Applied Statistics with `R`

David Dalpiaz

2016-07-25

Contents

Chapter 1

Introduction

Welcome to Applied Statistics with `R`!

1.1 About This Book

This book was originally (and currently) designed for use with STAT 420, Methods of Applied Statistics, at the University of Illinois at Urbana-Champaign. It may certainly be used elsewhere, but any references to "this course" in this book specifically refer to STAT 420.

This book is under active development. When possible, it would be best to always access the text online to be sure you are using the most up-to-date version. (Also, the html version provides additional features such as changing text size, font, and colors.) If you are in need of a local copy, a **pdf version** is continuously maintained.

Since this book is under active development you may encounter errors ranging from typos to broken code to poorly explained topics. If you do, please let us know! Simply send an email and we'll make the changes ASAP. (`dalpiaz2 AT illinois DOT edu`) Or, if you know RMarkdown and are familiar with GitHub, make a pull request and fix an issue yourself! (This process is partially automated by the edit button in the top-left corner of the html version.)

This text uses MathJax to render mathematical notation for the web. Occasionally, but rarely, a JavaScript error will prevent MathJax from rendering correctly. (In which case, will see the "code" instead of the expected mathematical equations.) From experience, this is almost always fixed by simply refreshing the page. You'll also notice that if you right-click any equation you can obtain the MathML Code (for copying into Microsoft Word) or the TeX command used to generate the equation.

$$a^2 + b^2 = c^2$$

1.2 Conventions

`R` code will be typeset using a `monospace` font which is syntax highlighted.

```
a = 3
b = 4
sqrt(a ^ 2 + b ^ 2)
```

`R` output lines, which would appear in the console will begin with `##`. They will generally not be syntax highlighted.

```
## [1] 5
```

1.3 Acknowledgements

Material in this book was heavily influenced by:

- Alex Stepanov
- David Unger
- James Balamuta

Additional corrections or suggestions provided by:

- Daniel McQuillan
- Mason Rubenstein
- Yuhang Wang
- Zhao Liu

1.4 License

Chapter 2

Introduction to R

> "Measuring programming progress by lines of code is like measuring aircraft building progress by weight."
>
> — **Bill Gates**

After reading this chapter you will be able to:

- Interact with R using RStudio.
- Use R as a calculator.
- Work with data as vectors and data frames.
- Make basic data visualizations.
- Write your own R functions.
- Perform hypothesis tests using R.
- Perform basic simulations in R.

2.1 R Resources

R is both a programming language and software environment for statistical computing, which is *free* and *open-source*. To get started, you will need to install two pieces of software:

- R, the actual programming language.
  - Chose your operating system, and select the most recent version, 3.3.1.
- RStudio, an excellent IDE for working with R.
  - Note, you must have R installed to use RStudio. RStudio is simply an interface used to interact with R.

The popularity of R is on the rise, and everyday it becomes a better tool for statistical analysis. It even generated this book! (A skill you will learn in this course.) There are many good resources for learning R. They are not necessary for this course, but you may find them useful if you would like a deeper understanding of R:

- Try R from Code School.
  - An interactive introduction to the basics of R. Could be very useful for getting up to speed on R's syntax.

- Quick-R by Robert Kabacoff.
  - A good reference for `R` basics.
- R Tutorial by Chi Yau.
  - A combination reference and tutorial for `R` basics.
- R Markdown from RStudio.
  - Reference materials for RMarkdown.
- The Art of `R` Programming by Norman Matloff.
  - Gentle introduction to the programming side of `R`. (Whereas we will focus more on the data analysis side.) A free electronic version is available through the Illinois library.
- Advanced `R` by Hadley Wickham.
  - From the author of several extremely popular `R` packages. Good follow-up to The Art of `R` Programming. (And more up-to-date material.)
- `R` for Data Science by Hadley Wickham and Garrett Grolemund.
  - Similar to Advanced `R`, but focuses more on data analysis, while still introducing programming concepts. At the time of writing, currently under development.
- The R Inferno by Patrick Burns.
  - Likens learning the tricks of `R` to descending through the levels of hell. Very advanced material, but may be important if `R` becomes a part of your everyday toolkit.

RStudio has a large number of useful keyboard shortcuts. A list of these can be found using a keyboard shortcut – the keyboard shortcut to rule them all:

- On Windows: `Alt` + `Shift` + `K`
- On Mac: `Option` + `Shift` + `K`

The RStudio team has developed a number of "cheatsheets" for working with both `R` and RStudio. This particular cheatseet for Base `R` will summarize many of the concepts in this document.

When programming, it is often a good practice to follow a style guide. (Where do spaces go? Tabs or spaces? Underscores or CamelCase when naming variables?) No style guide is "correct" but it helps to be aware of what others do. The more import thing is to be consistent within your own code.

- Hadley Wickham Style Guide from Advanced `R`
- Google Style Guide

For this course, our main deviation from these two guides is the use of `=` in place of `<-`. (More on that later.)

2.2 `R` Basics

2.2.1 Basic Calculations

To get started, we'll use `R` like a simple calculator. Note, in `R` the `#` symbol is used for comments. In this book, lines which begin with two such symbols, `##`, indicate output.

Addition, Subtraction, Multiplication and Division

```r
3 + 2
```

```
## [1] 5
```

```r
3 - 2
```

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

```r
3 * 2
```

```
## [1] 6
```

```r
3 / 2
```

```
## [1] 1.5
```

Exponents

```r
3 ^ 2
```

```
## [1] 9
```

```r
2 ^ (-3)
```

```
## [1] 0.125
```

```r
100 ^ (1 / 2)
```

```
## [1] 10
```

```r
sqrt(1 / 2)
```

```
## [1] 0.7071068
```

```r
exp(1)
```

```
## [1] 2.718282
```

Mathematical Constants

```r
pi
```

```
## [1] 3.141593
```

```
exp(1)
```

```
## [1] 2.718282
```

Logarithms

```
log(10)          # natural log
```

```
## [1] 2.302585
```

```
log10(1000)      # base 10 log
```

```
## [1] 3
```

```
log2(8)          # base 2 log
```

```
## [1] 3
```

```
log(16, base = 4) # base 4 log
```

```
## [1] 2
```

Trigonometry

```
sin(pi / 2)
```

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

```
cos(0)
```

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

2.2.2 Getting Help

In using `R` as a calculator, we have seen a number of functions: `sqrt()`, `exp()`, `log()` and `sin()`. To get documentation about a function in `R`, simply put a question mark in front of the function name and RStudio will display the documentation, for example:

```
?log
?sin
?paste
?lm
```

Frequently one of the most difficult things to do when learning `R` is asking for help. First, you need to decide to ask for help, then you need to know *how* to ask for help. Your very first line of defense should be to Google your error message or a short description of your issue. (The ability to solve problems using this method is quickly becoming an extremely valuable skill.) If that fails, and it eventually will, you should ask for help. There are a number of things you should include when emailing an instructor, or posting to a help website such as Stack Exchange.

- Describe what you expect the code to do.
- State the end goal you are trying to achieve. (Sometimes what you expect the code to do, is not what you want to actually do.)
- Provide the full text of any errors you have received.
- Provide enough code to recreate the error. Often for the purpose of this course, you could simply email your entire `.R` or `.Rmd` file.
- Sometimes it is also helpful to include a screenshot of your entire RStudio window when the error occurs.

If you follow these steps, you will get your issue resolved much quicker, and possibly learn more in the process. Do not be discouraged by running into errors and difficulties when learning `R`. (Or any technical skill.) It is simply part of the learning process.

2.2.3 Installing Packages

`R` comes with a number of built-in functions and datasets, but one of the main strengths of `R` as an open-source project is its package system. Packages add additional functions and data. Frequently if you want to do something in `R`, and it isn't available by default, there is a good chance that there is a package that will fulfill your needs.

To install a package, use the `install.packages()` function. Think of this as buying a recipe book from the store, bringing it home, and putting it on your shelf.

```r
install.packages("ggplot2")
```

Once a package is installed, it must be loaded into your current `R` session before being used. Think of this as taking the book off of the shelf and opening it up to read.

```r
library(ggplot2)
```

Once you close `R`, all the packages are closed and put back on the imaginary shelf. The next time you open `R`, you do not have to install the package again, but you do have to load any packages you intend to use by invoking `library()`.

2.2.4 Data Types

`R` has a number of basic data *types*.

- Numeric
    - Also known as Double. The default type when dealing with numbers.
    - Examples: `1`, `1.0`, `42.5`
- Integer
    - Examples: `1L`, `2L`, `42L`

- Complex
  - Example: `4 + 2i`
- Logical
  - Two possible values: `TRUE` and `FALSE`
  - You can also use `T` and `F`, but this is *not* recommended.
  - `NA` is also considered logical.
- Character
  - Examples: `"a"`, `"Statistics"`, `"1 plus 2."`

`R` also has a number of basic data *structures*. A data structure is either homogeneous (all elements are of the same data type) or heterogeneous (elements can be of more than one data type).

Dimension	**Homogeneous**	**Heterogeneous**
1	Vector	List
2	Matrix	Data Frame
3+	Array	

2.2.5 Vectors

Many operations in `R` make heavy use of **vectors**. Vectors in `R` are indexed starting at `1`. That is what the `[1]` in the output is indicating, that the first element of the row being displayed is the first element of the vector. Larger vectors will start additional rows with `[*]` where `*` is the index of the first element of the row.

Possibly the most common way to create a vector in `R` is using the `c()` function, which is short for "combine."" As the name suggests, it combines a list of numbers separated by commas.

```
c(1, 3, 5, 7, 8, 9)
```

```
## [1] 1 3 5 7 8 9
```

Here `R` simply outputs this vector. If we would like to store this vector in a **variable** we can do so with the **assignment** operator `=`. In this case the variable `x` now holds the vector we just created, and we can access the vector by typing `x`.

```
x = c(1, 3, 5, 7, 8, 9)
x
```

```
## [1] 1 3 5 7 8 9
```

As an aside, there is a long history of the assignment operator in `R`. For simplicity we will use `=`, but know that often you will see `<-` as the assignment operator. The pros and cons of these two are well beyond the scope of this book, but know that for our purposes you will have no issue if you simply use `=`.

Frequently you may wish to create a vector based on a sequence of numbers. The quickest and easiest way to do this is with the `:` operator, which creates a sequence of integers between two specified integers.

```
(y = 1:100)
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100
```

Here we see `R` labeling the rows after the first since this is a large vector. Also, we see that by putting parentheses around the assignment, `R` both stores the vector in a variable called `y` and automatically outputs `y` to the console.

To subset a vector, we use square brackets, `[]`.

```
x
```

```
## [1] 1 3 5 7 8 9
```

```
x[1]
```

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

```
x[3]
```

```
## [1] 5
```

We see that `x[1]` returns the first element, and `x[3]` returns the third element.

```
x[-2]
```

```
## [1] 1 5 7 8 9
```

We can also exclude certain indexes, in this case the second element.

```
x[1:3]
```

```
## [1] 1 3 5
```

```
x[c(1,3,4)]
```

```
## [1] 1 5 7
```

Lastly we see that we can subset based on a vector of indices.

One of the biggest strengths of `R` is its use of vectorized operations. (Frequently the lack of understanding of this concept leads of a belief that `R` is *slow*. `R` is not the fastest language, but it has a reputation for being slower than it really is.)

```r
x = 1:10
x + 1
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11
```

```r
2 * x
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```r
2 ^ x
```

```
##  [1]    2    4    8   16   32   64  128  256  512 1024
```

```r
sqrt(x)
```

```
##  [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
##  [9] 3.000000 3.162278
```

```r
log(x)
```

```
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
##  [8] 2.0794415 2.1972246 2.3025851
```

We see that when a function like `log()` is called on a vector `x`, a vector is returned which has applied the function to each element of the vector `x`.

```r
vec_1 = 1:10
vec_2 = 1:1000
vec_3 = 42
```

The length of a vector can be obtained with the `length()` function.

```r
length(vec_1)
```

```
## [1] 10
```

```r
length(vec_2)
```

```
## [1] 1000
```

```r
length(vec_3)
```

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

Note that scalars do not exists in `R`. They are simply vectors of length `1`.

If we want to create a sequence that isn't limited to integers and increasing by 1 at a time, we can use the `seq()` function.

```r
seq(from = 1.5, to = 4.2, by = 0.1)
```

```
##  [1] 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3
## [20] 3.4 3.5 3.6 3.7 3.8 3.9 4.0 4.1 4.2
```

We will discuss functions in detail later, but note here that the input labels `from`, `to`, and `by` are optional.

```r
seq(1.5, 4.2, 0.1)
```

```
##  [1] 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3
## [20] 3.4 3.5 3.6 3.7 3.8 3.9 4.0 4.1 4.2
```

Another common operation to create a vector is `rep()`, which can repeat a single value a number of times.

```r
rep("A", times = 10)
```

```
##  [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
```

Or, `rep()` can be used to repeat a vector a number of times.

```r
rep(x, times = 3)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5
## [26]  6  7  8  9 10
```

We have now seen four different ways to create vectors:

- `c()`
- `:`
- `seq()`
- `rep()`

So far we have mostly used them in isolation, but they are often used together.

```r
c(x, rep(seq(1, 9, 2), 3), c(1, 2, 3), 42, 2:4)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10  1  3  5  7  9  1  3  5  7  9  1  3  5  7  9
## [26]  1  2  3 42  2  3  4
```

2.2.6 Summary Statistics

`R` has built in functions for a large number of summary statistics.

```r
y
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100
```

Central Tendency

```r
mean(y)
```

```
## [1] 50.5
```

```r
median(y)
```

```
## [1] 50.5
```

Spread

```r
var(y)
```

```
## [1] 841.6667
```

```r
sd(y)
```

```
## [1] 29.01149
```

```r
IQR(y)
```

```
## [1] 49.5
```

```r
min(y)
```

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

```r
max(y)
```

```
## [1] 100
```

```r
range(y)
```

```
## [1]   1 100
```

2.2.7 Matrices

`R` can also be used for **matrix** calculations. Matrices have rows and columns containing a single data type. In a matrix, the order of rows and columns is important. (This is not true of *data frames*, which we will see later.)

Matrices can be created using the `matrix` function.

```
x = 1:9
x
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

```
X = matrix(x, nrow = 3, ncol = 3)
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Note here that we are using two different variables: lower case `x`, which stores a vector and capital `X`, which stores a matrix. (Following the usual mathematical convention.) We can do this because `R` is case sensitive.

By default the `matrix` function reorders a vector into columns, but we can also tell `R` to use rows instead.

```
Y = matrix(x, nrow = 3, ncol = 3, byrow = TRUE)
Y
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

We can also create a matrix of a specified dimension where every element is the same, in this case `0`.

```
Z = matrix(0, 2, 4)
Z
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    0    0
## [2,]    0    0    0    0
```

Like vectors, matrices can be subsetted using square brackets, `[]`. However, since matrices are two-dimensional, we need to specify both a row and a column when subsetting.

```
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
X[1, 2]
```

```
## [1] 4
```

Here we accessed the element in the first row and the second column. We could also subset an entire row or column.

```
X[1, ]
```

```
## [1] 1 4 7
```

```
X[, 2]
```

```
## [1] 4 5 6
```

We can also use vectors to subset more than one row or column at a time. Here we subset to the first and third column of the second row.

```
X[2, c(1, 3)]
```

```
## [1] 2 8
```

Matrices can also be created by combining vectors as columns, using `cbind`, or combining vectors as rows, using `rbind`.

```
x = 1:9
rev(x)
```

```
## [1] 9 8 7 6 5 4 3 2 1
```

```
rep(1, 9)
```

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

```
cbind(x, rev(x), rep(1, 9))
```

```
##        x
##  [1,] 1 9 1
##  [2,] 2 8 1
##  [3,] 3 7 1
##  [4,] 4 6 1
##  [5,] 5 5 1
##  [6,] 6 4 1
##  [7,] 7 3 1
##  [8,] 8 2 1
##  [9,] 9 1 1
```

```
rbind(x, rev(x), rep(1, 9))
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## x    1    2    3    4    5    6    7    8    9
##      9    8    7    6    5    4    3    2    1
##      1    1    1    1    1    1    1    1    1
```

`R` can then be used to perform matrix calculations.

```
x = 1:9
y = 9:1
X = matrix(x, 3, 3)
Y = matrix(y, 3, 3)
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
Y
```

```
##      [,1] [,2] [,3]
## [1,]    9    6    3
## [2,]    8    5    2
## [3,]    7    4    1
```

```
X + Y
```

```
##      [,1] [,2] [,3]
## [1,]   10   10   10
## [2,]   10   10   10
## [3,]   10   10   10
```

```
X - Y
```

```
##      [,1] [,2] [,3]
## [1,]   -8   -2    4
## [2,]   -6    0    6
## [3,]   -4    2    8
```

```
X * Y
```

```
##      [,1] [,2] [,3]
## [1,]    9   24   21
## [2,]   16   25   16
## [3,]   21   24    9
```

```
X / Y
```

```
##           [,1]      [,2]     [,3]
## [1,] 0.1111111 0.6666667 2.333333
## [2,] 0.2500000 1.0000000 4.000000
## [3,] 0.4285714 1.5000000 9.000000
```

Note that `X * Y` is not matrix multiplication. It is element by element multiplication. (Same for `X / Y`). Instead, matrix multiplication uses `%*%`. Other matrix functions include `t()` which gives the transpose of a matrix and `solve()` which returns the inverse of a square matrix if it is invertible.

```r
X %*% Y
```

```
##      [,1] [,2] [,3]
## [1,]   90   54   18
## [2,]  114   69   24
## [3,]  138   84   30
```

```r
t(X)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

```r
Z = matrix(c(9, 2, -3, 2, 4, -2, -3, -2, 16), 3, byrow = TRUE)
Z
```

```
##      [,1] [,2] [,3]
## [1,]    9    2   -3
## [2,]    2    4   -2
## [3,]   -3   -2   16
```

```r
solve(Z)
```

```
##             [,1]        [,2]       [,3]
## [1,]  0.12931034 -0.05603448 0.01724138
## [2,] -0.05603448  0.29094828 0.02586207
## [3,]  0.01724138  0.02586207 0.06896552
```

R has a number of matrix specific functions for obtaining dimension and summary information.

```r
X = matrix(1:6, 2, 3)
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```r
dim(X)
```

```
## [1] 2 3
```

```r
rowSums(X)
```

```
## [1]  9 12
```

```r
colSums(X)
```

```
## [1]  3  7 11
```

```r
rowMeans(X)
```

```
## [1] 3 4
```

```r
colMeans(X)
```

```
## [1] 1.5 3.5 5.5
```

The `diag()` function can be used in a number of ways. We can extract the diagonal of a matrix.

```r
diag(Z)
```

```
## [1]  9  4 16
```

Or create a matrix with specified elements on the diagonal. (And `0` on the off-diagonals.)

```r
diag(1:5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    2    0    0    0
## [3,]    0    0    3    0    0
## [4,]    0    0    0    4    0
## [5,]    0    0    0    0    5
```

Or, lastly, create a square matrix of a certain dimension with `1` for every element of the diagonal and `0` for the off-diagonals.

```r
diag(5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    1    0    0    0
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
## [5,]    0    0    0    0    1
```

2.2.8 Data Frames

We have previously seen vectors and matrices for storing data as we introduced `R`. We will now introduce a **data frame** which will be the most common way that we store and interact with data in this course.

```r
example_data = data.frame(x = c(1, 3, 5, 7, 9, 1, 3, 5, 7, 9),
                          y = rep("Hello", 10),
                          z = rep(c("TRUE", "FALSE"), 5))
```

Unlike a matrix, which can be thought of as a vector rearranged into rows and columns, a data frame is not required to have the same data type for each element. A data frame is a **list** of vectors. So, each vector must contain the same data type, but the different vectors can store different data types.

```
example_data
```

```
##     x     y     z
## 1   1 Hello  TRUE
## 2   3 Hello FALSE
## 3   5 Hello  TRUE
## 4   7 Hello FALSE
## 5   9 Hello  TRUE
## 6   1 Hello FALSE
## 7   3 Hello  TRUE
## 8   5 Hello FALSE
## 9   7 Hello  TRUE
## 10 9 Hello FALSE
```

The `data.frame()` function above is one way to create a data frame. We can also import data from various file types in into `R`, as well as use data stored in packages.

The example data above can also be found here as a .csv file. To read this data into `R`, we would use the `read.csv()` function.

```
example_data_from_csv = read.csv("data/example_data.csv")
```

This particular line of code assumes that the file `example_data.csv` exists in a folder called `data` in your current working directory.

Alternatively, we could use the "Import Dataset" feature in RStudio which can be found in the environment window. (By default, the top-right pane of RStudio.)

Once completed, this process will automatically generate the code to import a file. The resulting code will be shown in the console window.

Earlier we looked at installing packages, in particular the `ggplot2` package. (A package for visualization. While not necessary for this course, it is quickly growing in popularity.)

```
library(ggplot2)
```

Inside the `ggplot2` package is a dataset called `mpg`. By loading the package using the `library()` function, we can now access `mpg`.

When using data from inside a package, there are three things we would generally like to do:

- Look at the raw data.
- Understand the data. (Where did it come from? What are the variables? Etc.)
- Visualize the data.

To look at the data, we have two useful commands: `head()` and `str()`.

```
head(mpg, n = 10)
```

```
##    manufacturer      model displ year cyl      trans drv cty hwy fl   class
## 1          audi         a4   1.8 1999   4   auto(l5)   f  18  29  p compact
## 2          audi         a4   1.8 1999   4 manual(m5)   f  21  29  p compact
## 3          audi         a4   2.0 2008   4 manual(m6)   f  20  31  p compact
```

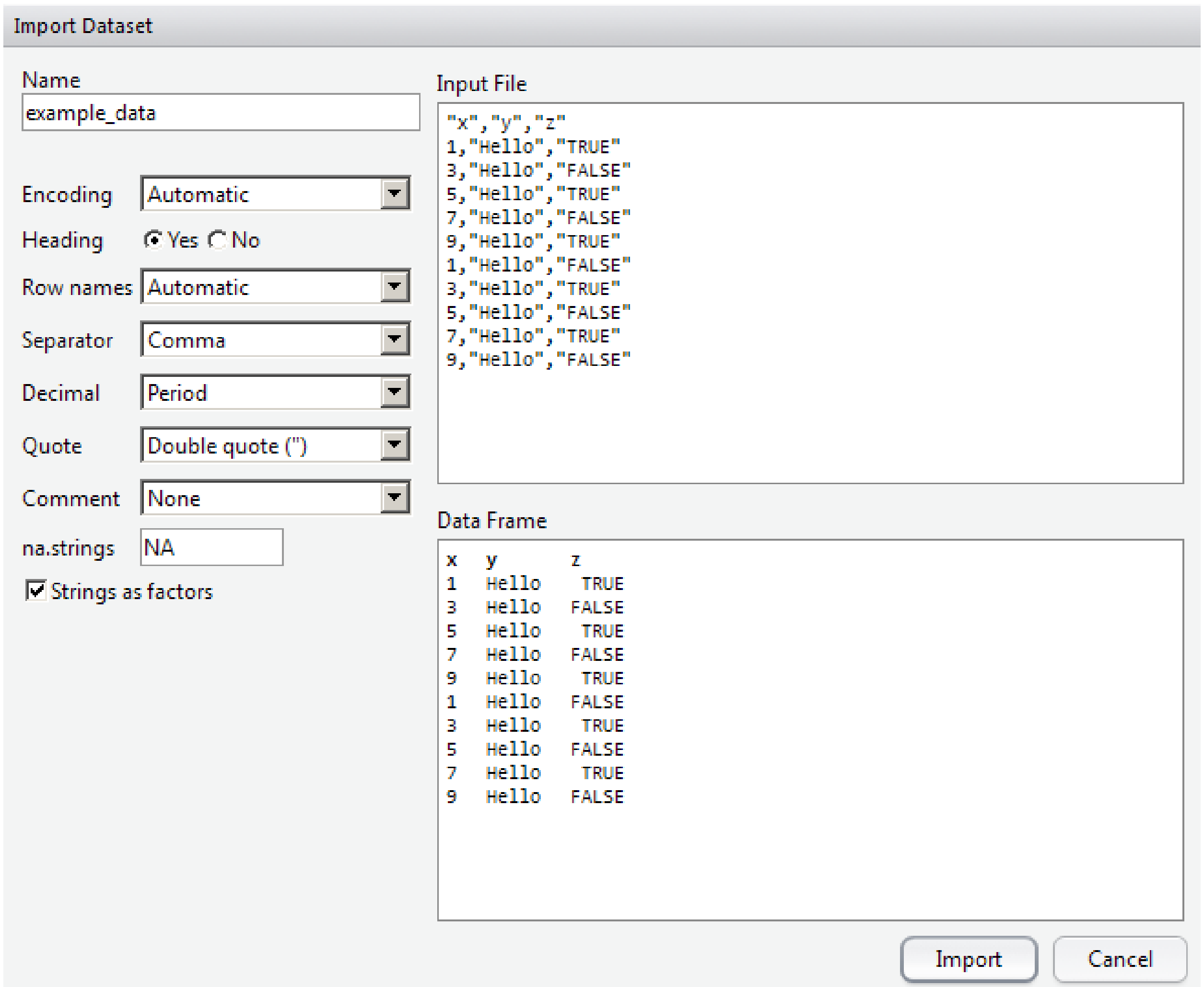


Figure 2.1: RStudio Import Screen

```
## 4          audi          a4   2.0 2008   4   auto(av)   f  21  30  p compact
## 5          audi          a4   2.8 1999   6   auto(l5)   f  16  26  p compact
## 6          audi          a4   2.8 1999   6 manual(m5)   f  18  26  p compact
## 7          audi          a4   3.1 2008   6   auto(av)   f  18  27  p compact
## 8          audi a4 quattro   1.8 1999   4 manual(m5)   4  18  26  p compact
## 9          audi a4 quattro   1.8 1999   4   auto(l5)   4  16  25  p compact
## 10         audi a4 quattro   2.0 2008   4 manual(m6)   4  20  28  p compact
```

The function `head()` will display the first `n` observations of the data frame.

```
str(mpg)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    234 obs. of  11 variables:
##  $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
##  $ model       : chr  "a4" "a4" "a4" "a4" ...
##  $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
##  $ year        : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
##  $ cyl         : int  4 4 4 4 6 6 6 4 4 4 ...
##  $ trans       : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
##  $ drv         : chr  "f" "f" "f" "f" ...
##  $ cty         : int  18 21 20 21 16 18 18 18 16 20 ...
##  $ hwy         : int  29 29 31 30 26 26 27 26 25 28 ...
##  $ fl          : chr  "p" "p" "p" "p" ...
##  $ class       : chr  "compact" "compact" "compact" "compact" ...
```

The function `str()` will display the "structure" of the data frame. It will display the number of **observations** and **variables**, list the variables, give the type of each variable, and show some elements of each variable.

It is important to note that while matrices have rows and columns, data frames instead have observations and variables. When displayed in the console or viewer, each row is an observation and each column is a variable. However generally speaking, their order does not matter, it is simply a side-effect of how the data was entered or stored.

In this dataset an observation is for a particular model-year of a car, and the variables describe attributes of the car, for example its highway fuel efficiency.

To understand more about the data set, we use the `?` operator to pull up the documentation for the data.

```
?mpg
```

`R` has a number of functions for quickly working with and extracting basic information from data frames. To quickly obtain a vector of the variable names, we use the `names()` function.

```
names(mpg)
```

```
##  [1] "manufacturer" "model"        "displ"        "year"         "cyl"
##  [6] "trans"        "drv"          "cty"          "hwy"          "fl"
## [11] "class"
```

To access one of the variables as a vector, we use the `$` operator.

```r
mpg$year
```

```
##   [1] 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 2008 1999 1999 2008 2008
##  [16] 1999 2008 2008 2008 2008 2008 1999 2008 1999 1999 2008 2008 2008 2008 2008
##  [31] 1999 1999 1999 2008 1999 2008 2008 1999 1999 1999 1999 2008 2008 2008 1999
##  [46] 1999 2008 2008 2008 2008 1999 1999 2008 2008 2008 1999 1999 1999 2008 2008
##  [61] 2008 1999 2008 1999 2008 2008 2008 2008 2008 2008 1999 1999 2008 1999 1999
##  [76] 1999 2008 1999 1999 1999 2008 2008 1999 1999 1999 1999 1999 2008 1999 2008
##  [91] 1999 1999 2008 2008 1999 1999 2008 2008 2008 1999 1999 1999 1999 1999 2008
## [106] 2008 2008 2008 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 2008 2008
## [121] 2008 2008 2008 2008 1999 1999 2008 2008 2008 2008 1999 2008 2008 1999 1999
## [136] 1999 2008 1999 2008 2008 1999 1999 1999 2008 2008 2008 2008 1999 1999 2008
## [151] 1999 1999 2008 2008 1999 1999 1999 2008 2008 1999 1999 2008 2008 2008 2008
## [166] 1999 1999 1999 1999 2008 2008 2008 2008 1999 1999 1999 1999 2008 2008 1999
## [181] 1999 2008 2008 1999 1999 2008 1999 1999 2008 2008 1999 1999 2008 1999 1999
## [196] 1999 2008 2008 1999 2008 1999 1999 2008 1999 1999 2008 2008 1999 1999 2008
## [211] 2008 1999 1999 1999 1999 2008 2008 2008 2008 1999 1999 1999 1999 1999 1999
## [226] 2008 2008 1999 1999 2008 2008 1999 1999 2008
```

```r
mpg$hwy
```

```
##   [1] 29 29 31 30 26 26 27 26 25 28 27 25 25 25 25 24 25 23 20 15 20 17 17 26 23
##  [26] 26 25 24 19 14 15 17 27 30 26 29 26 24 24 22 22 24 24 17 22 21 23 23 19 18
##  [51] 17 17 19 19 12 17 15 17 17 12 17 16 18 15 16 12 17 17 16 12 15 16 17 15 17
##  [76] 17 18 17 19 17 19 19 17 17 17 16 16 17 15 17 26 25 26 24 21 22 23 22 20 33
## [101] 32 32 29 32 34 36 36 29 26 27 30 31 26 26 28 26 29 28 27 24 24 24 22 19 20
## [126] 17 12 19 18 14 15 18 18 15 17 16 18 17 19 19 17 29 27 31 32 27 26 26 25 25
## [151] 17 17 20 18 26 26 27 28 25 25 24 27 25 26 23 26 26 26 26 25 27 25 27 20 20
## [176] 19 17 20 17 29 27 31 31 26 26 28 27 29 31 31 26 26 27 30 33 35 37 35 15 18
## [201] 20 20 22 17 19 18 20 29 26 29 29 24 44 29 26 29 29 29 29 23 24 44 41 29 26
## [226] 28 29 29 29 28 29 26 26 26
```

We can use the `dim()`, `nrow()` and `ncol()` functions to obtain information about the dimension of the data frame.

```r
dim(mpg)
```

```
## [1] 234  11
```

```r
nrow(mpg)
```

```
## [1] 234
```

```r
ncol(mpg)
```

```
## [1] 11
```

Here `nrow()` is also the number of observations, which in most cases is the *sample size*.

Subsetting data frames can work much like subsetting matrices using square brackets, `[,]`. Here, we find fuel efficient vehicles earning over 35 miles per gallon and only display `manufacturer`, `model` and `year`.

```
mpg[mpg$hwy > 35, c("manufacturer", "model", "year")]
```

```
##     manufacturer      model year
## 106        honda      civic 2008
## 107        honda      civic 2008
## 197       toyota    corolla 2008
## 213   volkswagen      jetta 1999
## 222   volkswagen new beetle 1999
## 223   volkswagen new beetle 1999
```

An alternative would be to use the `subset()` function, which has a much more readable syntax.

```
subset(mpg, subset = hwy > 35, select = c("manufacturer", "model", "year"))
```

Lastly, we could use the `filter` and `select` functions from the `dplyr` package which introduces the `%>%` operator from the `magrittr` package. This is not necessary for this course, however the `dplyr` package is something you should be aware of as it is becoming a popular tool in the `R` world.

```
library(dplyr)
mpg %>% filter(hwy > 35) %>% select(manufacturer, model, year)
```

All three approaches produce the same results. Which you use will be largely based on a given situation as well as user preference.

2.2.9 Plotting

Now that we have some data to work with, and we have learned about the data at the most basic level, our next tasks is to visualize the data. Often, a proper visualization can illuminate features of the data that can inform further analysis.

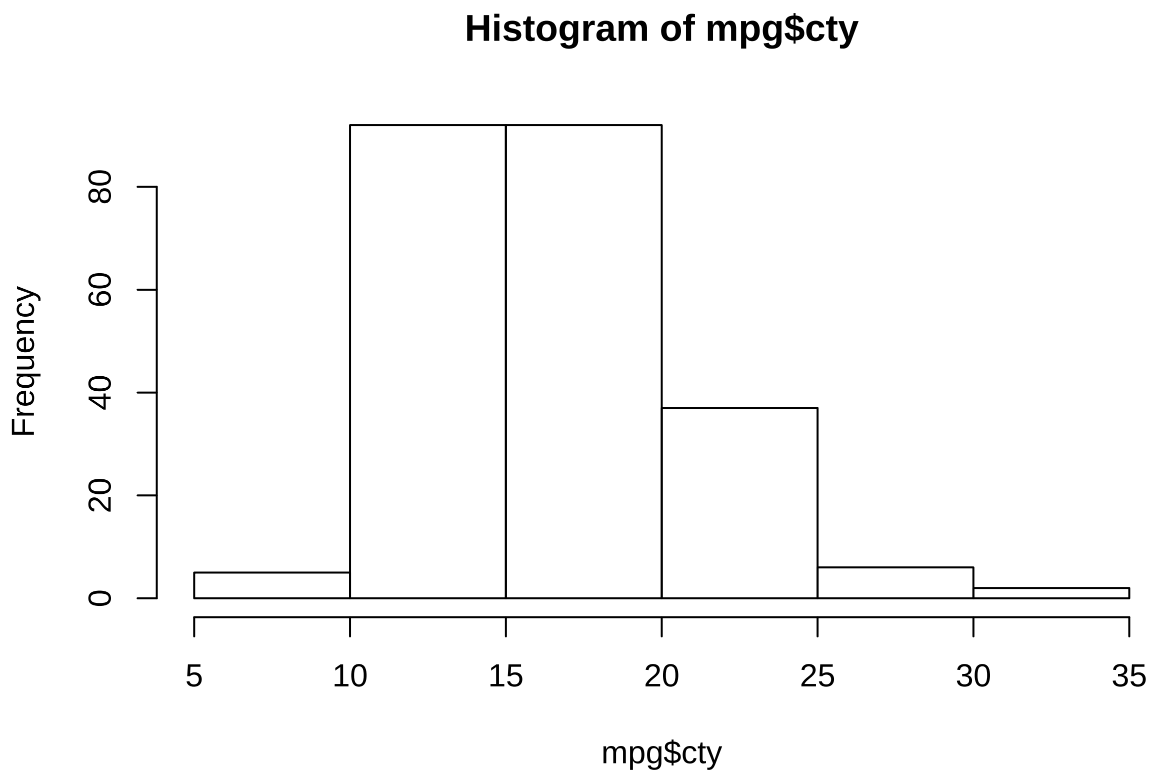
We will look at three methods of visualizing data that we will use throughout the course:

- Histograms
- Boxplots
- Scatterplots

2.2.9.1 Histograms

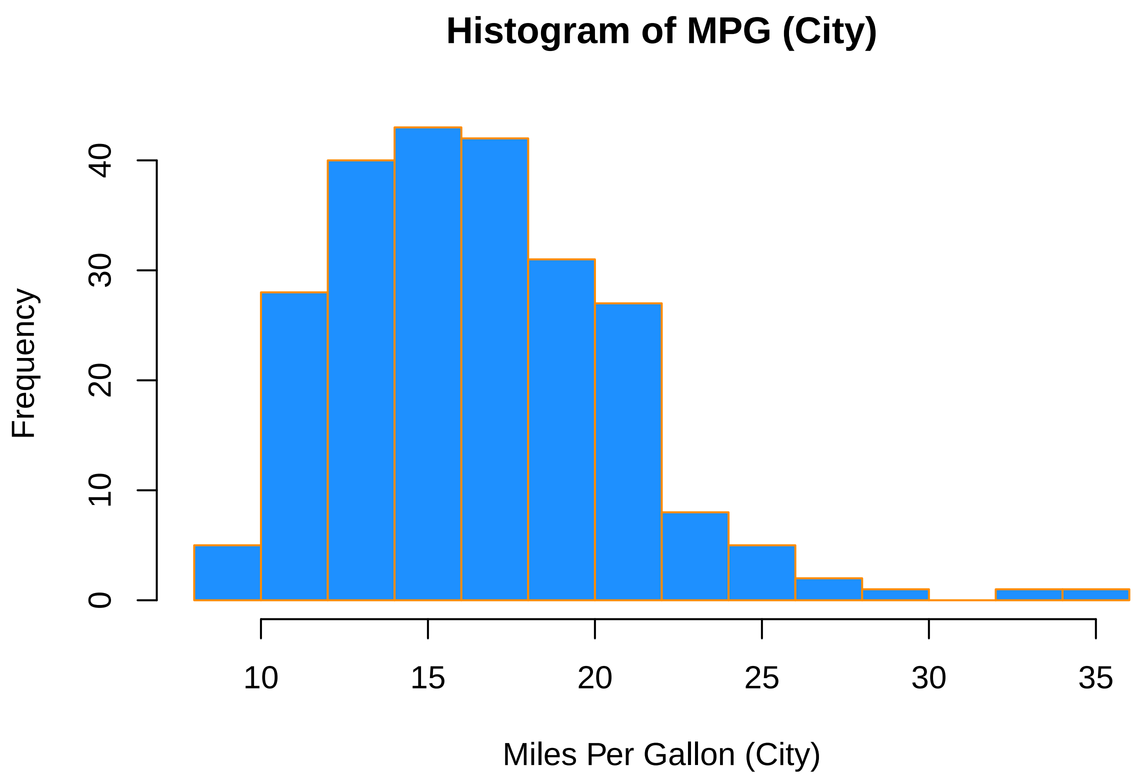
When visualizing a single numerical variable, a **histogram** will be our go-to tool, which can be created in `R` using the `hist()` function.

```
hist(mpg$cty)
```



The histogram function has a number of parameters which can be changed to make our plot look much nicer. Use the `?` operator to read the documentation for the `hist()` to see a full list of these parameters.

```
hist(mpg$cty,
     xlab   = "Miles Per Gallon (City)",
     main   = "Histogram of MPG (City)",
     breaks = 12,
     col    = "dodgerblue",
     border = "darkorange")
```



Importantly, you should always be sure to label your axes and give the plot a title. The argument `breaks` is specific to `hist()`. Entering an integer will give a suggestion to `R` for how many bars to use for the histogram. By default `R` will attempt to intelligently guess a good number of `breaks`, but as we can see here, it is sometimes useful to modify this yourself.

2.2.9.2 Boxplots

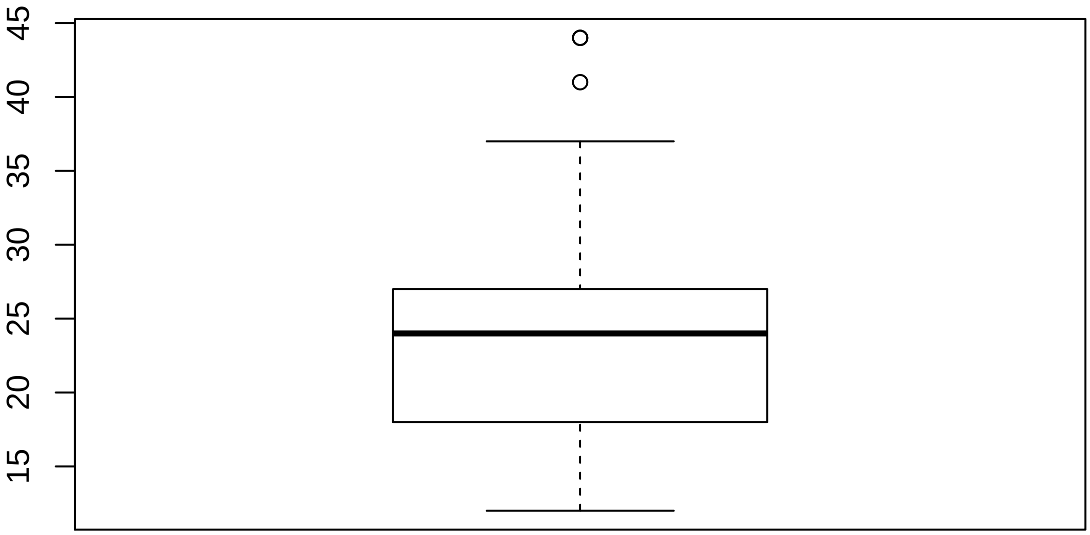
To visualize the relationship between a numerical and categorical variable, we will use a **boxplot**. In the `mpg` dataset, the `drv` variable takes a small, finite number of values. A car can only be front wheel drive, 4 wheel drive, or rear wheel drive.

```r
unique(mpg$drv)
```

```
## [1] "f" "4" "r"
```

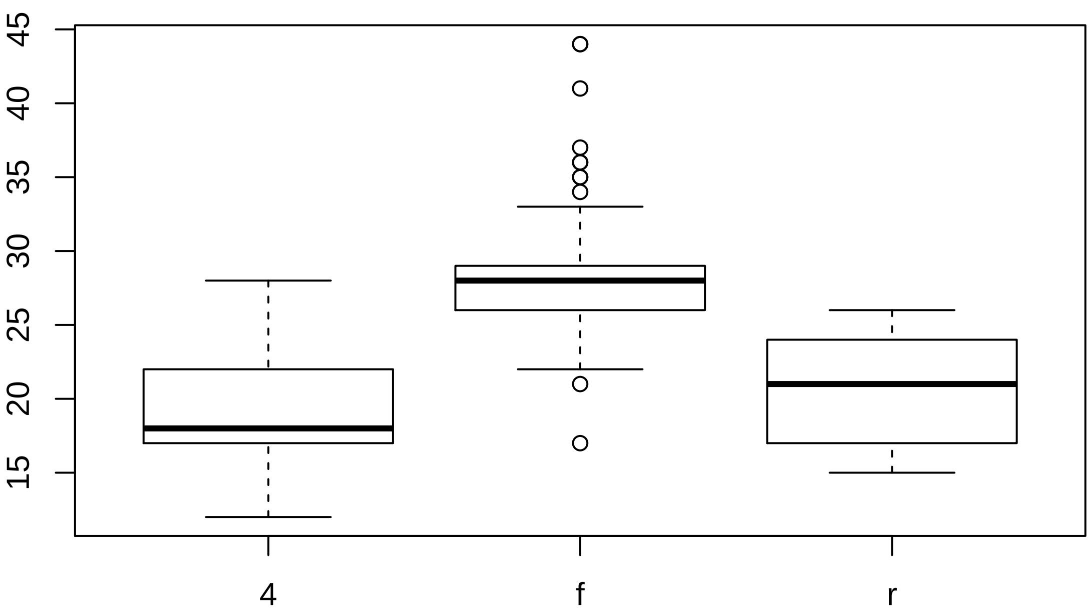
First note that we can use a single boxplot as an alternative to a histogram for visualizing a single numerical variable. To do so in `R`, we use the `boxplot()` function.

```r
boxplot(mpg$hwy)
```



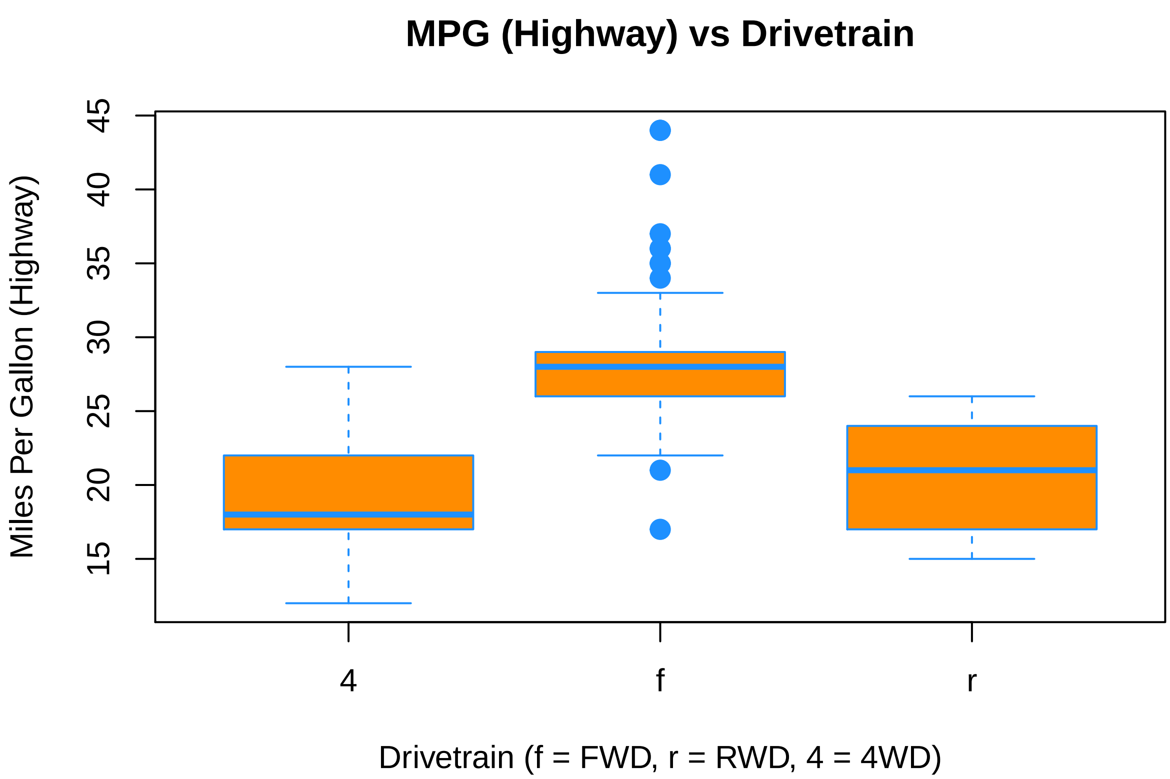
However, more often we will use boxplots to compare a numerical variable for different values of a categorical variable.

```
boxplot(hwy ~ drv, data = mpg)
```



Here used the `boxplot()` command to create side-by-side boxplots. However, since we are now dealing with two variables, the syntax has changed. The R syntax `hwy ~ drv, data = mpg` reads "Plot the `hwy` variable against the `drv` variable using the dataset `mpg`." We see the use of a `~` (which specifies a formula) and also a `data =` argument. This will be a syntax that is common to many functions we will use in this course.

```
boxplot(hwy ~ drv, data = mpg,
     xlab   = "Drivetrain (f = FWD, r = RWD, 4 = 4WD)",
     ylab   = "Miles Per Gallon (Highway)",
     main   = "MPG (Highway) vs Drivetrain",
     pch    = 20,
     cex    = 2,
     col    = "darkorange",
     border = "dodgerblue")
```

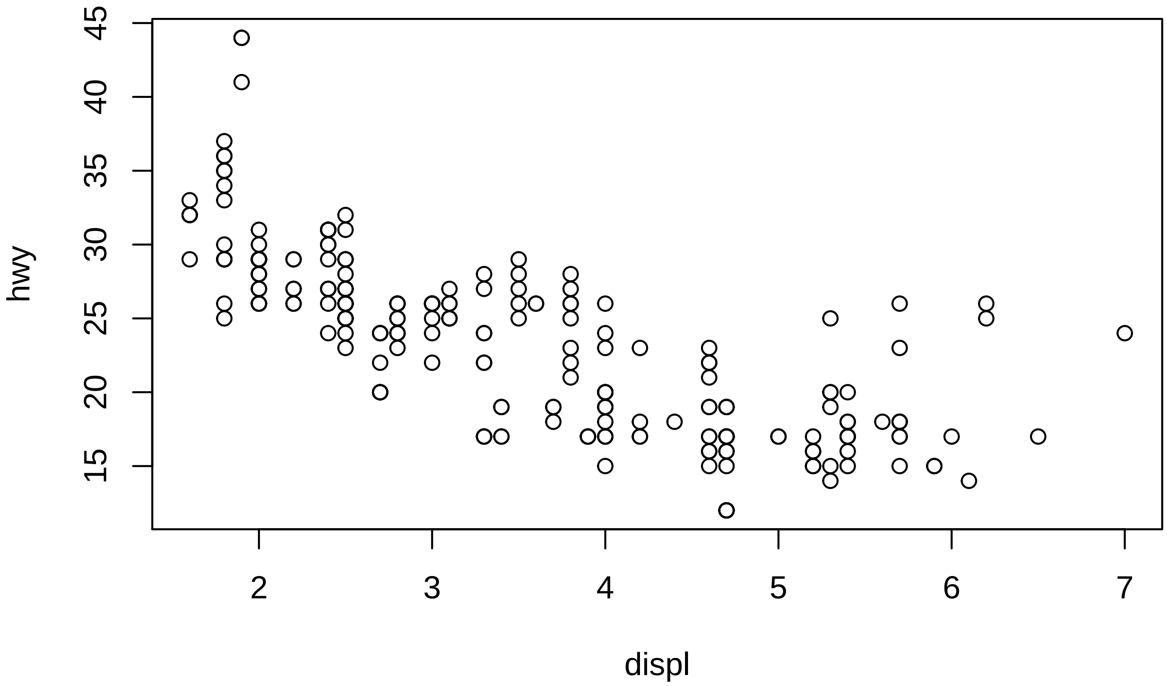


Again, `boxplot()` has a number of additional arguments which have the ability to make our plot more visually appealing.

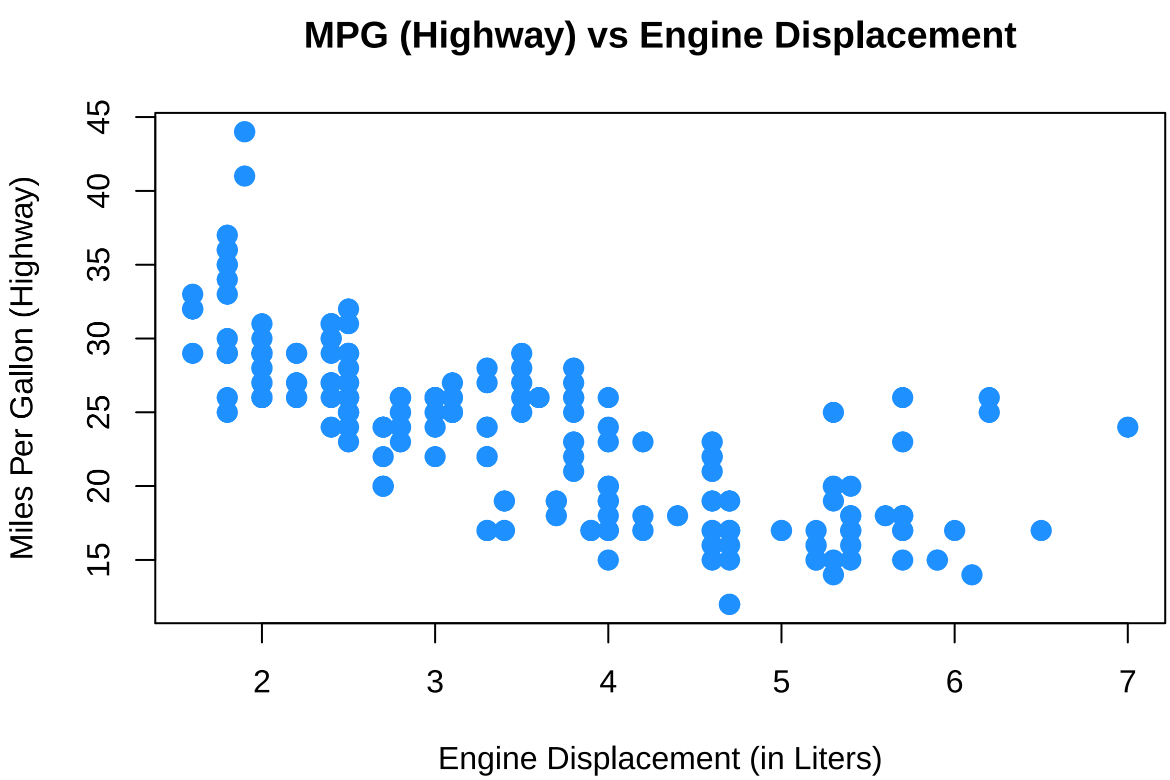
2.2.9.3 Scatterplots

Lastly, to visualize the relationship between two numeric variables we will use a **scatterplot**. This can be done with the `plot()` function and the `~` syntax we just used with a boxplot. (The function `plot()` can also be used more generally; see the documentation for details.)

```
plot(hwy ~ displ, data = mpg)
```




```
plot(hwy ~ displ, data = mpg,
     xlab = "Engine Displacement (in Liters)",
     ylab = "Miles Per Gallon (Highway)",
     main = "MPG (Highway) vs Engine Displacement",
     pch  = 20,
     cex  = 2,
     col  = "dodgerblue")
```



2.2.10 Distributions

When working with different statistical distributions, we often want to make probabilistic statements based on the distribution.

We typically want to know one of four things:

- The density (pdf) at a particular value.
- The distribution (cdf) at a particular value.
- The quantile value corresponding to a particular probability.
- A random draw of values from a particular distribution.

This used to be done with statistical tables printed in the back of textbooks. Now, `R` has functions for obtaining density, distribution, quantile and random values.

The general naming structure of the relevant `R` functions is:

- `dname` calculates density (pdf) at input `x`.
- `pname` calculates distribution (cdf) at input `x`.
- `qname` calculates the quantile at an input probability.
- `rname` generates a random draw from a particular distribution.

Note that `name` represents the name of the given distribution.

For example, consider a random variable X which is $N(\mu = 2, \sigma^2 = 25)$. (Note, we are parameterizing using the variance σ^2. `R` however uses the standard deviation.)

To calculate the value of the pdf at `x = 3`, that is, the height of the curve at `x = 3`, use:

```
dnorm(x = 3, mean = 2, sd = 5)
```

```
## [1] 0.07820854
```

To calculate the value of the cdf at `x = 3`, that is, $P(X \leq 3)$, the probability that X is less than or equal to `3`, use:

```
pnorm(q = 3, mean = 2, sd = 5)
```

```
## [1] 0.5792597
```

Or, to calculate the quantile for probability 0.975, use:

```
qnorm(p = 0.975, mean = 2, sd = 5)
```

```
## [1] 11.79982
```

Lastly, to generate a random sample of size `n = 10`, use:

```
rnorm(n = 10, mean = 2, sd = 5)
```

```
##  [1]  2.8058623  2.8562568 -0.8649388 -0.7076853  3.4211776  1.9255361
##  [7] -6.2589029  7.0421879 -2.7228485 -3.4093051
```

These functions exist for many other distributions, including but not limited to:

Command	Distribution
`*binom`	Binomial
`*t`	t
`*pois`	Poisson
`*f`	F
`*chisq`	Chi-Squared

Where `*` can be `d`, `p`, `q`, and `r`. Each distribution will have its own set of parameters which need to be passed to the functions as arguments. For example, `dbinom()` would not have arguments for `mean` and `sd`, since those are not parameters of the distribution. Instead a binomial distribution is usually parameterized by n and p, however `R` chooses to call them something else. To find the names that `R` uses we would use `?dbinom` and see that `R` instead calls the arguments `size` and `prob`. For example:

```
dbinom(x = 6, size = 10, prob = 0.75)
```

```
## [1] 0.145998
```

Also note that, when using the `dname` functions with discrete distributions, they are the pmf of the distribution. For example, the above command is $P(Y = 6)$ if $Y \sim b(n = 10, p = 0.75)$. (The probability of flipping an unfair coin `10` times and seeing `6` heads, if the probability of heads is `0.75`.)

2.3 Programming Basics

2.3.1 Logical Operators

Operator	Summary	Example	Result
`x < y`	`x` less than `y`	`3 < 42`	TRUE
`x > y`	`x` greater than `y`	`3 > 42`	FALSE
`x <= y`	`x` less than or equal to `y`	`3 <= 42`	TRUE
`x >= y`	`x` greater than or equal to `y`	`3 >= 42`	FALSE
`x == y`	`x`equal to `y`	`3 == 42`	FALSE
`x != y`	`x` not equal to `y`	`3 != 42`	TRUE
`!x`	not `x`	`!(3 > 42)`	TRUE
`x | y`	`x` or `y`	`(3 > 42) | TRUE`	TRUE
`x & y`	`x` and `y`	`(3 < 4) & ( 42 > 13)`	TRUE

In `R`, logical operators are vectorized. To demonstrate this, we will use the following height and weight data.

```
heights = c(110, 120, 115, 136, 205, 156, 175)
weights = c(64, 67, 62, 60, 77, 70, 66)
```

First, using the `<` operator, when can find which `heights` are less than `121`. Further, we could also find which `heights` are less than `121` or exactly equal to `156`.

```
heights < 121
```

```
## [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

```
heights < 121 | heights == 156
```

```
## [1]  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE
```

Often, a vector of logical values is useful for subsetting a vector. For example, we can find the `heights` that are larger than `150`. We can then use the resulting vector to subset the `heights` vector, thus actually returning the `heights` that are above `150`, instead of a vector of which values are above `150`. Here we also obtain the `weights` corresponding to `heights` above `150`.

```
heights > 150
```

```
## [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

```
heights[heights > 150]
```

```
## [1] 205 156 175
```

```
weights[heights > 150]
```

```
## [1] 77 70 66
```

When comparing vectors, be sure you are comparing vectors of the same length.

```
a = 1:10
b = 2:4
a < b
```

```
## Warning in a < b: longer object length is not a multiple of shorter object
## length
```

```
##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

What happened here? `R` still performed the operation, but it also gives us a warning. (To perform the operation, `R` automatically made `b` longer by repeating `b` as needed.)

The one exception to this behavior is comparing to a vector of length `1`. `R` does not warn us in this case, as comparing each value of a vector to a single value is a common operation that is usually reasonable to perform.

```
a > 5
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Often we will want to convert `TRUE` and `FALSE` values to `1` and `0`. When performing mathematical operations on `TRUE` and `FALSE`, this is done automatically through type coercion.

```
5 + (a > 5)
```

```
##  [1] 5 5 5 5 5 6 6 6 6 6
```

By calling `sum()` on a vector of logical values, we can essentially count the number of `TRUE` values.

```
sum(a > 5)
```

```
## [1] 5
```

Here we count the elements of `a` that are larger than `5`. This is an extremely useful feature.

2.3.2 Control Flow

In `R`, the if/else syntax is:

```
if (...) {
  some R code
} else {
  more R code
}
```

For example,

```
x = 1
y = 3
if (x > y) {
  z = x * y
  print("x is larger than y")
} else {
  z = x + 5 * y
  print("x is less than or equal to y")
}
```

```
## [1] "x is less than or equal to y"
```

```
z
```

```
## [1] 16
```

`R` also has a special function `ifelse()` which is very useful. It returns one of two specified values based on a conditional statement.

```
ifelse(4 > 3, 1, 0)
```

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

The real power of `ifelse()` comes from its ability to be applied to vectors.

```
fib = c(1, 1, 2, 3, 5, 8, 13, 21)
ifelse(fib > 6, "Foo", "Bar")
```

```
## [1] "Bar" "Bar" "Bar" "Bar" "Bar" "Foo" "Foo" "Foo"
```

Now a `for` loop example,

```
x = 11:15
for (i in 1:5) {
  x[i] = x[i] * 2
}

x
```

```
## [1] 22 24 26 28 30
```

Note that this `for` loop is very normal in many programming languages, but not in `R`. In `R` we would not use a loop, instead we would simply use a vectorized operation.

```
x = 11:15
x = x * 2
x
```

```
## [1] 22 24 26 28 30
```

2.3.3 Functions

So far we have been using functions, but haven't actually discussed some of their details.

```
function_name(arg1 = 10, arg2 = 20)
```

To use a function, you simply type its name, followed by an open parenthesis, then specify values of its arguments, then finish with a closing parenthesis.

An **argument** is a variable which is used in the body of the function. Specifying the values of the arguments is essentially providing the inputs to the function.

We can also write our own functions in `R`. For example, we often like to "standardize" variables, that is, subtracting the sample mean, and dividing by the sample standard deviation.

$$\frac{x - \bar{x}}{s}$$

In `R` we would write a function to do this. When writing a function, there are three thing you must do.

- Give the function a name. Preferably something that is short, but descriptive.
- Specify the arguments using `function()`
- Write the body of the function within curly braces, `{}`.

```
standardize = function(x) {
  m = mean(x)
  std = sd(x)
  result = (x - m) / std
  result
}
```

Here the name of the function is `standardize`, and the function has a single argument `x` which is used in the body of function. Note that the output of the final line of the body is what is returned by the function. In this case the function returns the vector stored in the variable `results`.

To test our function, we will take a random sample of size `n = 10` from a normal distribution with a mean of `2` and a standard deviation of `5`.

```
(test_sample = rnorm(n = 10, mean = 2, sd = 5))
```

```
##  [1] -4.5798797 -2.8449101  0.5883266 -7.2013093  3.6586112  3.2252645
##  [7]  4.9238427  0.3818837 -2.5859105  1.9709208
```

```
standardize(x = test_sample)
```

```
##  [1] -1.1010564 -0.6602415  0.2120630 -1.7670998  0.9921499  0.8820468
##  [7]  1.3136154  0.1596107 -0.5944358  0.5633476
```

This function could be written much more succinctly, simply performing all the operations on one line and immediately returning the result, without storing any of the intermediate results.

```
standardize = function(x) {
  (x - mean(x)) / sd(x)
}
```

When specifying arguments, you can provide default arguments.

```
power_of_num = function(num, power = 2) {
  num ^ power
}
```

Let's look at a number of ways that we could run this function to perform the operation `10^2` resulting in `100`.

```
power_of_num(10)
```

```
## [1] 100
```

```
power_of_num(10, 2)
```

```
## [1] 100
```

```
power_of_num(num = 10, power = 2)
```

```
## [1] 100
```

```
power_of_num(power = 2, num = 10)
```

```
## [1] 100
```

Note that without using the argument names, the order matters. The following code will not evaluate to the same output as the previous example.

```
power_of_num(2, 10)
```

```
## [1] 1024
```

Also, the following line of code would produce an error since arguments without a default value must be specified.

```
power_of_num(power = 5)
```

To further illustrate a function with a default argument, we will write a function that calculates sample standard deviation two ways.

By default, is will calculate the unbiased estimate of σ, which we will call s.

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

It will also have the ability to return the biased estimate (based on maximum likelihood) which we will call $\hat{\sigma}$.

$$\hat{\sigma} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x - \bar{x})^2}$$

```
get_sd = function(x, biased = FALSE) {
  n = length(x) - 1 * !biased
  sqrt((1 / n) * sum((x - mean(x)) ^ 2))
}
```

```
get_sd(test_sample)
```

```
## [1] 3.935824
```

```
get_sd(test_sample, biased = FALSE)
```

```
## [1] 3.935824
```

```
sd(test_sample)
```

```
## [1] 3.935824
```

We see the function is working as expected, and when returning the unbiased estimate it matches `R`'s built in function `sd()`. Finally, let's examine the biased estimate of σ.

```
get_sd(test_sample, biased = TRUE)
```

```
## [1] 3.73385
```

2.4 Hypothesis Tests in R

2.4.1 One Sample t-Test: Review

Suppose $x_i \sim \mathrm{N}(\mu, \sigma^2)$ and we want to test $H_0 : \mu = \mu_0$ versus $H_1 : \mu \neq \mu_0$.

Assuming σ is unknown, we use the one-sample Student's t test statistic:

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} \sim t_{n-1},$$

where $\bar{x} = \dfrac{\sum_{i=1}^{n} x_i}{n}$ and $s = \sqrt{\dfrac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$.

A $100(1-\alpha)\%$ confidence interval for μ is given by,

$$\bar{x} \pm t_{n-1}(\alpha/2)\frac{s}{\sqrt{n}}$$

where $t_{n-1}(\alpha/2)$ is the critical value such that $P\left(t > t_{n-1}(\alpha/2)\right) = \alpha/2$ for $n-1$ degrees of freedom.

2.4.2 One Sample t-Test: Example

Suppose a grocery store sells "16 ounce" boxes of *Captain Crisp* cereal. A random sample of 9 boxes was taken and weighed. The weight in ounces are stored in the data frame `capt_crisp`.

```
capt_crisp = data.frame(weight = c(15.5, 16.2, 16.1, 15.8, 15.6, 16.0, 15.8, 15.9, 16.2))
```

The company that makes *Captain Crisp* cereal claims that the average weight of a box is at least 16 ounces. We will assume the weight of cereal in a box is normally distributed and use a 0.05 level of significance to test the company's claim.

To test $H_0 : \mu \geq 16$ versus $H_1 : \mu < 16$, the test statistic is

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

The sample mean $\bar{x}$ and the sample standard deviation s can be easily computed using `R`. We also create variables which store the hypothesized mean and the sample size.

```
x_bar = mean(capt_crisp$weight)
s     = sd(capt_crisp$weight)
mu_0  = 16
n     = 9
```

We can then easily compute the test statistic.

```
t = (x_bar - mu_0) / (s / sqrt(n))
t
```

```
## [1] -1.2
```

Under the null hypothesis, the test statistic has a t distribution with $n-1$ degrees of freedom, in this case 8.

To complete the test, we need to obtain the p-value of the test. Since this is a one-sided test with a less-than alternative, we need to area to the left of -1.2 for a t distribution with 8 degrees of freedom. That is,

$$P(t_8 < -1.2)$$

```r
pt(t, df = n - 1)
```

```
## [1] 0.1322336
```

We now have the p-value of our test, which is greater than our significance level (0.05), so we fail to reject the null hypothesis.

Alternatively, this entire process could have been completed using one line of `R` code.

```r
t.test(x = capt_crisp$weight, mu = 16, alternative = c("less"), conf.level = 0.95)
```

```
##
##  One Sample t-test
##
## data:  capt_crisp$weight
## t = -1.2, df = 8, p-value = 0.1322
## alternative hypothesis: true mean is less than 16
## 95 percent confidence interval:
##      -Inf 16.05496
## sample estimates:
## mean of x
##      15.9
```

We supply `R` with the data, the hypothesized value of μ, the alternative, and the confidence level. `R` then returns a wealth of information including:

- The value of the test statistic.
- The degrees of freedom of the distribution under the null hypothesis.
- The p-value of the test.
- The confidence interval which corresponds to the test.
- An estimate of μ.

Since the test was one-sided, `R` returned a one-sided confidence interval. If instead we wanted a two-sided interval for the mean weight of boxes of *Captain Crisp* cereal we could modify our code.

```r
capt_test_results = t.test(capt_crisp$weight, mu = 16,
                           alternative = c("two.sided"), conf.level = 0.95)
```

This time we have stored the results. By doing so, we can directly access portions of the output from `t.test()`. To see what information is available we use the `names()` function.

```r
names(capt_test_results)
```

```
## [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
## [6] "null.value"  "alternative" "method"      "data.name"
```

We are interested in the confidence interval which is stored in `conf.int`.

```
capt_test_results$conf.int
```

```
## [1] 15.70783 16.09217
## attr(,"conf.level")
## [1] 0.95
```

Let's check this interval "by hand." The one piece of information we are missing is the critical value, $t_{n-1}(\alpha/2) = t_8(0.025)$, which can be calculated in R using the `qt()` function.

```
qt(0.975, df = 8)
```

```
## [1] 2.306004
```

So, the 95% CI for the mean weight of a cereal box is calculated by plugging into the formula,

$$\bar{x} \pm t_{n-1}(\alpha/2)\frac{s}{\sqrt{n}}$$

```
c(mean(capt_crisp$weight) - qt(0.975, df = 8) * sd(capt_crisp$weight) / sqrt(9),
  mean(capt_crisp$weight) + qt(0.975, df = 8) * sd(capt_crisp$weight) / sqrt(9))
```

```
## [1] 15.70783 16.09217
```

2.4.3 Two Sample t-Test: Review

Suppose $x_i \sim \mathrm{N}(\mu_x, \sigma^2)$ and $y_i \sim \mathrm{N}(\mu_y, \sigma^2)$.

Want to test $H_0 : \mu_x - \mu_y = \mu_0$ versus $H_1 : \mu_x - \mu_y \neq \mu_0$.

Assuming σ is unknown, use the two-sample Student's t test statistic:

$$t = \frac{(\bar{x} - \bar{y}) - \mu_0}{s_p\sqrt{\frac{1}{n} + \frac{1}{m}}} \sim t_{n+m-2},$$

where $\bar{x} = \dfrac{\sum_{i=1}^{n} x_i}{n}$, $\bar{y} = \dfrac{\sum_{i=1}^{m} y_i}{m}$, and $s_p^2 = \dfrac{(n-1)s_x^2 + (m-1)s_y^2}{n+m-2}$.

A $100(1-\alpha)\%$ CI for $\mu_x - \mu_y$ is given by

$$(\bar{x} - \bar{y}) \pm t_{n+m-2}(\alpha/2)\left(s_p\sqrt{\frac{1}{n} + \frac{1}{m}}\right),$$

where $t_{n+m-2}(\alpha/2)$ is the critical value such that $P\left(t > t_{n+m-2}(\alpha/2)\right) = \alpha/2$.

2.4.4 Two Sample t-Test: Example

Assume that the distributions of X and Y are $\mathrm{N}(\mu_1, \sigma^2)$ and $\mathrm{N}(\mu_2, \sigma^2)$, respectively. Given the $n = 6$ observations of X,

```
x = c(70, 82, 78, 74, 94, 82)
n = length(x)
```

and the $m = 8$ observations of Y,

```
y = c(64, 72, 60, 76, 72, 80, 84, 68)
m = length(y)
```

we will test $H_0 : \mu_1 = \mu_2$ versus $H_1 : \mu_1 > \mu_2$.

First, note that we can calculate the sample means and standard deviations.

```
x_bar = mean(x)
s_x   = sd(x)
y_bar = mean(y)
s_y   = sd(y)
```

We can then calculate the pooled standard deviation.

$$s_p = \sqrt{\frac{(n-1)s_x^2 + (m-1)s_y^2}{n+m-2}}$$

```
s_p = sqrt(((n - 1) * s_x ^ 2 + (m - 1) * s_y ^ 2) / (n + m - 2))
```

Thus, the relevant t test statistic is given by

$$t = \frac{(\bar{x} - \bar{y}) - \mu_0}{s_p\sqrt{\frac{1}{n} + \frac{1}{m}}}.$$

```
t = ((x_bar - y_bar) - 0) / (s_p * sqrt(1 / n + 1 / m))
t
```

```
## [1] 1.823369
```

Note that $t \sim t_{n+m-2} = t_{12}$, so we can calculate the p-value, which is

$$P(t_{12} > 1.8233692).$$

```
1 - pt(t, df = n + m - 2)
```

```
## [1] 0.04661961
```

But, then again, we could have simply performed this test in one line of `R`.

```
t.test(x, y, alternative = c("greater"), var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  x and y
## t = 1.8234, df = 12, p-value = 0.04662
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.1802451       Inf
## sample estimates:
## mean of x mean of y
##        80        72
```

Recall that a two-sample t-test can be done with or without an equal variance assumption. Here `var.equal = TRUE` tells R we would like to perform the test under the equal variance assumption.

Above we carried out the analysis using two vectors `x` and `y`. In general, we will have a preference for using data frames.

```
t_test_data = data.frame(values = c(x, y),
                         group  = c(rep("A", length(x)), rep("B", length(y))))
```

We now have the data stored in a single variables (`values`) and have created a second variable (`group`) which indicates which "sample" the value belongs to.

```
t_test_data
```

```
##    values group
## 1      70     A
## 2      82     A
## 3      78     A
## 4      74     A
## 5      94     A
## 6      82     A
## 7      64     B
## 8      72     B
## 9      60     B
## 10     76     B
## 11     72     B
## 12     80     B
## 13     84     B
## 14     68     B
```

Now to perform the test, we still use the `t.test()` function but with the `~` syntax and a `data` argument.

```
t.test(values ~ group, data = t_test_data,
       alternative = c("greater"), var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  values by group
## t = 1.8234, df = 12, p-value = 0.04662
```

```
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.1802451       Inf
## sample estimates:
## mean in group A mean in group B
##              80              72
```

2.5 Simulation

One of the biggest strengths of R is its ability to carry out simulations. We'll look at two examples here, however simulation will be a topic we revisit several times throughout the course.

2.5.1 Paired Differences

Consider the model:

$$X_{11}, X_{12}, \ldots, X_{1n} \sim N(\mu_1, \sigma^2)$$
$$X_{21}, X_{22}, \ldots, X_{2n} \sim N(\mu_2, \sigma^2)$$

Assume that $\mu_1 = 6$, $\mu_2 = 5$, $\sigma^2 = 4$ and $n = 25$.

Let

$$\bar{X}_1 = \frac{1}{n}\sum_{i=1}^{n} X_{1i}$$
$$\bar{X}_2 = \frac{1}{n}\sum_{i=1}^{n} X_{2i}$$
$$D = \bar{X}_1 - \bar{X}_2.$$

Suppose we would like to calculate $P(0 < D < 2)$. First we will need to obtain the distribution of D.

Recall,

$$\bar{X}_1 \sim N\left(\mu_1, \frac{\sigma^2}{n}\right)$$

and

$$\bar{X}_2 \sim N\left(\mu_2, \frac{\sigma^2}{n}\right).$$

Then,

$$D = \bar{X}_1 - \bar{X}_2 \sim N\left(\mu_1 - \mu_2, \frac{\sigma^2}{n} + \frac{\sigma^2}{n}\right) = N\left(6 - 5, \frac{4}{25} + \frac{4}{25}\right).$$

So,

$$D \sim N(\mu = 1, \sigma^2 = 0.32).$$

Thus,

$$P(0 < D < 2) = P(D < 2) - P(D < 0).$$

This can then be calculated using `R` without a need to first standardize, or use a table.

```
pnorm(2, mean = 1, sd = sqrt(0.32)) - pnorm(0, mean = 1, sd = sqrt(0.32))
```

```
## [1] 0.9229001
```

An alternative approach, would be to **simulate** a large number of observations of D then use the **empirical distribution** to calculate the probability.

Our strategy will be to repeatedly:

- Generate a sample of 25 random observations from $N(\mu_1 = 6, \sigma^2 = 4)$. Call the mean of this sample $\bar{x}_{1s}$.
- Generate a sample of 25 random observations from $N(\mu_1 = 5, \sigma^2 = 4)$. Call the mean of this sample $\bar{x}_{2s}$.
- Calculate the differences of the means, $d_s = \bar{x}_{1s} - \bar{x}_{2s}$.

We will repeat the process a large number of times. Then we will use the distribution of the simulated observations of d_s as an estimate for the true distribution of D.

```
set.seed(42)
num_samples = 10000
differences = rep(0, num_samples)
```

Before starting our `for` loop to perform the operation, we set a seed for reproducibility, create and set a variable `num_samples` which will define the number of repetitions, and lastly create a variables `differences` which will store the simulate values, d_s.

By using `set.seed()` we can reproduce the random results of `rnorm()` each time starting from that line.

```
for (s in 1:num_samples) {
  x1 = rnorm(n = 25, mean = 6, sd = 2)
  x2 = rnorm(n = 25, mean = 5, sd = 2)
  differences[s] = mean(x1) - mean(x2)
}
```

To estimate $P(0 < D < 2)$ we will find the proportion of values of d_s (among the 10000 values of d_s generated) that are between 0 and 2.

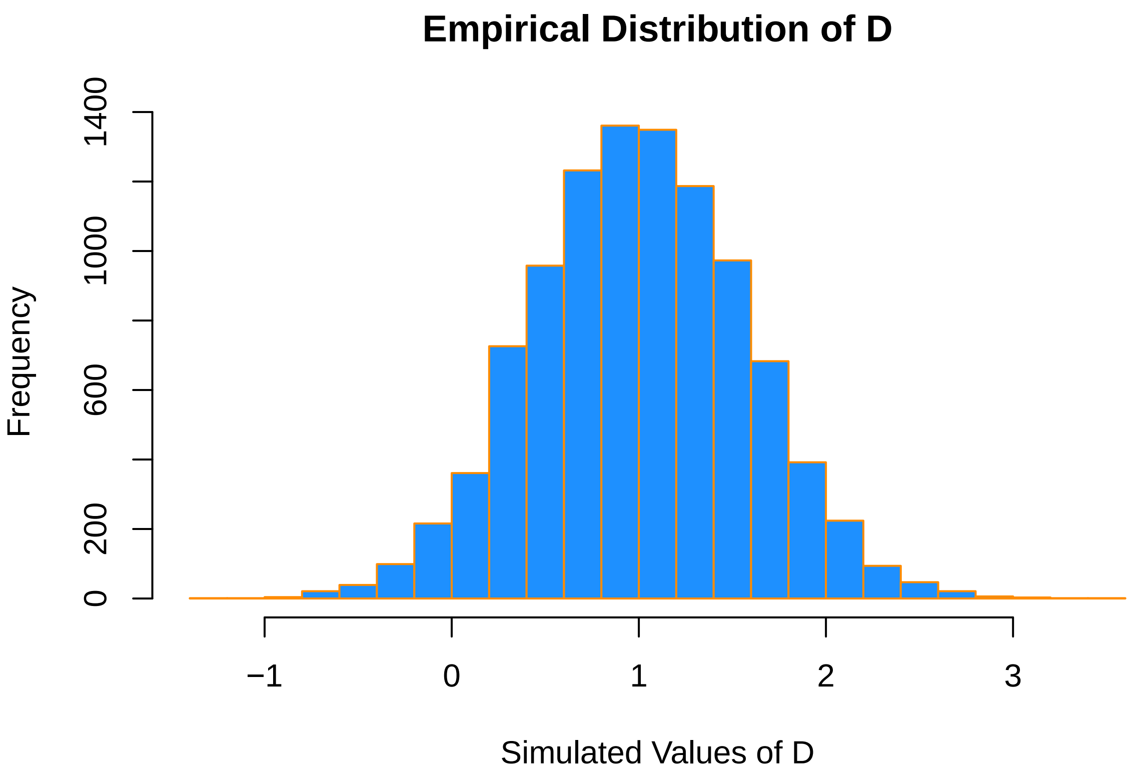
```
mean(0 < differences & differences < 2)
```

```
## [1] 0.9222
```

Recall that above we derived the distribution of D to be $N(\mu = 1, \sigma^2 = 0.32)$

If we look at a histogram of the differences, we find that it looks very much like a normal distribution.

```
hist(differences, breaks = 20,
     main   = "Empirical Distribution of D",
     xlab   = "Simulated Values of D",
     col    = "dodgerblue",
     border = "darkorange")
```



Also the sample mean and variance are very close to to what we would expect.

```
mean(differences)
```

```
## [1] 1.001423
```

```
var(differences)
```

```
## [1] 0.3230183
```

We could have also accomplished this task with a single line of more “idiomatic” `R`.

```
set.seed(42)
diffs = replicate(10000, mean(rnorm(25, 6, 2)) - mean(rnorm(25, 5, 2)))
```

Use `?replicate` to take a look at the documentation for the `replicate` function and see if you can understand how this line performs the same operations that our `for` loop above executed.

```
mean(differences == diffs)
```

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

We see that by setting the same seed for the randomization, we actually obtain identical results!

2.5.2 Distribution of a Sample Mean

For another example of simulation, we will simulate observations from a Poisson distribution, and examine the empirical distribution of the sample mean of these observations.

Recall, if

$$X \sim Pois(\mu)$$

then

$$E[X] = \mu$$

and

$$Var[X] = \mu.$$

Also, recall that for a random variable X with finite mean μ and finite variance σ^2, the central limit theorem tells us that the mean, $\bar{X}$ of a random sample of size n is approximately normal for *large* values of n. Specifically, as $n \to \infty$,

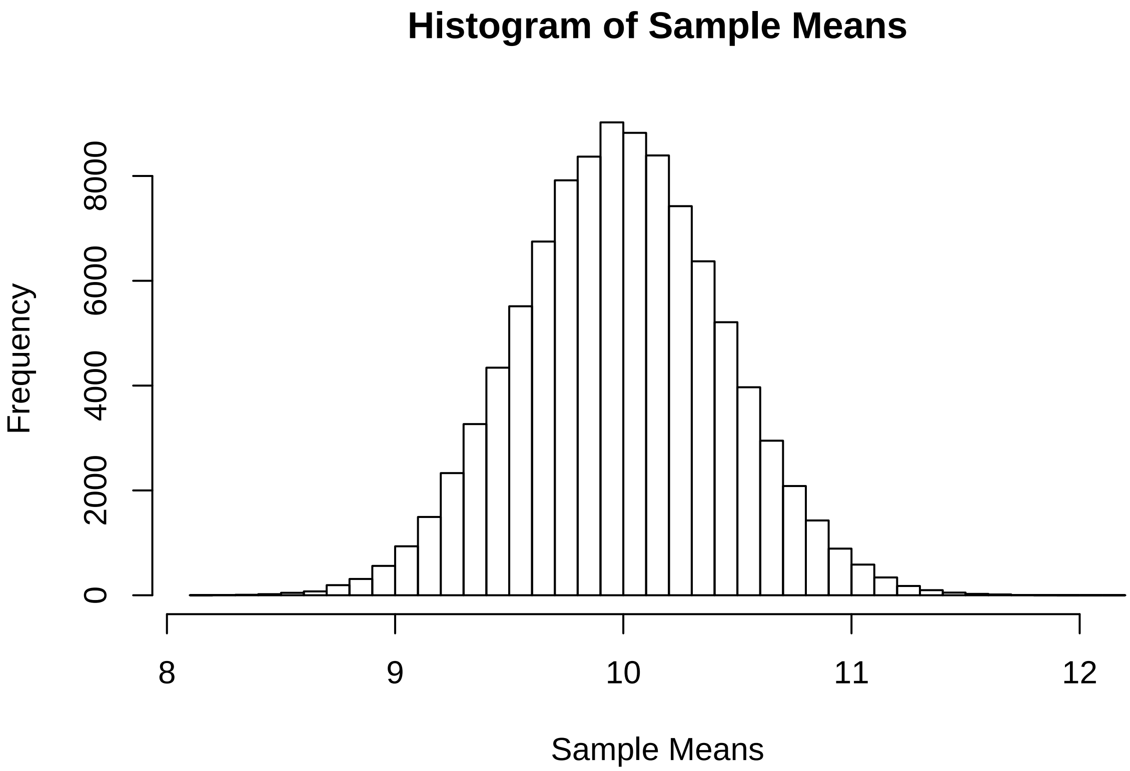
$$\bar{X} \overset{d}{\to} N\left(\mu, \frac{\sigma^2}{n}\right).$$

The following verifies this result for a Poisson distribution with $\mu = 10$ and a sample size of $n = 50$.

```
set.seed(1337)
mu          = 10
sample_size = 50
samples     = 100000
x_bars      = rep(0, samples)
```

```
for(i in 1:samples){
  x_bars[i] = mean(rpois(sample_size, lambda = mu))
}
```

```
x_bar_hist = hist(x_bars, breaks = 50,
                  main = "Histogram of Sample Means",
                  xlab = "Sample Means")
```



Now we will compare sample statistics from the empirical distribution with their known values based on the parent distribution.

```
c(mean(x_bars), mu)
```

```
## [1] 10.00008 10.00000
```

```
c(var(x_bars), mu / sample_size)
```

```
## [1] 0.1989732 0.2000000
```

```
c(sd(x_bars), sqrt(mu) / sqrt(sample_size))
```

```
## [1] 0.4460641 0.4472136
```

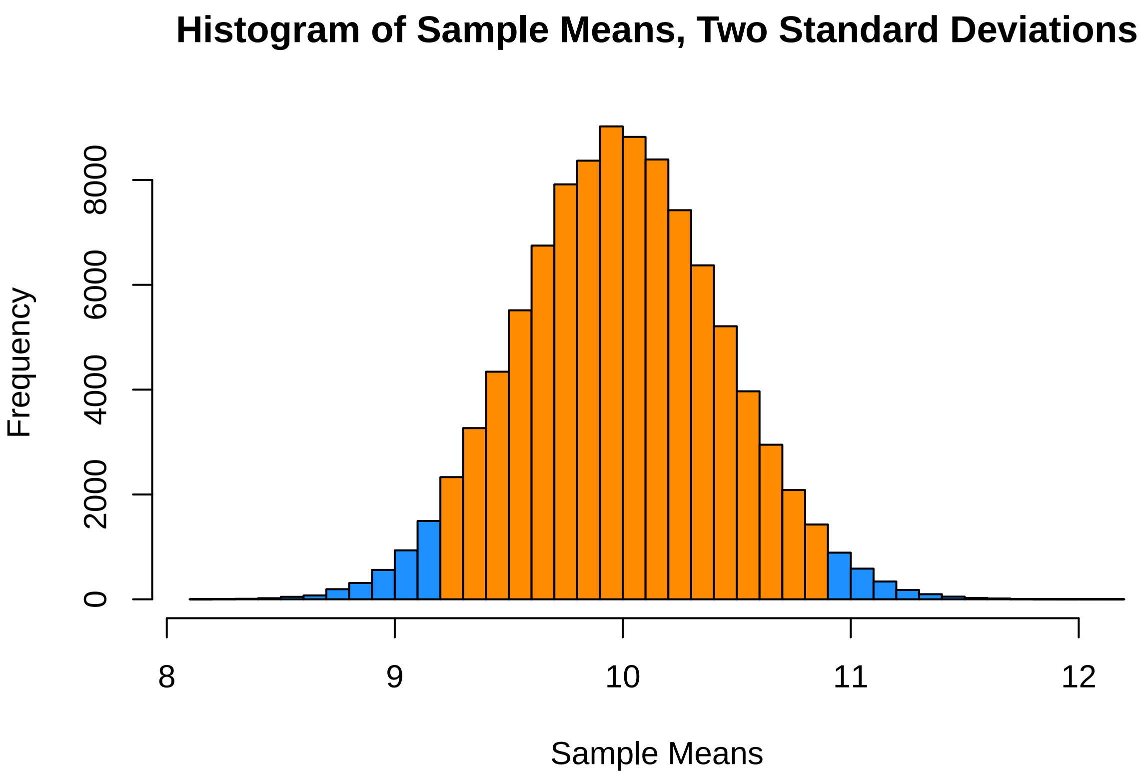
And here, we will calculate the proportion of sample means that are within 2 standard deviations of the population mean.

```
mean(x_bars > mu - 2 * sqrt(mu) / sqrt(sample_size) &
     x_bars < mu + 2 * sqrt(mu) / sqrt(sample_size))
```

```
## [1] 0.95429
```

This last histogram uses a bit of a trick to approximately shade the bars that are within two standard deviations of the mean.)

```
shading = ifelse(x_bar_hist$breaks > mu - 2 * sqrt(mu) / sqrt(sample_size) &
                   x_bar_hist$breaks < mu + 2 * sqrt(mu) / sqrt(sample_size),
                  "darkorange", "dodgerblue")

x_bar_hist = hist(x_bars, breaks = 50, col = shading,
                  main = "Histogram of Sample Means, Two Standard Deviations",
                  xlab = "Sample Means")
```



Chapter 3

Simple Linear Regression

> "All models are wrong, but some are useful."
>
> — **George E. P. Box**

After reading this chapter you will be able to:

- Understand the concept of a model.
- Describe two ways in which regression coefficients are derived.
- Estimate and visualize a regression model using `R`.
- Interpret regression coefficients and statistics in the context of real-world problems.
- Use a regression model to make predictions.

3.1 Modeling

Let's consider a simple example of how the speed of a car affects its stopping distance, that is, how far it travels before it comes to a stop. To examine this relationship, we will use the `cars` dataset which, is a default `R` dataset. Thus, we don't need to load a package first; it is immediately available.

To get a first look at the data you can use the `View()` function inside RStudio.

```
View(cars)
```

We could also take a look at the variable names, the dimension of the data frame, and some sample observations with `str()`.

```
str(cars)
```

```
## 'data.frame':    50 obs. of  2 variables:
##  $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
##  $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

As we have seen before with data frames, there are a number of additional functions to access some of this information directly.

```
dim(cars)
```

```
## [1] 50  2
```

```
nrow(cars)
```

```
## [1] 50
```

```
ncol(cars)
```

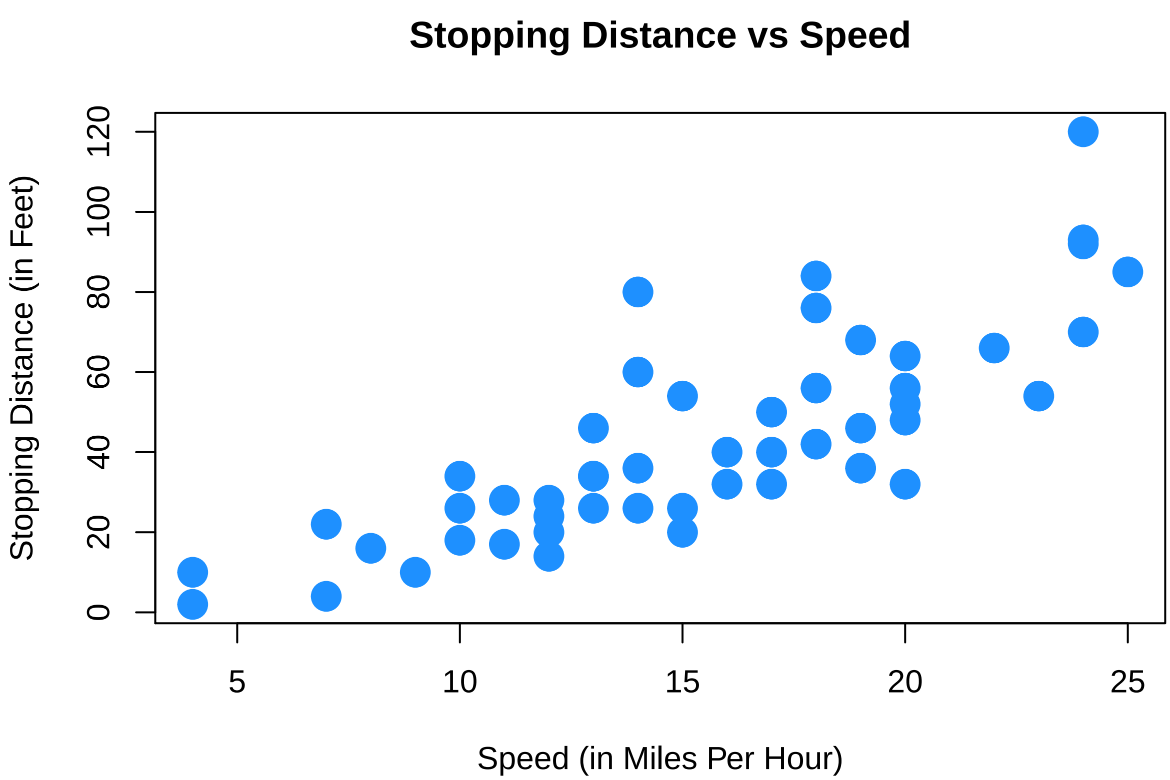
```
## [1] 2
```

Other than the two variable names and the number of observations, this data is still just a bunch of numbers, so we should probably obtain some context.

```
?cars
```

Reading the documentation we learn that this is data gathered during the 1920s about the speed of cars and the resulting distance it takes for the car to come to a stop. The interesting task here is to determine how far a car travels before stopping, when traveling at a certain speed. So, we will first plot the stopping distance against the speed.

```
plot(dist ~ speed, data = cars,
     xlab = "Speed (in Miles Per Hour)",
     ylab = "Stopping Distance (in Feet)",
     main = "Stopping Distance vs Speed",
     pch  = 20,
     cex  = 3,
     col  = "dodgerblue")
```



Let's now define some terminology. We have pairs of data, (x_i, y_i), for $i = 1, 2, \ldots n$, where n is the sample size of the dataset.

We use i as an index, simply for notation. We use x_i as the **predictor** (explanatory) variable. The predictor variable is used to help *predict* or explain the **response** (target, outcome) variable, y_i.

Other texts may use the term independent variable instead of predictor and dependent variable in place of response. However, those monikers imply mathematical characteristics that might not be true. While these other terms are not incorrect, independence is already a strictly defined concept in probability. For example, when trying to predict a person's weight given their height, would it be accurate to say that height is independent of weight? Certainly not, but that is an unintended implication of saying "independent variable." We prefer to stay away from this nomenclature.

In the `cars` example, we are interested in using the predictor variable `speed` to predict and explain the response variable `dist`.

Broadly speaking, we would like to model the relationship between X and Y using the form

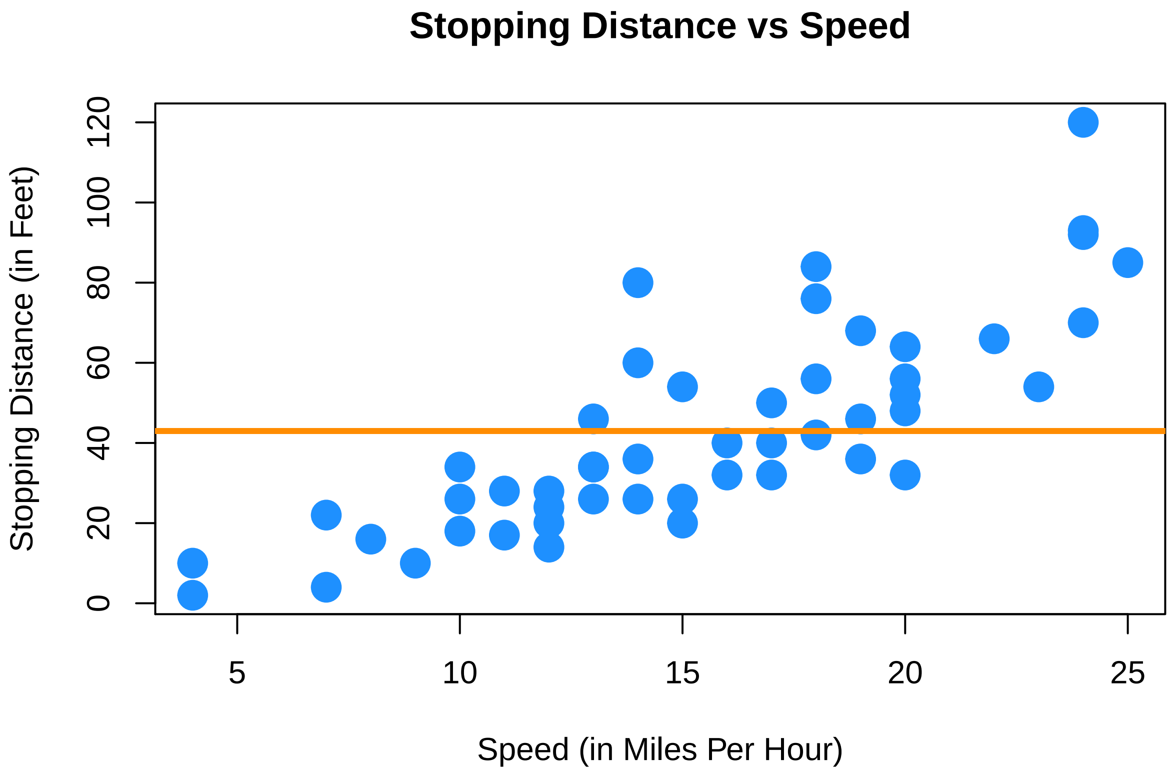
$$Y = f(X) + \epsilon.$$

The function f describes the functional relationship between the two variables, and the ϵ term is used to account for error. This indicates that if we plug in a given value of X as input, our output is a value of Y, within a certain range of error. You could think of this a number of ways:

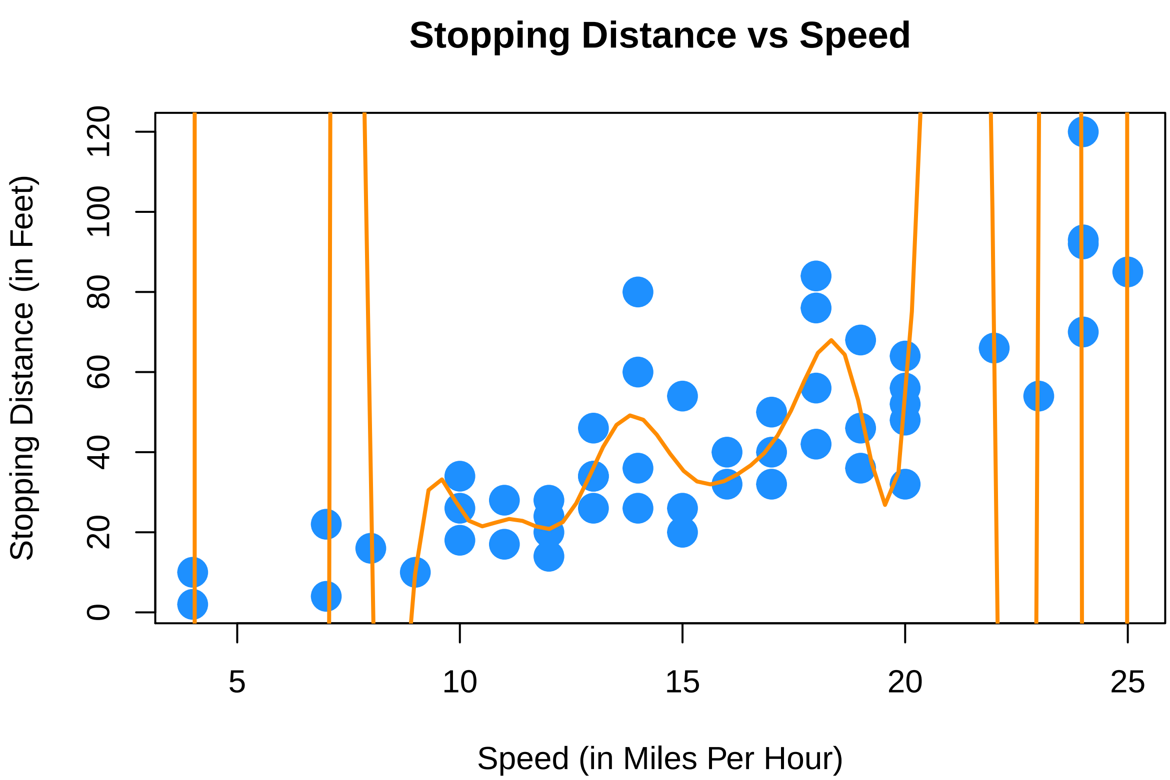
- Response = Prediction + Error
- Response = Signal + Noise
- Response = Model + Unexplained
- Response = Explainable + Unexplainable

What sort of function should we use for $f(X)$ for the `cars` data?

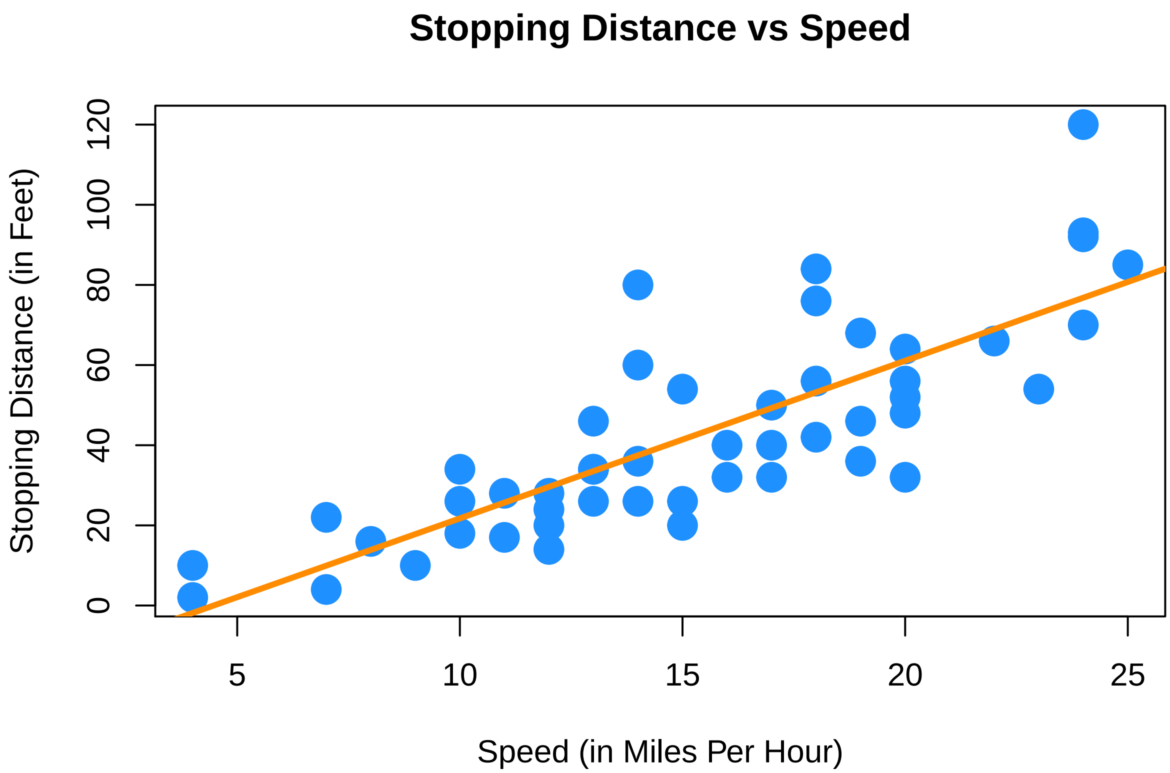
We could try to model the data with a horizontal line. That is, the model for y does not depend on the value of x. (Some function $f(X) = c$.) In the plot below, we see this doesn't seem to do a very good job. Many of the data points are very far from the orange line representing c. This is an example of **underfitting**. The obvious fix is to make the function $f(X)$ actually depend on x.



We could also try to model the data with a very "wiggly" function that tries to go through as many of the data points as possible. This also doesn't seem to work very well. The stopping distance for a speed of 5 mph shouldn't be off the chart! (Even in 1920.) This is an example of **overfitting**. (Note that in this example no function will go through every point, since there are some x values that have several possible y values in the data.)



Lastly, we could try to model the data with a well chosen line rather than one of the two extremes previously attempted. The line on the plot below seems to summarize the relationship between stopping distance and speed quite well. As speed increases, the distance required to come to a stop increases. There is still some variation about this line, but it seems to capture the overall trend.



With this in mind, we would like to restrict our choice of $f(X)$ to *linear* functions of X. We will write our model using β_1 for the slope, and β_0 for the intercept,

$$Y = \beta_0 + \beta_1 X + \epsilon.$$

3.1.1 Simple Linear Regression Model

We now define what we will call the simple linear regression model,

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$. That is, the ϵ_i are *independent and identically distributed* (iid) normal random variables with mean 0 and variance σ^2. This model has three parameters to be estimated: β_0, β_1, and σ^2, which are fixed, but unknown constants.

We have slightly modified our notation here. We are now using Y_i and x_i, since we will be fitting this model to a set of n data points, for $i = 1, 2, \ldots n$.

Recall that we use capital Y to indicate a random variable, and lower case y to denote a potential value of the random variable. Since we will have n observations, we have n random variables Y_i and their possible values y_i.

In the simple linear regression model, the x_i are assumed to be fixed, known constants, and are thus notated with a lower case variable. The response Y_i remains a random variable because of the random behavior of the error variable, ϵ_i. That is, each response Y_i is tied to an observable x_i and a random, unobservable, ϵ_i.

The random Y_i are a function of x_i, thus we can write its mean as a function of x_i,

$$E[Y_i] = \beta_0 + \beta_1 x_i.$$

However, its variance remains constant for each x_i,

$$Var[Y_i] = \sigma^2.$$

This is visually displayed in the image below. We see that for any value x, the expected value of Y is $\beta_0 + \beta_1 x$. At each value of x, Y has the same variance σ^2.

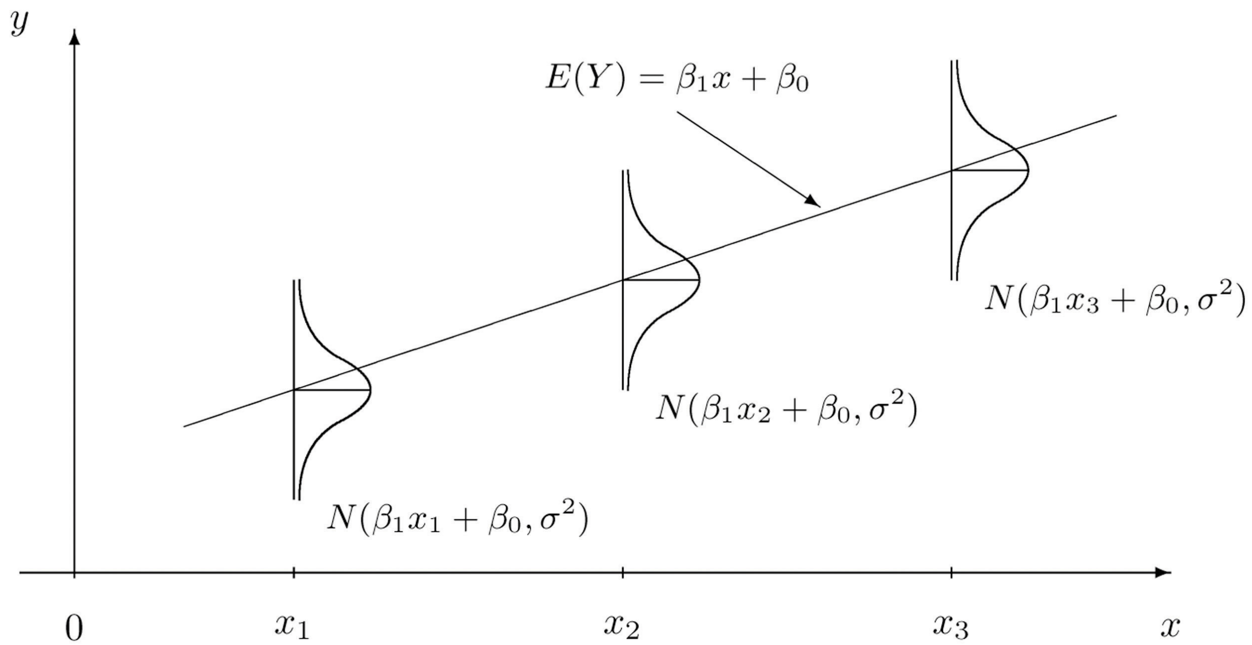


Figure 3.1: Simple Linear Regression Model UC David Stat Wiki

Often, we directly talk about the assumptions that this model makes. They can be cleverly shortened to **LINE**.

- **L**inear. The relationship between Y and x is linear, of the form $\beta_0 + \beta_1 x$.
- **I**ndepedent. The errors ϵ are independent.
- **N**ormal. The errors, ϵ are normally distributed. That is the "error" around the line follows a normal distribution.
- **E**qual Variance. At each value of x, the variance of Y is the same, σ^2.

As a side note, we will often refer to simple linear regression as **SLR**. Some explanation of the name SLR:

- **Simple** refers to the fact that we are using a single predictor variable. Later we will use multiple predictor variables.
- **Linear** tells us that our model for Y is a linear combination of the predictors X. (In this case just the one.) Right now, this always results in a model that is a line, but later we will see how this is not always the case.
- **Regression** simply means that we are attempting to measure the relationship between a response variable and (one or more) predictor variables.

So SLR models Y as a linear function of X, but how do we actually define a good line? There are an infinite number of lines we could use, so we will attempt to find one with "small errors." That is a line with as many points as close to it as possible. The questions now becomes, how do we find such a line? There are many approaches we could take.

We could find the line that has the smallest maximum distance from any of the points to the line. That is,

$$\underset{\beta_0,\beta_1}{\text{argmin}} \max |y_i - (\beta_0 + \beta_1 x_i)|.$$

We could find the line that minimizes the sum of all the distances from the points to the line. That is,

$$\underset{\beta_0,\beta_1}{\text{argmin}} \sum_{i=1}^{n} |y_i - (\beta_0 + \beta_1 x_i)|.$$

We could find the line that minimizes the sum of all the squared distances from the points to the line. That is,

$$\underset{\beta_0,\beta_1}{\text{argmin}} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i))^2.$$

This last option is called the method of **least squares**. It is essentially the de-facto method for fitting a line to data. (You may have even seen it before in a linear algebra course.) Its popularity is largely due to the fact that it is mathematically "easy." (Which was important historically, as computers are a modern contraption.) It is also very popular because many relationships are well approximated by a linear function.

3.2 Least Squares Approach

Given observations (x_i, y_i), for $i = 1, 2, \ldots n$, we want to find values of β_0 and β_1 which minimize

$$f(\beta_0, \beta_1) = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i))^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2.$$

We will call these values $\hat{\beta}_0$ and $\hat{\beta}_1$.

First, we take a partial derivative with respect to both β_0 and β_1.

$$\frac{\partial f}{\partial \beta_0} = -2 \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)$$
$$\frac{\partial f}{\partial \beta_1} = -2 \sum_{i=1}^{n} (x_i)(y_i - \beta_0 - \beta_1 x_i)$$

We then set each of the partial derivatives equal to zero and solving the resulting system of equations.

$$\sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i) = 0$$
$$\sum_{i=1}^{n} (x_i)(y_i - \beta_0 - \beta_1 x_i) = 0$$

While solving the system of equations, one common algebraic rearrangement results in the **normal equations**.

$$
\begin{aligned}
\sum_{i=1}^{n} y_i &= n\beta_0 + \beta_1 \sum_{i=1}^{n} x_i \\
\sum_{i=1}^{n} x_i y_i &= \beta_0 \sum_{i=1}^{n} x_i + \beta_1 \sum_{i=1}^{n} x_i^2
\end{aligned}
$$

Finally, we finish solving the system of equations.

$$
\begin{aligned}
\hat{\beta}_1 &= \frac{\sum_{i=1}^{n} x_i y_i - \frac{(\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n}}{\sum_{i=1}^{n} x_i^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n}} = \frac{S_{xy}}{S_{xx}} \\
\hat{\beta}_0 &= \bar{y} - \hat{\beta}_1 \bar{x}
\end{aligned}
$$

Here, we have defined some notation for the expression we've obtained. Note that they have alternative forms which are much easier to work with. (We won't do it here, but you can try to prove the equalities below on your own, for "fun.") We use the capital letter S to denote "summation" which replaces the capital letter Σ when we calculate these values based on observed data, (x_i, y_i). The subscripts such as xy denote over which variables the function $(z - \bar{z})$ is applied.

$$
\begin{aligned}
S_{xy} &= \sum_{i=1}^{n} x_i y_i - \frac{(\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \\
S_{xx} &= \sum_{i=1}^{n} x_i^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n} = \sum_{i=1}^{n}(x_i - \bar{x})^2 \\
S_{yy} &= \sum_{i=1}^{n} y_i^2 - \frac{(\sum_{i=1}^{n} y_i)^2}{n} = \sum_{i=1}^{n}(y_i - \bar{y})^2
\end{aligned}
$$

Note that these summations S are not to be confused with sample standard deviation s.

By using the above alternative expressions for S_{xy} and S_{xx}, we arrive at a cleaner, more useful expression for $\hat{\beta}_1$.

$$
\hat{\beta}_1 = \frac{S_{xy}}{S_{xx}} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}
$$

Traditionally we would now calculate $\hat{\beta}_0$ and $\hat{\beta}_1$ by hand for the `cars` dataset. However because we are living in the 21st century and are intelligent (or lazy or efficient, depending on your perspective), we will utilize `R` to do the number crunching for us.

To keep some notation consistent with above mathematics, we will store the response variable as `y` and the predictor variable as `x`.

```
x = cars$speed
y = cars$dist
```

We then calculate the three sums of squares defined above.

```
Sxy = sum((x - mean(x)) * (y - mean(y)))
Sxx = sum((x - mean(x)) ^ 2)
Syy = sum((y - mean(y)) ^ 2)
c(Sxy, Sxx, Syy)
```

```
## [1]  5387.40  1370.00 32538.98
```

Then finally calculate $\hat{\beta}_0$ and $\hat{\beta}_1$.

```
beta_1_hat = Sxy / Sxx
beta_0_hat = mean(y) - beta_1_hat * mean(x)
c(beta_0_hat, beta_1_hat)
```

```
## [1] -17.579095   3.932409
```

What do these values tell us about our dataset?

The slope *parameter* β_1 tells us that for an increase in speed of one mile per hour, the **mean** stopping distance increases by β_1. It is important to specify that we are talking about the mean. Recall that $\beta_0 + \beta_1 x$ is the estimated mean of Y, in this case stopping distance, for a particular value of x. (In this case speed.) So β_1 tells us how the mean of Y is affected by a change in x.

Similarly, the *estimate* $\hat{\beta}_1 = 3.932$ tells us that for an increase in speed of one mile per hour, the **estimated** *mean* stopping distance increases by 3.932 feet. Here we should be sure to specify we are discussing an estimated quantity. Recall that $\hat{y}$ is the estimated mean of Y, so $\hat{\beta}_1$ tells us how the estimated mean of Y is affected by changing x.

The intercept *parameter* β_0 tells us the **mean** stopping distance for a car traveling zero miles per hour. (Not moving.) The *estimate* $\hat{\beta}_0 = -17.579$ tells us that the **estimated** mean stopping distance for a car traveling zero miles per hour is -17.579 feet. So when you apply the brakes to a car that is not moving, it moves backwards? This doesn't seem right. (Extrapolation, which we will see later, is the issue here.)

3.2.1 Making Predictions

We can now write the **fitted** or estimated line,

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x.$$

In this case,

$$\hat{y} = -17.579 + 3.932x.$$

We can now use this line to make predictions. First, let's see the possible x values in the `cars` dataset. Since some x values may appear more than once, we use the `unique()` to return each unique value only once.

```
unique(cars$speed)
```

```
##  [1]  4  7  8  9 10 11 12 13 14 15 16 17 18 19 20 22 23 24 25
```

Let's make a prediction for the stopping distance of a car traveling at 8 miles per hour.

$$\hat{y} = -17.579 + 3.932 \times 8 = 13.88$$

```
beta_0_hat + beta_1_hat * 8
```

```
## [1] 13.88018
```

This tells us that the estimated mean stopping distance of a car traveling at 8 miles per hour is 13.88.

Now let's make a prediction for the stopping distance of a car traveling at 21 miles per hour. This is considered **interpolation** as 21 is not an observed value of x. (But is in the data range.) We can use the special `%in%` operator to quickly verify this in `R`.

```
8 %in% unique(cars$speed)
```

```
## [1] TRUE
```

```
21 %in% unique(cars$speed)
```

```
## [1] FALSE
```

```
min(cars$speed) < 21 & 21 < max(cars$speed)
```

```
## [1] TRUE
```

$$\hat{y} = -17.579 + 3.932 \times 21 = 65.001$$

```
beta_0_hat + beta_1_hat * 21
```

```
## [1] 65.00149
```

Lastly, we can make a prediction for the stopping distance of a car traveling at 50 miles per hour. This is considered **extrapolation** as 50 is not an observed value of x and is outside data range. We should be less confident in predictions of this type.

```
range(cars$speed)
```

```
## [1]  4 25
```

```
range(cars$speed)[1] < 50 & 50 < range(cars$speed)[2]
```

```
## [1] FALSE
```

$$\hat{y} = -17.579 + 3.932 \times 50 = 179.041$$

```
beta_0_hat + beta_1_hat * 50
```

```
## [1] 179.0413
```

Cars travel 50 miles per hour rather easily today, but not in the 1920s!

This is also an issue we saw when interpreting $\hat{\beta}_0 = -17.579$, which is equivalent to making a prediction at $x = 0$. We should not be confident in the estimated linear relationship outside of the range of data we have observed.

3.2.2 Residuals

If we think of our model as "Response = Prediction + Error," we can then write it as

$$y = \hat{y} + e.$$

We then define a **residual** to be the observed value minus the predicted value.

$$e_i = y_i - \hat{y}_i$$

Let's calculate the residual for the prediction we made for a car traveling 8 miles per hour. First, we need to obtain the observed value of y for this x value.

```
which(cars$speed == 8)
```

```
## [1] 5
```

```
cars[5, ]
```

```
##   speed dist
## 5     8   16
```

```
cars[which(cars$speed == 8), ]
```

```
##   speed dist
## 5     8   16
```

We can then calculate the residual.

$$e = 16 - 13.88 = 2.12$$

```
16 - (beta_0_hat + beta_1_hat * 8)
```

```
## [1] 2.119825
```

The positive residual value indicates that the observed stopping distance is actually 2.12 feet more than what was predicted.

3.2.3 Variance Estimation

We'll now use the residuals for each of the points to create an estimate for the variance, σ^2.

Recall that,

$$E[Y_i] = \beta_0 + \beta_1 x_i.$$

So,

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

is a natural estimate for the mean of Y_i for a given value of x_i.

Also, recall that when we specified the model, we had three unknown parameters; β_0, β_1, and σ^2. The method of least squares gave us estimates for β_0 and β_1, however, we have yet to see an estimate for σ^2. We will now define s_e^2 which will be an estimate for σ^2.

$$\begin{aligned} s_e^2 &= \frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \\ &= \frac{1}{n-2}\sum_{i=1}^{n} e_i^2 \end{aligned}$$

This probably seems like a natural estimate, aside from the use of $n-2$, which we will put off explaining until the next chapter. It should actually look rather similar to something we have seen before.

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

Here, s^2 is the estimate of σ^2 when we have a single random variable X. In this case $\bar{x}$ is an estimate of μ which is assumed to be the same for each x.

Now, in the regression case, with s_e^2 each y has a different mean because of the relationship with x. Thus, for each y_i, we use a different estimate of the mean, that is $\hat{y}_i$.

```
y_hat = beta_0_hat + beta_1_hat * x
e     = y - y_hat
n     = length(e)
s2_e  = sum(e^2) / (n - 2)
s2_e
```

```
## [1] 236.5317
```

Just as with the univariate measure of variance, this value of 236.532 doesn't have a practical interpretation in terms of stopping distance. Taking the square root, however, computes the standard deviation of the residuals, also known as *residual standard error*.

```
s_e = sqrt(s2_e)
s_e
```

```
## [1] 15.37959
```

This tells us that our estimates of mean stopping distance are "typically" off by 15.38 feet.

3.3 Decomposition of Variation

We can re-express $y_i - \bar{y}$, which measures the deviation of an observation from the sample mean, in the following way,

$$y_i - \bar{y} = (y_i - \hat{y}_i) + (\hat{y}_i - \bar{y}).$$

This is the common mathematical trick of "adding zero." In this case we both added and subtracted $\hat{y}_i$.

Here, $y_i - \hat{y}_i$ measures the deviation of an observation from the fitted regression line and $\hat{y}_i - \bar{y}$ measures the deviation of the fitted regression line from the sample mean.

If we square then sum both sides of the equation above, we can obtain the following,

$$\sum_{i=1}^{n}(y_i - \bar{y})^2 = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2.$$

This should be somewhat alarming or amazing. How is this true? For now we will leave this questions unanswered. (Think about this, and maybe try to prove it.) We will now define three of the quantities seen in this equation.

Sum of Squares Total

$$SST = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

The quantity "Sum of Squares Total," or SST, represents the **total variation** of the observed y values. This should be a familiar looking expression. Note that,

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2 = \frac{1}{n-1}SST.$$

Sum of Squares Regression

$$SSReg = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$$

The quantity "Sum of Squares Regression," $SSReg$, represents the **explained variation** of the observed y values.

Sum of Squares Error

$$SSE = RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The quantity "Sum of Squares Error," SSE, represents the **unexplained variation** of the observed y values. You will often see SSE written as RSS, or "Residual Sum of Squares."

```
SST   = sum((y - mean(y)) ^ 2)
SSReg = sum((y_hat - mean(y)) ^ 2)
SSE   = sum((y - y_hat) ^ 2)
c(SST = SST, SSReg = SSReg, SSE = SSE)
```

```
##      SST    SSReg      SSE
## 32538.98 21185.46 11353.52
```

Note that,

$$s_e^2 = \frac{SSE}{n - 2}.$$

```
SSE / (n - 2)
```

```
## [1] 236.5317
```

We can use `R` to verify that this matches our previous calculation of s_e^2.

```
s2_e == SSE / (n - 2)
```

```
## [1] TRUE
```

These three measures also do not have an important practical interpretation individually. But together, they're about to reveal a new statistic to help measure the strength of a SLR model.

3.3.1 Coefficient of Determination

The **coefficient of determination**, R^2, is defined as

$$R^2 = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = \frac{SSReg}{SST} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = 1 - \frac{\sum_{i=1}^{n} e_i^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = 1 - \frac{SSE}{SST}$$

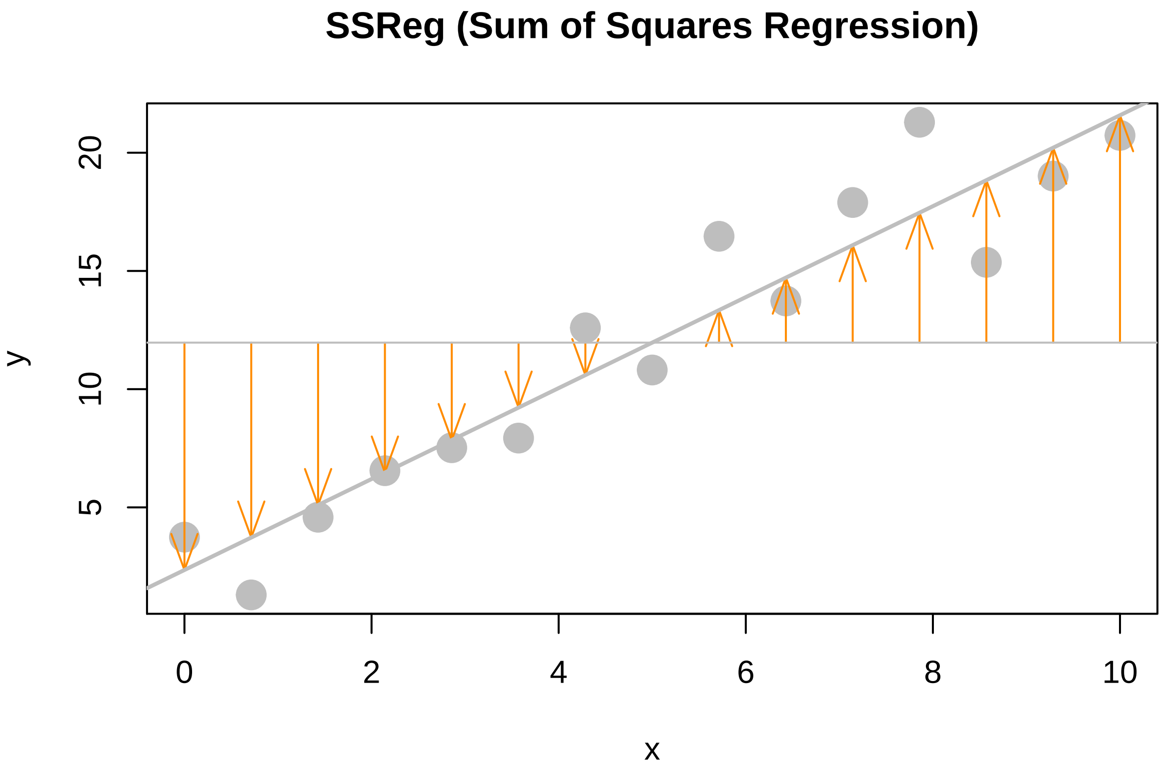
The coefficient of determination is interpreted as the proportion of observed variation in y that can be explained by the simple linear regression model.

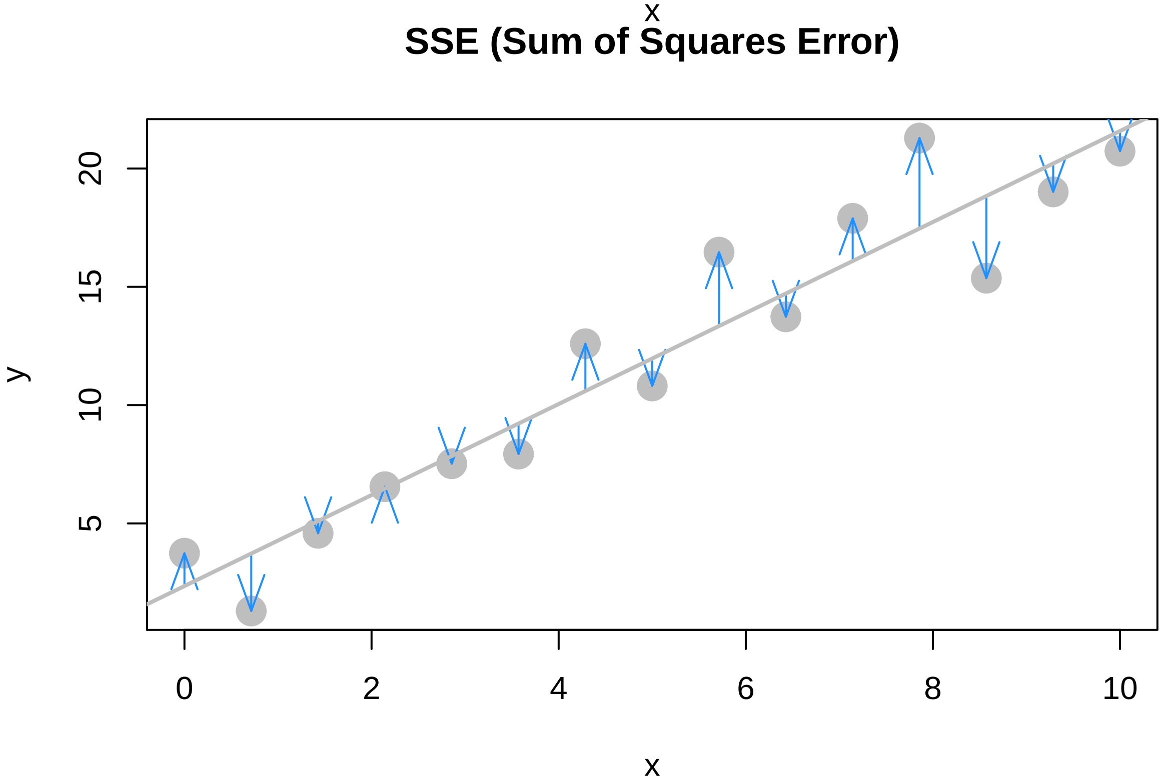
```
R2 = SSReg / SST
R2
```

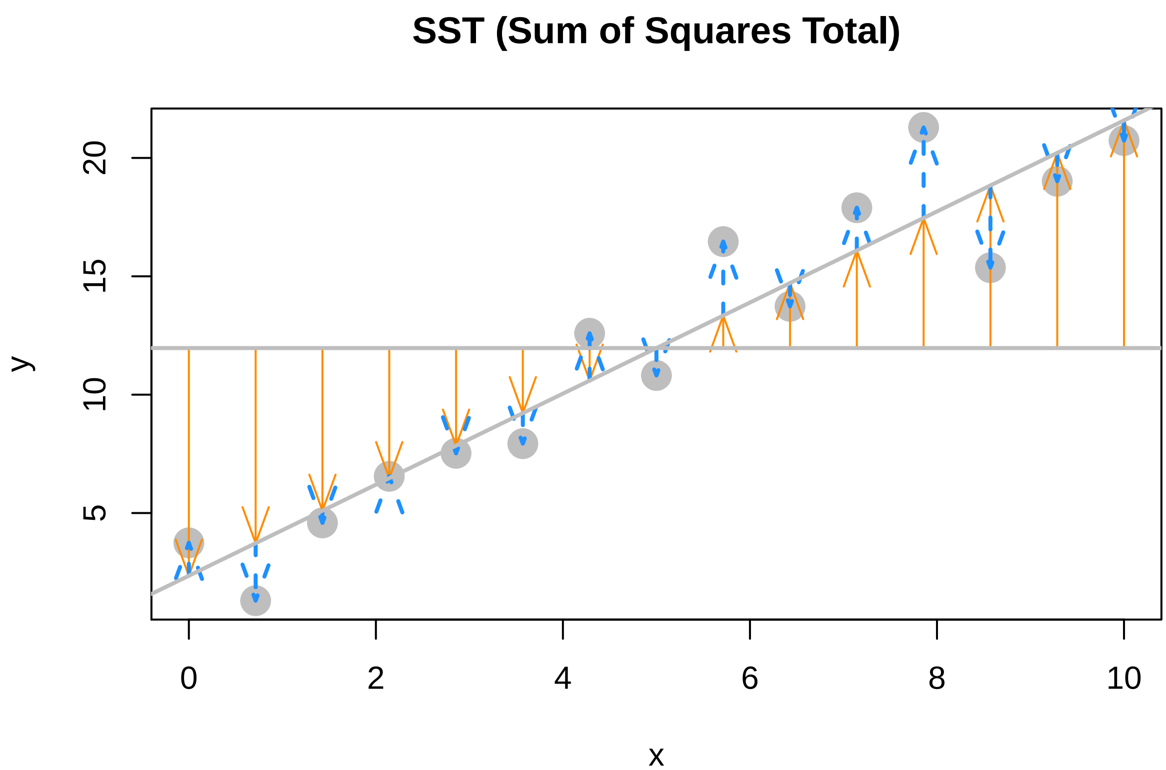
```
## [1] 0.6510794
```

For the `cars` example, we calculate $R^2 = 0.651$. We then say that 65.1% of the observed variability in stopping distance is explained by the linear relationship with speed.

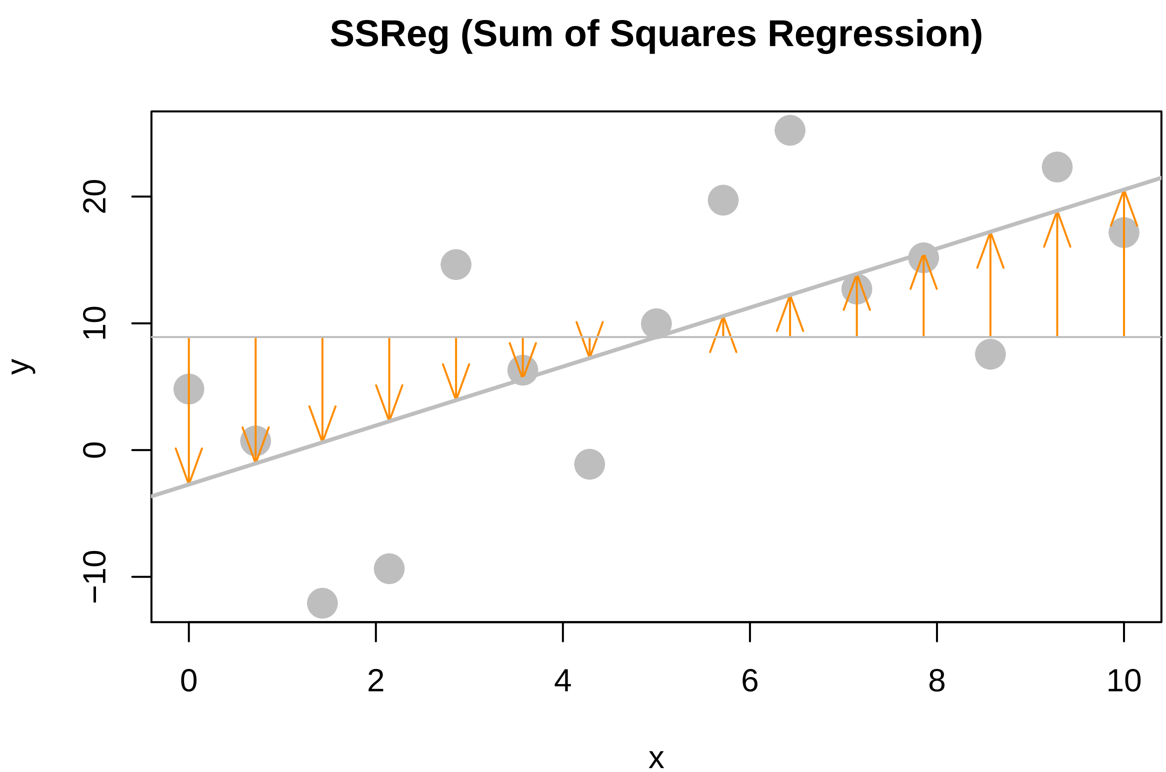
The following three plots visually demonstrate the three "sums of squares" for a simulated dataset which has $R^2 = 0.901$ which is a somewhat high value. Notice in the third plot, that the orange arrows account for a larger proportion of the total arrow.

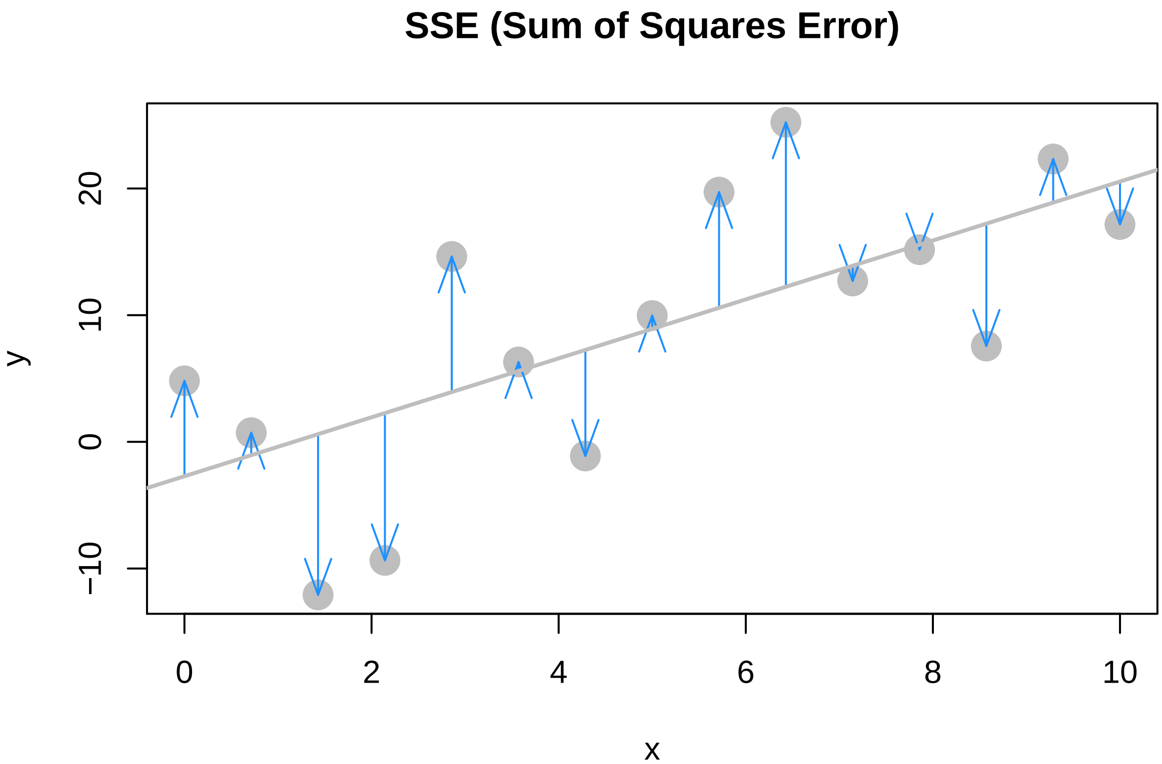

SSReg (Sum of Squares Regression)
y
5
10
15
20
0
2
4
6
8
10
x

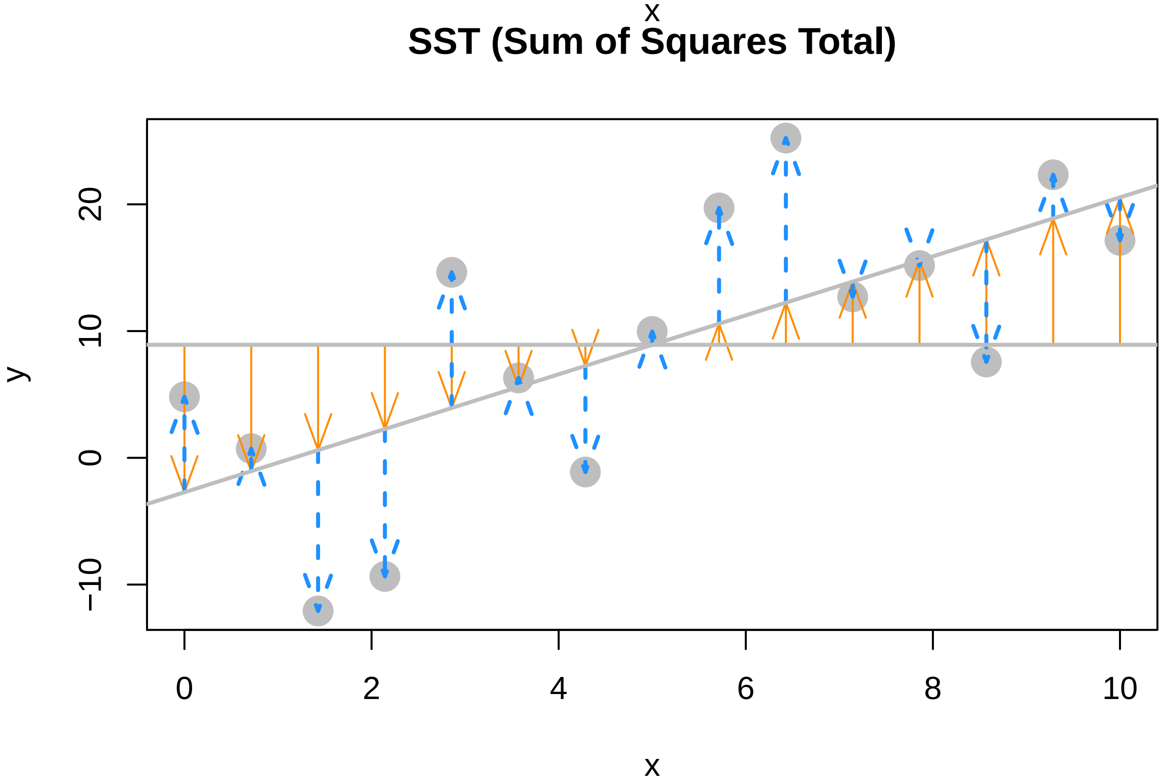

SSE (Sum of Squares Error)
y
5
10
15
20
0
2
4
6
8
10
x



The next three plots again visually demonstrate the three "sums of squares," this time for a simulated dataset which has $R^2 = 0.459$. Notice in the third plot, that now the blue arrows account for a larger proportion of the total arrow.







3.4 The `lm` Function

So far we have done regression by deriving the least squares estimates, then writing simple `R` commands to perform the necessary calculations. Since this is such a common task, this is functionality that is built directly into R via the lm() command.

The lm() command is used to fit **linear models** which actually account for a broader class of models than simple linear regression, but we will use SLR as our first demonstration of lm(). The lm() function will be one of our most commonly used tools, so you may want to take a look at the documentation by using ?lm. You'll notice there is a lot of information there, but we will start with just the very basics. This is documentation you will want to return to often.

We'll continue using the cars data, and essentially use the lm() function to check the work we had previously done.

```
stop_dist_model = lm(dist ~ speed, data = cars)
```

This line of code fits our very first linear model. The syntax should look somewhat familiar. We use the dist ~ speed syntax to tell R we would like to model the response variable dist as a linear function of the predictor variable speed. In general, you should think of the syntax as response ~ predictor. The data = cars argument then tells R that that dist and speed variables are from the dataset cars. We then store this result in a variable stop_dist_model.

The variable stop_dist_model now contains a wealth of information, and we will now see how to extract and use that information. The first thing we will do is simply output whatever is stored immediately in the variable stop_dist_model.

```
stop_dist_model
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Coefficients:
## (Intercept)        speed
##     -17.579        3.932
```

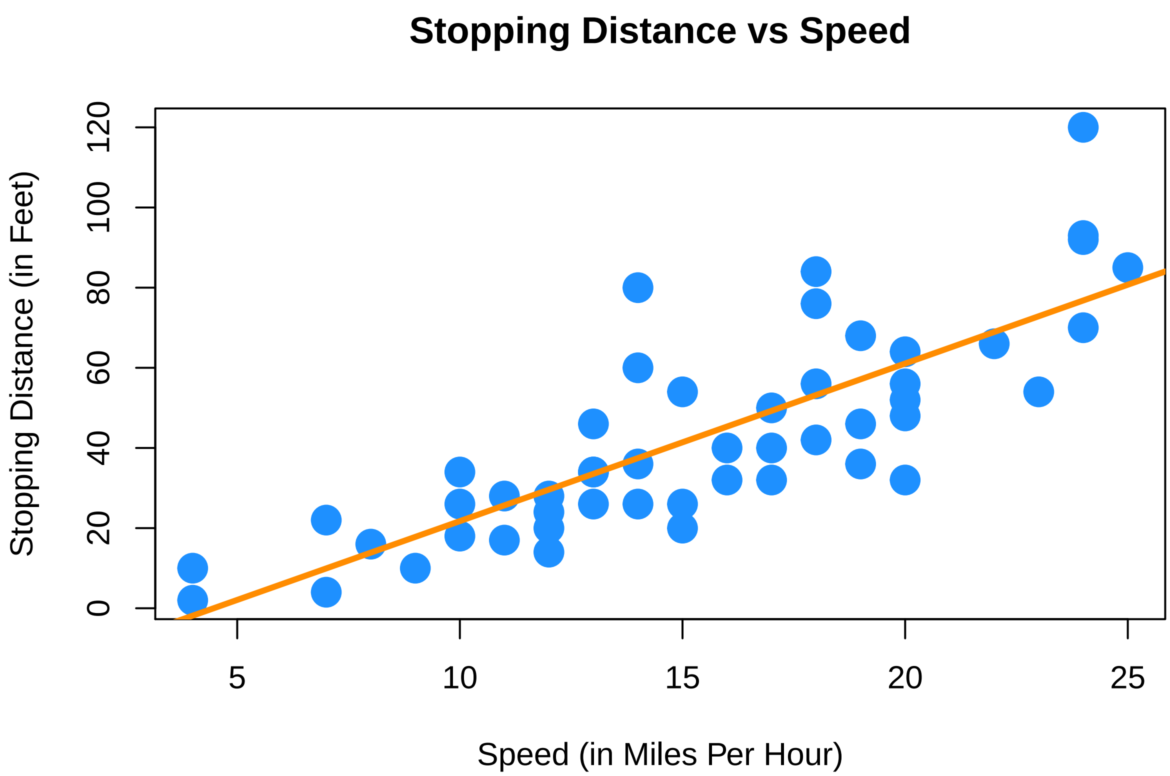
We see that it first tells us the formula we input into R, that is lm(formula = dist ~ speed, data = cars). We also see the coefficients of the model. We can check that these are what we had calculated previously. (Minus some rounding that R is doing to display the results.)

```
c(beta_0_hat, beta_1_hat)
```

```
## [1] -17.579095   3.932409
```

Next, it would be nice to add the fitted line to the scatterplot. To do so we will use the abline() function.

```
plot(dist ~ speed, data = cars,
     xlab = "Speed (in Miles Per Hour)",
     ylab = "Stopping Distance (in Feet)",
     main = "Stopping Distance vs Speed",
     pch  = 20,
     cex  = 3,
     col  = "dodgerblue")
abline(stop_dist_model, lwd = 3, col = "darkorange")
```



The `abline()` function is used to add lines of the form $a + bx$ to a plot. (Hence **`abline`**.) When we give it `stop_dist_model` as an argument, it automatically extracts the regression coefficient estimates ($\hat{\beta}_0$ and $\hat{\beta}_1$) and uses them as the slope and intercept of the line. Here we also use `lwd` to modify the width of the line, as well as `col` to modify the color of the line.

The "thing" that is returned by the `lm()` function is actually an object of class `lm` which is a list. The exact details of this are unimportant unless you are seriously interested in the inner-workings of `R`, but know that we can determine the names of the elements of the list using the `names()` command.

```
names(stop_dist_model)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

When can then use this information to, for example, access the residuals using the `$` operator.

```
stop_dist_model$residuals
```

```
##          1          2          3          4          5          6          7
##   3.849460  11.849460  -5.947766  12.052234   2.119825  -7.812584  -3.744993
##          8          9         10         11         12         13         14
##   4.255007  12.255007  -8.677401   2.322599 -15.609810  -9.609810  -5.609810
##         15         16         17         18         19         20         21
##  -1.609810  -7.542219   0.457781   0.457781  12.457781 -11.474628  -1.474628
##         22         23         24         25         26         27         28
##  22.525372  42.525372 -21.407036 -15.407036  12.592964 -13.339445  -5.339445
##         29         30         31         32         33         34         35
```

```
## -17.271854  -9.271854   0.728146 -11.204263   2.795737  22.795737  30.795737
##         36         37         38         39         40         41         42
## -21.136672 -11.136672  10.863328 -29.069080 -13.069080  -9.069080  -5.069080
##         43         44         45         46         47         48         49
##   2.930920  -2.933898 -18.866307  -6.798715  15.201285  16.201285  43.201285
##         50
##   4.268876
```

Another way to access stored information in `stop_dist_model` are the `coef()`, `resid()`, and `fitted()` functions. These return the coefficients, residuals, and fitted values, respectively.

```
coef(stop_dist_model)
```

```
## (Intercept)       speed
##  -17.579095    3.932409
```

```
resid(stop_dist_model)
```

```
##          1          2          3          4          5          6          7
##   3.849460  11.849460  -5.947766  12.052234   2.119825  -7.812584  -3.744993
##          8          9         10         11         12         13         14
##   4.255007  12.255007  -8.677401   2.322599 -15.609810  -9.609810  -5.609810
##         15         16         17         18         19         20         21
##  -1.609810  -7.542219   0.457781   0.457781  12.457781 -11.474628  -1.474628
##         22         23         24         25         26         27         28
##  22.525372  42.525372 -21.407036 -15.407036  12.592964 -13.339445  -5.339445
##         29         30         31         32         33         34         35
## -17.271854  -9.271854   0.728146 -11.204263   2.795737  22.795737  30.795737
##         36         37         38         39         40         41         42
## -21.136672 -11.136672  10.863328 -29.069080 -13.069080  -9.069080  -5.069080
##         43         44         45         46         47         48         49
##   2.930920  -2.933898 -18.866307  -6.798715  15.201285  16.201285  43.201285
##         50
##   4.268876
```

```
fitted(stop_dist_model)
```

```
##         1         2         3         4         5         6         7         8
## -1.849460 -1.849460  9.947766  9.947766 13.880175 17.812584 21.744993 21.744993
##         9        10        11        12        13        14        15        16
## 21.744993 25.677401 25.677401 29.609810 29.609810 29.609810 29.609810 33.542219
##        17        18        19        20        21        22        23        24
## 33.542219 33.542219 33.542219 37.474628 37.474628 37.474628 37.474628 41.407036
##        25        26        27        28        29        30        31        32
## 41.407036 41.407036 45.339445 45.339445 49.271854 49.271854 49.271854 53.204263
##        33        34        35        36        37        38        39        40
## 53.204263 53.204263 53.204263 57.136672 57.136672 57.136672 61.069080 61.069080
##        41        42        43        44        45        46        47        48
## 61.069080 61.069080 61.069080 68.933898 72.866307 76.798715 76.798715 76.798715
##        49        50
## 76.798715 80.731124
```

An R function that is useful in many situations is `summary()`. We see that when it is called on our model, it returns a good deal of information. By the end of the course, you will know what every value here is used for. For now, you should immediately notice the coefficient estimates, and you may recognize the R^2 value we saw earlier.

```
summary(stop_dist_model)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

The `summary()` command also returns a list, and we can again use `names()` to learn what about the elements of this list.

```
names(summary(stop_dist_model))
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

So, for example, if we wanted to directly access the value of R^2, instead of copy and pasting it out of the printed statement from `summary()`, we could do so.

```
summary(stop_dist_model)$r.squared
```

```
## [1] 0.6510794
```

Another value we may want to access is s_e, which R calls `sigma`.

```
summary(stop_dist_model)$sigma
```

```
## [1] 15.37959
```

Note that this is the same result seen earlier as `s_e`. You may also notice that this value was display above as a result of the `summary()` command, which R labeled the "Residual Standard Error."

$$s_e = RSE = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n} e_i^2}$$

Often it is useful to talk about s_e (or RSE) instead of s_e^2 because of their units. The units of s_e in the `cars` example is feet, while the units of s_e^2 is feet-squared.

Another useful function, which we will use almost as often as `lm()` is the `predict()` function.

```
predict(stop_dist_model, data.frame(speed = 8))
```

```
##        1
## 13.88018
```

The above code reads "predict the stopping distance of a car traveling 8 miles per hour using the `stop_dist_model`." Importantly, the second argument to `predict()` is a data frame that we make in place. We do this so that we can specify that `8` is a value of `speed`, so that predict knows how to use it with the model stored in `stop_dist_model`. We see that this result is what we had calculated "by hand" previously.

We could also predict multiple values at once.

```
predict(stop_dist_model, data.frame(speed = c(8, 21, 50)))
```

```
##         1         2         3
##  13.88018  65.00149 179.04134
```

$$\begin{aligned}
\hat{y} &= -17.579 + 3.932 \times 8 = 13.88 \\
\hat{y} &= -17.579 + 3.932 \times 21 = 65.001 \\
\hat{y} &= -17.579 + 3.932 \times 50 = 179.041
\end{aligned}$$

Or we could calculate the fitted value for each of the original data points.

```
predict(stop_dist_model, data.frame(speed = cars$speed))
```

```
##         1         2         3         4         5         6         7         8
## -1.849460 -1.849460  9.947766  9.947766 13.880175 17.812584 21.744993 21.744993
##         9        10        11        12        13        14        15        16
## 21.744993 25.677401 25.677401 29.609810 29.609810 29.609810 29.609810 33.542219
##        17        18        19        20        21        22        23        24
## 33.542219 33.542219 33.542219 37.474628 37.474628 37.474628 37.474628 41.407036
##        25        26        27        28        29        30        31        32
## 41.407036 41.407036 45.339445 45.339445 49.271854 49.271854 49.271854 53.204263
##        33        34        35        36        37        38        39        40
## 53.204263 53.204263 53.204263 57.136672 57.136672 57.136672 61.069080 61.069080
##        41        42        43        44        45        46        47        48
## 61.069080 61.069080 61.069080 68.933898 72.866307 76.798715 76.798715 76.798715
##        49        50
## 76.798715 80.731124
```

This is actually equivalent to simply calling `predict()` on `stop_dist_model` without a second argument.

```
predict(stop_dist_model)
```

```
##         1         2         3         4         5         6         7         8
## -1.849460 -1.849460  9.947766  9.947766 13.880175 17.812584 21.744993 21.744993
##         9        10        11        12        13        14        15        16
## 21.744993 25.677401 25.677401 29.609810 29.609810 29.609810 29.609810 33.542219
##        17        18        19        20        21        22        23        24
## 33.542219 33.542219 33.542219 37.474628 37.474628 37.474628 37.474628 41.407036
##        25        26        27        28        29        30        31        32
## 41.407036 41.407036 45.339445 45.339445 49.271854 49.271854 49.271854 53.204263
##        33        34        35        36        37        38        39        40
## 53.204263 53.204263 53.204263 57.136672 57.136672 57.136672 61.069080 61.069080
##        41        42        43        44        45        46        47        48
## 61.069080 61.069080 61.069080 68.933898 72.866307 76.798715 76.798715 76.798715
##        49        50
## 76.798715 80.731124
```

Note that then in this case, this is the same as using `fitted()`.

```
fitted(stop_dist_model)
```

```
##         1         2         3         4         5         6         7         8
## -1.849460 -1.849460  9.947766  9.947766 13.880175 17.812584 21.744993 21.744993
##         9        10        11        12        13        14        15        16
## 21.744993 25.677401 25.677401 29.609810 29.609810 29.609810 29.609810 33.542219
##        17        18        19        20        21        22        23        24
## 33.542219 33.542219 33.542219 37.474628 37.474628 37.474628 37.474628 41.407036
##        25        26        27        28        29        30        31        32
## 41.407036 41.407036 45.339445 45.339445 49.271854 49.271854 49.271854 53.204263
##        33        34        35        36        37        38        39        40
## 53.204263 53.204263 53.204263 57.136672 57.136672 57.136672 61.069080 61.069080
##        41        42        43        44        45        46        47        48
## 61.069080 61.069080 61.069080 68.933898 72.866307 76.798715 76.798715 76.798715
##        49        50
## 76.798715 80.731124
```

3.5 Maximum Likelihood Estimation (MLE) Approach

Recall the model,

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where $e_i \sim N(0, \sigma^2)$.

Then we can find the mean and variance of each Y_i.

$$E[Y_i] = \beta_0 + \beta_1 x_i$$

and

$$Var[Y_i] = \sigma^2.$$

Recall that the pdf of a random variable $X \sim N(\mu, \sigma^2)$ is given by

$$f_X(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right].$$

Then we can write the pdf of each of the Y_i as

$$f_{Y_i}(y_i; x_i, \beta_0, \beta_1, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}\left(\frac{y_i - (\beta_0 + \beta_1 x_i)}{\sigma}\right)^2\right].$$

Given n data points (x_i, y_i) we can write the likelihood, which is a function of the three parameters β_0, β_1, and σ^2. Since the data have been observed, we use lower case y_i to denote that these values are no longer random.

$$L(\beta_0, \beta_1, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}\left(\frac{y_i - \beta_0 - \beta_1 x_i}{\sigma}\right)^2\right]$$

Our goal is to find values of β_0, β_1, and σ^2 which maximize this function, which is a straightforward multivariate calculus problem.

We'll start by doing a bit of rearranging to make our task easier.

$$L(\beta_0, \beta_1, \sigma^2) = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left[-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2\right]$$

Then, as is often the case when finding MLEs, for mathematical convenience we will take the natural logarithm of the likelihood function since log is a monotonically increasing function. Then we will proceed to maximize the log-likelihood, and the resulting estimates will be the same as if we had not taken the log.

$$\log L(\beta_0, \beta_1, \sigma^2) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$$

Note that we use log to mean the natural logarithm. We now take a partial derivative with respect to each of the parameters.

$$\begin{aligned}
\frac{\partial \log L(\beta_0, \beta_1, \sigma^2)}{\partial \beta_0} &= \frac{1}{\sigma^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i) \\
\frac{\partial \log L(\beta_0, \beta_1, \sigma^2)}{\partial \beta_1} &= \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i)(y_i - \beta_0 - \beta_1 x_i) \\
\frac{\partial \log L(\beta_0, \beta_1, \sigma^2)}{\partial \sigma^2} &= -\frac{n}{2\sigma^2} + \frac{1}{2(\sigma^2)^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2
\end{aligned}$$

We then set each of the partial derivatives equal to zero and solve the resulting system of equations.

$$
\begin{aligned}
\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i) &= 0 \\
\sum_{i=1}^{n}(x_i)(y_i - \beta_0 - \beta_1 x_i) &= 0 \\
-\frac{n}{2\sigma^2} + \frac{1}{2(\sigma^2)^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2 &= 0
\end{aligned}
$$

You may notice that the first two equations also appear in the least squares approach. Then, skipping the issue of actually checking if we have found a maximum, we then arrive at our estimates. We call these estimates the maximum likelihood estimates.

$$
\begin{aligned}
\hat{\beta}_1 &= \frac{\sum_{i=1}^{n} x_i y_i - \frac{(\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n}}{\sum_{i=1}^{n} x_i^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n}} = \frac{S_{xy}}{S_{xx}} \\
\hat{\beta}_0 &= \bar{y} - \hat{\beta}_1 \bar{x} \\
\hat{\sigma}^2 &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2
\end{aligned}
$$

Note that $\hat{\beta}_0$ and $\hat{\beta}_1$ are the same as the least squares estimates. However we now have a new estimate of σ^2, that is $\hat{\sigma}^2$. So we now have two different estimates of σ^2.

$$
\begin{aligned}
s_e^2 &= \frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n-2}\sum_{i=1}^{n} e_i^2 \quad \text{Least Squares} \\
\hat{\sigma}^2 &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n} e_i^2 \quad \text{MLE}
\end{aligned}
$$

In the next chapter, we will discuss in detail the difference between these two estimates, which involves biasedness.

3.6 Simulating SLR

We return again to more examples of simulation. This will be a common theme!

In practice you will almost never have a true model, and you will use data to attempt to recover information about the unknown true model. With simulation, we decide the true model and simulate data from the it. Then we apply a method to the data, in this case least squares. Now, since we know the true model, we can assess how well it did.

For this example, we will simulate `n = 20` observations from the model

$$y_i = 5 + 2x_i + \epsilon_i.$$

That is $\beta_0 = 5$, $\beta_1 = 2$, and let $\epsilon_i \sim N(\mu = 0, \sigma^2 = 1)$.

We first set the parameters of the simulation.

```
n       = 20
beta_0  = 5
beta_1  = 2
sigma   = 1
```

Next, we obtain simulated values of ϵ_i.

```
epsilon = rnorm(n, mean = 0, sd = sigma)
```

Now, since the x_i values in SLR are considered fixed and known, we simply generate them from a uniform distribution. Know, that this is an arbitrary, but common practice.

```
x = runif(n, 0, 10)
```

We then generate the y values according the specified functional relationship.

```
y = beta_0 + beta_1 * x + epsilon
```

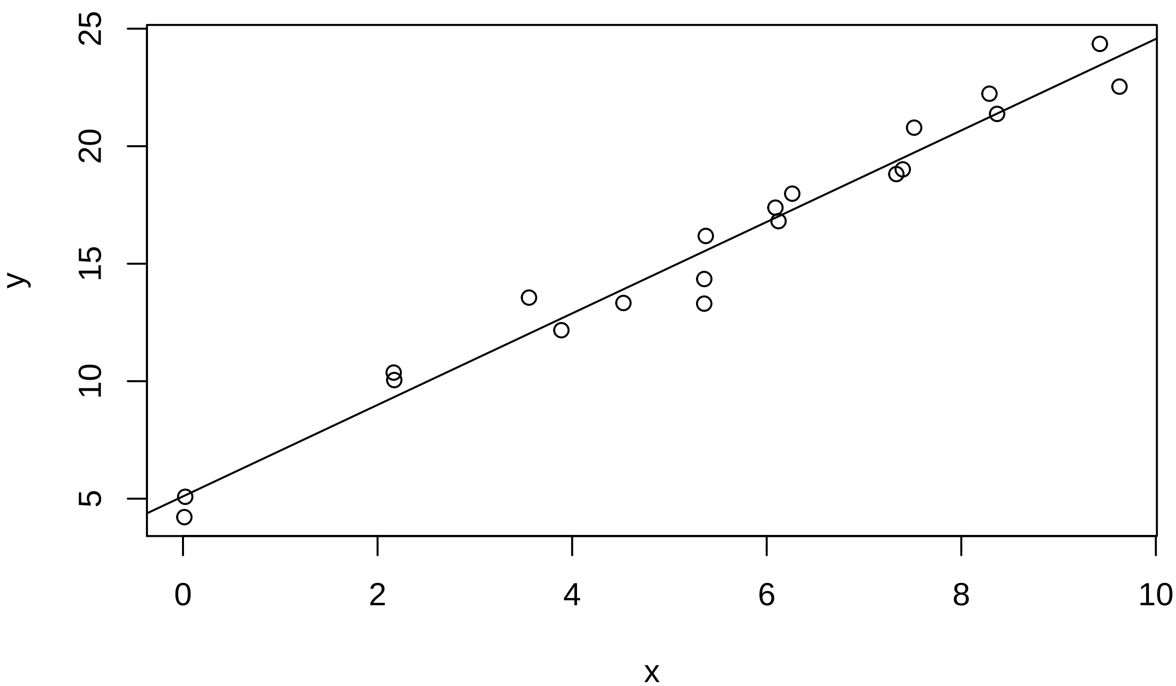
Now to check how well the method of least squares works, we use `lm()` to fit the model to our data, then take a look at the estimated coefficients.

```
sim_fit = lm(y ~ x)
coef(sim_fit)
```

```
## (Intercept)           x
##    5.098490    1.946622
```

And look at that, they aren't too far from the parameters we specified!

```
plot(y ~ x)
abline(sim_fit)
```



We should say here, that we're being sort of lazy, and not the good kinda of lazy that could be considered efficient. Any time you simulate data, you should consider doing two things: writing a function, and storing the data in a data frame.

The function below, `sim_slr()` can be used for the same task as above, but is much more flexible.

```
sim_slr = function(n, beta_0 = 10, beta_1 = 5, sigma = 1, xmin = 0, xmax = 10) {
  epsilon = rnorm(n, mean = 0, sd = sigma)
  x       = runif(n, xmin, xmax)
  y       = beta_0 + beta_1 * x + epsilon
  data.frame(predictor = x, response = y)
}
```

Here, we use the function to repeat the analysis above.

```
sim_data = sim_slr(n = 20, beta_0 = 5, beta_1 = 2, sigma = 1)
```

This time, the simulated observations are stored in a data frame.

```
head(sim_data)
```

```
##   predictor  response
## 1  9.348230 23.329224
## 2  5.504941 16.195112
## 3  6.017662 17.617148
## 4  1.969945 10.339627
## 5  5.352366 14.977440
## 6  1.795557  9.893657
```

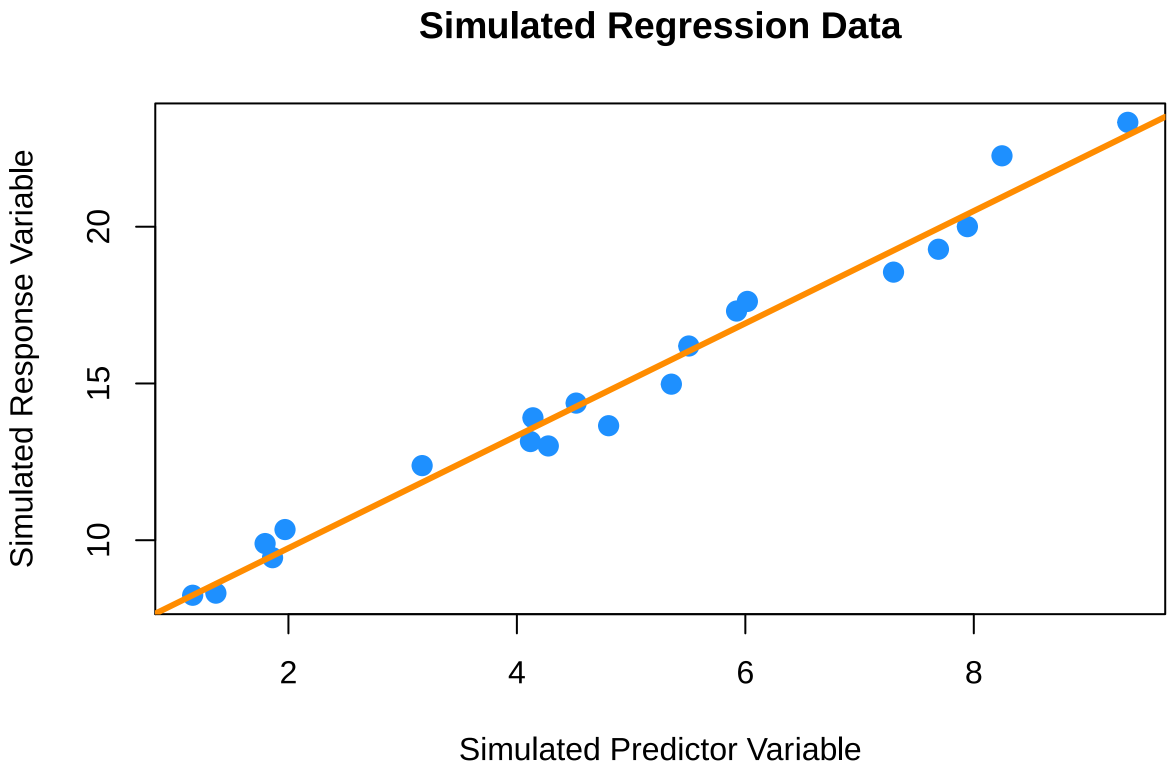
Now when we fit the model with `lm()` we can use a `data` argument, a very good practice.

```
sim_fit = lm(response ~ predictor, data = sim_data)
coef(sim_fit)
```

```
## (Intercept)   predictor
##    6.162023    1.792461
```

And this time, we'll make the plot look a lot nicer.

```
plot(response ~ predictor, data = sim_data,
     xlab = "Simulated Predictor Variable",
     ylab = "Simulated Response Variable",
     main = "Simulated Regression Data",
     pch  = 20,
     cex  = 2,
     col  = "dodgerblue")
abline(sim_fit, lwd = 3, col = "darkorange")
```



3.7 History

For some brief background on the history of linear regression, see "Galton, Pearson, and the Peas: A Brief History of Linear Regression for Statistics Instructors" from the Journal of Statistics Education as well as the Wikipedia page on the history of regression analysis and lastly the article for regression to the mean which details the origins of the term "regression."

Chapter 4

Inference for Simple Linear Regression

> "There are three types of lies: lies, damn lies, and statistics."
>
> — **Benjamin Disraeli**

After reading this chapter you will be able to:

- Understand the distributions of regression estimates.
- Create interval estimates for regression parameters.
- Test for significance of regression.

Last chapter we defined the simple linear regression model,

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$. We then used observations (x_i, y_i), for $i = 1, 2, \ldots n$, to find values of β_0 and β_1 which minimized

$$f(\beta_0, \beta_1) = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_i))^2.$$

We called these values $\hat{\beta}_0$ and $\hat{\beta}_1$, which we found to be

$$\begin{aligned}
\hat{\beta}_1 &= \frac{S_{xy}}{S_{xx}} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\
\hat{\beta}_0 &= \bar{y} - \hat{\beta}_1 \bar{x}.
\end{aligned}$$

We also estimated σ^2 using s_e^2. In other words, we found that s_e is an estimate of σ, where;

$$s_e = RSE = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n} e_i^2}$$

which we also called RSE, for "Residual Standard Error."

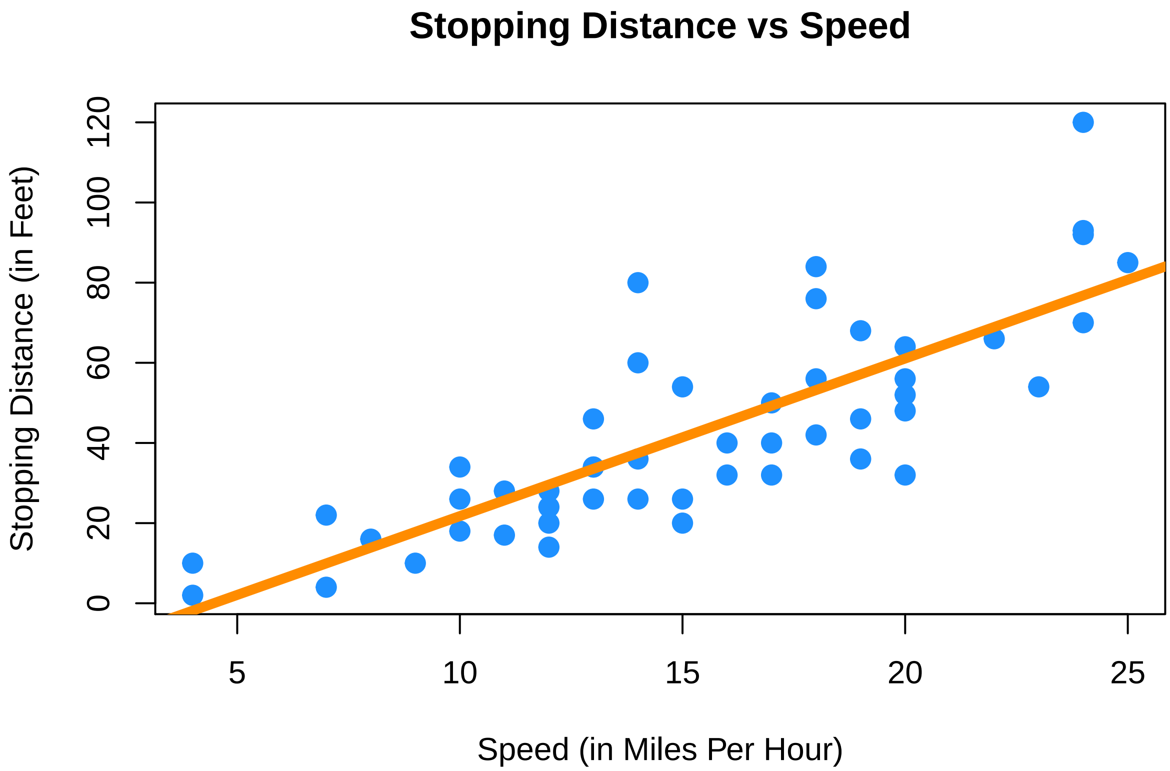
When applied to the `cars` data, we obtained the following results:

```r
stop_dist_model = lm(dist ~ speed, data = cars)
summary(stop_dist_model)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

Last chapter, we only discussed the `Estimate`, `Residual standard error`, and `Multiple R-squared` values. In this chapter, we will discuss all of the information under `Coefficients` as well as `F-statistic`.

```r
plot(dist ~ speed, data = cars,
     xlab = "Speed (in Miles Per Hour)",
     ylab = "Stopping Distance (in Feet)",
     main = "Stopping Distance vs Speed",
     pch  = 20,
     cex  = 2,
     col  = "dodgerblue")
abline(stop_dist_model, lwd = 5, col = "darkorange")
```



To get started, we'll note that there is another equivalent expression for S_{xy} which we did not see last chapter,

$$S_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^{n}(x_i - \bar{x})y_i.$$

This may be a surprising equivalence. (Maybe try to prove it.) However, it will be useful for illustrating concepts in this chapter.

Note that, $\hat{\beta}_1$ is a **statistic** when calculated with observed data as written above, as is $\hat{\beta}_0$.

However, in this chapter it will often be convenient to use both $\hat{\beta}_1$ and $\hat{\beta}_0$ as **random variables**, that is, we have not yet observed the values for each Y_i. When this is the case, we will use a slightly different notation, substituting in capital Y_i for lower case y_i.

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})Y_i}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$
$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1\bar{x}$$

Last chapter we argued that these estimates of unknown model parameters β_0 and β_1 were good because we obtained them by minimizing errors. We will now discuss the Gauss–Markov theorem which takes this idea further, showing that these estimates are actually the "best" estimates, from a certain point of view.

4.1 Gauss–Markov Theorem

The **Gauss–Markov theorem** tells us that when estimating the parameters of the simple linear regression model β_0 and β_1, the $\hat{\beta}_0$ and $\hat{\beta}_1$ which we derived are the **best linear unbiased estiamtes**, or *BLUE* for short. (The actual conditions for the Gauss–Markov theorem are more relaxed than the SLR model.)

We will now discuss *linear*, *unbiased*, and *best* as is relates to these estimates.

Linear

Recall, in the SLR setup that the x_i values are considered fixed and known quantities. Then a **linear** estimate is one which can be written as a linear combination of the Y_i. In the case of $\hat{\beta}_1$ we see

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})Y_i}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \sum_{i=1}^{n} k_i Y_i = k_1 Y_1 + k_2 Y_2 + \cdots k_n Y_n$$

where $k_i = \dfrac{(x_i - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$.

In a similar fashion, we could show that $\hat{\beta}_0$ can be written as a linear combination of the Y_i. Thus both $\hat{\beta}_0$ and $\hat{\beta}_1$ are linear estimators.

Unbiased

Now that we know our estimates are *linear*, how good are these estimates? One measure of the "goodness" of an estimate is its **bias**. Specifically, we prefer estimates that are **unbiased**, meaning their expected value is the parameter being estimated.

In the case of the regression estimates, we have,

$$E[\hat{\beta}_0] = \beta_0$$
$$E[\hat{\beta}_1] = \beta_1.$$

This tells us that, when the conditions of the SLR model are met, on average our estimates will be correct. However, as we saw last chapter when simulating from the SLR model, that does not mean that each individual estimate will be correct. Only that, if we repeated the process an infinite number of times, on average the estimate would be correct.

Best

Now, if we restrict ourselves to both *linear* and *unbiased* estimates, how do we define the *best* estimate? The estimate with the **minimum variance**.

First note that it is very easy to create an estimate for β_1 that has very low variance, but is not unbiased. For example, define:

$$\hat{\theta}_{BAD} = 5.$$

Then, since $\hat{\theta}_{BAD}$ is a constant value,

$$Var[\hat{\theta}_{BAD}] = 0.$$

However since,

$$E[\hat{\theta}_{BAD}] = 5$$

we say that $\hat{\theta}_{BAD}$ is a biased estimator unless $\beta_1 = 5$, which we would not know ahead of time. For this reason, it is a terrible estimate (unless by chance $\beta_1 = 5$) even though it has the smallest possible variance.

This is part of the reason we restrict ourselves to *unbiased* estimates. What good is an estimate, if it estimates the wrong quantity?

So now, the natural question is, what are the variances of $\hat{\beta}_0$ and $\hat{\beta}_1$? They are,

$$
\begin{aligned}
Var[\hat{\beta}_0] &= \sigma^2 \left(\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}\right) \\
Var[\hat{\beta}_1] &= \frac{\sigma^2}{S_{xx}}.
\end{aligned}
$$

These quantify the variability of the estimates due to random chance during sampling. Are these "the best"? Are these variances as small as we can possibility get? You'll just have to take our word for it that they are because showing that this is true is beyond the score of this course.

4.2 Sampling Distributions

Now that we have "redefined" the estimates for $\hat{\beta}_0$ and $\hat{\beta}_1$ as random variables, we can discuss their **sampling distribution**, which is the distribution when a statistic is considered a random variable.

Since both $\hat{\beta}_0$ and $\hat{\beta}_1$ are a linear combination of the Y_i and each Y_i is normally distributed, then both $\hat{\beta}_0$ and $\hat{\beta}_1$ also follow a normal distribution.

Then, putting all of the above together, we arrive at the distributions of $\hat{\beta}_0$ and $\hat{\beta}_1$.

For $\hat{\beta}_1$ we say,

$$
\hat{\beta}_1 = \frac{S_{xy}}{S_{xx}} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})Y_i}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \sim N\left(\beta_1, \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\right).
$$

Or more succinctly,

$$
\hat{\beta}_1 \sim N\left(\beta_1, \frac{\sigma^2}{S_{xx}}\right).
$$

And for $\hat{\beta}_0$,

$$
\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1\bar{x} \sim N\left(\beta_0, \frac{\sigma^2 \sum_{i=1}^{n} x_i^2}{n\sum_{i=1}^{n}(x_i - \bar{x})^2}\right).
$$

Or more succinctly,

$$
\hat{\beta}_0 \sim N\left(\beta_0, \sigma^2\left(\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}\right)\right)
$$

At this point we have neglected to prove a number of these results. Instead of working through the tedious derivations of these sampling distributions, we will instead justify these results to ourselves using simulation.

A note to current readers: These derivations and proofs may be added to an appendix at a later time. You can also find these results in nearly any standard linear regression textbook. At UIUC, these results will likely be presented in both STAT 424 and STAT 425. However, since you will not be asked to perform derivations of this type in this course, they are for now omitted.

4.2.1 Simulating Sampling Distributions

To verify the above results, we will simulate samples of size $n = 100$ from the model

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$. In this case, the parameters are known to be:

- $\beta_0 = 3$
- $\beta_1 = 6$
- $\sigma^2 = 4$

Then, based on the above, we should find that

$$\hat{\beta}_1 \sim N\left(\beta_1, \frac{\sigma^2}{S_{xx}}\right)$$

and

$$\hat{\beta}_0 \sim N\left(\beta_0, \sigma^2\left(\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}\right)\right).$$

First we need to decide ahead of time what our x values will be for this simulation, since the x values in SLR are also considered known quantities. The choice of x values is arbitrary. Here we also set a seed for randomization, and calculate S_{xx} which we will need going forward.

```
set.seed(42)
sample_size = 100 # this is n
x = seq(-1, 1, length = sample_size)
Sxx = sum((x - mean(x)) ^ 2)
```

We also fix our parameter values.

```
beta_0 = 3
beta_1 = 6
sigma  = 2
```

With this information, we know the sampling distributions should be:

```
(var_beta_1_hat = sigma ^ 2 / Sxx)
```

```
## [1] 0.1176238
```

```
(var_beta_0_hat = sigma ^ 2 * (1 / sample_size + mean(x) ^ 2 / Sxx))
```

```
## [1] 0.04
```

$$\hat{\beta}_1 \sim N(6, 0.1176238)$$

and

$$\hat{\beta}_0 \sim N(3, 0.04).$$

That is,

$$E[\hat{\beta}_1] = 6$$
$$Var[\hat{\beta}_1] = 0.1176238$$

and

$$E[\hat{\beta}_0] = 3$$
$$Var[\hat{\beta}_0] = 0.04.$$

We now simulate this model 10,000 times. Note this may not be the most `R` way of doing the simulation. We perform the simulation in this manner in an attempt at clarity.

```
num_samples = 10000
beta_0_hats = rep(0, num_samples)
beta_1_hats = rep(0, num_samples)

for(i in 1:num_samples) {
  eps = rnorm(sample_size, mean = 0, sd = sigma)
  y   = beta_0 + beta_1 * x + eps

  sim_model = lm(y ~ x)

  beta_0_hats[i] = coef(sim_model)[1]
  beta_1_hats[i] = coef(sim_model)[2]
}
```

The variables `beta_0_hats` and `beta_1_hats` now store 10,000 simulated values of $\hat{\beta}_0$ and $\hat{\beta}_1$ respectively.

We first verify the distribution of $\hat{\beta}_1$.

```
mean(beta_1_hats) # empirical mean
```

```
## [1] 6.001998
```

```
beta_1            # true mean
```

```
## [1] 6
```

```
var(beta_1_hats)  # empirical variance
```

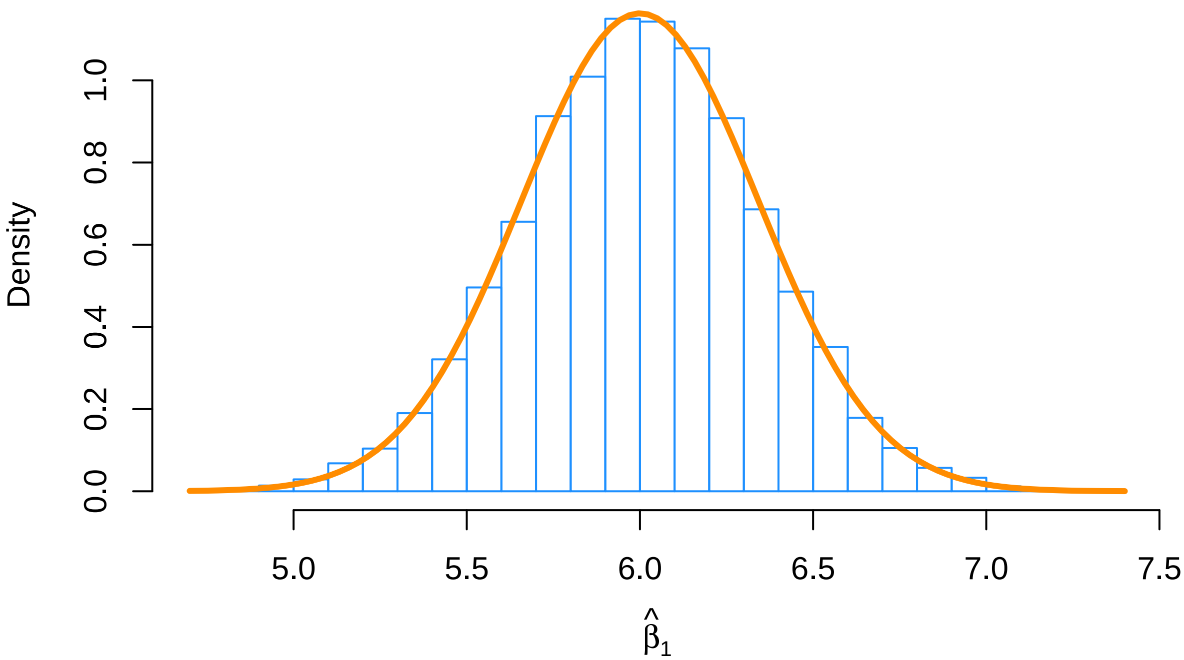
```
## [1] 0.11899
```

```
var_beta_1_hat    # true variance
```

```
## [1] 0.1176238
```

We see that the empirical and true means and variances are *very* similar. We also verify that the empirical distribution is normal. We plot a histogram of the `beta_1_hats`, and add the curve for the true distribution of β_1. We use `prob = TRUE` to put the histogram on the same scale as the normal curve.

```
# note need to use prob = TRUE
hist(beta_1_hats, prob = TRUE, breaks = 20,
     xlab = expression(hat(beta)[1]), main = "", border = "dodgerblue")
curve(dnorm(x, mean = beta_1, sd = sqrt(var_beta_1_hat)),
      col = "darkorange", add = TRUE, lwd = 3)
```



We then repeat the process for $\hat{\beta}_0$.

```
mean(beta_0_hats) # empirical mean
```

```
## [1] 3.001147
```

```
beta_0            # true mean
```

```
## [1] 3
```

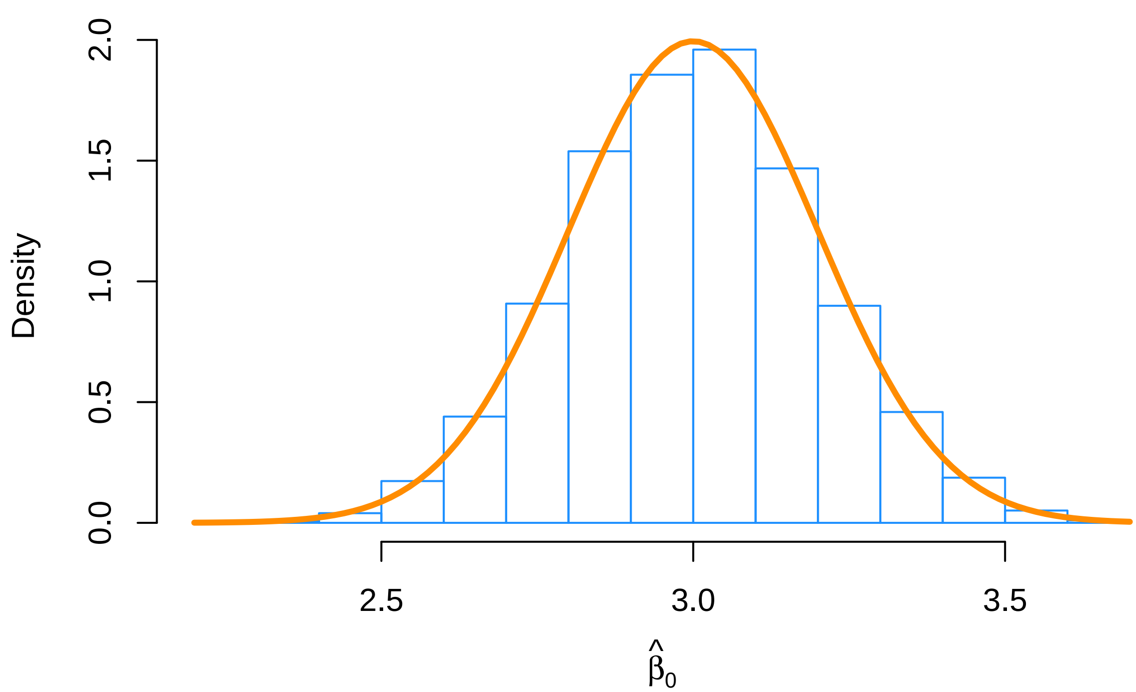
```
var(beta_0_hats)  # empirical variance
```

```
## [1] 0.04017924
```

```
var_beta_0_hat     # true variance
```

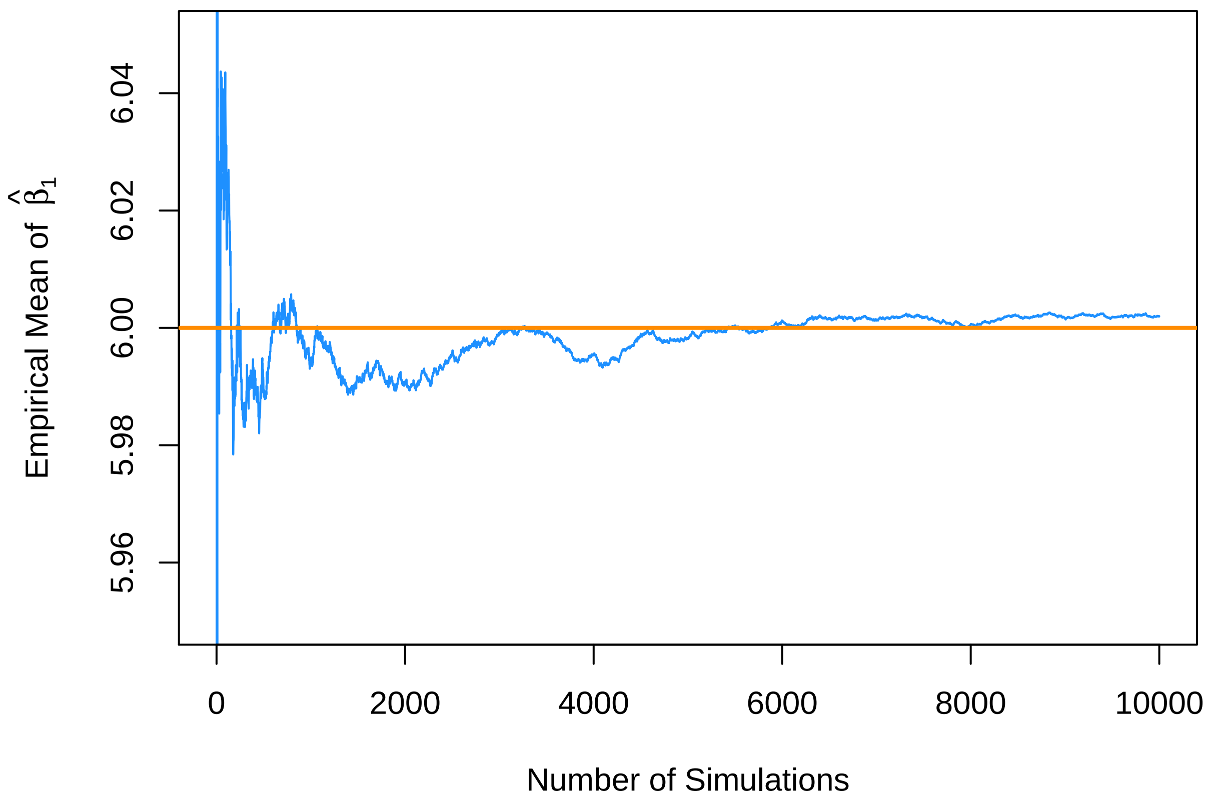
```
## [1] 0.04
```

```
hist(beta_0_hats, prob = TRUE, breaks = 20,
     xlab = expression(hat(beta)[0]), main = "", border = "dodgerblue")
curve(dnorm(x, mean = beta_0, sd = sqrt(var_beta_0_hat)),
      col = "darkorange", add = TRUE, lwd = 3)
```

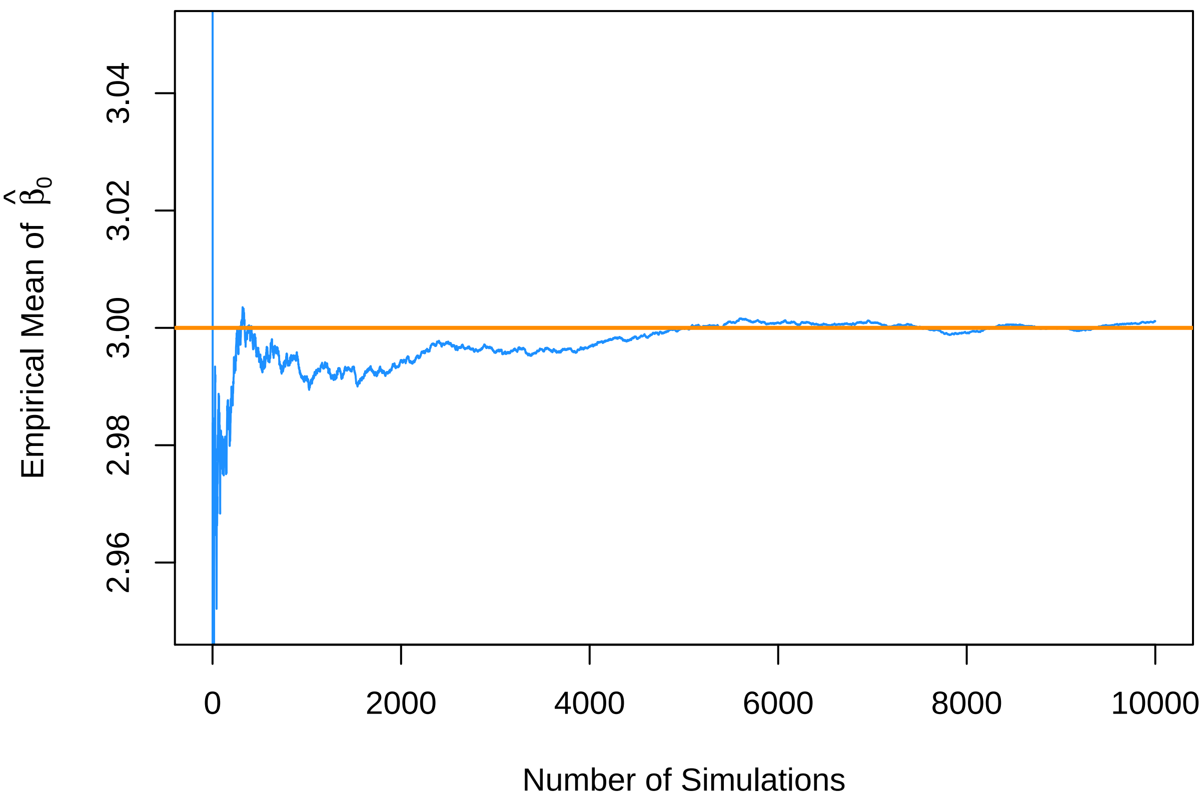


In this simulation study, we have only simulated a finite number of samples. To truly verify the distributional results, we would need to observe an infinite number of samples. However, the following plot should make it clear that if we continued simulating, the empirical results would get closer and closer to what we should expect.

```
par(mar = c(5, 5, 1, 1)) # adjusted plot margins, otherwise the "hat" does not display
plot(cumsum(beta_1_hats) / (1:length(beta_1_hats)), type = "l", ylim = c(5.95, 6.05),
     xlab = "Number of Simulations",
     ylab = expression("Empirical Mean of " ~ hat(beta)[1]),
     col  = "dodgerblue")
abline(h = 6, col = "darkorange", lwd = 2)
```




```r
par(mar = c(5, 5, 1, 1)) # adjusted plot margins, otherwise the "hat" does not display
plot(cumsum(beta_0_hats) / (1:length(beta_0_hats)), type = "l", ylim = c(2.95, 3.05),
     xlab = "Number of Simulations",
     ylab = expression("Empirical Mean of " ~ hat(beta)[0]),
     col  = "dodgerblue")
abline(h = 3, col = "darkorange", lwd = 2)
```



4.3 Standard Errors

So now we believe the two distributional results,

$$\hat{\beta}_0 \sim N\left(\beta_0, \sigma^2\left(\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}\right)\right)$$
$$\hat{\beta}_1 \sim N\left(\beta_1, \frac{\sigma^2}{S_{xx}}\right).$$

Then by standardizing these results we find that

$$\frac{\hat{\beta}_0 - \beta_0}{SD[\hat{\beta}_0]} \sim N(0,1)$$

and

$$\frac{\hat{\beta}_1 - \beta_1}{SD[\hat{\beta}_1]} \sim N(0,1)$$

where

$$SD[\hat{\beta}_0] = \sigma\sqrt{\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}}$$

and

$$SD[\hat{\beta}_1] = \frac{\sigma}{\sqrt{S_{xx}}}.$$

Since we don't know σ in practice, we will have to estimate it using s_e, which we plug into our existing expression for the standard deviations of our estimates. We choose s_e instead of $\hat{\sigma}$ because, as you've seen recently, we prize unbiased estimators over biased ones.

These two new expressions are called **standard errors** which are the *estimated* standard deviations of the sampling distributions.

$$SE[\hat{\beta}_0] = s_e \sqrt{\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}}$$

$$SE[\hat{\beta}_1] = \frac{s_e}{\sqrt{S_{xx}}}$$

Now if we divide by the standard error, instead of the standard deviation, we obtain the following results which will allow us to make confidence intervals and perform hypothesis testing.

$$\frac{\hat{\beta}_0 - \beta_0}{SE[\hat{\beta}_0]} \sim t_{n-2}$$

$$\frac{\hat{\beta}_1 - \beta_1}{SE[\hat{\beta}_1]} \sim t_{n-2}$$

To see this, first note that,

$$\frac{RSS}{\sigma^2} = \frac{(n-2)s_e^2}{\sigma^2} \sim \chi^2_{n-2}.$$

Then we use the classic trick of "multiply by 1" and some rearranging to arrive at

$$\begin{aligned}
\frac{\hat{\beta}_1 - \beta_1}{SE[\hat{\beta}_1]} &= \frac{\hat{\beta}_1 - \beta_1}{s_e/\sqrt{S_{xx}}} \\
&= \frac{\hat{\beta}_1 - \beta_1}{s_e/\sqrt{S_{xx}}} \cdot \frac{\sigma/\sqrt{S_{xx}}}{\sigma/\sqrt{S_{xx}}} \\
&= \frac{\hat{\beta}_1 - \beta_1}{\sigma/\sqrt{S_{xx}}} \cdot \frac{\sigma/\sqrt{S_{xx}}}{s_e/\sqrt{S_{xx}}} \\
&= \frac{\hat{\beta}_1 - \beta_1}{\sigma/\sqrt{S_{xx}}} \Bigg/ \sqrt{\frac{s_e^2}{\sigma^2}} \sim \frac{Z}{\sqrt{\frac{\chi^2_{n-2}}{n-2}}} \sim t_{n-2}
\end{aligned}$$

where $Z \sim N(0, 1)$.

Recall that a random variable T defined as,

$$T = \frac{Z}{\sqrt{\frac{\chi^2_d}{d}}}$$

follows a t distribution with d degrees of freedom, where χ^2_d is a χ^2 random variable with d degrees of freedom.

That is,

$$T \sim t_d.$$

4.4 Confidence Intervals for Slope and Intercept

Recall that confidence intervals for means often take the form:

$$EST \pm CRIT \cdot SE$$

or

$$EST \pm MARGIN$$

where EST is an estimate for the parameter of interest, SE is the standard error of the estimate and $MARGIN = CRIT \cdot SE$.

Then, for β_0 and β_1 we can create confidence intervals using

$$\hat{\beta}_0 \pm t_{\alpha/2,n-2} \cdot SE[\hat{\beta}_0] \qquad \hat{\beta}_0 \pm t_{\alpha/2,n-2} \cdot s_e \sqrt{\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}}$$

and

$$\hat{\beta}_1 \pm t_{\alpha/2,n-2} \cdot SE[\hat{\beta}_1] \qquad \hat{\beta}_1 \pm t_{\alpha/2,n-2} \cdot \frac{s_e}{\sqrt{S_{xx}}}$$

where $t_{\alpha/2,n-2}$ is the critical value such that $P(t_{n-2} > t_{\alpha/2,n-2}) = \alpha/2$.

4.5 Hypothesis Tests

> "We may speak of this hypothesis as the 'null hypothesis', and it should be noted that the null hypothesis is never proved or established, but is possibly disproved, in the course of experimentation."
>
> — **Ronald Aylmer Fisher**

Recall that a test statistic (TS) for testing means often take the form:

$$TS = \frac{EST - HYP}{SE}$$

where EST is an estimate for the parameter of interest, HYP is a hypothesized value of the parameter, and SE is the standard error of the estimate.

So, to test

$$H_0 : \beta_0 = \beta_{00} \quad \text{vs} \quad H_1 : \beta_0 \neq \beta_{00}$$

we use the test statistic

$$t = \frac{\hat{\beta}_0 - \beta_{00}}{SE[\hat{\beta}_0]} = \frac{\hat{\beta}_0 - \beta_{00}}{s_e\sqrt{\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}}}$$

which, under the null hypothesis, follows a t distribution with $n-2$ degrees of freedom. We use β_{00} to denote the hypothesized value of β_0.

Similarly, to test

$$H_0 : \beta_1 = \beta_{10} \quad \text{vs} \quad H_1 : \beta_1 \neq \beta_{10}$$

we use the test statistic

$$t = \frac{\hat{\beta}_1 - \beta_{10}}{SE[\hat{\beta}_1]} = \frac{\hat{\beta}_1 - \beta_{10}}{s_e / \sqrt{S_{xx}}}$$

which again, under the null hypothesis, follows a t distribution with $n-2$ degrees of freedom. We now use β_{10} to denote the hypothesized value of β_1.

4.6 cars Example

We now return to the `cars` example from last chapter to illustrate these concepts. We first fit the model using `lm()` then use `summary()` to view the results in greater detail.

```
stop_dist_model = lm(dist ~ speed, data = cars)
summary(stop_dist_model)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

4.6.1 Tests in R

We will now discuss the results displayed called `Coefficients`. First recall that we can extract this information directly.

```r
names(summary(stop_dist_model))
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

```r
summary(stop_dist_model)$coefficients
```

```
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) -17.579095  6.7584402 -2.601058 1.231882e-02
## speed         3.932409  0.4155128  9.463990 1.489836e-12
```

The `names()` function tells us what information is available, and then we use the `$` operator and `coefficients` to extract the information we are interested in. Two values here should be immediately familiar.

$$\hat{\beta}_0 = -17.5790949$$

and

$$\hat{\beta}_1 = 3.9324088$$

which are our estimates for the model parameters β_0 and β_1.

Let's now focus on the second row of output, which is relevant to β_1.

```r
summary(stop_dist_model)$coefficients[2,]
```

```
##     Estimate   Std. Error      t value     Pr(>|t|)
## 3.932409e+00 4.155128e-01 9.463990e+00 1.489836e-12
```

Again, the first value, `Estimate` is

$$\hat{\beta}_1 = 3.9324088.$$

The second value, `Std. Error`, is the standard error of $\hat{\beta}_1$,

$$SE[\hat{\beta}_1] = \frac{s_e}{\sqrt{S_{xx}}} = 0.4155128.$$

The third value, `t value`, is the value of the test statistics for testing $H_0 : \beta_1 = 0$ vs $H_1 : \beta_1 \neq 0$,

$$t = \frac{\hat{\beta}_1 - 0}{SE[\hat{\beta}_1]} = \frac{\hat{\beta}_1 - 0}{s_e / \sqrt{S_{xx}}} = 9.46399.$$

Lastly, `Pr(>|t|)`, gives us the p-value of that test.

$$\text{p-value} = 1.4898365 \times 10^{-12}$$

Note here, we are specifically testing whether or not $\beta_1 = 0$.

The first row of output reports the same values, but for β_0.

```
summary(stop_dist_model)$coefficients[1,]
```

```
##     Estimate   Std. Error      t value     Pr(>|t|)
## -17.57909489   6.75844017  -2.60105800   0.01231882
```

In summary, the following code stores the information of `summary(stop_dist_model)$coefficients` in a new variable `stop_dist_model_test_info`, then extracts each element into a new variable which describes the information it contains.

```
stop_dist_model_test_info = summary(stop_dist_model)$coefficients

beta_0_hat      = stop_dist_model_test_info[1,1] # Estimate
beta_0_hat_se   = stop_dist_model_test_info[1,2] # Std. Error
beta_0_hat_t    = stop_dist_model_test_info[1,3] # t value
beta_0_hat_pval = stop_dist_model_test_info[1,4] # Pr(>|t|)

beta_1_hat      = stop_dist_model_test_info[2,1] # Estimate
beta_1_hat_se   = stop_dist_model_test_info[2,2] # Std. Error
beta_1_hat_t    = stop_dist_model_test_info[2,3] # t value
beta_1_hat_pval = stop_dist_model_test_info[2,4] # Pr(>|t|)
```

We can then verify some equivalent expressions: the t test statistic for $\hat{\beta}_1$ and the two-sided p-value associated with that test statistic.

```
(beta_1_hat - 0) / beta_1_hat_se
```

```
## [1] 9.46399
```

```
beta_1_hat_t
```

```
## [1] 9.46399
```

```
2 * pt(abs(beta_1_hat_t), df = length(resid(stop_dist_model)) - 2, lower.tail = FALSE)
```

```
## [1] 1.489836e-12
```

```
beta_1_hat_pval
```

```
## [1] 1.489836e-12
```

4.6.2 Significance of Regression, t-Test

We pause to discuss the **significance of regression** test. First, note that based on the above distributional results, we could test β_0 and β_1 against any particular value, and perform both one and two-sided tests.

However, one very specific test,

$$H_0 : \beta_1 = 0 \quad \text{vs} \quad H_1 : \beta_1 \neq 0$$

is used most often. Let's think about this test in terms of the simple linear regression model,

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i.$$

If we assume the null hypothesis is true, then $\beta_1 = 0$ and we have the model,

$$Y_i = \beta_0 + \epsilon_i.$$

In this model, the response does **not** depend on the predictor. So then we could think of this test in the following way,

- Under H_0 there is not a significant linear relationship between x and y.
- Under H_1 there is a significance **linear** relationship between x and y.

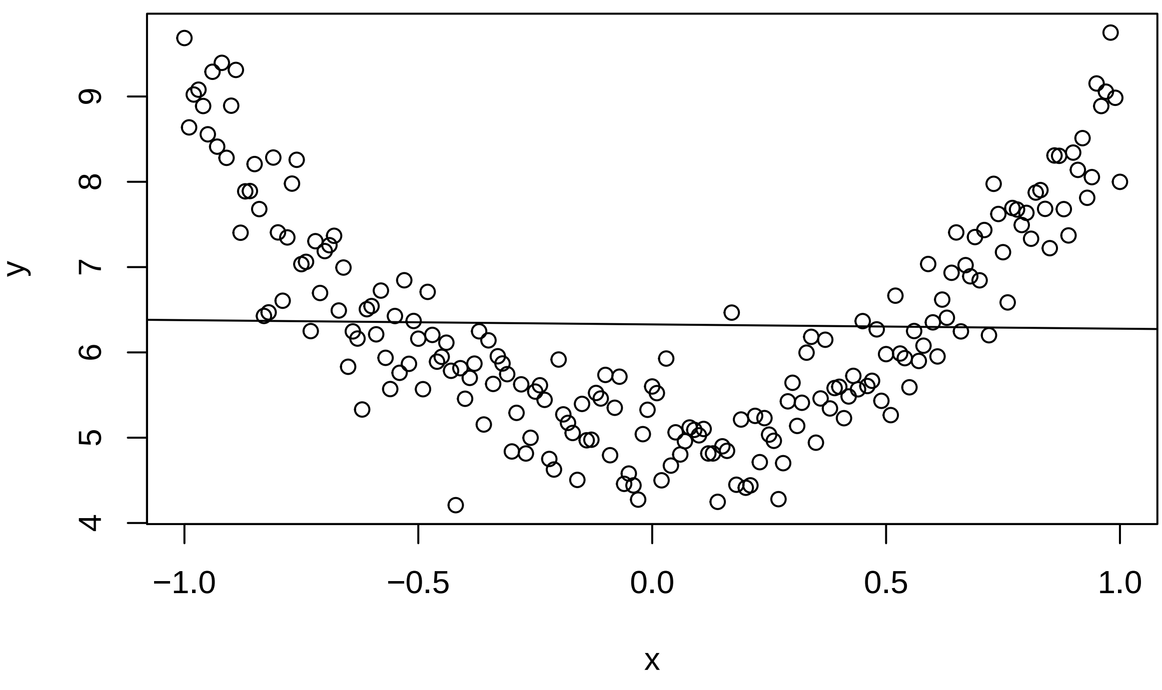
For the `cars` example,

- Under H_0 there is not a significant linear relationship between speed and stopping distance.
- Under H_1 there is a significant **linear** relationship between speed and stopping distance.

Again, that test is seen in the output from `summary()`,

$$\text{p-value} = 1.4898365 \times 10^{-12}.$$

With this extremely low p-value, we would reject the null hypothesis at any reasonable α level, say for example $\alpha = 0.01$. So we say there is a significant **linear** relationship between speed and stopping distance. Notice that we emphasize **linear**.



In this plot of simulated data, we see a clear relationship between x and y, however it is not a linear relationship. If we fit a line to this data, it is very flat. The resulting test for $H_0 : \beta_1 = 0$ vs $H_1 : \beta_1 \neq 0$ gives a large p-value, in this case 0.7564548, so we would fail to reject and say that there is no significant linear relationship between x and y. We will see later how to fit a curve to this data using a "linear" model, but for now, realize that testing $H_0 : \beta_1 = 0$ vs $H_1 : \beta_1 \neq 0$ can only detect straight line relationships.

4.6.3 Confidence Intervals in R

Using R we can very easily obtain the confidence intervals for β_0 and β_1.

```
confint(stop_dist_model, level = 0.99)
```

```
##                  0.5 %    99.5 %
## (Intercept) -35.706610 0.5484205
## speed         2.817919 5.0468988
```

This automatically calculates 99% confidence intervals for both β_0 and β_1, the first row for β_0, the second row for β_1.

For the `cars` example when interpreting these intervals, we say, we are 99% confident that for an increase in speed of 1 mile per hour, the average increase in stopping distance is between 2.8179187 and 5.0468988 feet, which is the interval for β_1.

Note that this 99% confidence interval does **not** contain the hypothesized value of 0. Since it does not contain 0, it is equivalent to rejecting the test of $H_0 : \beta_1 = 0$ vs $H_1 : \beta_1 \neq 0$ at $\alpha = 0.01$, which we had seen previously.

You should be somewhat suspicious of the confidence interval for β_0, as it covers negative values, which correspond to negative stopping distances. Technically the interpretation would be that we are 99% confident that the average stopping distance of a car traveling 0 miles per hour is between -35.7066103 and 0.5484205 feet, but we don't really believe that, since we are actually certain that it would be non-negative.

Note, we can extract specific values from this output a number of ways. This code is not run, and instead, you should check how it relates to the output of the code above.

```
confint(stop_dist_model, level = 0.99)[1,]
confint(stop_dist_model, level = 0.99)[1,1]
confint(stop_dist_model, level = 0.99)[1,2]
confint(stop_dist_model, parm = "(Intercept)", level = 0.99)
confint(stop_dist_model, level = 0.99)[2,]
confint(stop_dist_model, level = 0.99)[2,1]
confint(stop_dist_model, level = 0.99)[2,2]
confint(stop_dist_model, parm = "speed", level = 0.99)
```

4.7 Confidence Interval for Mean Response

In addition to confidence intervals for β_0 and β_1 there two other common interval estimates used with regression. The first is called a **confidence interval for the mean response**. Often, we would like an interval estimate for the mean, $E[Y]$ for a particular value of x.

In this situation we use $\hat{y}$ as our estimate of $E[Y]$.

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

Recall that,

$$E[Y] = \beta_0 + \beta_1 x$$

Thus, $\hat{y}$ is a good estimate since it is unbiased:

$$E[\hat{y}] = \beta_0 + \beta_1 x.$$

We could then derive,

$$Var[\hat{y}] = \sigma^2 \left(\frac{1}{n} + \frac{(x - \bar{x})^2}{S_{xx}} \right).$$

Like the other estimates we have seen, $\hat{y}$ also follows a normal distribution,

$$\hat{y} \sim N\left(\beta_0 + \beta_1 x, \sigma^2 \left(\frac{1}{n} + \frac{(x - \bar{x})^2}{S_{xx}} \right) \right).$$

And lastly, since we need to estimate this variance, we arrive at the standard error of our estimate,

$$SE[\hat{y}] = s_e \sqrt{\frac{1}{n} + \frac{(x - \bar{x})^2}{S_{xx}}}.$$

We can then use this to find the confidence interval for the mean response,

$$\hat{y} \pm t_{\alpha/2, n-2} \cdot s_e \sqrt{\frac{1}{n} + \frac{(x - \bar{x})^2}{S_{xx}}}$$

To find confidence intervals for the mean response using `R`, we use the `predict()` function. We give the function our fitted model as well as new data, stored as a data frame. (This is important, so that `R` knows the name of the predictor variable.) Here, we are finding the confidence interval for the mean stopping distance when a car is travelling 5 miles per hour and when a car is travelling 21 miles per hour.

```
new_speeds = data.frame(speed = c(5, 21))
predict(stop_dist_model, newdata = new_speeds,
        interval = c("confidence"), level = 0.99)
```

```
##         fit       lwr      upr
## 1  2.082949 -10.89309 15.05898
## 2 65.001489  56.45836 73.54462
```

4.8 Prediction Interval for New Observations

Sometimes we would like an interval estimate for a new observation, Y for a particular value of x. This is very similar to an interval for the mean response, $E[Y]$, but different in one very important way.

Our best guess for a new observation is still $\hat{y}$. The estimated mean is still the best prediction we can make. The difference is in the amount of variability, since we know that observations will vary about the true regression line according to a $N(0, \sigma^2)$ distribution. Because of this we add an extra factor of σ^2 to our estimates variability.

$$Var[\hat{y}] = \sigma^2 \left(1 + \frac{1}{n} + \frac{(x - \bar{x})^2}{S_{xx}} \right)$$

$$\hat{y} \sim N\left(\beta_0 + \beta_1 x, \sigma^2 \left(1 + \frac{1}{n} + \frac{(x - \bar{x})^2}{S_{xx}} \right) \right)$$

$$SE[\hat{y}] = s_e \sqrt{1 + \frac{1}{n} + \frac{(x - \bar{x})^2}{S_{xx}}}$$

We can then find a **prediction interval** using,

$$\hat{y} \pm t_{\alpha/2, n-2} \cdot s_e \sqrt{1 + \frac{1}{n} + \frac{(x - \bar{x})^2}{S_{xx}}}.$$

To calculate this for a set of points in `R` notice there is only a minor change in syntax from finding a confidence interval for the mean response.

```
predict(stop_dist_model, newdata = new_speeds,
        interval = c("prediction"), level = 0.99)
```

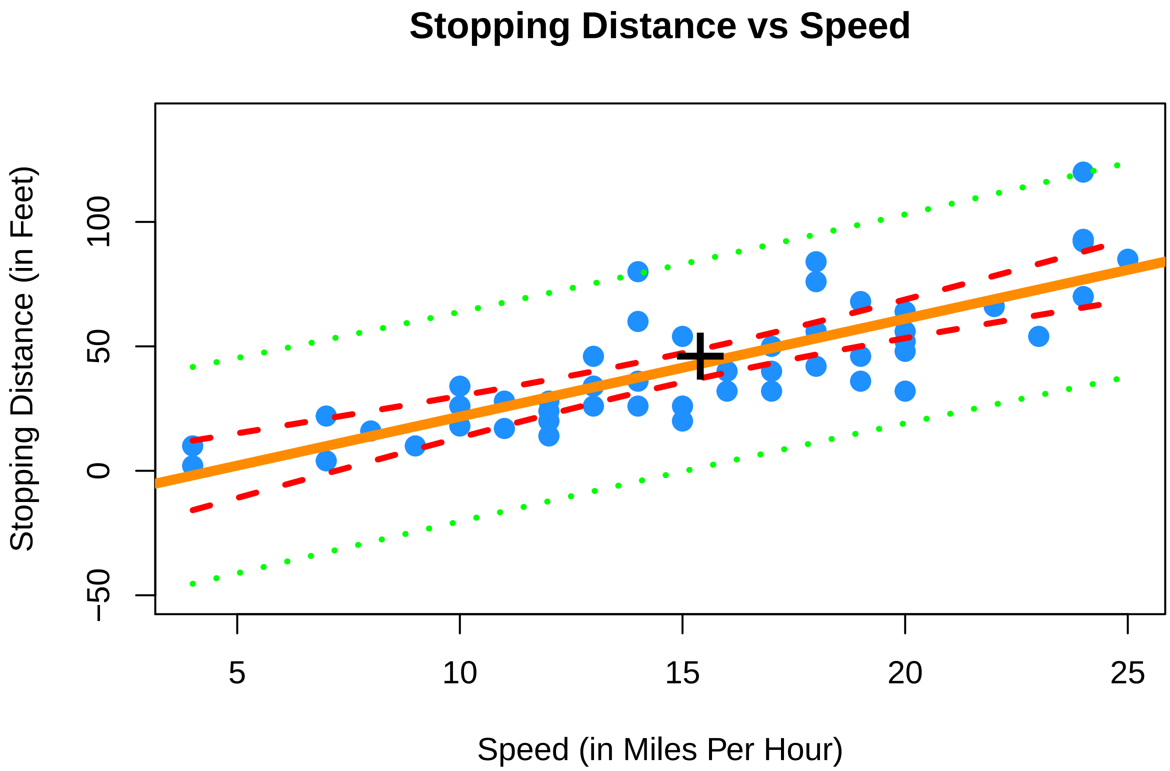
```
##         fit       lwr       upr
## 1  2.082949 -41.16099  45.32689
## 2 65.001489  22.87494 107.12803
```

Also notice that these two intervals are wider than the corresponding confidence intervals for the mean response.

4.9 Confidence and Prediction Bands

Often we will like to plot both confidence intervals for the mean response and prediction intervals for all possible values of x. We calls these confidence and prediction bands.

```
speed_grid = seq(min(cars$speed), max(cars$speed), by = 0.01)
dist_ci_band = predict(stop_dist_model,
                       newdata = data.frame(speed = speed_grid),
                       interval = "confidence", level = 0.99)
dist_pi_band = predict(stop_dist_model,
                       newdata = data.frame(speed = speed_grid),
                       interval = "prediction", level = 0.99)

plot(dist ~ speed, data = cars,
     xlab = "Speed (in Miles Per Hour)",
     ylab = "Stopping Distance (in Feet)",
     main = "Stopping Distance vs Speed",
     pch  = 20,
     cex  = 2,
     col  = "dodgerblue",
     ylim = c(-50, 140))
abline(stop_dist_model, lwd = 5, col = "darkorange")

lines(speed_grid, dist_ci_band[,"lwr"], col = "red", lwd = 3, lty = 2)
lines(speed_grid, dist_ci_band[,"upr"], col = "red", lwd = 3, lty = 2)
lines(speed_grid, dist_pi_band[,"lwr"], col = "green", lwd = 3, lty = 3)
lines(speed_grid, dist_pi_band[,"upr"], col = "green", lwd = 3, lty = 3)
points(mean(cars$speed), mean(cars$dist), pch = "+", cex = 3)
```



Some things to notice:

- We use the `ylim` argument to stretch the y-axis of the plot, since the bands extend further than the points.
- We add a point at the point $(\bar{x}, \bar{y})$.
  - This is a point that the regression line will **always** pass through. (Think about why.)
  - This is the point where both the confidence and prediction bands are the narrowest. Look at the standard errors of both to understand why.
- The prediction bands (green) are less curved than the confidence bands (red). This is a result of the extra factor of σ^2 added to the variance at any value of x.

4.10 Significance of Regression, F-Test

In the case of simple linear regression, the t test for the significance of the regression is equivalent to another test, the F test for the significance of the regression. This equivalence will only be true for simple linear regression, and in the next chapter we will only use the F test for the significance of the regression.

Recall from last chapter the decomposition of variance we saw before calculating R^2,

$$\sum_{i=1}^{n}(y_i - \bar{y})^2 = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2,$$

or, in short,

$$SST = SSReg + SSE.$$

To develop the F test, we will arrange this information in an **ANOVA** table,

Source	Sum of Squares	Degrees of Freedom	Mean Square	F
Regression	$\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$	1	$SSReg/1$	$MSReg/MSE$
Error	$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	$n-2$	$SSE/(n-2)$	
Total	$\sum_{i=1}^{n}(y_i - \bar{y})^2$	$n-1$		

ANOVA, or Analysis of Variance will be a concept we return to often in this course. For now, we will focus on the results of the table, which is the F statistic,

$$F = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2/1}{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2/(n-2)} \sim F_{1,n-2}$$

which follows an F distribution with degrees of freedom 1 and $n-2$ under the null hypothesis. An F distribution is a continuous distribution which takes only positive values and has two parameters, which are the two degrees of freedom.

Recall, in the significance of the regression test, Y does **not** depend on x in the null hypothesis.

$$H_0: \beta_1 = 0 \qquad Y_i = \beta_0 + \epsilon_i$$

While in the alternative hypothesis Y may depend on x.

$$H_1: \beta_1 \neq 0 \qquad Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

We can use the F statistic to perform this test.

$$F = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2/1}{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2/(n-2)}$$

In particular, we will reject the null when the F statistic is large, that is, when there is a low probability that the observations could have come from the null model by chance. We will let `R` calculate the p-value for us.

To perform the F test in `R` you can look at the last row of the output from `summary()` called `F-statistic` which gives the value of the test statistic, the relevant degrees of freedom, as well as the p-value of the test.

```
summary(stop_dist_model)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

Additionally, you can use the `anova()` function to display the information in an ANOVA table.

```r
anova(stop_dist_model)
```

```
## Analysis of Variance Table
##
## Response: dist
##           Df Sum Sq Mean Sq F value   Pr(>F)
## speed      1  21186 21185.5  89.567 1.49e-12 ***
## Residuals 48  11354   236.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This also gives a p-value for the test. You should notice that the p-value from the t test was the same. You might also notice that the value of the test statistic for the t test, 9.46399, can be squared to obtain the value of the F statistic, 89.5671065.

Note that there is another equivalent way to do this in `R`, which we will return to often to compare two models.

```r
anova(lm(dist ~ 1, data = cars), lm(dist ~ speed, data = cars))
```

```
## Analysis of Variance Table
##
## Model 1: dist ~ 1
## Model 2: dist ~ speed
##   Res.Df   RSS Df Sum of Sq      F   Pr(>F)
## 1     49 32539
## 2     48 11354  1     21186 89.567 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The model statement `lm(dist ~ 1, data = cars)` applies the model $Y_i = \beta_0 + \epsilon_i$ to the cars data. Note that $\hat{y} = \bar{y}$ when $Y_i = \beta_0 + \epsilon_i$.

The model statement `lm(dist ~ speed, data = cars)` applies the model $Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$.

We can then think of this usage of `anova()` as directly comparing the two models. (Notice we get the same p-value again.)

Chapter 5

Multiple Linear Regression

> "Life is really simple, but we insist on making it complicated."
>
> — **Confucius**

After reading this chapter you will be able to:

- Construct and interpret linear regression models with more than one predictor.
- Understand how regression models are derived using matrices.
- Create interval estimates and perform hypothesis tests for multiple regression parameters.
- Formulate and interpret interval estimates for the mean response under various conditions.
- Compare nested models using an ANOVA F-Test.

The last two chapters we saw how to fit a model that assumed a linear relationship between a response variable and a single predictor variable. Specifically, we defined the simple linear regression model,

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$.

However, it is rarely the case that a dataset will have a single predictor variable. It is also rarely the case that a response variable will only depend on a single variable. So in this chapter, we will extend our current linear model to allow a response to depend on *multiple* predictors.

```
# read the data from the web
autompg = read.table(
  "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data",
  quote = "\"",
  comment.char = "",
  stringsAsFactors = FALSE)
# give the dataframe headers
colnames(autompg) = c("mpg", "cyl", "disp", "hp", "wt", "acc", "year", "origin", "name")
# remove missing data, which is stored as "?"
autompg = subset(autompg, autompg$hp != "?")
# remove the plymouth reliant, as it causes some issues
autompg = subset(autompg, autompg$name != "plymouth reliant")
# give the dataset row names, based on the engine, year and name
rownames(autompg) = paste(autompg$cyl, "cylinder", autompg$year, autompg$name)
```

```
# remove the variable for name, as will as origin
autompg = subset(autompg, select = c("mpg", "cyl", "disp", "hp", "wt", "acc", "year"))
# change horsepower from character to numeric
autompg$hp = as.numeric(autompg$hp)
str(autompg)
```

```
## 'data.frame':    390 obs. of  7 variables:
##  $ mpg : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cyl : int  8 8 8 8 8 8 8 8 8 8 ...
##  $ disp: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ hp  : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ wt  : num  3504 3693 3436 3433 3449 ...
##  $ acc : num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year: int  70 70 70 70 70 70 70 70 70 70 ...
```

We will once again discuss a dataset with information about cars. This dataset, which can be found at the UCI Machine Learning Repository contains a response variable `mpg` which stores the city fuel efficiency of cars, as well as several predictor variables for the attributes of the vehicles. We load the data, and perform some basic tidying before moving on to analysis.

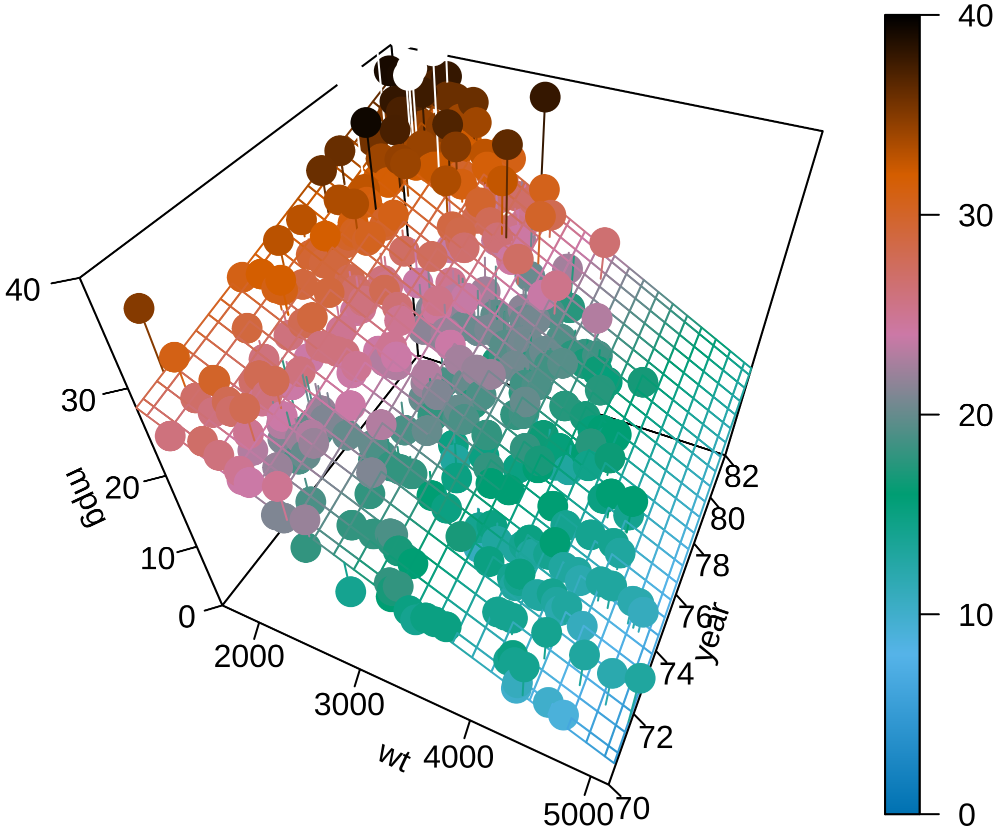
For now we will focus on using two variables, `wt` and `year`, as predictor variables. That is, we would like to model the fuel efficiency (`mpg`) of a car as a function of its weight (`wt`) and model year (`year`). To do so, we will define the following linear model,

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \epsilon_i, \qquad i = 1, 2, \ldots, n$$

where $\epsilon_i \sim N(0, \sigma^2)$. In this notation we will define:

- x_{i1} as the weight (`wt`) of the ith car.
- x_{i2} as the model year (`year`) of the ith car.

The picture below will visualize what we would like to accomplish. The data points (x_{i1}, x_{i2}, y_i) now exist in 3-dimensional space, so instead of fitting a line to the data, we will fit a plane. (We'll soon move to higher dimensions, so this will be the last example that is easy to visualize and think about this way.)



How do we find such a plane? Well, we would like a plane that is as close as possible to the data points. That is, we would like it to minimize the errors it is making. How will we define these errors? Squared distance of course! So, we would like to minimize

$$f(\beta_0, \beta_1, \beta_2) = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}))^2$$

with respect to β_0, β_1, and β_2. How do we do so? It is another straightforward multivariate calculus problem. All we have done is add an extra variable since we did this last time. So again, we take a derivative with respect to each of β_0, β_1, and β_2 and set them equal to zero, then solve the resulting system of equations. That is,

$$\frac{\partial f}{\partial \beta_0} = 0$$
$$\frac{\partial f}{\partial \beta_1} = 0$$
$$\frac{\partial f}{\partial \beta_2} = 0$$

After doing so, we will once again obtain the **normal equations.**

$$
\begin{aligned}
n\beta_0 + \beta_1 \sum_{i=1}^{n} x_{i1} + \beta_2 \sum_{i=1}^{n} x_{i2} &= \sum_{i=1}^{n} y_i \\
\beta_0 \sum_{i=1}^{n} x_{i1} + \beta_1 \sum_{i=1}^{n} x_{i1}^2 + \beta_2 \sum_{i=1}^{n} x_{i1}x_{i2} &= \sum_{i=1}^{n} x_{i1}y_i \\
\beta_0 \sum_{i=1}^{n} x_{i2} + \beta_1 \sum_{i=1}^{n} x_{i1}x_{i2} + \beta_2 \sum_{i=1}^{n} x_{i2}^2 &= \sum_{i=1}^{n} x_{i2}y_i
\end{aligned}
$$

We now have three equations and three variables, which we could solve, or we could simply let `R` solve for us.

```
mpg_model = lm(mpg ~ wt + year, data = autompg)
coef(mpg_model)
```

```
##   (Intercept)            wt          year
## -14.637641945  -0.006634876   0.761401955
```

$$
\hat{y} = -14.6376419 + -0.0066349 x_1 + 0.761402 x_2
$$

Here we have once again fit our model using `lm()`, however we have introduced a new syntactical element. The formula `mpg ~ wt + year` now reads: "model the response variable `mpg` as a linear function of `wt` and `year`". That is, it will estimate an intercept, as well as slope coefficients for `wt` and `year`. We then extract these as we have done before using `coef()`.

In the multiple linear regression setting, some of the interpretations of the coefficients change slightly.

Here, $\hat{\beta}_0 = -14.6376419$ is our estimate for β_0, the mean miles per gallon for a car that weighs 0 pounds and was built in 1900. We see our estimate here is negative, which is a physical impossibility. However, this isn't unexpected, as we shouldn't expect our model to be accurate for cars from 1900 which weigh 0 pounds. (Because they never existed!) This isn't much of a change from SLR. That is, β_0 is still simply the mean when all of the predictors are 0.

The interpretation of the coefficients in front of our predictors are slightly different than before. For example $\hat{\beta}_1 = -0.0066349$ is our estimate for β_1, the average change in miles per gallon for an increase in weight (x_1) of one-pound **for a car of a certain model year**, that is, for a fixed value of x_2. Note that this coefficient is actually the same for any given value of x_2. Later, we will look at models that allow for a different change in mean response for different values of x_2. Also note that this estimate is negative, which we would expect since, in general, fuel efficiency decreases for larger vehicles. Recall that in the multiple linear regression setting, this interpretation is dependent on a fixed value for x_2, that is, "for a car of a certain model year." It is possible that the indirect relationship between fuel efficiency and weight does not hold when an additional factor, say year, is included, and thus we could have the sign of our coefficient flipped.

Lastly, $\hat{\beta}_2 = 0.761402$ is our estimate for β_2, the average change in miles per gallon for a one-year increase in model year (x_2) for a car of a certain weight, that is, for a fixed value of x_1. It is not surprising that the estimate is positive. We expect that as time passes and the years march on, technology would improve so that a car of a specific weight would get better mileage now as compared to their predecessors. And yet, the coefficient could have been negative because we are also including weight as variable, and not strictly as a fixed value.

5.1 Matrix Approach to Regression

In our above example we used two predictor variables, but it will only take a little more work to allow for an arbitrary number of predictor variables and derive their coefficient estimates. We can consider the model,

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_{p-1} x_{i(p-1)} + \epsilon_i, \qquad i = 1, 2, \ldots, n$$

where $\epsilon_i \sim N(0, \sigma^2)$. In this model, there are $p-1$ predictor variables, $x_1, x_2, \cdots, x_{p-1}$. There are a total of p β-parameters and a single parameter σ^2 for the variance of the errors. (It should be noted that almost as often, authors will use p as the number of predictors, making the total number of β parameters $p+1$. This is always something you should be aware of when reading about multiple regression. There is not a standard that is used most often.)

If we were to stack together the n linear equations that represent each Y_i into a column vector, we get the following.

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1(p-1)} \\ 1 & x_{21} & x_{22} & \cdots & x_{2(p-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{n(p-1)} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{p-1} \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

$$Y = X\beta + \epsilon$$

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1(p-1)} \\ 1 & x_{21} & x_{22} & \cdots & x_{2(p-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{n(p-1)} \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{p-1} \end{bmatrix}, \quad \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

So now with data,

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Just as before, we can estimate β by minimizing,

$$f(\beta_0, \beta_1, \beta_2, \cdots, \beta_{p-1}) = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_{p-1} x_{i(p-1)}))^2,$$

which would require taking p derivatives, which result in following **normal equations**.

$$\begin{bmatrix} n & \sum_{i=1}^n x_{i1} & \sum_{i=1}^n x_{i2} & \cdots & \sum_{i=1}^n x_{i(p-1)} \\ \sum_{i=1}^n x_{i1} & \sum_{i=1}^n x_{i1}^2 & \sum_{i=1}^n x_{i1}x_{i2} & \cdots & \sum_{i=1}^n x_{i1}x_{i(p-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ \sum_{i=1}^n x_{i(p-1)} & \sum_{i=1}^n x_{i(p-1)}x_{i1} & \sum_{i=1}^n x_{i(p-1)}x_{i2} & \cdots & \sum_{i=1}^n x_{i(p-1)}^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{p-1} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_{i1}y_i \\ \vdots \\ \sum_{i=1}^n x_{i(p-1)}y_i \end{bmatrix}$$

The normal equations can be written much more succinctly in matrix notation,

$$X^\top X\beta = X^\top y.$$

We can then solve this expression by multiplying both sides by the inverse of $X^\top X$, which exists, provided the columns of X are linearly independent. Then as always, we denote our solution with a hat.

$$\hat{\beta} = \left(X^\top X\right)^{-1} X^\top y$$

To verify that this is what `R` has done for us in the case of two predictors, we create an X matrix. Note that the first column is all 1s, and the remaining columns contain the data.

```
n = nrow(autompg)
p = length(coef(mpg_model))
X = cbind(rep(1, n), automgp$wt, autompg$year)
y = autompg$mpg

(beta_hat = solve(t(X) %*% X) %*% t(X) %*% y)
```

```
##                [,1]
## [1,] -14.637641945
## [2,]  -0.006634876
## [3,]   0.761401955
```

```
coef(mpg_model)
```

```
##   (Intercept)            wt          year 
## -14.637641945  -0.006634876   0.761401955
```

$$\hat{\beta} = \begin{bmatrix} -14.6376419 \\ -0.0066349 \\ 0.761402 \end{bmatrix}$$

In our new notation, the fitted values can be written

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

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix}$$

Then, we can create a vector for the residual values,

$$e = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix}.$$

And lastly, we can update our estimate for σ^2.

$$s_e^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n - p} = \frac{e^\top e}{n - p}$$

Recall, we like this estimate because it is unbiased, that is,

$$E[s_e^2] = \sigma^2$$

Note that the change from the SLR estimate to now is in the denominator. Specifically we now divide by $n - p$ instead of $n - 2$. Or actually, we should note that in the case of SLR, there are two β parameters and thus $p = 2$.

Also note that if we fit the model $Y_i = \beta + \epsilon_i$ that $\hat{y} = \bar{y}$ and $p = 1$ and s_e^2 would become

$$s_e^2 = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{n - 1}$$

which is likely the very first sample standard deviation you saw in a mathematical statistics class. The same reason for $n - 1$ in this case, that we estimated one parameter, so we lose one degree of freedom. Now, in general, we are estimating p parameters, the β parameters, so we lose p degrees of freedom.

Also, recall that most often we will be interested in s_e, the residual standard error as `R` calls it,

$$s_e = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n - p}}.$$

In `R`, we could directly access s_e for a fitted model, as we have seen before.

```
summary(mpg_model)$sigma
```

```
## [1] 3.431367
```

And we can now verify that our math above is indeed calculating the same quantities.

```
y_hat = X %*% solve(t(X) %*% X) %*% t(X) %*% y
e     = y - y_hat
sqrt(t(e) %*% e / (n - p))
```

```
##          [,1]
## [1,] 3.431367
```

```
sqrt(sum((y - y_hat) ^ 2) / (n - p))
```

```
## [1] 3.431367
```

5.2 Sampling Distribution

As we can see in the output below, the results of calling `summary()` are similar to SLR, but there are some differences, most obviously a new row for the added predictor variable.

```r
summary(mpg_model)
```

```
##
## Call:
## lm(formula = mpg ~ wt + year, data = autompg)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -8.852 -2.292 -0.100  2.039 14.325
##
## Coefficients:
##                Estimate  Std. Error t value Pr(>|t|)
## (Intercept) -14.6376419   4.0233914  -3.638 0.000312 ***
## wt           -0.0066349   0.0002149 -30.881  < 2e-16 ***
## year          0.7614020   0.0497266  15.312  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.431 on 387 degrees of freedom
## Multiple R-squared:  0.8082, Adjusted R-squared:  0.8072
## F-statistic: 815.6 on 2 and 387 DF,  p-value: < 2.2e-16
```

To understand these differences in detail, we will need to first obtain the sampling distribution of $\hat{\beta}$.

The derivation of the sampling distribution of $\hat{\beta}$ involves the multivariate normal distribution. These brief notes from semesters past give a basic overview. These are simply for your information, as we will not present the derivation in full here.

Our goal now is to obtain the distribution of the $\hat{\beta}$ vector,

$$\hat{\beta} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \hat{\beta}_2 \\ \vdots \\ \hat{\beta}_{p-1} \end{bmatrix}$$

Recall from last time that when discussing sampling distributions, we now consider $\hat{\beta}$ to be a random vector, thus we use Y instead of the data vector y.

$$\hat{\beta} = \left(X^\top X\right)^{-1} X^\top Y$$

Then it is a consequence of the multivariate normal distribution that,

$$\hat{\beta} \sim N\left(\beta, \sigma^2 \left(X^\top X\right)^{-1}\right).$$

We then have

$$E[\hat{\beta}] = \beta$$

and for any $\hat{\beta}_j$ we have

$$E[\hat{\beta}_j] = \beta_j.$$

We also have

$$Var[\hat{\beta}] = \sigma^2 \left(X^\top X\right)^{-1}$$

and for any $\hat{\beta}_j$ we have

$$Var[\hat{\beta}_j] = \sigma^2 C_{jj}$$

where

$$C = \left(X^\top X\right)^{-1}$$

and the elements of C are denoted

$$C = \begin{bmatrix} C_{00} & C_{01} & C_{02} & \cdots & C_{0(p-1)} \\ C_{10} & C_{11} & C_{12} & \cdots & C_{1(p-1)} \\ C_{20} & C_{21} & C_{22} & \cdots & C_{2(p-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{(p-1)0} & C_{(p-1)1} & C_{(p-1)2} & \cdots & C_{(p-1)(p-1)} \end{bmatrix}.$$

Essentially, the diagonal elements correspond to the β vector.

Then the standard error for the $\hat{\beta}$ vector is given by

$$SE[\hat{\beta}] = s_e \sqrt{\left(X^\top X\right)^{-1}}$$

and for a particular $\hat{\beta}_j$

$$SE[\hat{\beta}_j] = s_e \sqrt{C_{jj}}.$$

Lastly, each of the $\hat{\beta}_j$ follows a normal distribution,

$$\hat{\beta}_j \sim N\left(\beta_j, \sigma^2 C_{jj}\right).$$

thus

$$\frac{\hat{\beta}_j - \beta_j}{s_e \sqrt{C_{jj}}} \sim t_{n-p}.$$

Now that we have the necessary distributional results, we can move on to perform tests and make interval estimates.

5.2.1 Single Parameter Tests

The first test we will see is a test for a single β_j.

$$H_0 : \beta_j = 0 \quad \text{vs} \quad H_1 : \beta_j \neq 0$$

Again, the test statistic takes the form

$$TS = \frac{EST - HYP}{SE}.$$

In particular,

$$t = \frac{\hat{\beta}_j - \beta_j}{SE[\hat{\beta}_j]} = \frac{\hat{\beta}_j - 0}{s_e \sqrt{C_{jj}}},$$

which, under the null hypothesis, follows a t distribution with $n - p$ degrees of freedom.

Recall our model for `mpg`,

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \epsilon_i, \qquad i = 1, 2, \ldots, n$$

where $\epsilon_i \sim N(0, \sigma^2)$.

- x_{i1} as the weight (`wt`) of the ith car.
- x_{i2} as the model year (`year`) of the ith car.

Then the test

$$H_0 : \beta_1 = 0 \quad \text{vs} \quad H_1 : \beta_1 \neq 0$$

can be found in the `summary()` output, in particular:

```r
summary(mpg_model)$coef
```

```
##                 Estimate   Std. Error    t value      Pr(>|t|)
## (Intercept) -14.637641945 4.0233913563  -3.638135  3.118311e-04
## wt           -0.006634876 0.0002148504 -30.881372 1.850466e-106
## year          0.761401955 0.0497265950  15.311765  1.036597e-41
```

The estimate (`Estimate`), standard error (`Std. Error`), test statistic (`t value`), and p-value (`Pr(>|t|)`) for this test are displayed in the second row, labeled `wt`. Remember that the p-value given here is specifically for a two-sided test, where the hypothesized value is 0.

Also note in this case, by hypothesizing that $\beta_1 = 0$ the null and alternative essentially specify two different models:

- H_0: $Y = \beta_0 + \beta_2 x_2 + \epsilon$
- H_1: $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$

This is important. We are not simply testing whether or not there is a relationship between weight and fuel efficiency. We are testing if there is a relationship between weight and fuel efficiency, given that a term for year is in the model. (Note, we dropped some indexing here, for readability.)

5.2.2 Confidence Intervals

Since $\hat{\beta}_j$ is our estimate for β_j and we have

$$E[\hat{\beta}_j] = \beta_j$$

as well as the standard error,

$$SE[\hat{\beta}_j] = s_e \sqrt{C_{jj}}$$

and the sampling distribution of $\hat{\beta}_j$ is Normal, then we can easily construct confidence intervals for each of the $\hat{\beta}_j$.

$$\hat{\beta}_j \pm t_{\alpha/2, n-p} \cdot s_e \sqrt{C_{jj}}$$

We can find these in `R` using the same method as before. Now there will simply be additional rows for the additional β.

```
confint(mpg_model, level = 0.99)
```

```
##                     0.5 %       99.5 %
## (Intercept) -25.052563681 -4.222720208
## wt           -0.007191036 -0.006078716
## year          0.632680051  0.890123859
```

5.2.3 Confidence Intervals for Mean Response

As we saw in SLR, we can create confidence intervals for mean response, that is, an interval estimate for $E[Y]$. In SLR, the mean of Y was only dependent on a single value x. Now, in multiple regression, $E[Y]$ is dependent on the value of each of the predictors, so we define the vector x_0 to be,

$$x_0 = \begin{bmatrix} 1 \\ x_{01} \\ x_{02} \\ \vdots \\ x_{0(p-1)} \end{bmatrix}.$$

Then our estimate of $E[Y]$ for a set of values x_0 is given by

$$\begin{aligned} \hat{y} &= x_0^\top \hat{\beta} \\ &= \hat{\beta}_0 + \hat{\beta}_1 x_{01} + \hat{\beta}_2 x_{02} + \cdots + \hat{\beta}_{p-1} x_{0(p-1)}. \end{aligned}$$

As with SLR, this is an unbiased estimate.

$$\begin{aligned} E[\hat{y}] &= x_0^\top \beta \\ &= \beta_0 + \beta_1 x_{01} + \beta_2 x_{02} + \cdots + \beta_{p-1} x_{0(p-1)} \end{aligned}$$

To make an interval estimate, we will also need its standard error.

$$
SE[\hat{y}] = s_e \sqrt{x_0^{\top} \left(X^{\top} X\right)^{-1} x_0}
$$

Putting it all together, we obtain a confidence interval for the mean response.

$$
\hat{y} \pm t_{\alpha/2,n-p} \cdot s_e \sqrt{x_0^{\top} \left(X^{\top} X\right)^{-1} x_0}
$$

The math has changed a bit, but the process in R remains almost identical. Here, we create a data frame for two additional cars. One car that weighs 3500 pounds produced in 1976, as well as a second car that weighs 5000 pounds which was produced in 1981.

```
new_cars = data.frame(wt = c(3500, 5000), year = c(76, 81))
new_cars
```

```
##     wt year
## 1 3500   76
## 2 5000   81
```

We can then use the `predict()` function with `interval = "confidence"` to obtain intervals for the mean fuel efficiency for both new car. Again, it is important to make the data passed to `newdata` a data frame, so that R knows which values are for which variables.

```
predict(mpg_model, newdata = new_cars, interval = "confidence", level = 0.99)
```

```
##        fit     lwr      upr
## 1 20.00684 19.4712 20.54248
## 2 13.86154 12.3341 15.38898
```

R then reports the estimate $\hat{y}$ (`fit`) for each, as well as the lower (`lwr`) and upper (`upr`) bounds for the interval at a desired level (99%).

A word of caution here: one of these estimates is good while one is suspect.

```
new_cars$wt
```

```
## [1] 3500 5000
```

```
range(autompg$wt)
```

```
## [1] 1613 5140
```

Note that both of the weights of the new cars are within the range of observed values.

```
new_cars$year
```

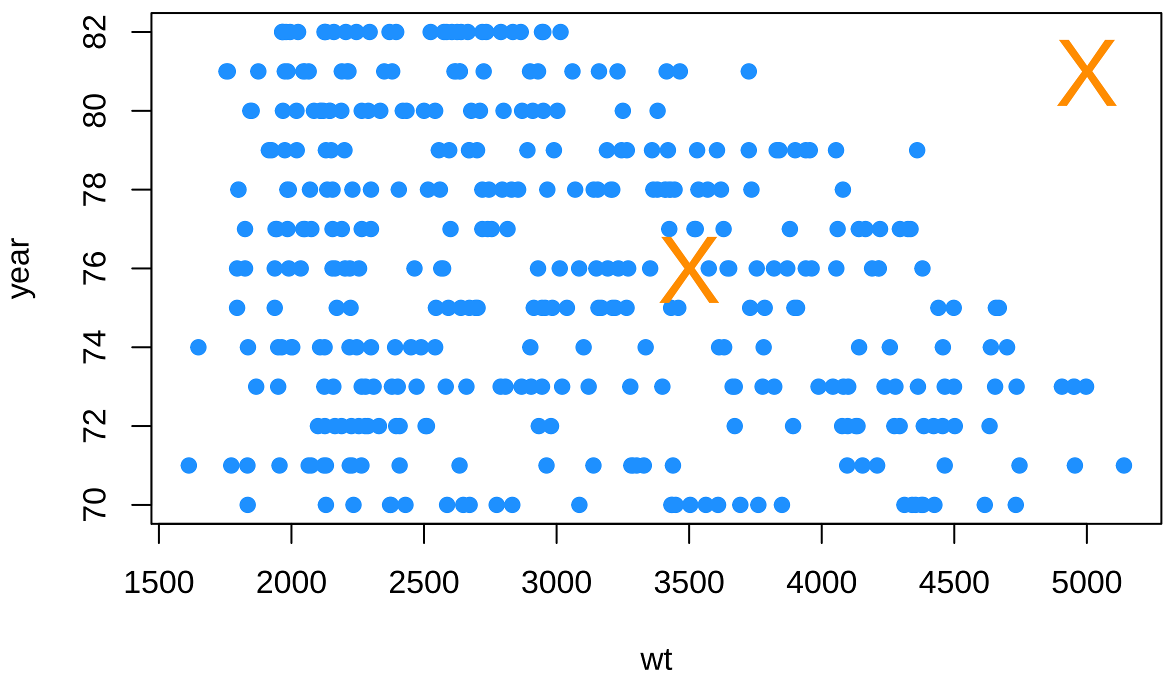
```
## [1] 76 81
```

```r
range(autompg$year)
```

```
## [1] 70 82
```

As are the years of each of the new cars.

```r
plot(year ~ wt, data = autompg, pch = 20, col = "dodgerblue", cex = 1.5)
points(new_cars, col = "darkorange", cex = 3, pch = "X")
```



However, we have to consider weight and year together now. And based on the above plot, one of the new cars is within the "blob" of observed values, while the other, the car from 1981 weighing 5000 pounds, is noticeably outside of the observed values. This is a hidden extrapolation which you should be aware of when using multiple regression.

Shifting gears back to the new data pair that can be reasonably estimated, we do a quick verification of some of the mathematics in R.

```r
x0 = c(1, 3500, 76)
x0 %*% beta_hat
```

```
##          [,1]
## [1,] 20.00684
```

$$x_0 = \begin{bmatrix} 1 \\ 3500 \\ 76 \end{bmatrix}$$

$$\hat{\beta} = \begin{bmatrix} -14.6376419 \\ -0.0066349 \\ 0.761402 \end{bmatrix}$$

$$\hat{y} = x_0^\top \hat{\beta} = \begin{bmatrix} 1 & 3500 & 76 \end{bmatrix} \begin{bmatrix} -14.6376419 \\ -0.0066349 \\ 0.761402 \end{bmatrix} = 20.0068411$$

Also note that, using a particular value for x_0, we can essentially extract certain $\hat{\beta}_j$ values.

```
x0 = c(0, 0, 1)
x0 %*% beta_hat
```

```
##          [,1]
## [1,] 0.761402
```

```
beta_hat
```

```
##               [,1]
## [1,] -14.637641945
## [2,]  -0.006634876
## [3,]   0.761401955
```

With this in mind, confidence intervals for the individual $\hat{\beta}_j$ are actually a special case of a confidence interval for mean response.

5.2.4 Prediction Intervals

As with SLR, creating prediction intervals involves one slight change to the standard error to account for the fact that we are now considering an observation, instead of a mean.

Here we use $\hat{y}_{Pred}$ to estimate Y_0, the value of Y for the predictor vector x_0

$$\begin{aligned} \hat{y}_{Pred} &= x_0^\top \hat{\beta} \\ &= \hat{\beta}_0 + \hat{\beta}_1 x_{01} + \hat{\beta}_2 x_{02} + \cdots + \hat{\beta}_{p-1} \end{aligned}$$

$$\begin{aligned} E[\hat{y}_{Pred}] &= Y_0 \\ &= x_0^\top \beta \\ &= \beta_0 + \beta_1 x_{01} + \beta_2 x_{02} + \cdots + \beta_{p-1} x_{0(p-1)} \end{aligned}$$

$$SE[\hat{y}_{Pred}] = s_e \sqrt{1 + x_0^\top \left(X^\top X\right)^{-1} x_0}$$

$$\hat{y}_{Pred} \pm t_{\alpha/2, n-p} \cdot s_e \sqrt{1 + x_0^\top \left(X^\top X\right)^{-1} x_0}$$

```
new_cars
```

```
##     wt year
## 1 3500   76
## 2 5000   81
```

```
predict(mpg_model, newdata = new_cars, interval = "prediction", level = 0.99)
```

```
##        fit       lwr      upr
## 1 20.00684 11.108294 28.90539
## 2 13.86154  4.848751 22.87432
```

5.3 Signifiance of Regression

The decomposition of variation that we had seen in SLR still remains true,

$$\sum_{i=1}^{n}(y_i - \bar{y})^2 = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2.$$

Which means that, we can still calculate R^2 in the same manner as before, which `R` continues to do automatically.

```
summary(mpg_model)$r.squared
```

```
## [1] 0.8082355
```

The interpretation changes slightly as compared to SLR. In this MLR case, we say that 80.82% for the observed variation in miles per gallon is explained by the linear relationship with the two predictor variables, weight and year.

We can also create the ANOVA table as before and perform the significance of regression test. In multiple regression, the significance of regression test is

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_{p-1} = 0.$$

Here, we see that the null hypothesis sets all of the β_j equal to 0, *except* the intercept, β_0. We could then say that the null model, or "model under the null hypothesis" is

$$Y_i = \beta_0 + \epsilon_i.$$

This is a model where the regression is insignificant. None of the predictors have a significant linear relationship with the response. Notationally, we will denote the fitted values of this model as $\hat{y}_{0i}$, which in this case happens to be:

$$\hat{y}_{0i} = \bar{y}.$$

The alternative hypothesis here is that at least one of the β_j from the null hypothesis is not 0.

$$H_1 : \text{At least one of } \beta_j \neq 0, j = 1, 2, \cdots, (p-1)$$

We could then say that the full model, or "model under the alternative hypothesis" is

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{i(p-1)} + \epsilon_i$$

This is a model where the regression is significant. At least one of the predictors has a significant linear relationship with the response. We will denote the fitted values of this model as $\hat{y}_{1i}$. The ANOVA table is then nearly identical to the ANOVA table from SLR, with two exceptions in the degrees of freedom column.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F
Regression	$\sum_{i=1}^{n}(\hat{y}_{1i} - \bar{y})^2$	$p - 1$	$SSReg/(p-1)$	$MSReg/MSE$
Error	$\sum_{i=1}^{n}(y_i - \hat{y}_{1i})^2$	$n - p$	$SSE/(n-p)$	
Total	$\sum_{i=1}^{n}(y_i - \bar{y})^2$	$n - 1$		

In summary, the F statistic is

$$F = \frac{\sum_{i=1}^{n}(\hat{y}_{1i} - \bar{y})^2/(p-1)}{\sum_{i=1}^{n}(y_i - \hat{y}_{1i})^2/(n-p)},$$

and the p-value is calculated as

$$P(F_{p-1,n-p} > F)$$

since we reject for large values of F. Here $F_{p-1,n-p}$ represents a random variable which follows an F distribution with $p-1$ and $n-p$ degrees of freedom.

To perform this test in `R`, we first explicitly specify the two models in `R` and save the results in different variables. We then use `anova()` to compare the two models, giving `anova()` the null model first and the alternative (full) model second.

In this case,

- H_0: $Y_i = \beta_0 + \epsilon_i$
- H_1: $Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \epsilon_i$

That is, in the null model, we use neither of the predictors, whereas in the full (alternative) model, at least one of the predictors is useful.

```r
null_mpg_model = lm(mpg ~ 1, data = autompg)
full_mpg_model = lm(mpg ~ wt + year, data = autompg)
anova(null_mpg_model, full_mpg_model)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ 1
## Model 2: mpg ~ wt + year
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1    389 23761.7
## 2    387  4556.6  2     19205 815.55 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

First, notice that `R` does not display the results in the same manner as the table above. More important than the layout of the table are its contents. We see that the value of the F statistic is 815.55, and the p-value is extremely low, so we reject the null hypothesis at any reasonable α and say that the regression is significant. At least one of `wt` or `year` has a useful linear relationship with `mpg`.

```r
summary(mpg_model)
```

```
##
## Call:
## lm(formula = mpg ~ wt + year, data = autompg)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -8.852 -2.292 -0.100  2.039 14.325
##
## Coefficients:
##               Estimate  Std. Error t value Pr(>|t|)
## (Intercept) -14.6376419   4.0233914  -3.638 0.000312 ***
## wt           -0.0066349   0.0002149 -30.881  < 2e-16 ***
## year          0.7614020   0.0497266  15.312  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.431 on 387 degrees of freedom
## Multiple R-squared:  0.8082, Adjusted R-squared:  0.8072
## F-statistic: 815.6 on 2 and 387 DF,  p-value: < 2.2e-16
```

Notice that the value reported in the row for `F-statistic` is indeed the F test statistic for the significance of the regression test, and additionally it reports the two relevant degrees of freedom.

Also, note that none of the individual t-tests are equivalent to the F-test as they were in SLR. This equivalence only holds for SLR because the individual test for β_1 is the same as testing for all non-intercept parameters, since there is only one.

We can also verify the sums of squares and degrees of freedom directly in `R`. You should match these to the table from `R` and use this to match `R`'s output to the written table above.

```r
# SSReg
sum((fitted(full_mpg_model) - fitted(null_mpg_model)) ^ 2)
```

```
## [1] 19205.03
```

```r
# SSE
sum(resid(full_mpg_model) ^ 2)
```

```
## [1] 4556.646
```

```r
# SST
sum(resid(null_mpg_model) ^ 2)
```

```
## [1] 23761.67
```

```r
# Degrees of Freedom: Regression
length(coef(full_mpg_model)) - length(coef(null_mpg_model))
```

```
## [1] 2
```

```
# Degrees of Freedom: Error
length(resid(full_mpg_model)) - length(coef(full_mpg_model))
```

```
## [1] 387
```

```
# Degrees of Freedom: Total
length(resid(null_mpg_model)) - length(coef(null_mpg_model))
```

```
## [1] 389
```

5.4 Nested Models

The significance of the regression test is actually a special case of testing what we will call **nested models**. More generally we can compare two models, where one model is "nested" inside the other, meaning one model contains a subset of the predictors from only the larger model.

Consider the following full model,

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{i(p-1)} + \epsilon_i$$

This model has $p - 1$ predictors, for a total of p β-parameters. We will denote the fitted values of this model as $\hat{y}_{1i}$.

Let the null model be

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_{(q-1)} x_{i(q-1)} + \epsilon_i$$

where $q < p$. This model has $q - 1$ predictors, for a total of q β-parameters. We will denote the fitted values of this model as $\hat{y}_{0i}$.

The difference between these two models can be codified by the null hypothesis of a test.

$$H_0 : \beta_q = \beta_{q+1} = \cdots = \beta_{p-1} = 0.$$

Specifically, the β-parameters from the full model that are not in the null model are zero. The resulting model, which is nested, is the null model.

We can then perform this test using an F-test, which is the result of the following ANOVA table.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F
Diff	$\sum_{i=1}^{n}(\hat{y}_{1i} - \hat{y}_{0i})^2$	$p - q$	$SSD/(p - q)$	MSD/MSE
Full	$\sum_{i=1}^{n}(y_i - \hat{y}_{1i})^2$	$n - p$	$SSE/(n - p)$	
Null	$\sum_{i=1}^{n}(y_i - \hat{y}_{0i})^2$	$n - q$		

$$F = \frac{\sum_{i=1}^{n}(\hat{y}_{1i} - \hat{y}_{0i})^2/(p - 1)}{\sum_{i=1}^{n}(y_i - \hat{y}_{1i})^2/(n - p)}.$$

Notice that the row for "Diff" compares the sum of the squared differences of the fitted values. The degrees of freedom is then the difference of the number of β-parameters estimated between the two models.

For example, the `autompg` dataset has a number of additional variables that we have yet to use.

```
names(autompg)
```

```
## [1] "mpg"  "cyl"  "disp" "hp"   "wt"   "acc"  "year"
```

We'll continue to use `mpg` as the response, but now we will consider two different models.

- Full: `mpg ~ wt + year + cyl + disp + hp + acc`
- Null: `mpg ~ wt + year`

Note that these are nested models, as the null model contains a subset of the predictors from the full model, and no additional predictors. Both models have an intercept β_0 as well as a coefficient in front of each of the predictors. We could then write the null hypothesis for comping these two models as,

$$H_0 : \beta_{cyl} = \beta_{disp} = \beta_{hp} = \beta_{acc} = 0$$

The alternative is simply that at least one of the β_j from the null is not 0.

To perform this test in `R` we first define both models, then give them to the `anova()` commands.

```
null_mpg_model = lm(mpg ~ wt + year, data = autompg)
#full_mpg_model = lm(mpg ~ wt + year + cyl + disp + hp + acc, data = autompg)
full_mpg_model = lm(mpg ~ ., data = autompg)
anova(null_mpg_model, full_mpg_model)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ wt + year
## Model 2: mpg ~ cyl + disp + hp + wt + acc + year
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    387 4556.6
## 2    383 4530.5  4     26.18 0.5533 0.6967
```

Here we have used the formula `mpg ~ .` to define to full model. This is the same as the commented out line. Specifically, this a common shortcut in `R` which reads, "model `mpg` as the response with each of the remaining variables in the data frame as predictors."

Here we see that the value of the F statistic is 0.553, and the p-value is very large, so we fail to reject the null hypothesis at any reasonable α and say that none of `cyl`, `disp`, `hp`, and `acc` are significant with `wt` and `year` already in the model.

Again, we verify the sums of squares and degrees of freedom directly in `R`. You should match these to the table from `R`, and use this to match `R`'s output to the written table above.

```
# SSDiff
sum((fitted(full_mpg_model) - fitted(null_mpg_model)) ^ 2)
```

```
## [1] 26.17981
```

```
# SSE (For Full)
sum(resid(full_mpg_model) ^ 2)
```

```
## [1] 4530.466
```

```
# SST (For Null)
sum(resid(null_mpg_model) ^ 2)
```

```
## [1] 4556.646
```

```
# Degrees of Freedom: Diff
length(coef(full_mpg_model)) - length(coef(null_mpg_model))
```

```
## [1] 4
```

```
# Degrees of Freedom: Full
length(resid(full_mpg_model)) - length(coef(full_mpg_model))
```

```
## [1] 383
```

```
# Degrees of Freedom: Null
length(resid(null_mpg_model)) - length(coef(null_mpg_model))
```

```
## [1] 387
```

5.5 Simulation

Since we ignored the derivation of certain results, we will again use simulation to convince ourselves of some of the above results. In particular, we will simulate samples of size `n = 100` from the model

$$Y_i = 5 + -2x_{i1} + 6x_{i2} + \epsilon_i, \qquad i = 1, 2, \ldots, n$$

where $\epsilon_i \sim N(0, \sigma^2 = 16)$. Here we have two predictors, so $p = 3$.

```
set.seed(1337)
n = 100 # sample size
p = 3

beta_0 = 5
beta_1 = -2
beta_2 = 6
sigma  = 4
```

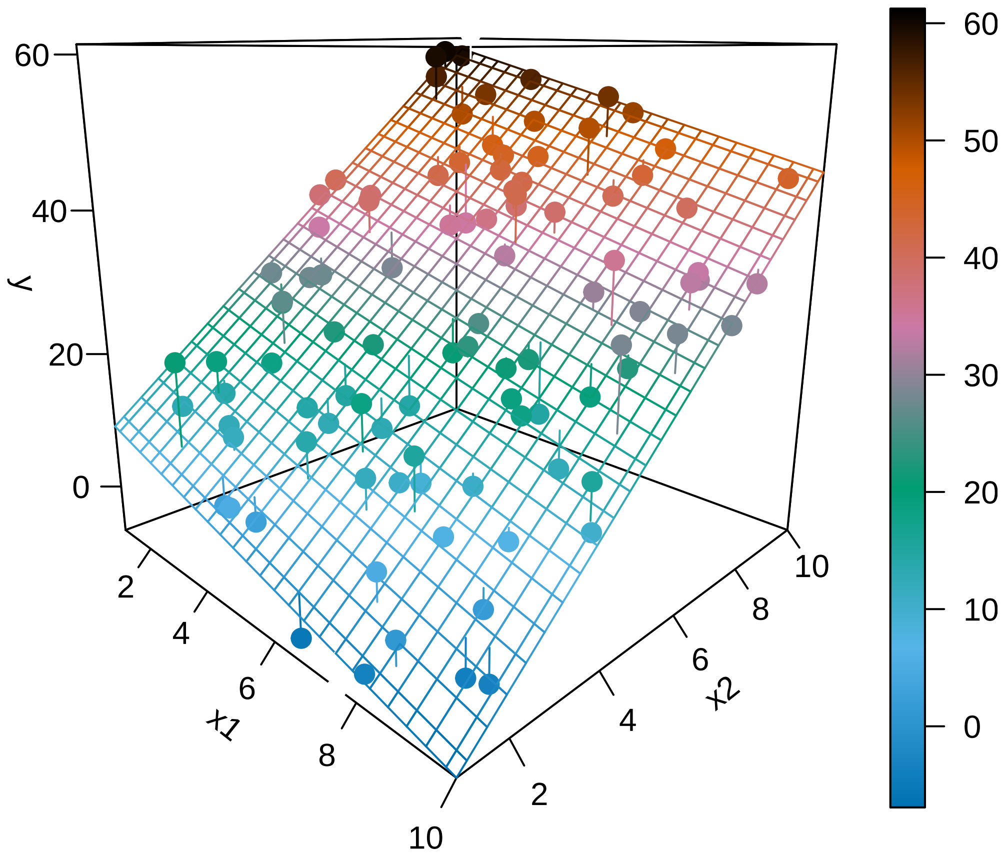
As is the norm with regression, the x values are considered fixed and known quantities, so we will simulate those first, and they remain the same for the rest of the simulation study. Also note we create an `x0` which is all `1`, which we need to create our `X` matrix. If you look at the matrix formulation of regression, this unit vector of all `1`s is a "predictor" that puts the intercept into the model. We also calculate the `C` matrix for later use.

```
x0 = rep(1, n)
x1 = sample(seq(1, 10, length = n))
x2 = sample(seq(1, 10, length = n))
X = cbind(x0, x1, x2)
C = solve(t(X) %*% X)
```

We then simulate the response according the model above. Lastly, we place the two predictors and response into a data frame. Note that we do **not** place `x0` in the data frame. This is a result of `R` adding an intercept by default.

```
eps      = rnorm(n, mean = 0, sd = sigma)
y        = beta_0 + beta_1 * x1 + beta_2 * x2 + eps
sim_data = data.frame(x1, x2, y)
```

Plotting this data and fitting the regression produces the following plot.



We then calculate

$$\hat{\beta} = \left(X^\top X\right)^{-1} X^\top y.$$

```
(beta_hat = C %*% t(X) %*% y)
```

```
##          [,1]
## x0  5.293609
## x1 -1.798593
## x2  5.775081
```

Notice that these values are the same as the coefficients found using `lm()` in `R`.

```
coef(lm(y ~ x1 + x2, data = sim_data))
```

```
## (Intercept)          x1          x2
##    5.293609   -1.798593    5.775081
```

Also, these values are close to what we would expect.

```
c(beta_0, beta_1, beta_2)
```

```
## [1]  5 -2  6
```

We then calculated the fitted values in order to calculate s_e, which we see is the same as the `sigma` which is returned by `summary()`.

```
y_hat = X %*% beta_hat
(s_e = sqrt(sum((y - y_hat) ^ 2) / (n - p)))
```

```
## [1] 3.976044
```

```
summary(lm(y ~ x1 + x2, data = sim_data))$sigma
```

```
## [1] 3.976044
```

So far so good. Everything checks out. Now we will finally simulate from this model repeatedly in order to obtain an empirical distribution of $\hat{\beta}_2$.

We expect $\hat{\beta}_2$ to follow a normal distribution,

$$\hat{\beta}_2 \sim N\left(\beta_2, \sigma^2 C_{22}\right).$$

In this case,

$$\hat{\beta}_2 \sim N\left(\mu = 6, \sigma^2 = 16 \times 0.0014777 = 0.0236438\right).$$

$$\hat{\beta}_2 \sim N\left(\mu = 6, \sigma^2 = 0.0236438\right).$$

Note that C_{22} corresponds to the element in the **third** row and **third** column since `R` is indexed starting at `1`, but we index the C matrix starting at `0` to match the diagonal elements to the corresponding β_j.

Note that C_{22} corresponds to the element in the **third** row and **third** column since β_2 is the **third** parameter in the model and because `R` is indexed starting at `1`. However, we index the C matrix starting at `0` to match the diagonal elements to the corresponding β_j.

```
C[3, 3]
```

```
## [1] 0.00147774
```

```r
C[2 + 1, 2 + 1]
```

```
## [1] 0.00147774
```

```r
sigma ^ 2 * C[2 + 1, 2 + 1]
```

```
## [1] 0.02364383
```

We now perform the simulation a large number of times. Each time, we update the `y` variable in the data frame, leaving the `x` variables the same. We then fit a model, and store $\hat{\beta}_2$.

```r
num_sims = 10000
beta_hat_2 = rep(0, num_sims)
for(i in 1:num_sims) {
  eps           = rnorm(n, mean = 0 , sd = sigma)
  sim_data$y    = beta_0 * x0 + beta_1 * x1 + beta_2 * x2 + eps
  fit           = lm(y ~ x1 + x2, data = sim_data)
  beta_hat_2[i] = coef(fit)[3]
}
```

We then see that the mean of the simulated values is close to the true value of β_2.

```r
mean(beta_hat_2)
```

```
## [1] 5.99871
```

```r
beta_2
```

```
## [1] 6
```

We also see that the variance of the simulated values is close to the true variance of $\hat{\beta}_2$.

$$Var[\hat{\beta}_2] = \sigma^2 C_{22} = 16 \times 0.0014777 = 0.0236438$$

```r
var(beta_hat_2)
```

```
## [1] 0.02360853
```

```r
sigma ^ 2 * C[2 + 1, 2 + 1]
```

```
## [1] 0.02364383
```

The standard deviations found from the simulated data and the parent population are also very close.

```r
sd(beta_hat_2)
```

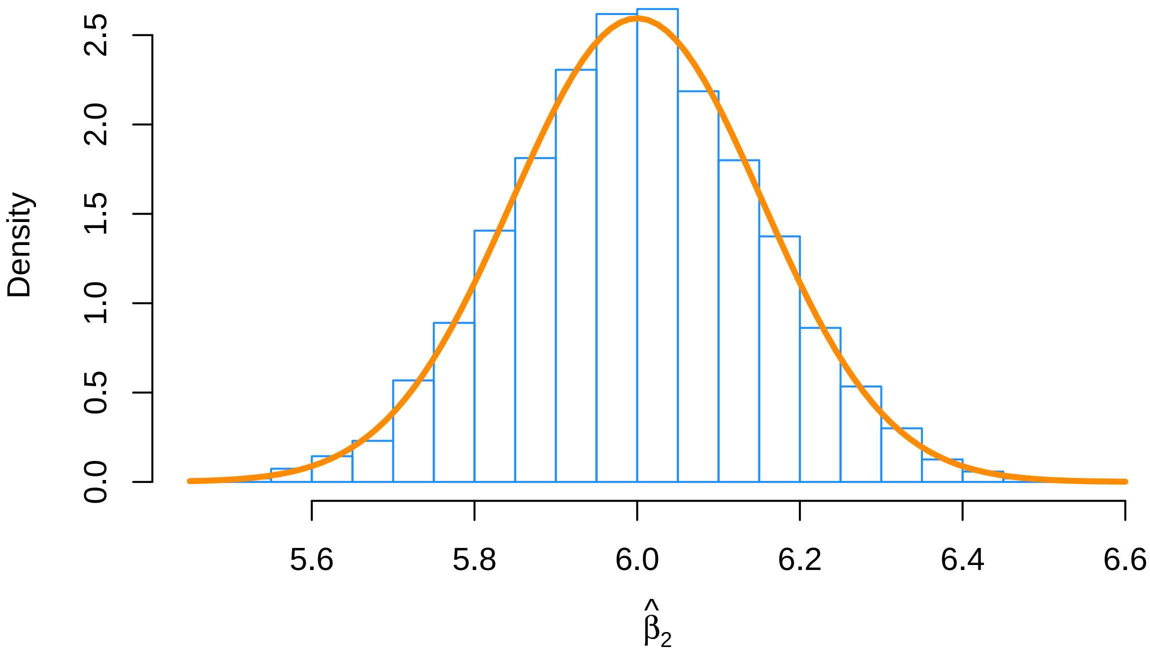
```
## [1] 0.1536507
```

```
sqrt(sigma ^ 2 * C[2 + 1, 2 + 1])
```

```
## [1] 0.1537655
```

Lastly, we plot a histogram of the simulated values, and overlay the true distribution.

```
hist(beta_hat_2, prob = TRUE, breaks = 20,
     xlab = expression(hat(beta)[2]), main = "", border = "dodgerblue")
curve(dnorm(x, mean = beta_2, sd = sqrt(sigma ^ 2 * C[2 + 1, 2 + 1])),
      col = "darkorange", add = TRUE, lwd = 3)
```



This looks good! The simulation-based histogram appears to be Normal with mean 6 and spread of about 0.15 as you measure from center to inflection point. That matches really well with the sampling distribution of $\hat{\beta}_2 \sim N\left(\mu = 6, \sigma^2 = 0.0236438\right)$.

One last check, we verify the $68 - 95 - 99.7$ rule.

```
sd_bh2 = sqrt(sigma ^ 2 * C[2 + 1, 2 + 1])
# We expect these to be: 0.68, 0.95, 0.997
mean(beta_2 - 1 * sd_bh2 < beta_hat_2 & beta_hat_2 < beta_2 + 1 * sd_bh2)
```

```
## [1] 0.6811
```

```
mean(beta_2 - 2 * sd_bh2 < beta_hat_2 & beta_hat_2 < beta_2 + 2 * sd_bh2)
```

```
## [1] 0.955
```

```
mean(beta_2 - 3 * sd_bh2 < beta_hat_2 & beta_hat_2 < beta_2 + 3 * sd_bh2)
```

```
## [1] 0.9972
```

Chapter 6

Categorical Predictors and Interactions

> "The greatest value of a picture is when it forces us to notice what we never expected to see."
> — **John Tukey**

After reading this chapter you will be able to:

- Include and interpret categorical variables in a linear regression model by way of dummy variables.
- Understand the implications of using a model with a categorical variable in two ways: levels serving as unique predictors versus levels serving as a comparison to a baseline.
- Construct and interpret linear regression models with interaction terms.
- Identify categorical variables in a data set and convert them into factor variables, if necessary, using R.

So far in each of our analyses, we have only used numeric variables as predictors. We have also only used *additive models*, meaning the effect any predictor had on the response was not dependent on the other predictors. In this chapter, we will remove both of these restrictions. We will fit models with categorical predictors, and use models that allow predictors to *interact*. The mathematics of multiple regression will remain largely unchanging, however, we will pay close attention to interpretation, as well as some difference in `R` usage.

6.1 Dummy Variables

For this chapter, we will briefly use the built in dataset `mtcars` before returning to our `autompg` dataset that we created in the last chapter. The `mtcars` dataset is somewhat smaller, so we'll quickly take a look at the entire dataset.

```
mtcars
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
```

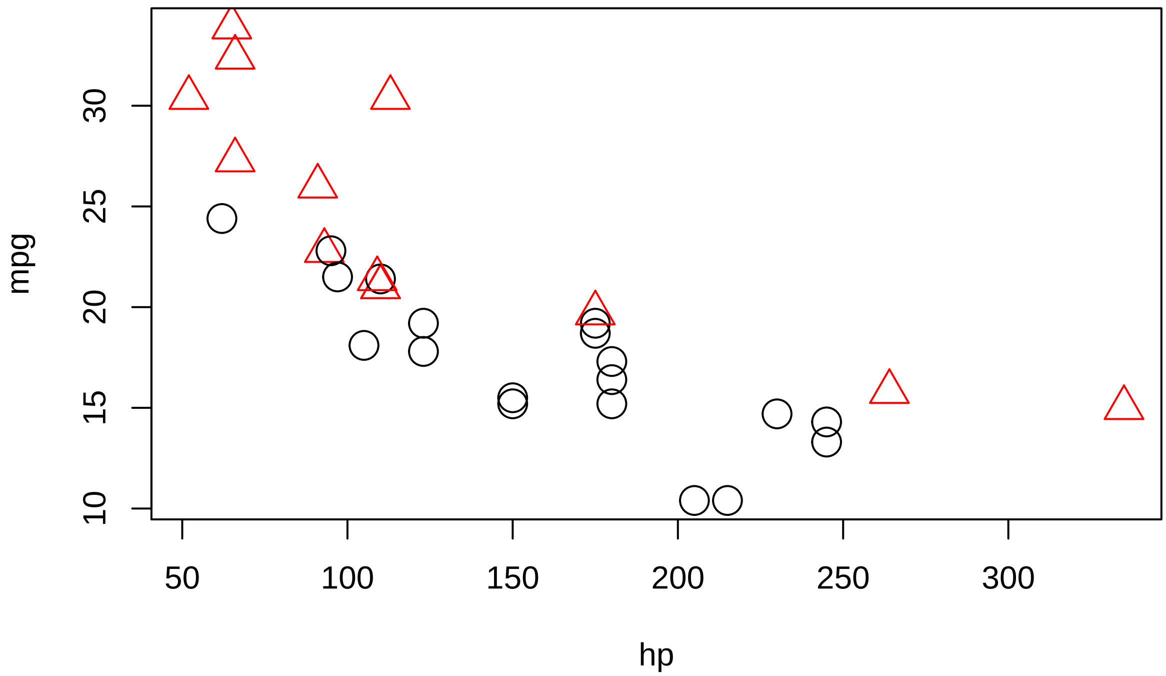
```
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

We will be interested in three of the variables: `mpg`, `hp`, and `am`.

- `mpg`: fuel efficiency, in miles per gallon.
- `hp`: horsepower, in foot-pounds per second.
- `am`: transmission. Automatic or manual.

As we often do, we will start by plotting the data. We are interested in `mpg` as the response variable, and `hp` as a predictor.

```
plot(mpg ~ hp, data = mtcars, col = am + 1, pch = am + 1, cex = 2)
```



We used a common R "trick" when plotting this data. The `am` variable takes two possible values; `0` for automatic transmission, and `1` for manual transmissions. R can use numbers to represent colors, however the color for `0` is white. So we take the `am` vector and add `1` to it. Then observations with automatic transmissions are now represented by `1`, which is black in R, and manual transmission are represented by `2`, which is red in R. (Note, we are only adding `1` inside the call to `plot()`, we are not actually modifying the values stored in `am`.)

We now fit the SLR model

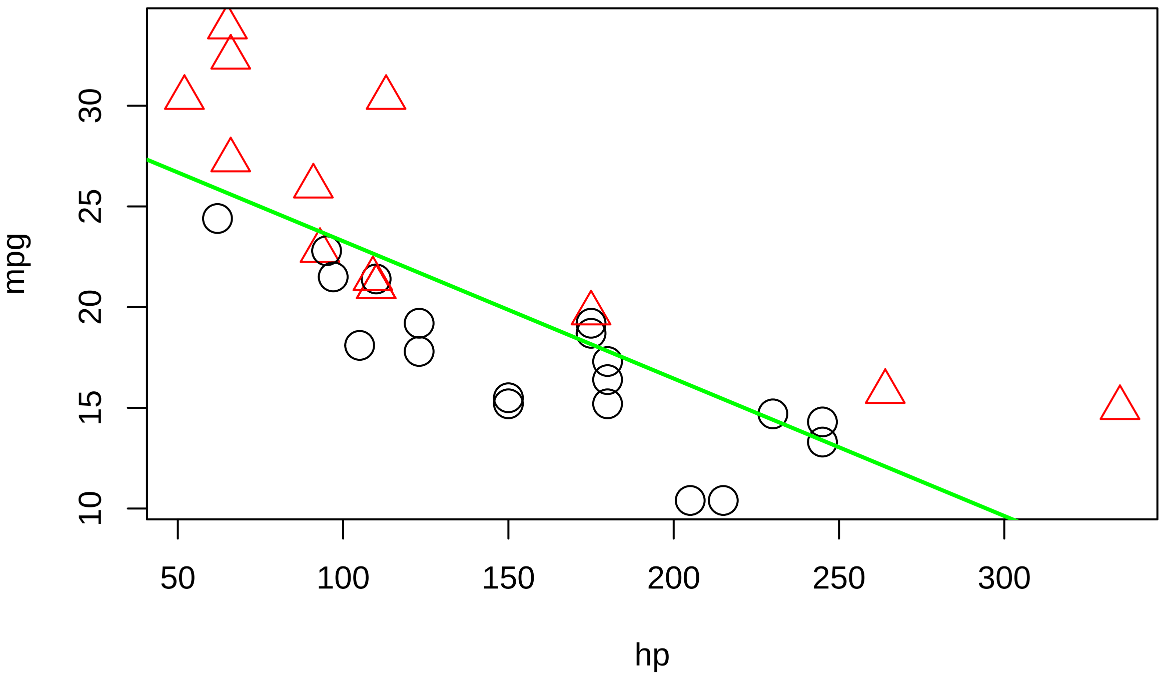
$$Y = \beta_0 + \beta_1 x_1 + \epsilon,$$

where Y is `mpg` and x_1 is `hp`. For notational brevity, we drop the index i for observations.

```
mpg_hp_slr = lm(mpg ~ hp, data = mtcars)
```

We then re-plot the data and add the fitted line to the plot.

```
plot(mpg ~ hp, data = mtcars, col = am + 1, pch = am + 1, cex = 2)
abline(mpg_hp_slr, lwd = 2, col = "green")
```



We should notice a pattern here. The red, manual observations largely fall above the line, while the black, automatic observations are mostly below the line. This means our model underestimates the fuel efficiency of manual transmissions, and overestimates the fuel efficiency of automatic transmissions. To correct for this, we will add a predictor to our model, namely, `am` as x_2.

Our new model is

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon,$$

where x_1 and Y remain the same, but now

$$x_2 = \begin{cases} 1 & \text{manual transmission} \\ 0 & \text{automatic transmission} \end{cases}.$$

In this case, we call x_2 a **dummy variable**. A dummy variable is somewhat unfortunately named, as it is in no way "dumb". In fact, it is actually somewhat clever. A dummy variable is a numerical variable that is used in a regression analysis to "code" for a binary categorical variable. Let's see how this works.

First, note that `am` is already a dummy variable, since it uses the values `0` and `1` to represent automatic and manual transmissions. Often, a variable like `am` would store the character values `auto` and `man` and we would either have to convert these to `0` and `1`, or, as we will see later, `R` will take care of creating dummy variables for us.

So, to fit the above model, we do so like any other multiple regression model we have seen before.

```
mpg_hp_add = lm(mpg ~ hp + am, data = mtcars)
```

Briefly checking the output, we see that `R` has estimated the three β parameters.

```
mpg_hp_add
```

```
##
## Call:
## lm(formula = mpg ~ hp + am, data = mtcars)
##
## Coefficients:
## (Intercept)           hp           am
##    26.58491     -0.05889      5.27709
```

Since x_2 can only take values `0` and `1`, we can effectively write two different models, one for manual and one for automatic transmissions.

For automatic transmissions, that is $x_2 = 0$, we have,

$$Y = \beta_0 + \beta_1 x_1 + \epsilon.$$

Then for manual transmissions, that is $x_2 = 1$, we have,

$$Y = (\beta_0 + \beta_2) + \beta_1 x_1 + \epsilon.$$

Notice that these models share the same slope, β_1, but have different intercepts, differing by β_2. So the change in `mpg` is the same for both models, but on average `mpg` differs by β_2 between the two transmission types.

We'll now calculate the estimated slope and intercept of these two models so that we can add them to a plot. Note that:

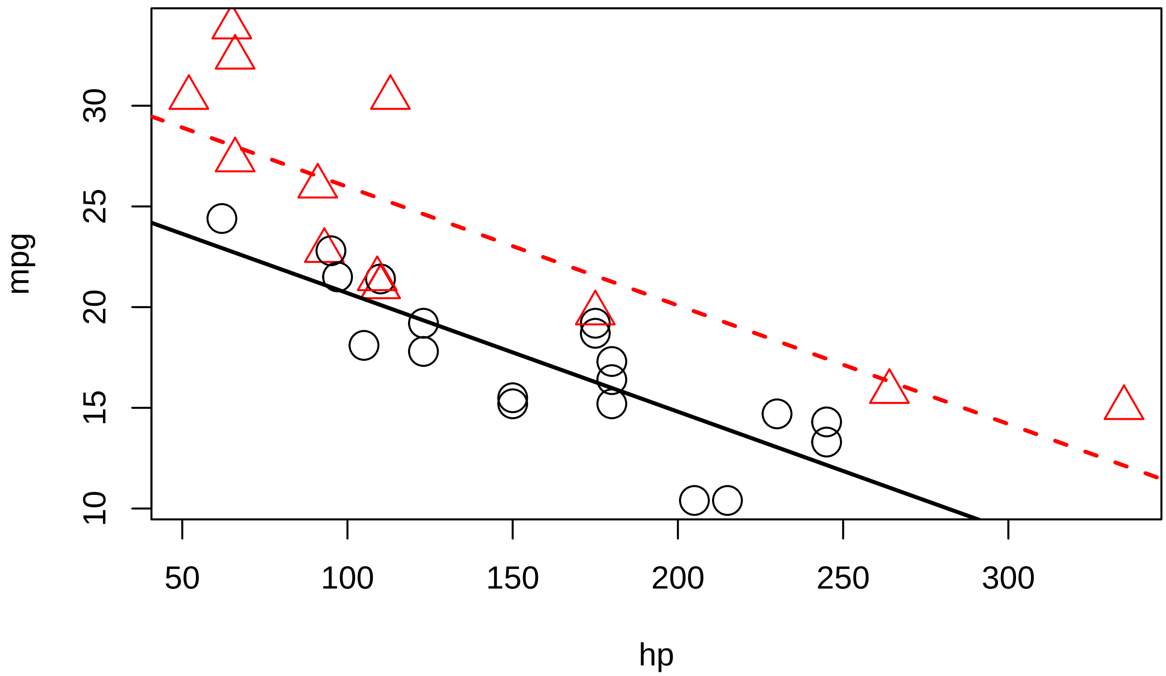
- $\hat{\beta}_0$ = `coef(mpg_hp_add)[1]` = 26.5849137
- $\hat{\beta}_1$ = `coef(mpg_hp_add)[2]` = -0.0588878
- $\hat{\beta}_2$ = `coef(mpg_hp_add)[3]` = 5.2770853

We can then combine these to calculate the estimated slope and intercepts.

```r
int_auto = coef(mpg_hp_add)[1]
int_manu = coef(mpg_hp_add)[1] + coef(mpg_hp_add)[3]

slope_auto = coef(mpg_hp_add)[2]
slope_manu = coef(mpg_hp_add)[2]
```

Re-plotting the data, we use these slopes and intercepts to add the "two" fitted models to the plot.

```r
plot(mpg ~ hp, data = mtcars, col = am + 1, pch = am + 1, cex = 2)
abline(int_auto, slope_auto, col = 1, lty = 1, lwd = 2) # add line for auto
abline(int_manu, slope_manu, col = 2, lty = 2, lwd = 2) # add line for manual
```



We notice right away that the points are no longer systematically incorrect. The red, manual observations vary about the red line in no particular pattern without underestimating the observations as before. The black, automatic points vary about the black line, also without an obvious pattern.

They say a picture is worth a thousand words, but as a statistician, sometimes a picture is worth an entire analysis. The above picture makes it plainly obvious that β_2 is significant, but let's verify mathematically. Essentially we would like to test:

$$H_0 : \beta_2 = 0 \quad \text{vs} \quad H_1 : \beta_2 \neq 0.$$

This is nothing new. Again, the math is the same as the multiple regression analyses we have seen before. We could perform either a t or F test here. The only difference is a slight change in interpretation. We could think of this as testing a model with a single line (H_0) against a model that allows two lines (H_1).

To obtain the test statistic and p-value for the t-test, we would use

```
summary(mpg_hp_add)$coef[3,]
```

```
##      Estimate    Std. Error       t value      Pr(>|t|)
## 5.27708530818 1.07954057578 4.88826953480 0.00003460318
```

To do the same for the F test, we would use

```
anova(mpg_hp_slr, mpg_hp_add)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ hp
## Model 2: mpg ~ hp + am
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1     30 447.67
```

```
## 2     29 245.44  1    202.24 23.895 0.0000346 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notice that these are indeed testing the same thing, as the p-values are exactly equal. (And the F test statistic is the t test statistic squared.)

Recapping some interpretations:

- $\hat{\beta}_0 = 26.5849137$ is the estimated average `mpg` for a car with an automatic transmission and **0** `hp`.
- $\hat{\beta}_0 + \hat{\beta}_2 = 31.8619991$ is the estimated average `mpg` for a car with a manual transmission and **0** `hp`.
- $\hat{\beta}_2 = 5.2770853$ is the estimated **difference** in average `mpg` for cars with manual transmissions as compared to those with automatic transmission, for **any** `hp`.
- $\hat{\beta}_1 = -0.0588878$ is the estimated change in average `mpg` for an increase in one `hp`, for **either** transmission types.

We should take special notice of those last two. In the model,

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon,$$

we see β_1 is the average change in Y for an increase in x_1, *no matter* the value of x_2. Also, β_2 is always the difference in the average of Y for *any* value of x_1. These are two restrictions we won't always want, so we need a way to specify a more flexible model.

Here we restricted ourselves to a single numerical predictor x_1 and one dummy variable x_2. However, the concept of a dummy variable can be used with larger multiple regression models. We only use a single numerical predictor here for ease of visualization since we can think of the "two lines" interpretation. But in general, we can think of a dummy variable as creating "two models," one for each category of a binary categorical variable.

6.2 Interactions

To remove the "same slope" restriction, we will now discuss **interaction**. To illustrate this concept, we will return to the `autompg` dataset we created in the last chapter, with a few more modifications.

```r
# read data frame from the web
autompg = read.table(
  "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data",
  quote = "\"",
  comment.char = "",
  stringsAsFactors = FALSE)
# give the dataframe headers
colnames(autompg) = c("mpg", "cyl", "disp", "hp", "wt", "acc", "year", "origin", "name")
# remove missing data, which is stored as "?"
autompg = subset(autompg, autompg$hp != "?")
# remove the plymouth reliant, as it causes some issues
autompg = subset(autompg, autompg$name != "plymouth reliant")
# give the dataset row names, based on the engine, year and name
rownames(autompg) = paste(autompg$cyl, "cylinder", autompg$year, autompg$name)
# remove the variable for name
```

```
autompg = subset(autompg, select = c("mpg", "cyl", "disp", "hp", "wt", "acc", "year", "origin"))
# change horsepower from character to numeric
autompg$hp = as.numeric(autompg$hp)
# create a dummary variable for foreign vs domestic cars. domestic = 1.
autompg$domestic = as.numeric(autompg$origin == 1)
# remove 3 and 5 cylinder cars (which are very rare.)
autompg = autompg[autompg$cyl != 5,]
autompg = autompg[autompg$cyl != 3,]
# the following line would verify the remaining cylinder possibilities are 4, 6, 8
#unique(autompg$cyl)
# change cyl to a factor variable
autompg$cyl = as.factor(autompg$cyl)
```

```
str(autompg)
```

```
## 'data.frame':    383 obs. of  9 variables:
##  $ mpg     : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cyl     : Factor w/ 3 levels "4","6","8": 3 3 3 3 3 3 3 3 3 3 ...
##  $ disp    : num  307 350 318 304 302 429 454 440 455 390 ...
##  $ hp      : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ wt      : num  3504 3693 3436 3433 3449 ...
##  $ acc     : num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year    : int  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ domestic: num  1 1 1 1 1 1 1 1 1 1 ...
```

We've removed cars with `3` and `5` cylinders , as well as created a new variable `domestic` which indicates whether or not a car was built in the United States. Removing the `3` and `5` cylinders is simply for ease of demonstration later in the chapter and would not be done in practice. The new variable `domestic` takes the value `1` if the car was built in the United States, and `0` otherwise, which we will refer to as "foreign." (We are arbitrarily using the United States as the reference point here.) We have also made `cyl` and `origin` into factor variables, which we will discuss later.

We'll now be concerned with three variables: `mpg`, `disp`, and `domestic`. We will use `mpg` as the response. We can fit a model,

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon,$$

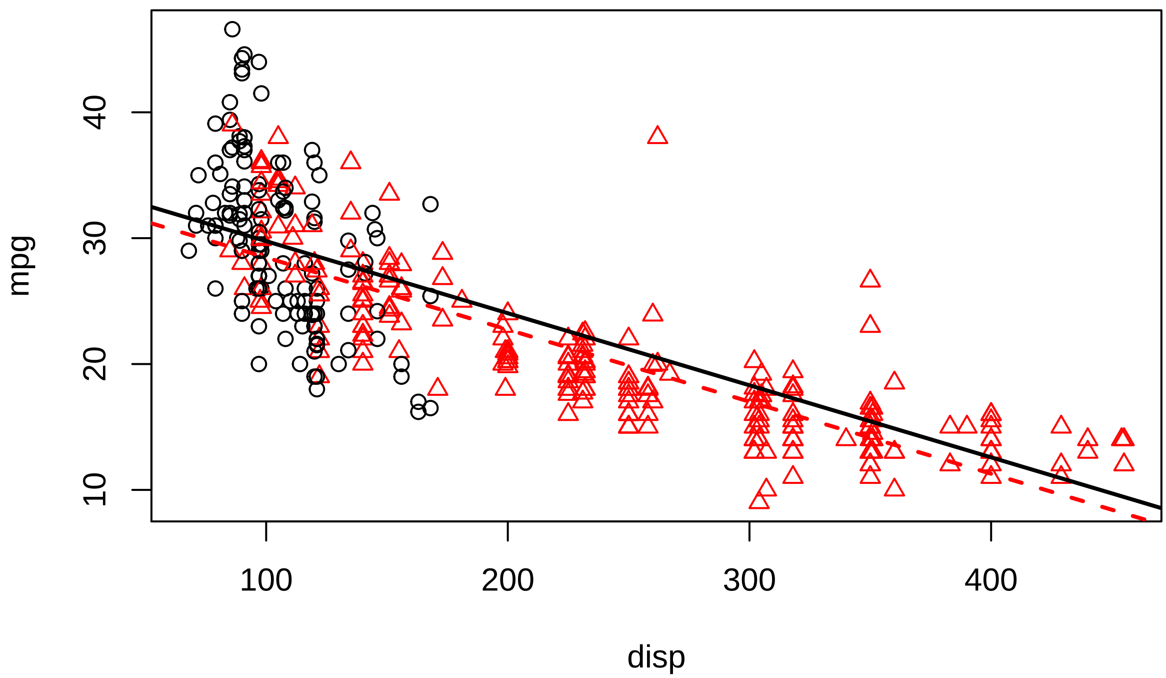
where

- Y is `mpg`, the fuel efficiency in miles per gallon,
- x_1 is `disp`, the displacement in cubic inches,
- x_2 is `domestic` as described above, which is a dummy variable.

We will fit this model, extract the slope and intercept for the "two lines," plot the data and add the lines.

```
mpg_disp_add = lm(mpg ~ disp + domestic, data = autompg)

int_for = coef(mpg_disp_add)[1]
int_dom = coef(mpg_disp_add)[1] + coef(mpg_disp_add)[3]

slope_for = coef(mpg_disp_add)[2]
```

```
slope_dom = coef(mpg_disp_add)[2]

plot(mpg ~ disp, data = autompg, col = domestic + 1, pch = domestic + 1)
abline(int_for, slope_for, col = 1, lty = 1, lwd = 2) # add line for foreign cars
abline(int_dom, slope_dom, col = 2, lty = 2, lwd = 2) # add line for domestic cars
```



This is a model that allows for two *parallel* lines, meaning the mpg can be different on average between foreign and domestic cars of the same engine displacement, but the change in average mpg for an increase in displacement is the same for both. We can see this model isn't doing very well here. The red line fits the red points fairly well, but the black line isn't doing very well for the black points, it should clearly have a more negative slope. Essentially, we would like a model that allows for two different slopes.

Consider the following model,

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \epsilon,$$

where x_1, x_2, and Y are the same as before, but we have added a new **interaction** term $x_1 x_2$ which multiplies x_1 and x_2, so we also have an additional β parameter β_3.

This model essentially creates two slopes and two intercepts, β_2 being the difference in intercepts and β_3 being the difference in slopes. To see this, we will break down the model into the two "sub-models" for foreign and domestic cars.

For foreign cars, that is $x_2 = 0$, we have

$$Y = \beta_0 + \beta_1 x_1 + \epsilon.$$

For domestic cars, that is $x_2 = 1$, we have

$$Y = (\beta_0 + \beta_2) + (\beta_1 + \beta_3) x_1 + \epsilon.$$

These two models have both different slopes and intercepts.

- β_0 is the average `mpg` for a foreign car with **0** `disp`.
- β_1 is the change in average `mpg` for an increase of one `disp`, for **foreign** cars.
- $\beta_0 + \beta_2$ is the average `mpg` for a domestic car with **0** `disp`.
- $\beta_1 + \beta_3$ is the change in average `mpg` for an increase of one `disp`, for **domestic** cars.

How do we fit this model in `R`? There are a number of ways.

One method would be to simply create a new variable, then fit a model like any other.

```
autompg$x3 = autompg$disp * autompg$domestic # THIS CODE NOT RUN!
do_not_do_this = lm(mpg ~ disp + domestic + x3, data = autompg) # THIS CODE NOT RUN!
```

You should only do this as a last resort. We greatly prefer not to have to modify our data simply to fit a model. Instead, we can tell `R` we would like to use the existing data with an interaction term, which it will create automatically when we use the `:` operator.

```
mpg_disp_int = lm(mpg ~ disp + domestic + disp:domestic, data = autompg)
```

An alternative method, which will fit the exact same model as above would be to use the `*` operator. This method automatically creates the interaction term, as well as any "lower order terms," which in this case are the first order terms for `disp` and `domestic`

```
mpg_disp_int2 = lm(mpg ~ disp * domestic, data = autompg)
```

We can quickly verify that these are doing the same thing.

```
coef(mpg_disp_int)
```

```
##   (Intercept)          disp      domestic disp:domestic
##    46.0548423    -0.1569239   -12.5754714     0.1025184
```

```
coef(mpg_disp_int2)
```

```
##   (Intercept)          disp      domestic disp:domestic
##    46.0548423    -0.1569239   -12.5754714     0.1025184
```

We see that both the variables, and their coefficient estimates are indeed the same for both models.

```
summary(mpg_disp_int)
```

```
##
## Call:
## lm(formula = mpg ~ disp + domestic + disp:domestic, data = autompg)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.8332  -2.8956  -0.8332   2.2828  18.7749
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)    46.05484    1.80582  25.504  < 2e-16 ***
## disp           -0.15692    0.01668  -9.407  < 2e-16 ***
## domestic      -12.57547    1.95644  -6.428 3.90e-10 ***
## disp:domestic   0.10252    0.01692   6.060 3.29e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.308 on 379 degrees of freedom
## Multiple R-squared:  0.7011, Adjusted R-squared:  0.6987
## F-statistic: 296.3 on 3 and 379 DF,  p-value: < 2.2e-16
```

We see that using `summary()` gives the usual output for a multiple regression model. We pay close attention to the row for `disp:domestic` which tests,

$$H_0 : \beta_3 = 0.$$

In this case, testing for $\beta_3 = 0$ is testing for two lines with parallel slopes versus two lines with possibly different slopes. The `disp:domestic` line in the `summary()` output uses a t-test to perform the test.

We could also use an ANOVA F-test. The additive model, without interaction is our null model, and the interaction model is the alternative.

```r
anova(mpg_disp_add, mpg_disp_int)
```

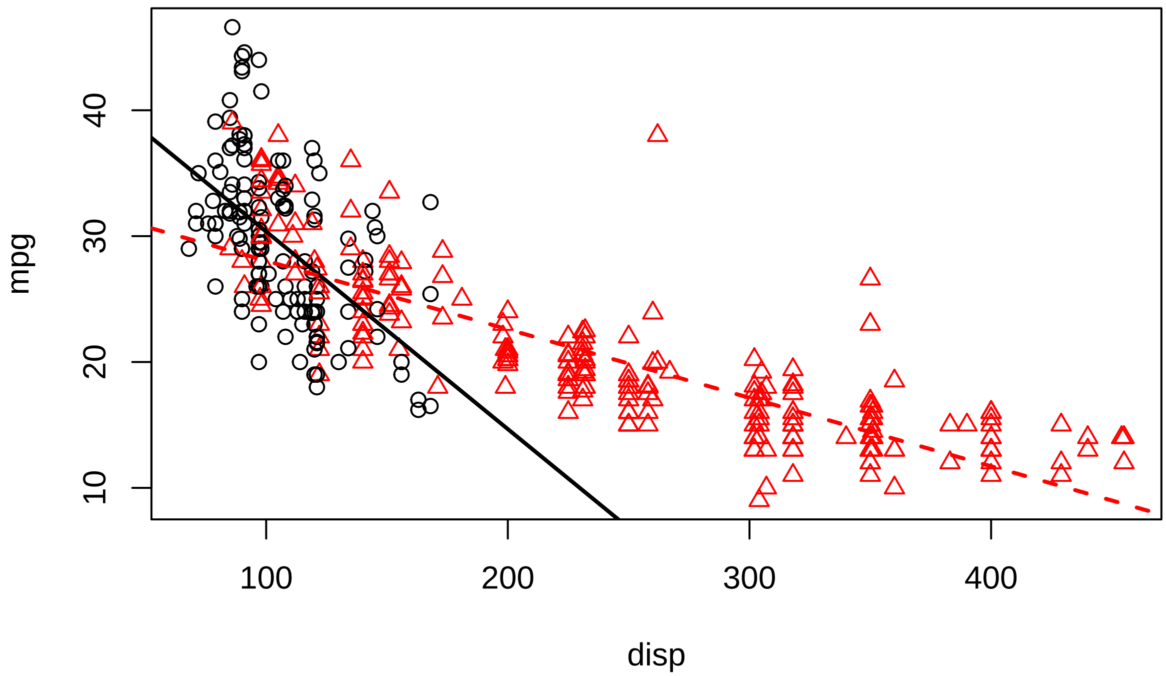
```
## Analysis of Variance Table
##
## Model 1: mpg ~ disp + domestic
## Model 2: mpg ~ disp + domestic + disp:domestic
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    380 7714.0
## 2    379 7032.6  1    681.36 36.719 3.294e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Again we see this test has the same p-value as the t-test. Also the p-value is extremely low, so between the two, we choose the interaction model.

```r
int_for = coef(mpg_disp_int)[1]
int_dom = coef(mpg_disp_int)[1] + coef(mpg_disp_int)[3]

slope_for = coef(mpg_disp_int)[2]
slope_dom = coef(mpg_disp_int)[2] + coef(mpg_disp_int)[4]
```

Here we again calculate the slope and intercepts for the two lines for use in plotting.

```r
plot(mpg ~ disp, data = autompg, col = domestic + 1, pch = domestic + 1)
abline(int_for, slope_for, col = 1, lty = 1, lwd = 2) # add line for foreign cars
abline(int_dom, slope_dom, col = 2, lty = 2, lwd = 2) # add line for domestic cars
```



We see that these lines fit the data much better, which matches the result of our tests.

So far we have only seen interaction between a categorical variable (`domestic`) and a numerical variable (`disp`). While this is easy to visualize, since it allows for different slopes for two lines, it is not the only type of interaction we can use in a model. We can also consider interactions between two numerical variables.

Consider the model,

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \epsilon,$$

where

- Y is `mpg`, the fuel efficiency in miles per gallon,
- x_1 is `disp`, the displacement in cubic inches,
- x_2 is `hp`, the horsepower, in foot-pounds per second.

How does `mpg` change based on `disp` in this model? We can rearrange some terms to see how.

$$Y = \beta_0 + (\beta_1 + \beta_3 x_2) x_1 + \beta_2 x_2 + \epsilon$$

So, for a one unit increase in x_1 (`disp`), the mean of Y (`mpg`) increases $\beta_1 + \beta_3 x_2$, which is a different value depending on the value of x_2 (`hp`)!

Since we're now working in three dimensions, this model can't be easily justified via visualizations like the previous example. Instead, we will have to rely on a test.

```
mpg_disp_add_hp = lm(mpg ~ disp + hp, data = autompg)
mpg_disp_int_hp = lm(mpg ~ disp * hp, data = autompg)
summary(mpg_disp_int_hp)
```

```
##
## Call:
```

```
## lm(formula = mpg ~ disp * hp, data = autompg)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.7849  -2.3104  -0.5699   2.1453  17.9211
##
## Coefficients:
##               Estimate  Std. Error t value Pr(>|t|)
## (Intercept) 52.40819978  1.52272673   34.42   <2e-16 ***
## disp        -0.10017377  0.00663825  -15.09   <2e-16 ***
## hp          -0.21981997  0.01986944  -11.06   <2e-16 ***
## disp:hp      0.00056583  0.00005165   10.96   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.896 on 379 degrees of freedom
## Multiple R-squared:  0.7554, Adjusted R-squared:  0.7535
## F-statistic: 390.2 on 3 and 379 DF,  p-value: < 2.2e-16
```

Using `summary()` we focus on the row for `disp:hp` which tests,

$$H_0 : \beta_3 = 0.$$

Again, we see a very low p-value so we reject the null (additive model) in favor of the interaction model. Again, there is an equivalent F-test.

```r
anova(mpg_disp_add_hp, mpg_disp_int_hp)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ disp + hp
## Model 2: mpg ~ disp * hp
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    380 7576.6
## 2    379 5754.2  1    1822.3 120.03 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We can take a closer look at the coefficients of our fitted interaction model.

```r
coef(mpg_disp_int_hp)
```

```
##   (Intercept)          disp            hp       disp:hp
## 52.4081997848 -0.1001737655 -0.2198199720  0.0005658269
```

- $\hat{\beta}_0 = 52.4081998$ is the estimated average `mpg` for a car with 0 `disp` and 0 `hp`.
- $\hat{\beta}_1 = -0.1001738$ is the estimated change in average `mpg` for an increase in 1 `disp`, **for a car with 0 `hp`**.
- $\hat{\beta}_2 = -0.21982$ is the estimated change in average `mpg` for an increase in 1 `hp`, **for a car with 0 `disp`**.
- $\hat{\beta}_3 = 0.0005658$ is an estimate of the modification to the change in average `mpg` for an increase in `disp`, for a car of a certain `hp` (or vice versa).

That last coefficient needs further explanation. Recall the rearrangement we made earlier

$$Y = \beta_0 + (\beta_1 + \beta_3 x_2)x_1 + \beta_2 x_2 + \epsilon.$$

So, our estimate for $\beta_1 + \beta_3 x_2$, is $\hat{\beta}_1 + \hat{\beta}_3 x_2$, which in this case is

$$-0.1001738 + 0.0005658 x_2.$$

This says that, for an increase of one `disp` we see an estimated change in average `mpg` of $-0.1001738 + 0.0005658 x_2$. So how `disp` and `mpg` are related, depends on the `hp` of the car.

So for a car with 50 `hp`, the estimated change in average `mpg` for an increase of one `disp` is

$$-0.1001738 + 0.0005658 \cdot 50 = -0.0718824$$

And for a car with 350 `hp`, the estimated change in average `mpg` for an increase of one `disp` is

$$-0.1001738 + 0.0005658 \cdot 350 = 0.0978657$$

Notice the sign changed!

6.3 Factor Variables

So far in this chapter, we have limited our use of categorical variables to binary categorical variables. Specifically, we have limited ourselves to dummy variables which take a value of `0` or `1` and represent a categorical variable numerically.

We will now discuss **factor** variables, which is a special way that `R` deals with categorical variables. With factor variables, a human user can simply think about the categories of a variable, and `R` will take care of the necessary dummy variables without any 0/1 assignment being done by the user.

```
is.factor(autompg$domestic)
```

```
## [1] FALSE
```

Earlier when we used the `domestic` variable, it was **not** a factor variable. It was simply a numerical variable that only took two possible values, `1` for domestic, and `0` for foreign. Let's create a new variable `origin` that stores the same information, but in a different way.

```
autompg$origin[autompg$domestic == 1] = "domestic"
autompg$origin[autompg$domestic == 0] = "foreign"
head(autompg$origin)
```

```
## [1] "domestic" "domestic" "domestic" "domestic" "domestic" "domestic"
```

Now the `origin` variable stores `"domestic"` for domestic cars and `"foreign"` for foreign cars.

```
is.factor(autompg$origin)
```

```
## [1] FALSE
```

However, this is simply a vector of character values. A vector of car models is a character variable in `R`. A vector of Vehicle Identification Numbers (VINs) is a character variable as well. But those don't represent a short list of levels that might influence a response variable. We will want to **coerce** this origin variable to be something more: a factor variable.

```
autompg$origin = as.factor(autompg$origin)
```

Now when we check the structure of the `autompg` dataset, we see that `origin` is a factor variable.

```
str(autompg)
```

```
## 'data.frame':    383 obs. of  9 variables:
##  $ mpg     : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cyl     : Factor w/ 3 levels "4","6","8": 3 3 3 3 3 3 3 3 3 3 ...
##  $ disp    : num  307 350 318 304 302 429 454 440 455 390 ...
##  $ hp      : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ wt      : num  3504 3693 3436 3433 3449 ...
##  $ acc     : num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year    : int  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin  : Factor w/ 2 levels "domestic","foreign": 1 1 1 1 1 1 1 1 1 1 ...
##  $ domestic: num  1 1 1 1 1 1 1 1 1 1 ...
```

Factor variables have **levels** which are the possible values (categories) that the variable may take, in this case foreign or domestic.

```
levels(autompg$origin)
```

```
## [1] "domestic" "foreign"
```

Recall that previously we have fit the model

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \epsilon,$$

where

- Y is `mpg`, the fuel efficiency in miles per gallon,
- x_1 is `disp`, the displacement in cubic inches,
- x_2 is `domestic` a dummy variable where `1` indicates a domestic car.

```
(mod_dummy = lm(mpg ~ disp * domestic, data = autompg))
```

```
## 
## Call:
## lm(formula = mpg ~ disp * domestic, data = autompg)
## 
## Coefficients:
##   (Intercept)           disp       domestic  disp:domestic  
##       46.0548        -0.1569       -12.5755         0.1025
```

So here we see that

$$\hat{\beta}_0 + \hat{\beta}_2 = 46.0548423 + -12.5754714 = 33.4793709$$

is the estimated average `mpg` for a **domestic** car with 0 `disp`.

Now let's try to do the same, but using our new factor variable.

```
(mod_factor = lm(mpg ~ disp * origin, data = autompg))
```

```
##
## Call:
## lm(formula = mpg ~ disp * origin, data = autompg)
##
## Coefficients:
##       (Intercept)                disp       originforeign  disp:originforeign
##          33.47937            -0.05441            12.57547            -0.10252
```

It seems that it doesn't produce the same results. Right away we notice that the intercept is different, as is the the coefficient in front of `disp`. We also notice that the remaining two coefficients are of the same magnitude as their respective counterparts using the domestic variable, but with a different sign. Why is this happening?

It turns out, that by using a factor variable, `R` is automatically creating a dummy variable for us. However, it is not the dummy variable that we had originally used ourselves.

`R` is fitting the model

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \epsilon,$$

where

- Y is `mpg`, the fuel efficiency in miles per gallon,
- x_1 is `disp`, the displacement in cubic inches,
- x_2 **is a dummy variable create by `R`.** It uses `1` to represent a **foreign car**.

So now,

$$\hat{\beta}_0 = 33.4793709$$

is the estimated average `mpg` for a **domestic** car with 0 `disp`, which is indeed the same as before.

When `R` created x_2, the dummy variable, it used domestic cars as the **reference** level, that is the default value of the factor variable. So when the dummy variable is `0`, the model represents this reference level, which is domestic. (`R` makes this choice because domestic comes before foreign alphabetically.)

So the two models have different estimated coefficients, but due to the different model representations, they are actually the same model.

6.3.1 Factors with More Than Two Levels

Let's now consider a factor variable with more than two levels. In this dataset, `cyl` is an example.

```r
is.factor(autompg$cyl)
```

```
## [1] TRUE
```

```r
levels(autompg$cyl)
```

```
## [1] "4" "6" "8"
```

Here the `cyl` variable has three possible levels: `4`, `6`, and `8`. You may wonder, why not simply use `cyl` as a numerical variable? You certainly could.

However, that would force the difference in average `mpg` between `4` and `6` cylinders to be the same as the difference in average mpg between `6` and `8` cylinders. That usually make senses for a continuous variable, but not for a discrete variable with so few possible values. In the case of this variable, there is no such thing as a 7-cylinder engine or a 6.23-cylinder engine in personal vehicles. For these reasons, we will simply consider `cyl` to be categorical. This is a decision that will commonly need to be made with ordinal variables. Often, with a large number of categories, the decision to treat them as numerical variables is appropriate because a large number of dummy variables are then needed to represent these variables.

Let's define three dummy variables related to the `cyl` factor variable.

$$v_1 = \begin{cases} 1 & \text{4 cylinder} \\ 0 & \text{not 4 cylinder} \end{cases}$$

$$v_2 = \begin{cases} 1 & \text{6 cylinder} \\ 0 & \text{not 6 cylinder} \end{cases}$$

$$v_3 = \begin{cases} 1 & \text{8 cylinder} \\ 0 & \text{not 8 cylinder} \end{cases}$$

Now, let's fit an additive model in `R`, using `mpg` as the response, and `disp` and `cyl` as predictors. This should be a model that uses "three regression lines" to model `mpg`, one for each of the possible `cyl` levels. They will all have the same slope (since it is an additive model), but each will have its own intercept.

```r
(mpg_disp_add_cyl = lm(mpg ~ disp + cyl, data = autompg))
```

```
##
## Call:
## lm(formula = mpg ~ disp + cyl, data = autompg)
##
## Coefficients:
## (Intercept)         disp         cyl6         cyl8
##    34.99929     -0.05217     -3.63325     -2.03603
```

The question is, what is the model that `R` has fit here? It has chosen to use the model

$$Y = \beta_0 + \beta_1 x + \beta_2 v_2 + \beta_3 v_3 + \epsilon,$$

where

- Y is `mpg`, the fuel efficiency in miles per gallon,
- x is `disp`, the displacement in cubic inches,
- v_2 and v_3 are the dummy variables define above.

Why doesn't `R` use v_1? Essentially because it doesn't need to. To create three lines, it only needs two dummy variables since it is using a reference level, which in this case is a 4 cylinder car. The three "sub models" are then:

- 4 Cylinder: $Y = \beta_0 + \beta_1 x + \epsilon$
- 6 Cylinder: $Y = (\beta_0 + \beta_2) + \beta_1 x + \epsilon$
- 8 Cylinder: $Y = (\beta_0 + \beta_3) + \beta_1 x + \epsilon$

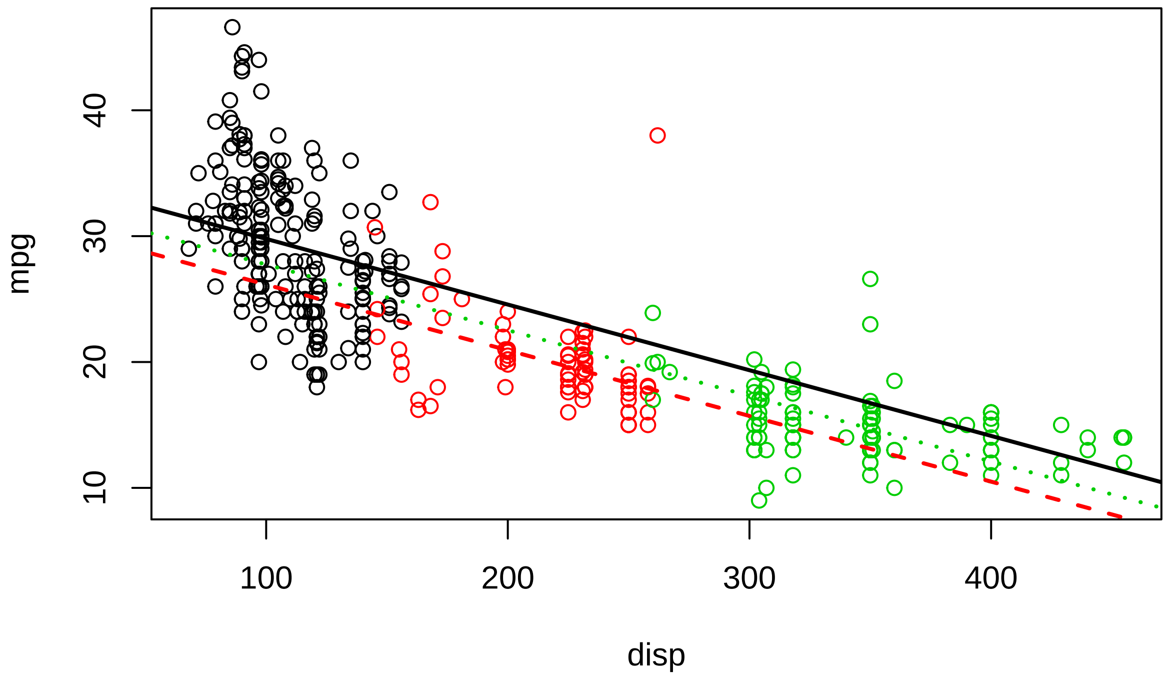
Notice that they all have the same slope. However, using the two dummy variables, we achieve the three intercepts.

- β_0 is the average `mpg` for a 4 cylinder car with 0 `disp`.
- $\beta_0 + \beta_2$ is the average `mpg` for a 6 cylinder car with 0 `disp`.
- $\beta_0 + \beta_3$ is the average `mpg` for a 8 cylinder car with 0 `disp`.

So because 4 cylinder is the reference level, β_0 is specific to 4 cylinders, but β_2 and β_3 are used to represent quantities relative to 4 cylinders.

As we have done before, we can extract these intercepts and slopes for the three lines, and plot them accordingly.

```
int_4cyl = coef(mpg_disp_add_cyl)[1]
int_6cyl = coef(mpg_disp_add_cyl)[1] + coef(mpg_disp_add_cyl)[3]
int_8cyl = coef(mpg_disp_add_cyl)[1] + coef(mpg_disp_add_cyl)[4]

slope_all_cyl = coef(mpg_disp_add_cyl)[2]

plot(mpg ~ disp, data = autompg, col = cyl)
abline(int_4cyl, slope_all_cyl, col = 1, lty = 1, lwd = 2)
abline(int_6cyl, slope_all_cyl, col = 2, lty = 2, lwd = 2)
abline(int_8cyl, slope_all_cyl, col = 3, lty = 3, lwd = 2)
```



On this plot, we have

- 4 Cylinder: black dots, solid black line.
- 6 Cylinder: red dots, dashed red line.
- 8 Cylinder: green dots, dotted green line.

The odd result here is that we're estimating that 8 cylinder cars have better fuel efficiency than 6 cylinder cars at **any** displacement! The dotted green line is always above the dashed red line. That doesn't seem right. Maybe for very large displacement engines that could be true, but that seems wrong for medium to low displacement.

To attempt to fix this, we will try using an interaction model, that is, instead of simply three intercepts and one slope, we will allow for three slopes. Again, we'll let `R` take the wheel, (no pun intended) then figure out what model it has applied.

```
(mpg_disp_int_cyl = lm(mpg ~ disp * cyl, data = autompg))
```

```
##
## Call:
## lm(formula = mpg ~ disp * cyl, data = autompg)
##
## Coefficients:
## (Intercept)         disp         cyl6         cyl8    disp:cyl6    disp:cyl8
##    43.59052     -0.13069    -13.20026    -20.85706      0.08299      0.10817
```

```
# could also use mpg ~ disp + cyl + disp:cyl
```

`R` has again chosen to use 4 cylinder cars as the reference level, but this also now has an effect on the interaction terms. `R` has fit the model.

$$Y = \beta_0 + \beta_1 x + \beta_2 v_2 + \beta_3 v_3 + \gamma_2 x v_2 + \gamma_3 x v_3 + \epsilon$$

We're using γ like a β parameter for simplicity, so that, for example β_2 and γ_2 are both associated with v_2.

Now, the three "sub models" are:

- 4 Cylinder: $Y = \beta_0 + \beta_1 x + \epsilon$.
- 6 Cylinder: $Y = (\beta_0 + \beta_2) + (\beta_1 + \gamma_2)x + \epsilon$.
- 8 Cylinder: $Y = (\beta_0 + \beta_3) + (\beta_1 + \gamma_3)x + \epsilon$.

Interpreting some parameters and coefficients then:

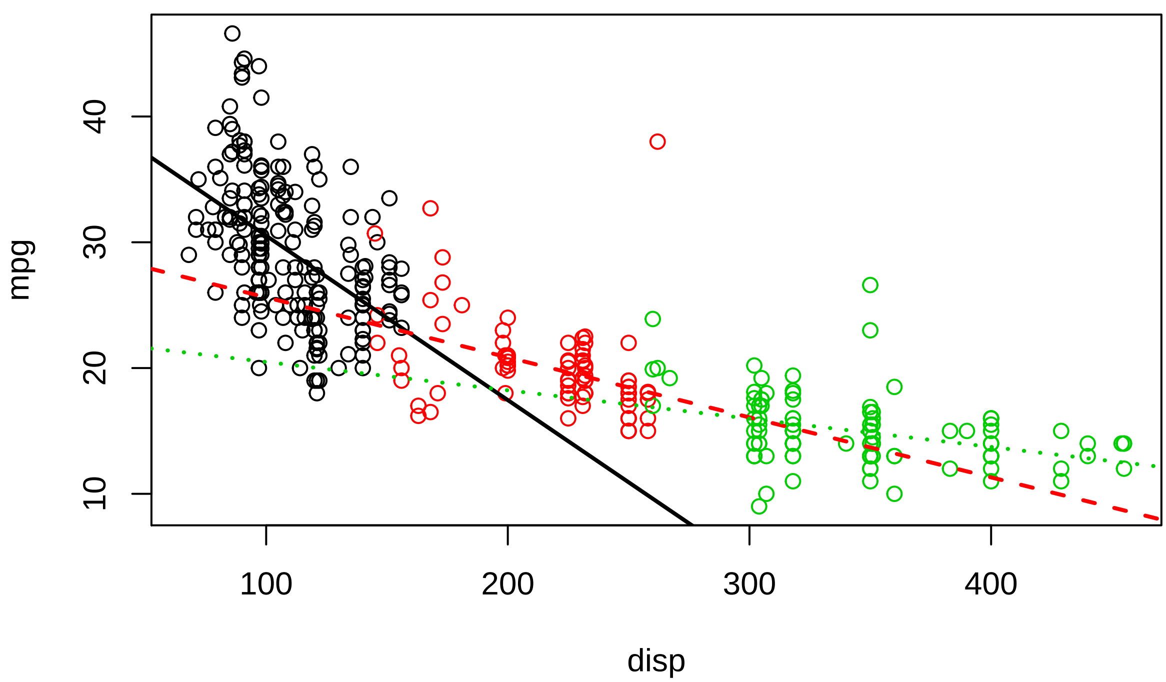
- $(\beta_0 + \beta_2)$ is the average `mpg` of a 6 cylinder car with 0 `disp`
- $(\hat{\beta}_1 + \hat{\gamma}_3) = -0.1306935 + 0.1081714 = -0.0225221$ is the estimated change in average `mpg` for an increase of one `disp`, for an 8 cylinder car.

So, as we have seen before β_2 and β_3 change the intercepts for 6 and 8 cylinder cars relative to the reference level of β_0 for 4 cylinder cars.

Now, similarly γ_2 and γ_3 change the slopes for 6 and 8 cylinder cars relative to the reference level of β_1 for 4 cylinder cars.

Once again, we extract the coefficients and plot the results.

```
int_4cyl = coef(mpg_disp_int_cyl)[1]
int_6cyl = coef(mpg_disp_int_cyl)[1] + coef(mpg_disp_int_cyl)[3]
int_8cyl = coef(mpg_disp_int_cyl)[1] + coef(mpg_disp_int_cyl)[4]

slope_4cyl = coef(mpg_disp_int_cyl)[2]
slope_6cyl = coef(mpg_disp_int_cyl)[2] + coef(mpg_disp_int_cyl)[5]
slope_8cyl = coef(mpg_disp_int_cyl)[2] + coef(mpg_disp_int_cyl)[6]

plot(mpg ~ disp, data = autompg, col = cyl)
abline(int_4cyl, slope_4cyl, col = 1, lty = 1, lwd = 2)
abline(int_6cyl, slope_6cyl, col = 2, lty = 2, lwd = 2)
abline(int_8cyl, slope_8cyl, col = 3, lty = 3, lwd = 2)
```



This looks much better! We can see that for medium displacement cars, 6 cylinder cars now perform better than 8 cylinder cars, which seems much more reasonable than before.

To completely justify the interaction model (i.e., a unique slope for each `cyl` level) compared to the additive model (single slope), we can perform an F-test. Notice first, that there is no t-test that will be able to do this since the difference between the two models is not a single parameter.

We will test,

$$H_0 : \gamma_2 = \gamma_3 = 0$$

which represents the parallel regression lines we saw before,

$$Y = \beta_0 + \beta_1 x + \beta_2 v_2 + \beta_3 v_3 + \epsilon.$$

Again, this is a difference of two parameters, thus no t-test will be useful.

```
anova(mpg_disp_add_cyl, mpg_disp_int_cyl)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ disp + cyl
## Model 2: mpg ~ disp * cyl
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    379 7299.5
## 2    377 6551.7  2    747.79 21.515 1.419e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As expected, we see a very low p-value, and thus reject the null. We prefer the interaction model over the additive model.

Recapping a bit:

- Null Model: $Y = \beta_0 + \beta_1 x + \beta_2 v_2 + \beta_3 v_3 + \epsilon$
    - Number of parameters: $q = 4$
- Full Model: $Y = \beta_0 + \beta_1 x + \beta_2 v_2 + \beta_3 v_3 + \gamma_2 x v_2 + \gamma_3 x v_3 + \epsilon$
    - Number of parameters: $p = 6$

```
length(coef(mpg_disp_int_cyl)) - length(coef(mpg_disp_add_cyl))
```

```
## [1] 2
```

We see there is a difference of two parameters, which is also displayed in the resulting ANOVA table from `R`. Notice that the following two values also appear on the ANOVA table.

```
nrow(autompg) - length(coef(mpg_disp_int_cyl))
```

```
## [1] 377
```

```
nrow(autompg) - length(coef(mpg_disp_add_cyl))
```

```
## [1] 379
```

6.4 Parameterization

So far we have been simply letting R decide how to create the dummy variables, and thus R has been deciding the parameterization of the models. To illustrate the ability to use alternative parameterizations, we will recreate the data, but directly creating the dummy variables ourselves.

```
new_param_data = data.frame(
  y = autompg$mpg,
  x = autompg$disp,
  v1 = 1 * as.numeric(autompg$cyl == 4),
  v2 = 1 * as.numeric(autompg$cyl == 6),
  v3 = 1 * as.numeric(autompg$cyl == 8))

head(new_param_data, 20)
```

```
##     y   x v1 v2 v3
## 1  18 307  0  0  1
## 2  15 350  0  0  1
## 3  18 318  0  0  1
## 4  16 304  0  0  1
## 5  17 302  0  0  1
## 6  15 429  0  0  1
## 7  14 454  0  0  1
## 8  14 440  0  0  1
## 9  14 455  0  0  1
## 10 15 390  0  0  1
## 11 15 383  0  0  1
## 12 14 340  0  0  1
## 13 15 400  0  0  1
## 14 14 455  0  0  1
## 15 24 113  1  0  0
## 16 22 198  0  1  0
## 17 18 199  0  1  0
## 18 21 200  0  1  0
## 19 27  97  1  0  0
## 20 26  97  1  0  0
```

Now,

- `y` is `mpg`
- `x` is `disp`, the displacement in cubic inches,
- `v1`, `v2`, and `v3` are dummy variables as defined above.

First let's try to fit an additive model using `x` as well as the three dummy variables.

```r
lm(y ~ x + v1 + v2 + v3, data = new_param_data)
```

```
##
## Call:
## lm(formula = y ~ x + v1 + v2 + v3, data = new_param_data)
##
## Coefficients:
## (Intercept)            x           v1           v2           v3
##    32.96326     -0.05217      2.03603     -1.59722           NA
```

What is happening here? Notice that R is essentially ignoring v3, but why? Well, because R uses an intercept, it cannot also use v3. This is because

$$\mathbf{1} = v_1 + v_2 + v_3$$

which means that $\mathbf{1}$, v_1, v_2, and v_3 are linearly dependent. This would make the $X^\top X$ matrix singular, but we need to be able to invert it to solve the normal equations and obtain $\hat{\beta}$. With the intercept, v1, and v2, R can make the necessary "three intercepts". So, in this case v3 is the reference level.

If we remove the intercept, then we can directly obtain all "three intercepts" without a reference level.

```r
lm(y ~ 0 + x + v1 + v2 + v3, data = new_param_data)
```

```
##
## Call:
## lm(formula = y ~ 0 + x + v1 + v2 + v3, data = new_param_data)
##
## Coefficients:
##        x        v1        v2        v3
## -0.05217  34.99929  31.36604  32.96326
```

Here, we are fitting the model

$$Y = \mu_1 v_1 + \mu_2 v_2 + \mu_3 v_3 + \beta x + \epsilon.$$

Thus we have:

- 4 Cylinder: $Y = \mu_1 + \beta x + \epsilon$
- 6 Cylinder: $Y = \mu_2 + \beta x + \epsilon$
- 8 Cylinder: $Y = \mu_3 + \beta x + \epsilon$

We could also do something similar with the interaction model, and give each line an intercept and slope, without the need for a reference level.

```r
lm(y ~ 0 + v1 + v2 + v3 + x:v1 + x:v2 + x:v3, data = new_param_data)
```

```
##
## Call:
## lm(formula = y ~ 0 + v1 + v2 + v3 + x:v1 + x:v2 + x:v3, data = new_param_data)
##
## Coefficients:
##       v1        v2        v3      v1:x      v2:x      v3:x
## 43.59052  30.39026  22.73346  -0.13069  -0.04770  -0.02252
```

$$Y = \mu_1 v_1 + \mu_2 v_2 + \mu_3 v_3 + \beta_1 x v_1 + \beta_2 x v_2 + \beta_3 x v_3 + \epsilon$$

- 4 Cylinder: $Y = \mu_1 + \beta_1 x + \epsilon$
- 6 Cylinder: $Y = \mu_2 + \beta_2 x + \epsilon$
- 8 Cylinder: $Y = \mu_3 + \beta_3 x + \epsilon$

Using the original data, we have (at least) three equivalent ways to specify the interaction model with `R`.

```
lm(mpg ~ disp * cyl, data = autompg)
```

```
##
## Call:
## lm(formula = mpg ~ disp * cyl, data = autompg)
##
## Coefficients:
## (Intercept)         disp         cyl6         cyl8    disp:cyl6    disp:cyl8
##    43.59052     -0.13069    -13.20026    -20.85706      0.08299      0.10817
```

```
lm(mpg ~ 0 + cyl + disp : cyl, data = autompg)
```

```
##
## Call:
## lm(formula = mpg ~ 0 + cyl + disp:cyl, data = autompg)
##
## Coefficients:
##      cyl4       cyl6       cyl8  cyl4:disp  cyl6:disp  cyl8:disp
##  43.59052   30.39026   22.73346   -0.13069   -0.04770   -0.02252
```

```
lm(mpg ~ 0 + disp + cyl + disp : cyl, data = autompg)
```

```
##
## Call:
## lm(formula = mpg ~ 0 + disp + cyl + disp:cyl, data = autompg)
##
## Coefficients:
##      disp       cyl4       cyl6       cyl8  disp:cyl6  disp:cyl8
##  -0.13069   43.59052   30.39026   22.73346    0.08299    0.10817
```

They all fit the same model, importantly each using six parameters, but the coefficients mean slightly different things in each. However, once they are interpreted as slopes and intercepts for the "three lines" they will have the same result.

Use `?all.equal` to learn about the `all.equal()` function, and think about how the following code verifies that the residuals of the two models are the same.

```
all.equal(fitted(lm(mpg ~ disp * cyl, data = autompg)),
          fitted(lm(mpg ~ 0 + cyl + disp : cyl, data = autompg)))
```

```
## [1] TRUE
```

6.5 Building Larger Models

Now that we have seen how to incorporate categorical predictors as well as interaction terms, we can start to build much larger, much more flexible models which can potentially fit data better.

Let's define a "big" model,

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1 x_2 + \beta_5 x_1 x_3 + \beta_6 x_2 x_3 + \beta_7 x_1 x_2 x_3 + \epsilon.$$

Here,

- Y is `mpg`.
- x_1 is `disp`.
- x_2 is `hp`.
- x_3 is `domestic`, which is a dummy variable we defined, where `1` is a domestic vehicle.

First thing to note here, we have included a new term $x_1 x_2 x_3$ which is a three-way interaction. Interaction terms can be larger and larger, up to the number of predictors in the model.

Since we are using the three-way interaction term, we also use all possible two-way interactions, as well as each of the first order (**main effect**) terms. This is the concept of a **hierarchy**. Any time a "higher-order" term is in a model, the related "lower-order" terms should also be included. Mathematically their inclusion or exclusion is sometimes irrelevant, but from an interpretation standpoint, it is best to follow the hierarchy rules.

Let's do some rearrangement to obtain a "coefficient" in front of x_1.

$$Y = \beta_0 + \beta_2 x_2 + \beta_3 x_3 + \beta_6 x_2 x_3 + (\beta_1 + \beta_4 x_2 + \beta_5 x_3 + \beta_7 x_2 x_3) x_1 + \epsilon.$$

Specifically, the "coefficient" in front of x_1 is

$$(\beta_1 + \beta_4 x_2 + \beta_5 x_3 + \beta_7 x_2 x_3).$$

Let's discuss this "coefficient" to help us understand the idea of the *flexibility* of a model. Recall that,

- β_1 is the coefficient for a first order term,
- β_4 and β_5 are coefficients for two-way interactions,
- β_7 is the coefficient for the three-way interaction.

If the two and three way interactions were not in the model, the whole "coefficient" would simply be

$$\beta_1.$$

Thus, no matter the values of x_2 and x_3, β_1 would determine the relationship between x_1 (`disp`) and Y (`mpg`).

With the addition of the two-way interactions, now the "coefficient" would be

$$(\beta_1 + \beta_4 x_2 + \beta_5 x_3).$$

Now, changing x_1 (`disp`) has a different effect on Y (`mpg`), depending on the values of x_2 and x_3.

Lastly, adding the three-way interaction gives the whole "coefficient"

$$(\beta_1 + \beta_4 x_2 + \beta_5 x_3 + \beta_7 x_2 x_3)$$

which is even more flexible. Now changing x_1 (`disp`) has a different effect on Y (`mpg`), depending on the values of x_2 and x_3, but in a more flexible way which we can see with some more rearrangement. Now the "coefficient" in front of x_3 in this "coefficient" is dependent on x_2.

$$(\beta_1 + \beta_4 x_2 + (\beta_5 + \beta_7 x_2) x_3)$$

It is so flexible, it is becoming hard to interpret!

Let's fit this three-way interaction model in `R`.

```
big_model = lm(mpg ~ disp * hp * domestic, data = autompg)
summary(big_model)
```

```
##
## Call:
## lm(formula = mpg ~ disp * hp * domestic, data = autompg)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.9410  -2.2147  -0.4008   1.9430  18.4094
##
## Coefficients:
##                     Estimate  Std. Error t value  Pr(>|t|)
## (Intercept)       60.6457838   6.6000851   9.189   < 2e-16 ***
## disp              -0.1415870   0.0634395  -2.232    0.0262 *
## hp                -0.3544717   0.0812261  -4.364 0.0000165 ***
## domestic         -12.5718884   7.0643505  -1.780    0.0759 .
## disp:hp            0.0013690   0.0006727   2.035    0.0426 *
## disp:domestic      0.0493298   0.0640046   0.771    0.4414
## hp:domestic        0.1851530   0.0870881   2.126    0.0342 *
## disp:hp:domestic  -0.0009163   0.0006768  -1.354    0.1766
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.88 on 375 degrees of freedom
## Multiple R-squared:   0.76,  Adjusted R-squared:  0.7556
## F-statistic: 169.7 on 7 and 375 DF,  p-value: < 2.2e-16
```

Do we actually need this large of a model? Let's first test for the necessity of the three-way interaction term. That is,

$$H_0 : \beta_7 = 0.$$

So,

- Full Model: $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1 x_2 + \beta_5 x_1 x_3 + \beta_6 x_2 x_3 + \beta_7 x_1 x_2 x_3 + \epsilon$
- Null Model: $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1 x_2 + \beta_5 x_1 x_3 + \beta_6 x_2 x_3 + \epsilon$

We fit the null model in `R` as `two_way_int_mod`, then use `anova()` to perform an F-test as usual.

```r
two_way_int_mod = lm(mpg ~ disp * hp + disp * domestic + hp * domestic, data = autompg)
anova(two_way_int_mod, big_model)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ disp * hp + disp * domestic + hp * domestic
## Model 2: mpg ~ disp * hp * domestic
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    376 5673.2
## 2    375 5645.6  1    27.599 1.8332 0.1766
```

We see the p-value is somewhat large, so we would fail to reject. We prefer the smaller, less flexible, null model, without the three-way interaction.

A quick note here: the full model does still "fit better." Notice that it has a smaller RMSE than the null model, which means the full model makes smaller (squared) errors on average.

```r
mean(resid(big_model) ^ 2)
```

```
## [1] 14.74053
```

```r
mean(resid(two_way_int_mod) ^ 2)
```

```
## [1] 14.81259
```

However, it is not much smaller. We could even say that, the difference is insignificant. This is an idea we will return to later in greater detail.

Now that we have chosen the model without the three-way interaction, can we go further? Do we need the two-way interactions? Let's test

$$H_0 : \beta_4 = \beta_5 = \beta_6 = 0.$$

Remember we already chose $\beta_7 = 0$, so,

- Full Model: $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1 x_2 + \beta_5 x_1 x_3 + \beta_6 x_2 x_3 + \epsilon$
- Null Model: $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon$

We fit the null model in `R` as `additive_mod`, then use `anova()` to perform an F-test as usual.

```r
additive_mod = lm(mpg ~ disp + hp + domestic, data = autompg)
anova(additive_mod, two_way_int_mod)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ disp + hp + domestic
## Model 2: mpg ~ disp * hp + disp * domestic + hp * domestic
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    379 7369.7
## 2    376 5673.2  3    1696.5 37.478 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here the p-value is small, so we reject the null, and we prefer the full (alternative) model. Of the models we have considered, our final preference is for

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1 x_2 + \beta_5 x_1 x_3 + \beta_6 x_2 x_3 + \epsilon.$$

Chapter 7

Model Diagnostics

"Your assumptions are your windows on the world. Scrub them off every once in a while, or the light won't come in."

— **Isaac Asimov**

After reading this chapter you will be able to:

- Understand the assumptions of a regression model.
- Assess regression model assumptions using visualizations and tests.
- Understand leverage, outliers, and influential points.
- Be able to identify unusual observations in regression models.

7.1 Model Assumptions

Recall the multiple linear regression model that we have defined.

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_{p-1} x_{i(p-1)} + \epsilon_i, \qquad i = 1, 2, \ldots, n.$$

Using matrix notation, this model can be written much more succinctly as

$$Y = X\beta + \epsilon.$$

Given data, we found the estimates for the β parameters using

$$\hat{\beta} = \left(X^\top X\right)^{-1} X^\top y.$$

We than noted that these estimates had mean

$$E[\hat{\beta}] = \beta,$$

and variance

$$Var[\hat{\beta}] = \sigma^2 \left(X^\top X\right)^{-1}.$$

In particular, an individual parameter, say $\hat{\beta}_j$ had a normal distribution

$$\hat{\beta}_j \sim N\left(\beta_j, \sigma^2 C_{jj}\right)$$

where C was the matrix defined as

$$C = \left(X^\top X\right)^{-1}.$$

We then used this fact to define

$$\frac{\hat{\beta}_j - \beta_j}{s_e \sqrt{C_{jj}}} \sim t_{n-p},$$

which we used to perform hypothesis testing.

So far we have looked at various metrics such as RMSE, RSE and R^2 to determine how well our model fit our data. Each of these in some way considers the expression

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

So, essentially each of these looks at how close the data points are to the model. However is that all we care about?

- It could be that the errors are made in a systematic way, which means that our model is misspecified. We may need additional interaction terms, or polynomial terms which we will see later.
- It is also possible that at a particular set of predictor values, the errors are very small, but at a different set of predictor values, the errors are large.
- Perhaps most of the errors are very small, but some are very large. This would suggest that the errors do not follow a normal distribution.

Are these issues that we care about? If all we would like to do is predict, possibly not, since we would only care about the size of our errors. However, if we would like to perform inference, for example to determine if a particular predictor is important, we care a great deal. All of the distributional results, such as a t-test for a single predictor, are derived under the assumptions of our model.

Technically, the assumptions of the model are encoded directly in a model statement such as,

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_{p-1} x_{i(p-1)} + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$.

Often, the **assumptions of linear regression**, are stated as,

- **L**inearity: the response can be written as a linear combination of the predictors. (With noise about this true linear relationship.)
- **I**ndependence: the errors are independent.
- **N**ormality: the distribution of the errors should follow a normal distribution.
- **E**qual Variance: the error variance is the same at any set of predictor values.

The linearity assumption is encoded as

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_{p-1} x_{i(p-1)},$$

while the remaining three, are all encoded in

$$\epsilon_i \sim N(0, \sigma^2),$$

since the ϵ_i are *iid* normal random variables with constant variance.

If these assumptions are met, great! We can perform inference, **and it is valid**. If these assumptions are *not* met, we can still "perform" a t-test using `R`, but the results are **not valid**. The distributions of the parameter estimates will not be what we expect. Hypothesis tests will then accept or reject incorrectly. Essentially, **garbage in, garbage out.**

7.2 Checking Assumptions

We'll now look at a number of tools for checking the assumptions of a linear model.

7.2.1 Fitted versus Residuals Plot

Probably our most useful tool will be a **Fitted versus Residuals Plot**. It will be useful for checking both the **linearity** and **constant variance** assumptions.

First, let's consider a true SLR model,

$$Y_i = 3 + 5x_i + \epsilon_i \qquad \epsilon_i \sim N(0, 1)$$

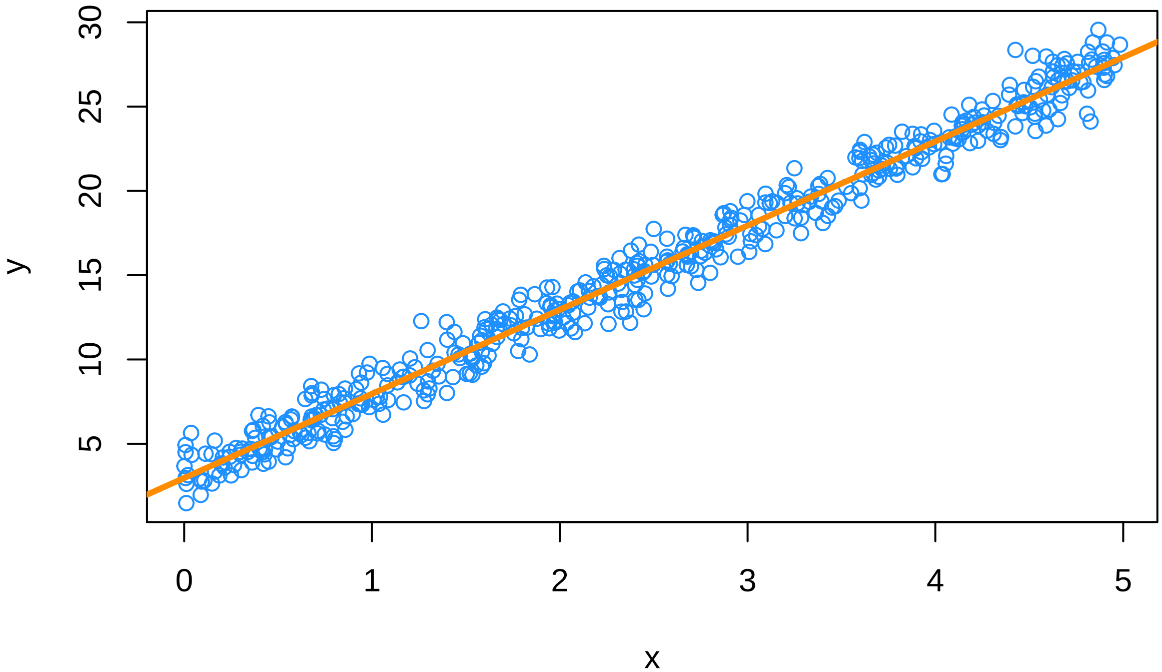
This model does not violate any the assumptions, so we'll use this to see what a good fitted versus residuals plot should look like. First, we'll simulate observations from this model.

```
n = 500
set.seed(42)
sim_data = data.frame(x = runif(n) * 5, y = rep(0, n))
sim_data$y = 3 + 5 * sim_data$x + rnorm(n, 0, 1)
head(sim_data)
```

```
##          x         y
## 1 4.574030 24.773995
## 2 4.685377 26.475936
## 3 1.430698  8.954993
## 4 4.152238 23.951210
## 5 3.208728 20.341344
## 6 2.595480 14.943525
```

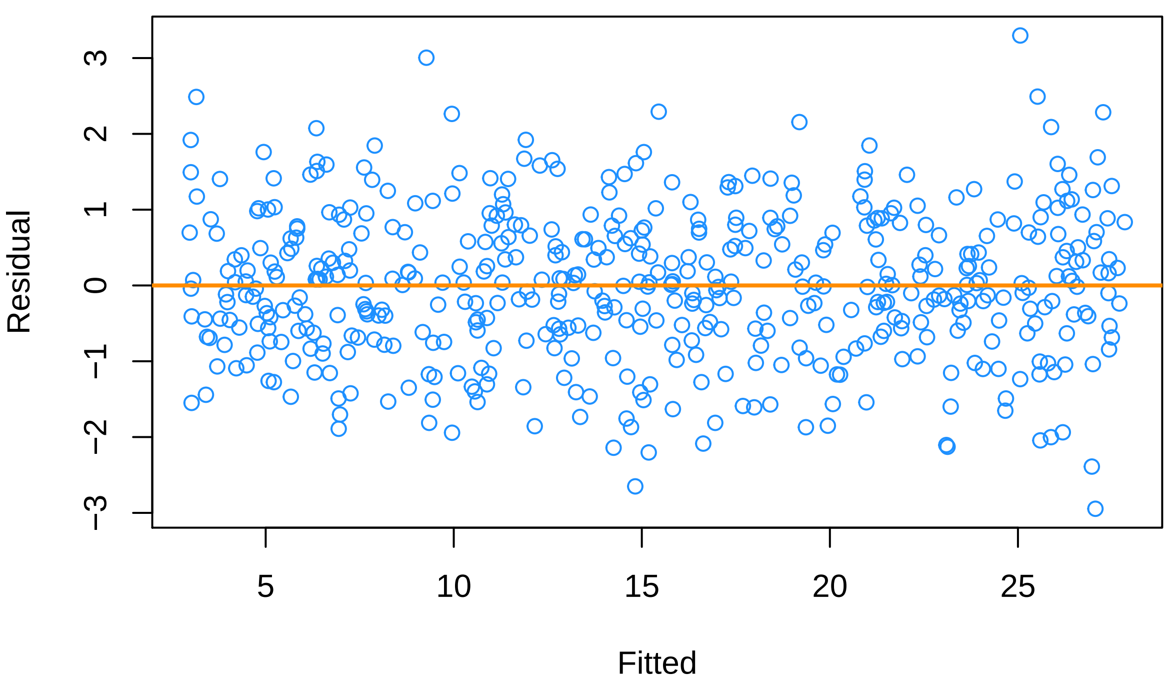
We then fit the model and add the fitted line to a scatterplot.

```
plot(y ~ x, data = sim_data, col = "dodgerblue")
fit1 = lm(y ~ x, data = sim_data)
abline(fit1, col = "darkorange", lwd = 3)
```



We now plot a fitted versus residuals plot. Note, this is residuals on the y-axis despite the ordering in the name. Sometimes you will see this called a residuals versus fitted, or residuals versus predicted plot.

```
plot(fitted(fit1), resid(fit1), col = "dodgerblue", xlab = "Fitted", ylab = "Residual")
abline(h = 0, col = "darkorange", lwd = 2)
```



We should look for two things in this plot.

- At any fitted value, the mean of the residuals should be roughly 0. If this is the case, the *linearity* assumption is valid. For this reason, we generally add a horizontal line at $y = 0$ to emphasize this point.

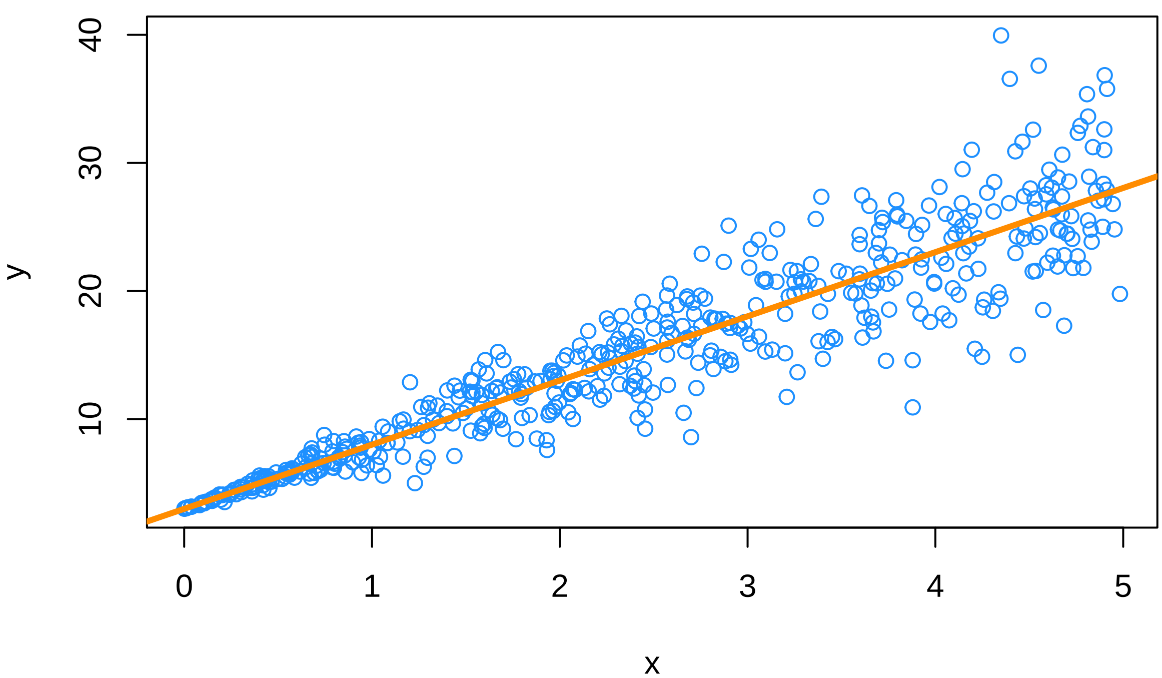
- At every fitted value, the spread of the residuals should be roughly the same. If this is the case, the *constant variance* assumption is valid.

Here we see this is the case for both.

To get a better idea of how a fitted versus residuals plot can be useful, we will simulate from models with violated assumptions.

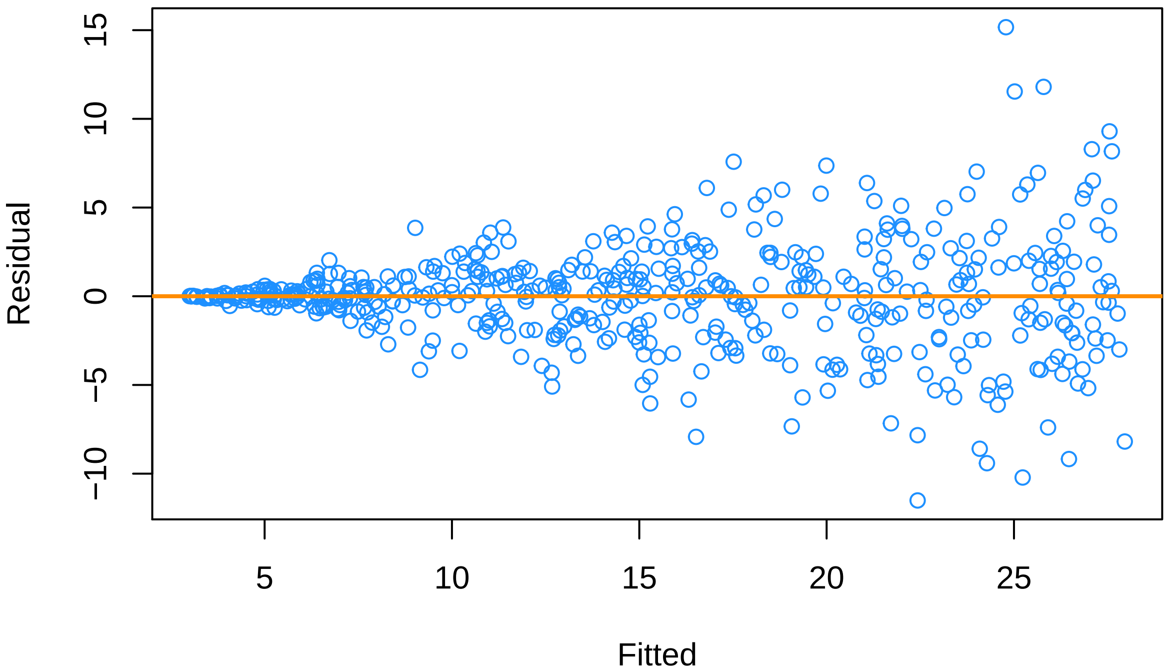
We'll first demonstrate a model with non-constant variance. In this case, the variance is larger for larger values of the predictor variable x.

```
sim_data2 = sim_data
sim_data2$y = 3 + 5 * sim_data2$x + rnorm(n, 0, sim_data2$x)
fit2 = lm(y ~ x, data = sim_data2)
plot(y ~ x, data = sim_data2, col = "dodgerblue")
abline(fit2, col = "darkorange", lwd = 3)
```



This actually is rather easy to see here by added the fitted line to a scatterplot. This is because we are only performing simple linear regression. With multiple regression, a fitted versus residuals plot is a necessity, since adding a fitted regression to a scatterplot isn't exactly possible.

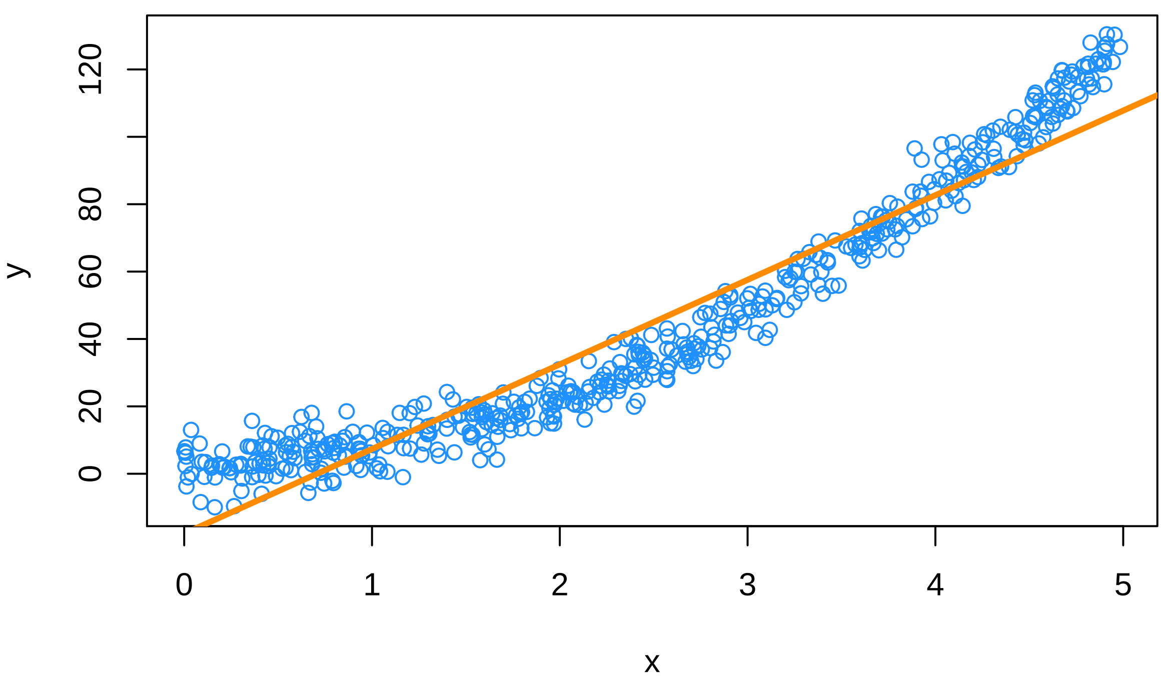
```
plot(fitted(fit2), resid(fit2), col = "dodgerblue", xlab = "Fitted", ylab = "Residual")
abline(h = 0, col = "darkorange", lwd = 2)
```



On the fitted versus residuals plot, we see two things very clearly. For any fitted value, the residuals seem roughly centered at 0. This is good! The linearity assumption is not violated. However, we also see very clearly, that for larger fitted values, the spread of the residuals is larger. This is bad! The constant variance assumption is violated here.

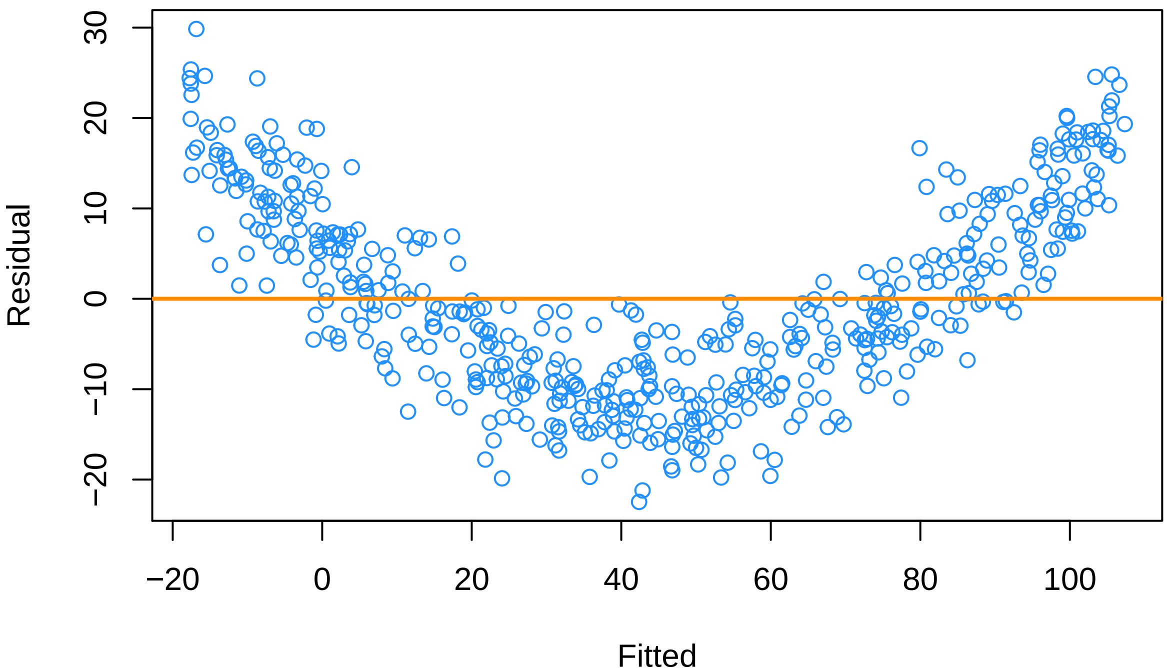
Now we will demonstrate a model which does not meet the linearity assumption.

```
sim_data3 = sim_data
sim_data3$y = 3 + 5 * sim_data3$x ^ 2 + rnorm(n, 0, 5)
fit3 = lm(y ~ x, data = sim_data3)
plot(y ~ x, data = sim_data3, col = "dodgerblue")
abline(fit3, col = "darkorange", lwd = 3)
```



Again, this is rather clear on the scatterplot, but again, we wouldn't be able to check this plot for multiple regression.

```
plot(fitted(fit3), resid(fit3), col = "dodgerblue", xlab = "Fitted", ylab = "Residual")
abline(h = 0, col = "darkorange", lwd = 2)
```



This time on the fitted versus residuals plot, for any fitted value, the spread of the residuals is about the same. However, they are not even close to centered at zero! At small and large fitted values the model is underestimating, while at medium fitted values, the model is overestimating. These are systematic errors, not random noise. So the constant variance assumption is met, but the linearity assumption is violated. Our model is simply wrong. We're trying to fit a line to a curve!

7.2.2 Breusch-Pagan Test

Constant variance is often called **homoscedasticity**. Conversely, non-constant variance is called **heteroscedasticity**. We've seen how we can use a fitted versus residuals plot to look for these attributes.

While a fitted versus residuals plot can give us an idea about homoscedasticity, sometimes we would prefer a more formal test. There are many tests for constant variance, but here we will present one, the **Breusch-Pagan Test**. The exact details of the test will omitted here, but importantly the null and alternative can be considered to be,

- H_0: Homoscedasticity. The errors have constant variance about the true model.
- H_1: Heteroscedasticity. The errors have non-constant variance about the true model.

Isn't that convenient? A test that will specifically test the **constant variance** assumption.

The Breusch-Pagan Test can not be performed by default in `R`, however the function `bptest` in the `lmtest` package implements the test.

```r
#install.packages("lmtest")
library(lmtest)
```

Let's try it on the three models we fit above. Recall,

- fit1 had no violation of assumptions,
- fit2 violated the constant variance assumption, but not linearity,
- fit3 violated linearity, but not constant variance.

```r
bptest(fit1)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  fit1
## BP = 1.0234, df = 1, p-value = 0.3117
```

For fit1 we see a large p-value, so we do not reject the null of homoscedasticity, which is what we would expect.

```r
bptest(fit2)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  fit2
## BP = 72.325, df = 1, p-value < 2.2e-16
```

For fit2 we see a small p-value, so we reject the null of homoscedasticity. The constant variance assumption is violated. This matches our findings with a fitted versus residuals plot.

```r
bptest(fit3)
```

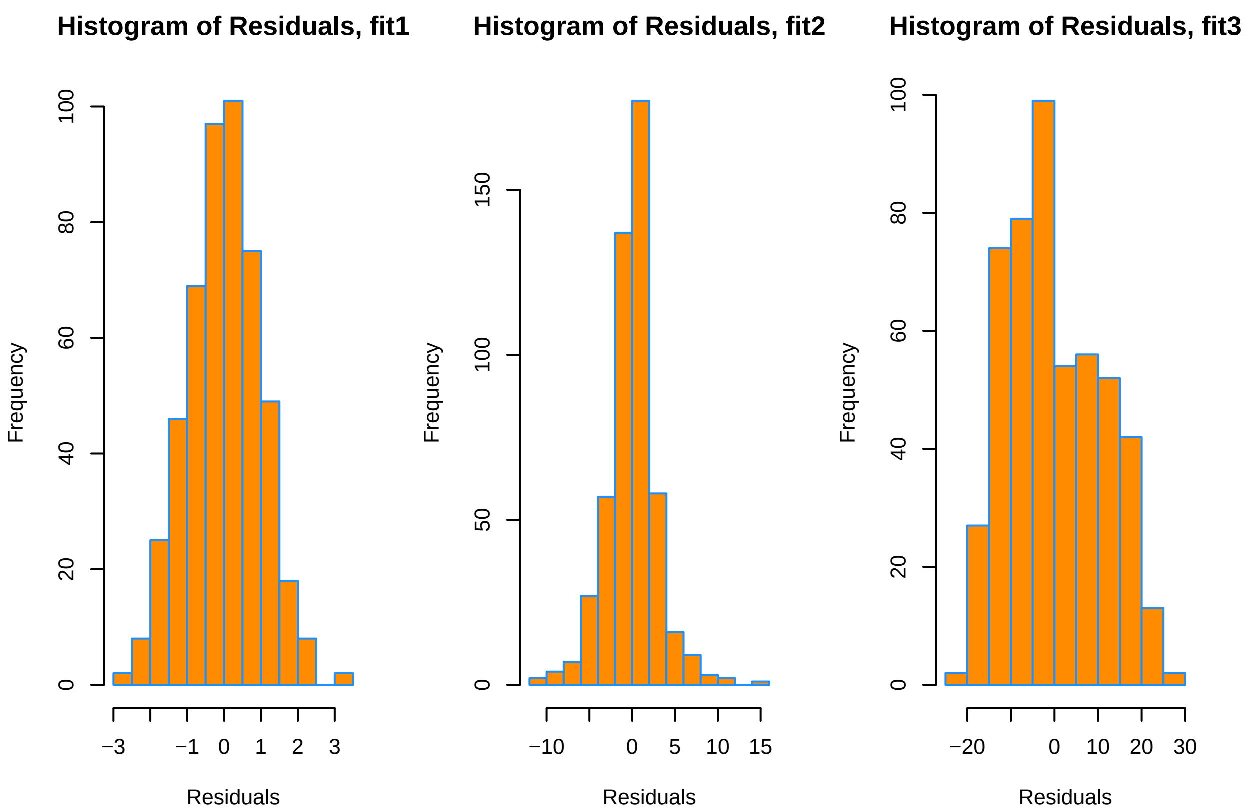
```
##
##  studentized Breusch-Pagan test
##
## data:  fit3
## BP = 0.24035, df = 1, p-value = 0.624
```

Lastly, for fit3 we again see a large p-value, so we do not reject the null of homoscedasticity, which matches our findings with a fitted versus residuals plot.

7.2.3 Histograms

We have a number of tools for assessing the normality assumption. The most obvious would be to make a histogram of the residuals. If it appears roughly normal, then we'll believe the errors could truly be normal.

```
par(mfrow = c(1, 3))
hist(resid(fit1),
     xlab   = "Residuals",
     main   = "Histogram of Residuals, fit1",
     col    = "darkorange",
     border = "dodgerblue")
hist(resid(fit2),
     xlab   = "Residuals",
     main   = "Histogram of Residuals, fit2",
     col    = "darkorange",
     border = "dodgerblue")
hist(resid(fit3),
     xlab   = "Residuals",
     main   = "Histogram of Residuals, fit3",
     col    = "darkorange",
     border = "dodgerblue")
```



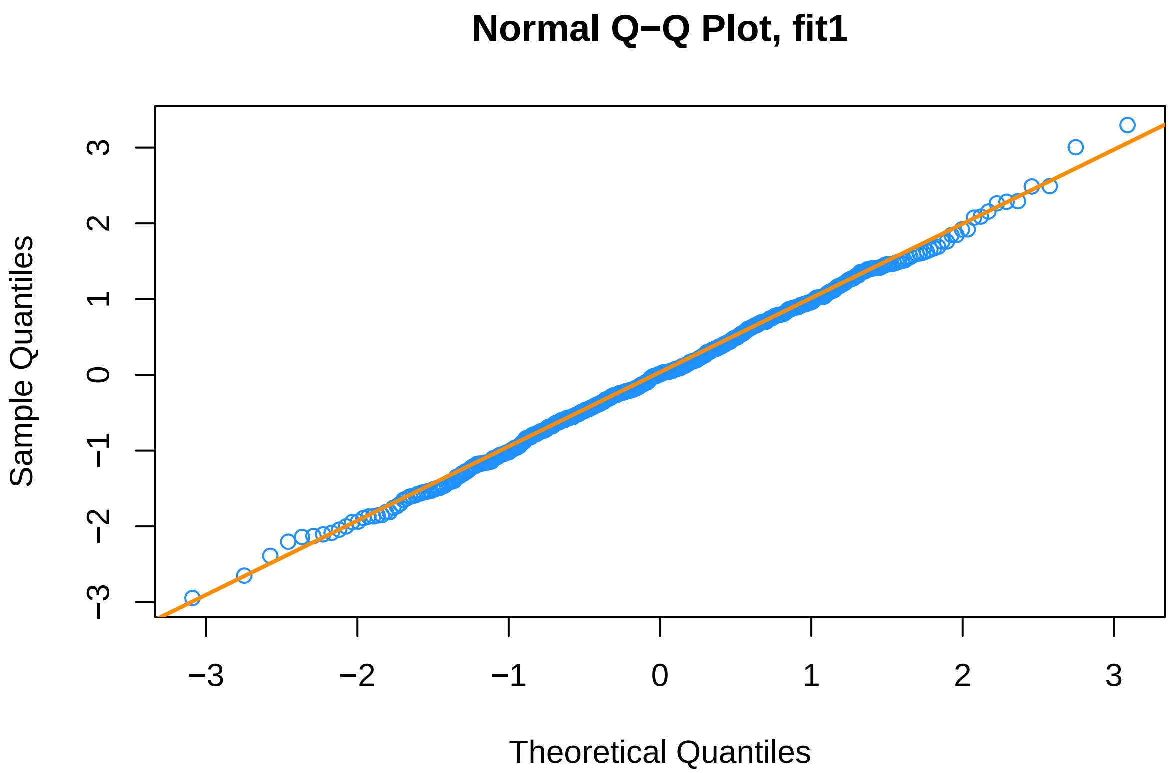
Above are histograms for each of the three regression we have been considering. Notice that the first, for `fit1` appears very normal. The third, for `fit3`, appears to be very non-normal. However `fit2` is not as clear. It does have a rough bell shape, however, it also has a very sharp peak. For this reason we will usually use more powerful tools such as **Q-Q plots** and the **Shapiro-Wilk test** for assessing the normality of errors.

7.2.4 Q-Q Plots

Another visual method for assessing the normality of errors, which is more powerful than a histogram, is a normal quantile-quantile plot, or **Q-Q plot** for short.

In R these are very easy to make. The `qqnorm()` function plots the points, and the `qqline()` function adds the necessary line. We create a Q-Q plot for the residuals of `fit1` to check if the errors could truly be normally distributed.

```
qqnorm(resid(fit1), main = "Normal Q-Q Plot, fit1", col = "dodgerblue")
qqline(resid(fit1), col = "darkorange", lwd = 2)
```



In short, if the points of the plot do not closely follow a straight line, this would suggest that the data do not come from a normal distribution.

The calculations required to create the plot vary depending on the implementation, but essentially the y-axis is the sorted data (observed, or sample quantiles), and the x-axis is the values we would expect if the data did come from a normal distribution (theoretical quantiles).

The Wikipedia page for Normal probability plots gives details on how this is implemented in R if you are interested.

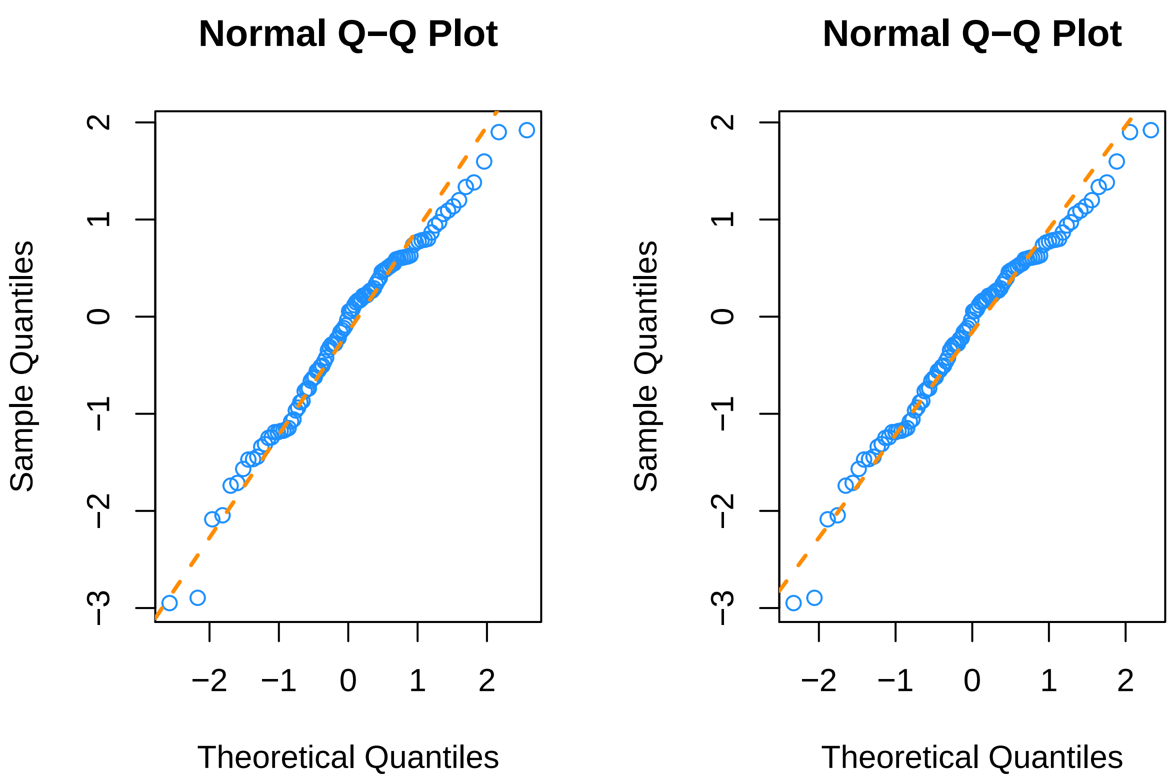
Also, to get a better idea of how Q-Q plots work, here is a quick function which creates a Q-Q plot:

```
qq_plot = function(w) {

  n = length(w)
  normal_quantiles = qnorm(((1:n) / (n + 1)))

  # plot theoretical verus observed quantiles
  plot(normal_quantiles, sort(w),
       xlab = c("Theoretical Quantiles"),
       ylab = c("Sample Quantiles"),
       col = "dodgerblue")
  title("Normal Q-Q Plot")
```

```
  ## calculate line through the first and third quartiles
  slope     = (quantile(w, 0.75) - quantile(w, 0.25)) / (qnorm(0.75) - qnorm(0.25))
  intercept = quantile(w, 0.25) - slope * qnorm(0.25)

  # add to existing plot
  abline(intercept, slope, lty = 2, lwd = 2, col = "darkorange")
}
```

We can then verify that it is essentially equivalent to using `qqnorm()` and `qqline()` in R. There are *slight* differences, but the general idea is the same.

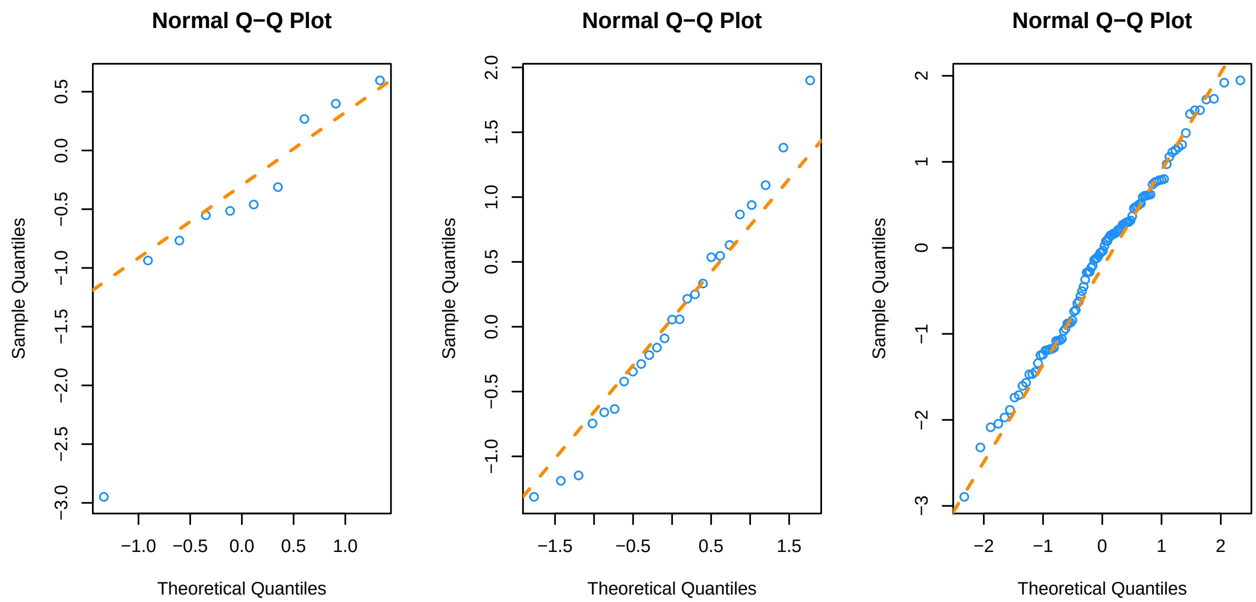
```
set.seed(420)
x = rnorm(100, mean = 0 , sd = 1)
par(mfrow = c(1, 2))
qqnorm(x, col = "dodgerblue")
qqline(x, lty = 2, lwd = 2, col = "darkorange")
qq_plot(x)
```



To get a better idea of what "close to the line" means, we perform a number of simulations, and create Q-Q plots.

First we simulate data from a normal distribution with different sample sizes, and each time create a Q-Q plot.

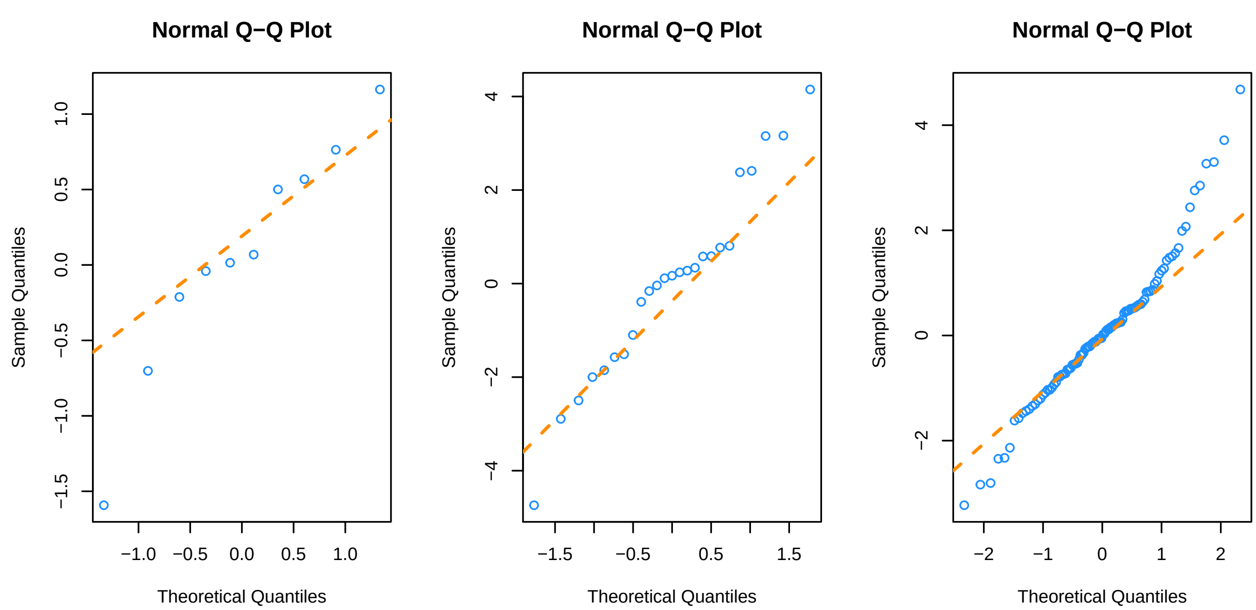
```
par(mfrow = c(1, 3))
set.seed(420)
qq_plot(rnorm(10))
qq_plot(rnorm(25))
qq_plot(rnorm(100))
```



Since this data **is** sampled from a normal distribution, these are all, by definition, good Q-Q plots. The points are "close to the line" and we would conclude that this data could have been sampled from a normal distribution. Notice in the first plot, one point is *somewhat* far from the line, but just one point, in combination with the small sample size, is not enough to make us worried. We see with the large sample size, all of the points are rather close to the line.

Next, we simulate data from a t distribution with a small degrees of freedom, for different sample sizes.

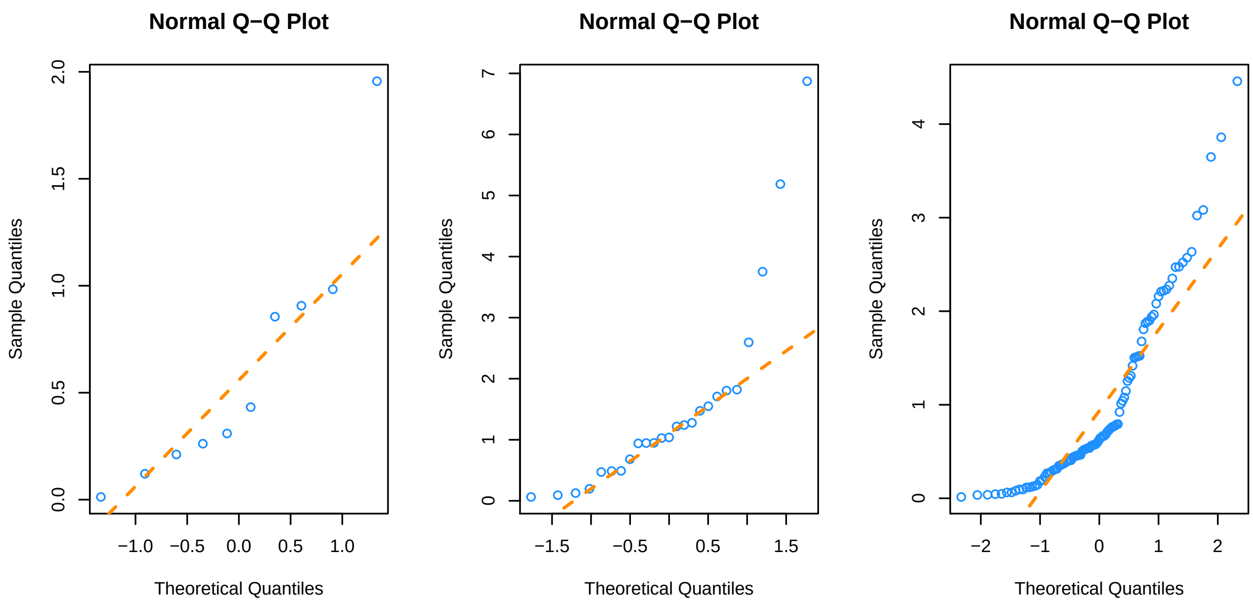
```r
par(mfrow = c(1, 3))
set.seed(420)
qq_plot(rt(10, df = 4))
qq_plot(rt(25, df = 4))
qq_plot(rt(100, df = 4))
```



Recall, that as the degrees of freedom for a t distribution become larger, the distribution becomes more and more similar to a normal. Here, using 4 degrees of freedom, we have a distribution that is somewhat normal, it is symmetrical and roughly bell-shaped, however it has "fat tails." This presents itself clearly in the third panel. While many of the points are close to the line, at the edges, there are large discrepancies. This indicates that the values are to small (negative) or too large (positive) compared to what we would expect for a normal distribution. So for the sample size of `100`, we would conclude that that normality assumption is violated. (If these were residuals of a model.) For sample sizes of `10` and `25` we may be suspicious, but not entirely confident. Reading Q-Q plots, is a bit of an art, not completely a science.

Next, we simulate data from an exponential distribution.

```
par(mfrow = c(1, 3))
set.seed(420)
qq_plot(rexp(10))
qq_plot(rexp(25))
qq_plot(rexp(100))
```



This is a distribution that is not very similar to a normal, so in all three cases, we see points that are far from the lines, so we would think that the normality assumption is violated.

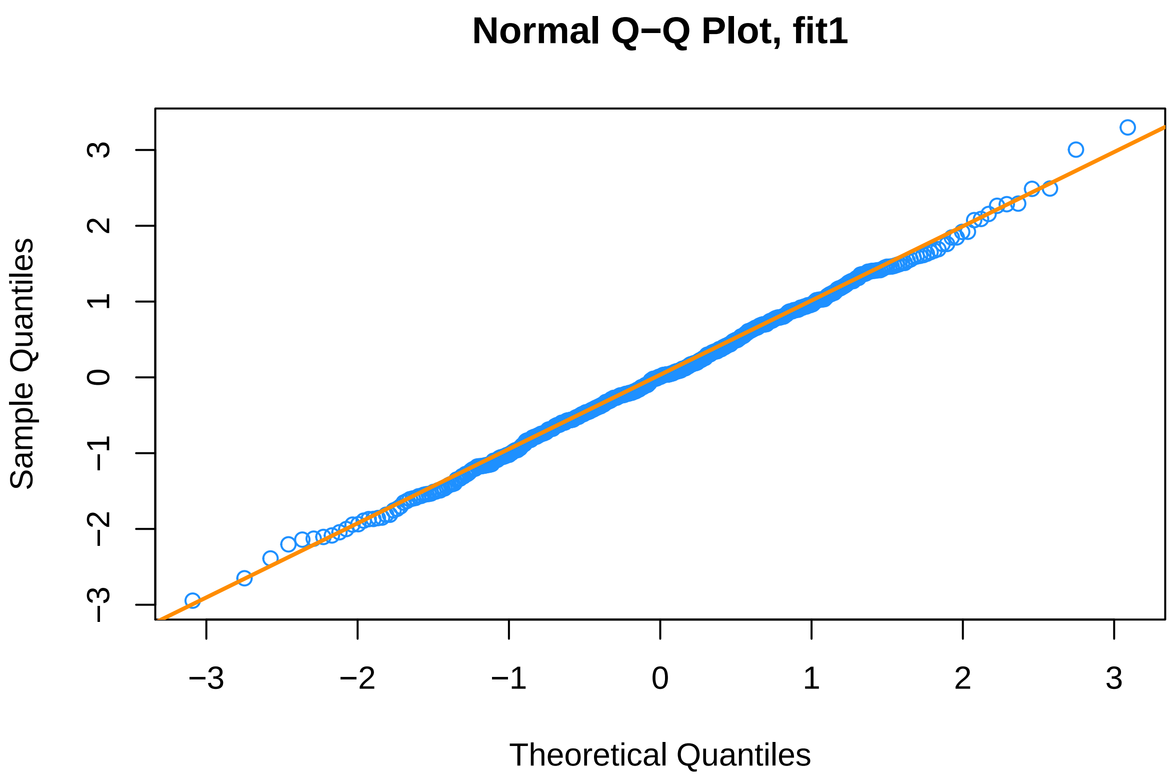
For a better understanding of which Q-Q plots are "good," repeat the simulations above a number of times (without setting the seed) and pay attention to the differences between those that are simulated from normal, and those that are not. Also consider different samples sizes and distribution parameters.

Returning to our three regressions, recall,

- `fit1` had no violation of assumptions,
- `fit2` violated the constant variance assumption, but not linearity,
- `fit3` violated linearity, but not constant variance.

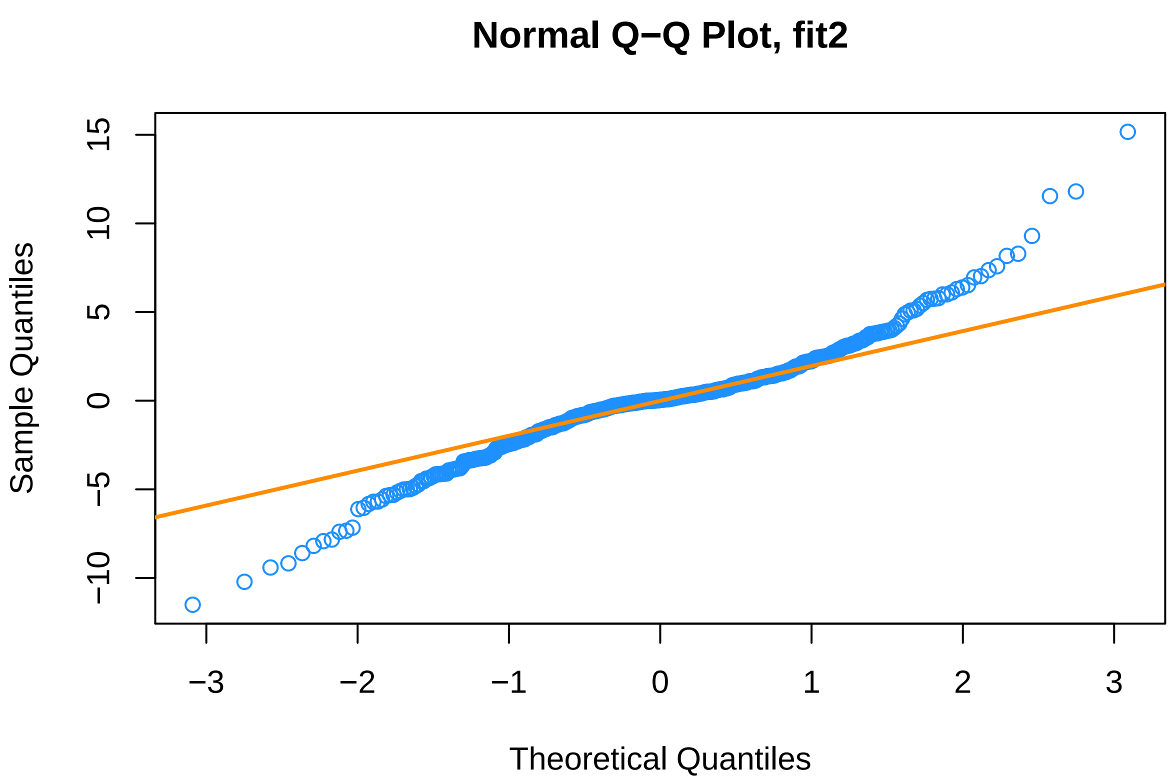
We'll now create a Q-Q plot for each to asses normality of errors.

```
qqnorm(resid(fit1), main = "Normal Q-Q Plot, fit1", col = "dodgerblue")
qqline(resid(fit1), col = "darkorange", lwd = 2)
```



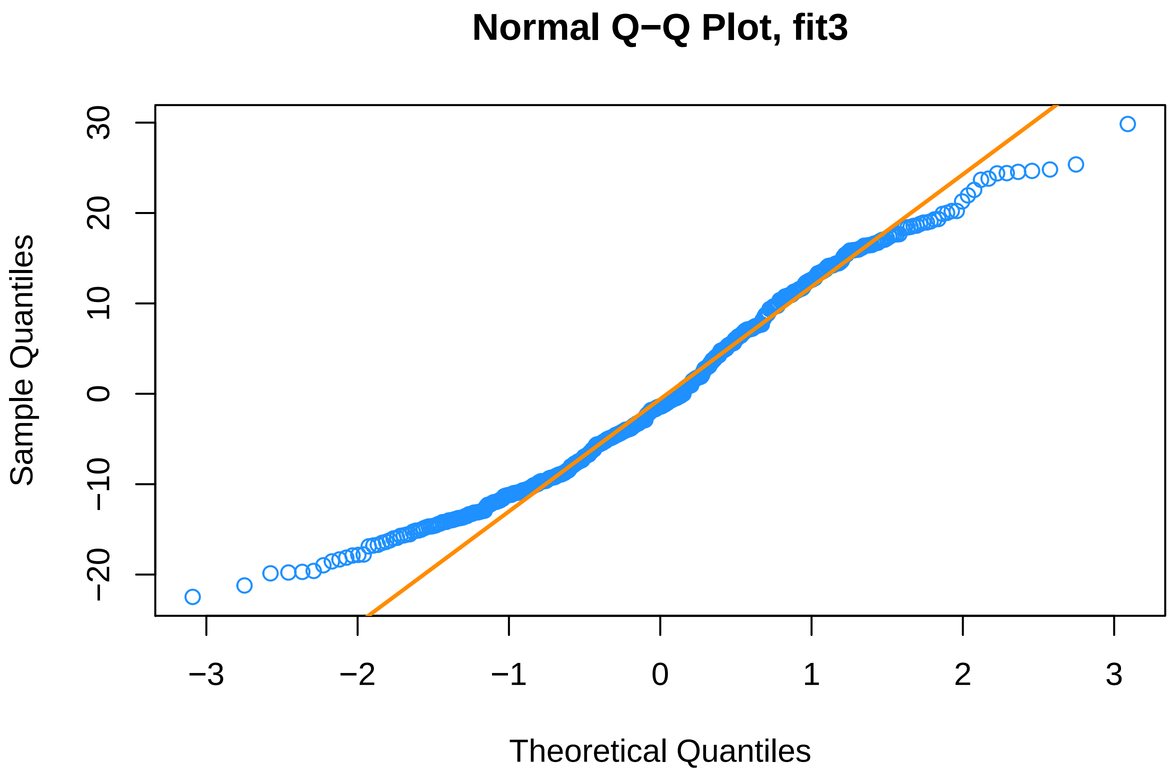
For `fit1`, we have a near perfect Q-Q plot. We would believe the errors follow a normal distribution.

```
qqnorm(resid(fit2), main = "Normal Q-Q Plot, fit2", col = "dodgerblue")
qqline(resid(fit2), col = "darkorange", lwd = 2)
```



For `fit2`, we have a suspect Q-Q plot. We would probably **not** believe the errors follow a normal distribution.

```
qqnorm(resid(fit3), main = "Normal Q-Q Plot, fit3", col = "dodgerblue")
qqline(resid(fit3), col = "darkorange", lwd = 2)
```



Lastly, for `fit3`, we again have a suspect Q-Q plot. We would probably **not** believe the errors follow a normal distribution.

7.2.5 Shapiro-Wilk Test

Histograms and Q-Q Plots give a nice visual representation of the residuals distribution, however if we are interested in formal testing, there are a number of options available. A commonly used test is the **Shapiro–Wilk test**, which is implemented in `R`.

```
set.seed(42)
shapiro.test(rnorm(25))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  rnorm(25)
## W = 0.9499, p-value = 0.2495
```

```
shapiro.test(rexp(25))
```

```
##
##  Shapiro-Wilk normality test
```

```
##
## data:  rexp(25)
## W = 0.71164, p-value = 0.0000105
```

This gives us the value of the test statistic and its p-value. The null hypothesis assumes the data follow a normal distribution, thus a small p-value indicates we believe there is only a small probability the data follow a normal distribution.

For details, see: Wikipedia: Shapiro–Wilk test.

In the above examples, we see we fail to reject for the data sampled from normal, and reject on the non-normal data.

Returning again to `fit1`, `fit2` and `fit3`, we see the result of running `shapiro.test()` on the residuals of each, returns a result for each that matches for decisions based on the Q-Q plots.

```
shapiro.test(resid(fit1))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(fit1)
## W = 0.99858, p-value = 0.9622
```

```
shapiro.test(resid(fit2))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(fit2)
## W = 0.94799, p-value = 3.002e-12
```

```
shapiro.test(resid(fit3))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(fit3)
## W = 0.97432, p-value = 0.000000109
```

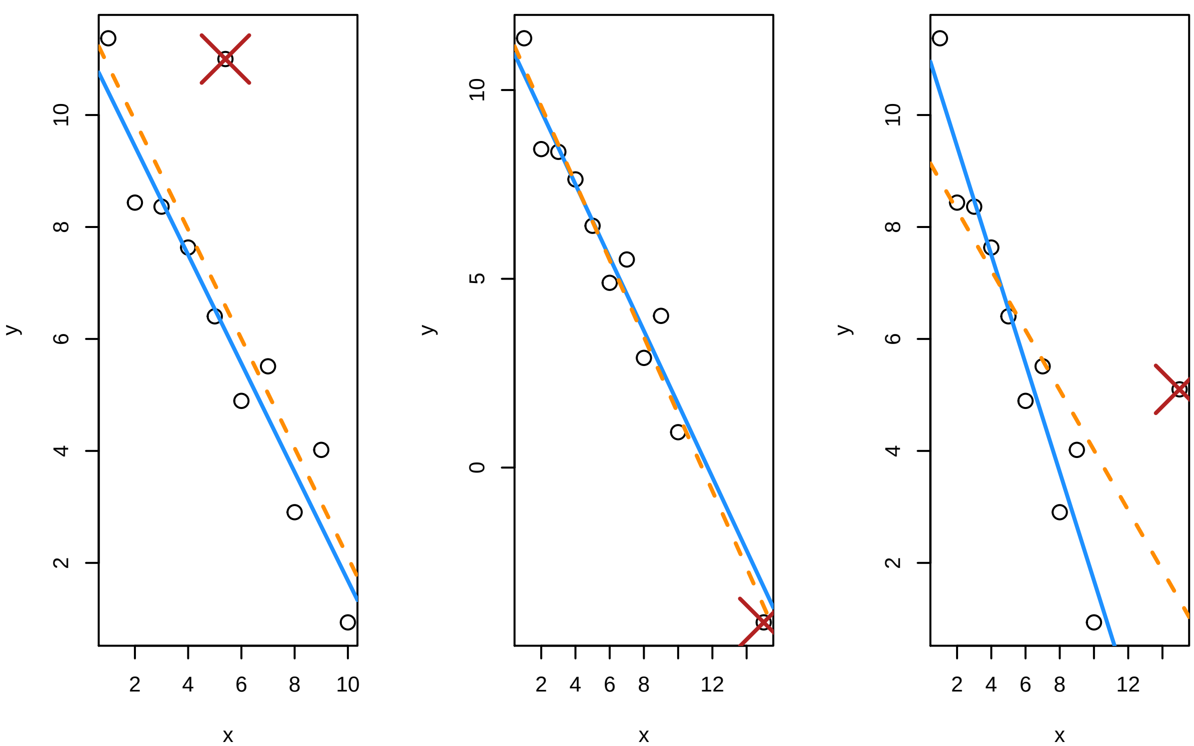
7.3 Unusual Observations

In addition to checking the assumptions of regression, we also look for any "unusual observations" in the data. Often a small number of data points can have an extremely large influence on a regression, sometimes so much so that the regression assumptions are violated as a result of these points.

The following three plots are inspired by an example from Linear Models with R.

```
par(mfrow = c(1, 3))
set.seed(42)
ex_data  = data.frame(x = 1:10, y = 10:1 + rnorm(10))
ex_model = lm(y ~ x, data = ex_data)
```

```
# low leverage, yes outlier, small influence
point1 = c(5.4, 11)
model1 = lm(y ~ x, data = rbind(ex_data, point1))
plot(y ~ x, data = rbind(ex_data, point1), cex = 1.5)
points(x = point1[1], y = point1[2], pch = 4, cex = 5, col = "firebrick", lwd = 2)
abline(ex_model, col = "dodgerblue", lwd = 2)
abline(model1, lty = 2, col = "darkorange", lwd = 2)

# high leverage, not outlier, low influence
point2 = c(15, -4.1)
model2 = lm(y ~ x, data = rbind(ex_data, point2))
plot(y ~ x, data = rbind(ex_data, point2), cex = 1.5)
points(x = point2[1], y = point2[2], pch = 4, cex = 5, col = "firebrick", lwd = 2)
abline(ex_model, col = "dodgerblue", lwd = 2)
abline(model2, lty = 2, col = "darkorange", lwd = 2)

# high leverage, yes outlier, large influence
point3 = c(15, 5.1)
model3 = lm(y ~ x, data = rbind(ex_data, point3))
plot(y ~ x, data = rbind(ex_data, point3), cex = 1.5)
points(x = point3[1], y = point3[2], pch = 4, cex = 5, col = "firebrick", lwd = 2)
abline(ex_model, col = "dodgerblue", lwd = 2)
abline(model3, lty = 2, col = "darkorange", lwd = 2)
```



The blue solid line in each plot is the regression fitted to the 10 original data points stored in `ex_data`. The dashed orange line in each plot is the result of adding a single point to the original data in `ex_data`. This additional point is indicated by the large red "X" in each plot.

The slope of the regression for the original ten points, the solid blue line, is given by:

```r
coef(ex_model)[2]
```

```
##          x
## -0.9696033
```

The added point in the first plot has a *small* effect on the slope, which becomes:

```r
coef(model1)[2]
```

```
##          x
## -0.9749534
```

We will say that this point has low leverage, is an outlier due to its large residual, but has small influence.

The added point in the second plot also has a *small* effect on the slope, which is:

```r
coef(model2)[2]
```

```
##         x
## -1.018734
```

We will say that this point has high leverage, is not an outlier due to its small residual, and has small influence.

Lastly, the added point in the third plot has a *large* effect on the slope, which is now:

```r
coef(model3)[2]
```

```
##          x
## -0.5358609
```

This added point is influential. It both has high leverage, and is an outlier due to its large residual.

We've now mentioned three new concepts: leverage, outliers, and influential points, each of which we will discuss in detail.

7.3.1 Leverage

A data point with high **leverage**, is a data point that *could* have a large influence when fitting the model.

Recall that,

$$\hat{\beta} = \left(X^\top X\right)^{-1} X^\top y.$$

Thus,

$$\hat{y} = X\hat{\beta} = X\left(X^\top X\right)^{-1} X^\top y$$

Now we define,

$$H = X \left(X^\top X\right)^{-1} X^\top$$

which we will refer to as the *hat matrix*. The hat matrix is used to project onto the subspace spanned by the columns of X. It is also simply known as a projection matrix.

The hat matrix, is a matrix that takes the original y values, and adds a hat!

$$\hat{y} = Hy$$

The diagonal elements of this matrix are called the **leverages**

$$H_{ii} = h_i,$$

where h_i is the leverage for the ith observation.

Large values of h_i indicate extreme values in X, which may influence regression. Note that leverages only depend on X.

Here, p the number of βs is also the trace (and rank) of the hat matrix.

$$\sum_{i=1}^{n} h_i = p$$

What is a value of h_i that would be considered large? There is no exact answer to this question. A common heuristic would be to compare each leverage to two times the average leverage. A leverage larger than this is considered an observation to be aware of. That is, if

$$h_i > 2\bar{h}$$

we say that observation i has large leverage. Here,

$$\bar{h} = \frac{\sum_{i=1}^{n} h_i}{n} = \frac{p}{n}.$$

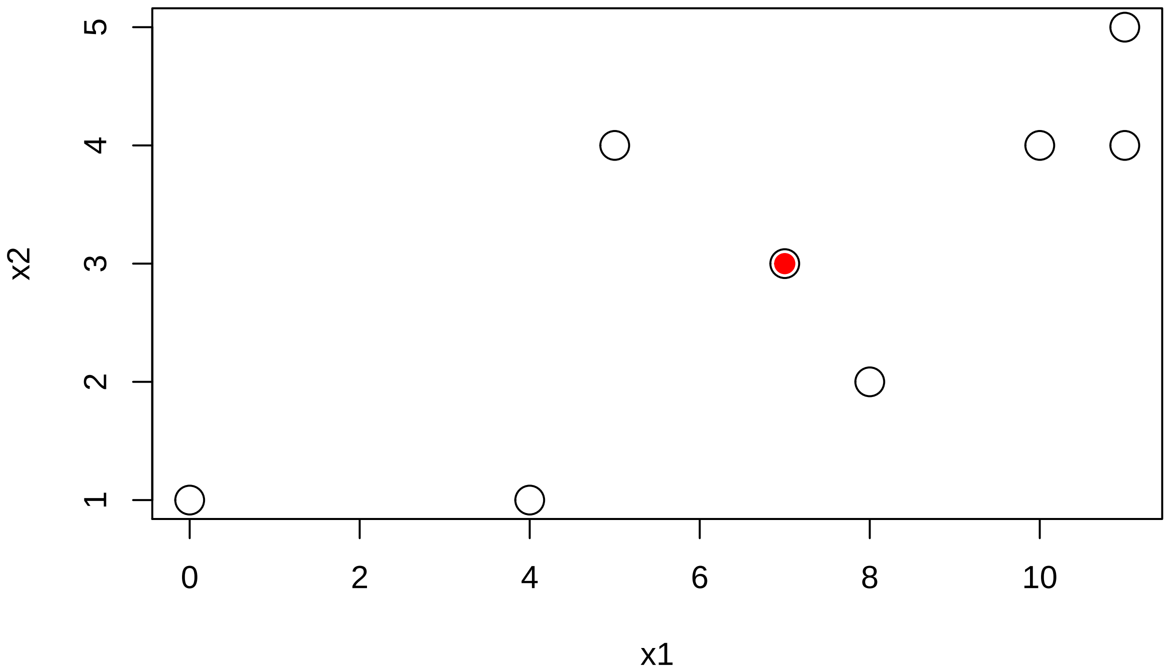
For simple linear regression, the leverage for each point is given by

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{S_{xx}}.$$

This expression should be familiar. (Think back to inference for SLR.) It suggests that the large leverages occur when x values are far from their mean. Recall that the regression goes through the point $(\bar{x}, \bar{y})$.

There are multiple ways to find leverages in `R`.

```
lev_ex = data.frame(
  x1 = c(0, 11, 11, 7, 4, 10, 5, 8),
  x2 = c(1, 5, 4, 3, 1, 4, 4, 2),
  y  = c(11, 15, 13, 14, 0, 19, 16, 8))

plot(x2 ~ x1, data = lev_ex, cex = 2)
points(7, 3, pch = 20, col = "red", cex = 2)
```



Here we've created some multivariate data. Notice that we have plotted the x values, not the y values. The red point is $(7, 3)$ which is the mean of `x1` and the mean of `x2` respectively.

We could calculate the leverages using the expressions defined above. We first create the X matrix, then calculate H as defined, and extract the diagonal elements.

```
X = cbind(rep(1, 8), lev_ex$x1, lev_ex$x2)
H = X %*% solve(t(X) %*% X) %*% t(X)
diag(H)
```

```
## [1] 0.6000 0.3750 0.2875 0.1250 0.4000 0.2125 0.5875 0.4125
```

Notice here, we have two predictors, so the regression would have 3 β parameters, so the sum of the diagonal elements is 3.

```
sum(diag(H))
```

```
## [1] 3
```

Alternatively, the method we will use more often, is to simply fit a regression, then use the `hatvalues()` function, which returns the leverages.

```
lev_fit = lm(y ~ ., data = lev_ex)
hatvalues(lev_fit)
```

```
##      1      2      3      4      5      6      7      8
## 0.6000 0.3750 0.2875 0.1250 0.4000 0.2125 0.5875 0.4125
```

Again, note that here we have "used" the y values to fit the regression, but `R` still ignores them when calculating the leverages, as leverages only depend on the x values.

```r
coef(lev_fit)
```

```
## (Intercept)          x1          x2
##         3.7        -0.7         4.4
```

Let's see what happens to these coefficients when we modify the `y` value of the point with the highest leverage.

```r
which.max(hatvalues(lev_fit))
```

```
## 1
## 1
```

```r
lev_ex[which.max(hatvalues(lev_fit)),]
```

```
##   x1 x2  y
## 1  0  1 11
```

We see that the original `y` value is 11. We'll create a copy of the data, and modify this point to have a `y` value of `20`.

```r
lev_ex_1 = lev_ex
lev_ex_1$y[1] = 20
lm(y ~ ., data = lev_ex_1)
```

```
##
## Call:
## lm(formula = y ~ ., data = lev_ex_1)
##
## Coefficients:
## (Intercept)           x1           x2
##       8.875       -1.375        4.625
```

Notice the **large** changes in the coefficients. Also notice that each of the coefficients has changed in some way. Note that the leverages of the points would not have changed, as we have not modified any of the x values.

Now let's see what happens to these coefficients when we modify the `y` value of the point with the lowest leverage.

```r
which.min(hatvalues(lev_fit))
```

```
## 4
## 4
```

```r
lev_ex[which.min(hatvalues(lev_fit)),]
```

```
##   x1 x2  y
## 4  7  3 14
```

We see that the original `y` value is 14. We'll again create a copy of the data, and modify this point to have a `y` value of `30`.

```r
lev_ex_2 = lev_ex
lev_ex_2$y[4] = 30
lm(y ~ ., data = lev_ex_2)
```

```
##
## Call:
## lm(formula = y ~ ., data = lev_ex_2)
##
## Coefficients:
## (Intercept)           x1           x2
##         5.7         -0.7          4.4
```

This time despite a large change in the y value, there is only small change in the coefficients. Also, only the intercept has changed!

```r
mean(lev_ex$x1)
```

```
## [1] 7
```

```r
mean(lev_ex$x2)
```

```
## [1] 3
```

```r
lev_ex[4,]
```

```
##   x1 x2  y
## 4  7  3 14
```

Notice that this point was the mean of both of the predictors.

Returning to our three plots, each with an added point, we can calculate the leverages for each. Note that the 11th data point each time is the added data point.

```r
hatvalues(model1)
```

```
##          1          2          3          4          5          6          7
## 0.33534597 0.23860732 0.16610842 0.11784927 0.09382988 0.09405024 0.11851036
##          8          9         10         11
## 0.16721022 0.24014985 0.33732922 0.09100926
```

```r
hatvalues(model2)
```

```
##          1          2          3          4          5          6          7
## 0.26574586 0.20662983 0.15966851 0.12486188 0.10220994 0.09171271 0.09337017
##          8          9         10         11
## 0.10718232 0.13314917 0.17127072 0.54419890
```

```r
hatvalues(model3)
```

```
##          1          2          3          4          5          6          7
## 0.26574586 0.20662983 0.15966851 0.12486188 0.10220994 0.09171271 0.09337017
##          8          9         10         11
## 0.10718232 0.13314917 0.17127072 0.54419890
```

Are any of these large?

```r
hatvalues(model1) > 2 * mean(hatvalues(model1))
```

```
##     1     2     3     4     5     6     7     8     9    10    11
## FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```r
hatvalues(model2) > 2 * mean(hatvalues(model2))
```

```
##     1     2     3     4     5     6     7     8     9    10    11
## FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```

```r
hatvalues(model3) > 2 * mean(hatvalues(model3))
```

```
##     1     2     3     4     5     6     7     8     9    10    11
## FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```

We see that in the second and third plots, the added point is a point of high leverage. Recall that only in the third plot did that have an influence on the regression. To understand why, we'll need to discuss outliers.

7.3.2 Outliers

Outliers are points which do not fit the model well. They may or may not have a large affect on the model. To identify outliers, we will look for observations with large residuals.

Note,

$$e = y - \hat{y} = Iy - Hy = (I - H)y$$

Then, under the assumptions of linear regression,

$$Var(e_i) = (1 - h_i)\sigma^2$$

and thus estimating σ^2 with s_e^2 gives

$$SE[e_i] = s_e\sqrt{(1 - h_i)}.$$

We can then look at the **standardized residual** for each observation, $i = 1, 2, \ldots n$,

$$r_i = \frac{e_i}{s_e\sqrt{1 - h_i}} \overset{approx}{\sim} N(\mu = 0, \sigma^2 = 1)$$

when n is large.

We can use this fact to identify "large" residuals. For example, standardized residuals greater than 2 in magnitude should only happen approximately 5 percent of the time.

Returning again to our three plots, each with an added point, we can calculate the residuals and standardized residuals for each. Standardized residuals can be obtained in `R` by using `rstandard()` where we would normally use `resid()`.

```
resid(model1)
```

```
##          1          2          3          4          5          6          7
##  0.4949887 -1.4657145 -0.5629345 -0.3182468 -0.5718877 -1.1073271  0.4852728
##          8          9         10         11
## -1.1459548  0.9420814 -1.1641029  4.4138254
```

```
rstandard(model1)
```

```
##          1          2          3          4          5          6          7
##  0.3464701 -0.9585470 -0.3517802 -0.1933575 -0.3428264 -0.6638841  0.2949482
##          8          9         10         11
## -0.7165857  0.6167268 -0.8160389  2.6418234
```

```
rstandard(model1)[abs(rstandard(model1)) > 2]
```

```
##       11
## 2.641823
```

In the first plot, we see that the 11th point, the added point, is a large standardized residual.

```
resid(model2)
```

```
##          1          2          3          4          5          6          7
##  0.7820254 -1.1348974 -0.1883370  0.1001310 -0.1097294 -0.6013884  1.0349919
##          8          9         10         11
## -0.5524553  1.5793613 -0.4830427 -0.4266595
```

```
rstandard(model2)
```

```
##          1          2          3          4          5          6          7
##  1.0653318 -1.4873251 -0.2398267  0.1249447 -0.1351833 -0.7365982  1.2688468
##          8          9         10         11
## -0.6825005  1.9801429 -0.6193932 -0.7377020
```

```
rstandard(model2)[abs(rstandard(model2)) > 2]
```

```
## named numeric(0)
```

In the second plot, we see that there are no points with large standardized residuals.

```r
resid(model3)
```

```
##          1          2          3          4          5          6          7
##  2.5356166  0.1358208  0.5995083  0.4051034 -0.2876300 -1.2621619 -0.1086545
##          8          9         10         11
## -2.1789746 -0.5300310 -3.0753079  3.7667107
```

```r
rstandard(model3)
```

```
##           1           2           3           4           5           6
##  1.45289045  0.07486882  0.32110154  0.21261803 -0.14904562 -0.65024338
##           7           8           9          10          11
## -0.05602801 -1.13225246 -0.27951271 -1.65865000  2.73934714
```

```r
rstandard(model3)[abs(rstandard(model3)) > 2]
```

```
##       11
## 2.739347
```

In the last plot, we see that the 11th point, the added point, is a large standardized residual.

Recall that the added point in plots two and three were both high leverage, but now only the point in plot three has a large residual. We will now combine this information and discuss influence.

7.3.3 Influence

As we have now seen in the three plots, some outliers only change the regression a small amount (plot one) and some outliers have a large effect on the regression (plot three). Observations that fall into the later category, points of *high leverage* **and** *large residual*, we will call **influential**.

A common measure of influence is **Cook's Distance**, which is defined as

$$D_i = \frac{1}{p} r_i^2 \frac{h_i}{1 - h_i}.$$

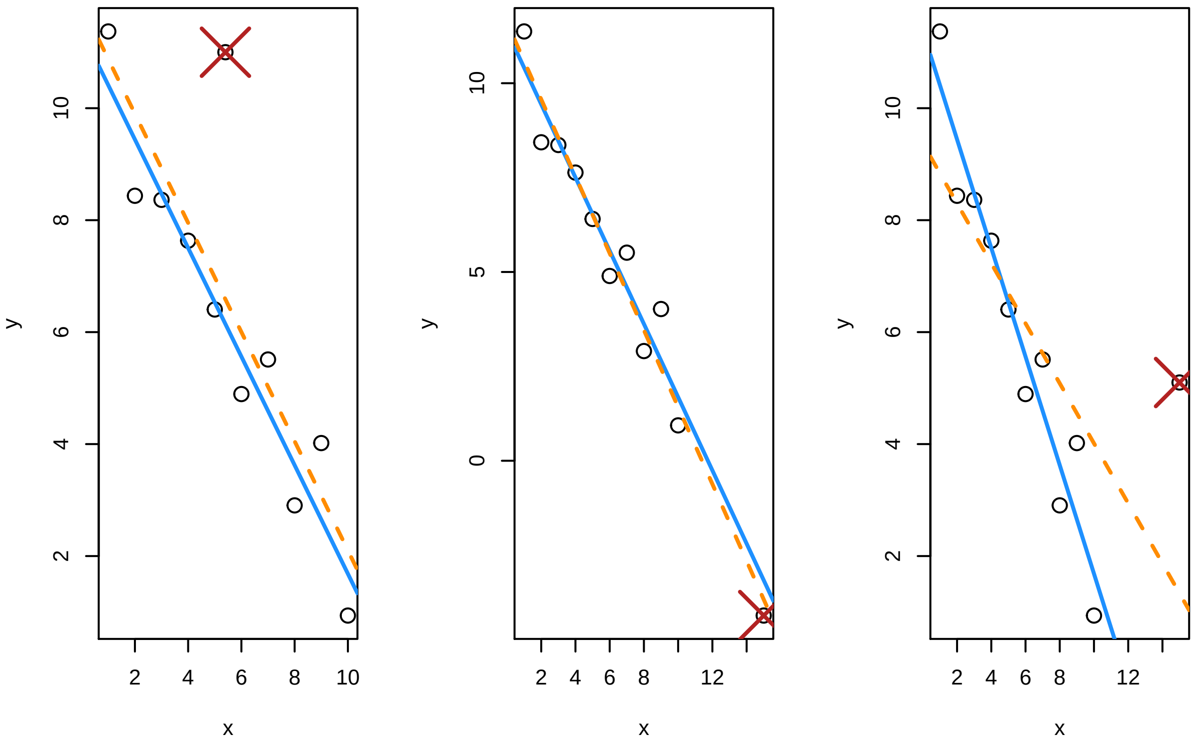
Notice that this is a function of both *leverage* and *standardized residuals*.

A Cook's Distance is considered large if

$$D_i > \frac{4}{n}$$

and an observation with a large Cook's Distance is called influential. This is again simply a heuristic, and not an exact rule.

The Cook's distance for each point of a regression can be calculated using `cooks.distance()` which is a default function in `R`. Let's look for influential points in the three plots we had been considering.



Recall that the added points (large red "X") in each plot have different characteristics:

- Plot One: low leverage, large residual.
- Plot Two: high leverage, small residual.
- Plot Three: high leverage, large residual.

We'll now directly check if each of these is influential.

```
cooks.distance(model1)[11] > 4 / length(cooks.distance(model1))
```

```
##    11
## FALSE
```

```
cooks.distance(model2)[11] > 4 / length(cooks.distance(model2))
```

```
##    11
## FALSE
```

```
cooks.distance(model3)[11] > 4 / length(cooks.distance(model3))
```

```
##   11
## TRUE
```

And, as expected, the added point in the third plot, with high leverage and a large residual is considered influential!

7.4 Data Analysis Examples

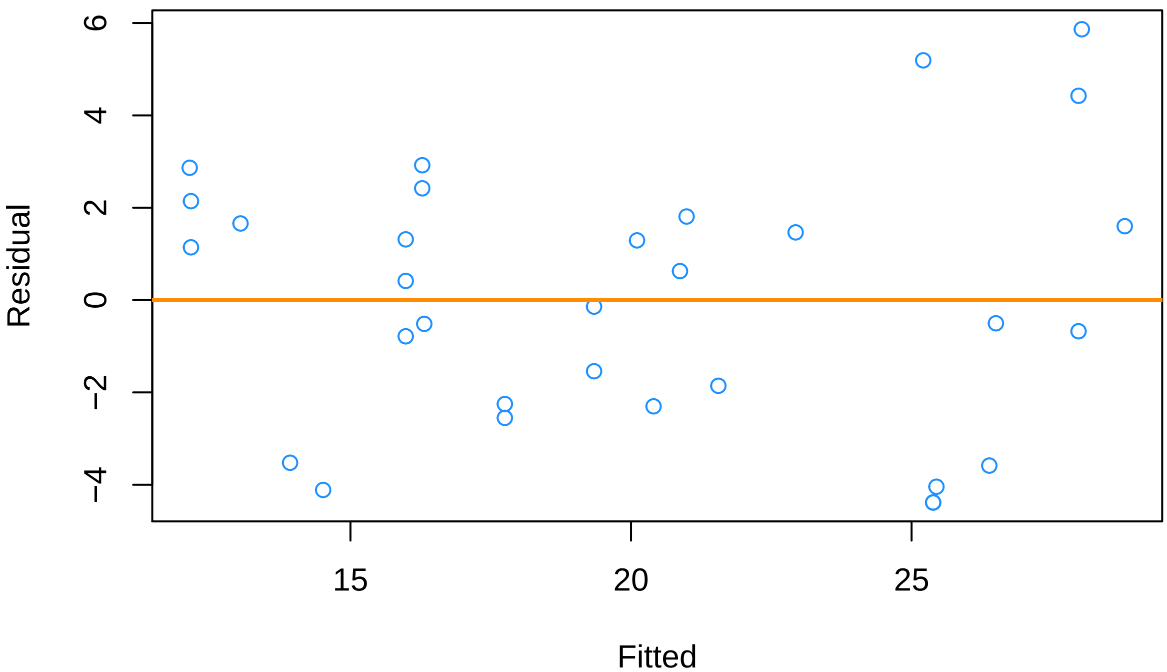
7.4.1 Good Diagnostics

Last chapter we fit an additive regression to the `mtcars` data with `mpg` as the response and `hp` and `am` as predictors. Let's perform some diagnostics on this model.

First, fit the model as we did last chapter.

```
mpg_hp_add = lm(mpg ~ hp + am, data = mtcars)
```

```
plot(fitted(mpg_hp_add), resid(mpg_hp_add), col = "dodgerblue", xlab = "Fitted", ylab = "Residual")
abline(h = 0, col = "darkorange", lwd = 2)
```



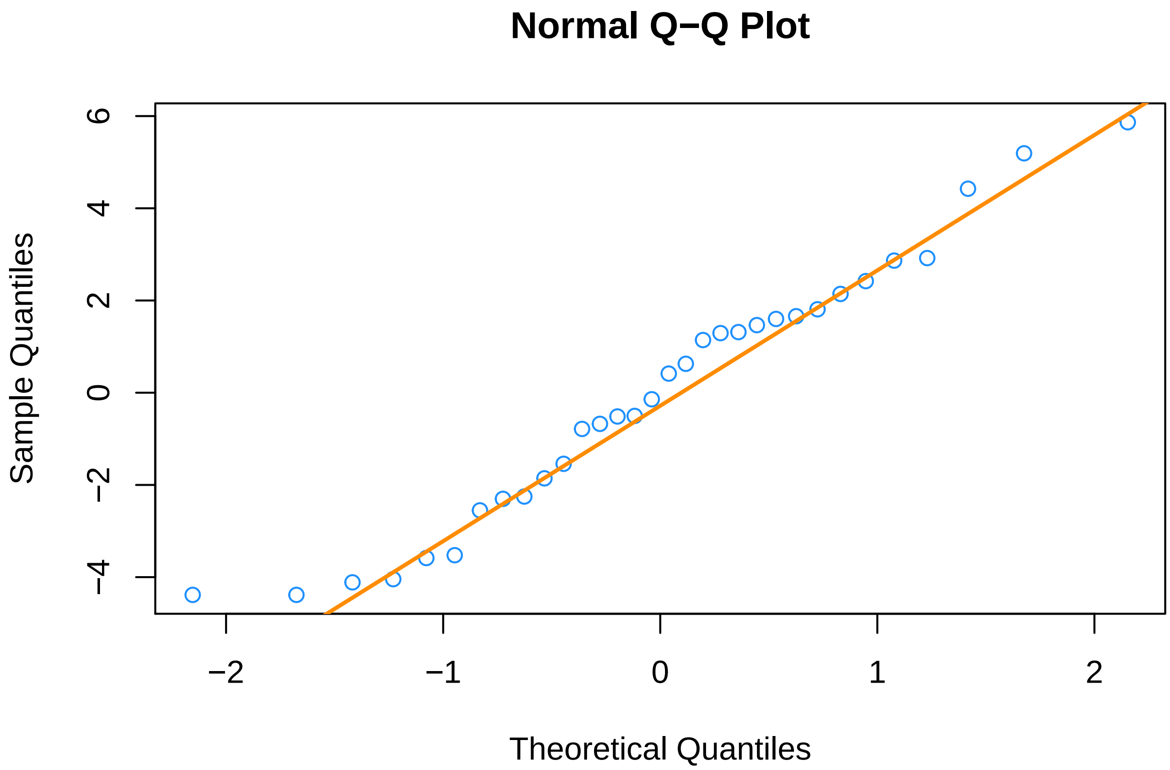
The fitted versus residuals plot looks good. We don't see any obvious pattern, and the variance looks roughly constant. (Maybe a little larger for large fitted values, but not enough to worry about.)

```
bptest(mpg_hp_add)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  mpg_hp_add
## BP = 7.5858, df = 2, p-value = 0.02253
```

The Breusch-Pagan test verifies this, at least for a small α value.

```
qqnorm(resid(mpg_hp_add), col = "dodgerblue")
qqline(resid(mpg_hp_add), col = "darkorange", lwd = 2)
```



The Q-Q plot looks extremely good and the Shapiro-Wilk test agrees.

```
shapiro.test(resid(mpg_hp_add))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(mpg_hp_add)
## W = 0.96485, p-value = 0.3706
```

```
sum(hatvalues(mpg_hp_add) > 2 * mean(hatvalues(mpg_hp_add)))
```

```
## [1] 2
```

We see that there are two points of large leverage.

```
sum(abs(rstandard(mpg_hp_add)) > 2)
```

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

There is also one point with a large residual. Do these result in any points that are considered influential?

```
cd_mpg_hp_add = cooks.distance(mpg_hp_add)
sum(cd_mpg_hp_add > 4 / length(cd_mpg_hp_add))
```

```
## [1] 2
```

```
large_cd_mpg = cd_mpg_hp_add > 4 / length(cd_mpg_hp_add)
cd_mpg_hp_add[large_cd_mpg]
```

```
## Toyota Corolla  Maserati Bora
##      0.1772555      0.3447994
```

We find two influential points. Interestingly, they are **very** different cars.

```
coef(mpg_hp_add)
```

```
## (Intercept)          hp          am
##  26.5849137  -0.0588878   5.2770853
```

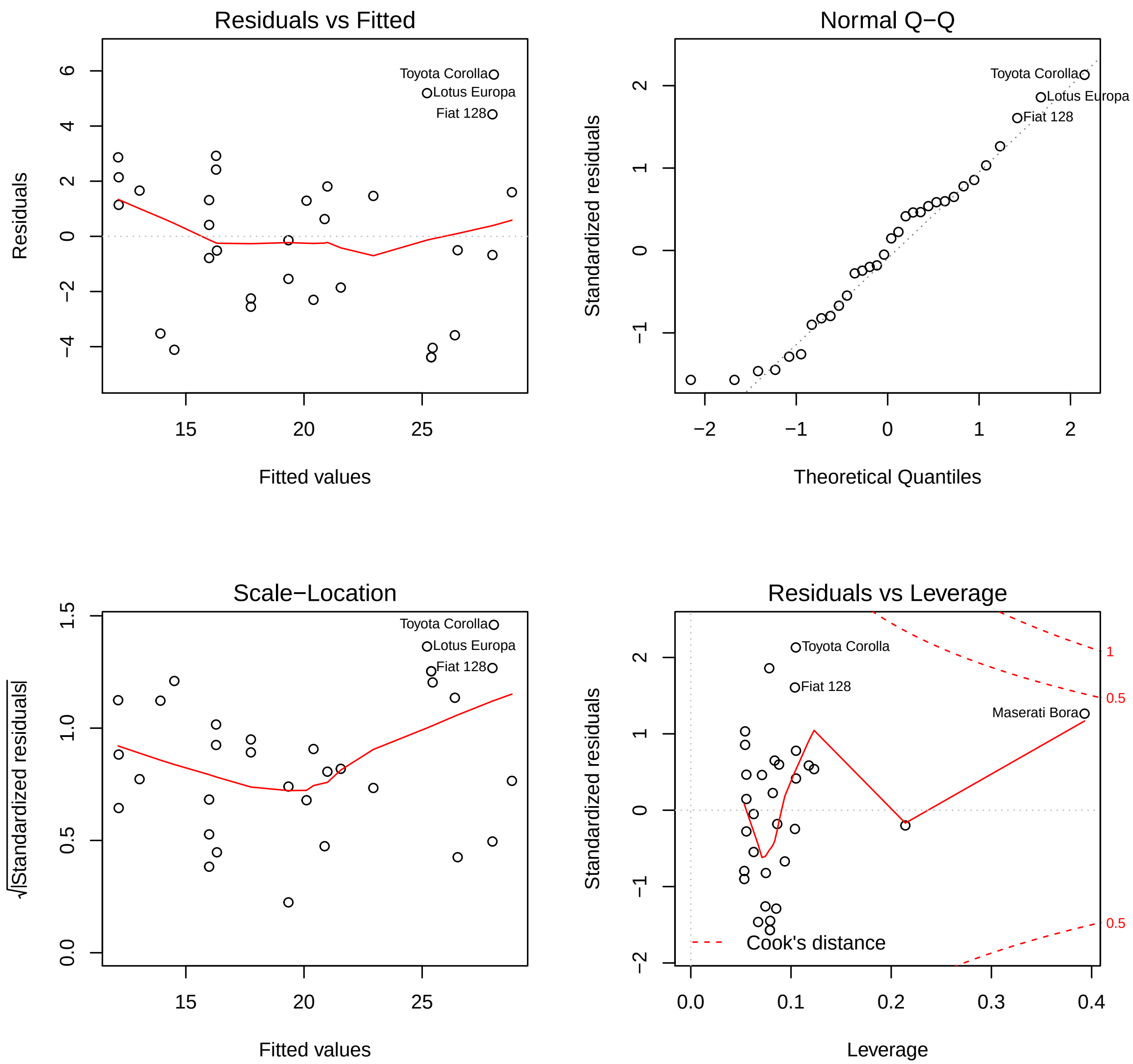
Since the diagnostics looked good, there isn't much need to worry about these two points, but let's see how much the coefficients change if we remove them.

```
mpg_hp_add_fix = lm(mpg ~ hp + am, data = mtcars, subset = cd_mpg_hp_add <= 4 / length(cd_mpg_hp_add))
coef(mpg_hp_add_fix)
```

```
## (Intercept)          hp          am
## 27.22190933 -0.06286249  4.29765867
```

It seems there isn't much of a change in the coefficients as a results of removing the supposed influential points. Notice we did not create a new dataset to accomplish this. We instead used the `subset` argument to `lm()`. Think about what the code `cd_mpg_hp_add <= 4 / length(cd_mpg_hp_add)` does here.

```
par(mfrow = c(2, 2))
plot(mpg_hp_add)
```



Notice that, calling `plot()` on a variable which stores an object created by `lm()` outputs four diagnostic plots by default. Use `?plot.lm` to learn more. The first two should already be familiar.

7.4.2 Suspect Diagnostics

Let's consider the model `big_model` from last chapter which was fit to the `autompg` dataset. It used `mpg` as the response, and considered many interaction terms between the predictors `disp`, `hp`, and `domestic`.

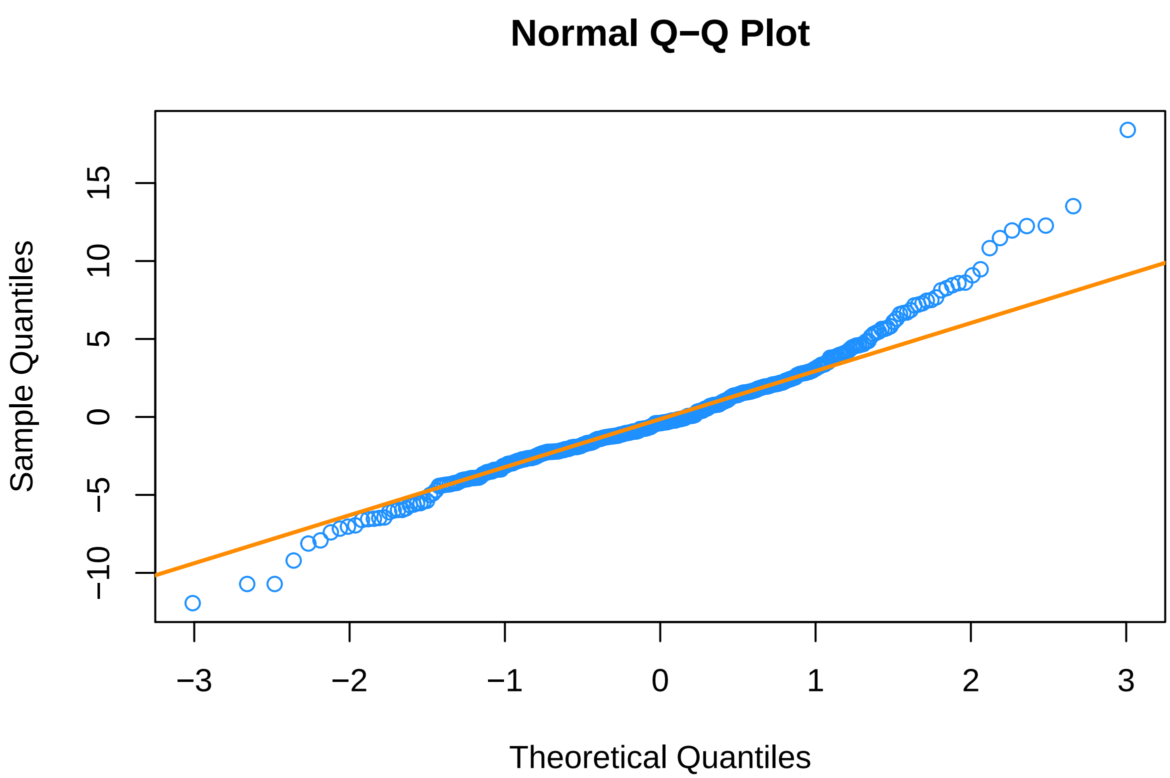
```
str(autompg)
```

```
## 'data.frame':    383 obs. of  9 variables:
##  $ mpg      : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cyl      : Factor w/ 3 levels "4","6","8": 3 3 3 3 3 3 3 3 3 3 ...
##  $ disp     : num  307 350 318 304 302 429 454 440 455 390 ...
##  $ hp       : num  130 165 150 150 140 198 220 215 225 190 ...
```

```
##  $ wt      : num  3504 3693 3436 3433 3449 ...
##  $ acc     : num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year    : int  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ domestic: num  1 1 1 1 1 1 1 1 1 1 ...
```

```
big_model = lm(mpg ~ disp * hp * domestic, data = autompg)
```

```
qqnorm(resid(big_model), col = "dodgerblue")
qqline(resid(big_model), col = "darkorange", lwd = 2)
```




```
shapiro.test(resid(big_model))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(big_model)
## W = 0.96161, p-value = 0.00000001824
```

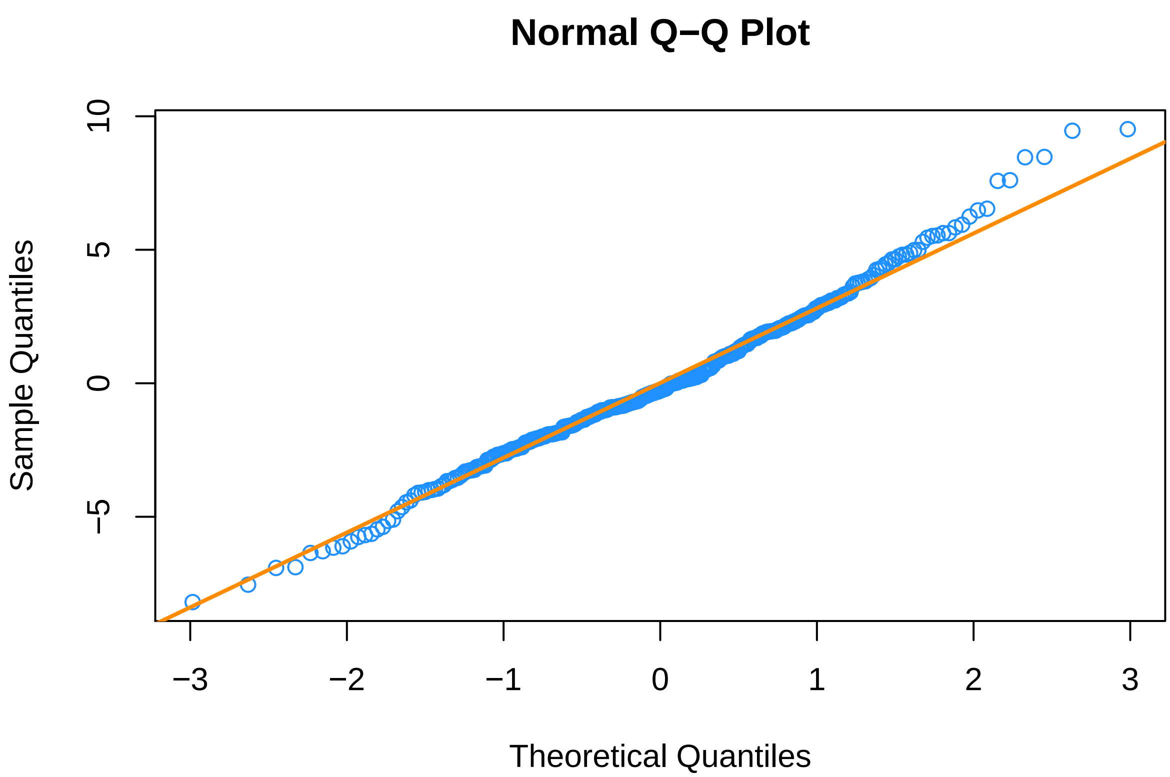
Here both the Q-Q plot, and the Shapiro-Wilk test suggest that the normality assumption is violated.

```
big_mod_cd = cooks.distance(big_model)
sum(big_mod_cd > 4 / length(big_mod_cd))
```

```
## [1] 31
```

Here, we find 31, so perhaps removing them will help!

```
big_model_fix = lm(mpg ~ disp * hp * domestic, data = autompg, subset = big_mod_cd < 4 / length(big_mod
qqnorm(resid(big_model_fix), col = "dodgerblue")
qqline(resid(big_model_fix), col = "darkorange", lwd = 2)
```




```
shapiro.test(resid(big_model_fix))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(big_model_fix)
## W = 0.99035, p-value = 0.02068
```

Removing these points results in a much better Q-Q plot, and now Shapiro-Wilk fails to reject for a low α.

We've now seen that sometimes modifying the data can fix issues with regression. However, next chapter, instead of modifying the data, we will modify the model via *transformations*.

Chapter 8

Transformations

> "Give me a lever long enough and a fulcrum on which to place it, and I shall move the world."
>
> — **Archimedes**

After reading this chapter you will be able to:

- Understand the concept of a variance stabilizing transformation.
- Use transformations of the response to improve regression models.
- Use polynomial terms as predictors to fit more flexible regression models.

Last chapter we checked the assumptions of regression models and looked at ways to diagnose possible issues. This chapter we will use transformations of both response and predictor variables in order to correct issues with model diagnostics, and to also potentially simply make a model fit data better.

8.1 Response Transformation

Let's look at some (fictional) salary data from the (fictional) company *Initech*. We will try to model `salary` as a function of `years` of experience. The data can be found in `initech.csv`.

```
initech = read.csv("data/initech.csv")
```

We first fit a simple linear model.

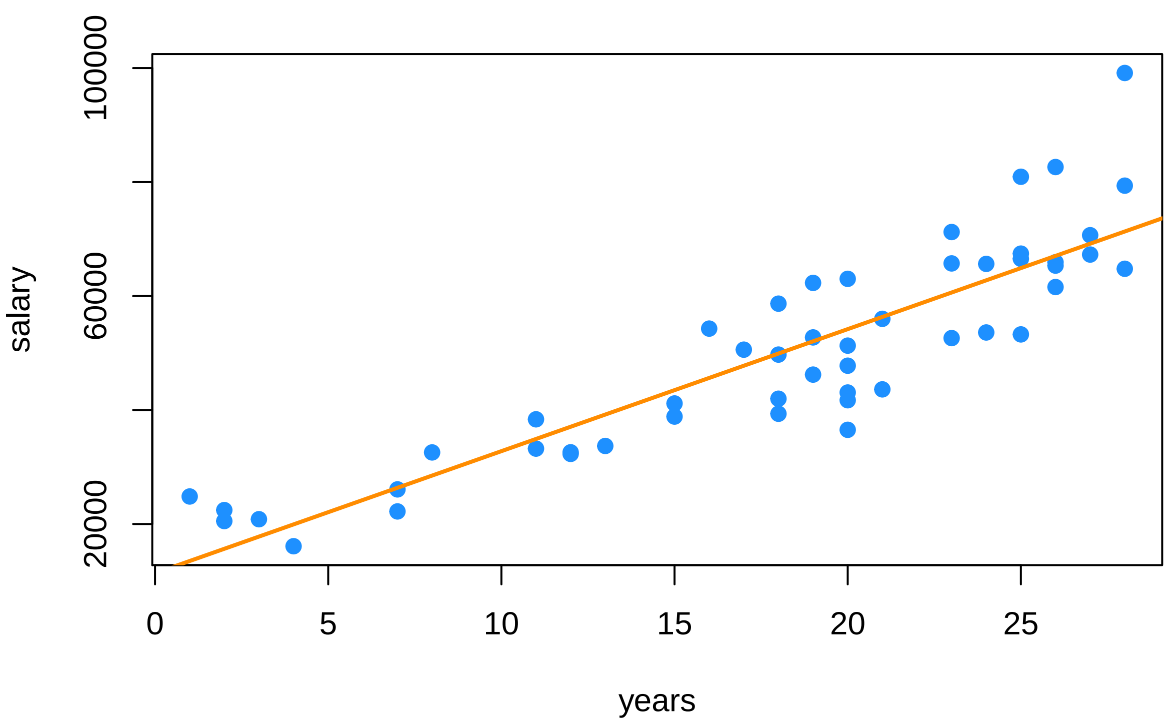
```
initech_fit = lm(salary ~ years, data = initech)
summary(initech_fit)
```

```
##
## Call:
## lm(formula = salary ~ years, data = initech)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17665.6  -5497.7   -725.7   4667.3  27812.9
##
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  11369.4     3160.2   3.598 0.000757 ***
## years         2141.3      160.8  13.314  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8642 on 48 degrees of freedom
## Multiple R-squared:  0.7869, Adjusted R-squared:  0.7825
## F-statistic: 177.3 on 1 and 48 DF,  p-value: < 2.2e-16
```

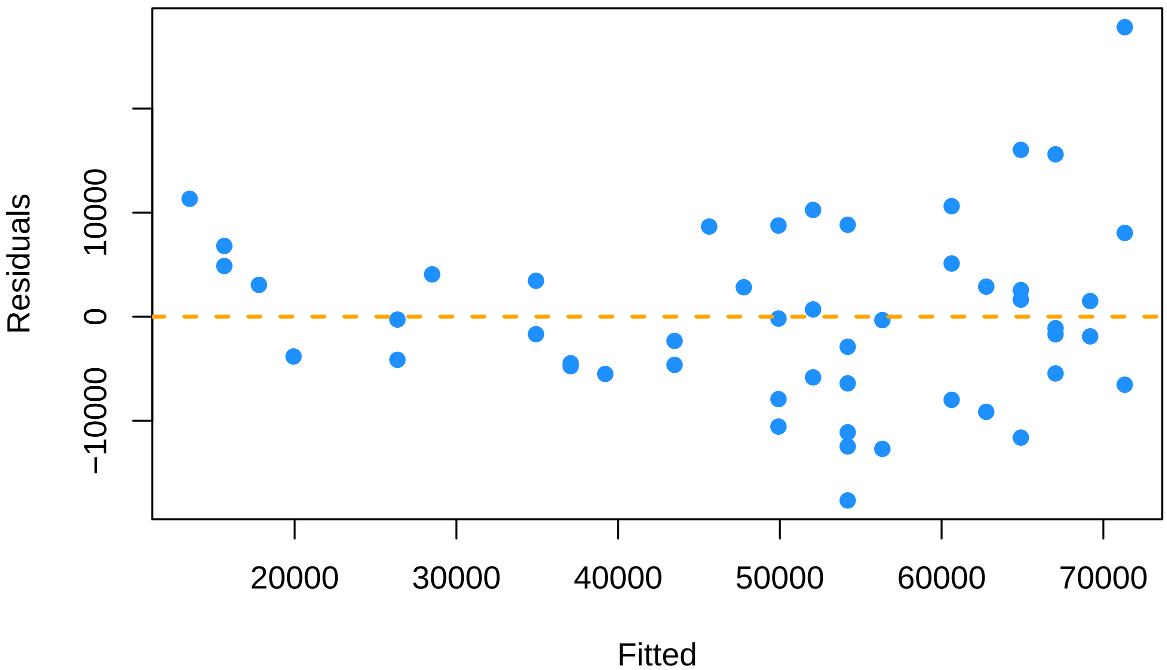
This model appears significant, but does it meet the model assumptions?

```r
plot(salary ~ years, data = initech, col = "dodgerblue", pch = 20, cex = 1.5)
abline(initech_fit, col = "darkorange", lwd = 2)
```



Adding the fitted line to the plot, we see that the linear relationship appears correct.

```r
plot(fitted(initech_fit), resid(initech_fit), col = "dodgerblue",
     pch = 20, cex = 1.5, xlab = "Fitted", ylab = "Residuals")
abline(h = 0, lty = 2, col = "orange", lwd = 2)
```



However, from the fitted versus residuals plot it appears there is non-constant variance. Specifically, the variance increases as the fitted value increases.

8.1.1 Variance Stabilizing Transformations

Recall the fitted value is our estimate of the mean at a particular value of x. Under our usual assumptions,

$$\epsilon_i \sim N(0, \sigma^2)$$

and thus

$$Var[Y \mid X = x] = \sigma^2$$

which is a constant value for any value of x.

However, here we see that the variance is a function of the mean,

$$Var[Y \mid X = x] = h(\mu).$$

In this case, h is some increasing function.

In order to correct for this, we would like to find some function of Y, $g(Y)$ such that,

$$Var[g(Y) \mid X = x] = c$$

where c is a constant that does not depend on μ. A transformation that accomplishes this is called a **variance stabilizaing transformation.**

A common variance stabilizing transformation (VST) when we see increasing variance in a fitted versus residuals plot is $\log(Y)$. Also, if the values of a variable range over more than one order of magnitude and the variable is *strictly positive*, then replacing the variable by its logarithm is likely to be helpful.

A reminder, that for our purposes, log and ln are both the natural log. `R` uses `log` to mean the natural log, unless a different base is specified.

We will now use a log transformed response for the *Initech* data,

$$\log(y_i) = \beta_0 + \beta_1 x_i + \epsilon_i.$$

Note, if we re-scale the data from a log scale back to the original scale of the data, we now have

$$y_i = \exp(\beta_0 + \beta_1 x_i) \cdot \exp(\epsilon_i)$$

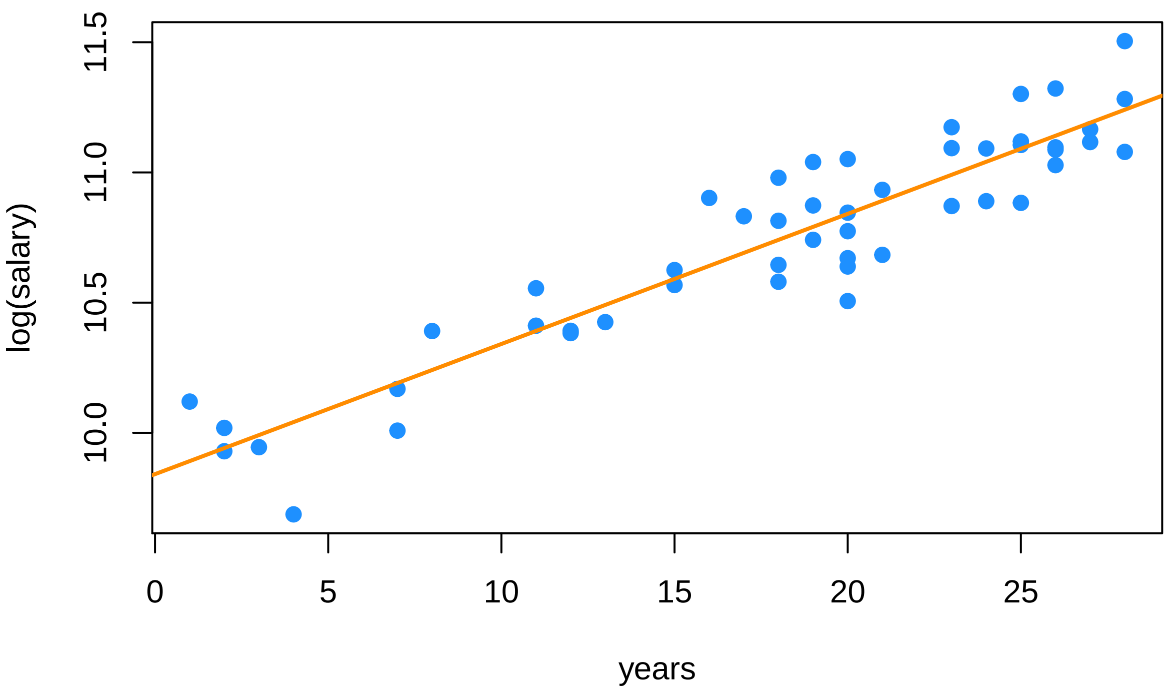
which has the errors entering the model in a multiplicative fashion.

Fitting this model in `R` requires only a minor modification to our formula specification.

```
initech_fit_log = lm(log(salary) ~ years, data = initech)
```

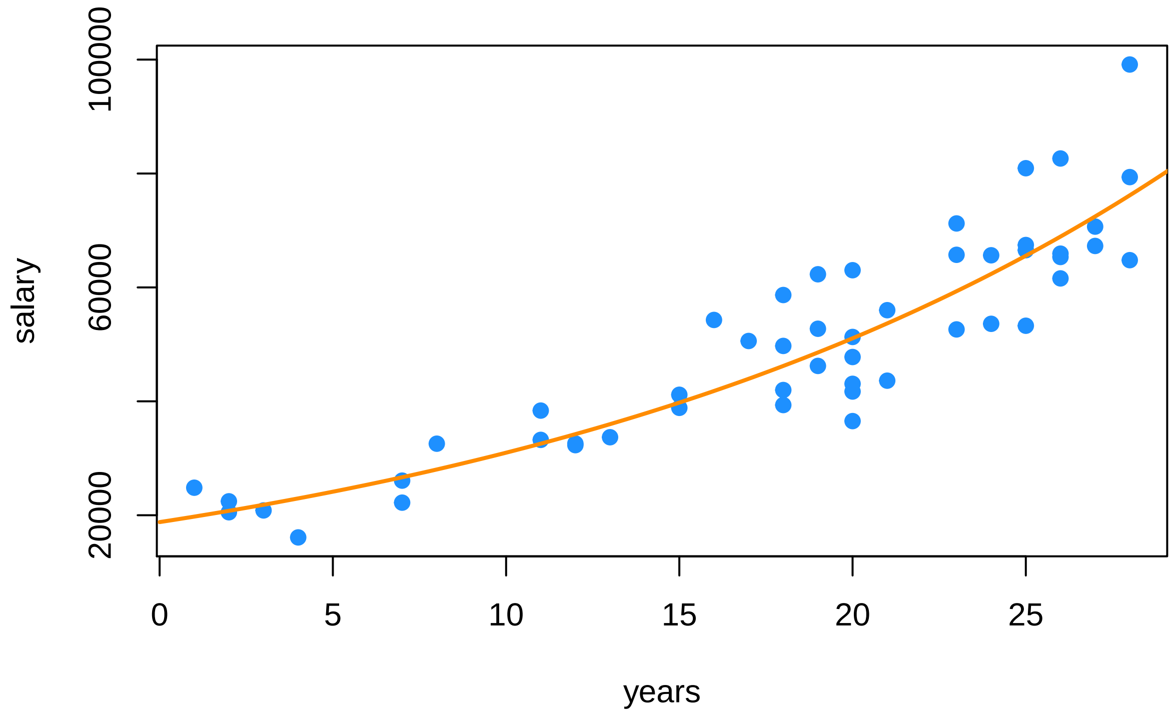
Note that while `log(y)` is considered the new response variable, we do not actually create a new variable in `R`, but simply transform the variable inside the model formula.

```
plot(log(salary) ~ years, data = initech, col = "dodgerblue", pch = 20, cex = 1.5)
abline(initech_fit_log, col = "darkorange", lwd = 2)
```



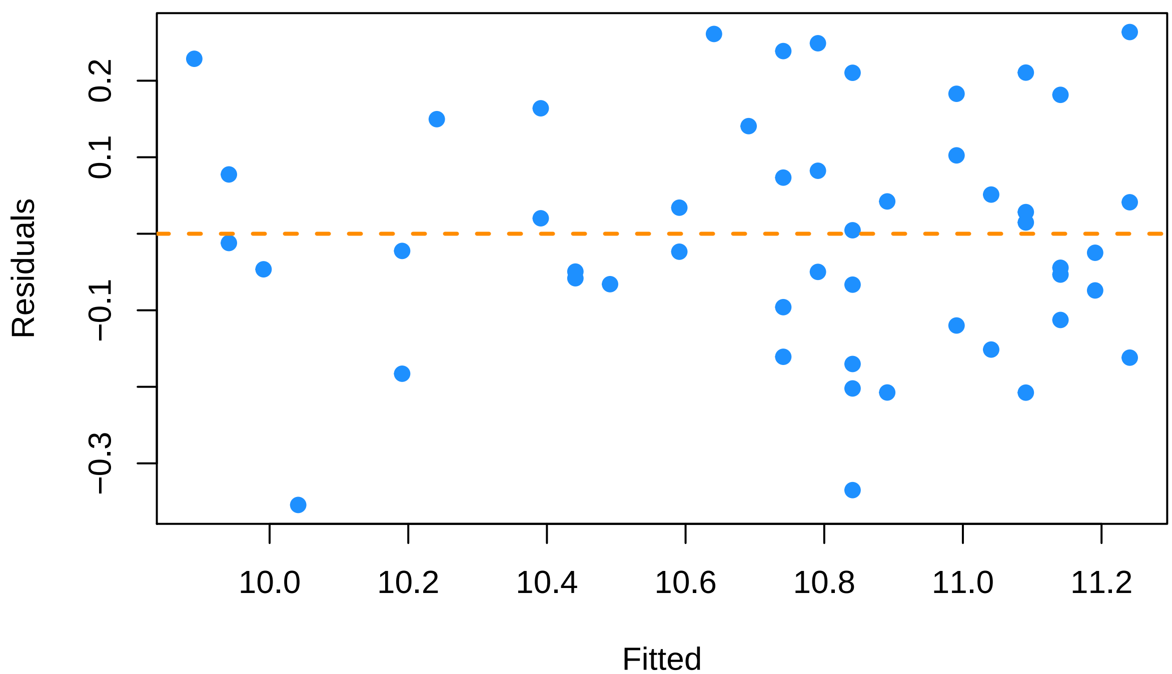
Plotting the data on the transformed log scale and adding the fitted line, the relationship again appears linear, and we can already see that the variation about the fitted line looks constant.

```
plot(salary ~ years, data = initech, col = "dodgerblue", pch = 20, cex = 1.5)
curve(exp(initech_fit_log$coef[1] + initech_fit_log$coef[2] * x),
      from = 0, to = 30, add = TRUE, col = "darkorange", lwd = 2)
```



By plotting the data on the original scale, and adding the fitted regression, we see an exponential relationship. However, this is still a *linear* model, since the new transformed response, $\log(y)$, is still a *linear* combination of the predictors.

```
plot(fitted(initech_fit_log), resid(initech_fit_log), col = "dodgerblue",
     pch = 20, cex = 1.5, xlab = "Fitted", ylab = "Residuals")
abline(h = 0, lty = 2, col = "darkorange", lwd = 2)
```



The fitted versus residuals plot looks much better. It appears the constant variance assumption is no longer violated.

Comparing the RMSE using the original and transformed response, we also see that the log transformed model simply fits better, with a smaller average squared error.

```
sqrt(mean(resid(initech_fit) ^ 2))
```

```
## [1] 8467.647
```

```
sqrt(mean(resid(initech_fit_log) ^ 2))
```

```
## [1] 0.1509989
```

But wait, that isn't fair, this difference is simply due to the different scales being used.

```
sqrt(mean((initech$salary - fitted(initech_fit)) ^ 2))
```

```
## [1] 8467.647
```

```
sqrt(mean((initech$salary - exp(fitted(initech_fit_log))) ^ 2))
```

```
## [1] 7874.517
```

Transforming the fitted values of the log model back to the data scale, we do indeed see that it fits better!

```
summary(initech_fit_log)
```

```
##
## Call:
## lm(formula = log(salary) ~ years, data = initech)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.35435 -0.09045 -0.01726  0.09740  0.26357
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 9.841325   0.056355  174.63   <2e-16 ***
## years       0.049978   0.002868   17.43   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1541 on 48 degrees of freedom
## Multiple R-squared:  0.8635, Adjusted R-squared:  0.8607
## F-statistic: 303.6 on 1 and 48 DF,  p-value: < 2.2e-16
```

Again, the transformed response is a *linear* combination of the predictors,

$$\log(\hat{y}) = \hat{\beta}_0 + \hat{\beta}_1 x = 9.84 + 0.05x.$$

But now, if we re-scale the data from a log scale back to the original scale of the data, we now have

$$\hat{y} = \exp(\hat{\beta}_0)\exp(\hat{\beta}_1 x) = \exp(9.84)\exp(0.05x).$$

We see that for every one additional year of experience, average salary increases $\exp(0.05) = 1.051$ times. We are now multiplying, not adding.

While using a log transform is possibly the most common response variable transformation, many others exist. We will now consider a family of transformations and choose the best from among them, which includes the log transform.

8.1.2 Box-Cox Transformations

The Box-Cox method considers a family of transformations on strictly positive response variables,

$$g_\lambda(y) = \begin{cases} \dfrac{y^\lambda - 1}{\lambda} & \lambda \neq 0 \\ \\ \log(y) & \lambda = 0 \end{cases}$$

The λ parameter is chosen by numerically maximizing the log-likelihood,

$$L(\lambda) = -\frac{n}{2}\log(RSS_\lambda / n) + (\lambda - 1)\sum \log(y_i).$$

A $100(1-\alpha)\%$ confidence interval for λ is,

$$\left\{ \lambda : L(\lambda) > L(\hat{\lambda}) - \frac{1}{2}\chi^2_{1,\alpha} \right\}$$

which `R` will plot for us to help quickly select an appropriate λ value. We often choose a "nice" value from within the confidence interval, instead of the value of λ that truly maximizes the likelihood.

```
library(MASS)
library(faraway)
```

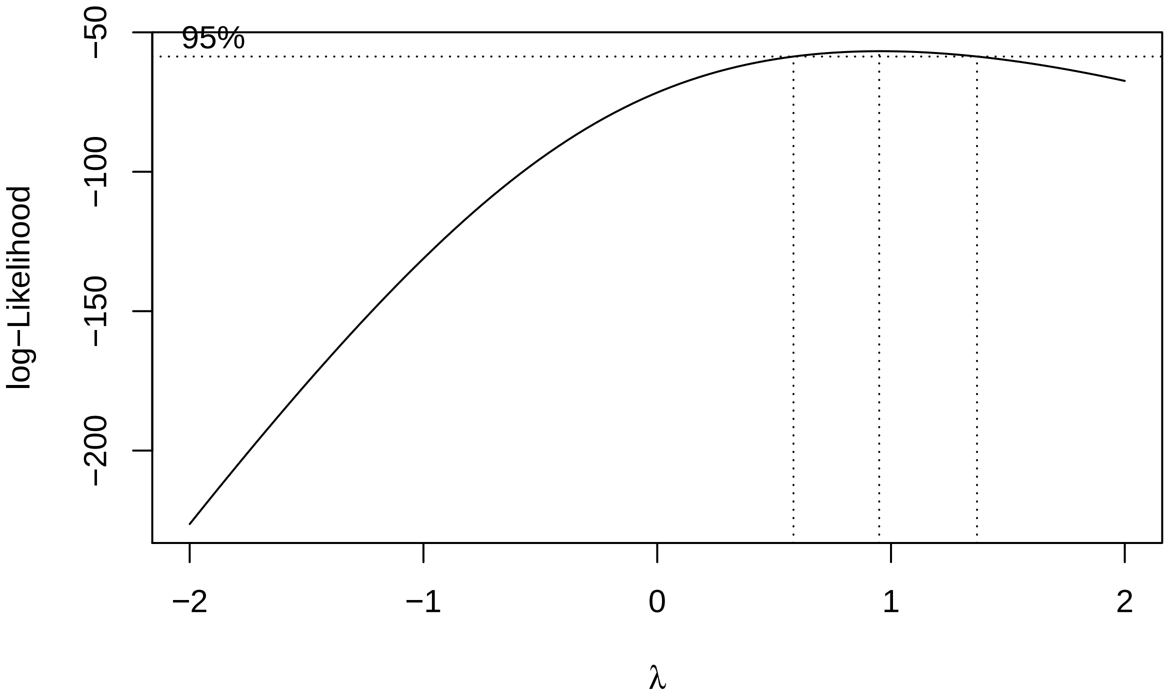
Here we need the `MASS` package for the `boxcox()` function, and we will consider a couple of datasets from the `faraway` package.

First we will use the `savings` dataset as an example of using the Box-Cox method to justify the use of no transformation. We fit an additive multiple regression model with `sr` as the response and each of the other variables as predictors.

```
savings_model = lm(sr ~ ., data = savings)
```

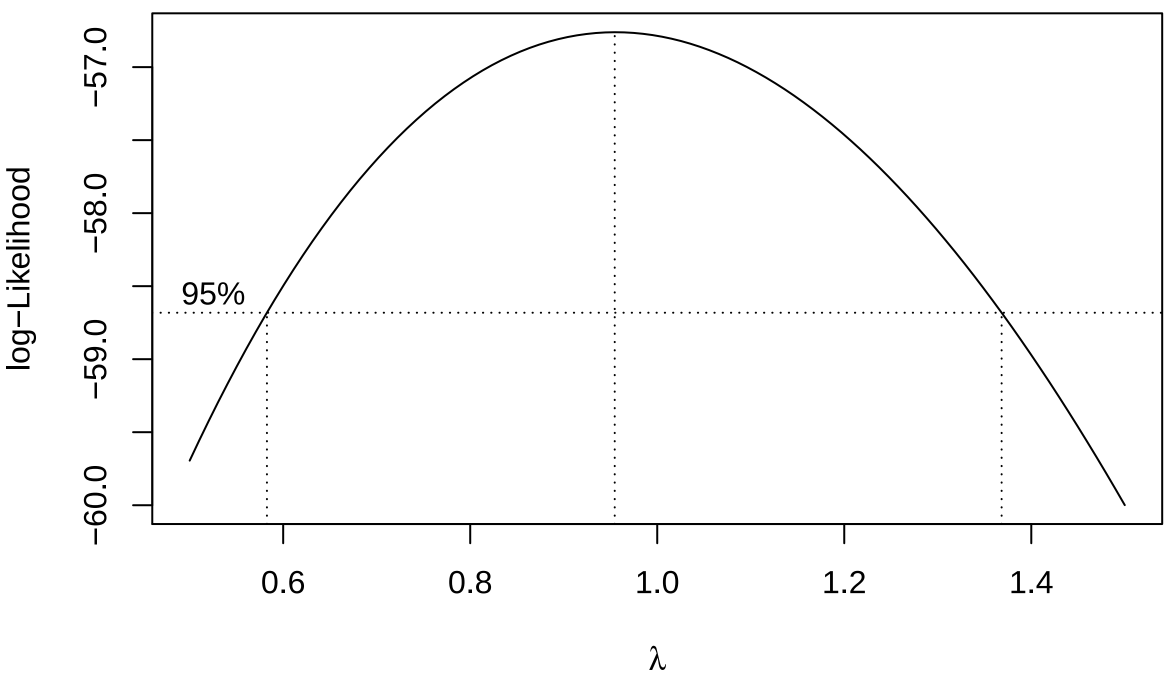
We then use the `boxcox()` function to find the best transformation of the form considered by the Box-Cox method.

```
boxcox(savings_model, plotit = TRUE)
```



R automatically plots the log-Likelihood as a function of possible λ values. It indicates both the value that maximizes the log-likelihood, as well as a confidence interval for the λ value that maximizes the log-likelihood.

```
boxcox(savings_model, plotit = TRUE, lambda = seq(0.5, 1.5, by = 0.1))
```

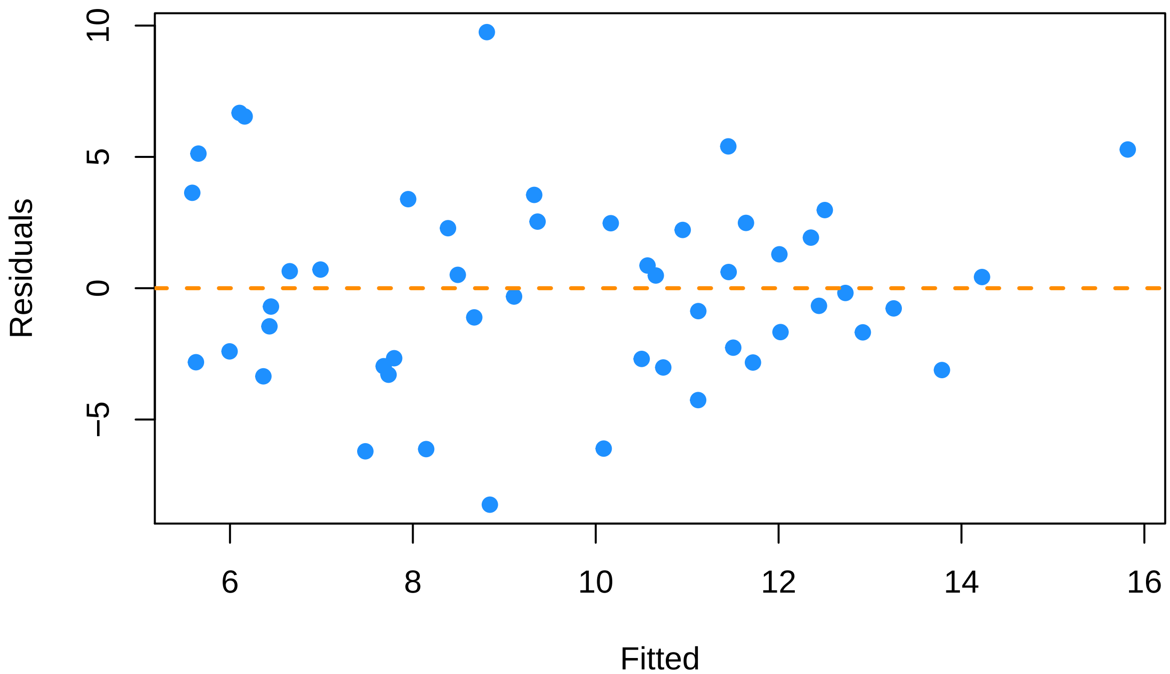


Note that we can specify a range of λ values to consider and thus be plotted. We often specify a range that is more visually interesting. Here we see that $\lambda = 1$ is both in the confidence interval, and is extremely close to the maximum. This suggests a transformation of the form

$$\frac{y^\lambda - 1}{\lambda} = \frac{y^1 - 1}{1} = y - 1.$$

This is essentially not a transformation. It would not change the variance or make the model fit better. By subtracting 1 from every value, we would only change the intercept of the model, and the resulting errors would be the same.

```
plot(fitted(savings_model), resid(savings_model), col = "dodgerblue",
     pch = 20, cex = 1.5, xlab = "Fitted", ylab = "Residuals")
abline(h = 0, lty = 2, col = "darkorange", lwd = 2)
```



Looking at a fitted versus residuals plot verifies that there likely are not any issue with the assumptions of this model, which Breusch-Pagan and Shapiro-Wilk tests verify.

```
library(lmtest)
bptest(savings_model)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  savings_model
## BP = 4.9852, df = 4, p-value = 0.2888
```

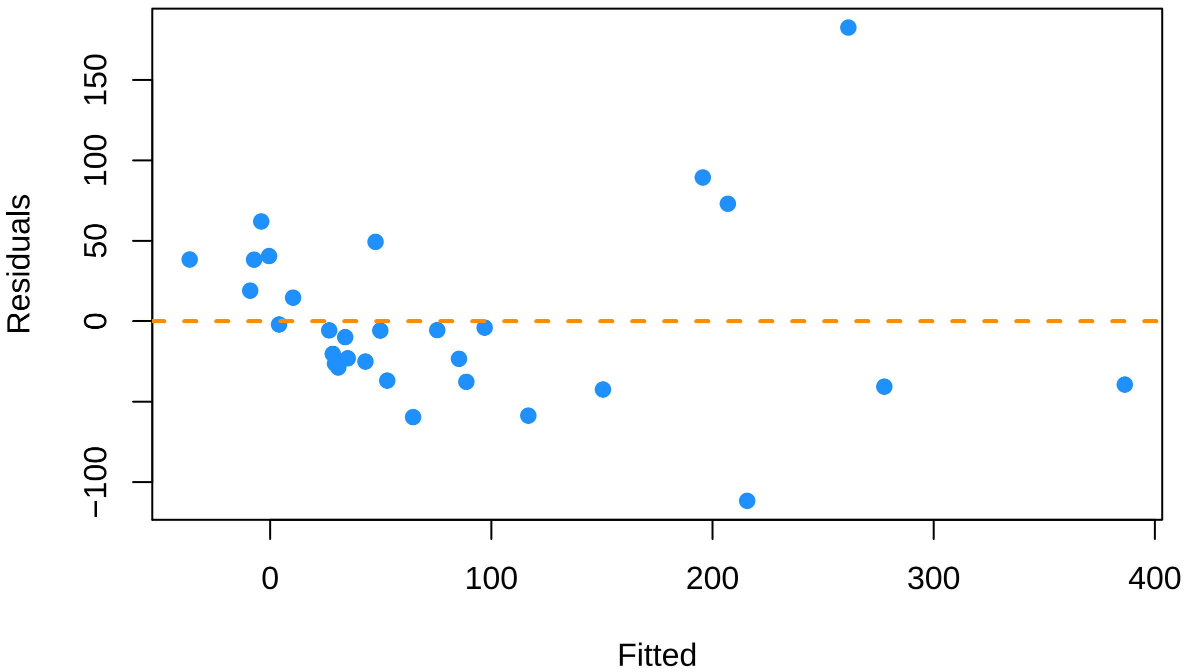
```
shapiro.test(resid(savings_model))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(savings_model)
## W = 0.98698, p-value = 0.8524
```

Now we will use the `gala` dataset as an example of using the Box-Cox method to justify a transformation other than log. We fit an additive multiple regression model with `Species` as the response and most of the other variables as predictors.

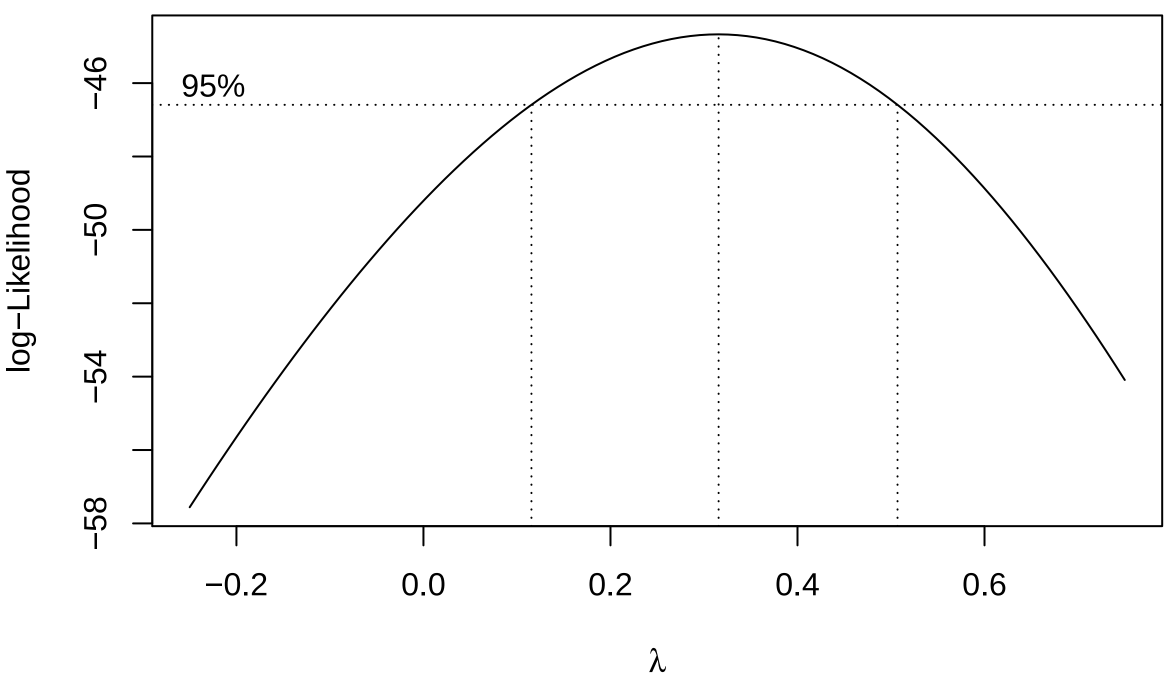
```
gala_model = lm(Species ~ Area + Elevation + Nearest + Scruz + Adjacent, data = gala)
```

```
plot(fitted(gala_model), resid(gala_model), col = "dodgerblue",
     pch = 20, cex = 1.5, xlab = "Fitted", ylab = "Residuals")
abline(h = 0, lty = 2, col = "darkorange", lwd = 2)
```



Even though there is not a lot of data for large fitted values, it still seems very clear that the constant variance assumption is violated.

```
boxcox(gala_model, lambda = seq(-0.25, 0.75, by = 0.05), plotit = TRUE)
```



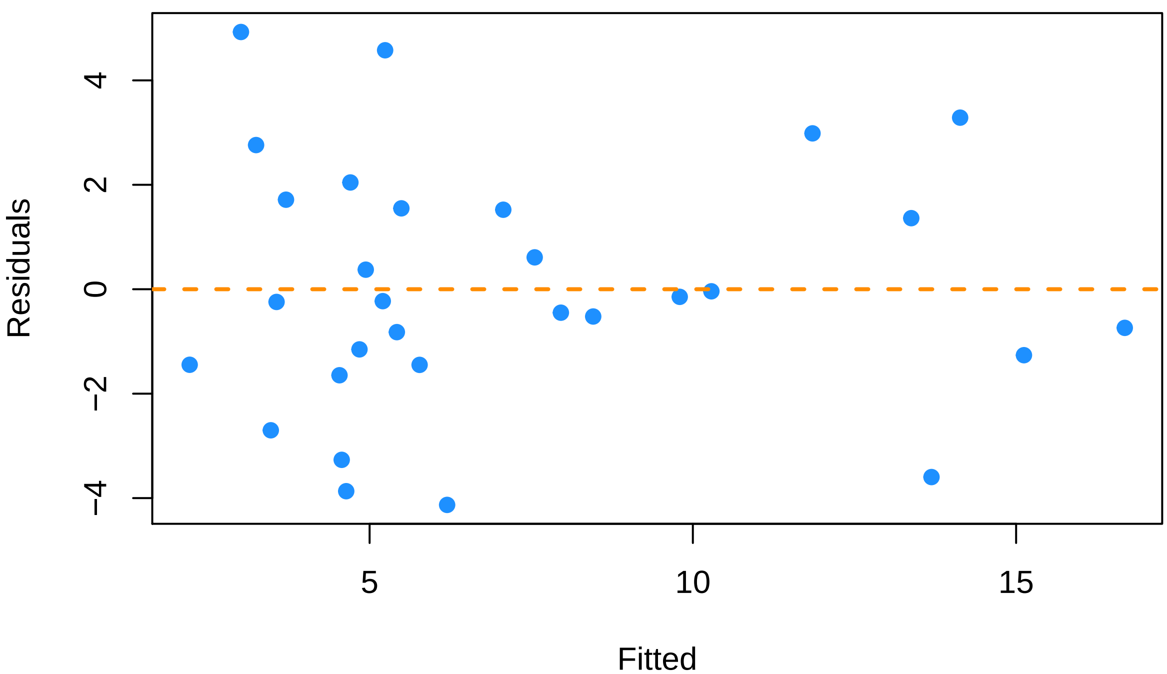
Using the Box-Cox method, we see that $\lambda = 0.3$ is both in the confidence interval, and is extremely close to the maximum, which suggests a transformation of the form

$$\frac{y^\lambda - 1}{\lambda} = \frac{y^{0.3} - 1}{0.3}.$$

We then fit a model with this transformation applied to the response.

```
gala_model_cox = lm((((Species ^ 0.3) - 1) / 0.3) ~ Area + Elevation + Nearest + Scruz + Adjacent, data
```

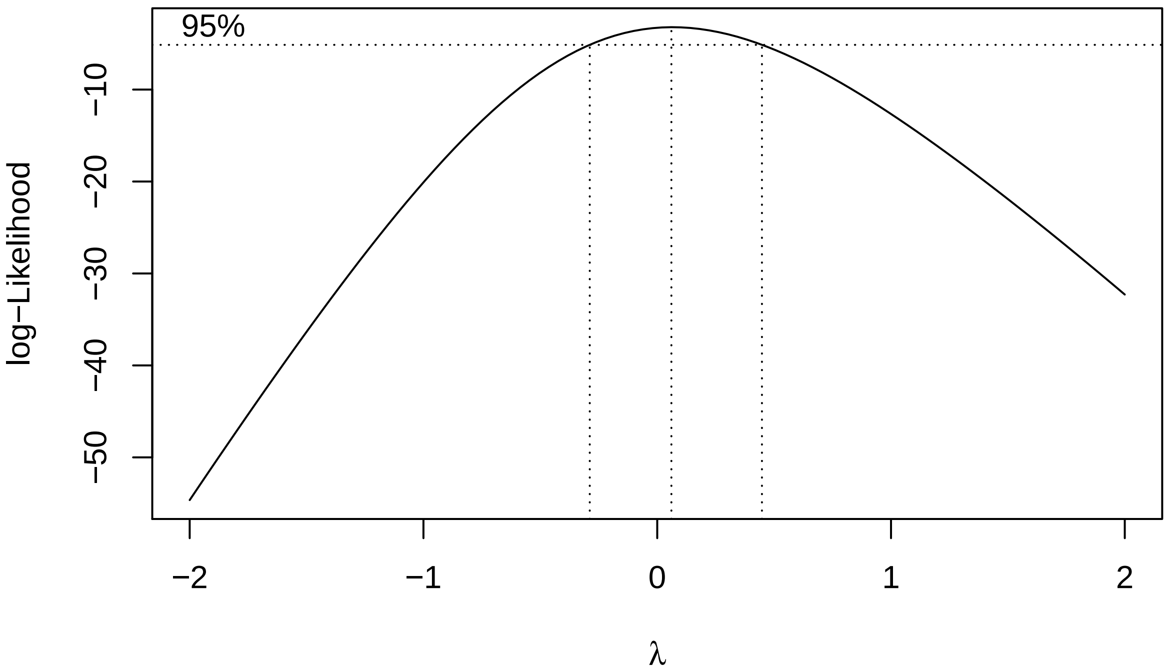
```
plot(fitted(gala_model_cox), resid(gala_model_cox), col = "dodgerblue",
     pch = 20, cex = 1.5, xlab = "Fitted", ylab = "Residuals")
abline(h = 0, lty = 2, col = "darkorange", lwd = 2)
```



The resulting fitted versus residuals plot looks much better!

Lastly, we return to the `initech` data, and the `initech_fit` model we had used earlier. Recall, that this was the untransformed model, that we used a log transform to fix.

```
boxcox(initech_fit)
```



Using the Box-Cox method, we see that $\lambda = 0$ is both in the interval, and extremely close to the maximum, which suggests a transformation of the form

$$\log(y).$$

So the Box-Cox method justifies our previous choice of a log transform!

8.2 Predictor Transformation

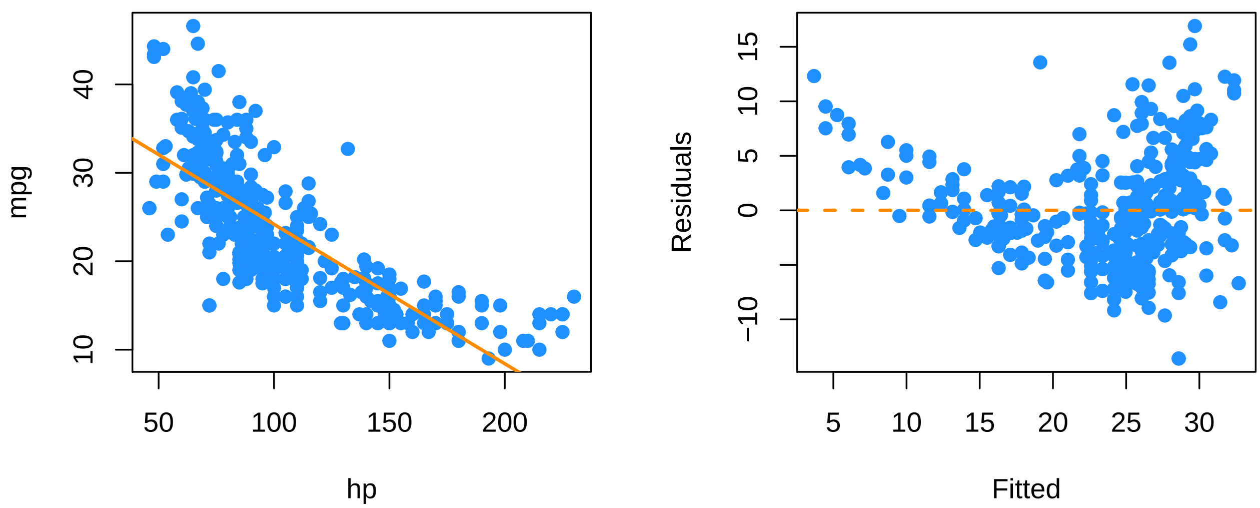
In addition to transformation of the response variable, we can also consider transformations of predictor variables. Sometimes these transformations can help with violation of model assumptions, and other times they can be used to simply fit a more flexible model.

```
str(autompg)
```

```
## 'data.frame':    383 obs. of  9 variables:
##  $ mpg     : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cyl     : Factor w/ 3 levels "4","6","8": 3 3 3 3 3 3 3 3 3 3 ...
##  $ disp    : num  307 350 318 304 302 429 454 440 455 390 ...
##  $ hp      : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ wt      : num  3504 3693 3436 3433 3449 ...
##  $ acc     : num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year    : int  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ domestic: num  1 1 1 1 1 1 1 1 1 1 ...
```

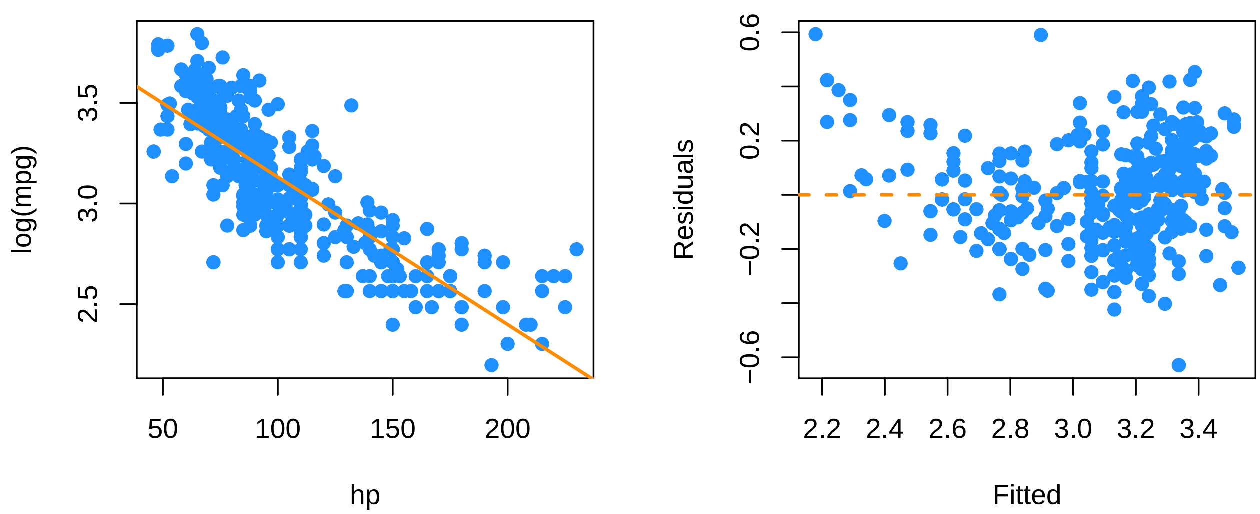
Recall the `autompg` dataset from the previous chapter. Here we will attempt to model `mpg` as a function of `hp`.

```
par(mfrow = c(1, 2))
plot(mpg ~ hp, data = autompg, col = "dodgerblue", pch = 20, cex = 1.5)
mpg_hp = lm(mpg ~ hp, data = autompg)
abline(mpg_hp, col = "darkorange", lwd = 2)
plot(fitted(mpg_hp), resid(mpg_hp), col = "dodgerblue",
     pch = 20, cex = 1.5, xlab = "Fitted", ylab = "Residuals")
abline(h = 0, lty = 2, col = "darkorange", lwd = 2)
```



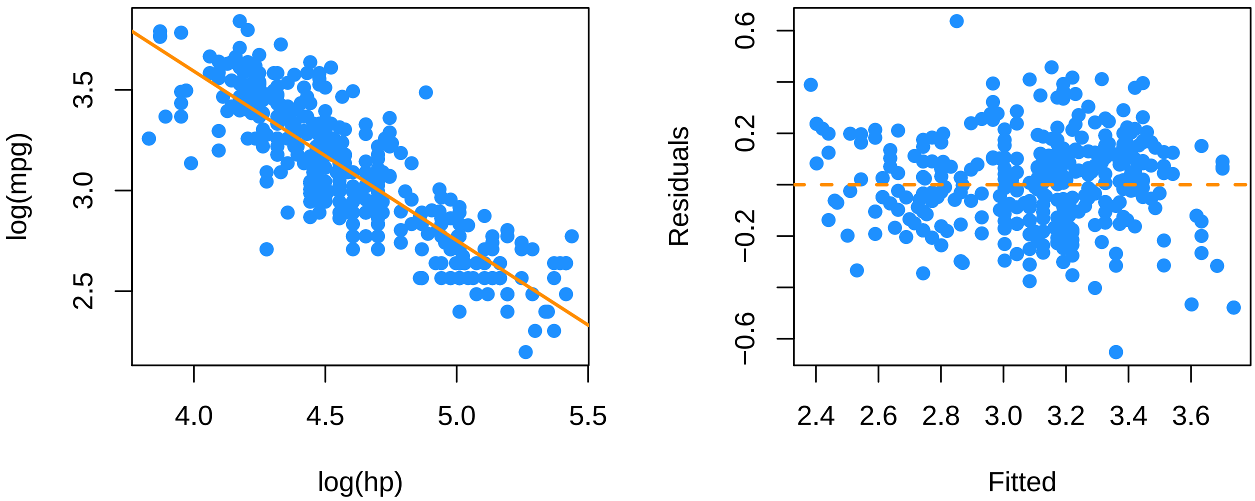
We first attempt SLR, but we see a rather obvious pattern in the fitted versus residuals plot, which includes increasing variance, so we attempt a log transform of the response.

```
par(mfrow = c(1, 2))
plot(log(mpg) ~ hp, data = autompg, col = "dodgerblue", pch = 20, cex = 1.5)
mpg_hp_log = lm(log(mpg) ~ hp, data = autompg)
abline(mpg_hp_log, col = "darkorange", lwd = 2)
plot(fitted(mpg_hp_log), resid(mpg_hp_log), col = "dodgerblue",
     pch = 20, cex = 1.5, xlab = "Fitted", ylab = "Residuals")
abline(h = 0, lty = 2, col = "darkorange", lwd = 2)
```



After performing the log transform of the response, we still have some of the same issues with the fitted versus response. Now, we will try also log transforming the **predictor**.

```
par(mfrow = c(1, 2))
plot(log(mpg) ~ log(hp), data = autompg, col = "dodgerblue", pch = 20, cex = 1.5)
mpg_hp_loglog = lm(log(mpg) ~ log(hp), data = autompg)
abline(mpg_hp_loglog, col = "darkorange", lwd = 2)
plot(fitted(mpg_hp_loglog), resid(mpg_hp_loglog), col = "dodgerblue",
     pch = 20, cex = 1.5, xlab = "Fitted", ylab = "Residuals")
abline(h = 0, lty = 2, col = "darkorange", lwd = 2)
```



Now our fitting versus residuals plot looks good.

8.2.1 Polynomials

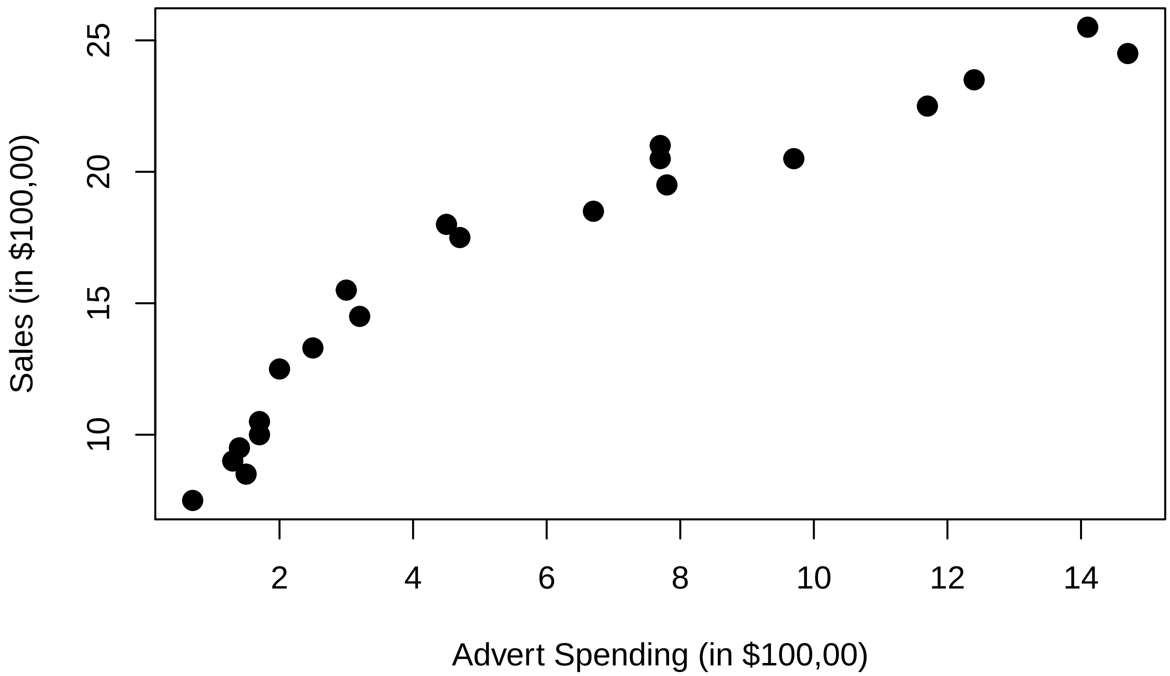
Another very common "transformation" of a predictor variable is the use of polynomial transformations. They are extremely useful as they allow for more flexible models, but do not change the units of the variables.

It should come as no surprise that sales of a product are related to the advertising budget for the product, but there are diminishing returns. A company cannot always expect linear returns based on an increased advertising budget.

Consider monthly data for the sales of *Initech* widgets, y, as a function of *Initech*'s advertising expenditure for said widget, x, both in ten thousand dollars. The data can be found in `marketing.csv`.

```
marketing = read.csv("data/marketing.csv")
```

```
plot(sales ~ advert, data = marketing,
     xlab = "Advert Spending (in $100,00)", ylab = "Sales (in $100,00)",
     pch = 20, cex = 2)
```



We would like to fit the model,

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$ for $i = 1, 2, \cdots 21$.

The response y is now a **linear** function of "two" variables which now allows y to be a non-linear function of the original single predictor x. We consider this a transformation, although we have actually in some sense added another predictor.

Thus, our X matrix is,

$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ \cdots & \cdots & \cdots \\ 1 & x_n & x_n^2 \end{bmatrix}$$

We can then proceed to fit the model as we have in the past for multiple linear regression.

$$\hat{\beta} = \left(X^\top X\right)^{-1} X^\top y.$$

Our estimates will have the usual properties. The mean is still

$$E[\hat{\beta}] = \beta,$$

and variance

$$Var[\hat{\beta}] = \sigma^2 \left(X^\top X\right)^{-1}.$$

We also maintain the same distributional results

$$\hat{\beta}_j \sim N\left(\beta_j, \sigma^2 C_{jj}\right).$$

```
mark_mod = lm(sales ~ advert, data = marketing)
summary(mark_mod)
```

```
##
## Call:
## lm(formula = sales ~ advert, data = marketing)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7845 -1.4762 -0.5103  1.2361  3.1869
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   9.4502     0.6806   13.88 2.13e-11 ***
## advert        1.1918     0.0937   12.72 9.65e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.907 on 19 degrees of freedom
## Multiple R-squared:  0.8949, Adjusted R-squared:  0.8894
## F-statistic: 161.8 on 1 and 19 DF,  p-value: 9.646e-11
```

While the SLR model is significant, the fitted versus residuals plot would have a very clear pattern.

```
mark_mod_poly2 = lm(sales ~ advert + I(advert ^ 2), data = marketing)
summary(mark_mod_poly2)
```

```
##
## Call:
## lm(formula = sales ~ advert + I(advert^2), data = marketing)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.9175 -0.8333 -0.1948  0.9292  2.1385
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.76161    0.67219  10.059 8.16e-09 ***
## advert       2.46231    0.24830   9.917 1.02e-08 ***
## I(advert^2) -0.08745    0.01658  -5.275 5.14e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.228 on 18 degrees of freedom
## Multiple R-squared:  0.9587, Adjusted R-squared:  0.9541
## F-statistic:   209 on 2 and 18 DF,  p-value: 3.486e-13
```

To add the second order term we need to use the `I()` function in the model specification around our newly created predictor. We see that with the first order term in the model, the quadratic term is also significant.

```r
n = length(marketing$advert)
X = cbind(rep(1, n), marketing$advert, marketing$advert ^ 2)
t(X) %*% X
```

```
##         [,1]     [,2]      [,3]
## [1,]   21.00   120.70   1107.95
## [2,]  120.70  1107.95  12385.86
## [3,] 1107.95 12385.86 151369.12
```

```r
solve(t(X) %*% X) %*% t(X) %*% marketing$sales
```

```
##             [,1]
## [1,]  6.76161045
## [2,]  2.46230964
## [3,] -0.08745394
```

Here we verify the parameter estimates were found as we would expect.

We could also add higher order terms, such as a third degree predictor. This is easy to do. Our X matrix simply becomes larger again.

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i$$

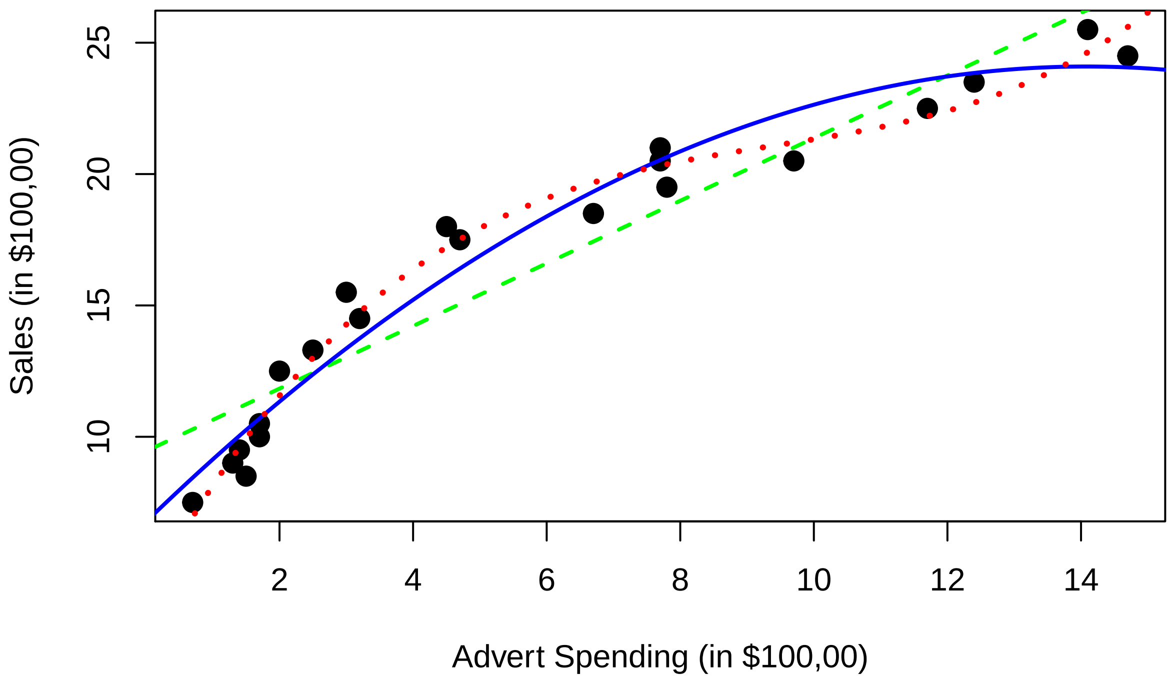
$$\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix}$$

```r
mark_mod_poly3 = lm(sales ~ advert + I(advert ^ 2) + I(advert ^ 3), data = marketing)
summary(mark_mod_poly3)
```

```
##
## Call:
## lm(formula = sales ~ advert + I(advert^2) + I(advert^3), data = marketing)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.44322 -0.61310 -0.01527  0.68131  1.22517
##
## Coefficients:
##              Estimate Std. Error t value     Pr(>|t|)
## (Intercept)  3.890070   0.761956   5.105 0.0000879488 ***
## advert       4.681864   0.501032   9.344 0.0000000414 ***
## I(advert^2) -0.455152   0.078977  -5.763 0.0000229561 ***
## I(advert^3)  0.016131   0.003429   4.704     0.000205 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8329 on 17 degrees of freedom
## Multiple R-squared:  0.9821, Adjusted R-squared:  0.9789
## F-statistic: 310.2 on 3 and 17 DF,  p-value: 4.892e-15
```

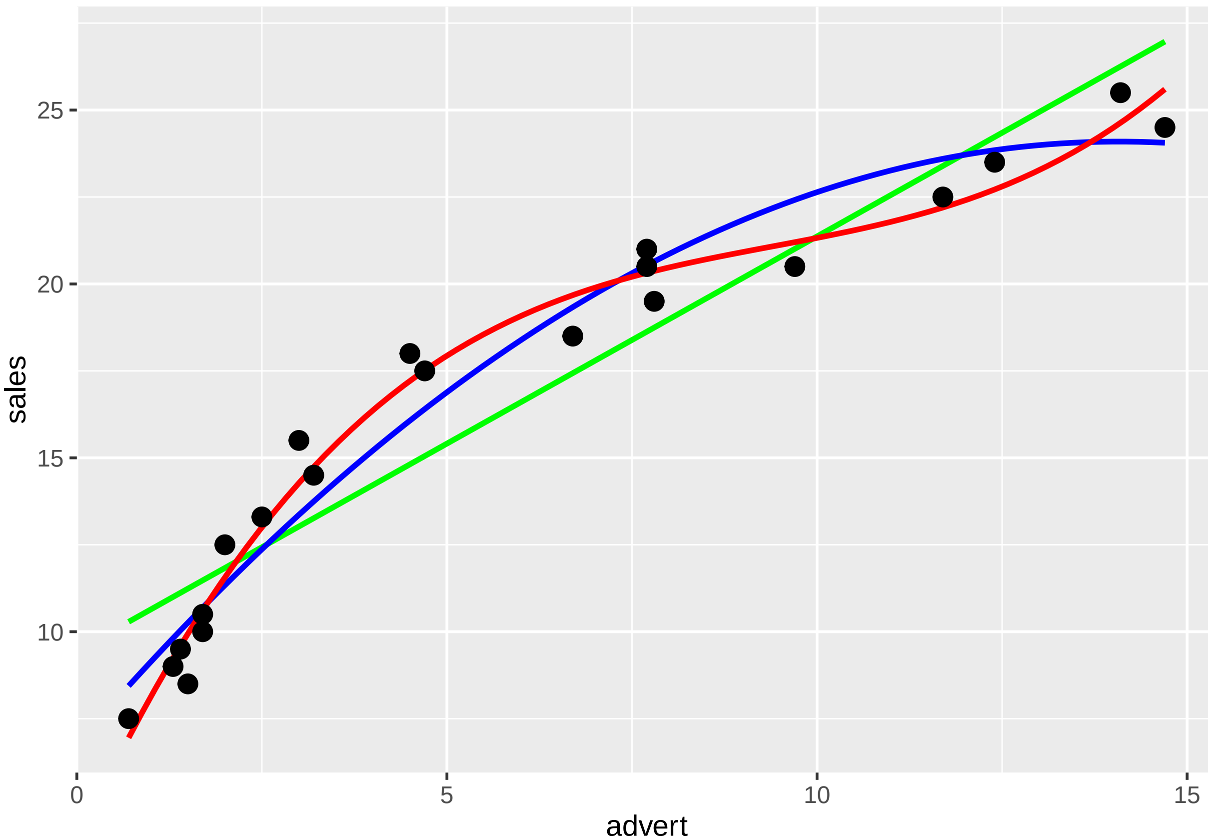
Now we see that with the first and second order terms in the model, the third order term is also significant. But does this make sense practically? The following plot should gives hints as to why it doesn't. (The model with the third order term doesn't have diminishing returns!)

```
plot(sales ~ advert, data = marketing,
     xlab = "Advert Spending (in $100,00)", ylab = "Sales (in $100,00)",
     pch = 20, cex = 2)
abline(mark_mod, lty = 2, col = "green", lwd = 2)
xplot = seq(0, 16, by = 0.01)
lines(xplot, predict(mark_mod_poly2, newdata = data.frame(advert = xplot)),
      col = "blue", lwd = 2)
lines(xplot, predict(mark_mod_poly3, newdata = data.frame(advert = xplot)),
      col = "red", lty = 3, lwd = 3)
```



The previous plot was made using base graphics in `R`. The next plot was made using the package `ggplot2`, an increasingly popular plotting method in `R`.

```
library(ggplot2)
ggplot(data = marketing, aes(x = advert, y = sales)) +
  stat_smooth(method = "lm", se = FALSE, color = "green", formula = y ~ x) +
  stat_smooth(method = "lm", se = FALSE, color = "blue", formula = y ~ x + I(x ^ 2)) +
  stat_smooth(method = "lm", se = FALSE, color = "red", formula = y ~ x + I(x ^ 2)+ I(x ^ 3)) +
  geom_point(colour = "black", size = 3)
```



Note we could fit a polynomial of an arbitrary order,

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_{p-1} x_i^{p-1} + \epsilon_i$$

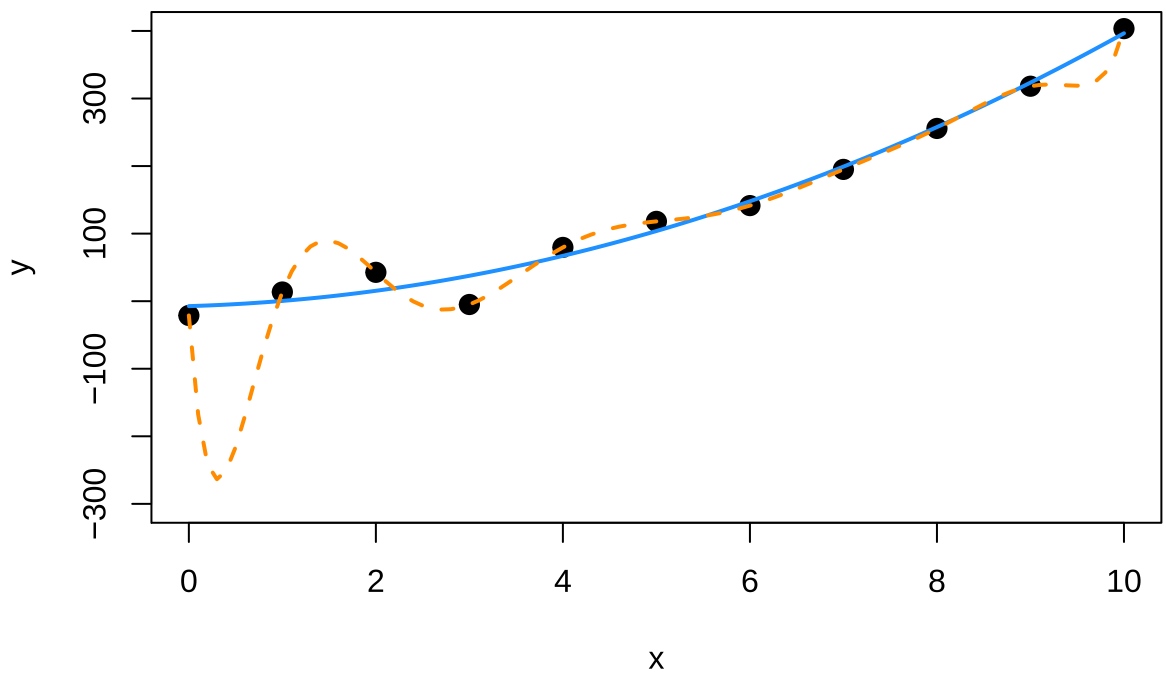
However, we should be careful about over-fitting, since with a polynomial of degree one less than the number of observations, it is sometimes possible to fit a model perfectly.

```
set.seed(1234)
x = seq(0, 10)
y = 3 + x + 4 * x ^ 2 + rnorm(11, 0, 20)
plot(x, y, ylim = c(-300, 400), cex = 2, pch = 20)
fit = lm(y ~ x + I(x ^ 2))
#summary(fit)
fit_perf = lm(y ~ x + I(x ^ 2) + I(x ^ 3) + I(x ^ 4) + I(x ^ 5) + I(x ^ 6)
               + I(x ^ 7) + I(x ^ 8) + I(x ^ 9) + I(x ^ 10))
summary(fit_perf)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) +
##     I(x^7) + I(x^8) + I(x^9) + I(x^10))
##
## Residuals:
## ALL 11 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
```

```
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -21.141315         NA      NA       NA
## x           -1918.260330         NA      NA       NA
## I(x^2)       4969.169159         NA      NA       NA
## I(x^3)      -4932.231427         NA      NA       NA
## I(x^4)       2580.602473         NA      NA       NA
## I(x^5)       -803.533255         NA      NA       NA
## I(x^6)        156.982335         NA      NA       NA
## I(x^7)        -19.465675         NA      NA       NA
## I(x^8)          1.489665         NA      NA       NA
## I(x^9)         -0.064240         NA      NA       NA
## I(x^10)         0.001195         NA      NA       NA
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 10 and 0 DF,  p-value: NA
```

```r
xplot = seq(0, 10, by = 0.1)
lines(xplot, predict(fit, newdata = data.frame(x = xplot)),
      col = "dodgerblue", lwd = 2, lty = 1)
lines(xplot, predict(fit_perf, newdata = data.frame(x = xplot)),
      col = "darkorange", lwd = 2, lty = 2)
```



Notice in the summary, `R` could not calculate standard errors. This is a result of being "out" of degrees of freedom. With 11 β parameters and 11 data points, we use up all the degrees of freedom before we can estimate σ.

In this example, the true relationship is quadratic, but the order 10 polynomial's fit is "perfect". Next chapter we will focus on the trade-off between goodness of fit (minimizing errors) and complexity of model.

Suppose you work for an automobile manufacturer which makes a large luxury sedan. You would like to know how the car performs from a fuel efficiency standpoint when it is driven at various speeds. Instead of testing the car at every conceivable speed (which would be impossible) you create an experiment where the car is driven at speeds of interest in increments of 5 miles per hour.

Our goal then, is to fit a model to this data in order to be able to predict fuel efficiency when driving at certain speeds. The data from this example can be found in `fuel_econ.csv`.

```
econ = read.csv("data/fuel_econ.csv")
```

In this example, we will be frequently looking a the fitted versus residuals plot, so we *should* write a function to make our life easier, but this is left as an exercise for homework.

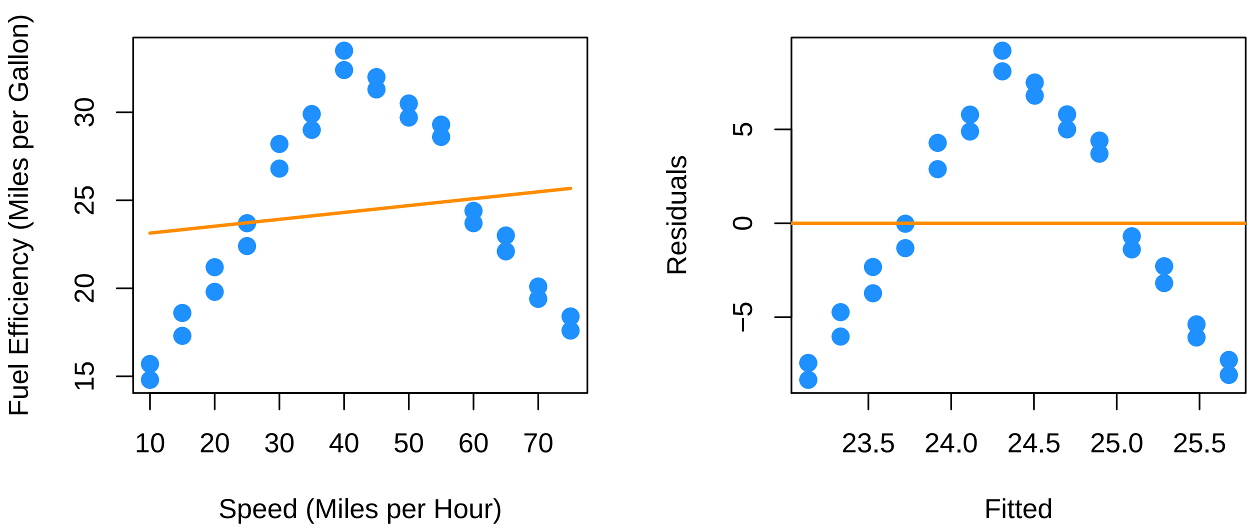
We will also be adding fitted curves to scatterplots repeatedly, so smartly we will write a function to do so.

```
plot_econ_curve = function(model){
  plot(mpg ~ mph, data = econ, xlab = "Speed (Miles per Hour)",
       ylab = "Fuel Efficiency (Miles per Gallon)", col = "dodgerblue",
       pch = 20, cex =2)
  xplot = seq(10, 75, by = 0.1)
  lines(xplot, predict(model, newdata = data.frame(mph = xplot)),
        col = "darkorange", lwd = 2, lty = 1)
}
```

So now we first fit a simple linear regression to this data.

```
fit1 = lm(mpg ~ mph, data = econ)
```

```
par(mfrow = c(1, 2))
plot_econ_curve(fit1)
plot(fitted(fit1), resid(fit1), xlab = "Fitted", ylab = "Residuals",
     col = "dodgerblue", pch = 20, cex =2)
  abline(h = 0, col = "darkorange", lwd = 2)
```



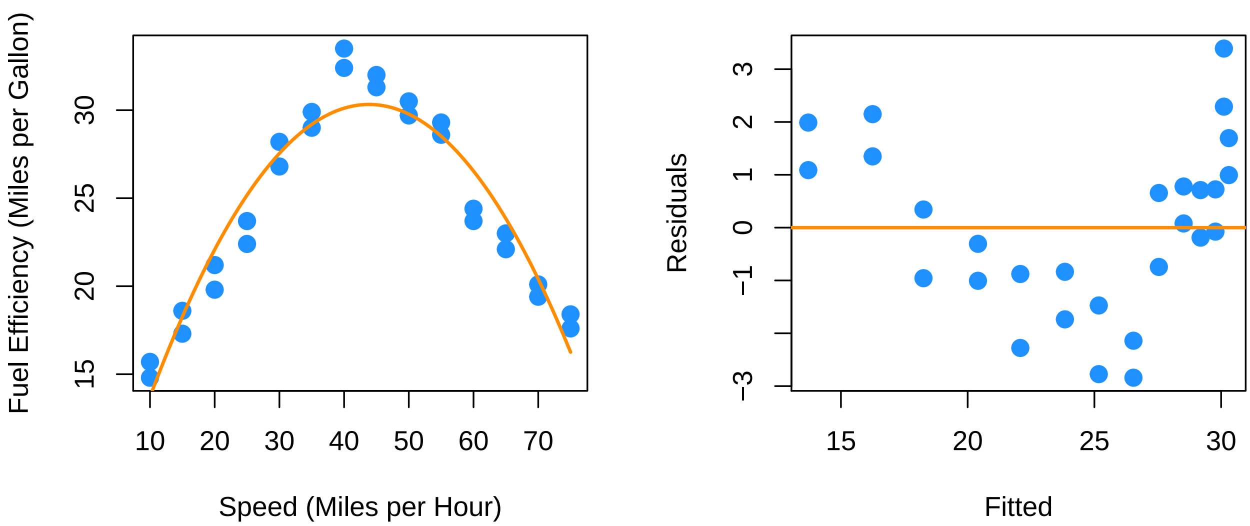
Pretty clearly we can do better. Yes fuel efficiency does increase as speed increases, but only up to a certain point.

We will now add polynomial terms until we fit a suitable fit.

```
fit2 = lm(mpg ~ mph + I(mph ^ 2), data = econ)
summary(fit2)
```

```
##
## Call:
## lm(formula = mpg ~ mph + I(mph^2), data = econ)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8411 -0.9694  0.0017  1.0181  3.3900
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.4444505  1.4241091   1.716   0.0984 .
## mph          1.2716937  0.0757321  16.792 3.99e-15 ***
## I(mph^2)    -0.0145014  0.0008719 -16.633 4.97e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.663 on 25 degrees of freedom
## Multiple R-squared:  0.9188, Adjusted R-squared:  0.9123
## F-statistic: 141.5 on 2 and 25 DF,  p-value: 2.338e-14
```

```r
par(mfrow = c(1, 2))
plot_econ_curve(fit2)
plot(fitted(fit2), resid(fit2), xlab = "Fitted", ylab = "Residuals",
     col = "dodgerblue", pch = 20, cex =2)
  abline(h = 0, col = "darkorange", lwd = 2)
```



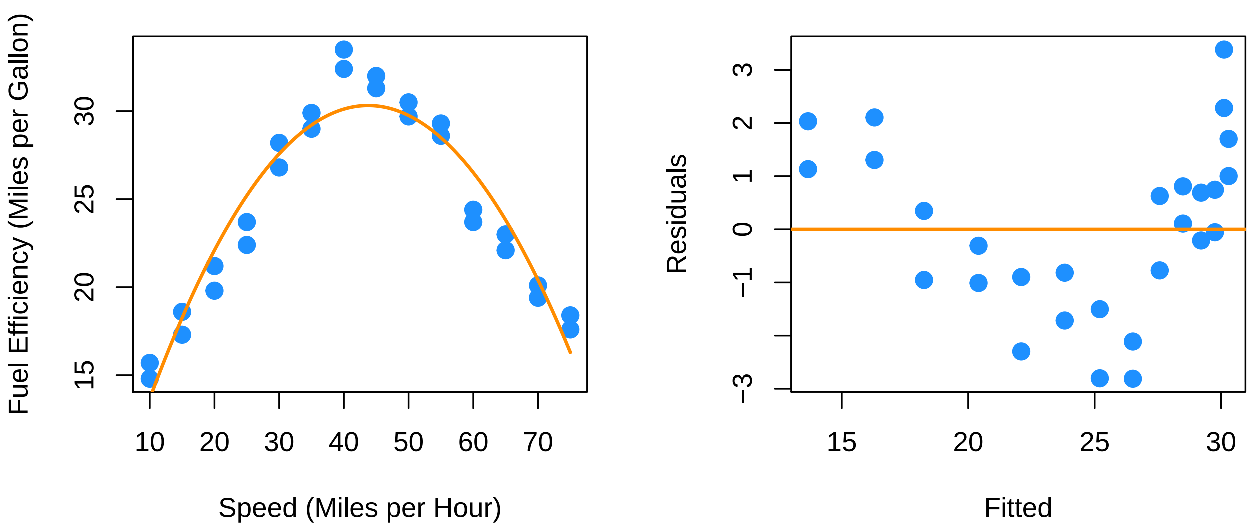
While this model clearly fits much better, and the second order term is significant, we still see a pattern in the fitted versus residuals plot which suggests higher order terms will help. Also, we would expect the curve to flatten as speed increases or decreases, not go sharply downward as we see here.

```r
fit3 = lm(mpg ~ mph + I(mph ^ 2) + I(mph ^ 3), data = econ)
summary(fit3)
```

```
##
## Call:
## lm(formula = mpg ~ mph + I(mph^2) + I(mph^3), data = econ)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8112 -0.9677  0.0264  1.0345  3.3827
##
## Coefficients:
##                 Estimate   Std. Error t value Pr(>|t|)
## (Intercept)  2.257842158  2.767928398   0.816   0.4227
## mph          1.290771239  0.252928479   5.103 0.000032 ***
## I(mph^2)    -0.015019730  0.006603861  -2.274   0.0322 *
## I(mph^3)     0.000004066  0.000051323   0.079   0.9375
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.697 on 24 degrees of freedom
## Multiple R-squared:  0.9188, Adjusted R-squared:  0.9087
## F-statistic: 90.56 on 3 and 24 DF,  p-value: 3.17e-13
```

```
par(mfrow = c(1, 2))
plot_econ_curve(fit3)
plot(fitted(fit3), resid(fit3), xlab = "Fitted", ylab = "Residuals",
     col = "dodgerblue", pch = 20, cex =2)
  abline(h = 0, col = "darkorange", lwd = 2)
```



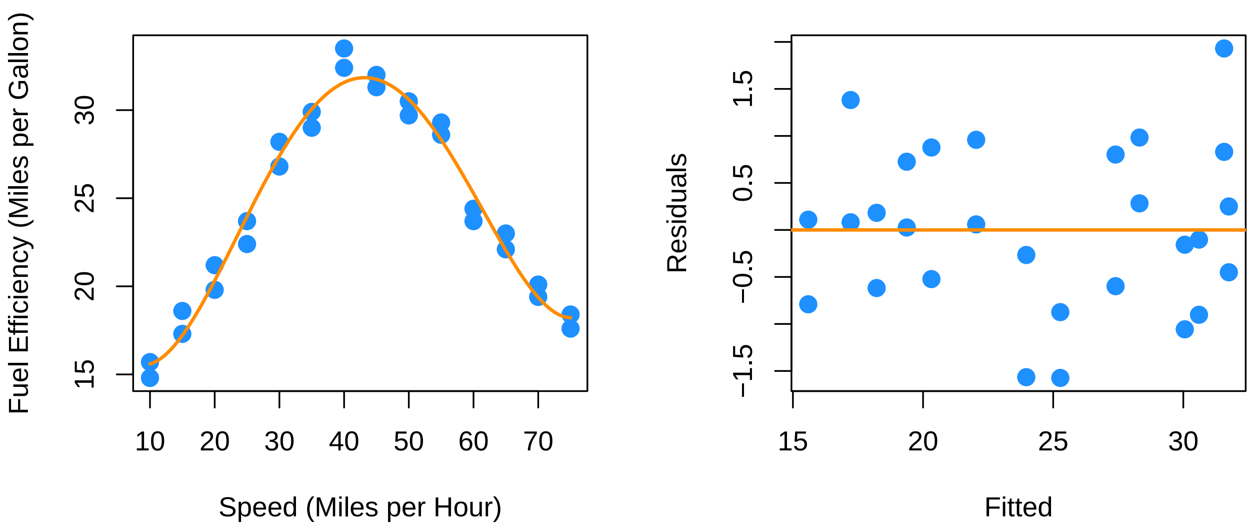
Adding the third order term doesn't seem to help at all. The fitted curve hardly changes. This makes sense, since what we would like is for the curve to flatten at the extremes. For this we will need an even degree polynomial term.

```
fit4 = lm(mpg ~ mph + I(mph ^ 2) + I(mph ^ 3) + I(mph ^ 4), data = econ)
summary(fit4)
```

```
##
## Call:
## lm(formula = mpg ~ mph + I(mph^2) + I(mph^3) + I(mph^4), data = econ)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -1.57410 -0.60308  0.04236  0.74481  1.93038
##
## Coefficients:
##                Estimate   Std. Error t value    Pr(>|t|)
## (Intercept) 21.460464535  2.964796563   7.238 0.000000228 ***
## mph         -1.467700706  0.391271891  -3.751     0.00104 **
## I(mph^2)     0.108111931  0.016728286   6.463 0.000001354 ***
## I(mph^3)    -0.002129559  0.000284392  -7.488 0.000000131 ***
## I(mph^4)     0.000012551  0.000001665   7.539 0.000000117 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9307 on 23 degrees of freedom
## Multiple R-squared:  0.9766, Adjusted R-squared:  0.9726
## F-statistic: 240.2 on 4 and 23 DF,  p-value: < 2.2e-16
```

```r
par(mfrow = c(1, 2))
plot_econ_curve(fit4)
plot(fitted(fit4), resid(fit4), xlab = "Fitted", ylab = "Residuals",
     col = "dodgerblue", pch = 20, cex =2)
  abline(h = 0, col = "darkorange", lwd = 2)
```



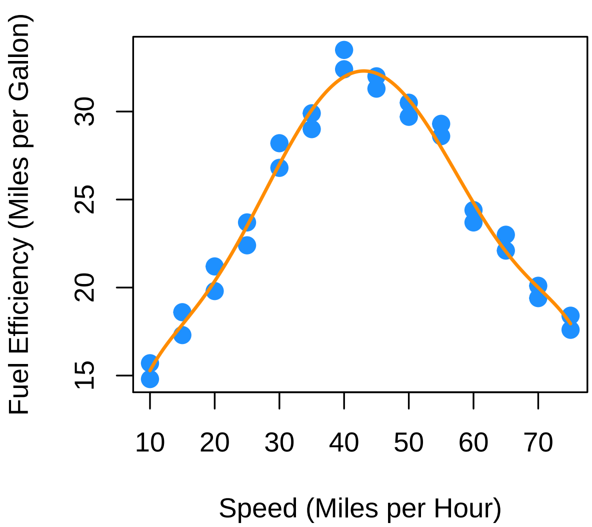
Now we are making progress. The fourth order term is significant with the other terms in the model. Also we are starting to see what we expected for low and high speed. However, there still seems to be a bit of a pattern in the residuals, so we will again try more higher order terms. We will add the fifth and sixth together, since adding the fifth will be similar to adding the third.

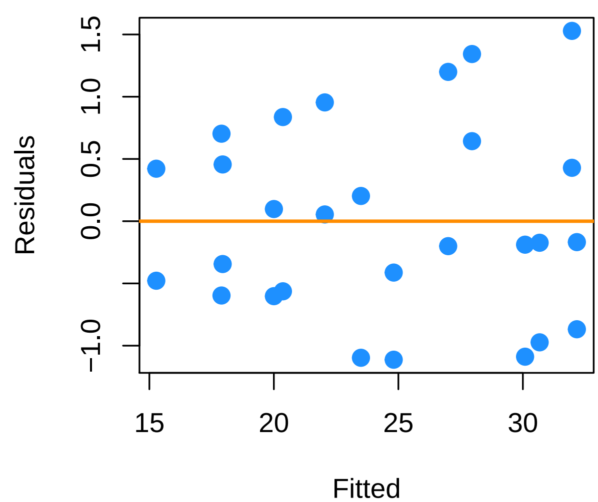
```r
fit6 = lm(mpg ~ mph + I(mph ^ 2) + I(mph ^ 3) + I(mph ^ 4) + I(mph ^ 5) + I(mph^6), data = econ)
summary(fit6)
```

```
##
## Call:
## lm(formula = mpg ~ mph + I(mph^2) + I(mph^3) + I(mph^4) + I(mph^5) +
##     I(mph^6), data = econ)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1129 -0.5717 -0.1707  0.5026  1.5288
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.206e+00  1.204e+01  -0.349   0.7304
## mph          4.203e+00  2.553e+00   1.646   0.1146
## I(mph^2)    -3.521e-01  2.012e-01  -1.750   0.0947 .
## I(mph^3)     1.579e-02  7.691e-03   2.053   0.0527 .
## I(mph^4)    -3.473e-04  1.529e-04  -2.271   0.0338 *
## I(mph^5)     3.585e-06  1.518e-06   2.362   0.0279 *
## I(mph^6)    -1.402e-08  5.941e-09  -2.360   0.0280 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8657 on 21 degrees of freedom
## Multiple R-squared:  0.9815, Adjusted R-squared:  0.9762
## F-statistic:   186 on 6 and 21 DF,  p-value: < 2.2e-16
```

```
par(mfrow = c(1, 2))
plot_econ_curve(fit6)
plot(fitted(fit6), resid(fit6), xlab = "Fitted", ylab = "Residuals",
     col = "dodgerblue", pch = 20, cex =2)
  abline(h = 0, col = "darkorange", lwd = 2)
```





Again the sixth order term is significant with the other terms in the model and here we see less pattern in the residuals plot. Let's now test for which of the previous two models we prefer. We will test

$$H_0 : \beta_5 = \beta_6 = 0.$$

```
anova(fit4, fit6)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ mph + I(mph^2) + I(mph^3) + I(mph^4)
```

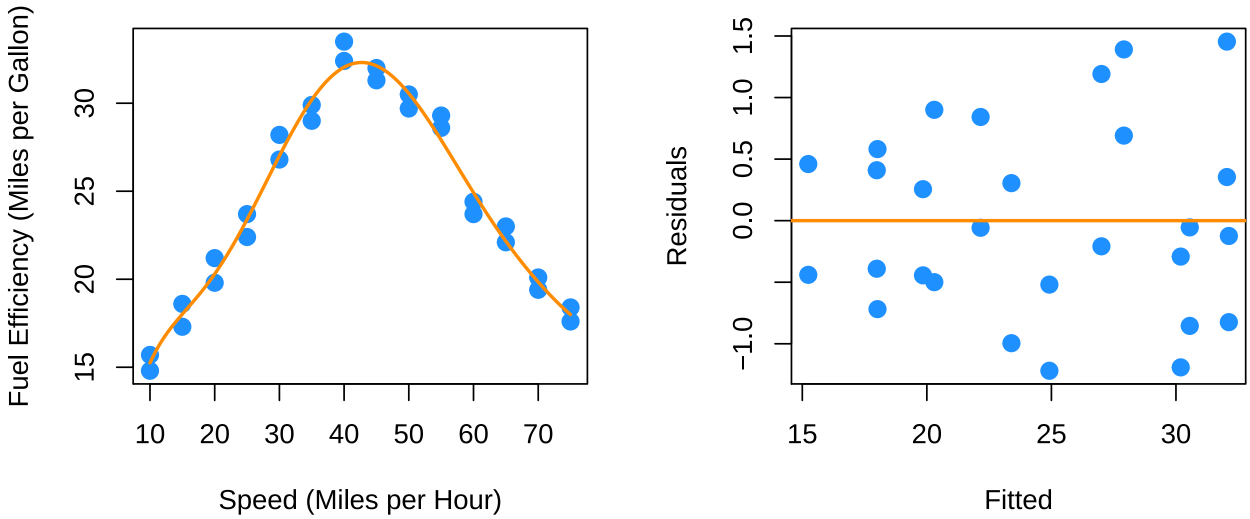
```
## Model 2: mpg ~ mph + I(mph^2) + I(mph^3) + I(mph^4) + I(mph^5) + I(mph^6)
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     23 19.922
## 2     21 15.739  2    4.1828 2.7905 0.0842 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

So, this test does not reject the null hypothesis at a level of significance of $\alpha = 0.05$, however the p-value is still rather small, and the fitted versus residuals plot is much better for the model with the sixth order term. This makes the sixth order model a good choice. We could repeat this process one more time.

```r
fit8 = lm(mpg ~ mph + I(mph ^ 2) + I(mph ^ 3) + I(mph ^ 4) + I(mph ^ 5)
          + I(mph ^ 6) + I(mph ^ 7) + I(mph ^ 8), data = econ)
summary(fit8)
```

```
##
## Call:
## lm(formula = mpg ~ mph + I(mph^2) + I(mph^3) + I(mph^4) + I(mph^5) +
##     I(mph^6) + I(mph^7) + I(mph^8), data = econ)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.21938 -0.50464 -0.09105  0.49029  1.45440
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.202e+01  7.045e+01  -0.171    0.866
## mph          6.021e+00  2.014e+01   0.299    0.768
## I(mph^2)    -5.037e-01  2.313e+00  -0.218    0.830
## I(mph^3)     2.121e-02  1.408e-01   0.151    0.882
## I(mph^4)    -4.008e-04  5.017e-03  -0.080    0.937
## I(mph^5)     1.789e-06  1.080e-04   0.017    0.987
## I(mph^6)     4.486e-08  1.381e-06   0.032    0.974
## I(mph^7)    -6.456e-10  9.649e-09  -0.067    0.947
## I(mph^8)     2.530e-12  2.835e-11   0.089    0.930
##
## Residual standard error: 0.9034 on 19 degrees of freedom
## Multiple R-squared:  0.9818, Adjusted R-squared:  0.9741
## F-statistic: 128.1 on 8 and 19 DF,  p-value: 7.074e-15
```

```r
par(mfrow = c(1, 2))
plot_econ_curve(fit8)
plot(fitted(fit8), resid(fit8), xlab = "Fitted", ylab = "Residuals",
     col = "dodgerblue", pch = 20, cex =2)
  abline(h = 0, col = "darkorange", lwd = 2)
```




```r
summary(fit8)
```

```
##
## Call:
## lm(formula = mpg ~ mph + I(mph^2) + I(mph^3) + I(mph^4) + I(mph^5) +
##     I(mph^6) + I(mph^7) + I(mph^8), data = econ)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.21938 -0.50464 -0.09105  0.49029  1.45440
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.202e+01  7.045e+01  -0.171    0.866
## mph          6.021e+00  2.014e+01   0.299    0.768
## I(mph^2)    -5.037e-01  2.313e+00  -0.218    0.830
## I(mph^3)     2.121e-02  1.408e-01   0.151    0.882
## I(mph^4)    -4.008e-04  5.017e-03  -0.080    0.937
## I(mph^5)     1.789e-06  1.080e-04   0.017    0.987
## I(mph^6)     4.486e-08  1.381e-06   0.032    0.974
## I(mph^7)    -6.456e-10  9.649e-09  -0.067    0.947
## I(mph^8)     2.530e-12  2.835e-11   0.089    0.930
##
## Residual standard error: 0.9034 on 19 degrees of freedom
## Multiple R-squared:  0.9818, Adjusted R-squared:  0.9741
## F-statistic: 128.1 on 8 and 19 DF,  p-value: 7.074e-15
```

```r
anova(fit6, fit8)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ mph + I(mph^2) + I(mph^3) + I(mph^4) + I(mph^5) + I(mph^6)
## Model 2: mpg ~ mph + I(mph^2) + I(mph^3) + I(mph^4) + I(mph^5) + I(mph^6) +
##     I(mph^7) + I(mph^8)
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     21 15.739
```

```
## 2     19 15.506  2    0.2324 0.1424 0.8682
```

Here we would clearly stick with `fit6`. The eighth order term is not significant with the other terms in the model and the F-test does not reject.

As an aside, be aware that there is a quicker way to specify a model with many higher order terms.

```r
fit6_alt = lm(mpg ~ poly(mph, 6), data = econ)
all.equal(fitted(fit6), fitted(fit6_alt))
```

```
## [1] TRUE
```

We first verify that this method produces the same fitted values. However, the estimated coefficients are different.

```r
coef(fit6)
```

```
##       (Intercept)               mph           I(mph^2)           I(mph^3)
## -4.20622377616269  4.20338221905924 -0.35214523989512  0.01579340288449
##          I(mph^4)          I(mph^5)           I(mph^6)
## -0.00034726647879  0.00000358520124 -0.00000001401995
```

```r
coef(fit6_alt)
```

```
##   (Intercept) poly(mph, 6)1 poly(mph, 6)2 poly(mph, 6)3 poly(mph, 6)4
##   24.40714286    4.16769628  -27.66685755    0.13446747    7.01671480
## poly(mph, 6)5 poly(mph, 6)6
##    0.09288754   -2.04307796
```

This is because `poly()` uses *orthogonal polynimials*, which solves an issue we will discuss in the next chapter.

```r
summary(fit6)
```

```
## 
## Call:
## lm(formula = mpg ~ mph + I(mph^2) + I(mph^3) + I(mph^4) + I(mph^5) + 
##     I(mph^6), data = econ)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -1.1129 -0.5717 -0.1707  0.5026  1.5288 
## 
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)  
## (Intercept) -4.206e+00  1.204e+01  -0.349   0.7304  
## mph          4.203e+00  2.553e+00   1.646   0.1146  
## I(mph^2)    -3.521e-01  2.012e-01  -1.750   0.0947 .
## I(mph^3)     1.579e-02  7.691e-03   2.053   0.0527 .
## I(mph^4)    -3.473e-04  1.529e-04  -2.271   0.0338 *
## I(mph^5)     3.585e-06  1.518e-06   2.362   0.0279 *
## I(mph^6)    -1.402e-08  5.941e-09  -2.360   0.0280 *
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8657 on 21 degrees of freedom
## Multiple R-squared:  0.9815, Adjusted R-squared:  0.9762
## F-statistic:   186 on 6 and 21 DF,  p-value: < 2.2e-16
```

```
summary(fit6_alt)
```

```
##
## Call:
## lm(formula = mpg ~ poly(mph, 6), data = econ)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1129 -0.5717 -0.1707  0.5026  1.5288
##
## Coefficients:
##                 Estimate Std. Error t value     Pr(>|t|)
## (Intercept)     24.40714    0.16360 149.184      < 2e-16 ***
## poly(mph, 6)1   4.16770    0.86571   4.814 0.0000930613 ***
## poly(mph, 6)2 -27.66686    0.86571 -31.958      < 2e-16 ***
## poly(mph, 6)3   0.13447    0.86571   0.155        0.878
## poly(mph, 6)4   7.01671    0.86571   8.105 0.0000000668 ***
## poly(mph, 6)5   0.09289    0.86571   0.107        0.916
## poly(mph, 6)6  -2.04308    0.86571  -2.360        0.028 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8657 on 21 degrees of freedom
## Multiple R-squared:  0.9815, Adjusted R-squared:  0.9762
## F-statistic:   186 on 6 and 21 DF,  p-value: < 2.2e-16
```

Notice though that the p-value for testing the degree 6 term is the same. Because of this, for the most part we can use these interchangeably.

To use `poly()` to obtain the same results as using `I()` repeatedly, we would need to set `raw = TRUE`.

```
fit6_alt2 = lm(mpg ~ poly(mph, 6, raw = TRUE), data = econ)
coef(fit6_alt2)
```

```
##              (Intercept) poly(mph, 6, raw = TRUE)1 poly(mph, 6, raw = TRUE)2
##        -4.20622377616269          4.20338221905924         -0.35214523989512
## poly(mph, 6, raw = TRUE)3 poly(mph, 6, raw = TRUE)4 poly(mph, 6, raw = TRUE)5
##         0.01579340288449         -0.00034726647879          0.00000358520124
## poly(mph, 6, raw = TRUE)6
##        -0.00000001401995
```

We've now seen how to transform predictor and response variables. In this chapter we have mostly focused on using this in the context of fixing SLR models. However, these concepts can easily be used together with categorical variables and interactions to build larger, more flexible models. In the next chapter, we will discuss how to choose a good model from a collection of possible models.

Please note: some data currently used in this chapter was used, changed, and passed around over the years in STAT 420 at UIUC. Its original sources, if they exist, are at this time unknown to the author. As a result, they should only be considered for use with STAT 420 in Summer 2016. Going forward they will likely be replaced with alternative sourceable data that illustrates the same concepts.

Chapter 9

Collinearity

> "If I look confused it is because I am thinking."
>
> — **Samuel Goldwyn**

After reading this chapter you will be able to:

- Identify collinearity in regression.
- Understand the effect of collinearity on regression models.

9.1 Exact Collinearity

Let's create a dataset where one of the predictors, x_3, is a linear combination of the other predictors.

```
gen_exact_collin_data = function(num_samples = 100) {
  x1 = rnorm(n = num_samples, mean = 80, sd = 10)
  x2 = rnorm(n = num_samples, mean = 70, sd = 5)
  x3 = 2 * x1 + 4 * x2 + 3
  y = 3 + x1 + x2 + rnorm(n = num_samples, mean = 0, sd = 1)
  data.frame(y, x1, x2, x3)
}
```

Notice that the way we are generating this data, the response y only really depends on x_1 and x_2.

```
set.seed(42)
exact_collin_data = gen_exact_collin_data()
head(exact_collin_data)
```

```
##          y       x1       x2       x3
## 1 170.7135 93.70958 76.00483 494.4385
## 2 152.9106 74.35302 75.22376 452.6011
## 3 152.7866 83.63128 64.98396 430.1984
## 4 170.6306 86.32863 79.24241 492.6269
## 5 152.3320 84.04268 66.66613 437.7499
## 6 151.3155 78.93875 70.52757 442.9878
```

What happens when we attempt to fit a regression model in R using all of the predictors?

```
exact_collin_fit = lm(y ~ x1 + x2 + x3, data = exact_collin_data)
summary(exact_collin_fit)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2 + x3, data = exact_collin_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.57662 -0.66188 -0.08253  0.63706  2.52057
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.957336   1.735165   1.704   0.0915 .
## x1          0.985629   0.009788 100.702   <2e-16 ***
## x2          1.017059   0.022545  45.112   <2e-16 ***
## x3                NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.014 on 97 degrees of freedom
## Multiple R-squared:  0.9923, Adjusted R-squared:  0.9921
## F-statistic:  6236 on 2 and 97 DF,  p-value: < 2.2e-16
```

We see that R simply decides to exclude a variable. Why is this happening?

```
X = cbind(1, as.matrix(exact_collin_data[,-1]))
solve(t(X) %*% X)
```

If we attempt to find $\hat{\boldsymbol{\beta}}$ using $\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}$, we see that this is not possible, due to the fact that the columns of $\boldsymbol{X}$ are linearly dependent. The previous lines of code were not run, because they produce an error!

When this happens, we say there is **exact collinearity** in the dataset.

As a result of this issue, R essentially chose to fit the model `y ~ x1 + x2`. However notice that two other models would accomplish exactly the same fit.

```
fit1 = lm(y ~ x1 + x2, data = exact_collin_data)
fit2 = lm(y ~ x1 + x3, data = exact_collin_data)
fit3 = lm(y ~ x2 + x3, data = exact_collin_data)
```

We see that the fitted values for each of the three models are exactly the same. This is a result of x_3 containing all of the information from x_1 and x_2. As long as one of x_1 or x_2 are included in the model, x_3 can be used to recover the information from the variable not included.

```
all.equal(fitted(fit1), fitted(fit2))
```

```
## [1] TRUE
```

```r
all.equal(fitted(fit2), fitted(fit3))
```

```
## [1] TRUE
```

While their fitted values are all the same, their estimated coefficients are wildly different. The sign of x_2 is switched in two of the models! So only `fit1` properly *explains* the relationship between the variables, `fit2` and `fit3` still *predict* as well as `fit1`, despite the coefficients having little to no meaning, a concept we will return to later.

```r
coef(fit1)
```

```
## (Intercept)          x1          x2
##   2.9573357   0.9856291   1.0170586
```

```r
coef(fit2)
```

```
## (Intercept)          x1          x3
##   2.1945418   0.4770998   0.2542647
```

```r
coef(fit3)
```

```
## (Intercept)          x2          x3
##   1.4788921  -0.9541995   0.4928145
```

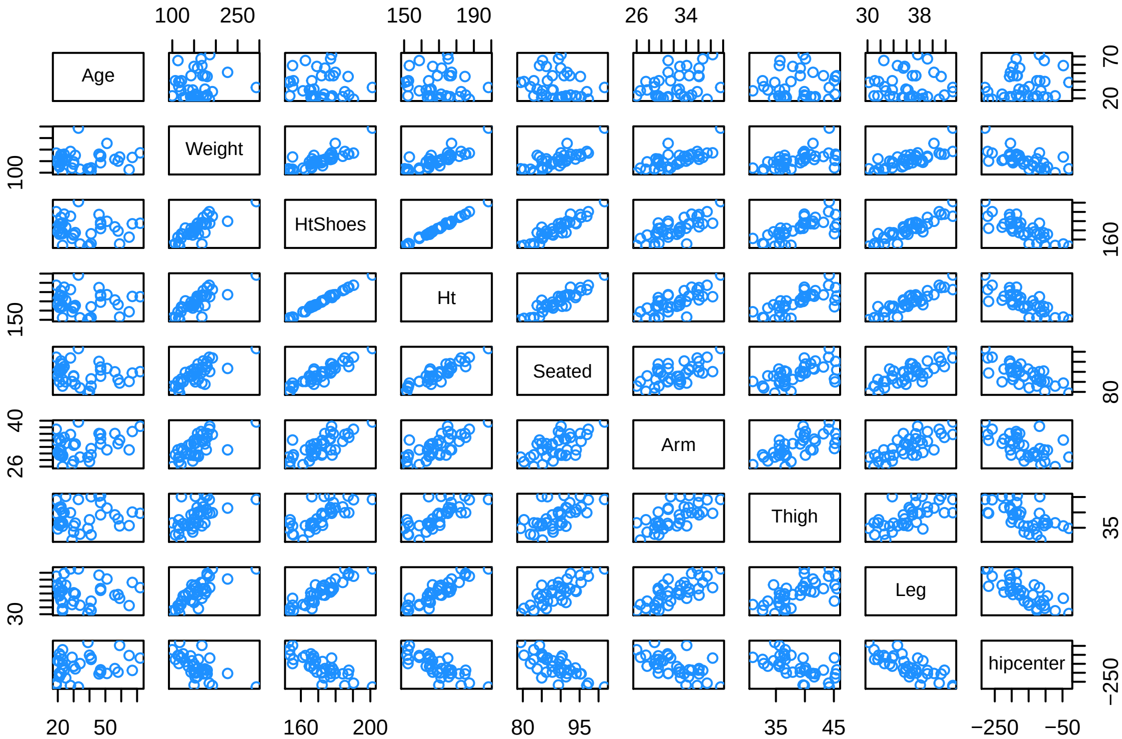
9.2 Collinearity

Exact collinearity is an extreme example of **collinearity**, which occurs in multiple regression when predictor variables are highly correlated. Collinearity is often called *multicollinearity*, since it is a phenomenon that really only occurs during multiple regression.

Looking at the `seatpos` dataset from the `faraway` package, we will see an example of this concept. The predictors in this dataset are various attributes of car drivers, such as their height, weight and age. The response variable `hipcenter` measures the "horizontal distance of the midpoint of the hips from a fixed location in the car in mm." Essentially, it measures the position of the seat for a given driver. This is potentially useful information for car manufacturers considering comfort and safety when designing vehicles.

We will attempt to fit a model that predicts `hipcenter`. Two predictor variables are immediately interesting to us: `HtShoes` and `Ht`. We certainly expect a person's height to be highly correlated to their height when wearing shoes. We'll pay special attention to these two variables when fitting models.

```r
library(faraway)
pairs(seatpos, col = "dodgerblue")
```




```r
round(cor(seatpos), 2)
```

```
##              Age Weight HtShoes    Ht Seated   Arm Thigh   Leg hipcenter
## Age         1.00   0.08   -0.08 -0.09  -0.17  0.36  0.09 -0.04      0.21
## Weight      0.08   1.00    0.83  0.83   0.78  0.70  0.57  0.78     -0.64
## HtShoes    -0.08   0.83    1.00  1.00   0.93  0.75  0.72  0.91     -0.80
## Ht         -0.09   0.83    1.00  1.00   0.93  0.75  0.73  0.91     -0.80
## Seated     -0.17   0.78    0.93  0.93   1.00  0.63  0.61  0.81     -0.73
## Arm         0.36   0.70    0.75  0.75   0.63  1.00  0.67  0.75     -0.59
## Thigh       0.09   0.57    0.72  0.73   0.61  0.67  1.00  0.65     -0.59
## Leg        -0.04   0.78    0.91  0.91   0.81  0.75  0.65  1.00     -0.79
## hipcenter   0.21  -0.64   -0.80 -0.80  -0.73 -0.59 -0.59 -0.79      1.00
```

After loading the `faraway` package, we do some quick checks of correlation between the predictors. Visually, we can do this with the `pairs()` function, which plots all possible scatterplots between pairs of variables in the dataset.

We can also do this numerically with the `cor()` function, which when applied to a dataset, returns all pairwise correlations. Notice this is a symmetric matrix. Recall that correlation measures strength and direction of the linear relationship between to variables. The correlation between `Ht` and `HtShoes` is extremely high. So high, that rounded to two decimal places, it appears to be 1!

Unlike exact collinearity, here we can still fit a model with all of the predictors, but what effect does this have?

```r
hip_model = lm(hipcenter ~ ., data = seatpos)
summary(hip_model)
```

```
##
## Call:
```

```
## lm(formula = hipcenter ~ ., data = seatpos)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -73.827 -22.833  -3.678  25.017  62.337
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 436.43213  166.57162   2.620   0.0138 *
## Age           0.77572    0.57033   1.360   0.1843
## Weight        0.02631    0.33097   0.080   0.9372
## HtShoes      -2.69241    9.75304  -0.276   0.7845
## Ht            0.60134   10.12987   0.059   0.9531
## Seated        0.53375    3.76189   0.142   0.8882
## Arm          -1.32807    3.90020  -0.341   0.7359
## Thigh        -1.14312    2.66002  -0.430   0.6706
## Leg          -6.43905    4.71386  -1.366   0.1824
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 37.72 on 29 degrees of freedom
## Multiple R-squared:  0.6866, Adjusted R-squared:  0.6001
## F-statistic:  7.94 on 8 and 29 DF,  p-value: 0.00001306
```

One of the first things we should notice is that the F-test for the regression tells us that the regression is significant, however each individual predictor is not. Another interesting result is the opposite signs of the coefficients for `Ht` and `HtShoes`. This should seem rather counter-intuitive. Increasing `Ht` increases `hipcenter`, but increasing `HtShoes` decreases `hipcenter`?

This happens as a result of the predictors being highly correlated. For example, the `HtShoe` variable explains a large amount of the variation in `Ht`. When they are both in the model, their effects on the response are lessened individually, but together they still explain a large portion of the variation of `hipcenter`.

We define R_j^2 to be the proportion of observed variation in the j-th predictor explained by the other predictors. In other words R_j^2 is the multiple R-Squared for the regression of x_j on each of the other predictors.

```r
ht_shoes_model = lm(HtShoes ~ . - hipcenter, data = seatpos)
summary(ht_shoes_model)$r.squared
```

```
## [1] 0.9967472
```

Here we see that the other predictors explain 99.67% of the variation in `HtShoe`. When fitting this model, we removed `hipcenter` since it is not a predictor.

9.2.1 Variance Inflation Factor.

Now note that the variance of $\hat{\beta}_j$ can be written as

$$Var(\hat{\beta}_j) = \sigma^2 C_{jj} = \sigma^2 \left(\frac{1}{1 - R_j^2} \right) \frac{1}{S_{x_j x_j}}$$

where $S_{x_j x_j} = \sum (x_{ij} - \bar{x}_j)^2$. This gives us a way to understand how collinearity affects our regression estimates.

We will call,

$$\frac{1}{1 - R_j^2}$$

the **variance inflation factor.** The variance inflation factor quantifies the effect of collinearity on the variance of our regression estimates. When R_j^2 is large, that is close to 1, x_j is well explained by the other predictors. With a large R_j^2 the variance inflation factor becomes large. This tells us that when x_j is highly correlated with other predictors, our estimate of β_j is highly variable.

The `vif` function from the `faraway` package calculates the VIFs for each of the predictors of a model.

```
vif(hip_model)
```

```
##        Age     Weight    HtShoes         Ht     Seated        Arm      Thigh
##   1.997931   3.647030 307.429378 333.137832   8.951054   4.496368   2.762886
##        Leg
##   6.694291
```

In practice it is common to say that any VIF greater than 5 is cause for concern. So in this example we see there is a huge multicollinearity issue as many of the predictors have a VIF greater than 5.

Let's further investigate how the presence of collinearity actually effects a model. If we add a small amount of noise to the data, we see that the estimates of the coefficients change drastically. This is a rather undesirable effect. Adding random noise should not effect the coefficients of a model.

```
hip_model_noise = lm(hipcenter + rnorm(38, mean = 0, sd = 10) ~ ., data = seatpos)
```

Adding the noise had such a large effect, the sign of the coefficient for `Ht` has changed.

```
hip_model
```

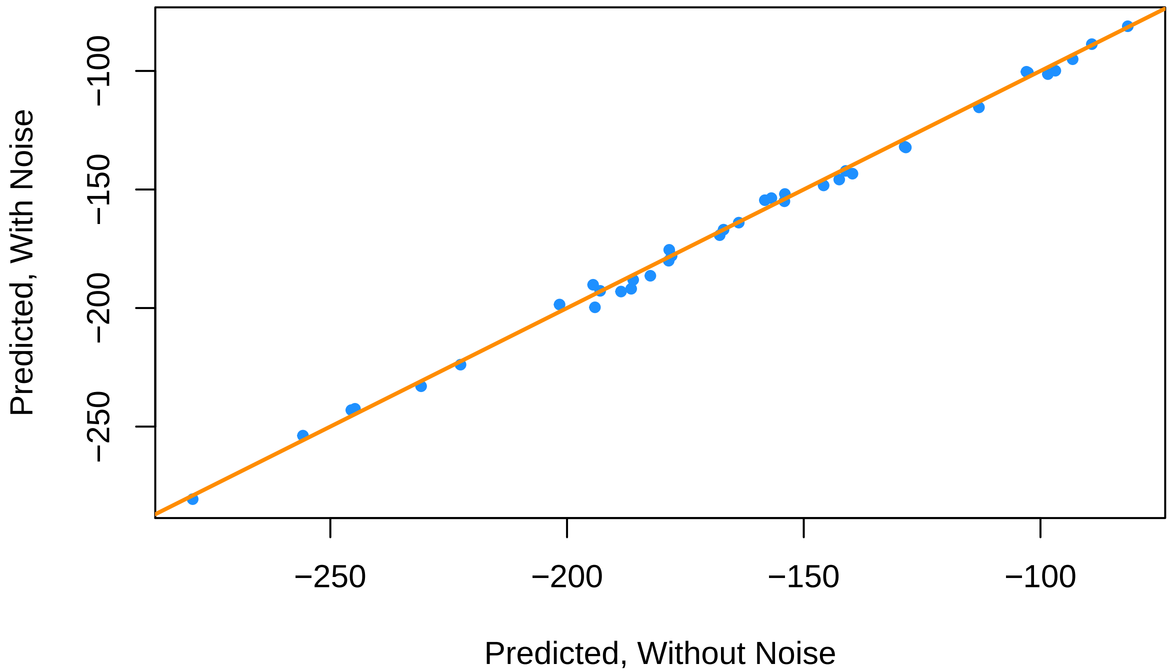
```
##
## Call:
## lm(formula = hipcenter ~ ., data = seatpos)
##
## Coefficients:
## (Intercept)          Age       Weight      HtShoes           Ht       Seated
##   436.43213      0.77572      0.02631     -2.69241      0.60134      0.53375
##         Arm        Thigh          Leg
##    -1.32807     -1.14312     -6.43905
```

```
hip_model_noise
```

```
##
## Call:
## lm(formula = hipcenter + rnorm(38, mean = 0, sd = 10) ~ ., data = seatpos)
##
## Coefficients:
## (Intercept)          Age       Weight      HtShoes           Ht       Seated
##   462.93357      0.63646      0.01359     -2.13828     -0.13234      0.04610
##         Arm        Thigh          Leg
##    -0.20175     -1.40641     -5.72109
```

This tells us that a model with collinearity is bad at explaining the relationship between the response and the predictors. We cannot even be confident in the direction of the relationship. However, does collinearity effect prediction?

```
plot(fitted(hip_model), fitted(hip_model_noise), col = "dodgerblue", pch = 20,
     xlab = "Predicted, Without Noise", ylab = "Predicted, With Noise")
abline(a = 0, b = 1, col = "darkorange", lwd = 2)
```



We see that by plotting the predicted values using both models against each other, they are actually rather similar.

Let's now look at a smaller model,

```
hip_model_small = lm(hipcenter ~ Age + Weight + Ht, data = seatpos)
summary(hip_model_small)
```

```
##
## Call:
## lm(formula = hipcenter ~ Age + Weight + Ht, data = seatpos)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -91.526 -23.005   2.164  24.950  53.982
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 528.297729 135.312947   3.904 0.000426 ***
## Age           0.519504   0.408039   1.273 0.211593
## Weight        0.004271   0.311720   0.014 0.989149
## Ht           -4.211905   0.999056  -4.216 0.000174 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 36.49 on 34 degrees of freedom
## Multiple R-squared:  0.6562, Adjusted R-squared:  0.6258
## F-statistic: 21.63 on 3 and 34 DF,  p-value: 0.00000005125
```

```r
vif(hip_model_small)
```

```
##      Age   Weight       Ht
## 1.093018 3.457681 3.463303
```

Immediately we see that multicollinearity isn't an issue here.

```r
anova(hip_model_small, hip_model)
```

```
## Analysis of Variance Table
##
## Model 1: hipcenter ~ Age + Weight + Ht
## Model 2: hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh +
##     Leg
##   Res.Df   RSS Df Sum of Sq      F Pr(>F)
## 1     34 45262
## 2     29 41262  5    4000.3 0.5623 0.7279
```

Also notice that using an F-test to compare the two models, we would prefer the smaller model.

We now investigate the effect of adding another variable to this smaller model. Specifically we want to look at adding the variable `HtShoes`. So now our possible predictors are `HtShoes`, `Age`, `Weight`, and `Ht`. Our response is still `hipcenter`.

To quantify this effect we will look at a **variable added plot** and a **partial correlation coefficient**. For both of these, we will look at the residuals of two models:

- Regressing the response (`hipcenter`) against all of the predictors except the predictor of interest (`HtShoes`).
- Regressing the predictor of interest (`HtShoes`) against the other predictors (`Age`, `Weight`, and `Ht`).

```r
ht_shoes_model_small = lm(HtShoes ~ Age + Weight + Ht, data = seatpos)
```

So now, the residuals of `hip_model_small` give us the variation of `hipcenter` that is *unexplained* by `Age`, `Weight`, and `Ht`. Similarly, the residuals of `ht_shoes_model_small` give us the variation of `HtShoes` unexplained by `Age`, `Weight`, and `Ht`.

The correlation of these two residuals gives us the **partial correlation coefficient** of `HtShoes` and `hipcenter` with the effects of `Age`, `Weight`, and `Ht` removed.

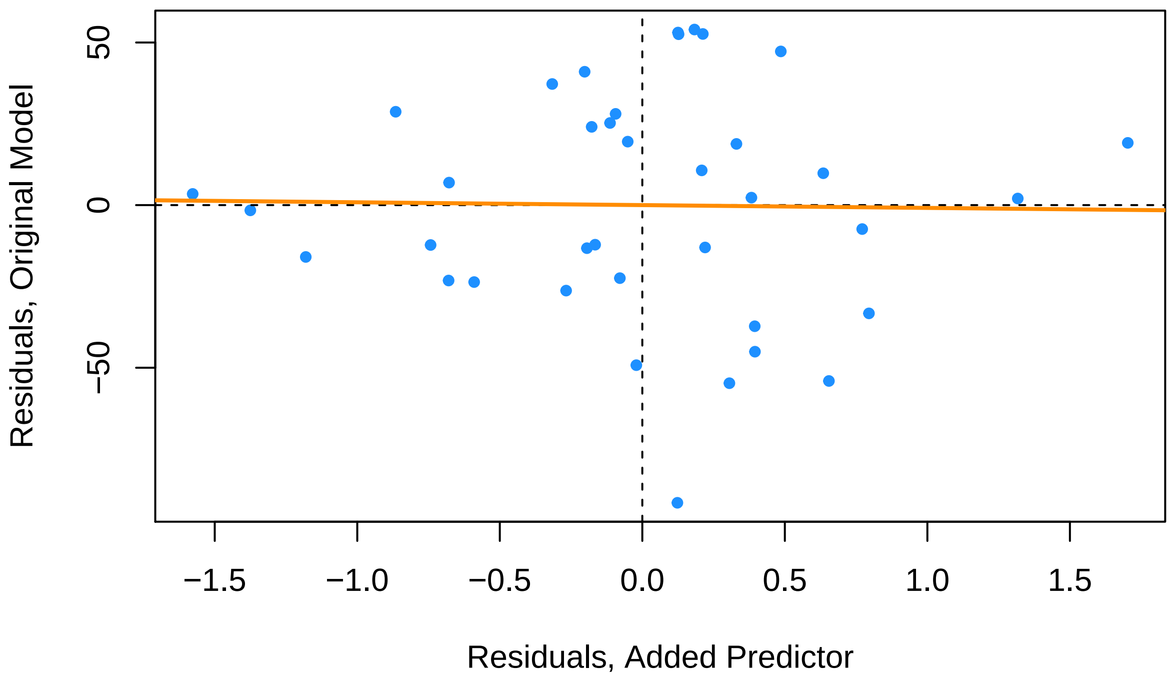
```r
cor(resid(ht_shoes_model_small), resid(hip_model_small))
```

```
## [1] -0.01650317
```

Since this value is small, close to zero, it means that the variation of `hipcenter` that is unexplained by `Age`, `Weight`, and `Ht` shows very little correlation with the variation of `HtShoes` that is not explained by `Age`, `Weight`, and `Ht`. Thus adding `HtShoes` to the model would likely be of little benefit.

Similarly a **variable added plot** visualizes these residuals against each other. It is also helpful to regress the residuals of the response against the residuals of the predictor and add the regression line to the plot.

```r
plot(resid(hip_model_small) ~ resid(ht_shoes_model_small), col = "dodgerblue", pch = 20,
     xlab = "Residuals, Added Predictor", ylab = "Residuals, Original Model")
abline(h = 0, lty = 2)
abline(v = 0, lty = 2)
abline(lm(resid(hip_model_small) ~ resid(ht_shoes_model_small)),
       col = "darkorange", lwd = 2)
```



Here the variable added plot shows almost no linear relationship. This tells us that adding `HtShoes` to the model would probably not be worthwhile. Since its variation is largely explained by the other predictors, adding it to the model will not do much to improve the model. However it will increase the variation of the estimates and make the model much harder to interpret.

Had there been a strong linear relationship here, thus a large partial correlation coefficient, it would likely have been useful to add the additional predictor to the model.

This trade off is mostly true in general. As a model gets more predictors, errors will get smaller and its *prediction* will be better, but it will be harder to interpret. This is why, if we are interested in *explaining* the relationship between the predictors and the response, we often want a model that fits well, but with a small number of predictors with little correlation.

Next chapter we will learn about methods to find models that both fit well, but also have a small number of predictors. We will also discuss *overfitting*. Although, adding additional predictors will always make errors smaller, sometimes we will be "fitting the noise" and such a model will not generalize to additional observations well.

9.3 Simulation

Here we simulate examples data with and without collinearity. We will note the difference in the distribution of the estimates of the β parameters, in particular their variance. However, we will also notice the similarity in their MSE.

We will use the model,

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

where $\epsilon \sim N(\mu = 0, \sigma^2 = 25)$ and the β coefficients defined below.

```
set.seed(42)
beta_0 = 7
beta_1 = 3
beta_2 = 4
sigma  = 5
```

We will use a sample size of 10, and 2000 simulations for both situations.

```
sample_size = 10
num_sim     = 2000
```

We'll first consider the situation with a collinearity issue, so we manually create the two predictor variables.

```
x1 = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
x2 = c(1, 2, 3, 4, 5, 7, 6, 10, 9, 8)
```

```
sd(x1)
```

```
## [1] 3.02765
```

```
sd(x2)
```

```
## [1] 3.02765
```

```
cor(x1, x2)
```

```
## [1] 0.9393939
```

Notice that they have extremely high correlation.

```
true_line    = beta_0 + beta_1 * x1 + beta_2 * x2
beta_hat_bad = matrix(0, num_sim, 3)
mse_bad      = rep(0, num_sim)
```

We perform the simulation 2000 times, each time fitting a regression model, and storing the estimated coefficients and the MSE.

```
for (s in 1:num_sim) {
  y = true_line + rnorm(n = sample_size, mean = 0, sd = sigma)
  reg_out = lm(y ~ x1 + x2)
  beta_hat_bad[s, ] = coef(reg_out)
  mse_bad[s] = mean(resid(reg_out) ^ 2)
}
```

Now we move to the situation without a collinearity issue, so we again manually create the two predictor variables.

```r
x1 = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
x2 = c(9, 2, 7, 4, 5, 6, 3, 8, 1, 10)
```

Notice that the standard deviations of each are the same as before, however, now the correlation is extremely close to 0.

```r
sd(x1)
```

```
## [1] 3.02765
```

```r
sd(x2)
```

```
## [1] 3.02765
```

```r
cor(x1, x2)
```

```
## [1] 0.03030303
```

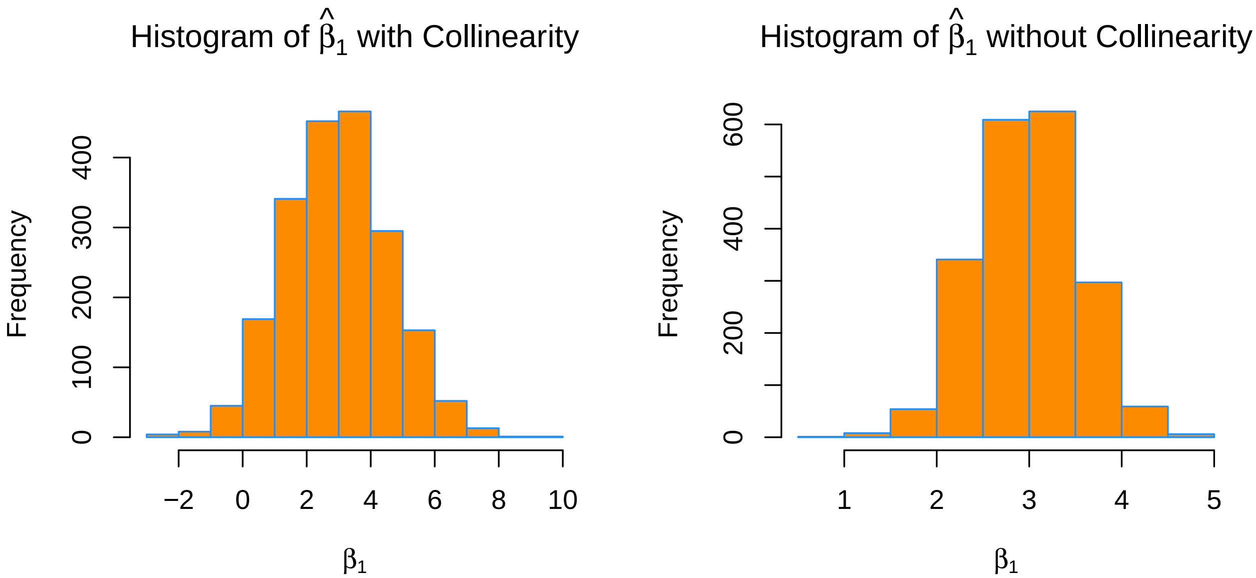
```r
true_line     = beta_0 + beta_1 * x1 + beta_2 * x2
beta_hat_good = matrix(0, num_sim, 3)
mse_good      = rep(0, num_sim)
```

We then perform simulations and store the same results.

```r
for (s in 1:num_sim) {
  y = true_line + rnorm(n = sample_size, mean = 0, sd = sigma)
  reg_out = lm(y ~ x1 + x2)
  beta_hat_good[s, ] = coef(reg_out)
  mse_good[s] = mean(resid(reg_out) ^ 2)
}
```

We'll now investigate the differences.

```r
par(mfrow = c(1, 2))
hist(beta_hat_bad[, 2],
     col = "darkorange",
     border = "dodgerblue",
     main = expression("Histogram of " *hat(beta)[1]* " with Collinearity"),
     xlab = expression(beta[1])
)
hist(beta_hat_good[, 2],
     col = "darkorange",
     border = "dodgerblue",
     main = expression("Histogram of " *hat(beta)[1]* " without Collinearity"),
     xlab = expression(beta[1])
)
```



First, for β_1, which has a true value of 3, we see that both with and without collinearity, the simulated values are centered near 3.

```
mean(beta_hat_bad[,2])
```

```
## [1] 2.969413
```

```
mean(beta_hat_good[,2])
```

```
## [1] 2.985792
```

The way the predictors were created, the $S_{x_j x_j}$ portion of the variance is the same for the predictors in both cases, but the variance is still much larger in the simulations performed with collinearity. The variance is so large in the collinear case, that sometimes the estimated coefficient for β_1 is negative!

```
sd(beta_hat_bad[,2])
```

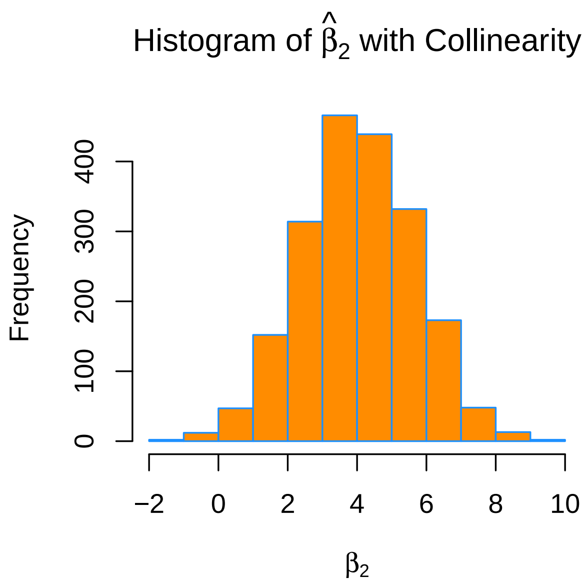
```
## [1] 1.631838
```

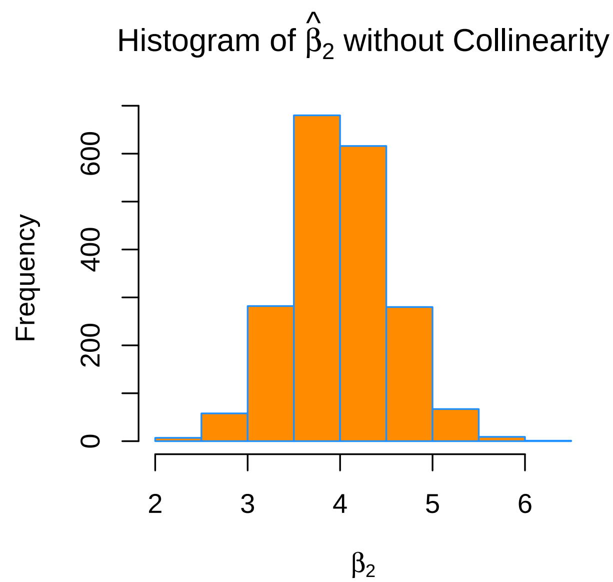
```
sd(beta_hat_good[,2])
```

```
## [1] 0.5572147
```

```
par(mfrow = c(1, 2))
hist(beta_hat_bad[, 3],
     col = "darkorange",
     border = "dodgerblue",
     main = expression("Histogram of " *hat(beta)[2]* " with Collinearity"),
     xlab = expression(beta[2])
)
hist(beta_hat_good[, 3],
     col = "darkorange",
```

```
      border = "dodgerblue",
      main = expression("Histogram of " *hat(beta)[2]* " without Collinearity"),
      xlab = expression(beta[2])
)
```





We see the same issues with β_2. On average the estimates are correct, but the variance is again much larger with collinearity.

```
mean(beta_hat_bad[,3])
```

```
## [1] 4.034139
```

```
mean(beta_hat_good[,3])
```

```
## [1] 4.001728
```

```
sd(beta_hat_bad[,3])
```

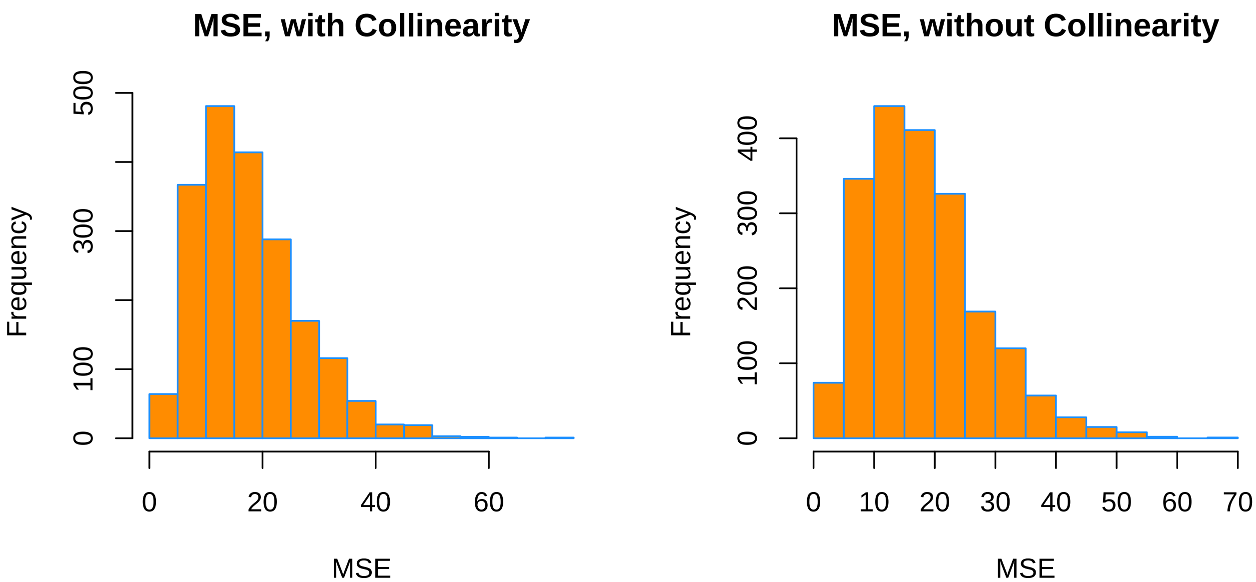
```
## [1] 1.640392
```

```
sd(beta_hat_good[,3])
```

```
## [1] 0.5533393
```

```
par(mfrow = c(1, 2))
hist(mse_bad,
     col = "darkorange",
     border = "dodgerblue",
     main = "MSE, with Collinearity",
     xlab = "MSE"
)
```

```r
hist(mse_good,
     col = "darkorange",
     border = "dodgerblue",
     main = "MSE, without Collinearity",
     xlab = "MSE"
)
```



Interestingly, in both cases, the MSE is roughly the same on average. Again, this is because collinearity effects a model's ability to *explain*, but not predict.

```r
mean(mse_bad)
```

```
## [1] 17.68559
```

```r
mean(mse_good)
```

```
## [1] 17.98918
```

Bibliography