A Progressive Introduction to Linear Models

Joshua French

Contents

Preliminaries

I designed this book to progressively introduce you to the analysis of data using linear models. My goal is to provide you with the skills needed to perform a linear regression analysis sooner rather than later. Some material that could be covered together (for example, all the different types of statistical tests and confidence intervals) has been broken into two sections: an early one to give you foundational knowledge about the topic and then later material to advance your understanding. Most of the detailed derivations have been placed in **Going Deeper** sections or in their own chapter, which can be skipped over to more quickly progress through the material if you do not want to focus as much on theory.

Acknowledgments

The **bookdown** package (Xie 2022) was used to generate this book. The **kableExtra** package (Zhu 2021) was to format the tables. The writing of the book was partially supported by the Colorado Department of Higher Education as part of the proposal "OER for the Creation of Interactive Computational Notebooks and a Computational Pathway in Mathematics and Statistics".

Chapter 1

R Foundations

Meaningful data analysis requires the use of computer software.

R statistical software is one of the most popular tools for data analysis in academia, industry, and government. In what follows, I will attempt to lay a foundation of basic knowledge and skills with R that you will need for data analysis. I make no attempt to be exhaustive, and many other important aspects of using R (like plotting) will be discussed later, as needed.

1.1 Setting up R and RStudio Desktop

What is R?

R is a programming language and environment designed for statistical computing. It was introduced by Robert Gentleman and Robert Ihaka in 1993 as a free implementation of the *S* programming language developed at Bell Laboratories (https://www.r-project.org/about.html)

Some important facts about R are that:

- R is free, open source, and runs on many different types of computers (Windows, Mac, Linux, and others).
- R is an interactive programming language.
  - You type and run a command in the Console for immediate feedback, in contrast to a compiled programming language, which compiles a program that is then executed.
- R is highly extendable.
  - Many user-created packages are available to extend the functionality beyond what is installed by default.
  - Users can write their own functions and easily add software libraries to R.

Installing R

To install R on your personal computer, you will need to download an installer program from the R Project's website (https://www.r-project.org/). Links to download the installer program for your operating system *should* be found at https://cloud.r-project.org/. Click on the download link appropriate for your computer's operating system and install R on your computer. If you have a Windows computer, a stable link for the most current installer program is available at https://cloud.r-project.org/bin/windows/base/release.html. (Similar links are not currently available for Mac and Linux computers.)

Installing RStudio

RStudio Desktop is a free "front end" for R provided by RStudio (https://rstudio.com/). RStudio Desktop makes doing data analysis with R much easier by adding an Integrated Development Environment (IDE) and providing many other features. Currently, you may download RStudio at https://rstudio.com/products/rstudio/download/. You may need to navigate the RStudio website directly if this link no longer functions. Download the Free version of RStudio Desktop appropriate for your computer and install it.

Having installed both R and RStudio Desktop, you will want to open RStudio Desktop as you continue to learn about R.

RStudio Layout

RStudio Desktop has four panes:

1. Console: the pane where commands are run.
2. Source: the pane where you prepare commands to be run.
3. Environment/History: the pane where you can see all the objects in your workspace, your command history, and other information.
4. The Files/Plot/Packages/Help: the pane where you navigate between directories, where plots can be viewed, where you can see the packages available to be loaded, and where you can get help.

To see all RStudio panes, press the keys `Ctrl + Alt + Shift + 0` on a PC or `Cmd + Option + Shift + 0` on a Mac.

Figure 1.1 displays a labeled graphic of the panes. Your panes are likely in a different order than the graphic shown because I have customized my workspace for my own needs.

Customizing the RStudio workspace

At this point, I would highly encourage you to make one small workspace customization that will likely save you from experiencing future frustration. R provides a "feature" of that allows you to "save a workspace". This allows you to easily pick up where you left off your last analysis. The issue with this is that over time you accumulate a lot of environmental artifacts that can conflict with each other. This can lead to errors and incorrect results that you will need to

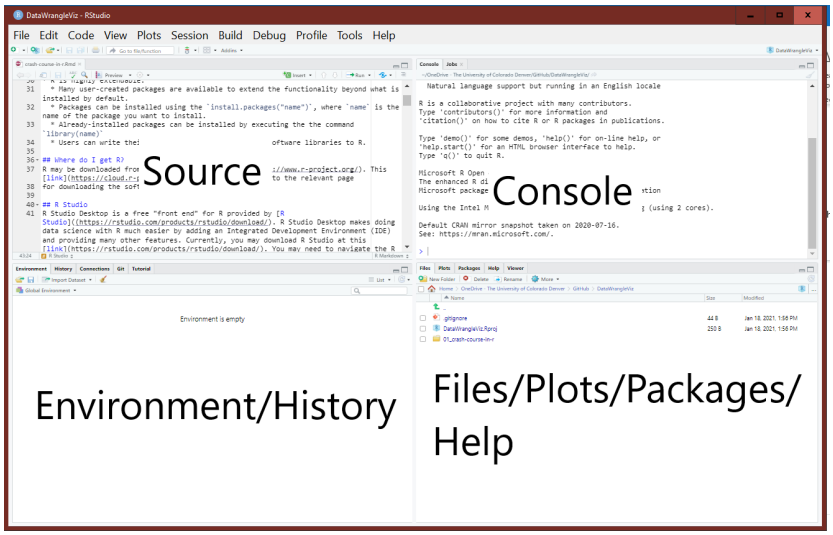


Figure 1.1: The RStudio panes labeled for convenience.

deal with. Additionally, this "feature" hinders the ability of others to reproduce your analysis because other users are unlikely to have the same workspace.

To turn off this feature, in the RStudio menu bar click Tools → Global Options and then make sure the "General" option is selected. Then make the following changes (if necessary):

1. Uncheck the box for "Restore .RData into workspace at startup".
2. Change the toggle box for "Save workspace to .RData on exit" to "Never".
3. Click Apply then OK to save the changes.

Figure 1.2 displays what these options should look like.

1.2 Running code, scripts, and comments

You can run code in R by typing it in the Console next to the `>` symbol and pressing the Enter key.

If you need to successively run multiple commands, it's better to write your commands in a "script" file and then save the file. The commands in a Script file are often generically referred to as "code".

Script files make it easy to:

- Reproduce your data analysis without retyping all your commands.
- Share your code with others.

A new Script file can be obtained by:

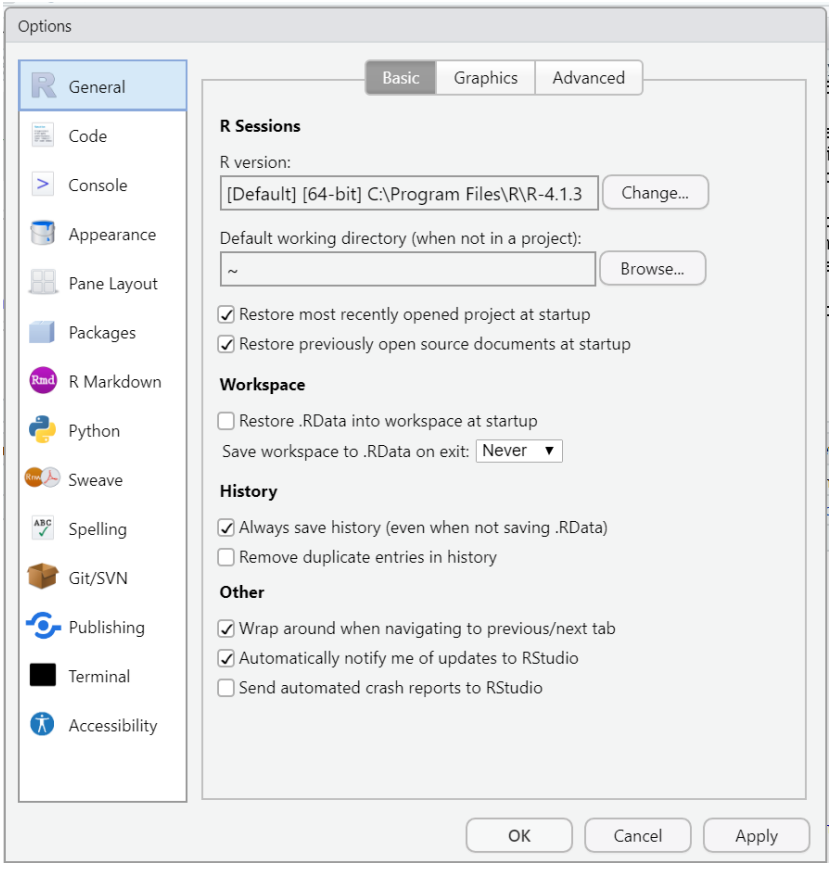
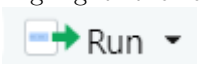


Figure 1.2: The General options window.

- Clicking File → New File → R Script in the RStudio menu bar.
- Pressing `Ctrl + Shift + n` on a PC or `Cmd + Shift + n` on a Mac.

There are various ways to run code from a Script file. The most common ones are:

1. Highlight the code you want to run and click the Run button 

at the top of the Script pane.
2. Highlight the code you want to run and press "Ctrl + Enter" on your keyboard. If you don't highlight anything, by default, RStudio runs the command the cursor currently lies on.

To save a Script file:

- Click File → Save in the RStudio menu bar.
- Press `Ctrl + s` on a PC or `Cmd + s` on a Mac.

A comment is a set of text ignored by R when submitted to the Console.

A comment is indicated by the `#` symbol. Nothing to the right of the `#` is executed by the Console.

To comment (or uncomment) multiple lines of code in the Source pane of RStudio, highlight the code you want to comment and press `Ctrl + Shift + c` on a PC or `Cmd + Shift + c` on a Mac.

Your turn
Perform the following tasks:

1. Type `1+1` in the Console and press Enter.
2. Open a new Script in RStudio.
3. Type `mean(1:3)` in your Script file.
4. Type `# mean(1:3)` in your Script file.
5. Run the commands from the Script using an approach mentioned above.
6. Save your Script file.
7. Use the keyboard shortcut to "comment out" some of the lines of your Script file.

1.3 Assignment

R works on various types of objects that we'll learn more about later.

To store an object in the computer's memory we must assign it a name using the assignment operator `<-` or the equal sign `=`.

Some comments:

- In general, both `<-` and `=` can be used for assignment.

- Pressing `Alt + -` on a PC or `Option + -` on a Mac will insert `<-` into the R Console and Script files.
  - If you are creating an R Markdown file, then this shortcut will only insert `<-` if you are in an R code block.
- `<-` and `=` are NOT synonyms, but can be used identically most of the time.

It is best to use `<-` for assigning a name to an object and reserving `=` for specifying function arguments. See Section 1.14.1 for an explanation.

Once an object has been assigned a name, it can be printed by running the name of the object in the Console or using the `print` function.

Your turn
Run the following commands in the Console:

```
# compute the mean of 1, 2, ..., 10 and assign the name m
m <- mean(1:10)
m # print m
print(m) # print m a different way
```

After the comment, we compute the sample mean of the values $1, 2, \ldots, 10$, then assign it the name `m`. The next two lines are different mechanisms for printing the information contained in the object `m` (which is just the number 5.5).

1.4 Functions

A function is an object that performs a certain action or set of actions based on objects it receives from its arguments. We use a sequence of function calls to perform data analysis.

To use a function, you type the function's name in the Console (or Script) and then supply the function's "arguments" between parentheses, `()`.

The arguments of a function are pieces of data or information the function needs to perform the requested task (i.e., the function "inputs"). Each argument you supply is separated by a comma, `,`. Some functions have default values for certain arguments and do not need to specified unless something beside the default behavior is desired.

e.g., the `mean` function computes the sample mean of an R object `x`. (How do I know? Because I looked at the documentation for the function by running `?mean` in the Console. We'll talk more about getting help with R shortly.) The `mean` function also has a `trim` argument that indicates the, "... fraction ... of observations to be trimmed from each end of `x` before the mean is computed" (R Core Team (2022), `?mean`).

Consider the examples below, in which we compute the mean of the set of values `1, 5, 3, 2, 10`.

```
mean(c(1, 5, 3, 4, 10))
## [1] 4.6
mean(c(1, 5, 3, 4, 10), trim = 0.2)
## [1] 4
```

The output differs for the two function calls because in the first we compute `(1 + 5 + 3 + 4 + 10)/5 = 23/5 = 4.6` while in the second we remove the first 20% and last 20% of the values (i.e., dropping `1` and `10`) and compute `(5 + 3 + 4)/3 = 12/3 = 4`.

1.5 Packages

Packages are collections of functions, data, and other objects that extend the functionality available in R by default.

R packages can be installed using the `install.packages` function and loaded using the `library` function.

Your turn

The **tidyverse** (https://www.tidyverse.org, Wickham (2022d)) is a popular ecosystem of R packages used for manipulating, tidying, and plotting data. Currently, the **tidyverse** is comprised of the following packages:

- **ggplot2**: A package for plotting based on the "Grammar of Graphics" (Wickham, Chang, et al. 2022).
- **purrr**: A package for functional programming (Henry and Wickham 2022).
- **tibble**: A package providing a more advanced data frame (Müller and Wickham 2022).
- **dplyr**: A package for manipulating data. More specifically, it provides " a grammar of data manipulation" (Wickham, François, et al. 2022).
- **tidyr**: A package to help create "tidy" data (Wickham and Girlich 2022). Tidy data is an data organization style often convenient for data analysis.
- **stringr**: A package for working with character/string data (Wickham 2022b).
- **readr**: A package for importing data (Wickham, Hester, and Bryan 2022).
- **forcats**: A package for working with categorical data (Wickham 2022a).

Install the set of **tidyverse** R packages by running the command below in the Console.

```
install.packages("tidyverse")
```

After you install **tidyverse**, load the package(s) by running the command below.

```
library(tidyverse)
```

You should see something like the following output:

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Different packages may use the same function name to provide certain functionality. The functions will likely be used for different tasks or require different arguments. E.g., You may have noticed when you loaded the **tidyverse** above that `dplyr::lag()` masks `stats::lag()`. What this means is that both the

dplyr and **stats** packages have a function called `lag`.

To refer to a function in a specific package, we should add `package::` prior to the function name. In the code below, we run `stats::lag` and `dplyr::lag` on two different objects using the `::` syntax.

```
stats::lag(1:10, 2)
##  [1]  1  2  3  4  5  6  7  8  9 10
## attr(,"tsp")
## [1] -1  8  1
dplyr::lag(1:10, 2)
##  [1] NA NA  1  2  3  4  5  6  7  8
```

The output returned by the two functions is different because the functions are intended to do different things. The `stats::lag` function call shifts the time base of the provided time series object back 2 units, while the call to `dplyr::lag` provides the values 2 positions earlier in the object. Note: you don't need to understand the `lag` function in the example above. The example is provided to demonstrate how to use the `::` syntax to call to a function in a specific package when the function name has conflicts in multiple packages.

1.6 Getting help

There are many ways to get help in R.

If you know the command for which you want help, then run `?command` (where command is replaced the name of the relevant command) in the Console, to access the documentation for the object. This approach will also work with data sets, package names, object classes, etc. If you need to refer to a function in a specific package, you can use `?package::function` to get help on a specific function, e.g., `?dplyr::filter`.

The documentation will provide:

- A **Description** section with general information about the function or object.
- A **Usage* section with a generic template for using the function or object.
- An **Arguments** section summarizing the function inputs the function needs.
- A **Details** section may be provided with additional information about how the function or object.
- A **Value** section that describes what is returned by the function.
- A **Examples** section providing examples of how to use the function. Usually, these can be copied and pasted into the Console to better understand the function arguments and what it produced.

If you need to find a command to help you with a certain *topic*, then `??topic` will search for the topic through all installed documentation and bring up any

vignettes, code demonstrations, or help pages that include the topic for which you searched.

If you are trying to figure out why an error is being produced, what packages can be used to perform a certain analysis, how to perform a complex task that you can't seem to figure out, etc., then simply do a web search for what you're trying to figure out! Because R is such a popular programming language, it is likely you will find a stackoverflow response, a helpful blog post, an R users forum response, etc., that at least partially addresses your question.

Do the following:

1. Run `?lm` in the Console to get help on the `lm` function, which is one of the main functions used for fitting linear models.
2. Run `??logarithms` in the Console to search the R documentation for information about logarithms. It is likely that you will see multiple Help pages that mention "logarithm", so you may end up needing to find the desired entry via trial and error.
3. Run a web search for something along the lines of "How do I change the size of the axis labels in an R plot?".

1.7 Data types and structures

1.7.1 Basic data types

R has 6 basic ("atomic") vector types (https://cran.r-project.org/doc/manuals/r-release/R-lang.html#Basic-types) (R Core Team 2022):

1. character: collections of characters. E.g., `"a"`, `"hello world!"`.
2. double: decimal numbers. e.g., `1.2`, `1.0`.
3. integer: whole numbers. In R, you must add `L` to the end of a number to specify it as an integer. E.g., `1L` is an integer but `1` is a double.
4. logical: boolean values, `TRUE` and `FALSE`.
5. complex: complex numbers. E.g., `1+3i`.
6. raw: a type to hold raw bytes.

Both double and integer values are specific types of numeric values.

The `typeof` function returns the R internal type or storage mode of any object.

Consider the following commands and output:

```
# determine basic data type
typeof(1)
## [1] "double"
typeof(1L)
## [1] "integer"
```

```
typeof("hello world!")
## [1] "character"
```

1.7.2 Other important object types

There are other important types of objects in R that are not basic. We will discuss a few. The R Project manual provides additional information about available types (https://cran.r-project.org/doc/manuals/r-release/R-lang.html#Basic-types).

1.7.2.1 Numeric

An object is `numeric` if it is of type `integer` or `double`. In that case, it's `mode` is said to be `numeric`.

The `is.numeric` function tests whether an object can be interpreted as numbers. We can use it to determine whether an object is `numeric`, as in the code run below.

```
# is the object numeric?
is.numeric("hello world!")
## [1] FALSE
is.numeric(1)
## [1] TRUE
is.numeric(1L)
## [1] TRUE
```

1.7.2.2 NULL

`NULL` is a special object to indicate an object is absent. An object having a length of zero is not the same thing as an object being absent.

1.7.2.3 NA

A "missing value" occurs when the value of something isn't known. R uses the special object `NA` to represent a missing value.

If you have a missing value, you should represent that value as `NA`. Note: `"NA"` is not the same thing as `NA`.

1.7.2.4 Functions

From R's perspective, a function is simply another data type.

1.7.2.5 A comment about classes

Every R object has a `class` that may be distinct from its type. Many functions will operate differently depending on an object's `class`.

1.7.3 Data structures

R operates on data structures. A data structure is a "container" that holds certain kinds of information.

R has 5 basic data structures:

1. vector.
2. matrix.
3. array.
4. data frame.
5. list.

Vectors, matrices, and arrays are homogeneous objects that can only store a single data type at a time. Data frames and lists can store multiple data types.

Vectors and lists are considered one-dimensional objects. A list is technically a vector. Vectors of a single type are atomic vectors (https://cran.r-project.org/doc/manuals/r-release/R-lang.html#List-objects). Matrices and data frames are considered two-dimensional objects. Arrays can have 1 or more dimensions.

The relationship between dimensionality and data type for the basic data structures is summarized in Table **??**, which is based on a table in the first edition of Hadley Wickham's *Advanced R* (https://adv-r.had.co.nz/Data-structures.html#data-structure).

	data type	
# of dimensions	homogeneous	heterogeneous
1	atomic vector	list
2	matrix	data frame
1 or more	array	

1.8 Vectors

A *vector* is a one-dimensional set of data of the same type.

1.8.1 Creation

The most basic way to create a vector is the `c` (combine) function. The `c` function combines values into an atomic vector or list.

The following commands create vectors of type `numeric`, `character`, and `logical`, respectively.

- `c(1, 2, 5.3, 6, -2, 4)`
- `c("one", "two", "three")`
- `c(TRUE, TRUE, FALSE, TRUE)`

R provides two main functions for creating vectors with specific patterns: `seq` and `rep`.

The `seq` (sequence) function is used to create an equidistant series of numeric values. Some examples:

- `seq(1, 10)` creates a sequence of numbers from 1 to 10 in increments of 1.
- `1:10` creates a sequence of numbers from 1 to 10 in increments of 1.
- `seq(1, 20, by = 2)` creates a sequence of numbers from 1 to 20 in increments of 2.
- `seq(10, 20, len = 100)` creates a sequence of numbers from 10 to 20 of length 100.

The `rep` (replicate) function can be used to create a vector by replicating values. Some examples:

- `rep(1:3, times = 3)` replicates the sequence `1, 2, 3` three times in a row.
- `rep(c("trt1", "trt2", "trt3"), times = 1:3)` replicates `"trt1"` once, `"trt2"` twice, and `"trt3"` three times.
- `rep(1:3, each = 3)` replicates each element of the sequence 1, 2, 3 three times.

Multiple vectors can be combined into a new vector object using the `c` function. E.g., `c(v1, v2, v3)` would combine vectors `v1`, `v2`, and `v3`.

Your turn
Run the commands below in the Console to see what is printed. After you do that, try to answer the following questions:

- What does the `by` argument of the `seq` function control?
- What does the `len` argument of the `seq` function control?
- What does the `times` argument of the `rep` function control?
- What does the `each` argument of the `rep` function control?

```
# vector creation
c(1, 2, 5.3, 6, -2, 4)
c("one", "two", "three")
c(TRUE, TRUE, FALSE, TRUE)
# sequences of values
seq(1, 10)
1:10
seq(1, 20, by = 2)
seq(10, 20, len = 100)
# replicated values
rep(1:3, times = 3)
rep(c("trt1", "trt2", "trt3"), times = 1:3)
rep(1:3, each = 3)
```

Next, we can practice combining multiple vectors using `c`. Run the commands below in the Console.

```
v1 <- 1:5 # create a vector, v1
v2 <- c(1, 10, 11) # create another vector, v2
v3 <- rep(1:2, each = 3) # crate a third vector, v3
new <- c(v1, v2, v3) # combine v1, v2, and v3 into a new vector
new # print the combined vector
```

1.8.2 Categorical vectors

Categorical data should be stored as a `factor` in R. Even though your code related to categorical data may work when stored as `character` or `numeric` data because a cautious developer planned for that possibility, it is best to use good coding practices that minimize potential issues.

The `factor` function takes a vector of values that can be coerced to type `character` and converts them to an object of class `factor`. In the code chunk below, we create two `factor` objects from vectors.

```
# create some factor variables
f1 <- factor(rep(1:6, times = 3))
f1
##  [1] 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6
## Levels: 1 2 3 4 5 6
```

```
f2 <- factor(c("a", 7, "blue", "blue", FALSE))
f2
## [1] a     7     blue  blue  FALSE
## Levels: 7 a blue FALSE
```

Note that when a `factor` object is printed that it lists the `Levels` (i.e., unique categories) of the object.

Some additional comments:

- `factor` objects aren't technically vectors (e.g., running `is.factor(f2)` based on the above code will return `FALSE`) though they essentially behave like vectors, which is why they are included here.
- The `is.factor` function can be used to determine whether an object is a `factor`.
- You can create `factor` objects with specific orderings of categories using the `level` and `ordered` arguments of the `factor` function (see `?factor` for more details).

Your turn
Attempt to complete the following tasks:

1. Create a vector named `grp` that has two levels: `a` and `b`, where the first 7 values are `a` and the second 4 values are `b`.
2. Run `is.factor(grp)` in the Console.
3. Run `is.vector(grp)` in the Console.
4. Run `typeof(grp)` in the Console.

Related to the last task, a `factor` object is *technically* a collection of integers that have labels associated with each unique integer value.
Let's look at creating ordered `factor` objects. Suppose we have categorical data with the categories `small`, `medium`, and `large`. We create a `size` vector with hypothetical data below.

```
size <- c("small", "medium", "small", "large", "medium", "medium", "large")
```

If we convert `size` to a factor, R will automatically order the levels of size alphabetically.

```
factor(size)
## [1] small  medium small  large  medium medium large
## Levels: large medium small
```

This is not technically a problem, but can result in undesirable side effects such as plots with levels in an undesirable order.
To create an ordered vector, we specify the desired order of the levels and set the `ordered` argument to `TRUE`, as in the code below.

```
factor(size, levels = c("small", "medium", "large"), ordered = TRUE)
## [1] small  medium small  large  medium medium large
## Levels: small < medium < large
```

1.8.3 Extracting parts of a vector

Parts a vector can be extracted by appending an index vector in square brackets `[]` to the name of the vector, where the index vector indicates which parts of the vector to retain or exclude. We can include either numbers or logical values in our index vector. We discuss both approaches below.

1.8.3.1 Selection use a numeric index vector

Let's create a `numeric` vector `a` with the values 2, 4, 6, 8, 10, 12, 14, 16.

```
# define a sequence 2, 4, ..., 16
a <- seq(2, 16, by = 2)
a
## [1]  2  4  6  8 10 12 14 16
```

To extract the 2nd, 4th, and 6th elements of `a`, we can use the code below. The

code indicates that the 2nd, 4th, and 6th elements of `a` should be extracted.

```
# extract subset of vector
a[c(2, 4, 6)]
## [1]  4  8 12
```

You can also use "negative" indexing to indicate the elements of the vector you want to exclude. Specifically, supplying a negative index vector indicates the values you want to exclude from your selection.

In the example below, we use the minus (`-`) sign in front of the index vector `c(2, 4, 6)` to indicate we want all elements of `a` EXCEPT the 2nd, 4th, and 6th. The last line of code excludes the 3rd through 6th elements of `a`.

```
# extract part of vector using negative indexing
a[-c(2, 4, 6)] # select all but element 2, 4, 6
## [1]  2  6 10 14 16
a[-(3:6)] # select all but elements 3-6
## [1]  2  4 14 16
```

1.8.3.2 Logical expressions

A logical expression uses one or more logical operators to determine which elements of an object satisfy the specified statement. The basic logical operators are:

- `<`, `<=`: less than, less than or equal to.
- `>`, `>=`: greater than, greater than or equal to.
- `==`: equal to.
- `!=`: not equal to.

Creating a logical expression with a vector will result in a logical vector indicating whether each element satisfies the logical expression.

Your turn
Run the following commands in R and see what is printed. What task is each statement performing?

```
a > 10  # which elements of a are > 10?
a <= 4  # which elements of a are <= 10?
a == 10 # which elements of a are equal to 10?
a != 10 # which elements of a are not equal to 10?
```

We can create more complicated logical expressions using the "and", "or", and "not" operators.

- `&`: and.
- `|`: or.
- `!`: not, i.e., not true.

The `&` operator returns `TRUE` if all logical values connected by the `&` are `TRUE`, otherwise it returns `FALSE`. On the other hand, the `|` operator returns `TRUE` if any logical values connected by the `|` are `TRUE`, otherwise it returns `FALSE`. The `!` operator returns the complement of a logical value or expression.

Your turn
Run the following commands below in the Console.

```
TRUE & TRUE & TRUE
TRUE & TRUE & FALSE
FALSE | TRUE | FALSE
FALSE | FALSE | FALSE
!TRUE
!FALSE
```

What role does `&` serve in a sequence of logical values? Similarly, what roles do `|` and `!` serve in a sequence of logical values?

Logical expressions can be connected via `&` and `|` (and impacted via `!`), in which case the operators are applied elementwise (i.e., to all of the first elements in the expressions, then all the second eleemnts in the expressions, etc).

Your turn
Run the following commands in R and see what is printed. What task is each statement performing? Note that the parentheses `()` are used to group logical expressions to more easily understand what is being done. This is a good coding style to follow.

```
# which elements of a are > 6 and <= 10
(a > 6) & (a <= 10)
# which elements of a are <= 4 or >= 12
(a <= 4) | (a >= 12)
# which elements of a are NOT <= 4 or >= 12
!((a <= 4) | (a >= 12))
```

1.8.3.3 Selection using logical expressions

Logical expressions can be used to return parts of an object satisfying the appropriate criteria. Specifically, we pass logical expressions within the square brackets to access part of a data structure. This syntax will return each element of the object for which the expression is `TRUE`.

Table 1.2: Functions frequently useful for data analysis.

function	purpose
'length'	Determines the length/number of elements in an object.
'sum'	Sums the elements in the object.
'mean'	Computes the sample mean of the elements in an object.
'var'	Computes the sample variance of the elements in an object.
'sd'	Computes the sample standard deviation the elements of an object.
'range'	Determines the range (minimum and maximum) of the elements of an object.
'log'	Computes the (natural) logarithm of elements in an object.
'summary'	Returns a summary of an object. The output changes depending on the class type of the object.
'str'	Provides information about the structure of an object. Usually, the class of the object and some infor

Your turn
Run the following commands in R and see what is printed. What task is each statement performing?

```
# extract the parts of a with values < 6
a[a < 6]
# extract the parts of a with values equal to 10
a[a == 10]
# extract the parts of a with values < 6 or equal to 10
a[(a < 6)|(a == 10)]
```

1.9 Helpful functions

We provide a brief overview of R functions we often use in our data analysis.

1.9.1 General functions

For brevity, Table 1.2 provides a table of functions commonly useful for basic data analysis along with a description of their purpose.

Your turn
Run the following commands in the Console. Determine for yourself what task each command is performing.

```
# common functions
x <- rexp(100) # sample 100 iid values from an Exponential(1) distribution
length(x) # length of x
sum(x) # sum of x
mean(x) # sample mean of x
var(x) # sample variance of x
sd(x) # sample standard deviation of x
range(x) # range of x
log(x) # logarithm of x
summary(x) # summary of x
str(x) # structure of x
```

1.9.2 Functions related to statistical distributions

If you are doing a lot of data analysis, you are likely to be familiar with statistical concepts such as distributions. R is designed specifically for statistical analysis, so it natively includes functionality for determining properties of statistical distributions. R makes it easy to evaluate the cumulative distribution function (CDF) of a distribution, the quantiles of a distribution, the density or mass of a distribution, and to sample random values from a distribution.

Suppose that a random variable X has the `dist` distribution. The function templates in the list below describe how to obtain certain properties of X.

- `p[dist](q, ...)`: returns the cdf of X evaluated at `q`, i.e., $p = P(X \leq q)$.
- `q[dist](p, ...)`: returns the inverse cdf (or quantile function) of X evaluated at p, i.e., $q = \inf\{x : P(X \leq x) \geq p\}$.
- `d[dist](x, ...)`: returns the mass or density of X evaluated at x (depending on whether it's discrete or continuous).
- `r[dist](n, ...)`: returns an independent and identically distributed random sample of size `n` having the same distribution as X.
- The `...` indicates that additional arguments describing the parameters of the distribution may be required.

To determine the distributions available by default in R, run `?Distributions` in the R Console. We demonstrate some of this functionality in the practice below.

Note: If you are using the statistical distribution-related functions in R, it is imperative that you look at the associated documentation to determine the parameterization of the distribution, as this dramatically impacts the results.

Your turn
Run the following commands in R to see the output. What task is each command performing?

```
pnorm(1.96, mean = 0, sd = 1)
qunif(0.6, min = 0, max = 1)
dbinom(2, size = 20, prob = .2)
dexp(1, rate = 2)
rchisq(100, df = 5)
```

Here are descriptions of what each command performs:

- `pnorm(1.96, mean = 0, sd = 1)` returns the probability that a standard normal random variable is less than or equal to 1.96, i.e., $P(X \leq 1.96)$.
- `qunif(0.6, min = 0, max = 1)` returns the value x such that $P(X \leq x) = 0.6$ for a uniform random variable on the interval $[0, 1]$.
- `dbinom(2, size = 20, prob = .2)` returns the probability that X equals 2 when X has a Binomial distribution with $n = 20$ trials and the probability of a successful trial is 0.2.
- `dexp(1, rate = 2)` evaluates the density of an exponential random variable with mean $= 1/2$ (i.e., the reciprocal of the `rate`) at $x = 1$.
- `rchisq(100, df = 5)` draws a sample of 100 observations from a chi-squared random variable with 5 degrees of freedom.

1.10 Data Frames

Data frames are two-dimensional data objects. Each column of a data frame is a vector (or variable) of possibly different data types. This is a *fundamental* data structure used by most of R's modeling software. The class of a **base** R data frame is `data.frame`, which is technically a specially structured `list`.

In general, I recommend *tidy data*, which means that each variable forms a column of the data frame, and each observation forms a row.

1.10.1 Direct creation

Data frames are directly created by passing vectors into the `data.frame` function.

The names of the columns in the data frame are the names of the vectors you give the `data.frame` function. Consider the following simple example.

```
# create basic data frame
d <- c(1, 2, 3, 4)
e <- c("red", "white", "blue", NA)
```

```
f <- c(TRUE, TRUE, TRUE, FALSE)
df <- data.frame(d,e,f)
df
##   d     e     f
## 1 1   red  TRUE
## 2 2 white  TRUE
## 3 3  blue  TRUE
## 4 4  <NA> FALSE
```

The columns of a data frame can be renamed using the `names` function on the data frame and assigning a vector of names to the data frame.

```
# name columns of data frame
names(df) <- c("ID", "Color", "Passed")
df
##   ID Color Passed
## 1  1   red   TRUE
## 2  2 white   TRUE
## 3  3  blue   TRUE
## 4  4  <NA>  FALSE
```

The columns of a data frame can be named when you are first creating the data frame by using `name =` for each vector of data.

```
# create data frame with better column names
df2 <- data.frame(ID = d, Color = e, Passed = f)
df2
##   ID Color Passed
## 1  1   red   TRUE
## 2  2 white   TRUE
## 3  3  blue   TRUE
## 4  4  <NA>  FALSE
```

1.10.2 Importing Data

Direct creation of data frames is only appropriate for very small data sets. In practice, you are likely to have a file that contains the data you want to analyze and you want to import the data into R.

The `read.table` function imports data in table format from file into R as a data frame.

The basic usage of this function is: `read.table(file, header = TRUE, sep = ",")`

- `file` is the file path and name of the file you want to import into R.
  - If you don't know the file path, setting `file = file.choose()` will bring up a dialog box asking you to locate the file you want to import.

- `header` specifies whether the data file has a header (variable labels for each column of data in the first row of the data file).
  - If you don't specify this option in R or use `header = FALSE`, then R will assume the file doesn't have any headings.
  - `header = TRUE` tells R to read in the data as a data frame with column names taken from the first row of the data file.
- `sep` specifies the delimiter separating elements in the file.
  - If each column of data in the file is separated by a space, then use `sep = " "`.
  - If each column of data in the file is separated by a comma, then use `sep = ","`.
  - If each column of data in the file is separated by a tab, then use `sep = "\t"`.

Your turn
Consider reading in a csv (comma separated file) with a header. The file in question contains information related to COVID-19 cases and deaths as of February 4, 2021. The file is available on the internet in the author's GitHub repository. Notice that we specify the path of the file (`https://raw.githubusercontent.com/jfrench/DataWrangleViz/master/data/`) prior to specifying the file name (`covid_dec4.csv`). Since the file has a header, we specify `header = TRUE`. Since the data values are separated by commas, we specify `sep = ","`. Run the code below in your R Console.

```
# import data as data frame
dtf <- read.table(file = "https://raw.githubusercontent.com/jfrench/DataWrangleViz/master/data
                  header = TRUE,
                  sep = ",")
str(dtf)
## 'data.frame':    50 obs. of  7 variables:
##  $ state_name: chr  "Alabama" "Alaska" "Arizona" "Arkansas" ...
##  $ state_abb : chr  "AL" "AK" "AZ" "AR" ...
##  $ deaths    : int  3831 142 6885 2586 19582 2724 5146 782 19236 9725 ...
##  $ population: num  387000 96500 498000 238000 2815000 ...
##  $ income    : int  25734 35455 29348 25359 31086 35053 37299 32928 27107 28838 ...
##  $ hs        : num  82.1 91 85.6 82.9 80.7 89.7 88.6 87.7 85.5 84.3 ...
##  $ bs        : num  21.9 27.9 25.9 19.5 30.1 36.4 35.5 27.8 25.8 27.3 ...
```

Running `str` on the data frame gives us a general picture of the values stored in the data frame.

Note that the `read_table` function in the **readr** package (Wickham, Hester, and Bryan 2022) is perhaps a better way of reading in tabular data and uses similar syntax. To import data contained in Microsoft Excel files, you can use functions available in the **readxl** package (Wickham and Bryan 2022).

1.10.3 Extracting parts of a data frame

R provides many ways to extract parts of a data frame. We will provide several examples using the `mtcars` data frame in the **datasets** package.

The `mtcars` data frame 32 observations of 11 variables. The variables are:

- `mpg`: miles per gallon.
- `cyl`: number of cylinders.
- `disp`: engine displacement (cubic inches).
- `hp`: horsepower.
- `drat`: rear axle ratio.
- `wt`: weight in 1000s of pounds.
- `qsec`: time in seconds to travel 0.25 of a mile.
- `vs`: engine shape (0 = V-shaped, 1 = straight).
- `am`: transmission type (0 = automatic, 1 = manual).
- `gear`: number of forward gears.
- `carb`: number of carburetors.

We load the data set and examine the basic structure by running the commands below.

```
data(mtcars) # load data set
str(mtcars)  # examine data structure
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

We should do some data cleaning on this data set (see Chapter 2), but we will refrain from this for simplicity.

1.10.3.1 Direct extraction

The column variables of a data frame may be extracted from a data frame by specifying the data frame's name, then `$`, and then specifying the name of the desired variable. This pulls the actual variable vector out of the data frame, so the thing extracted is a vector, not a data frame.

Below, we extract the `mpg` variable from the `mtcars` data frame.

```
mtcars$mpg
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

Another way to extract a variable from a data frame as a vector is `df[, "var"]`, where `df` is the name of our data frame and `var` is the desired variable name. This syntax uses a `df[rows, columns]` style syntax, where `rows` and `columns` indicate the desired rows or columns. If either the `rows` or `columns` are left blank, then all `rows` or `columns`, respectively, are extracted.

```
mtcars[,"mpg"]
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

Once again, this action returns a vector, not a data frame. The is because the `[` operator has an argument `drop` that is set to `TRUE` by default by when using `[rows, columns]` style extraction. The `drop` argument controls whether the result is coerced to the lowest possible dimension.

To get around this behavior we can change the `drop` argument to `FALSE`, as shown below (some output suppressed).

```
# extract mpg variable, keep as data frame
mtcars[,"mpg", drop = FALSE]
##                      mpg
## Mazda RX4           21.0
## Mazda RX4 Wag       21.0
## Datsun 710          22.8
....
```

An easier approach to avoid the default `drop` behavior is the slightly different syntax `df["var"]` (notice we no longer have the comma to separate rows and columns). We use this syntax below, suppressing part of the output, for the `mpg` variable in `mtcars`.

```
# extract mpg variable, keep as data frame
mtcars["mpg"]
##                      mpg
## Mazda RX4           21.0
## Mazda RX4 Wag       21.0
## Datsun 710          22.8
....
```

To select multiple variables in a data frame, we can provide a character vector with multiple variable names between `[]`. In the example below, we extract both the `mpg` and `cyl` variables from `mtcars`.

```
mtcars[c("mpg", "cyl")]
##                    mpg cyl
## Mazda RX4         21.0   6
## Mazda RX4 Wag     21.0   6
## Datsun 710        22.8   4
....
```

You can also use numeric indices to directly indicate the rows or columns of the data frame that you would like to extract. Alternatively, you can use variable names for the columns.

- `df[1,]` would access the first row of `df`.
- `df[1:2,]` would access the first two rows of `df`.
- `df[,2]` would access the second column of `df`.
- `df[1:2, 2:3]` would access the information in rows 1 and 2 of columns 2 and 3 of `df`.
- `df[c(1, 3, 5), c("var1", "var2")]` would access the information in rows 1, 3, and 5 of the `var1` and `var2` variables.

We practice these techniques below.

Run the following commands in the Console. Determine what task each command is performing.

```
# Extract parts of a data frame
df3 <- data.frame(numbers = 1:5,
                  characters = letters[1:5],
                  logicals = c(TRUE, TRUE, FALSE, TRUE, FALSE))
df3 # print df3
df3$logicals # extract the logicals vector of df3
df3[1, ] # extract the first column of df3
df3[, 3] # extract the third column of df3
df3[, 2:3] # extract column 2 and 3 of df3
# extract the numbers and logical columns of df3
df3[, c("numbers", "logicals")]
df3[c("numbers", "logicals")]
```

1.10.3.2 Extraction using logical expressions

Logical expressions can be used to subset a data frame.

To select specific rows of a data frame, we use the syntax `df[logical vector, ]`, where `logical vector` is a valid logical vector whose length matches the number of rows in the data frame. Usually, the logical vector is created using a logical expression involving one or more data frame variables. In the code below, we extract the rows of the `mtcars` data frame for which the `hp` variable is more than 250.

```
# extract rows with hp > 250
mtcars[mtcars$hp > 250,]
##                 mpg cyl disp  hp drat   wt qsec vs am gear carb
## Ford Pantera L 15.8   8  351 264 4.22 3.17 14.5  0  1    5    4
## Maserati Bora  15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
```

We can make the logical expression more complicated and also select specific variables using the syntax discussed in Section 1.10.3.1. Below, we extract the rows of `mtcars` with 8 cylinders and `mpg > 17`, while extracting only the `mpg`, `cyl`, `disp`, and `hp` variables.

```
# return rows with `cyl == 8` and `mpg > 17`
# return columns mpg, cyl, disp, hp
mtcars[mtcars$cyl == 8 & mtcars$mpg > 17,
       c("mpg", "cyl", "disp", "hp")]
##                    mpg cyl  disp  hp
## Hornet Sportabout 18.7   8 360.0 175
## Merc 450SL        17.3   8 275.8 180
## Pontiac Firebird  19.2   8 400.0 175
```

1.10.3.3 Extraction using the `subset` function

The techniques for extracting parts of a data frame discussed in Sections 1.10.3.1 and 1.10.3.2 are the fundamental approaches for selecting desired parts of a data frame. However, these techniques can seem complex and difficult to interpret, particularly when looking back at code you have written in the past. A sleeker approach to extracting part of a data frame is to use the `subset` function.

The `subset` function returns the part of a data frame that meets the specified conditions. The basic usage of this function is: `subset(x, subset, select, drop = FALSE)`

- `x` is the object you want to subset.
  - `x` can be a vector, matrix, or data frame.
- `subset` is a logical expression that indicates the elements or rows of `x` to keep (`TRUE` means keep).
- `select` is a vector that indicates the columns to keep.
- `drop` is a logical value indicating whether the data frame should "drop" into a vector if only a single row or column is kept. The default is `FALSE`, meaning that a data frame will always be returned by the `subset` function by default.

There are many clever ways of using `subset` to select specific parts of a data frame. We encourage the reader to run `?base::subset` in the Console for more details.

Your turn
Run the following commands in the Console to use the `subset` function to extract parts of the `mtcars` data framea.

- `subset(mtcars, subset = gear > 4)`. This command will subset the rows of `mtcars` that have more than 4 gears. Note any variables referred to in the `subset` function are assumed to be part of the supplied data frame or are available in memory.
- `subset(mtcars, select = c(disp, hp, gear))`. This command will select the `disp`, `hp`, and `gear` variables of `mtcars` but will exclude the other columns.
- `subset(mtcars, subset = gear > 4, select = c(disp, hp, gear))` combines the previous two subsets into a single command.

An advantage of the `subset` function is that it makes code easily readable. This is important for collaborating with others, including your future self! Using base R, the final code example above would be: `mtcars[mtcars$gear>4, c("disp", "hp", "gear")]`

It is difficult to look at base R code and immediately tell what it happening, so the `subset` function adds clarity.

1.11 Using the pipe operator

R's native pipe operator (`|>`) allows you to "pipe" the object on the left side of the operator into the first argument of the function on the right side of the operator. There are ways to modify this default behavior, but we will not discuss them.

The pipe operator is a convenient way to string together numerous steps in a string of commands. This coding style is generally considered more readable than other approaches because you can incrementally modify the object through each pipe and each step of the pipe is easy to understand. Ultimately, it's a stylistic choice that can decide to adopt or ignore.

Consider the following approaches to extracting part of `mtcars`. We choose the rows for which engine displacement is more than 400 and only keep the `mpg`, `disp`, and `hp` columns. We can do this in a single function call, but the piping approach breaks the action into smaller parts.

```
# two styles for select certain rows and columns of mtcars
subset(mtcars,
       subset = disp > 400,
       select = c(mpg, disp, hp))
##                      mpg disp  hp
## Cadillac Fleetwood  10.4  472 205
## Lincoln Continental 10.4  460 215
```

```
## Chrysler Imperial   14.7  440 230
mtcars |>
  subset(subset = disp > 400) |>
  subset(select = c(mpg, disp, hp))
##                      mpg disp  hp
## Cadillac Fleetwood  10.4  472 205
## Lincoln Continental 10.4  460 215
## Chrysler Imperial   14.7  440 230
```

When reading code with pipes, the pipe can be thought of as the word "then". In the code above, we take `mtcars` *then* subset it based on `disp` and *then* select some columns.

Most parts of the world do not use miles per gallon to measure fuel economy because they don't measure distance in miles nor volume in gallons. A common measure of fuel economy is the liters of fuel required to travel 100 kilometers. Noting that 3.8 liters is (approximately) equivalent to 1 (U.S.) gallon and 1.6 kilometers is (approxiomately) equivalent to 1 mile, we can convert fuel economy of x miles per gallon to liters per 100 kilometers by noting:

$$\frac{1}{x}\frac{\text{gal}}{\text{mi}} \times \frac{3.8}{1}\frac{\text{L}}{\text{gal}} \times \frac{1}{1.6}\frac{\text{mi}}{\text{km}} \times \frac{100\text{ km}}{100\text{ km}} = \frac{237.5}{x}\frac{\text{L}}{100\text{ km}}.$$

Thus, to convert from miles per gallon to liters per 100 kilometers, we take 237.5 and divide by the number of miles per gallon.

In the next set of code, we create a new variable, `lp100km`, in the `mtcars` data frame that describes the liters of fuel each car requires to travel 100 kilometers. Then we select only the columns `mpg` and `lp100km`. We then look at only the first 5 observations. To create the new variable, `lp100km`, we use the `base::transform` function, which allows you to create a new variable from the existing columns of a data frame. Run `?base::transform` in the Console for more details and examples.

```
# create new variable
mtcars2 <- transform(mtcars, lp100km = 237.5/mpg)
# select certain columns
mtcars3 <- subset(mtcars2, select = c(mpg, lp100km))
# print first 5 rows
head(mtcars3, n = 5)
##                    mpg  lp100km
## Mazda RX4         21.0 11.30952
## Mazda RX4 Wag     21.0 11.30952
## Datsun 710        22.8 10.41667
## Hornet 4 Drive    21.4 11.09813
## Hornet Sportabout 18.7 12.70053
```

Next, we perform the actions above with pipes.

```
# create new variable, select columns, extract first 5 rows
mtcars |>
  transform(lp100km = 237.5/mpg) |>
  subset(select = c(mpg, lp100km)) |>
  head(n = 5)
##                   mpg  lp100km
## Mazda RX4         21.0 11.30952
## Mazda RX4 Wag     21.0 11.30952
## Datsun 710        22.8 10.41667
## Hornet 4 Drive    21.4 11.09813
## Hornet Sportabout 18.7 12.70053
```

If we allow ourselves to use parts of the **tidyverse**, we can simplify the code even further, as shown below.

```
mtcars |>
  transform(lp100km = 237.5/mpg) |>
  subset(select = c(mpg, lp100km)) |>
  dplyr::arrange(dplyr::desc(lp100km)) |>
  head(n = 5)
##                      mpg  lp100km
## Cadillac Fleetwood  10.4 22.83654
## Lincoln Continental 10.4 22.83654
## Camaro Z28          13.3 17.85714
## Duster 360          14.3 16.60839
## Chrysler Imperial   14.7 16.15646
```

The function `dplyr::arrange` orders the rows of a data frame based on a column variable, while the `dplyr::desc` causes this to be done in descending order.

1.12 Dealing with common problems

You are going to have to deal with many errors and problems as you use R because of inexperience, simple mistakes, misunderstanding. It happens even to the best programmers.

Every problem is unique, but there are common mistakes that we try to provide insight for below.

`Error in ...: could not find function "..."`. You probably forgot to load the package needed to use the function. You also may have misspelled the function name.

`Error: object '...' not found`. The object doesn't exist in loaded memory. Perhaps you forget to assign that name to an object or misspelled the name of the object you are trying to access.

`Error in plot.new() : figure margins too large`. This typically happens because your Plots pane is too small. Increase its size and try again.

Code was working, but isn't anymore. You may have run code out of order. It may work if you run it in order. Or you may have run something in the Console that you don't have in your Script file. It is good practice to clear your environment (the objects R has loaded in memory) using the broom icon  in the Environment pane and rerun your entire Script file to ensure it behaves as expected.

1.13 Ecosystem debate

We typically prefer performing analysis using the functionality of **base** R, which means we try to perform our analysis with features R offers by default. This will be impossible as we move to more complicated aspects of regression analysis, so we will introduce new packages and functions as we progress.

Many readers may have previous experience working with the **tidyverse** (https://www.tidyverse.org) and wonder how frequently we use **tidyverse** functionality. The **tidyverse** offers a unified framework for data manipulation and visualization that tends to be more consistent than **base** R. However, there are many situations where a **base** R solution is more straightforward than a **tidyverse** solution, not to mention the fact that there are many aspects of R programming (e.g., S3 and S4 objects, method dispatch) that require knowledge of **base** R features. Because the R universe is vast and there are many competing coding styles, we will prioritize analysis approaches using **base** R, which gives users a stronger programming foundation. However, we use analysis approaches from the **tidyverse** when it greatly simplifies analysis, data manipulation, or visualization because it provides an extremely useful feature set.

1.14 Additional information

1.14.1 Comparing assignment operators

As previously mentioned in Section 1.3, both `<-` and `=` can mostly be used interchangeably for assignment. But there are times when using `=` for assignment can be problematic. Consider the examples below where we want to use `system.time` to time how long it takes to draw 100 values from a standard normal distribution and assign it the name `result`.

This code works:

```
system.time(result <- rnorm(100))
##    user  system elapsed
##       0       0       0
```

This code doesn't work:

```
system.time(result = rnorm(100))
## Error in system.time(result = rnorm(100)): unused argument (result = rnorm(100))
```

What's the difference? In the second case, R thinks you are setting the `result` argument of the `system.time` function (which doesn't exist) to the value produced by `rnorm(100)`.

Thus, it is best to use `<-` for assigning a name to an object and reserving `=` for specifying function arguments.

Chapter 2

Data cleaning and exploration

You should explore every data set numerically and visually prior to modeling it. The data exploration process will aid you in finding errors in your data, locating missing values, identifying outliers and unusual observations, finding patterns in your data, deciding on a modeling approach, etc.

In order to properly explore the data, it is likely that you will need to prepare the data. Many data sets are initially poorly structured, have store the variables with an as an inappropriate data type, have poor variable names, etc. The process of preparing the data into a friendly format is known as "cleaning".

A systematic exploration of the data is essential to performing a correct analysis. We will demonstrate a systematic (but not exhaustive) exploration of the `penguins_raw` data set from the **palmerpenguins** package (Horst, Hill, and Gorman 2022).

For brevity, we exclude some of the code output in analysis that follows. The excluded portion will be noted with ....

2.1 Raw Palmer penguins data

A data set in a package can be loaded into memory using the `data` function, specifying the `name` of the data set to be loaded, and specifying the package that contains the data. We do this for the `penguins_raw` data below.

```
data(penguins, package = "palmerpenguins")
```

This command actually loads two data sets: `penguins_raw`, the data set we will be looking at, and `penguins`, a simplified version of `penguins_raw`. Note

that this particular data set loads in a nonstandard way; we would normally expect to load the `penguins_raw` data using `data(penguins_raw, package = "palmerpenguins")`.

We could have also loaded the data set by running the following commands in the Console.

```
library(palmerpenguins)
data(penguins)
```

This second approach is overkill and loads everything the package includes into memory. If you are going to be using many functions or objects from a package, then the second approach is sensible. Otherwise, the first approach is more precise and is better coding practice.

The `penguins_raw` data set provides data related to various penguin species measured in the Palmer Archipelago (Antarctica), originally provided by Gorman, Williams, and Fraser (2014).

The data set includes 344 observations of 17 variables. The variables are:

- `studyName`: a `character` variable indicating the expedition from which the data were collected.
- `Sample Number`: a `numeric` variable denoting the continuous number sequence for each sample.
- `Species`: a `character` variable indicating the penguin species.
- `Region`: a `character` variable denoting the region of the Palmer LTER sampling grid the sample was obtained.
- `Island`: a `character` variable indicating the island on which the penguin was observed.
- `Stage`: a `character` variable indicating the reproductive stage of the observation.
- `Individual ID`: a `character` variable indicating the unique identification number for each individual in the data set.
- `Clutch Completion`: a `character` variable indicating whether the study nest was observed with a "full clutch" of 2 eggs.
- `Date Egg`: a `Date` variable indicating the date that the study nest was observed with 1 egg.
- `Culman Length (mm)`: a `numeric` variable indicating the length of the dorsal ridge of the penguin's bill in millimeters.
- `Culmen Depth (mm)`: a `numeric` variable indicating the indicating the depth of the dorsal ridge of the penguin's bill in millimeters.
- `Flipper Length (mm)`: a `numeric` variable indicating the penguin's flipper length in millimeters.
- `Body Mass (g)`: a `numeric` variable indicating the penguin's body mass in grams.
- `Sex`: a `character` variable indicating the penguin's sex (`FEMALE`, `MALE`)
- `Delta 15 N (o/oo)`: a `numeric` variable indicating the ratio of stable

isotopes 15N:14N.
- `Delta 13 C (o/oo)`: a `numeric` variable indicating the ratio of stable isotopes 15C:12C.
- `Comments`: a `character` variable providing additional information about the observation.

2.2 Initial data cleaning

The `str` function is a great first function to apply on a newly loaded data set that you aren't familiar with because it provides a general overview of the data's structure.

```
str(penguins_raw, give.attr = FALSE)
## tibble [344 x 17] (S3: tbl_df/tbl/data.frame)
##  $ studyName          : chr [1:344] "PAL0708" "PAL0708" "PAL0708" "PAL0708" ...
##  $ Sample Number      : num [1:344] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Species            : chr [1:344] "Adelie Penguin (Pygoscelis adeliae)" "Adelie Penguin (Pyg
##  $ Region             : chr [1:344] "Anvers" "Anvers" "Anvers" "Anvers" ...
##  $ Island             : chr [1:344] "Torgersen" "Torgersen" "Torgersen" "Torgersen" ...
##  $ Stage              : chr [1:344] "Adult, 1 Egg Stage" "Adult, 1 Egg Stage" "Adult, 1 Egg St
##  $ Individual ID      : chr [1:344] "N1A1" "N1A2" "N2A1" "N2A2" ...
##  $ Clutch Completion  : chr [1:344] "Yes" "Yes" "Yes" "Yes" ...
##  $ Date Egg           : Date[1:344], format: "2007-11-11" ...
##  $ Culmen Length (mm) : num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
##  $ Culmen Depth (mm)  : num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
##  $ Flipper Length (mm): num [1:344] 181 186 195 NA 193 190 181 195 193 190 ...
##  $ Body Mass (g)      : num [1:344] 3750 3800 3250 NA 3450 ...
##  $ Sex                : chr [1:344] "MALE" "FEMALE" "FEMALE" NA ...
##  $ Delta 15 N (o/oo)  : num [1:344] NA 8.95 8.37 NA 8.77 ...
##  $ Delta 13 C (o/oo)  : num [1:344] NA -24.7 -25.3 NA -25.3 ...
....
```

We see that the `penguins_raw` object is a `tibble`, a special kind of data frame provided by the **`tibble`** package (Müller and Wickham 2022) as part of the broader **`tidyverse`**. It has 344 rows and 17 columns. In general, a `tibble` will function like a standard data frame, though its behavior may change when **`tidyverse`** packages are loaded. In the code below, we confirm that `penguins_raw` qualifies as a **`base`** R `data.frame`.

```
is.data.frame(penguins_raw)
## [1] TRUE
```

An alternative to `str` is the `glimpse` function from the `dplyr` package. `dplyr::glimpse` also summarizes the structure of an object, but also automatically formats the printed output to the size of the Console to make it more readable. An example is provided below.

```
dplyr::glimpse(penguins_raw)
## Rows: 344
## Columns: 17
## $ studyName             <chr> "PAL0708", "PAL0708", "PAL07~
## $ `Sample Number`       <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 1~
## $ Species               <chr> "Adelie Penguin (Pygoscelis ~
## $ Region                <chr> "Anvers", "Anvers", "Anvers"~
## $ Island                <chr> "Torgersen", "Torgersen", "T~
## $ Stage                 <chr> "Adult, 1 Egg Stage", "Adult~
## $ `Individual ID`       <chr> "N1A1", "N1A2", "N2A1", "N2A~
## $ `Clutch Completion`   <chr> "Yes", "Yes", "Yes", "Yes", ~
## $ `Date Egg`            <date> 2007-11-11, 2007-11-11, 200~
## $ `Culmen Length (mm)`  <dbl> 39.1, 39.5, 40.3, NA, 36.7, ~
## $ `Culmen Depth (mm)`   <dbl> 18.7, 17.4, 18.0, NA, 19.3, ~
## $ `Flipper Length (mm)` <dbl> 181, 186, 195, NA, 193, 190,~
## $ `Body Mass (g)`       <dbl> 3750, 3800, 3250, NA, 3450, ~
## $ Sex                   <chr> "MALE", "FEMALE", "FEMALE", ~
## $ `Delta 15 N (o/oo)`   <dbl> NA, 8.94956, 8.36821, NA, 8.~
## $ `Delta 13 C (o/oo)`   <dbl> NA, -24.69454, -25.33302, NA~
## $ Comments              <chr> "Not enough blood for isotop~
```

Another thing that we notice about `penguins_raw` is that it has terrible variable names. The variable names have a mixture of lowercase and uppercase letters, parentheses, and even spaces! This makes it complicated to access variables in the data frame. To access the flipper length variable, we would have to use something like the command below. Note the `` ` `` around "Flipper Length (mm)" because of the spaces in the variable name.

```
penguins_raw$`Flipper Length (mm)`
##   [1] 181 186 195  NA 193 190 181 195 193 190 186 180 182
##  [14] 191 198 185 195 197 184 194 174 180 189 185 180 187
##  [27] 183 187 172 180 178 178 188 184 195 196 190 180 181
##  [40] 184 182 195 186 196 185 190 182 179 190 191 186 188
##  [53] 190 200 187 191 186 193 181 194 185 195 185 192 184
....
```

In *The tidyverse style guide* (Wickham 2022c), Hadley Wickham recommends:

> Variable and function names should use only lowercase letters, numbers, and _. Use underscores (_) (so called snake case) to separate words within a name.

We will apply this recommendation to the `penguins_raw` data below.

Additionally, many variables will be extraneous for our future analyses, so we will select only the ones that we will use in the future. We the `subset` function to select the `Species`, `Island`, `Culmen Length (mm)`, `Culmen Depth (mm)`, `Flipper Length (mm)`, `Body Mass (g)`, and `Sex` variables of `penguins_raw` and assign

the subsetted data frame the name `penguins_clean`.

```
# select certain columns of penguins_raw, assign new name
penguins_clean <-
  penguins_raw |>
  subset(select = c("Species", "Island", "Culmen Length (mm)", "Culmen Depth (mm)", "Flipper Leng
```

To rename the columns of `penguins_clean`, we use the `names` function to access the column names of the data frame and replace it with a vector containing the desired column names. A second usage of `names` confirms that the data frame now has improved column names.

```
# access column names and replace with new names
names(penguins_clean) <- c("species", "island", "bill_length", "bill_depth", "flipper_length", "b
# look at new column names
names(penguins_clean)
## [1] "species"        "island"         "bill_length"
## [4] "bill_depth"     "flipper_length" "body_mass"
## [7] "sex"
```

There are still some issues with `penguins_clean`. The most notable issues are that the `species`, `island`, and `sex` variables are categorical, but are represented as `character` vectors. These variables should each be converted to a `factor`. We use the `transform` function to convert the each variable to a `factor`. Notice that we must replace the original `penguins_clean` object with the transformed object using the assignment operator. We then run the `str` function to confirm the structure change occurred.

```
# convert sex variable to factor, replace original object
penguins_clean <-
  penguins_clean |>
  transform(species = factor(species), island = factor(island), sex = factor(sex))
# view structure
str(penguins_clean)
## 'data.frame':    344 obs. of  7 variables:
##  $ species       : Factor w/ 3 levels "Adelie Penguin (Pygoscelis adeliae)",..: 1 1 1 1 1 1 1
##  $ island        : Factor w/ 3 levels "Biscoe","Dream",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ bill_length   : num  39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
##  $ bill_depth    : num  18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
##  $ flipper_length: num  181 186 195 NA 193 190 181 195 193 190 ...
##  $ body_mass     : num  3750 3800 3250 NA 3450 ...
##  $ sex           : Factor w/ 2 levels "FEMALE","MALE": 2 1 1 NA 1 2 1 2 NA NA ...
```

Our conversion of `species`, `island`, and `sex` to `factor` variables was successful. However, we notice that the `levels` of `sex` are `MALE` and `FEMALE`, which is visually unappealing. Also, the levels of `species` are extremely long, which can create formatting challenges. We simplify both below. First, we confirm the factor levels of the two variables.

```
# determine levels of species and sex
levels(penguins_clean$species)
## [1] "Adelie Penguin (Pygoscelis adeliae)"
## [2] "Chinstrap penguin (Pygoscelis antarctica)"
## [3] "Gentoo penguin (Pygoscelis papua)"
levels(penguins_clean$sex)
## [1] "FEMALE" "MALE"
```

We now change the levels of each variable in the same order they are printed above and confirm that the changes were successful.

```
# update factor levels of species and sex
levels(penguins_clean$species) <- c("adelie", "chinstrap", "gentoo")
levels(penguins_clean$sex) <- c("female", "male")
# confirm that changes took effect
str(penguins_clean)
## 'data.frame':    344 obs. of  7 variables:
##  $ species       : Factor w/ 3 levels "adelie","chinstrap",..: 1 1 1 1 1 1 1 1 1 1
##  $ island        : Factor w/ 3 levels "Biscoe","Dream",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ bill_length   : num  39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
##  $ bill_depth    : num  18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
##  $ flipper_length: num  181 186 195 NA 193 190 181 195 193 190 ...
##  $ body_mass     : num  3750 3800 3250 NA 3450 ...
##  $ sex           : Factor w/ 2 levels "female","male": 2 1 1 NA 1 2 1 2 NA NA ...
```

Our initial data cleaning process is now completed. As we explore our data further, it may become clear that additional data cleaning is needed.

2.3 Numerical summarization of data

Numerical exploration of a data set generally consists of computing various relevant statistics for each of the variables in a data set in order to summarize the data. A relevant statistic will depend on the type of data being analyzed.

Table 2.1 provides an overview of common numerical summaries used to explore data, the type of data that can be summarized, what is summarized, and the function to compute the summary.

We provide additional explanation about the numeric summaries in what follows.

2.3.1 Numeric data

Numerical exploration of a set of `numeric` values usually focuses on determining the:

1. center
2. spread

Table 2.1: A summary of the numeric summaries frequently used for different types of data, the types of variable they are used for, the information provided, and the R function used to compute it.

numeric summary	variable type	summarizes	R function
mean	‘numeric‘	center	‘mean‘
median	‘numeric‘	center	‘median‘
variance	‘numeric‘	spread	‘var‘
standard deviation	‘numeric‘	spread	‘sd‘
interquartile range	‘numeric‘	spread	‘quantile‘ (modified)
quantiles	‘numeric‘	center and spread	‘quantile‘
correlation	‘numeric‘	similarity	‘cor‘
frequency distribution	‘factor‘	counts	‘table‘
relative frequency distribution	‘factor‘	proportions	‘table‘ (modified)

3. quantiles (less common).

It can also be useful to compute the correlation between two `numeric` variables.

2.3.1.1 Measures of center

The sample mean and median are the most common statistics used to represent the “center” of a set of numeric values.

The sample mean or average is obtained by adding all values in the sample and dividing by the number of observations. The sample mean is the most commonly used measure of center. A weakness of the sample mean is that it is easily affected by outliers (values that are very large or small compared to the rest of the data values). Formally, if $x_1, x_2, \ldots, x_n$ are a sample of n numeric values, then the sample mean is computed as

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

The sample mean can be computed in R using `mean`.

The sample median is the middle value of an ordered set of values (the actual middle value for when the number of values is odd and the average of the two middle values if there are an even number of values). Alternatively, the median is identical to the 0.5 quantile of the data. The median is considered more “resistant” because it is not so greatly affected by outliers. The median of the values 1, 8, 7, 6, 100 is 7. The median of the values 1, 8, 7, 6, 100, 4 is 6.5. The median can be computed in R using `median`.

We compute the mean of the `body_mass` variable of the `penguins_clean` data in the code below.

```
mean(penguins_clean$body_mass)
## [1] NA
```

Why is the result `NA` instead of a number? In general, an `NA` value "poisons" any calculation it is part of, with the function returning `NA`. Even a single `NA` value will cause a calculation to return `NA`, even if there are thousands or millions of non-`NA` values. If you want to compute the sample mean of the non-`NA` values, then you must change the `na.rm` argument of `mean` to `TRUE`, which makes the `mean` function to temporarily remove `NA`s prior to calculation. The `na.rm` argument is provided in many other functions, as we'll see in subsequent examples. We now compute the sample mean and median of the `body_mass` variable in `penguins_clean`, ignoring `NA` values.

```
# compute sample mean and median body_mass, ignoring NAs
mean(penguins_clean$body_mass, na.rm = TRUE)
## [1] 4201.754
median(penguins_clean$body_mass, na.rm = TRUE)
## [1] 4050
```

We see the that average penguin `body_mass` is approximately 4201 grams, while the median value is 4050 grams. Since the median is less than the mean (i.e., large values are pulling the mean in the positive direction) the data *may* be positively skewed, but we will need to look at a histogram or density plot of the data to be sure (these plots are discussed in Sections 2.4.2.3 and 2.4.2.4).

2.3.1.2 Quantiles

We introduce quantiles before spread because we refer to quantiles in the discussion of spread.

Informally, the pth quantile (where $0 \leq p \leq 1$) of a set of values is the value that separates the smallest $100p\%$ of the values from the upper $100(1-p)\%$ of the values. e.g., the 0.25 sample quantile (often called Q1) of a set of values is the value that separates the smallest 25% of the values from the largest 75% of the values.

The `quantile` function is used to compute sample quantiles. There are actually many competing approaches to computing sample quantiles (which we don't discuss or worry about). Run `?quantile` in the Console if you are interested in learning more about the approaches used by R.

Quantiles are useful quantifying both the center (median) and spread (minimum and maxmimum or interquartile range) of a set of values.

We use the `quantile` function to compute the minimum (0 quantile), Q1 (0.25 quantile), median (0.5 quantile), Q3 (0.75 quantile), and maximum (1 quantile) of `body_mass` in the code below. The desired quantiles are provided as a `numeric` vector to the `probs` argument.

```
quantile(penguins_clean$body_mass, probs = c(0, 0.25, 0.5, 0.75, 1),
         na.rm = TRUE)
##   0%  25%  50%  75% 100%
## 2700 3550 4050 4750 6300
```

We see that the smallest and largest body masses are 2700 grams and 6300 grams, so the range of the data is $6300 - 2700 = 3600$ grams. Q1 is 3550 grams, while Q3 is 4750 grams. Since Q3 and the maximum are further from the median than Q1 and the minimum, respectively, this is additional evidence that this variable may be positively skewed (stretched out in the postive direction), but we really need to visualize the data to confirm this.

2.3.1.3 Measures of spread

In addition to identifying the center of a set of values, it is important to measure their spread, i.e., how variable the values are.

The sample variance and standard deviation are the most common measures of spread for numeric values. The sample variance of a set of values is the (approximate) average of the squared deviation of each observation from the sample mean, i.e., the sample variance is

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$

The sample standard deviation is the square root of the sample variance and is generally a more useful measure of spread because it is has the same units as the original data. The larger the standard deviation or variance of a set of values, the more they vary from their sample mean. The sample standard deviation and variance can be greatly affected by outliers. The `var` function computes the sample variance while `sd` computes the sample standard deviation.

The interquartile range is a more resistant measure of spread based on quantiles. The interquartile range is the difference between the 0.75 and 0.25 quantiles of a data set.

The minimum and maximum (in relation to the sample mean or median) can also be used to ascertain the spread of a data set. The minimum and maximum values are computed using the `min` and `max` functions, respectively.

We compute these measures of spread for the `body_mass` variable below.

```
# sample variance
var(penguins_clean$body_mass, na.rm = TRUE)
## [1] 643131.1
# sample standard deviation
sd(penguins_clean$body_mass, na.rm = TRUE)
## [1] 801.9545
# interquartile range (names = FALSE removes text above the results)
```

```
quantile(penguins_clean$body_mass, probs = 0.75,
         na.rm = TRUE, names = FALSE) -
  quantile(penguins_clean$body_mass, probs = 0.25,
           na.rm = TRUE, names = FALSE)
## [1] 1200
# minimum
min(penguins_clean$body_mass, na.rm = TRUE)
## [1] 2700
# maximum
max(penguins_clean$body_mass, na.rm = TRUE)
## [1] 6300
```

The sample variance of `body_mass` is 643131.1 gramsˆ2, which isn't easy to interpret. The sample standard deviation is almost 802 grams. So the "typical" deviation of a `body_mass` value from the sample mean is about 800 grams. The interquartile range is 1200 grams. The minimum and maximum values match what we saw the Section 2.3.1.2.

2.3.1.4 Correlation

The correlation between two `numeric` variables quantifies the strength and direction of their linear relationship. The most common correlation statistic is Pearson's correlation statistic. If x_1, x_2, x_n and $y_1, y_2, \ldots, y_n$ are two sets of `numeric` values, then the sample correlation statistic is computed as

$$r = \frac{1}{n-1} \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{s_x} \right) \left(\frac{y_i - \bar{y}}{s_y} \right),$$

where $\bar{x}$ and s_x denote the sample mean and standard deviation of the x's with $\bar{y}$ and s_y denoting the same thing for the y's. r must be between -1 and 1. The `cor` function can be used to compute the sample correlation between two `numeric` variables.

The closer r is to -1 or 1, the closer the data values fall to a straight line when we plot (x_i, y_i), $i = 1, 2, \ldots, n$ in a scatter plot (discussed in Section). Values close to 0 indicate that there is no linear relationship between the two variables. Negative r values indicate a negative relationship between the two variables (as values for one variable increase, the values for the other variable tend to decrease). Positive r values indicate a positive linear relationship between the two variables (as values for one variable increase, the values of the other variable also tend to increase).

In the code below, we compute the sample correlation between all `numeric` variables in `penguins_clean`. We set `use = "pairwise.complete.obs"` so that all non-`NA` pairs of values are used in the calculation.

```
# determine whether each variable is numeric
num_col <- unlist(lapply(penguins_clean, is.numeric))
# observe results
num_col
##        species         island    bill_length     bill_depth
##          FALSE          FALSE           TRUE           TRUE
## flipper_length      body_mass            sex
##           TRUE           TRUE          FALSE
# compute correlation of numeric variables
cor(penguins_clean[, num_col], use = "pairwise.complete.obs")
##                bill_length bill_depth flipper_length
## bill_length      1.0000000 -0.2350529      0.6561813
## bill_depth      -0.2350529  1.0000000     -0.5838512
## flipper_length   0.6561813 -0.5838512      1.0000000
## body_mass        0.5951098 -0.4719156      0.8712018
##                 body_mass
## bill_length     0.5951098
## bill_depth     -0.4719156
## flipper_length  0.8712018
## body_mass       1.0000000
```

Commenting on the output, we see that the values of each variable are perfectly correlated with themselves (this is always true since the values in each pairs are identical). The correlation between `bill_length` and `body_mass` is 0.87, so the larger a penguin is, the larger its bill tends to be. Perhaps surprisingly, the correlation between `bill_length` and `bill_depth` is -0.24, so the longer a bill becomes, the shallower (narrower) we expect the depth to be. Similarly, the correlation between `bill_depth` and `body_mass` is -0.47, so larger penguins tend to have narrower bills.

We briefly explain why we didn't simply use the `cor` function on `penguins_clean` directly. If we try to use `cor` on `penguins_clean` naively, then R will return an error because not all variables in `penguins_clean` are `numeric`. To account for this, we create a vector that determines whether a variable in `penguins_clean` is `numeric`. Recall that a `data.frame` object is a specially-structured `list` object, with each variable being an element of the `list`. The `lapply` function applies a function (`is.numeric` in this case) to each element of the supplied `list`. In our case, we use this to determine whether each variable is `numeric`. The result is returned as a `list`, so we use the `unlist` function to simplify the `list` to a `vector`.

2.3.2 Categorical data

The statistics mentioned in the previous section are generally not appropriate for a categorical variable. Instead, a frequency distribution or relative frequency distribution might be a useful numeric summary of categorical data.

The `table` function returns a contingency table summarizing the number of observations having each level. Note that by default, the table ignores `NA` values.

We see that for the `sex` variable, there are 165 female penguins and 168 male penguins.

```
table(penguins_clean$sex)
##
## female   male
##    165    168
```

To count the `NA` values (if present), we can set the `useNA` argument of `table` to `"ifany"`.

We see that 11 of the observations had no available information on `sex`.

```
table(penguins_clean$sex, useNA = "ifany")
##
## female   male   <NA>
##    165    168     11
```

A relative frequency distribution summarizes the proportion or percentage of observation with each level of a categorical variable. To compute the relative frequency distribution of a variable, we must divide the frequency distribution by the number of observations. If you want to ignore `NA`s, then you can use the following code, which takes the frequency distribution of `sex` and divides by the number of non-`NA` `sex` values.

```
# divide the frequence distribution of sex by the number of non-NA values
table(penguins_clean$sex)/sum(!is.na(penguins_clean$sex))
##
##    female      male
## 0.4954955 0.5045045
```

We see that slightly under 50% of the `sex` values are female (not accounting for `NA` values) are just over 50% are male.

Are you wondering what `sum(!is.na(penguins_clean$sex))` is doing in the code above? The `is.na` function returns a `TRUE` for each value that is `NA` but otherwise returns `FALSE`. The `!` in front of `is.na` inverts the logical expression so that we are determining whether each value is NOT an `NA` (and returns `TRUE` if the value is not `NA`). It is common in programming associated `TRUE` with `1` and `FALSE` with `0`. So if we `sum` the the values that are not `NA`, that is equivalent to counting the number of non-`NA` observations.

If we want to include the `NA` values in our table, we can use the code below.

```
table(penguins_clean$sex, useNA = "ifany")/length(penguins_clean$sex)
##
##     female       male       <NA>
## 0.47965116 0.48837209 0.03197674
```

We do not know the `sex` of approximately 3% of the penguins observations.

2.3.3 The `summary` function

The `summary` function provides a simple approach for quickly quantifying the center and spread of each `numeric` variable in a data frame or determining the frequency distribution of a `factor` variable. More specifically, the `summary` function will compute the minimum, 0.25 quantile, mean, median, 0.75 quantile, and maximum value of a `numeric` variable and will return the frequency distribution (including `NA` values) of `factor` variable.

A `summary` method is available for a `data.frame` object, which means that we can apply the `summary` function directly to our `penguins_clean` data frame, which we do so below.

```
summary(penguins_clean)
##       species          island     bill_length
##  adelie   :152   Biscoe   :168   Min.   :32.10
##  chinstrap: 68   Dream    :124   1st Qu.:39.23
##  gentoo   :124   Torgersen: 52   Median :44.45
##                                  Mean   :43.92
##                                  3rd Qu.:48.50
##                                  Max.   :59.60
##                                  NA's   :2
##    bill_depth    flipper_length    body_mass
##  Min.   :13.10   Min.   :172.0   Min.   :2700
##  1st Qu.:15.60   1st Qu.:190.0   1st Qu.:3550
##  Median :17.30   Median :197.0   Median :4050
##  Mean   :17.15   Mean   :200.9   Mean   :4202
##  3rd Qu.:18.70   3rd Qu.:213.0   3rd Qu.:4750
##  Max.   :21.50   Max.   :231.0   Max.   :6300
##  NA's   :2       NA's   :2       NA's   :2
##      sex
##  female:165
##  male  :168
##  NA's  : 11
##
##
##
##
```

We conveniently get a numeric summary of all of the variables in our data set (you will see different results for variables that are not `factor` or `numeric` type). The `summary` function all makes it easy to identify the presence of any `NA`s in a variable.

Table 2.2: A summary of common plot types used to explore data, the type of variable(s) they summarize, the number of variables summarized, and the base R and ggplot2 functions used to create the plot.

plot type	variable types	number of variables	base R	ggplo
box plot	'numeric'	univariate	'boxplot'	'geon
histogram	'numeric'	univariate	'hist'	'geon
density plot	'numeric'	univariate	'plot', 'density'	'geon
bar plot	'factor'	univariate	'plot' or 'barplot', 'table'	'geon
scatter plot	2 'numeric'	bivariate	'plot'	'geon
parallel box plot	1 'numeric', 1 'factor'	bivariate	'plot' or 'boxplot'	'geon
grouped scatter plot	2 'numeric', 1 'factor'	multivariate	'plot'	'geon
facetted plots	mixed	multivariate	none	'facet
interactive plots	mixed	multivariate	none	'plotl

2.4 Visual summaries of data

Visual summaries (i.e., plots) of data are vital to understanding your data prior to modeling. They help us to spot errors in our data, unusual observations, and simple patterns. They are also important after modeling to communicate the results of your analysis.

We will introduce basic visualization approaches using **base** R functions as well as the popular **ggplot2** package (Wickham, Chang, et al. 2022). It is important to know the basic plotting capabilities of **base** R (particularly the `plot` function, which has been extended by many packages to provide standard plots for complex objects produced by those packages). However, **ggplot2** is able to produce complex graphics with automated legends in a consistent, systematic way, which provides it advantages over **base** graphics in many contexts.

Table 2.2 provides an overview of different plots types that can be used to summarize data, the type of data being summarized, whether the plot is for univariate (one variable), bivariate (two variable), or multivariate (3 or more variables) data, the **base** R functions create the plot, and the main **ggplot2** functions need to create the plot. The table is not intended to be an exhaustive list of useful graphics you should use for data exploration.

2.4.1 The ggplot recipe

There are 4 main components needed to produce a graphic using **ggplot2**.

1. A data frame containing your data.
    - Each column should be a variable and each row should be an observation of data.
2. A `ggplot` object.
    - This is initialized using the `ggplot` function.

3. A geometric object.
    - These are called "geoms" for short.
    - geoms indicate the geometric object used to visualize the data. E.g., points, lines, polygons etc. More generally, geoms indicate the type of plot that is desired, e.g., histogram, density, or boxplot, which aren't exactly a simple geometric argument.
4. An aesthetic.
    - An aesthetic mapping indicates what role a variable plays in the plot.
    - e.g., which variable will play the "x" variable in the plot, the "y" variable in the plot, control the "color" of the observations, etc.

We add "layers" of information to a `ggplot`, such as geoms, scales, or other customizations, using `+`.

2.4.2 Univariate plots

A univariate plot is a plot that only involves a single variable. Examples include bar plots, boxplots, histograms, density plots, dot plots, pie charts, etc. (the last two are are generally poor choices.)

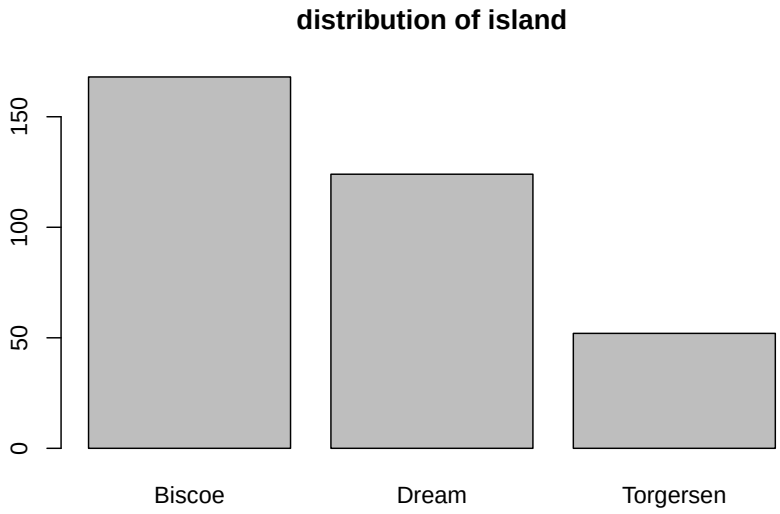
2.4.2.1 Bar plots

A bar plot (or bar chart) displays the number or proportion of observations in each category of a categorical variable (or using R terminology, each `level` of a `factor` variable).

What are you looking for? Generally, categories that have substantially more or fewer observations than the other categories.

The simplest way to create a bar plot in base R is using the `plot` function on a `factor`. In the code below, we create a bar plot for the `island` variable of `penguins_clean`. We use the `main` argument to add a title to the plot.

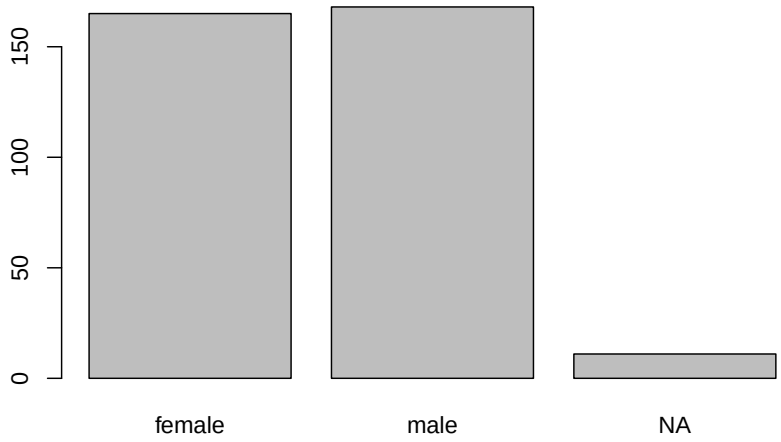
```
plot(penguins$island, main = "distribution of island")
```



We see that the largest number of penguins were obserevd on Biscoe allowed, followed by Dream and Torgersen islands, respectively.

Alternatively, we can combine `barplot` with the `table` function. We do so below for the `sex` variable. To account for `NA`s in the `sex` variable, we specify `useNA = "ifany"` in the `table` function. Also, we specify `names.arg = ...` to specify the bar names, otherwise the bar for `NA` will be blank.

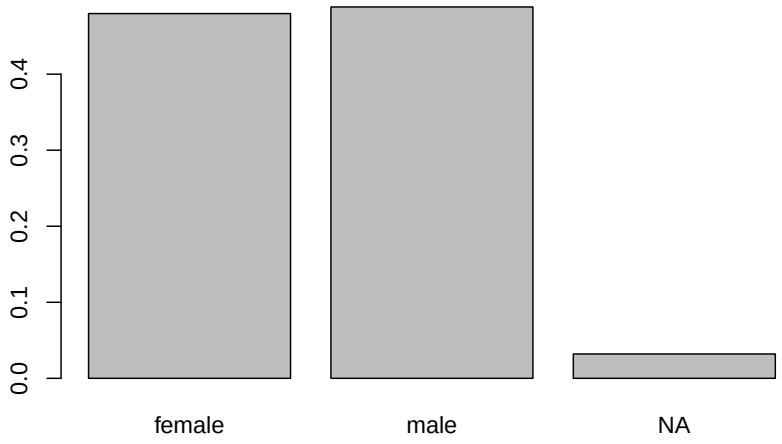
```
barplot(table(penguins_clean$sex, useNA = "ifany"),
        names.arg = c("female", "male", "NA"))
```



We see that that there are slightly more male than female penguins, with a handful of observations missing this characteristic.

To create a relatively frequency bar plot, we should divide the results of `table` by the number of relevant observations. For this particular example, we could use the code below. We use the `length` function to determine the number of observations to divide the counts with.

```
barplot(table(penguins_clean$sex, useNA = "ifany") /
          length(penguins_clean$sex),
        names.arg = c("female", "male", "NA"))
```

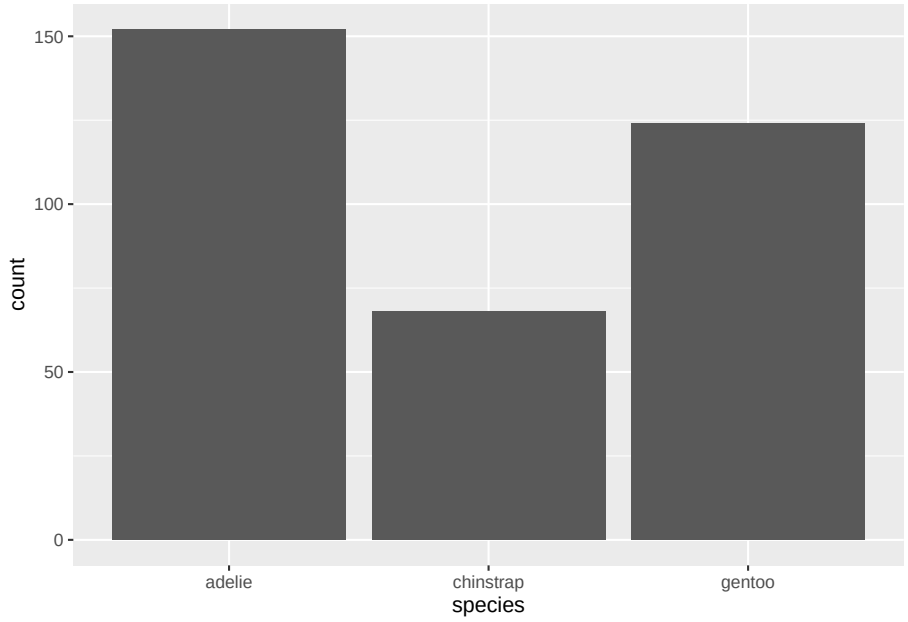


To create a bar plot with **ggplot2**, we first create a basic `ggplot` object containing our data. Make sure to load the **ggplot2** package prior to creating the plot, otherwise you'll get errors!

```
# load ggplot2 package
library(ggplot2)
# create generic ggplot object with our data
gg_penguin <- ggplot(data = penguins_clean)
```

`gg_penguin` is a minimal `ggplot` object with the raw information needed to produce future graphics. To create a bar plot, we add the geom `geom_bar` and map the `species` variable (in this example) to the `x` aesthetic using the `aes` function.

```
# create bar plot for species variable
gg_penguin + geom_bar(aes(x = species))
```



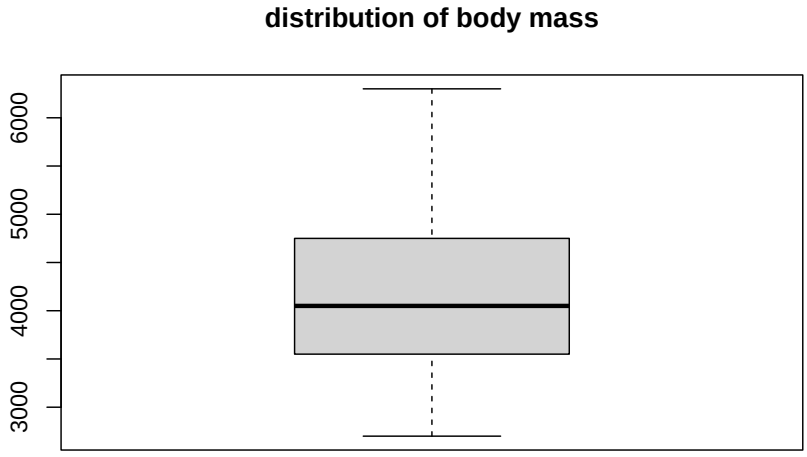
2.4.2.2 Box plots

A box plot is a simple graphic showing critical quantiles of a `numeric` variable, as well as outliers. A box plot indicates the median, 0.25 quantile (Q1), 0.75 quantile (Q3), and extend bars to the largest and smallest observations that are not outliers. Outliers are usually marked with starts or dots. The standard definition of an outlier in the context of box plots is an value that is more than Q3 + 1.5 (Q3 - Q1) and less than Q1 - 1.5 (Q3 - Q1). The box of a box plot extends from Q1 to Q3, with a line in the box indicating the median.

Box plots are useful for identifying outliers and skewness in the variable. However, box plot throw away a lot of information, so be cautious in making conclusions about skewness and modality without seeking a histogram or density plot of the data.

The `boxplot` function is the easiest approach for producing a box plot using base R. We do so for the `body_mass` variable below.

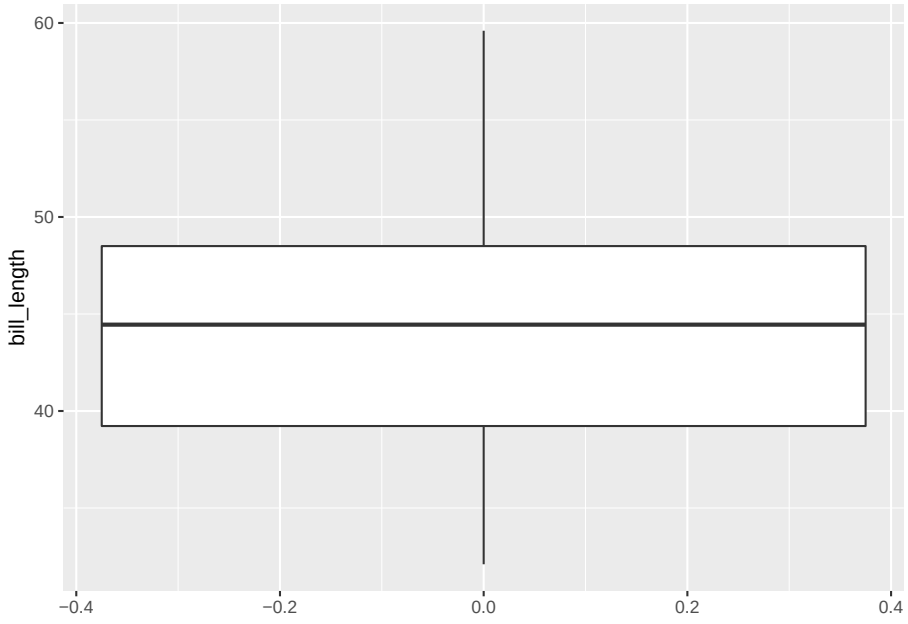
```
boxplot(penguins_clean$body_mass, data = penguins_clean,
        main = "distribution of body mass")
```



The `body_mass` variable doesn't have any outliers. It's has perhaps a slight positive skew since the upper tail and upper part of the box are longer than the lower tail and lower part of the box. The median value is a bit larger than 4000 grams, while 50% of the data being between approximatly 3500 and 4750 grams (i.e., between Q1 and Q3).

To create a box plot using **ggplot2**, we use `geom_boxplot`. We create a box plot for the `bill_length` variable below. We map `bill_length` to the `y` aesthetic so that we get a vertically-oriented box plot (mapping it to `x` will produce a horizontal box plot).

```
gg_penguin + geom_boxplot(aes(y = bill_length))
## Warning: Removed 2 rows containing non-finite values
## (stat_boxplot).
```



There are not `bill_length` outliers. The minimum value is approximately 32 mm and the maximum is almost 60 mm. Q1 is approximately 39 mm, the median is approximately 44 mm, and Q3 is approximately 48 mm. It is difficult to assess the skewness of this data. The upper tail is longer than the shorter tail, but the upper box is shorter than the lower box, so the evidence is inconsistent.

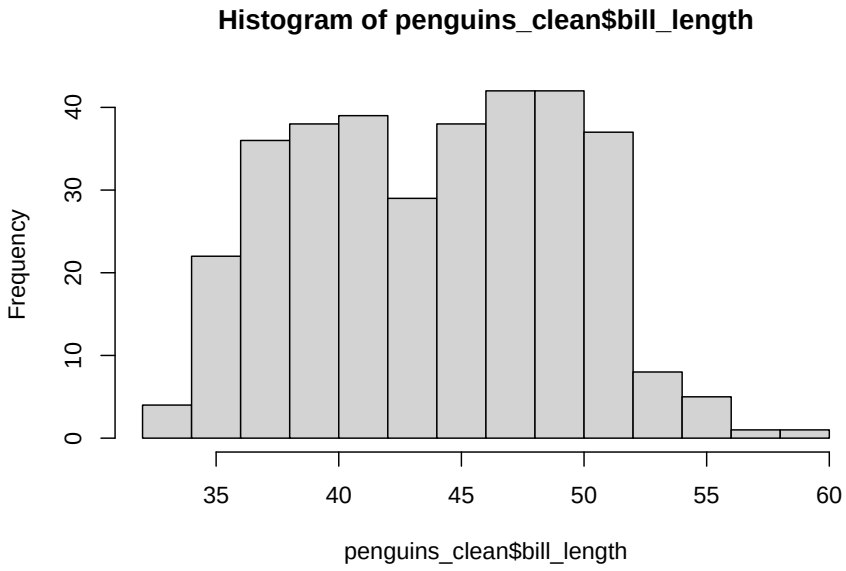
2.4.2.3 Histograms

A histogram displays the distribution of a `numeric` variable. A histogram counts the number of values falling into (usually) equal-sized "bins" running from the smallest value to the largest value. The number of bins and width of the bins affect the shape of the histogram.

Histograms are used to assess skewness, modality (the number of clear "peaks" in the plot), and to some extent, outliers.

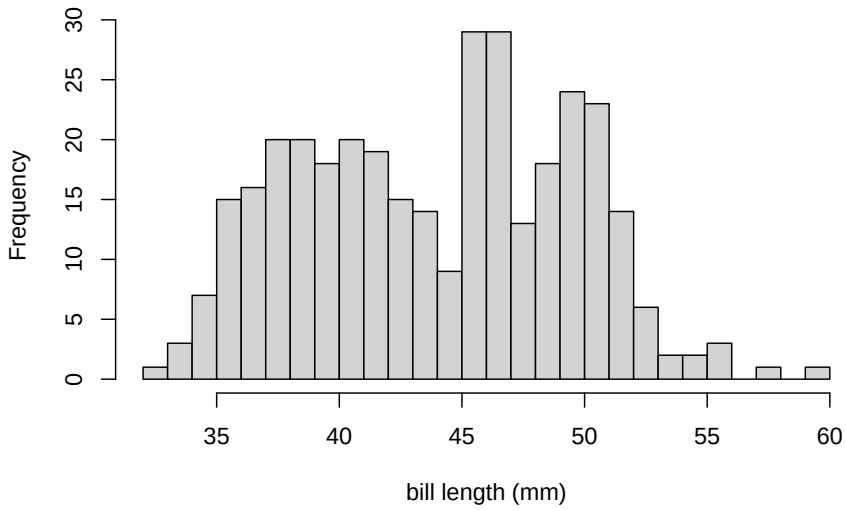
The `hist` function is used create a histogram of a `numeric` variable. We augment the information already learned about `bill_length` with a histogram in the code below.

```
hist(penguins_clean$bill_length)
```



The title and x-axis label are visually unappealling, so we use the `main` and `xlab` arguments to change them to nothing (i.e., no title) and `bill length (mm)`. We also increase the number of bins using the `breaks` argument.

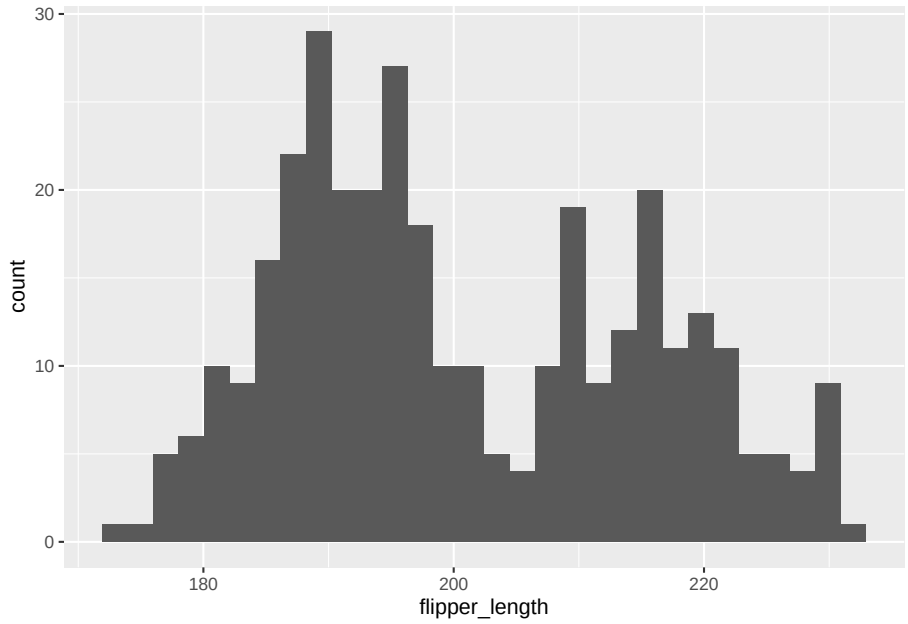
```
hist(penguins_clean$bill_length, main = "",
     xlab = "bill length (mm)", breaks = 20)
```



This distribution of `bill_length` is bimodal (has two prominent peaks or modes). That is why the skewness information on the box plot of `bill_length` was inconsistent. This also demonstrates why should should never draw conclusions like modality or skewness from numeric summaries alone.

We use `geom_histogram` to create a histogram using **`ggplot2`**, mapping the variable to the `x` aesthetic. We do so for the `flipper_length` variable below.

```
gg_penguin + geom_histogram(aes(x = flipper_length))
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
## Warning: Removed 2 rows containing non-finite values
## (stat_bin).
```



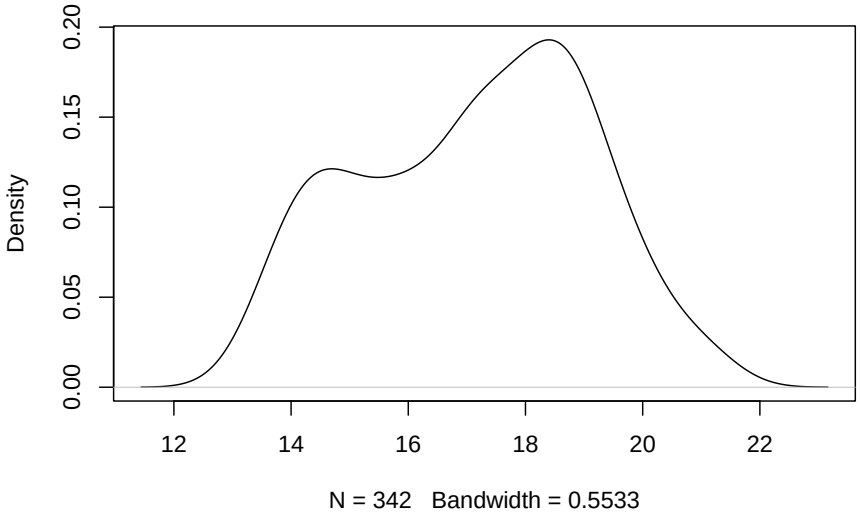
Flipper length is also bimodal, with prominent peaks approximately centered around 190 and 220 mm.

2.4.2.4 Density plots

A density plot is similar to a smoothed histogram and the area under the smoothed curve must equal 1. In general, density plots are more visually appealing than histograms, but both communicate similar information. However, density plots can sometimes have problems near the edges of a variable with a fixed upper or lower bound because it is difficult to know how to smooth the data in that case.

The `plot` and `density` function can be combined to construct a density plot using **`base`** R. We do that below for the `bill_depth` variable below. Note the use of `na.rm` to remove `NA` that would otherwise poison the density calculation, and use `main` to have a blank title.

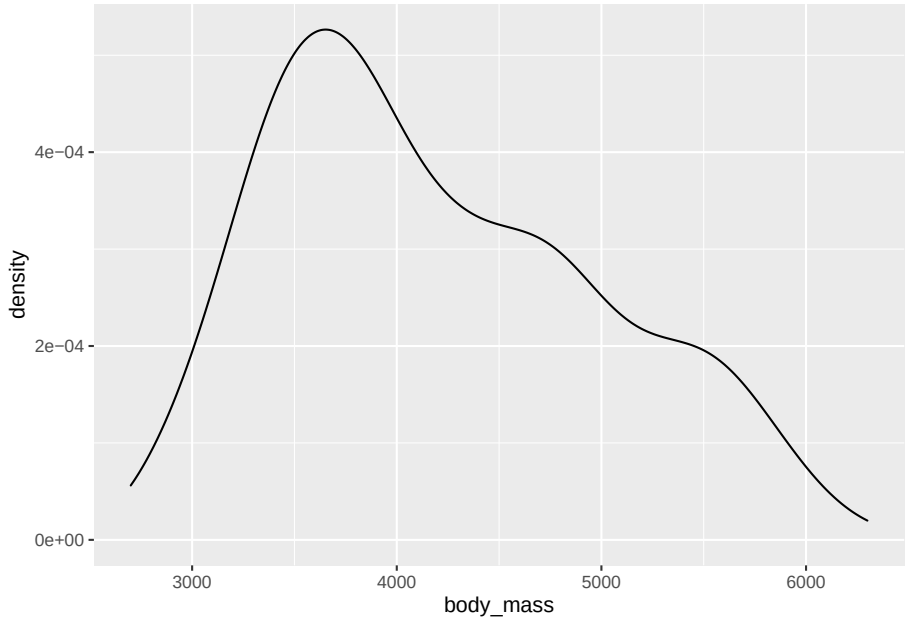
```
plot(density(penguins_clean$bill_depth, na.rm = TRUE), main = "")
```



The `bill_depth` variable is bimodal with peaks around 14 mm and 18 mm. The graphic also indicates that 342 observations were used to estimate the density and the bandwidth parameter was 0.5533. The bandwidth parameter controls the amount of smoothing and can be changed. Run `?stats::density` in the Console for more details.

We create a density plot with **ggplot2** using `geom_density`. We do so for the `body_mass` variable, mapping it to the `x` aesthetic.

```
gg_penguin + geom_density(aes(x = body_mass))
## Warning: Removed 2 rows containing non-finite values
## (stat_density).
```



The `body_mass` variable is unimodal, with a peak around 3700 grams. It is also positively skewed.

2.4.3 Bivariate plots

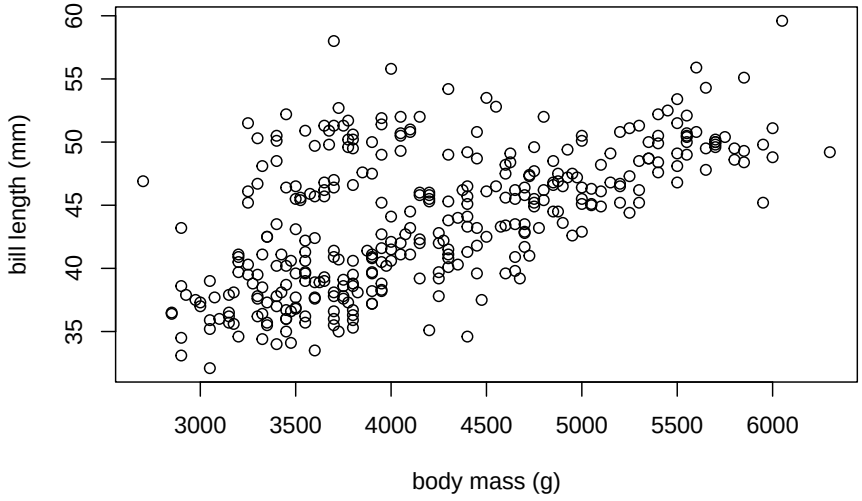
A `bivariate` plot is a plot involving two variables. A `bivariate` plot can involve more than one data type.

2.4.3.1 Scatter plots

Scatter plots can be used to identify the relationship between two `numeric` variables.

We use the `plot` function to create a scatter plot of `bill_length` versus `body_mass` (the `y` variable versus the `x` variable) using `base` R below. The `plot` function is very flexible and can be used multiple ways to produce a scatter plot, but we will use the `formula` method that takes a formula describing the variables (`y ~ x`) and the data frame from which the variables come.

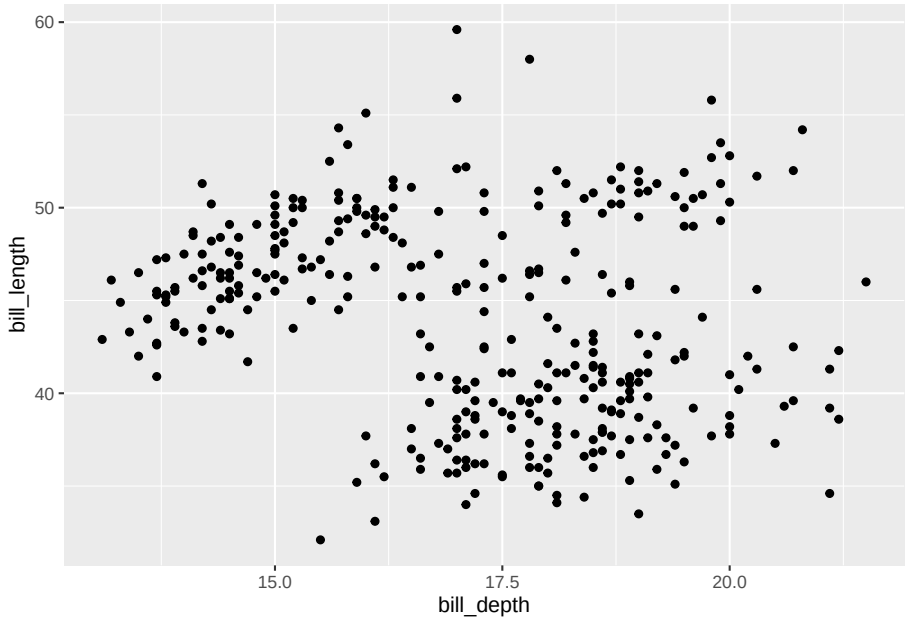
```
# xlab and ylab are used to customize the x-axis and y-axis labels
plot(bill_length ~ body_mass, data = penguins_clean,
     xlab = "body mass (g)", ylab = "bill length (mm)")
```



There is a positive linear relationship between `body_mass` and `bill_length`. As `body_mass` increases, `bill_length` tends to increase.

The `geom_point` function can be used to create a scatter plot with **`ggplot2`**. we make the variables to be plotted to the `x` and `y` aesthetics. We create a scatter plot of `bill_length` versus `bill_depth`.

```
gg_penguin + geom_point(aes(x = bill_depth, y = bill_length))
## Warning: Removed 2 rows containing missing values
## (geom_point).
```



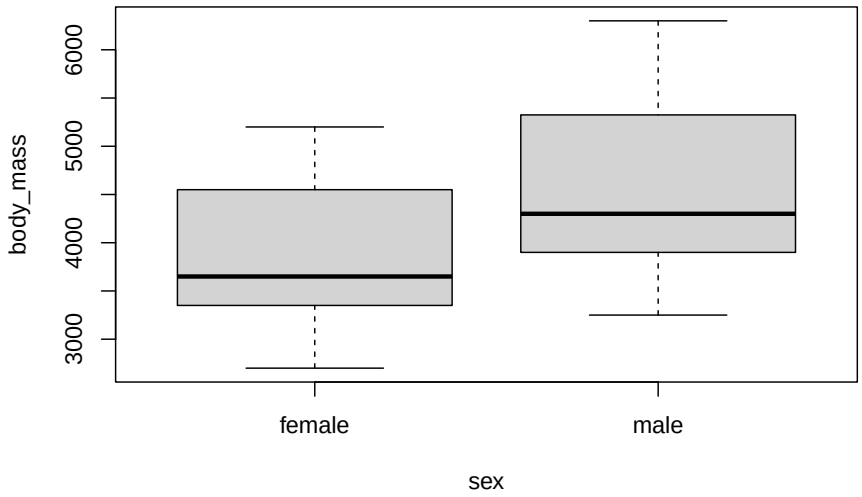
It's difficult to make any definitive conclusion about the relationship of these two variables from this plot, as the plot is basically a blob of points.

2.4.3.2 Parallel box plots

A parallel box plot is used to display the distribution of a `numeric` variable whose values are grouped based on each `level` of a `factor` variable. Specifically, a box plot of the `numeric` variable is is constructed for all the values associated with the levels of the `factor`. Parallel box plot are useful for determining if the distribution of a `numeric` variable substantially changes based on whether an observation has a certain `level` of a `factor`.

Once again use the `plot` function with a `formula`, we can create a parallel box plot easily, with the syntax of the formula being `numeric variable ~ factor variable` (you can reverse these to change the box plot orientation). We parallel box plots of `body_mass` versus `sex` below.

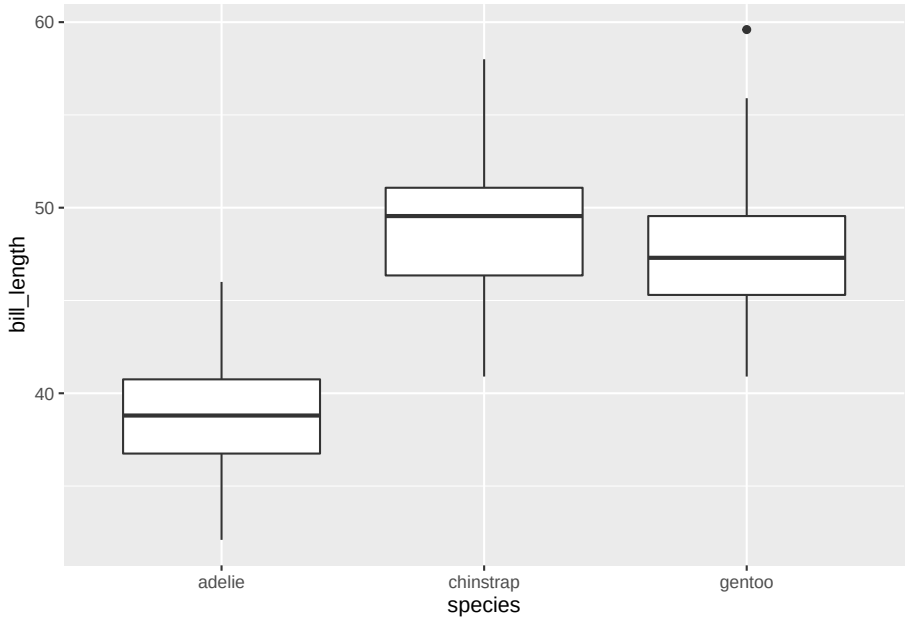
```
plot(body_mass ~ sex, data = penguins_clean)
```



We can see that the `body_mass` values tend to be larger for the male penguins compared to the female penguins.

We can produce something similar with **ggplot2** by specifying both the `y` and `x` aesthetics of for `geom_boxplot`. We do so below to compare `bill_length` for the different penguin `species`.

```
gg_penguin + geom_boxplot(aes(x = species, y = bill_length))
## Warning: Removed 2 rows containing non-finite values
## (stat_boxplot).
```



From a quick glance, we are able to see from the graphic above that the chinstrap penguins tend to have slightly larger bill lengths compared to the gentoo penguins, which typically have larger bill lengths than the adelie penguins.

2.4.4 Multivariate plots

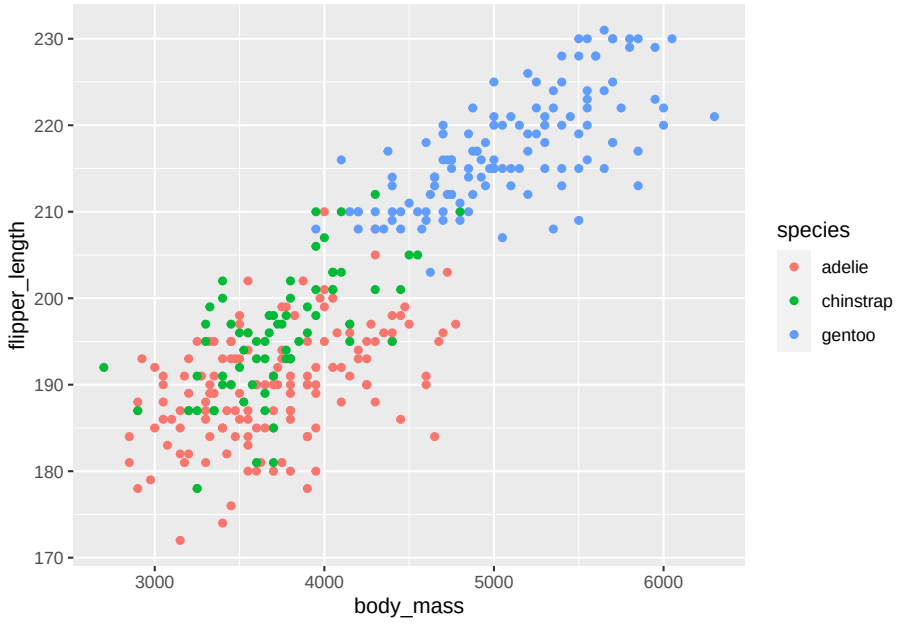
A multivariate plot displays relationships between 2 or more variables (so bivariate plots are technically multivariate plots). We focus on multivariate plots using `ggplot2`. While the same graphics can be created with `base` R, it is substantially quicker to create an initial version of multivariate graphics with `ggplot2`.

2.4.4.1 Grouped scatter plot

A grouped scatter plot is a scatter plot that uses colors or symbols (or both) to indicate the `level` of a `factor` variable that each point corresponds to. You can actually use more than one `factor` variable, but interpretation often becomes much more difficult. The most common way to create a grouped scatter plot with `ggplot2` is to map a `factor` variable to the `color` or `shape` aesthetic of `geom_point`. `ggplot2` will automatically map the `factor` variable to unique colors or shapes and then indicates the mapping in a legend (this process is known as "scaling".

In the example below, we create a scatter plot of `flipper_length` versus `body_mass` that distinguishes the different `species` using `color`.

```
gg_penguin + geom_point(aes(x = body_mass, y = flipper_length,
                            color = species))
## Warning: Removed 2 rows containing missing values
## (geom_point).
```



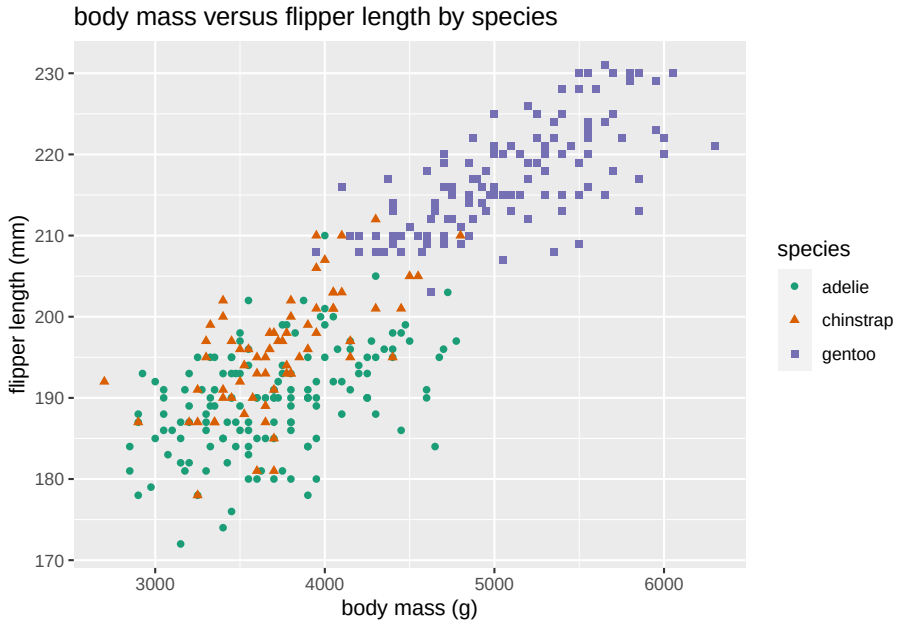
The flipper length and body mass of gentoo penguins tend to be noticeably larger than the other two species, and there is the cluster of gentoo penguis is noticeably different in the plot. Chinstrap and adelie penguins tend to have similar flipper length and body mass, with chinstrap penguins tending to have slightly longer flipper length.

The graphic above can be made better in two ways. 1. Using better colors, and 2. Using more than one visual approach to distinguishing the levels of a `factor` variable.

Color blindness is a common visual impairment. The colors used above use both red and green, which may be difficult to distinguish. We should use a more friendly color palette. An excellent resource for choosing a color palette is https://colorbrewer2.org (Brewer (2022)). The webpage allows you to choose a color palette based on certain desired characteristics such as whether the palette is colorblind-friendly, printer friendly, etc. The recommend palettes can be accessed using the `scale_color_brewer` function. We use colorblind-friendly palette below. We also added a few additional customizations below.

```
gg_penguin +
  geom_point(aes(x = body_mass, y = flipper_length,
```

```
                  color = species, shape = species)) +
  scale_color_brewer(type = "qual", palette = "Dark2") +
  xlab("body mass (g)") + ylab("flipper length (mm)") +
  ggtitle("body mass versus flipper length by species")
## Warning: Removed 2 rows containing missing values
## (geom_point).
```

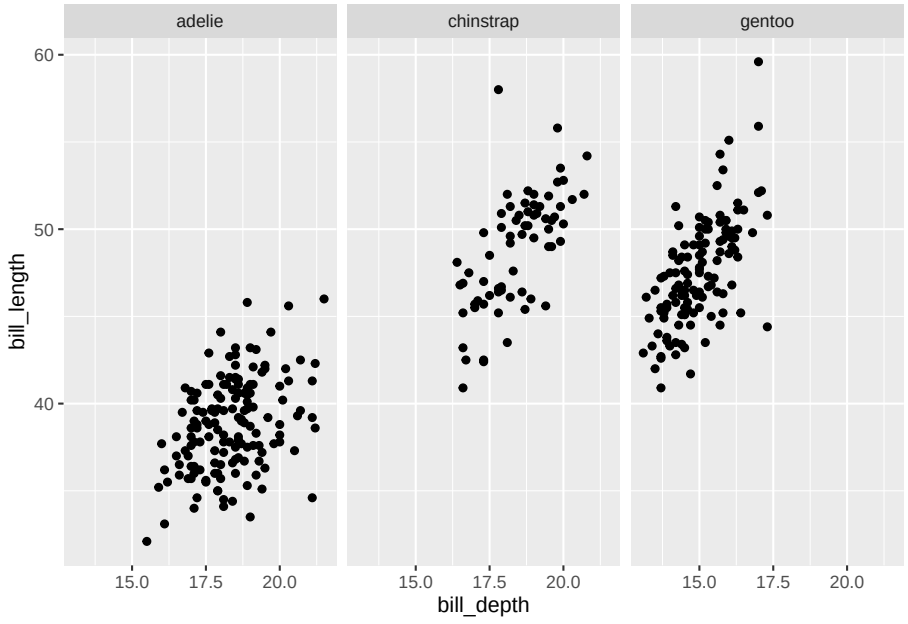


2.4.5 Facetted plots (and alternatives)

Facetting creates separate panels (facets) of plots based on one or more facetting variables. The key functions to do this with **ggplot2** are the `facet_grid` and `facet_wrap` functions. `facet_grid` is used to create a grid of plots based on one or two `factor` variables, while `facet_wrap` wraps facets of panels around the plot. We

Below, we facet scatter plots of `bill_length` versus `bill_depth` by `species`.

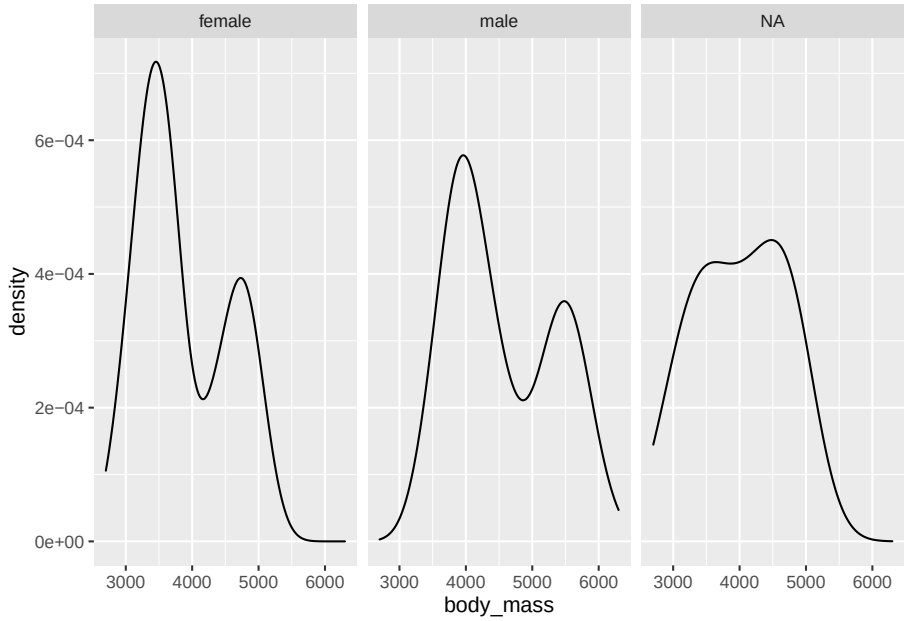
```
gg_penguin +
  geom_point(aes(x = bill_depth, y = bill_length)) +
  facet_grid(~ species)
## Warning: Removed 2 rows containing missing values
## (geom_point).
```



Whereas we previously couldn't discern a relationship between bill length and depth based on a single scatter plot, facetting by `species` makes it clear there is a positive relationship between `bill_length` and `bill_depth`. We could have used a group scatter plot for the same thing.

A simpler facetting example would be to facet density plots of `body_mass` by `sex` as shown below.

```
gg_penguin + geom_density(aes(x = body_mass)) + facet_grid(~sex)
## Warning: Removed 2 rows containing non-finite values
## (stat_density).
```



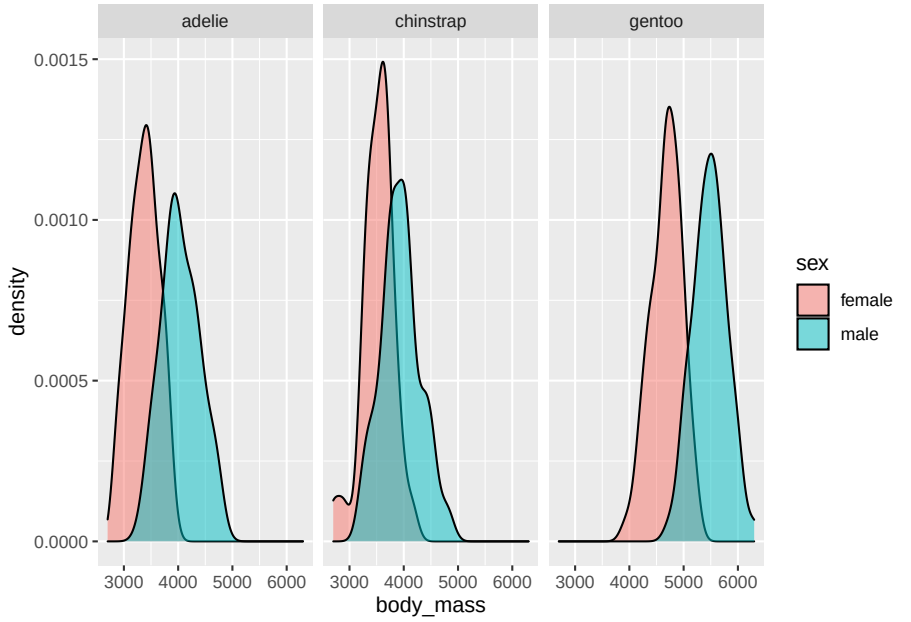
This plot is a bit difficult to interpret. We see that body mass is bimodal for the males and females. Perhaps this is related to `species`. Since the density plots are in different panels, its a bit tricky to see how they relate to each other. Also, the `NA` panel is probably not needed.

To get rid of the `NA` panel, we need to remove all of the observations with `NA` values. We do this below, using `subset` to select the desired columns and then using `na.omit` to remove any rows that have `NA` values for `body_mass`, `sex`, or `species`. Note that order matters here because `na.omit` removes any observation of the data frame that has an `NA` row. We save the filtered object as `penguins_temp`.

```
penguins_temp <-
  penguins_clean |>
  subset(select = c(body_mass, sex, species)) |>
  na.omit()
```

In the next plot, we create density plots of the `body_mass` variable. However, we use the `fill` aesthetic to scale the `sex` variable so that the we distinguish the densities of male and female penguins with different colors. We set the `alpha` argument to 0.5 OUTSIDE the `aes` function (because it is being manually specified) so that the colors are translucent and blend. We also facet by species to see what the patterns look like for the different species.

```
ggplot(data = penguins_temp) +
  geom_density(aes(x = body_mass, fill = sex), alpha = 0.5) +
  facet_grid(~ species)
```



We see that for all species, the body mass of the males tends to be larger than the females.

The examples above provide a small taste of the complicated graphics you can create with `ggplot2` using only a few lines of code.

2.4.5.1 Interactive graphics

There are many tools for creating interactive graphics in R. We have found the `ggiraph` package (Gohel and Skintzos 2022) useful for creating interactive graphics based on `ggplot2`. However, it is a bit too complex to discuss here.

The `plotly` package (Sievert et al. 2021) is an R package to provide the capabilities of plotly https://plotly.com/, a well-known tool for creating interactive scientific plots. The `ggplotly` function will instantly make a `ggplot` interactive (though you may need to customize it for your needs). We provide two examples below.

First, we load the `plotly` package to have access to the `ggplotly` function. We then take our previous grouped scatter plot that plotted `flipp_length` versus `body_mass` distguishing by `species` and assign it the name `ggi`. We then use the `ggplotly` function in make the graphic interactive. When you hover over a point, the plot interactively provides the exact `body_mass` value, `flipper_length` value, and `species` of the observation.

```
# load plotly package
library(plotly)
```

```
##
## Attaching package: 'plotly'
## The following object is masked from 'package:ggplot2':
##
##     last_plot
## The following object is masked from 'package:stats':
##
##     filter
## The following object is masked from 'package:graphics':
##
##     layout
# assign grouped scatter plot name
ggi <-
  gg_penguin +
  geom_point(aes(x = body_mass, y = flipper_length,
                 color = species, shape = species)) +
  scale_color_brewer(type = "qual", palette = "Dark2") +
  xlab("body mass (g)") + ylab("flipper length (mm)") +
  ggtitle("body mass versus flipper length by species")
# make plot interactive
ggplotly(ggi)
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is
```

```
# assign parallel box plot name
ggi2 <-
  gg_penguin +
  geom_boxplot(aes(x = species, y = bill_length))
# make plot interactive
ggplotly(ggi2)
## Warning: Removed 2 rows containing non-finite values
## (stat_boxplot).
```

The interactive parallel box plot provides information about the box plot of each species (such as the minimum `bill_length`, Q1, median, Q3, etc.)

2.5 A plan for data cleaning and exploration

We have provides many examples of data cleaning and exploration using the `penguins_raw` data. The analysis above is NOT exhaustive, and there are many additional numeric and visual summaries we could consider. We summarize a basic plan below for initial data cleaning and exploration that we have found useful. You will likely augment this process based on your own preferences and things you learn from the data. We assume you are working with a data frame, which is the most common data structure used for data analysis in R.

1. Import or the data set.

2. Use the `str` function to get an idea of the initial structure. This can help to identify clear issues you may have had in importing the data, problems with variable names and types, etc.
3. Clean the variable names based on your preferences.
4. Convert the variables to the appropriate type (e.g., categorical variables to `factor`).
5. Run the `summary` function on your data frame. Take note of `NAs`, impossible values that are data entry errors, etc. Perhaps perform some additional cleaning based on this information.
6. Compute any additional numeric summaries of the different variables, as desired.
7. Create univariate plots of all variables you are considering. Use histograms for discrete `numeric` variables, density plots for continuous `numeric` variables, and bar plots for `factor` variables. Take note of any interesting patterns such as modality, skewness, overall shape, outliers, etc.
8. Create bivariate plots of any pairs of variables. Use scatter plots for two `numeric` variables. Use parallel boxplots for `numeric` and `factor` variables or perhaps create histogram plots of the `numeric` variable facetted by the `factor` variable, or density plots of the `numeric` variables filled with different colors by the `factor` variable. Once again, notice any patterns.
9. Create multivariate and interactive graphics based on what you learned in the previous steps.

2.6 Final notes on missing or erroneous data

What should you do with your data when observations are missing information or the information is clearly erroneous.

If the data are clearly erroneous, attempt to get the correct value. If the values cannot be corrected, replace them with `NA` since you don't have that information.

What should you do about `NA`s. There are many approaches for dealing with `NA`s. The proper approach depends a lot on WHY the data are missing. Speaking informally, if there is not systematic reason why the data are missing, then ignoring the observations with missing data isn't a terrible approach. However, if there is a systematic reason why the data are missing (such as individuals not wanting to answer a sensitive question, subjects dying for a specific reason) then ignoring that data can lead to erroneous conclusions.

In what follows, we will generally ignore missing data, assuming the problem is not systematic.

Chapter 3

Linear model estimation

3.1 A simple motivating example

Suppose you observe data related to the heights of 5 mothers and their adult daughters. The observed heights (measured in inches) are provided in Table 3.1. Figure 3.1 displays a scatter plot of the height data provided in Table 3.1. Would it be reasonable to use a mother's height to predict the height of her adult daughter?

A **regression analysis** is the process of building a model describing the typical relationship between a set of observed variables. In the present context, we want to model the height of adult daughters using their height of their mothers. The model we build is known as a **regression model**.

The variables in a regression analysis may be divided into two types: the response variable and the predictor variables.

The outcome variable we are trying to predict is known as the **response variable**. Response variables are also known as **outcome**, **output**, or **dependent** variables. The response variable is denoted by Y.

Table 3.1: Heights of mothers and their adult daughters (in).

observation	mother's height (in)	daughter's height (in)
1	57.5	61.5
2	60.5	63.5
3	63.5	63.5
4	66.5	66.5
5	69.5	66.5

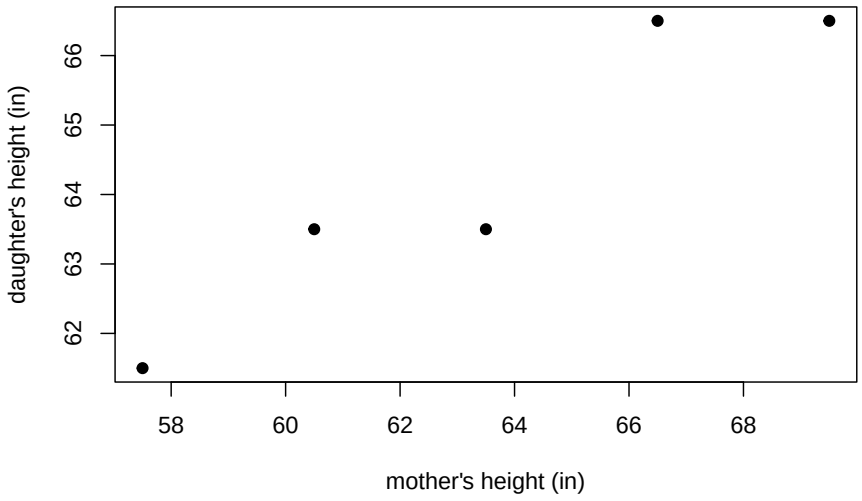


Figure 3.1: A scatter plot displaying pairs of heights for a mother and her adult daughter.

The variables available to model the response variable are known as **predictors variables**. Predictor variables are also known as **explanatory**, **regressor**, **input**, **dependent** variables, or simply as **features**. Following the convention of Weisberg (2014), we use the term **regressor** to refer to the variables used in our regression model, whether that is the original predictor variable, some transformation of a predictor, some combination of predictors, etc. Thus, every predictor can be a regressor but not all regressors are a predictor. The regressor variables are denoted as $X_1, X_2, \ldots, X_{p-1}$.

A regression analysis assumes that we have observed variables $X_1, X_2, \ldots, X_{p-1}$, and Y for each of n subjects from some population, with $x_{i,j}$ denoting the value of X_j for observation i and Y_i denoting the value of Y for observation i. If there is only a single regressor, we can denote the single regressor as X and the observed values of X as $x_1, x_2, \ldots, x_n$. For the height data, the 5 pairs of observed data are denoted

$$(x_1, Y_1), (x_2, Y_2), \ldots, (x_5, Y_5),$$

with (x_i, Y_i) denoting the data for observation i. x_i denotes the mother's height for observation i and Y_i denotes the daughter's height for observation i. Referring to Table 3.1, we see that, e.g., $x_3 = 63.5$ and $Y_5 = 66.5$.

Suppose we want to find the straight line that best fits the plot of mother and

daughter heights in Figure 3.1. How do we determine the "best fitting" model? Consider Figure 3.2, in which 2 potential "best fitting" lines are drawn on the scatter plot of the height data. Which one is best?

The rest of this chapter focuses on defining and estimating the parameters of a *linear* regression model. We will start with the simplest type of linear regression, called simple linear regression, which only uses a single regressor variable to model the response. We will then start looking at more complicated linear regression models. After that, we discuss how to evaluate how well an estimated regression model fits the data. We conclude with a summary of some important concepts from the chapter.

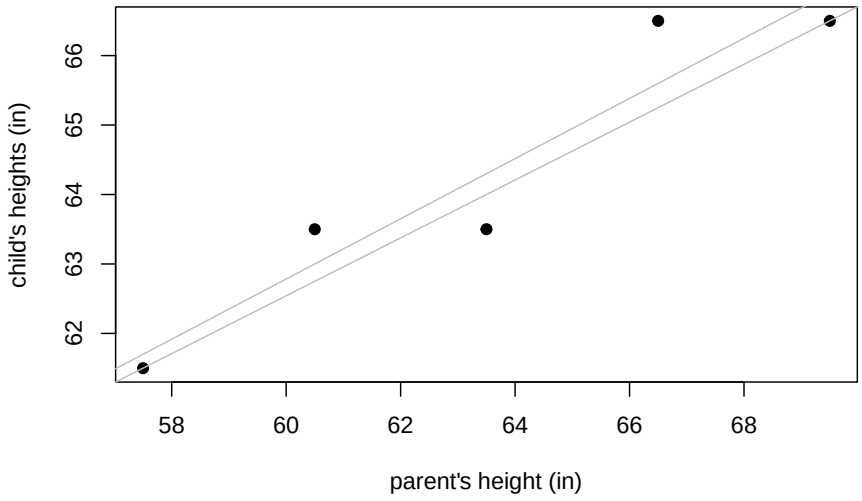


Figure 3.2: Comparison of two potential fitted models to some observed data. The fitted models are shown in grey.

3.2 Estimation of the simple linear regression model

Parameter estimation is the process of using observed data to estimate values for the regression coefficients. There are many different methods of parameter estimation in statistics: method-of-moments, maximum likelihood, Bayesian, etc. The most common parameter estimation method for linear models is the **least squares method**, which is commonly called **Ordinary Least Squares (OLS)** estimation. OLS estimation estimates the regression coefficients with the values

that minimize the residual sum of squares (RSS), which we will define shortly.

3.2.1 Model definition, fitted values, residuals, and RSS

The regression model for Y as a function of X, denoted $E(Y \mid X)$, is the expected value of Y conditional on the regressor X. Thus, a regression model specifically refers to the expected relationship between the response and regressors.

The **simple linear regression model** for a response variable assumes the mean of Y conditional on a single regressor X is

$$E(Y \mid X) = \beta_0 + \beta_1 X.$$

The response variable Y is modeled as

$$\begin{aligned} Y &= E(Y \mid X) + \epsilon \\ &= \beta_0 + \beta_1 X + \epsilon, \end{aligned}$$

where ϵ is known as the model error.

The error term ϵ is literally the deviation of the response variable from its mean. It is standard to assume that conditional on the regressor variable, the error term has mean 0 and variance σ^2, which can be written as

$$E(\epsilon \mid X) = 0$$

and

$$\text{var}(\epsilon \mid X) = \sigma^2.$$

Using the response values $Y_1, \ldots, Y_n$ and their associated regressor values $x_1, \ldots, x_n$, the observed data are modeled as

$$\begin{aligned} Y_i &= \beta_0 + \beta_1 x_i + \epsilon \\ &= E(Y \mid X = x_i) + \epsilon_i, \end{aligned}$$

for $i = 1, 2, \ldots, n$, where ϵ_i denotes the error for observation i.

The **estimated regression model** is defined as

$$\hat{E}(Y|X) = \hat{\beta}_0 + \hat{\beta}_1 X,$$

where $\hat{\beta}_j$ denotes the estimated value of β_j for $j = 0, 1$.

The ith **fitted value** is defined as

$$\hat{Y}_i = \hat{E}(Y|X = x_i) = \hat{\beta}_0 + \hat{\beta}_1 x_i.$$

Thus, the ith fitted value is the estimated mean of Y when the regressor $X = x_i$. More specifically, the ith fitted value is the estimated mean response based on the regressor value observed for the ith observation.

The ith **residual** is defined as

$$\hat{\epsilon}_i = Y_i - \hat{Y}_i.$$

The ith residual is the difference between the response and estimated mean response of observation i.

The **residual sum of squares (RSS)** of a regression model is the sum of its squared residuals. The RSS for a simple linear regression model, as a function of the estimated regression coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$, is defined as

$$RSS(\hat{\beta}_0, \hat{\beta}_1) = \sum_{i=1}^{n} \hat{\epsilon}_i^2.$$

Using the various objects defined above, there are many equivalent expressions for the RSS. Notably, Equation (3.2.1) can be rewritten using Equations (3.2.1) and (3.2.1) as

$$\begin{aligned} RSS(\hat{\beta}_0, \hat{\beta}_1) &= \sum_{i=1}^{n} \hat{\epsilon}_i^2 \\ &= \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \\ &= \sum_{i=1}^{n} (Y_i - \hat{E}(Y|X = x_i))^2 \\ &= \sum_{i=1}^{n} (Y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2. \end{aligned}$$

The **fitted model** is the estimated model that minimizes the RSS and is written as

$$\hat{Y} = \hat{E}(Y|X) = \hat{\beta}_0 + \hat{\beta}_1 X.$$

Both $\hat{Y}$ and $\hat{E}(Y|X)$ are used to denote a fitted model. $\hat{Y}$ is used for brevity while $\hat{E}(Y|X)$ is used for clarity. In a simple linear regression context, the fitted model is known as the **line of best fit**.

In Figure 3.3, we visualize the response values, fitted values, residuals, and fitted model in a simple linear regression context. Note that:

- The fitted model is shown as the dashed grey line and minimizes the RSS.
- The response values, shown as black dots, are the observed values of Y.
- The fitted values, shown as blue x's, are the values returned by evaluating the fitted model at the observed regressor values.
- The residuals, shown as solid orange lines, indicate the distance and direction between the observed responses and their corresponding fitted value. If the response is larger than the fitted value then the residual is positive, otherwise it is negative.
- The RSS is the sum of the squared vertical distances between the response and fitted values.

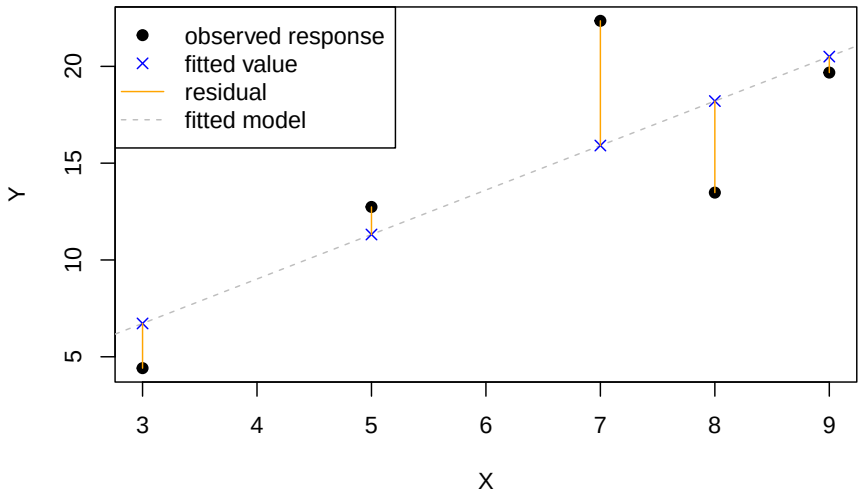


Figure 3.3: Visualization of the fitted model, response values, fitted values, and residuals.

3.2.2 OLS estimators of the simple linear regression parameters

The estimators of β_0 and β_1 that minimize the RSS for a simple linear regression model can be obtained analytically using basic calculus under minimal assumptions. Specifically, the optimal analytical solutions for $\hat{\beta}_0$ and $\hat{\beta}_1$ are valid as long as the regressor values are not a constant value, i.e, $x_i \neq x_j$ for at least some $i, j \in \{1, 2, \ldots, n\}$.

Define $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$. The expression $\bar{x}$ is read "x bar", and it is the sample mean of the observed x_i values. The OLS estimators of the simple linear regression coefficients that minimize the RSS are

$$\begin{aligned}
\hat{\beta}_1 &= \frac{\sum_{i=1}^{n} x_i Y_i - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} Y_i\right)}{\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2} \\
&= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\
&= \frac{\sum_{i=1}^{n}(x_i - \bar{x})Y_i}{\sum_{i=1}^{n}(x_i - \bar{x})x_i}
\end{aligned}$$

and

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{x}.$$

The various expressions given in Equation (3.2.2) are equivalent. In fact, in Equation (3.2.2), all of the numerators are equivalent, and all of the denominators are equivalent. We provide derivations of the estimators for $\hat{\beta}_0$ and $\hat{\beta}_1$ in Section 3.12.2.

In addition to the regression coefficients, the other parameter we mentioned in Section 3.2.1 is the error variance, σ^2. The most common estimator of the error variance is

$$\hat{\sigma}^2 = \frac{RSS}{\mathrm{df}_R SS}.$$

where df_{RSS} is the **degrees of freedom** of the RSS. In a simple linear regression context, the denominator of Equation (3.2.2) is $n - 2$. See Section 3.12.1 for more comments about degrees of freedom.

3.3 Penguins simple linear regression example

We will use the `penguins` data set in the **palmerpenguins** package (Horst, Hill, and Gorman 2022) to illustrate a very basic simple linear regression analysis.

The `penguins` data set provides data related to various penguin species measured in the Palmer Archipelago (Antarctica), originally provided by Gorman, Williams, and Fraser (2014). We start by loading the data into memory.

```
data(penguins, package = "palmerpenguins")
```

The data set includes 344 observations of 8 variables. The variables are:

- `species`: a `factor` indicating the penguin species.
- `island`: a `factor` indicating the island the penguin was observed.
- `bill_length_mm`: a `numeric` variable indicating the bill length in millimeters.
- `bill_depth_mm`: a `numeric` variable indicating the bill depth in millimeters.
- `flipper_length_mm`: an `integer` variable indicating the flipper length in millimeters
- `body_mass_g`: an `integer` variable indicating the body mass in grams.
- `sex`: a `factor` indicating the penguin sex (`female`, `male`).
- `year`: an integer denoting the study year the penguin was observed (`2007`, `2008`, or `2009`).

We begin by creating a scatter plot of `bill_length_mm` versus `body_mass_g` (y-axis versus x-axis) in Figure 3.4.

```
plot(bill_length_mm ~ body_mass_g, data = penguins,
     ylab = "bill length (mm)", xlab = "body mass (g)",
     main = "Penguin size measurements")
```

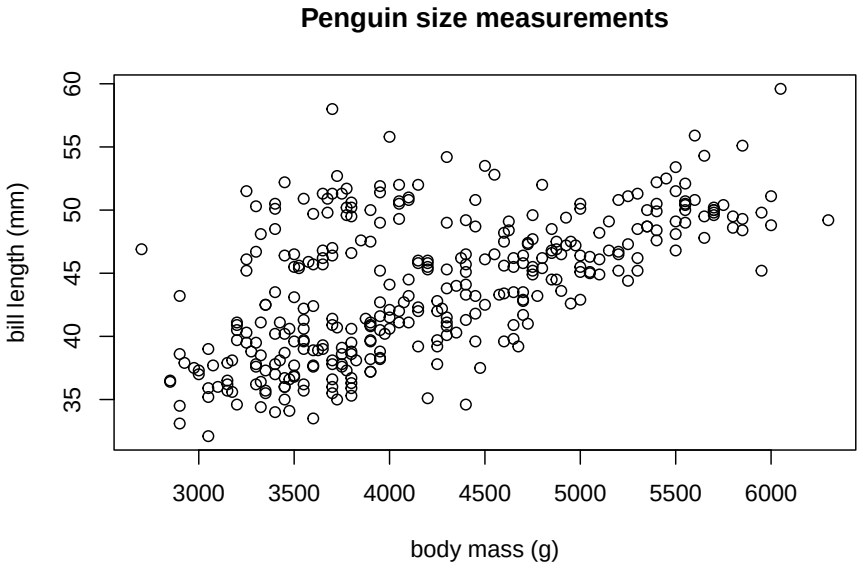


Figure 3.4: A scatter plot of penguin bill length (mm) versus body mass (g)

We see a clear positive association between body mass and bill length: as the body mass increases, the bill length tends to increase. The pattern is linear, i.e., roughly a straight line.

We will build a simple linear regression model that regresses `bill_length_mm` on `body_mass_g`. More specifically, we want to estimate the parameters of the regression model $E(Y \mid X) = \beta_0 + \beta_1 X$, with $Y =$ `bill_length_mm` and $X =$ `body_mass_g`, i.e., we want to estimate the parameters of the model

$$E(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}) = \beta_0 + \beta_1 \, \texttt{body\_mass\_g}.$$

The `lm` function uses OLS estimation to fit a linear model to data. The function has two main arguments:

- `data`: the data frame in which the model variables are stored. This can be omitted if the variables are already stored in memory.
- `formula`: a Wilkinson and Rogers (1973) style formula describing the linear regression model. For complete details, run `?stats:formula` in the Console. If `y` is the response variable and `x` is an available numeric predictor, then `formula = y ~ x` tells `lm` to fit the simple linear regression model $E(Y|X) = \beta_0 + \beta_1 X$.

We use the code below to fit a linear model regressing `bill_length_mm` on `body_mass_g` using the `penguins` data frame and assign the result the name

`lmod`. `lmod` is an object of class `lm`.

```
lmod <- lm(bill_length_mm ~ body_mass_g, data = penguins) # fit model
class(lmod) # class of lmod
## [1] "lm"
```

The `summary` function is commonly used to summarize the results of our fitted model. When an `lm` object is supplied to the `summary` function, it returns:

- `Call`: the function call used to fit the model.
- `Residuals`: A 5-number summary of the $\hat{\epsilon}_1, \ldots, \hat{\epsilon}_n$.
- `Coefficients`: A table that lists:
  - The regressors in the fitted model.
  - `Estimate`: the estimated coefficient for each regressor.
  - `Std. Error`: the *estimated* standard error of the estimated coefficients.
  - `t value`: the computed test statistic associated with testing H_0 : $\beta_j = 0$ versus H_a : $\beta_j \neq 0$ for each regression coefficient in the model.
  - `Pr(>|t|)`: the associated p-value of each test.
- Various summary statistics:
  - `Residual standard error` is the value of $\hat{\sigma}$, the estimate of the error standard deviation. The degrees of freedom is df_{RSS}, the number of observations minus the number of estimated coefficients in the model.
  - `Multiple R-squared` is an estimate of model fit discussed in Section 3.10.
  - `Adjusted R-squared` is a modified version of `Multiple R-squared`.
  - `F-statistic` is the test statistic for the test that compares the model with an only an intercept to the fitted model. The `DF` (degrees of freedom) values relate to the statistic under the null hypothesis, and the `p-value` is the p-value for the test.

We use the `summary` function on `lmod` to produce the output below.

```
# summarize results stored in lmod
summary(lmod)
##
## Call:
## lm(formula = bill_length_mm ~ body_mass_g, data = penguins)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.1251  -3.0434  -0.8089   2.0711  16.1109
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.690e+01  1.269e+00   21.19   <2e-16 ***
## body_mass_g 4.051e-03  2.967e-04   13.65   <2e-16 ***
## ---
```

```
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.394 on 340 degrees of freedom
##   (2 observations deleted due to missingness)
## Multiple R-squared:  0.3542, Adjusted R-squared:  0.3523
## F-statistic: 186.4 on 1 and 340 DF,  p-value: < 2.2e-16
```

Using the output above, we see that the estimated parameters are $\hat{\beta}_0 = 26.9$ and $\hat{\beta}_1 = 0.004$. Thus, our fitted model is

$$\widehat{\texttt{bill\_length\_mm}} = 26.9 + 0.004\,\texttt{body\_mass\_g}.$$

In the context of a simple linear regression model, the intercept term is the expected response when the value of the regressor is zero, while the slope is the expected change in the response when the regressor increases by 1 unit. Thus, based on the model we fit to the `penguin` data, we can make the following interpretations:

- $\hat{\beta}_1$: If a penguin has a body mass 1 gram larger than another penguin, we expect the larger penguin's bill length to be 0.004 millimeters longer.
- $\hat{\beta}_0$: A penguin with a body mass of 0 grams is expected to have a bill length of 26.9 millimeters.

The latter interpretation is nonsensical. It doesn't make sense to discuss a penguin with a body mass of 0 grams unless we are talking about an embryo, in which case it doesn't even make sense to discuss bill length. This is caused by the fact that we are extrapolating far outside the observed body mass values. Our data only includes information for adult penguins, so we should be cautious about drawing conclusions for penguins at other life stages.

The `abline` function can be used to automatically overlay the fitted model on the observed data. We run the code below to produce Figure 3.5. The fit of the model to our observed data seems reasonable.

```
plot(bill_length_mm ~ body_mass_g, data = penguins, main = "Penguin size measurements"
     ylab = "bill length (mm)", xlab = "body mass (g)")
# draw fitted line of plot
abline(lmod)
```

R provides many additional methods (generic functions that do something specific when applied to a certain type of object) for `lm` objects. Commonly used ones include:

- `residuals`: extracts the residuals, $\hat{\epsilon}_1, \ldots, \hat{\epsilon}_n$ from an `lm` object.
- `fitted`: extracts the fitted values, $\hat{Y}_1, \ldots, \hat{Y}_n$ from an `lm` object.
- `predict`: by default, computes $\hat{Y}_1, \ldots, \hat{Y}_n$ for an `lm` object. It can also be used to make arbitrary predictions for the `lm` object.

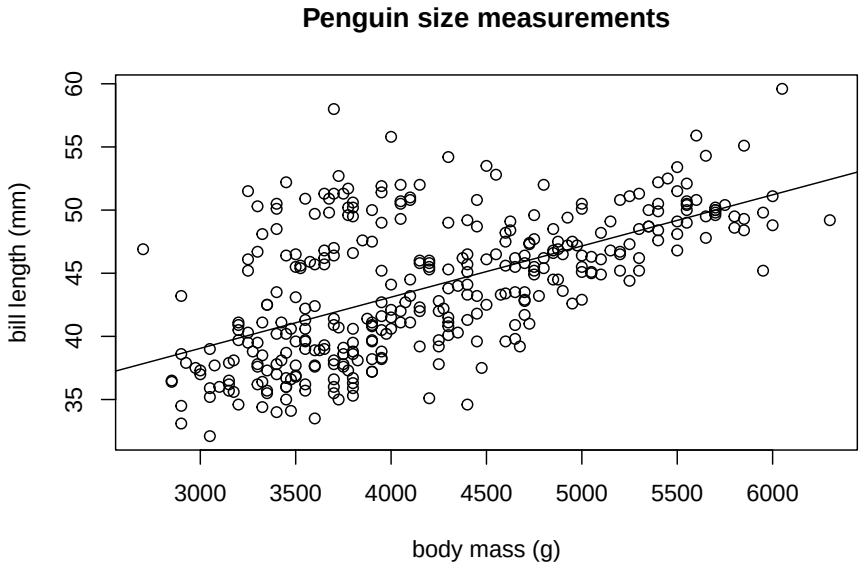


Figure 3.5: The fitted model overlaid on the penguin data.

- `coef` or `coefficients`: extracts the estimated coefficients from an `lm` object.
- `deviance`: extracts the RSS from an `lm` object.
- `df.residual`: extracts df_{RSS}, the degrees of freedom for the RSS, from an `lm` object.
- `sigma`: extracts $\hat{\sigma}$ from an `lm` object.

We now use some of the methods to extract important characteristics of our fitted model.

```
(coeffs <- coef(lmod)) # extract, assign, and print coefficients
##  (Intercept)  body_mass_g
## 26.898872424  0.004051417
ehat <- residuals(lmod) # extract and assign residuals
head(ehat) # first few residuals
##          1          2          3          5          6
## -2.9916846 -2.7942554  0.2340237 -4.1762596 -2.3865430
##          7
## -2.6852575
yhat <- fitted(lmod) # extract and assign fitted values
head(yhat) # first few fitted values
##        1        2        3        5        6        7
## 42.09168 42.29426 40.06598 40.87626 41.68654 41.58526
```

```
yhat2 <- predict(lmod) # compute and assign fitted values
head(yhat2) # first few fitted values
##        1        2        3        5        6        7
## 42.09168 42.29426 40.06598 40.87626 41.68654 41.58526
(rss <- deviance(lmod)) # extract, assign, and print rss
## [1] 6564.494
(dfr <- df.residual(lmod)) # extract n - p
## [1] 340
(sigmasqhat <- sigma(lmod)^2) # estimated error variance
## [1] 19.30734
```

From the output above, we that the the first 3 residuals are -2.99, -2.79, and 0.23. The first 3 fitted values are 42.09, 42.29, and 40.07. The RSS for the fitted model is 6564.49 with 340 degrees of freedom. The estimated error variance, $\hat{\sigma}^2$, is 19.31.

We use the `methods` function to obtain a full list of methods available for `lm` objects using the code below.

```
methods(class = "lm")
##  [1] add1           alias          anova
##  [4] case.names     coerce         confint
##  [7] cooks.distance deviance       dfbeta
## [10] dfbetas        drop1          dummy.coef
## [13] effects        extractAIC     family
## [16] formula        fortify        hatvalues
## [19] influence      initialize     kappa
## [22] labels         logLik         model.frame
## [25] model.matrix   nobs           plot
## [28] predict        print          proj
## [31] qr             residuals      rstandard
## [34] rstudent       show           simulate
## [37] slotsFromS3    summary        variable.names
## [40] vcov
## see '?methods' for accessing help and source code
```

3.4 Defining a linear model

3.4.1 Necessary components and notation

We now wish to discuss linear models in a broader context. We begin by defining notation for the components of a linear model and provide some of their important properties. We repeat some of the previous discussion for clarity.

- Y denotes the response variable.
  - The response variable is treated as a random variable.

  - We will observe realizations of this random variable for each observation in our data set.
- X denotes a single regressor variable. $X_1, X_2, \ldots, X_{p-1}$ denote distinct regressor variables if we are performing regression with multiple regressor variables.
  - The regressor variables are treated as non-random variables.
  - The observed values of the regressor variables are treated as fixed, known values.
- $\mathbb{X} = \{X_0, X_1, \ldots, X_{p-1}\}$ denotes the collection of all regressors under consideration, though this notation is really only needed in the context of multiple regression. X_0 is usually the constant regressor 1, which is needed to include an intercept in the regression model.
- β_0, β_1, ..., β_{p-1} denote **regression coefficients**.
  - Regression coefficients are statistical parameters that we will estimate from our data.
  - The regression coefficients are treated as fixed, non-random but unknown values.
  - Regression coefficients are not observable.
- ϵ denotes model **error**.
  - The model error is more accurately described as random variation of each observation from the regression model.
  - The error is treated as a random variable.
  - The error is assumed to have mean 0 for all values of the regressors. We write this as $E(\epsilon \mid \mathbb{X}) = 0$, which is read as, "The expected value of ϵ conditional on knowing all the regressor values equals 0". The notation "$\mid \mathbb{X}$" extends the notation used in Equation (4.1.1) to multiple regressors.
  - The variance of the errors is assumed to be a constant value for all values of the regressors. We write this assumption as $\text{var}(\epsilon \mid \mathbb{X}) = \sigma^2$.
  - The error is never observable (except in the context of a simulation study where the experimenter literally defines the true model).

3.4.2 Standard definition of linear model

In general, a linear regression model can have an arbitrary number of regressors. A **multiple linear regression** model has two or more regressors.

A **linear model** for Y is defined by the equation

$$\begin{aligned} Y &= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_{p-1} X_{p-1} + \epsilon \\ &= E(Y \mid \mathbb{X}) + \epsilon. \end{aligned}$$

We write the linear model in this way to emphasize the fact *the response value equals the expected response for that combination of regressor values plus some error*. It should be clear from comparing Equation (3.4.2) with the previous line that

$$E(Y \mid \mathbb{X}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_{p-1} X_{p-1},$$

which we prove in Chapter 5.

More generally, one can say that a regression model is linear if the mean function can be written as a linear combination of the regression coefficients and known values created from our regressor variables, i.e.,

$$E(Y \mid \mathbb{X}) = \sum_{j=0}^{p-1} c_j \beta_j,$$

where $c_0, c_1, \ldots, c_{p-1}$ are known functions of the regressor variables, e.g., $c_1 = X_1 X_2 X_3$, $c_3 = X_2^2$, $c_8 = \ln(X_1)/X_2^2$, etc. Thus, if $g_0, \ldots, g_{p-1}$ are functions of $\mathbb{X}$, then we can say that the regression model is linear if it can be written as

$$E(Y \mid \mathbb{X}) = \sum_{j=0}^{p-1} g_j(\mathbb{X}) \beta_j.$$

A model is linear because of its *form*, not the shape it produces. Many of the linear model examples given below do not result in a straight line or surface, but are curved. Some examples of linear regression models are:

- $E(Y|X) = \beta_0$.
- $E(Y|X) = \beta_0 + +\beta_1 X + \beta_2 X^2$.
- $E(Y|X_1, X_2) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$.
- $E(Y|X_1, X_2) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2$.
- $E(Y|X_1, X_2) = \beta_0 + \beta_1 \ln(X_1) + \beta_2 X_2^{-1}$.
- $E(\ln(Y)|X_1, X_2) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$.
- $E(Y^{-1}|X_1, X_2) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$.

Some examples of non-linear regression models are:

- $E(Y|X) = \beta_0 + e^{\beta_1 X}$.
- $E(Y|X) = \beta_0 + \beta_1 X/(\beta_2 + X)$.

The latter regression models are non-linear models because there is no way to express them using the expression in Equation (3.4.2).

3.5 Estimation of the multiple linear regression model

Suppose we want to estimate the parameters of a linear model with 1 or more regressors, i.e., when we use the model

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_{p-1} X_{p-1} + \epsilon.$$

The system of equations relating the responses, the regressors, and the errors for all n observations can be written as

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_{p-1} x_{i,p-1} + \epsilon_i, \quad i = 1, 2, \ldots, n.$$

3.5.1 Using matrix notation to represent a linear model

We can simplify the linear model described in Equation (3.5) using matrix notation. It may be useful to refer to Appendix A for a brief overview of matrix-related information.

We use the following notation:

- $\mathbf{y} = [Y_1, Y_2, \ldots, Y_n]$ denotes the column vector containing the n observed response values.
- $\mathbf{X}$ denotes the matrix containing a column of 1s and the observed regressor values for $X_1, X_2, \ldots, X_{p-1}$. This may be written as

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,p-1} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,p-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,p-1} \end{bmatrix}.$$

- $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_{p-1}]$ denotes the column vector containing the p regression coefficients.
- $\boldsymbol{\epsilon} = [\epsilon_1, \epsilon_2, \ldots, \epsilon_n]$ denotes the column vector contained the n errors.

The system of equations defining the linear model in Equation (3.5) can be written as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}.$$

Thus, matrix notation can be used to represent a system of linear equations. A model that cannot be represented as a system of linear equations using matrices is not a linear model.

3.5.2 Residuals, fitted values, and RSS for multiple linear regression

We now discuss of residuals, fitted values, and RSS for the multiple linear regression context using matrix notation.

The vector of estimated values for the coefficients contained in $\boldsymbol{\beta}$ is denoted

$$\hat{\boldsymbol{\beta}} = [\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_{p-1}].$$

The vector of regressor values for the ith observation is denoted

$$\mathbf{x}_i = [1, x_{i,1}, \ldots, x_{i,p-1}],$$

where the 1 is needed to account for the intercept in our model.

Extending the original definition of a fitted value in Equation (3.2.1), the ith

fitted value in the context of multiple linear regression is defined as

$$\begin{aligned}\hat{Y}_i &= \hat{E}(Y \mid \mathbb{X} = \mathbf{x}_i) \\ &= \hat{\beta}_0 + \hat{\beta}_1 x_{i,1} + \cdots + \hat{\beta}_{p-1} x_{i,p-1} \\ &= \mathbf{x}_i^T \hat{\boldsymbol{\beta}}.\end{aligned}$$

The notation "$\mathbb{X} = \mathbf{x}_i$" is a concise way of saying "$X_0 = 1, X_1 = x_{i,1}, \ldots, X_{p-1} = x_{i,p-1}$".

The column vector of fitted values is defined as

$$\hat{\mathbf{y}} = [\hat{Y}_1, \ldots, \hat{Y}_n],$$

and can be computed as

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}.$$

Extending the original definition of a residual in Equation (3.2.1), the ith **residual** in the context of multiple linear regression can be written as

$$\hat{\epsilon}_i = Y_i - \hat{Y}_i = Y_i - \mathbf{x}_i^T \hat{\boldsymbol{\beta}},$$

using Equation (3.5.2).

The column vector of residuals is defined as

$$\hat{\boldsymbol{\epsilon}} = [\hat{\epsilon}_1, \ldots, \hat{\epsilon}_n].$$

Using Equations (3.5.2) and (3.5.2), equivalent expressions for the residual vector are

$$\hat{\boldsymbol{\epsilon}} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}.$$

The RSS for a multiple linear regression model, as a function of the estimated regression coefficients, is

$$\begin{aligned}RSS(\hat{\boldsymbol{\beta}}) &= \sum_{i=1}^{n} \hat{\epsilon}_i^2 \\ &= \hat{\boldsymbol{\epsilon}}^T \hat{\boldsymbol{\epsilon}} \\ &= (\mathbf{y} - \hat{\mathbf{y}})^T (\mathbf{y} - \hat{\mathbf{y}}) \\ &= (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})^T (\mathbf{y} - \hat{\boldsymbol{\beta}}).\end{aligned}$$

The various expressions in Equation (3.5.2) are equivalent (cf. Equation (3.5.2)).

3.5.3 OLS estimator of the regression coefficients

The OLS estimator of the regression coefficient vector, $\boldsymbol{\beta}$, is

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}.$$

Equation (3.5.3) assumes that $\mathbf{X}$ has full-rank ($n > p$ and none of the columns of $\mathbf{X}$ are linear combinations of other columns in $\mathbf{X}$), which is a very mild assumption. We provide a derivation of the estimator for β in Section 3.12.5.

The general estimator of the σ^2 in the context of multiple linear regression is

$$\hat{\sigma}^2 = \frac{RSS}{n-p},$$

which is consistent with the previous definition given in Equation (3.2.2).

3.6 Penguins multiple linear regression example

We continue our analysis of the `penguins` data introduced in Section 3.3. We will fit a multiple linear regression model regressing `bill_length_mm` on `body_mass_g` and `flipper_length_mm`, and will once again do so using the `lm` function.

Before we do that, we provide some additional discussion of the of the `formula` argument of the `lm` function. This will be very important as we discuss more complicated models. Assume `y` is the response variable and `x`, `x1`, `x2`, `x3` are available numeric predictors. Then: - `y ~ x` describes the simple linear regression model $E(Y|X) = \beta_0 + \beta_1 X$. - `y ~ x1 + x2` describes the multiple linear regression model $E(Y|X_1, X_2) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$. - `y ~ x1 + x2 + x1:x2` and `y ~ x1 * x2` describe the multiple linear regression model $E(Y|X_1, X_2) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2$. - `y ~ -1 + x1 + x2` describe a multiple linear regression model without an intercept, in this case, $E(Y|X_1, X_2) = \beta_1 X_1 + \beta_2 X_2$. The `-1` tells R not to include an intercept in the fitted model. - `y ~ x + I(x^2)` describe the multiple linear regression model $E(Y|X) = \beta_0 + \beta_1 X + \beta_2 X^2$. The `I()` function is a special function that tells R to create a regressor based on the syntax inside the `()` and include that regressor in the model.

In the code below, we fit the linear model regressing `bill_length_mm` on `body_mass_g` and `flipper_length_mm` and then use the `coef` and `deviance` functions to extract the estimated coefficients and RSS of the fitted model, respectively.

```
# fit model
mlmod <- lm(bill_length_mm ~ body_mass_g + flipper_length_mm, data = penguins)
# extract estimated coefficients
coef(mlmod)
##       (Intercept)       body_mass_g flipper_length_mm
##     -3.4366939266      0.0006622186      0.2218654584
# extract RSS
deviance(mlmod)
## [1] 5764.585
```

The fitted model is

$$\widehat{\texttt{bill\_length\_mm}} = -3.44 + 0.0007\,\texttt{body\_mass\_g} + 0.22\,\texttt{flipper\_length\_mm}.$$

We discuss how to interpret the coefficients of a multiple linear regression model in Chapter 4.

The RSS for the fitted model is 5764.59, which is substantially less than the RSS of the fitted simple linear regression model in Section 3.3.

3.7 Types of linear models

We provide a brief overview of different types of linear regression models. Our discussion is not exhaustive, nor are the types exclusive (a model can sometimes be described using more than one of these labels). We have previously discussed some of the terms, but include them for completeness.

- **Simple**: a model with an intercept and a single regressor.
- **Multiple**: a model with 2 or more regressors.
- **Polynomial**: a model with squared, cubic, quartic predictors, etc. E.g, $E(Y \mid X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$ is a 4th-degree polynomial.
- **First-order**: a model in which each predictor is used to create no more than one regressor.
- **Main effect**: a model in which none of the regressors are functions of more than one predictor. A predictor can be used more than once, but each regressor is only a function of one predictor. E.g., if X_1 and X_2 are different predictors, then the regression model $E(Y \mid X_1, X_2) = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_2$ would be a main effect model, but not a first-order model since X_1 was used to create two regressors.
- **Interaction**: a model in which some of the regressors are functions of more than 1 predictor. E.g., if X_1 and X_2 are different predictors, then the regression model $E(Y \mid X_1, X_2) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2$ is a very simple interaction model since the third regressor is the product of X_1 and X_2.
- **Analysis of variance (ANOVA)**: a model for which all predictors used in the model are categorical.
- **Analysis of covariance (ANCOVA)**: a model that uses at least one numeric predictor and at least one categorical predictor.
- **Generalized (GLM)**: a "generalized" linear regression model in which the responses do not come from a normal distribution.

3.8 Categorical predictors

Categorical predictors can greatly improve the explanatory power or predictive capability of a fitted model when different patterns exist for different levels of the variables. We discuss two basic uses of categorical predictors in linear regression models. We will briefly introduce the:

- **parallel lines regression model**, which is a main effect regression model that has a single numeric regressor and a single categorical predictor. The model produces parallel lines for each level of the categorical variable.
- **separate lines regression model**, which adds an interaction term between the numeric regressor and categorical predictor of the parallel lines regression model. The model produces separate lines for each level of the categorical variable.

3.8.1 Indicator variables

In order to compute $\hat{\boldsymbol{\beta}}$ using Equation (3.5.3), both $\mathbf{X}$ and $\mathbf{y}$ must contain numeric values. How can we use a categorical predictor in our regression model when its values are not numeric? To do so, we must transform the categorical predictor into one or more **indicator** or **dummy variables**, which we explain in more detail below.

An **indicator function** is a function that takes the value 1 if a certain property is true and 0 otherwise. An **indicator variable** is the variable that results from applying an indicator function to each observation of a variable. Many notations exist for indicator functions. We use the notation,

$$I_S(x) = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{if } x \notin S \end{cases},$$

which is shorthand for a function that returns 1 if x is in the set S and 0 otherwise.

Let C denote a categorical predictor with levels L_1 and L_2. The C stands for "categorical", while the L stands for "level". Let c_i denote the value of C for observation i.

Let D_j denote the indicator (dummy) variable for factor level L_j of C. The value of D_j for observation i is denoted $d_{i,j}$, with

$$d_{i,j} = I_{\{L_j\}}(c_i),$$

i.e., $d_{i,j}$ is 1 if c_i has factor level L_j and 0 otherwise.

3.8.2 Parallel and separate lines models

Assume we want to build a linear regression model using a single numeric regressor X and a two-level categorical predictor C.

The standard simple linear regression model is

$$E(Y \mid X) = \beta_0 + \beta_1 X.$$

To create a parallel lines regression model, we add regressor D_2 to the simple linear regression model. Thus, the parallel lines regression model is

$$E(Y \mid X, C) = \beta_0 + \beta_1 X + \beta_2 D_2.$$

Since $D_2 = 0$ when $C = L_1$ and $D_2 = 1$ when $C = L_2$, we see that the model in Equation (3.8.2) simplifies to

$$E(Y \mid X, C) = \begin{cases} \beta_0 + \beta_1 X & \text{if } C = L_1 \\ (\beta_0 + \beta_2) + \beta_1 X & \text{if } C = L_2 \end{cases}.$$

The parallel lines will be separated vertically by the distance β_2.

To create a separate lines regression model, we add regressor D_2 and the interaction regressor XD_2 to our simple linear regression model. Thus, the separate lines regression model is

$$E(Y \mid X, C) = \beta_0 + \beta_1 X + \beta_2 D_2 + \beta_3 X D_2,$$

which, similar to the previous model, simplifies to

$$E(Y \mid X, C) = \begin{cases} \beta_0 + \beta_1 X & \text{if } C = L_1 \\ (\beta_0 + \beta_2) + (\beta_1 + \beta_3) X & \text{if } C = L_2 \end{cases}.$$

3.8.3 Extensions

We have presented the most basic regression models that include a categorical predictor. If you had a categorical predictor C with K levels $L_1, L_2, \ldots, L_K$, then you could add indicator variables $D_2, D_3, \ldots, L_K$ to a simple linear regression model to create a parallel lines model for each level of C. Similarly, you could add regressors $D_2, D_3, \ldots, D_K, XD_2, XD_3, \ldots, XD_K$ to a simple linear regression model to create a separate lines model for each level of C.

It is easy to imagine using multiple categorical predictors in a model, interacting one or more categorical predictors with one or more numeric regressors in model, etc. These models can be fit easily using R (as we'll see below), though interpretation becomes more challenging.

3.8.4 Avoiding an easy mistake

Why didn't we add D_1 to the parallel lines model? Or D_1 and $D_1 X$ to the separate lines model? First, we notice that we don't *need* to add them. E.g., if an observation doesn't havel level L_2 ($D_2 = 0$), then it must have level L_1. More importantly, we didn't do this because it will create linear dependencies in the columns of the regressor matrix $\mathbf{X}$.

Let $\mathbf{d}_1 = [d_{1,1}, d_{2,1}, \ldots, d_{n,1}]$ be the column vector of observed values for indicator variable D_1 and $\mathbf{d}_2$ be the column vector for D_2. Then for a two-level categorical variable, $\mathbf{d}_1 + \mathbf{d}_2$ is an $n \times 1$ vector of 1s, meaning that D_1 and D_2 will be linearly dependent with the intercept column of our $\mathbf{X}$ matrix. Thus, adding D_1 to the parallel lines model would result in $\mathbf{X}$ having linearly dependent columns, which creates estimation problems.

For a categorical predictor with K levels, we only need indicator variables for $K-1$ levels of the categorical predictor. The level without an indicator variable in the regression model is known as the **reference level**, which is explained in Chapter 4. Technically, you can choose any level to be your reference level, but R automatically chooses the first level of a categorical (`factor`) variable to be the reference level, so we adopt that convention.

3.9 Penguins example with categorical predictor

We return once again to the `penguins` data previously introduced. We use the code below to produce Figure 3.6, which displays the grouped scatter plot of `bill_length_mm` versus `body_mass_g` that distinguishes the `species` of each observation. It is very clear that the relationship between bill length and body mass changes depending on the species of penguin.

```
library(ggplot2) # load package
# create grouped scatterplot
ggplot(data = penguins) +
  geom_point(aes(x = body_mass_g, y = bill_length_mm, shape = species, color = species)) +
  xlab("body mass (g)") + ylab("bill length (mm)")
```

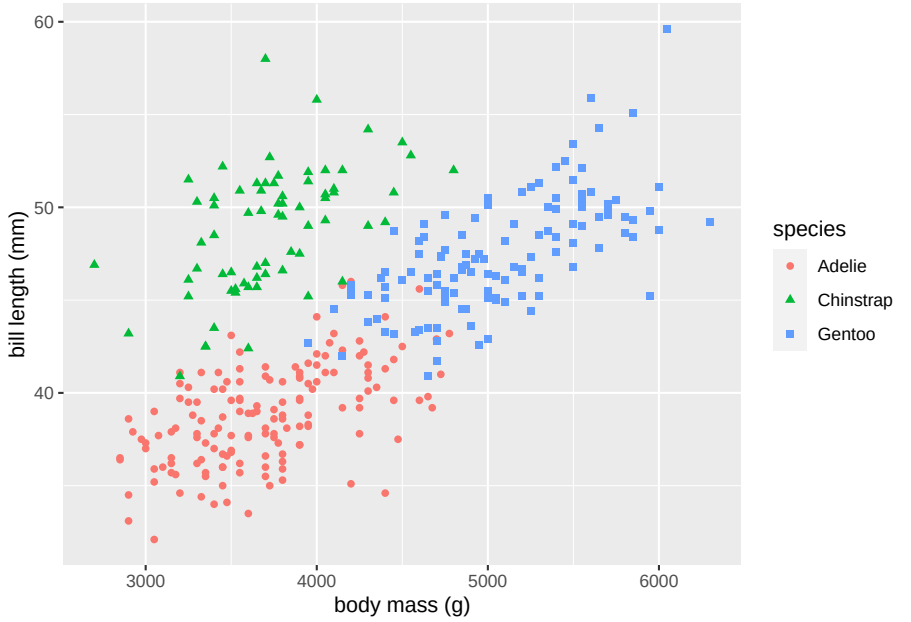


Figure 3.6: A grouped scatter plot of body mass versus bill length that distinguishes penguin species.

How do you use a categorical variable in R's `lm` function? Recall that we

should represent our categorical variables as a `factor` in R. The `lm` function will automatically convert a `factor` variable to the correct number of indicator variables when you include the `factor` variable in your `formula` argument. R will automatically choose the reference level to be the first level of the `factor` variable. To add a main effect term for a categorical predictor, we simply add the term to our `lm` formula. To create an interaction term, we use `:` between the interacting variables. E.g., if `c` is a `factor` variable and `x` is a `numeric` variable, you can use the notation `c:x` in your `formula` to get all the interactions between `c` and `x`.

In our present context, the categorical predictor of interest is `species`, which has the levels `Adelie`, `Chinstrap`, and `Gentoo`. The `species` variable is already a `factor`. Since the variable has 3 levels, it will be transformed into 2 indicator variables by R. The first level of species is `Adelie`, so R will treat that level as the reference level, and automatically create indicator variables for the levels `Chinstrap` and `Gentoo`. (Reminder: to determine the level order of a `factor` variable `c`, run the commend `levels(c)`, or in this case `levels(penguins$species)`.)

Let D_C denote the indicator variable for the `Chinstrap` level and D_G denote the indicator variable for the `Gentoo` level. To fit the parallel lines regression model

$$E(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species}) = \beta_0 + \beta_1 \texttt{body\_mass\_g} + \beta_2 D_C + \beta_3 D_G,$$

we run the code below. The `coef` function is used to extract the estimated coefficients from our fitted model in `lmodp`.

```
# fit parallel lines model
lmodp <- lm(bill_length_mm ~ body_mass_g + species, data = penguins)
# extract coefficients
coef(lmodp)
##      (Intercept)      body_mass_g speciesChinstrap
##     24.919470977      0.003748497      9.920884113
##    speciesGentoo
##      3.557977539
```

Thus, the fitted parallel lines model is

$$\begin{aligned}&\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species})\\ &= 24.92 + 0.004\texttt{body\_mass\_g} + 9.92 D_C + 3.56 D_G.\end{aligned}$$

Note that `speciesChinstrap` and `speciesGentoo` are the indicator variables related to the `Chinstrap` and `Gentoo` levels of `species`, respectively, i.e., they represent D_C and D_G. When an observation has `species` level `Adelie`, then Equation (3.9) simplifies to

$$\begin{aligned}&\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species} = \texttt{Adelie})\\ &= 24.92 + 0.004\texttt{body\_mass\_g} + 9.92 \cdot 0 + 3.56 \cdot 0\\ &= 24.92 + 0.004\texttt{body\_mass\_g}.\end{aligned}$$

When an observation has `species` level `Chinstrap`, then Equation (3.9) simplifies to

$$\begin{aligned}&\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species} = \texttt{Chinstrap})\\&= 24.92 + 0.004\texttt{body\_mass\_g} + 9.92 \cdot 1 + 3.56 \cdot 0\\&= 34.84 + 0.004\texttt{body\_mass\_g}.\end{aligned}$$

Lastly, when an observation has `species` level `Gentoo`, then Equation (3.9) simplifies to

$$\begin{aligned}&\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species} = \texttt{Gentoo})\\&= 24.92 + 0.004\texttt{body\_mass\_g} + 9.92 \cdot 0 + 3.56 \cdot 1\\&= 28.48 + 0.004\texttt{body\_mass\_g}.\end{aligned}$$

Adding fitted lines for each `species` level to the scatter plot in Figure 3.6 is a bit more difficult than before. One technique is to use `predict` to get the fitted values of each observation, use the `transform` function to add those values as a column to the original the data frame, then use `geom_line` to connect the fitted values from each group.

We start by adding our fitted values to the `penguins` data frame. We use the `predict` function to obtained the fitted values of our fitted model and then use the `transform` function to add those values as the `pl_fitted` variable in the `penguins` data frame.

```
penguins <-
  penguins |>
  transform(pl_fitted = predict(lmodp))
## Error in data.frame(structure(list(species = structure(c(1L, 1L, 1L, 1L, : arguments imply di
```

We just received a nasty error. What is going on? The original `penguins` data frame has 344 rows. However, two rows had `NA` observations such that when we used the `lm` function to fit our parallel lines model, those observations were removed prior to fitting. The `predict` function produces fitted values for the observations used in the fitting process, so there are only 342 predicted values. There is a mismatch between the number of rows in `penguins` and the number of values we attempt to add in the new column `pl_fitted`, so we get an error.

To handle this error, we refit our model while setting the `na.action` argument to `na.exclude`. As stated Details section of the documentation for the `lm` function (run `?lm` in the Console):

> ... when `na.exclude` is used the residuals and predictions are padded to the correct length by inserting `NA`s for cases omitted by `na.exclude`.

We refit the parallel lines model below with `na.action = na.exclude`, then use the `predict` function to add the fitted values to the `penguins` data frame via the `transform` function.

```
# refit parallel lines model with new na.action behavior
lmodp <- lm(bill_length_mm ~ body_mass_g + species, data = penguins, na.action = na.exc
# add fitted values to penguins data frame
penguins <-
  penguins |>
  transform(pl_fitted = predict(lmodp))
```

We now use the `geom_line` function to add the fitted lines for each `species` level to our scatter plot. Figure 3.7 displays the results from running the code below. The parallel lines model shown in Figure 3.7 fits the `penguins` data better than the simple linear regression model shown in Figure 3.5.

```
# create plot
# create scatterplot
# customize labels
# add lines for each level of species
ggplot(data = penguins) +
  geom_point(aes(x = body_mass_g, y = bill_length_mm,
                 shape = species, color = species)) +
  xlab("body mass (g)") + ylab("bill length (mm)") +
  geom_line(aes(x = body_mass_g, y = pl_fitted, color = species))
```

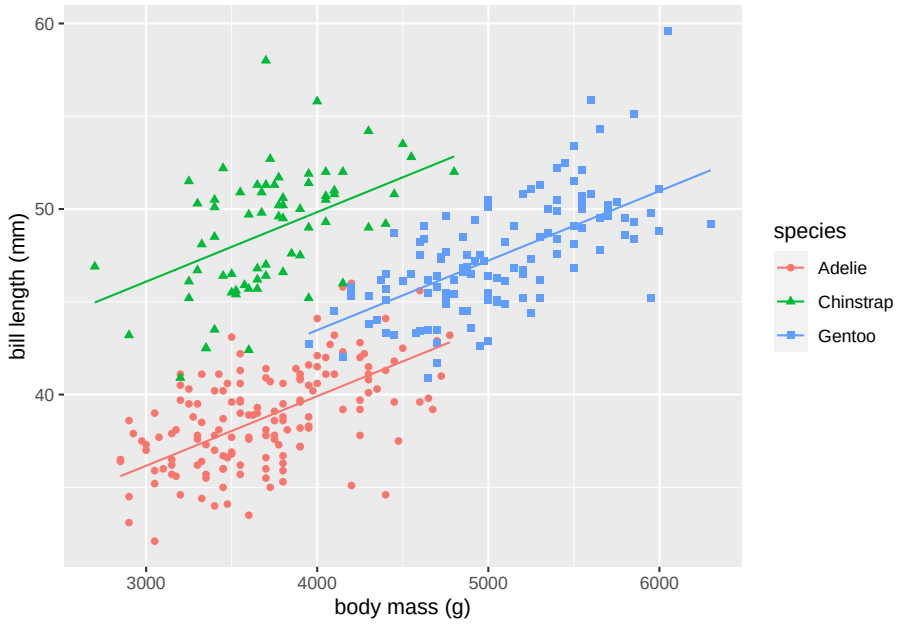


Figure 3.7: The fitted lines from the separate lines model for each level of `species` is added to the grouped scatter plot of `bill_length_mm` versus `body_mass_g`.

We now fit a separate lines regression model to the `penguins` data. Specifically, we fit the model

$$E(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species}) = \beta_0 + \beta_1 \texttt{body\_mass\_g} + \beta_2 D_C + \beta_3 D_G + \beta_4 \texttt{body\_mass\_g} D_C + \beta_5 \texttt{body\_mass\_g} D_G,$$

using the code below, using the `coef` function to extract the estimated coefficients. The terms with `:` are interaction variables, e.g., `body_mass_g:speciesChinstrap` is $\hat{\beta}_4$, the coefficient for the interaction between regressor $\texttt{body\_mass\_g} D_C$.

```
# fit separate lines model
# na.omit = na.exclude used to change predict behavior
lmods <- lm(bill_length_mm ~ body_mass_g + species + body_mass_g:species,
            data = penguins, na.action = na.exclude)
# extract estimated coefficients
coef(lmods)
##                   (Intercept)                   body_mass_g
##                 26.9941391367                 0.0031878758
##              speciesChinstrap                speciesGentoo
##                  5.1800537287                -0.2545906615
## body_mass_g:speciesChinstrap    body_mass_g:speciesGentoo
##                  0.0012748183                 0.0009029956
```

Thus, the fitted separate lines model is

$$\begin{aligned}\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species}) &= 26.99 + 0.003\texttt{body\_mass\_g} + 5.18 D_C - 0.25 D_G \\ &\quad + 0.001 \texttt{body\_mass\_g} D_C + 0.0009 \texttt{body\_mass\_g} D_G,\end{aligned}$$

When an observation has `species` level `Adelie`, then Equation (3.9) simplifies to

$$\begin{aligned}\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species} = \texttt{Adelie}) &= 26.99 + 0.003\texttt{body\_mass\_g} + 5.18 \cdot 0 - 0.25 \cdot 0 \\ &\quad + 0.001 \cdot \texttt{body\_mass\_g} \cdot 0 + 0.0009 \cdot \texttt{body\_mass\_g} \cdot 0 \\ &= 26.99 + 0.003\texttt{body\_mass\_g}.\end{aligned}$$

When an observation has `species` level `Chinstrap`, then Equation (3.9) simplifies to

$$\begin{aligned}\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species} = \texttt{Chinstrap}) &= 26.99 + 0.003\texttt{body\_mass\_g} + 5.18 \cdot 1 - 0.25 \cdot 0 \\ &\quad + 0.001 \cdot \texttt{body\_mass\_g} \cdot 1 + 0.0009 \cdot \texttt{body\_mass\_g} \cdot 0 \\ &= 31.17 + 0.004\texttt{body\_mass\_g}.\end{aligned}$$

When an observation has `species` level `Gentoo`, then Equation (3.9) simplifies

to

$$\begin{aligned}
&\hat{E}(\texttt{bill_length_mm} \mid \texttt{body_mass_g}, \texttt{species} = \texttt{Chinstrap}) \\
&= 26.99 + 0.003\texttt{body_mass_g} + 5.18 \cdot 0 - 0.25 \cdot 1 \\
&\quad + 0.001 \cdot \texttt{body_mass_g} \cdot 0 + 0.0009 \cdot \texttt{body_mass_g} \cdot 1 \\
&= 26.74 + 0.004\texttt{body_mass_g}.
\end{aligned}$$

We use the code below to display the fitted lines for the separate lines model on the `penguins` data. Figure 3.8 shows the results. The fitted lines match the observed data behavior reasonably well.

```
# add separate lines fitted values to penguins data frame
penguins <-
  penguins |>
  transform(sl_fitted = predict(lmods))
# use geom_line to add fitted lines to plot
ggplot(data = penguins) +
  geom_point(aes(x = body_mass_g, y = bill_length_mm, shape = species, color = species)
  xlab("body mass (g)") + ylab("bill length (mm)") +
  geom_line(aes(x = body_mass_g, y = sl_fitted, col = species))
```

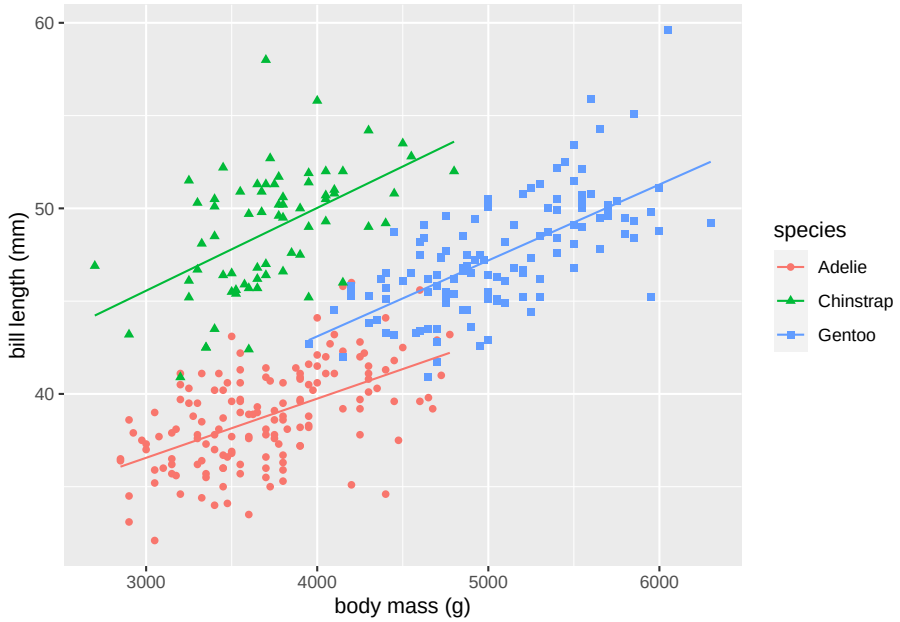


Figure 3.8: The fitted model for each level of `species` is added to the grouped scatter plot of `bill_length_mm` versus `body_mass_g`.

Having fit several models for the `penguins` data, we may be wondering how to evaluate how well the models fit the data. We discuss that in the next section.

3.10 Evaluating model fit

The most basic statistic measuring the fit of a regression model is the **coefficient of determination**, which is defined as

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2},$$

where $\bar{Y}$ is the sample mean of the observed response values.

To interpret this statistic, we need to introduce some new "sum-of-squares" statistics similar to the RSS.

The **total sum of squares** (corrected for the mean) is computed as

$$TSS = \sum_{i=1}^{n}(Y_i - \bar{Y})^2.$$

The TSS is the sum of the squared deviations of the response values from the sample mean. However, it has a more insightful interpretation. Consider the **constant mean model**, which is the model

$$E(Y) = \beta_0.$$

Using basic calculus, you can show that the OLS estimator of β_0 for the model in Equation (3.10) is $\hat{\beta}_0 = \bar{Y}$. For the constant mean model, the fitted value of every observation is $\hat{\beta}_0$, i.e., $\hat{Y}_i = \hat{\beta}_0$ for $i = 1, 2, \ldots, n$. Thus, the RSS of the constant mean model is $\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n}(Y_i - \bar{Y})^2$. Thus, *the TSS is the RSS for the constant mean model.*

The **regression sum-of-squares** or **model sum-of-squares** is defined as

$$SS_{reg} = \sum_{i=1}^{n}(\hat{Y}_i - \bar{Y})^2.$$

Thus, $\mathrm{SS}_{\mathrm{reg}}$ is the sum of the squared deviations between the fitted values of a model and the fitted values of the constant mean model. More helpfully, we have the following equation relating TSS, RSS, and $\mathrm{SS}_{\mathrm{reg}}$:

$$TSS = RSS + SS_{reg}.$$

Thus, $SS_{reg} = TSS - RSS$. This means that *SS_{reg} measures the reduction in RSS when comparing the fitted model to the constant mean model.*

Comparing Equations (3.2.1), (3.10), (3.10), and (3.10), we can express R^2 as:

$$\begin{aligned} R^2 &= 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2} \\ &= 1 - \frac{RSS}{TSS} \\ &= \frac{TSS - RSS}{TSS} \\ &= \frac{SS_{reg}}{TSS} \\ &= [\text{cor}(\mathbf{y}, \hat{\mathbf{y}})]^2. \end{aligned}$$

The last expression is the squared sample correlation between the observed and fitted values, and is a helpful way to express the coefficient of determination because it extends to regression models that are not linear.

Looking at Equation (3.10) in particular, we can say that *the coefficient of determination is the proportional reduction in RSS when comparing the fitted model to the constant mean model.*

Some comments about the coefficient of determination:

- $0 \leq R^2 \leq 1$.
- $R^2 = 0$ for the constant mean model.
- $R^2 = 1$ for a fitted model that perfectly fits the data (the fitted values match the observed response values).
- Generally, larger values of R^2 suggest that the model explains a lot of the variation in the response variable. Smaller R^2 values suggest the fitted model does not explain a lot of the response variation.
- The `Multiple R-squared` value printed by the `summary` of an `lm` object is R^2.
- To extract R^2 from a fitted model, you can use the syntax `summary(lmod)$r.squared`, where `lmod` is your fitted model.

Figure 3.9 provides examples of the R^2 value for various fitted simple linear regression models. The closer the points fall to a straight line, the larger R^2 tends to be. However, as shown in the bottom right panel of Figure 3.9, a poorly fit model can result in a lower R^2 model even if there is a clear relationship between the points (the points have a perfect quadratic relationship).

The coefficient of determination for the parallel lines model fit to the `penguins` data in Section 3.9 is 0.81, as shown in the R output below. By adding the `body_mass_g` regressor and `species` predictor to the constant mean model of `bill_length_mm`, we reduced the RSS by 81%.

```
summary(lmodp)$r.squared
## [1] 0.8079566
```

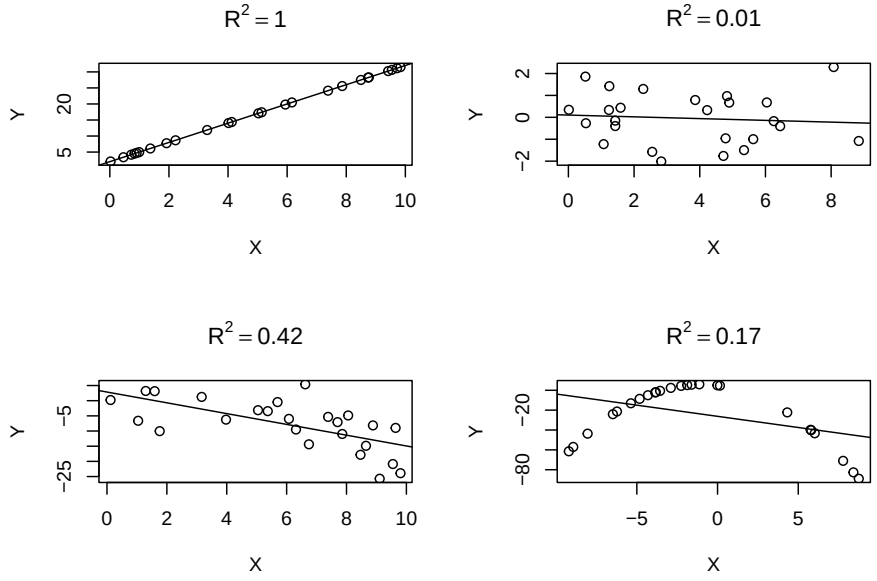


Figure 3.9: The coefficient of determination values for 4 different data sets.

It may seem sensible to choose between models based on the value of R^2. This is unwise for two reasons:

1. R^2 never decreases as regressors are added to an existing model. Basically, you can increase R^2 by simply adding regressors to your existing model, even if they are non-sensical.
2. R^2 doesn't tell you whether a model adequately describes the pattern of the observed data. R^2 is a useful statistic for measuring model fit when there is approximately a linear relationship between the response values and fitted values.

Regarding point 1, consider what happens when we add a regressor of random values to the parallel lines model fit to the `penguins` data. The code below sets a random number seed so that we can get the same results each time we run the code, creates the regressor `noisyx` by sampling 344 values randomly drawn from a $\mathcal{N}(0, 1)$ distribution, adds `noisyx` as a regressor to the parallel lines regression model stored in `lmodp`, and then extracts the R^2 value. We use the `update` method to update our existing model. The `update` function takes an existing model as its first argument and then the `formula` for the updated model. The syntax `. ~ .` means "keep the same response (on the left) and the same regressors (on the right)". We can then add or subtract regressors using the typical `formula` syntax. We use this approach to add the `noisyx` regressor to the regressors already in `lmodp`.

```
set.seed(28) # for reproducibility
# create regressor of random noise
noisyx <- rnorm(344)
# add noisyx as regressor to lmodp
lmod_silly <- update(lmodp, . ~ . + noisyx)
# extract R^2 from fitted model
summary(lmod_silly)$r.squared
## [1] 0.8087789
```

The R^2 value increased from 0.8080 to 0.8088! So clearly, choosing the model with the largest R^2 can be a mistake, as it will tend to favor models with more regressors.

Regarding point 2, R^2 can mislead you into thinking an inappropriate model fits better than it actually does. Anscombe (1973) provided a canonical data set known as "Anscombe's quartet" that illustrates this point. The data set is comprised of 4 different data sets. When a simple linear regression model is fit to each data set, we find that $\hat{\beta}_0 = 3$, $\hat{\beta}_1 = 0.5$, and that $R^2 = 0.67$. However, as we will see, not all models describe the data particularly well!

Anscombe's quartet is available as the `anscombe` data set in the **datasets** package. The data set includes 11 observations of 8 variables. The variables are:

- `x1`, `x2`, `x3` `x4`: the regressor variable for each individual data set.
- `y1`, `y2`, `y3` `y4`: the response variable for each individual data set.

We fit the simple linear regression model to the four data sets in the code below, then extract the coefficients and R^2 to verify the information provided above.

```
# fit model to first data set
lmod_a1 <- lm(y1 ~ x1, data = anscombe)
# extract coefficients from fitted model
coef(lmod_a1)
## (Intercept)          x1
##   3.0000909   0.5000909
# extract R^2 from fitted model
summary(lmod_a1)$r.squared
## [1] 0.6665425
# fit model to second data set
lmod_a2 <- lm(y2 ~ x2, data = anscombe)
coef(lmod_a2)
## (Intercept)          x2
##    3.000909    0.500000
summary(lmod_a2)$r.squared
## [1] 0.666242
# fit model to third data set
lmod_a3 <- lm(y3 ~ x3, data = anscombe)
coef(lmod_a3)
```

```
## (Intercept)          x3
##   3.0024545   0.4997273
summary(lmod_a3)$r.squared
## [1] 0.666324
# fit model to fourth data set
lmod_a4 <- lm(y4 ~ x4, data = anscombe)
coef(lmod_a4)
## (Intercept)          x4
##   3.0017273   0.4999091
summary(lmod_a4)$r.squared
## [1] 0.6667073
```

Figure 3.10 provides a scatter plot each data set and overlays their fitted models.

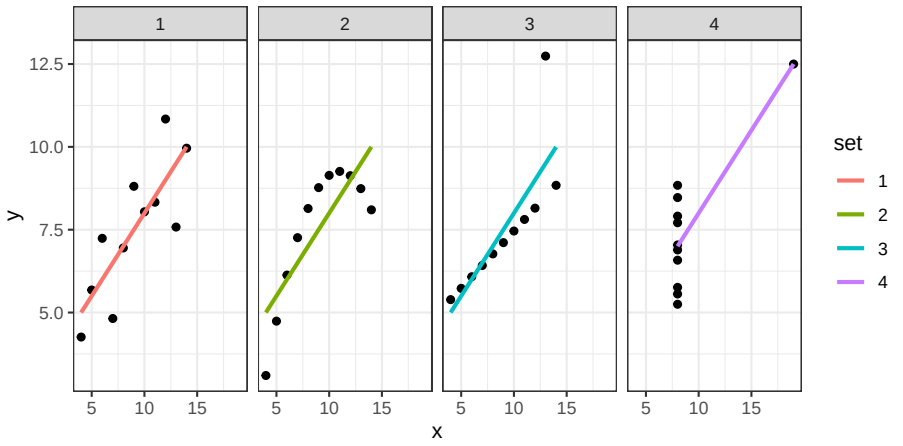


Figure 3.10: Scatter plots of the four Anscombe data sets along with their line of best fit.

While the fitted model and R^2 value is essentially the same for each model, the fitted model is only appropriate for data set 1. The fitted model for the second data set fails to model the curve of the data. The third fitted model doesn't handle the outlier in the data. Lastly, the fourth data set has a single point on the far right side driving the model fit, so the fitted model is highly questionable.

To address the problem with R^2 that it cannot decrease as regressors are added to a model, Ezekiel (1930) proposed the adjusted R-squared statistic for measuring model fit. The adjusted R^2 statistic is defined as

$$R^2_a = 1 - (1 - R^2)\frac{n-1}{n-p} = 1 - \frac{RSS/(n-p)}{TSS/(n-1)}.$$

Practically speaking, R^2_a will only increase when a regressors substantively improves the fit of the model to the observed data. We favor models with larger

values of R_a^2. To extract the adjusted R-squared from a fitted model, we can use the syntax `summary(lmod)$adj.R.squared`, where `lmod` is the fitted model.

Using the code below, we extract the R_a^2 for the 4 models we previously fit to the `penguins` data. Specifically, we extract R_a^2 for the simple linear regression model fit in Section 3.3, the multiple linear regression model in Section 3.6, and the parallel and separate lines models fit in Section 3.9.

```
# simple linear regression model
summary(lmod)$adj.r.squared
## [1] 0.3522562
# multiple linear regression model
summary(mlmod)$adj.r.squared
## [1] 0.4295084
# parallel lines model
summary(lmodp)$adj.r.squared
## [1] 0.8062521
# separate lines model
summary(lmods)$adj.r.squared
## [1] 0.8069556
```

With an R_a^2 of 0.8070, the separate lines regression model appears to be slightly favored over the other 3 models fit to the `penguins` data. To confirm that this statistic is meaningful (i.e., that the model provides a reasonable fit to the data), we use the code below to create a scatter plot of the response versus fitted values shown in Figure 3.11. The points in Figure 3.11 follow a linear pattern, so the separate lines model seems to be a reasonable model for the `penguins` data.

```
plot(penguins$bill_length_mm ~ fitted(lmods),
     xlab = "fitted values", ylab = "bill length (mm)")
```

3.11 Summary

In this chapter, we learned:

- What a linear model is.
- What various objects are, such as coefficients, residuals, fitted values, etc.
- How to estimate the coefficients of a linear model using ordinary least squares estimation.
- How to fit a linear model using R.
- How to include a categorical predictor in a linear model.
- How to evaluate the fit of a model.

3.11.1 Summary of terms

We have introduced many terms to define a linear model. It can be difficult to keep track of their notation, their purpose, whether they are observable, and

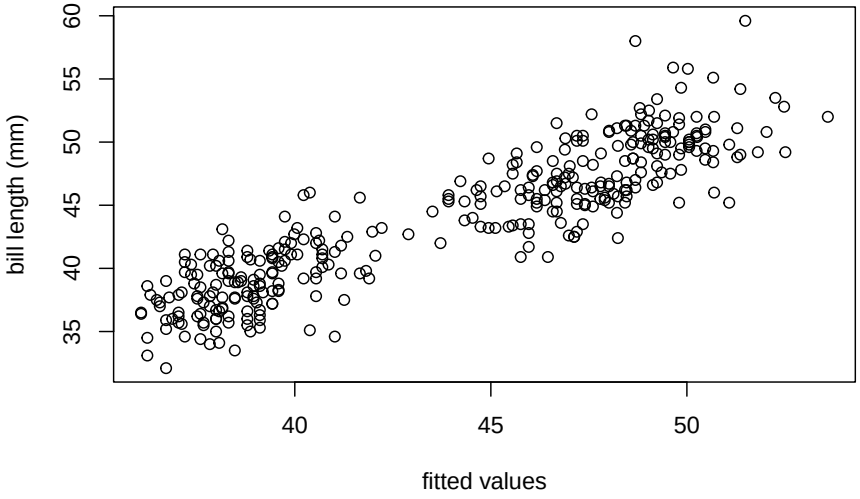


Figure 3.11: A scatter plot of the observed bill length versus the fitted values of the separate lines model for the `penguins` data.

whether they are treated as random variables or vectors. We discuss various terms below, and then summarize the discussion in Table 3.2.

We've already talked about observing the response variable and the predictor/regressor variables. So these objects are observable. However, we have no way to measure the regression coefficients or the error. These are not observable. One way to distinguish observable versus non-observable variables is that observable variables are denoted using Phoenician letters (e.g., X and Y) while non-observable variables are denoted using Greek letters (e.g., β_j, ϵ, σ^2).

We treat the response variable as a random variable. Perhaps surprisingly, we treat the predictor and regressor variables as fixed, non-random variables. The regression coefficients are treated as fixed, non-random but unknown values. This is standard for parameters in a statistical model. The errors are also treated as random variables. In fact, since both the regressor variables and the regression coefficients are non-random, the only way for the responses in Equation (3.5) to be random variables is for the errors to be random.

3.11.2 Summary of functions

We have used many functions in this Chapter. We summarize some of the most important ones in Table 3.3.

Table 3.2: An overview of terms used to define a linear model.

Term	Description	Observable?	Random?
Y	response variable	Yes	Yes
Y_i	response value for the ith observation	Yes	Yes
$\mathbf{y}$	the $n \times 1$ column vector of response values	Yes	Yes
X	regressor variable	Yes	No
X_j	the jth regressor variable	Yes	No
$x_{i,j}$	the value of the jth regressor variable for the ith observation	Yes	No
$\mathbf{X}$	the $n \times p$ matrix of regressor values	Yes	No
$\mathbf{x}_i$	the $p \times 1$ vector of regressor values for the ith observation	Yes	No
β_j	the coefficient associated with the jth regressor variable	No	No
$\boldsymbol{\beta}$	the $p \times 1$ column vector of regression coefficients	No	No
ϵ	the model error	No	Yes
ϵ_i	the error for the ith observation	No	Yes

Table 3.3: An overview of important functions discussed in this chapter.

Function	Purpose
‘lm‘	Fits a linear model based on a provided ‘formula‘
‘summary‘	Provides summary information about the fitted model
‘coef‘	Extracts the vector of estimated regression coefficients from the fitted model
‘residuals‘	Extracts the vector of residuals from the fitted model
‘fitted‘	Extracts the vector of fitted values from the fitted model
‘predict‘	Computes the fitted values (or arbitrary predictions) based on a fitted model
‘deviance‘	Extracts the RSS of a fitted model
‘sigma‘	Extracts $\hat{\sigma}$ from the fitted model
‘update‘	Updates a fitted model to remove or add regressors

3.12 Going Deeper

3.12.1 Degrees of freedom

The degrees of freedom of a statistics refers to the number of independent pieces of information that go into its calculation.

Consider the sample mean

$$\bar{x} = \sum_{i=1}^{n} x_i.$$

The calculation uses n pieces of information to compute, but the statistic only has $n-1$ degrees of freedom. Once you know the sample mean, only $n-1$ values are independent, while the last is constrained to be a certain value.

Let's say $n = 3$ and $\bar{x} = 10$. Then x_1 and x_2 can be any numbers, but the last value MUST equal $30 - x_1 - x_2$ so that $x_1 + x_2 + x_3 = 30$ (otherwise the sample mean won't equal 10). To be more specific, if $x_1 = 5$ and $x_2 = 25$, then x_3 must be 0, otherwise the sample mean won't be 10.

3.12.2 Derivation of the OLS estimators of the simple linear regression model coefficients

Assume a simple linear regression model with n observations. The residual sum of squares for the simple linear regression model is

$$RSS(\hat{\beta}_0, \hat{\beta}_1) = \sum_{i=1}^{n} (Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2.$$

OLS estimator of β_0

First, we take the partial derivative of the RSS with respect to $\hat{\beta}_0$ and simplify:

$$\begin{aligned}
\frac{\partial RSS(\hat{\beta}_0, \hat{\beta}_1)}{\partial \hat{\beta}_0} &= \frac{\partial}{\partial \hat{\beta}_0} \sum_{i=1}^{n} (Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2 && \text{(substituting the formula for the RSS)} \\
&= \sum_{i=1}^{n} \frac{\partial}{\partial \hat{\beta}_0} (Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2 && \text{(by the linearity property of derivatives)} \\
&= -2 \sum_{i=1}^{n} (Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i). && \text{(chain rule, factoring out -2)}
\end{aligned}$$

Next, we set the partial derivative equal to zero and rearrange the terms to solve

for $\hat{\beta}_0$:

$$
\begin{aligned}
0 &= \frac{\partial RSS(\hat{\beta}_0, \hat{\beta}_1)}{\partial \hat{\beta}_0} \\
0 &= -2\sum_{i=1}^{n}(Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) && \text{(substitute partial deriviative)} \\
0 &= \sum_{i=1}^{n}(Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) && \text{(divide both sides by -2)} \\
0 &= \sum_{i=1}^{n} Y_i - \sum_{i=1}^{n}\hat{\beta}_0 - \sum_{i=1}^{n}\hat{\beta}_1 x_i && \text{(by linearity of sum)} \\
0 &= \sum_{i=1}^{n} Y_i - n\hat{\beta}_0 - \sum_{i=1}^{n}\hat{\beta}_1 x_i && \left(\text{summing } \hat{\beta}_0 \text{ } n \text{ times equals } n\hat{\beta}_0\right) \\
n\hat{\beta}_0 &= \sum_{i=1}^{n} Y_i - \sum_{i=1}^{n}\hat{\beta}_1 x_i. && \text{(algebra rearrange)}
\end{aligned}
$$

Finally, we divide both sides by n to get the OLS estimator for $\hat{\beta}_0$ in terms of $\hat{\beta}_1$:

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{x}$$

OLS Estimator of β_1

Similar to the previous derivation, we differentiate the RSS with respect to the parameter estimate of interest, set the derivative equal to zero, and solve for the parameter.

We start by taking the partial derivative of the RSS with respect ot $\hat{\beta}_1$ and simplify.

$$
\begin{aligned}
\frac{\partial RSS(\hat{\beta}_0, \hat{\beta}_1)}{\partial \hat{\beta}_1} &= \frac{\partial}{\partial \hat{\beta}_1}\sum_{i=1}^{n}(Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2 && \text{(substitute formula for RSS)} \\
&= \sum_{i=1}^{n}\frac{\partial}{\partial \hat{\beta}_1}(Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2 && \text{(linearity property of derivatives)} \\
&= -2\sum_{i=1}^{n}(Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)x_i && \text{(chain rule, factor out -2)}
\end{aligned}
$$

We now set this derivative equal to 0 and rearrange the terms to solve for $\hat{\beta}_1$:

$$
\begin{aligned}
0 &= \frac{\partial RSS(\hat{\beta}_0, \hat{\beta}_1)}{\partial \hat{\beta}_1} \\
0 &= -2\sum_{i=1}^{n}(Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)x_i && \text{(substitute partial derivative)} \\
0 &= \sum_{i=1}^{n}(Y_i - (\bar{Y} - \hat{\beta}_1\bar{x}) - \hat{\beta}_1 x_i)x_i && \text{(substitute OLS estimator of } \hat{\beta}_0\text{, divide both sides by -2)} \\
0 &= \sum_{i=1}^{n} x_i Y_i - \sum_{i=1}^{n} x_i\bar{Y} + \hat{\beta}_1\bar{x}\sum_{i=1}^{n} x_i - \hat{\beta}_1\sum_{i=1}^{n} x_i^2. && \text{(expand sum, use linearity of sum)}
\end{aligned}
$$

Continuing from the previous line, we move the terms involving $\hat{\beta}_1$ to the other side of the equality to get

$$
\begin{aligned}
\hat{\beta}_1\sum_{i=1}^{n} x_i^2 - \hat{\beta}_1\bar{x}\sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} x_i Y_i - \sum_{i=1}^{n} x_i\bar{Y} && \text{(move estimator to other side)} \\
\hat{\beta}_1\sum_{i=1}^{n} x_i^2 - \hat{\beta}_1\frac{1}{n}\sum_{i=1}^{n} x_i\sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} x_i Y_i - \sum_{i=1}^{n} x_i\frac{1}{n}\sum_{i=1}^{n} Y_i && \text{(rewrite using definition of sample means)} \\
\hat{\beta}_1\sum_{i=1}^{n} x_i^2 - \hat{\beta}_1\frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2 &= \sum_{i=1}^{n} x_i Y_i - \frac{1}{n}\sum_{i=1}^{n} x_i\sum_{i=1}^{n} Y_i && \text{(reorder and simplify)} \\
\hat{\beta}_1\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right) &= \sum_{i=1}^{n} x_i Y_i - \frac{1}{n}\sum_{i=1}^{n} x_i\sum_{i=1}^{n} Y_i, && \text{(factoring)}
\end{aligned}
$$

which allows us to obtain

$$
\hat{\beta}_1 = \frac{\sum_{i=1}^{n} x_i Y_i - \frac{1}{n}\sum_{i=1}^{n} x_i\sum_{i=1}^{n} Y_i}{\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2}.
$$

Thus, we have the OLS estimators of the simple linear regression coefficients are

$$
\begin{aligned}
\hat{\beta}_0 &= \bar{Y} - \hat{\beta}_1\bar{x}, \\
\hat{\beta}_1 &= \frac{\sum_{i=1}^{n} x_i Y_i - \frac{1}{n}\sum_{i=1}^{n} x_i\sum_{i=1}^{n} Y_i}{\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2}.
\end{aligned}
$$

3.12.3 Unbiasedness of OLS estimators

We now show that the OLS estimators we derived in Section 3.12.2 are unbiased. An estimator is unbiased if the expected value is equal to the parameter it is estimating.

The OLS estimator assumes we know the value of the regressor variables for all observations. Thus, we must condition our expectation on knowing the regressor matrix $\mathbf{X}$. Thus, we want to show that

$$E(\hat{\beta}_0 \mid \mathbf{X}) = \beta_0,$$

where "$\mid \mathbf{X}$" is convenient notation to indicate that we are conditioning our expectation on knowing the regressor values for every observation.

In Section 3.2, we noted that we assume $E(\epsilon \mid X) = 0$, which is true for every error in our model, i.e. $E(\epsilon_i \mid X = x_i) = 0$ for $i = 1, 2, \ldots, n$. Thus,

$$\begin{aligned}
E(Y_i \mid X = x_i) &= E(\beta_0 + \beta_1 x_i + \epsilon_i \mid X = x_i) && \text{(substiute definition of } Y_i\text{)} \\
&= E(\beta_0 \mid X = x_i) + E(\beta_1 x_i \mid X = X_i) + E(\epsilon_i | X = x_i) && \text{(linearity property of expectation)} \\
&= \beta_0 + \beta_1 x_i + E(\epsilon_i | X = x_i) && \text{(the } \beta\text{s and } x_i \text{ are non-random values)} \\
&= \beta_0 + \beta_1 x_i + 0 && \text{(assumption about errors)} \\
&= \beta_0 + \beta_1 x_i.
\end{aligned}$$

In the derivations below, every sum is over all values of i, i.e., $\sum \equiv \sum_{i=1}^{n}$. We drop the index for simplicity.

Next, we note:

$$\begin{aligned}
E\left(\sum x_i Y_i \,\middle|\, \mathbf{X}\right) &= \sum E(x_i Y_i \mid \mathbf{X}) && \text{(by the linearity of the expectation operator)} \\
&= \sum x_i E(Y_i \mid \mathbf{X}) && (x_i \text{ is a fixed value, so it can be brought out}) \\
&= \sum x_i (\beta_0 + \beta_1 x_i) && \text{(substitute expected value of } Y_i\text{)} \\
&= \sum x_i \beta_0 + \sum x_i \beta_1 x_i && \text{(distribute sum)} \\
&= \beta_0 \sum x_i + \beta_1 \sum x_i^2. && \text{(factor out constants)}
\end{aligned}$$

Also,

$$\begin{aligned}
E(\bar{Y} \mid \mathbf{X}) &= E\left(\frac{1}{n}\sum Y_i \,\middle|\, \mathbf{X}\right) && \text{(definition of sample mean)} \\
&= \frac{1}{n} E\left(\sum Y_i \,\middle|\, \mathbf{X}\right) && \text{(factor out constant)} \\
&= \frac{1}{n}\sum E\left(Y_i \mid \mathbf{X}\right) && \text{(linearity of expectation)} \\
&= \frac{1}{n}\sum (\beta_0 + \beta_1 x_i) && \text{(substitute expected value of } Y_i\text{)} \\
&= \frac{1}{n}\left(\sum \beta_0 + \sum \beta_1 x_i\right) && \text{(distribute sum)} \\
&= \frac{1}{n}\left(n\beta_0 + \beta_1 \sum x_i\right) && \text{(simplify, factor out constant)} \\
&= \beta_0 + \beta_1 \bar{x}. && \text{(simplify)}
\end{aligned}$$

To simplify our derivation below, define

$$SSX = \sum x_i^2 - \frac{1}{n}\left(\sum x_i\right)^2.$$

Thus,

$$
\begin{aligned}
&E(\hat{\beta}_1 \mid \mathbf{X}) \\
&= E\left(\frac{\sum x_i Y_i - \frac{1}{n}\sum x_i \sum Y_i}{\sum x_i^2 - \frac{1}{n}\left(\sum x_i\right)^2} \,\middle|\, \mathbf{X}\right) && \text{(substitute OLS estimator)} \\
&= \frac{1}{SSX} E\left(\sum x_i Y_i - \frac{1}{n}\sum x_i \sum Y_i \,\middle|\, \mathbf{X}\right) && \text{(factor out constant denominator, substitute } SSX\text{)} \\
&= \frac{1}{SSX}\left[E\left(\sum x_i Y_i \,\middle|\, \mathbf{X}\right) - E\left(\frac{1}{n}\sum x_i \sum Y_i \,\middle|\, \mathbf{X}\right)\right] && \text{(linearity of expectation)} \\
&= \frac{1}{SSX}\left[E\left(\sum x_i Y_i \,\middle|\, \mathbf{X}\right) - \left(\sum x_i\right) E\left(\bar{Y} \mid \mathbf{X}\right)\right] && \text{(factor out constant } \sum x_i\text{, use definition of } \bar{Y}\text{)} \\
&= \frac{1}{SSX}\left[\left(\beta_0 \sum x_i + \beta_1 \sum x_i^2\right) - \left(\sum x_i\right)\left(\beta_0 + \beta_1 \bar{x}\right)\right] && \text{(substitute previous derivations)} \\
&= \frac{1}{SSX}\left[\beta_0 \sum x_i + \beta_1 \sum x_i^2 - \beta_0 \sum x_i - \beta_1 \bar{x} \sum x_i\right] && \text{(expand product and reorder)} \\
&= \frac{1}{SSX}\left[\beta_1 \sum x_i^2 - \beta_1 \bar{x} \sum x_i\right] && \text{(cancel terms)} \\
&= \frac{1}{SSX}\left[\beta_1 \sum x_i^2 - \beta_1 \frac{1}{n}\sum x_i \sum x_i\right] && \text{(using definition of sample mean)} \\
&= \frac{1}{SSX}\beta_1\left[\sum x_i^2 - \frac{1}{n}\left(\sum x_i\right)^2\right] && \text{(factor out } \beta_1\text{, simplify)} \\
&= \frac{1}{SSX}\beta_1[SSX] && \text{(substitute } SSX\text{)} \\
&= \beta_1. && \text{(simplify)}
\end{aligned}
$$

Therefore, $\hat{\beta}_1$ is an unbiased estimator of β_1.

Next, we show that $\hat{\beta}_0$ is unbiased:

$$
\begin{aligned}
E(\hat{\beta}_0 \mid \mathbf{X}) &= E(\bar{Y} - \hat{\beta}_1 \bar{x} \mid \mathbf{X}) && \text{(OLS estimator of } \beta_0\text{)} \\
&= E(\bar{Y} \mid \mathbf{X}) - E(\hat{\beta}_1 \bar{x} \mid \mathbf{X}) && \text{(linearity of expectation)} \\
&= E(\bar{Y} \mid \mathbf{X}) - \bar{x} E(\hat{\beta}_1 \mid \mathbf{X}) && \text{(factor out constant)} \\
&= \beta_0 + \beta_1 \bar{x} - \bar{x}\beta_1 && \text{(substitute previous derivations)} \\
&= \beta_0. && \text{(cancel terms)}
\end{aligned}
$$

Therefore, $\hat{\beta}_0$ is an unbiased estimator of β_0.

3.12.4 Manual calculation Penguins simple linear regression example

In this section, we manually produce (i.e., without the `lm` function) the `penguins` simple linear regression example in Section 3.3.

First, we will manually fit a simple linear regression model that regresses `bill_length_mm` on `body_mass_g`.

Using the `summary` function on the `penguins` data frame, we see that both `bill_length_mm` and `body_mass_g` have `NA` values.

```
summary(penguins)
##       species          island    bill_length_mm
##  Adelie   :152   Biscoe   :168   Min.   :32.10
##  Chinstrap: 68   Dream    :124   1st Qu.:39.23
##  Gentoo   :124   Torgersen: 52   Median :44.45
##                                  Mean   :43.92
##                                  3rd Qu.:48.50
##                                  Max.   :59.60
##                                  NA's   :2
##  bill_depth_mm   flipper_length_mm  body_mass_g
##  Min.   :13.10   Min.   :172.0     Min.   :2700
##  1st Qu.:15.60   1st Qu.:190.0     1st Qu.:3550
##  Median :17.30   Median :197.0     Median :4050
##  Mean   :17.15   Mean   :200.9     Mean   :4202
##  3rd Qu.:18.70   3rd Qu.:213.0     3rd Qu.:4750
##  Max.   :21.50   Max.   :231.0     Max.   :6300
##  NA's   :2       NA's   :2         NA's   :2
##      sex           year        pl_fitted
##  female:165   Min.   :2007   Min.   :35.60
##  male  :168   1st Qu.:2007   1st Qu.:38.98
##  NA's  : 11   Median :2008   Median :45.67
##               Mean   :2008   Mean   :43.92
##               3rd Qu.:2009   3rd Qu.:48.34
##               Max.   :2009   Max.   :52.83
##                              NA's   :2
##    sl_fitted
##  Min.   :36.08
##  1st Qu.:38.95
##  Median :45.40
##  Mean   :43.92
##  3rd Qu.:48.42
##  Max.   :53.60
##  NA's   :2
```

This is important to note because the `lm` function automatically removes any observation with `NA` values for any of the variables specified in the `formula` argument. In order to replicate our results, we must remove the same observations.

We want to remove the rows of `penguins` where either `body_mass_g` or `bill_length_mm` have `NA` values. We do that below using the `na.omit` function (selecting only the relevant variables) and assign the cleaned object the name

penguins_clean.

```
# remove rows of penguins where bill_length_mm or body_mass_g have NA values
penguins_clean <-
  penguins |>
  subset(select = c("bill_length_mm", "body_mass_g")) |>
  na.omit()
```

We extract the `bill_length_mm` variable from the `penguins` data frame and assign it the name `y` since it will be the response variable. We extract the `body_mass_g` variable from the `penguins` data frame and assign it the name `x` since it will be the regressor variable. We also determine the number of observations and assign that value the name `n`.

```
# extract response and regressor from penguins_clean
y <- penguins_clean$bill_length_mm
x <- penguins_clean$body_mass_g
# determine number of observations
n <- length(y)
```

We now compute $\hat{\beta}_1$ and $\hat{\beta}_0$ using Equations (3.2.2) and (3.2.2). Note that placing `()` around the assignment operations will both perform the assignment and print the results.

```
# compute OLS estimate of beta1
(b1 <- (sum(x * y) - sum(x) * sum(y) / n)/(sum(x^2) - sum(x)^2/n))
## [1] 0.004051417
# compute OLS estimate of beta0
(b0 <- mean(y) - b1 * mean(x))
## [1] 26.89887
```

The estimated value of β_0 is $\hat{\beta}_0 = 26.90$ and the estimated value of β_1 is $\hat{\beta}_1 = 0.004$.

We can also compute the residuals, the fitted values, the RSS, and the estimated error variance. Using the code below, the RSS for our model is 6564.49 and the estimated error variance if $\hat{\sigma}^2 = 19.31$.

```
yhat <- b0 + b1 * x # compute fitted values
ehat <- y - yhat # compute residuals
(rss <- sum(ehat^2)) # sum of the squared residuals
## [1] 6564.494
(sigmasqhat <- rss/(n-2)) # estimated error variance
## [1] 19.30734
```

3.12.5 Derivation of the OLS estimator for the multiple linear regression model coefficients

We want to determine the value of $\hat{\boldsymbol{\beta}}$ that will minimize

$$\begin{aligned} RSS(\hat{\boldsymbol{\beta}}) &= \sum_{i=1}^{n} \hat{\epsilon_i}^2 \\ &= \hat{\boldsymbol{\epsilon}}^T \hat{\boldsymbol{\epsilon}} \\ &= (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})^T (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}) \\ &= \mathbf{y}^T\mathbf{y} - 2\hat{\boldsymbol{\beta}}^T\mathbf{X}^T\mathbf{y} + \hat{\boldsymbol{\beta}}^T\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}}, \end{aligned}$$

where the second term in the last line comes from the fact that $\hat{\boldsymbol{\beta}}^T\mathbf{X}^T\mathbf{y}$ is a 1×1 matrix, and is thus symmetric. Consequently, $\hat{\boldsymbol{\beta}}^T\mathbf{X}^T\mathbf{y} = (\hat{\boldsymbol{\beta}}^T\mathbf{X}^T\mathbf{y})^T = \mathbf{y}^T\mathbf{X}\hat{\boldsymbol{\beta}}$.

To find the local extrema of $RSS(\hat{\boldsymbol{\beta}})$, we set its derivative with respect to $\hat{\boldsymbol{\beta}}$ equal to 0, and solve for $\hat{\boldsymbol{\beta}}$).

Using the results in Appendix A.5, we see that

$$\frac{\partial RSS(\hat{\boldsymbol{\beta}})}{\partial \hat{\boldsymbol{\beta}}} = -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}}.$$

Setting $\partial RSS(\hat{\boldsymbol{\beta}})/\partial\hat{\boldsymbol{\beta}} = 0$ and using some simple algebra, we derive the **normal equations**

$$\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}^T\mathbf{y}.$$

Assuming the $\mathbf{X}^T\mathbf{X}$ is invertible, which it will be when $\mathbf{X}$ is full-rank, our solution is

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

To show that the OLS estimator of $\boldsymbol{\beta}$ minimizes $RSS(\hat{\boldsymbol{\beta}})$, we technically need to show that the Hessian matrix of $RSS(\hat{\boldsymbol{\beta}})$, the matrix of second-order partial derivatives, is positive definite. In our context, the Hessian matrix is

$$\begin{aligned} \frac{\partial^2 RSS(\hat{\boldsymbol{\beta}})}{\partial \hat{\boldsymbol{\beta}}^2} &= \frac{\partial}{\partial\hat{\boldsymbol{\beta}}}(-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}}) \\ &= 2\mathbf{X}^T\mathbf{X}. \end{aligned}$$

The $p \times p$ matrix $2\mathbf{X}^T\mathbf{X}$ is positive definite, but it is beyond the scope of the course to prove this.

Therefore, the OLS estimator of $\boldsymbol{\beta}$,

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

minimizes the RSS.

3.12.6 Manual calculation of Penguins multiple linear regression example

We manually verify the calculations for the `penguins` example given in Section @ref{s:penguins-mlr}, where we fit the multiple linear regression model regressing `bill_length_mm` on `body_mass_g` and `flipper_length_mm`. We refit the model below, specifying the argument `y = TRUE` so we can get the response vector used in the model.

```
# fit regression model, retaining the y vector
mlmod <- lm(bill_length_mm ~ body_mass_g + flipper_length_mm,
            data = penguins, y = TRUE)
```

We can use `model.matrix` to extract the $\mathbf{X}$ matrix from our fitted model. And because we specified `y = TRUE` in our call to `lm`, we can also extract `y` from the fitted model using the code below.

```
# extract X matrix from fitted model
X <- model.matrix(mlmod)
# extract y vector from fitted model
y <- mlmod$y
```

We'll need to learn a few new commands in R to do the calculations:

- `t` is the transpose of a matrix.
- `%*%` is the multiplication operator for two matrices.
- `solve(A, b)` computes $\mathbf{A}^{-1}\mathbf{b}$.

Thus, we compute $\hat{\boldsymbol{\beta}}$ using the code below, which matches the estimate from the `lm` function.

```
# manually calculate betahat
solve(t(X) %*% X, t(X) %*% y)
##                              [,1]
## (Intercept)       -3.4366939266
## body_mass_g        0.0006622186
## flipper_length_mm  0.2218654584
# betahat from lm function
coef(mlmod)
##       (Intercept)       body_mass_g flipper_length_mm
##     -3.4366939266      0.0006622186      0.2218654584
```

Chapter 4

Interpreting a fitted linear model

Interpreting a fitted model is a critical part of a regression analysis and aids us in determining the role and impact that each variable plays in describing the behavior of the response variable.

4.1 Interpretation of coefficients

The standard approach to interpreting the coefficients of a fitted linear model is to consider the expected change in the response in relation to changes in the regressors in the model.

Consider the typical multiple linear regression model of the response

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_{p-1} X_{p-1} + \epsilon.$$

As discussed in Chapter 3, we treat the values of our regressor variables as being fixed, known values. The error term is treated as a random variable, and consequently, the response variable is also a random variable. Additionally, we assume that the errors all have mean 0, conditional on the values of the regressor variables. More formally, we write this assumption as

$$E(\epsilon \mid X_1, X_2, \ldots, X_{p-1}) = 0.$$

Recall that we use the notation $\mathbb{X} = \{X_1, \ldots, X_{p-1}\}$ to denote the set of all regressors, which will help us simplify the derivations below. Thus, the assumption in Equation (4.1) can be expressed as $E(\epsilon \mid \mathbb{X}) = 0$. Using the assumption in Equation (4.1) and applying it to the model in Equation (4.1), we see that

$$
\begin{aligned}
&E(Y \mid X_1, X_2, \ldots, X_{p-1}) \\
&= E(Y \mid \mathbb{X}) \\
&= E(\beta_0 + \beta_1 X_1 + \ldots + \beta_{p-1} X_{p-1} + \epsilon \mid \mathbb{X}) \\
&= E(\beta_0 + \beta_1 X_1 + \ldots + \beta_{p-1} X_{p-1} \mid \mathbb{X}) + E(\epsilon \mid \mathbb{X}) \\
&= \beta_0 + \beta_1 X_1 + \ldots + \beta_{p-1} X_{p-1}
\end{aligned}
$$

since all terms in the first summand of line 4 are fixed, non-random values conditional on $\mathbb{X}$ and the second summand is 0 by assumption. If you are rusty with properties of random variables, consider reviewing the material in Appendix B.

Using the facts above, we discuss interpretation of simple linear regression models, multiple linear regression models with basic numeric predictors as regressors, and interpretation for parallel and separate lines regression model.

4.1.1 Interpretation for simple linear regression

Suppose we have a simple linear regression model, so that

$$E(Y \mid X) = \beta_0 + \beta_1 X.$$

The interpretations of the coefficients are:

- β_0 is the expected response when the regressor is 0, i.e., $\beta_0 = E(Y \mid X = 0)$.
- β_1 is the expected change in the response when the regressor increases 1 unit, i.e., $\beta_1 = E(Y \mid X = x^* + 1) - E(Y \mid X = x^*)$, where x^* is a fixed, real number.

Regarding the interpretation of β_0, from the regression model in Equation (4.1.1), notice that

$$
\begin{aligned}
E(Y \mid X = 0) &= \beta_0 + \beta_1 \cdot 0 \\
&= \beta_0.
\end{aligned}
$$

This is why β_0 is the expected value of the response variable when the regressor is zero.

Similarly, for β_1, we notice that

$$
\begin{aligned}
E(Y \mid X = x^* + 1) - E(Y \mid X = x^*) &= [\beta_0 + \beta_1 (x^* + 1)] - [\beta_0 + \beta_1 x^*] \\
&= \beta_1.
\end{aligned}
$$

Thus, β_1 literally equals the change in the expected response when the regressor increases by 1 unit.

The regressors we use in our regression analysis are often observational in nature, meaning that we do not control them. And even if we could control them, they may be difficult to change. E.g., if one of our regressors was the size of an island, how would we realistically go about increasing the size of the island?

thus, in the context of observational data, it may not make sense to say "we increase X by 1 unit" or "when X increases by 1 unit". An alternative approach, alluded to by Faraway (2014), is to consider the expected response difference between observations that are identical with respect to all regressors we include in our model except the regressor under consideration, which varies by only a single unit. While mathematically, the result is the same, the interpretation is more philosophically palatable. Regardless, interpretation is a bit of an art. There can be many correct ways to interpret a coefficient or the impact of a variable. Always double-check that the mathematics of your model supports your conclusions.

To illustrate the interpretations given above, we interpret the simple linear regression model fit to the `penguins` data in Section 3.3. From Section 3.3, the fitted simple linear regression model of `body_mass_g` regressed on `body_mass_g` is

$$\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}) = 26.9 + 0.004\,\texttt{body\_mass\_g}.$$

Some basic interpretations of the coefficients are:

- Intercept: The expected bill length of a penguin with a body mass of 0 grams is 26.9 mm. We discussed the absurdity of this interpretation in Section 3.3.
- `body_mass_g`: A penguin 1 gram heavier than another penguin is expected to have a bill length 0.004 mm longer than the smaller penguin.

The scale of the latter interpretation is difficult to comprehend. A weight difference of 1 gram is negligible in the context of penguin weights, and a bill length change of 0.004 mm is unlikely to be noticed by the naked eye. This suggests that rescaling our `body_mass_g` predictor could result in a more natural interpretation of the associated coefficient. In the code below, we divide the `body_mass_g` variable by 1000 to convert the variable from grams to kilograms, and consequently, save the rescaled variable as `body_mass_kg`. We then fit the model regressing `bill_length_mm` on `body_mass_kg` and extract the estimated coefficients.

```
# load penguins data
data(penguins, package = "palmerpenguins")
# transform body mass variable from g to kg
penguins <- penguins |> transform(body_mass_kg = body_mass_g/1000)
# fit model with body_mass_kg
slmod_scaled <- lm(bill_length_mm ~ body_mass_kg, data = penguins)
# extract coefficients
coefficients(slmod_scaled)
##  (Intercept) body_mass_kg
##    26.898872     4.051417
```

The fitted model from above is

$$\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_kg}) = 26.9 + 4.05\,\texttt{body\_mass\_kg}.$$

Thus, we can interpret the estimated coefficient for `body_mass_kg` as something like, "A penguin that is 1 kg larger than another penguin is expected to have a bill length 4 mm longer than the smaller penguin".

Comparing the estimated coefficients from Equations (4.1.1) and (4.1.1), we see that dividing `body_mass_g` by 1000 resulted in the estimated coefficient changing by a factor of 1000. More generally, if $\hat{\beta}_j$ is the estimated coefficient for X_j, then the regressor $(X_j + a)/c$ will have an estimated coefficient of $c\hat{\beta}_j$, where a and c are fixed, real numbers and assuming nothing else in the fitted model changes.

4.1.2 Interpretation for first-order multiple linear regression models

Suppose we have a multiple linear regression model with $p-1$ *numeric* regressors, so that

$$E(Y \mid X_1, \ldots, X_{p-1}) = \beta_0 + \beta_1 X_1 + \cdots + \beta_{p-1} X_{p-1}.$$

Relying on the definition of $\mathbb{X}$, we denote the set of regressors without X_j as $\mathbb{X}_{-j} = \mathbb{X} \setminus \{X_j\}$.

The interpretations of the coefficients from the model in Equation (4.1.2) are:

- β_0 is the expected response when all regressors are 0, i.e., $\beta_0 = E(Y \mid X_1 = 0, \ldots, X_{p-1} = 0)$.
- β_j, $j = 1, \ldots, p-1$, represents the expected change in the response when regressor j increases 1 unit and the other regressors stay the same, i.e., $\beta_j = E(Y \mid \mathbb{X}_{-j} = \mathbf{x}^*_{-j}, X_{j+1} = x^*_j + 1) - E(Y \mid \mathbb{X}_{-j} = \mathbf{x}^*_{-j}, X_{j+1} = x^*_j)$ where $\mathbf{x}^*_{-j} = [x^*_1, \ldots, x^*_{j-1}, x^*_{j+1}, \ldots, x^*_{p-1}] \in \mathbb{R}^{p-2}$ is a vector with $p-2$ fixed values (the number of regressors excluding X_j) and x^*_j is a fixed real number. The non-intercept coefficients of a multiple linear regression model are known as **partial slopes**.

Regarding the interpretation of β_0, from the regression model in Equation (4.1.2), notice that

$$\begin{aligned} E(Y \mid X_1 = 0, \ldots, X_{p-1} = 0) &= \beta_0 + \beta_1 \cdot 0 + \cdots + \beta_{p-1} \cdot 0 \\ &= \beta_0. \end{aligned}$$

It is quite common for the mathematical interpretation of the intercept to be nonsensical because we are extrapolating outside the range of the observed data.

For β_j, $j = 1, \ldots, p-1$, we notice that

$$\begin{aligned}
&E(Y \mid \mathbb{X}_{-j} = \mathbf{x}^*_{-j}, X_j = x^*_j + 1) - E(Y \mid \mathbb{X}_{-j} = \mathbf{x}^*_{-j}, X_j = x^*_j) \\
&= \left[\beta_0 + \sum_{k=1}^{j-1} \beta_k x^*_k + \beta_j (x^*_j + 1) + \sum_{k=j+1}^{p-1} \beta_k x^*_k\right] \\
&\quad - \left[\beta_0 + \sum_{k=1}^{j-1} \beta_k x^*_k + \beta_j x^*_j + \sum_{k=j+1}^{p-1} \beta_k x^*_k\right] \\
&= \beta_j.
\end{aligned}$$

A notable problem with the standard mathematical interpretation of multiple regression models is that a single predictor can be used more than once in the model. E.g., in the 2nd-degree polynomial regression model

$$E(Y \mid X) = \beta_0 + \beta_1 X + \beta_2 X^2,$$

X is used in both the second and third terms. So it is not possible to increase X while keeping X^2 fixed. The mathematical interpretation given in this section is applicable to first-order linear regression models (cf. Section 3.7), where a *first-order linear regression model* is a multiple linear regression model in which no regressor is a function of any other regressor.

Additionally, the interpretation given above for the partial slope coefficients applies to the coefficients of numeric predictors that can increase by 1 unit; that interpretation doesn't apply to the coefficients for categorical predictors, which are discussed in Section 4.3.

To illustrate the interpretations given above, we interpret the first-order multiple linear regression model fit to the `penguins` data in Section 3.6. The fitted multiple linear regression model is

$$\begin{aligned}
&\hat{E}(\texttt{bill_length_mm} \mid \texttt{body_mass_g}, \texttt{flipper_length_mm}) \\
&= -3.44 + 0.0007\,\texttt{body_mass_g} + 0.22\,\texttt{flipper_length_mm}.
\end{aligned}$$

Some basic interpretations of the coefficients are:

- Intercept: We expect a penguin with a body mass of 0 grams and a flipper length of 0 mm to to have a bill length of -3.44 mm.
- `body_mass_g`: For two penguins that have the same flipper length but one penguin has a body mass 1 gram larger, we expect the heavier penguin to have a bill length 0.0007 mm longer than the other penguin.
- `flipper_length_mm`: For two penguins with the same body mass but whose flipper lengths differ by 1 mm, we expect the penguin with longer flippers to have a bill length 0.22 mm longer than the other penguin.

4.1.3 Roles of regressor variables

Did you notice that the estimated coefficients for the intercept and the `body_mass_g` regressor changed between the fitted simple linear model and the fitted multiple linear regression model for the `penguins` example in the sections above (cf. Equations (4.1.1) and (4.1.2))? Why does this happen? What is going on?

The role a regressor plays in a regression model depends on what other regressors are in the model. Recall a team setting you've been in where you had to work with others to accomplish something; it could be related to school, work, sports, etc. Depending on the skills and knowledge your team members have, you will try to find a role in which you can effectively help the team. Something similar happens in regression models. Generally, we can't provide a definitive interpretation of a regressor's role in a fitted model without knowing what other regressors are in thee model. Thus, when interpreting a regressor, it is common to include the something like *after accounting for the other variables in the model*. We do this because we are giving the interpretation of that regressor's role when the other variables are also in the model. If our model had different variables, then our interpretation would be different.

There is also a technical reason why the estimated coefficients change as you add or remove regressors from a model. If a regressor is correlated with other regressors in a model, then adding or removing that regressor will impact the estimated coefficients in the new model. The more correlated the regressors are, the more they tend to affect each others' estimated coefficients. More formally, a regressor will impact the estimated coefficients of the other regressors in a model unless it is **orthogonal** to the other regressors. Orthogonality is related to correlation, but there are important differences. See Section 4.5.1), which provides an extensive discussion of orthogonality and how it affects estimation.

4.2 Effect plots

An effect plot is a visual display that aids in helping us intuitively interpret the impact of a *predictor* in a model. As stated by Fox and Weisberg (2020):

> Summarization of the effects of predictors using tables of coefficient estimates is often incomplete. Effects, and particularly plots of effects, can in many instances reveal the relationship of the response to the predictors more clearly. This conclusion is especially true for models with linear predictors that include interactions and multiple-coefficient terms such as regression splines and polynomials ....

More formally, an **effect plot** is a plot of the estimated mean response as a function of a *focal predictor* with the other *predictors* being held at "typical values". The distinction between predictor and regressor is important when discussing effect plots, because we can create effect plots for each predictor but

not necessarily each regressor. Recall from Section 3.1 that a predictor variable is a variable available to model the response variable, while a regressor variable is a variable used in our regression model, whether that is an unmodified predictor variable, some transformation of a predictor, some combination of predictors, etc.

We first discuss how to create an effect plot for a first-order linear regression models with numeric regressors since these are the simplest to construct. In a first-order linear model, none of the regressors interact, i.e., none of the regressors are functions of each other. Fox and Weisberg (2020) use the terminology **fixed group** to refer to the group of predictors that do not interact with the focal predictor. For a first-order regression model, all of the non-focal predictors are part of the fixed group. To create our effect plot, we must first find the equation for the estimated mean response as a function of a focal predictor while holding the other predictors at their "typical" values. We set numeric fixed group predictors equal to their sample means when finding this function.

We now construct effect plots for the estimated regression model of the `penguins` data that regressed `bill_length_mm` on `body_mass_g` and `flipper_length_mm` (previously considered in Section 3.6). We refit that model using the code below and assign it the name `mlmod`. The fitted model is

$$\begin{aligned}&\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{flipper\_length\_mm})\\ &= -3.44 + 0.0007\,\texttt{body\_mass\_g} + 0.22\,\texttt{flipper\_length\_mm}.\end{aligned}$$

```
# load penguins data since it hasn't been loaded in this chapter
data(penguins, package = "palmerpenguins")
# refit multiple linear regression model
mlmod <- lm(bill_length_mm ~ body_mass_g + flipper_length_mm, data = penguins)
# extract estimated coefficients
coef(mlmod)
##       (Intercept)       body_mass_g flipper_length_mm
##     -3.4366939266      0.0006622186      0.2218654584
```

There are two predictors in the model in Equation (4.2), `body_mass_g` and `flipper_length_mm`, so we can create effect plots for both variables. Since each predictor is numeric and doesn't interact with other predictors, the "typical" value used to determine the estimated mean response will be the sample mean of the observed predictor values. When fitting a linear model in R, R will automatically drop any observations that have `NA` values for any variables used in our model. Thus, the sample means used in our effect plot should correspond to the sample mean of the values actually used to fit the model. If our fitted model was assigned the name `lmod`, the response and predictor values used to fit the model may be extracted using `lmod$model`. Since the response variable and predictor variables used to fit `mlmod` are all numeric, we can use the `colMeans` function to get the sample mean of each variable. The sample means of the `body_mass_g` and `flipper_length_mm` values used to fit `mlmod` are $\overline{\texttt{body\_mass\_g}} = 4201.75$

and $\overline{\texttt{flipper\_length\_mm}} = 200.92$, respectively, as shown in the code below.

```
colMeans(mlmod$model)
##    bill_length_mm       body_mass_g flipper_length_mm
##          43.92193        4201.75439         200.91520
```

The effect plot for `body_mass_g` (on the response `bill_length_mm`) is a plot of

$$\begin{aligned}
&\hat{E}(\texttt{bill_length_mm} \mid \texttt{body_mass_g}, \texttt{flipper_length_mm} = 200.92) \\
&= -3.44 + 0.0007\,\texttt{body_mass_g} + 0.22 \cdot 200.92 \\
&= 41.14 + 0.0007\,\texttt{body_mass_g}
\end{aligned}$$

as a function of `body_mass_g`. Note: we used exact values in the calculation above. The intercept will be 40.76 instead of 41.14 if you use the rounded values. Similarly, the effect plot for `flipper_length_mm` is a plot of

$$\begin{aligned}
&\hat{E}(\texttt{bill_length_mm} \mid \texttt{body_mass_g} = 4201.75, \texttt{flipper_length_mm}) \\
&= -3.44 + 0.0007 \cdot 4201.75 + 0.22\,\texttt{flipper_length_mm} \\
&= -0.65 + 0.22\,\texttt{flipper_length_mm}
\end{aligned}$$

as a function of `flipper_length_mm`.

The **effects** package (Fox et al. 2022) can be used to generate effect plots for the predictors of a fitted linear model. We start by using the `effects::predictorEffect` function to compute the information needed to draw the plot, then the `plot` function to display the information.

The `predictorEffect` function computes the estimated mean response for different values of the focal predictor while holding the other predictors at their typical values. The main arguments of `predictorEffect` are:

- `predictor`: the name of the predictor you want to plot. This is the "focal predictor".
- `mod`: the fitted model. The function works with `lm` objects, but also many other types of fitted models.

The `plot` function will take the output of the `predictorEffect` function and produce the desired effect plot.

In the code below, we load the **effects** package (so that we can use the `predictorEffect` function) and then combine calls to the `plot` and `predictorEffect` functions to create an effect plot for `body_mass_g`, which is shown in Figure 4.1.

```
# load effects package
library(effects)
## Loading required package: carData
## lattice theme set by effectsTheme()
## See ?effectsTheme for details.
```

```
# draw effect plot for body_mass_g
plot(predictorEffect("body_mass_g", mlmod))
```

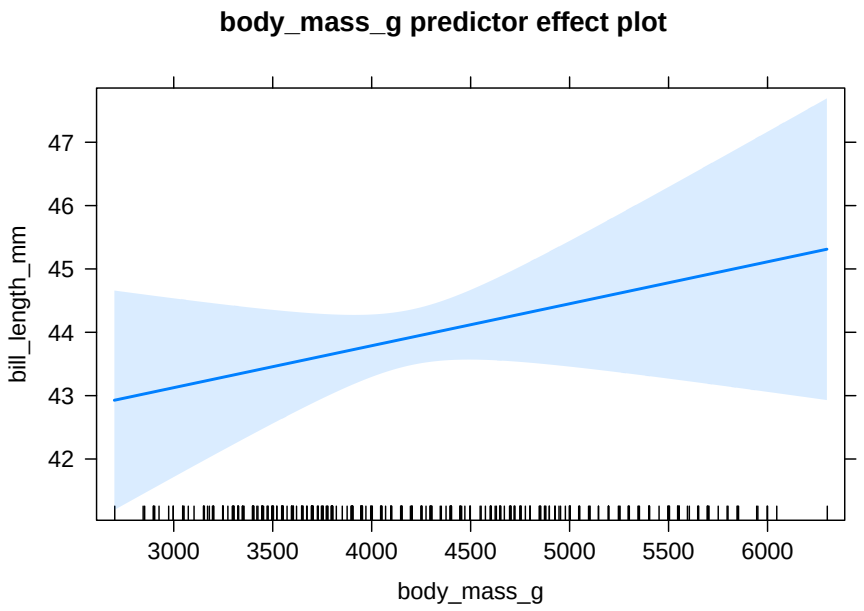


Figure 4.1: Effect plot for body mass based on the fitted model in Equation (4.2).

We see from Figure 4.1 that there is a clear positive association between `body_mass_g` and `bill_length_mm` after accounting for the `flipper_length_mm` variable. The shaded area indicates the 95% confidence interval bands for the estimated mean response. We do not discuss confidence interval bands here, except to say that they provide a visual picture of the uncertainty of our estimated mean (wider bands indicate greater uncertainty). Chapter 6 discusses confidence intervals for linear models in some detail. The many tick marks along the the x-axis of the effect plot indicate observed values of the x-axis variable.

We next create an effect plot for `flipper_length_mm`, which is shown in Figure 4.2, using the code below. There is a clear positive association between `flipper_length_mm` and `bill_length_mm` after accounting for `body_mass_g`.

```
# draw effect plot for flipper_length_mm
plot(predictorEffect("flipper_length_mm", mlmod))
```

Alternatively, we could use `effects::allEffects` to compute the necessary effect plot information for all predictors simultaneously, then use `plot` to create a display of the effect plots for all predictors in one graphic. This approach is quicker, but the individual effect plots can sometimes be too small for practical

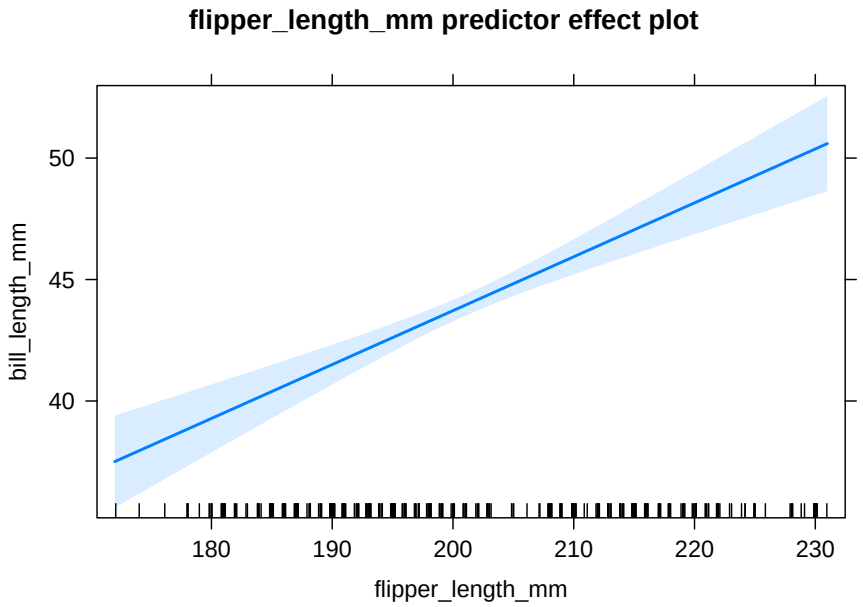


Figure 4.2: Effect plot for flipper length based on the fitted model in Equation (4.2).

use. We demonstrate this faster approach in the code below, which produces Figure 4.3.

```
plot(allEffects(mlmod))
```

4.3 Interpretation for categorical predictors

We now discuss the interpretation of regression coefficients in the context of a parallel lines and separate lines models.

4.3.1 Coefficient interpretation for parallel lines models

Consider a parallel lines model with numeric regressor X and categorical predictor C with levels L_1, L_2, and L_3. Following the discussion in Section 3.8, predictor C must be transformed into two indicator variables, D_2 and D_3, for category levels L_2 and L_3, to be included in our linear model. L_1 is the reference level. The parallel lines model is formulated as

$$E(Y \mid X, C) = \beta_{int} + \beta_X X + \beta_{L_2} D_2 + \beta_{L_3} D_3,$$

where we replace the usual β_0, β_1, β_2, and β_3 with notation the indicates the regressor each coefficient is associated with.

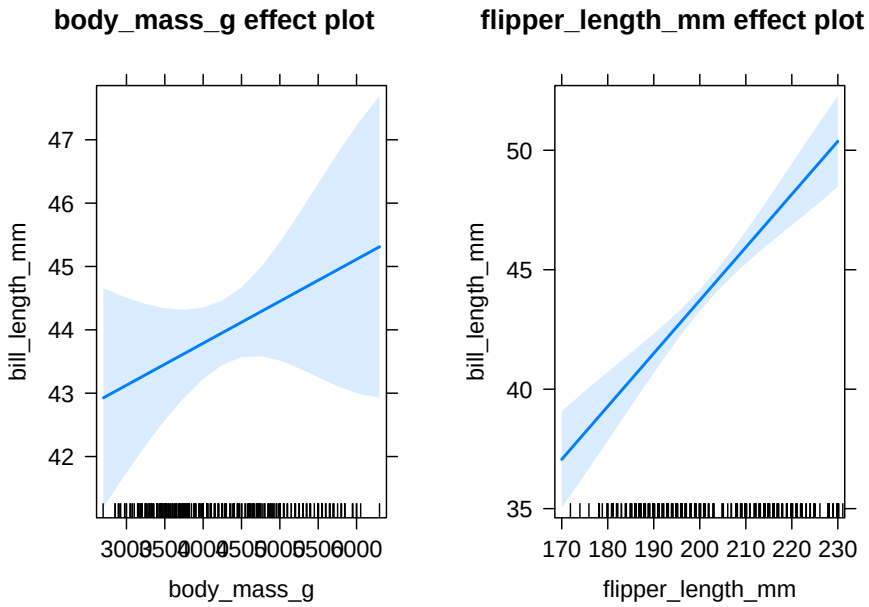


Figure 4.3: All effect plots for predictors of the fitted model in Equation (4.2).

When an observation has level L_1 and $X = 0$, then the expected response is

$$\begin{aligned} E(Y|X = 0, C = L_1) &= \beta_{int} + \beta_X \cdot 0 + \beta_{L_2} \cdot 0 + \beta_{L_3} \cdot 0 \\ &= \beta_{int}. \end{aligned}$$

Thus, β_{int} is the expected response for an observation with level L_1 when $X = 0$.

When an observation has a fixed level L_j (it doesn't matter which level) and X increases from x^* to $x^* + 1$, then the change in the expected response is

$$E(Y|X = x^* + 1, C = L_j) - E(Y|X = x^*, C = L_j) = \beta_X.$$

Thus, β_X is the expected change in the response for an observation with fixed level L_j when X increases by 1 unit.

When an observation has level L_2, the expected response is

$$\begin{aligned} E(Y \mid X = x^*, C = L_2) &= \beta_{int} + \beta_X x^* + \beta_{L_2} \cdot 1 + \beta_{L_3} \cdot 0 \\ &= \beta_{int} + \beta_X x^* + \beta_{L_2}. \end{aligned}$$

Thus,

$$\begin{aligned} &E(Y \mid X = x^*, C = L_2) - E(Y \mid X = x^*, C = L_1) \\ &= (\beta_{int} + \beta_X x^* + \beta_{L_2}) - (\beta_{int} + \beta_X x^*) \\ &= \beta_{L_2}. \end{aligned}$$

Thus, β_{L_2} is the expected change in the response for a fixed value of X when comparing on observation having level L_1 to level L_2 of predictor C. More specifically, β_{L_2} indicates the distance between the estimated regression lines for observations having levels L_1 and L_2. A similar interpretation holds for β_{L_3} when comparing observations having level L_3 to observations having level L_1. L_1 is known as as the reference level of C because we must refer to it to interpret our model with respect to other levels of C.

To summarize the interpretation of the coefficients in parallel lines models like Equation (4.3.1), assuming categorical predictor C has K levels instead of 3:

- β_{int} represents the expected response for observations having the reference level when the numeric regressor $X = 0$.
- β_X is the expected change in the response when X increases by 1 unit for a fixed level of C.
- β_{L_j}, for $j = 2, \ldots, K$, represents the expected change in the response when comparing observations having level L_1 and L_j with X fixed at the same value.

In Section 3.9, we fit a parallel lines model to the `penguins` data that used both `body_mass_g` and `species` to explain the behavior of `bill_length_mm`. Letting D_C denote the indicator variable for the `Chinstrap` level and D_G denote the indicator variable for the `Gentoo` level, the fitted parallel lines model was

$$\begin{aligned}&\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species})\\ &= 24.92 + 0.004\texttt{body\_mass\_g} + 9.92D_C + 3.56D_G.\end{aligned}$$

In the context of this model:

- The expected bill length for an Adelie penguin with a body mass of 0 grams is 24.92 mm.
- If two penguins are of the same species, but one penguin has a body mass 1 gram larger, then the larger penguin is expected to have a bill length 0.004 mm longer than the smaller penguin.
- A Chinstrap penguin is expected to have a bill length 9.92 mm longer than an Adelie penguin, assuming their body mass is the same.
- A Gentoo penguin is expected to have a bill length 3.56 mm longer than an Adelie penguin, assuming their body mass is the same.

4.3.2 Effect plots for fitted models with non-interacting categorical predictors

How do we create an effect plot for a numeric focal predictor when a non-interacting categorical predictor is in the model (such as for the parallel lines model we have been discussing)? In short, we determine the fitted model as a function of the focal predictor for each level of the categorical predictor, and then compute the weighted average of the equation with the weights being proportional to the number of observations in each group.

Let's construct an effect plot for the `body_mass_g` predictor in the context of the `penguins` parallel lines model discussed in the previous section. In Section 3.9, we determined that the fitted parallel lines model simplified to the following equations depending on the level of `species`:

$$\begin{aligned}
&\hat{E}(\texttt{bill_length_mm} \mid \texttt{body_mass_g}, \texttt{species} = \texttt{Adelie}) \\
&= 24.92 + 0.004\texttt{body_mass_g} \\
&\hat{E}(\texttt{bill_length_mm} \mid \texttt{body_mass_g}, \texttt{species} = \texttt{Chinstrap}) \\
&= 34.84 + 0.004\texttt{body_mass_g} \\
&\hat{E}(\texttt{bill_length_mm} \mid \texttt{body_mass_g}, \texttt{species} = \texttt{Gentoo}) \\
&= 28.48 + 0.004\texttt{body_mass_g}.
\end{aligned}$$

We recreate the fitted model producing these equations in R using the code below.

```
# refit the parallel lines model
lmodp <- lm(bill_length_mm ~ body_mass_g + species, data = penguins)
# double-check coefficients
coef(lmodp)
##      (Intercept)      body_mass_g speciesChinstrap
##     24.919470977      0.003748497      9.920884113
##    speciesGentoo
##      3.557977539
```

The code below determines the number of observations with each level of `species` for the data used in the fitted model `lmodp`. We see that 151 Adelie, 68 Chinstrap, and 123 Gentoo penguins (342 total penguins) were used to fit the model stored in `lmodp`.

```
table(lmodp$model$species)
##
##    Adelie Chinstrap    Gentoo
##       151        68       123
```

The equation used to create the effect plot of `body_mass_g` is the weighted average of the equations in Equation (4.3.2), with weights proportional to the number of observational having each level of the categorical predictor. Specifically,

$$\begin{aligned}
&\hat{E}(\texttt{bill_length_mm} \mid \texttt{body_mass_g}, \texttt{species} = \texttt{typical}) \\
&= \frac{151}{342}(24.92 + 0.004\texttt{body_mass_g}) \\
&\quad + \frac{68}{342}(34.84 + 0.004\texttt{body_mass_g}) \\
&\quad + \frac{123}{342}(28.48 + 0.004\texttt{body_mass_g}) \\
&= 28.17 + 0.004\texttt{body_mass_g}.
\end{aligned}$$

The effect plot for `body_mass_g` for the fitted parallel lines model is shown in Figure 4.4. The association between `bill_length_mm` and `body_mass_g` is positive after accounting for `species`.

```
# draw effect plot for body_mass_g
plot(predictorEffect("body_mass_g", lmodp))
```

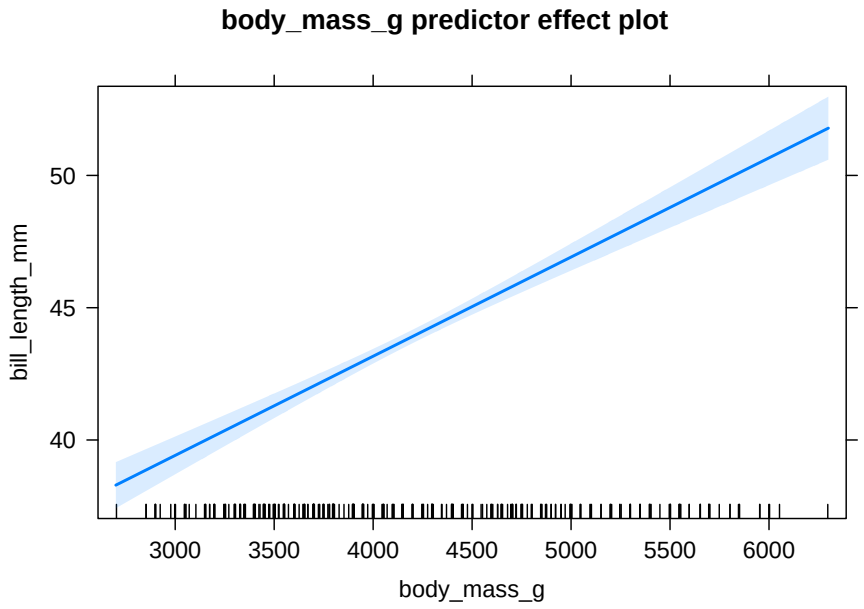


Figure 4.4: The effect plot of `body_mass_g` for the fitted parallel lines model.

An effect plot for a categorical predictor, assuming all other predictors in the model are non-interacting numerical predictors (i.e., fixed group predictors), is a plot of the estimated mean response for each level of the categorical variable when the fixed group group predictors are held at their sample mean. The sample mean of the `body_mass_g` values used to fit `lmodp` is 4201.75, as shown in the code below.

```
# sample mean of body_mass_g variable used to fit lmodp
mean(lmodp$model$body_mass_g)
## [1] 4201.754
```

Using the first equation in Equation (4.3.2), the estimated mean for the Adelie species when `body_mass_g` is fixed at 4201.75 is

$$\begin{aligned}
&\hat{E}(\texttt{bill_length_mm} \mid \texttt{body_mass_g} = 4201.75, \texttt{species} = \texttt{Adelie}) \\
&= 24.92 + 0.004 \cdot 4201.75 \\
&= 40.67.
\end{aligned}$$

Similarly, $\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g} = 4201.75, \texttt{species} = \texttt{Chinstrap}) = 50.59$ and $\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g} = 4201.75, \texttt{species} = \texttt{Gentoo}) = 44.23$.

The code below produces the effect plot for `species`, which is shown in Figure 4.5. We see that after accounting for `body_mass_g`, the `bill_length_mm` tends to be largest for Chinstrap penguins, second largest for Gentoo penguins, and smallest for Adelie penguins. The confidence bands for the estimated mean response are shown by the vertical bars.

```
plot(predictorEffect("species", lmodp))
```

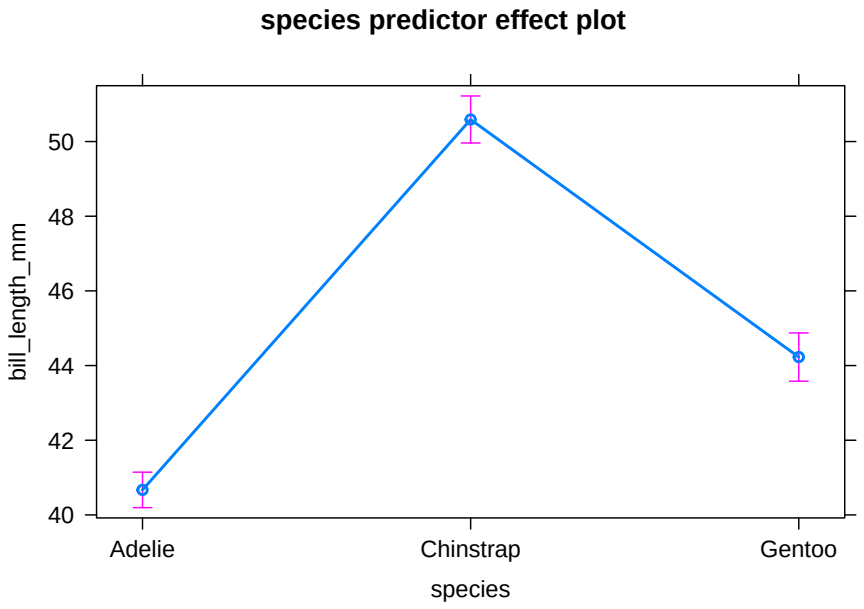


Figure 4.5: An effect plot for `species` after accounting for `body_mass_g`.

4.3.3 Coefficient interpretation for separate lines models

Consider a separate lines model with numeric regressor X and categorical predictor C with levels L_1, L_2, and L_3. The predictor C will be transformed into two indicator variables, D_2 and D_3, for category levels L_2 and L_3, with L_1 being the reference level. The separate lines model is formulated as

$$E(Y \mid X, C) = \beta_{int} + \beta_X X + \beta_{L_2} D_2 + \beta_{L_3} D_3 + \beta_{XL_2} X D_2 + \beta_{XL_3} X D_3.$$

When an observation has level L_1 and $X = x^*$, then the expected response is

$$\begin{aligned}
&E(Y \mid X = x^*, C = L_1) \\
&= \beta_{int} + \beta_X \cdot x^* + \beta_{L_2} \cdot 0 + \beta_{L_3} \cdot 0 + \beta_{XL_2} \cdot x^* \cdot 0 + \beta_{XL_3} \cdot x^* \cdot 0 \\
&= \beta_{int} + \beta_X x^*.
\end{aligned}$$

Using Equation (4.3.3), we can verify that:

- $\beta_{int} = E(Y \mid X = 0, C = L_1)$.
- $\beta_X = E(Y \mid X = x^* + 1, C = L_1) - E(Y \mid X = x^*, C = L_1)$.

Similarly, when $C = L_2$,

$$\begin{aligned}
&E(Y|X = x^*, C = L_2) \\
&= \beta_{int} + \beta_X \cdot x^* + \beta_{L_2} \cdot 1 + \beta_{L_3} \cdot 0 + \beta_{XL_2} \cdot x^* \cdot 1 + \beta_{XL_3} \cdot x^* \cdot 0 \\
&= \beta_{int} + \beta_X x^* + \beta_{L_2} + \beta_{XL_2} x^* \\
&= (\beta_{int} + \beta_{L_2}) + (\beta_X + \beta_{XL_2}) x^*.
\end{aligned}$$

Following this same pattern, when $C = L_3$ we have

$$E(Y|X = x^*, C = L_3) = (\beta_{int} + \beta_{L_3}) + (\beta_X + \beta_{XL_3}) x^*.$$

Using Equations (4.3.3), (4.3.3), and (4.3.3), we can verify that for $j = 2, 3$,

$$\beta_{L_j} = E(Y \mid X = 0, C = L_j) - E(Y \mid X = 0, C = L_1),$$

and

$$\begin{aligned}
&\beta_{XL_j} \\
&= [E(Y \mid X = x^* + 1, C = L_j) - E(Y \mid X = x^*, C = L_j)] \\
&\quad - [E(Y \mid X = x^* + 1, C = L_1) - E(Y \mid X = x^*, C = L_1)].
\end{aligned}$$

To summarize the interpretation of the coefficients in separate lines models like Equation (4.3.3), assuming categorical predictor C has K levels instead of 3:

- β_{int} represents the expected response for observations having the reference level when the numeric regressor $X = 0$.
- β_{L_j}, for $j = 2, \ldots, K$, represents the expected change in the response when comparing observations having level L_1 and L_j with $X = 0$.
- β_X represents the expected change in the response when X increases by 1 unit for observations having the reference level.
- $\beta_X L_j$, for $j = 2, \ldots, K$, represents the difference in the expected response between observations having the reference level in comparison to level L_j when X increases by 1 unit. More simply, these terms represent the difference in the rate of change for observations having level L_j compared to the reference level.

We illustrate these interpretation using the separate lines model to the `penguins` data in Section in Section 3.9. The fitted separate lines model was

$$\begin{aligned}&\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species})\\ &= 26.99 + 0.003\texttt{body\_mass\_g} + 5.18D_C - 0.25D_G\\ &\quad + 0.001D_C\texttt{body\_mass\_g} + 0.0009D_G\texttt{body\_mass\_g}.\end{aligned}$$

In the context of this model:

- The expected bill length for an Adelie penguin with a body mass of 0 grams is 26.99 mm.
- If an Adelie penguin has a body mass 1 gram larger than another Adelie penguin, then the larger penguin is expected to have a bill length 0.003 mm longer than the smaller penguin.
- A Chinstrap penguin is expected to have a bill length 5.18 mm longer than an Adelie penguin when both have a body mass of 0 grams.
- A Gentoo penguin is expected to have a bill length 0.25 mm shorter than an Adelie penguin when both have a body mass of 0 grams.
- For each 1 gram increase in body mass, we expect the change in bill length by Chinstrap penguins to be 0.001 mm larger than the corresponding change in bill length by Adelie penguins.
- For each 1 gram increase in body mass, we expect the change in bill length by Gentoo penguins to be 0.0009 mm larger than the corresponding change in bill length by Adelie penguins.

4.3.4 Effect plots for interacting categorical predictors

We now discuss construction of effect plots for a separate lines model, which has an interaction between a categorical and numeric predictor. In additional to the focal and fixed group predictors we have previously discussed, Fox and Weisberg (2020) also discuss predictors in the **conditioning group**, which is the set of predictors that interact with the focal predictor.

When some predictors interact with the focal predictor, the effect plot of the focal predictor is a plot of the estimated mean response when the fixed group predictors are held at their typical values and the conditioning group predictors vary over different combinations of discrete values. The process of determining the estimated mean response as function of the focal predictor is similar to before, but there are more predictor values on which to condition. By default, to compute the estimated mean response as a function of the focal predictor, we:

- Hold the numeric fixed group predictors at their sample means.
- Average the estimated mean response equation across the different levels of a fixed group categorical predictor, with weights equal to the number of observations with each level.
- We compute the estimated mean response function for 5 discrete values of numeric predictors in the conditioning group.

- We compute the estimated mean response function for different levels of a categorical predictor in the conditioning group.

We provide examples of the effect plots for the `body_mass_g` and `species` predictors for the separate lines model fit to the `penguins` data and given in Equation (4.3.3). We first run the code below to fit the separate lines model previously fit in Section 3.9.

```
# fit separate lines model
lmods <- lm(bill_length_mm ~ body_mass_g + species + body_mass_g:species,
            data = penguins)
# extract estimated coefficients
coef(lmods)
##                   (Intercept)                   body_mass_g
##                 26.9941391367                0.0031878758
##              speciesChinstrap                 speciesGentoo
##                  5.1800537287                -0.2545906615
## body_mass_g:speciesChinstrap    body_mass_g:speciesGentoo
##                  0.0012748183                 0.0009029956
```

We previously determined in Section 3.9 that the model simplifies depending on the level of species. Specifically, we have that

$$\begin{aligned}
&\hat{E}(\texttt{bill_length_mm} \mid \texttt{body_mass_g}, \texttt{species} = \texttt{Adelie}) \\
&= 26.99 + 0.003\texttt{body_mass_g}, \\
&\hat{E}(\texttt{bill_length_mm} \mid \texttt{body_mass_g}, \texttt{species} = \texttt{Chinstrap}) \\
&= 31.17 + 0.004\texttt{body_mass_g}, \\
&\hat{E}(\texttt{bill_length_mm} \mid \texttt{body_mass_g}, \texttt{species} = \texttt{Chinstrap}) \\
&= 26.74 + 0.004\texttt{body_mass_g}.
\end{aligned}$$

The effect plot of `body_mass_g` for the separate lines model will be a plot of each equation given in Equation (4.3.4). Figure 4.6 displays this effect plot, which was created using the code below. We use the `axes` argument to rotate the x-axis labels (otherwise the text overlaps) and the `lines` arguments to display all three lines in one graphic instead of a separate panel for each level of `species`. We notice Chinstrap penguins tend to have the largest bill lengths for a given value of body mass and the bill lengths increase more quickly as a function of body mass then for the Adelie and Gentoo penguins. Similarly, the Adelie penguins tend to have the smallest bill length for a fixed value of body mass and the bill length tends to increase more slowly as body mass increases compared to the other two types of penguins.

```
# effect plot of body mass for separate lines model
# axes ... rotates the x-axis labels 90 degrees
# lines ... plots the effect of body mass in one graphic
plot(predictorEffect("body_mass_g", lmods),
```

```
     axes = list(x = list(rotate = 90)),
     lines = list(multiline = TRUE))
```

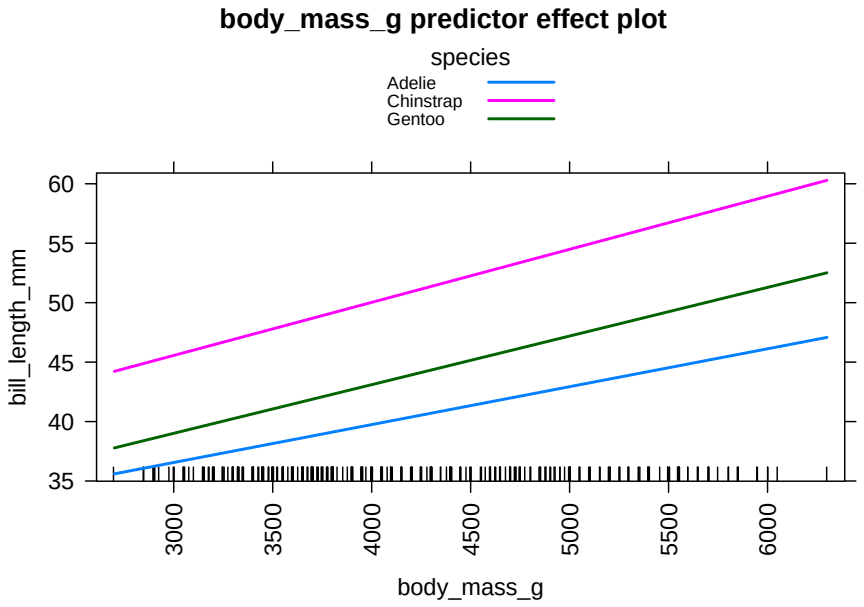


Figure 4.6: Effect plot for body mass based on the equations in Equation (4.3.4).

The effect plot of `species` for the separate lines model will be a plot of the estimated mean response in Equation (4.3.4) for each level of `species` when varying `body_mass_g` over 5 discrete values. Figure 4.7 displays this effect plot, which was created using the code below. By specifying `lines = list(multiline = TRUE)`, the estimated mean responses for each level of `species` are connected for each discrete value of `body_mass_g`. Figure 4.7 allows use to determine the effect of `species` on `bill_length_mm` when we vary `body_mass_g` over 5 discrete values. When varying `body_mass_g` across the values 2700, 3600, 4500, 5300, and 6300 g, we see greater changes in the estimated mean of `bill_lengh_mm` for Chinstrap penguins in comparison to Adelie and Gentoo penguins.

```
# effect plot of body mass for separate lines model
# axes ... rotates the x-axis labels 90 degrees
# lines ... plots the effect of species in one graphic
plot(predictorEffect("species", lmods),
     axes = list(x = list(rotate = 90)),
     lines=list(multiline = FALSE))
```

We refer the reader to the "Predictor effects gallery" vignette in the **`effects`** package (run `vignette("predictor-effects-gallery", package = "effects")`

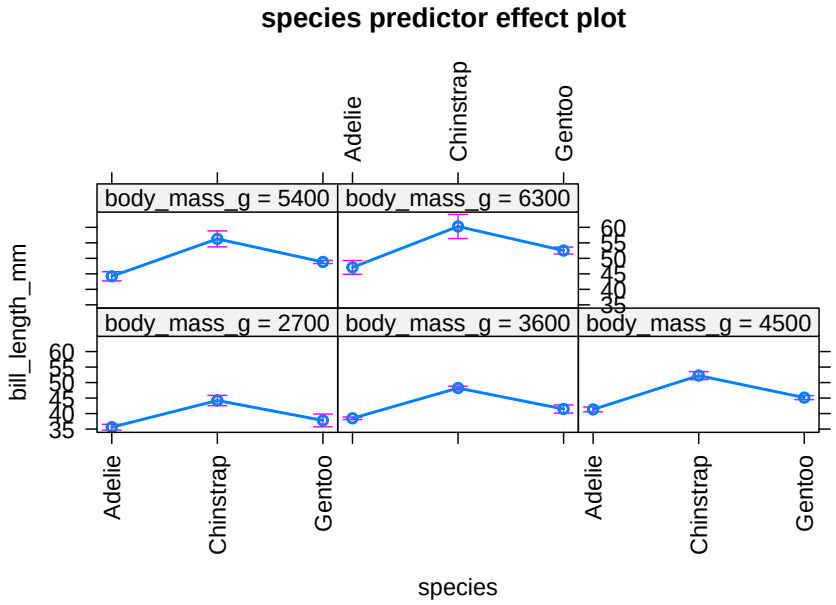


Figure 4.7: Effect plot for species based on the equations in Equation (4.3.4).

in the Console) for more details about how to construct effect plots in different settings.

4.4 Added-variable and leverage plots

An **added-variable plot** or **partial regression plot** displays the marginal effect of a regressor on the response after accounting for the other regressors in the model (Mosteller and Tukey 1977). While an effect plot is a plot of the estimated mean relationship between the response and a focal predictor while holding the model's predictors at typical values, the added-variable plot is a plot of two sets of residuals against one other.

We create an added-variable plot for regressor X_j in the following way:

1. Compute the residuals of the model regressing the response Y on all regressors except X_j. We denote these residuals $\hat{\boldsymbol{\epsilon}}(Y \mid \mathbb{X}_{-j})$. These residuals represent the part of the response variable not explained by the regressors in $\mathbb{X}_{-j}$.
2. Compute the residuals of the model regressing the regressor X_j on all regressors except X_j. We denote these residuals $\hat{\boldsymbol{\epsilon}}(X_j \mid \mathbb{X}_{-j})$. These residuals represent the part of the X_j not explained by the regressors in $\mathbb{X}_{-j}$. Alternatively, these residuals represent the amount of additional information X_j provides after accounting for the regressors in $\mathbb{X}_{-j}$.

3. The added-variable plot for X_j is a plot of $\hat{\boldsymbol{\epsilon}}(Y \mid \mathbb{X}_{-j})$ on the y-axis and $\hat{\boldsymbol{\epsilon}}(X_j \mid \mathbb{X}_{-j})$ on the x-axis.

Added-variable plots allow us to visualize the impact a regressor has when added to an existing regression model. We can use the added-variable plot for X_j to visually estimate the partial slope $\hat{\beta}_j$ (Sheather 2009). In fact, the simple linear regression line that minimizes the RSS for the added-variable plot of X_j will have slope $\hat{\beta}_j$.

We can use an added-variable plot in several ways:

1. To assess the marginal relationship between X_j and Y after accounting for all of the other variables in the model.
2. To assess the strength of this marginal relationship.
3. To identify deficiencies in our fitted model.
4. To identify outliers and observations influential in determining the estimated partial slope.

We focus on points 1 and 2 above, as they are directly related to interpreting our fitted model. We discuss points 3 and 4 in the context of diagnostics for assessing the appropriateness of our model.

In regards to point 1 and 2:

- If the added-variable plot for X_j is essentially a scatter of points with slope zero, then X_j can do little to explain Y after accounting for the other regressors. There is little to gain in adding X_j as an additional regressor to the model regressing Y on $\mathbb{X}_{-j}$. Figure 4.8 (a) provides an example of this situation.
- If the points in an added-variable plot for X_j have a linear relationship, then adding X_j to the model regressing Y on $\mathbb{X}_{-j}$ is expected to improve our model's ability to predict the behavior of Y. The stronger the linear relationship of the points in the added-variable plot, the more important this variable tends to be in our model. Figure 4.8 (b) provides an example of this situation.
- If the points in an added-variable plot for X_j are curved, it indicates that that there is a deficiency in the fitted model (likely because we need to include one or more additional regressors to the model). This is related to point 3 above, but we do not expand on this here. Figure 4.8 (c) provides an example of this situation.

The `car` package (Fox, Weisberg, and Price 2022) provides the `avPlot` and `avPlots` functions for creating added-variable plots. The `avPlot` function will produce an added-variable plot for a single regressor while the `avPlots` function will produce added-variable plots for one or more regressors.

The main arguments to the `avPlot` function are:

- `model`: the fitted `lm` (or `glm`) object.

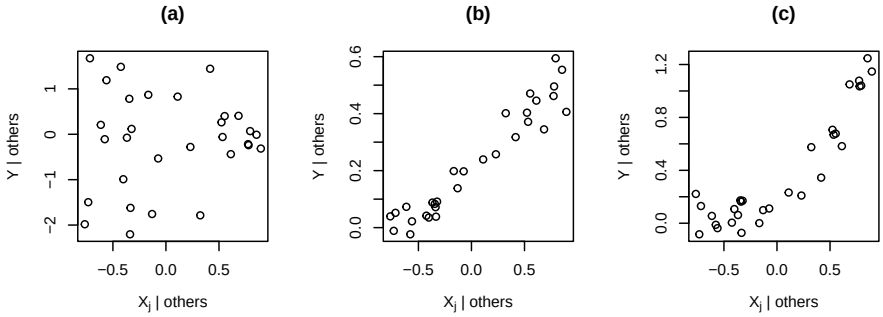


Figure 4.8: Examples of added-variable plots

- `variable`: the regressor for which to create an added-variable plot.
- `id`: a logical value indicating whether unusual observations should be identified. By default, the value is `TRUE`, which means the 2 points with the largest residuals and the 2 points with the largest partial leverage are identified, though this can be customized.

The `avPlots` function replaces the `variable` argument with the `terms` argument. The `terms` argument should be a one-sided formula to indicate the regressors for which you want to construct added-variable plots (one plot for each term). By default, an added-variable plot is created for each regressor. Run `car::avPlot` in the Console for information about about the arguments and details of the `avPlot` and `avPlots` functions.

We now create and interpret added-variable plots for the model regressing `bill_length_mm` on `body_mass_g` and `flipper_length_mm`, which was previously assigned the name `mlmod`. We first load the `car` package and then use the `avPlots` function to construct added-variable plots for `body_mass_g` and `flipper_length_mm`. Figure 4.9 displays the added-variable plots for `body_mass_g` and `flipper_length_mm`. The blue line is the simple linear regression model that minimizes the RSS of the points. The added-variable plot for `body_mass_g` exhibits a weak positive linear relationship between the points. After using the `flipper_length_mm` variable to explain the behavior of `bill_length_mm`, `body_mass_g` likely has some additional explanatory

power, but it doesn't explain a lot of additional response variation. The added-variable plot for `flipper_length_mm` exhibits a slightly stronger positive linear relationship. The `flipper_length_mm` variable seems to have some additional explanatory power when added to the model regressing `bill_length_mm` on `body_mass_g`.

```
library(car)
##
## Attaching package: 'car'
## The following object is masked from 'package:dplyr':
##
##     recode
## The following object is masked from 'package:purrr':
##
##     some
# create added-variable plots for all regressors in mlmod
avPlots(mlmod)
```

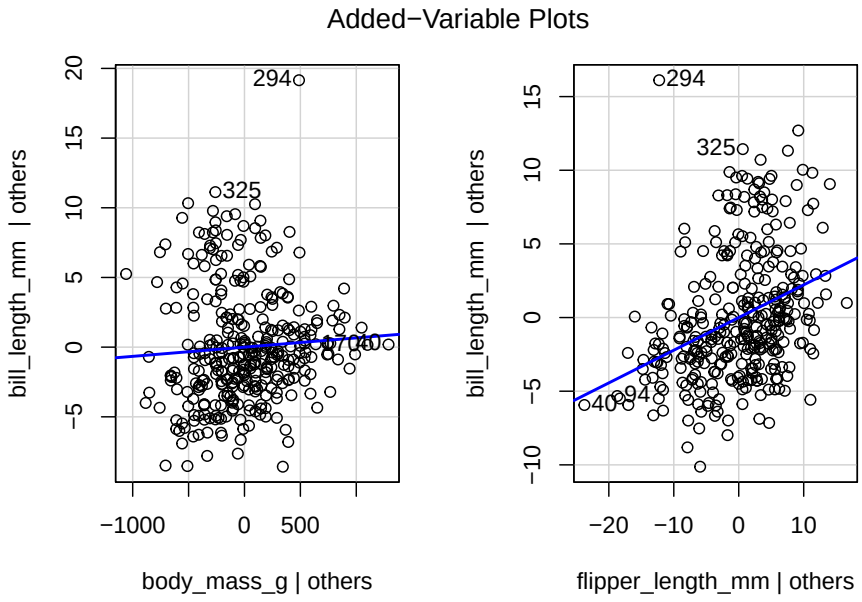


Figure 4.9: The added-variable plots of all regressors for the model regressing `bill_length_mm` on `body_mass_g` and `flipper_length_mm`.

The added-variable plots of fitted models with categorical predictors often show "clusters" of points related to the categorical predictors. These clusters aren't anything to worry about unless the overall pattern of the points is non-linear. We use the code below to create the added-variable plots for all regressors in the

parallel lines model previously fit to the `penguins` data. The fitted model was

$$\begin{aligned}&\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species})\\&= 24.92 + 0.004\texttt{body\_mass\_g} + 9.92D_C + 3.56D_G,\end{aligned}$$

where D_C and D_G are indicator variables for the Chinstrap and Gentoo penguin species (Adelie penguins are the reference species). Figure 4.10 displays the added-variable plots for the `body_mass_g` regressor and the indicator variables for Chinstrap and Gentoo penguins. The added-variable plot for `body_mass_g` has a moderately strong linear relationship, so adding `body_mass_g` to the model regressing `bill_length_mm` on `species` seems to be beneficial. The other two variable plots are show a linear relationship. Clustering patterns are apparent in the added-variable plot for the Chinstrap penguins indicator variable (`speciesChinstrap`) but not for the Gentoo penguins indicator variable.

```
avPlots(lmodp)
```

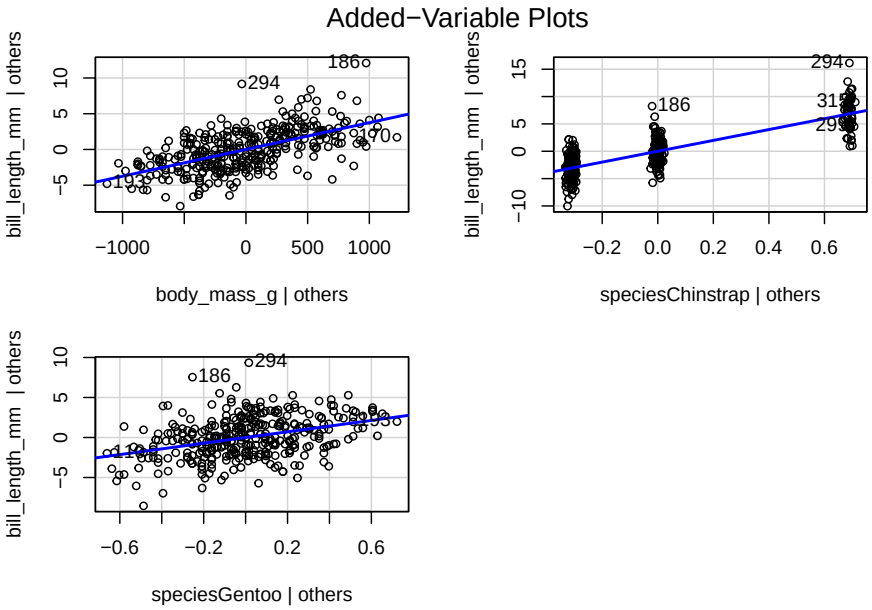


Figure 4.10: The added-variable plots for all regressors in the parallel lines model fit to the `penguins` data.

A challenge in interpreting the added-variable plots of indicator variable regressors is that it often doesn't make sense to talk about the effect of adding a single indicator regressor when all of the other regressors are in the model. Specifically, we either add the categorical *predictor* to our regression model or we do not. When we add a categorical predictor to our model, we simultaneously add $K - 1$ indicator variables as regressors; we do not add the indicator variables

one-at-a-time. In general, we refer to regressors with this behavior as "multiple degrees-of-freedom terms". A categorical variable with 3 or more levels is the most basic of multiple degrees-of-freedom term, but you could also consider regressors related to the interaction between two or more predictors, polynomial regressors, etc.

A **leverage plot** is an extension of the added-variable plot that allows us to visualize the impact of multiple degrees-of-freedom terms. Sall (1990) originally proposed leverage plots to visualize hypothesis tests of linear hypotheses. The interpretation of leverage plots is similar to the interpretation of added-variable plots, though we refer to "predictors" or "terms" instead regressors (which may be combined into one plot). The `leveragePlot` and `leveragePlots` functions in the `car` package produce single or multiple leverage plots, respectively, with arguments similar to the `avPlot` and `avPlots` functions.

We use the code below to create Figure 4.11, which displays leverage plots for the `body_mass_g` and `species` predictors of the parallel lines model previously fit to the `penguins` data. The leverage plot for `body_mass_g` has a moderate linear relationship, so we expect `body_mass_g` to have moderate value in explaining the behavior of `bill_length_mm` after accounting for `species`. Similarly, the points in the leverage plot for `species` have a moderately strong linear relationship, so we expect `species` to have moderate value in explaining the behavior of `bill_length_mm` after accounting for `body_mass_g`.

```
leveragePlots(lmodp)
```

We next examine the leverage plot for the separate lines model fit to the `penguins` data. The fitted separate lines model is

$$\begin{aligned}&\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species})\\ &= 26.99 + 0.003\texttt{body\_mass\_g} + 5.18D_C - 0.25D_G\\ &\quad + 0.001D_C\texttt{body\_mass\_g} + 0.0009D_G\texttt{body\_mass\_g},\end{aligned}$$

which has 6 estimated coefficients. However, the fitted model has only 3 non-intercept terms. Recall the formula we fit for the separate lines model:

```
# function call for separate lines model
lm(formula = bill_length_mm ~ body_mass_g + species + body_mass_g:species,
   data = penguins)
```

Thus, we have terms for `body_mass_g`, `species`, and the interaction term `body_mass_g:species`.

We use the code below to create the leverage plots shown in Figure 4.12. The leverage plot for `body_mass_g` has a moderate linear relationship, so we expect `body_mass_g` to have moderate additional value in explaining the behavior of `bill_length_mm` after accounting for `species` and the interaction term `body_mass_g:species`. It is unlikely we would include the `body_mass_g:species` term in our model prior to including `body_mass_g` , so

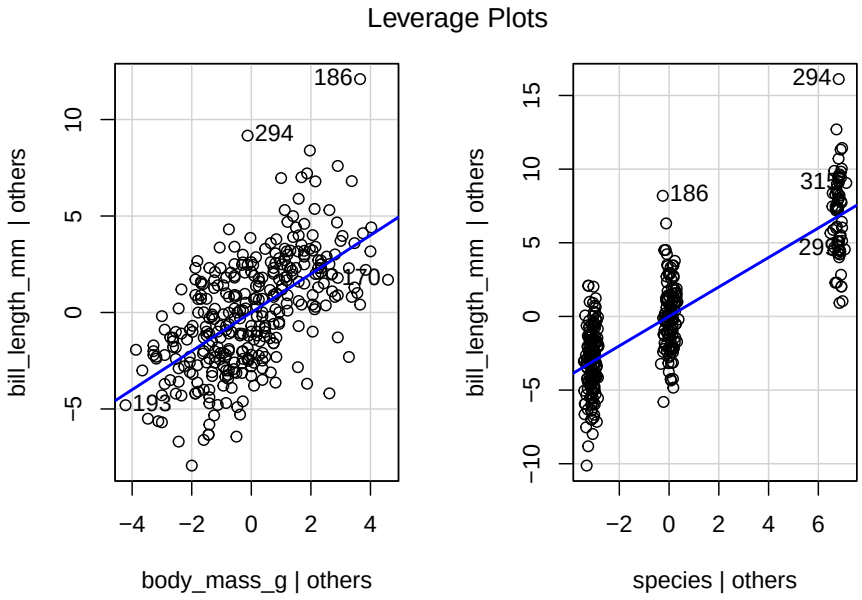


Figure 4.11: Leverage plots for the predictors in the parallel lines model fit to the `penguins` data.

philosophically, this plot provides little useful information. Similarly, interpreting the leverage plot for `species` has limited utility because the leverage plot includes the influence of the interaction term `body_mass_g:species`. We are unlikely to fit a model that includes the interaction term without also including the `species` term directly. Instead it makes more sense to judge the utility of adding `species` to the model regressing `bill_length_mm` on `body_mass_g` alone, which we already considered in Figure 4.11. Examining the leverage plot for the interaction term `body_mass_g:species` , we see the points have only a weak linear relationship. Thus, we expect limited utility in adding the interaction term `body_mass_g:species` to the parallel lines regression model that regresses `bill_length_mm` on `body_mass_g` and `species`.

```
leveragePlots(lmods)
```

4.5 Going deeper

4.5.1 Orthogonality

Let

$$\mathbf{X}_{[j]} = [x_{1,j}, \ldots, x_{n,j}]$$

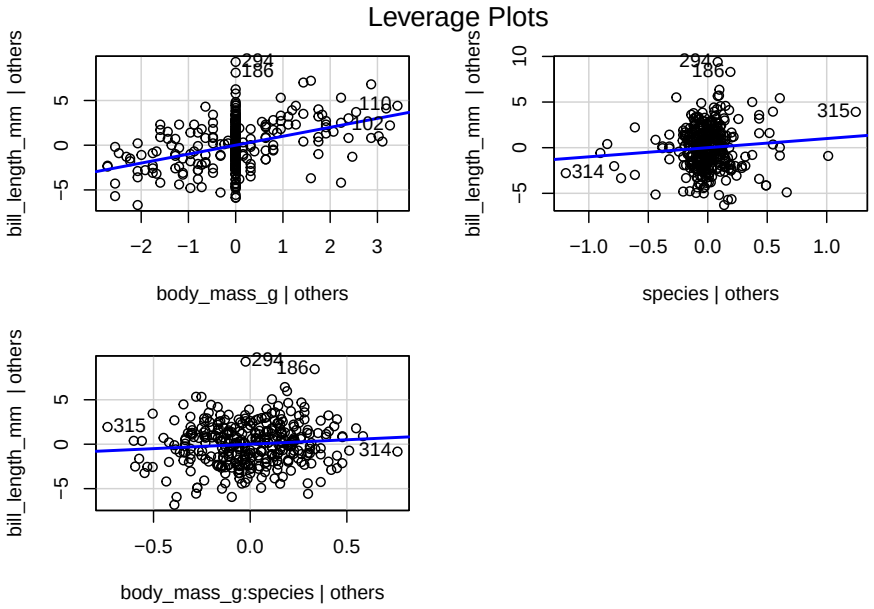


Figure 4.12: Leverage plots for the terms in the separate lines model fit to the `penguins` data.

denote the $n \times 1$ column vector of observed values for regressor X_j. (We can't use the notation $\mathbf{x}_j$ because that is the $p \times 1$ vector of regressor values for the jth observation). Regressors $\mathbf{X}_{[j]}$ and $\mathbf{X}_{[k]}$ are **orthogonal** if $\mathbf{X}_{[j]}^T \mathbf{X}_{[k]} = 0$.

Let $\mathbf{1}_{n\times 1}$ denote an $n \times 1$ column vector of 1s. The definition of orthogonal vectors above implies that $\mathbf{X}_{[j]}$ is orthogonal to $\mathbf{1}_{n\times 1}$ if

$$\mathbf{X}_{[j]}^T \mathbf{1}_{n\times 1} = \sum_{i=1}^{n} x_{i,j} = 0,$$

i.e., if the values in $\mathbf{X}_{[j]}$ sum to zero.

Let $\bar{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{i,j}$ denote the sample mean of $\mathbf{X}_{[j]}$ and $\bar{\mathbf{x}}_j = \bar{x}_j \mathbf{1}_{n\times 1}$ denote the column vector that repeats $\bar{x}_j$ n times.

Centering $\mathbf{X}_{[j]}$ involves subtracting the sample mean of $\mathbf{X}_{[j]}$ from $\mathbf{X}_{[j]}$, i.e., $\mathbf{X}_{[j]} - \bar{\mathbf{x}}_j$.

Regressors $\mathbf{X}_{[j]}$ and $\mathbf{X}_{[k]}$ are **uncorrelated** if they are orthogonal after being centered, i.e., if

$$(\mathbf{X}_{[j]} - \bar{\mathbf{x}}_j)^T (\mathbf{X}_{[k]} - \bar{\mathbf{x}}_k) = 0.$$

Note that the sample covariance between vectors $\mathbf{X}_{[j]}$ and $\mathbf{X}_{[k]}$ is

$$\begin{aligned}\widehat{\text{cov}}(\mathbf{X}_{[j]}, \mathbf{X}_{[k]}) &= \frac{1}{n-1}\sum_{i=1}^{n}(x_{i,j} - \bar{x}_j)(x_{i,k} - \bar{x}_k) \\ &= \frac{1}{n-1}(\mathbf{X}_{[j]} - \bar{\mathbf{x}}_j)^T(\mathbf{X}_{[k]} - \bar{\mathbf{x}}_k).\end{aligned}$$

Thus, two centered regressors are orthogonal if their covariance is zero.

It is a desirable to have orthogonal regressors in your fitted model because they simplify estimating the relationship between the regressors and the response. Specifically:

If a regressor is orthogonal to all other regressors (and the column of 1s) in a model, adding or removing the orthogonal regressor from your model will not impact the estimated regression coefficients of the other regressors.

Since most linear regression models include an intercept, we should assess whether our regressors are orthogonal to other regressors and the column of 1s.

We consider a simple example with $n = 5$ observations to demonstrate how orthogonality of regressors impacts the estimated regression coefficients. In the code below:

- `y` is a vector of response values.
- `ones` is the column vector of 1s.
- `X1` is a column vector of regressor values.
- `X2` is a column vector of regressor values chosen to be orthogonal to `x1` but not to `ones`.
- `X3` is a column vector of regressor values orthogonal to both `x1` and `ones`.
- `X4` is a column vector of regressor values orthogonal to `ones`, `x1`, and `x3`, but not `x2`.
- `X5` is a column vector of regressor values orthogonal to `ones` and `x1`, but not the other regressor vectors.

In the code below, we define vectors `y`, `X1`, and `X2`.

```
y <- c(1, 4, 6, 8, 9)        # create an arbitrary response vector
X1 <- c(7, 5, 5, 7, 7)       # create regressor 1
X2 <- c(-1, 2, -3, 1, 5/7)   # create regressor 2 to be orthogonal to x1
```

Note that the `crossprod` function computes the crossproduct of two vectors or matrices, so that `crossprod(A, B)` computes $\mathbf{A}^T B$, where the vectors or matrices must have the correct dimension for the multiplication to be performed.

The regressor vectors `X1` and `X2` are orthogonal since their crossproduct $\mathbf{X}_{[1]}^T\mathbf{X}_{[2]}$ (in R, `crossprod(X1, X2)`) equals zero, as shown in the code below.

```
# crossproduct is zero, so X1 and X2 are orthogonal
crossprod(X1, X2)
```

```
##      [,1]
## [1,]    0
```

In the code below, we regress `y` on `x1` without an intercept (`lmod1`). The estimated coefficient for `X1` is $\hat{\beta}_1 = 0.893$. Next, we then regress `y` on `X1` and `X2` without an intercept (`lmod2`). The estimated coefficients for `X1` and `X2` are $\hat{\beta}_1 = 0.893$ and $\hat{\beta}_2 = 0.221$, respectively. Because `X1` and `X2` are orthogonal (and because there are no other regressors to consider in the model), the estimated coefficient for `X1` stays the same in both models.

```
# y regressed on X1 without an intercept
lmod1 <- lm(y ~ x1 - 1)
coef(lmod1)
##       x1
## 0.893401
# y regressed on X1 and X2 without an intercept
lmod2 <- lm(y ~ x1 + x2 - 1)
coef(lmod2)
##        x1        x2
## 0.8934010 0.2210526
```

The previous models (`lmod1` and `lmod2`) neglect an important characteristic of a typical linear model: we usually include an intercept coefficient (a columns of 1s as a regressor) in our model. If the regressors are not orthogonal to the column of 1s in our $\mathbf{X}$ matrix, then the coefficients for the other regressors in the model will change when the regressors are added or removed from the model because they are not orthogonal to the column of 1s.

However, neither `X1` nor `X2` is orthogonal with the column of ones. We define the vector `ones` below, which is a column of 1s, and compute the crossproduct between `ones` and the two regressors. Since the crossproducts are not zero, `X1` and `X2` are not orthogonal to the column of ones.

```
ones <- rep(1, 5)   # column of 1s
crossprod(ones, X1) # not zero, so not orthogonal
##      [,1]
## [1,]   31
crossprod(ones, X2) # not zero, so not orthogonal
##            [,1]
## [1,] -0.2857143
```

We create `lmod3` by adding adding a column of ones to `lmod2` (i.e., if we include the intercept in the model). The the coefficients for both `X1` and `X2` change when going from `lmod2` to `lmod3` because these regressors are not orthogonal to the column of 1s. Comparing the coefficients `lmod2` above and `lmod3`, $\hat{\beta}_1$ changes from 0.893 to 0.397 and $\hat{\beta}_2$ changes from 0.221 to 0.279.

```
coef(lmod2) # coefficients for lmod2
##        x1        x2
## 0.8934010 0.2210526
# y regressed on X1 and X2 with an intercept
lmod3 <- lm(y ~ x1 + x2)
coef(lmod3) # coefficients for lmod3
## (Intercept)          x1          x2
##   3.1547101   0.3969746   0.2791657
```

For orthogonality of our regressors to be most impactful, the model's regressors should be orthogonal to each other and the column of 1s. In that context, adding or removing any of the regressors doesn't impact the estimated coefficients of the other regressors. In the code below, we define centered regressors `x3` and `x4` to be uncorrelated, i.e., `X3` and `X4` have sample mean zero and are orthogonal to each other.

```
X3 <-  c(0, -1, 1, 0, 0) # sample mean is zero
X4 <- c(0, 0, 0, 1, -1)  # sample mean is zero
cov(X3, X4)              # 0, so X3 and X4 are uncorrelated and orthogonal
## [1] 0
```

If we fit linear regression models with any combination of `ones`, `X3`, or `X4` as regressors, the associated regression coefficients will not change. To demonstrate this, we consider all possible combinations of the three variables in the models below. We do not run the code to save space, but we summarize the results below.

```
coef(lm(y ~ 1))           # only column of 1s
coef(lm(y ~ x3 - 1))      # only x3
coef(lm(y ~ x4 - 1))      # only x4
coef(lm(y ~ x3))          # 1s and x3
coef(lm(y ~ x4))          # 1s and x4
coef(lm(y ~ x3 + x4 - 1)) # x3 and x4
coef(lm(y ~ x3 + x4))     # 1s, x3, and x4
```

We simply note that in each of the previous models, because all of the regressors (and the column of 1s) are orthogonal to each other, adding or removing any regressor doesn't impact the estimated coefficients for the other regressors in the model. Thus, the estimated coefficients were $\hat{\beta}_0 = 5.6$, $\hat{\beta}_3 = 1.0$, $\hat{\beta}_4 = -0.5$ when the relevant regressor was included in the model.

The easiest way to determine which vectors are orthogonal to each other and the intercept is to compute the crossproduct of the $\mathbf{X}$ matrix for the largest set of regressors you are considering. Consider the matrix of crossproducts for the columns of 1s, `x1`, `x2`, `x3`, and `x4`.

```
crossprod(model.matrix(~ X1 + X2 + X3 + X4))
##             (Intercept)           X1           X2 X3
```

```
## (Intercept)   5.0000000 3.100000e+01 -2.857143e-01  0
## X1           31.0000000 1.970000e+02  1.110223e-16  0
## X2           -0.2857143 1.110223e-16  1.551020e+01 -5
## X3            0.0000000 0.000000e+00 -5.000000e+00  2
## X4            0.0000000 0.000000e+00  2.857143e-01  0
##                    X4
## (Intercept) 0.0000000
## X1          0.0000000
## X2          0.2857143
## X3          0.0000000
## X4          2.0000000
```

Consider the sequence of models below.

```
coef(lm(y ~ 1))
## (Intercept)
##         5.6
```

The model with only an intercept has an estimated coefficient of $\hat{\beta}_{int} = 5.6$. If we add the `X1` to the model with an intercept, then both coefficients change because they are not orthogonal to each other.

```
lmod4 <- lm(y ~ x1) # model with 1s and x1
coef(lmod4)
## (Intercept)          x1
##         2.5         0.5
```

If we add `X2` to `lmod4`, we might think that only $\hat{\beta}_0$ will change because `X1` and `X2` are orthogonal to each other. However, because `X2` is not orthogonal to all of the other regressors in the model (`X1` and the column of 1s), both $\hat{\beta}_0$ and $\hat{\beta}_1$ will change. The easiest way to realize this is to look at `lmod2` above with only `x1` and `x2`. When we add the column of 1s to `lmod2`, both $\hat{\beta}_1$ and $\hat{\beta}_2$ will change because neither regressor is orthogonal to the column of 1s needed to include the intercept term.

```
coef(lm(y ~ x1 + x2))
## (Intercept)          x1          x2
##   3.1547101   0.3969746   0.2791657
```

However, note that `X3` is orthogonal to the column of 1s and `X1`. Thus, if we add `X3` to `lmod4`, which includes both a column of 1s and `X1`, `X3` will not change the estimated coefficients for the intercept or `X1`.

```
coef(lm(y ~ x1 + x3))
## (Intercept)          x1          x3
##         2.5         0.5         1.0
```

Additionally, since `X4` is orthogonal to the column of 1s, `x1`, and `x3`, adding `X4` to the previous model will not change the estimated coefficients for any of the

other variables already in the model.

```
coef(lm(y ~ x1 + x3 + x4))
## (Intercept)          x1          x3          x4
##         2.5         0.5         1.0        -0.5
```

Lastly, if we can partition our $\mathbf{X}$ matrix such that $\mathbf{X}^T\mathbf{X}$ is a block diagonal matrix, then none of the blocks of variables will affect the estimated coefficients of the other variables.

Define a new regressor `X5` below. `X5` is orthogonal to the column of 1s and `X1`, but not `X4`.

```
X5 <- c(1, 0, 0, -1, 0) # orthogonal to ones, x1, not x4
# note block of 0s
crossprod(cbind(ones, X1, X4, X5))
##      ones  X1 X4 X5
## ones    5  31  0  0
## X1     31 197  0  0
## X4      0   0  2 -1
## X5      0   0 -1  2
```

Note the block of zeros in the lower left and upper right corners of the crossproduct matrix above. The block containing `ones` and `X1` is orthogonal to the block containing `X4` and `X5`. This means that if we fit the model with only the column of 1s and `X1`, the model only with `X4` and `X5`, and then fit the model with the column of 1s, `x1`, `x4`, and `x5`, then the coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$ are not impacted when `X4` and `X5` are added to the model. Similarly, $\hat{\beta}_4$ and $\hat{\beta}_5$ are not impacted when the column of 1s and `X1` are added to the model with `X4` and `X5`. See the output below.

```
lm(y ~ x1)           # model with 1s and x1
##
## Call:
## lm(formula = y ~ x1)
##
## Coefficients:
## (Intercept)           x1
##         2.5          0.5
lm(y ~ x4 + x5 - 1)  # model with x4 and x5 only
##
## Call:
## lm(formula = y ~ x4 + x5 - 1)
##
## Coefficients:
## x4  x5
## -3  -5
lm(y ~ x1 + x4 + x5) # model with 1s, x1, x4, x5
```

```
##
## Call:
## lm(formula = y ~ x1 + x4 + x5)
##
## Coefficients:
## (Intercept)           x1           x4           x5
##         2.5          0.5         -3.0         -5.0
```

Chapter 5

Basic theoretical results for linear models

In this chapter we discuss many basic theoretical results for linear models. The results are not interesting by themselves, but they are foundational for the inferential results discussed in Chapter 6. Appendices A and B provide an overview of properties related to matrices and random vectors that are needed for the derivations below.

We assume the responses can be modeled as

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \ldots + \beta_{p-1} x_{i,-1} + \epsilon_i, \quad i = 1, 2, \ldots, n,$$

or using matrix formulation, as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}.$$

using the notation defined in Chapter 3.

5.1 Standard assumptions

We assume that the components of our linear model have the characteristics previously described in Section 3.11.1. For the results we will derive below, we also need to make several specific assumptions about the errors. We have mentioned some of them previously, but discuss them all for completeness.

The first error assumption is that conditional on the regressors, the mean of the errors is zero. This means that $E(\epsilon_i \mid \mathbb{X} = \mathbf{x}_i) = 0$ for $i = 1, 2, \ldots, n$, or using matrix notation,

$$E(\boldsymbol{\epsilon} \mid \mathbf{X}) = 0_{n \times 1},$$

where "$\mid \mathbf{X}$" is notation meaning "conditional on knowing the regressor values for all observations".

We also assume that the errors have constant variances and are uncorrelated, conditional on knowing the regressors, i.e.,

$$\text{var}(\epsilon_i \mid \mathbb{X} = \mathbf{x}_i) = \sigma^2, \quad i = 1, 2, \ldots, n,$$

and

$$\text{cov}(\epsilon_i, \epsilon_j \mid \mathbf{X}) = 0, \quad i, j = 1, 2, \ldots, n, \quad i \neq j.$$

In matrix notation, this is stated as

$$\text{var}(\mathbf{y} \mid \mathbf{X}) = \sigma^2 \mathbf{I}_{n \times n}.$$

Additionally, we assume that the errors are identically distributed. Formally, this may be written as

$$\epsilon_i \sim F, i = 1, 2, \ldots, n,$$

where F is some arbitrary distribution. The $\sim$ is read as "distributed as". In other words, Equation (5.1) means, "ϵ_i is distributed as F for i equal to $1, 2, \ldots, n$". In practice, it is common to assume the errors have a normal (Gaussian) distribution. Two uncorrelated normal random variables are also independent (this is true for normal random variables, but is not generally true for other distributions). Thus, we may concisely state the typical error assumptions as

$$\epsilon_1, \epsilon_2, \ldots, \epsilon_n \mid \mathbf{X} \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2),$$

or using matrix notation as

$$\boldsymbol{\epsilon} \mid \mathbf{X} \sim \mathsf{N}(\mathbf{0}_{n \times 1}, \sigma^2 \mathbf{I}_{n \times n}),$$

where $\mathbf{0}_{n \times 1}$ is the $n \times 1$ vector of zeros and $\mathbf{I}_{n \times n}$ is the $n \times n$ identity matrix. Equation (5.1) combines the following assumptions:

1. $E(\epsilon_i \mid \mathbb{X} = \mathbf{x}_i) = 0$ for $i = 1, 2, \ldots, n$.
2. $\text{var}(\epsilon_i \mid \mathbb{X} = \mathbf{x}_i) = \sigma^2$ for $i = 1, 2, \ldots, n$.
3. $\text{cov}(\epsilon_i, \epsilon_j \mid \mathbf{X}) = 0$ for $i \neq j$ with $i, j = 1, 2, \ldots, n$.
4. ϵ_i has a normal distribution for $i = 1, 2, \ldots, n$.

5.2 Summary of results

For the linear model given in Equation (5) and under the assumptions summarized in Equation (5.1), we have the following results:

1. $\mathbf{y} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\beta}, \sigma^2 \mathbf{I}_{n \times n})$.
2. $\hat{\boldsymbol{\beta}} \mid \mathbf{X} \sim \mathcal{N}(\boldsymbol{\beta}, \sigma^2 (\mathbf{X}^T \mathbf{X})^{-1})$.
3. $\hat{\boldsymbol{\epsilon}} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{0}_{n \times 1}, \sigma^2 (\mathbf{I}_{n \times n} - \mathbf{H}))$.
4. $\hat{\boldsymbol{\beta}}$ has the minimum variance among all unbiased estimators of $\boldsymbol{\beta}$ with the additional assumptions that the model is correct and $\mathbf{X}$ is full-rank.

We prove these results in the sections below. To simplify the derivations below, we let $\mathbf{I} = \mathbf{I}_{n \times n}$ for the duration of this chapter.

5.3 Results for y

Theorem 5.1. *For the linear model given in Equation* (5) *and under the assumptions summarized in Equation* (5.1)*,*

$$E(\mathbf{y} \mid \mathbf{X}) = \mathbf{X}\boldsymbol{\beta}.$$

Proof.

$$\begin{aligned} E(\mathbf{y}|\mathbf{X}) &= E(\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}|\mathbf{X}) && \text{(by definition)} \\ &= E(\mathbf{X}\boldsymbol{\beta}|\mathbf{X}) + E(\boldsymbol{\epsilon}|\mathbf{X}) && \text{(linearity of expectation)} \\ &= E(\mathbf{X}\boldsymbol{\beta}|\mathbf{X}) + \mathbf{0}_{n\times 1} && \text{(by assumption about } \epsilon\text{)} \\ &= \mathbf{X}\boldsymbol{\beta} && \text{(since } \mathbf{X} \text{ and } \boldsymbol{\beta} \text{ are constant)} \end{aligned}$$

□

Theorem 5.2. *For the linear model given in Equation* (5) *and under the assumptions summarized in Equation* (5.1)*,*

$$\text{var}(\mathbf{y} \mid \mathbf{X}) = \sigma^2\mathbf{I}.$$

Proof.

$$\begin{aligned} \text{var}(\mathbf{y}|\mathbf{X}) &= \text{var}(\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}|\mathbf{X}) && \text{(by definition)} \\ &= \text{var}(\boldsymbol{\epsilon}|\mathbf{X}) && (\mathbf{X}\boldsymbol{\beta} \text{ is constant}) \\ &= \sigma^2\mathbf{I}. && \text{(by assumption)} \end{aligned}$$

□

Theorem 5.3. *For the linear model given in Equation* (5) *and under the assumptions summarized in Equation* (5.1)*,*

$$\mathbf{y} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\beta}, \sigma^2\mathbf{I}).$$

Proof. We know that $E(\mathbf{y} \mid \mathbf{X}) = \mathbf{X}\boldsymbol{\beta}$ from Theorem 5.1 and $\text{var}(\mathbf{yX}) = \sigma^2\mathbf{I}$ from Theorem 5.2.

Since $\mathbf{y}$ is a linear function of the multivariate normal vector $\boldsymbol{\epsilon}$, then $\mathbf{y}$ must also have a multivariate normal distribution. □

5.4 Results for $\hat{\boldsymbol{\beta}}$

Theorem 5.4. *For the linear model given in Equation* (5) *and under the assumptions summarized in Equation* (5.1)*, the OLS estimator for* $\boldsymbol{\beta}$*,*

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^T\mathbf{X}^T\mathbf{y},$$

is an unbiased estimator for $\boldsymbol{\beta}$*, i.e.,*

$$E(\hat{\boldsymbol{\beta}} \mid \mathbf{X}) = \boldsymbol{\beta}.$$

Proof. We previously derived the following results,

$$E(\mathbf{y}|\mathbf{X}) = \mathbf{X}\boldsymbol{\beta}$$

$$\text{var}(\mathbf{y}|\mathbf{X}) = \sigma^2\mathbf{I}$$

Then,

$$\begin{aligned}
E(\hat{\boldsymbol{\beta}}|\mathbf{X}) &= E((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}|\mathbf{X}) && \text{(substitute OLS formula)} \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T E(\mathbf{y}|\mathbf{X}) && \text{(factor non-random terms)} \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X} && \text{(above result)} \\
&= \mathbf{I}_{p\times p}\boldsymbol{\beta} && \text{(property of inverse matrices)} \\
&= \boldsymbol{\beta}.
\end{aligned}$$

□

Theorem 5.5. *For the linear model given in Equation* (5) *and under the assumptions summarized in Equation* (5.1)*,*

$$\text{var}(\hat{\boldsymbol{\beta}} \mid \mathbf{X}) = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}.$$

Proof.

$$\begin{aligned}
\text{var}(\hat{\boldsymbol{\beta}}|\mathbf{X}) &= \text{var}((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}|\mathbf{X}) && \text{(by OLS formula)} \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\text{var}(\mathbf{y}|\mathbf{X})((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T)^T && \text{(pull constants out of variance)} \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\text{var}(\mathbf{y}|\mathbf{X})\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} && \text{(simplification)} \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\sigma^2\mathbf{I})\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} && \text{(previous result)} \\
&= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} && (\sigma^2 \text{ is a scalar}) \\
&= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}. && \text{(simplification)}
\end{aligned}$$

□

Theorem 5.6. *For the linear model given in Equation* (5) *and under the assumptions summarized in Equation* (5.1)*,*

$$\hat{\boldsymbol{\beta}} \mid \mathbf{X} \sim \mathcal{N}(\boldsymbol{\beta}, \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}).$$

Proof. Since $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ is a linear combination of $\mathbf{y}$, and $\mathbf{y}$ is a multivariate normal random vector, then $\hat{\boldsymbol{\beta}}$ is also a multivariate normal random vector. Using the previous two results for the expectation and variance,

$$\hat{\boldsymbol{\beta}}|\mathbf{X} \sim N(\boldsymbol{\beta}, \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}).$$

□

5.5 Results for the residuals

The residual vector can be expressed in various equivalent ways, such as

$$\begin{aligned}\hat{\boldsymbol{\epsilon}} &= \mathbf{y} - \hat{\mathbf{y}} \\ &= \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}.\end{aligned}$$

The **hat** matrix is denoted as

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T.$$

Thus, using the substitution $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ and the definition for $\mathbf{H}$ in Equation (5.5), we see that

$$\begin{aligned}\hat{\boldsymbol{\epsilon}} &= \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}} \\ &= \mathbf{y} - \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \\ &= \mathbf{y} - \mathbf{H}\mathbf{y} \\ &= (\mathbf{I} - \mathbf{H})\mathbf{y}.\end{aligned}$$

The hat matrix is an important theoretical matrix, as it projects $\mathbf{y}$ into the space spanned by the vectors in $\mathbf{X}$. It also has some properties that we will exploit in some of the derivations below.

Theorem 5.7. *The hat matrix* $\mathbf{H}$ *is symmetric and idempotent.*

Proof. Notice that,

$$\begin{aligned}
\mathbf{H}^T &= (\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T)^T && \text{(definition of } \mathbf{H}\text{)} \\
&= (\mathbf{X}^T)^T((\mathbf{X}^T\mathbf{X})^{-1})^T\mathbf{X}^T && \text{(apply transpose to matrix product)} \\
&= \mathbf{X}((\mathbf{X}^T\mathbf{X})^T)^{-1}\mathbf{X}^T && \text{(simplification, reversibility of inverse and transpose)} \\
&= \mathbf{X}(\mathbf{X}^T(\mathbf{X}^T)^T)^{-1}\mathbf{X}^T && \text{(apply transpose to matrix product)} \\
&= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T && \text{(simplification)} \\
&= \mathbf{H}.
\end{aligned}$$

Thus, $\mathbf{H}$ is symmetric.

Additionally,

$$\begin{aligned}
\mathbf{HH} &= (\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T)(\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T) && \text{(definition of } \mathbf{H}\text{)} \\
&= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X})(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^t && \text{(associative property of matrices)} \\
&= \mathbf{X}\mathbf{I}_{p\times p}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T && \text{(property of inverse matrices)} \\
&= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T && \text{(simplification)} \\
&= \mathbf{H}.
\end{aligned}$$

Therefore, $\mathbf{H}$ is idempotent. □

Theorem 5.8. *The matrix* $\mathbf{I}-\mathbf{H}$ *is symmetric and idempotent.*

Proof. First, notice that,

$$\begin{aligned}
(\mathbf{I}-\mathbf{H})^T &= \mathbf{I}^T - \mathbf{H}^T && \text{(transpose to matrix sum)} \\
&= \mathbf{I}-\mathbf{H}. && \text{(since } \mathbf{I} \text{ and } \mathbf{H} \text{ are symmetric)}
\end{aligned}$$

Thus, $\mathbf{I}-\mathbf{H}$ is symmetric.

Next,

$$\begin{aligned}
(\mathbf{I}-\mathbf{H})(\mathbf{I}-\mathbf{H}) &= \mathbf{I}-2\mathbf{H}+\mathbf{HH} && \text{(transpose to matrix sum)} \\
&= \mathbf{I}-2\mathbf{H}+\mathbf{H} && \text{(since H is idempotent)} \\
&= \mathbf{I}-\mathbf{H}. && \text{(simplification)}
\end{aligned}$$

Thus, $\mathbf{I}-\mathbf{H}$ is idempotent. □

Theorem 5.9. *For the linear model given in Equation* (5) *and under the assumptions summarized in Equation* (5.1)*,*

$$E(\hat{\boldsymbol{\epsilon}} \mid \mathbf{X}) = \mathbf{0}_{n\times 1}.$$

Proof.

$$\begin{aligned}
E(\hat{\boldsymbol{\epsilon}}|\mathbf{X}) &= E((\mathbf{I}-\mathbf{H})\mathbf{y}|\mathbf{X}) \\
&= (\mathbf{I}-\mathbf{H})E(\mathbf{y}|\mathbf{X}) && (\mathbf{I}-\mathbf{H} \text{ is non-random}) \\
&= (\mathbf{I}-\mathbf{H})\mathbf{X}\boldsymbol{\beta} && \text{(earlier result)} \\
&= \mathbf{X}\boldsymbol{\beta} - \mathbf{X}\boldsymbol{\beta} && \text{(distribute the product)} \\
&= \mathbf{X}\boldsymbol{\beta} - \mathbf{X}^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} && \text{(definition of H)} \\
&= \mathbf{X}\boldsymbol{\beta} - \mathbf{X}\mathbf{I}_{p\times p}\boldsymbol{\beta} && \text{(property of inverse matrix)} \\
&= \mathbf{X}\boldsymbol{\beta} - \mathbf{X}\boldsymbol{\beta} && \text{(simplification)} \\
&= \mathbf{0}_{n\times 1}. && \text{(simplification)}
\end{aligned}$$

□

Theorem 5.10. *For the linear model given in Equation* (5) *and under the assumptions summarized in Equation* (5.1)*,*

$$\text{var}(\hat{\boldsymbol{\epsilon}} \mid \mathbf{X}) = \sigma^2(\mathbf{I}-\mathbf{H}).$$

Proof.

$$\begin{aligned}
\text{var}(\hat{\boldsymbol{\epsilon}}|\mathbf{X}) &= \text{var}((\mathbf{I}-\mathbf{H})\mathbf{y}|\mathbf{X}) \\
&= (\mathbf{I}-\mathbf{H})\text{var}(\mathbf{y}|\mathbf{X})(\mathbf{I}-\mathbf{H})^T && (\mathbf{I}-\mathbf{H} \text{ is nonrandom}) \\
&= (\mathbf{I}-\mathbf{H})\sigma^2(\mathbf{I}-\mathbf{H})^T && \text{(earlier result)} \\
&= \sigma^2(\mathbf{I}-\mathbf{H})(\mathbf{I}-\mathbf{H}) && (\mathbf{I}-\mathbf{H} \text{ is symmetric}) \\
&= \sigma^2(\mathbf{I}-\mathbf{H}). && (\mathbf{I}-\mathbf{H} \text{ is idempotent})
\end{aligned}$$

□

Theorem 5.11. *For the linear model given in Equation* (5) *and under the assumptions summarized in Equation* (5.1)*,*

$$\hat{\boldsymbol{\epsilon}} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{0}_{n\times 1}, \sigma^2(\mathbf{I}-\mathbf{H})).$$

Proof. Since $\hat{\boldsymbol{\epsilon}}$ is a linear combination of multivariate normal vectors, and using previous results, it has mean $\mathbf{0}_{n\times 1}$ and variance matrix $\sigma^2(\mathbf{I}-\mathbf{H})$. □

Theorem 5.12. *The RSS can be represented as,*

$$RSS = \mathbf{y}^T(\mathbf{I}-\mathbf{H})\mathbf{y}.$$

Proof.

$$\begin{aligned} RSS &= \hat{\boldsymbol{\epsilon}}^T \hat{\boldsymbol{\epsilon}} && \text{(matrix representation of RSS)} \\ &= ((\mathbf{I}-\mathbf{H})\mathbf{y})^T(\mathbf{I}-\mathbf{H})\mathbf{y} && \text{(previous result)} \\ &= \mathbf{y}^T(\mathbf{I}-\mathbf{H})^T(\mathbf{I}-\mathbf{H})\mathbf{y} && \text{(apply transpose)} \\ &= \mathbf{y}^T(\mathbf{I}-\mathbf{H})(\mathbf{I}-\mathbf{H})\mathbf{y} && (\mathbf{I}-\mathbf{H} \text{ is symmetric}) \\ &= \mathbf{y}^T(\mathbf{I}-\mathbf{H})\mathbf{y} && (\mathbf{I}-\mathbf{H} \text{ is idempotent}) \end{aligned}$$

□

5.6 The Gauss-Markov Theorem

Suppose we will fit the regression model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}.$$

Assume that

1. $E(\boldsymbol{\epsilon} \mid \mathbf{X}) = 0$.
2. $\text{var}(\boldsymbol{\epsilon} \mid \mathbf{X}) = \sigma^2\mathbf{I}$, i.e., the errors have constant variance and are uncorrelated.
3. $E(\mathbf{y} \mid \mathbf{X}) = \mathbf{X}\boldsymbol{\beta}$
4. $\mathbf{X}$ is a full-rank matrix.

Then the **Gauss-Markov** states that the OLS estimator of $\boldsymbol{\beta}$,

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^T\mathbf{X}^T\mathbf{y},$$

has the minimum variance among all unbiased estimators of $\boldsymbol{\beta}$ and this estimator is unique.

Some comments:

- Assumption 3 guarantees that we have hypothesized the correct model, i.e., that we have included exactly the correct regressors in our model. Not only are we fitting a linear model to the data, but our hypothesized model is actually correct.
- Assumption 4 ensures that the OLS estimator can be computed (otherwise, there is no unique solution).
- The Gauss-Markov theorem only applies to unbiased estimators of $\boldsymbol{\beta}$. Biased estimators could have a smaller variance.
- The Gauss-Markov theorem states that no unbiased estimator of $\boldsymbol{\beta}$ can have a smaller variance than $\hat{\boldsymbol{\beta}}$.
- The OLS estimator uniquely has the minimum variance property, meaning that if an $\tilde{\boldsymbol{\beta}}$ is another unbiased estimator of $\boldsymbol{\beta}$ and $\text{var}(\tilde{\boldsymbol{\beta}}) = \text{var}(\hat{\boldsymbol{\beta}})$, then in fact the two estimators are identical and $\tilde{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}$.

We do not prove this theorem.

Chapter 6

Linear model inference and prediction

6.1 Overview of inference and prediction

In this chapter we will discuss statistical inference and prediction. Inference and prediction are often intertwined, so we discuss them together.

Wasserman (2004) states

> Statistical inference, or "learning" as it is called in computer science, is the process of using data to infer the distribution that generated the data.

In short, statistical inference is the process by which we use a sample of data to draw conclusions about some aspect of the population distribution from which the sample came.

There are two primary types of statistical inference:

1. Confidence intervals
2. Hypothesis tests.

We will discuss both types of inference for linear regression models under standard distributional assumptions. Appendix C provides an overview of both confidence intervals and hypothesis tests in a more general context.

We will also discuss prediction in this chapter. While inference focuses on drawing conclusions about the data-generating distribution, prediction focuses on selecting a plausible value or range of values for an unobserved response. It is common to make predictions using estimated parameters we find as part of the inferential process, though this isn't required.

We will also introduce and discuss solutions for the multiple comparisons problem, which arises when we make multiple inferences or predictions simultaneously.

6.2 Necessary notation

We briefly introduce some notation related to random variables and distributions that we will need in our discussion below.

We let t_ν denote a random variable having a t distribution with ν degrees of freedom. We will use the notation t_ν^α to denote the $1-\alpha$ quantile of a t distribution with ν degrees of freedom. The t distribution is a symmetric bell-shaped distribution like the normal distribution but has a larger standard deviation. As the degrees of freedom of a t random variable increases it behaves more and more similarly to a random variable with a standard normal distribution (a $\mathcal{N}(0,1)$ distribution). Figure 6.1 displays the density of a t distribution with 10 degrees of freedom while also indicating the 0.95 quantile of that distribution. Additional information about the t distribution is available on Wikipedia at https://en.wikipedia.org/wiki/Student%27s_t-distribution.

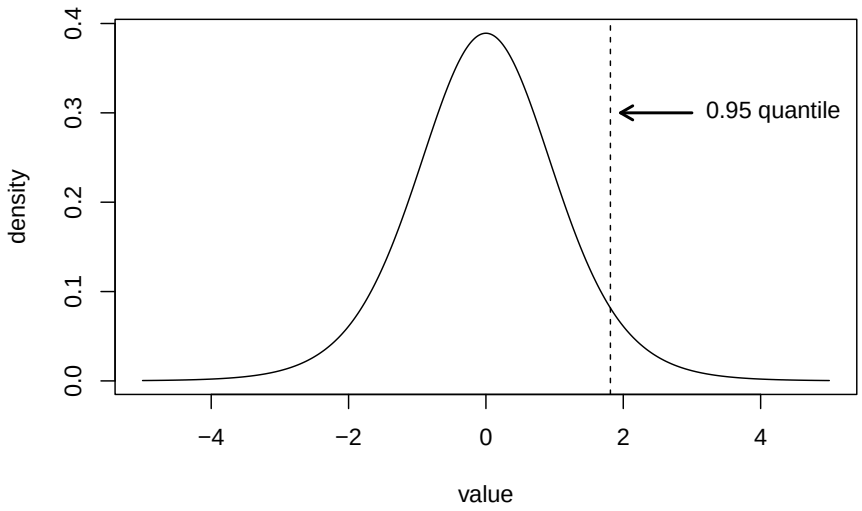


Figure 6.1: The solid line shows the density of a t random variable with 10 degrees of freedom. The dashed vertical line indicates $t_{10}^{0.05}$, the 0.95 quantile of a t distribution with 10 degrees of freedom. The area to the left of the line is 0.95 while the area to the right is 0.05.

We use the notation F_{ν_1,ν_2} to denote a random variable having an F distribution

with ν_1 numerator degrees of freedom and ν_2 denominator degrees of freedom. We let $F^{\alpha}_{\nu_1,\nu_2}$ denote the $1-\alpha$ quantile of an F random variables with ν_1 numerator degrees of freedom and ν_2 denominator degrees of freedom. In fact, $[t_\nu]^2 = F_{1,\nu}$, i.e., the square of a t random variable with ν degrees of freedom is equivalent to an F random variable with 1 numerator degree of freedom and ν denominator degrees of freedom.

6.3 Properties of the OLS estimator

We continue by reviewing some of the properties of $\hat{\boldsymbol{\beta}}$, the OLS estimator of the regression coefficient vector.

We assume that

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

using standard matrix notation.

We also assume that the model in Equation (6.3) is correct (i.e., we have correctly specified the true model that generated the data) and that

$$\boldsymbol{\epsilon} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{0}_{n\times 1}, \sigma^2\mathbf{I}_{n\times n}).$$

This assumption applies to all errors, so we believe that all errors, observed and future, will have mean 0, variance σ^2, will be uncorrelated, and have a normal distribution.

Under these assumptions, we showed in Chapter 5 that

$$\mathbf{y} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\beta}, \sigma^2\mathbf{I}_{n\times n}).$$

and

$$\hat{\boldsymbol{\beta}} \mid \mathbf{X} \sim \mathcal{N}(\boldsymbol{\beta}, \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}).$$

6.4 Parametric confidence intervals for regression coefficients

6.4.1 Standard t-based confidence intervals

Under the assumptions in Equations (6.3) and (6.3), we can prove (though we won't) that

$$\frac{\hat{\beta}_j - \beta_j}{\hat{\text{se}}(\hat{\beta}_j)} \sim t_{n-p}, \quad j = 0, 1, \ldots, p-1,$$

where $\hat{\text{se}}(\hat{\beta}_j) = \hat{\sigma}(\mathbf{X}^T\mathbf{X})^{-1}_{j+1,j+1}$ is the estimated standard error of $\hat{\beta}_j$. Recall that estimated standard error is the estimated standard deviation of the sampling distribution of $\hat{\beta}_j$. Also, recall that the notation $(\mathbf{X}^T\mathbf{X})^{-1}_{j+1,j+1}$ indicates the

element in row $j+1$, column $j+1$, of the matrix $(\mathbf{X}^T\mathbf{X})^{-1}$. Thus, $(\hat{\beta}_j - \beta_j)/\hat{\text{se}}(\hat{\beta}_j)$ is a pivotal quantity with a t distribution, and it can be used to derive a confidence interval

A confidence interval for β_j with confidence level $1-\alpha$ is given by the expression

$$\hat{\beta}_j \pm t_{n-p}^{\alpha/2}\hat{\text{se}}(\hat{\beta}_j), \quad j = 0, 2, \ldots, p-1.$$

It is critical to note that the $1-\alpha$ confidence level refers to the procedure for a single interval, not the family of intervals we can produce for all p coefficients. We discuss this issue in more detail in Section 6.4.2.

The `confint` function returns confidence intervals for the regression coefficients of a fitted model. Technically, the `confint` function is a generic function that has methods for many different object classes, but we only discuss its usage with `lm` objects. The `confint` function has 3 main arguments:

- `object`: a fitted model object. In our case, the object produced by the `lm` function.
- `parm`: a vector of numbers or names indicating the parameters for which we want to construct confidence intervals. By default, confidence intervals are constructed for all parameters.
- `level`: the confidence level desired for the confidence interval. The default value is `0.95`, which will produce 95% confidence intervals.

We once again use the `penguins` data from the **palmerpenguins** package (Horst, Hill, and Gorman 2022) to illustrate what we have learned. Consider the regression model

$$\begin{aligned} &E(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{flipper\_length\_mm}) \\ &= \beta_0 + \beta_1 \texttt{body\_mass\_g} + \beta_2 \texttt{flipper\_length\_mm}. \end{aligned}$$

We estimate the parameters of this model in R using the code below.

```
# load data
data(penguins, package = "palmerpenguins")
# fit model
mlmod <- lm(bill_length_mm ~ body_mass_g + flipper_length_mm, data = penguins)
```

We obtain the 95% confidence intervals for the 3 regression coefficients by running the code below.

```
confint(mlmod)
##                          2.5 %      97.5 %
## (Intercept)       -1.244658e+01 5.573192182
## body_mass_g       -4.534709e-04 0.001777908
## flipper_length_mm  1.582365e-01 0.285494420
```

The 95% confidence interval for the intercept parameter is [-12.45, 5.58]. We are 95% confident that the mean penguin bill length is between -12.25 and 5.58

mm for a penguin with a body mass of 0 g and a flipper length of 0 mm. (This really isn't sensible).

The 95% confidence interval for the `body_mass_g` coefficient is [-0.00046, 0.002]. We are 95% confident the regression coefficient for `body_mass_g` is between -0.00046 and 0.002, assuming the `flipper_length_mm` regressor is also in the model.

If we wanted to get the 90% confidence interval for the `flipper_length_mm` coefficient by itself, we could use either of the commands shown below.

```
# two styles for determining the CI for a single parameter (at a 90% level)
confint(mlmod, parm = 3, level = 0.90)
##                          5 %      95 %
## flipper_length_mm 0.1685112 0.2752197
confint(mlmod, parm = "flipper_length_mm", level = 0.90)
##                          5 %      95 %
## flipper_length_mm 0.1685112 0.2752197
```

We discuss how to "manually" construct these intervals using R in Section 6.8.1.

6.4.2 The multiple comparisons problem

Our linear models typically have multiple regression coefficients, and thus, we typically want to construct confidence intervals for all of the coefficients.

While individual confidence intervals have utility in providing us with plausible values of the unknown coefficients, the confidence level of the procedure described in Section 6.4.1 is only valid for a single interval. Since we are constructing multiple intervals, the simultaneous confidence level of the procedure for the family of intervals is less than $1 - \alpha$. This is an example of the multiple comparisons problem.

A **multiple comparisons problem** occurs anytime we make multiple inferences (confidence intervals, hypothesis tests, prediction intervals, etc.). We are more likely to draw erroneous conclusions if we do not adjust for the fact that we are making multiple inferential statements. E.g., a confidence interval procedure with level 0.95 will produce intervals that contain the target parameter with probability 0.95. If we construct two confidence intervals with level 0.95, then the family-wise confidence level (i.e., the probability that both intervals simultaneously contain their respective target parameters) will be less than 0.95. (We can guarantee that our family-wise confidence level will be at least 0.90, but we can't determine the exact value without more information). In general, the **family-wise confidence level** is the probability that all intervals under consideration simultaneously contain their target parameter. The family-wise confidence level is also known as the **simultaneous** or **overall** confidence level.

A **multiple comparisons procedure** is a procedure designed to adjust for multiple inferences. In the context of confidence intervals, a multiple comparisons

procedure will produce a family of intervals that have a family-wise confidence level above some threshold. We discuss two basic multiple comparisons procedures for confidence intervals below.

6.4.3 Adjusted confidence intervals for regression coefficients

Bonferroni (1936) proposed a simple multiple comparisons procedure that is applicable in many contexts. This general procedure is known as the **Bonferroni correction**. We describe its application below.

Suppose we are constructing k confidence intervals simultaneously. We control the family-wise confidence level of our intervals at $1-\alpha$ if we construct the individual confidence intervals with the level $1-\alpha/k$. We sketch a proof of this below.

Boole's inequality (Boole 1847) states that for a countable set of events $A_1, A_2, A_3 \ldots$,

$$P(\cup_{j=1}^{\infty} A_j) \leq \sum_{j=1}^{\infty} P(A_j).$$

This is a generalization of the fact that

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \leq P(A) + P(B)$$

for two events A and B. We can use Boole's inequality to show that the Bonferroni correction controls the family-wise confidence level of our confidence intervals at $1-\alpha$.

Suppose that we construct a family of k confidence intervals with individual confidence level $1-\alpha/k$ (and all assumptions are satisfied.) Then the probability that the confidence interval procedure for a specific interval doesn't contain the target parameter is α/k. Then

$$\begin{aligned}
&P(\text{All } k \text{ intervals contain the target parameter}) \\
&= 1 - P(\text{At least one of the } k \text{ intervals misses the target parameter}) \\
&= 1 - P(\cup_{j=1}^{k} \text{interval } j \text{ misses the target parameter}) \\
&\geq 1 - \sum_{j=1}^{k} P(\text{interval } j \text{ misses the target parameter}) \\
&= 1 - k(\alpha/k) \\
&= 1 - \alpha.
\end{aligned}$$

Thus, the family-wise confidence level of all k intervals is AT LEAST $1-\alpha$ when the Bonferroni correction is used.

The Bonferroni correction is known to be conservative, which means that the family-wise confidence level is typically much larger than $1-\alpha$. This might

sound like a desirable property, but conservative methods can have low power. In the context of our confidence intervals, this means our intervals are much wider than they need to be, so we aren't able to draw precise conclusions about the plausible values of our regression coefficients.

Let's construct simultaneous confidence intervals for our `penguins` example using the Bonferroni correction. If we want to control the family-wise confidence level of our $k = 3$ intervals at 0.95, then $\alpha = 0.05$ and the Bonferroni correction suggests that we should construct the individual intervals at a confidence level of $1 - 0.05/3 = 0.983$. We construct the Bonferroni-adjusted confidence intervals using the code below.

```
# Simultaneous 95% confidence intervals for mlmod
confint(mlmod, level = 1 - 0.05/3)
##                          0.833 %    99.167 %
## (Intercept)       -1.445714e+01 7.583749811
## body_mass_g       -7.024372e-04 0.002026874
## flipper_length_mm  1.440377e-01 0.299693234
```

Alternatively, we can use the `confint_adjust` function from the **api2lm** package (French 2022) to construct this interval. The `confint_adjust` function works identically to the `confint` function except that it has an additional argument to indicate the type of adjustment to make when constructing the confidence intervals. Specifying `method = "bonferroni"` will produce Bonferroni-corrected intervals, as demonstrated in the code below.

```
library(api2lm)
confint_adjust(mlmod, method = "bonferroni")
##
## Bonferroni-adjusted confidence intervals
##
## Family-wise confidence level of at least 0.95
##
##               term           lwr         upr
##        (Intercept) -1.445714e+01 7.583749811
##        body_mass_g -7.024372e-04 0.002026874
##  flipper_length_mm  1.440377e-01 0.299693234
```

Working and Hotelling (1929) developed another multiple comparisons procedure that can be used to preserve the family-wise confidence level of the intervals at $1 - \alpha$. The Working-Hotelling multiple comparisons procedure is valid for ALL linear combination of the regression coefficients, meaning that we can construct an arbitrarily large number of confidence intervals for linear combinations of the regression coefficients with this procedure and the family-wise confidence level will be at least $1 - \alpha$ (Weisberg 2014).

The Working-Hotelling procedure guarantees that if we construct the individual confidence intervals in the following way, then the family-wise confidence level

will be at least $1 - \alpha$:

$$\hat{\beta}_j \pm \sqrt{pF^{\alpha}_{p,n-p}}\hat{\text{se}}(\hat{\beta}_j), \quad j = 0, 2, \ldots, p-1.$$

The `confint_adjust` function from the **api2lm** package will produce these intervals when setting the `method` argument to `"wh"`. We construct Working-Hotelling-adjusted intervals with family-wise confidence level of at least 0.95 for the `penguins` example using the code below.

```
# 95% family-wise CIs using Working-Hotelling
confint_adjust(mlmod, method = "wh")
##
## Working-Hotelling-adjusted confidence intervals
##
## Family-wise confidence level of at least 0.95
##
##               term           lwr         upr
##        (Intercept) -1.630614e+01 9.432751051
##        body_mass_g -9.313981e-04 0.002255835
##  flipper_length_mm  1.309798e-01 0.312751116
```

In this example, the Bonferroni-adjusted intervals are narrower than the Working-Hotelling-adjusted intervals. The Working-Hotelling intervals tends to be narrower for small p (e.g., $p = 1$ or 2) and small $n - p$ (e.g., $n - p = 1$ or 2) (Mi and Sampson 1993). Additionally, as the number of intervals increases, the Working-Hotelling intervals will eventually be narrower than the Bonferroni-adjusted intervals.

6.5 Prediction: mean response versus new response

It is common to make two types of predictions in a regression context: prediction of a mean response and prediction of a new response. In either context, we want to make predictions with respect to a specific combination of regressor values, which we denote $\mathbf{x}_0$. Using our previous notation, the mean response for a specific combination of regressors is denoted $E(Y \mid \mathbb{X} = \mathbf{x}_0)$. We do not have notation to describe a new response for a specific combination of regressor values, so we will use the notation $Y(\mathbf{x}_0)$.

$E(Y \mid \mathbb{X} = \mathbf{x}_0)$ represents the average response when the regressor values are $\mathbf{x}_0$. Conceptually, this is the number we would get if were able to determine the average of an infinite number of responses with regressor values being fixed at $\mathbf{x}_0$.

$Y(\mathbf{x}_0)$ represents the actual response we will will observe for a new observation with regressor values $\mathbf{x}_0$. Conceptually, we can think of $Y(\mathbf{x}_0)$ as the mean

response for that combination of regressor values plus some error, or more formally,

$$Y(\mathbf{x}_0) = E(Y \mid \mathbb{X} = \mathbf{x}_0) + \epsilon(\mathbf{x}_0),$$

where $\epsilon(\mathbf{x_0})$ denotes the error for our new observation.

Suppose we want to rent a new apartment or buy a new house. If we look through the available listings, we will likely filter our search results by certain characteristics. We might limit our search to dwellings with 3 bedrooms, 2 bathrooms, that are within a certain distance of public transportation, and have a certain amount of square footage. If we averaged the monthly rent or asking price of all the dwellings matching our specifications, then that would be an approximation of the mean response (we would need all the possible dwellings matching those considerations to get the true average). This average would give us an idea of the "typical" price of dwellings with those characteristics. On the other hand, we likely want to know the price of the dwelling we actually end up in. This is the "new response" we want to predict.

Though we can discuss predictions for both the mean response and a new response, it is common to distinguish the two scenarios by using the terminology "estimating the mean response" to refer to prediction of the the mean and "prediction a new response" when we want to predict a new observation. We use this convention in what follows to distinguish the two contexts.

6.6 Parametric confidence interval for the mean response

Consider a typical linear regression model with p regression coefficients given by

$$E(Y|\mathbb{X}) = \beta_0 + \beta_1 X_1 + \ldots \beta_{p-1} X_{p-1}.$$

We want to estimate the mean response for a specific combination of regressor values. The mean response for that combination of regressors is obtained via the equivalent expressions

$$\begin{aligned} E(Y \mid \mathbb{X} = \mathbf{x}_0) &= \beta_0 + \sum_{j=1}^{p-1} x_{0,j}\beta_j \\ &= \mathbf{x}_0^T \boldsymbol{\beta}. \end{aligned}$$

To simplify our notation, we drop the "$\mathbb{X} =$" in our discussion below, so $E(Y \mid \mathbb{X} = \mathbf{x}_0) \equiv E(Y \mid \mathbf{x}_0)$.

What does $E(Y \mid \mathbf{x}_0)$ represent? It represents the average response we will observe if we somehow managed to observe infinitely many responses with $\mathbb{X} = \mathbf{x}_0$.

The Gauss-Markov Theorem discussed in Chapter 3 indicates that the best linear unbiased estimator of the mean response is given by the equation

$$\begin{aligned}\hat{E}(Y \mid \mathbf{x}_0) &= \hat{\beta}_0 + \sum_{j=1}^{p-1} x_{0,j}\hat{\beta}_j \\ &= \mathbf{x}_0^T \hat{\boldsymbol{\beta}},\end{aligned}$$

which replaces the unknown, true coefficients by their OLS estimates.

We want to create a confidence interval for $E(Y \mid \mathbf{x}_0)$. If we divide the estimation error of the mean response, i.e., $E(Y \mid \mathbf{x}_0) - \hat{E}(Y \mid \mathbf{x}_0)$, by its estimated standard deviation, then we obtain a pivotal quantity. More specifically, we have

$$\frac{E(Y \mid \mathbf{x}_0) - \hat{E}(Y \mid \mathbf{x}_0)}{\sqrt{\hat{\text{var}}\left(E(Y \mid \mathbf{x}_0) - \hat{E}(Y \mid \mathbf{x}_0)\right)}} = \frac{E(Y \mid \mathbf{x}_0) - \mathbf{x}_0\hat{\boldsymbol{\beta}}}{\hat{\text{se}}(\mathbf{x}_0\hat{\boldsymbol{\beta}})} \sim t_{n-p},$$

with

$$\hat{\text{se}}(\mathbf{x}_0^T\hat{\boldsymbol{\beta}}) = \hat{\sigma}\sqrt{\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0}.$$

A confidence interval for $E(Y \mid \mathbf{x}_0)$ with confidence level $1-\alpha$ is given by the expression

$$\mathbf{x}_0\hat{\boldsymbol{\beta}} \pm t_{n-p}^{\alpha/2}\hat{\text{se}}(\mathbf{x}_0\hat{\boldsymbol{\beta}}).$$

The `predict` function is a generic function used to make predictions based on fitted models. We can use this function to estimate the mean response for multiple combinations of predictor variables, compute the estimated standard error of each estimate, and obtain confidence intervals for the mean response. The primary arguments to the `lm` method for `predict` are:

- `object`: A fitted model from the `lm` function.
- `newdata`: A data frame of predictor values. All predictors used in the formula used to fit `object` must be provided. Each row contains the predictor values for the mean response we want to estimate. If this is not provided, then the fitted values for each observation are returned.
- `se.fit`: A logical value indicating whether we want to explicitly compute the standard errors of each estimated mean, i.e., $\hat{\text{se}}(\hat{E}(Y \mid \mathbf{x}_0))$ for each estimate.
- `interval`: The type of interval to compute. The default is `none`, meaning no interval is provided. Setting `interval = "confidence"` will return a confidence interval for the mean response associated with each row of `newdata`. Setting `interval = "prediction"` will return a prediction interval for a new response, which we will discuss in the next section.
- `level`: The confidence level of the interval.

Run `?predict.lm` in the Console for additional details about this function.

We will estimate the mean response for the parallel lines model previously fit to the `penguins` data. We do this to emphasize the fact that the `predict` function asks you to specify the values of the *predictor* variables for each estimate you want to make, not the complete set of regressors.

Recall that in Section 3.9, we fit a parallel lines model to the `penguins` data that used both `body_mass_g` and `species` to explain the behavior of `bill_length_mm`. Letting D_C denote the indicator variable for the `Chinstrap` level and D_G denote the indicator variable for the `Gentoo` level, the fitted parallel lines model was

$$\begin{aligned}&\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species})\\&= 24.92 + 0.004\texttt{body\_mass\_g} + 9.92D_C + 3.56D_G.\end{aligned}$$

We fit this model below, assigning it the name `lmodp`.

```
# fit parallel lines model to penguins data
lmodp <- lm(bill_length_mm ~ body_mass_g + species,
            data = penguins)
coef(lmodp)
##       (Intercept)       body_mass_g  speciesChinstrap
##     24.919470977       0.003748497       9.920884113
##    speciesGentoo
##       3.557977539
```

Let's estimate the mean response for the "typical" `body_mass_g` of each `species`. We compute the mean `body_mass_g` of each `species` using the code below, assigning the resulting data frame the name `newpenguins`. We use the `group_by` and `summarize` functions from the **dplyr** package (Wickham, François, et al. 2022) to simplify this process.

```
# mean body_mass_g of each species
newpenguins <- penguins |>
  dplyr::group_by(species) |>
  dplyr::summarize(body_mass_g = mean(body_mass_g, na.rm = TRUE))
newpenguins
## # A tibble: 3 x 2
##   species   body_mass_g
##   <fct>           <dbl>
## 1 Adelie          3701.
## 2 Chinstrap       3733.
## 3 Gentoo          5076.
```

We now have a data frame with variables for the two predictors, `species` and `body_mass_g` , used to fit `lmodp`. Each row of `newpenguins` contains the mean `body_mass_g` for each level of `species` and is suitable for use in the `predict` function.

In the code below, we estimate the mean `flipper_length_mm` based on the fitted model in `lmodp` for the mean `body_mass_g` of each level of `species`. We

also choose to compute the estimated standard errors of each estimate by setting the `se.fit` argument to `TRUE`.

```
# estimate mean and standard error for 3 combinations of predictors
predict(lmodp, newdata = newpenguins, se.fit = TRUE)
## $fit
##        1        2        3
## 38.79139 48.83382 47.50488
##
## $se.fit
##         1         2         3
## 0.1955643 0.2914228 0.2166833
##
## $df
## [1] 338
##
## $residual.scale
## [1] 2.403134
```

An Adelie penguin with a body mass of 3700.662 g is estimated to have a mean flipper length of 38.79 mm with an estimated standard error of 0.196 mm. A Chinstrap penguin with a body mass of 3733.09 g is estimated to have a mean flipper length of 48.83 mm with an estimated standard error of 0.29 mm. A Gentoo penguin with a body mass of 5076.02 g is estimated to have a mean flipper length of 47.50 mm with an estimated standard error of 0.22 mm.

To create each 98% confidence intervals for the mean response for each combination of predictors, we specify `level = 0.98` and `interval = "confidence"` in the `predict` function in the code below.

```
predict(lmodp, newdata = newpenguins, level = 0.98, interval = "confidence")
##        fit      lwr      upr
## 1 38.79139 38.33427 39.24851
## 2 48.83382 48.15264 49.51500
## 3 47.50488 46.99840 48.01136
```

The 98% confidence interval for the mean flipper length of an Adelie penguin with a body mass of 3700.662 g is [38.33, 39.25]. We are 98% confident that the mean flipper length of Adelie penguins with a body mass of 3700.662 is between 38.33 g and 39.25 g. Note: we are constructing a confidence interval for the mean flipper length of ALL Adelie penguins with this body mass, i.e., for

$$E(\texttt{flipper\_length\_mm} \mid \texttt{body\_mass\_g} = 3700.662, \texttt{species} = \texttt{Adelie}),$$

which is an unknown characteristic of the population of all Adelie penguins. Similarly, the 98% confidence interval for the mean flipper length of Chinstrap penguins with a body mass of 3733.09 g is [48.15, 49.52]. Lastly, the mean flipper length of Gentoo penguins with a body mass of 5076.02 g is [46.99, 48.02].

We provide details about manually computing confidence intervals for the mean response in Section 6.8.3.

We are once again faced with a multiple comparisons problem because we are making 3 inferences. To control the family-wise confidence level of our intervals, we can use the Bonferroni or Working-Hotelling corrections previously discussed in Section 6.4.3. Both corrections are implemented in the `predict_adjust` function in the **api2lm** package, which is intended to work identically to the `predict` function, but produces intervals that adjust for multiple comparisons. The only additional argument is `method`, with choices `"none"` (no correction), `"bonferroni"` (Bonferroni adjustment), or `"wh"` (Working-Hotelling adjustment). We produce both types of adjusted intervals in the code below. The Bonferroni-adjusted confidence intervals are slightly narrower in this example.

```
# bonferroni-adjusted confidence intervals for the mean response
predict_adjust(lmodp, newdata = newpenguins, level = 0.98,
               interval = "confidence", method = "bonferroni")
##
## Bonferroni-adjusted confidence intervals
##
## Family-wise confidence level of at least 0.98
##
##        fit      lwr      upr
## 1 38.79139 38.25751 39.32527
## 2 48.83382 48.03826 49.62939
## 3 47.50488 46.91335 48.09641
# working-hotelling-adjusted confidence intervals for the mean response
predict_adjust(lmodp, newdata = newpenguins, level = 0.98,
               interval = "confidence", method = "wh")
##
## Working-Hotelling-adjusted confidence intervals
##
## Family-wise confidence level of at least 0.98
##
##        fit      lwr      upr
## 1 38.79139 38.11858 39.46420
## 2 48.83382 47.83122 49.83642
## 3 47.50488 46.75941 48.25035
```

6.7 Parametric prediction interval for a new response

We once again assume the regression model

$$E(Y|\mathbb{X}) = \beta_0 + \beta_1 X_1 + \ldots \beta_{p-1} X_{p-1}.$$

We want to predict a new response for a specific combination of regressor values, $\mathbf{x}_0$. The new response will be the mean response for that combination of regressors, $E(Y \mid \mathbf{x}_0)$, plus the error for that observation, $\epsilon(\mathbf{x}_0)$, i.e.,

$$Y(\mathbf{x}_0) = E(Y \mid \mathbf{x}_0) + \epsilon(\mathbf{x}_0).$$

Thus, our predicted new response, $\hat{Y}(\mathbf{x})$ is the estimated mean plus the estimated error, i.e.,

$$\hat{Y}(\mathbf{x}_0) = \hat{E}(Y \mid \mathbf{x}_0) + \hat{\epsilon}(\mathbf{x}_0).$$

We have already used $\hat{E}(Y \mid \mathbf{x}_0) = \mathbf{x}_0^T \hat{\boldsymbol{\beta}}$ as the estimator for $E(Y \mid \mathbf{x}_0)$ since it is the best linear unbiased estimator according to the Gauss-Markov theorem. What should we use for $\hat{\epsilon}(\mathbf{x}_0)$? In short, because all of the errors, $\epsilon_1, \epsilon_2, \ldots, \epsilon_n, \epsilon(\mathbf{x}_0)$ are uncorrelated, our best guess is the mean value of the errors, which is zero. This is not true when the errors are correlated, but we do not discuss that in detail. Thus, the best predictor of a new response is

$$\begin{aligned} \hat{Y}(\mathbf{x}_0) &= \hat{E}(Y \mid \mathbf{x}_0) + \hat{\epsilon}(\mathbf{x}_0) \\ &= \mathbf{x}_0^T \hat{\boldsymbol{\beta}} + 0 \\ &= \mathbf{x}_0^T \hat{\boldsymbol{\beta}}. \end{aligned}$$

We want to create a prediction interval for $Y(\mathbf{x}_0)$. A prediction interval is basically the same thing as a confidence interval, but the interval estimator is for a random variable instead of a parameter. If we divide the estimation error of the new response, i.e., $Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0)$, by its estimated standard deviation, $\widehat{\text{sd}}(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0))$, then we obtain a t-distributed pivotal quantity:

$$\frac{Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0)}{\widehat{\text{sd}}(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0))} \sim t_{n-p},$$

where

$$\widehat{\text{sd}}(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0)) = \hat{\sigma}\sqrt{1 + \mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0}.$$

The standard deviation of the prediction error is sometimes known as the **standard error of prediction**, but we do not use that terminology. A detailed derivation of $\widehat{\text{sd}}(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0))$ is provided in Section 6.8.4.

There is a conceptual and mathematical relationship between the standard deviation of the estimation error for the mean response and the prediction error for a new observation. If we compare the expression for $\hat{\text{se}}(\hat{E}(Y \mid \mathbf{x}_0)) - E(Y \mid \mathbf{x}_0))$ in Equation (6.6) and $\widehat{\text{sd}}(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0))$ in Equation (6.7), we see that $\widehat{\text{sd}}(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0))$ has an extra "1 +" under the square root. Because of this, *prediction intervals for a new response are always wider than a confidence interval for the mean response* when considering the same regressor values, $\mathbf{x}_0$, and confidence level. Conceptually, prediction intervals are wider because there are two sources of uncertainty in our prediction: estimating the mean response

and predicting the error of the new response. Estimating the mean response does not require us to predict the error of a new response, so the uncertainty of our estimate is less. We can see that this is formally true through the derivations provided in Section 6.8.4.

A prediction interval for $Y(\mathbf{x}_0)$ with confidence level $1 - \alpha$ is given by the expression

$$\mathbf{x}_0\hat{\boldsymbol{\beta}} \pm t_{n-p}^{\alpha/2}\hat{\text{sd}}(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0)).$$

The `predict` function can be used to create predictions and prediction intervals for a new response in the same way that it can be used to estimate the mean response and produce associated confidence intervals. If the `interval` argument is `"none"`, then predictions for a new response are returned. As we have already seen, $\hat{E}(Y \mid \mathbf{x}_0)$ and $\hat{Y}(\mathbf{x}_0)$ are the same number. If the `interval` argument is `"prediction"`, then the predicted new response and associated prediction interval will be produced for each row of data supplied to the `newdata` argument.

Continuing the example started in Section 6.6, we want to predict the response value for new, unobserved penguins having the observed mean `body_mass_g` for each level of `species`. The fitted model is stored in `lmodp` and the data frame with the predictors for the new responses is stored in `newpenguins`. We print the coefficients of the fitted model and the data frame of new predictors below for clarity.

```
coef(lmodp)
##      (Intercept)      body_mass_g speciesChinstrap
##     24.919470977      0.003748497      9.920884113
##    speciesGentoo
##      3.557977539
newpenguins
## # A tibble: 3 x 2
##   species   body_mass_g
##   <fct>           <dbl>
## 1 Adelie          3701.
## 2 Chinstrap       3733.
## 3 Gentoo          5076.
```

In the code below, we predict the `flipper_length_mm` for new penguins for the predictor values stored in `newpenguins` based on the fitted model in `lmodp`. We also construct the associated 99% prediction intervals.

```
# predict new response and compute prediction intervals
# for 3 combinations of predictors
predict(lmodp, newdata = newpenguins,
        interval = "prediction", level = 0.99)
##        fit      lwr      upr
## 1 38.79139 32.54561 45.03718
```

```
## 2 48.83382 42.56301 55.10464
## 3 47.50488 41.25442 53.75534
```

An new Adelie penguin with a body mass of 3700.662 g is predicted to have a flipper length of 38.79 mm. We are 99% confident that a new Adelie penguins with a body mass of 3700.662 g will have a flipper length between 32.54 mm and 45.04 mm. Similarly, the 99% prediction interval for the flipper length of a new Chinstrap penguin with a body mass of 3733.09 g is [48.56, 55.11]. The flipper length of a new Gentoo penguin with a body mass of 5076.02 g is between 41.25 and 53.75 with a confidence level of 0.99.

Since we are making 3 predictions, our inferences suffer from the multiple comparisons problem. To control the family-wise confidence level of our intervals, we can use the Bonferroni correction with $k = 3$. The Working-Hotelling correction discussed in Section 6.4.3 does not apply to new responses. However, a similar adjustment proposed by Scheffé does apply (Kutner et al. 2005). The prediction interval multiplier $t_{n-p}^{\alpha/2}$ used for a single prediction interval with confidence level $1 - \alpha$ is replaced by $\sqrt{kF_{k,n-p}^{1-\alpha}}$ to control the family-wise confidence level at $1 - \alpha$ for a family of k prediction intervals. Recall that the Working-Hotelling multiplier was $\sqrt{pF_{p,n-p}^{1-\alpha}}$. Thus, the Scheffé multiplier scales with the number of predictions being made while the Working-Hotelling multiplier scales with the number of estimated regression coefficients in the fitted model.

Both the Bonferroni and Scheffé multiple comparisons corrections are implemented in the `predict_adjust` function in the **api2lm** package. In the prediction interval setting, we can choose the correction `method` argument to be `"none"` (no correction), `"bonferroni"` (Bonferroni adjustment), or `"scheffe"` (Scheffe adjustment). We produce both types of adjusted intervals in the code below. The Bonferroni-adjusted confidence intervals are slightly narrower in this example.

```
# bonferroni-adjusted prediction intervals for new responses
predict_adjust(lmodp, newdata = newpenguins, level = 0.99,
               interval = "prediction",
               method = "bonferroni")
##
## Bonferroni-adjusted prediction intervals
##
## Family-wise confidence level of at least 0.99
##
##        fit      lwr      upr
## 1 38.79139 31.66373 45.91905
## 2 48.83382 41.67760 55.99004
## 3 47.50488 40.37188 54.63787
# sheffe-adjusted prediction intervals for new responses
predict_adjust(lmodp, newdata = newpenguins, level = 0.99,
```

```
               interval = "prediction",
               method = "scheffe")
##
## Scheffe-adjusted prediction intervals
##
## Family-wise confidence level of at least 0.99
##
##        fit      lwr      upr
## 1 38.79139 30.60787 46.97492
## 2 48.83382 40.61751 57.05014
## 3 47.50488 39.31523 55.69453
```

6.8 Going deeper

6.8.1 Manual calculation of the standard t-based confidence interval for a regression coefficient

Consider the summary of the fitted linear model below, which summarizes a first-order linear model fit to the `penguins` data earlier in this chapter.

```
summary(mlmod)
##
## Call:
## lm(formula = bill_length_mm ~ body_mass_g + flipper_length_mm,
##     data = penguins)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.8064 -2.5898 -0.7053  1.9911 18.8288
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -3.4366939  4.5805532  -0.750    0.454
## body_mass_g        0.0006622  0.0005672   1.168    0.244
## flipper_length_mm  0.2218655  0.0323484   6.859 3.31e-11
##
## (Intercept)
## body_mass_g
## flipper_length_mm ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.124 on 339 degrees of freedom
##   (2 observations deleted due to missingness)
```

```
## Multiple R-squared:  0.4329, Adjusted R-squared:  0.4295
## F-statistic: 129.4 on 2 and 339 DF,  p-value: < 2.2e-16
```

If we want to manually produce 95% confidence intervals for the true regression coefficients for this model using Equation (6.4.1), then we need to acquire some basic information. We need:

- the estimated regression coefficients
- the degrees of freedom for the fitted model $n - p$
- the $1 - \alpha/2$ quantile of a t random variable with $n - p$ degrees of freedom
- the estimated standard error of the estimated regression coefficients.

The estimated regression coefficients can be extracted from our fitted model, `mlmod`, using the `coef` function. We extract and print the estimated coefficients using the code below while simultaneously assigning the vector the name `betahats`.

```
# extract estimated coefficients from mlmod
(betahats <- coef(mlmod))
##       (Intercept)       body_mass_g flipper_length_mm
##     -3.4366939266      0.0006622186      0.2218654584
```

The degrees of freedom $n - p$ is referred to as the residual degrees of freedom and can be obtained by using the `df.residual` function on the fitted model. Using the code below, we see that the residual degrees of freedom is 339 (this information was also in the summary of the fitted model).

```
df.residual(mlmod)
## [1] 339
```

To construct a 95% confidence interval for our coefficients, we need to determine the 0.975 quantile of a t random variable with 339 degrees of freedom. This can be found using the `qt` function, as is done in the code below. We assign this value the name `mult`. Notice that value is a bit above 1.96, which is the value often seen in introductory statistics courses for confidence intervals based on the standard normal distribution ($\mathcal{N}(0, 1)$ distribution).

```
(mult <- qt(0.975, df = 339))
## [1] 1.966986
```

The estimated standard errors of each coefficient are shown in the summary of the fitted model. They are (approximately) 4.58, 0.00057, and 0.032 and can be obtained by extracting the 2nd column of the `coefficients` element produced by the `summary` function.

```
(sehats <- summary(mlmod)$coefficients[,2])
##       (Intercept)       body_mass_g flipper_length_mm
##      4.5805531903      0.0005672075      0.0323484492
```

A alternative approach is to use 3 step process below:

1. Use the `vcov` function to obtain the estimated variance matrix of $\hat{\boldsymbol{\beta}}$, i.e., $\hat{\text{var}}(\hat{\boldsymbol{\beta}})$.
2. Use the `diag` function to extract the diagonal elements of this matrix, which gives us $\hat{\text{var}}(\hat{\beta}_0), \hat{\text{var}}(\hat{\beta}_1), \ldots, \hat{\text{var}}(\hat{\beta}_{p-1})$.
3. Use the `sqrt` function to calculate the estimated standard errors from the estimated variances of the estimated coefficients.

We use this process in the code below, which returns the same estimated standard errors we previously obtained.

```
# 2nd approach to obtaining estimated standard errors
sqrt(diag(vcov(mlmod)))
##       (Intercept)       body_mass_g flipper_length_mm
##      4.5805531903      0.0005672075      0.0323484492
```

Using the estimated coefficients (`betahats`), the appropriate quantile of the t distribution (`mult`), and the estimated standard errors (`sehats`), we can manually produce the standard t-based confidence intervals using the code below.

```
data.frame(lb = betahats - mult * sehats,
           ub = betahats - mult * sehats)
##                              lb            ub
## (Intercept)       -1.244658e+01 -1.244658e+01
## body_mass_g       -4.534709e-04 -4.534709e-04
## flipper_length_mm  1.582365e-01  1.582365e-01
```

6.8.2 Details about estimation of the mean response

Consider the estimated mean response for a specific combination of regressor values, denoted by $\mathbf{x}_0$, so that

$$\hat{E}(Y \mid \mathbf{x}_0) = \mathbf{x}_0^T \hat{\boldsymbol{\beta}}.$$

Using some of the matrix-related results from Appendix B and the result in Equation (6.3), we can determine that

$$\begin{aligned} \text{var}\left(\hat{E}(Y \mid \mathbf{x}_0)\right) &= \text{var}(\mathbf{x}_0^T \hat{\boldsymbol{\beta}}) \\ &= \mathbf{x}_0^T \text{var}(\hat{\boldsymbol{\beta}}) \mathbf{x}_0 \\ &= \sigma^2 \mathbf{x}_0^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{x}_0. \end{aligned}$$

To get the simplest expression for the confidence interval for $E(Y \mid \mathbf{x}_0)$, we have to make a number of connections that are often glossed over. We discuss them explicitly. Since the error variance, σ^2, in Equation (6.8.2) isn't known, we replace it with the typical estimator $\hat{\sigma}^2 = RSS/(n-p)$ to get

$$\hat{\text{var}}\left(\hat{E}(Y \mid \mathbf{x}_0)\right) = \hat{\sigma}^2 \mathbf{x}_0^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{x}_0.$$

The standard error of an estimator is the standard deviation of the estimator's variance, so we have $\text{se}\left(\hat{E}(Y \mid \mathbf{x}_0)\right) = \sqrt{\text{var}\left(\hat{E}(Y \mid \mathbf{x}_0)\right)}$. Similarly, we have that the estimated standard error for $\hat{E}(Y \mid \mathbf{x}_0)$ is $\hat{\text{se}}\left(\hat{E}(Y \mid \mathbf{x}_0)\right) = \sqrt{\hat{\text{var}}\left(\hat{E}(Y \mid \mathbf{x}_0)\right)}$. Taking the square root of Equation (6.8.2), we see that

$$\hat{\text{se}}\left(\hat{E}(Y \mid \mathbf{x}_0)\right) = \hat{\sigma}\sqrt{\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0}.$$

Additionally, since $E(Y \mid \mathbf{x}_0)$ is an (unknown) constant, we also have that

$$\text{var}\left(E(Y \mid \mathbf{x}_0) - \hat{E}(Y \mid \mathbf{x}_0)\right) = \text{var}\left(\hat{E}(Y \mid \mathbf{x}_0)\right).$$

If we divide the estimation error of the mean response, i.e., $E(Y \mid \mathbf{x}_0) - \hat{E}(Y \mid \mathbf{x}_0)$, by its estimated standard deviation, then we obtain a pivotal quantity. More specifically, we have

$$\frac{E(Y \mid \mathbf{x}_0) - \hat{E}(Y \mid \mathbf{x}_0)}{\sqrt{\hat{\text{var}}\left(E(Y \mid \mathbf{x}_0) - \hat{E}(Y \mid \mathbf{x}_0)\right)}} = \frac{E(Y \mid \mathbf{x}_0) - \mathbf{x}_0\hat{\boldsymbol{\beta}}}{\hat{\text{se}}(\mathbf{x}_0\hat{\boldsymbol{\beta}})} \sim t_{n-p}.$$

6.8.3 Manual calculation of confidence intervals for the mean response

We discuss manual computation of the estimated mean response and associated confidence interval for a particular combination of regressor values based on the `penguins` example discussed in Section 6.6.

Specifically, we estimate the mean response for the fitted parallel lines model given by

$$\begin{aligned}&\hat{E}(\texttt{bill\_length\_mm} \mid \texttt{body\_mass\_g}, \texttt{species})\\ &= 24.92 + 0.004\texttt{body\_mass\_g} + 9.92D_C + 3.56D_G,\end{aligned}$$

where D_C and D_G denote the indicator variables for the `Chinstrap` and `Gentoo` levels of the `species` variable. This fitted model is stored in `lmodp`. The estimated coefficients are shown in the code below.

```
coef(lmodp)
##      (Intercept)      body_mass_g speciesChinstrap
##     24.919470977      0.003748497      9.920884113
##    speciesGentoo
##      3.557977539
```

We want to estimate the mean response for the following combination of predictors stored in the `newpenguins` data frame, which is printed in the code below.

```
# mean body_mass_g of each species
newpenguins
## # A tibble: 3 x 2
##   species   body_mass_g
##   <fct>           <dbl>
## 1 Adelie          3701.
## 2 Chinstrap       3733.
## 3 Gentoo          5076.
```

We want to use the `newpenguins` data frame to generate the matrix of regressors used to estimate the associated mean responses. We can create the matrix of regressors using the `model.matrix` function. The main arguments of `model.matrix` are:

- `object`: an object of the appropriate class. In our case, it is a formula or fitted model.
- `data`: a data frame with the predictors needed to construct the matrix.

We need only the right side of the formula used to fit the model in `lmodp` to create our matrix of regressor values. We can use the `formula` function to extract the formula used to fit `lmodp`. We then use `model.matrix` to create the matrix of regressor by values needed for estimating the mean. The matrix produced by `model.matrix`, `X0`, includes a column for the intercept term and the indicator variables in `lmodp`.

```
# determine formula used to fit lmodp
formula(lmodp)
## bill_length_mm ~ body_mass_g + species
# create matrix of regressor values from newpenguins
(X0 <- model.matrix(~ body_mass_g + species, data = newpenguins))
##   (Intercept) body_mass_g speciesChinstrap speciesGentoo
## 1           1    3700.662                0             0
## 2           1    3733.088                1             0
## 3           1    5076.016                0             1
## attr(,"assign")
## [1] 0 1 2 2
## attr(,"contrasts")
## attr(,"contrasts")$species
## [1] "contr.treatment"
```

We can obtain the estimated mean by taking the product of `X0` and the estimated coefficients for `lmodp`. We assign the estimated mean responses the name `est_means`.

```
(est_means <- X0 %*% coef(lmodp))
##       [,1]
## 1 38.79139
## 2 48.83382
```

```
## 3 47.50488
```

We now use Equation (6.6) to get the estimated standard error of each estimated mean response. First, we use `model.matrix` to extract the original matrix of regressors, $\mathbf{X}$, from the fitted model `lmodp`. The `sigma` function is used to extract $\hat{\sigma}$ from `lmodp`. Each *row* of `X0` contains a particular instance regressor values. In the code below, we extract each row of `X0`, and then use Equation (6.6) to compute the estimated standard error associated with each estimated mean response; these are assigned the names `se1`, `se2`, and `se3`. That approach isn't scalable, so we provide a scalable version of these computations that is assigned the name `sehat`.

```
# original matrix of regressors
X <- model.matrix(lmodp)
sigmahat <- sigma(lmodp)
# compute estimated standard error for each estimated mean response
# crossprod(X) = t(X) %*% X
(sehat1 <- sigmahat * sqrt(t(X0[1,]) %*% solve(crossprod(X), X0[1,])))
##           [,1]
## [1,] 0.1955643
(sehat2 <- sigmahat * sqrt(t(X0[2,]) %*% solve(crossprod(X), X0[2,])))
##           [,1]
## [1,] 0.2914228
(sehat3 <- sigmahat * sqrt(t(X0[3,]) %*% solve(crossprod(X), X0[3,])))
##           [,1]
## [1,] 0.2166833
# scalable computation of estimated standard errors
(sehat <- sigmahat * sqrt(diag(X0 %*% solve(crossprod(X), t(X0)))))
##         1         2         3
## 0.1955643 0.2914228 0.2166833
```

To finish the computation of our confidence intervals, we determine the correct multiplier, $t_{n-p}^{\alpha/2}$. For a 98% confidence interval, $\alpha = 0.02$ and $\alpha/2 = 0.01$. The degrees of freedom, $n - p$, is 338 (as shown in the code below). So the multiplier can be represented as $t_{338}^{0.01}$, which is the 0.99 quantile of a t distribution with 338 degrees of freedom. We provide this information to the `qt` function, which provides the quantiles of a t distribution, using the code below to get the correct multiplier.

```
# degrees of freedom, n-p
df.residual(lmodp)
## [1] 338
# multiplier for confidence interval
(mult <- qt(0.99, df = df.residual(lmodp)))
## [1] 2.337431
```

Thus, our 98% confidence intervals for each mean response can be computed

using the code below, which matches the results we obtained in Section 6.6.

```
data.frame(lb = est_means - mult * sehat,
           ub = est_means + mult * sehat)
##         lb       ub
## 1 38.33427 39.24851
## 2 48.15264 49.51500
## 3 46.99840 48.01136
```

6.8.4 Details about prediction interval for a new response

We want to predict the value of a new response for a specific combination of regressor values, $\mathbf{x}_0$. The value of the new response is denoted by $Y(\mathbf{x}_0)$ and its prediction by $\hat{Y}(\mathbf{x}_0)$.

In Section 6.7, we briefly discussed that the new response may be written as

$$Y(\mathbf{x}_0) = E(Y \mid \mathbf{x}_0) + \epsilon(\mathbf{x}_0),$$

and that the predicted new response, under our standard assumptions, is given by

$$\hat{Y}(\mathbf{x}_0) = \mathbf{x}_0^T \hat{\boldsymbol{\beta}}.$$

Using these relationships, we can determine that the variance of the prediction error for a new response is given by

$$\begin{aligned} \text{var}\left(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0)\right) &= \text{var}(Y(\mathbf{x}_0) - \mathbf{x}_0^T \hat{\boldsymbol{\beta}}) \\ &= \text{var}(\mathbf{x}_0^T \boldsymbol{\beta} + \epsilon(\mathbf{x}_0) - \mathbf{x}_0^T \hat{\boldsymbol{\beta}}) \end{aligned}$$

Recall from Appendix B that the variance of a constant plus a random variable is equal to the variance of the random variable (since a constant doesn't vary!). Thus, Equation (6.8.4) simplifies to

$$\text{var}\left(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0)\right) = \text{var}(\epsilon(\mathbf{x}_0) - \mathbf{x}_0^T \hat{\boldsymbol{\beta}}).$$

Using results from Appendix B related to the variance of a sum of random variables, we have that

$$\begin{aligned} &\text{var}\left(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0)\right) \\ &= \text{var}(\epsilon(\mathbf{x}_0) - \mathbf{x}_0^T \hat{\boldsymbol{\beta}}) \\ &= \text{var}\left(\epsilon(\mathbf{x}_0)\right) + \text{var}(-\mathbf{x}_0^T \hat{\boldsymbol{\beta}}) + 2\text{cov}\left(\epsilon(\mathbf{x}_0), -\mathbf{x}_0^T \hat{\boldsymbol{\beta}}\right) \\ &= \text{var}\left(\epsilon(\mathbf{x}_0)\right) + (-1)^2 \text{var}(\mathbf{x}_0^T \hat{\boldsymbol{\beta}}) - 2\text{cov}\left(\epsilon(\mathbf{x}_0), \mathbf{x}_0^T \hat{\boldsymbol{\beta}}\right) \\ &= \text{var}\left(\epsilon(\mathbf{x}_0)\right) + \text{var}(\mathbf{x}_0^T \hat{\boldsymbol{\beta}}) - 2\text{cov}\left(\epsilon(\mathbf{x}_0), \mathbf{x}_0^T \hat{\boldsymbol{\beta}}\right). \end{aligned}$$

The covariance term in the final line of Equation (6.8.4) is 0 because $\epsilon(\mathbf{x}_0)$ and $\mathbf{x}_0^T\hat{\boldsymbol{\beta}}$ are uncorrelated. Why are they uncorrelated? Recall that $\epsilon(\mathbf{x}_0), \epsilon_1, \ldots, \epsilon_n$ are uncorrelated. Also, recall that $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. Since $\mathbf{y} = [Y_1, Y_2, \ldots, Y_n]$ and $Y_i = \mathbf{x}_i^T\boldsymbol{\beta} + \epsilon_i$ for $i = 1, 2, \ldots, n$, the "randomness" of $\mathbf{x}_0^T\hat{\boldsymbol{\beta}}$ comes from $\epsilon_1, \epsilon_2, \ldots, \epsilon_n$. Since $\epsilon(\mathbf{x}_0)$ is uncorrelated with $\epsilon_1, \epsilon_2, \ldots, \epsilon_n$, we conclude that $\epsilon(\mathbf{x}_0)$ and $\mathbf{x}_0^T\hat{\boldsymbol{\beta}}$ are uncorrelated, so their covariance is zero. Thus, we have

$$\text{var}\left(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0)\right) = \text{var}\left(\epsilon(\mathbf{x}_0)\right) + \text{var}(\mathbf{x}_0^T\hat{\boldsymbol{\beta}}).$$

From our assumptions in Section 6.3, we have that $\text{var}(\epsilon(\mathbf{x}_0)) = \sigma^2$. We determined in Section 6.8.2 that

$$\text{var}(\mathbf{x}_0^T\hat{\boldsymbol{\beta}}) = \sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0.$$

Using these two facts, we can conclude that

$$\begin{aligned}\text{var}\left(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0)\right) &= \sigma^2 + \sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0 \\ &= \sigma^2\left(1 + \mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0\right).\end{aligned}$$

Replacing σ^2 by the typical estimator $\hat{\sigma}^2 = RSS/(n-p)$ to get the estimated variance of the prediction error and taking the square root of the estimated variance to get the estimated standard deviation of the prediction error, we have that

$$\widehat{\text{sd}}\left(Y(\mathbf{x}_0) - \hat{Y}(\mathbf{x}_0)\right) = \hat{\sigma}\sqrt{1 + \mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0}.$$

Appendix A

Overview of matrix facts

In this chapter we provide an overview of vectors, matrices, and matrix operations that are useful for data modeling (though their application will not be discussed here).

A **matrix** is a two-dimensional array of values, symbols, or other objects (depending on the context). We will assume that our matrices contain numbers or random variables. Context will make it clear which is being represented.

A.1 Notation

Matrices are commonly denoted by bold capital letters like $\mathbf{A}$ or $\mathbf{B}$, but this will sometimes be simplified to capital letters like A or B. A matrix $\mathbf{A}$ with m rows and n columns (an $m \times n$ matrix) will be denoted as

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{2,1} & \cdots & \mathbf{A}_{1,n} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,1} & \cdots & \mathbf{A}_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{m,1} & \mathbf{A}_{m,2} & \cdots & \mathbf{A}_{m,n} \end{bmatrix},$$

where $\mathbf{A}_{i,j}$ denotes the element in row i and column j of matrix $\mathbf{A}$.

A **column vector** is a matrix with a single column. A **row vector** is a matrix with a single row.

- Vectors are commonly denoted with bold lowercase letters such as $\mathbf{a}$ or $\mathbf{b}$, but this may be simplified to lowercase letters such as a or b.

A $p \times 1$ column vector $\mathbf{a}$ may constructed as

$$\mathbf{a} = [a_1, a_2, \ldots, a_p] = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix}.$$

A vector is assumed to be a column vector unless otherwise indicated.

A.2 Basic mathematical operations

A.2.1 Addition and subtraction

Consider matrices $\mathbf{A}$ and $\mathbf{B}$ with identical sizes $m \times n$.

We add $\mathbf{A}$ and $\mathbf{B}$ by adding the element in position i, j of $\mathbf{B}$ with the element in position i, j of A, i.e.,

$$(\mathbf{A} + \mathbf{B})_{i,j} = \mathbf{A}_{i,j} + \mathbf{B}_{i,j}.$$

Similarly, if we subtract $\mathbf{B}$ from matrix $\mathbf{A}$, then we subtract the element in position i, j of $\mathbf{B}$ from the element in position i, j of $\mathbf{A}$, i.e.,

$$(\mathbf{A} - \mathbf{B})_{i,j} = \mathbf{A}_{i,j} - \mathbf{B}_{i,j}.$$

Example:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 & 9 & 1 \\ 1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 11 & 4 \\ 5 & 8 & 7 \end{bmatrix}.$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} - \begin{bmatrix} 2 & 9 & 1 \\ 1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -7 & 2 \\ 3 & 2 & 5 \end{bmatrix}.$$

A.2.2 Scalar multiplication

A matrix multiplied by a scalar value $c \in \mathbb{R}$ is the matrix obtained by multiplying each element of the matrix by c. If $\mathbf{A}$ is a matrix and $c \in \mathbb{R}$, then

$$(c\mathbf{A})_{i,j} = c\mathbf{A}_{i,j}.$$

Example:

$$3 \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 3 \cdot 1 & 3 \cdot 2 & 3 \cdot 3 \\ 3 \cdot 4 & 3 \cdot 5 & 3 \cdot 6 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \end{bmatrix}.$$

A.2.3 Matrix multiplication

Consider two matrices $\mathbf{A}$ and $\mathbf{B}$. The matrix product $\mathbf{AB}$ is only defined if the number of columns in $\mathbf{A}$ matches the number of rows in $\mathbf{B}$.

Assume $\mathbf{A}$ is an $m \times n$ matrix and $\mathbf{B}$ is an $n \times p$ matrix. $\mathbf{AB}$ will be an $m \times p$ matrix and

$$(\mathbf{AB})_{i,j} = \sum_{k=1}^{n} \mathbf{A}_{i,k}\mathbf{B}_{k,j}.$$

Example:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3 & 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 \\ 4 \cdot 1 + 5 \cdot 2 + 6 \cdot 3 & 4 \cdot 4 + 5 \cdot 5 + 6 \cdot 6 \end{bmatrix} = \begin{bmatrix} 14 & 32 \\ 32 & 77 \end{bmatrix}.$$

A.2.4 Transpose

The **transpose** of a matrix $\mathbf{A}$, denoted $\mathbf{A}^T$, exchanges the rows and columns of the matrix. More formally, the i, j element of $\mathbf{A}^T$ is the j, i element of $\mathbf{A}$, i.e., $(\mathbf{A}^T)_{i,j} = \mathbf{A}_{j,i}$.

Example:

$$\begin{bmatrix} 2 & 9 & 3 \\ 4 & 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 2 & 4 \\ 9 & 5 \\ 3 & 6 \end{bmatrix}.$$

A.3 Basic mathematical properties

A.3.1 Associative property

Addition and multiplication satisfy the associative property for matrices. Assuming that the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ have the sizes required to do the operations below, then

$$(\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C})$$

and

$$(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC}).$$

A.3.2 Distributive property

Matrix operations satisfy the distributive property. Assuming that the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ have the sizes required to do the operations below, then

$$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC} \quad \text{and} \quad (\mathbf{A} + \mathbf{B})\mathbf{C} = \mathbf{AC} + \mathbf{BC}.$$

A.3.3 No commutative property

In general, matrix multiplication does not satisfy the commutative property, i.e.,

$$\mathbf{AB} \neq \mathbf{BA},$$

even when the matrix sizes allow the operation to be performed.

Example:

$$\begin{bmatrix}1 & 2\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}5\end{bmatrix}$$

while

$$\begin{bmatrix}1\\2\end{bmatrix}\begin{bmatrix}1 & 2\end{bmatrix} = \begin{bmatrix}1 & 2\\2 & 4\end{bmatrix}.$$

A.3.4 Transpose-related properties

Assume that the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ have the sizes required to perform the operations below. Additionally, assume that $c \in \mathbb{R}$ is a scalar constant.

The following properties are true:

- $c^T = c$.
- $(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$.
- $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$, which can be extended to $(\mathbf{ABC})^T = \mathbf{C}^T\mathbf{B}^T\mathbf{A}^T$, etc.
- $(\mathbf{A}^T)^T = \mathbf{A}$.

A.4 Special matrices

A.4.1 Square matrices

A matrix is **square** if the number of rows equals the number of columns.

The **diagonal elements** of an $n \times n$ square matrix $\mathbf{A}$ are the elements $\mathbf{A}_{i,i}$ for $i = 1, 2, \ldots, n$. Any non-diagonal elements of $\mathbf{A}$ are called **off-diagonal** elements.

A.4.2 Identity matrix

The $n \times n$ identity matrix $\mathbf{I}_{n\times n}$ is 1 for its diagonal elements and 0 for its off-diagonal elements. Context often makes it clear what the dimensions of an identity matrix are, so $\mathbf{I}_{n\times n}$ is often simplified to $\mathbf{I}$ or I.

Example:

$$\mathbf{I}_{3\times 3} = \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}.$$

A.4.3 Diagonal matrices

A square matrix $\mathbf{A}$ is **diagonal** if all its off-diagonal elements are zero. A 3×3 diagonal matrix and will look something like

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 5 \end{bmatrix},$$

where the non-zero values coudl be replaced by any real number.

A.4.4 Symmetric matrices

A matrix $\mathbf{A}$ is **symmetric** if $\mathbf{A} = \mathbf{A}^T$, i.e., $\mathbf{A}_{i,j} = \mathbf{A}_{j,i}$ for all potential i, j.

A symmetric matrix must be square.

A.4.5 Idempotent matrices

A matrix $\mathbf{A}$ is **idempotent** if $\mathbf{AA} = \mathbf{A}$.

An idempotent matrix must be square.

A.4.6 Positive definite matrices

A matrix $\mathbf{A}$ is positive definite if

$$\mathbf{a}^T\mathbf{Aa} > 0,$$

for every vector of real values $\mathbf{a}$ whose values are not identically 0.

A.4.7 Inverse matrix

An $n \times n$ matrix $\mathbf{A}$ is invertible if there exists a matrix $\mathbf{B}$ such that $\mathbf{AB} = \mathbf{BA} = \mathbf{I}_{n\times n}$. The inverse of $\mathbf{A}$ is denoted $\mathbf{A}^{-1}$.

Inverse matrices only exist for square matrices.

Some other properties related to the inverse operator:

- If $n \times n$ matrices $\mathbf{A}$ and $\mathbf{B}$ are invertible then $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$.
- If $\mathbf{A}$ is invertible then $(\mathbf{A}^{-1})^T = (\mathbf{A}^T)^{-1}$.

A.5 Matrix derivatives

We start with some basic calculus results.

Let $f(y)$ be a function of a scalar value b and $\frac{df(y)}{dy}$ denote the derivative of the function with respect to y. Assume $c \in \mathbb{R}$ is a constant. Then the results in Table A.1 are true.

Table A.1: Some basic calculus results for scalar functions taking scalar inputs.

$f(y)$	$\frac{df(y)}{dy}$
cy	c
y^2	$2y$
cy^2	$2cy$

Table A.2: Some basic calculus results for scalar functions taking vector inputs.

$f(\mathbf{y})$	$\frac{df(\mathbf{y})}{d\mathbf{y}}$
$\mathbf{y}^T\mathbf{A}$	$\mathbf{A}$
$\mathbf{y}^T\mathbf{y}$	$2\mathbf{y}$
$\mathbf{y}^T\mathbf{A}\mathbf{y}$	$2\mathbf{A}\mathbf{y}$

Now let's look at the derivative of a scalar function f with respect to a vector (i.e., the function takes a vector of values and produces a single real number).

Let $f(\mathbf{y})$ be a function of a $p \times 1$ column vector $\mathbf{y} = [y_1, y_2, \ldots, y_p]^T$. The derivative of $f(\mathbf{y})$ with respect to $\mathbf{y}$ is denoted $\frac{\partial f(\mathbf{y})}{\partial \mathbf{y}}$ and

$$\frac{\partial f(\mathbf{y})}{\partial \mathbf{y}} = \begin{bmatrix} \frac{\partial f(\mathbf{y})}{\partial y_1} \\ \frac{\partial f(\mathbf{y})}{\partial y_2} \\ \vdots \\ \frac{\partial f(\mathbf{y})}{\partial y_p} \end{bmatrix}.$$

In words, the derivative of a scalar function with respect to its input vector is the vector of partial derivatives with respect the elements of the input vector.

Assume $\mathbf{A}$ is an $m \times p$ matrix of constant values. Then the results in Table A.2 are true.

Comparing Tables A.1 and A.2, one can make parallels with the derivative results from the two contexts.

A.6 Additional topics

A.6.1 Determinant

The determinant of a matrix is a special function that is applied to a square matrix and returns a scalar value. The determinant of a matrix $\mathbf{A}$ is usually denoted $|\mathbf{A}|$ or $\det(\mathbf{A})$. We do not discuss how to compute the determinant of a matrix, which is not needed for our purposes. You can find out more about matrix determinants at https://en.wikipedia.org/wiki/Determinant.

A.6.2 Linearly independent vectors

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be a set of n vectors of size $p \times 1$.

Then $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ are **linearly dependent** if there exists $\mathbf{a} = [a_1, a_2, \ldots, a_m)] \neq 0_{n\times 1}$ such

$$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_n\mathbf{x}_n = 0_{p\times 1}.$$

Let $\mathbf{X}$ be an $n \times p$ matrix such that $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$ make up its n rows and vectors $\mathbf{X}_{[1]}, \mathbf{X}_{[2]}, \ldots, \mathbf{X}_{[p]}$ make up its columns, so that

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix} = \begin{bmatrix} \mathbf{X}_{[1]} & \mathbf{X}_{[1]} & \cdots & \mathbf{X}_{[p]} \end{bmatrix}.$$

The columns vectors of $\mathbf{X}$ are linearly independent if there is no $\mathbf{a} = [a_1, a_2, \ldots, a_p] \neq 0_{p\times 1}$ such that

$$a_1\mathbf{X}_{[1]} + a_2\mathbf{X}_{[2]} + \cdots + a_p\mathbf{X}_{[p]} = 0_{p\times 1}.$$

The row vectors of $\mathbf{X}$ are linearly independent if there is no $\mathbf{a} = [a_1, a_2, \ldots, a_n] \neq 0_{n\times 1}$ such that

$$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_n\mathbf{x}_n = 0_{n\times 1}.$$

You can learn more about linear independence at https://en.wikipedia.org/wiki/Linear_independence.

A.6.3 Rank

The **rank** of a matrix is the number of linearly independent columns of the matrix.

If $\mathbf{X}$ is an $n \times p$ matrix that has linearly dependent columns, but removing a single column results in a linearly independent matrix, then the rank of $\mathbf{X}$ would be $p - 1$.

An $n \times p$ matrix has **full rank** its rank equals $\min(n, p)$, i.e., the smallers of its number of rows and columns.

Appendix B

Overview of probability, random variables, and random vectors

B.1 Probability Basics

The mathematical field of probability attempts to quantify how likely certain outcomes are, where the outcomes are produced by a random experiment (defined below). In what follows, we assume you have a basic understanding of set theory and notation.

We review some basic probability-related terminology in Table B.1.

Some comments about the terms in Table B.1:

- **Outcomes** may also be referred to as **points**, **realizations**, or **elements**.
- The set of outcomes of Ω are referred to as the **elements** of Ω.
- An **event** is a set of outcomes.
- The **empty set** is a subset of Ω but not an outcome of Ω.

Table B.1: Basic terminology used in probability.

term	notation	definition
experiment	N/A	A mechanism that produces outcomes that cannot be predicted with absolute cert
outcome	ω	The simplest kind of result produced by an experiment.
sample space	Ω	The set of all possible outcomes an experiment can produce.
event	A, A_i, B, etc.	Any subset of Ω.
empty set	$\emptyset$	The event that includes no outcomes.

- The **empty set** is a subset of every event $A \subseteq \Omega$.

We now review some basic set operations adn related facts. Let A and B be two events contained in Ω.

- The **intersection** of A and B, denoted $A \cap B$ is the set of outcomes that are common to both A and B, i.e., $A \cap B = \{\omega \in \Omega : \omega \in A \text{ and } \omega \in B\}$.
  - Events A and B are **disjoint** if $A \cap B = \emptyset$, i.e., if there are no outcomes common to events A and B.
- The **union** of A and B, denoted $A \cup B$ is the set of outcomes that are in A or B or both, i.e., $A \cup B = \{\omega \in \Omega : \omega \in A \text{ or } \omega \in B\}$.
- The **complement** of A, denoted A^c is the set of outcomes that are in Ω but are not in A, i.e., $A^c = \{\omega \in \Omega : \omega \notin A\}$.
  - The complement of A may also be denoted as $\overline{A}$ or A'.
- The set **difference** between A and B, denoted $A \setminus B$, is the elements of A that are not in B, i.e., $A \setminus B = \{\omega \in A : \omega \notin B\}$.
  - The set difference between A and B may also be denoted by $A - B$.
  - The set difference is order specific, i.e., $(A \setminus B) \neq (B \setminus A)$ in general.

A function P that assigns a real number $P(A)$ to every event A is a probability distribution if it satisfies three properties:

1. $P(A) \geq 0$ for all $A \in \Omega$
2. $P(\Omega) = P(\omega \in \Omega) = 1$
3. If $A_1, A_2, \ldots$ are disjoint, then $P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i)$.

A set of events $\{A_i : i \in I\}$ are **independent** if

$$P\left(\cap_{i \in J} A_i\right) = \prod_{i \in J} P(A_i)$$

for every finite subset $J \subseteq I$.

The **conditional probability** of A given B, denoted as $P(A \mid B)$, is the probability that A occurs given that B has occurred, and is defined as

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}, \quad P(B) > 0.$$

Some additional facts about probabilities:

- **Complement rule**: $P(A^c) = 1 - P(A)$.
- **Addition rule**: $P(A \cap B) = P(A) + P(B) - P(A \cap B)$.
- **Bayes' rule**: Assuming $\$P(A) > 0\ \$$ and $P(B) > 0$, then

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}.$$

- **Law of Total Probability**: Let $B_1, B_2, \ldots$ be a countably infinite partition of Ω. Then

$$P(A) = \sum_{i=1}^{\infty} P(A \cap B_i) = \sum_{i=1}^{\infty} P(A \mid B_i)P(B_i).$$

B.2 Random Variables

A **random variable** Y is a mapping/function

$$Y : \Omega \to \mathbb{R}$$

that assigns a real number $Y(\omega)$ to each outcome ω. We typically drop the (ω) notation for simplicity.

The **cumulative distribution function (CDF)** of Y, F_Y, is a function $F_Y : \mathbb{R} \to [0, 1]$ defined by

$$F_Y(y) = P(Y \leq y).$$

The subscript of F indicates the random variable the CDF describes. E.g., F_X denotes the CDF of the random variable X and F_Y denotes the CDF of the random variable Y. The subscript can be dropped when the context makes it clear what random variable the CDF describes. An F-distributed random variable is one that has the F distribution.

The **support** of Y, $\mathcal{S}$, is the smallest set such that $P(Y \in \mathcal{S}) = 1$.

B.2.1 Discrete random variables

Y is a **discrete** random variable if it takes countably many values $\{y_1, y_2, \dots\} = \mathcal{S}$.

The **probability mass function (pmf)** for Y is $f_Y(y) = P(Y = y)$, where $y \in \mathbb{R}$, and must have the following properties:

1. $0 \leq f_Y(y) \leq 1$.
2. $\sum_{y \in \mathcal{S}} f_Y(y) = 1$.

Additionally, the following statements are true:

- $F_Y(c) = P(Y \leq c) = \sum_{y \in \mathcal{S}: y \leq c} f_Y(y)$.
- $P(Y \in A) = \sum_{y \in A} f_Y(y)$ for some event A.
- $P(a \leq Y \leq b) = \sum_{y \in \mathcal{S}: a \leq y \leq b} f_Y(y)$.

The **expected value**, **mean**, or first moment of Y is defined as

$$E(Y) = \sum_{y \in \mathcal{S}} y f_Y(y),$$

assuming the sum is well-defined.

The **variance** of Y is defined as

$$\text{var}(Y) = E(Y - E(Y))^2 = \sum_{y \in \mathcal{S}} (y - E(Y))^2 f_Y(y).$$

Note that $\text{var}(Y) = E(Y - E(Y))^2 = E(Y^2) - [E(Y)]^2$. The last expression is often easier to compute.

The **standard deviation** of Y is

$$SD(Y) = \sqrt{\text{var}(Y)}.$$

B.2.1.1 Bernoulli distribution example

A random variable Y is said to have a Bernoulli distribution with probability π, denoted $Y \sim \mathsf{Bernoulli}(\pi)$, if $\mathcal{S} = \{0, 1\}$ and $P(Y = 1) = \pi$, where $\pi \in (0, 1)$.

The pmf of a Bernoulli random variable is

$$f_Y(y) = \pi^y (1 - \pi)^{(1-y)}.$$

The mean of a Bernoulli random variable is

$$E(Y) = 0(1 - \pi) + 1(\pi) = \pi.$$

The variance of a Bernoulli random variable is

$$\text{var}(Y) = (0 - \pi)^2(1 - \pi) + (1 - \pi)^2\pi = \pi(1 - \pi).$$

B.2.2 Continuous random variables

Y is a **continuous** random variable if there exists a function $f_Y(y)$ such that:

1. $f_Y(y) \geq 0$ for all y,
2. $\int_{-\infty}^{\infty} f_Y(y)dy = 1$,
3. $a \leq b$, $P(a < Y < b) = \int_a^b f_Y(y)dy$.

The function f_Y is called the **probability density function (pdf)**.

Additionally, $F_Y(y) = \int_{-\infty}^{y} f_Y(y)dy$ and $f_Y(y) = F'_Y(y)$ for any point y at which F_Y is differentiable.

The **mean** of a continuous random variables Y is defined as

$$E(Y) = \int_{-\infty}^{\infty} y f_Y(y)dy = \int_{y\in\mathcal{S}} y f_Y(y).$$

assuming the integral is well-defined.

The **variance** of a continuous random variable Y is defined by

$$\text{var}(Y) = E(Y - E(Y))^2 = \int_{-\infty}^{\infty} (y - E(Y))^2 f_Y(y)dy = \int_{y\in\mathcal{S}} (y - E(Y))^2 f_Y(y)dy.$$

B.2.2.1 Example (Exponential distribution)

A random variable Y is said to have an exponential distribution rate parameter λ, denoted with $Y \sim \mathsf{Exp}(\lambda)$ if $\mathcal{S} = y \in \mathbb{R} : y \geq 0$ and

$$f_Y(y) = \lambda \exp(-\lambda y).$$

The mean of an exponential random variable is

$$E(Y) = \int_0^\infty y\lambda \exp(-\lambda y)\, dy = -\exp(-\lambda y)(\lambda^{-1} + y)\Big]_0^\infty = \lambda^{-1}.$$

Note that this process involves integration by parts, which is not shown. Similarly, $E(Y^2) = 2\lambda^{-2}$. Thus,

$$\text{var}(Y) = E(Y^2) - [E(Y)]^2 = 2\lambda^{-2} - [\lambda^{-1}]^2 = \lambda^{-2}.$$

B.2.3 Useful facts for transformations of random variables

Let Y be a random variable and $a \in \mathbb{R}$ be a constant. Then:

- $E(a) = a$.
- $E(Y) = aE(Y)$.
- $E(a + Y) = a + E(Y)$.
- $\text{var}(a) = 0$.
- $\text{var}(aY) = a^2\text{var}(Y)$.
- $\text{var}(a + Y) = \text{var}(Y)$.
- For a discrete random variable and a function g,

  $$E(g(Y)) = \sum_{y \in \mathcal{S}} g(Y) f_Y(y),$$

  assuming the sum is well-defined.
- For a continuous random variable and a function g,

  $$E(g(Y)) = \int_{y \in \mathcal{S}} g(Y) f_Y(y)\, dy,$$

  assuming the integral is well-defined.

B.3 Multivariate distributions

B.3.1 Basic properties

Let $Y_1, Y_2, \ldots, Y_n$ denote n random variables with supports $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$, respectively.

If the random variables are **jointly discrete** (i.e., all discrete), then the joint pmf $f(y_1, \ldots, y_n) = P(Y_1 = y_1, \ldots, Y_n = y_n)$ satisfies the following properties:

1. $0 \leq f(y_1, \ldots, y_n) \leq 1$,
2. $\sum_{y_1 \in \mathcal{S}_1} \cdots \sum_{y_n \in \mathcal{S}_n} f(y_1, \ldots, y_n) = 1$,
3. $P((Y_1, \ldots, Y_n) \in A) = \sum_{(y_1, \ldots, y_n) \in A} f(y_1, \ldots, y_n)$.

In this context,

$$E(Y_1 \cdots Y_n) = \sum_{y_1 \in \mathcal{S}_1} \cdots \sum_{y_n \in \mathcal{S}_n} y_1 \cdots y_n f(y_1, \ldots, y_n).$$

In general,

$$E(g(Y_1, \ldots, Y_n)) = \sum_{y_1 \in \mathcal{S}_1} \cdots \sum_{y_n \in \mathcal{S}_n} g(y_1, \ldots, y_n) f(y_1, \ldots, y_n),$$

where g is a function of the random variables.

If the random variables are **jointly continuous**, then $f(y_1, \ldots, y_n) = P(Y_1 = y_1, \ldots, Y_n = y_n)$ is the joint pdf if it satisfies the following properties:

1. $f(y_1, \ldots, y_n) \geq 0$,
2. $\int_{y_1 \in \mathcal{S}_1} \cdots \int_{y_n \in \mathcal{S}_n} f(y_1, \ldots, y_n) dy_n \cdots dy_1 = 1$,
3. $P((Y_1, \ldots, Y_n) \in A) = \int \cdots \int_{(y_1, \ldots, y_n) \in A} f(y_1, \ldots, y_n) dy_n \ldots dy_1$.

In this context,

$$E(Y_1 \cdots Y_n) = \int_{y_1 \in \mathcal{S}_1} \cdots \int_{y_n \in \mathcal{S}_n} y_1 \cdots y_n f(y_1, \ldots, y_n) dy_n \ldots dy_1.$$

In general,

$$E(g(Y_1, \ldots, Y_n)) = \int_{y_1 \in \mathcal{S}_1} \cdots \int_{y_n \in \mathcal{S}_n} g(y_1, \ldots, y_n) f(y_1, \ldots, y_n) dy_n \cdots dy_1,$$

where g is a function of the random variables.

B.3.2 Marginal distributions

If the random variables are jointly discrete, then the marginal pmf of Y_1 is

$$f_{Y_1}(y_1) = \sum_{y_2 \in \mathcal{S}_2} \cdots \sum_{y_n \in \mathcal{S}_n} f(y_1, \ldots, y_n).$$

Similarly, if the random variables are jointly continuous, then the marginal pdf of Y_1 is

$$f_{Y_1}(y_1) = \int_{y_2 \in \mathcal{S}_2} \cdots \int_{y_n \in \mathcal{S}_n} f(y_1, \ldots, y_n) dy_n \cdots dy_2.$$

B.3.3 Independence of random variables

Random variables X and Y are independent if

$$F(x, y) = F_X(x)F_Y(y).$$

Alternatively, X and Y are independent if

$$f(x, y) = f_X(x)f_Y(y).$$

B.3.4 Conditional distributions

Let X and Y be random variables. Then assuming $f_Y(y) > 0$, the conditional distribution of X given $Y = y$, denoted $X|Y = y$ is

$$f(x|y) = \frac{f(x, y)}{f_Y(y)}, \quad f_Y(y) > 0.$$

B.3.5 Covariance

The covariance between random variables X and Y is

$$\text{cov}(X, Y) = E[(X - E(X))(Y - E(Y))] = E(XY) - E(X)E(Y).$$

B.3.6 Useful facts for transformations of multiple random variables

Let a and b be scalar constants. Let Y and Z be random variables. Then:

- $E(aY + bZ) = aE(Y) + bE(Z)$
- $\text{var}(Y + Z) = \text{var}(Y) + \text{var}(Z) + 2\text{cov}(Y, Z)$
- $\text{cov}(a, Y) = 0$
- $\text{cov}(Y, Y) = \text{var}(Y)$
- $\text{cov}(aY, bZ) = ab\text{cov}(Y, Z)$
- $\text{cov}(a + Y, b + Z) = \text{cov}(Y, Z)$

If Y and Z are also independent, then:

- $E(YZ) = E(Y)E(Z)$.
- $\text{cov}(Y, Z) = 0$.

In general, if $Y_1, Y_2, \ldots, Y_n$ are a set of random variables, then:

- $E(\sum_{i=1}^n Y_i) = \sum_{i=1}^n E(Y_i)$, i.e., the expectation of the sum of random variables is the sum of the expectation of the random variables.
- $\text{var}(\sum_{i=1}^n Y_i) = \sum_{i=1}^n \text{var}(Y_i) + \sum_{j=1}^n \sum_{1 \le i < j \le n} 2\text{cov}(Y_i, Y_j)$, i.e., the variance of the sum of random variables is the sum fo the variables' variances plus the sum of twice all possible pairwise covariances.

If in addition, $Y_1, Y_2, \ldots, Y_n$ are all independent of each other, then:

- $\text{var}(\sum_{i=1}^n Y_i) = \sum_{i=1}^n \text{var}(Y_i)$ since all pairwise covariances are 0.

B.3.7 Binomial distribution example

A random variable Y is said to have a Binomial distribution with n trials and probability of success θ, denoted $Y \sim \mathsf{Bin}(n, \theta)$ when $\mathcal{S} = \{0, 1, 2, \ldots, n\}$ and the pmf is

$$f(y \mid \theta) = \binom{n}{y} \theta^y (1-\theta)^{(n-y)}.$$

An alternative explanation of a Binomial random variable is that it is the sum of n independent and identically-distributed Bernoulli random variables. Alternatively, let $Y_1, Y_2, \ldots, Y_n \overset{i.i.d.}{\sim} \mathsf{Bernoulli}(\theta)$, where i.i.d. stands for independent and identically distributed, i.e., $Y_1, Y_2, \ldots, Y_n$ are independent random variables with identical distributions. Then $Y = \sum_{i=1}^{n} Y_i \sim \mathsf{Bin}(n, \theta)$.

Using this information and the facts above, we can easily determine the mean and variance of Y.

Using our results from Section B.2.1.1, we can see that $E(Y_i) = \theta$ for $i = 1, 2, \ldots, n$. Similarly, $\text{var}(Y_i) = \theta(1-\theta)$ for $i = 1, 2, \ldots, n$.

We determine that:

$$E(Y) = E\left(\sum_{i=1}^{n} Y_i\right) = \sum_{i=1}^{n} E(Y_i) = \sum_{i=1}^{n} \theta = n\theta.$$

Similarly, since $Y_1, Y_2, \ldots, Y_n$ are i.i.d. and using the fact in Section B.3.6, we see that

$$\text{var}(Y) = \text{var}(\sum_{i=1}^{n} Y_i) = \sum_{i=1}^{n} \text{var}(Y_i) = \sum_{i=1}^{n} \theta(1-\theta) = n\theta(1-\theta).$$

B.3.8 Continuous bivariate distribution example

Hydration is important for health. Like many people, the author has a water bottle he uses to say hydrated through the day and drinks several liters of water per day. Let's say the author refills his water bottle every 3 hours. Let Y denote the proportion of the water bottle filled with water at the beginning of the 3-hour window. Let X denote the amount of water the author consumes in the 3-hour window (measured in the the proportion of total water bottle capacity. We know that $0 \leq X \leq Y \leq 1$. The joint density of the random variables is

$$f(x, y) = 4y^2, \quad 0 \leq x \leq y \leq 1,$$

and 0 otherwise.

We answer a series of questions about this distribution.

Q1: Determine $P(0.5 \leq X \leq 1, 0.75 \leq Y)$.

Note that the comma between the two events means "and".

Since X must be no more than Y, we can answer this question as

$$\int_{0.75}^{1}\int_{0.5}^{y} 4y^2\,dx\,dy = 229/768 \approx 0.30.$$

Q2: Determine the marginal distributions of X and Y.

To find the marginal distribution of X, we must integrate the joint pdf with respect to the limits of Y. Don't forget to include the support of the pdf of X (which after integrating out Y, must be between 0 and 1).

$$\begin{aligned} f_X(x) &= \int_x^1 4y^2\,dy \\ &= \frac{4}{3}(1-x^3), \quad 0 \le x \le 1. \end{aligned}$$

Similarly,

$$\begin{aligned} f_Y(y) &= \int_0^y 4y^2\,dx \\ &= 4y^3, \quad 0 \le y \le 1. \end{aligned}$$

Q3: Determine the means of X and Y.

The mean of X is the integral of $xf_X(x)$ over the support of X, i.e.,

$$E(X) = \int_0^1 x\left(\frac{4}{3}(1-x^3)\right)dx = 2/5.$$

Similarly,

$$E(Y) = \int_0^1 y(4y^3)\,dy = 4/5.$$

Q4: Determine the variances of X and Y. We use the formula $\text{var}(X) = E(X^2) - [E(X)^2]$ to compute the variances. First,

$$E(X^2) = \int_0^1 x^2\left(\frac{4}{3}(1-x^3)\right)dx = 2/9.$$

Second,

$$E(Y^2) = \int_0^1 y^2(4y^3)\,dy = 2/3.$$

Thus, $\text{var}(X) = 2/5 - (2/9)^2 = 142/405$ and $\text{var}(Y) = 4/5 - (2/3)^2 = 16/45$.

Q5: Determine the mean of XY.

The mean of XY requires us to integrate the product of xy and the joint pdf over the joint support of X and Y. Specifically,

$$E(XY) = \int_0^1 \int_0^y xy(4y^2)\, dx\, dy = 1/12.$$

Q6: Determine the covariance of X and Y. Using our previous work, we see that

$$\text{cov}(X, Y) = E(XY) - E(X)E(Y) = 1/12 - (2/5)(4/5) = -71/300.$$

Q7: Determine the mean and variance of $Y - X$, i.e., the average amount of water remaining after a 3-hour window and the variability of that amount.

Using the results in Section B.3.6, we have that

$$E(Y - X) = E(Y) - E(X) = 4/5 - 2/5 = 2/5,$$

and

$$\text{var}(Y-X) = \text{var}(Y)+\text{var}(X)-2\text{cov}(Y, X) = 16/45+142/405-2(-71/300) \approx 1.18.$$

B.4 Random vectors

B.4.1 Definition

A **random vector** is a vector of random variables. A random vector is assumed to be a column vector unless otherwise specified.

Additionally, a **random matrix** is a matrix of random variables.

B.4.2 Mean, variance, and covariance

Let $\mathbf{y} = [Y_1, Y_2, \ldots, Y_n]$ be an $n \times 1$ random vector.

The mean of a random vector is the vector containing the means of the random variables in the vector. More specifically, the mean of $\mathbf{y}$ is defined as

$$E(\mathbf{y}) = \begin{bmatrix} E(Y_1) \\ E(Y_2) \\ \vdots \\ E(Y_n) \end{bmatrix}.$$

The variance of a random vector isn't a number. Instead, it is the matrix of all covariances of all pairs of random variables in the random vector. The variance

of $\mathbf{y}$ is

$$\begin{aligned}\text{var}(\mathbf{y}) &= E(\mathbf{y}\mathbf{y}^T) - E(\mathbf{y})E(\mathbf{y})^T \\ &= \begin{bmatrix} \text{var}(Y_1) & \text{cov}(Y_1, Y_2) & \dots & \text{cov}(Y_1, Y_n) \\ \text{cov}(Y_2, Y_1) & \text{var}(Y_2) & \dots & \text{cov}(Y_2, Y_n) \\ \vdots & \vdots & \vdots & \vdots \\ \text{cov}(Y_n, Y_1) & \text{cov}(Y_n, Y_2) & \dots & \text{var}(Y_n) \end{bmatrix}.\end{aligned}$$

Alternatively, the variance of $\mathbf{y}$ is called the **covariance matrix** of $\mathbf{y}$ or the **variance-covariance matrix** of $\mathbf{y}$.

Let $\mathbf{x} = [X_1, X_2, \dots, X_n]$ be an $n \times 1$ random vector.

The covariance matrix between $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$\text{cov}(\mathbf{x}, \mathbf{y}) = E(\mathbf{x}\mathbf{y}^T) - E(\mathbf{x})E(\mathbf{y})^T.$$

Note that $\text{var}(\mathbf{y}) = \text{cov}(\mathbf{y}, \mathbf{y})$.

B.4.3 Properties of transformations of random vectors

Define:

- $\mathbf{a}$ to be an $n \times 1$ vector of constants (not necessarilly the same constant).
- $\mathbf{A}$ to be an $m \times n$ matrix of constants (not necessarily the same constant).
- $\mathbf{x} = [X_1, X_2, \dots, X_n]$ to be an $n \times 1$ random vector.
- $\mathbf{y} = [Y_1, Y_2, \dots, Y_n]$ to be an $n \times 1$ random vector.
- $\mathbf{z} = [Z_1, Z_2, \dots, Z_n]$ to be an $n \times 1$ random vector.
- $0_{n \times n}$ to be an $n \times n$ matrix of zeros.

Then:

- $E(\mathbf{A}\mathbf{y}) = \mathbf{A}E(\mathbf{y})$.
- $E(\mathbf{y}\mathbf{A}^T) = E(\mathbf{y})\mathbf{A}^T$.
- $E(\mathbf{x} + \mathbf{y}) = E(\mathbf{x}) + E(\mathbf{y})$.
- $\text{var}(\mathbf{A}\mathbf{y}) = \mathbf{A}\text{var}(\mathbf{y})\mathbf{A}^T$.
- $\text{cov}(\mathbf{x} + \mathbf{y}, \mathbf{z}) = \text{cov}(\mathbf{x}, \mathbf{z}) + \text{cov}(\mathbf{y}, \mathbf{z})$.
- $\text{cov}(\mathbf{x}, \mathbf{y} + \mathbf{z}) = \text{cov}(\mathbf{x}, \mathbf{y}) + \text{cov}(\mathbf{x}, \mathbf{z})$.
- $\text{cov}(\mathbf{A}\mathbf{x}, \mathbf{y}) = \mathbf{A}\ \text{cov}(\mathbf{x}, \mathbf{y})$.
- $\text{cov}(\mathbf{x}, \mathbf{A}\mathbf{y}) = \text{cov}(\mathbf{x}, \mathbf{y})\mathbf{A}^T$.
- $\text{var}(\mathbf{a}) = 0_{n \times n}$.
- $\text{cov}(\mathbf{a}, \mathbf{y}) = 0_{n \times n}$.
- $\text{var}(\mathbf{a} + \mathbf{y}) = \text{var}(\mathbf{y})$.

B.4.4 Continuous bivariate distribution example continued

Using the definitions and results in B.4, we want to answer **Q7** of the example in B.3.8. Summarizing only the essential details, we have a random vector

$\mathbf{z} = [X, Y]$ with mean $E(\mathbf{z}) = [2/5, 4/5]$ and covariance matrix

$$\text{var}(\mathbf{z}) = \begin{bmatrix} 142/405 & -71/300 \\ -71/300 & 16/45 \end{bmatrix}.$$

We want to determine $E(Y - X)$ and $\text{var}(Y - X)$.

Define $\mathbf{A} = [-1, 1]^T$ (the ROW vector with 1 and -1). Then

$$\mathbf{Az} = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = Y - X,$$

and

$$E(Y - X) = E(\mathbf{Az}) = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 2/5 \\ 4/5 \end{bmatrix} = -2/5 + 4/5 = 2/5.$$

Additionally,

$$\begin{aligned} &\text{var}(Y - X) \\ &= \text{var}(\mathbf{Az}) \\ &= \mathbf{A}\text{var}(\mathbf{z})\mathbf{A}^T \\ &= \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 142/405 & -71/300 \\ -71/300 & 16/45 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} -142/405 - 71/300 & 71/300 + 16/45 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \\ &= 142/405 + 16/45 + 2(71/300) \\ &\approx 1.18. \end{aligned}$$

B.5 Multivariate normal (Gaussian) distribution

B.5.1 Definition

The random vector $\mathbf{y} = [Y_1, \ldots, Y_n]$ has a multivariate normal distribution with mean $E(\mathbf{y}) = \boldsymbol{\mu}$ (an $n \times 1$ vector) and covariance matrix $\text{var}(\mathbf{y}) = \boldsymbol{\Sigma}$ (an $n \times n$ matrix) if its joint pdf is

$$f(\mathbf{y}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{y} - \boldsymbol{\mu})\right),$$

where $|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$. Note that $\boldsymbol{\Sigma}$ must be symmetric and positive definite.

In this case, we would denote the distribution of $\mathbf{y}$ as

$$\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}).$$

B.5.2 Linear functions of a multivariate normal random vector

A linear function of a multivariate normal random vector (i.e., $\mathbf{a} + \mathbf{A}\mathbf{y}$, where $\mathbf{a}$ is an $m \times 1$ vector of constant values and $\mathbf{A}$ is an $m \times n$ matrix of constant values) is also multivariate normal (though it could collapse to a single random variable if $\mathbf{A}$ is a $1 \times n$ vector).

Application: Suppose that $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. For an $m \times n$ matrix of constants $\mathbf{A}$, $\mathbf{A}\mathbf{y} \sim \mathcal{N}(\mathbf{A}\boldsymbol{\mu}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T)$.

More generally, the most common estimators used in linear regression are linear combinations of a (typically) multivariate normal random vector, meaning that many of the estimators also have a (multivariate) normal distribution.

B.5.3 OLS example

Ordinary least squares regression is a method for fitting a linear regression model to data. Suppose that we have observed variables $X_1, X_2, X_3, \ldots, X_{p-1}, Y$ for each of n subjects from some population, with $X_{i,j}$ denoting the value of X_j for observation i and Y_i denoting the value of Y for observation i. In general, we want to use $X_1, \ldots, X_{p-1}$ to predict the value of Y. Let

$$\mathbf{X} = \begin{bmatrix} 1 & X_{1,1} & X_{1,2} & \cdots & X_{1,n} \\ 1 & X_{2,1} & X_{2,2} & \cdots & X_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_{n,1} & X_{n,2} & \cdots & X_{n,n} \end{bmatrix}$$

be a full-rank matrix of size $n \times p$ and

$$\mathbf{y} = (Y_1, Y_2, \ldots, Y_n)^T,$$

be an $n \times 1$ vector of responses. It is common to assume that

$$\mathbf{y} \sim \mathcal{N}(0_{n\times 1}, \sigma^2 \mathbf{I}_{n\times n}).$$

The matrix $\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ projects $\mathbf{y}$ into the space spanned by the vectors in $\mathbf{X}$. Because of what we know about linear functions of a multivariate normal random vector (B.5.2), we can determine that

$$\mathbf{H}\mathbf{y} \sim \mathcal{N}(0_{n\times n}, \sigma^2 \mathbf{H}).$$

Appendix C

Review of Estimation and Inference

A primary purpose of statistics is taking a sample of values from a population and using the sample to draw conclusions about that population. In what follows, we discuss statistical concepts related to estimation, hypothesis testing, and confidence intervals.

C.1 Estimation

A **parameter** is a numeric characteristic that describes a population. E.g., the population mean, standard deviation, or cumulative distribution function.

The **target** parameter or **parameter of interest** is the population parameter we would like to estimate.

There are different kinds of estimates:

- A **point estimate** is a single number that we *hope* is close to the true value of the target parameter.
- An **interval estimate** is an interval of numbers that we *hope* will contain the target parameter.

An estimate and an estimator are different but related concepts.

- An estimate is a specific number (for a point estimate) or a specific range of numbers (for an interval estimate).
- An **estimator** is a formula we use to calculate an estimate (once we get a sample of data).

Once the data are observed, an estimate is fixed. An estimator is a random variable. An estimator produces different estimates based on the sample of data

we obtain from the population.

The **sampling distribution** of an estimator is the distribution of the estimates we get when we use the estimator to compute estimates from all possible samples of a fixed size n from the population of interest.

A point estimator, $\hat{\theta}$, is an **unbiased estimator** of a target parameter, θ, if $E(\hat{\theta}) = \theta$. An estimator is biased if it is not biased.

The **bias** of an estimator is defined as

$$B(\hat{\theta}) = E(\hat{\theta}) - \theta.$$

The **variance of an estimator** is defined as

$$\text{var}(\hat{\theta}) = E[\hat{\theta} - E(\hat{\theta})]^2.$$

The **standard error of an estimator** is the standard deviation of the estimator, i.e.,

$$\text{se}(\hat{\theta}) \equiv \text{sd}(\hat{\theta}) = \sqrt{\text{var}(\hat{\theta})}.$$

Typically, we cannot compute the standard error of an estimator because it is a function of parameters that we do not know. Instead, we use the sample data to estimate the standard error. When we hear or read the term "standard error", we must carefully evaluate whether the "standard error" presented is the theoretical standard error or the estimated standard error (and it's nearly always the latter).

The **mean square error** of a point estimator is

$$MSE(\hat{\theta}) = E(\hat{\theta} - \theta)^2,$$

which is equivalent to

$$MSE(\hat{\theta}) = \text{var}(\hat{\theta}) + [B(\hat{\theta})]^2.$$

The MSE formula makes it clear that there is a "bias-variance trade off" when choosing between point estimators. Typically, unbiased point estimators will have larger variance (and correspondingly, MSE). Biased estimators will often have smaller MSE, but are (obviously) biased. It's a trade off we have to balance.

C.2 Hypothesis Testing

A **statistical test of hypotheses** or **hypothesis test** is a statistical procedure used to decide between a null hypothesis, H_0, and an alternative hypothesis, H_a or H_1. The null hypothesis is usually a hypothesis that "nothing interesting is going on". The alternative hypothesis is generally the complement of the null hypothesis and is usually what we want to show is true.

A **test statistic** is a number used to decide between H_0 and H_a. A test statistic is a function of the data, and generally, parameters in the hypotheses. A test statistic measures the compatibility of the observed data with H_0. A "small" test statistic suggests the observed data are consistent with H_0, while an "extreme" test statistic indicates that the observed data are inconsistent with H_0, which we take as evidence that H_a is true.

The **null distribution** allows us to identify the values of the test statistic that are typical or unusual when H_0 is true. Formally, the null distribution is the distribution of the test statistic under the assumption that H_0 is true.

There are two types of errors we can make when doing hypothesis testing:

- Type I error: rejecting H_0 when H_0 is true.
- Type II error: failing to reject H_0 when H_a is true.

We can control the Type I error rate at a specified level, α, called the **significance level**, since we know the distribution of our test statistic under the assumption that H_0 is true. We reject H_0 and conclude that H_a is true if the test statistic falls in the **rejection region** of the null distribution, which is the set of test statistics that are the $100\alpha\%$ most unlikely test statistics if H_0 is true.

Instead of using the test statistic directly to decide between H_0 and H_a, we generally use the test statistic to compute a p-value. The **p-value** of a test statistic is the probability of seeing a test statistic at least as supportive of H_a when H_0 is true. If we specify the significance level, α, prior to performing our hypothesis test (which is the ethical thing to do), then we reject H_0 and conclude H_a is true when the p-value $< \alpha$. Otherwise, we fail to reject H_0.

Researchers sometimes say that the smaller the p-value, the stronger the evidence that H_a is true and H_0 is false. This isn't definitively true because the p-value doesn't have the ability to distinguish between the following options: (1) H_0 is true but our observed data were very unlikely, (2) H_0 is false. When H_0 is true, then the test statistic (for simple hypotheses and continuous test statistics) has a uniform distribution over the interval [0, 1]. However, if the H_a is true, then the p-value is more likely to be small, which makes us think H_a is true for small p-values. However, unless we know the **power** of our test, which is the probability that we reject H_0 when H_a is true, then it is very difficult to assess how much evidence for H_a a small p-value provides. Gibson (2021) point out the p-values can be interpreted naturally on a $\log_{10}$ scale. Gibson (2021) states:

> The p-value can be expressed as $p = c \times 10^{-k}$ so that $\log_{10}(p) = -\log_{10}(c) + k$, where c is a constant and k is an integer, which implies that only the magnitude k measures the actual strength of evidence (Boos and Stefanski 2011). ... This would suggest that $p = 0.01$ ($k = 2$) could be interpreted as twice the evidence [for H_a as] $p = 0.10$ ($k = 1$).

Gibson (2021) provides a thorough review of p-value interpretation. Table C.1 summarizes common strength of evidence interpretations for p-values.

Table C.1: An summary of common strength-of-evidence interpretations for p-values.

p-value	Interpretation
more than - 0.10	no evidence for H_a
≤ 0.10	weak evidence for H_a
≤ 0.05	moderate evidence for H_a
≤ 0.01	strong evidence for H_a
≤ 0.001	very strong evidence for H_a

Example

Suppose that $Y_1, \ldots, Y_n$ is a random sample (in other words, an independent and identically distributed sample) from a population having a normal distribution with unknown mean μ and variance $\sigma^2 = 1$.

We would like to decide between the following two hypotheses: $H_0 : \mu = 0$ and $H_a : \mu \neq 0$.

Consider the statistic $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$. If H_0 is true, then $\bar{Y} \sim \mathcal{N}(0, 1/n)$ and the test statistic

$$Z^* = \bar{Y}/(1/\sqrt{n}) = \sqrt{n}\bar{Y} \sim \mathcal{N}(0, 1),$$

i.e., the null distribution of Z^* is $\mathcal{N}(0, 1)$.

If $\alpha = 0.10$, then the 10% of test statistics that are most unlikely if H_0 is true (i.e., most supportive of H_a) are more extreme than $Z^{0.95}$ and $Z^{0.05}$, the 0.05 and 0.95 quantiles of a standard normal distribution, respectively. (The superscript in the quantile notation indicates the area to the right of the quantile in the CDF). The 0.05 quantile of the standard normal distribution is -1.65 and the 0.95 quantile is 1.65. In R, we can find these quantile using the `qnorm` function, where the first argument of `qnorm` is `p`, which is the vector of probabilities to the LEFT of the quantile. We verify these quantiles using the code below.

```
qnorm(c(0.05,0.95))
## [1] -1.644854  1.644854
```

H_0 should be rejected when Z^* is less than -1.65 or more than 1.65, i.e., the rejection region is $(-\infty, -1.65) \cup (1.65, \infty)$.

Alternatively, we could compute the p-value using the formula $2P(Z \geq |Z^*|)$ in order to make our choice between the hypotheses.

Suppose $z^* = 1.74$ and $\alpha = 0.10$. The test statistic is in the rejection region, so we would conclude that H_a is true. The p-value is $2P(Z \geq 1.74) = 0.082$. Using the p-value approach we would conclude that H_a is true. (Note that the rejection region and p-value approaches to deciding between hypotheses must agree). In R, we can compute the p-value using the code below.

```
2*(1 - pnorm(1.74))
## [1] 0.08185902
```

In straightforward language, our interpretation could be: there is weak evidence that the population mean differs from 0.

Simulation study

We provide a brief simulation study to better understand the null distribution and also how rejection regions are chosen to control the type I error rate.

Assume that we sample $n = 10$ values from a $\mathcal{N}(\mu, \sigma^2)$ population but that we don't know the mean or the standard deviation. Assume the hypotheses we want to test are $H_0 : \mu = 2$ versus $H_a : \mu > 2$ (implicitly, the null hypothesis is $H_0 : \mu \leq 2$).

In this context, it is common to use the test statistic

$$T^* = \frac{\bar{Y} - \mu}{s/\sqrt{n}},$$

where s is the sample standard deviation of the measurements. Under the null hypothesis, $\mu = 2$, and statistical theory tells us that $T^* \sim t_{n-1}$, i.e., the test statistic has a t distribution with $n - 1$ degrees of freedom. Since $n = 10$, our test statistic has a t distribution with 9 degrees of freedom if the null hypothesis is true.

Recall that the null distribution of a test statistic is its sampling distribution under the assumption that the null hypothesis is true. In the code below, we:

1. Draw $B = 1,000$ samples of size $n = 10$ from our $\mathcal{N}(\mu, \sigma^2)$ population assuming the null hypothesis is true, i.e., with $\mu = 2$.
2. Compute the test statistic for each sample.

```
# set number seed for reproducible results
set.seed(12)
# create matrix to store samples
samples <- matrix(nrow = 10000, ncol = 10)
# draw 10000 samples of size 10 from a N(4, 4^2)
for (i in 1:10000) {
  samples[i,] <- rnorm(n = 10, mean = 2, sd = 4)
}
# compute sample mean and sd for each sample
means <-  rowMeans(samples)
sds <- apply(samples, 1, sd)
# compute test statistic for each sample
tstats <- (means - 2)/(sds/sqrt(10))
```

We now use the code below to compute the empirical density of the computed test statistics and overlay a density for a t random variable with 9 degrees of

freedom.

To draw our sample, we must choose a value of σ^2. We will use $\sigma^2 = 16$. The exact value isn't important, but choosing a fixed number for σ^2 is critical for the example below. Figure C.1 displays the results. We see that the two distributions match up very well.

```
# plot empirical null distribution
plot(density(tstats),
     xlab = "test statistic",
     ylab = "density",
     main = "empirical versus true null distribution")
# sequence to plot null density over
s <- seq(-5, 5, len = 1000)
lines(s, dt(s, df = 9), lty = 2, col = "blue")
```

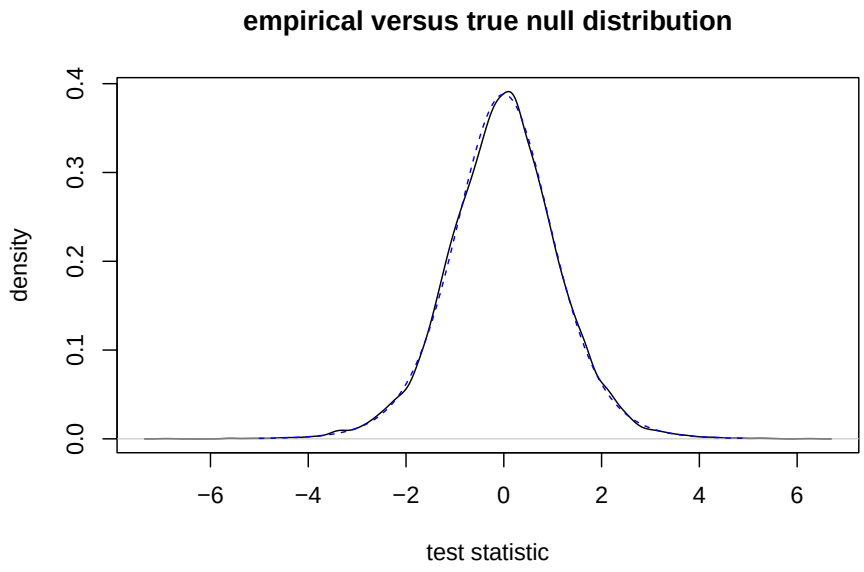


Figure C.1: Comparison of empirical null distribution to theoretical null distribution. The empirical null distribution is shown by the solid black line and the theoretical null distribution by a dashed blue line.

What should we takeaway from this example? The null distribution of a hypothesis test is the sampling distribution of the test statistic under the assumption that H_0 is true. We approximated the null distribution of our test statistic by drawing 10,000 samples from the population distribution under the assumption that H_0 was true.

How does the null distribution relate to choosing the rejection region? First, the type I error rate is the probability of rejecting H_0 when H_0 is true. Since we know the null distribution, we know what behavior to expect from our test statistic if H_0 is true. Thus, we know what test statistics are most unlikely if H_0 is true. Let's say we want to control the type I error at $\alpha = 0.05$. For this upper-tailed test, we should reject H_0 when the test statistic is greater than $t_9^{0.05}$, i.e., the 0.95 quantile of a t distribution with 9 degrees of freedom. Why do we use this threshold? Because if H_0 is true, this will only lead to erroneous rejections of H_0 (i.e., a type I error) 5% of the time. In the code below, we compute the sample proportion of test statistics from our null distribution that are more than $t_9^{0.05}$. Our sample proportion is very close to 0.05, and this number will converge to 0.05 as we increase the number samples used in our simulation.

```
mean(tstats > qt(0.95, df = 9))
## [1] 0.0489
```

C.3 Confidence Intervals

A **confidence interval** provides us with plausible values of a target parameter. It is the most common type of interval estimator.

A confidence interval procedure has an associated **confidence level**. When independent random samples are taken repeatedly from the population, a confidence interval procedure will produce intervals containing the target parameter with probability equal to the confidence level. Confidence level is associated with a confidence interval *procedure*, not a specific interval. A 95% confidence interval procedure will produce intervals that contain the target parameter 95% of the time. A specific interval estimate will either contain the target parameter or it will not.

The formulas for confidence intervals are usually derived from a pivotal quantity. A **pivotal quantity** is a function of the data and the target parameter whose distribution does not depend on the value of the target parameter.

Example:

Suppose $Y_1, Y_2, \ldots, Y_n \overset{i.i.d.}{\sim} \mathcal{N}(\mu, 1)$.

The random variable $Z = (\bar{Y} - \mu)/(1/\sqrt{n}) \sim \mathcal{N}(0, 1)$ is a pivotal quantity.

Since $P(-1.96 \leq Z \leq 1.96) = 0.95$, we can derive that

$$P(\bar{Y} - 1.96 \times 1/\sqrt{n} \leq \mu \leq \bar{Y} + 1.96 \times 1/\sqrt{n}) = 0.95.$$

Our 95% confidence interval for μ in this context is

$$[\bar{Y} - 1.96 \times 1/\sqrt{n}, \bar{Y} + 1.96 \times 1/\sqrt{n}].$$

If $\bar{Y} = 0.551$ and $n = 10$, then the associated 95% confidence interval for μ is [-0.070,1.171].

More discussion of confidence level

The CI formula given above is supposed to produce 95% confidence intervals (i.e., the confidence level of the procedure is 0.95). If produce 100 intervals from independent data sets, then about 95% of them would contain the true mean, but about 5% would not. To illustrate this further, we use a small simulation example below to produce 100 95% confidence intervals using a sample of size n = 10 for a $\mathcal{N}(\prime, \infty)$ population.

First, we obtain 100 samples of size 10 from the population and then compute the sample mean of each sample. We do this in the code below.

```
# create matrix to store samples
samples <- matrix(0, nrow = 100, ncol = 10)
# obtain 100 samples from the population
for (i in 1:100) {
  samples[i,] <- rnorm(n = 10, mean = 0, sd = 1)
}
# calculate the sample mean for each sample
means = rowMeans(samples) #calculates the mean of each row
```

Next, we use the formula above to determine the lower and upper bound of the confidence interval associated with each interval. Since we want 95% confidence intervals, we use the 0.975 quantile of the standard normal distribution to construct our interval.

```
#calculate the lower and upper bounds for the 95% CIs
lb = means - qnorm(.975) * 1/sqrt(10)
ub = means + qnorm(.975) * 1/sqrt(10)
```

Next, we plot each interval. The intervals are orange if the interval doesn't contain the true population mean, which is 0. Figure C.2 displays our results.

```
#create blank plot
plot(range(c(lb, ub)), c(1,100),
     xlab = "", ylab = "interval", type = "n")
# title plot
title("Interpretation of a Confidence Interval")
# plot true mean
abline(v = 0, col = "blue")
#plot interval each sample
for (i in 1:100) lines(c(lb[i], ub[i]), c(i, i))
#highlight intervals missing 0 in orange
for (i in 1:100) {
  if (lb[i] > 0 | ub[i] < 0 )   {
    lines(c(lb[i],ub[i]), c(i,i), col = "orange")
  }
}
```

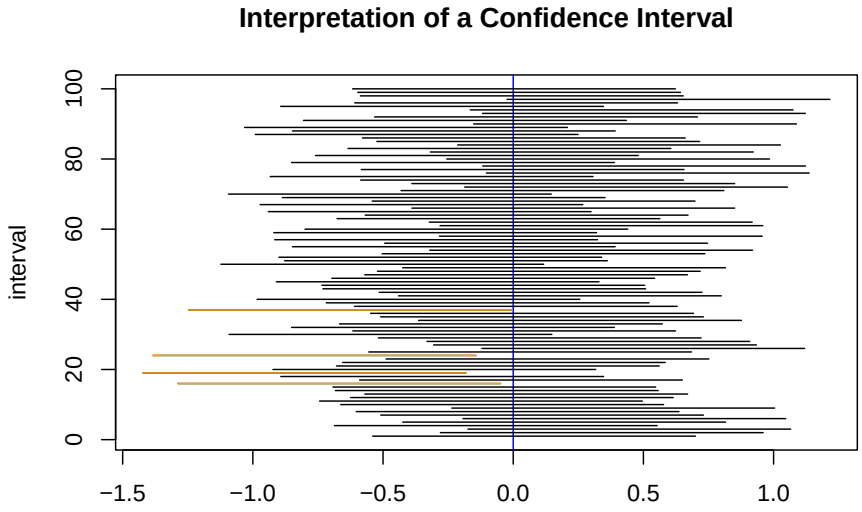


Figure C.2: A plot of 100 95% confidence intervals for a population mean produced from independent samples from a $\mathcal{N}(0, 1)$ population.

In this example, 96 out of 100 intervals for the population mean contained the true population mean of 0. Notice how the intervals move around. This is because each sample provides us with slightly different values, so the intervals move around because of the samples obtained. Each interval either contains the true mean of 0 or it does not. But as a whole, the procedure we are using will produce confidence intervals that contain the true mean 95% of the time.

C.4 Linking Hypothesis Tests and Confidence Intervals

CIs are directly linked to hypothesis tests.

A $100(1 - \alpha)\%$ two-sided confidence interval for target parameter θ is linked with a hypothesis test of $H_0 : \theta = c$ versus $H_a : \theta \neq c$ tested at level α.

- Any point that lies within the $100(1 - \alpha)\%$ confidence interval for θ represents a value of c for which the associated null hypothesis would not be rejected at significance level θ.
- Any point outside of the confidence interval is a value of c for which the associated null hypothesis would be rejected.

Similar relationships hold for one-sided CIs and hypothesis tests.

Example:

Consider the 95% confidence interval for μ we previously constructed: [-0.070,1.171].

That interval is conceptually linked to the statistical test of $H_0 : \mu = c$ versus $H_a : \mu \neq c$ using $\alpha = 0.05$.

We would reject H_0 for any hypothesized values of c less than -0.070 or more than 1.171. We would fail to reject H_0 for any values of c between -0.070 and 1.171.

A confidence interval provides us with much of the same information as a hypothesis test, but it doesn't provide the p-value or allow us to do hypothesis tests at different significance levels.

Confidence intervals are often preferred over hypothesis tests because they provide additional information in the form of plausible parameters values while giving us enough information to perform a hypothesis test.

References

Anscombe, Francis J. 1973. "Graphs in Statistical Analysis." *The American Statistician* 27 (1): 17–21.

Bonferroni, Carlo. 1936. "Teoria Statistica Delle Classi e Calcolo Delle Probabilita." *Pubblicazioni Del R Istituto Superiore Di Scienze Economiche e Commericiali Di Firenze* 8: 3–62.

Boole, George. 1847. *The Mathematical Analysis of Logic*. Philosophical Library.

Boos, Dennis D., and Leonard A. Stefanski. 2011. "P-Value Precision and Reproducibility." *The American Statistician* 65 (4): 213–21. https://doi.org/10.1198/tas.2011.10129.

Brewer, Cynthia A. 2022. "ColorBrewer2.org." https://colorbrewer2.org.

Ezekiel, Mordecai. 1930. "Methods of Correlation Analysis."

Faraway, Julian J. 2014. *Linear Models with r*. Second. Chapman; Hall/CRC.

Fox, John, and Sanford Weisberg. 2020. "Predictor Effects Graphics Gallery." https://cran.r-project.org/web/packages/effects/vignettes/predictor-effects-gallery.pdf.

Fox, John, Sanford Weisberg, and Brad Price. 2022. *Car: Companion to Applied Regression*. https://CRAN.R-project.org/package=car.

Fox, John, Sanford Weisberg, Brad Price, Michael Friendly, and Jangman Hong. 2022. *Effects: Effect Displays for Linear, Generalized Linear, and Other Models*. https://CRAN.R-project.org/package=effects.

French, Joshua P. 2022. *Api2lm: Functions and Data Sets for the Book "a Progressive Introduction to Linear Models"*. https://CRAN.R-project.org/package=api2lm.

Gibson, Eric W. 2021. "The Role of p-Values in Judging the Strength of Evidence and Realistic Replication Expectations." *Statistics in Biopharmaceutical Research* 13 (1): 6–18. https://doi.org/10.1080/19466315.2020.1724560.

Gohel, David, and Panagiotis Skintzos. 2022. *Ggiraph: Make Ggplot2 Graphics Interactive*. https://davidgohel.github.io/ggiraph/.

Gorman, Kristen B., Tony D. Williams, and William R. Fraser. 2014. "Ecological Sexual Dimorphism and Environmental Variability Within a Community of Antarctic Penguins (Genus Pygoscelis)." *PLOS ONE* 9 (3): 1–14. https://doi.org/10.1371/journal.pone.0090081.

Henry, Lionel, and Hadley Wickham. 2022. *Purrr: Functional Programming Tools*. https://CRAN.R-project.org/package=purrr.

Horst, Allison, Alison Hill, and Kristen Gorman. 2022. *Palmerpenguins: Palmer Archipelago (Antarctica) Penguin Data*. https://CRAN.R-project.org/package=palmerpenguins.

Kutner, Michael H, Christopher J Nachtsheim, John Neter, and William Li. 2005. *Applied Linear Statistical Models, 5th Edition*. McGraw-Hill/Irwin, New York.

Mi, Jie, and Allan R. Sampson. 1993. "A Comparison of the Bonferroni and Scheffé Bounds." *Journal of Statistical Planning and Inference* 36 (1): 101–5. https://doi.org/https://doi.org/10.1016/0378-3758(93)90105-F.

Mosteller, Frederick, and John W Tukey. 1977. *Data Analysis and Regression. A Second Course in Statistics. Addison-Wesley, Reading, MA*.

Müller, Kirill, and Hadley Wickham. 2022. *Tibble: Simple Data Frames*. https://CRAN.R-project.org/package=tibble.

R Core Team. 2022. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Sall, John. 1990. "Leverage Plots for General Linear Hypotheses." *The American Statistician* 44 (4): 308–15.

Sheather, Simon. 2009. *A Modern Approach to Regression with r*. Springer, New York.

Sievert, Carson, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec, and Pedro Despouy. 2021. *Plotly: Create Interactive Web Graphics via Plotly.js*. https://CRAN.R-project.org/package=plotly.

Wasserman, Larry. 2004. *All of Statistics: A Concise Course in Statistical Inference*. Vol. 26. Springer.

Weisberg, Sanford. 2014. *Applied Linear Regression*. Fourth. Hoboken NJ: Wiley. http://z.umn.edu/alr4ed.

Wickham, Hadley. 2022a. *Forcats: Tools for Working with Categorical Variables (Factors)*. https://CRAN.R-project.org/package=forcats.

———. 2022b. *Stringr: Simple, Consistent Wrappers for Common String Operations*. https://CRAN.R-project.org/package=stringr.

———. 2022c. *The Tidyverse Style Guide*. https://style.tidyverse.org/.

———. 2022d. *Tidyverse: Easily Install and Load the Tidyverse*. https://CRAN.R-project.org/package=tidyverse.

Wickham, Hadley, and Jennifer Bryan. 2022. *Readxl: Read Excel Files*. https://CRAN.R-project.org/package=readxl.

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2022. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. https://CRAN.R-project.org/package=ggplot2.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2022. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.

Wickham, Hadley, and Maximilian Girlich. 2022. *Tidyr: Tidy Messy Data*. https://CRAN.R-project.org/package=tidyr.

Wickham, Hadley, Jim Hester, and Jennifer Bryan. 2022. *Readr: Read Rectan-*

gular Text Data. https://CRAN.R-project.org/package=readr.

Wilkinson, GN, and CE Rogers. 1973. "Symbolic Description of Factorial Models for Analysis of Variance." *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 22 (3): 392–99.

Working, Holbrook, and Harold Hotelling. 1929. "Applications of the Theory of Error to the Interpretation of Trends." *Journal of the American Statistical Association* 24 (165A): 73–85. https://doi.org/10.1080/01621459.1929.10506274.

Xie, Yihui. 2022. *Bookdown: Authoring Books and Technical Documents with r Markdown*. https://CRAN.R-project.org/package=bookdown.

Zhu, Hao. 2021. *kableExtra: Construct Complex Table with Kable and Pipe Syntax*. https://CRAN.R-project.org/package=kableExtra.