Rmd-ex-paper-more-03Oct2021

See GitHub

03/10/2021

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

> "Statisticians, like artists, have the bad habit of falling in love with their models."
>
> — **George Box**

Let's take a step back and consider the process of finding a model for data at a higher level. We are attempting to find a model for a response variable y based on a number of predictors $x_1, x_2, x_3, \ldots, x_{p-1}$.

Essentially, we are trying to discover the functional relationship between y and the predictors. In the previous chapter we were fitting models for a car's fuel efficiency (`mpg`) as a function of its attributes (`wt`, `year`, `cyl`, `disp`, `hp`, `acc`). We also consider y to be a function of some noise. Rarely if ever do we expect there to be an *exact* functional relationship between the predictors and the response.

$$y = f(x_1, x_2, x_3, \ldots, x_{p-1}) + \epsilon$$

We can think of this as

$$\text{response} = \text{signal} + \text{noise}.$$

We *could* consider all sorts of complicated functions for f. You will likely encounter several ways of doing this in future machine learning courses. So far in this course we have focused on (multiple) linear regression. That is

$$\begin{aligned} y &= f(x_1, x_2, x_3, \ldots, x_{p-1}) + \epsilon \\ &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_{p-1} x_{p-1} + \epsilon \end{aligned}$$

In the big picture of possible models that we could fit to this data, this is a rather restrictive model. What do we mean by a restrictive model?

Family, Form, and Fit

When modeling data, there are a number of choices that need to be made.

- What **family** of models will be considered?
- What **form** of the model will be used?
- How will the model be **fit**?

Let's work backwards and discuss each of these.

Fit

Consider one of the simplest models we could fit to data, simple linear regression.

$$y = f(x_1, x_2, x_3, \ldots, x_{p-1}) + \epsilon = \beta_0 + \beta_1 x_1 + \epsilon$$

So here, despite having multiple predictors, we chose to use only one. How is this model **fit**? We will almost exclusively use the method of least squares, but recall, we had seen alternative methods of fitting this model.

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

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

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

Any of these methods (we will always use the last, least squares) will obtain estimates of the unknown parameters β_0 and β_1. Since those are the only unknowns of the specified model, we have then *fit* the model. The fitted model is then

$$\hat{y} = \hat{f}(x_1, x_2, x_3, \ldots, x_{p-1}) = \hat{\beta}_0 + \hat{\beta}_1 x_1$$

Note that, now we have dropped the term for the noise. We don't make any effort to model the noise, only the signal.

Form

What are the different **forms** a model could take? Currently, for the linear models we have considered, the only method for altering the form of the model is to control the predictors used. For example, one form of the multiple linear regression model is simple linear regression.

$$y = f(x_1, x_2, x_3, \ldots, x_{p-1}) + \epsilon = \beta_0 + \beta_1 x_1 + \epsilon$$

We could also consider a SLR model with a different predictor, thus altering the form of the model.

$$y = f(x_1, x_2, x_3, \ldots, x_{p-1}) + \epsilon = \beta_0 + \beta_2 x_2 + \epsilon$$

Often, we'll use multiple predictors in our model. Very often, we will at least try a model with all possible predictors.

$$\begin{aligned} y &= f(x_1, x_2, x_3, \ldots, x_{p-1}) + \epsilon \\ &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_{p-1} x_{p-1} + \epsilon \end{aligned}$$

We could also use some, but not all of the predictors.

$$\begin{aligned} y &= f(x_1, x_2, x_3, \ldots, x_{p-1}) + \epsilon \\ &= \beta_0 + \beta_1 x_1 + \beta_3 x_3 + \beta_5 x_5 + \epsilon \end{aligned}$$

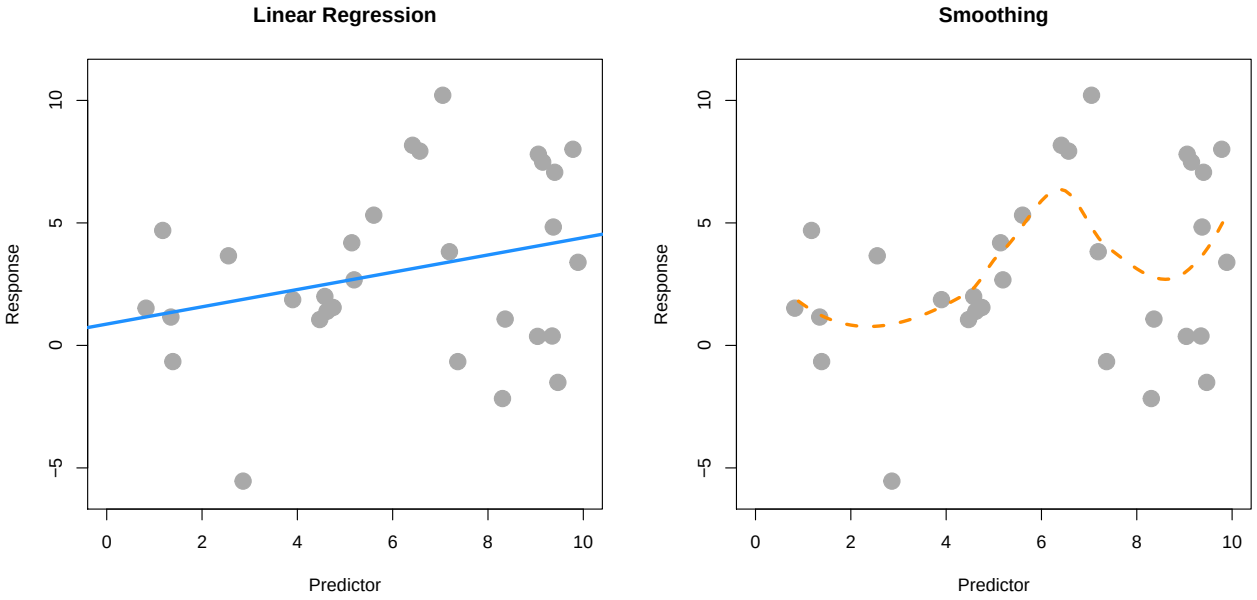
These forms are **restrictive** in two senses. First, they only allow for linear relationships between the response and the predictors. This seems like an obvious restriction of linear models, but in fact, we will soon see how to use linear models for *non-linear* relationships. (It will involve transforming variables.) Second, how one variable affects the response is the same for **any** values of the other predictors. Soon we will see how to create models where the effect of x_1 can be different for different values of x_2. We will discuss the concept of *interaction*.

Family

A **family** of models is a broader grouping of many possible *forms* of a model. For example, above we saw several forms of models from the family of linear models. We will only ever concern ourselves with linear models, which model a response as a linear combination of predictors. There are certainly other families of models.

For example, there are several families of *non-parametric* regression. Smoothing is a broad family of models. As are regression trees.

In linear regression, we specified models with parameters, β_j and fit the model by finding the best values of these parameters. This is a *parametric* approach. A non-parametric approach skips the step of specifying a model with parameters, and are often described as more of an algorithm. Non-parametric models are often used in machine learning.



Here, SLR (parametric) is used on the left, while smoothing (non-parametric) is used on the right. SLR finds the best slope and intercept. Smoothing produces the fitted y value at a particular x value by considering the y values of the data in a neighborhood of the x value considered. (Local smoothing.)

Why the focus on **linear models**? Two big reasons:

- Linear models are **the** go-to model. Linear models have been around for a long time, and are computationally easy. A linear model may not be the final model you use, but often, it should be the first model you try.
- The ideas behind linear models can be easily transferred to other modeling techniques.

Assumed Model, Fitted Model

When searching for a model, we often need to make assumptions. These assumptions are codified in the **family** and **form** of the model. For example

$$y = \beta_0 + \beta_1 x_1 + \beta_3 x_3 + \beta_5 x_5 + \epsilon$$

assumes that y is a linear combination of x_1, x_3, and x_5 as well as some noise. This assumes that the effect of x_1 on y is β_1, which is the same for all values of x_3 and x_5. That is, we are using the *family* of linear models with a particular *form*.

Suppose we then *fit* this model to some data and obtain the **fitted model**. For example, in `R` we would use

```
fit = lm(y ~ x1 + x3 + x5, data = some_data)
```

This is `R`'s way of saying the *family* is *linear* and specifying the *form* from above. An additive model with the specified predictors as well as an intercept. We then obtain

$$\hat{y} = 1.5 + 0.9x_1 + 1.1x_3 + 2.3x_5.$$

This is our best guess for the function f in

$$y = f(x_1, x_2, x_3, \ldots, x_{p-1}) + \epsilon$$

for the assumed **family** and **form**. Fitting a model only gives us the best fit for the family and form that we specify. So the natural question is; how do we choose the correct family and form? We'll focus on *form* since we are focusing on the *family* of linear models.

Explanation versus Prediction

What is the purpose of fitting a model to data? Usually it is to accomplish one of two goals. We can use a model to **explain** the relationship between the response and the predictors. Models can also be used to **predict** the response based on the predictors. Often, a good model will do both, but we'll discuss both goals separately since the process of finding models for explaining and predicting have some differences.

For our purposes, since we are only considering linear models, searching for a good model is essentially searching for a good **form** of a model.

Explanation

If the goal of a model is to explain the relationship between the response and the predictors, we are looking for a model that is **small** and **interpretable**, but still fits the data well. When discussing linear models, the **size** of a model is essentially the number of β parameters used.

Suppose we would like to find a model that explains fuel efficiency (`mpg`) based on a car's attributes (`wt`, `year`, `cyl`, `disp`, `hp`, `acc`). Perhaps we are a car manufacturer trying to engineer a fuel efficient vehicle. If this is the case, we are interested in both which predictor variables are useful for explaining the car's fuel efficiency, as well as how those variables effect fuel efficiency. By understanding this relationship, we can use this knowledge to our advantage when designing a car.

To explain a relationship, we are interested in keeping models as small as possible, since smaller models are easy to interpret. The fewer predictors the less considerations we need to make in our design process.

Note that *linear* models of any size are rather interpretable to begin with. Later in your data analysis careers, you will see more complicated models that may fit data better, but are much harder, if not impossible to interpret. These models aren't nearly as useful for explaining a relationship. This is another reason to always attempt a linear model. If it fits as well as more complicated methods, it will be the easiest to understand.

To find small and interpretable models, we will eventually use selection procedures, which search among many possible forms of a model. For now we will do this in a more ad-hoc manner using **inference** techniques we

have already encountered. To use inference as we have seen it, we need an additional assumption in addition to the family and form of the model.

$$y = \beta_0 + \beta_1 x_1 + \beta_3 x_3 + \beta_5 x_5 + \epsilon$$

Our additional assumption is about the error term.

$$\epsilon \sim N(0, \sigma^2)$$

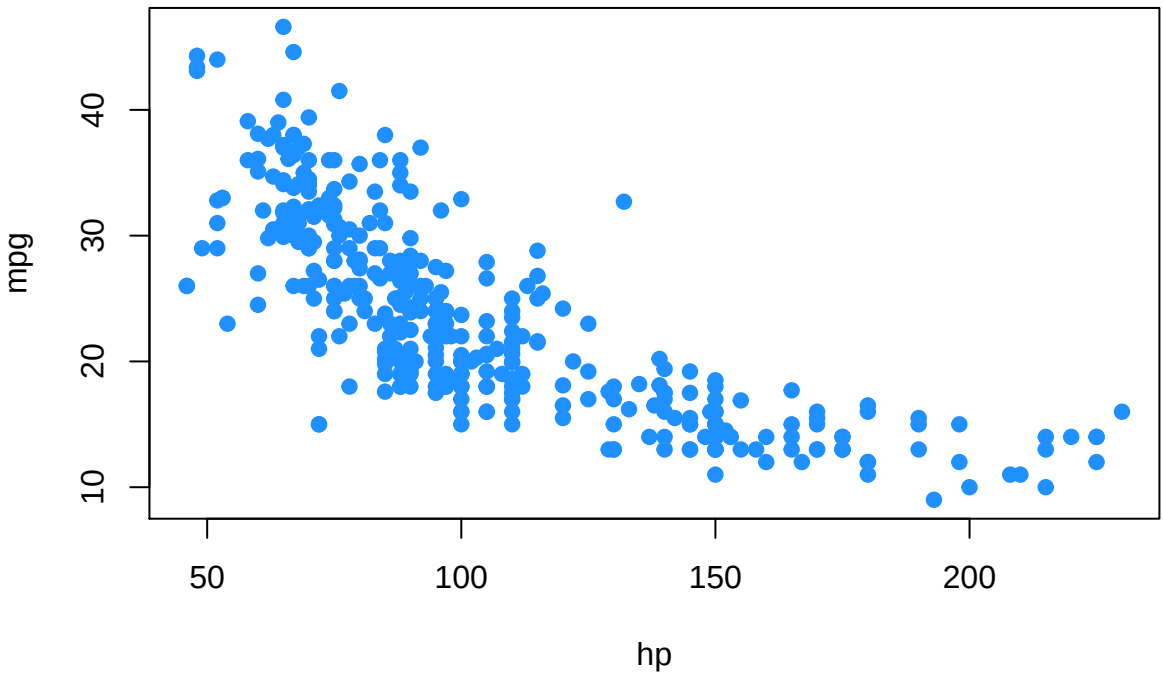
This assumption, that the errors are normally distributed with some common variance is the key to all of the inference we have done so far. We will discuss this in great detail later.

So with our inference tools (ANOVA and t-test) we have two potential strategies. Start with a very small model (no predictors) and attempt to add predictors. Or, start with a big model (all predictors) and attempt to remove predictors.

Correlation and Causation A word of caution when using a model to *explain* a relationship. There are two terms often used to describe a relationship between two variables: *causation* and *correlation*. Correlation is often also referred to as association.

Just because two variables are correlated does not necessarily mean that one causes the other. For example, consider modeling `mpg` as only a function of `hp`.

```
plot(mpg ~ hp, data = autompg, col = "dodgerblue", pch = 20, cex = 1.5)
```



Does an increase in horsepower cause a drop in fuel efficiency? Or, perhaps the causality is reversed and an increase in fuel efficiency cause a decrease in horsepower. Or, perhaps there is a third variable that explains both!

The issue here is that we have **observational** data. With observational data, we can only detect *associations*. To speak with confidence about *causality*, we would need to run **experiments**. Often, this decision is made for us, before we ever see data, so we can only modify our interpretation.

This is a concept that you should encounter often in your statistics education. For some further reading, and some related fallacies, see: Wikipedia: Correlation does not imply causation.

We'll discuss this further when we discuss experimental design and traditional ANOVA techniques. (All of which has recently been re-branded as A/B testing.)

Prediction

If the goal of a model is to predict the response, then the **only** consideration is how well the model fits the data. For this, we will need a metric. In regression problems, this is most often RMSE.

$$\text{RMSE(model, data)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where

- y_i are the actual values of the response for the given data
- $\hat{y}_i$ are the predicted values using the fitted model and the predictors from the data

Correlation and causation are *not* an issue here. If a predictor is correlated with the response, it is useful for prediction. For example, in elementary school aged children their shoe size certainly doesn't *cause* them to read at a higher level, however we could very easily use shoe size to make a prediction about a child's reading ability. The larger their shoe size, the better they read. There's a lurking variable here though, their age! (Don't send your kids to school with size 14 shoes, it won't make them read better!)

Also, since we are not performing inference, the extra assumption about the errors is not needed. The only thing we care about is how close the fitted model is to the data. Least squares is least squares. For a specified model, it will find the values of the parameters which will minimize the squared error loss. Your results might be largely uninterpretable and useless for inference, but for prediction none of that matters.

Suppose instead of the manufacturer who would like to build a car, we are a consumer who wishes to purchase a new car. However this particular car is so new, it has not been rigorously tested, so we are unsure of what fuel efficiency to expect. (And, as skeptics, we don't trust what the manufacturer is telling us.) In this case, we would like to use the model to help *predict* the fuel efficiency of this car based on its attributes, which are the predictors of the model. The smaller the errors the model makes, the more confident we are in its prediction.

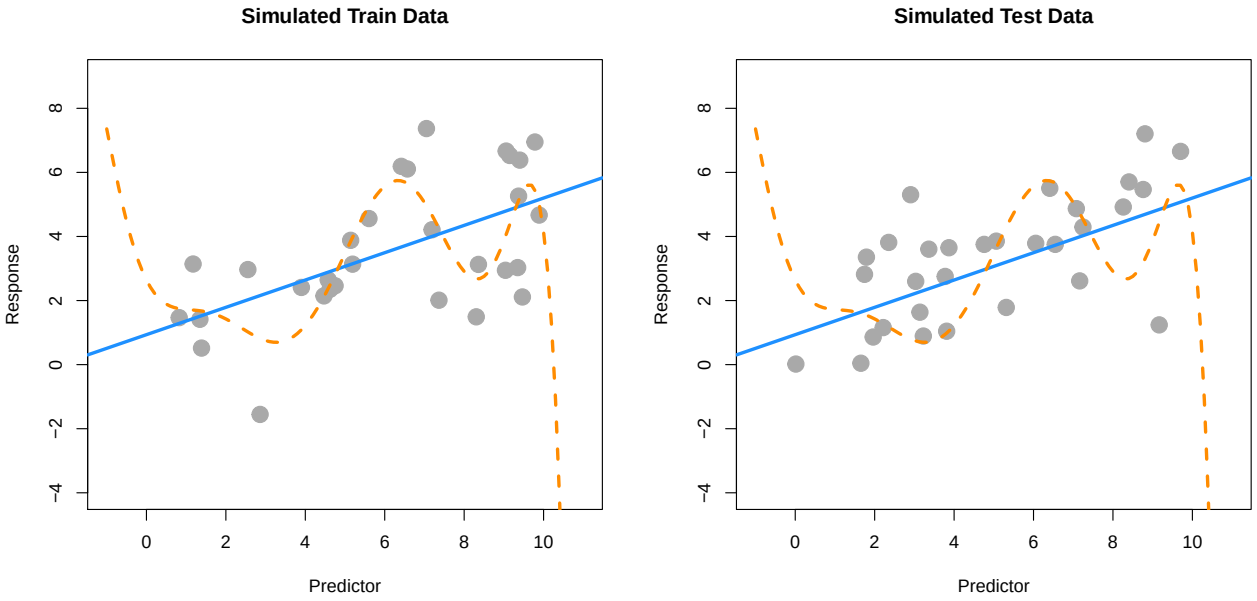
Test-Train Split The trouble with using RMSE to identify how well a model fits data, is that RMSE is **always** (equal or) lower for a larger model. This would suggest that we should always use the largest model possible when looking for a model that predicts well. The problem with this is the potential to **overfit** to the data. So, we want a model that fits well, but does not overfit. To understand overfitting, we need to think about applying a model to seen and unseen data.

Suppose we fit a model using all data available and we evaluate RMSE on this fitted model and all of the seen data. We will call this data the **training** data, and this RMSE the **train** RMSE.

Now, suppose we magically encounter some additional data. To truly asses how well the model predicts, we should evaluate how well our models predicts the response of this data. We will call this data the **test** data and this RMSE the **test** RMSE.

- Train RMSE: model fit on seen data, evaluated on **seen** data
- Test RMSE: model fit on seen data, evaluated on **unseen** data

Below, we simulate some data and fit two models. We will call the solid blue line the "simple" model. The dashed orange line will be called the "complex" model, which was fit with methods we do not yet know.



The left panel shows the data that was used to fit the two models. Clearly the "complex" model fits the data much better. The right panel shows additional data that was simulated in the same manner as the original data. Here we see that the "simple" model fits much better. The dashed orange line almost seems random.

Model	**Train** RMSE	**Test** RMSE
Simple	1.71	1.45
Complex	1.41	2.07

The more "complex", wiggly, model fits the training data much better as it has a much lower train RMSE. However, we see that the "simple" model fits the test data much better, with a much lower test RMSE. This means that the complex model has *overfit* the data, and we prefer the simple model. When choosing a model for prediction, we prefer a model that predicts unseen data.

In practice, you can't simply generate more data to evaluate your models. Instead we split existing data into data used to fit the model (train) and data used to evaluate the model (test). Never fit a model with test data.

Summary

Models can be used to **explain** relationships and **predict** observations.

When using model to,

- **explain**; we prefer *small* and *interpretable* models.
- **predict**; we prefer models that make the smallest errors possible, without *overfitting*.

Linear models can accomplish both these goals. Later, we will see that often a linear model that accomplishes one of these goals, usually accomplishes the other.

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.

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
```

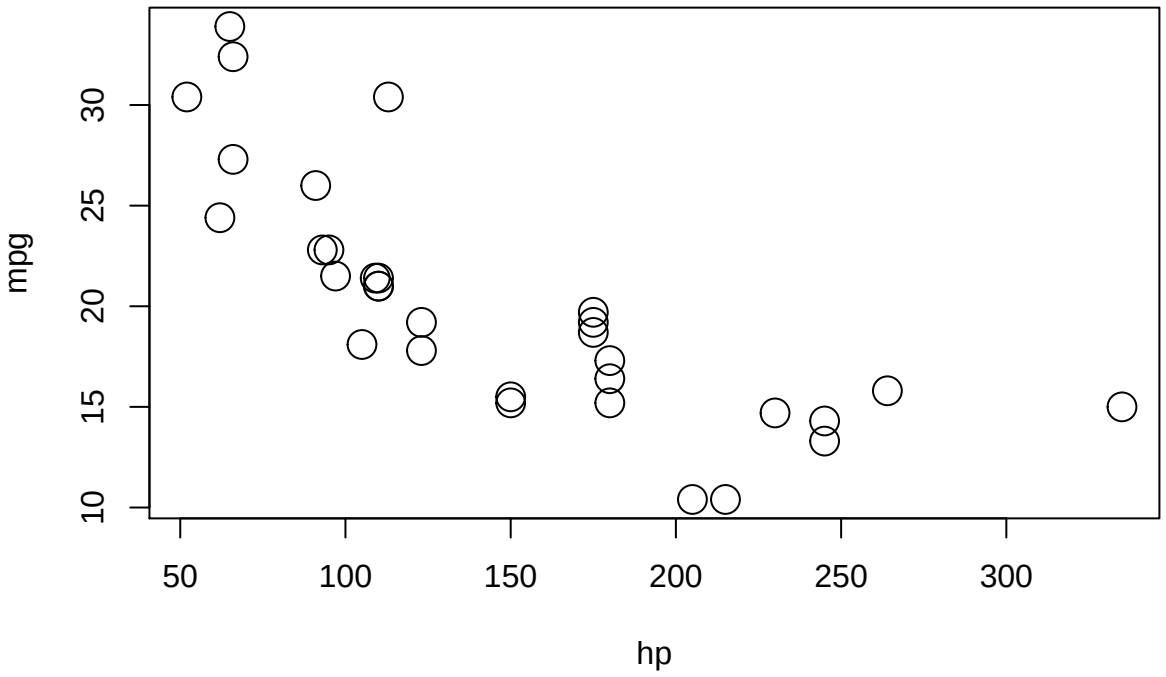
```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag      21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 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.

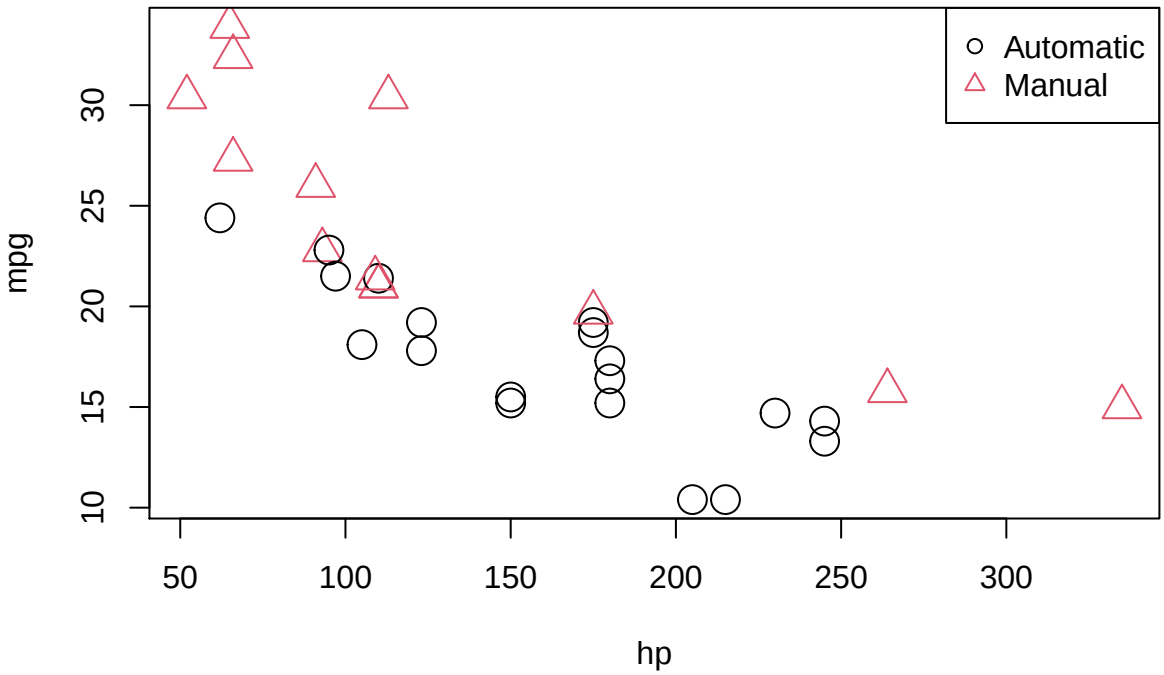
As we often do, we will start by plotting the data. We are interested in `mpg` as the response variable, and `hp` as a predictor.

```
plot(mpg ~ hp, data = mtcars, cex = 2)
```



Since we are also interested in the transmission type, we could also label the points accordingly.

```
plot(mpg ~ hp, data = mtcars, col = am + 1, pch = am + 1, cex = 2)
legend("topright", c("Automatic", "Manual"), col = c(1, 2), pch = c(1, 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

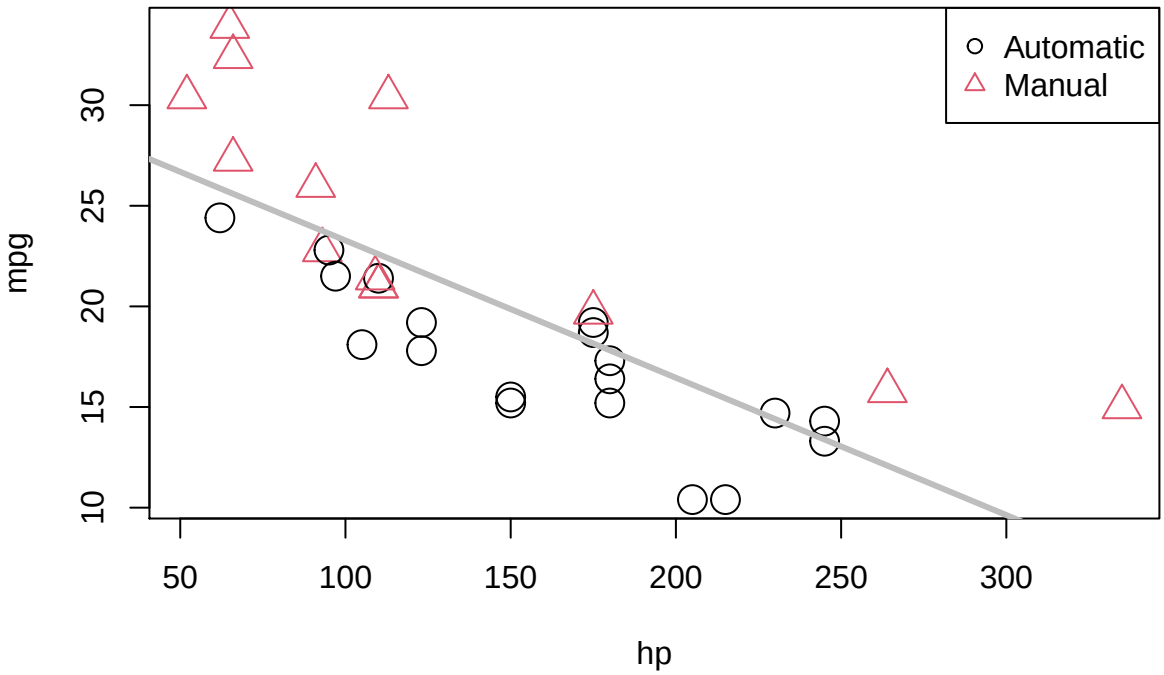
$$Y = \beta_0 + \beta_1 x_1 + \epsilon,$$

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 = 3, col = "grey")
legend("topright", c("Automatic", "Manual"), col = c(1, 2), pch = c(1, 2))
```



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:

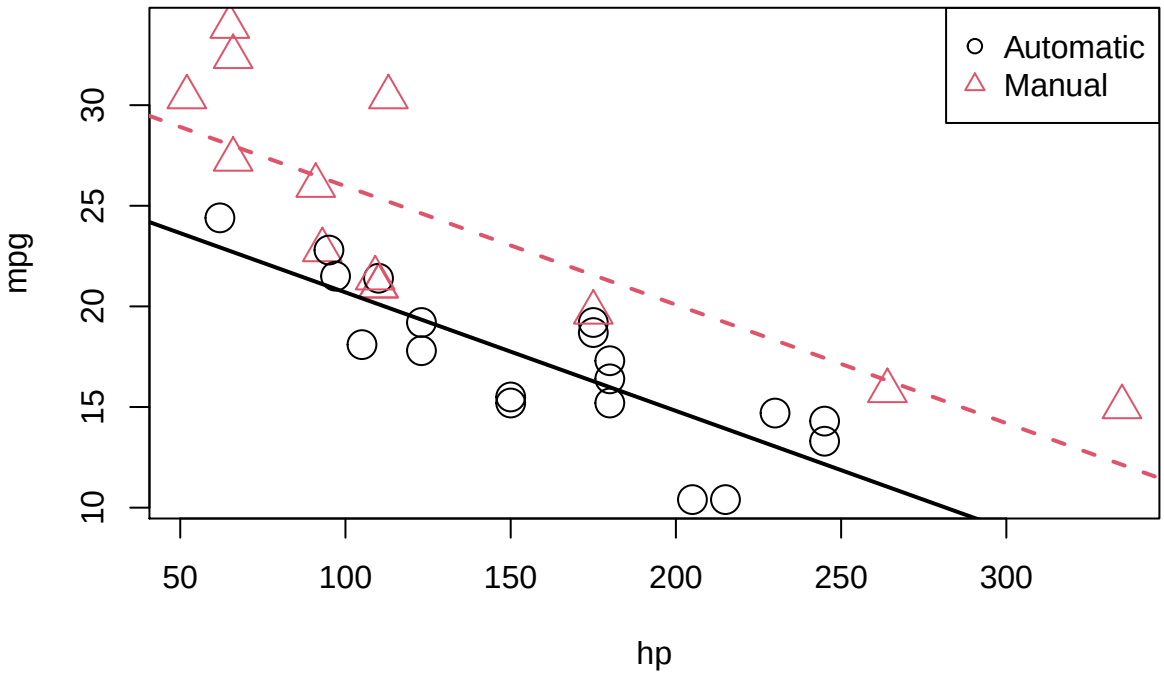
- $\hat{\beta}_0$ = `coef(mpg_hp_add)[1]` = 26.5849137
- $\hat{\beta}_1$ = `coef(mpg_hp_add)[2]` = -0.0588878
- $\hat{\beta}_2$ = `coef(mpg_hp_add)[3]` = 5.2770853

We can then combine these to calculate the estimated slope and intercepts.

```
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.

```
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
legend("topright", c("Automatic", "Manual"), col = c(1, 2), pch = c(1, 2))
```



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)$coefficients["am",]
```

```
##     Estimate   Std. Error      t value     Pr(>|t|)
## 5.277085e+00 1.079541e+00 4.888270e+00 3.460318e-05
```

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 3.46e-05 ***
## ---
## 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.

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.

```
# 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$hp)
# create a dummy variable for foreign vs domestic cars. domestic = 1.
autompg$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,

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon,$$

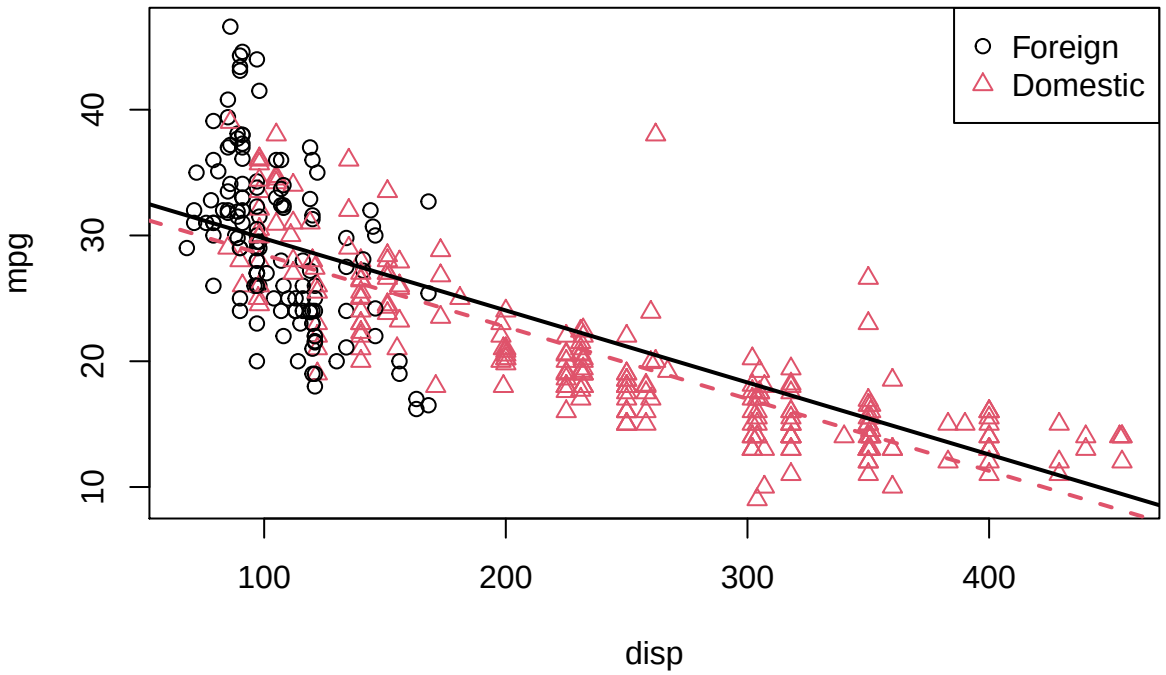
where

- 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.

$$x_2 = \begin{cases} 1 & \text{Domestic} \\ 0 & \text{Foreign} \end{cases}$$

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
legend("topright", c("Foreign", "Domestic"), pch = c(1, 2), col = c(1, 2))
```



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.

```
anova(mpg_disp_add, mpg_disp_int)
```

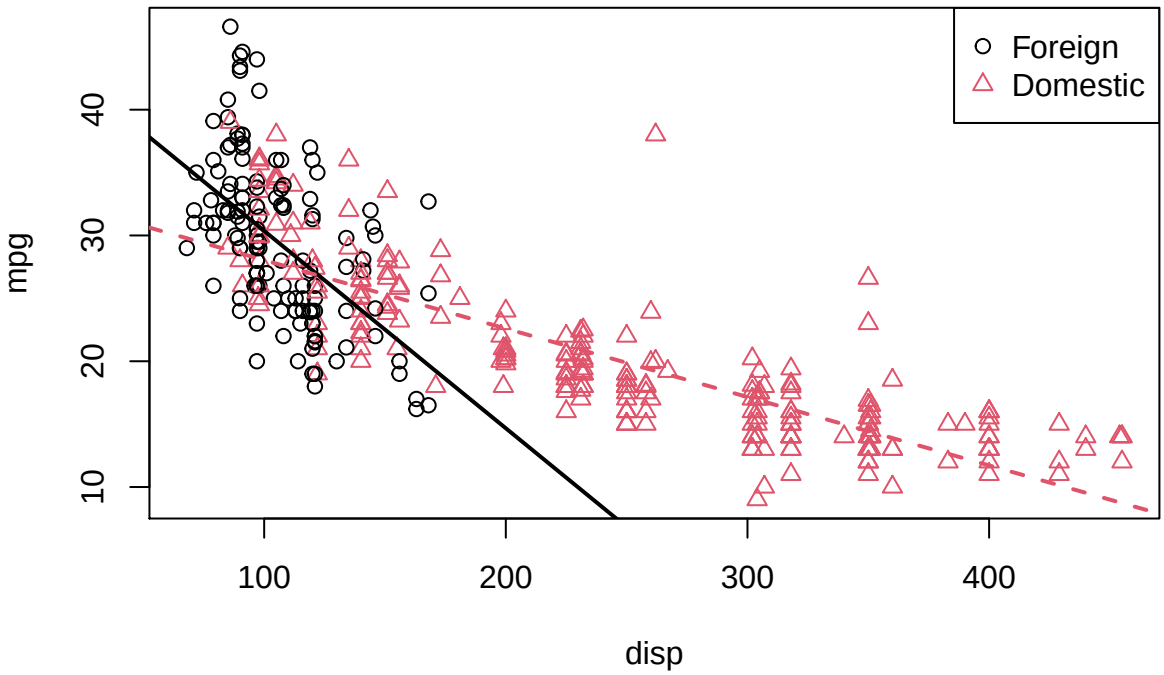
```
## Analysis of Variance Table
##
## Model 1: mpg ~ disp + domestic
## Model 2: mpg ~ disp + domestic + disp:domestic
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    380 7714.0
## 2    379 7032.6  1    681.36 36.719 3.294e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 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.

```
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.

```
plot(mpg ~ disp, data = autompg, col = domestic + 1, pch = domestic + 1)
abline(int_for, slope_for, col = 1, lty = 1, lwd = 2) # line for foreign cars
abline(int_dom, slope_dom, col = 2, lty = 2, lwd = 2) # line for domestic cars
legend("topright", c("Foreign", "Domestic"), pch = c(1, 2), col = c(1, 2))
```



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)  5.241e+01  1.523e+00   34.42   <2e-16 ***
## disp        -1.002e-01  6.638e-03  -15.09   <2e-16 ***
## hp          -2.198e-01  1.987e-02  -11.06   <2e-16 ***
## disp:hp      5.658e-04  5.165e-05   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.

```
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.

```
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`**.
- $\hat{\beta}_3 = 5.658269 \times 10^{-4}$ 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

$$-0.1001738 + 5.658269 \times 10^{-4} x_2.$$

This says that, for an increase of one `disp` we see an estimated change in average `mpg` of $-0.1001738 + 5.658269 \times 10^{-4} x_2$. 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

$$-0.1001738 + 5.658269 \times 10^{-4} \cdot 50 = -0.0718824$$

And for a car with 350 `hp`, the estimated change in average `mpg` for an increase of one `disp` is

$$-0.1001738 + 5.658269 \times 10^{-4} \cdot 350 = 0.0978657$$

Notice the sign changed!

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 created 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.

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.

```
is.factor(autompg$cyl)
```

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

```
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, otherwise, 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.

```
(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: $Y = (\beta_0 + \beta_3) + \beta_1 x + \epsilon$

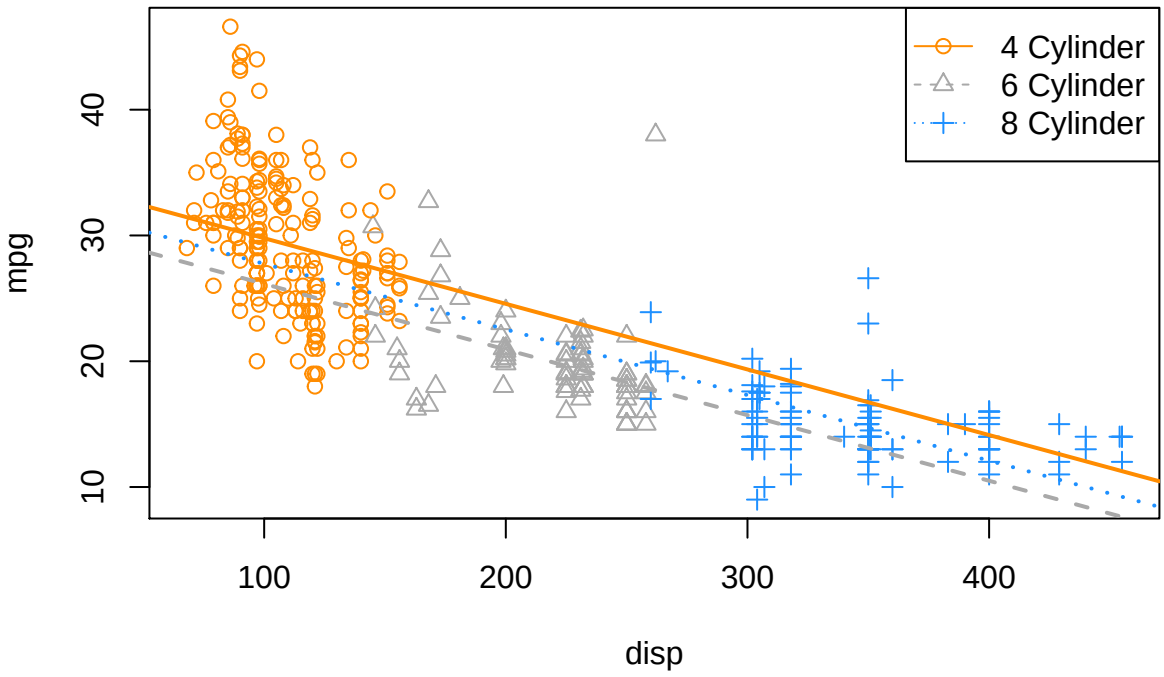
Notice that they all have the same slope. However, using the two dummy variables, we achieve the three intercepts.

- 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_colors = c("Darkorange", "Darkgrey", "Dodgerblue")
plot(mpg ~ disp, data = autompg, col = plot_colors[cyl], pch = as.numeric(cyl))
abline(int_4cyl, slope_all_cyl, col = plot_colors[1], lty = 1, lwd = 2)
abline(int_6cyl, slope_all_cyl, col = plot_colors[2], lty = 2, lwd = 2)
abline(int_8cyl, slope_all_cyl, col = plot_colors[3], lty = 3, lwd = 2)
legend("topright", c("4 Cylinder", "6 Cylinder", "8 Cylinder"),
       col = plot_colors, lty = c(1, 2, 3), pch = c(1, 2, 3))
```



On this plot, we have

- 4 Cylinder: orange dots, solid orange line.
- 6 Cylinder: grey dots, dashed grey line.
- 8 Cylinder: blue dots, dotted blue 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 blue line is always above the dashed grey 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:

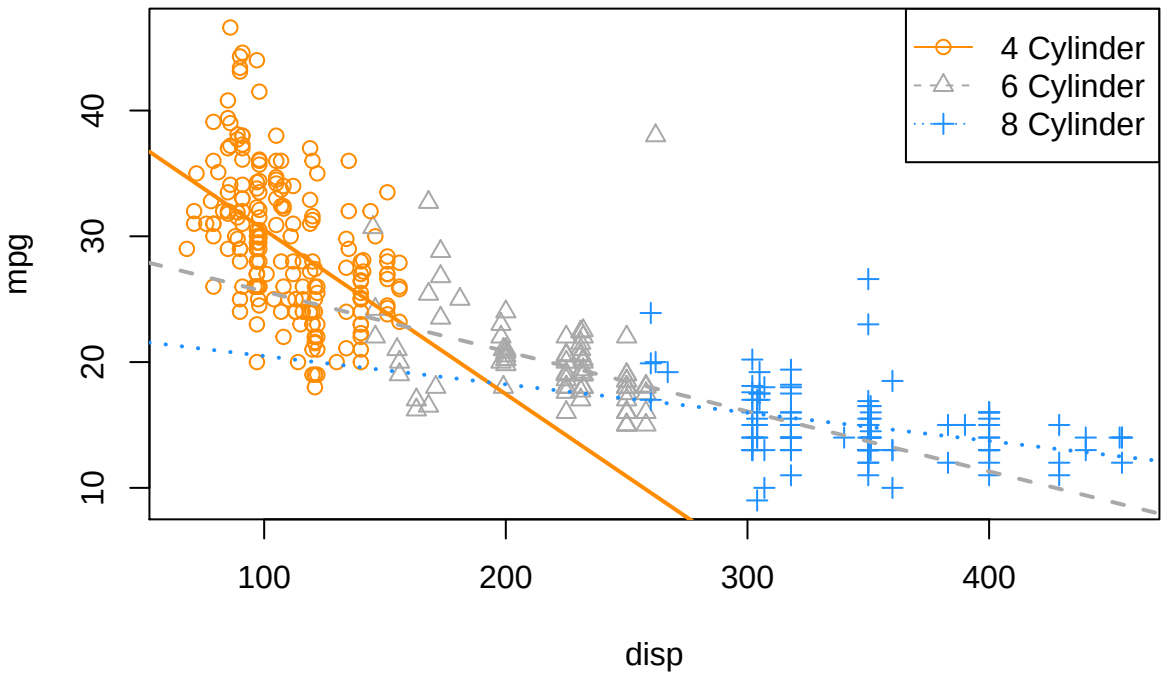
- $(\beta_0 + \beta_2)$ is the average `mpg` of a 6 cylinder car with 0 `disp`
- $(\hat{\beta}_1 + \hat{\gamma}_3) = -0.1306935 + 0.1081714 = -0.0225221$ is the estimated change in average `mpg` for an increase of one `disp`, for an 8 cylinder car.

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_colors = c("Darkorange", "Darkgrey", "Dodgerblue")
plot(mpg ~ disp, data = autompg, col = plot_colors[cyl], pch = as.numeric(cyl))
abline(int_4cyl, slope_4cyl, col = plot_colors[1], lty = 1, lwd = 2)
abline(int_6cyl, slope_6cyl, col = plot_colors[2], lty = 2, lwd = 2)
abline(int_8cyl, slope_8cyl, col = plot_colors[3], lty = 3, lwd = 2)
legend("topright", c("4 Cylinder", "6 Cylinder", "8 Cylinder"),
       col = plot_colors, lty = c(1, 2, 3), pch = c(1, 2, 3))
```



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
```

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.

```
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.

```
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.

```
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
```

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)       6.065e+01  6.600e+00   9.189  < 2e-16 ***
## disp             -1.416e-01  6.344e-02  -2.232   0.0262 *
## hp               -3.545e-01  8.123e-02  -4.364 1.65e-05 ***
## domestic         -1.257e+01  7.064e+00  -1.780   0.0759 .
## disp:hp           1.369e-03  6.727e-04   2.035   0.0426 *
## disp:domestic     4.933e-02  6.400e-02   0.771   0.4414
```

```
## hp:domestic       1.852e-01  8.709e-02   2.126   0.0342 *
## disp:hp:domestic -9.163e-04  6.768e-04  -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.

```
two_way_int_mod = lm(mpg ~ disp * hp + disp * domestic + hp * domestic, data = autompg)
#two_way_int_mod = lm(mpg ~ (disp + hp + domestic) ^ 2, 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.

```
mean(resid(big_model) ^ 2)
```

```
## [1] 14.74053
```

```
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.

```
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

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1 x_2 + \beta_5 x_1 x_3 + \beta_6 x_2 x_3 + \epsilon.$$

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.

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 then noted that these estimates had mean

$$\mathrm{E}[\hat{\beta}] = \beta,$$

and variance

$$\text{Var}[\hat{\beta}] = \sigma^2 \left(X^\top X\right)^{-1}.$$

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.**

Checking Assumptions

We'll now look at a number of tools for checking the assumptions of a linear model. To test these tools, we'll use data simulated from three models:

$$\text{Model 1:} \quad Y = 3 + 5x + \epsilon, \quad \epsilon \sim N(0, 1)$$

$$\text{Model 2:} \quad Y = 3 + 5x + \epsilon, \quad \epsilon \sim N(0, x^2)$$

$$\text{Model 3:} \quad Y = 3 + 5x^2 + \epsilon, \quad \epsilon \sim N(0, 25)$$

```
sim_1 = function(sample_size = 500) {
  x = runif(n = sample_size) * 5
  y = 3 + 5 * x + rnorm(n = sample_size, mean = 0, sd = 1)
  data.frame(x, y)
}

sim_2 = function(sample_size = 500) {
  x = runif(n = sample_size) * 5
  y = 3 + 5 * x + rnorm(n = sample_size, mean = 0, sd = x)
  data.frame(x, y)
}

sim_3 = function(sample_size = 500) {
  x = runif(n = sample_size) * 5
  y = 3 + 5 * x ^ 2 + rnorm(n = sample_size, mean = 0, sd = 5)
  data.frame(x, y)
}
```

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.

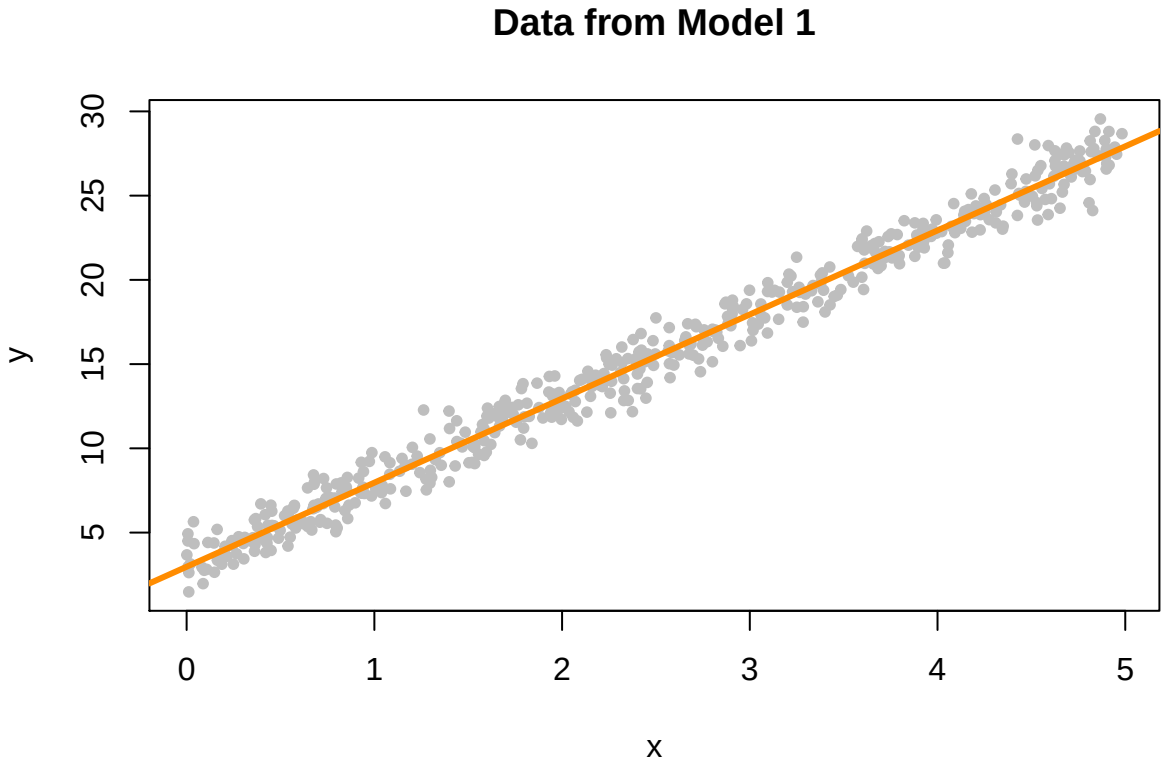
Data generated from Model 1 above should not show any signs of violating assumptions, so we'll use this to see what a good fitted versus residuals plot should look like. First, we'll simulate observations from this model.

```
set.seed(42)
sim_data_1 = sim_1()
head(sim_data_1)
```

```
##          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
```

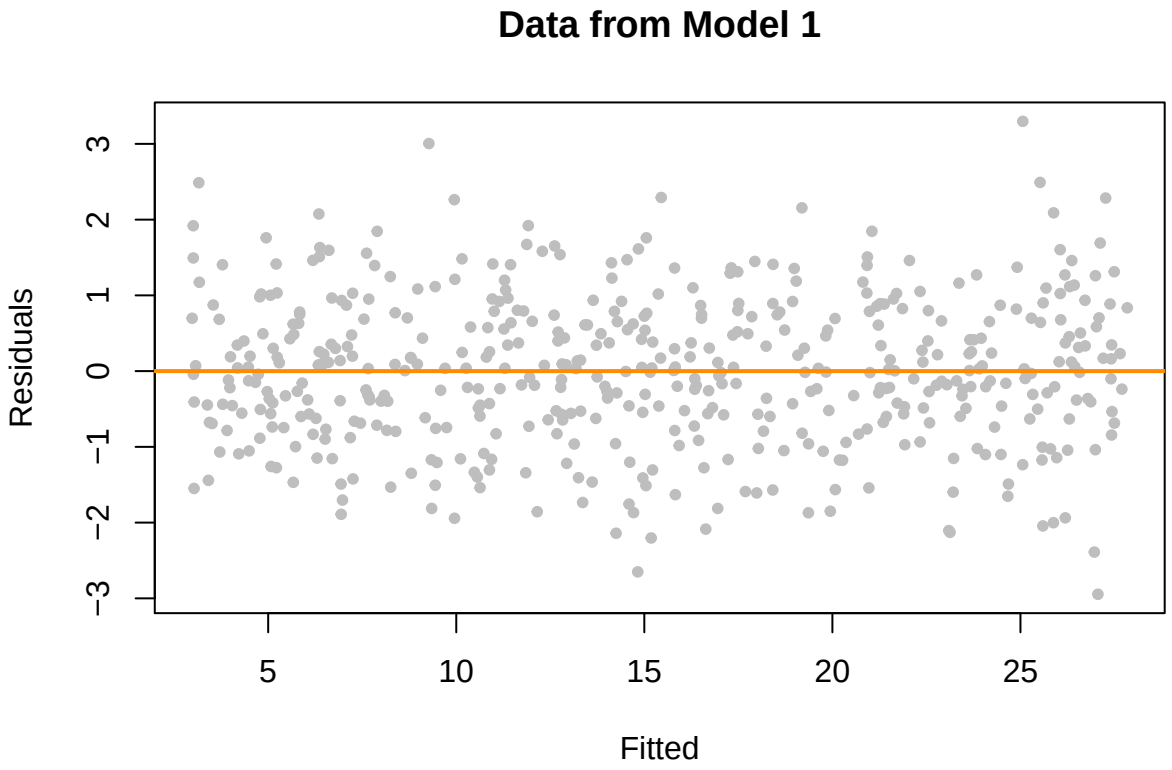
We then fit the model and add the fitted line to a scatterplot.

```
plot(y ~ x, data = sim_data_1, col = "grey", pch = 20,
     main = "Data from Model 1")
fit_1 = lm(y ~ x, data = sim_data_1)
abline(fit_1, 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(fit_1), resid(fit_1), col = "grey", pch = 20,
     xlab = "Fitted", ylab = "Residuals", main = "Data from Model 1")
abline(h = 0, col = "darkorange", lwd = 2)
```



We should look for two things in this plot.

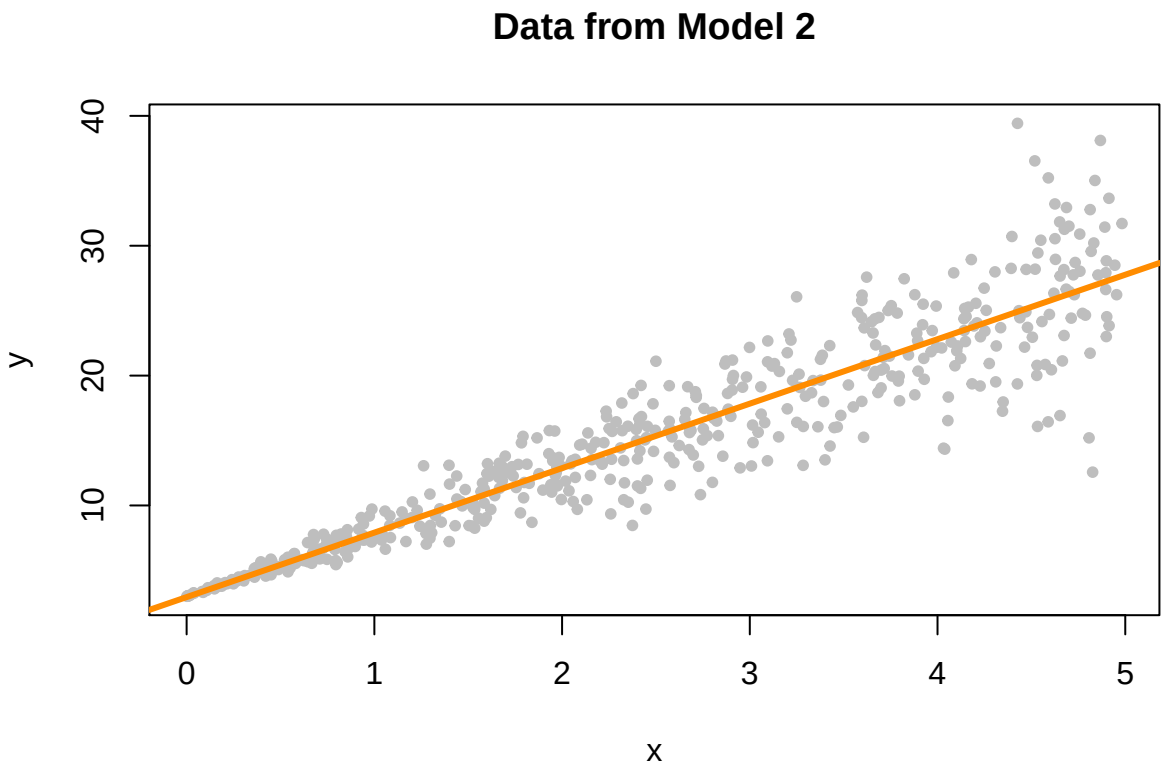
- At any fitted value, the mean of the residuals should be roughly 0. If this is the case, the *linearity* assumption is valid. For this reason, we generally add a horizontal line at $y = 0$ to emphasize this point.
- At every fitted value, the spread of the residuals should be roughly the same. If this 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.

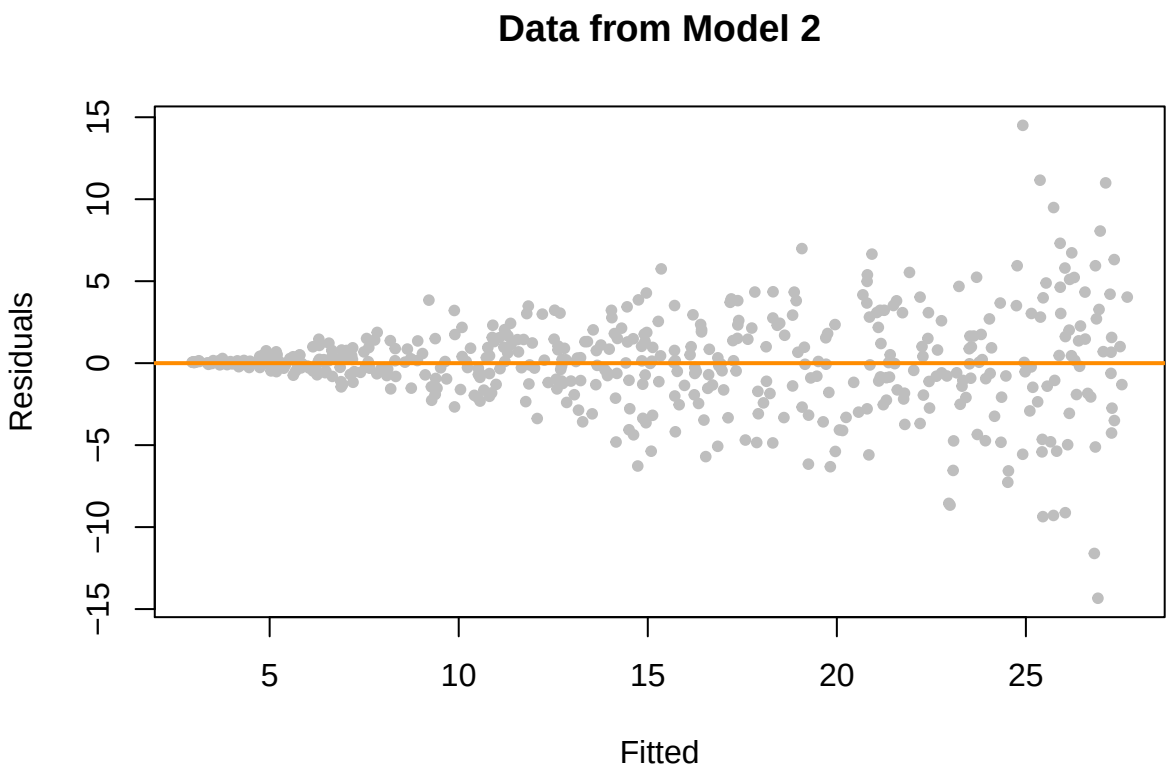
Model 2 is an example of non-constant variance. In this case, the variance is larger for larger values of the predictor variable x.

```
set.seed(42)
sim_data_2 = sim_2()
fit_2 = lm(y ~ x, data = sim_data_2)
plot(y ~ x, data = sim_data_2, col = "grey", pch = 20,
     main = "Data from Model 2")
abline(fit_2, col = "darkorange", lwd = 3)
```



This actually is rather easy to see here by adding 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.

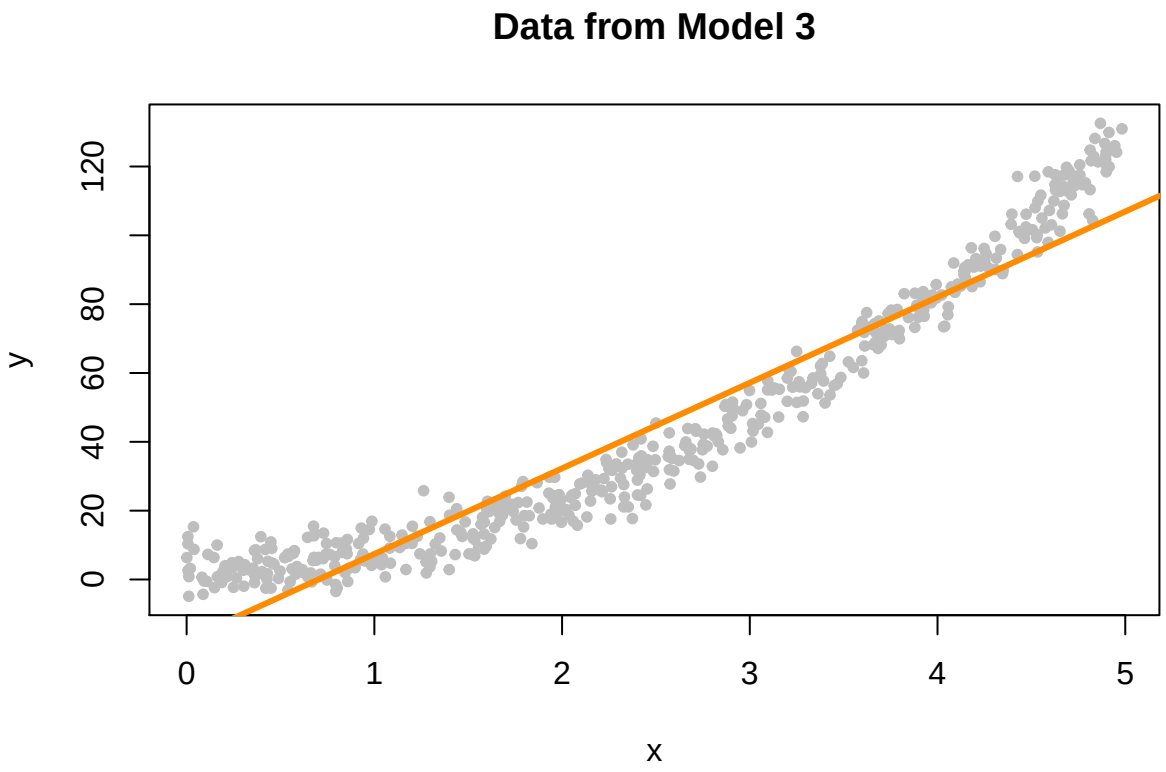
```
plot(fitted(fit_2), resid(fit_2), col = "grey", pch = 20,
     xlab = "Fitted", ylab = "Residuals", main = "Data from Model 2")
abline(h = 0, col = "darkorange", lwd = 2)
```



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.

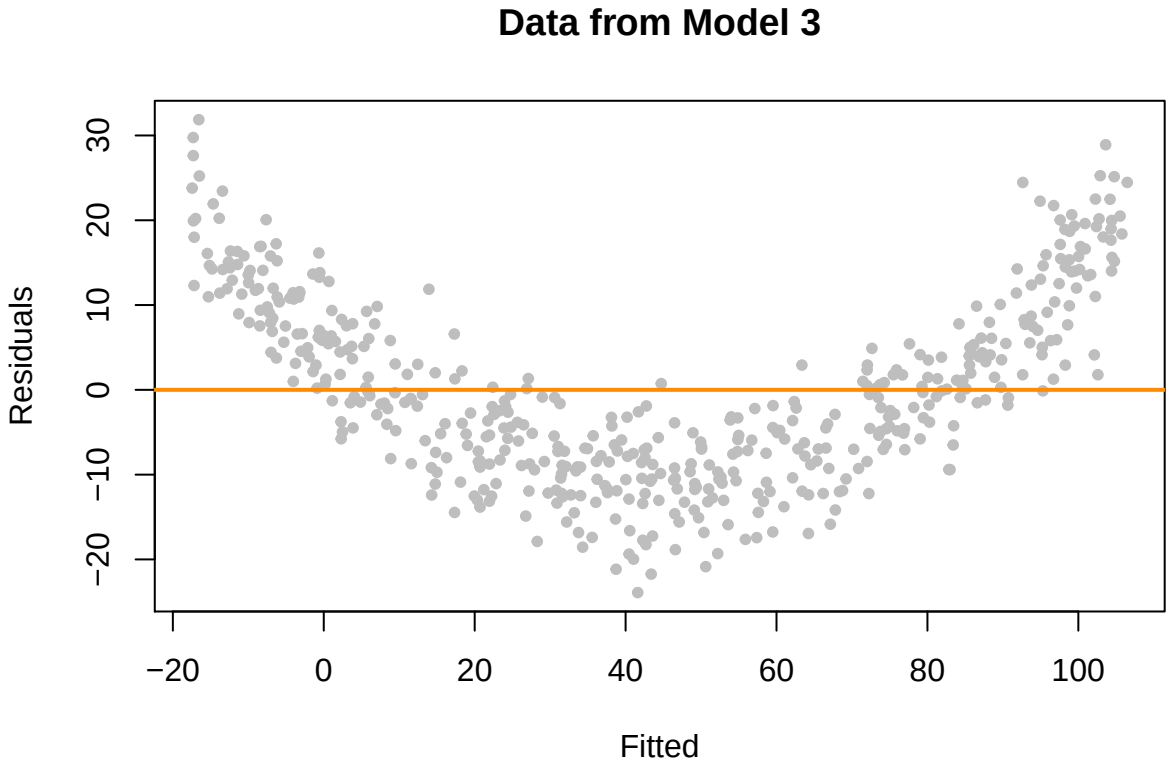
Now we will demonstrate a model which does not meet the linearity assumption. Model 3 is an example of a model where Y is not a linear combination of the predictors. In this case the predictor is x, but the model uses x^2. (We'll see later that this is something that a "linear" model can deal with. The fix is simple, just make x^2 a predictor!)

```
set.seed(42)
sim_data_3 = sim_3()
fit_3 = lm(y ~ x, data = sim_data_3)
plot(y ~ x, data = sim_data_3, col = "grey", pch = 20,
     main = "Data from Model 3")
abline(fit_3, 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(fit_3), resid(fit_3), col = "grey", pch = 20,
     xlab = "Fitted", ylab = "Residuals", main = "Data from Model 3")
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. The form of our model is simply wrong. We're trying to fit a line to a curve!

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.

```
#install.packages("lmtest")
library(lmtest)
```

Let's try it on the three models we fit above. Recall,

- `fit_1` had no violation of assumptions,
- `fit_2` violated the constant variance assumption, but not linearity,
- `fit_3` violated linearity, but not constant variance.

```
bptest(fit_1)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  fit_1
## BP = 1.0234, df = 1, p-value = 0.3117
```

For `fit_1` we see a large p-value, so we do not reject the null of homoscedasticity, which is what we would expect.

```
bptest(fit_2)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  fit_2
## BP = 76.693, df = 1, p-value < 2.2e-16
```

For `fit_2` 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.

```
bptest(fit_3)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  fit_3
```

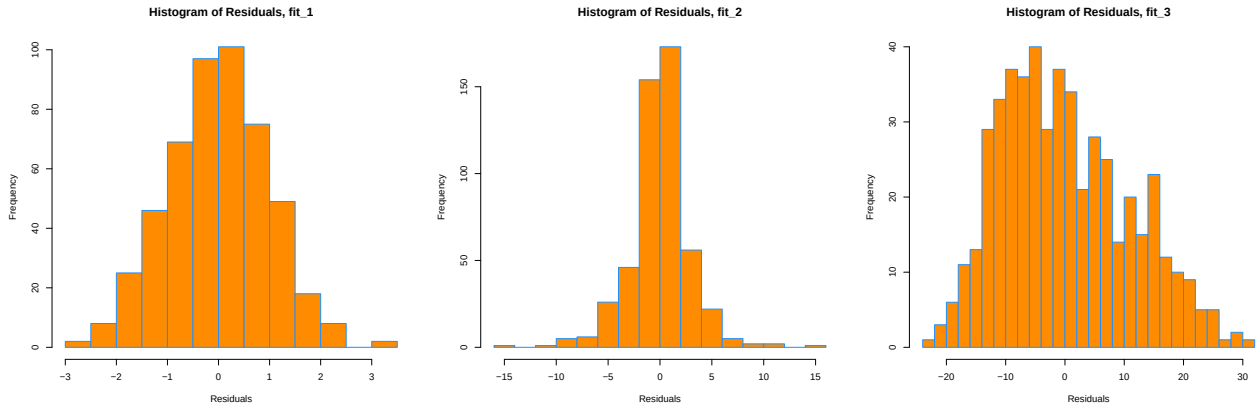
```
## BP = 0.33466, df = 1, p-value = 0.5629
```

Lastly, for `fit_3` 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.

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(fit_1),
     xlab   = "Residuals",
     main   = "Histogram of Residuals, fit_1",
     col    = "darkorange",
     border = "dodgerblue",
     breaks = 20)
hist(resid(fit_2),
     xlab   = "Residuals",
     main   = "Histogram of Residuals, fit_2",
     col    = "darkorange",
     border = "dodgerblue",
     breaks = 20)
hist(resid(fit_3),
     xlab   = "Residuals",
     main   = "Histogram of Residuals, fit_3",
     col    = "darkorange",
     border = "dodgerblue",
     breaks = 20)
```



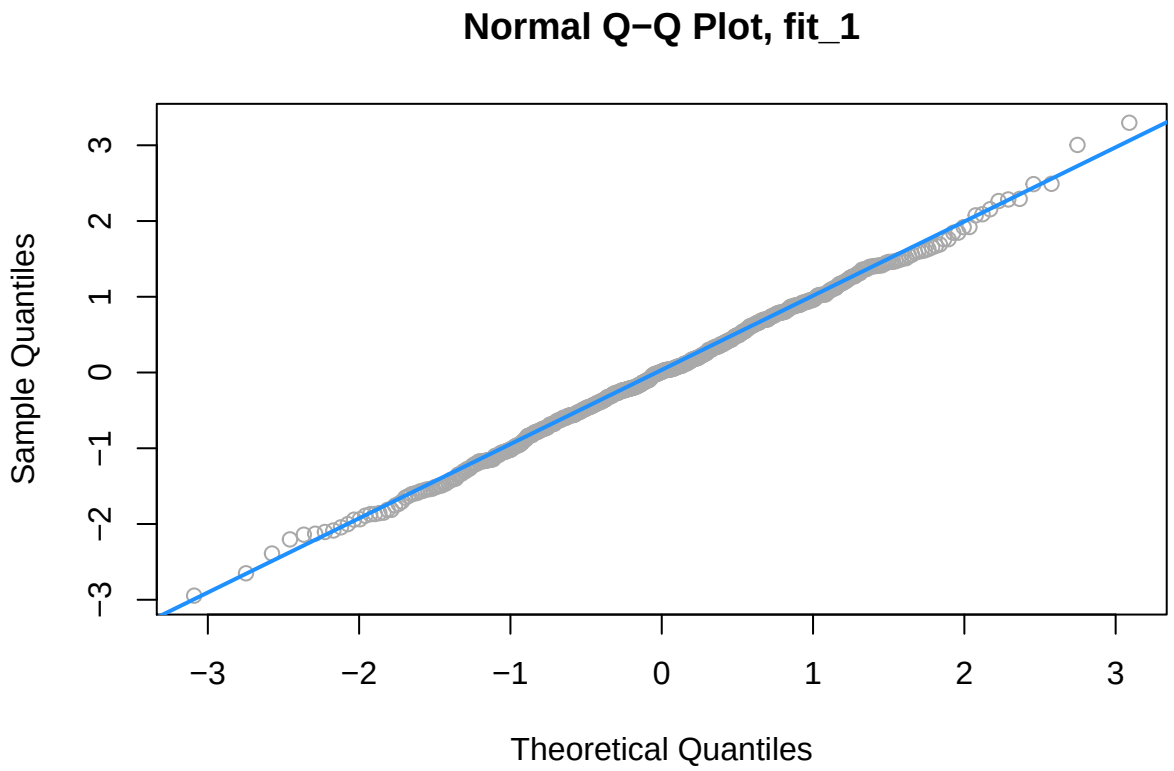
Above are histograms for each of the three regression we have been considering. Notice that the first, for `fit_1` appears very normal. The third, for `fit_3`, appears to be very non-normal. However `fit_2` is not as clear. It does have a rough bell shape, however, it also has a very sharp peak. For 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.

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 `fit_1` to check if the errors could truly be normally distributed.

```r
qqnorm(resid(fit_1), main = "Normal Q-Q Plot, fit_1", col = "darkgrey")
qqline(resid(fit_1), col = "dodgerblue", 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.

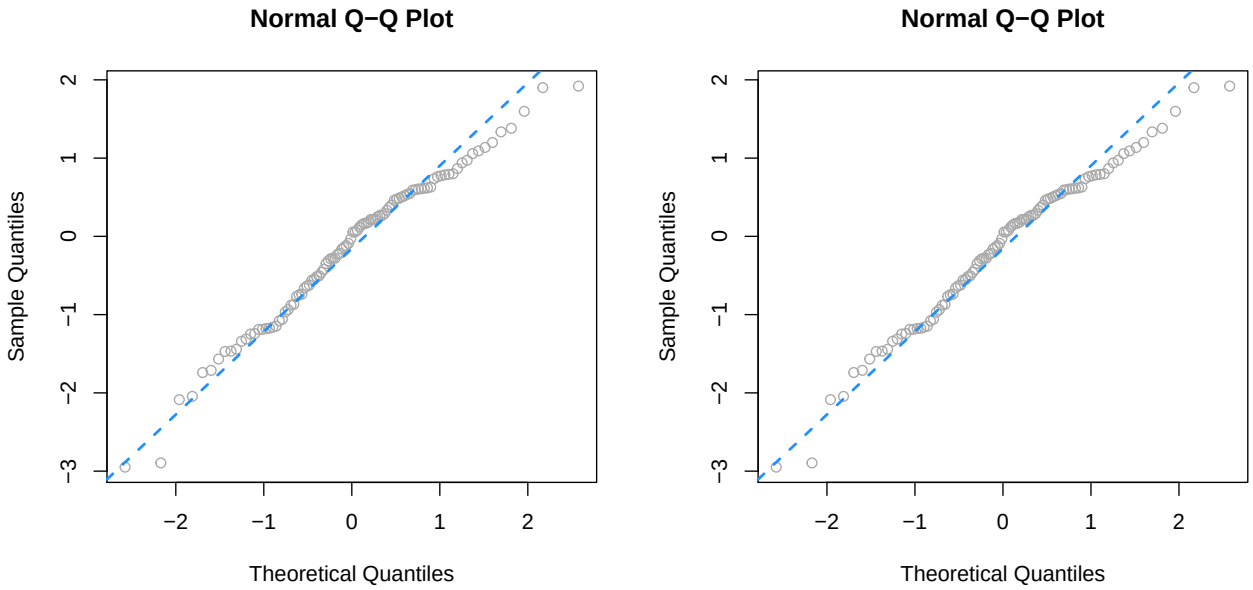
Also, to get a better idea of how Q-Q plots work, here is a quick function which creates a Q-Q plot:

```r
qq_plot = function(e) {

  n = length(e)
  normal_quantiles = qnorm(((1:n - 0.5) / n))
  # normal_quantiles = qnorm(((1:n) / (n + 1)))

  # plot theoretical verus observed quantiles
  plot(normal_quantiles, sort(e),
       xlab = c("Theoretical Quantiles"),
       ylab = c("Sample Quantiles"),
       col = "darkgrey")
  title("Normal Q-Q Plot")

  # calculate line through the first and third quartiles
  slope     = (quantile(e, 0.75) - quantile(e, 0.25)) / (qnorm(0.75) - qnorm(0.25))
  intercept = quantile(e, 0.25) - slope * qnorm(0.25)
```

```
  # add to existing plot
  abline(intercept, slope, lty = 2, lwd = 2, col = "dodgerblue")
}
```

We can then verify that it is essentially equivalent to using `qqnorm()` and `qqline()` in `R`.

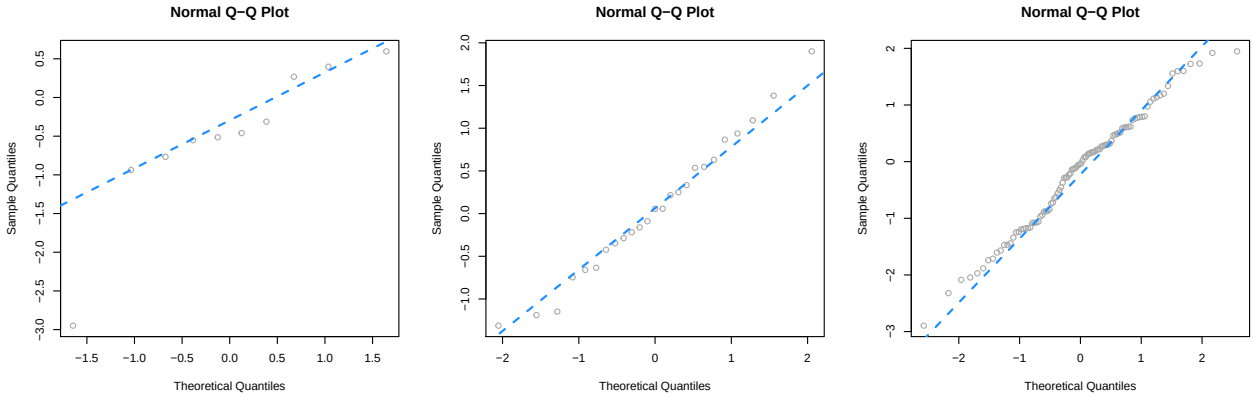
```
set.seed(420)
x = rnorm(100, mean = 0 , sd = 1)
par(mfrow = c(1, 2))
qqnorm(x, col = "darkgrey")
qqline(x, lty = 2, lwd = 2, col = "dodgerblue")
qq_plot(x)
```



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.

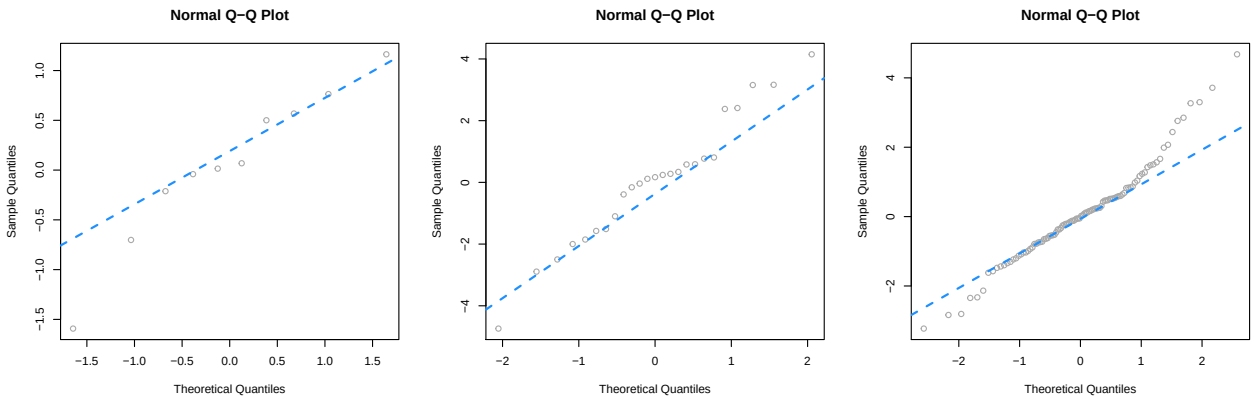
```
par(mfrow = c(1, 3))
set.seed(420)
qq_plot(rnorm(10))
qq_plot(rnorm(25))
qq_plot(rnorm(100))
```



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.

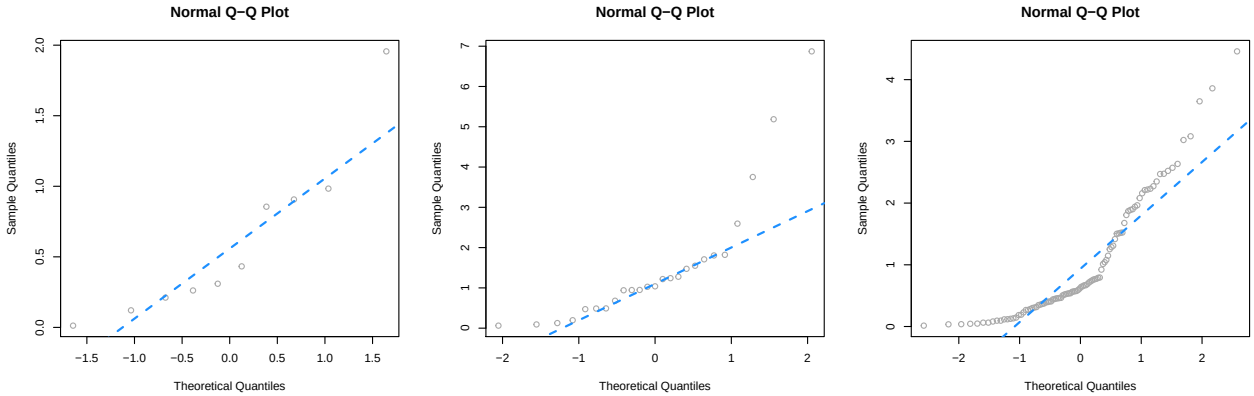
```
par(mfrow = c(1, 3))
set.seed(420)
qq_plot(rt(10, df = 4))
qq_plot(rt(25, df = 4))
qq_plot(rt(100, df = 4))
```



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 too 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.

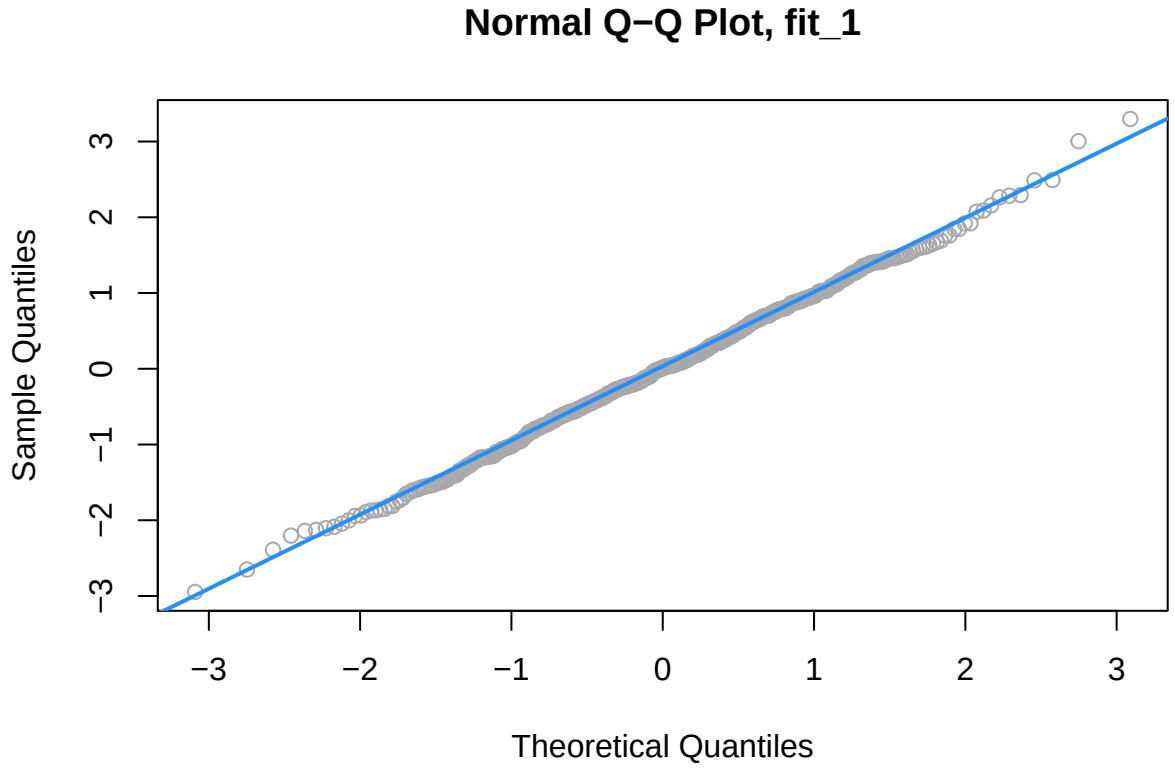
For a better understanding of which Q-Q plots are "good," repeat the simulations above a number of times (without setting the seed) and pay attention to the differences between those that are simulated from normal, and those that are not. Also consider different samples sizes and distribution parameters.

Returning to our three regressions, recall,

- `fit_1` had no violation of assumptions,
- `fit_2` violated the constant variance assumption, but not linearity,
- `fit_3` violated linearity, but not constant variance.

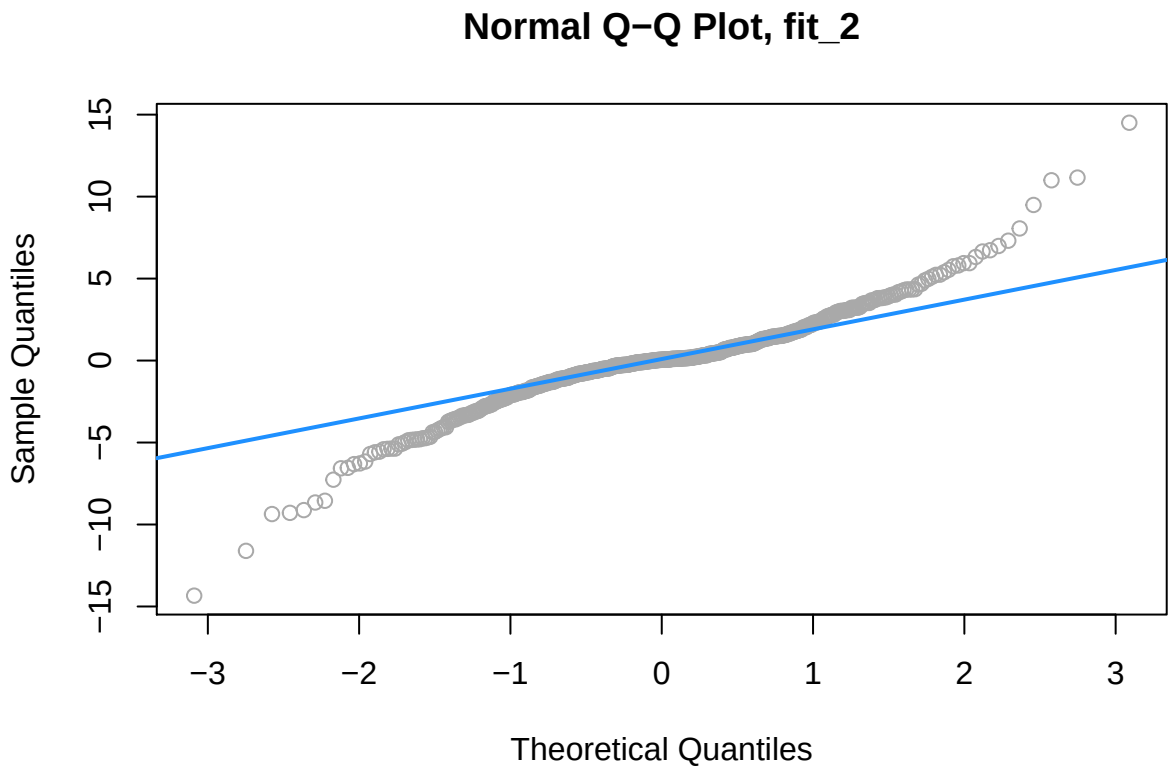
We'll now create a Q-Q plot for each to assess normality of errors.

```
qqnorm(resid(fit_1), main = "Normal Q-Q Plot, fit_1", col = "darkgrey")
qqline(resid(fit_1), col = "dodgerblue", lwd = 2)
```



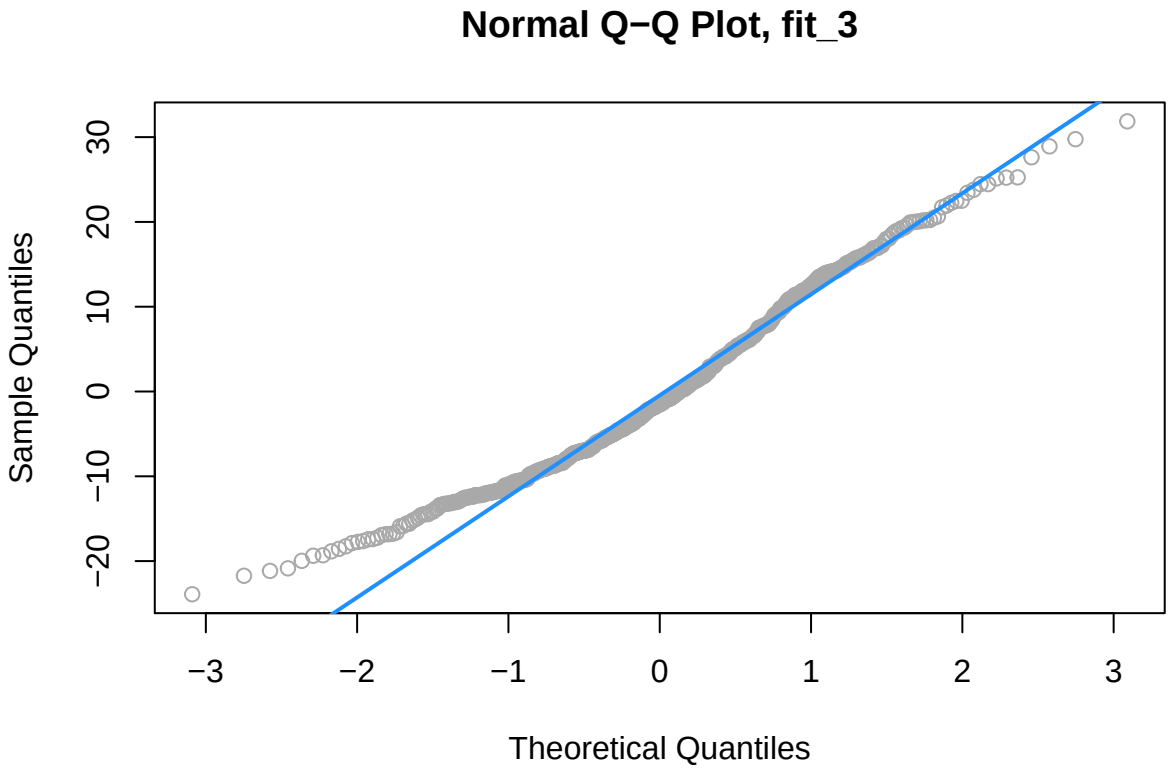
For `fit_1`, we have a near perfect Q-Q plot. We would believe the errors follow a normal distribution.

```
qqnorm(resid(fit_2), main = "Normal Q-Q Plot, fit_2", col = "darkgrey")
qqline(resid(fit_2), col = "dodgerblue", lwd = 2)
```



For `fit_2`, we have a suspect Q-Q plot. We would probably **not** believe the errors follow a normal distribution.

```
qqnorm(resid(fit_3), main = "Normal Q-Q Plot, fit_3", col = "darkgrey")
qqline(resid(fit_3), col = "dodgerblue", lwd = 2)
```



Lastly, for `fit_3`, we again have a suspect Q-Q plot. We would probably **not** believe the errors follow a normal distribution.

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 = 1.05e-05
```

This gives us the value of the test statistic and its p-value. The null hypothesis assumes the data were sampled from a normal distribution, thus a small p-value indicates we believe there is only a small probability the data could have been sampled from 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, for any reasonable α.

Returning again to `fit_1`, `fit_2` and `fit_3`, 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(fit_1))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(fit_1)
## W = 0.99858, p-value = 0.9622
```

```
shapiro.test(resid(fit_2))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(fit_2)
## W = 0.93697, p-value = 1.056e-13
```

```
shapiro.test(resid(fit_3))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(fit_3)
## W = 0.97643, p-value = 3.231e-07
```

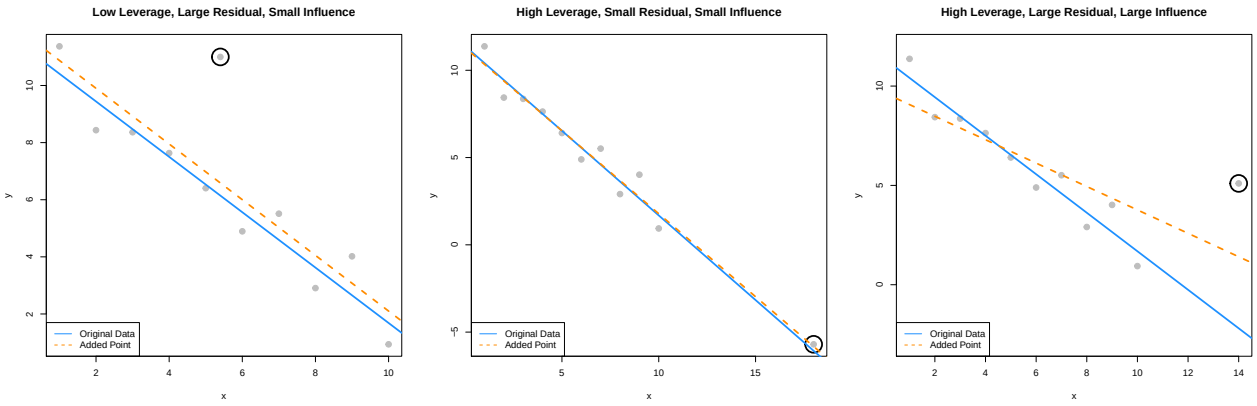
Unusual Observations

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(n = 10))
ex_model = lm(y ~ x, data = ex_data)

# low leverage, large residual, small influence
point_1 = c(5.4, 11)
ex_data_1 = rbind(ex_data, point_1)
model_1 = lm(y ~ x, data = ex_data_1)
plot(y ~ x, data = ex_data_1, cex = 2, pch = 20, col = "grey",
     main = "Low Leverage, Large Residual, Small Influence")
points(x = point_1[1], y = point_1[2], pch = 1, cex = 4, col = "black", lwd = 2)
abline(ex_model, col = "dodgerblue", lwd = 2)
abline(model_1, lty = 2, col = "darkorange", lwd = 2)
legend("bottomleft", c("Original Data", "Added Point"),
       lty = c(1, 2), col = c("dodgerblue", "darkorange"))

# high leverage, small residual, small influence
point_2 = c(18, -5.7)
```

```
ex_data_2 = rbind(ex_data, point_2)
model_2 = lm(y ~ x, data = ex_data_2)
plot(y ~ x, data = ex_data_2, cex = 2, pch = 20, col = "grey",
     main = "High Leverage, Small Residual, Small Influence")
points(x = point_2[1], y = point_2[2], pch = 1, cex = 4, col = "black", lwd = 2)
abline(ex_model, col = "dodgerblue", lwd = 2)
abline(model_2, lty = 2, col = "darkorange", lwd = 2)
legend("bottomleft", c("Original Data", "Added Point"),
       lty = c(1, 2), col = c("dodgerblue", "darkorange"))

# high leverage, large residual, large influence
point_3 = c(14, 5.1)
ex_data_3 = rbind(ex_data, point_3)
model_3 = lm(y ~ x, data = ex_data_3)
plot(y ~ x, data = ex_data_3, cex = 2, pch = 20, col = "grey", ylim = c(-3, 12),
     main = "High Leverage, Large Residual, Large Influence")
points(x = point_3[1], y = point_3[2], pch = 1, cex = 4, col = "black", lwd = 2)
abline(ex_model, col = "dodgerblue", lwd = 2)
abline(model_3, lty = 2, col = "darkorange", lwd = 2)
legend("bottomleft", c("Original Data", "Added Point"),
       lty = c(1, 2), col = c("dodgerblue", "darkorange"))
```



The blue solid line in each plot is a regression fit 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 circled point.

The slope of the regression for the original ten points, the solid blue line, is given by:

```
coef(ex_model)[2]
```

```
##          x
## -0.9696033
```

The added point in the first plot has a *small* effect on the slope, which becomes:

```
coef(model_1)[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:

```
coef(model_2)[2]
```

```
##          x
## -0.9507397
```

We will say that this point has high leverage, is not an outlier due to its small residual, and has a very small influence.

Lastly, the added point in the third plot has a *large* effect on the slope, which is now:

```
coef(model_3)[2]
```

```
##          x
## -0.5892241
```

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.

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,

$$\bar{h} = \frac{\sum_{i=1}^{n} h_i}{n} = \frac{p}{n}.$$

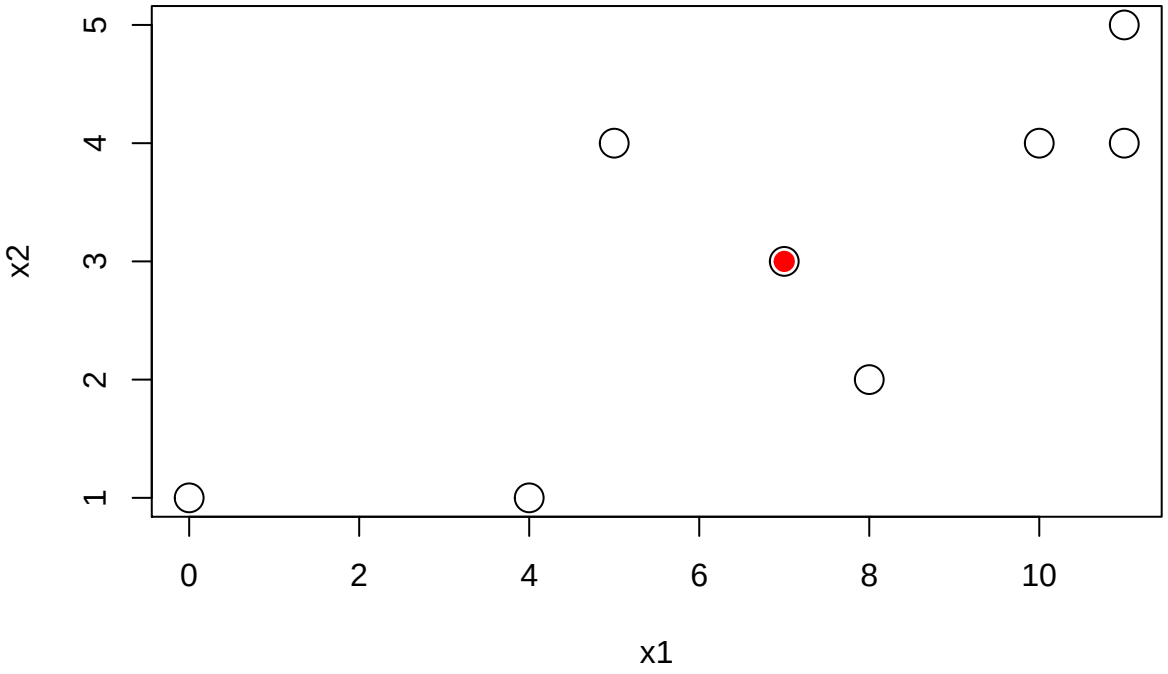
For simple linear regression, the leverage for each point is given by

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{S_{xx}}.$$

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.

```
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.

```
which.max(hatvalues(lev_fit))
```

```
## 1
## 1
```

```
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.

```
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.

```
which.min(hatvalues(lev_fit))
```

```
## 4
## 4
```

```
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`.

```
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!

```
mean(lev_ex$x1)
```

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

```
mean(lev_ex$x2)
```

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

```
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.

```
hatvalues(model_1)
```

```
##          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
```

```
hatvalues(model_2)
```

```
##          1          2          3          4          5          6          7
## 0.23238866 0.18663968 0.14979757 0.12186235 0.10283401 0.09271255 0.09149798
##          8          9         10         11
## 0.09919028 0.11578947 0.14129555 0.66599190
```

```
hatvalues(model_3)
```

```
##           1           2           3           4           5           6           7
## 0.27852761 0.21411043 0.16319018 0.12576687 0.10184049 0.09141104 0.09447853
##           8           9          10          11
## 0.11104294 0.14110429 0.18466258 0.49386503
```

Are any of these large?

```
hatvalues(model_1) > 2 * mean(hatvalues(model_1))
```

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

```
hatvalues(model_2) > 2 * mean(hatvalues(model_2))
```

```
##     1     2     3     4     5     6     7     8     9    10    11
## FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```

```
hatvalues(model_3) > 2 * mean(hatvalues(model_3))
```

```
##     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.

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,

$$\text{Var}(e_i) = (1 - h_i)\sigma^2$$

and thus estimating σ^2 with s_e^2 gives

$$\text{SE}[e_i] = s_e\sqrt{(1 - h_i)}.$$

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(model_1)
```

```
##          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(model_1)
```

```
##          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(model_1)[abs(rstandard(model_1)) > 2]
```

```
##       11
## 2.641823
```

In the first plot, we see that the 11th point, the added point, is a large standardized residual.

```
resid(model_2)
```

```
##          1          2          3          4          5          6
##  1.03288292 -0.95203397 -0.07346766  0.14700626 -0.13084829 -0.69050140
##          7          8          9         10         11
##  0.87788484 -0.77755647  1.28626601 -0.84413207  0.12449986
```

```
rstandard(model_2)
```

```
##          1          2          3          4          5          6
##  1.41447023 -1.26655590 -0.09559792  0.18822094 -0.16574677 -0.86977220
##          7          8          9         10         11
##  1.10506546 -0.98294409  1.64121833 -1.09295417  0.25846620
```

```
rstandard(model_2)[abs(rstandard(model_2)) > 2]
```

```
## named numeric(0)
```

In the second plot, we see that there are no points with large standardized residuals.

```
resid(model_3)
```

```
##          1          2          3          4          5          6
##  2.30296166 -0.04347087  0.47357980  0.33253808 -0.30683212 -1.22800087
##          7          8          9         10         11
## -0.02113027 -2.03808722 -0.33578039 -2.82769411  3.69191633
```

```
rstandard(model_3)
```

```
##          1          2          3          4          5          6
##  1.41302755 -0.02555591  0.26980722  0.18535382 -0.16873216 -0.67141143
##          7          8          9         10         11
## -0.01157256 -1.12656475 -0.18882474 -1.63206526  2.70453408
```

```
rstandard(model_3)[abs(rstandard(model_3)) > 2]
```

```
##       11
## 2.704534
```

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.

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 latter category, points with (some combination of) *high leverage* **and** *large residual*, we will call **influential**.

A common measure of influence is **Cook's Distance**, which is defined as

$$D_i = \frac{1}{p} r_i^2 \frac{h_i}{1 - h_i}.$$

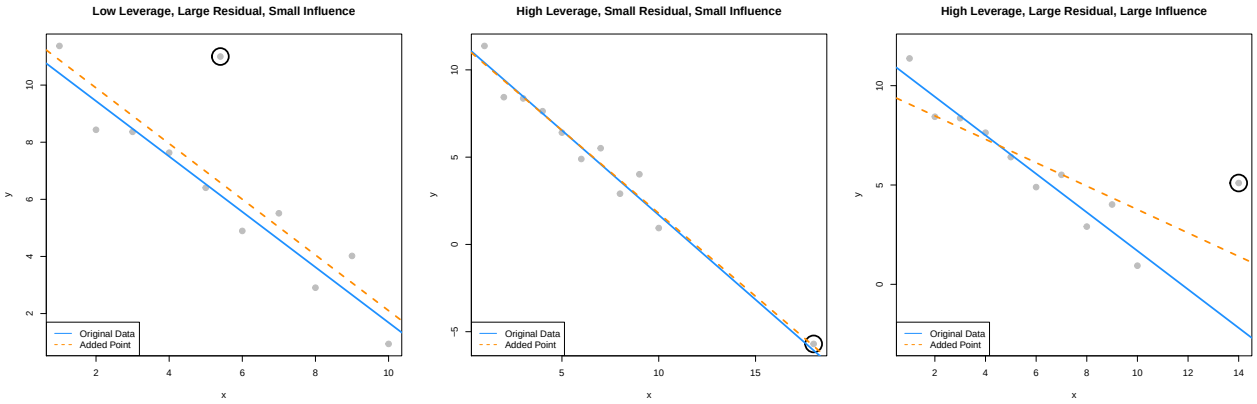
Notice that this is a function of both *leverage* and *standardized residuals*.

A Cook's Distance is often 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 circled points 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(model_1)[11] > 4 / length(cooks.distance(model_1))
```

```
##    11
## FALSE
```

```
cooks.distance(model_2)[11] > 4 / length(cooks.distance(model_2))
```

```
##    11
## FALSE
```

```
cooks.distance(model_3)[11] > 4 / length(cooks.distance(model_3))
```

```
##   11
## TRUE
```

And, as expected, the added point in the third plot, with high leverage and a large residual is considered influential!

Data Analysis Examples

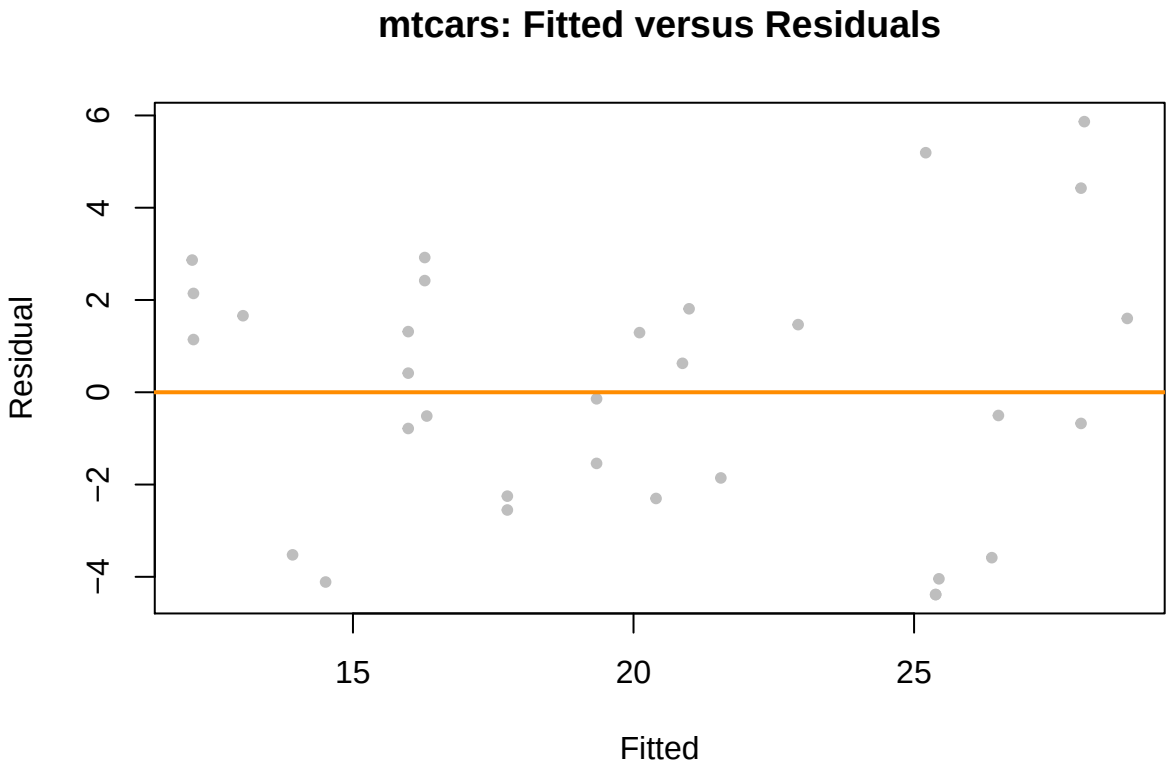
Good Diagnostics

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 = "grey", pch = 20,
     xlab = "Fitted", ylab = "Residual",
     main = "mtcars: Fitted versus Residuals")
abline(h = 0, col = "darkorange", lwd = 2)
```



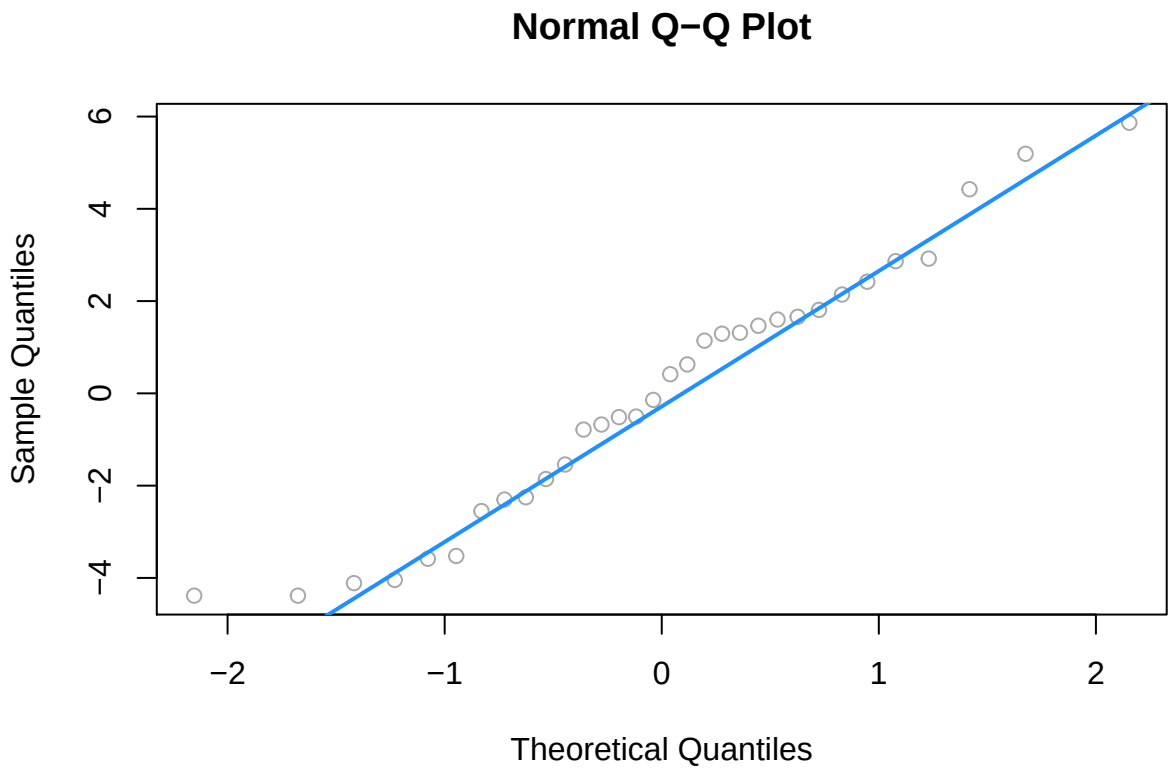
The fitted versus residuals plot looks good. We don't see any obvious pattern, and the variance looks roughly constant. (Maybe a little larger for large fitted values, but not enough to worry about.)

```
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 = "darkgrey")
qqline(resid(mpg_hp_add), col = "dodgerblue", 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
```

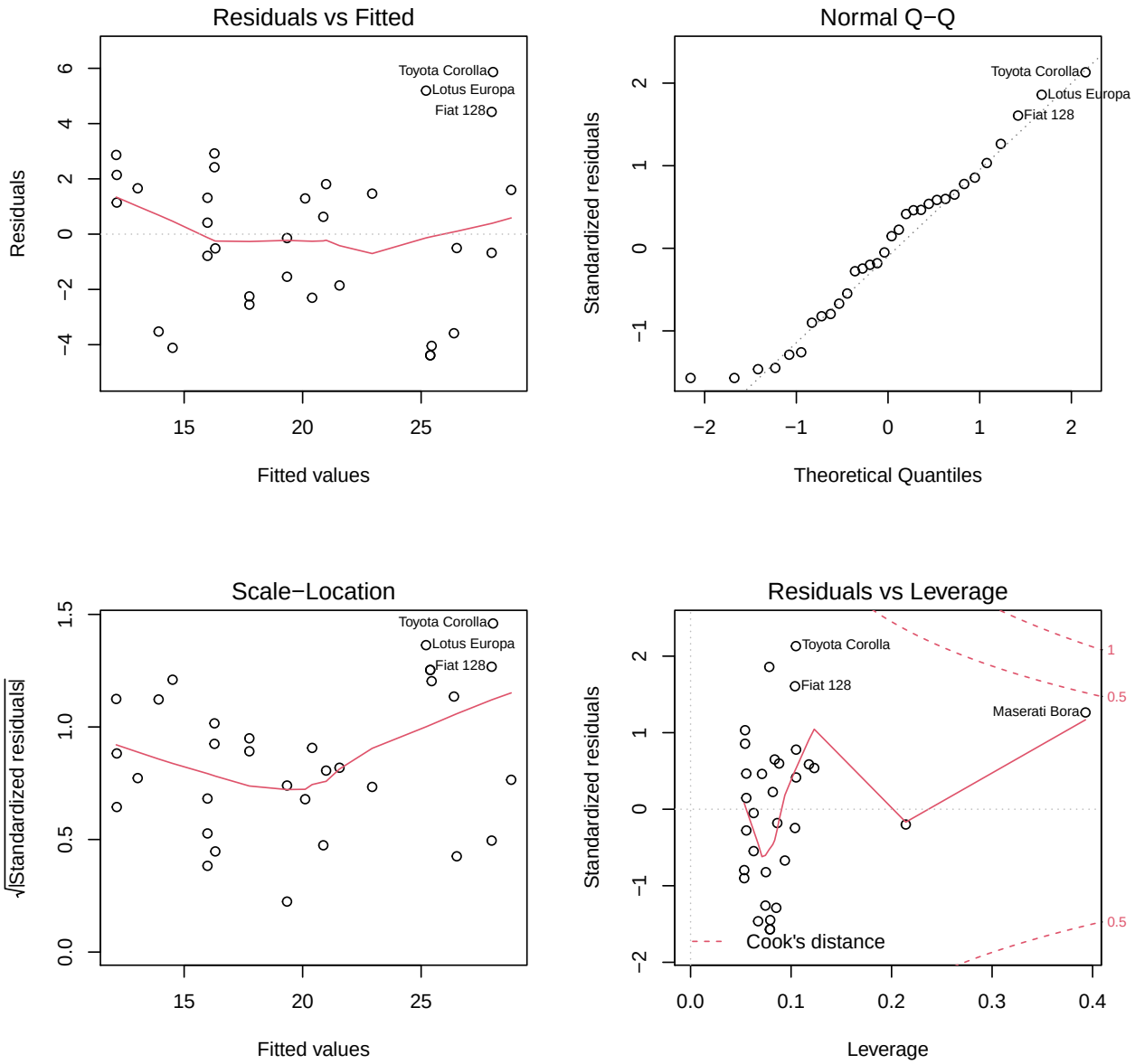
Since the diagnostics looked good, there isn't much need to worry about these two points, but let's see how much the coefficients change if we remove them.

```
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.

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`.

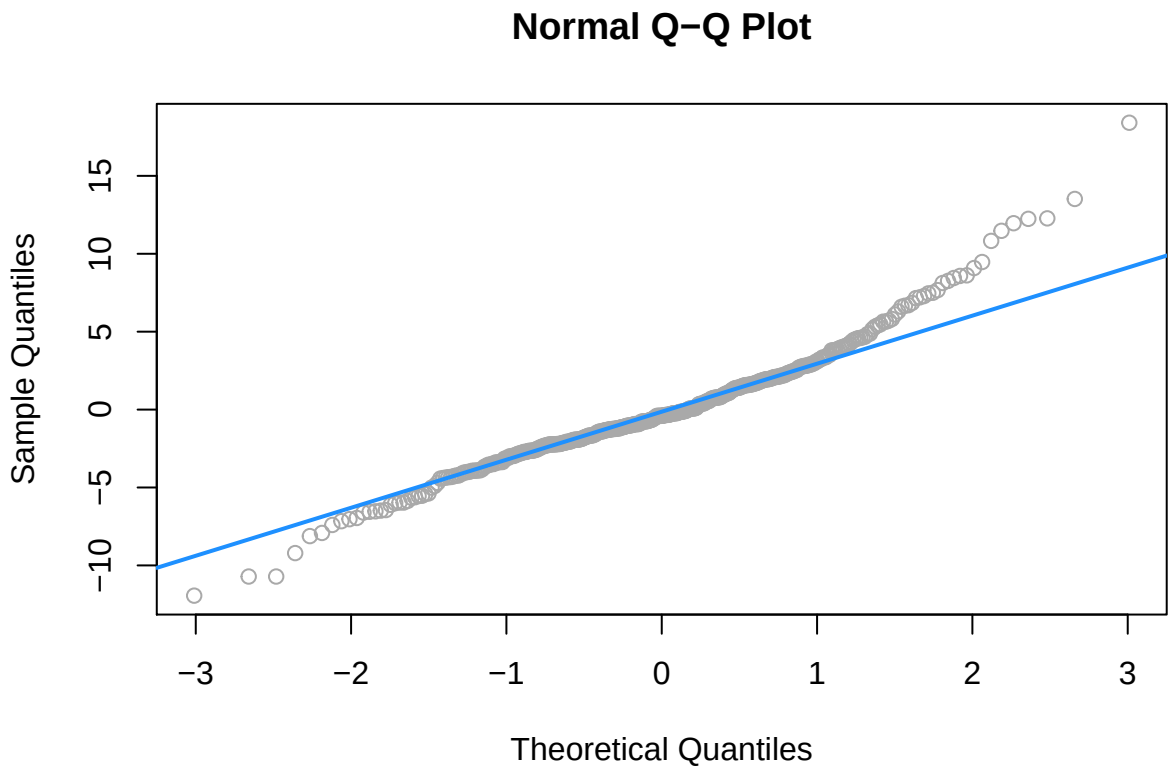
```
str(autompg)
```

```
## '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 = "darkgrey")
qqline(resid(big_model), col = "dodgerblue", lwd = 2)
```




```
shapiro.test(resid(big_model))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(big_model)
## W = 0.96161, p-value = 1.824e-08
```

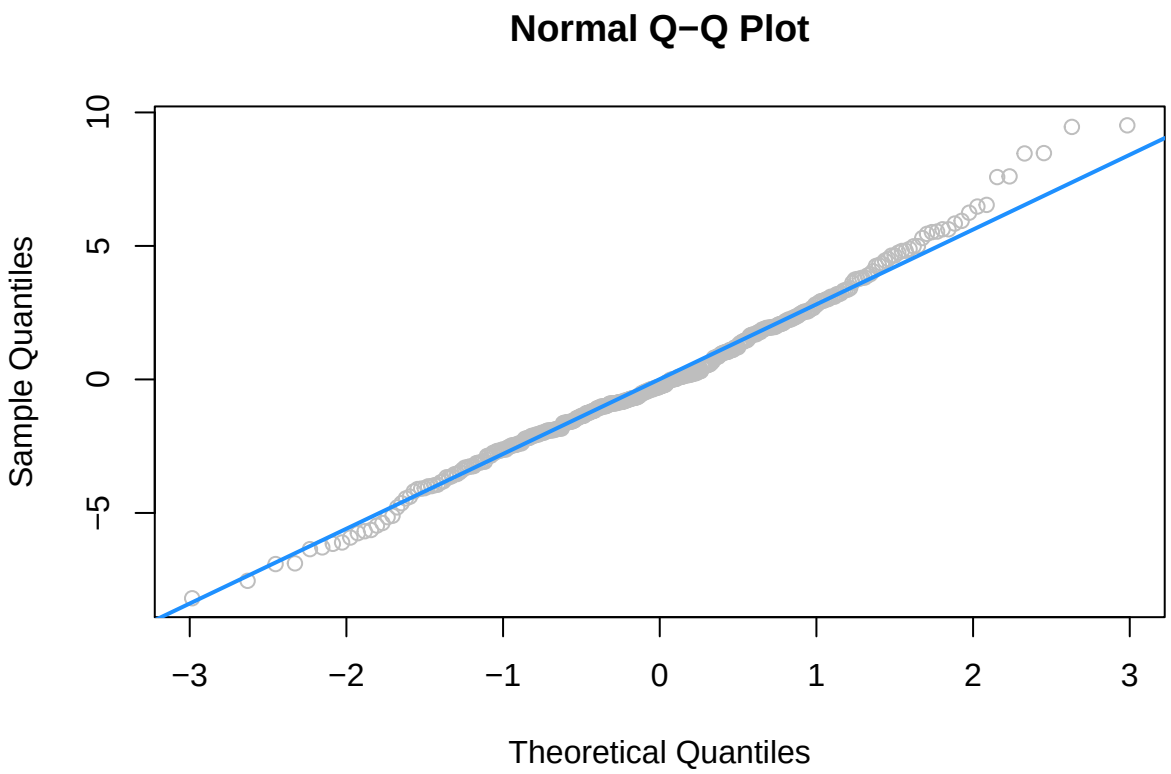
Here both the Q-Q plot, and the Shapiro-Wilk test suggest that the normality assumption is violated.

```
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_cd))
qqnorm(resid(big_model_fix), col = "grey")
qqline(resid(big_model_fix), col = "dodgerblue", 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*.

Transformations

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. Going forward they will likely be replaced with alternative sourceable data that illustrates the same concepts. At the end of this chapter you can find code seen in videos for Week 8 for STAT 420 in the MCS-DS program. It is currently in the process of being merged into the narrative of this chapter.

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.

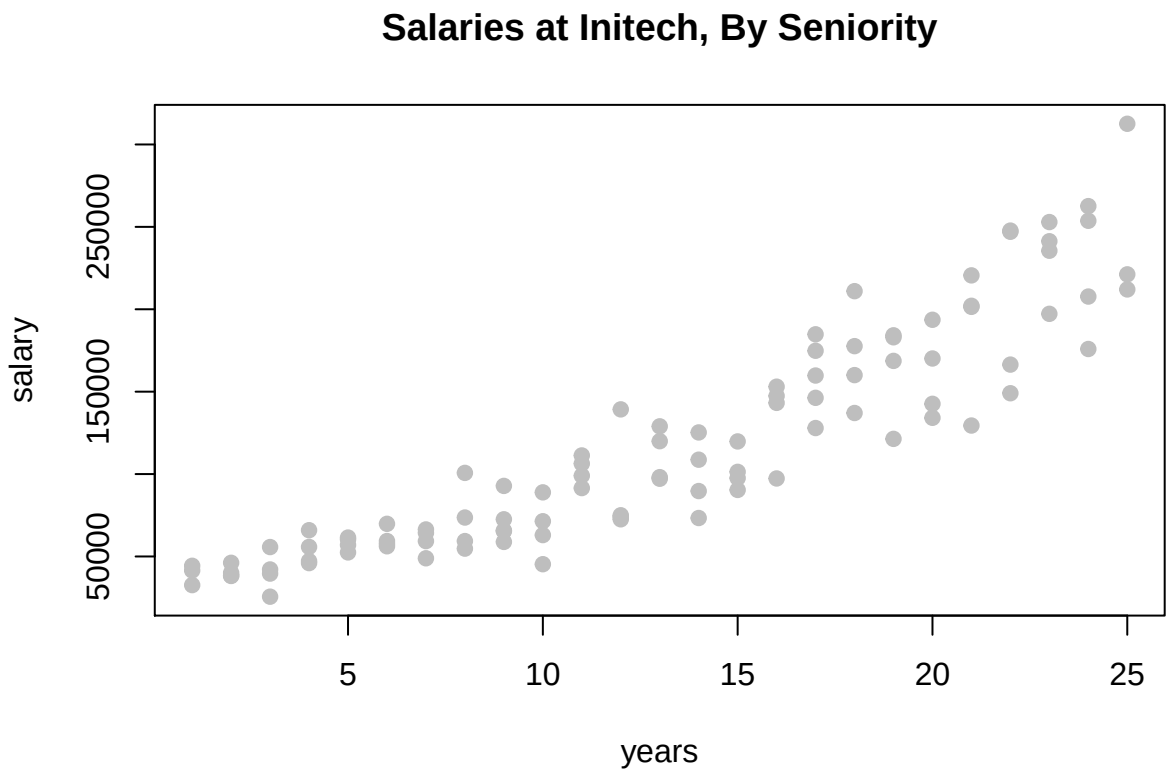
Last chapter we checked the assumptions of regression models and looked at ways to diagnose possible issues. This chapter we will use transformations of both response and predictor variables in order to correct issues with model diagnostics, and to also potentially simply make a model fit data better.

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("C:\\Users\\root\\Documents\\Rprojects\\Rmd-ex-paper-more-03Oct2021\\data\\initech.cs
```

```
plot(salary ~ years, data = initech, col = "grey", pch = 20, cex = 1.5,
     main = "Salaries at Initech, By Seniority")
```



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
## -57225 -18104    241  15589  91332
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)     5302       5750   0.922    0.359
## years           8637        389  22.200   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 27360 on 98 degrees of freedom
## Multiple R-squared:  0.8341, Adjusted R-squared:  0.8324
```

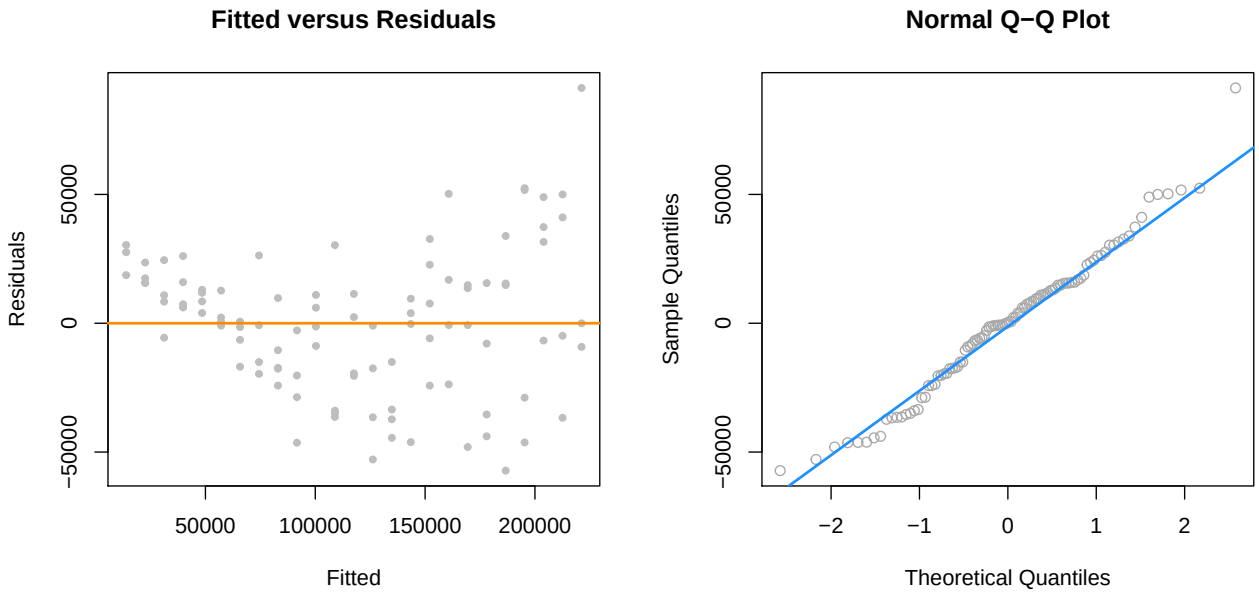
```
## F-statistic: 492.8 on 1 and 98 DF,  p-value: < 2.2e-16
```

This model appears significant, but does it meet the model assumptions?

```
plot(salary ~ years, data = initech, col = "grey", pch = 20, cex = 1.5,
     main = "Salaries at Initech, By Seniority")
abline(initech_fit, col = "darkorange", lwd = 2)
```

Adding the fitted line to the plot, we see that the linear relationship appears correct.

```
par(mfrow = c(1, 2))

plot(fitted(initech_fit), resid(initech_fit), col = "grey", pch = 20,
     xlab = "Fitted", ylab = "Residuals", main = "Fitted versus Residuals")
abline(h = 0, col = "darkorange", lwd = 2)

qqnorm(resid(initech_fit), main = "Normal Q-Q Plot", col = "darkgrey")
qqline(resid(initech_fit), col = "dodgerblue", 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.

Variance Stabilizing Transformations

Recall the fitted value is our estimate of the mean at a particular value of x. Under our usual assumptions,

$$\epsilon \sim N(0, \sigma^2)$$

and thus

$$\text{Var}[Y|X = x] = \sigma^2$$

which is a constant value for any value of x.

However, here we see that the variance is a function of the mean,

$$\text{Var}[Y \mid X = x] = h(\text{E}[Y \mid X = x]).$$

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,

$$\text{Var}[g(Y) \mid X = x] = c$$

where c is a constant that does not depend on the mean, $\text{E}[Y \mid X = x]$. 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 model with 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 model 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.

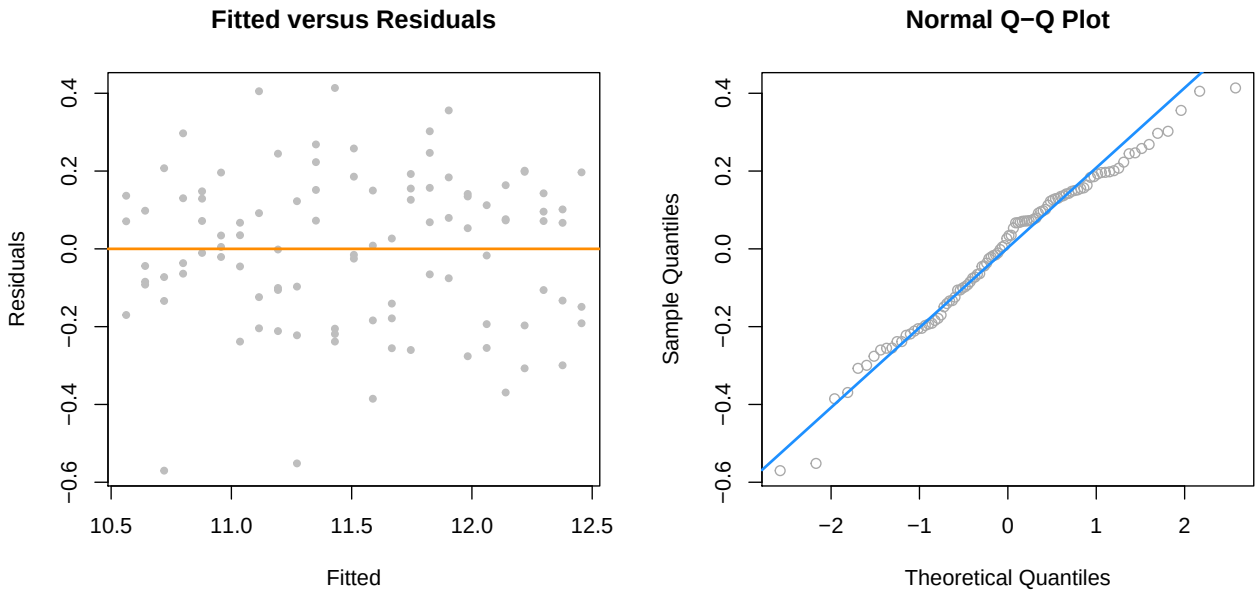
```
plot(log(salary) ~ years, data = initech, col = "grey", pch = 20, cex = 1.5,
     main = "Salaries at Initech, By Seniority")
abline(initech_fit_log, col = "darkorange", lwd = 2)
```

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 = "grey", pch = 20, cex = 1.5,
     main = "Salaries at Initech, By Seniority")
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.

```
par(mfrow = c(1, 2))

plot(fitted(initech_fit_log), resid(initech_fit_log), col = "grey", pch = 20,
     xlab = "Fitted", ylab = "Residuals", main = "Fitted versus Residuals")
abline(h = 0, col = "darkorange", lwd = 2)

qqnorm(resid(initech_fit_log), main = "Normal Q-Q Plot", col = "darkgrey")
qqline(resid(initech_fit_log), col = "dodgerblue", 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] 27080.16
```

```
sqrt(mean(resid(initech_fit_log) ^ 2))
```

```
## [1] 0.1934907
```

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] 27080.16
```

```
sqrt(mean((initech$salary - exp(fitted(initech_fit_log))) ^ 2))
```

```
## [1] 24280.36
```

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
## -0.57022 -0.13560  0.03048  0.14157  0.41366
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.48381    0.04108  255.18   <2e-16 ***
## years        0.07888    0.00278   28.38   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1955 on 98 degrees of freedom
## Multiple R-squared:  0.8915, Adjusted R-squared:  0.8904
## F-statistic: 805.2 on 1 and 98 DF,  p-value: < 2.2e-16
```

Again, the transformed response is a *linear* combination of the predictors,

$$\log(\hat{y}(x)) = \hat{\beta}_0 + \hat{\beta}_1 x = 10.484 + 0.079x.$$

But now, if we re-scale the data from a log scale back to the original scale of the data, we now have

$$\hat{y}(x) = \exp(\hat{\beta}_0)\exp(\hat{\beta}_1 x) = \exp(10.484)\exp(0.079x).$$

We see that for every one additional year of experience, average salary increases $\exp(0.079) = 1.0822$ 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.

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)
```

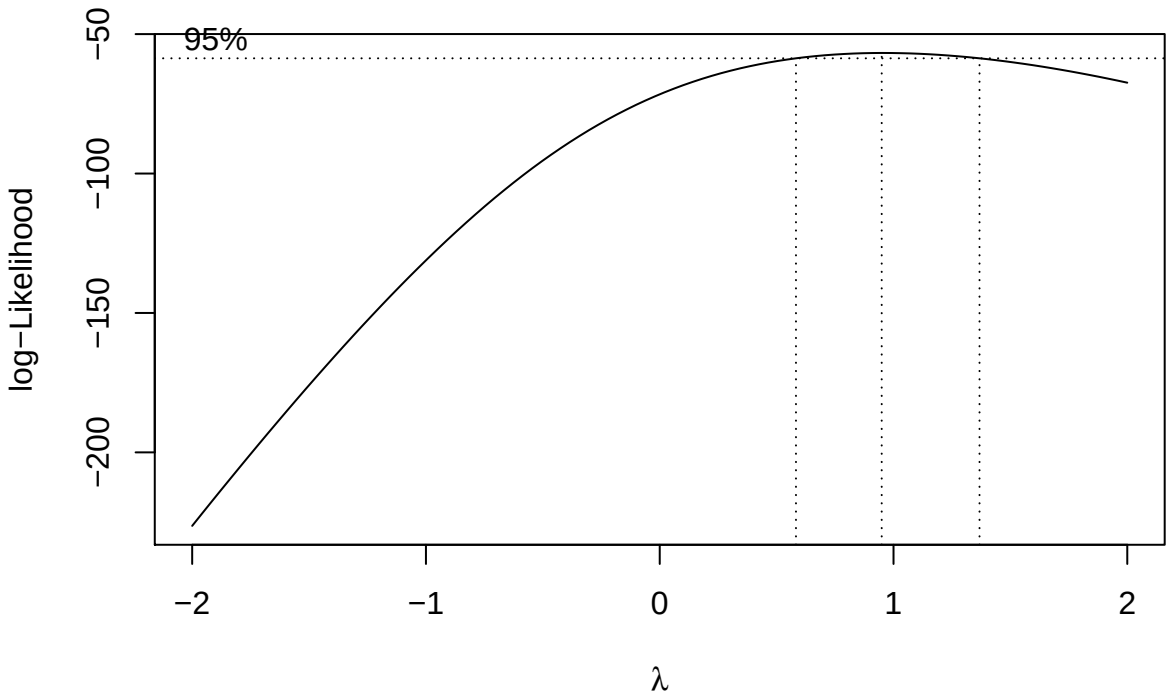
Here we need the `MASS` package for the `boxcox()` function, and we will consider a couple of datasets from the `faraway` package.

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)
```

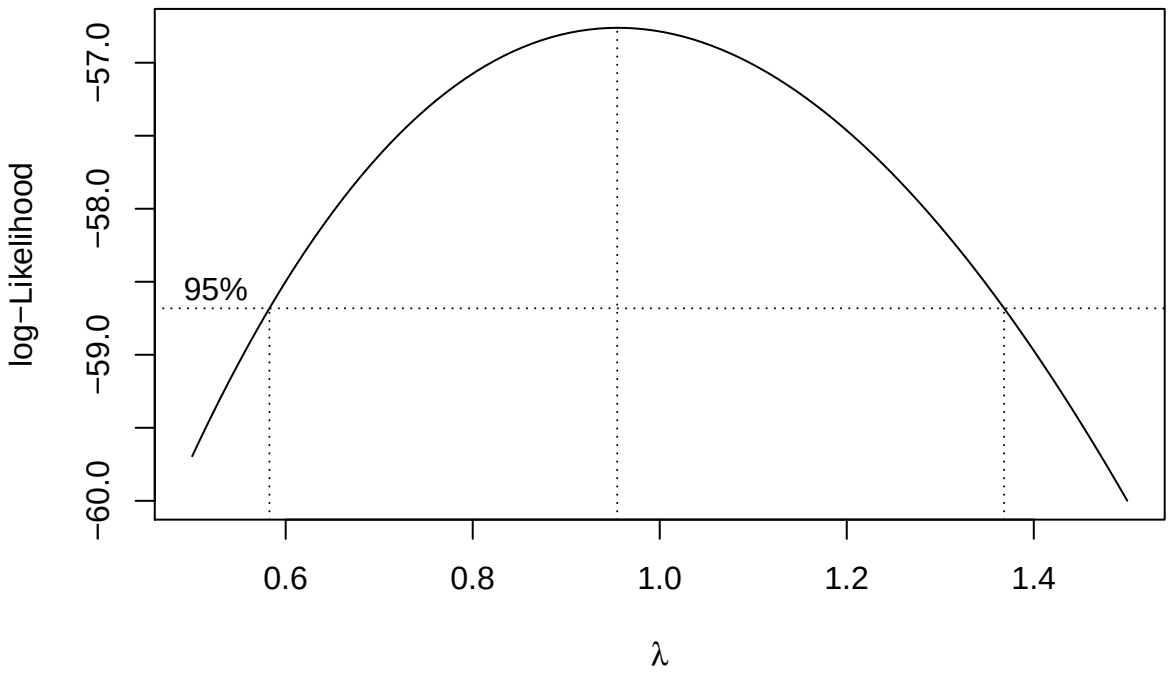
We then use the `boxcox()` function to find the best transformation of the form considered by the Box-Cox method.

```
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))
```



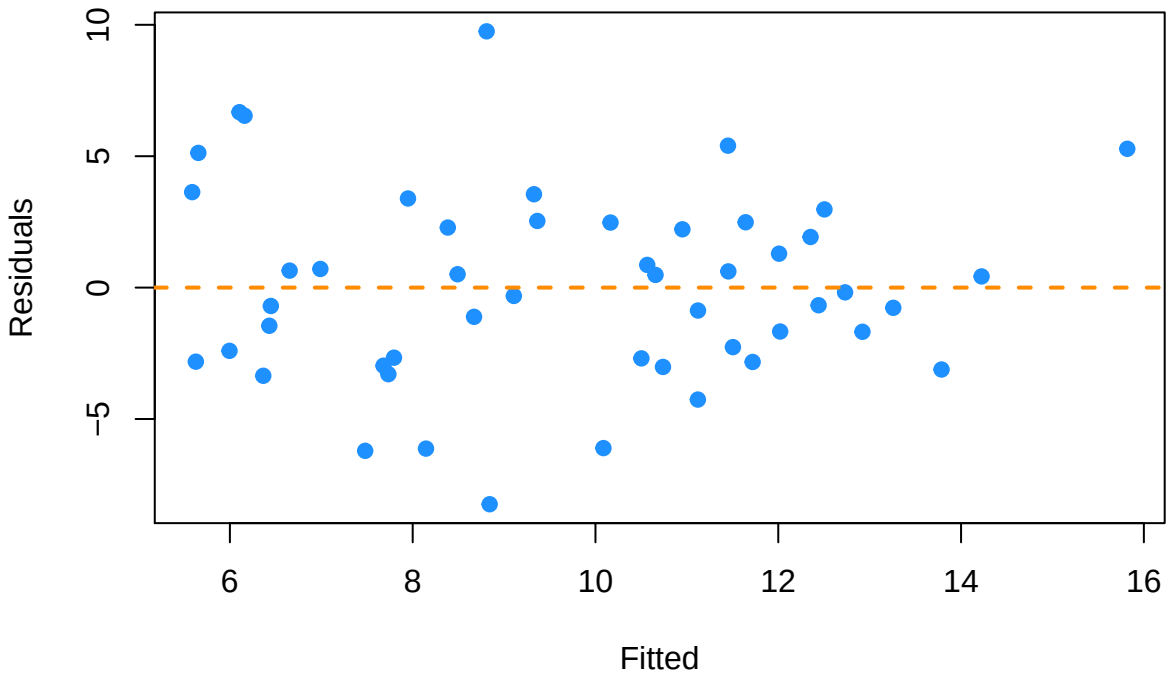
Note that we can specify a range of λ values to consider and thus be plotted. We often specify a range that is more visually interesting. Here we see that $\lambda = 1$ is both in the confidence interval, and is extremely close

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))
```

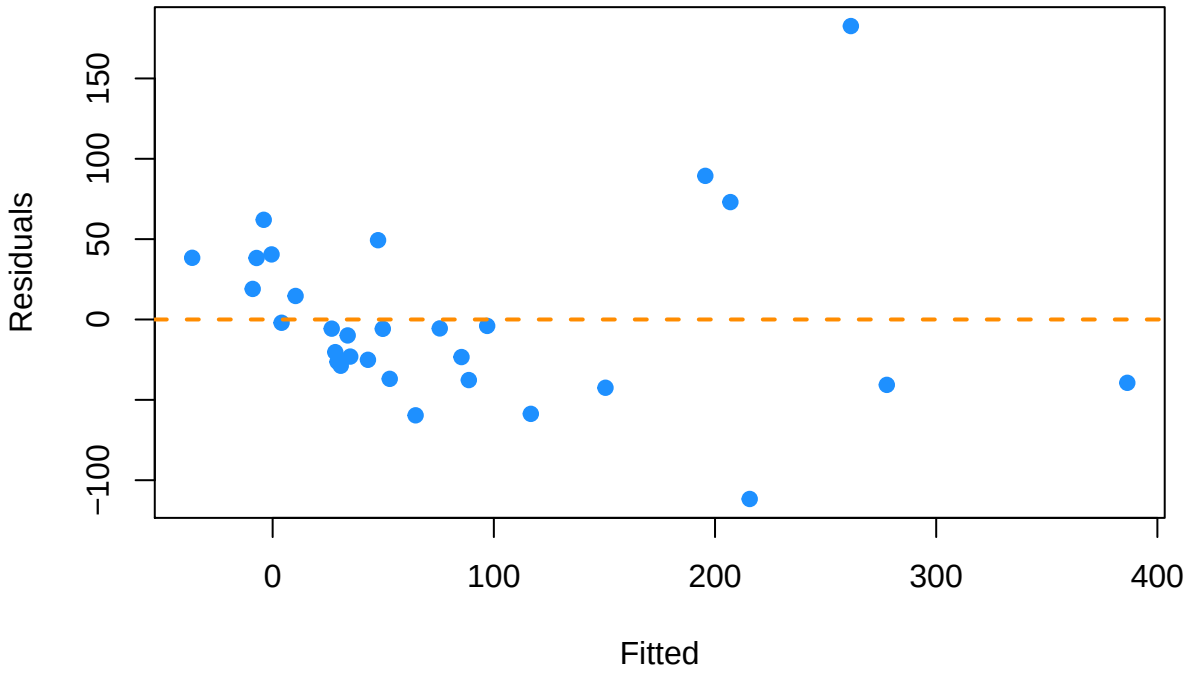
```
##
##  Shapiro-Wilk normality test
##
## data:  resid(savings_model)
## W = 0.98698, p-value = 0.8524
```

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.

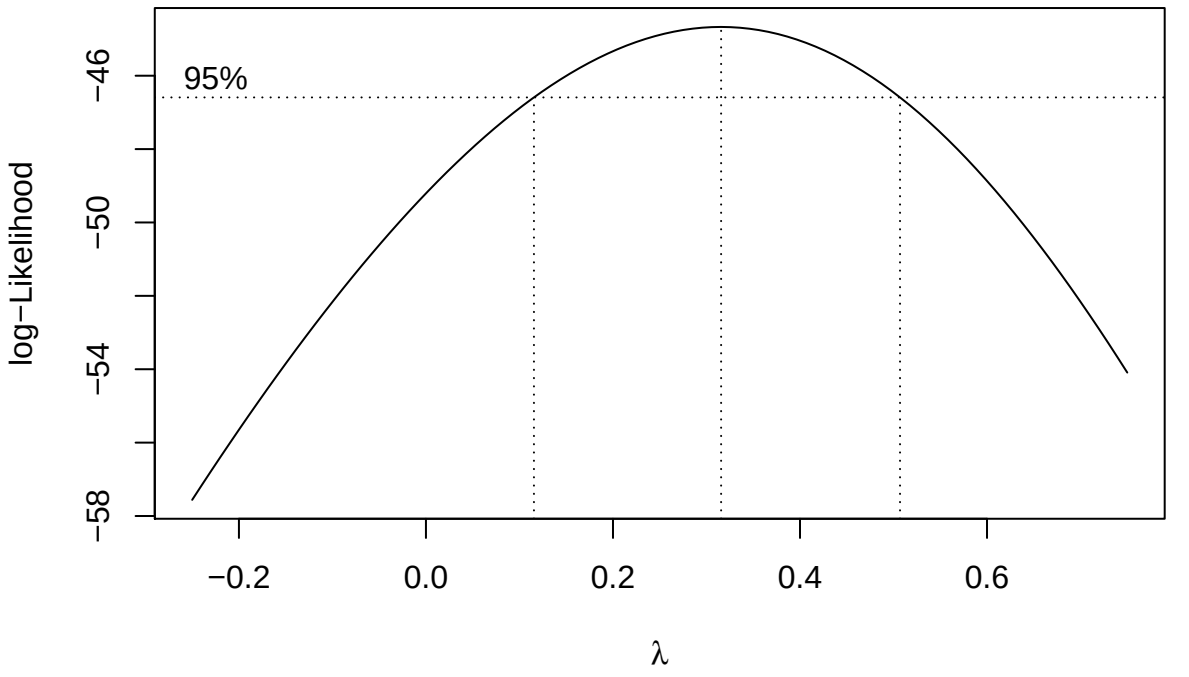
```
gala_model = lm(Species ~ Area + Elevation + Nearest + Scruz + Adjacent, data = gala)
```

```
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)
```



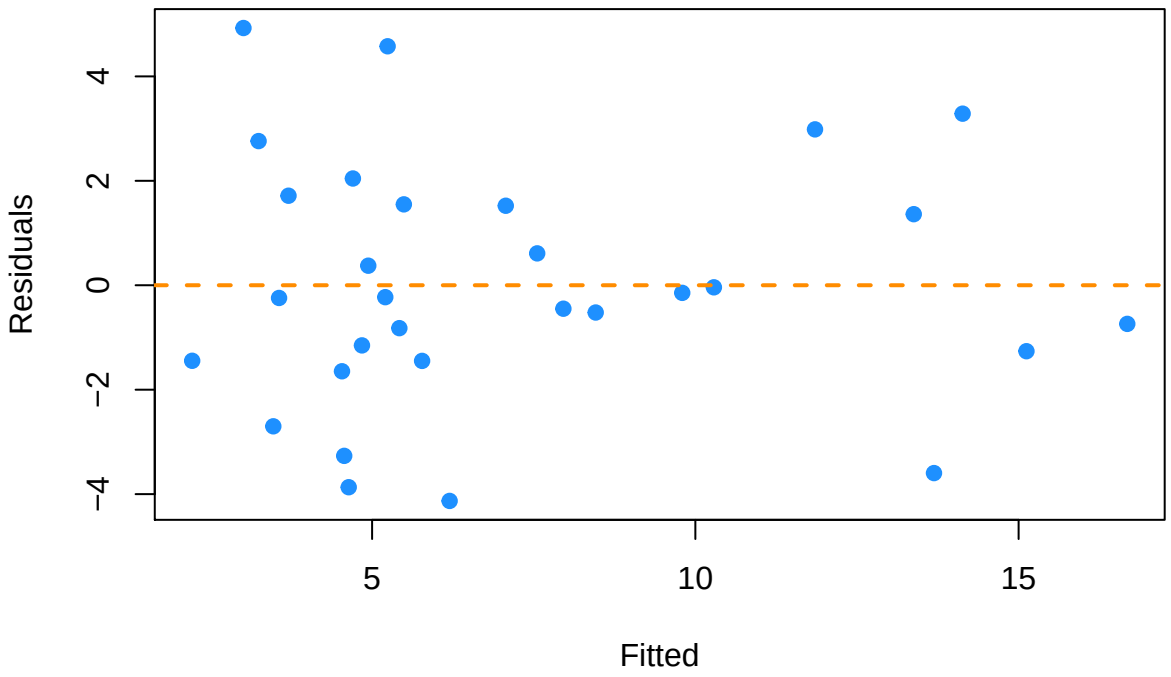
Using the Box-Cox method, we see that $\lambda = 0.3$ is both in the confidence interval, and is extremely close to the maximum, which suggests a transformation of the form

$$\frac{y^\lambda - 1}{\lambda} = \frac{y^{0.3} - 1}{0.3}.$$

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
```

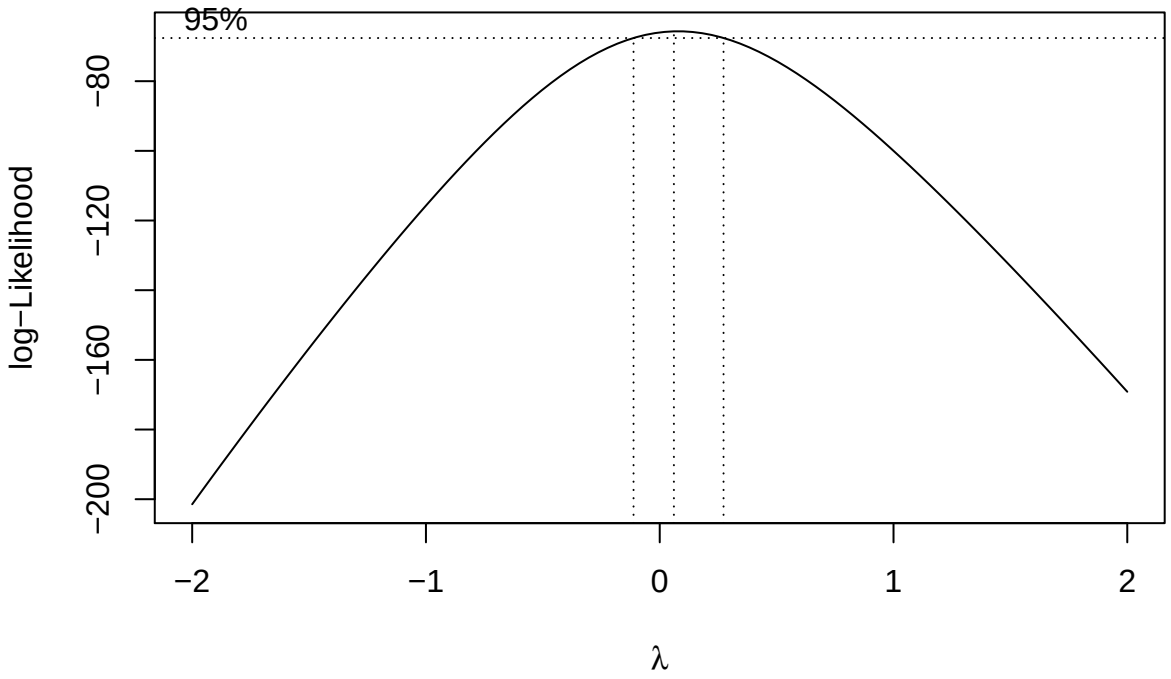
```
plot(fitted(gala_model_cox), resid(gala_model_cox), col = "dodgerblue",
     pch = 20, cex = 1.5, xlab = "Fitted", ylab = "Residuals")
abline(h = 0, lty = 2, col = "darkorange", lwd = 2)
```



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!

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.

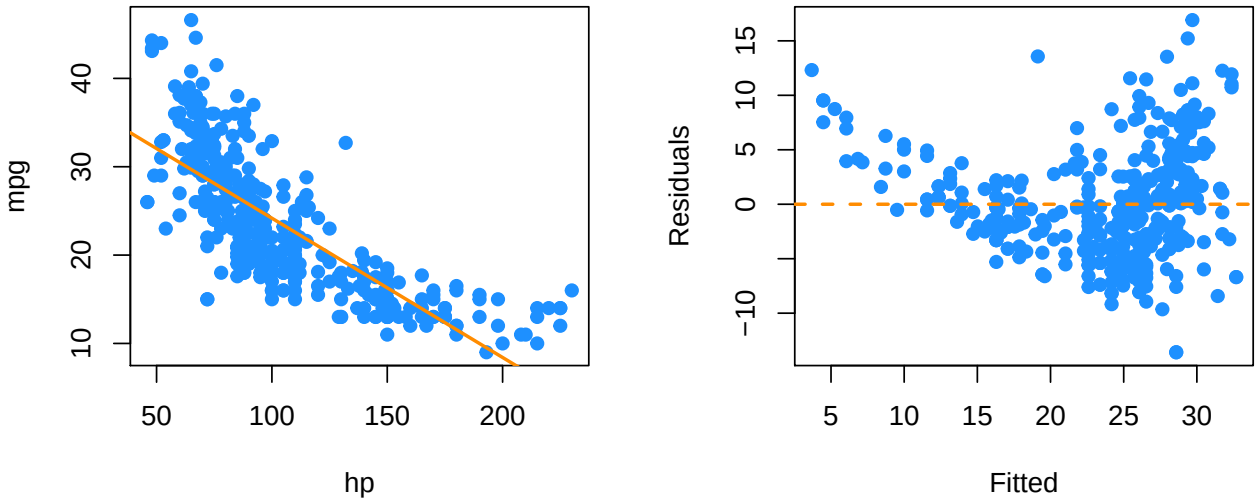
```
str(autompg)
```

```
## '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`.

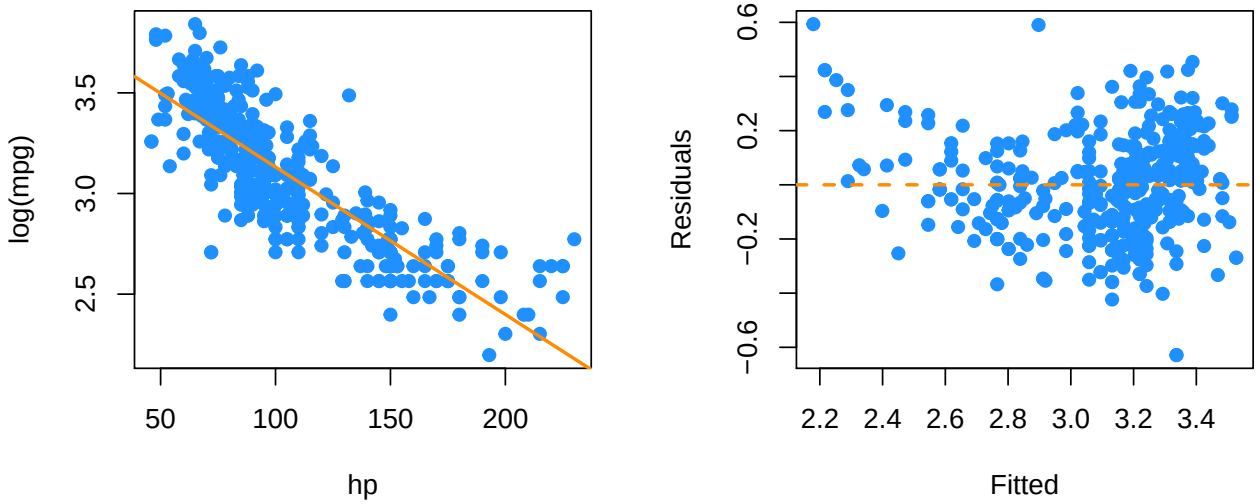
```
par(mfrow = c(1, 2))
plot(mpg ~ hp, data = autompg, col = "dodgerblue", pch = 20, cex = 1.5)
mpg_hp = lm(mpg ~ hp, data = autompg)
abline(mpg_hp, col = "darkorange", lwd = 2)
plot(fitted(mpg_hp), resid(mpg_hp), col = "dodgerblue",
```

```
     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.

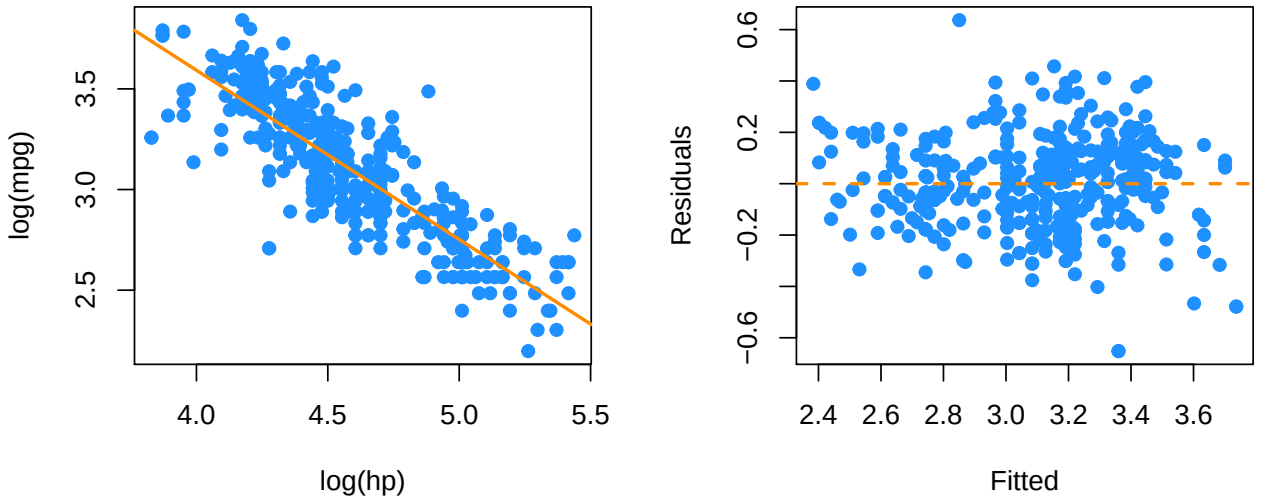
```
par(mfrow = c(1, 2))
plot(log(mpg) ~ hp, data = autompg, col = "dodgerblue", pch = 20, cex = 1.5)
mpg_hp_log = lm(log(mpg) ~ hp, data = autompg)
abline(mpg_hp_log, col = "darkorange", lwd = 2)
plot(fitted(mpg_hp_log), resid(mpg_hp_log), col = "dodgerblue",
     pch = 20, cex = 1.5, xlab = "Fitted", 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)
```



Here, our fitted versus residuals plot looks good.

Polynomials

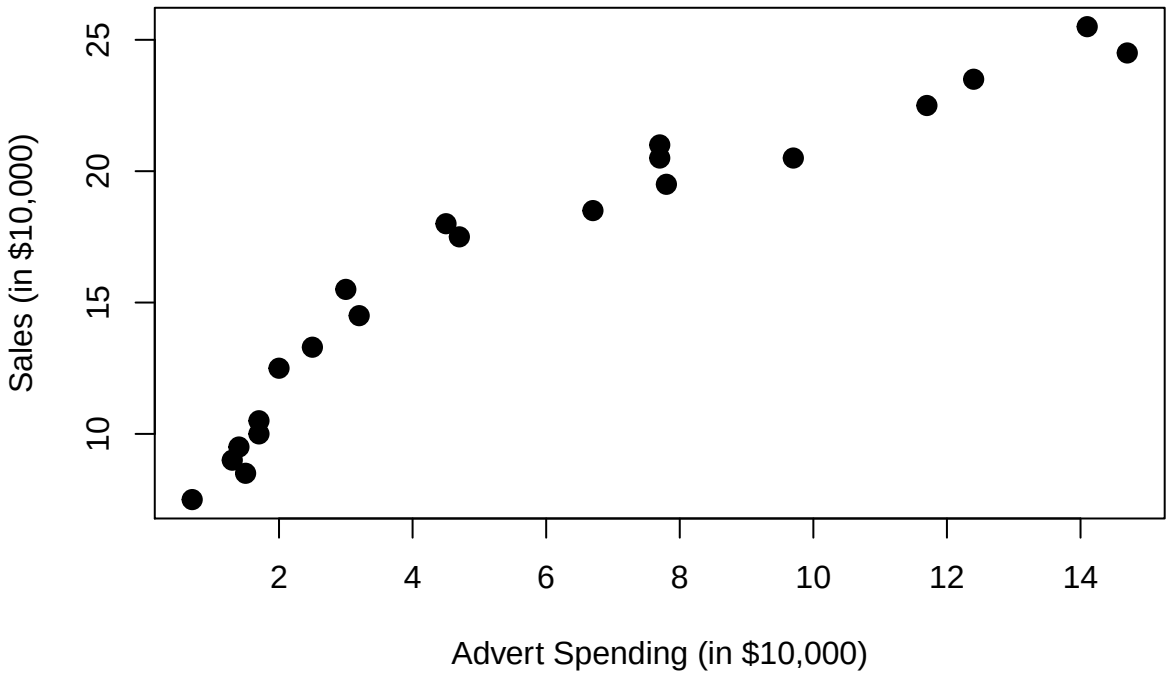
Another very common "transformation" of a predictor variable is the use of polynomial transformations. They are extremely useful as they allow for more flexible models, but do not change the units of the variables.

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("C:\\Users\\root\\Documents\\Rprojects\\Rmd-ex-paper-more-03Oct2021\\data\\marketi
```

```
plot(sales ~ advert, data = marketing,
     xlab = "Advert Spending (in $10,000)", ylab = "Sales (in $10,000)",
     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,

$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix}$$

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

$$\text{Var}[\hat{\beta}] = \sigma^2 \left(X^\top X\right)^{-1}.$$

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.

```
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
```

```
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.

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i$$

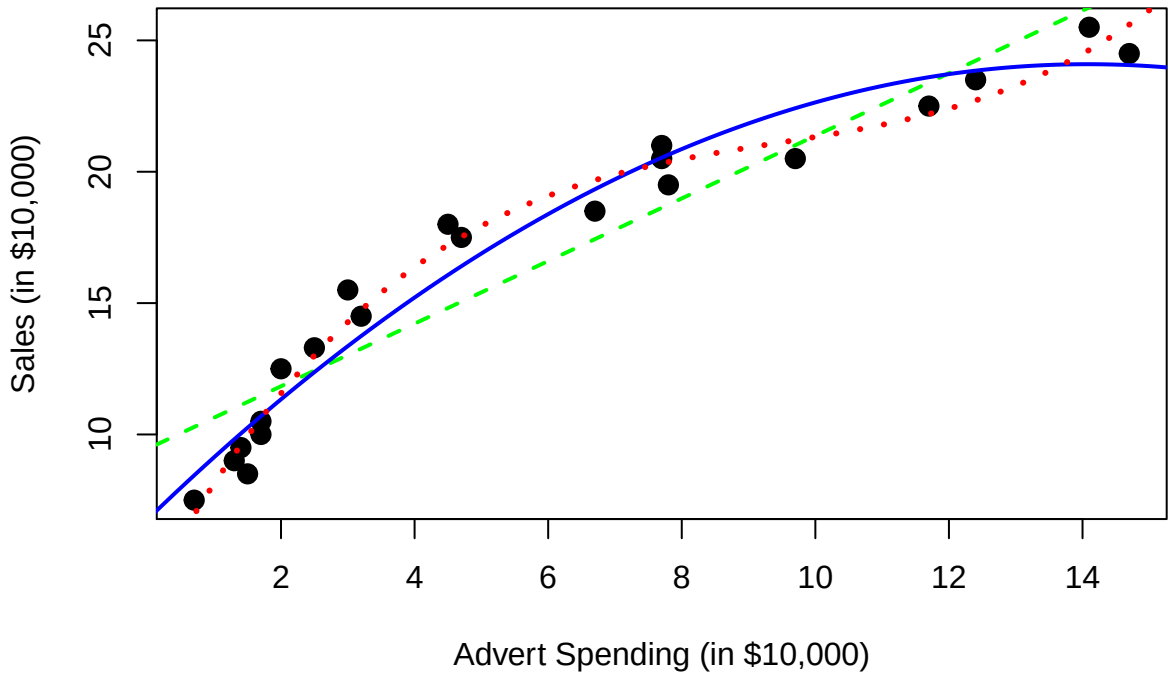
$$\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix}$$

```
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 8.79e-05 ***
## advert       4.681864   0.501032   9.344 4.14e-08 ***
## I(advert^2) -0.455152   0.078977  -5.763 2.30e-05 ***
## 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
```

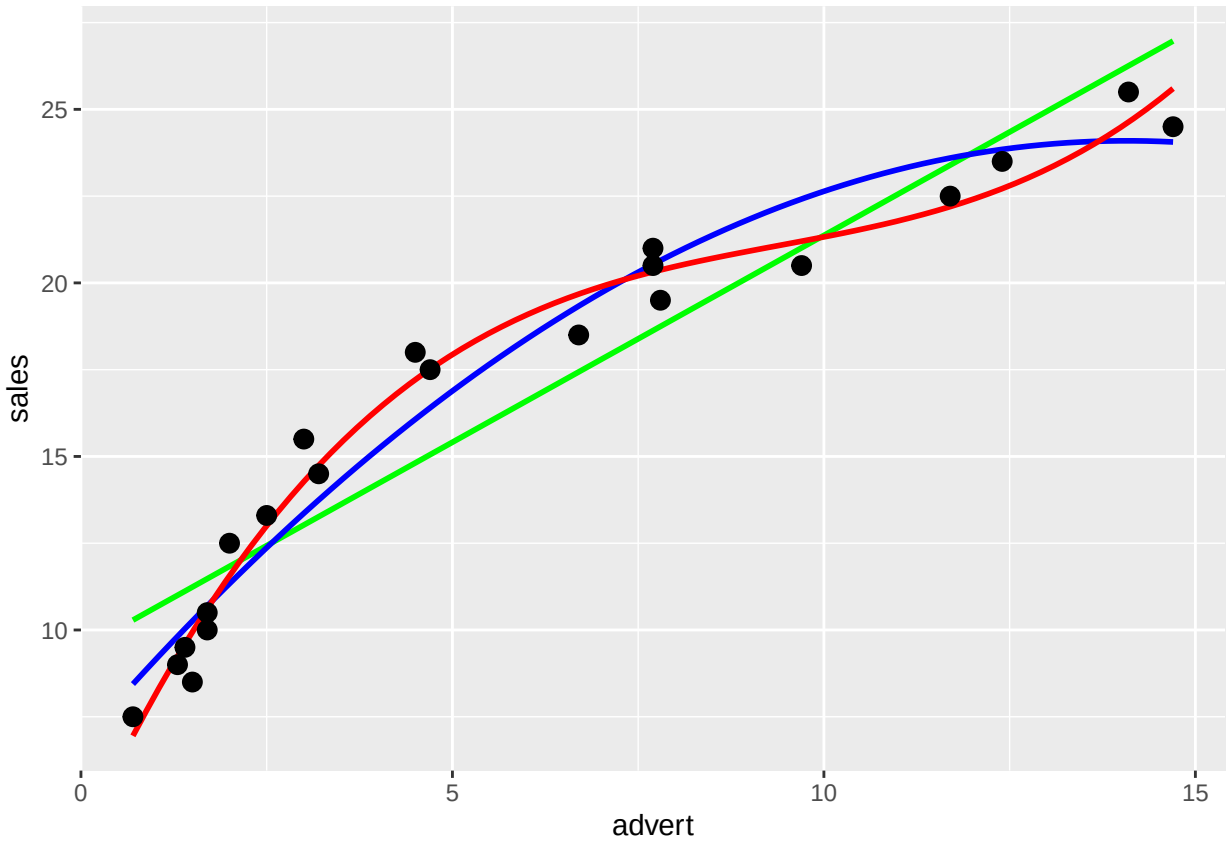
Now we see that with the first and second order terms in the model, the third order term is also significant. But does this make sense practically? The following plot should gives hints as to why it doesn't. (The model with the third order term doesn't have diminishing returns!)

```
plot(sales ~ advert, data = marketing,
     xlab = "Advert Spending (in $10,000)", ylab = "Sales (in $10,000)",
     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,

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_{p-1} x_i^{p-1} + \epsilon_i$$

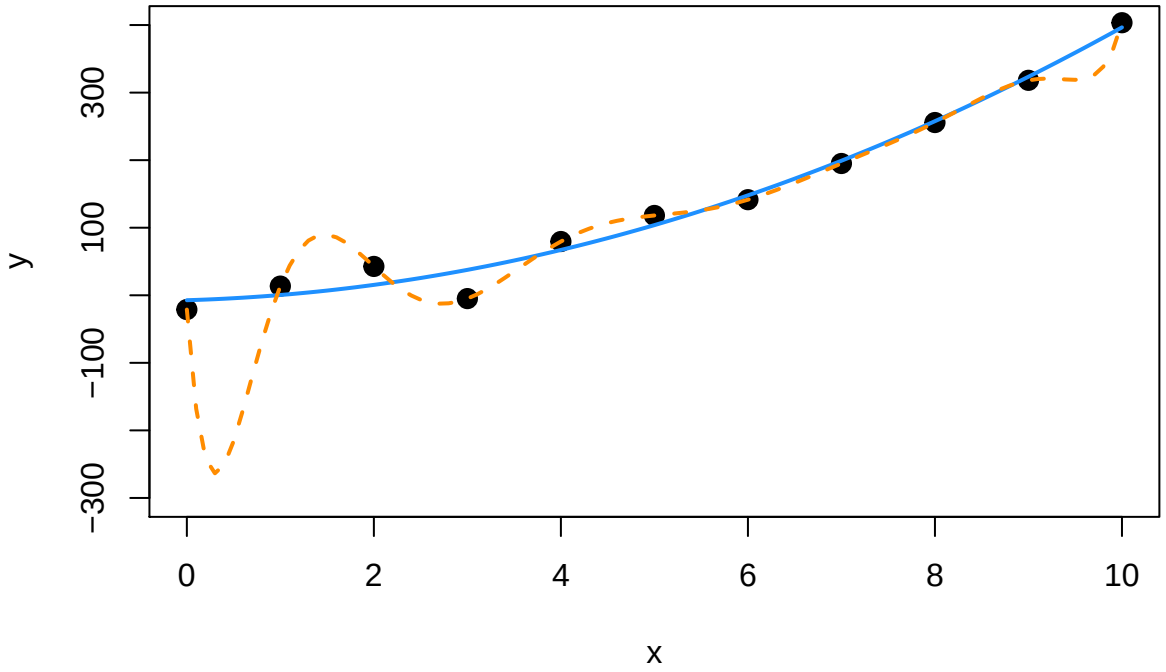
However, we should be careful about over-fitting, since with a polynomial of degree one less than the number of observations, it is sometimes possible to fit a model perfectly.

```
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) -2.114e+01         NA      NA       NA
```

```
## x           -1.918e+03         NA       NA       NA
## I(x^2)       4.969e+03         NA       NA       NA
## I(x^3)      -4.932e+03         NA       NA       NA
## I(x^4)       2.581e+03         NA       NA       NA
## I(x^5)      -8.035e+02         NA       NA       NA
## I(x^6)       1.570e+02         NA       NA       NA
## I(x^7)      -1.947e+01         NA       NA       NA
## I(x^8)       1.490e+00         NA       NA       NA
## I(x^9)      -6.424e-02         NA       NA       NA
## I(x^10)      1.195e-03         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
```

```
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("C:\\Users\\root\\Documents\\Rprojects\\Rmd-ex-paper-more-03Oct2021\\data\\fuel_econ.cs
```

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.

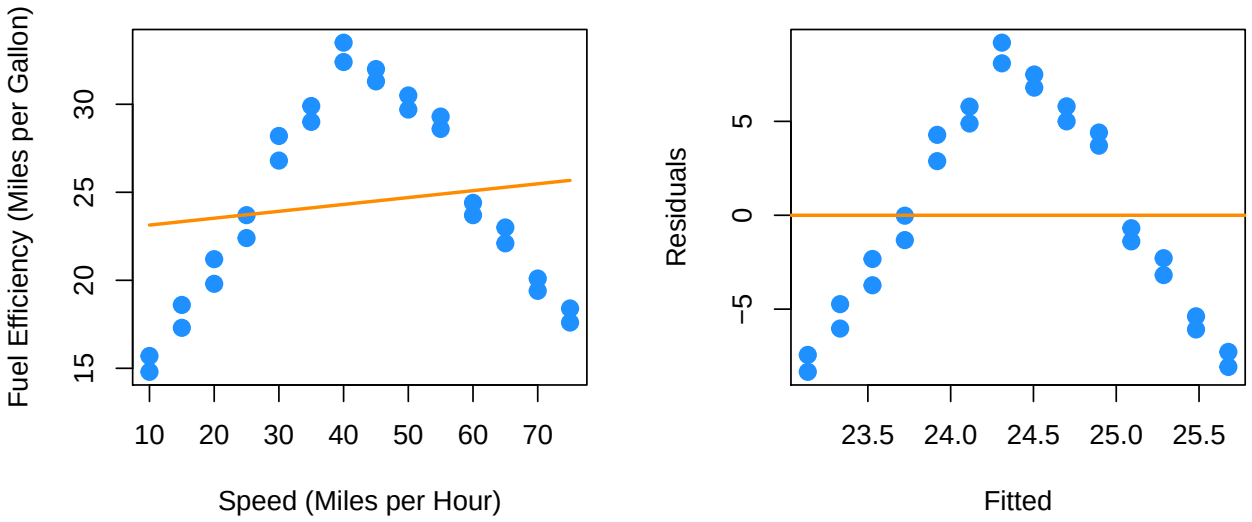
We will also be adding fitted curves to scatterplots repeatedly, so smartly we will write a function to do so.

```
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.

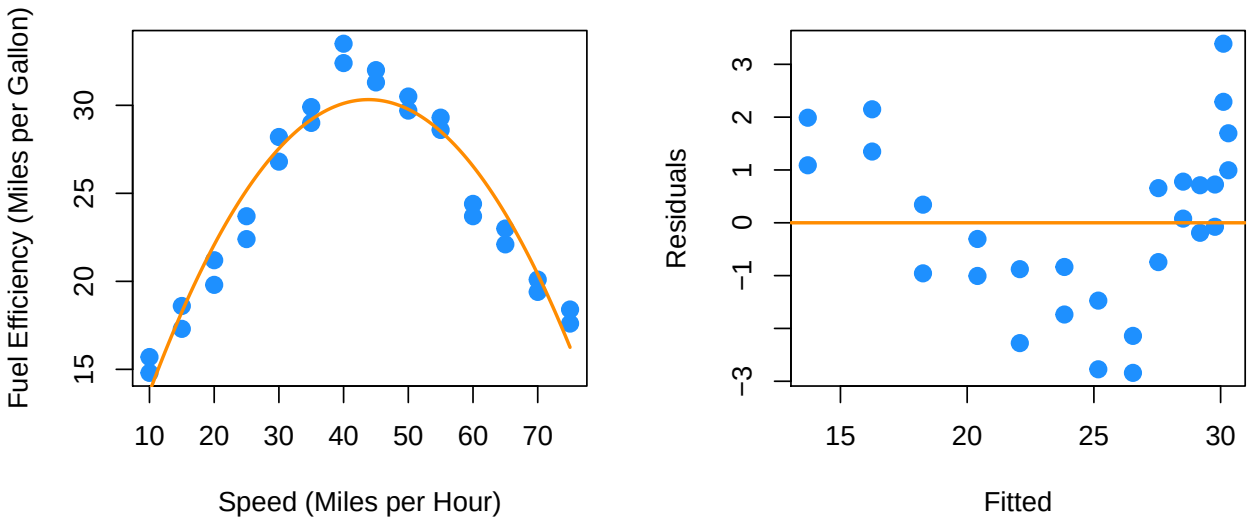
We will now add polynomial terms until we fit a suitable fit.

```
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
```

```
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)
```



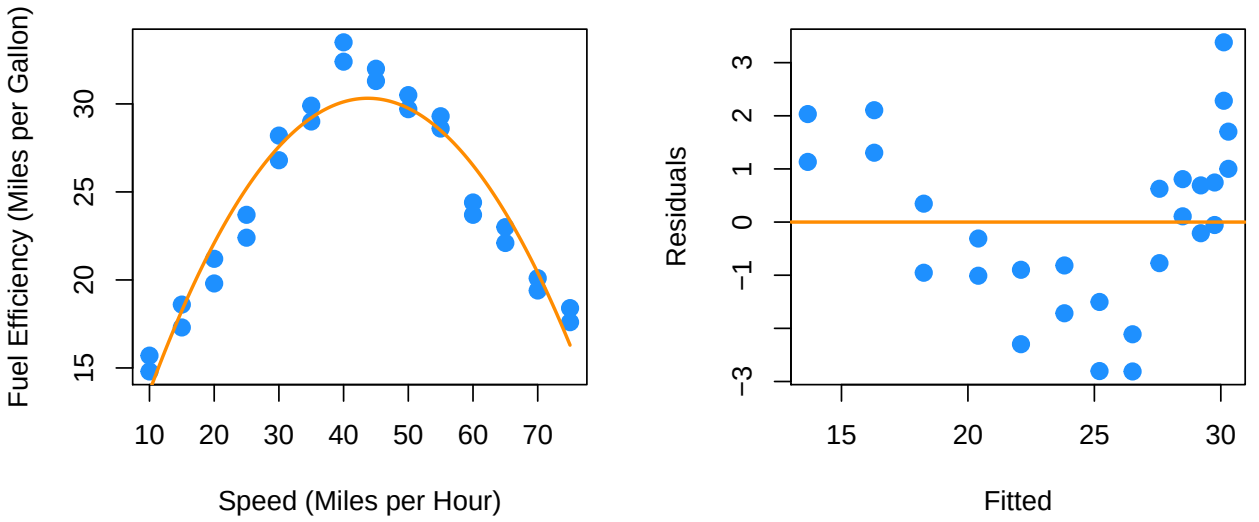
While this model clearly fits much better, and the second order term is significant, we still see a pattern in the fitted versus residuals plot which suggests higher order terms will help. Also, we would expect the curve to flatten as speed increases or decreases, not go sharply downward as we see here.

```
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)  2.258e+00  2.768e+00   0.816   0.4227
## mph          1.291e+00  2.529e-01   5.103  3.2e-05 ***
## I(mph^2)    -1.502e-02  6.604e-03  -2.274   0.0322 *
```

```
## I(mph^3)     4.066e-06  5.132e-05   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)
```



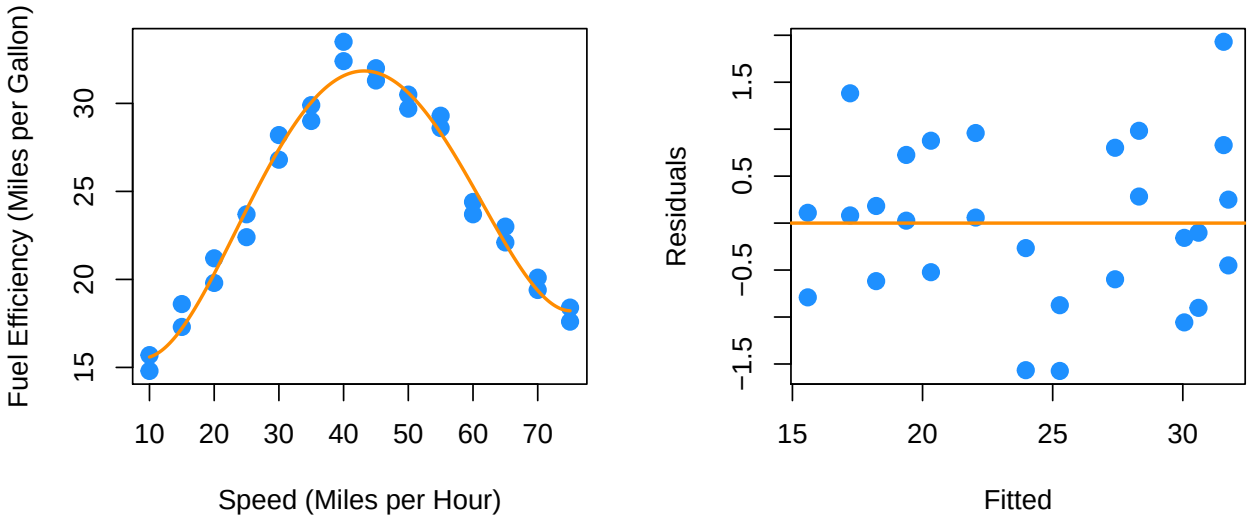
Adding the third order term doesn't seem to help at all. The fitted curve hardly changes. This makes sense, since what we would like is for the curve to flatten at the extremes. For this we will need an even degree polynomial term.

```
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)  2.146e+01  2.965e+00   7.238 2.28e-07 ***
## mph         -1.468e+00  3.913e-01  -3.751  0.00104 **
## I(mph^2)     1.081e-01  1.673e-02   6.463 1.35e-06 ***
## I(mph^3)    -2.130e-03  2.844e-04  -7.488 1.31e-07 ***
## I(mph^4)     1.255e-05  1.665e-06   7.539 1.17e-07 ***
## ---
## 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
```

```
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)
```



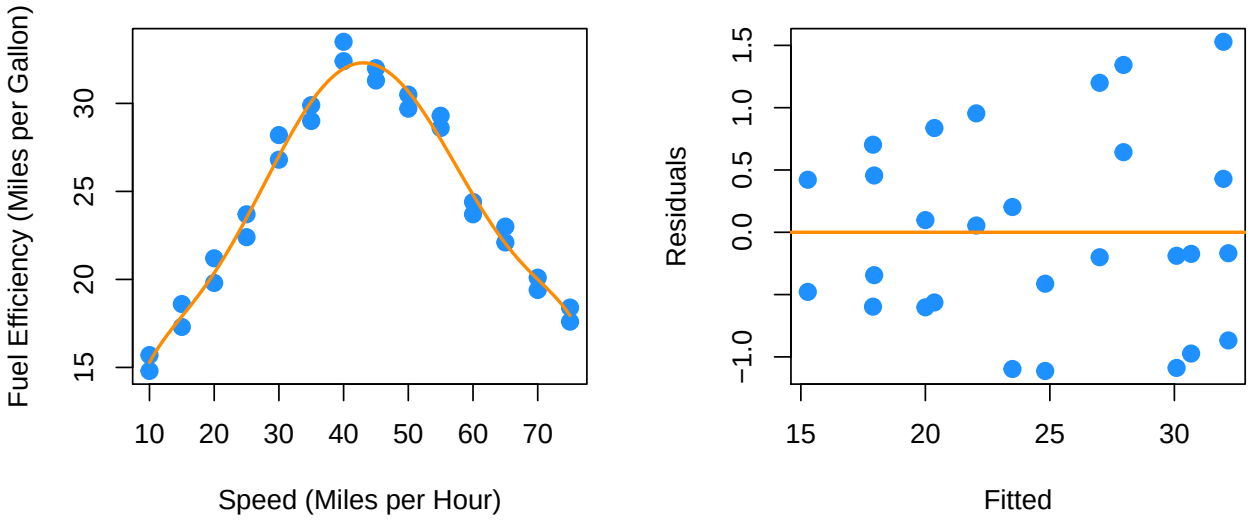
Now we are making progress. The fourth order term is significant with the other terms in the model. Also we are starting to see what we expected for low and high speed. However, there still seems to be a bit of a pattern in the residuals, so we will again try more higher order terms. We will add the fifth and sixth together, since adding the fifth will be similar to adding the third.

```
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
```

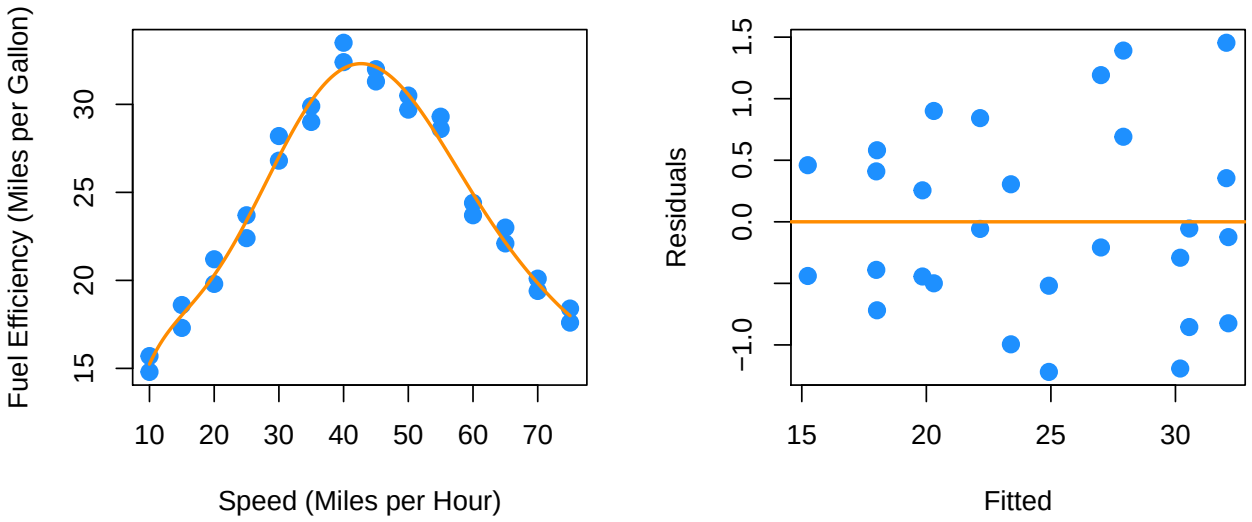
So, this test does not reject the null hypothesis at a level of significance of $\alpha = 0.05$, however the p-value is still rather small, and the fitted versus residuals plot is much better for the model with the sixth order term. This makes the sixth order model a good choice. We could repeat this process one more time.

```
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
```

```
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)
```




```
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
```

```
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.

```
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.

```
coef(fit6)
```

```
##   (Intercept)           mph      I(mph^2)      I(mph^3)      I(mph^4)
## -4.206224e+00  4.203382e+00 -3.521452e-01  1.579340e-02 -3.472665e-04
##      I(mph^5)      I(mph^6)
##  3.585201e-06 -1.401995e-08
```

```
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 polynomials*, which solves an issue we will discuss in the next chapter.

```
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 9.31e-05 ***
## 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 6.68e-08 ***
## 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
##             -4.206224e+00              4.203382e+00             -3.521452e-01
## poly(mph, 6, raw = TRUE)3 poly(mph, 6, raw = TRUE)4 poly(mph, 6, raw = TRUE)5
##              1.579340e-02             -3.472665e-04              3.585201e-06
## poly(mph, 6, raw = TRUE)6
##             -1.401995e-08
```

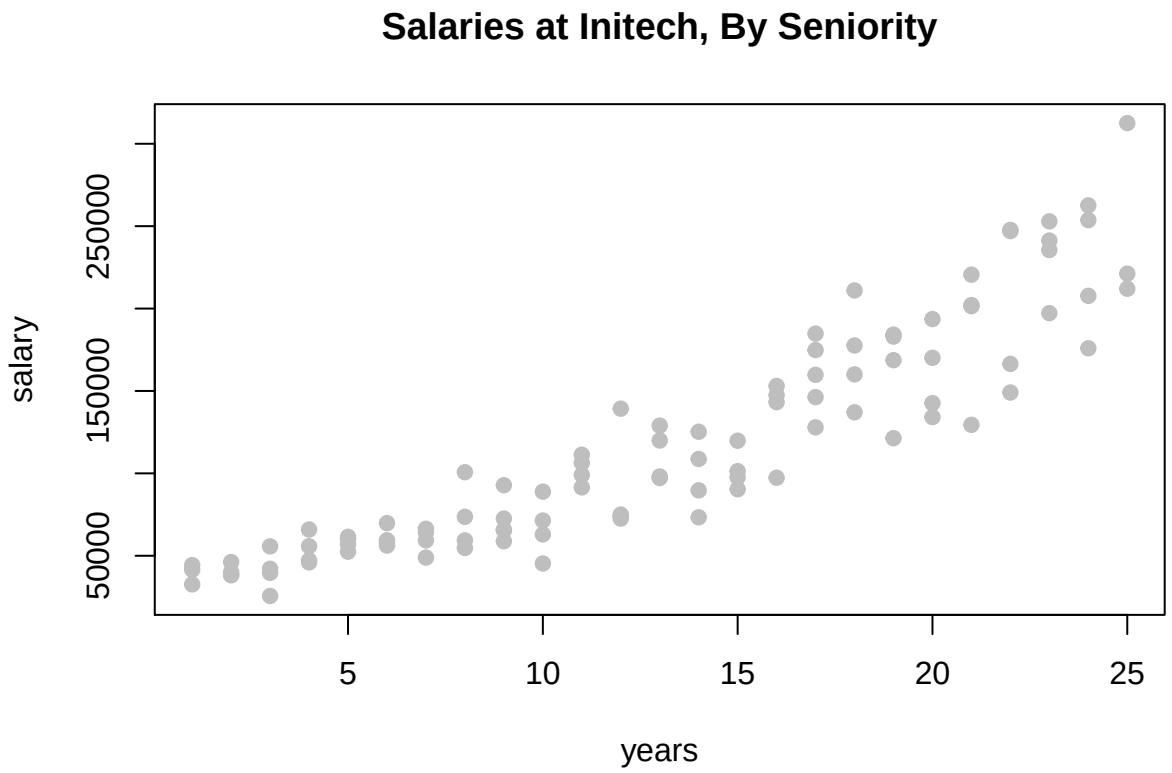
We've now seen how to transform predictor and response variables. In this chapter we have mostly focused on using this in the context of fixing SLR models. However, these concepts can easily be used together with categorical variables and interactions to build larger, more flexible models. In the next chapter, we will discuss how to choose a good model from a collection of possible models.

Material below here is currently being merged into the content above.

Response Transformations

```
initech = read.csv("C:\\Users\\root\\Documents\\Rprojects\\Rmd-ex-paper-more-03Oct2021\\data\\initech.cs
```

```
plot(salary ~ years, data = initech, col = "grey", pch = 20, cex = 1.5,
     main = "Salaries at Initech, By Seniority")
```



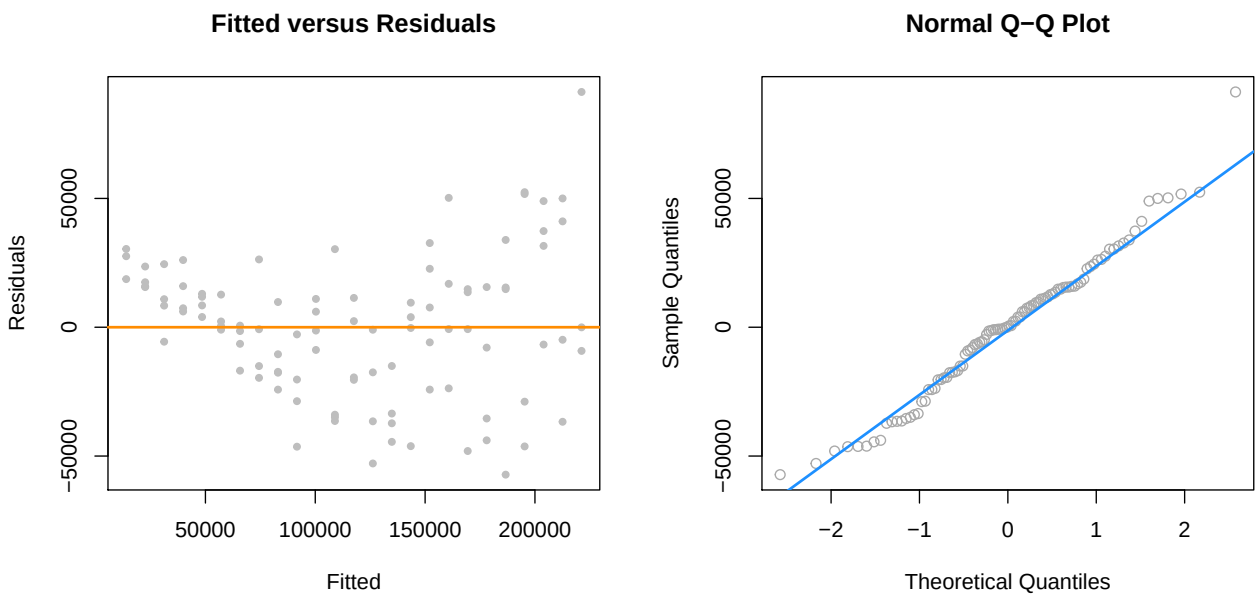

```
initech_fit = lm(salary ~ years, data = initech)
summary(initech_fit)
```

```
##
## Call:
## lm(formula = salary ~ years, data = initech)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -57225 -18104    241  15589  91332
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)     5302       5750   0.922    0.359
## years           8637        389  22.200   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 27360 on 98 degrees of freedom
## Multiple R-squared:  0.8341, Adjusted R-squared:  0.8324
## F-statistic: 492.8 on 1 and 98 DF,  p-value: < 2.2e-16
```

```r
plot(salary ~ years, data = initech, col = "grey", pch = 20, cex = 1.5,
     main = "Salaries at Initech, By Seniority")
abline(initech_fit, col = "darkorange", lwd = 2)
```


```r
par(mfrow = c(1, 2))

plot(fitted(initech_fit), resid(initech_fit), col = "grey", pch = 20,
     xlab = "Fitted", ylab = "Residuals", main = "Fitted versus Residuals")
abline(h = 0, col = "darkorange", lwd = 2)

qqnorm(resid(initech_fit), main = "Normal Q-Q Plot", col = "darkgrey")
qqline(resid(initech_fit), col = "dodgerblue", lwd = 2)
```




```
initech_fit_log = lm(log(salary) ~ years, data = initech)
```

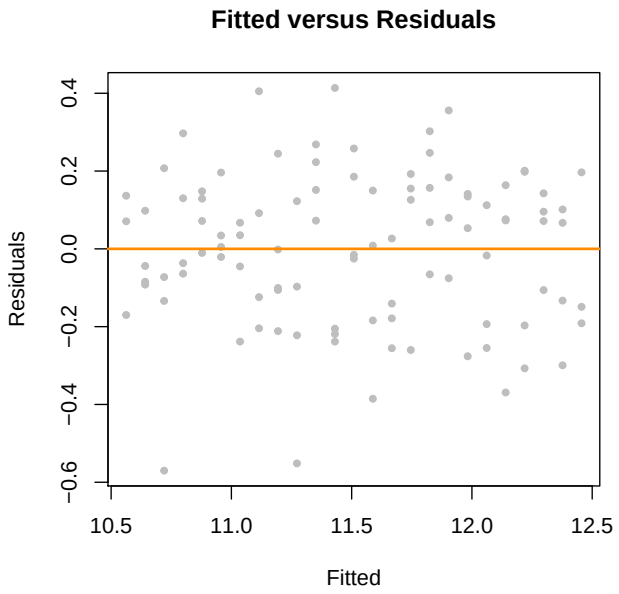
$$\log(Y_i) = \beta_0 + \beta_1 x_i + \epsilon_i$$

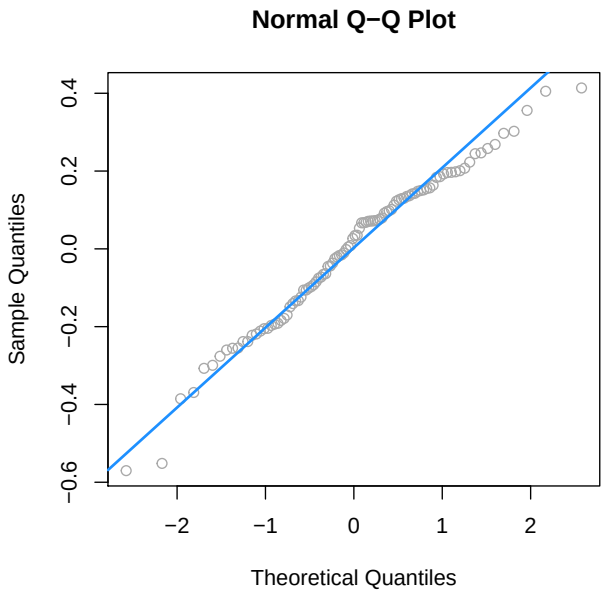
```
plot(log(salary) ~ years, data = initech, col = "grey", pch = 20, cex = 1.5,
     main = "Salaries at Initech, By Seniority")
abline(initech_fit_log, col = "darkorange", lwd = 2)
```

$$Y_i = \exp(\beta_0 + \beta_1 x_i) \cdot \exp(\epsilon_i)$$

```
plot(salary ~ years, data = initech, col = "grey", pch = 20, cex = 1.5,
     main = "Salaries at Initech, By Seniority")
curve(exp(initech_fit_log$coef[1] + initech_fit_log$coef[2] * x),
      from = 0, to = 30, add = TRUE, col = "darkorange", lwd = 2)
```


```
par(mfrow = c(1, 2))

plot(fitted(initech_fit_log), resid(initech_fit_log), col = "grey", pch = 20,
     xlab = "Fitted", ylab = "Residuals", main = "Fitted versus Residuals")
abline(h = 0, col = "darkorange", lwd = 2)

qqnorm(resid(initech_fit_log), main = "Normal Q-Q Plot", col = "darkgrey")
qqline(resid(initech_fit_log), col = "dodgerblue", lwd = 2)
```






```
sqrt(mean(resid(initech_fit) ^ 2))
```

```
## [1] 27080.16
```

```
sqrt(mean(resid(initech_fit_log) ^ 2))
```

```
## [1] 0.1934907
```

```
sqrt(mean((initech$salary - fitted(initech_fit)) ^ 2))
```

```
## [1] 27080.16
```

```
sqrt(mean((initech$salary - exp(fitted(initech_fit_log))) ^ 2))
```

```
## [1] 24280.36
```

Predictor Transformations

A Quadratic Model

```
sim_quad = function(sample_size = 500) {
  x = runif(n = sample_size) * 5
  y = 3 + 5 * x ^ 2 + rnorm(n = sample_size, mean = 0, sd = 5)
  data.frame(x, y)
}
```

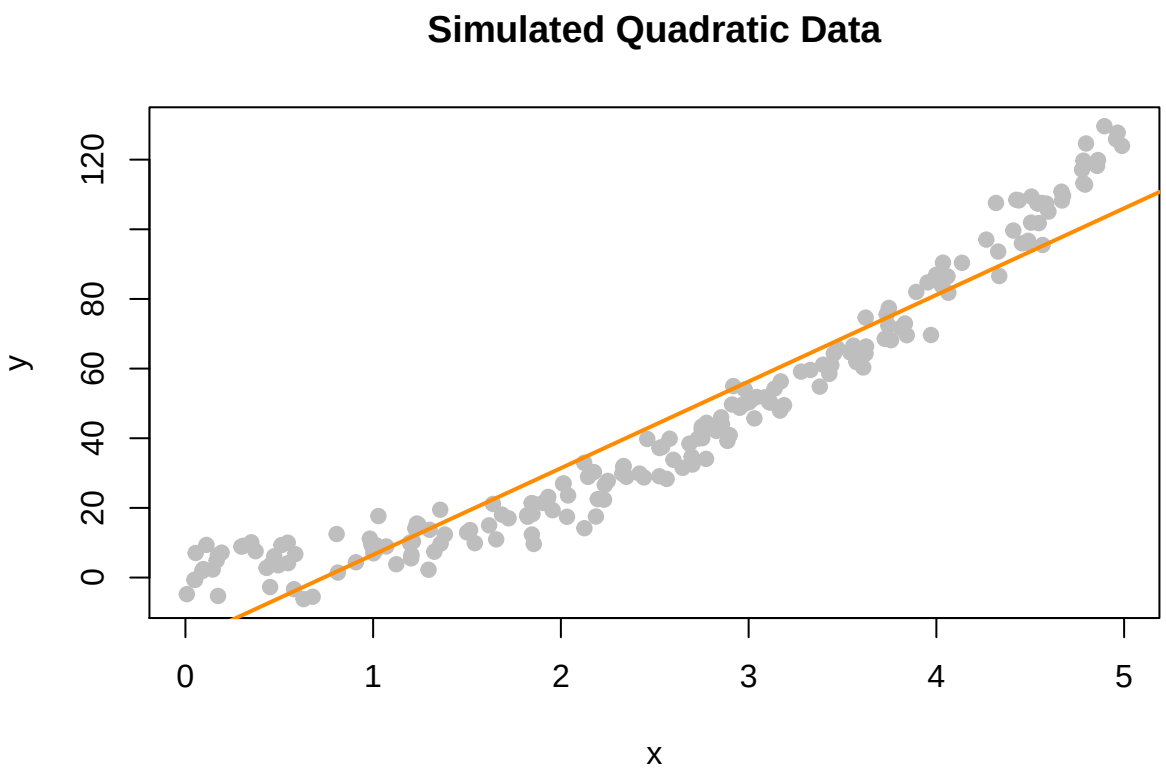
```
set.seed(314)
quad_data = sim_quad(sample_size = 200)
```

```
lin_fit = lm(y ~ x, data = quad_data)
summary(lin_fit)
```

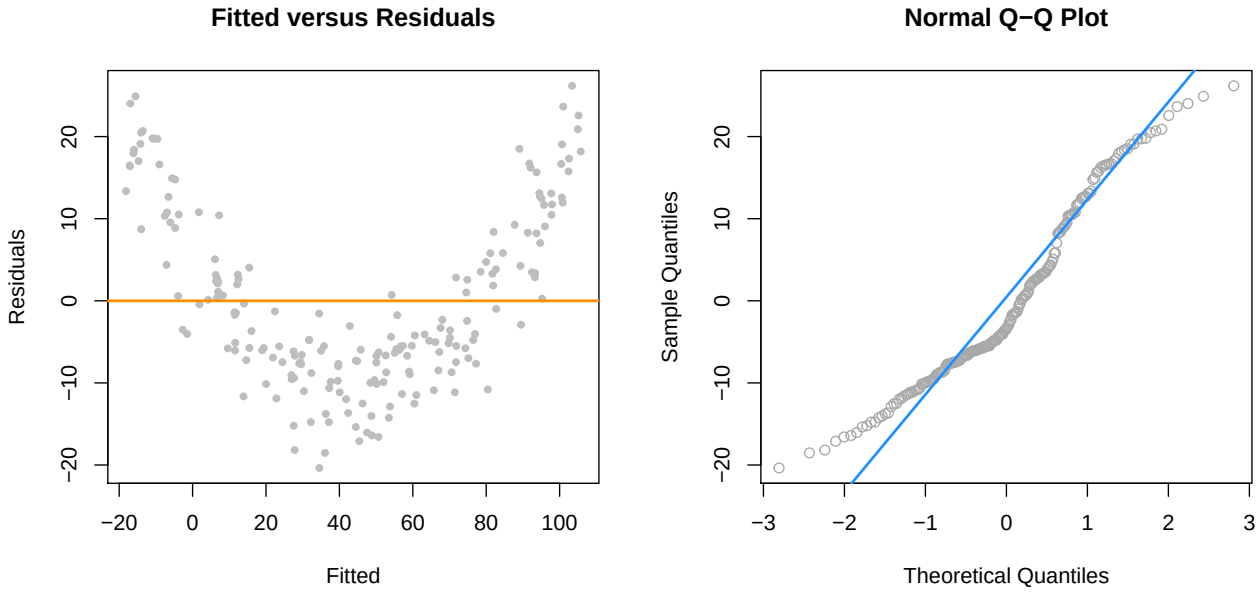
```
##
## Call:
## lm(formula = y ~ x, data = quad_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -20.363  -7.550  -3.416   8.472  26.181
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -18.3271     1.5494  -11.83   <2e-16 ***
## x            24.8716     0.5343   46.55   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.79 on 198 degrees of freedom
## Multiple R-squared:  0.9163, Adjusted R-squared:  0.9158
## F-statistic:  2167 on 1 and 198 DF,  p-value: < 2.2e-16
```

```r
plot(y ~ x, data = quad_data, col = "grey", pch = 20, cex = 1.5,
     main = "Simulated Quadratic Data")
abline(lin_fit, col = "darkorange", lwd = 2)
```




```r
par(mfrow = c(1, 2))

plot(fitted(lin_fit), resid(lin_fit), col = "grey", pch = 20,
     xlab = "Fitted", ylab = "Residuals", main = "Fitted versus Residuals")
abline(h = 0, col = "darkorange", lwd = 2)

qqnorm(resid(lin_fit), main = "Normal Q-Q Plot", col = "darkgrey")
qqline(resid(lin_fit), col = "dodgerblue", lwd = 2)
```

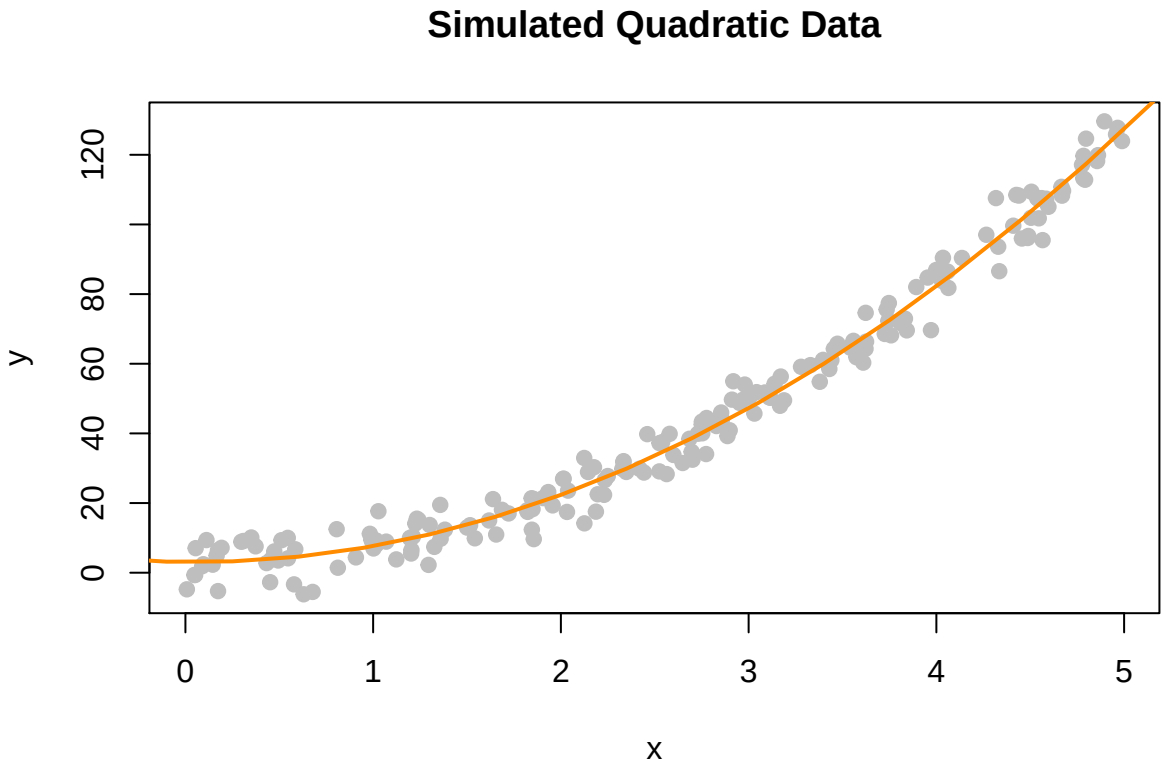


$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$$

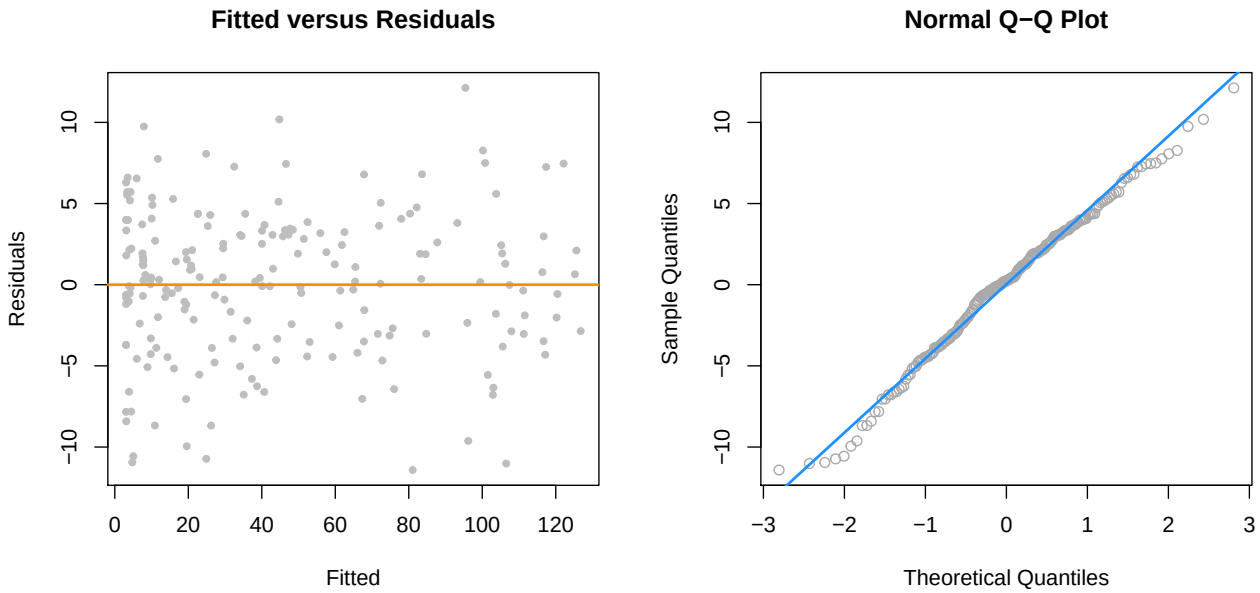
```
quad_fit = lm(y ~ x + I(x^2), data = quad_data)
summary(quad_fit)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2), data = quad_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.4167  -3.0581   0.2297   3.1024  12.1256
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.0649     0.9577   3.200   0.0016 **
## x            -0.5108     0.8637  -0.591   0.5549
## I(x^2)        5.0740     0.1667  30.433   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.531 on 197 degrees of freedom
## Multiple R-squared:  0.9853, Adjusted R-squared:  0.9852
## F-statistic:  6608 on 2 and 197 DF,  p-value: < 2.2e-16
```

```
plot(y ~ x, data = quad_data, col = "grey", pch = 20, cex = 1.5,
     main = "Simulated Quadratic Data")
curve(quad_fit$coef[1] + quad_fit$coef[2] * x + quad_fit$coef[3] * x ^ 2,
      from = -5, to = 30, add = TRUE, col = "darkorange", lwd = 2)
```




```
par(mfrow = c(1, 2))

plot(fitted(quad_fit), resid(quad_fit), col = "grey", pch = 20,
     xlab = "Fitted", ylab = "Residuals", main = "Fitted versus Residuals")
abline(h = 0, col = "darkorange", lwd = 2)

qqnorm(resid(quad_fit), main = "Normal Q-Q Plot", col = "darkgrey")
qqline(resid(quad_fit), col = "dodgerblue", lwd = 2)
```



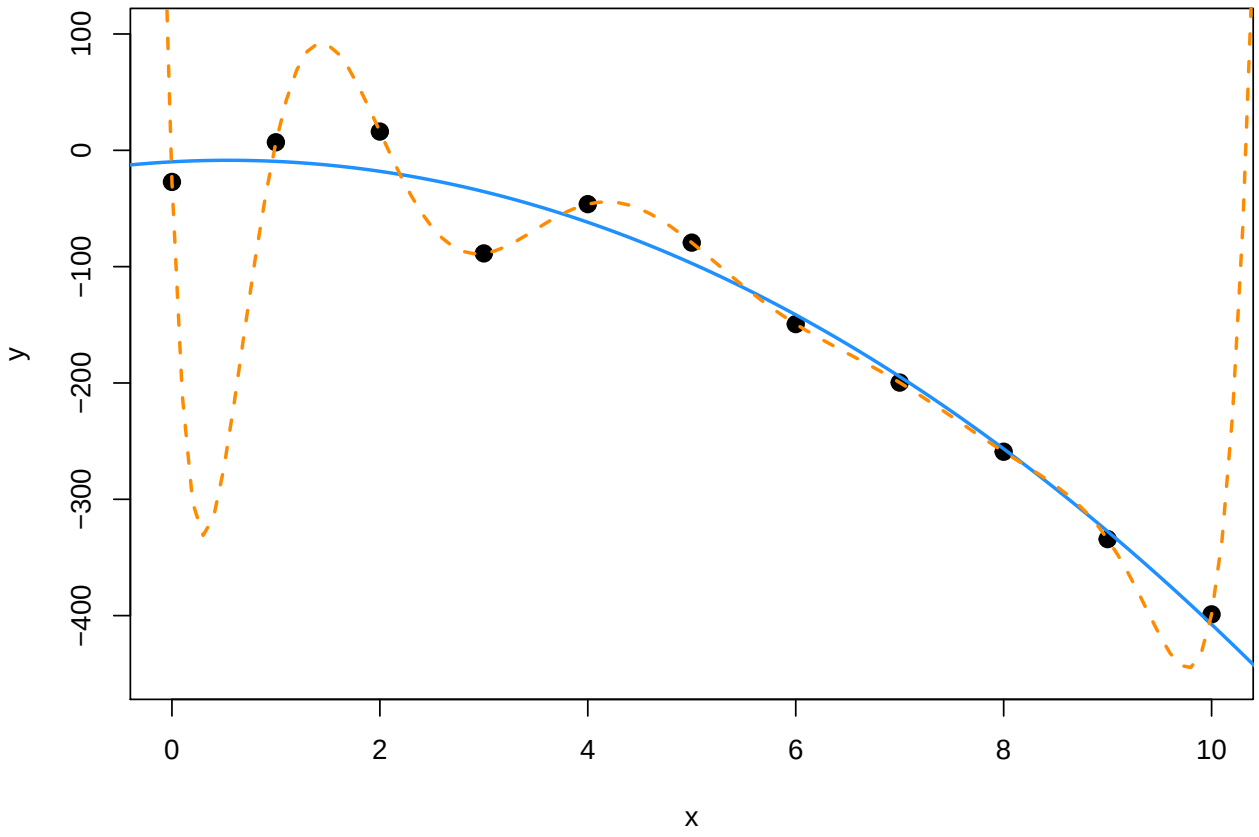
Overfitting and Extrapolation

```
sim_for_perf = function() {
  x = seq(0, 10)
  y = 3 + x - 4 * x ^ 2 + rnorm(n = 11, mean = 0, sd = 25)
  data.frame(x, y)
}
```

```
set.seed(1234)
data_for_perf = sim_for_perf()
```

```
fit_correct = lm(y ~ x + I(x ^ 2), data = data_for_perf)
fit_perfect = 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),
                 data = data_for_perf)
```

```
x_plot = seq(-5, 15, by = 0.1)
plot(y ~ x, data = data_for_perf, ylim = c(-450, 100), cex = 2, pch = 20)
lines(x_plot, predict(fit_correct, newdata = data.frame(x = x_plot)),
      col = "dodgerblue", lwd = 2, lty = 1)
lines(x_plot, predict(fit_perfect, newdata = data.frame(x = x_plot)),
      col = "darkorange", lwd = 2, lty = 2)
```



Comparing Polynomial Models

```
sim_higher = function(sample_size = 250) {
  x = runif(n = sample_size, min = -1, max = 1) * 2
  y = 3 + -6 * x ^ 2 + 1 * x ^ 4 + rnorm(n = sample_size, mean = 0, sd = 3)
```

```
  data.frame(x, y)
}
```

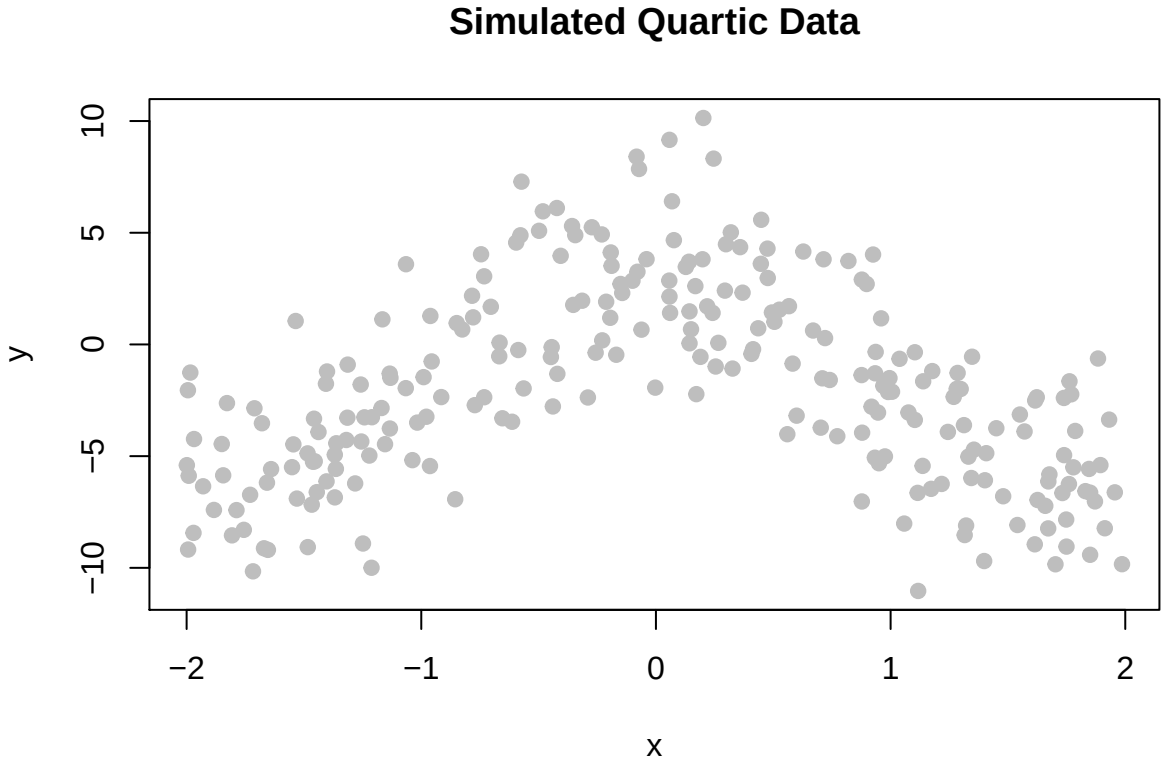
$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$$

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 x_i^4 + \epsilon_i$$

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 x_i^4 + \beta_5 x_i^5 + \beta_6 x_i^6 + \epsilon_i$$

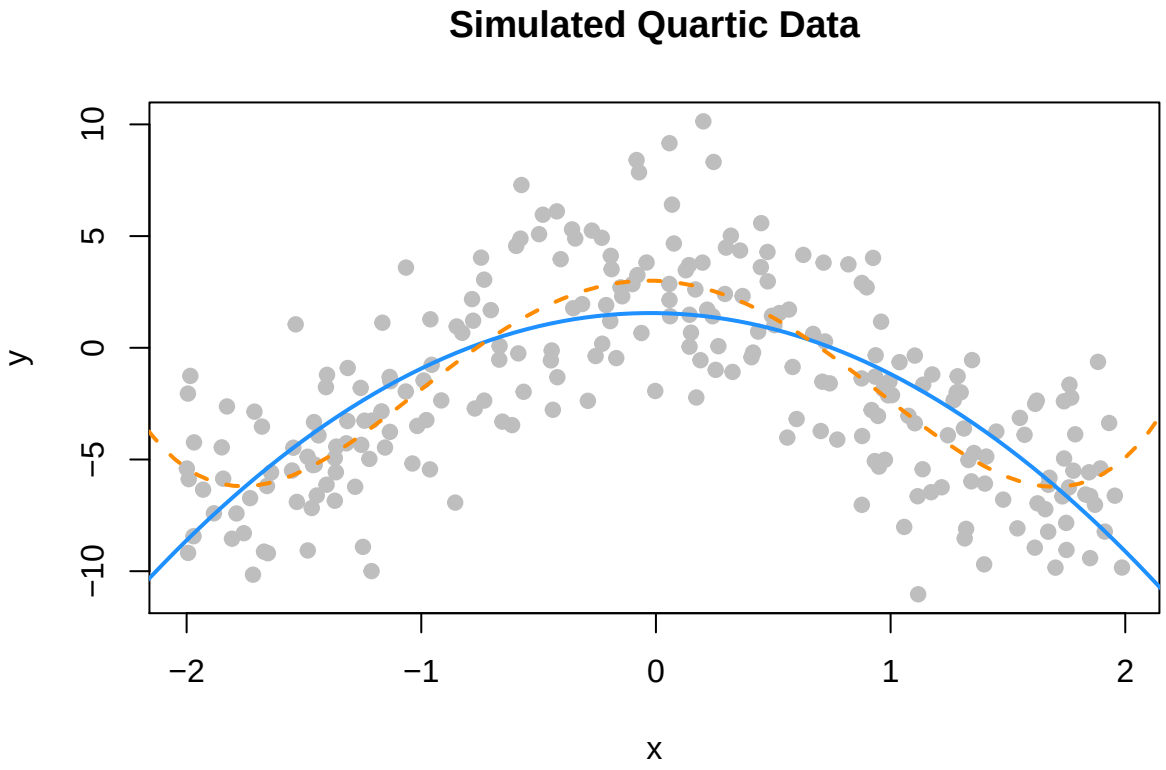
```
set.seed(42)
data_higher = sim_higher()
```

```
plot(y ~ x, data = data_higher, col = "grey", pch = 20, cex = 1.5,
     main = "Simulated Quartic Data")
```

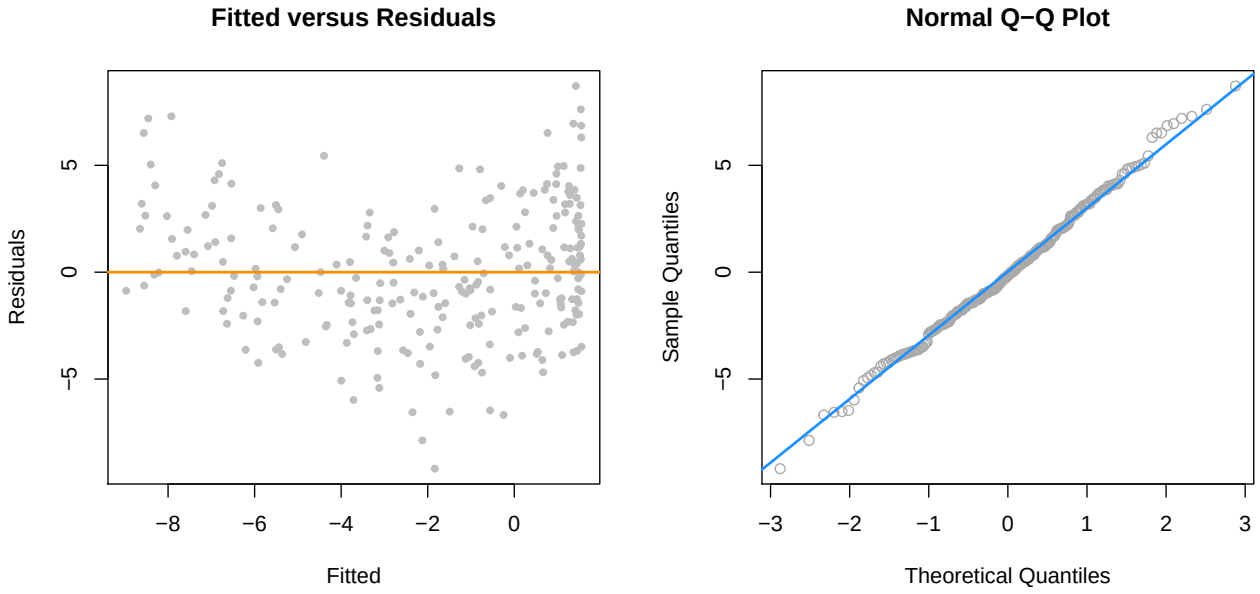



```
fit_2 = lm(y ~ poly(x, 2), data = data_higher)
fit_4 = lm(y ~ poly(x, 4), data = data_higher)
```

```
plot(y ~ x, data = data_higher, col = "grey", pch = 20, cex = 1.5,
     main = "Simulated Quartic Data")
x_plot = seq(-5, 5, by = 0.05)
lines(x_plot, predict(fit_2, newdata = data.frame(x = x_plot)),
      col = "dodgerblue", lwd = 2, lty = 1)
lines(x_plot, predict(fit_4, newdata = data.frame(x = x_plot)),
      col = "darkorange", lwd = 2, lty = 2)
```

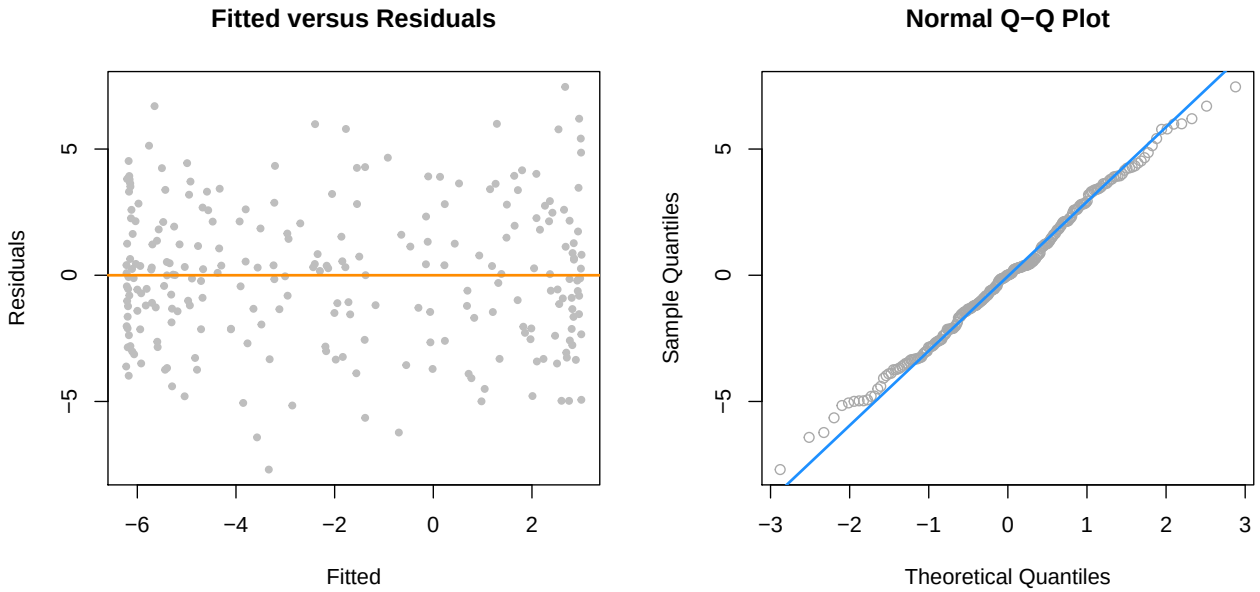



```
par(mfrow = c(1, 2))

plot(fitted(fit_2), resid(fit_2), col = "grey", pch = 20,
     xlab = "Fitted", ylab = "Residuals", main = "Fitted versus Residuals")
abline(h = 0, col = "darkorange", lwd = 2)

qqnorm(resid(fit_2), main = "Normal Q-Q Plot", col = "darkgrey")
qqline(resid(fit_2), col = "dodgerblue", lwd = 2)
```




```
par(mfrow = c(1, 2))
```

```
plot(fitted(fit_4), resid(fit_4), col = "grey", pch = 20,
     xlab = "Fitted", ylab = "Residuals", main = "Fitted versus Residuals")
abline(h = 0, col = "darkorange", lwd = 2)

qqnorm(resid(fit_4), main = "Normal Q-Q Plot", col = "darkgrey")
qqline(resid(fit_4), col = "dodgerblue", lwd = 2)
```




```
anova(fit_2, fit_4)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ poly(x, 2)
## Model 2: y ~ poly(x, 4)
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    247 2334.1
## 2    245 1912.6  2    421.51 26.997 2.536e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
fit_6 = lm(y ~ poly(x, 6), data = data_higher)
```

```
anova(fit_4, fit_6)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ poly(x, 4)
## Model 2: y ~ poly(x, 6)
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    245 1912.6
## 2    243 1904.4  2    8.1889 0.5224 0.5937
```

`poly()` Function and Orthogonal Polynomials

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 x_i^4 + \epsilon_i$$

```
fit_4a = lm(y ~ poly(x, degree = 4), data = data_higher)
fit_4b = lm(y ~ poly(x, degree = 4, raw = TRUE), data = data_higher)
fit_4c = lm(y ~ x + I(x^2) + I(x^3) + I(x^4), data = data_higher)
```

```
coef(fit_4a)
```

```
##          (Intercept) poly(x, degree = 4)1 poly(x, degree = 4)2
##            -1.980036           -2.053929          -49.344752
## poly(x, degree = 4)3 poly(x, degree = 4)4
##             0.669874           20.519759
```

```
coef(fit_4b)
```

```
##                      (Intercept) poly(x, degree = 4, raw = TRUE)1
##                        2.9996256                       -0.3880250
## poly(x, degree = 4, raw = TRUE)2 poly(x, degree = 4, raw = TRUE)3
##                       -6.1511166                        0.1269046
## poly(x, degree = 4, raw = TRUE)4
##                        1.0282139
```

```
coef(fit_4c)
```

```
## (Intercept)           x      I(x^2)      I(x^3)      I(x^4)
##   2.9996256  -0.3880250  -6.1511166   0.1269046   1.0282139
```

```
unname(coef(fit_4a))
```

```
## [1]  -1.980036  -2.053929 -49.344752   0.669874  20.519759
```

```
unname(coef(fit_4b))
```

```
## [1]  2.9996256 -0.3880250 -6.1511166  0.1269046  1.0282139
```

```
unname(coef(fit_4c))
```

```
## [1]  2.9996256 -0.3880250 -6.1511166  0.1269046  1.0282139
```

```
all.equal(fitted(fit_4a),
          fitted(fit_4b))
```

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

```
all.equal(resid(fit_4a),
          resid(fit_4b))
```

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

```
summary(fit_4a)
```

```
##
## Call:
## lm(formula = y ~ poly(x, degree = 4), data = data_higher)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6982 -2.0334  0.0042  1.9532  7.4626
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -1.9800     0.1767 -11.205  < 2e-16 ***
```

```
## poly(x, degree = 4)1  -2.0539     2.7940  -0.735    0.463
## poly(x, degree = 4)2 -49.3448     2.7940 -17.661  < 2e-16 ***
## poly(x, degree = 4)3   0.6699     2.7940   0.240    0.811
## poly(x, degree = 4)4  20.5198     2.7940   7.344 3.06e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.794 on 245 degrees of freedom
## Multiple R-squared:  0.5993, Adjusted R-squared:  0.5928
## F-statistic: 91.61 on 4 and 245 DF,  p-value: < 2.2e-16
```

```
summary(fit_4c)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4), data = data_higher)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6982 -2.0334  0.0042  1.9532  7.4626
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.9996     0.3315   9.048  < 2e-16 ***
## x            -0.3880     0.3828  -1.014    0.312
## I(x^2)       -6.1511     0.5049 -12.183  < 2e-16 ***
## I(x^3)        0.1269     0.1456   0.871    0.384
## I(x^4)        1.0282     0.1400   7.344 3.06e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.794 on 245 degrees of freedom
## Multiple R-squared:  0.5993, Adjusted R-squared:  0.5928
## F-statistic: 91.61 on 4 and 245 DF,  p-value: < 2.2e-16
```

Inhibit Function

```
coef(lm(y ~ x + x ^ 2, data = quad_data))
```

```
## (Intercept)           x
##   -18.32715    24.87163
```

```
coef(lm(y ~ x + I(x ^ 2), data = quad_data))
```

```
## (Intercept)           x      I(x^2)
##   3.0649446  -0.5108131   5.0739805
```

```
coef(lm(y ~ x + x:x, data = quad_data))
```

```
## (Intercept)           x
##   -18.32715    24.87163
```

```
coef(lm(y ~ x * x, data = quad_data))
```

```
## (Intercept)           x
##   -18.32715    24.87163
```

```
coef(lm(y ~ x ^ 2, data = quad_data))
```

```
## (Intercept)           x
##   -18.32715    24.87163
```

```
coef(lm(y ~ x + x ^ 2, data = quad_data))
```

```
## (Intercept)           x
##   -18.32715    24.87163
```

```
coef(lm(y ~ I(x + x), data = quad_data))
```

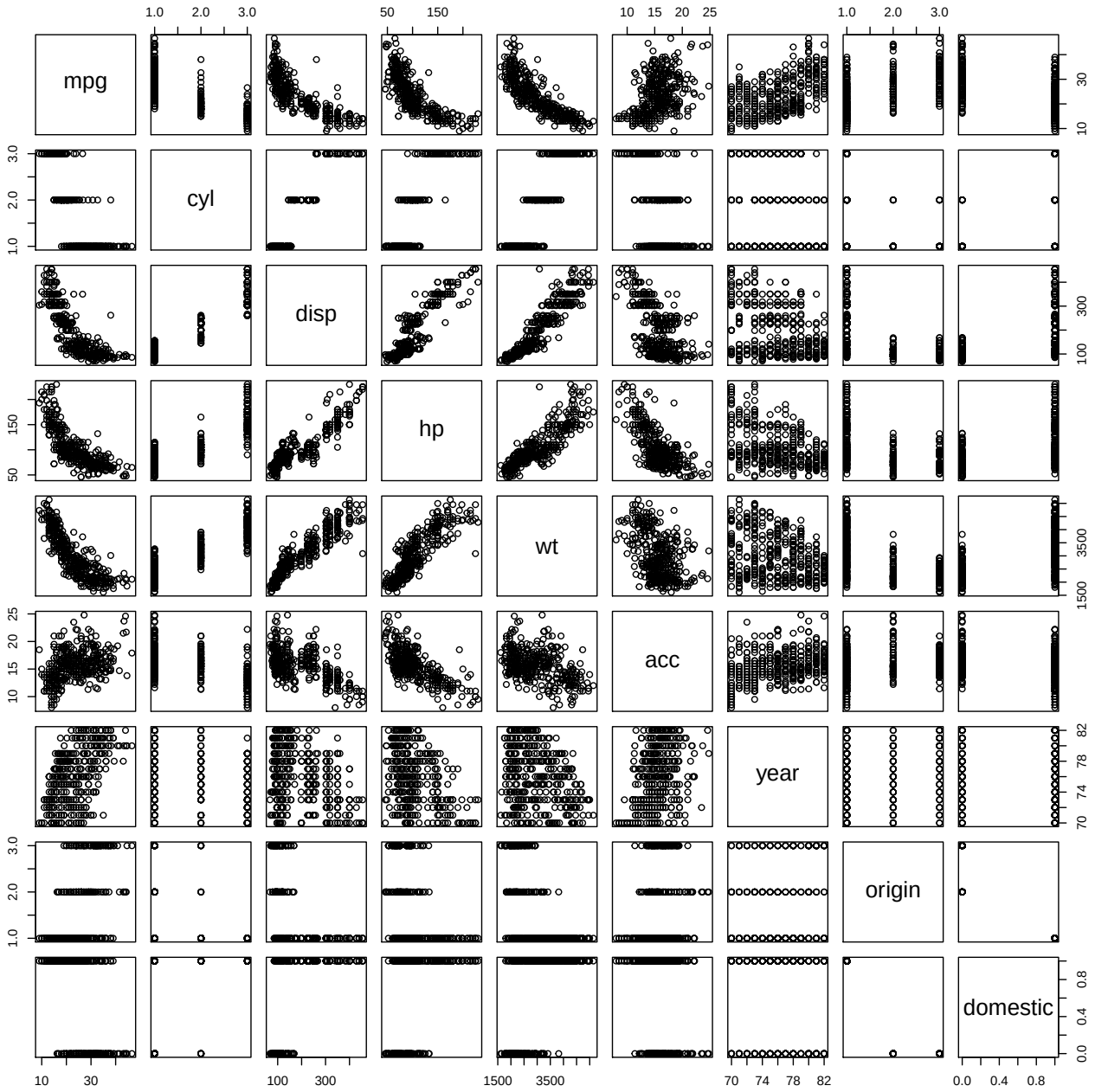
```
## (Intercept)    I(x + x)
##   -18.32715    12.43582
```

```
coef(lm(y ~ x + x, data = quad_data))
```

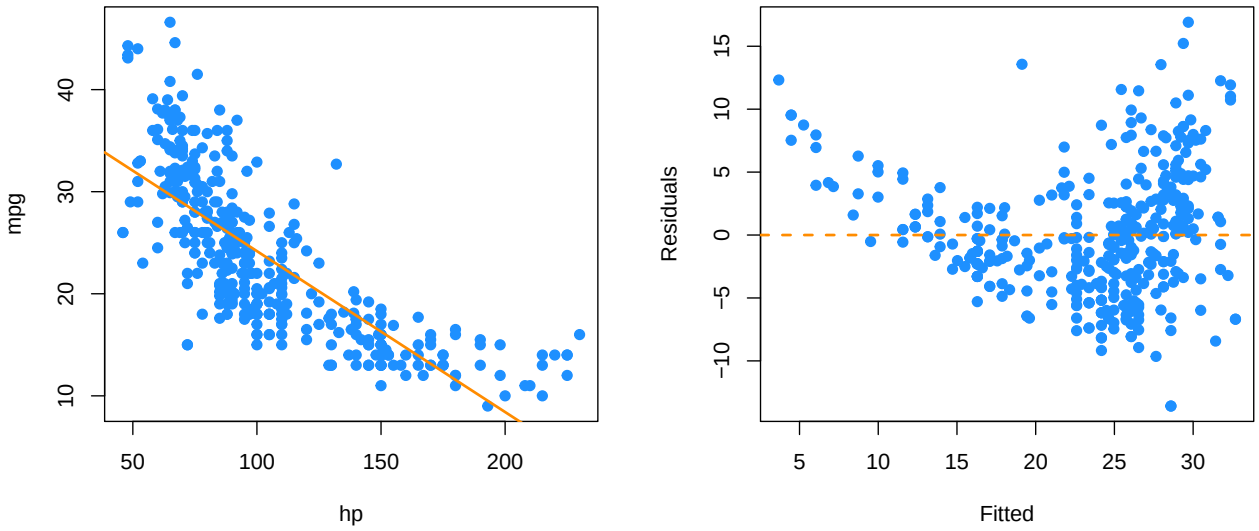
```
## (Intercept)           x
##   -18.32715    24.87163
```

Data Example

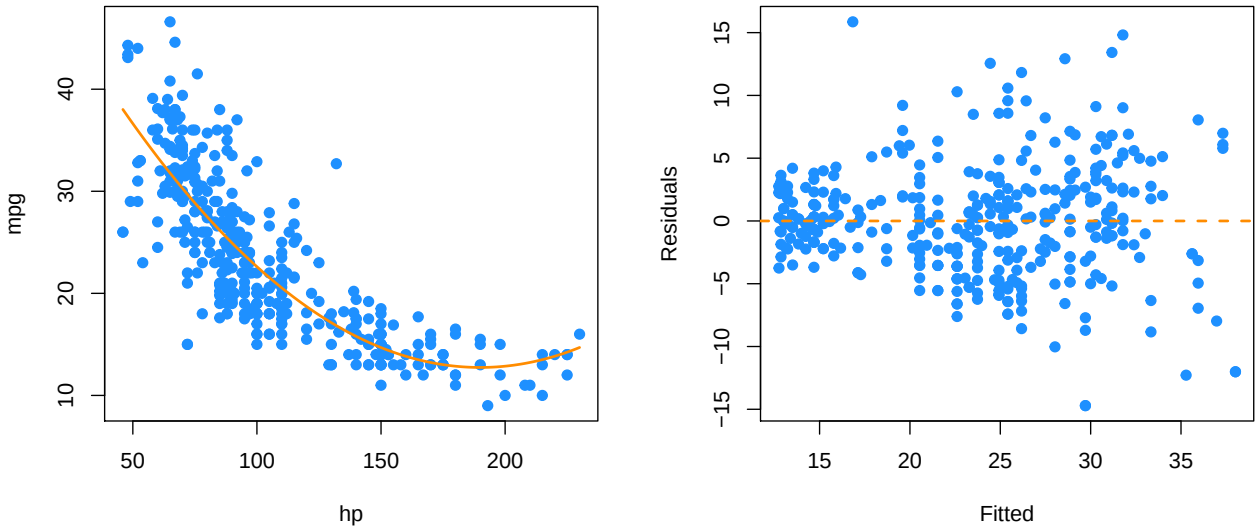
```
pairs(autompg)
```




```
mpg_hp = lm(mpg ~ hp, data = autompg)

par(mfrow = c(1, 2))

plot(mpg ~ hp, data = autompg, col = "dodgerblue", pch = 20, cex = 1.5)
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)
```

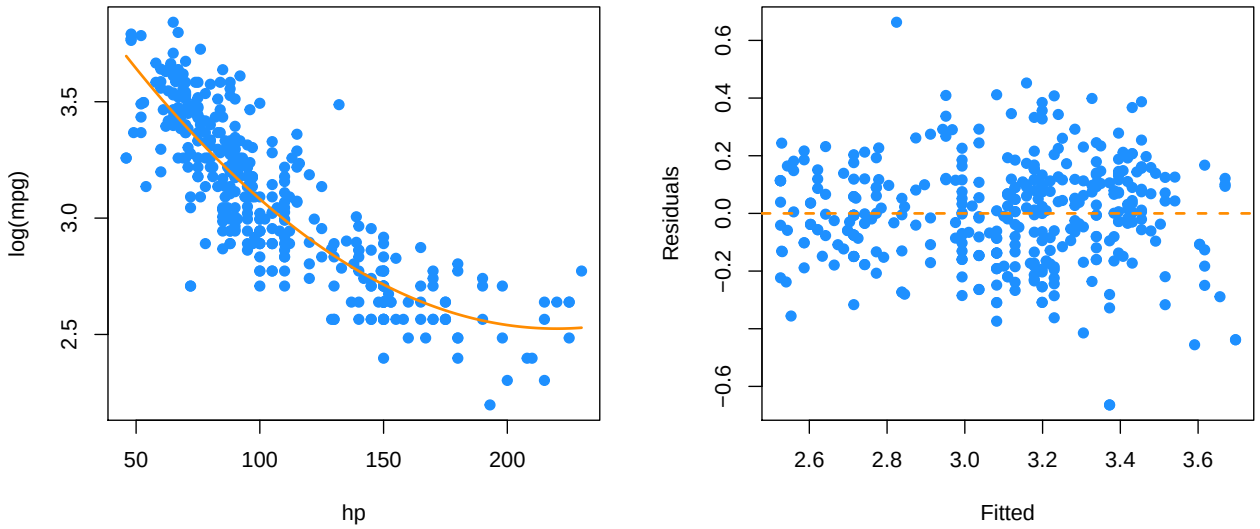



```
mpg_hp_log = lm(mpg ~ hp + I(hp ^ 2), data = autompg)

par(mfrow = c(1, 2))

plot(mpg ~ hp, data = autompg, col = "dodgerblue", pch = 20, cex = 1.5)
xplot = seq(min(autompg$hp), max(autompg$hp), by = 0.1)
lines(xplot, predict(mpg_hp_log, newdata = data.frame(hp = xplot)),
      col = "darkorange", lwd = 2, lty = 1)

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)
```

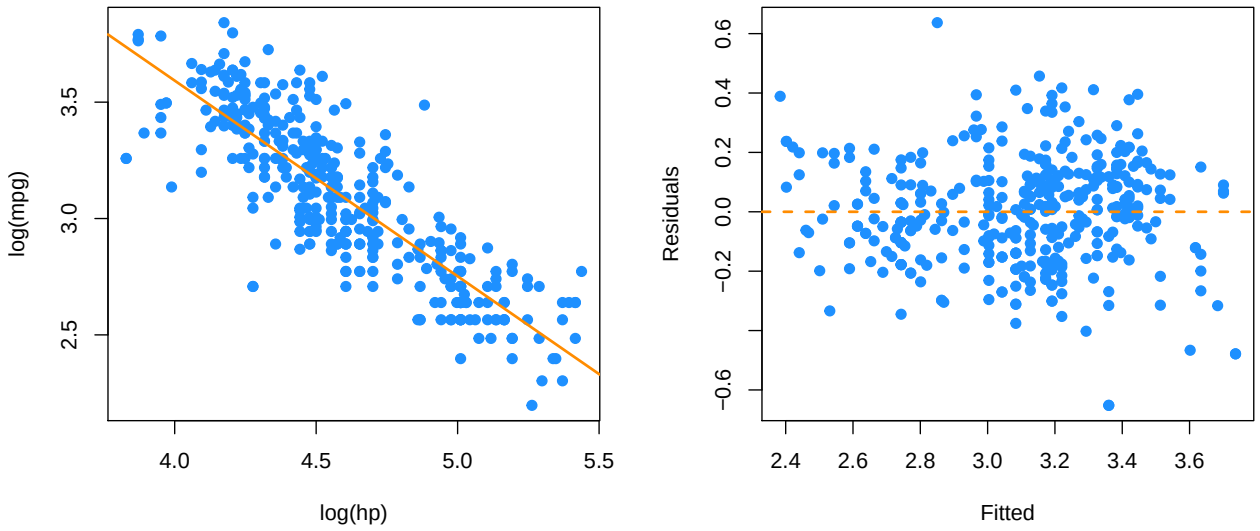



```
mpg_hp_log = lm(log(mpg) ~ hp + I(hp ^ 2), data = autompg)

par(mfrow = c(1, 2))

plot(log(mpg) ~ hp, data = autompg, col = "dodgerblue", pch = 20, cex = 1.5)
xplot = seq(min(autompg$hp), max(autompg$hp), by = 0.1)
lines(xplot, predict(mpg_hp_log, newdata = data.frame(hp = xplot)),
```

```
      col = "darkorange", lwd = 2, lty = 1)

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)
```

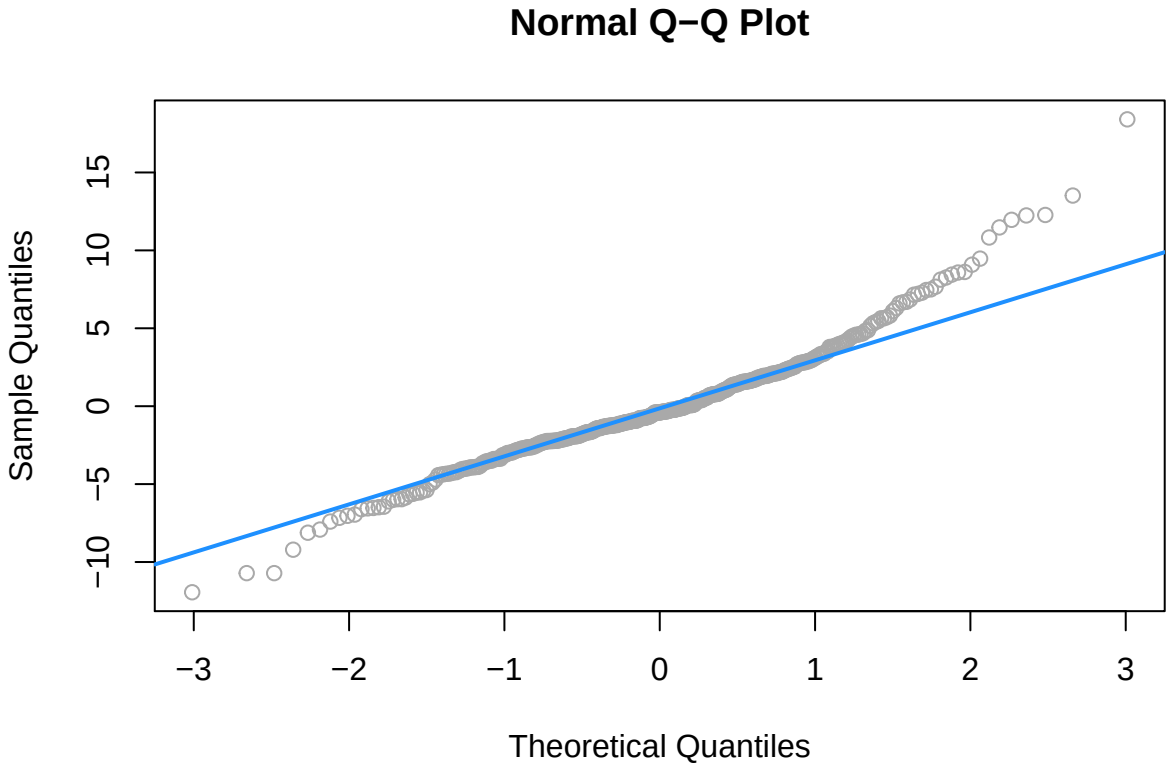



```
mpg_hp_loglog = lm(log(mpg) ~ log(hp), data = autompg)

par(mfrow = c(1, 2))
plot(log(mpg) ~ log(hp), data = autompg, col = "dodgerblue", pch = 20, cex = 1.5)
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)
```




```
big_model = lm(mpg ~ disp * hp * domestic, data = autompg)
```

```
qqnorm(resid(big_model), col = "darkgrey")
qqline(resid(big_model), col = "dodgerblue", lwd = 2)
```

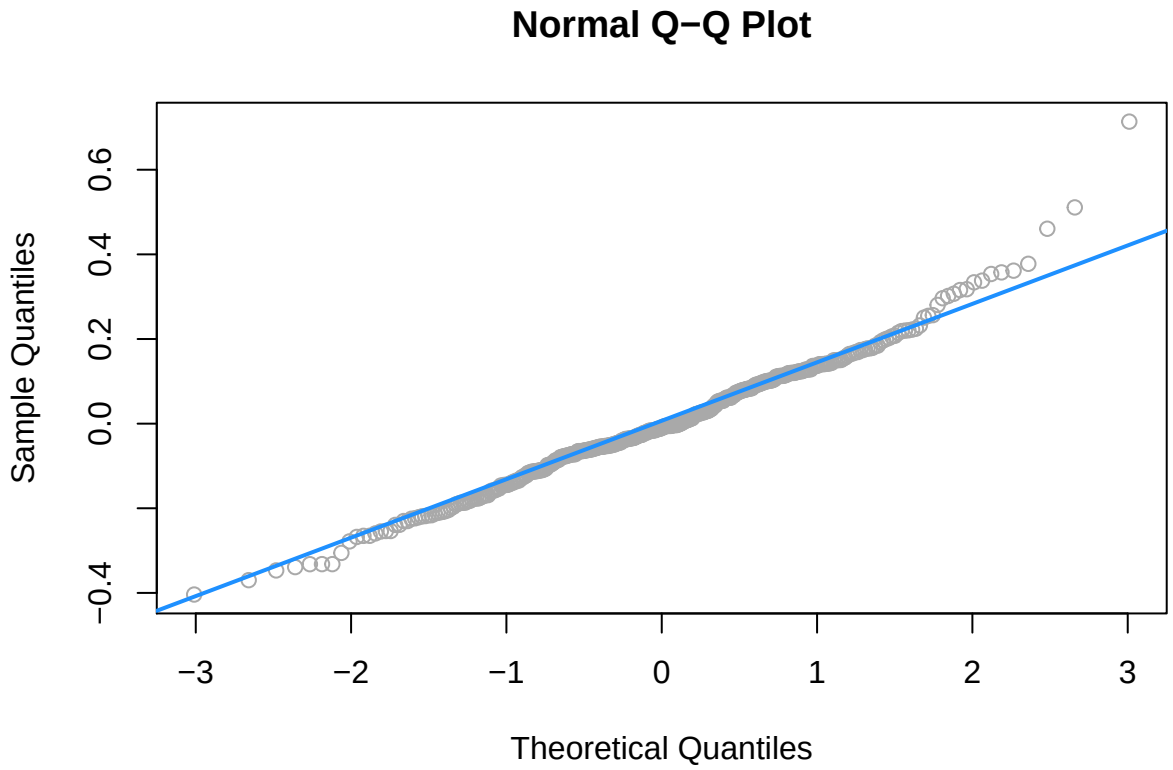



```
bigger_model = lm(log(mpg) ~ disp * hp * domestic +
                I(disp ^ 2) + I(hp ^ 2), data = autompg)
summary(bigger_model)
```

```
##
## Call:
## lm(formula = log(mpg) ~ disp * hp * domestic + I(disp^2) + I(hp^2),
##     data = autompg)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.40381 -0.08635 -0.01040  0.09995  0.71365
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        4.276e+00  2.564e-01  16.674   <2e-16 ***
## disp              -5.289e-03  2.565e-03  -2.062   0.0399 *
## hp                -7.386e-03  3.309e-03  -2.232   0.0262 *
## domestic          -2.496e-01  2.787e-01  -0.896   0.3710
## I(disp^2)          8.552e-06  4.141e-06   2.065   0.0396 *
## I(hp^2)           -1.565e-05  1.679e-05  -0.932   0.3519
## disp:hp            2.685e-05  3.082e-05   0.871   0.3842
## disp:domestic     -1.101e-03  2.526e-03  -0.436   0.6631
## hp:domestic        7.560e-03  3.689e-03   2.049   0.0411 *
## disp:hp:domestic -2.311e-05  2.662e-05  -0.868   0.3859
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1507 on 373 degrees of freedom
## Multiple R-squared:  0.8107, Adjusted R-squared:  0.8062
```

```
## F-statistic: 177.5 on 9 and 373 DF,  p-value: < 2.2e-16
```

```
qqnorm(resid(bigger_model), col = "darkgrey")
qqline(resid(bigger_model), col = "dodgerblue", lwd = 2)
```



Collinearity

After reading this chapter you will be able to:

- Identify collinearity in regression.
- Understand the effect of collinearity on regression models.

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
```

```
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.

```
coef(fit1)
```

```
## (Intercept)          x1          x2
##   2.9573357   0.9856291   1.0170586
```

```
coef(fit2)
```

```
## (Intercept)          x1          x3
##   2.1945418   0.4770998   0.2542647
```

```
coef(fit3)
```

```
## (Intercept)          x2          x3
##   1.4788921  -0.9541995   0.4928145
```

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.

```
library(faraway)
pairs(seatpos, col = "dodgerblue")
```


```
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?

```
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: 1.306e-05
```

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.

```
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.

Variance Inflation Factor.

Now note that the variance of $\hat{\beta}_j$ can be written as

$$\text{Var}(\hat{\beta}_j) = \sigma^2 C_{jj} = \sigma^2 \left(\frac{1}{1 - R_j^2} \right) \frac{1}{S_{x_j x_j}}$$

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 affects a model. If we add a moderate 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 affect the coefficients of a model.

```
set.seed(1337)
noise = rnorm(n = nrow(seatpos), mean = 0, sd = 5)
hip_model_noise = lm(hipcenter + noise ~ ., data = seatpos)
```

Adding the noise had such a large effect, the sign of the coefficient for `Ht` has changed.

```
coef(hip_model)
```

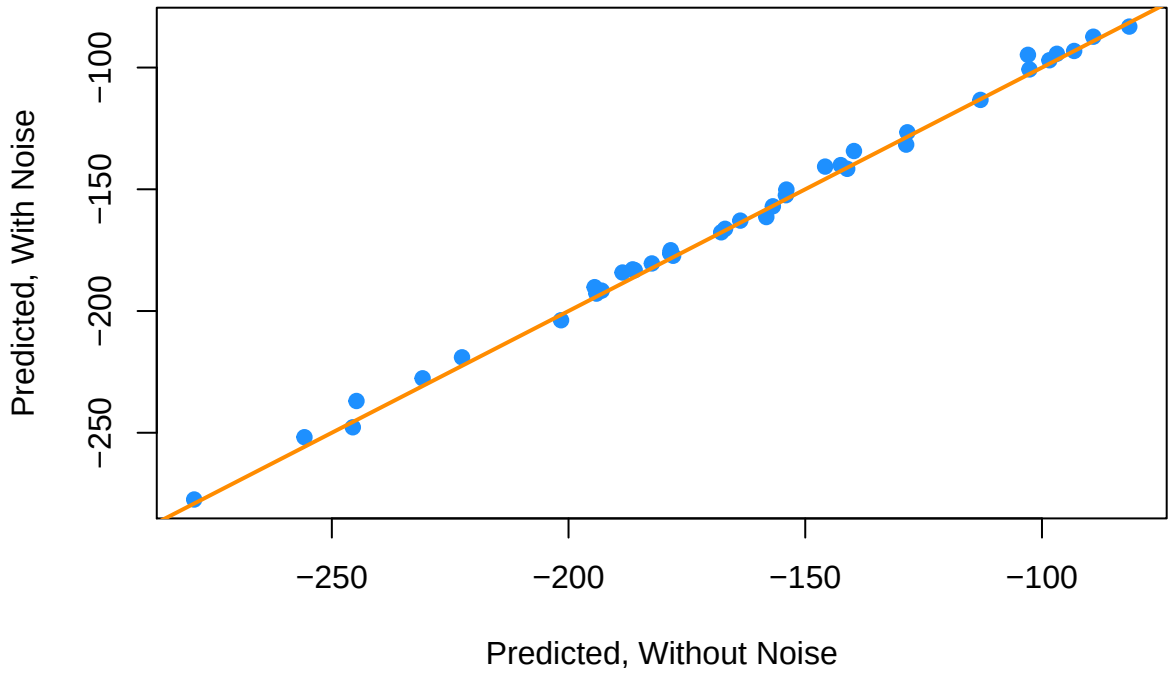
```
##  (Intercept)          Age       Weight      HtShoes           Ht       Seated
## 436.43212823   0.77571620   0.02631308  -2.69240774   0.60134458   0.53375170
##          Arm        Thigh          Leg
##  -1.32806864  -1.14311888  -6.43904627
```

```
coef(hip_model_noise)
```

```
##  (Intercept)          Age       Weight      HtShoes           Ht       Seated
## 415.32909380   0.76578240   0.01910958  -2.90377584  -0.12068122   2.03241638
##          Arm        Thigh          Leg
##  -1.02127944  -0.89034509  -5.61777220
```

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 affect prediction?

```
plot(fitted(hip_model), fitted(hip_model_noise), col = "dodgerblue", pch = 20,
     xlab = "Predicted, Without Noise", ylab = "Predicted, With Noise", cex = 1.5)
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 + Arm + Ht, data = seatpos)
summary(hip_model_small)
```

```
##
## Call:
## lm(formula = hipcenter ~ Age + Arm + Ht, data = seatpos)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -82.347 -24.745  -0.094  23.555  58.314
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 493.2491   101.0724   4.880 2.46e-05 ***
## Age           0.7988     0.5111   1.563  0.12735
## Arm          -2.9385     3.5210  -0.835  0.40979
## Ht           -3.4991     0.9954  -3.515  0.00127 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 36.12 on 34 degrees of freedom
```

```
## Multiple R-squared:  0.6631, Adjusted R-squared:  0.6333
## F-statistic:  22.3 on 3 and 34 DF,  p-value: 3.649e-08
```

```
vif(hip_model_small)
```

```
##      Age      Arm       Ht
## 1.749943 3.996766 3.508693
```

Immediately we see that multicollinearity isn't an issue here.

```
anova(hip_model_small, hip_model)
```

```
## Analysis of Variance Table
##
## Model 1: hipcenter ~ Age + Arm + Ht
## Model 2: hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh +
##     Leg
##   Res.Df   RSS Df Sum of Sq      F Pr(>F)
## 1     34 44354
## 2     29 41262  5    3091.9 0.4346 0.8207
```

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`, `Arm`, 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`, `Arm`, and `Ht`).

```
ht_shoes_model_small = lm(HtShoes ~ Age + Arm + Ht, data = seatpos)
```

So now, the residuals of `hip_model_small` give us the variation of `hipcenter` that is *unexplained* by `Age`, `Arm`, and `Ht`. Similarly, the residuals of `ht_shoes_model_small` give us the variation of `HtShoes` unexplained by `Age`, `Arm`, and `Ht`.

The correlation of these two residuals gives us the **partial correlation coefficient** of `HtShoes` and `hipcenter` with the effects of `Age`, `Arm`, and `Ht` removed.

```
cor(resid(ht_shoes_model_small), resid(hip_model_small))
```

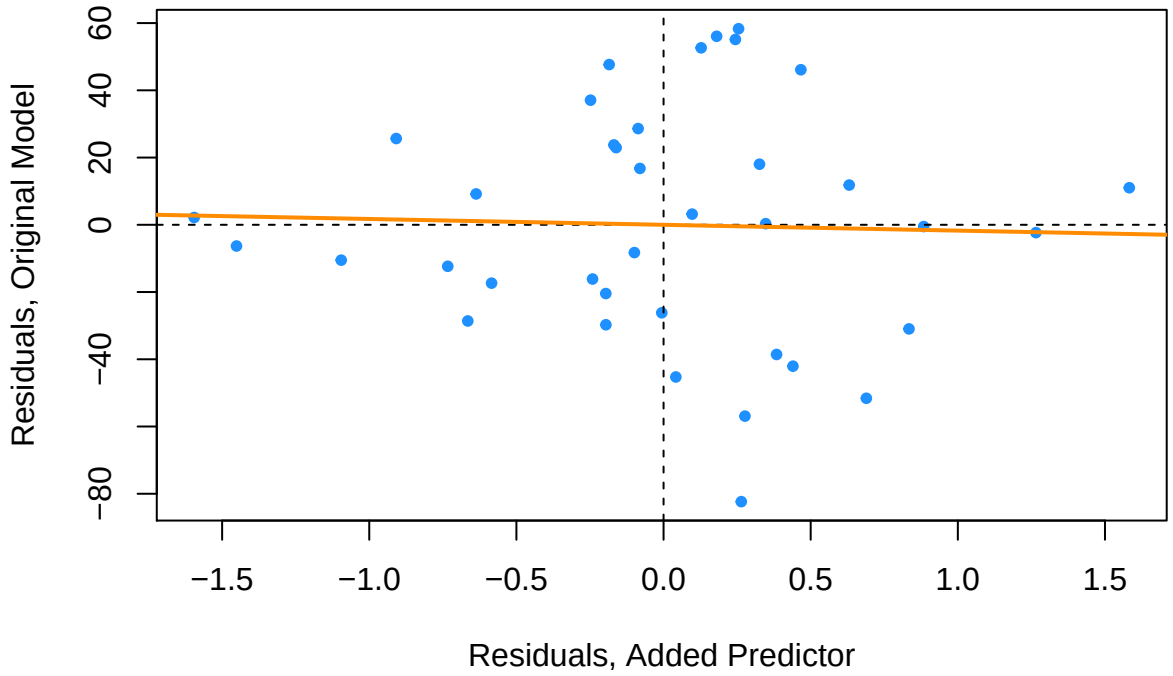
```
## [1] -0.03311061
```

Since this value is small, close to zero, it means that the variation of `hipcenter` that is unexplained by `Age`, `Arm`, and `Ht` shows very little correlation with the variation of `HtShoes` that is not explained by `Age`, `Arm`, 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.

```
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.

Simulation

Here we simulate example 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 2500 simulations for both situations.

```
sample_size = 10
num_sim     = 2500
```

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)
```

```
c(sd(x1), sd(x2))
```

```
## [1] 3.02765 3.02765
```

```
cor(x1, x2)
```

```
## [1] 0.9393939
```

Notice that they have extremely high correlation.

```
true_line_bad = beta_0 + beta_1 * x1 + beta_2 * x2
beta_hat_bad  = matrix(0, num_sim, 2)
mse_bad       = rep(0, num_sim)
```

We perform the simulation 2500 times, each time fitting a regression model, and storing the estimated coefficients and the MSE.

```
for (s in 1:num_sim) {
  y = true_line_bad + rnorm(n = sample_size, mean = 0, sd = sigma)
  reg_out = lm(y ~ x1 + x2)
  beta_hat_bad[s, ] = coef(reg_out)[-1]
  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.

```
z1 = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
z2 = 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.

```
c(sd(z1), sd(z2))
```

```
## [1] 3.02765 3.02765
```

```
cor(z1, z2)
```

```
## [1] 0.03030303
```

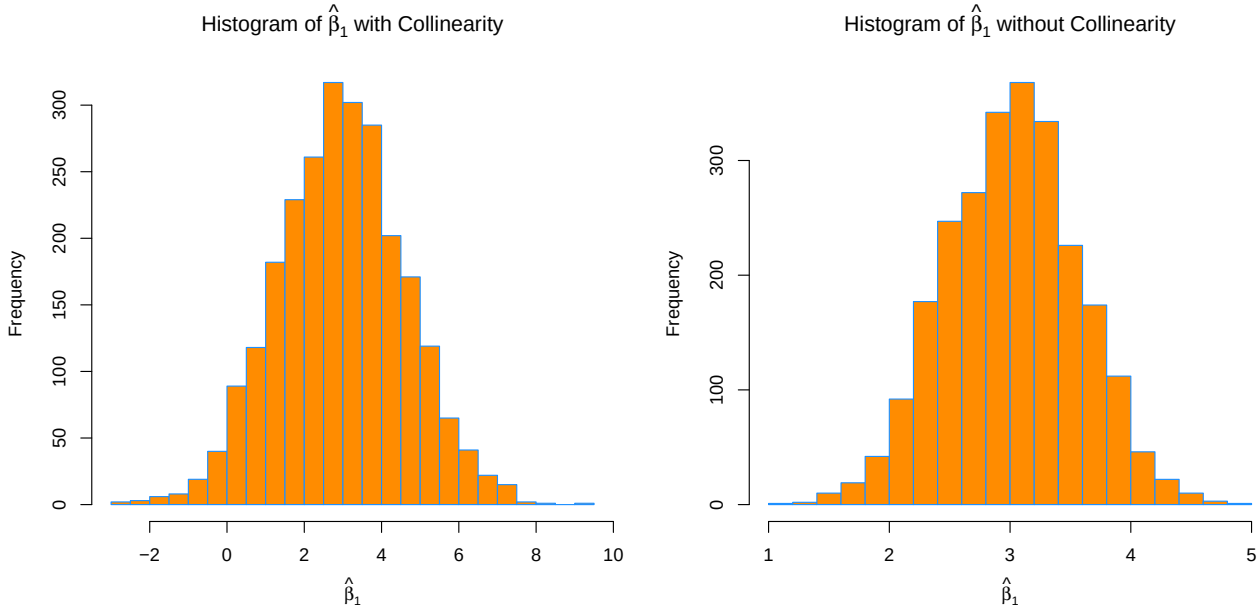
```
true_line_good = beta_0 + beta_1 * z1 + beta_2 * z2
beta_hat_good  = matrix(0, num_sim, 2)
mse_good       = rep(0, num_sim)
```

We then perform simulations and store the same results.

```
for (s in 1:num_sim) {
  y = true_line_good + rnorm(n = sample_size, mean = 0, sd = sigma)
  reg_out = lm(y ~ z1 + z2)
  beta_hat_good[s, ] = coef(reg_out)[-1]
  mse_good[s] = mean(resid(reg_out) ^ 2)
}
```

We'll now investigate the differences.

```
par(mfrow = c(1, 2))
hist(beta_hat_bad[, 1],
     col = "darkorange",
     border = "dodgerblue",
     main = expression("Histogram of " *hat(beta)[1]* " with Collinearity"),
     xlab = expression(hat(beta)[1]),
     breaks = 20)
hist(beta_hat_good[, 1],
     col = "darkorange",
     border = "dodgerblue",
     main = expression("Histogram of " *hat(beta)[1]* " without Collinearity"),
     xlab = expression(hat(beta)[1]),
     breaks = 20)
```



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[, 1])
```

```
## [1] 2.963325
```

```
mean(beta_hat_good[, 1])
```

```
## [1] 3.013414
```

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!

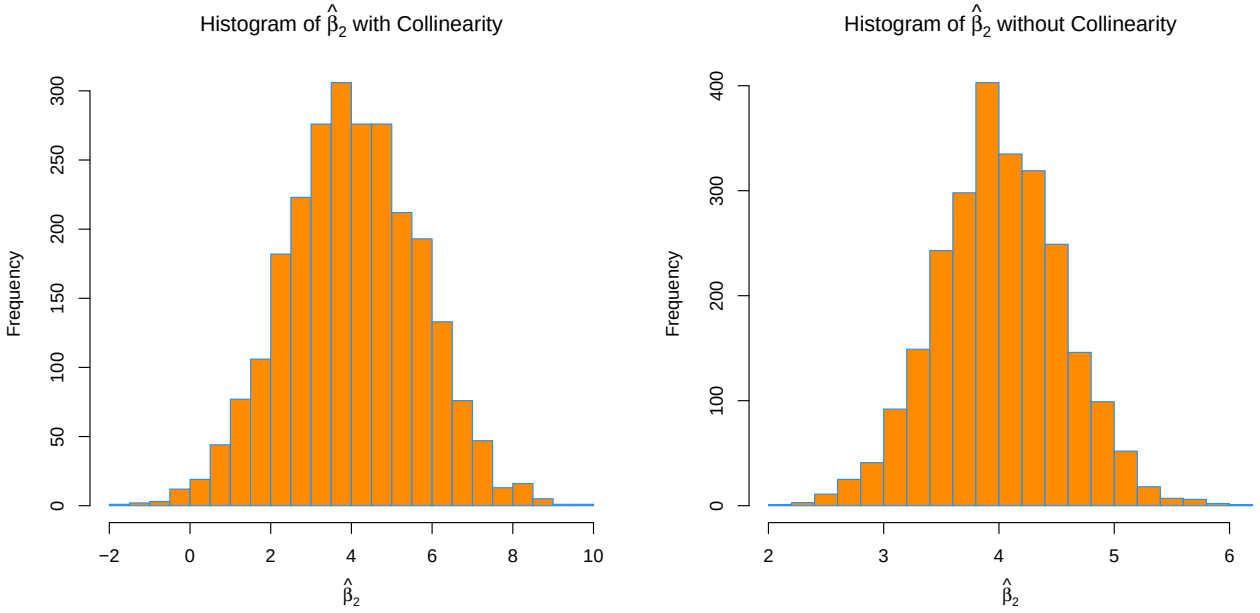
```
sd(beta_hat_bad[, 1])
```

```
## [1] 1.633294
```

```
sd(beta_hat_good[, 1])
```

```
## [1] 0.5484684
```

```
par(mfrow = c(1, 2))
hist(beta_hat_bad[, 2],
     col = "darkorange",
     border = "dodgerblue",
     main = expression("Histogram of " *hat(beta)[2]* " with Collinearity"),
     xlab = expression(hat(beta)[2]),
     breaks = 20)
hist(beta_hat_good[, 2],
     col = "darkorange",
     border = "dodgerblue",
     main = expression("Histogram of " *hat(beta)[2]* " without Collinearity"),
     xlab = expression(hat(beta)[2]),
     breaks = 20)
```



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[, 2])
```

```
## [1] 4.025059
```

```
mean(beta_hat_good[, 2])
```

```
## [1] 4.004913
```

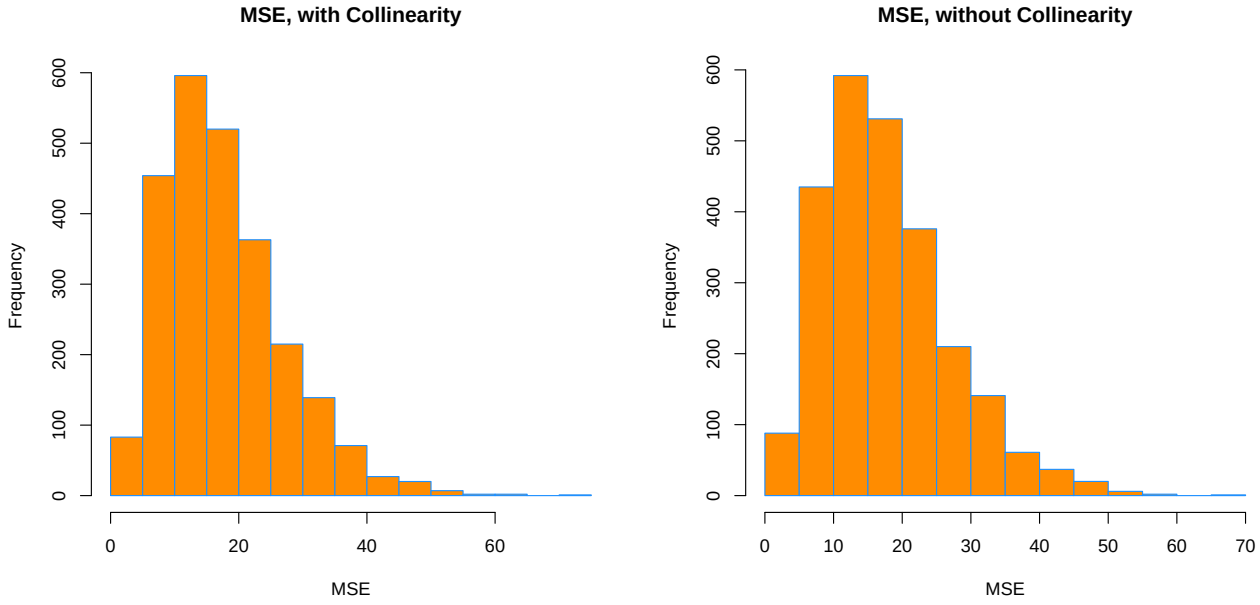
```
sd(beta_hat_bad[, 2])
```

```
## [1] 1.642592
```

```
sd(beta_hat_good[, 2])
```

```
## [1] 0.5470381
```

```
par(mfrow = c(1, 2))
hist(mse_bad,
     col = "darkorange",
     border = "dodgerblue",
     main = "MSE, with Collinearity",
     xlab = "MSE")
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 affects a model's ability to *explain*, but not predict.

```
mean(mse_bad)
```

```
## [1] 17.7186
```

```
mean(mse_good)
```

```
## [1] 17.70513
```

Variable Selection and Model Building

After reading this chapter you will be able to:

- Understand the trade-off between goodness-of-fit and model complexity.
- Use variable selection procedures to find a good model from a set of possible models.
- Understand the two uses of models: explanation and prediction.

Last chapter we saw how correlation between predictor variables can have undesirable effects on models. We used variance inflation factors to assess the severity of the collinearity issues caused by these correlations. We

also saw how fitting a smaller model, leaving out some of the correlated predictors, results in a model which no longer suffers from collinearity issues. But how should we chose this smaller model?

This chapter, we will discuss several *criteria* and *procedures* for choosing a "good" model from among a choice of many.

Quality Criterion

So far, we have seen criteria such as R^2 and RMSE for assessing quality of fit. However, both of these have a fatal flaw. By increasing the size of a model, that is adding predictors, that can at worst not improve. It is impossible to add a predictor to a model and make R^2 or RMSE worse. That means, if we were to use either of these to chose between models, we would *always* simply choose the larger model. Eventually we would simply be fitting to noise.

This suggests that we need a quality criteria that takes into account the size of the model, since our preference is for small models that still fit well. We are willing to sacrifice a small amount of "goodness-of-fit" for obtaining a smaller model. (Here we use "goodness-of-fit" to simply mean how far the data is from the model, the smaller the errors the better. Often in statistics, goodness-of-fit can have a more precise meaning.) We will look at three criteria that do this explicitly: AIC, BIC, and Adjusted R^2. We will also look at one, Cross-Validated RMSE, which implicitly considers the size of the model.

Akaike Information Criterion

The first criteria we will discuss is the Akaike Information Criterion, or AIC for short. (Note that, when *Akaike* first introduced this metric, it was simply called *An* Information Criterion. The *A* has changed meaning over the years.)

Recall, the maximized log-likelihood of a regression model can be written as

$$\log L(\hat{\boldsymbol{\beta}}, \hat{\sigma}^2) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log\left(\frac{\text{RSS}}{n}\right) - \frac{n}{2},$$

where $\text{RSS} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ and $\hat{\boldsymbol{\beta}}$ and $\hat{\sigma}^2$ were chosen to maximize the likelihood.

Then we can define AIC as

$$\text{AIC} = -2\log L(\hat{\boldsymbol{\beta}}, \hat{\sigma}^2) + 2p = n + n\log(2\pi) + n\log\left(\frac{\text{RSS}}{n}\right) + 2p,$$

which is a measure of quality of the model. The smaller the AIC, the better. To see why, let's talk about the two main components of AIC, the **likelihood** (which measures "goodness-of-fit") and the **penalty** (which is a function of the size of the model).

The likelihood portion of AIC is given by

$$-2\log L(\hat{\boldsymbol{\beta}}, \hat{\sigma}^2) = n + n\log(2\pi) + n\log\left(\frac{\text{RSS}}{n}\right).$$

For the sake of comparing models, the only term here that will change is $n\log\left(\frac{\text{RSS}}{n}\right)$, which is function of RSS. The

$$n + n\log(2\pi)$$

terms will be constant across all models applied to the same data. So, when a model fits well, that is, has a low RSS, then this likelihood component will be small.

Similarly, we can discuss the penalty component of AIC which is,

$$2p,$$

where p is the number of β parameters in the model. We call this a penalty, because it is large when p is large, but we are seeking to find a small AIC

Thus, a good model, that is one with a small AIC, will have a good balance between fitting well, and using a small number of parameters. For comparing models

$$\text{AIC} = n \log\left(\frac{\text{RSS}}{n}\right) + 2p$$

is a sufficient expression, as $n + n\log(2\pi)$ is the same across all models for any particular dataset.

Bayesian Information Criterion

The Bayesian Information Criterion, or BIC, is similar to AIC, but has a larger penalty. BIC also quantifies the trade-off between a model which fits well and the number of model parameters, however for a reasonable sample size, generally picks a smaller model than AIC. Again, for model selection use the model with the smallest BIC.

$$\text{BIC} = -2\log L(\hat{\boldsymbol{\beta}}, \hat{\sigma}^2) + \log(n)p = n + n\log(2\pi) + n\log\left(\frac{\text{RSS}}{n}\right) + \log(n)p.$$

Notice that the AIC penalty was

$$2p,$$

whereas for BIC, the penalty is

$$\log(n)p.$$

So, for any dataset where $log(n) > 2$ the BIC penalty will be larger than the AIC penalty, thus BIC will likely prefer a smaller model.

Note that, sometimes the penalty is considered a general expression of the form

$$k \cdot p.$$

Then, for AIC $k = 2$, and for BIC $k = \log(n)$.

For comparing models

$$\text{BIC} = n \log\left(\frac{\text{RSS}}{n}\right) + \log(n)p$$

is again a sufficient expression, as $n + n\log(2\pi)$ is the same across all models for any particular dataset.

Adjusted R-Squared

Recall,

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

We now define

$$R^2_a = 1 - \frac{\text{SSE}/(n-p)}{\text{SST}/(n-1)} = 1 - \left(\frac{n-1}{n-p}\right)(1 - R^2)$$

which we call the Adjusted R^2.

Unlike R^2 which can never become smaller with added predictors, Adjusted R^2 effectively penalizes for additional predictors, and can decrease with added predictors. Like R^2, larger is still better.

Cross-Validated RMSE

Each of the previous three metrics explicitly used p, the number of parameters, in their calculations. Thus, they all explicitly limit the size of models chosen when used to compare models.

We'll now briefly introduce **overfitting** and **cross-validation**.

```
make_poly_data = function(sample_size = 11) {
  x = seq(0, 10)
  y = 3 + x + 4 * x ^ 2 + rnorm(n = sample_size, mean = 0, sd = 20)
  data.frame(x, y)
}
```

```
set.seed(1234)
poly_data = make_poly_data()
```

Here we have generated data where the mean of Y is a quadratic function of a single predictor x, specifically,

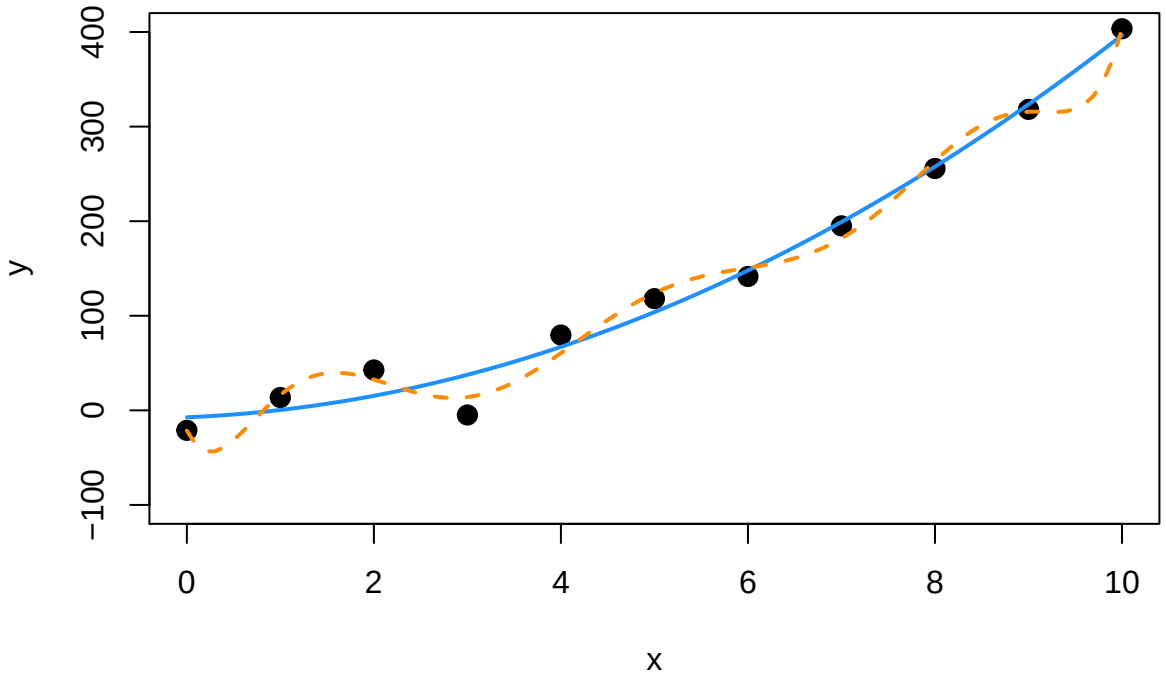
$$Y = 3 + x + 4x^2 + \epsilon.$$

We'll now fit two models to this data, one which has the correct form, quadratic, and one that is large, which includes terms up to and including an eighth degree.

```
fit_quad = lm(y ~ poly(x, degree = 2), data = poly_data)
fit_big  = lm(y ~ poly(x, degree = 8), data = poly_data)
```

We then plot the data and the results of the two models.

```
plot(y ~ x, data = poly_data, ylim = c(-100, 400), cex = 2, pch = 20)
xplot = seq(0, 10, by = 0.1)
lines(xplot, predict(fit_quad, newdata = data.frame(x = xplot)),
      col = "dodgerblue", lwd = 2, lty = 1)
lines(xplot, predict(fit_big, newdata = data.frame(x = xplot)),
      col = "darkorange", lwd = 2, lty = 2)
```



We can see that the solid blue curve models this data rather nicely. The dashed orange curve fits the points better, making smaller errors, however it is unlikely that it is correctly modeling the true relationship between x and y. It is fitting the random noise. This is an example of **overfitting**.

We see that the larger model indeed has a lower RMSE.

```
sqrt(mean(resid(fit_quad) ^ 2))
```

```
## [1] 17.61812
```

```
sqrt(mean(resid(fit_big) ^ 2))
```

```
## [1] 10.4197
```

To correct for this, we will introduce cross-validation. We define the leave-one-out cross-validated RMSE to be

$$\text{RMSE}_{\text{LOOCV}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_{[i]}^2}.$$

The $e_{[i]}$ are the residual for the ith observation, when that observation is **not** used to fit the model.

$$e_{[i]} = y_i - \hat{y}_{[i]}$$

That is, the fitted value is calculated as

$$\hat{y}_{[i]} = \boldsymbol{x}_i^\top \hat{\beta}_{[i]}$$

where $\hat{\beta}_{[i]}$ are the estimated coefficients when the ith observation is removed from the dataset.

In general, to perform this calculation, we would be required to fit the model n times, once with each possible observation removed. However, for leave-one-out cross-validation and linear models, the equation can be rewritten as

$$\text{RMSE}_{\text{LOOCV}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{e_i}{1-h_i}\right)^2},$$

where h_i are the leverages and e_i are the usual residuals. This is great, because now we can obtain the LOOCV RMSE by fitting only one model! In practice 5 or 10 fold cross-validation are much more popular. For example, in 5-fold cross-validation, the model is fit 5 times, each time leaving out a fifth of the data, then predicting on those values. We'll leave in-depth examination of cross-validation to a machine learning course, and simply use LOOCV here.

Let's calculate LOOCV RMSE for both models, then discuss *why* we want to do so. We first write a function which calculates the LOOCV RMSE as defined using the shortcut formula for linear models.

```
calc_loocv_rmse = function(model) {
  sqrt(mean((resid(model) / (1 - hatvalues(model))) ^ 2))
}
```

Then calculate the metric for both models.

```
calc_loocv_rmse(fit_quad)
```

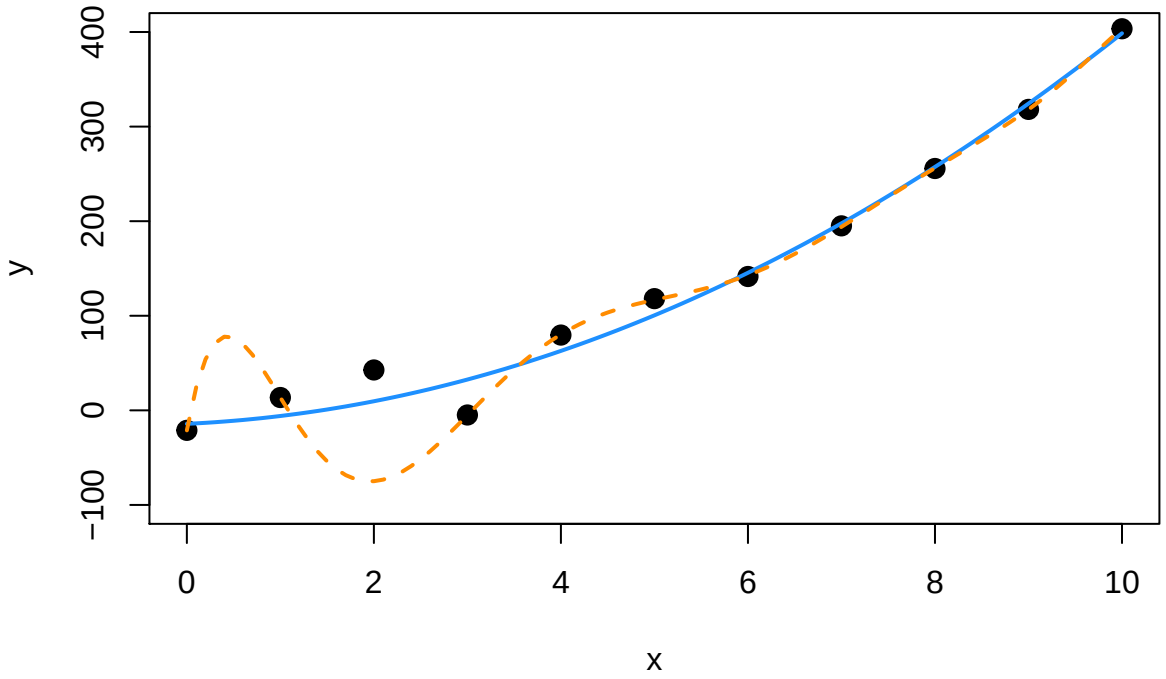
```
## [1] 23.57189
```

```
calc_loocv_rmse(fit_big)
```

```
## [1] 1334.357
```

Now we see that the quadratic model has a much smaller LOOCV RMSE, so we would prefer this quadratic model. This is because the large model has *severely* over-fit the data. By leaving a single data point out and fitting the large model, the resulting fit is much different than the fit using all of the data. For example, let's leave out the third data point and fit both models, then plot the result.

```
fit_quad_removed = lm(y ~ poly(x, degree = 2), data = poly_data[-3, ])
fit_big_removed  = lm(y ~ poly(x, degree = 8), data = poly_data[-3, ])

plot(y ~ x, data = poly_data, ylim = c(-100, 400), cex = 2, pch = 20)
xplot = seq(0, 10, by = 0.1)
lines(xplot, predict(fit_quad_removed, newdata = data.frame(x = xplot)),
      col = "dodgerblue", lwd = 2, lty = 1)
lines(xplot, predict(fit_big_removed, newdata = data.frame(x = xplot)),
      col = "darkorange", lwd = 2, lty = 2)
```



We see that on average, the solid blue line for the quadratic model has similar errors as before. It has changed very slightly. However, the dashed orange line for the large model, has a huge error at the point that was removed and is much different that the previous fit.

This is the purpose of cross-validation. By assessing how the model fits points that were not used to perform the regression, we get an idea of how well the model will work for future observations. It assess how well the model works in general, not simply on the observed data.

Selection Procedures

We've now seen a number of model quality criteria, but now we need to address which models to consider. Model selection involves both a quality criterion, plus a search procedure.

```
library(faraway)
hipcenter_mod = lm(hipcenter ~ ., data = seatpos)
coef(hipcenter_mod)
```

```
##  (Intercept)          Age       Weight      HtShoes           Ht       Seated
## 436.43212823   0.77571620   0.02631308  -2.69240774   0.60134458   0.53375170
##          Arm        Thigh          Leg
##  -1.32806864  -1.14311888  -6.43904627
```

Let's return to the `seatpos` data from the `faraway` package. Now, let's consider only models with first order terms, thus no interactions and no polynomials. There are *eight* predictors in this model. So if we consider all possible models, ranging from using 0 predictors, to all eight predictors, there are

$$\sum_{k=0}^{p-1} \binom{p-1}{k} = 2^{p-1} = 2^8 = 256$$

possible models.

If we had 10 or more predictors, we would already be considering over 1000 models! For this reason, we often search through possible models in an intelligent way, bypassing some models that are unlikely to be considered good. We will consider three search procedures: backwards, forwards, and stepwise.

Backward Search

Backward selection procedures start with all possible predictors in the model, then considers how deleting a single predictor will effect a chosen metric. Let's try this on the `seatpos` data. We will use the `step()` function in `R` which by default uses AIC as its metric of choice.

```
hipcenter_mod_back_aic = step(hipcenter_mod, direction = "backward")
```

```
## Start:  AIC=283.62
## hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh +
##     Leg
##
##           Df Sum of Sq   RSS    AIC
## - Ht       1      5.01 41267 281.63
## - Weight   1      8.99 41271 281.63
## - Seated   1     28.64 41290 281.65
## - HtShoes  1    108.43 41370 281.72
## - Arm      1    164.97 41427 281.78
## - Thigh    1    262.76 41525 281.87
## <none>                 41262 283.62
## - Age      1   2632.12 43894 283.97
## - Leg      1   2654.85 43917 283.99
##
## Step:  AIC=281.63
## hipcenter ~ Age + Weight + HtShoes + Seated + Arm + Thigh + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Weight   1     11.10 41278 279.64
## - Seated   1     30.52 41297 279.66
## - Arm      1    160.50 41427 279.78
## - Thigh    1    269.08 41536 279.88
## - HtShoes  1    971.84 42239 280.51
## <none>                 41267 281.63
## - Leg      1   2664.65 43931 282.01
## - Age      1   2808.52 44075 282.13
##
## Step:  AIC=279.64
## hipcenter ~ Age + HtShoes + Seated + Arm + Thigh + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Seated   1     35.10 41313 277.67
## - Arm      1    156.47 41434 277.78
## - Thigh    1    285.16 41563 277.90
## - HtShoes  1    975.48 42253 278.53
## <none>                 41278 279.64
## - Leg      1   2661.39 43939 280.01
## - Age      1   3011.86 44290 280.31
##
## Step:  AIC=277.67
## hipcenter ~ Age + HtShoes + Arm + Thigh + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Arm      1    172.02 41485 275.83
## - Thigh    1    344.61 41658 275.99
## - HtShoes  1   1853.43 43166 277.34
```

```
## <none>                 41313 277.67
## - Leg      1   2871.07 44184 278.22
## - Age      1   2976.77 44290 278.31
##
## Step:  AIC=275.83
## hipcenter ~ Age + HtShoes + Thigh + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Thigh    1     472.8 41958 274.26
## <none>                 41485 275.83
## - HtShoes  1    2340.7 43826 275.92
## - Age      1    3501.0 44986 276.91
## - Leg      1    3591.7 45077 276.98
##
## Step:  AIC=274.26
## hipcenter ~ Age + HtShoes + Leg
##
##           Df Sum of Sq   RSS    AIC
## <none>                 41958 274.26
## - Age      1    3108.8 45067 274.98
## - Leg      1    3476.3 45434 275.28
## - HtShoes  1    4218.6 46176 275.90
```

We start with the model `hipcenter ~ .`, which is otherwise known as `hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh + Leg`. `R` will then repeatedly attempt to delete a predictor until it stops, or reaches the model `hipcenter ~ 1`, which contains no predictors.

At each "step", `R` reports the current model, its AIC, and the possible steps with their RSS and more importantly AIC.

In this example, at the first step, the current model is `hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh + Leg` which has an AIC of `283.62`. Note that when `R` is calculating this value, it is using `extractAIC()`, which uses the expression

$$\text{AIC} = n \log\left(\frac{\text{RSS}}{n}\right) + 2p,$$

which we quickly verify.

```
extractAIC(hipcenter_mod) # returns both p and AIC
```

```
## [1]   9.000 283.624
```

```
n = length(resid(hipcenter_mod))
(p = length(coef(hipcenter_mod)))
```

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

```
n * log(mean(resid(hipcenter_mod) ^ 2)) + 2 * p
```

```
## [1] 283.624
```

Returning to the first step, `R` then gives us a row which shows the effect of deleting each of the current predictors. The `-` signs at the beginning of each row indicates we are considering removing a predictor. There is also a row with `<none>` which is a row for keeping the current model. Notice that this row has the smallest RSS, as it is the largest model.

We see that every row above `<none>` has a smaller AIC than the row for `<none>` with the one at the top, `Ht`, giving the lowest AIC. Thus we remove `Ht` from the model, and continue the process.

Notice, in the second step, we start with the model `hipcenter ~ Age + Weight + HtShoes + Seated + Arm + Thigh + Leg` and the variable `Ht` is no longer considered.

We continue the process until we reach the model `hipcenter ~ Age + HtShoes + Leg`. At this step, the row for `<none>` tops the list, as removing any additional variable will not improve the AIC This is the model which is stored in `hipcenter_mod_back_aic`.

```
coef(hipcenter_mod_back_aic)
```

```
## (Intercept)         Age     HtShoes         Leg
## 456.2136538   0.5998327  -2.3022555  -6.8297461
```

We could also search through the possible models in a backwards fashion using BIC. To do so, we again use the `step()` function, but now specify `k = log(n)`, where `n` stores the number of observations in the data.

```
n = length(resid(hipcenter_mod))
hipcenter_mod_back_bic = step(hipcenter_mod, direction = "backward", k = log(n))
```

```
## Start:  AIC=298.36
## hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh +
##     Leg
##
##           Df Sum of Sq   RSS    AIC
## - Ht       1      5.01 41267 294.73
## - Weight   1      8.99 41271 294.73
## - Seated   1     28.64 41290 294.75
## - HtShoes  1    108.43 41370 294.82
## - Arm      1    164.97 41427 294.88
## - Thigh    1    262.76 41525 294.97
## - Age      1   2632.12 43894 297.07
## - Leg      1   2654.85 43917 297.09
## <none>                 41262 298.36
##
## Step:  AIC=294.73
## hipcenter ~ Age + Weight + HtShoes + Seated + Arm + Thigh + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Weight   1     11.10 41278 291.10
## - Seated   1     30.52 41297 291.12
## - Arm      1    160.50 41427 291.24
## - Thigh    1    269.08 41536 291.34
## - HtShoes  1    971.84 42239 291.98
## - Leg      1   2664.65 43931 293.47
## - Age      1   2808.52 44075 293.59
## <none>                 41267 294.73
##
## Step:  AIC=291.1
## hipcenter ~ Age + HtShoes + Seated + Arm + Thigh + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Seated   1     35.10 41313 287.50
## - Arm      1    156.47 41434 287.61
## - Thigh    1    285.16 41563 287.73
## - HtShoes  1    975.48 42253 288.35
## - Leg      1   2661.39 43939 289.84
## - Age      1   3011.86 44290 290.14
```

```
## <none>                 41278 291.10
##
## Step:  AIC=287.5
## hipcenter ~ Age + HtShoes + Arm + Thigh + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Arm      1    172.02 41485 284.02
## - Thigh    1    344.61 41658 284.18
## - HtShoes  1   1853.43 43166 285.53
## - Leg      1   2871.07 44184 286.41
## - Age      1   2976.77 44290 286.50
## <none>                 41313 287.50
##
## Step:  AIC=284.02
## hipcenter ~ Age + HtShoes + Thigh + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Thigh    1     472.8 41958 280.81
## - HtShoes  1    2340.7 43826 282.46
## - Age      1    3501.0 44986 283.46
## - Leg      1    3591.7 45077 283.54
## <none>                 41485 284.02
##
## Step:  AIC=280.81
## hipcenter ~ Age + HtShoes + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Age      1    3108.8 45067 279.89
## - Leg      1    3476.3 45434 280.20
## <none>                 41958 280.81
## - HtShoes  1    4218.6 46176 280.81
##
## Step:  AIC=279.89
## hipcenter ~ HtShoes + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Leg      1    3038.8 48105 278.73
## <none>                 45067 279.89
## - HtShoes  1    5004.4 50071 280.25
##
## Step:  AIC=278.73
## hipcenter ~ HtShoes
##
##           Df Sum of Sq    RSS    AIC
## <none>                  48105 278.73
## - HtShoes  1     83534 131639 313.35
```

The procedure is exactly the same, except at each step we look to improve the BIC, which R still labels AIC in the output.

The variable `hipcenter_mod_back_bic` stores the model chosen by this procedure.

```
coef(hipcenter_mod_back_bic)
```

```
## (Intercept)     HtShoes
##  565.592659   -4.262091
```

We note that this model is *smaller*, has fewer predictors, than the model chosen by AIC, which is what we would expect. Also note that while both models are different, neither uses both `Ht` and `HtShoes` which are extremely correlated.

We can use information from the `summary()` function to compare their Adjusted R^2 values. Note that either selected model performs better than the original full model.

```
summary(hipcenter_mod)$adj.r.squared
```

```
## [1] 0.6000855
```

```
summary(hipcenter_mod_back_aic)$adj.r.squared
```

```
## [1] 0.6531427
```

```
summary(hipcenter_mod_back_bic)$adj.r.squared
```

```
## [1] 0.6244149
```

We can also calculate the LOOCV RMSE for both selected models, as well as the full model.

```
calc_loocv_rmse(hipcenter_mod)
```

```
## [1] 44.44564
```

```
calc_loocv_rmse(hipcenter_mod_back_aic)
```

```
## [1] 37.58473
```

```
calc_loocv_rmse(hipcenter_mod_back_bic)
```

```
## [1] 37.40564
```

We see that we would prefer the model chosen via BIC if using LOOCV RMSE as our metric.

Forward Search

Forward selection is the exact opposite of backwards selection. Here we tell `R` to start with a model using no predictors, that is `hipcenter ~ 1`, then at each step `R` will attempt to add a predictor until it finds a good model or reaches `hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh + Leg`.

```
hipcenter_mod_start = lm(hipcenter ~ 1, data = seatpos)
hipcenter_mod_forw_aic = step(
  hipcenter_mod_start,
  scope = hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh + Leg,
  direction = "forward")
```

```
## Start:  AIC=311.71
## hipcenter ~ 1
##
##           Df Sum of Sq    RSS    AIC
## + Ht       1     84023  47616 275.07
## + HtShoes  1     83534  48105 275.45
## + Leg      1     81568  50071 276.98
## + Seated   1     70392  61247 284.63
## + Weight   1     53975  77664 293.66
## + Thigh    1     46010  85629 297.37
## + Arm      1     45065  86574 297.78
## <none>                 131639 311.71
## + Age      1      5541 126098 312.07
##
```

```
## Step:  AIC=275.07
## hipcenter ~ Ht
##
##           Df Sum of Sq   RSS    AIC
## + Leg      1   2781.10 44835 274.78
## <none>                 47616 275.07
## + Age      1   2353.51 45262 275.14
## + Weight   1    195.86 47420 276.91
## + Seated   1    101.56 47514 276.99
## + Arm      1     75.78 47540 277.01
## + HtShoes  1     25.76 47590 277.05
## + Thigh    1      4.63 47611 277.06
##
## Step:  AIC=274.78
## hipcenter ~ Ht + Leg
##
##           Df Sum of Sq   RSS    AIC
## + Age      1   2896.60 41938 274.24
## <none>                 44835 274.78
## + Arm      1    522.72 44312 276.33
## + Weight   1    445.10 44390 276.40
## + HtShoes  1     34.11 44801 276.75
## + Thigh    1     32.96 44802 276.75
## + Seated   1      1.12 44834 276.78
##
## Step:  AIC=274.24
## hipcenter ~ Ht + Leg + Age
##
##           Df Sum of Sq   RSS    AIC
## <none>                 41938 274.24
## + Thigh    1    372.71 41565 275.90
## + Arm      1    257.09 41681 276.01
## + Seated   1    121.26 41817 276.13
## + Weight   1     46.83 41891 276.20
## + HtShoes  1     13.38 41925 276.23
```

Again, by default `R` uses AIC as its quality metric when using the `step()` function. Also note that now the rows begin with a `+` which indicates addition of predictors to the current model from any step.

```
hipcenter_mod_forw_bic = step(
  hipcenter_mod_start,
  scope = hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh + Leg,
  direction = "forward", k = log(n))
```

```
## Start:  AIC=313.35
## hipcenter ~ 1
##
##           Df Sum of Sq    RSS    AIC
## + Ht       1     84023  47616 278.34
## + HtShoes  1     83534  48105 278.73
## + Leg      1     81568  50071 280.25
## + Seated   1     70392  61247 287.91
## + Weight   1     53975  77664 296.93
## + Thigh    1     46010  85629 300.64
## + Arm      1     45065  86574 301.06
```

```
## <none>                 131639 313.35
## + Age      1      5541 126098 315.35
##
## Step:  AIC=278.34
## hipcenter ~ Ht
##
##           Df Sum of Sq   RSS    AIC
## <none>                 47616 278.34
## + Leg      1   2781.10 44835 279.69
## + Age      1   2353.51 45262 280.05
## + Weight   1    195.86 47420 281.82
## + Seated   1    101.56 47514 281.90
## + Arm      1     75.78 47540 281.92
## + HtShoes  1     25.76 47590 281.96
## + Thigh    1      4.63 47611 281.98
```

We can make the same modification as last time to instead use BIC with forward selection.

```
summary(hipcenter_mod)$adj.r.squared
```

```
## [1] 0.6000855
```

```
summary(hipcenter_mod_forw_aic)$adj.r.squared
```

```
## [1] 0.6533055
```

```
summary(hipcenter_mod_forw_bic)$adj.r.squared
```

```
## [1] 0.6282374
```

We can compare the two selected models' Adjusted R^2 as well as their LOOCV RMSE The results are very similar to those using backwards selection, although the models are not exactly the same.

```
calc_loocv_rmse(hipcenter_mod)
```

```
## [1] 44.44564
```

```
calc_loocv_rmse(hipcenter_mod_forw_aic)
```

```
## [1] 37.62516
```

```
calc_loocv_rmse(hipcenter_mod_forw_bic)
```

```
## [1] 37.2511
```

Stepwise Search

Stepwise search checks going both backwards and forwards at every step. It considers the addition of any variable not currently in the model, as well as the removal of any variable currently in the model.

Here we perform stepwise search using AIC as our metric. We start with the model `hipcenter ~ 1` and search up to `hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh + Leg`. Notice that at many of the steps, some row begin with `-`, while others begin with `+`.

```
hipcenter_mod_both_aic = step(
  hipcenter_mod_start,
  scope = hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh + Leg,
  direction = "both")
```

```
## Start:  AIC=311.71
## hipcenter ~ 1
```

```
##
##           Df Sum of Sq    RSS    AIC
## + Ht       1     84023  47616 275.07
## + HtShoes  1     83534  48105 275.45
## + Leg      1     81568  50071 276.98
## + Seated   1     70392  61247 284.63
## + Weight   1     53975  77664 293.66
## + Thigh    1     46010  85629 297.37
## + Arm      1     45065  86574 297.78
## <none>                 131639 311.71
## + Age      1      5541 126098 312.07
##
## Step:  AIC=275.07
## hipcenter ~ Ht
##
##           Df Sum of Sq    RSS    AIC
## + Leg      1      2781  44835 274.78
## <none>                  47616 275.07
## + Age      1      2354  45262 275.14
## + Weight   1       196  47420 276.91
## + Seated   1       102  47514 276.99
## + Arm      1        76  47540 277.01
## + HtShoes  1        26  47590 277.05
## + Thigh    1         5  47611 277.06
## - Ht       1     84023 131639 311.71
##
## Step:  AIC=274.78
## hipcenter ~ Ht + Leg
##
##           Df Sum of Sq   RSS    AIC
## + Age      1    2896.6 41938 274.24
## <none>                  44835 274.78
## - Leg      1    2781.1 47616 275.07
## + Arm      1     522.7 44312 276.33
## + Weight   1     445.1 44390 276.40
## + HtShoes  1      34.1 44801 276.75
## + Thigh    1      33.0 44802 276.75
## + Seated   1       1.1 44834 276.78
## - Ht       1    5236.3 50071 276.98
##
## Step:  AIC=274.24
## hipcenter ~ Ht + Leg + Age
##
##           Df Sum of Sq   RSS    AIC
## <none>                  41938 274.24
## - Age      1    2896.6 44835 274.78
## - Leg      1    3324.2 45262 275.14
## - Ht       1    4238.3 46176 275.90
## + Thigh    1     372.7 41565 275.90
## + Arm      1     257.1 41681 276.01
## + Seated   1     121.3 41817 276.13
## + Weight   1      46.8 41891 276.20
## + HtShoes  1      13.4 41925 276.23
```

We could again instead use BIC as our metric.

```
hipcenter_mod_both_bic = step(
  hipcenter_mod_start,
  scope = hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh + Leg,
  direction = "both", k = log(n))
```

```
## Start:  AIC=313.35
## hipcenter ~ 1
##
##           Df Sum of Sq    RSS    AIC
## + Ht       1     84023  47616 278.34
## + HtShoes  1     83534  48105 278.73
## + Leg      1     81568  50071 280.25
## + Seated   1     70392  61247 287.91
## + Weight   1     53975  77664 296.93
## + Thigh    1     46010  85629 300.64
## + Arm      1     45065  86574 301.06
## <none>                 131639 313.35
## + Age      1      5541 126098 315.35
##
## Step:  AIC=278.34
## hipcenter ~ Ht
##
##           Df Sum of Sq    RSS    AIC
## <none>                  47616 278.34
## + Leg      1      2781  44835 279.69
## + Age      1      2354  45262 280.05
## + Weight   1       196  47420 281.82
## + Seated   1       102  47514 281.90
## + Arm      1        76  47540 281.92
## + HtShoes  1        26  47590 281.96
## + Thigh    1         5  47611 281.98
## - Ht       1     84023 131639 313.35
```

Adjusted R^2 and LOOCV RMSE comparisons are similar to backwards and forwards, which is not at all surprising, as some of the models selected are the same as before.

```
summary(hipcenter_mod)$adj.r.squared
```

```
## [1] 0.6000855
```

```
summary(hipcenter_mod_both_aic)$adj.r.squared
```

```
## [1] 0.6533055
```

```
summary(hipcenter_mod_both_bic)$adj.r.squared
```

```
## [1] 0.6282374
```

```
calc_loocv_rmse(hipcenter_mod)
```

```
## [1] 44.44564
```

```
calc_loocv_rmse(hipcenter_mod_both_aic)
```

```
## [1] 37.62516
```

```
calc_loocv_rmse(hipcenter_mod_both_bic)
```

```
## [1] 37.2511
```

Exhaustive Search

Backward, forward, and stepwise search are all useful, but do have an obvious issue. By not checking every possible model, sometimes they will miss the best possible model. With an extremely large number of predictors, sometimes this is necessary since checking every possible model would be rather time consuming, even with current computers.

However, with a reasonably sized dataset, it isn't too difficult to check all possible models. To do so, we will use the `regsubsets()` function in the `R` package `leaps`.

```
library(leaps)
all_hipcenter_mod = summary(regsubsets(hipcenter ~ ., data = seatpos))
```

A few points about this line of code. First, note that we immediately use `summary()` and store those results. That is simply the intended use of `regsubsets()`. Second, inside of `regsubsets()` we specify the model `hipcenter ~ .`. This will be the largest model considered, that is the model using all first-order predictors, and `R` will check all possible subsets.

We'll now look at the information stored in `all_hipcenter_mod`.

```
all_hipcenter_mod$which
```

```
##   (Intercept)   Age Weight HtShoes    Ht Seated   Arm Thigh   Leg
## 1        TRUE FALSE  FALSE   FALSE  TRUE  FALSE FALSE FALSE FALSE
## 2        TRUE FALSE  FALSE   FALSE  TRUE  FALSE FALSE FALSE  TRUE
## 3        TRUE  TRUE  FALSE   FALSE  TRUE  FALSE FALSE FALSE  TRUE
## 4        TRUE  TRUE  FALSE    TRUE FALSE  FALSE FALSE  TRUE  TRUE
## 5        TRUE  TRUE  FALSE    TRUE FALSE  FALSE  TRUE  TRUE  TRUE
## 6        TRUE  TRUE  FALSE    TRUE FALSE   TRUE  TRUE  TRUE  TRUE
## 7        TRUE  TRUE   TRUE    TRUE FALSE   TRUE  TRUE  TRUE  TRUE
## 8        TRUE  TRUE   TRUE    TRUE  TRUE   TRUE  TRUE  TRUE  TRUE
```

Using `$which` gives us the best model, according to RSS, for a model of each possible size, in this case ranging from one to eight predictors. For example the best model with four predictors ($p = 5$) would use `Age`, `HtShoes`, `Thigh`, and `Leg`.

```
all_hipcenter_mod$rss
```

```
## [1] 47615.79 44834.69 41938.09 41485.01 41313.00 41277.90 41266.80 41261.78
```

We can obtain the RSS for each of these models using `$rss`. Notice that these are decreasing since the models range from small to large.

Now that we have the RSS for each of these models, it is rather easy to obtain AIC, BIC, and Adjusted R^2 since they are all a function of RSS Also, since we have the models with the best RSS for each size, they will result in the models with the best AIC, BIC, and Adjusted R^2 for each size. Then by picking from those, we can find the overall best AIC, BIC, and Adjusted R^2.

Conveniently, Adjusted R^2 is automatically calculated.

```
all_hipcenter_mod$adjr2
```

```
## [1] 0.6282374 0.6399496 0.6533055 0.6466586 0.6371276 0.6257403 0.6133690
## [8] 0.6000855
```

To find which model has the highest Adjusted R^2 we can use the `which.max()` function.

```
(best_r2_ind = which.max(all_hipcenter_mod$adjr2))
```

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

We can then extract the predictors of that model.

```
all_hipcenter_mod$which[best_r2_ind, ]
```

```
## (Intercept)         Age      Weight     HtShoes          Ht      Seated
##        TRUE        TRUE       FALSE       FALSE        TRUE       FALSE
##         Arm       Thigh         Leg
##       FALSE       FALSE        TRUE
```

We'll now calculate AIC and BIC for each of the models with the best RSS. To do so, we will need both n and the p for the largest possible model.

```
p = length(coef(hipcenter_mod))
n = length(resid(hipcenter_mod))
```

We'll use the form of AIC which leaves out the constant term that is equal across all models.

$$\text{AIC} = n \log\left(\frac{\text{RSS}}{n}\right) + 2p.$$

Since we have the RSS of each model stored, this is easy to calculate.

```
hipcenter_mod_aic = n * log(all_hipcenter_mod$rss / n) + 2 * (2:p)
```

We can then extract the predictors of the model with the best AIC.

```
best_aic_ind = which.min(hipcenter_mod_aic)
all_hipcenter_mod$which[best_aic_ind,]
```

```
## (Intercept)         Age      Weight     HtShoes          Ht      Seated
##        TRUE        TRUE       FALSE       FALSE        TRUE       FALSE
##         Arm       Thigh         Leg
##       FALSE       FALSE        TRUE
```

Let's fit this model so we can compare to our previously chosen models using AIC and search procedures.

```
hipcenter_mod_best_aic = lm(hipcenter ~ Age + Ht + Leg, data = seatpos)
```

The `extractAIC()` function will calculate the AIC defined above for a fitted model.

```
extractAIC(hipcenter_mod_best_aic)
```

```
## [1]   4.0000 274.2418
```

```
extractAIC(hipcenter_mod_back_aic)
```

```
## [1]   4.0000 274.2597
```

```
extractAIC(hipcenter_mod_forw_aic)
```

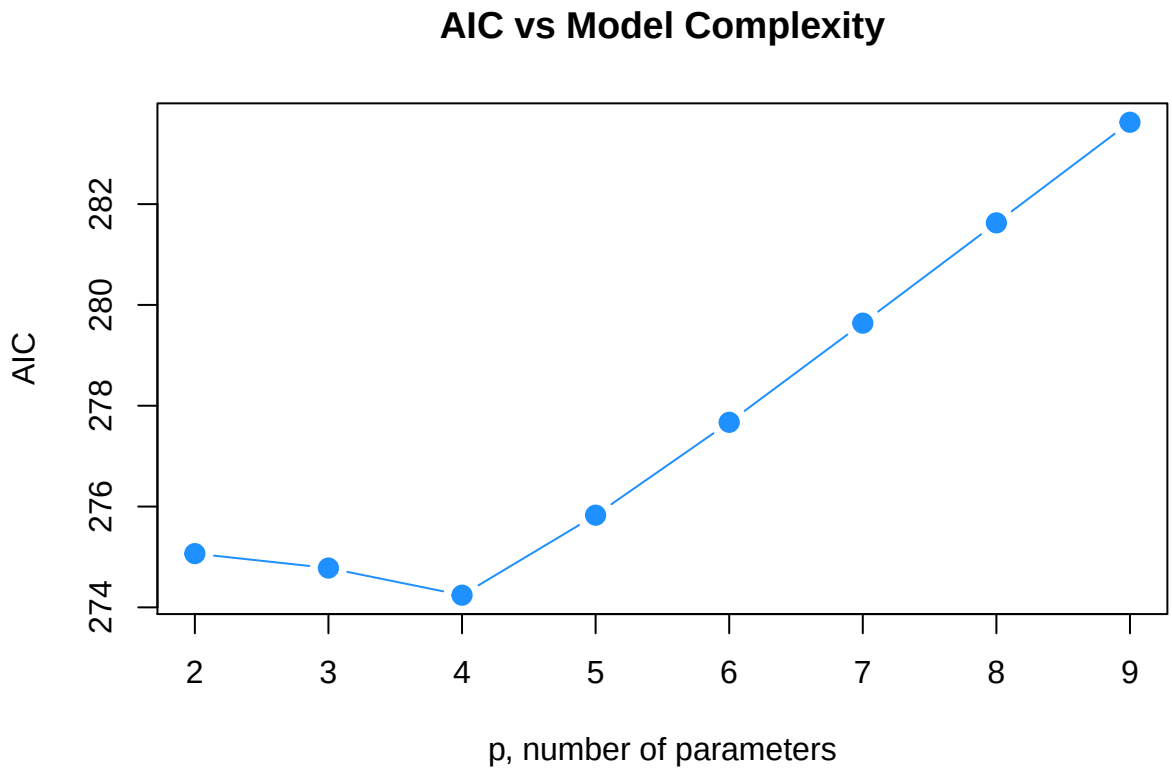
```
## [1]   4.0000 274.2418
```

```
extractAIC(hipcenter_mod_both_aic)
```

```
## [1]   4.0000 274.2418
```

We see that two of the models chosen by search procedures have the best possible AIC, as they are the same model. This is however never guaranteed. We see that the model chosen using backwards selection does not achieve the smallest possible AIC.

```
plot(hipcenter_mod_aic ~ I(2:p), ylab = "AIC", xlab = "p, number of parameters",
     pch = 20, col = "dodgerblue", type = "b", cex = 2,
     main = "AIC vs Model Complexity")
```



We could easily repeat this process for BIC.

$$\text{BIC} = n \log\left(\frac{\text{RSS}}{n}\right) + \log(n)p.$$

```
hipcenter_mod_bic = n * log(all_hipcenter_mod$rss / n) + log(n) * (2:p)
```

```
which.min(hipcenter_mod_bic)
```

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

```
all_hipcenter_mod$which[1,]
```

```
## (Intercept)         Age      Weight     HtShoes          Ht      Seated
##        TRUE       FALSE       FALSE       FALSE        TRUE       FALSE
##         Arm       Thigh         Leg
##       FALSE       FALSE       FALSE
```

```
hipcenter_mod_best_bic = lm(hipcenter ~ Ht, data = seatpos)
```

```
extractAIC(hipcenter_mod_best_bic, k = log(n))
```

```
## [1]   2.0000 278.3418
```

```
extractAIC(hipcenter_mod_back_bic, k = log(n))
```

```
## [1]   2.0000 278.7306
```

```
extractAIC(hipcenter_mod_forw_bic, k = log(n))
```

```
## [1]   2.0000 278.3418
```

```
extractAIC(hipcenter_mod_both_bic, k = log(n))
```

```
## [1]   2.0000 278.3418
```

Higher Order Terms

So far we have only allowed first-order terms in our models. Let's return to the `autompg` dataset to explore higher-order terms.

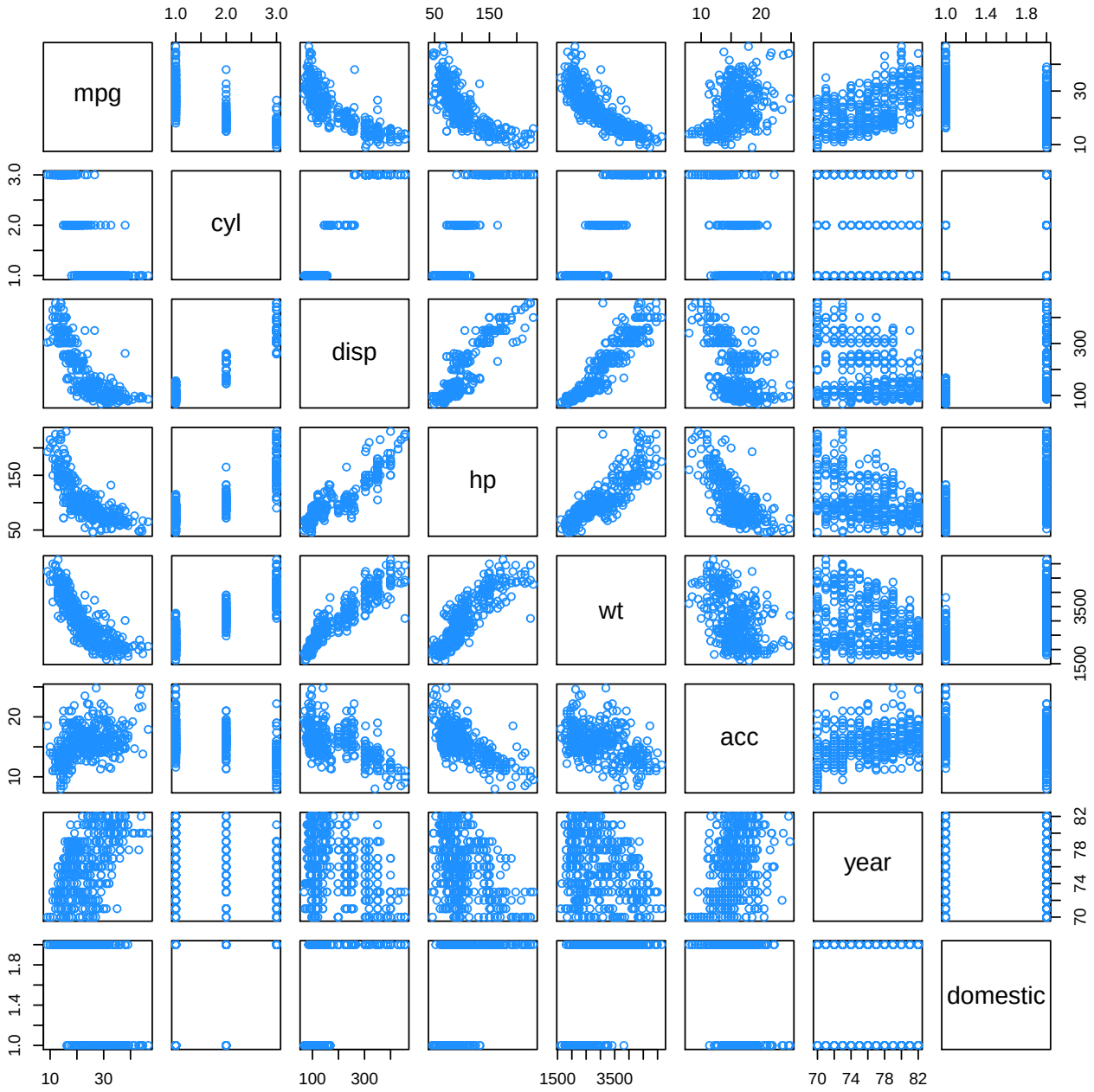
```
autompg = read.table(
  "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data",
  quote = "\"",
  comment.char = "",
  stringsAsFactors = FALSE)
colnames(autompg) = c("mpg", "cyl", "disp", "hp", "wt", "acc",
                      "year", "origin", "name")
autompg = subset(autompg, autompg$hp != "?")
autompg = subset(autompg, autompg$name != "plymouth reliant")
rownames(autompg) = paste(autompg$cyl, "cylinder", autompg$year, autompg$name)
autompg$hp = as.numeric(autompg$hp)
autompg$domestic = as.numeric(autompg$origin == 1)
autompg = autompg[autompg$cyl != 5,]
autompg = autompg[autompg$cyl != 3,]
autompg$cyl = as.factor(autompg$cyl)
autompg$domestic = as.factor(autompg$domestic)
autompg = subset(autompg, select = c("mpg", "cyl", "disp", "hp",
                                     "wt", "acc", "year", "domestic"))
```

```
str(autompg)
```

```
## 'data.frame':    383 obs. of  8 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 ...
##  $ domestic: Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
```

Recall that we have two factor variables, `cyl` and `domestic`. The `cyl` variable has three levels, while the `domestic` variable has only two. Thus the `cyl` variable will be coded using two dummy variables, while the `domestic` variable will only need one. We will pay attention to this later.

```
pairs(autompg, col = "dodgerblue")
```



We'll use the `pairs()` plot to determine which variables may benefit from a quadratic relationship with the response. We'll also consider all possible two-way interactions. We won't consider any three-order or higher. For example, we won't consider the interaction between first-order terms and the added quadratic terms.

So now, we'll fit this rather large model. We'll use a log-transformed response. Notice that `log(mpg) ~ . ^ 2` will automatically consider all first-order terms, as well as all two-way interactions. We use `I(var_name ^ 2)` to add quadratic terms for some variables. This generally works better than using `poly()` when performing variable selection.

```
autompg_big_mod = lm(
  log(mpg) ~ . ^ 2 + I(disp ^ 2) + I(hp ^ 2) + I(wt ^ 2) + I(acc ^ 2),
  data = autompg)
```

We think it is rather unlikely that we truly need all of these terms. There are quite a few!

```
length(coef(autompg_big_mod))
```

```
## [1] 40
```

We'll try backwards search with both AIC and BIC to attempt to find a smaller, more reasonable model.

```
autompg_mod_back_aic = step(autompg_big_mod, direction = "backward", trace = 0)
```

Notice that we used `trace = 0` in the function call. This suppress the output for each step, and simply stores the chosen model. This is useful, as this code would otherwise create a large amount of output. If we had viewed the output, which you can try on your own by removing `trace = 0`, we would see that `R` only considers the `cyl` variable as a single variable, despite the fact that it is coded using two dummy variables. So removing `cyl` would actually remove two parameters from the resulting model.

You should also notice that `R` respects hierarchy when attempting to remove variables. That is, for example, `R` will not consider removing `hp` if `hp:disp` or `I(hp ^ 2)` are currently in the model.

We also use BIC.

```
n = length(resid(autompg_big_mod))
autompg_mod_back_bic = step(autompg_big_mod, direction = "backward",
                            k = log(n), trace = 0)
```

Looking at the coefficients of the two chosen models, we see they are still rather large.

```
coef(autompg_mod_back_aic)
```

```
##    (Intercept)           cyl6           cyl8           disp             hp
##   3.671884e+00  -1.602563e-01  -8.581644e-01  -9.371971e-03   2.293534e-02
##             wt            acc           year      domestic1        I(hp^2)
##  -3.064497e-04  -1.393888e-01  -1.966361e-03   9.369324e-01  -1.497669e-05
##       cyl6:acc       cyl8:acc        disp:wt      disp:year         hp:acc
##   7.220298e-03   5.041915e-02   5.797816e-07   9.493770e-05  -5.062295e-04
##        hp:year       acc:year  acc:domestic1 year:domestic1
##  -1.838985e-04   2.345625e-03  -2.372468e-02  -7.332725e-03
```

```
coef(autompg_mod_back_bic)
```

```
##   (Intercept)          cyl6          cyl8          disp            hp
##  4.657847e+00 -1.086165e-01 -7.611631e-01 -1.609316e-03  2.621266e-03
##            wt           acc          year     domestic1      cyl6:acc
## -2.635972e-04 -1.670601e-01 -1.045646e-02  3.341579e-01  4.315493e-03
##      cyl8:acc       disp:wt        hp:acc      acc:year acc:domestic1
##  4.610095e-02  4.102804e-07 -3.386261e-04  2.500137e-03 -2.193294e-02
```

However, they are much smaller than the original full model. Also notice that the resulting models respect hierarchy.

```
length(coef(autompg_big_mod))
```

```
## [1] 40
```

```
length(coef(autompg_mod_back_aic))
```

```
## [1] 19
```

```
length(coef(autompg_mod_back_bic))
```

```
## [1] 15
```

Calculating the LOOCV RMSE for each, we see that the model chosen using BIC performs the best. That means that it is both the best model for prediction, since it achieves the best LOOCV RMSE, but also the best model for explanation, as it is also the smallest.

```
calc_loocv_rmse(autompg_big_mod)
```

```
## [1] 0.1112024
```

```
calc_loocv_rmse(autompg_mod_back_aic)
```

```
## [1] 0.1032888
```

```
calc_loocv_rmse(autompg_mod_back_bic)
```

```
## [1] 0.103134
```

Explanation versus Prediction

Throughout this chapter, we have attempted to find reasonably "small" models, which are good at **explaining** the relationship between the response and the predictors, that also have small errors which are thus good for making **predictions**.

We'll further discuss the model `autompg_mod_back_bic` to better explain the difference between using models for *explaining* and *predicting*. This is the model fit to the `autompg` data that was chosen using Backwards Search and BIC, which obtained the lowest LOOCV RMSE of the models we considered.

```
autompg_mod_back_bic
```

```
##
## Call:
## lm(formula = log(mpg) ~ cyl + disp + hp + wt + acc + year + domestic +
##     cyl:acc + disp:wt + hp:acc + acc:year + acc:domestic, data = autompg)
##
## Coefficients:
##   (Intercept)           cyl6           cyl8           disp             hp
##     4.658e+00     -1.086e-01     -7.612e-01     -1.609e-03      2.621e-03
##            wt            acc           year      domestic1       cyl6:acc
##    -2.636e-04     -1.671e-01     -1.046e-02      3.342e-01      4.315e-03
##      cyl8:acc        disp:wt         hp:acc       acc:year  acc:domestic1
##     4.610e-02      4.103e-07     -3.386e-04      2.500e-03     -2.193e-02
```

Notice this is a somewhat "large" model, which uses 15 parameters, including several interaction terms. Do we care that this is a "large" model? The answer is, **it depends**.

Explanation

Suppose we would like to use this model for explanation. Perhaps we are a car manufacturer trying to engineer a fuel efficient vehicle. If this is the case, we are interested in both what predictor variables are useful for explaining the car's fuel efficiency, as well as how those variables effect fuel efficiency. By understanding this relationship, we can use this knowledge to our advantage when designing a car.

To explain a relationship, we are interested in keeping models as small as possible, since smaller models are easy to interpret. The fewer predictors the less considerations we need to make in our design process. Also the fewer interactions and polynomial terms, the easier it is to interpret any one parameter, since the parameter interpretations are conditional on which parameters are in the model.

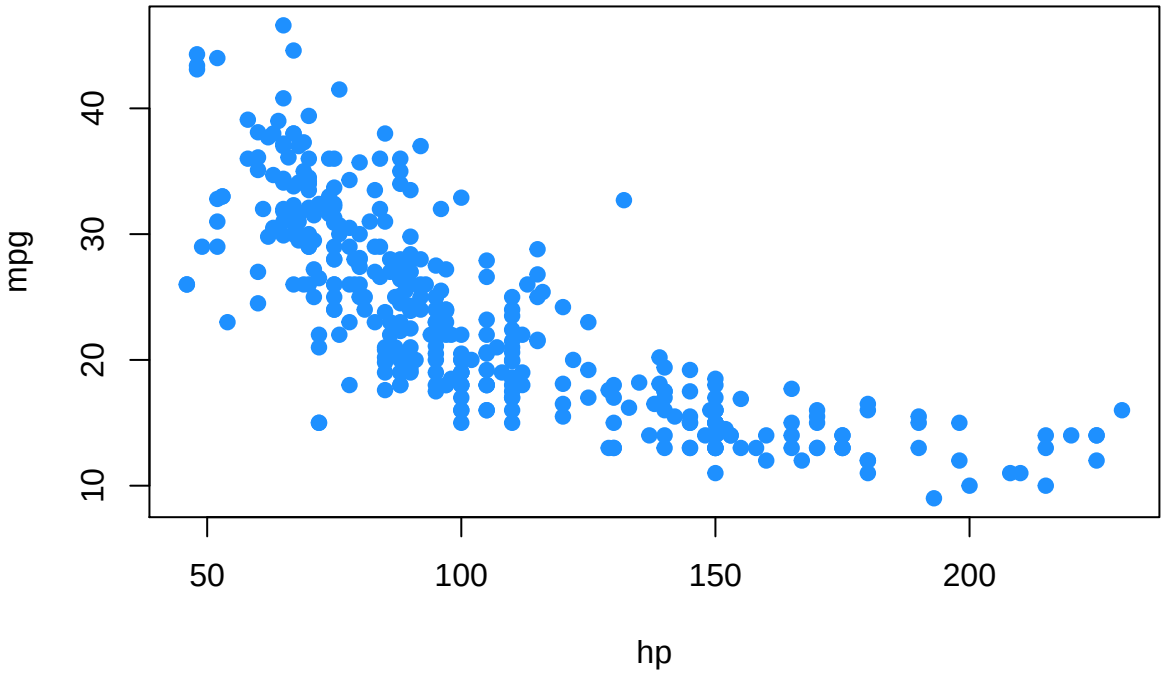
Note that *linear* models are rather interpretable to begin with. Later in your data analysis careers, you will see more complicated models that may fit data better, but are much harder, if not impossible to interpret. These models aren't very useful for explaining a relationship.

To find small and interpretable models, we would use selection criterion that *explicitly* penalize larger models, such as AIC and BIC. In this case we still obtained a somewhat large model, but much smaller than the model we used to start the selection process.

Correlation and Causation A word of caution when using a model to *explain* a relationship. There are two terms often used to describe a relationship between two variables: *causation* and *correlation*. Correlation is often also referred to as association.

Just because two variable are correlated does not necessarily mean that one causes the other. For example, considering modeling `mpg` as only a function of `hp`.

```
plot(mpg ~ hp, data = autompg, col = "dodgerblue", pch = 20, cex = 1.5)
```



Does an increase in horsepower cause a drop in fuel efficiency? Or, perhaps the causality is reversed and an increase in fuel efficiency cause a decrease in horsepower. Or, perhaps there is a third variable that explains both!

The issue here is that we have **observational** data. With observational data, we can only detect associations. To speak with confidence about causality, we would need to run **experiments**.

This is a concept that you should encounter often in your statistics education. For some further reading, and some related fallacies, see: Wikipedia: Correlation does not imply causation.

Prediction

Suppose now instead of the manufacturer who would like to build a car, we are a consumer who wishes to purchase a new car. However this particular car is so new, it has not been rigorously tested, so we are unsure of what fuel efficiency to expect. (And, as skeptics, we don't trust what the manufacturer is telling us.)

In this case, we would like to use the model to help *predict* the fuel efficiency of this car based on its attributes, which are the predictors of the model. The smaller the errors the model makes, the more confident we are in its prediction. Thus, to find models for prediction, we would use selection criterion that *implicitly* penalize larger models, such as LOOCV RMSE. So long as the model does not over-fit, we do not actually care how large the model becomes. Explaining the relationship between the variables is not our goal here, we simply want to know what kind of fuel efficiency we should expect!

If we **only** care about prediction, we don't need to worry about correlation vs causation, and we don't need to worry about model assumptions.

If a variable is correlated with the response, it doesn't actually matter if it causes an effect on the response, it can still be useful for prediction. For example, in elementary school aged children their shoe size certainly doesn't *cause* them to read at a higher level, however we could very easily use shoe size to make a prediction about a child's reading ability. The larger their shoe size, the better they read. There's a lurking variable here though, their age! (Don't send your kids to school with size 14 shoes, it won't make them read better!)

We also don't care about model assumptions. Least squares is least squares. For a specified model, it will find the values of the parameters which will minimize the squared error loss. Your results might be largely uninterpretable and useless for inference, but for prediction none of that matters.