::kable()
```

species	heavy_3	count
Adelie	hefty	7
Adelie	light	54
Adelie	medium	89
Chinstrap	hefty	2
Chinstrap	light	17
Chinstrap	medium	48
Gentoo	hefty	106
Gentoo	medium	16

Data viz

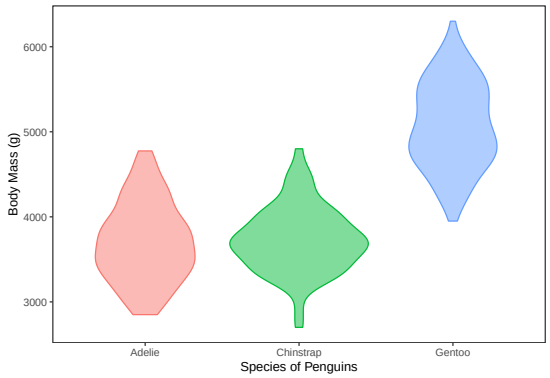
In this course, we've made lots of scatterplots and assessed lines of best fit that capture variance in continuous relationships. There is also value in grouping

the data and looking at descriptive statistics (e.g., Range, mean, spread of the scores) within each group. Below I will show three diverent versions of presenting group-based descriptive information using charts.

Violin Plot

The code below generates a violin plot showing the distributions of body mass for each species of penguin.

```
data %>%
  group_by(species) %>%
  ggplot(aes(x = species, y = body_mass, colour = species, fill = species)) +
  geom_violin(alpha = 0.5) +
  theme_bw() + theme(panel.grid = element_blank()) +
  theme(legend.position = "none") +
  labs(
    x = "Species of Penguins",
    y = "Body Mass (g)"
  )
```



- The violin plot provides qualitative information about the spread of scores in each group, which is communicated by the shape of the "violins".
- Each violin is the distribution of scores plotted vertically and mirrored.

Bar Chart

Some people are partial to seeing a bar chart depicting mean values plus or minus an index of spread of the scores. The computation that is used as the error bars is field-specfic: Some fields us standard deviation, others use standard error of the mean, and there are some that use the 95% confidence interval. Since

the error bars do not always represent the same thing, it is useful to provide information about what the error bars represent in the figure caption. Using the standard error of the mean is the norm in my field, so I will show it below.

- Rstudio does not have a built in formula to compute SEM, so I will compute it myself by entering the formula

$$se = \frac{sd}{\sqrt{n}}$$

- The standard error is equal to the standard deviation within a group divided by the square root of the number of scores in that group.

```
data %>%
  group_by(species) %>%
  na.omit() %>%
  summarise(
    n = n(),
    mean = mean(body_mass),
    sd = sd(body_mass),
    se = sd / sqrt(n)
  ) %>%
  ggplot(aes(x = species, y = mean, colour = species, fill = species)) +
  geom_bar(stat = "identity", alpha = 0.2) +
  geom_errorbar(aes(x = species, ymin = mean - se, ymax = mean + se), width = 0.5) +
  geom_jitter(data = data, aes(x = species, y = body_mass), width = 0.25, alpha = 0.3) +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  theme(legend.position = "none") +
  labs(
    x = "Species of Penguins",
    y = "Body Mass (g)"
  )
```

- In the code above, I first computed the mean and se for each group, then piped that information directly to `ggplot`
- I also added `geom_jitter()` which shows the individual datapoints overlain on top of the bars.

Boxplot

Some people like to use boxplots to showcase descriptive information about scores.

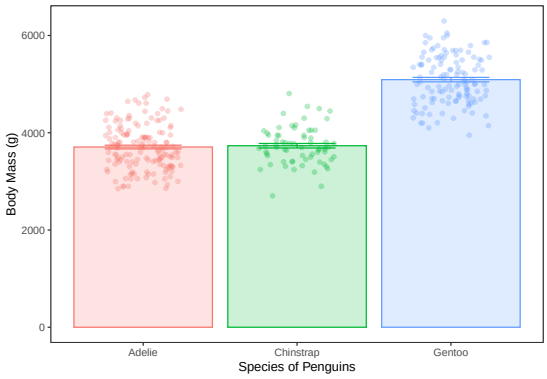
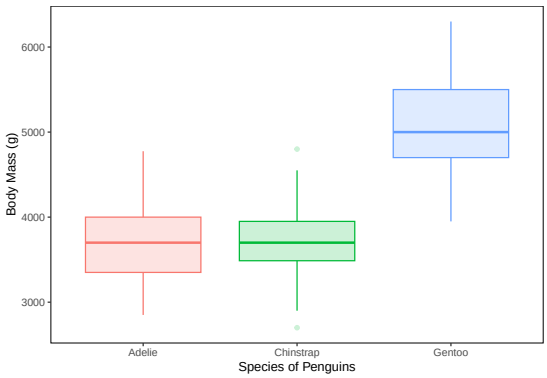


Figure 1: Figure caption: Average body weights for each penguin species. Data plotted as mean value +/- SEM.

```
data %>%
  ggplot(aes(x = species, y = body_mass, colour = species, fill = species)) +
  geom_boxplot(alpha = 0.2) +
  theme_bw() +
  theme(panel.grid = element_blank()) +
  theme(legend.position = "none") +
  labs(
    x = "Species of Penguins",
    y = "Body Mass (g)"
  )
```



- The horizonal centeral line on each bar represents the group's median score.
- the box represents the inter-quartile range. 50% of each group's scores fall in this range.

- The “wiskers” (error bars) represent the range

Week 11: November 20 2025