Quantitative Biology

Lecture Notes BSc Biology

Benjamin T. Martin & André M. de Roos

Institute for Biodiversity and Ecosystem Dynamics
University of Amsterdam

B.T.Martin@uva.nl, *A.M.deRoos@uva.nl*

Version January 23, 2026

Contents

Preface

This is an evolving set of lecture notes for the course on Quantitative Biology

Statistical Analysis of Biological Data

1 Statistical modelling

1.1 Statistical models

Statistics is about using models to understand data. Statistical models are similar to physical models, such as a map of a city, in that they are simplified but informative representations of our data.

We use statistical models for 3 main reasons:

Description - We use models to describe and communicate our data in a simple way. If I were to conduct a survey of 100 UvA students about how many hours per week they study and I wanted to communicate the results of the study to the UvA administrators, I wouldn't just send them an excel sheet with 100 responses and say "see!". To understand and communicate data we need to represent data in a simpler way. The formal way of simplifying data is through statistical modeling. In some cases, the model might be extremely simple. For example, if I tell you that the average student in my survey studied 20 hours per week with a standard deviation of 3 hrs, this already tells you a lot about the 100 data points with just two numbers! While you might not realize it, this is a model! Depending on the data set, we might also want to construct more complex models to describe our data. For example, the hours studied per week might depend on the year of study and the bachelor's program of the student. In this class, we will learn a flexible framework for modeling our data sets: **general linear models**.

Inference - We use models to tell us how likely the patterns we see in our data are real patterns vs. due to random chance. In the example above, we wanted to know how much the average UvA student studies, but we only asked 100 students. How can we say something about the general population of all UvA students if we only surveyed a small fraction of these students? What is the risk of coming to the wrong conclusions just by coincidentally asking students that happen to study more or less than usual? We use statistical models to help answer these questions. We can use statistical models to tell us how much uncertainty there is around an observed pattern in our data, or how likely this pattern could result from chance alone.

Prediction - Sometimes variables that are of interest to us are very difficult to measure, and we want to be able to develop models to predict the variable of interest from easier to measure variables. A good example of this comes from remote sensing where images from satellites are used to estimate various variables on earth, for example land surface temperatures, or primary productivity in the ocean. These models are generally developed by fitting collecting datasets where both the variable of interest and the easier to measure variables were measured at various times, and fitting models to find the relationship between the two. These models can then be used to make predictions in cases where the variable of interest was not measured.

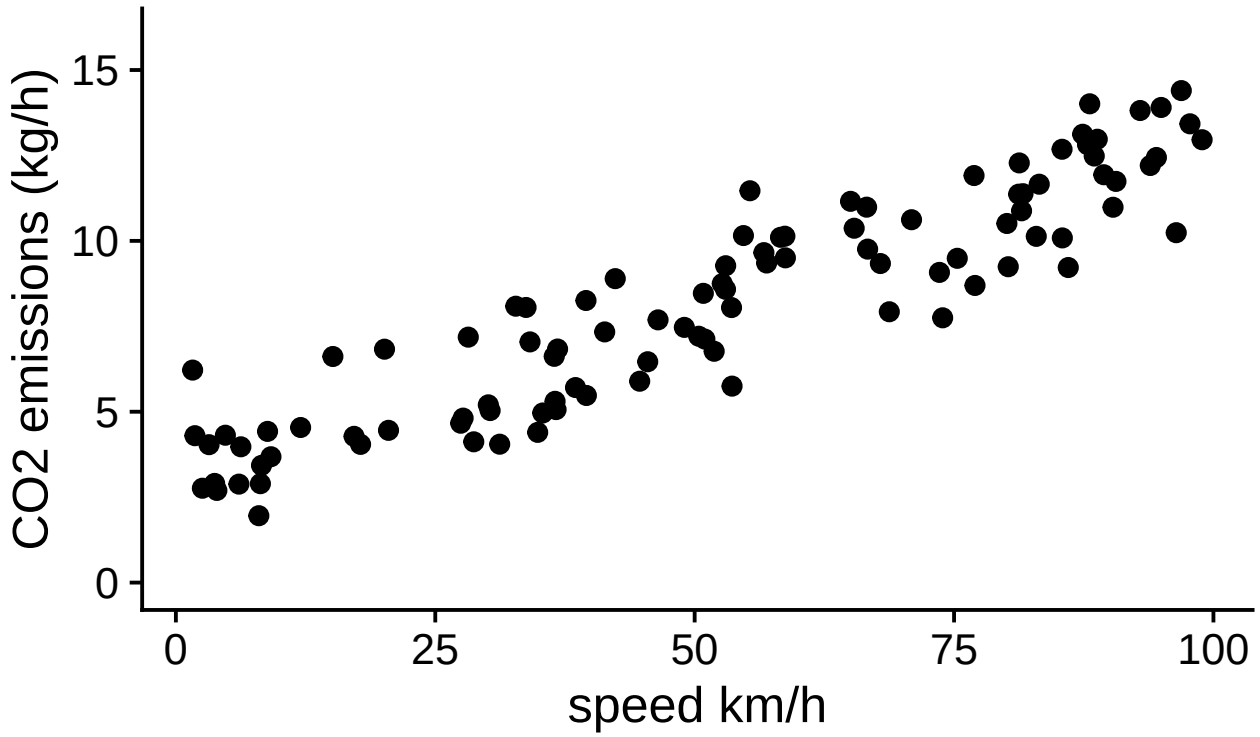
1.2 linear models

The basic structure of a statistical model is:

data = model + error

In the equation above, the "model" gives the expected value for any data point, and the error is the difference between expectations and reality. One question then is how should we model our data. In theory, we could use any type of model from complex simulation models to deep neural networks. But in general, we like to keep things as simple as possible. One of the simplest types of models one could use to model data, are linear models.

To illustrate linear models, consider the data below showing carbon emissions of cars as function of their speed:



From looking at the data, there appears to be a relationship between CO2 emissions and the speed a car travels. We can model this data, or represent the data in a simpler form, using the equation of a line, or linear equation:

$$\mathbf{y} = b_0 + b_1\mathbf{x}$$

In the context of this dataset, the predicted value of CO2 is **y** (the hat on top of the y indicates this is a predicted value). On the right hand side of the equation we see that **y** depends on the sum of two terms. The first is b_0 that does not depend on **x** (in our case speed), and the second is a term that increases linearly with **x** (speed). When speed is zero, then our predicted CO2 emission is equal to the value of the b_0, so b_0 is the intercept. In the equation y and x are written in bold because we can think of them as vectors, like columns in an excel file, where each row is a different observation and the length of these vectors is equal to the number of observations in our dataset. This equation provides a way of generating a predicted value of y for each value of x in our dataset.

1.3 Definitions

Dependent variable (y): in the example above the term on the left-hand side of the equation is called the dependent variable, (also: response variable). It is the variable that we model with our statistical model.

Independent variables (x): In statistical models, we model the dependent variable as a function of independent variables (also: predictor variables). We use the term independent variable because the dependent variable depends on the values of the independent variables, or the response variable depends on the value of the predictor variables.

Parameters (intercept (b_0) and slope (b_1)): The parameters determine the exact relationship between the dependent variable and the independent variables. In the example above, the intercept determines the modeled value of CO2 emissions for an idling car, and the slope parameter determines how much the modeled CO2 value increases per unit increase in speed.

Problem 1.1 In the next section, we will learn how to precisely fit models to data, but you should be able to look at the plot and get a rough estimate of what the model parameters need to be for the model predictions to approximate the data. Before moving on look at the figure above and estimate the two parameters of the statistical model.

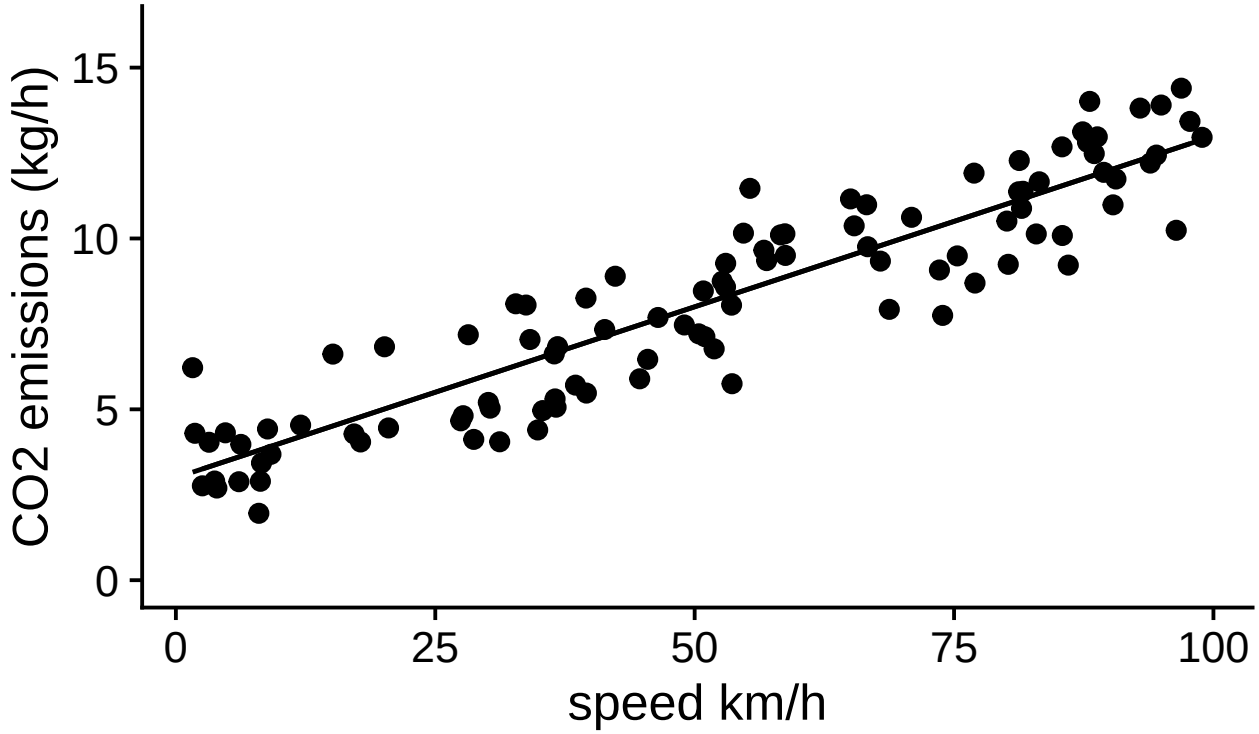
In this case, the b_0, the intercept is very easy to estimate. The observations for cars with speeds close to zero have

CO_2 emissions around 3 kg/h. For b_1, the slope, we can see that typical emissions go from around 3 to 13 kg/hr as we go from 0 to 100 km/hr. Thus the slope of this relationship is (13-3)/(100-0) ~ 0.1 (kg/hr)/(km/h).

If we then plot out the equation:

$$\hat{y} = 3 + 0.1x$$

we see that this simple linear model does a good job at capturing the trend in the data:



Importantly by using a model to describe our data, we are able to boil 100 observations down to 2 numbers, a slope and an intercept that capture the salient features of the data. This allows us to convey important information about the data, for example, how quickly does CO2 emissions go up with speed, or how much CO2 does the average idling car emit? It is important to note that there is variability in the data set, different cars have different CO2 emissions even when traveling at the same speed. This has several consequences.

1. Statistical models are not perfect representations of our data, for any observation there will be some deviation between the predicted and the observed value. This is related to the equation we first introduced: data = model + error. In our new notation the data is y, the model predictions are $\hat{y}$, and the difference between observations and model predictions is the error.

$$data = model + error$$

$$y = \hat{y} + error$$

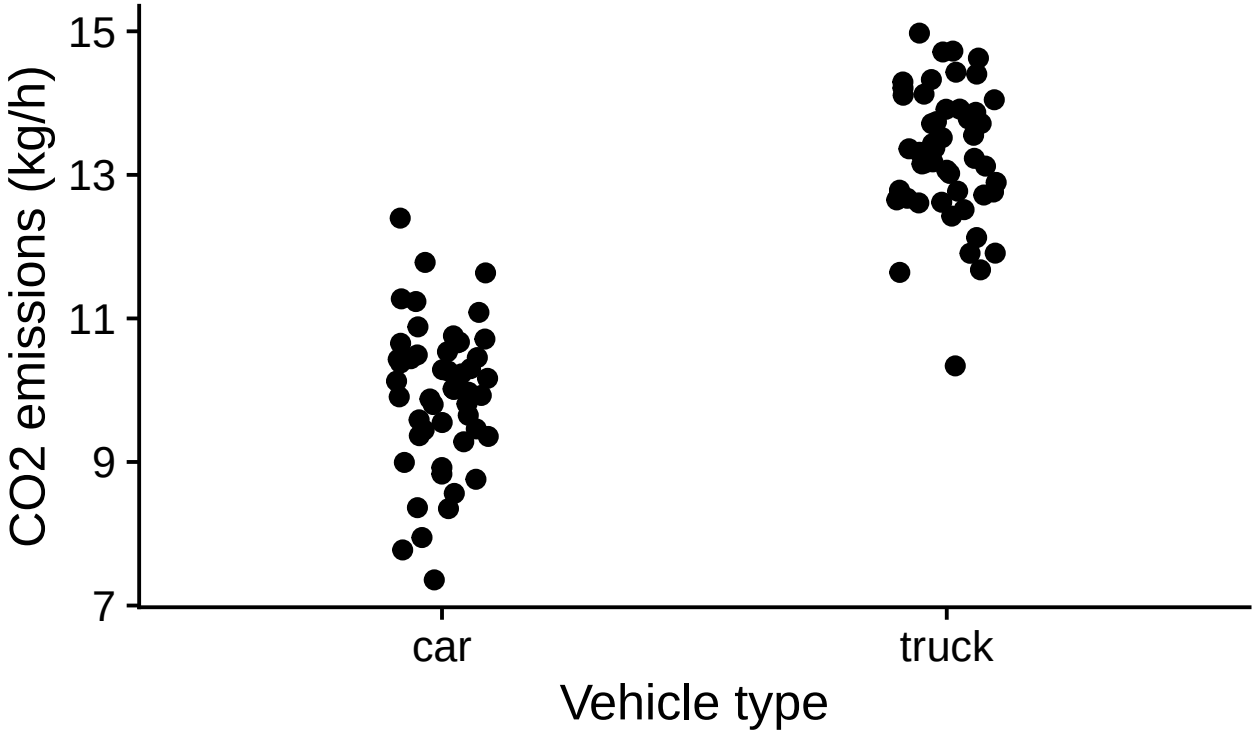
In the context of an entire dataset, we have a vector (column) of values for the variable, predictions and errors:

$$\mathbf{y} = \mathbf{y} + \mathbf{error}$$

2. Parameter estimates of our model in part depend on which random sample we collected in our study. If we repeated our study with a different random set of cars we would probably get a slightly different result. This is an important and general problem with data and one that we will use statistical models to help solve in chapter 3 (Uncertainty).

1.4 Categorical independent variables

In the example above the independent variable (speed) was continuous. But independent variables can also be categorical, for example, we might want to know carbon emission of cars vs trucks.



Here we have two classes of the independent variable that don't fall on a continuous scale. However, we can use the same linear model framework with categorical variables:

$$\mathbf{y} = b_0 + b_1\mathbf{x}$$

but now, each value in the vector **x** equals either a 1 or 0 depending on the class of the independent variable. For example, if the vehicle is a car, x = 0, and if it is a truck x = 1.

This means our predicted CO2 emission for cars would be:

$$\hat{y} = b_0$$

And for trucks would be:

$$\hat{y} = b_0 + b_1$$

Thus the parameters of our linear model have clear interpretations. The b_0 is the expected CO2 emissions for cars, and the b_1 parameter is the expected difference in emissions between trucks and cars. We would expect the b_1 parameter to be positive if trucks have higher emissions than cars.

It might seem arbitrary which class of a categorical variable is set to 0 and which is set to 1, and in fact it is, but either way results in the same interpretation.

Problem 1.2 Prove to yourself that the order of the dummy variables doesn't affect your interpretation of the result. By eye, estimate the parameters of the model for both possible orderings of the dummy variables (Case 1: x = 0 if car, x= 1 if truck; Case 2: x = 1 if car, x = 0 if truck).

You should have found that the magnitude of b_1 is the same in both cases, but that the sign depends on the ordering. In the first case, when x = 0 if the vehicle is a car, then b_1 is the estimated change in CO2 emissions if we switch from a car to a truck (b_1 is positive). And in the second case, (x=1 if the vehicle is a car), b_1 is the estimated change in CO2

emissions of switching from a truck to a car (b_1 is negative). In practice R will automatically assign dummy variables to categorical independent variables (based on alphabetical order). In some cases, specific orderings of the variables are more logical. For example, in an experiment we might want adopt x=0 for the control and x=1 for the treatment. You can override the default ordering of levels in R to change this if desired.

1.5 More than two classes of categorical variables

If we have more than two classes of an independent variable, for example, cars, trucks, and motorcycles, then we need to introduce more dummy variables:

$$\mathbf{y} = b_0 + b_1\mathbf{x_1} + b_2\mathbf{x_2}$$

In this formula, $\mathbf{x_1}$ and $\mathbf{x_2}$ are column vectors of 0's and 1's where the value of each row depends on the type of vehicle. For example, we could set x_1 equal to 1 if the vehicle is a truck and equal to 0 if not. Similarly, we can set x_2 equal to 1 if the vehicle is a motorcycle and 0 if not.

Problem 1.3 Assume that $x_1 = 1$ indicates a vehicle is a truck and $x_1 = 0$ that it is not and $x_2 = 1$ indicates a vehicle is a motorcycle and $x_2 = 0$ that it is not. What is then the interpretation of each of the parameters in the model?

For cars x_1 and x_2 will both equal 0 so the expected CO2 emission will equal b_0. For trucks x_1 = 1 and x_2 = 0 so b_1 is the expected difference between truck emissions and car emissions (which will be positive if trucks emit more CO2 than cars). For motorcycles $x_1 = 0$ and $x_2 = 1$ so b_2 is the expected difference between motorcycle emissions and car emissions (which will be negative if motorcycles emit less CO2 than cars).

1.6 Interpreting parameters

Throughout the course, and later on in your career, you will fit statistical models to data, and then need to interpret and communicate your findings to colleagues, stakeholders, or the public. You often hear the parameters in a linear model being referred to as "effects". For example, in the vehicle emissions example, you might say that the "effect" of vehicle type was 2, or the effect of speed on emissions was 0.1. The word "effect" is tempting to use (and I am sure I will accidentally use it during the course) because it is an easy shorthand for describe the model results. However, the problem with the work "effect" is that it implies the independent variable causally affects the dependent variable. This is related to the idea that correlation is not causation. Just because we find a linear relationship between the dependent variable and the independent variable it does not indicate causation. If you need to be convinced of this statement, I highly recommend you check out the website: https://www.tylervigen.com/spurious-correlations

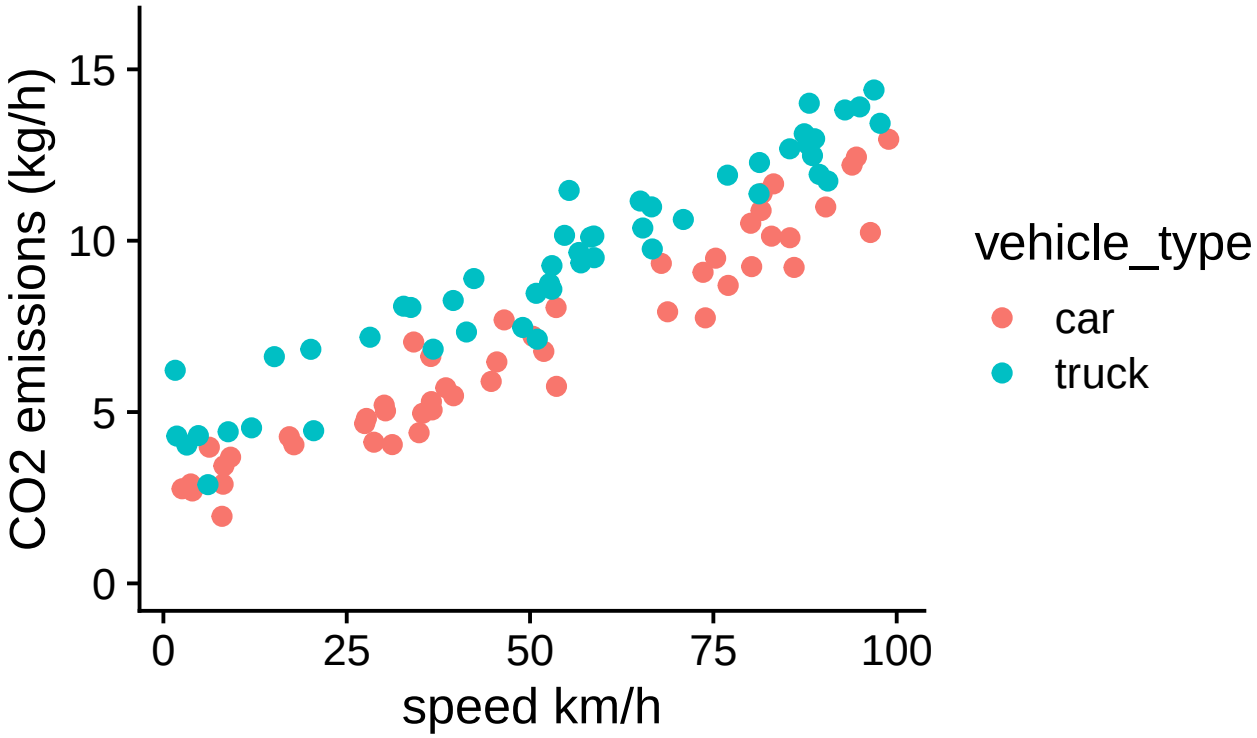
The safest way to interpret parameters in a statistical model is comparisons. For example, in CO2 emissions example, we can interpret the value of the parameter b_1, associated with vehicle type, as the estimated average difference in CO2 emissions of trucks compared to cars, which was roughly 2 kg CO2/hr.. And for b_2, the average difference in CO2 emissions of vehicles of the same type, but differing in speed by 1 km/hr is approximately 0.1 kg/hr. As you can see, it is a bit more awkward to describe model parameters as comparisons rather than effects, and in the case of CO2 emissions, the causal links between vehicle type and speed with CO2 emissions is straightforward, but it many cases this won't be true. For example in many observational studies, if you find an association between two variables, it could be due to a causal link, but it could also be because the independent variable is correlated with another variable that is causally affects the dependent variable. If we use language like the "effect of x" in these cases gives the impression that it we experimentally manipulated x, it should the effect on y predicted by the linear model, which will not always be the case.

The only time it is relatively safe to use causal language to describe parameters in statistical models is in the context of randomized experiments. In this case, if the experiment was designed appropriately, there should be on average no systematic differences between individuals except for the treatment assignment, which allows us to rule out other potential causal mechanisms. You could still observe differences between treatments in an experiment due to chance, but we can use statistical methods to quantify this risk.

1.7 More complex models

The generality of general linear models comes from the fact that we can add as many independent variables to the model as we want.

For example, we could model CO2 emissions as a function of both car type and speed:



Problem 1.4 Think about what type of simple model you could use to model the data that accounts for the effect of vehicle type and speed on carbon emissions. Write down the model as a linear equation and define all independent variables and parameters. By eye, estimate roughly what the parameters of this model should be to fit the data.

Looking at the data, we can see again the clear trend of vehicle type and speed on CO2 emissions. For a given speed, trucks have higher emissions than cars, and for both cars and trucks emissions increase with speed at roughly the same rate. We can account for these two factors in the model:

$$\mathbf{y} = b_0 + b_1\mathbf{x_1} + b_2\mathbf{x_2}$$

where

- x_1 is a dummy variable that equals 0 if the vehicle is a car and 1 if it is a truck
- x_2 is the speed of the vehicle
- b_0 is the expected CO2 emissions for a stationary car (roughly 2 kg /hr)
- b_1 is expected difference in emissions for trucks over cars (roughly 2-3 kg /hr)
- b_2 is the expected difference in CO2 emission for vehicles of the same type that differ in speed by 1 km/h (roughly 0.1 [kg/hr]/[km/h])

Note in this case we assumed that the effect of speed on emissions was the same for cars and trucks. This may not always be the case. Later in the class we will learn how to deal with cases where the effect of one independent variable depends on the value of another independent variable.

For each new independent variable included in our model, we will need a new parameter that relates the value of the independent variable to its effect on the expected value of the dependent variable.

1.8 What makes a linear model "linear"?

Linear models have the general framework:

$$\textbf{prediction} = intercept + parameter_1 * \textbf{variable}_\textbf{1} + ... + parameter_n * \textbf{variable}_\textbf{n}$$

or:

$$\mathbf{y} = b_0 + b_1\mathbf{x_1} + ... + b_n\mathbf{x_n}$$

The model is linear in the sense that the predicted values of the model is a linear combination of the independent variables:

$$\mathbf{y} = b_0 + \sum b_i\mathbf{x_i}$$

Linear in the context of linear statistical models means *linear in the parameters*, such that the change in $\hat{y}$ due to a change in the parameters does not depend on the value of the parameters. Thus if $\frac{d\hat{y}}{db_i}$ does not depend on the value of b_i for all of the parameters then the model can be written down as a linear statistical model.

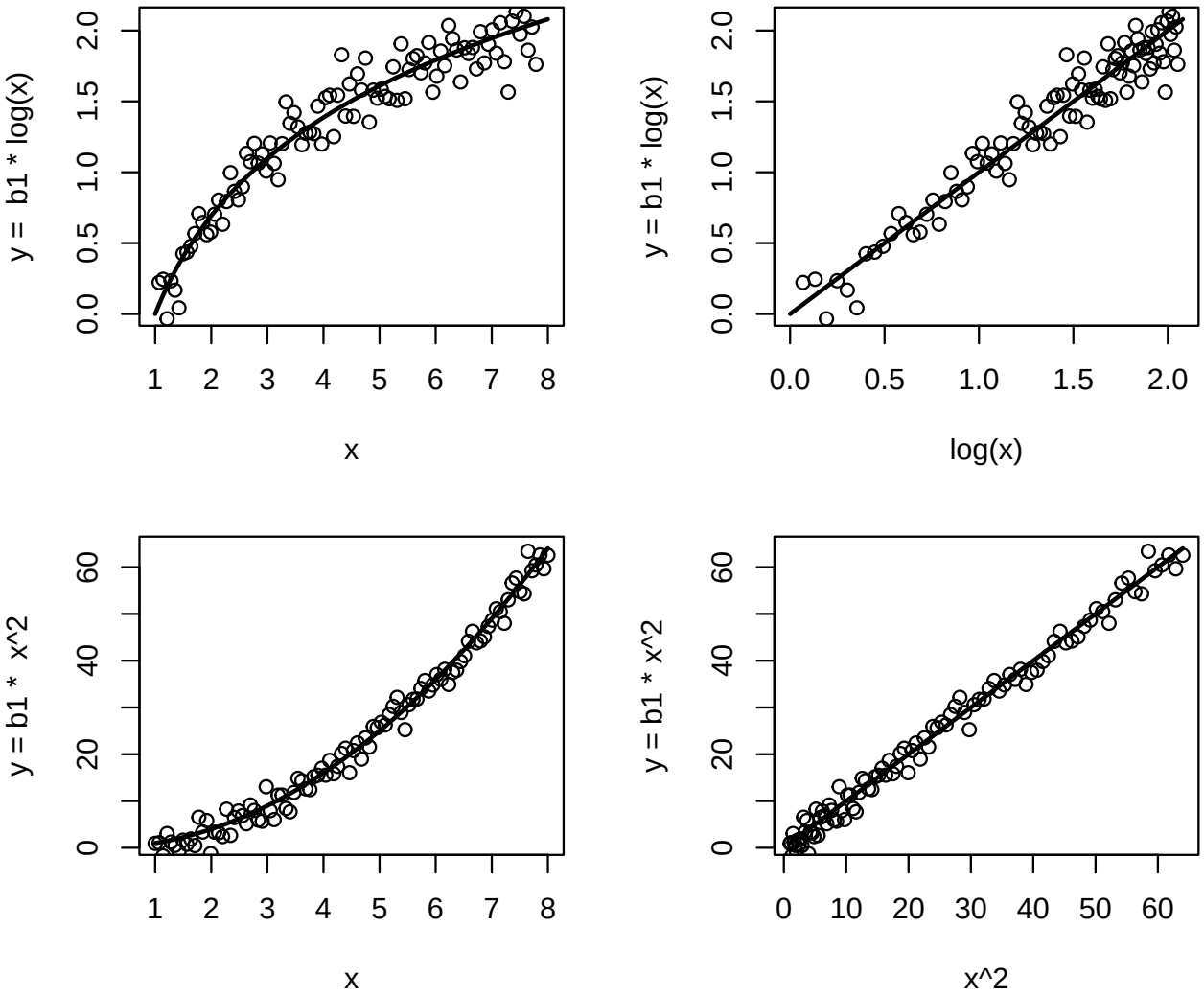
The fact that a given change in the value of a model parameter results in the a same change in $\hat{y}$ regardless of the value of the parameters makes it possible to analytically solve for the parameters that best fit the data. More advanced and computationally expensive methods are needed to fit non-linear to data, and in general, there is no guarantee that we will find the parameters that best fit the data. Fortunately, linear models are not quite as restrictive as you might think. This is because we can transform or combine independent variables in whatever what we see fit and still have a 'linear' model. For example, we could log transform an independent variable:

$$\hat{y} = b_0 + b_1 \log(x)$$

or square an independent variable:

$$\hat{y} = b_0 + b_1 x^2$$

and still have a linear model. We can think of $\log(x)$ or x^2 as new independent variables, and the predicted value of y is just a linear combination of the new independent variables (right panels). In this way, linear models can fit data that don't look linear (left panels).



Examples of *non-linear* models would be

$$\hat{y} = b_0 + x^{b_1}$$

or

$$\hat{y} = x_1/(b + x_1)$$

An easy way to tell if a term in a model is linear is if you can arrange the term so that you have the parameter times any combination or transformation of independent variables. For example the term:

$$\frac{b_0 x_1}{x_2 + x_1}$$

is linear because we can put parentheses around everything but the parameter:

$$b_0 \left(\frac{x_1}{x_2 + x_1} \right)$$

We can think of everything inside the parentheses as a new independent variable:

$$b_0 x_{new}$$

where

$$x_{new} = \left(\frac{x_1}{x_2 + x_1} \right)$$

Problem 1.5: Which of the following models can be made linear by substitutions? The "b"s are parameters and "x"s are independent variables.

1. $\widehat{y} = bx^2$
2. $\widehat{y} = b \sin(x)$
3. $\widehat{y} = \sin(bx)$
4. $\widehat{y} = b_0 \sin(b_1 x)$
5. $\widehat{y} = bx_1 x_2$
6. $\widehat{y} = bx_1^{x_2}$
7. $\widehat{y} = b \log(x_1 x_2)$

Answers: 1 Yes, 2 Yes, 3 No, 4 No, 5 Yes, 6 Yes, 7 Yes

1.9 General linear models

As introduced above, we will be using linear models to model the relationship between independent variables and dependent variables. More specifically, we will be working with a class of statistical models referred to as **General Linear Models**. In the equation:

data = model + error

General linear models models, assume that the "model" term is composed of linear relationships between independent and dependent variables as described above, but it also assumes that the errors are normally distributed. More specifically it assumes that

error ~ N(0,σ)

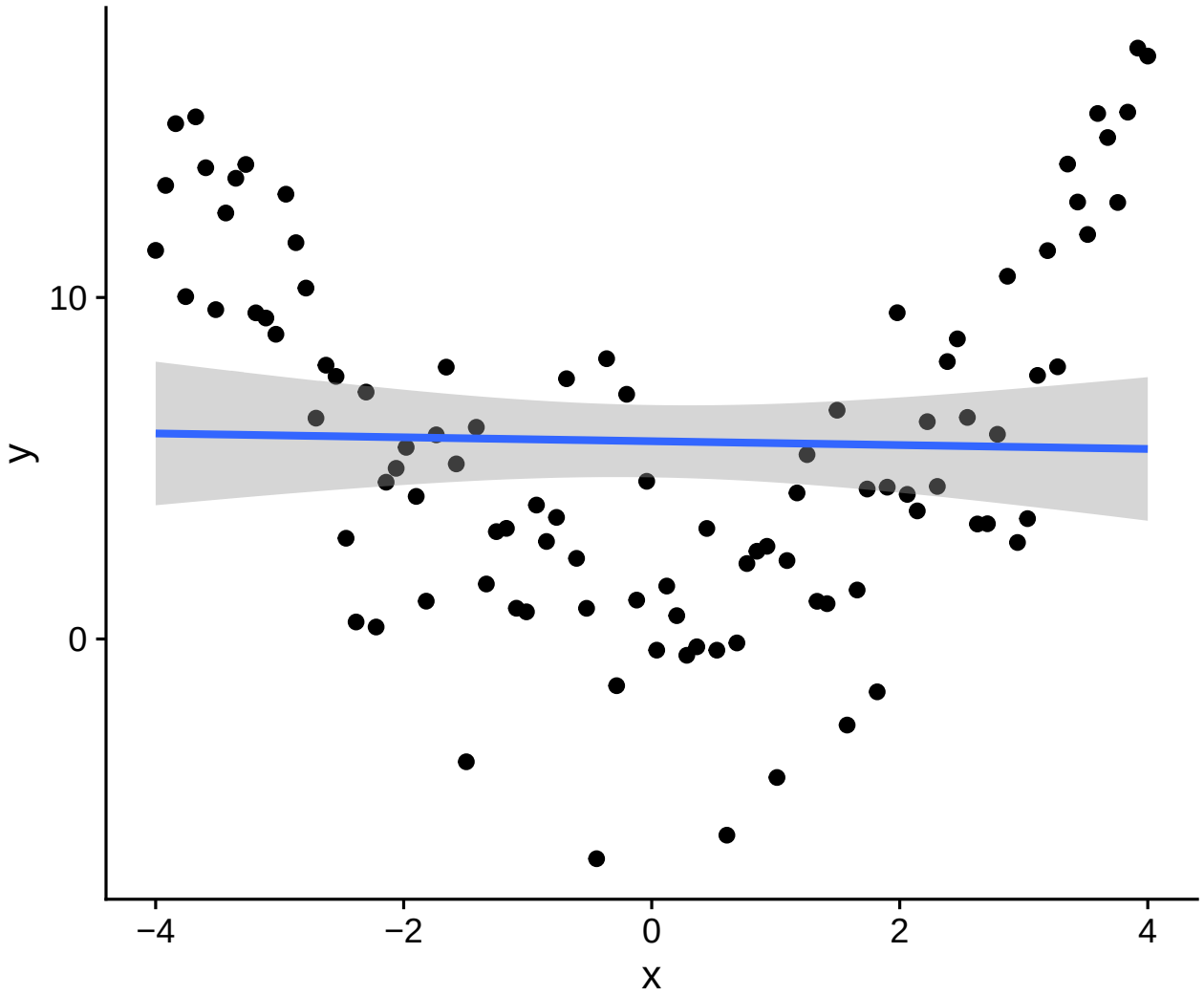
which means that the errors are drawn from a normal distribution with a mean equal to zero and a standard deviation equal to sigma. In many cases when the dependent variable is continuous (e.g. carbon emissions, blood pressure, height, weight, precipitation, ect.), a normal distribution is a decent approximation for the distribution of the errors. In Chapter 3, we discover why normal distributions tend to show up so frequently with continuous data.

1.10 Assumptions of General Linear Models

There are 4 main assumptions when we use general linear models.

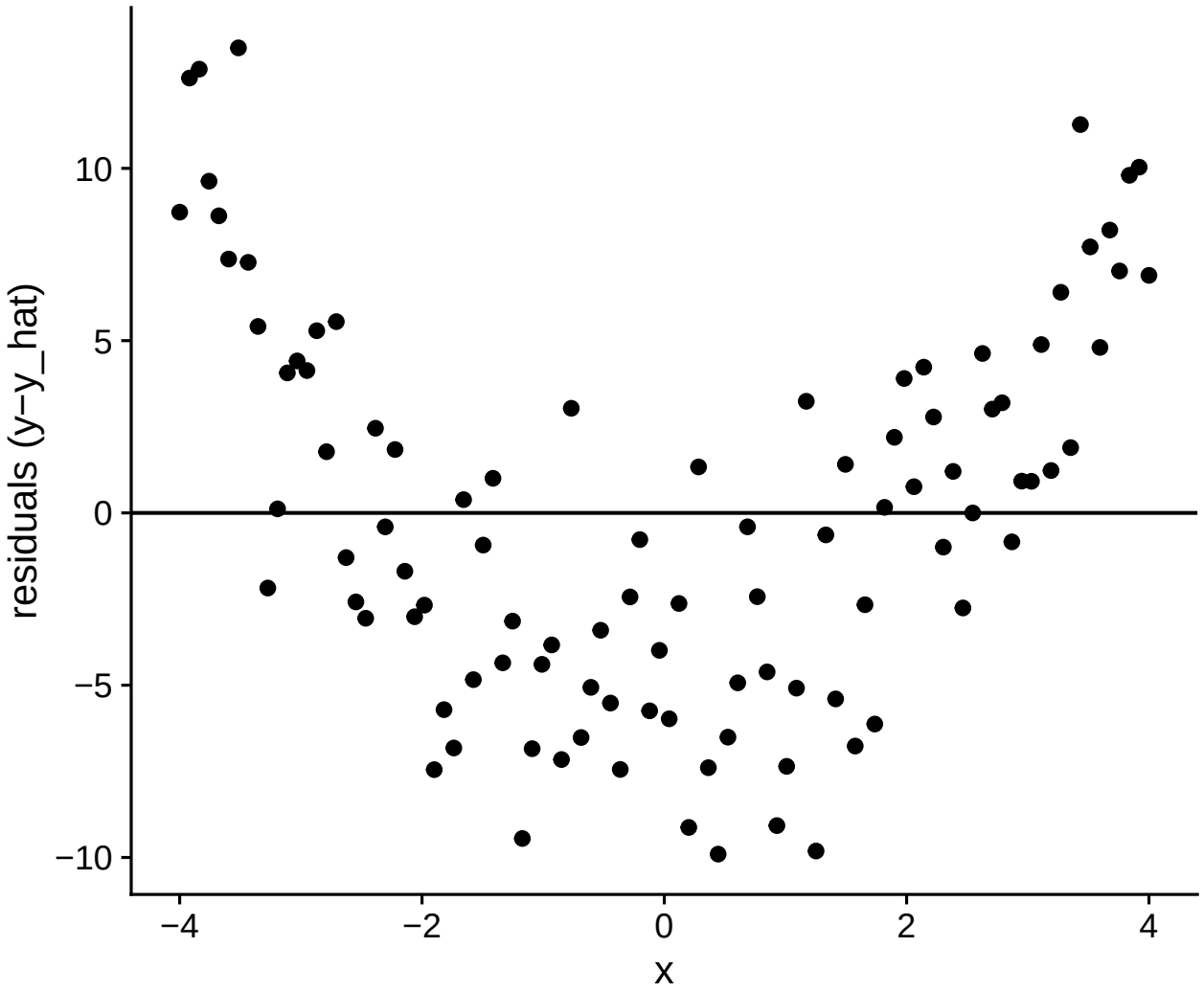
1.10.1 Linearity

General linear models assume that the relationship between dependent and independent variables is linear. For example, in the plot below, the relationship between x and y is not linear, and thus when we fit a linear model, the predictions of the model are systematically biased in a way that depends on the value of x.



In this particular case, we might estimate the slope of the linear relationship to be very close to zero. In a way, this inference is still correct; however, at the same time, this statistical model doesn't really capture the relationship between x and y. By fitting a more complex model, we can see that there is a relationship between x and y, it is just not linear.

Extreme violations of the linearity assumption can often be revealed by plotting the model prediction along with the data. If the residuals, or differences between observations and predictions, differ systematically as a function of the independent variable, this is a good sign that the wrong model is being used for the data at hand.

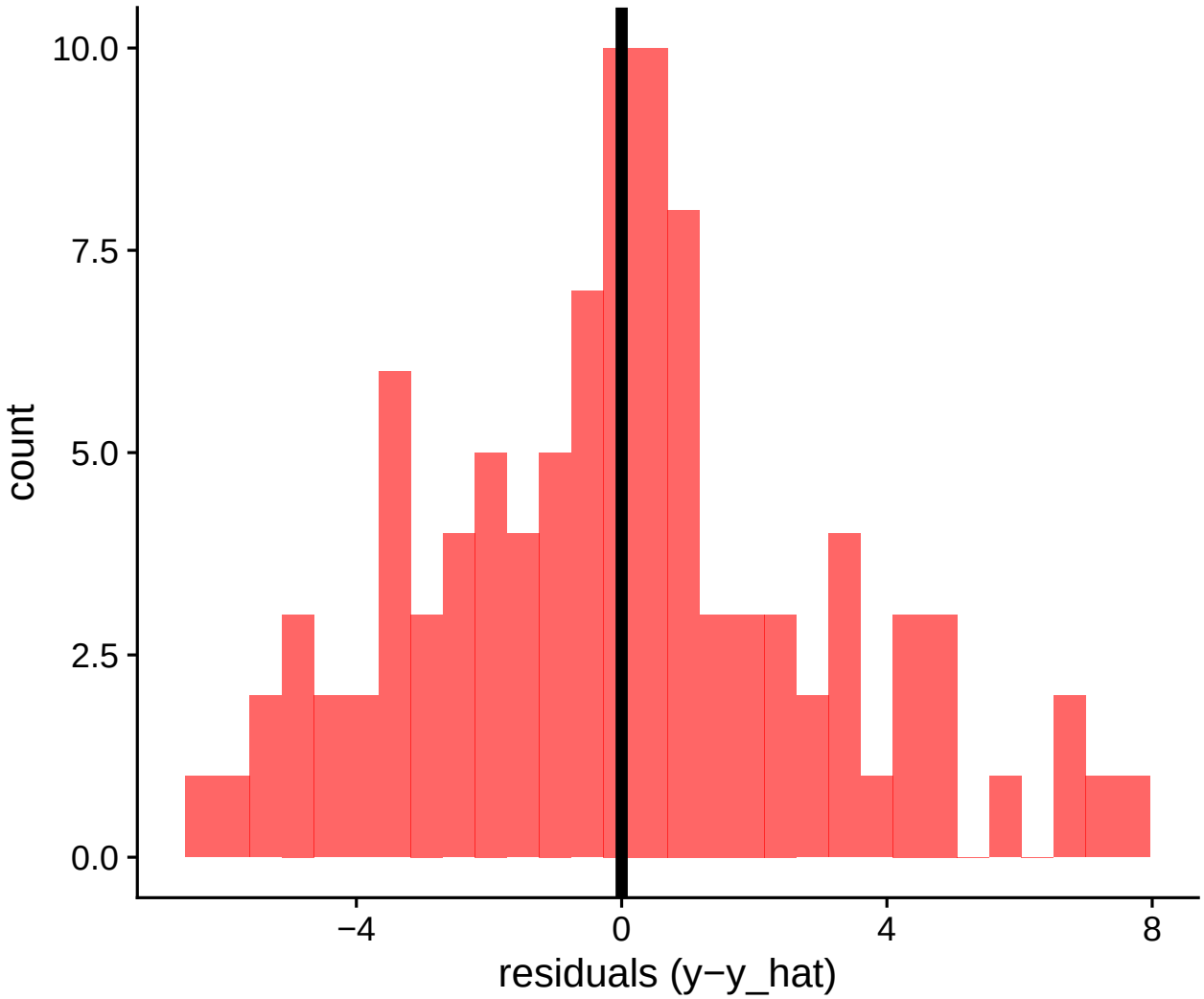


Later in the course, we will discuss how more complex (but still linear) models or data transformations can be used to avoid substantial violations of the linearity assumption.

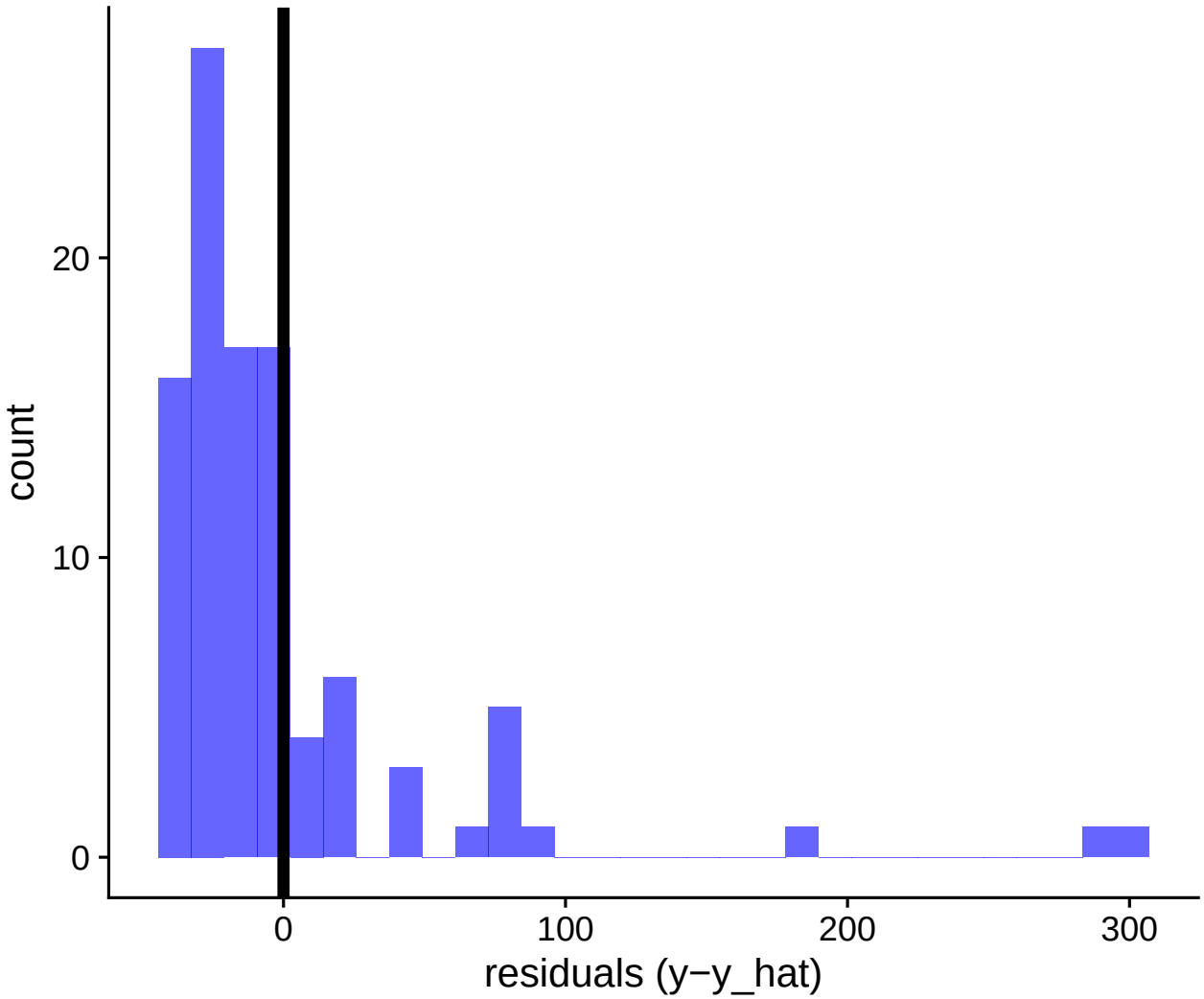
1.10.2 Normality

General Linear Models assume that errors (or residuals) are normally distributed. Thus if you were to plot a histogram of the model residuals, you would expect to see a distribution that is centered around zero, and roughly symmetric:

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



In other cases, our residuals might look heavily skewed:



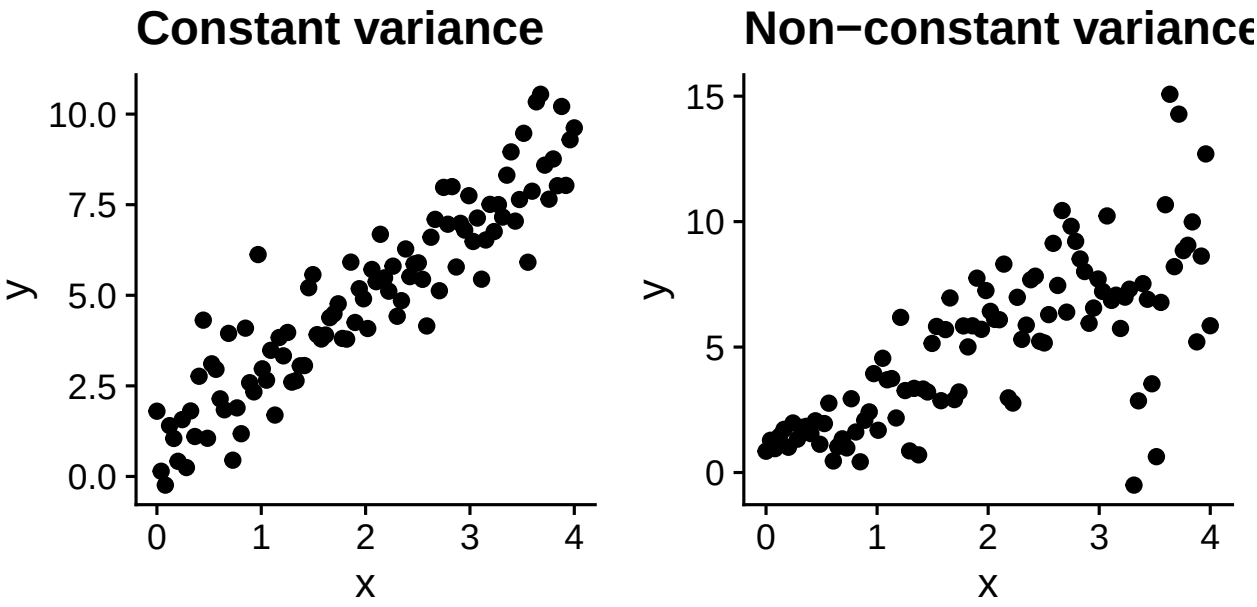
As we will see later in the course, even when residuals are clearly not normally distributed like in the example above, the statistical inferences are often still valid. However the main problem with the example above is that the means don't really reflect an "average" or typical value because most observations have values below the mean. In cases like this, sometimes transforming the data, for example, using log(y) rather than y as the dependent variable allows the mean to better capture the central tendency of the data.

1.10.3 Constant variance

As stated earlier, general linear models assume errors are drawn from a normal distribution with a mean of zero and a standard deviation of sigma, where σ is a specific value for a given dataset:

error ~ N(0,σ)

Because σ is a fixed value, this means that our model assumes that the variance doesn't systematically change with as a function of the independent or dependent variables. In the figure below, on the left is an example where the variance appear to be constant across the range of x values, and on the right is an example, where the variance appears to increase with x. Thus in the figure on the right, the constant variance assumption is not met. Again, like for the normality assumption, it turns out that the results from general linear models are fairly robust to violations of this assumption.



1.10.4 Independence

General linear models assume that errors are independent of each other. **This is by far the most important assumption, and even small violations can have important implications.**

Consider an example where a pollster randomly samples 1000 households to gauge political preferences. However, to increase their sample size, the pollster decided to not only poll the randomly selected individual, but also their partner. As a result the pollster end up with nearly double their originally planned sample size, and thus was able to get more precise statistical estimates of political preferences.

Can you think of a problem here?

The problem is that the statistical model is based on the assumption that we have 2000 independent data points, but in all likelihood, individuals are likely to share the political preferences of their partners, and thus the political preference of one individual is not independent of their partner's preferences. In the most extreme case, if people only partnered with individuals with identical political preferences, then there is no new information gained by asking partners about political preferences. In this case, even though the pollster asked 2000 individuals, they effectively only have 1000 true observations, and thus their analysis based on 2000 individuals will drastically overestimate our certainty about estimates of political preferences.

Non-independence of errors shows up in many contexts. For example observations might be correlated because they were taken from locations that are close in space or time, or observations taken from the same individual multiple times. In some cases non-independence of errors is inevitable, and thus we have to use more advanced statistical approaches that account for dependencies (for example, time-series analysis or spatial statistics). However, when possible it is best to design experiments of field studies to avoid dependent errors through randomization.

1.11 What you should know

1. The general structure of linear models, including distinguishing parameters, independent variables, and dependent variables

2. Be able to write general linear models to describe simple data sets with both continuous and categorical independent variables, and interpret the meaning of each parameter and variable in the model.

3. Be able to roughly estimate parameter values of simple linear models by looking at a figure (e.g. problem 1.1 and 1.4)

4. Be able to tell whether a statistical model is linear (e.g. problem 1.5)

5. The assumptions of general linear models

2 Fitting models to data

2.1 Recap

In Chapter 1 we introduced the general structure of a linear model. We will use linear models for the entire class to describe and communicate patterns in data, estimate how certain we are about the estimated relationships among variables, test hypotheses, and make predictions.

The first step of our analysis begins with deciding which variable we want to model (what is the dependent or response variable), and what independent variables we think the dependent variable depends on. Typically this is something you should decide before you start your study whether you are doing an experiment or collecting observational data. For example, if we were interested in the effect of nitrogen on plant growth we might conduct an experiment where we measure the growth rate of plants at different soil N levels. In this case, it is clear that the variable we are interested in modeling is plant growth, and we want to model it as a function of N:

$$growth = f(N)$$

The $f(N)$ means that plant growth is a function of N. In theory, we could use any equation or function but in this class we will focus on general linear models

Problem 2.1: Write down a linear model to predict plant growth as a function of N. Describe each parameter in the model:

Answer:

- $\mathbf{y} = b_0 + b_1\mathbf{N}$
- b_0: expected plant growth when N = 0
- b_1: the expected increase in plant growth per unit increase in N

2.2 Fitting models to data

After we have specified the model, the next step is to fit the model to the data. This means finding the parameter values that most accurately make the model match the data. We saw in chapter 1, that for simple models, you can get a rough estimate of parameters just by looking at a plot of the data. But ideally, we want a more precise and systematic way of fitting models to data. This is even more important for complex models with multiple variables for which visualizing all the relationships among variables on a 2-dimensional computer screen is difficult.

To fit a model to data we first need to quantitatively define what it means for a model to match data. In other words, we need to put a number on how well a model fits data. Once we have a quantitative metric for model fit, the next step is to search for the parameter values that optimize this metric.

An intuitive way of thinking about how well a model fits data is how closely the predictions of the model match the observations. The difference between observations and predictions are called errors residuals. Models that fit the data well should have small errors. It is straightforward to calculate the error for each observation, but then the question is how do we aggregate the errors for all of our data to come up with one number that quantifies how well our model matches the data.

The most intuitive thing to do would be to calculate the error for each observation and sum them. For example let's consider a data set with just 3 observations $\mathbf{y} = [4, 5, 6]$. Here we have no independent variable, so we just want to find the parameter b_0 that best fits the data:

$$\mathbf{y} = b_0$$

If you wanted to pick a value for b_0 to closely match the data, 5 would be a good choice:

Observation	Prediction	Residual
4	5	-1
5	5	0
6	5	1
sum		0

We can see that for the first and 3rd observation, the model is off by 1. However, one residual is positive and one is negative so the sum of all the residuals is 0. Thus using the sum of all the residuals as a metric of goodness of fit has a problem in that even though our model doesn't fit the data perfectly, the sum of the errors equals zero. This is particularly problematic if we want to compare how well models fit different data sets. For example, if we had a second data set $\mathbf{y} = [0, 5, 10]$, with $b_0 = 5$, the sum of the residuals again equals 0, in spite of the fact that the typical distance between observation and prediction is much higher:

Observation	Prediction	Residual
0	5	-5
5	5	0
10	5	5
sum		0

One way around this problem is to simply square the residuals and then sum them. If we do this for both example data sets we get the following:

Observation	Prediction	Residual	Residual^2
0	5	-1	1
5	5	0	0
10	5	1	1
sum		0	2

Observation	Prediction	Residual	Residual^2
0	5	-5	25
5	5	0	0
10	5	5	25
sum		0	50

We see that the sum of the squared residuals, or SS_{Error} is lower for the first dataset, which is consistent with the fact that the predictions of the model are more accurate for the first dataset.

Problem 2.2: In the example above, we squared the residuals so that they don't cancel each other out when we sum across observations. But couldn't we have done other transformations to avoid this problem? Think of two other ways of quantifying errors to avoid negative and positive residuals don't cancel each other out.

One simple way to keep positive and negative residuals from canceling each other out would be to sum the absolute value of the residuals. Alternatively we could raise the residuals to the 4th power. These different ways of counting errors have consequences for how influential specific observations are. For example we when square errors, large errors have a disproportionate influence on the goodness of fit compared to smaller errors, while if we just used the absolute value of residuals all residuals contribute to the total in proportion to their size.

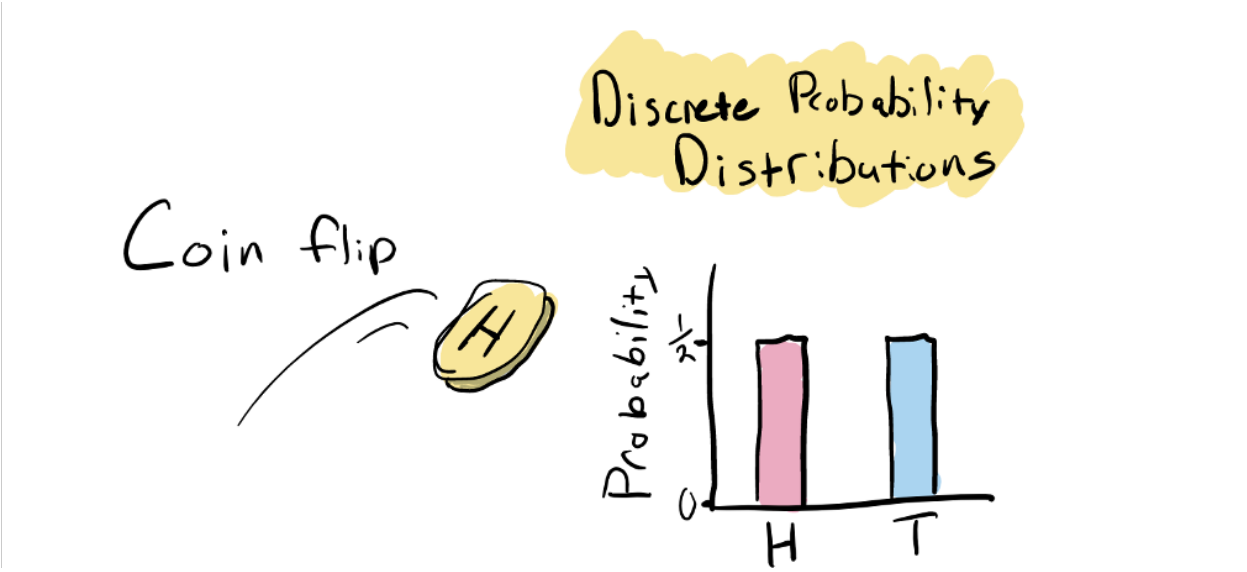
So there are many criteria we could use to quantify the goodness of fit, but in practice, when our dependent variable is continuous and we assume that errors are normally distributed, we typically use the SS_{Error}. The method of fitting

statistical models to data by finding the parameters that minimize SS_{Error} is often referred to as **Ordinary Least Squares** or OLS. To understand why we primarily use Ordinary Least Squares and not, for example, Ordinary Least Absolute Values, we need to understand a little bit about probability and likelihoods.

2.3 Discrete probability distributions

A deeper and more fundamental goal of fitting models to data, is to find the parameters of the model that make the data most likely. To be able to understand this, we need to review some basics about probability.

Probability is easiest to think about when we have a discrete number of outcomes. For example, in a coin flip there are two outcomes: heads or tails. If the coin is a fair coin, each outcome has equal probability:

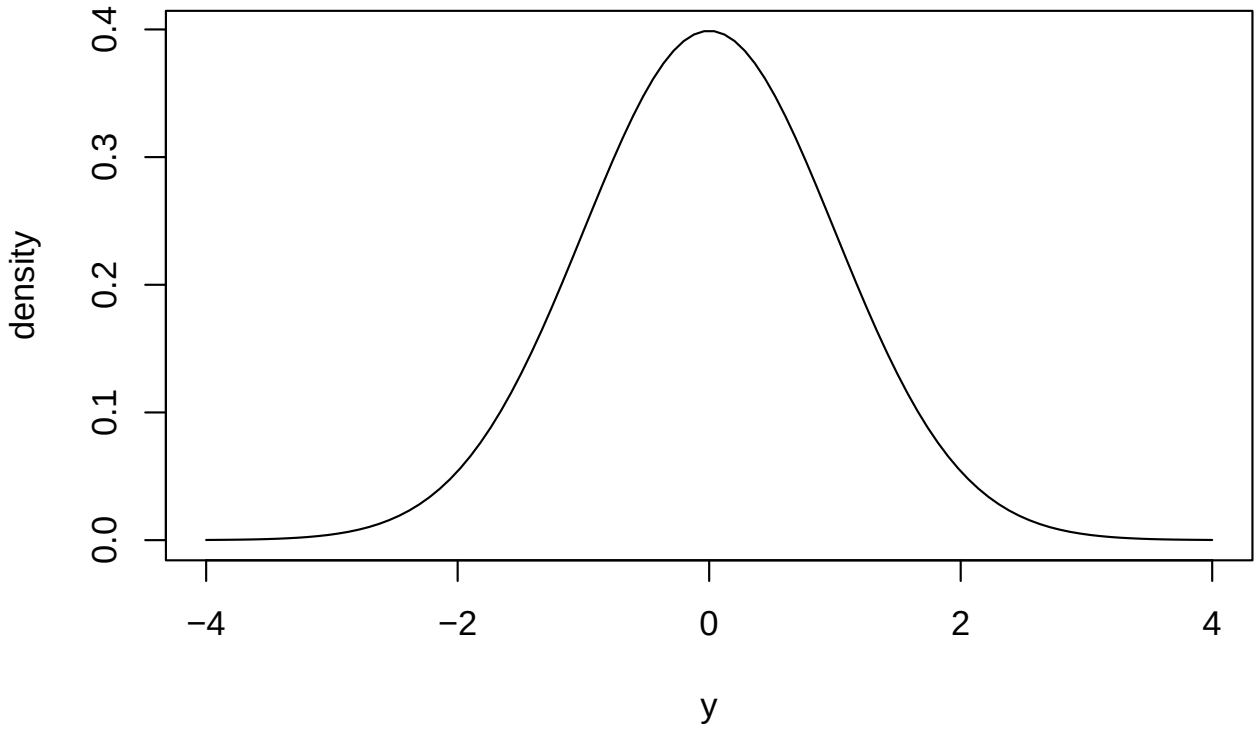


A probability distribution is simply a function that describes the probability of all possible outcomes. In the case of a fair coin there are just two outcomes, each with a probability of 0.5. An important point about all probability distribution functions is that the sum of the probabilities for all possible outcomes equals one (e.g. 0.5 (probability of heads) + 0.5 (tails) = 1). Other examples of a discrete probability distributions would be the probabilities of throwing rock, papers, or scissors in a game of rock-paper-scissors shoot, or the probabilities of rolling 1-6 on a 6-sided dice.

2.4 Probability density functions

The examples above are discrete probability distributions because there are a finite set of outcomes the observations can take. For continuous data, such as height, weight, temperature, etc., there are, in theory, an infinite number of values these measurements can take. For example, someone's height could be 170.1 cm, or 170.01 or 17.00000001 cm (although in practice we generally only measure down to some level of precision). ***As a result the probability of observing exactly any specific value is zero!***

For continuous variables, it doesn't make sense to talk about probabilities of specific outcomes because the probability of any outcome is zero. Instead, for continuous variables we use **probability densities**. A probability density function is a function that defines the probability density for a particular value of a random variable. A very important probability density function for this course is the normal distribution (AKA Gaussian distribution), which is defined by its mean and standard deviation, and looks like the following:



The plot above shows a probability density function of a normal distribution with mean = 0 and standard deviation = 1. The probability density for y is shown on the y-axis. Intuitively, you can look at this plot and see the probability density is highest around the mean (0) and decreases for higher or lower values. You can think of probability densities as the relative likelihood of observing specific values. In the example above, the probability density at $y = 0$ is roughly 0.4, and at 1 is roughly 0.2, thus values of y very close to 0 are twice as likely to be observed as values very close to 1.

More formally, the probability density is defined as the probability of observing a value of x between x and $x + \Delta$ as Δ approaches zero. While the absolute probability of observing any specific value of a continuous variable is zero, the relative likelihood of observing values close to a specific value are proportional to their probability density. **Importantly, the integral of a probability density function equals 1, and the probability of observing a value of x within a specific range (e.g. between 1 and 2) is equal to the area under the curve in this interval.** This will be important later in the course.

The equation that gives the probability density for the normal distribution is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

For now I just want you to note the parameters in the equation μ, the mean or predicted value of x, and the standard deviation, σ. Also notice that squared errors $(x - \mu)^2$ already show up in the normal probability density function, which already gives a clue about why we work with squared residuals in OLS. You can think of μ as determining where the distribution is centered, and σ determining the width of the distribution (when sigma is large the distribution is wide). If these parameters μ and σ are known, then we can calculate the probability density for an observation, y. In R you can calculate the density function of the normal distribution using the `dnorm()` function (the "d" stands for density, and "norm" for normal distribution). The syntax of the function is:

```
dnorm(y, mean = 0, sd = 1)
```

Importantly, we can use this probability density function to calculate the relative likelihood of a particular observation. Where y, is an observation and the mean = $\hat{y}$ or the predicted value for that data point, and sd is the standard deviation, σ, of the normal distribution.

Problem 2.3 A statistical model predicts $\hat{y} = 5$, with a $\sigma = 2$, but the observed value of y was 6.2. What is the probability density of this observation given the model?

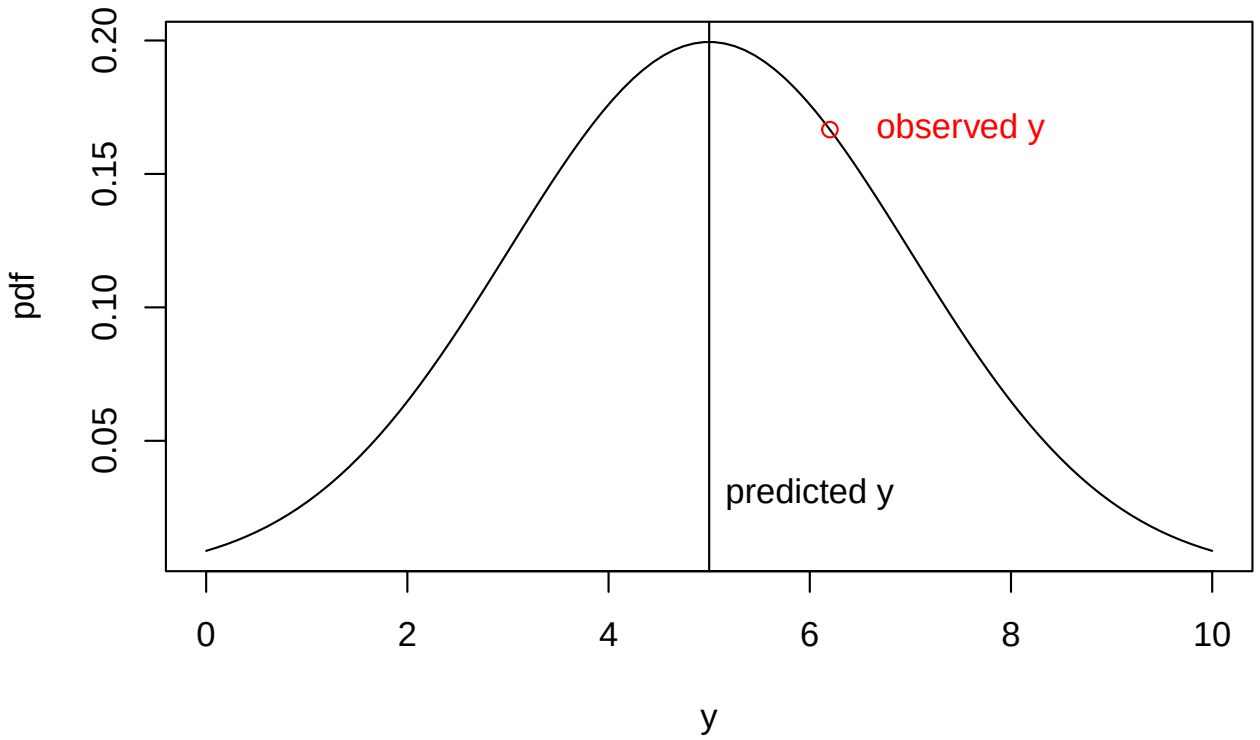
We can answer this question using the dnorm function

```
obs = 6.2
pred = 5
sigma = 2
pd =dnorm(obs,pred,sigma)
pd
```

```
[1] 0.1666123
```

We can inspect this answer visually, by plotting out the normal probability density function for $\mu = 5, \sigma = 2$ for a range of values and then plotting out the pd for $y = 6.2$ (red point):

```
y<-seq(0,10,length.out=100) #range of x values
pred = 5
sigma = 2
pdf =dnorm(y,pred,sigma)
plot(y,pdf,'l')
abline(v=5)
text(6, .03, "predicted y")
text(7.5, pd, "observed y",col='red')
points(6.2,pd,col='red')
```



In the example above, we only have one data point. If we wanted to find the parameters that maximized the likelihood of this data point, where the model is:

$$\hat{y} = b_0$$

then it is clear that b_0 should equal the value of the data point: y. However in real applications, we have more than one observation. The question then is how do we maximize the likelihood of the data when there are multiple observations? To answer this question, we need to understand joint probabilities.

2.5 Joint probability

Problem 2.4 If you roll a 6-sided dice two times, what is the probability of rolling two 6's?

The probability of two independent events occurring is equal to the product of their individual probabilities. The probability of rolling a 6 is 1/6 and thus the probability of rolling two sixes is 1/6 * 1/6 = 1/36.

The key point in the example above is that **when observations are independent their joint probability is the product of probabilities of each observation**. Thus if the errors are independent, the likelihood of a dataset given a particular set of parameters is equal to the product of the probability density function for each observation. As mentioned in Chapter 1, independent errors is an important assumption of general linear models, and here we can partially see why: our calculation of the likelihood of the data given a parameter set assumes each error is independent.

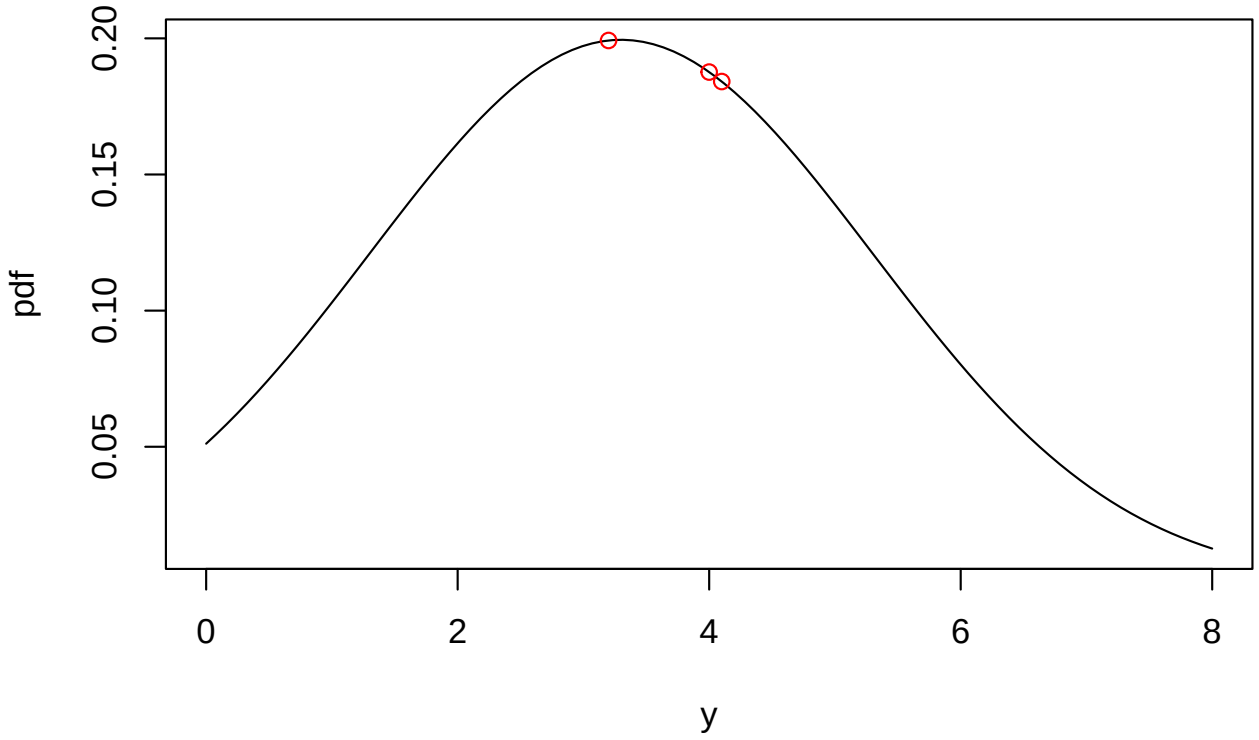
Problem 2.5 What is the likelihood of observing the dataset **y** = [3.2, 4.1, 4.0] assuming y is a normally distributed random variable with mean = 3.3 and σ = 2?

We can calculate the likelihood of the data set given the model parameters by taking the product of the probability density for each observation:

```
b0=3.3
sigma=2
dnorm(3.2,b0,sigma) * dnorm(4.1,b0,sigma) * dnorm(4.0,b0,sigma)
```

```
[1] 0.006882611
```

We can visualize this by looking at the probability density function (often shortened to *pdf*) for the parameters $b_0 = 3.3$ and $\sigma = 2$. Each of the red points corresponds to the probability density for the 3 observations, and the likelihood of the data given the parameters is given by the product of the tree probability densities.

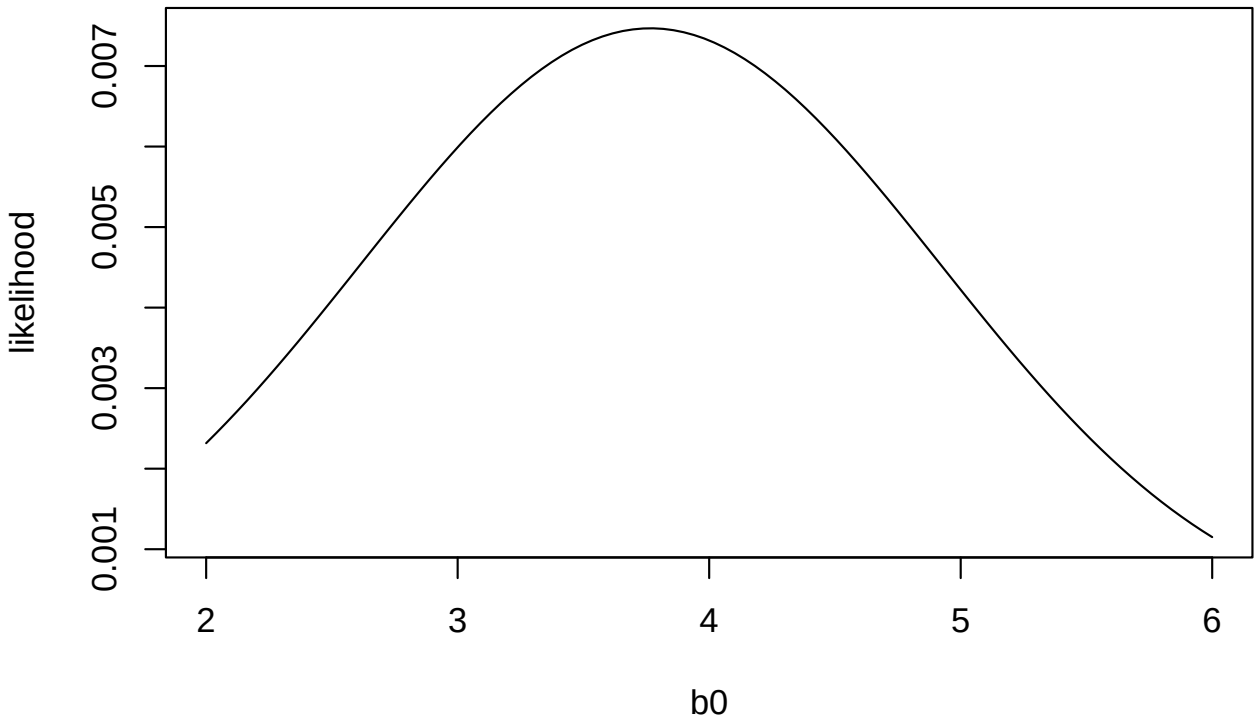


2.6 Maximum Likelihood Estimation

Problem 2.6 Can you find a value of b_0 that makes the observed data more likely? What is the value of b_0 that maximizes the likelihood of the data?

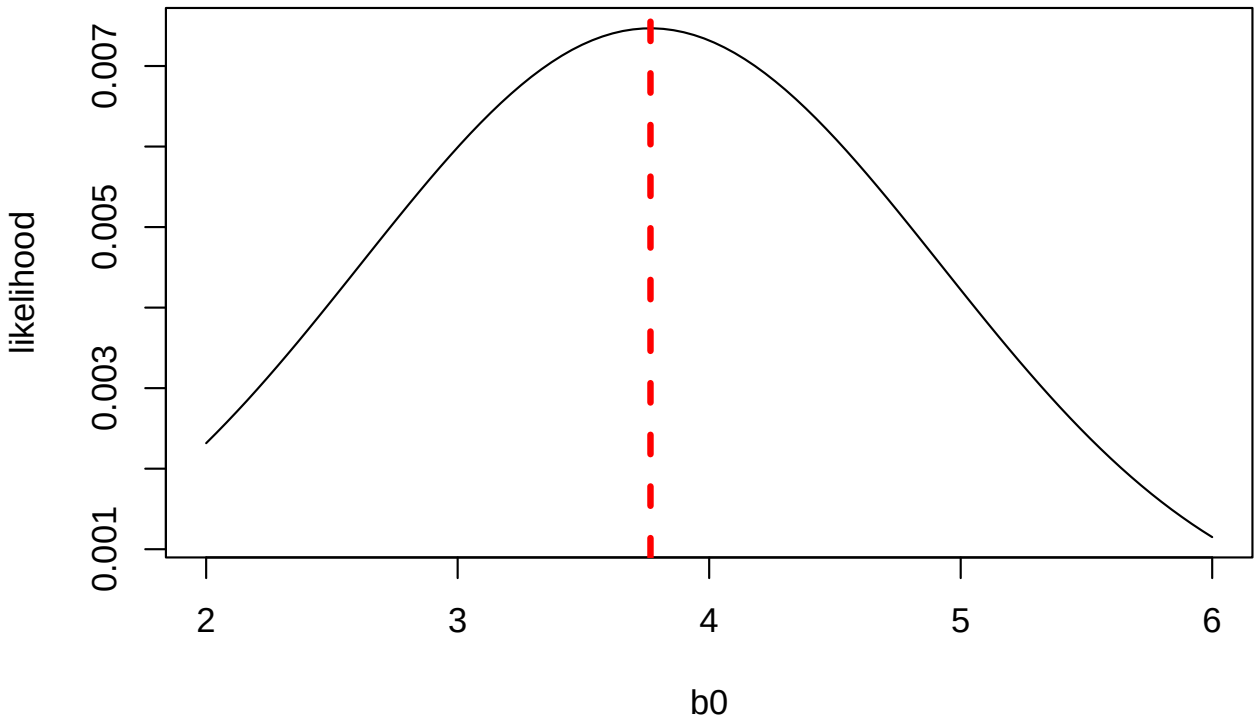
One way to answer this question is by brute force: we just try out a range of values for b_0 and see which one results in the highest likelihood. We can do this with a `for` loop.

```
# Create vector of length 100 with b0 values between 2 and 6
b0=seq(2,6,length.out=100)
# Create vector of equal length with NA values (NA = Not Available / Missing Value)
likelihood = NA * b0
sigma=2
for (i in 1:length(b0)){
likelihood[i] =  dnorm(3.2,b0[i],sigma) * dnorm(4.1,b0[i],sigma) * dnorm(4.0,b0[i],sigma)
}
plot(b0,likelihood,'l')
```



The plot above shows the likelihood of the data for different values of b_0. The value of b_0 that maximizes the likelihood of the data is called the **Maximum Likelihood Estimate**. We can see that the likelihood of the data is maximized when b_0 is less than, but close to 4. Your intuition would probably tell you that the value of b_0 that best matches the data would be equal to the mean of y. And if we compare the mean of y to the likelihood function, we see in fact the maximum likelihood estimate of b_0 is the mean of y:

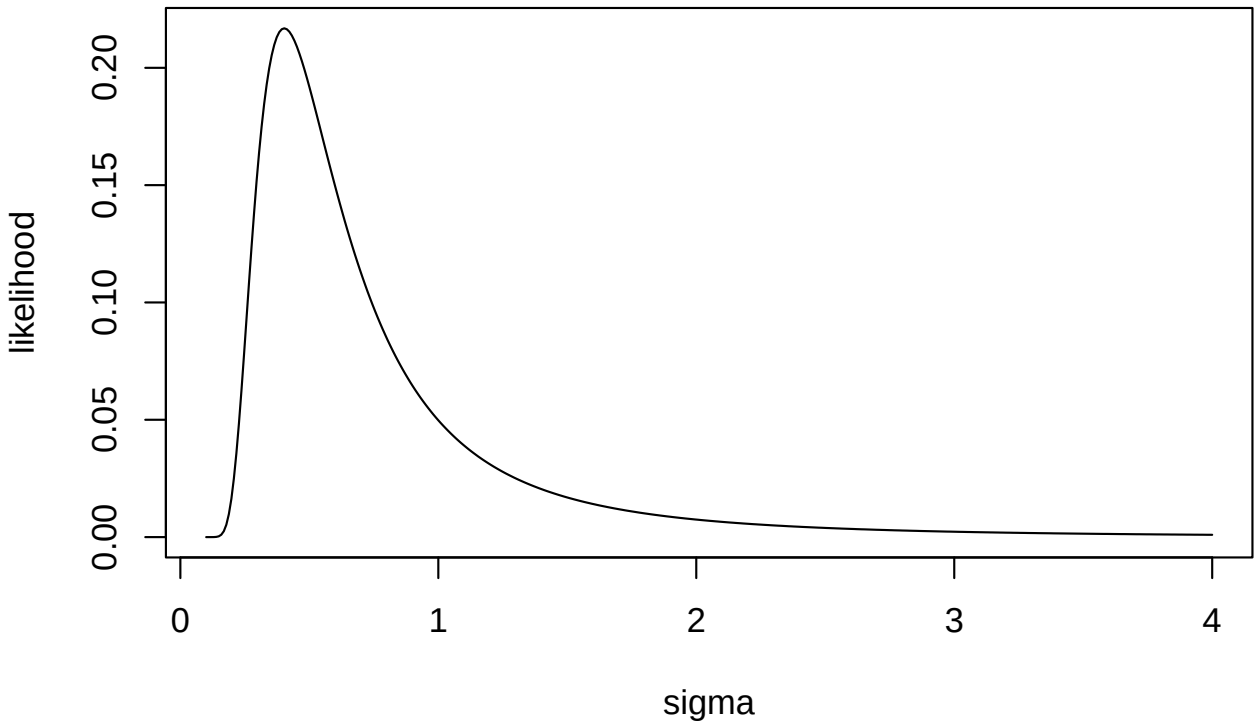
```
plot(b0,likelihood,'l')
abline(v = mean(c(3.2,4.1,4.0)), col="red", lwd=3, lty=2)
```



Problem 2.7 So far we have assumed $\sigma = 2$, but in general, σ isn't known, and has to be estimated from the data. Given that $b_0 = \text{mean}(y)$, what is the value of sigma that maximizes the likelihood of the data?

We can again use brute force to estimate sigma, with b_0 set to the mean of y.

```
b0=mean(c(3.2,4.1,4.0))
sigma=seq(0.1,4,length.out=400)
likelihood = NA * sigma # in
for (i in 1:length(sigma)){
likelihood[i] =  dnorm(3.2,b0,sigma[i]) * dnorm(4.1,b0,sigma[i]) * dnorm(4.0,b0,sigma[i])
}
plot(sigma,likelihood,'l')
```




```
sigma[which.max(likelihood)]
```

```
[1] 0.4030075
```

We see that the likelihood of the data is maximized at an intermediate value of σ. If σ is too small the pdf is very narrow, and observations far from the mean are very unlikely, but if σ is too wide, then the pdf gets spread across a wide range of possible values, and the overall likelihood is low. It turns out that the σ that maximizes the likelihood is equal to the square root of the sample variance,s:

$$s^2 = \frac{\sum_{n=i}^{n}(y_i - \hat{y}_i)}{n}$$

where n is the number of data points.

The estimate of σ that maximizes the likelihood of the data is just the square root of the the sample variance, s. Thus, s is the maximum likelihood estimate of the true population σ.

However you may notice that, you get a slightly different answer if you just calculate the standard deviation of the data, in R using the function `sd()`. This is because the maximum likelihood estimate of σ is slightly biased. It will on average underestimate the true population standard deviation. This is because if there is one parameter in the model and 3 data points, there are only 2 degrees of freedom to estimate the standard deviation. For example if we had only one data point, the mean would equal the value of the one data point, and the estimate of the variance would equal zero, when in fact we don't have enough information to estimate variance from a single data point. Thus the unbiased estimate of the population variance is:

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

However for large datasets (large n) you basically get the same answer.

The important take-home message here is that by taking the product of the probability densities for all observations from the probability density function we can calculate the likelihood of the data for a given parameter set. When we fit our model to data, the goal is to find the parameter set that maximizes the likelihood of observing the data.

The parameter set that maximizes the likelihood function is called the maximum likelihood estimate.

2.6.1 Ordinary Least Squares and Maximum Likelihood give the same answers when errors are normally distributed

Below, I will show that if errors are normally distributed, then the parameters that maximize the likelihood function are the same as those that minimize the sum of squared errors. So again, the likelihood of observing a dataset, given a model with parameters θ is calculated by taking the produce of the probability densities of the individual observations:

$$L(y|\theta) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y_1 - \hat{y}_1}{\sigma}\right)^2} \times ... \times \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y_n - \hat{y}_n}{\sigma}\right)^2}$$

You might have already noticed a difference between ordinary least squares and maximum likelihood. In OLS we add the squared errors together, but in MLE we multiply the probability densities. However, if we take the log of the likelihood function, it turns multiplications into additions:

$$\log L = \log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2}\left(\frac{y_i - \hat{y}_i}{\sigma}\right)^2 + ... + \log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2}\left(\frac{y_n - \hat{y}_n}{\sigma}\right)^2$$

Since σ is the same for each term, we can factor out everything that doesn't include a y or $\hat{y}$.

$$\log L = \log\left(\frac{n}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2\sigma^2}\left((y_1 - \hat{y}_1)^2 + ... + (y_n - \hat{y}_n)^2\right)$$

which is the same as:

$$\log L = \log\left(\frac{n}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

While this still looks complicated, keep in mind that we don't actually care what the absolute value of the log likelihood is, we just need to know the parameters that minimize the likelihood, and we can use the log likelihood because the parameters that maximize the log-likelihood also maximize the likelihood. By grouping terms that don't depend on the linear model parameter (the *b*'s) in a linear model, we can simplify the equation to:

$$\log L = C_1 - C_2 \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

which is just

$$\log L = C_1 - C_2 \times SS_{Error}$$

Here we can see that the the log likelihood (and thus the likelihood) will be maximized when the SS_{Error} is minimized.

2.6.2 Take-home messages about the relationship between OLS and MLE

This is a rather long demonstration of how OLS is related to MLE for data with normal distributions. I don't expect that you memorize these formulas, what is import though is that:

1. You have some idea of why we focus on the sum of squared errors as a measure of goodness of fit. We use SS_{Error} because it is fundamentally related to likelihood when errors are normally distributed.

2. You understand probability density functions because we will use them throughout the class.

3. After this course you might work on data sets where the dependent variable is not normally distributed. For example, if you work with binary data (e.g. presence/absence), or count data (values only take positive integer values), the errors will have a different error distribution. Maximum likelihood estimation is a general technique for fitting models to data and can be used for any type of data. In this case, you will want to use **Generalized Linear Models**, which are different from the topic of this course (general linear models) in that they generalize to different error distributions (e.g. binomial, multinomial, poisson).

2.7 Optimization (how do we find the best parameters)

We have shown that you can estimate parameters by brute force (trying a wide rang of values) for each parameter in the model. However this becomes much more difficult when there are many parameters. The beauty of linear models, and a major reason why we use them so often, is that it is very easy to fit them to data. Because the predicted value of a linear model is a linear combination of the independent variables, where the effect of each independent variables is weighted by its parameter value, you can use linear algebra and calculus to solve for the parameter values that minimize the sum of squared errors. All statistical software packages use these calculations to estimate model parameters. In our class we will us the `lm()` function in R, which will return the optimal parameter values for the specified model.

The details of how one can analytically solve for the best parameters is beyond the scope of this class, but for those interested feel free to check out these resources on linear optimization (you probably need some experience with linear algebra to follow):

https://www.khanacademy.org/math/linear-algebra/alternate-bases/orthogonal-projections/v/linear-algebra-least-squares-approximation

2.7.1 Non-linear optimization

In cases where the model is non-linear there is no analytical way to solve for the best parameters. In practice this means that that finding the best parameters is essentially done by trial and error. When there are many parameters, the brute force method is no longer feasible. Instead you generally provide a starting guess for the parameters, from which the SS_{Error} or log-likelihood can be calculated. Then various algorithms are used to search for parameters that can improve the fit. Non-linear models are beyond the scope of this class, but for an interesting introduction to non-linear optimization in the context of machine learning check out this video:

https://www.youtube.com/watch?v=IHZwWFHWa-w&t=421s&ab_channel=3Blue1Brown

2.8 Other important metrics of goodness of fit

As discussed above, we use SS_{Error} as a metric of the goodness of fit and optimize our models by finding the parameters that minimize SS_{Error}. But SS_{Error} isn't always the most useful metric for communicating how well our model fits the data. For example, we might want to know how far off the predictions of our model are on average, or we might also what to know how much of the variability in the data is explained by the independent variables in our model. In this section, we briefly discuss two additional but important metrics of goodness of fit: residual standard error and R^2.

2.8.1 Residual standard error

The residual standard error (RSE) is defined as:

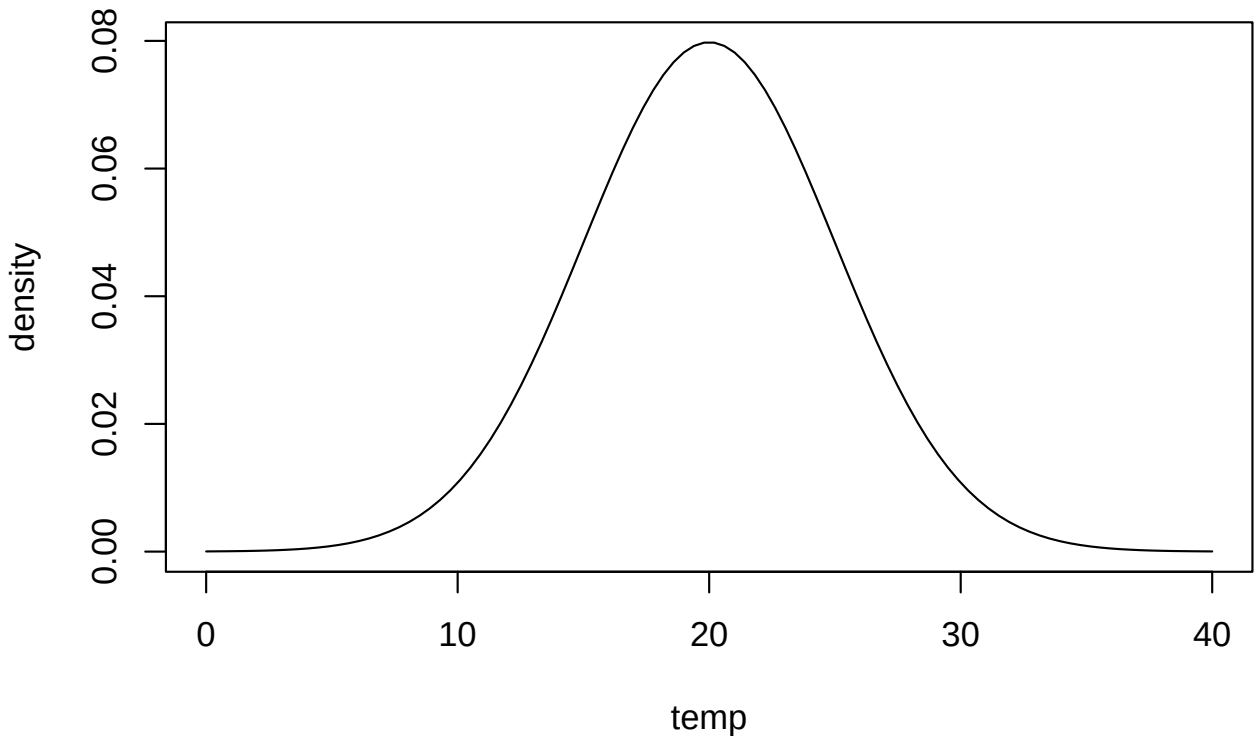
$$\text{RSE} = \sqrt{\frac{SS_{Error}}{df}}$$

where df are the degrees of freedom, which is given by $n-k$, where n is the number of data points and k is the number of parameters in the model. The residual standard error is also an unbiased estimate of the standard deviation of the errors, σ.

To calculate the RSE you take the square root of the sum of the squared errors ($\sum(y_i - \hat{y}_i)^2$) divided by the degrees of freedom. You divide by degrees of freedom rather than the number of data points because if you have a model with n data points and n parameters, the model can fit the data perfectly. For example, with two data points, a regression line can go perfectly through the data, so we need more than 2 data points to be able to quantify the typical variability in the data.

The residual standard error essentially tells how close model predictions are to data on average. If a weather model predicts tomorrow's temperature will be 20°C but has a residual standard error of 5°C, you should be prepared to encounter a wide range of conditions as shown in the following graph:

```
temp<-seq(0,40,length.out=100)
density <- dnorm(temp,20,5)
plot(temp,density,'l')
```



2.8.2 R-squared

Another useful way of quantifying goodness of fit is asking, how much of the variability in the data can be explained by the independent variables in the statistical model.

To do this, we can first quantify how much variability is in the data in the first place, by calculating the sum of squared errors in a model with no independent variables:

$$y = b_0$$

Where b_0 is just the mean of the data.

Doing this is essentially the same thing as calculating the summed variance of the data. We call this the total sum of squares SS_{Total}:

$$SS_{Total} = \sum (y_i - \bar{y})^2$$

where $\bar{y}$ is the mean of the dependent variable.

The idea of R^2 is to partition SS_{Total} into variation explained by the independent variables in the model SS_{Model} and variation not explained by the independent variables SS_{Error}.

$$SS_{Total} = SS_{Model} + SS_{Error}$$

We already know how to calculate SS_{Total} and SS_{Error}, and we can therefore calculate SS_{Model} as:

$$SS_{Model} = SS_{Total} - SS_{Error}$$

We can then calculate what fraction of SS_{Total} is explained by the model by dividing both sides of the equation by SS_{Total}:

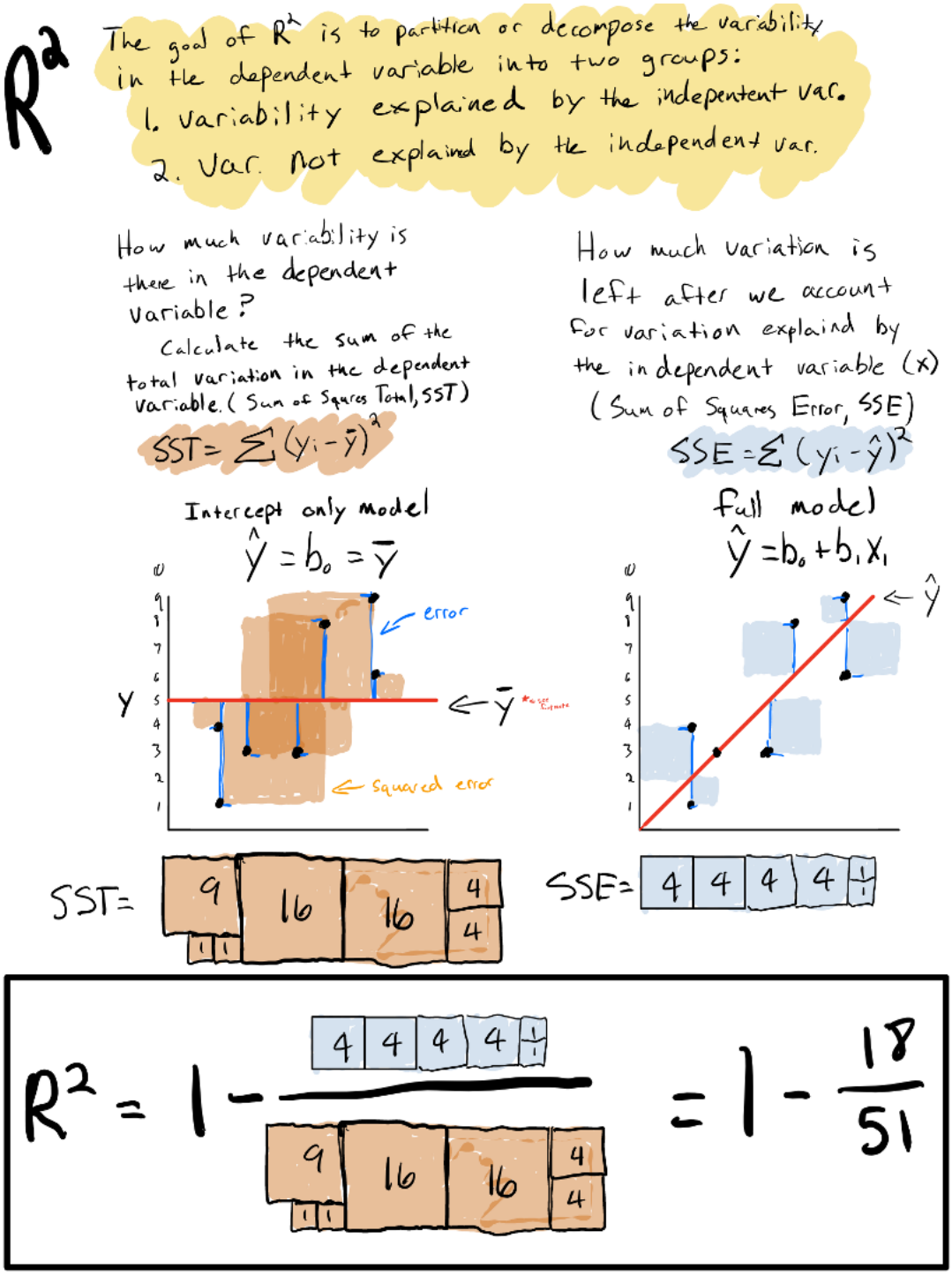
$$R^2 = \frac{SS_{Model}}{SS_{Total}} = \frac{SS_{Total}}{SS_{Total}} - \frac{SS_{Error}}{SS_{Total}}$$

which reduces to:

$$R^2 = 1 - \frac{SS_{Error}}{SS_{Total}}$$

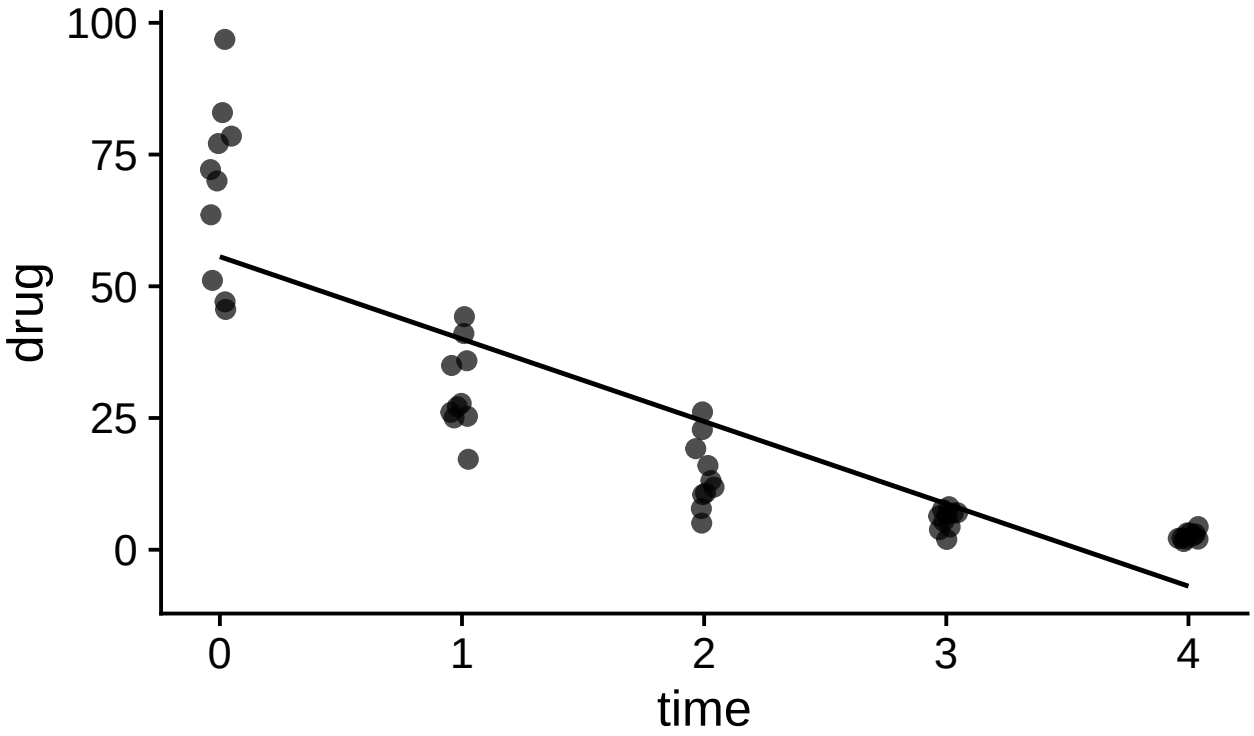
R^2 is a measure of the proportion of the variation in the dependent variable explained by the independent variables in the model. Because R^2 is a proportion, its value will fall between 0 and 1. When it is 0, it means the model doesn't explain any variability in the data, and when it is 1 it means that it explains all the variability in the data. For real data sets, R^2 will fall in between 0 and 1. The closer the value is to 1, the more variation is explained by the model.

Below is an graphical and numerical demonstration of R^2:



2.9 Introduction to transformations

Transformations involve converting a variable from one scale to another. Such transformations are sometimes useful to allow us to use general linear models (that assume linearity, normality and constant variance) to model data that don't meet initially these assumptions. For example, consider the data below that show the concentration of a drug in the blood of patients as a function of time (days) since the drug was taken, and the linear statistical model that was fit to the data:



Question: Can you identify some problems with using the linear model above to model the data?

Probably the most striking problem with the above model is that the assumption that the concentration of the drug decreases as a constant rate per unit time is clearly not supported by the data. For example, going from day 0 to day 1 results in a drop of about 30 mg, while from day 3 to day 4, the drop is only around 5 mg. Instead we see that the rate of drop seems to depend on amount of the drug in the blood; when the concentration is high the drop is high. We also can see the the variance among the data points seems to systematically decrease with time as the concentration of the drug is lower.

Question: Can you think of a equation that might better represent the relationship between time and drug concentration?

The data in the example above, look like an example of exponential decay, were during each interval of time, some fraction In cases, when the growth or decay rate of a quantity is proportional the value of the quantity, we expect concentration to either exponential increase or decrease over time:

$$C(t) = C_0 e^{rt}$$

Here, C_0 and r are the parameters of the model, representing the intial concentration, and growth rate respectively, and in terms of a statistical model, t, or time is the dependent variable. We can tell this model is not linear in the parameters because the parameter r shows up in the exponential along with the independent variable t.

However if we log transform both size of the equation:

$$\ln C(t) = \ln C_0 + rt$$

We see that this has the same structure of a linear model

$$\hat{y} = b_0 + b_1 t$$

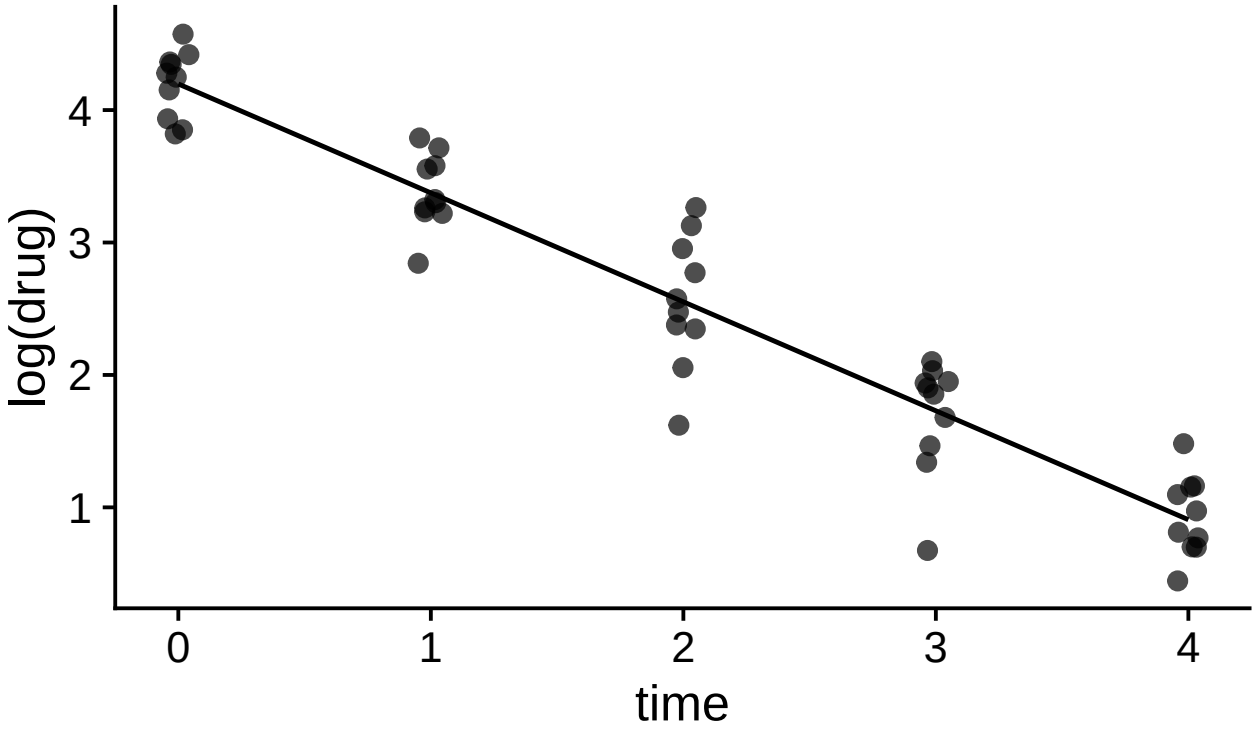
where,

$$\hat{y} = \ln C(t)$$

$$b_0 = \ln C_0$$
$$b_1 = r$$

Thus if our data were generated by an exponential process, either exponential growth or decay, than log transforming the dependent variable allows us to fit a linear statistical model:

```
mod<-lm(log(drug)~time,df)
df$pred<-predict(mod)
ggplot(df,aes(x=time,y=log(drug)))+
  geom_jitter(width=.05,height=0,alpha=.7)+
  geom_line(aes(x=time,y=pred))+
  theme_cowplot()
```



We can from the plot above that converting the dependent to log scale allows us to us a linear model to a non-linear relationship. We can also see that transforming the dependent variable solved the issue of non-constant variance as well. In linear scale, the variability appears to be proportional to the average of the dependent variable, however in log scale, the variability appears to be more or less constant across the data set.

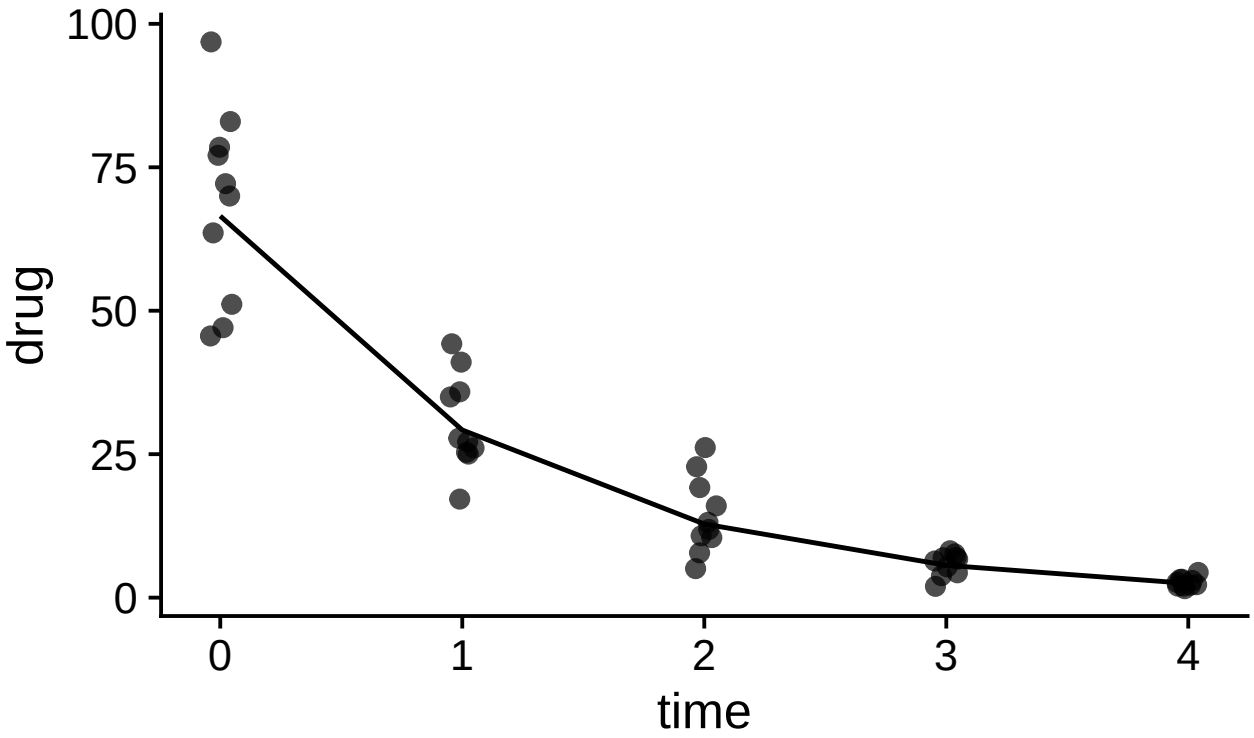
This brings up a general point, if we transform the dependent variable, it will not only affect the relationship between the dependent variable and the independent variables, it will also change the distribution of the error terms. In the example above, transforming the dependent variable solved both the linearity and constant variance, but this will not always be the case.

2.9.1 Converting back to linear scale

While transforming variables can sometimes make it easier to fit statistical models to data, the are often easier to interpret in their original scale. Thus we will often want to convert prediction of our model back to the original scale, or convert the parameters of our linear statistical model back to the parameters of the original non-linear equation.

We can convert the predictions of our model from the transformed to originla scale by applying the inverse function of our transformation. Because exponentiation undoes taking the log transformation, if we apply the exponential function to our prediction, we can convert predictions back to linear scale:

```
mod<-lm(log(drug)~time,df)
df$ln_pred<-predict(mod)
df$pred<-exp(df$ln_pred)
ggplot(df,aes(x=time,y=drug))+
  geom_jitter(width=.05,height=0,alpha=.7)+
  geom_line(aes(x=time,y=pred))+
  theme_cowplot()
```



To relate the linear statistical model parameters back to the parameters of the orininal non-linear equation, we simply need to look at the mathematical relationship between the two. This will depend on the non-linear equation and transformations applied, but in our case, because $b_0 = \ln(C_0)$, and $b_1 = r$, we know that $C_0 = e^{b_0}$ and $r = b_1$.

Thus by looking at the estimated linear parameters:

```
mod<-lm(log(drug)~time,df)
summary(mod)
```

```

Call:
lm(formula = log(drug) ~ time, data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-1.05469 -0.19718  0.03686  0.22040  0.71184

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.19725    0.08703   48.23   <2e-16 ***
time        -0.82260    0.03553  -23.15   <2e-16 ***
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3553 on 48 degrees of freedom
Multiple R-squared:  0.9178,    Adjusted R-squared:  0.9161
F-statistic:   536 on 1 and 48 DF,  p-value: < 2.2e-16
```

We can see that the decay rate is -0.85 and the estimated initial concentration in the blood is ~ e^4.2 or ~66.7.

2.9.2 Summary of transformations

I have just briefly introduced the idea of transforming variables, and but we will build on this throughout the class. For now it is important to know that

1. Transformations can be applied to either the dependent variable, independent variable(s) or both.

2. Transformations of the dependent variable will also affect the distribution of the residuals, while transformation of the independent variables generally should not affect the distribution of the residuals.

3. To convert predictions back to the original scale you have to apply the inverse function of your transformation to your model predictions.

2.10 What you should know

1. What SS_{Total}, SS_{Error}, R^2, and the residual standard errors (RSE) are, how to interpret them, and how to calculate them.

2. You should be able to use `dnorm()` to calculate probability densities, and be able to interpret a probability density function (pdf). You should have an intuitive understanding of the normal pdf, and be able to draw a rough sketch of it if I told you the mean and standard deviation.

3. You should have a general appreciation for why we fit parameters by minimizing SS_{Error} and know how it relates to the concept of maximum likelihood.

3 Uncertainty

3.1 Recap

In Chapter 1 we learned how to formulate statistical models to describe our data and in chapter 2 we learned how to fit these models to data. The estimated parameter values then can give us useful information related to a scientific question, for example, what is the relationship between GDP and carbon emissions among countries? Or what is the effectiveness of a vaccine relative to a control?

We can attempt to answer these questions by fitting statistical models to data. For example, we one wanted to study the effectiveness of a drug for reducing the blood pressure of patients with elevated blood pressure, we might design a study where we give patients either the drug or a placebo and then measure blood pressure after some time on the medication. For now, lets assume that there is no patient information we want to include in my statistical model, and the only independent variable is the treatment (drug or placebo).

$$y = b_0 + b_1 x$$

where y is blood pressure and $x = 0$ if patients were given the placebo and $x = 1$ if patients were given a drug.

Problem 3.1 If you were interested in testing the hypothesis that the drug is more effective than a placebo for patients with high blood pressure, which parameter would be of interest?

b_1 is the estimated difference in blood pressure between treatments (would be negative if the drug reduces blood pressure).

Let's say you fit this model to your data and estimate b_1 to be -5.1 mmHg. This means that patients that were prescribed the drug had on average a blood pressure that was 5.1 mmHg lower than patients prescribed the placebo. If this study had collected data for every person with high blood pressure in the world (the population of interest), we could probably stop here. However for almost any real study, we don't have data for the entire population, instead, we try make inferences about the population of interest from a smaller sample of that population. Instead of including every person with high blood pressure, we might try to get a representative sample of 100 individuals with high blood pressure. Working with samples instead of populations makes doing science feasible, but it also introduces a new problem: uncertainty. Some patients might be helped a lot by the drug, some not as at all. As a result, if we repeated our study, we would get a slightly different answer every time due to the randomness of who happens to be in our study. It also means that the estimated effect of the drug we get from our study will also be different from the true average effectiveness of the drug if we sampled the entire population of interest. The difference between our estimate from our sample, and the true population parameter is called the sampling error. This means that whenever we use samples to infer things about a population, we do so with some error and uncertainty. An important problem we use statistics for is then to estimate how uncertain our estimates and conclusions are given the randomness imposed by finite sample sizes. But how do we do this?

3.2 The sampling distribution and the Central Limit Theorem.

A sampling distribution is a hypothetical distribution of parameter estimates we would get if we repeated our study an infinite number of times. The best way to get intuition for sampling distributions (and many other concepts in statistics) is through simulation. We will illustrate the sampling distribution with an example.

In this example, we want to estimate the average height of students at UvA from a random sample of a small number of students. In this example, we aren't trying to explain student height as a function of any other variables, so our model is just:

$$\hat{y} = b_0$$

Let's assume that the true average height of UvA students is 170 cm with a standard deviation of 10 cm. The question is then, is how accurate are our estimates of this true population parameter for a given sample size?

The smallest sample size possible is 1. In this case we just randomly select a student, record their height, and then our estimate for the mean height of UvA students is equal to the height of the randomly selected individual. In this case our estimate of the mean UvA student height will be as variable as the individuals in the population.

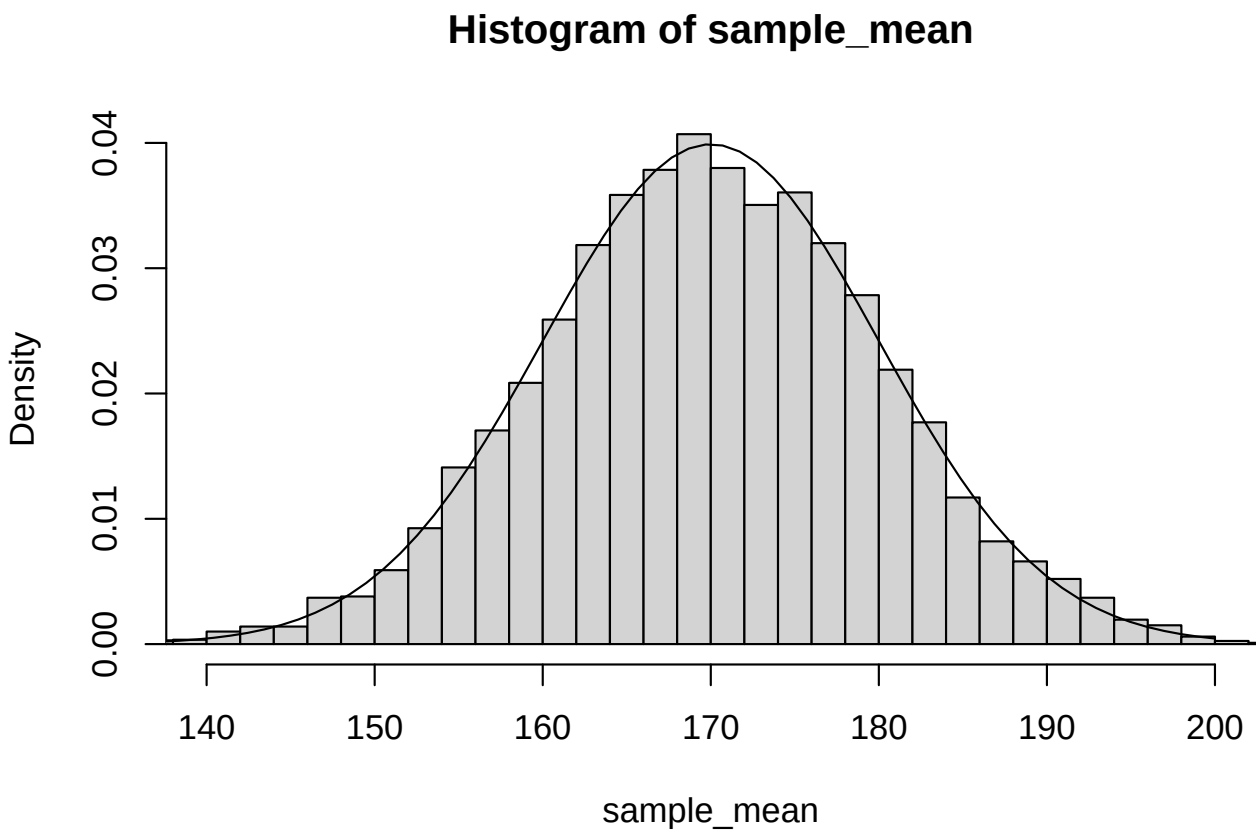
We can write an R program to simulate a large number of studies where we randomly draw one student from the population, and record their height, take the mean (which with a sample size of 1 just equals the height of the selected student). We use a for loop to repeat this many times (10000):

```
sample_size = 1
N_studies = 10000 # a large number to approximate infinite number of studies
sample_mean = rep(NA,N_studies)

# repeat the study many times
for (i in 1:N_studies){
sample_mean[i] <-mean(rnorm(sample_size,170,10))
}
```

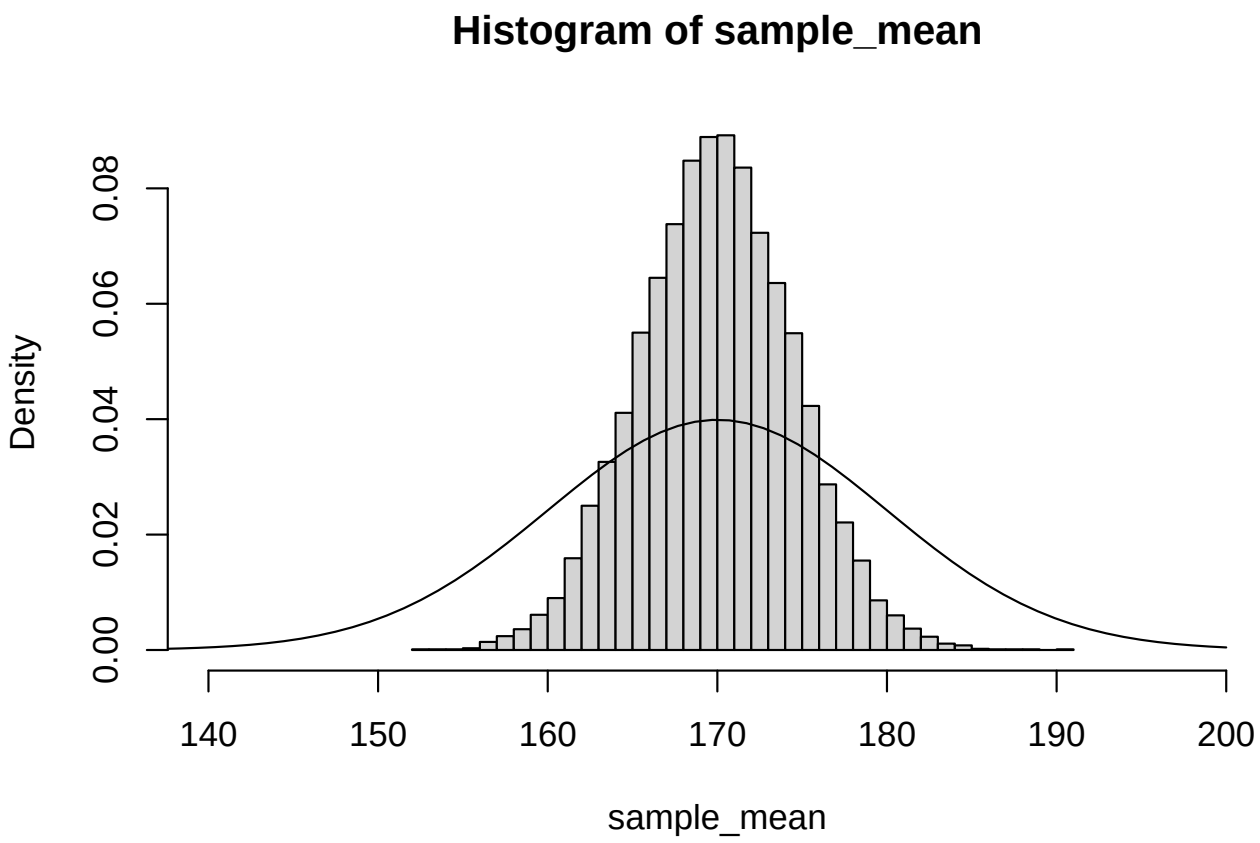
In the code above, `sample_mean` is a vector that contains the results of each of the 10000 studies we conducted, each measuring the height of one student. We simulated this in our code by using the `rnorm()` function in R, to draw 1 observation from the population distribution with mean 170 and standard deviation of 10.

If we plot out the distribution of the estimates along with the probability density function for the population we see that they are essentially identical:

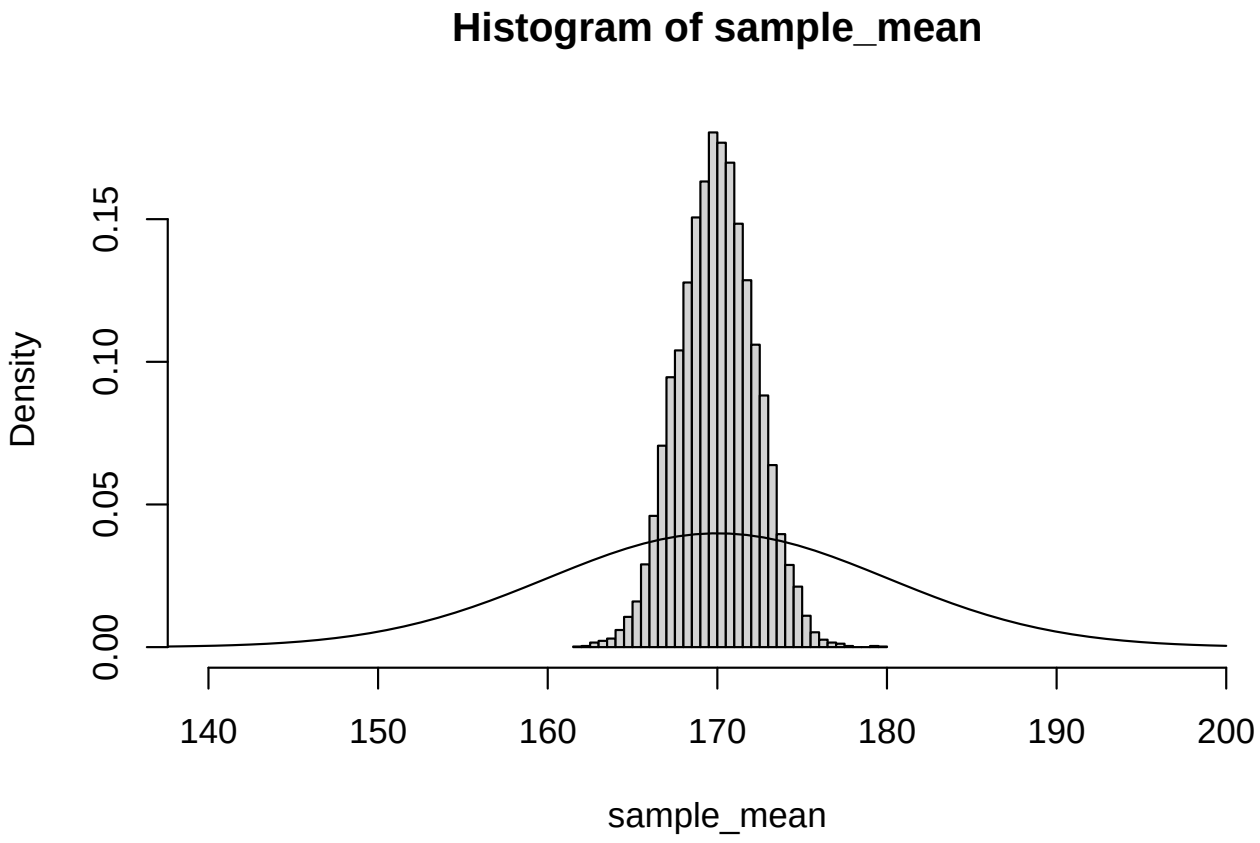
```
hist(sample_mean,freq=F,breaks=50,xlim=c(140,200))
x<-seq(100,200,length.out = 100)
pdf<-dnorm(x,170,10)
lines(x,pdf)
```



As you can see the estimates of our studies are highly variable when we only have a sample size of 1. If we happen to pick 1 unusually short or tall person, our estimate can be off by as much as 20 cm. To improve the precision of our estimate we could sample multiple individuals. For example, if we increased the sample size to 5, we see that our estimates of average height become much less variable:



and if we further increased the sample size to 20, the distribution of estimates of average height is even less variable:



The intuition for why this happens is that as you take multiple samples, it becomes more and more likely for the extreme values to cancel each other out. While it might not be that unlikely to randomly select a student that is over 190 cm (around 1 in 10 chance), it would be very unlikely to randomly select 5 students who were all over 190cm (0.1^5 = about 1 in 100000 chance). Thus as we take larger samples, our estimate of the average height of students at UvA more closely approximates the true population parameter.

An important point about sampling distributions is that for continuous variables they are approximately normally distributed. In the histogram plots above you can see clearly that they look almost exactly like a normal distribution.

Because sampling distributions are approximately normal, we can quantify the shape of the distribution by its standard deviation. The standard deviation of the sampling distribution tells us how variable the estimates of a model parameter are across replicate studies. In the UvA height example, we have the model parameter b_0, which is just the mean of the sample. The standard deviation of the sampling distribution for b_0 is referred to as ***the standard error of the parameter estimate***. In this context, because the parameter is a sample mean, it is also referred to as the standard error of the mean. But in general every estimated parameter in our model has sampling error, and the standard error of the parameter tells us how uncertain our estimate of that parameter is. When the standard error is very large, it means that the estimate of the parameter values is highly uncertain and the true population parameter may be quite different from the estimated parameter from a sample.

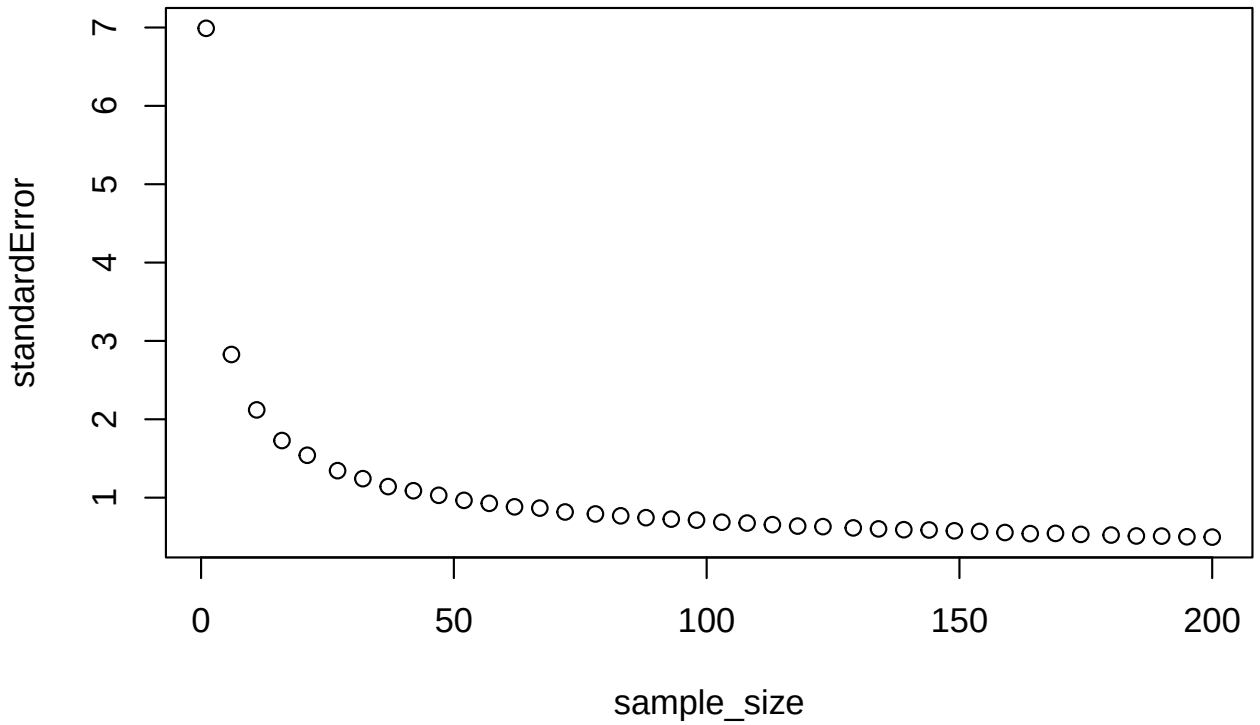
Interestingly, if you have a large enough sample size (~30 or more), the sampling distribution will approximate a normal distribution even if the raw data are not normally distributed. This result is known as the **Central Value Theorem**. In class we will show an example of how normal distributions emerge from averages of highly skewed data. The Central Value Theorem is really an astonishing and surprising result with some important implications. First it means that even when the raw data don't have normally distributed errors, the sampling distribution of parameters (for example a mean or regression slope) will be approximately normal. Thus our estimates of standard errors and confidence intervals are still approximately valid even when the raw data is not normal (assuming large enough sample sizes). Secondly, the Central Value Theorem helps explain why so many variables turn out to be normally distributed. Whenever a variable like someones' height, is determined by the aggregate effect of many random variables (e.g. many genes and environmental factors) the variable will be approximately normally distributed.

3.3 How does uncertainty in parameter estimates change with sample size?

How large does our sample size need to be to accurately estimate the population parameters of interest?

We can begin to answer this question by repeating our analysis for different sample sizes. In the code below I use an additional for loop to generate sampling distributions for sample sizes ranging from 1 to 200. For each sample size I then calculate the standard deviation of the sampling distribution (the standard error of a parameter estimate) and then plot it as a function of sample size:

```
sample_size = round(seq(1,200,length.out=40))
N_studies = 10000 # a large number to approximate infinite number of studies
sample_mean = rep(NA,N_studies)
standardError = rep(NA,length(sample_size))
# repeat study many times
for (j in 1:length(sample_size)){
  for (i in 1:N_studies){
    sample_mean[i] <-mean(rnorm(sample_size[j],170,7))
  }
standardError[j]=sd(sample_mean)
}
plot(sample_size,standardError)
```



As you can see the standard error goes down with sample size, but with diminishing returns. Standard error decreases more rapidly with sample size when sample size is low.

3.4 Calculating standard errors from a sample

In the example above, we could calculate standard errors of parameter estimates by simulating a study many, many times. But in the real world, we generally just have one study. Fortunately, we can estimate the standard error of parameters directly from a sample. In the case of a model with no independent variables (just a mean) there is a simple formula to calculate the standard error of the mean:

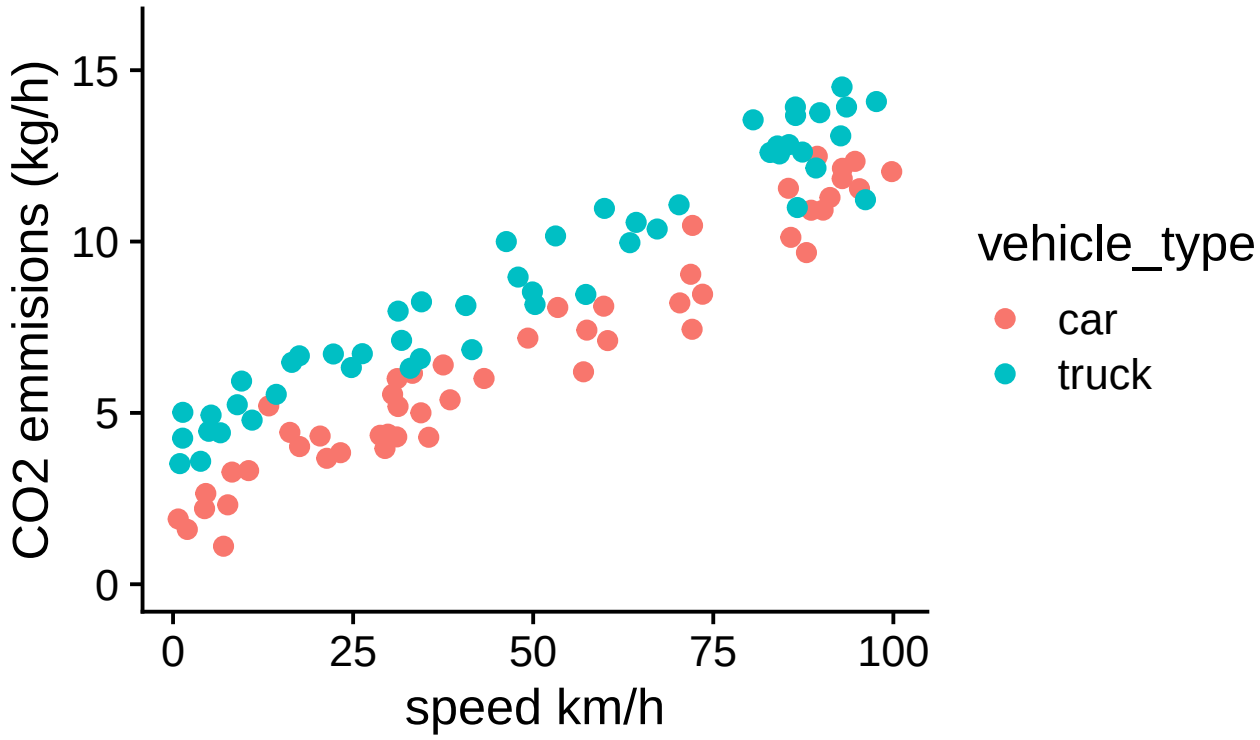
$$\hat{SEM} = \hat{\sigma}/\sqrt{n}$$

where $\hat{\sigma}$ is the residual standard error (the estimated standard deviation of the residuals or error) and n is the sample size.

For more complex models with multiple parameters, there isn't a simple formula to calculate standard errors, it requires linear algebra, but the essence of this formula holds also for more complex models. The standard error of parameter estimates will increase as the data becomes noisier (larger σ) and the standard error of parameters will decrease with increasing sample size, but again with diminishing returns.

When you fit a model to data in R or any other statistical software, it will return the parameter estimate for each parameter in the model, but also the standard error for each parameter value. The parameter estimate tells you the most likely value of the parameter given the data, and the standard error tells you how wide the range of likely values are for the parameter. If a parameter has a large standard error that means that there is a lot of uncertainty about the true population parameter, while if the standard error is small, then it suggests that the true population parameter is likely very close to the estimated value.

Going back to the CO2 emissions data set:



We can fit a model with two independent variables (speed and vehicle_type) to the data using the `lm()` function:

```
mod<- lm(CO2~speed+vehicle_type,df)
summary(mod)
```

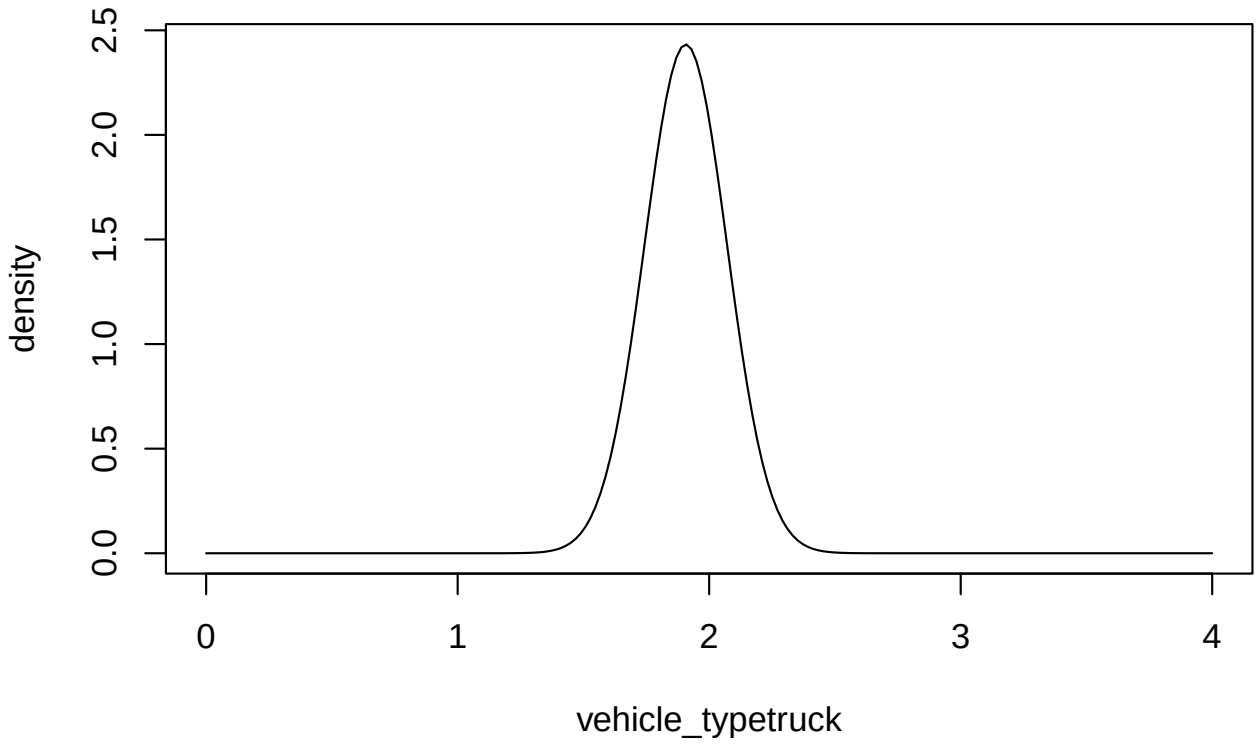
```

Call:
lm(formula = CO2 ~ speed + vehicle_type, data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-2.52676 -0.45656  0.02895  0.58776  1.96606

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       1.897667   0.167456   11.33   <2e-16 ***
speed             0.100947   0.002557   39.49   <2e-16 ***
vehicle_typetruck 2.145354   0.162546   13.20   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8124 on 97 degrees of freedom
Multiple R-squared:  0.9478,    Adjusted R-squared:  0.9468
F-statistic: 881.3 on 2 and 97 DF,  p-value: < 2.2e-16
```

In the "coefficients" section of summary table of the model, there are 3 rows. Each row corresponds to a parameter in the model, with the name of the parameter to the left of the table. In the table the first column shows the parameter estimate, and the 2nd shows the standard error. For now we will ignore the other columns. You should be able to look at the first two rows and get an idea of the likely range of the true population parameters.

Problem 3.2 Plot out the estimated probability density function for the parameter associated with vehicle_type (truck = 1).

We can answer this question using the `dnorm()` function and plugging in the parameter estimate for vehicle_typetruck and its standard error:

```
vehicle_typetruck<-seq(0,4,length.out=200)
density<-dnorm(vehicle_typetruck,1.907,0.164)
plot(vehicle_typetruck,density,'l')
```



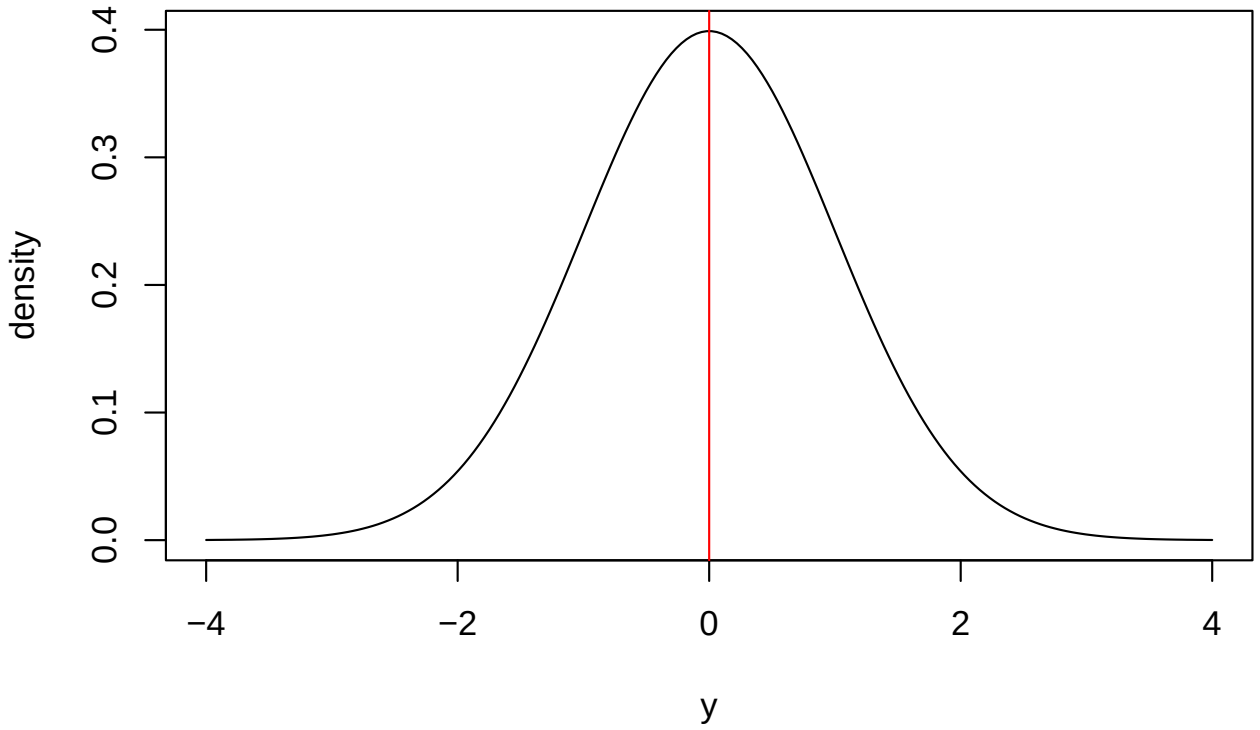
Problem 3.2 By looking at this figure above, we can see that the most likely value is a little less than 2. If you had to estimate by eye, what is the minimum value for the population parameter that seems likely? What about the maximum value?

With this problem I am asking you about your confidence that the true population parameter of the difference in mean CO2 emissions of trucks vs. cars falls within some bound. For example, you might want to say something like, the estimated difference in CO2 emissions in trucks vs. cars is 1.9, and I am 95% confident that the true difference between trucks and cars (if we collected data for every car and truck in the world) would be between 1.5 and 2.3. What I just described is a confidence interval. Here, I just guessed these values by eyeballing the figure, but below we will show how they can be calculated precisely. However before doing so, we first need to review the relationship between probability density functions and probabilities.

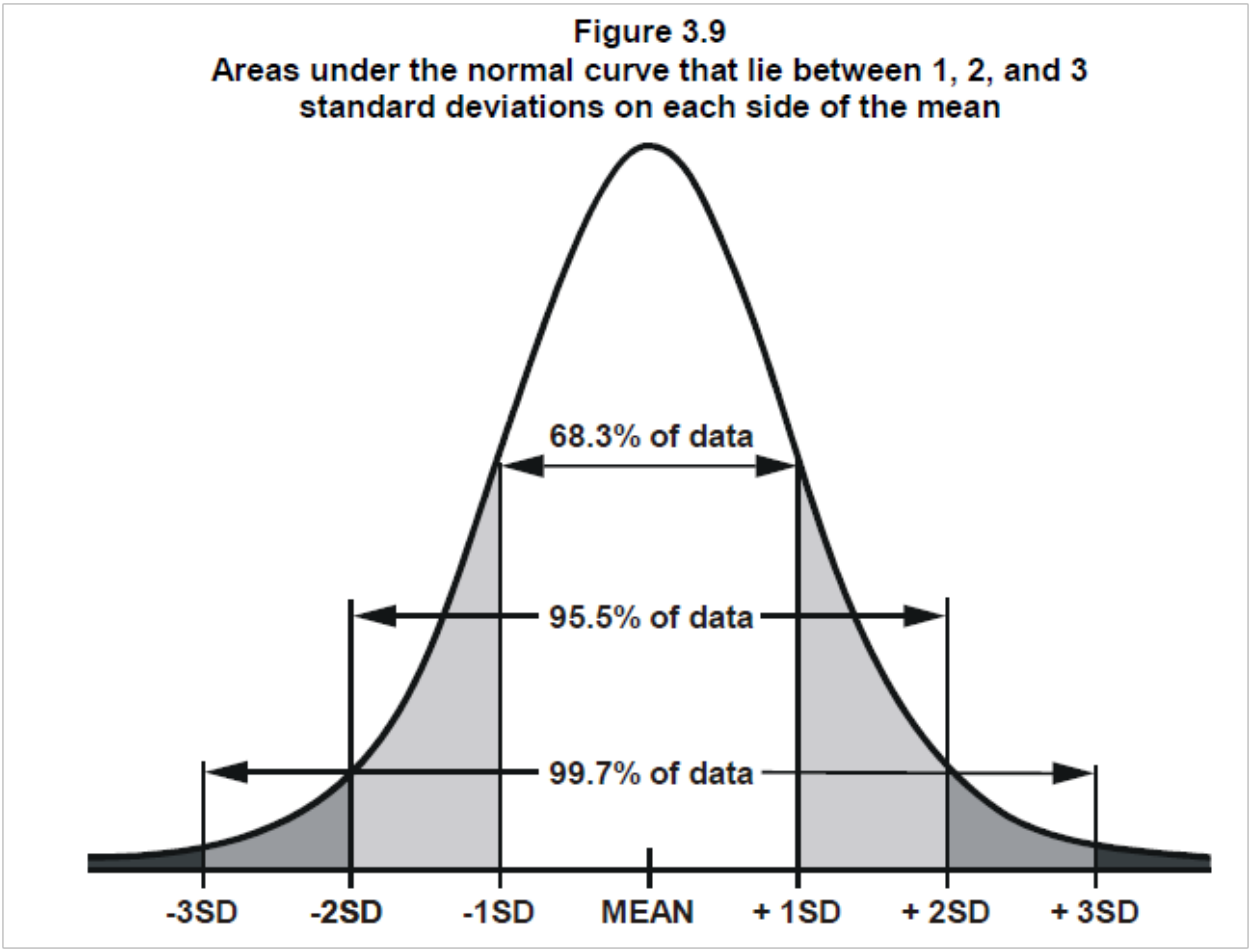
3.4.1 Review of probability density functions and probabilities

The plot above is another example of a normal probability density function. As a reminder, the integral of a pdf over the entire range of possible values, or the total area under the curve, equals 1. Moreover if we specify any range on this continuous scale, by a lower and upper value, the area under the curve between these two points is the probability the value falls within this range.

For example, if we have a random variable, y, with mean 0 and sd = 1:



the distribution is symmetric around the mean, with half the area under the curve occurring for values $y < 0$ and half the area under the curve for values of y above zero. Thus the probability of drawing a random number greater than zero would be 50%. We might also want to know what is the probability of an observation falling within some distance of the mean? For example, what fraction of observations drawn from a normal distribution do we expect to fall within 1 standard deviation of the mean. This is essentially asking the question what proportion of the total area under the curve occurs between -1 and 1 times the standard deviation (between -1 and 1 sd) from the mean. It turns out that approximately 68% percent of the area under the curve falls within 1 standard deviation of the mean:



You can also see that 95% percent of the area under the curve is within 2 standard deviations of the mean, and nearly all, 99.7% of the area under the curve is within 3 standard deviations of the mean.

We can calculate these numbers precisely using the `pnorm()` function in R. The syntax of pnorm is:

```
pnorm(x = 0, mean = 0, sd = 1)
```

`pnorm()` returns the integral (area under the curve) from -infinity up to the value of x in the first argument for a given mean and standard deviation sd. So the probability of observing a value of x less than zero when mean = 0 and sd = 1 can be computed with:

```
pnorm(0, 0, 1)
```

```
[1] 0.5
```

And the probability of observing a value less than -1 as:

```
pnorm(-1, 0, 1)
```

```
[1] 0.1586553
```

Because the normal distribution is symmetric the probability of observing a value less than -1 times the standard deviation is the same as the probability of observing a value greater than 1 times the standard deviation. So the probability of observing a value between -1 and 1 is:

```
1 - 2 * pnorm(-1, 0, 1)
```

```
[1] 0.6826895
```

which agrees with the approximate value stated earlier.

Problem 3.3 If y is a normally distributed random variable with mean 2.1 and standard deviation 1.3, what fraction of y values would you expect to fall between 0 and 1?

Plugging in the values we would expect:

```
pnorm(0, 2.1, 1.3)
```

```
[1] 0.05311371
```

about 5% of the data to be less than 0, and:

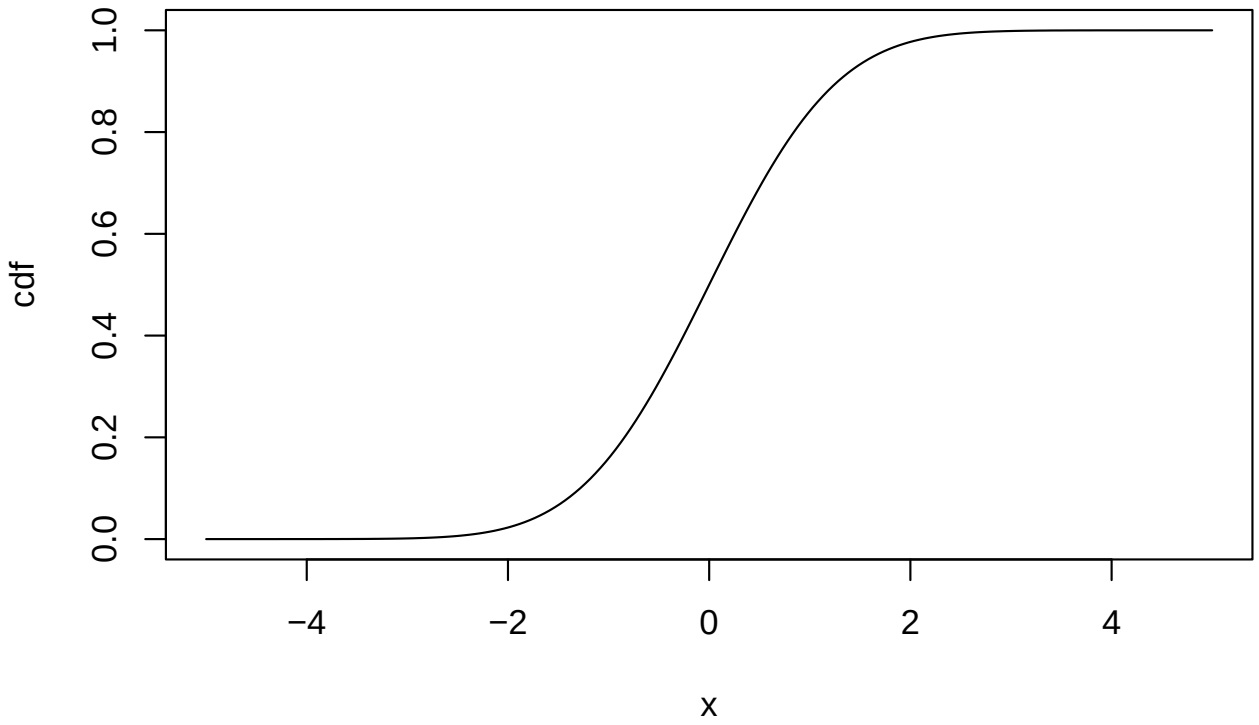
```
pnorm(1, 2.1, 1.3)
```

```
[1] 0.1987335
```

around 20% of the data to be less than 1, so $20 - 5 \approx 15\%$ of the data would be between 0 and 1.

Finally, if we plug in a vector of x values and plot the output of `pnorm()`:

```
x<-seq(-5, 5, length.out=200)
cdf<-pnorm(x, 0, 1)
plot(x, cdf, 'l')
```



we see a cumulative distribution function, where the y value for a given x value corresponds to the probability of observing a value less than x. You can see that the slope of the line increases most rapidly around the mean, because the slope of this line is equal to the probability density function, which is highest around the mean.

3.5 Confidence intervals

If you understand probability density functions and cumulative distribution functions, then confidence intervals are fairly straightforward. When you fit a model, you get a parameter estimate and standard error for each parameter. These two values can be used to produce a probability density function of the likely values of the parameter, and from this pdf we can calculate what interval is required to encompass 95% of the area under the curve.

The main distinction between confidence intervals and other applications of pdfs, is that confidence intervals don't tell you the probability that the true population parameter falls within the confidence interval. This is because the true population parameter is a fixed value. For example in the context of the vehicle CO2 emissions, the true population parameter for "vehicle_type" is the average difference in CO2 emissions between trucks and cars, if we measured emissions for every car and truck. It has one specific value, and it either is or is not within out confidence interval. In our study, we tried to estimate the population value with a random sample of cars and trucks. If we repeated our study many, many times, in the long run we should expect the true population parameter to fall within the 95% confidence intervals about 95% of the time. Thus for any one sample, we can be 95% "confident'' that the true population parameter falls within our confidence intervals.

3.6 The 2 SE method for estimating 95% CI

Because approximately, 95% of the area under a normal distribution falls within 2 standard deviations of the mean, one quick (but approximate) way to estimate 95% confidence intervals is:

```
upper_CI = estimate + 2 sd
lower_CI = estimate - 2 sd
```

I call this "the 2 SE method'' for estimating 95% confidence intervals. The values produced by this method are not precise, but are generally very close, and this method is an easy way to look at parameter estimates and standard

errors and get a sense for the range of likely parameter values.

We can calculate confidence intervals for data manually using data from the coefficients table from the summary output of the model:

```
mod <- lm(CO2~speed+vehicle_type,df)
summary(mod)$coefficients
```

```
                   Estimate  Std. Error  t value     Pr(>|t|)
(Intercept)       1.8976667 0.167455988 11.33233 1.838625e-19
speed             0.1009468 0.002556536 39.48578 1.421217e-61
vehicle_typetruck 2.1453539 0.162546470 13.19840 2.214078e-23
```

where `mod` is the name we use for the output of the `lm()` model fit.

If we want to access specific elements in the table we just need to specify the row and column number. For example the standard error of the intercept parameter is in the first row and 2nd column so we can access it by:

```
summary(mod)$coefficients[1,2]
```

```
[1] 0.167456
```

Problem 3.4 Use the quick and dirty 2 SE method to calculate the 95% confidence intervals for the parameter associated with `vehicle_type`.

You can easily do this manually, but using R you can do this by:

```
estimate = summary(mod)$coefficients[3,1]
s_e = summary(mod)$coefficients[3,2]
upper_CI = estimate + 2 * s_e
lower_CI = estimate - 2 * s_e

upper_CI
```

```
[1] 2.470447
```

```
lower_CI
```

```
[1] 1.820261
```

If we wanted to be more precise, we can use the `qnorm()` function to find the range of values needed to encompass 95% of the area under then normal curve. The 'q' stands for 'quantile function' and is the inverse function of `pnorm()`. For example if we have a normal distribution with mean = 0, sd = 1, we can calculate the probability of observing data less than -2 by:

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

```
[1] 0.02275013
```

Which means that 0.0227 of the area under the curve occurs at values less than -2. Conversely, if we ask what is the value of $xS below which 0.02275 of the area under the curve falls, we can use `qnorm()`:

```
qnorm(0.02275013, mean=0, sd=1)
```

```
[1] -2
```

So if you want to calculate how wide of an interval is needed to encompass 95% of the area under the normal distribution you would use qnorm:

```
qnorm(c(0.025, 0.975), mean=0, sd=1)
```

```
[1] -1.959964  1.959964
```

I used `c(0.025, 0.975)`, because for 95% CI, 2.5% of the area under the curve will be on the lower tail, and 2.5% will be in the upper tail. Thus we see that the quick and dirty 2 SE rule is only approximate, the exact value is closer to 1.96. We can then use this value to calculate confidence intervals:

```
estimate = summary(mod)$coefficients[3,1]
s_e = summary(mod)$coefficients[3,2]
upper_CI = estimate + 1.96 * s_e
lower_CI = estimate - 1.96 * s_e

upper_CI
```

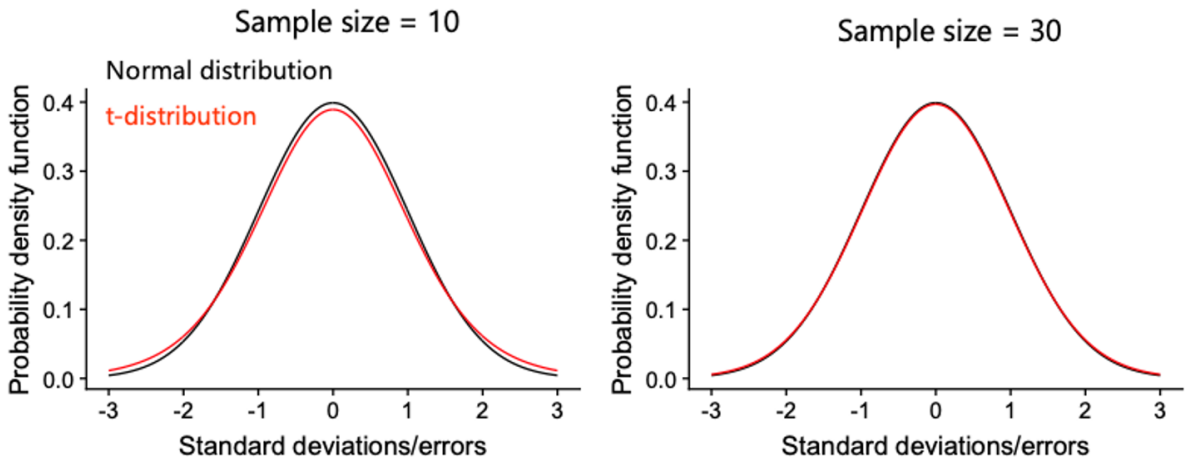
```
[1] 2.463945
```

```
lower_CI
```

```
[1] 1.826763
```

3.7 The *t*-distribution

Now you should have the basic idea of confidence intervals and where they come from. However there is one more minor detail. The calculations we used assume that the σ, the standard deviation of the errors is known, when in fact we just estimate it from our sample. The consequences of this is that to calculate the true confidence intervals, we have to use the *t*-distribution instead of the normal distribution. The *t*-distribution looks almost exactly like the normal distribution but it has wider tails. The shape of the distribution is also dependents on the amount of data we have, or more precisely the degrees of freedom. When we have lots of data (e.g. > 30 df) the *t*-distribution is almost indistinguishable from the normal distribution. It is only when we have very little data, that the distributions differ significantly. This is because when we have little data, there is more uncertainly about how variable the errors truly are.



In the case of the vehicle CO_2 emissions, we had 100 data points and 3 parameters, so 97 degrees of freedom. To calculate CI intervals using the *t*-distribution we need to calculate what interval on the *t*-distribution is needed to contain 95% of the area under the curve. For this we use the `qt()` function, which is like the `qnorm()` function but for the *t*-distribution instead of the normal distribution:

```
critical_value = qt(c(0.025, 0.975), df = 97)
critical_value
```

```
[1] -1.984723  1.984723
```

Here the first argument is the same as in `qnorm()`, and the second argument is the degrees of freedom. We see the the values we get are very similar to `qnorm()`, because in this case we have a lot of data. If we only had 5 degrees of freedom, you see that the interval would be wider and thus so would the confidence intervals:

```
qt(c(0.025, 0.975), df = 5)
```

```
[1] -2.570582  2.570582
```

Finally we can put this all together to precisely calculate the confidence intervals:

```
estimate = summary(mod)$coefficients[3, 1]
s_e = summary(mod)$coefficients[3, 2]
```

```
critical_value = qt(0.975, df = 97)
upper_CI = estimate + critical_value * s_e
lower_CI = estimate - critical_value * s_e

upper_CI
```

```
[1] 2.467964
```

```
lower_CI
```

```
[1] 1.822744
```

There is also a built-in function to calculate confidence intervals in R:

```
confint(mod)
```

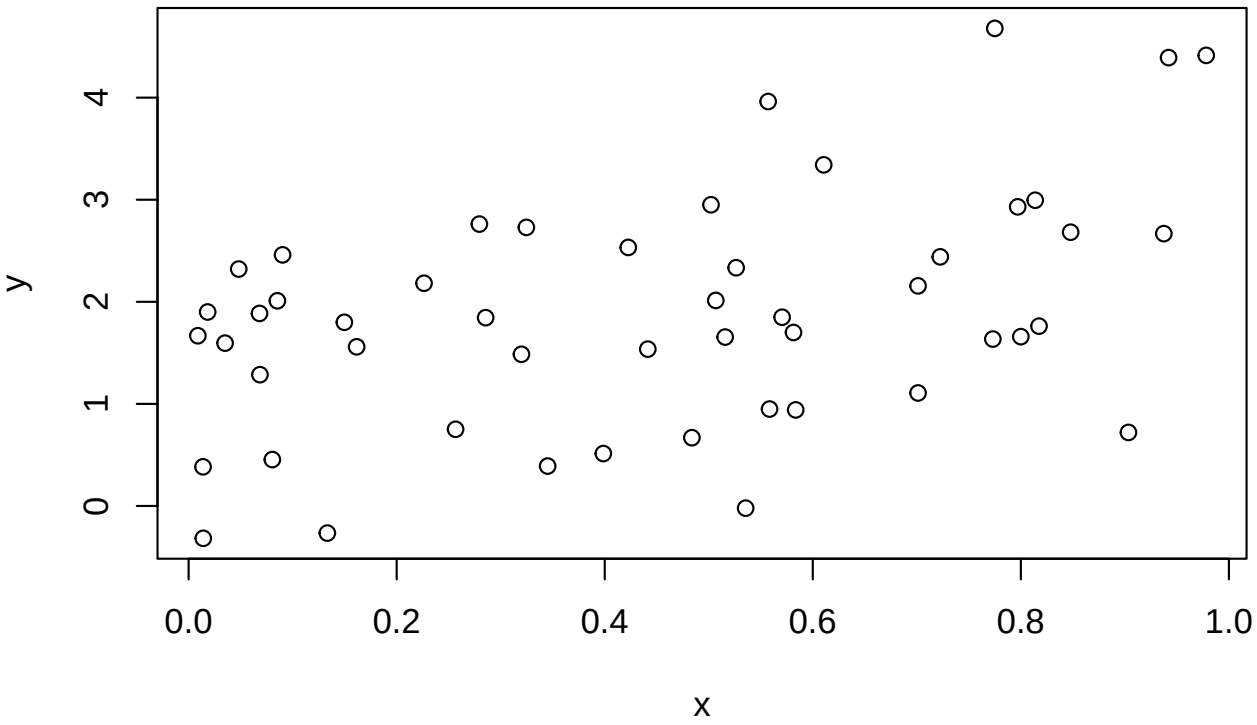
```
                     2.5 %    97.5 %
(Intercept)       1.5653130 2.2300205
speed             0.0958728 0.1060208
vehicle_typetruck 1.8227441 2.4679636
```

and we see the we got the same answer here as when we calculated the CI manually. We can see that the quick and dirty 2 SE method we used earlier to calculate 95% CI is pretty close: `[1.580352 - 2.315591]`.

3.8 Simulating data

Simulations are a very powerful tool both for getting intuition for statistical modelling, and exploring whether our statistical methods are working. In the code below I generate a simple data set with one continous independent variable:

```
# sample size
N = 50
b_0 = 1
b_1 = 2
sigma = 1
x = runif(N,0,1)
y = b_0 + b_1 * x + rnorm(N,0,sigma)
plot(x,y)
```



In the code above I needed to define the sample size, and model coefficients, and the standard deviation of the errors (sigma). Then I generated a range of x values from a uniform distribution between 0 and 1 (however you could change the range or even the distribution). Then to generate values of y, I account for the non-random component of y `b_0 + b_1 * x` and add to this the random component drawn from a normal distribution with mean 0 and standard deviation equal to sigma. The important thing here is that the size of the vector of random component needs to match the fixed part, which I accomplished by using drawing N values of x and N values from the normal distribution.

You can use this basic structure to generate many types of linear models. The advantage of simulations compared to the real world is that we know the true model that generated the data, and therefor we can check if our statistical inferences are correct. For example, the definition of confidence intervals is that we expect them to contain the true parameter about 95 percent of the time. With simulations, we can actually see if this is correct or not by repeating the simulations above many times, and checking how often the estimate 95% CI contain the true parameter (e.g. b_1 in the example above.)

Importantly, we can also use simulations to create cases where we know assumptions of our model are violated. For example, we can generate datasets where the errors are not normally distributed, or where variance increase systematically with either the dependent or independent variable and then check whether our statistical inferences are robust to these violations.

3.9 What you need to know

1. What a sampling distribution is and how it relates to standard errors: If we repeat a study many times, we would get a distribution of parameter estimates. The standard deviations of these distributions is the standard error of a parameter estimate. Although we generally don't repeat our studies many times, we can estimate standard errors for parameters from our sample. In general, standard errors increase in proportion to standard deviation of the errors/residuals, and decrease in proportion to the square root of sample size.

2. Be able to interpret standard errors and understand what they imply about the likely ranges of parameter values.

3. You should be able to understand the relationship between `dnorm()`, `pnorm()`, and `qnorm()`, and how to use them.

4. You should have an intuitive understanding of confidence intervals, how it relates to the area under the normal pdf, and be able to use the quick and dirty 2 SE value to calculate them.

4 Hypothesis testing

4.1 Recap

In Chapter 1, we learned how to formulate linear models to model data. In Chapter 2, we learned how these models are fit to data, and various ways to quantify goodness of fit. In Chapter 3, we learned how to quantify the uncertainty in parameter estimates given a particular dataset. In a way, if you understand these three chapters well, you are already prepared to begin using models to learn from data. Now, in Chapter 4, we will cover how we can use models to test (null) hypotheses.

Null hypothesis significance testing (NHST), p-values, and 'statistical significance' are concepts widely used and often misinterpreted in scientific research and media reports. In fact, there is an ongoing debate in science about whether and when NHST should be used at all. Some journals have even gone so far as to ban the use of NHST! You have already come across NHST in previous stats classes, but if you're like many practicing scientists, there is a good chance you don't quite understand how to use and interpret the results of NHST correctly. The goal of this chapter is to demystify p-values, statistical significance, and null hypothesis testing, so that you can better understand your own statistical results, but also be able to critically assess statistical results reported in scientific papers and the media.

4.2 The motivation behind p-values

Suppose you conducted a study to assess whether a particular drug can reduce blood pressure, and found that, on average, individuals who took the drug had lower blood pressure compared to those who took a placebo. You are excited about the result, but then you begin to worry: what if this result is just a coincidence? For example, if the drug truly had no effect, there would be a 50% chance of observing lower average blood pressures in the drug group (the chance that they are *exactly* the same on average is vanishing small). This is the problem that p-values a meant to help deal with. A p-value is the probability of observing the effect we observed (or a larger one) due to chance alone. By "effect" I mean the magnitude of a parameter value in a statistical model. If p is close to 1, it means that we should expect to see effects (a parameter estimate) as or more extreme than the one observed in out study quite often due to random chance alone. If the p-value is very low it means that we would expect to see the type of results we observed due to random chance only rarely.

A common statistical practice is to formulate a null hypothesis, which generally corresponds to there being no effect of an independent variable (parameter = 0). In the blood pressure example, a null hypothesis would correspond there being no true average difference in blood pressure between people receiving the drug vs. a placebo. So in the linear model:

$$\hat{y} = b_0 + b_1 x$$

$x_1 = 0$ if placebo, and $x_1 = 1$ if drug. Then b_1 is the estimated difference in blood pressure between those receiving the drug, vs. a control. The null hypothesis in this case would be $b_1 = 0$. If the null hypothesis is true, then any difference we observed between individuals on the drug. vs. placebo is due to random chance. We calculate the probability of observing the difference we observed in our study (or a larger difference) with the p-value.

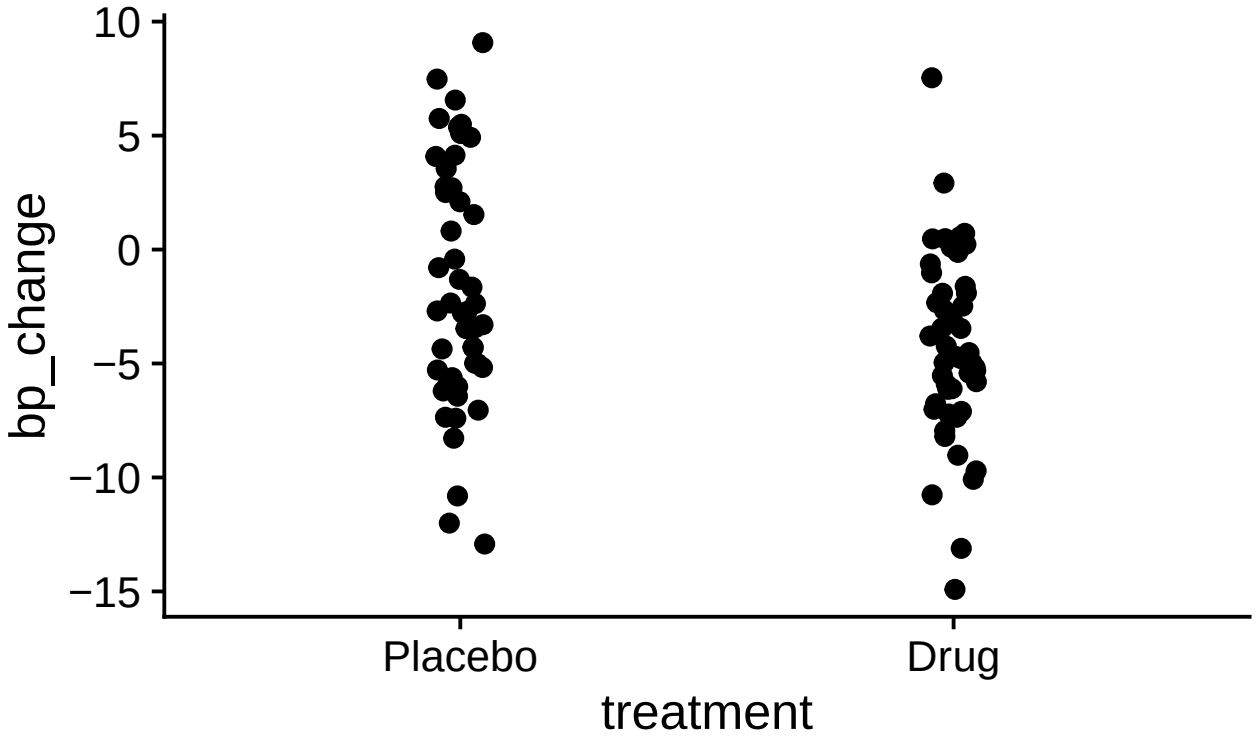
4.3 Conventional use of NHST

Null Hypothesis Significance Testing (NHST) involves using a p-value to arrive at a binary decision: rejecting or failing to reject the null hypothesis. If the p-value falls below a predetermined threshold ($alpha$), typically set at 0.05 in many disciplines, the null hypothesis is rejected. This means if there's less than a 5% chance of observing an effect as extreme as the study's findings if the null hypothesis is true, the null is rejected. This threshold is known as the false positive rate, signifying the probability of incorrectly rejecting a true null hypothesis. Conversely, if a p-value is greater than $alpha$, then we fail to reject the null hypothesis. Conventionally, results with p-values below 0.05 are deemed "statistically significant," while those above 0.05 are considered "non-significant." However, it's crucial to distinguish between statistical and practical significance, as well as to understand that failing to reject the null does not equate to accepting it as true.

4.4 P-values and NHST: an example

To get a better idea of how p-values are calculated, let's work through an example with data from the blood pressure study. The study had 100 participants, with 50 randomly assigned the drug, and 50 assigned the placebo. In this case the null hypothesis was that there is that there is no true difference between control and drug groups. So in the statistical model this is equivalent to $b_1 = 0$. They also decided that the will reject the null hypothesis if the probability of observing b_1 as or more extreme than the value measured in their study due to chance is less than 5.

The researchers collected the following data, where 'bp_change' was the change in blood pressure during the study, with negative values indicating a reduction in blood pressure.



In the plot above you can see that on average, patients that received the drug experienced a lower drop in blood pressure than those that received a control. But the question, is how likely would this extreme of a difference emerge just due to random chance when there were no true differences in the effectiveness of the drug over the placebo.

We can begin to answer his question by fitting our linear model, and calculating the parameter estimates and their standard errors.

```
mod<-lm(bp_change~treatment,df)
summary(mod)
```

```

Call:
lm(formula = bp_change ~ treatment, data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-11.2445  -3.0409  -0.6127   3.7715  11.9445

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    -1.6753     0.6714  -2.495  0.01426 *
```

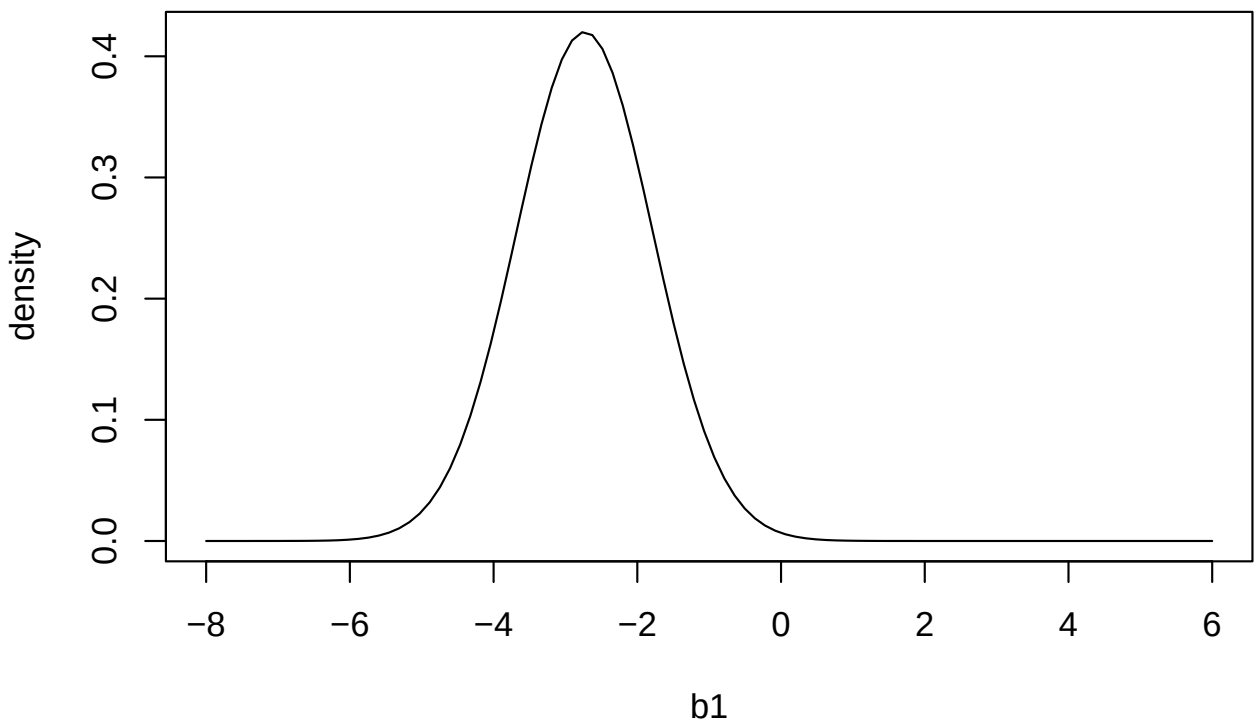
```
treatmentDrug  -2.7330     0.9495  -2.878  0.00491 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.747 on 98 degrees of freedom
Multiple R-squared:  0.07795,   Adjusted R-squared:  0.06854
F-statistic: 8.285 on 1 and 98 DF,  p-value: 0.004908
```

Problem 4.1 We see two rows in the coefficients tables. Which parameters do each of these rows correspond to in the linear model. What is the interpretation of each parameter? Which of these rows is most relevant for testing our null hypothesis that the drug has no effect relative to a placebo?

Hopefully it should be clear to you that the first row, corresponds to the intercept in the linear model, b0, which in this case is the estimated change in blood pressure for patients recieving the placebo. The second row corresponds to the estimated difference in the change in bp for individuals receiving the drug vs, the placebo. The null hypothesis pertains to the 2nd row in the data set, where the null hypothesis is that this parameter is 0.

By looking at the second row, we see that patients that took the drug had on average a 2.73 greater reduction in blood pressure than the control group. We can also see that the standard error for this parameter is 0.9495. We can use these values to calculate the range of parameter values of b_1 that are consistent with the data:

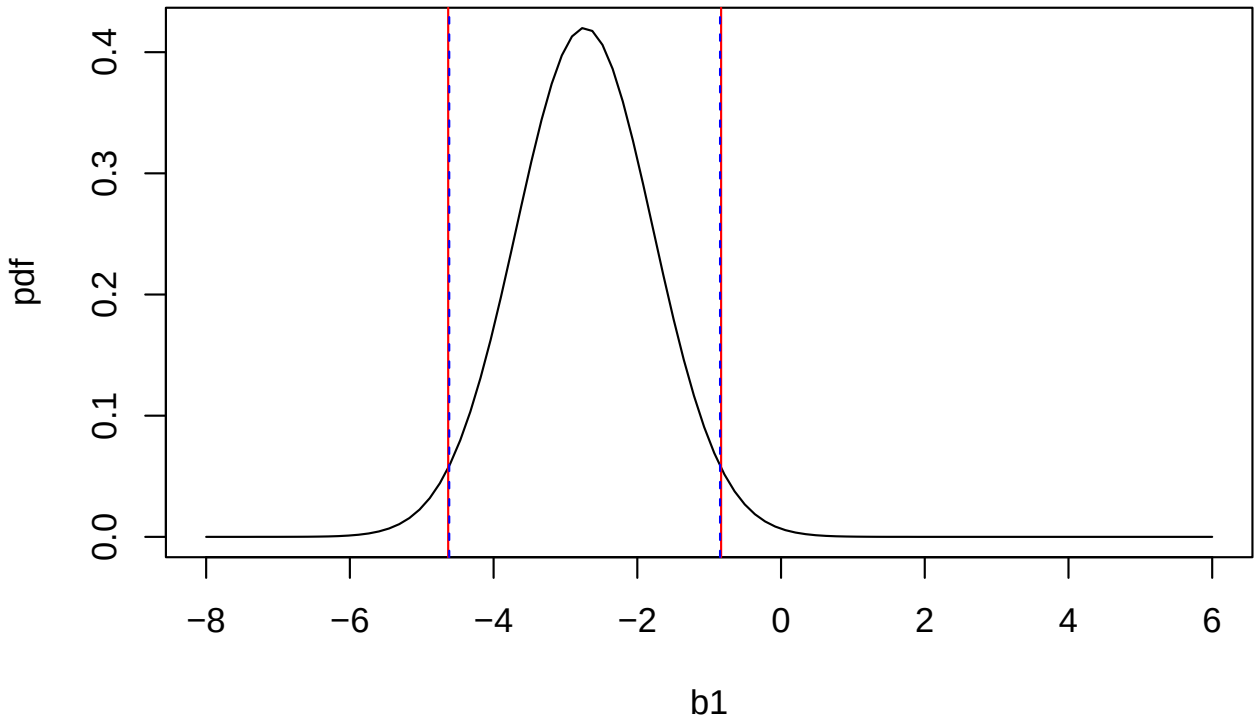
```
b1 = seq(-8,6,length.out=100)
pdf = dnorm(b1,summary(mod)$coefficients[2,1],summary(mod)$coefficients[2,2])
plot(b1,pdf,'l',ylab='density')
```



And if we wanted to 95% calculate confidence intervals for b_1, using both the 2 SE method and the more exact way:

```
# 2SE method
lower_CI_2SEmethod <- summary(mod)$coefficients[2,1] - 2 * summary(mod)$coefficients[2,2]
upper_CI_2SEmethod <- summary(mod)$coefficients[2,1] + 2 * summary(mod)$coefficients[2,2]
```

```r
# more precise method
deg_free = 98 # 100 data points - 2 parameters
crit_value <- qt(0.975,deg_free)

lower_CI_exact<- summary(mod)$coefficients[2,1] - crit_value * summary(mod)$coefficients[2,2]
upper_CI_exact <- summary(mod)$coefficients[2,1] + crit_value * summary(mod)$coefficients[2,2]

plot(b1,pdf,'l')
abline(v=lower_CI_2SEmethod,col='red')
abline(v=upper_CI_2SEmethod,col='red')
abline(v=lower_CI_exact,col='blue',lt='dashed')
abline(v=upper_CI_exact,col='blue',lt='dashed')
```

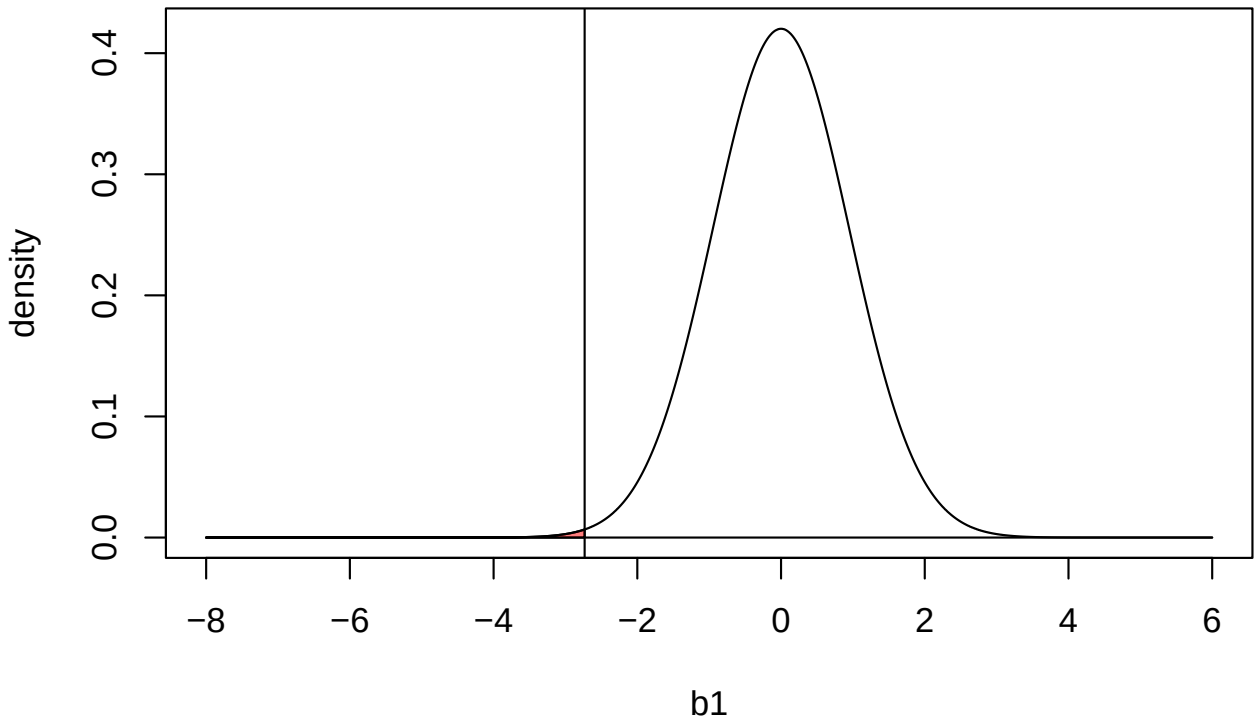


We see first, that the 2 SE method (red) is a very close approximation of the exact method (blue) for calculating 95% confidence intervals. We also see that the 95% confidence intervals do not include 0. Thus we can be 95% confident that the true population is between -4.62 and -0.85, and this range does not include the null hypothesis. This actually already tells us that we will reject the null hypothesis, with an $alpha$ of 0.05. But below we will go through and show how to calculate the p-value.

So we have estimated our parameter of interest it's uncertainty (standard error). The logic of null hypothesis testing is then to say, lets assume for a moment the null hypothesis is true $b_1 = 0$. Given how much uncertainty there is in the parameter value (determined by its standard error) how likely would it have been to observe a b_1 as far or further from zero than the one we observed. You can think of this, as taking the estimated probability density function in the figure above and centering it on 0, which is like assuming $b_1 = 0$. In the plot below we see the expected sampling distribution of b_1 under the null hypothesis. We can then compare this distribution to to observed b_1 value in our study (vertical line):

```r
b1 = seq(-8,6,length.out=20000)
est= summary(mod)$coefficients[2,1]
pdf = dnorm(b1,0,summary(mod)$coefficients[2,2])

plot(b1 ,pdf,'l',xlab='b1',ylab='density')
```

```
lines(b1,b1*0)
abline(v=summary(mod)$coefficients[2,1])
polygon(c(b1[b1<=est],est), c(pdf[b1<=est],0), col=alpha("red",.5))
```



The the area of the red shaded region in the plot above corresponds to the probability of observing a b_1 value less than the value measured in our study if the null hypothesis is true ($b_1 = 0$). Or in other words, how likely is it to have observed a difference in the reduction of blood pressure between the drug and placebo group of -2.73 or less due to random chance alone.

We can calculate this value with the pnorm() function, which is just the are under the curve in the plot above up to the observed value of b_1.

```
estimate = summary(mod)$coefficients[2,1]
se = summary(mod)$coefficients[2,2]
pnorm(estimate,0,se)
```

```
[1] 0.001998794
```

4.5 One-tailed and two-tailed tests

Thus the probability of observing a value of b_1 in the study of -2.73 or less is 0.002. This would be the p-value for what is know as a one tailed test null hypothesis test.

A confusing convention with p-values is that they primarily are used as the probability of seeing parameter value being as least as *"extreme"* (different from zero) as the observed value if the null hypothesis is true. This means we need to account for the probability of observing b_1 being less than -2.73 and also the probability of it being greater than 2.73. Because the normal distribution symmetric, the probability of b_1 being more extreme than our estimate b_1 > abs(estimate) is 2 times the probability of b_1 < estimate.

So the p-value for a 2 tailed test would be:

```
2 * pnorm(estimate,0,se)
```

```
[1] 0.003997588
```

This is referred to as a two-tailed test, because we are accounting for probability of observing a b_1 more extreme (further from zero) than the one observed in our study due to random chance. By convention, we mostly use two-tailed tests because they are more conservative (it is harder to reject the null hypothesis). This is despite the fact our hypotheses are often directional. For example in the case of the drug, we are really only interested in the case where it improves blood pressure relative to a placebo, not in cases where it makes patients worse, so it might make more sense to use a one tailed test. However, by convention, most researchers use a 2 tailed and an $alpha$ of 0.05.

You might notice that the p-value we calculated is slightly different than the value reported in the lm() summary table. This small discrepancy is because we used the normal distribution to calculate the probability of observing a more extreme value of b_1, but the more precise way is to use the t-distribution to account for uncertainty in the standard error of the parameter estimate. To calculate the p-value more precisely, we just take parameter estimate and divide by its standard error, which gives us a t-value:

```
summary(mod)$coefficients[2,1]/summary(mod)$coefficients[2,2]
```

```
[1] -2.878352
```

which tells us how many standard errors the estimated parameter value is from zero. We can then calculate the probability of observing a value less than this if the null hypothesis is true using pt() (similar to pnorm, but for the t-distribution):

```
pt(-2.878352,98)
```

```
[1] 0.002453941
```

where 98 is the degrees of freedom of the analysis. If we multiply the value above by 2 to account for both tails of the distribution, get the exact same value as in the lm() output table.

Problem 4.2 A study reported that primary productivity of grasslands increased linearly with the log of species diversity, with a slope of 0.4 and a standard error of 0.25. Approximately what would be the probability of observing this extreme of a slope due to random chance alone?

In the problem above, I didn't specify the degrees of freedom so we will have to settle for approximate answers. If we assume the null hypothesis is true slope = 0, the probability of observing as extreme as 0.4 in a study would be:

```
2 * pnorm(-0.4,0,0.25)
```

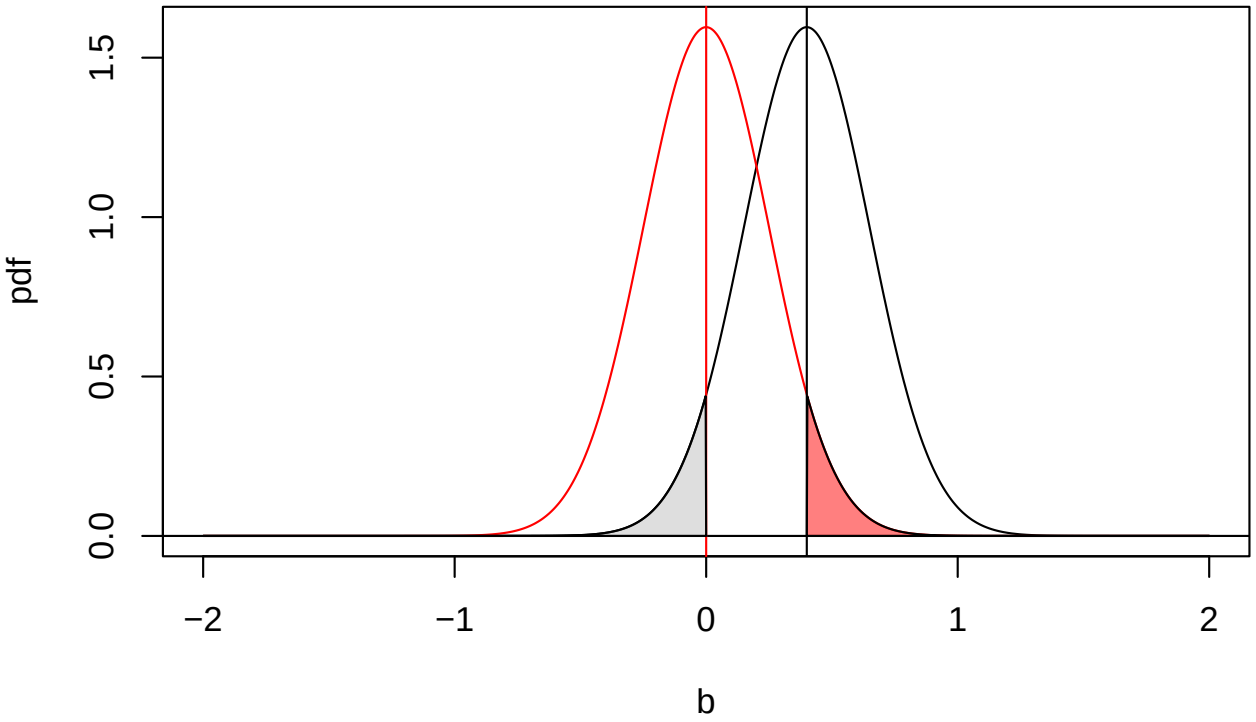
```
[1] 0.1095986
```

Thus the probability of observing a slope as or more extreme than 0.4 if the null hypothesis is true is approximately 11%. Notice that I put a negative sign in front of the 0.4 in my answer. Why did I do this?

4.6 The relationship between confidence intervals and t-tests

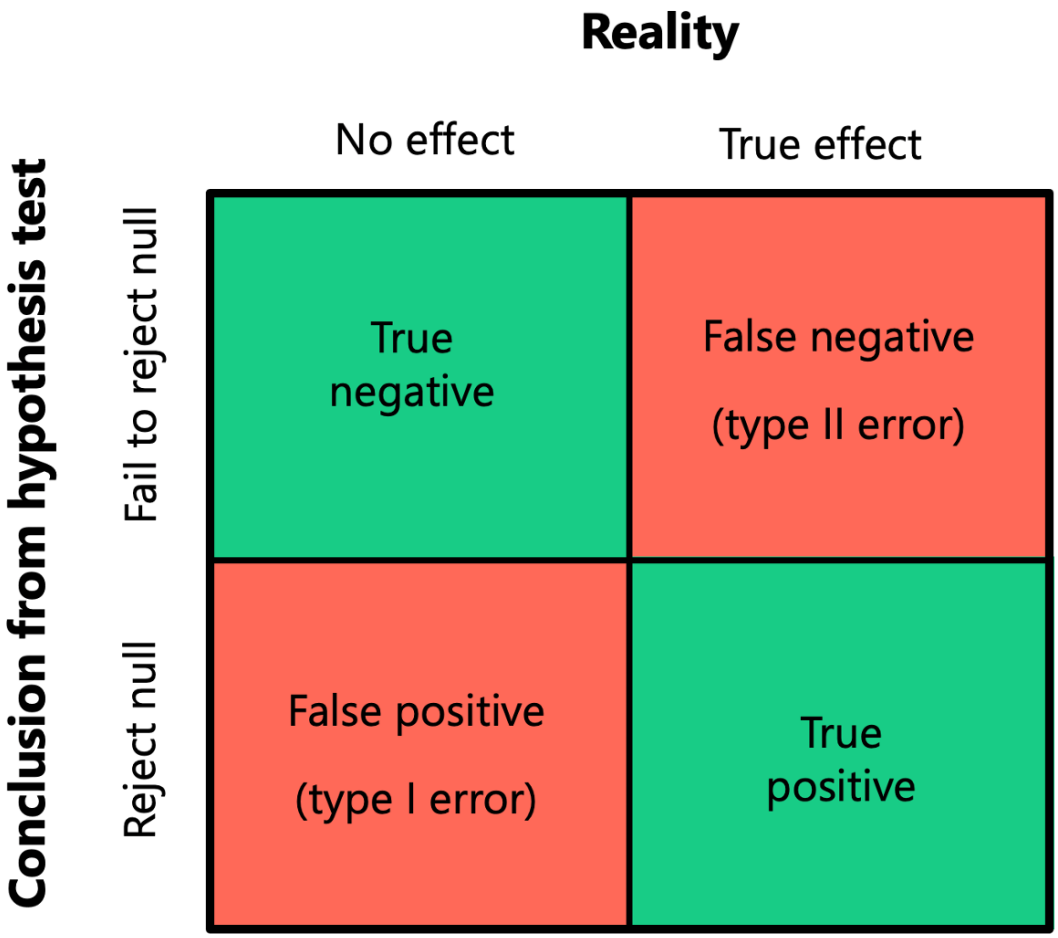
You might have noticed some similarities between t-tests and confidence intervals. In short, if the 95% confidence interval does not include zero, then you will reject the null hypothesis with an $alpha$ of 0.05. With a confidence interval you are calculating a range of parameter values that you are 95% confident will contain the true population parameter. With NHST, you assume the parameter of interest equal zero, and then calculate the probability of observing as extreme of a value as the one observed in your study.

I will illustrate the relationship between confidence intervals and hypothesis test with an example below. Here I assume a parameter of interest was estimated to be 0.4 with a standard error of 0.25. The black line shows the probability density function for the population parameter given the observed data set. The grey shaded region corresponds to our confidence level that the true population parameter is less than zero. The red line shows the expected sampling distribution if the null hypothesis is true ($b_1 = 0$). The shaded red region is the probability of observing a parameter value greater than 0.4 if the null hypothesis is true. Because the width of the red and black distributions are equal (determined by the standard error of the parameter value) the area of these shaded regions are identical. For a two-tailed test with an $alpha$ of 0.05, you would reject the null hypothesis if the area of the shaded region is less than 0.025 (divide 0.05 by to to account for both tails of the distribution).



4.7 False positives and negatives

The use of NHST takes as binary view of the world. First there is the truth: is the true population parameter 0 (null hypothesis) or not? Secondly, there is the conclusion from our study: We either reject null hypothesis or we fail to reject it. These give four possible outcomes:



The good outcomes: There are true positives: you reject the null hypothesis and in reality the null hypothesis is not true There are true negatives, you fail to reject the null hypothesis and in reality the null hypothesis is true

The bad outcomes: False positive (type I error): you reject the null hypothesis when in reality the null hypothesis is true False negative (type II error): you fail to reject the null hypothesis when in reality the null hypothesis is not true

A useful analogy is with a covid test. You either have covid or you do not. If the test comes out positive when you don't have covid this is a false positive and if the test comes out negative when you have covid this is a false negative.

We often want to know the probability that a given NHST is a false positive or false negative, however in practice it is impossible to know. What we can calculate, however, is the probability of a false positive or false negative conditioned on the null hypothesis either being true or false.

4.7.1 False positive rate when null hypothsis is true

If the null hypothesis is true (b = 0) then the probability of a false positive is equal to $alpha$. For example, if we reject the null hypothesis when p < 0.05, and we analyzed many datasets where the null hypothesis is true, we would reject the null 5% of time. These would be false positives because we would reject the null hypothesis even though the null hypothesis is true.

4.7.2 False negatives and power analyses

On the other hand, if there is a true effect, what is our probability of not rejecting the null hypothesis (a false negative)? This will depend on the magnitude of the true population parameter and then standard error, which will decrease

with sample size. An intuitive way to think about this is that if the true parameter is small, we need small standard errors to differentiate that parameter from zero, and doing this requires larger sample sizes.

Another way of looking at this is, if the true population parameter does not equal zero, what is the probability I reject the null hypothesis in my study. This is often referred to as the statistical power of a test:

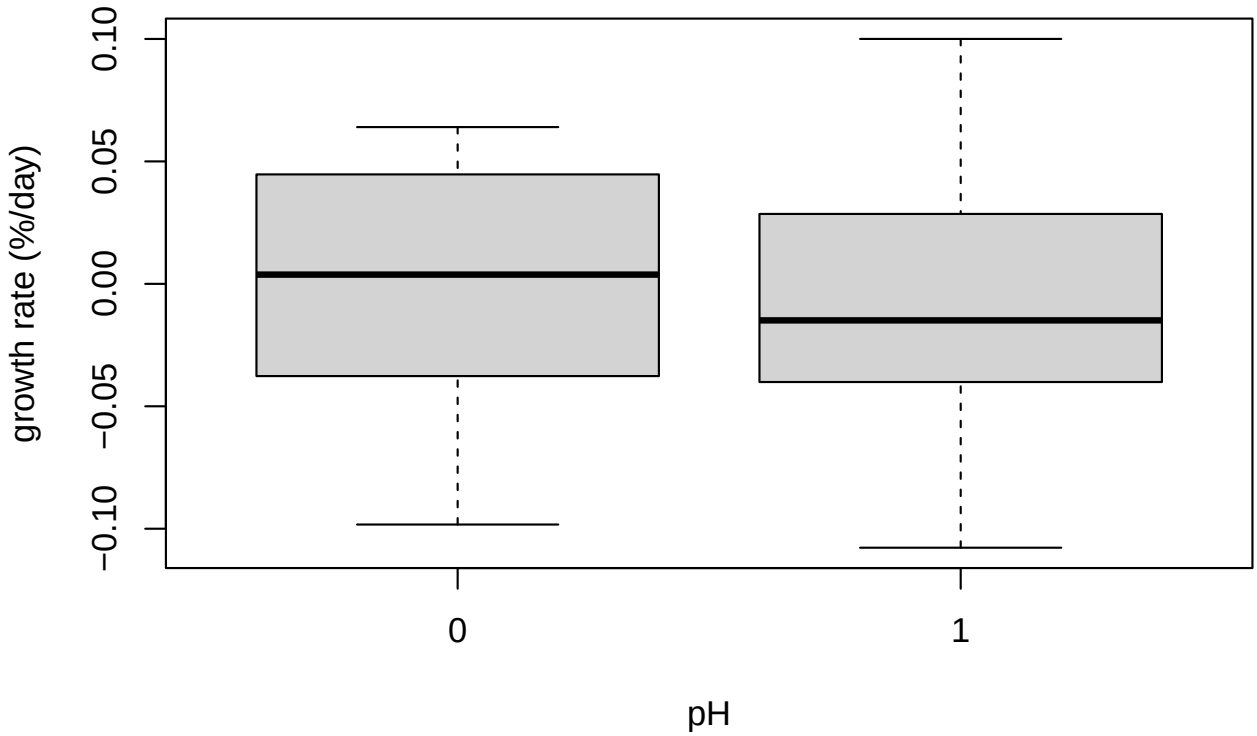
Power = 1 - P(false_negative)

To know the power of a specific test again need to know the true population parameter and the standard deviation of the errors. In practice you will rarely know these values. However, before you do an experiment, you can think about what the minimum effect size (e.g. magnitude of parameter value, which could be a difference between treatments or a regression slope) you would want to be able to detect. Generally, when you conduct an experiment or collect field data, you do it to test a hypothesis that one dependent variable is associated the value of another independent variable. Think about what the minimum effect size would be for your hypothesis to have practical importance. Then think about how noisy you expect the errors to be. This is often best done by looking for similar studies that measured the same dependent variable. Once you have a minimum effect size of interest, and an estimated standard deviation of the errors, you can calculate the power of your study under these conditions. In some cases there are formulas you can use to solve for the statisitcal power of a test, but a very general and easy way to calculate the power of your study is through simulation.

In this example, I want to conduct an experiment to measure the effect of ocean acidification on the growth rate of coral. I have one treatment with current pH levels and one with expected pH levels in the year 2100. My dependent variable is growth rate in % of body weight per day, and the minimum effect size that I think is biologically meaningful is -0.03%. Based on previous studies, I expect the standard deviation of the errors to be 0.05%. How many replicates per treatment would I need to have a 80% of detecting a the true effect?

To answer this question, I first need to write down an equation to generate the data of interest:

```
rep_N <- 10 #number of replicates per treatment
effect_size<- -0.03
sigma <- 0.05
pH<- c(rep(0,rep_N),rep(1,rep_N)) # generate dummy variable for pH (0 = current, 1 = future)
growth<- effect_size * pH + rnorm(rep_N*2,0,sigma)

plot(factor(pH),growth,xlab="pH",ylab="growth rate (%/day)")
```



The variable 'growth' represents the simulated dependent variable, which depends on the treatment (pH level) and an error term. Notice that I didn't add an intercept in this model, because it would not affect the power analysis. If you don't believe this you can adapt the simulation to check. The code above generates one random data set, for which we can fit a linear model and check if we reject the null hypothesis for the difference in growth in current (0) and future (1) conditions. In this case:

```
mod<-lm(growth~pH)
summary(mod)
```

```

Call:
lm(formula = growth ~ pH)

Residuals:
      Min        1Q    Median        3Q       Max
-0.099456 -0.031315  0.000209  0.040760  0.111305

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.001174   0.018311   0.064    0.950
pH          -0.012472   0.025895  -0.482    0.636

Residual standard error: 0.0579 on 18 degrees of freedom
Multiple R-squared:  0.01272,   Adjusted R-squared:  -0.04213
F-statistic: 0.232 on 1 and 18 DF,  p-value: 0.6359
```

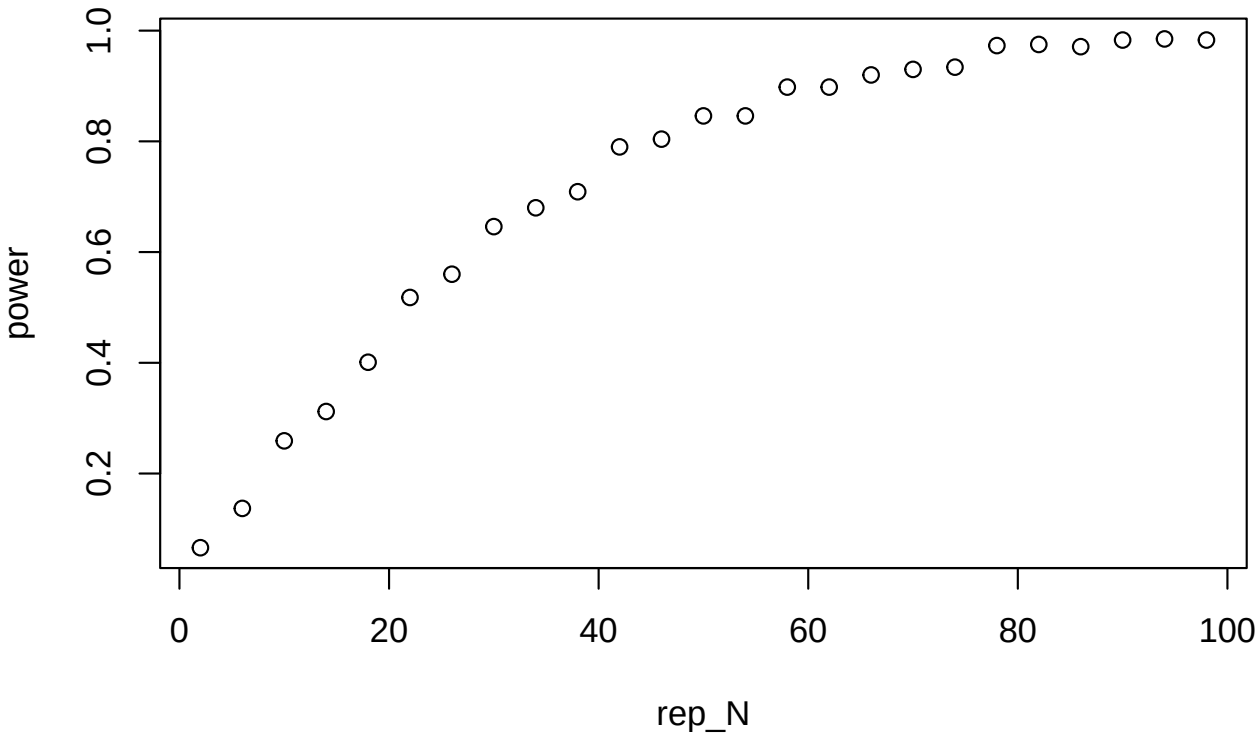
we see that in this simulated experiment, we would have failed to reject the null hypothesis, even though we know there was a true effect in the model used to generate the simulated data. However this is just one simulation, to get estimate the power of this test, we need to simulate many data sets and then calculate the fraction of times we reject the null. I can do this with a for loop:

```
rep_N <- 10
effect_size<- -0.03
sigma <- 0.05
p<-rep(NA,1000)
for (i in 1:1000){
pH<- c(rep(0,rep_N),rep(1,rep_N)) # generate dummy variable for pH
growth<- effect_size * pH + rnorm(rep_N*2,0,sigma)
mod<-lm(growth~pH)
p[i]<-summary(mod)$coefficient[2,4]
}

power = sum(p<0.05)/1000
power
```

```
[1] 0.225
```

In the for loop I generated 1000 datasets, conducted a null hypothesis test with $alpha$ of 0.05, and counted the fraction of datasets where I rejected the null. In this case, with the number of replicates set to 10 per treatment, we only rejected the null hypotheses in about 22% of the experiments, even though we know that there is a true effect. If you think about all the work it takes to conduct en experiment, you probably want to have better than a 25% of detecting an effect in the case that your hypothesis is actually true. This suggests for this study, we should probably use a higher sample size. We can estimate how big of a sample size we need by running the code above with different numbers of replicate per treatment (rep_N), with a nested for loop.

```
rep_N <- seq(2,100,4)
effect_size<- -0.03
sigma <- 0.05
p<-rep(NA,1000)
power<-rep(NA,length(rep_N))
for (j in 1:length(rep_N)){
  for (i in 1:1000){
    pH<- c(rep(0,rep_N[j]),rep(1,rep_N[j])) # generate dummy variable for pH
    growth<- effect_size * pH + rnorm(rep_N[j]*2,0,sigma)
    mod<-lm(growth~pH)
    p[i]<-summary(mod)$coefficient[2,4]
  }
  power[j] <-  sum(p<0.05)/1000
}
plot(rep_N,power)
```



The plot above shows the probability of rejecting the null as a function of the number of replicates per treatment. We can see in this case, that to have a roughly 80% chance of rejecting the null we would over 40 replicates per group! It is always a good idea to run these types of simulations before you conduct your experiment and decide on sample sizes. You can adapt the code above to model and kind of experiment or field survey. If you are going to go through the trouble of running an experiment to test a hypothesis, you probably want to avoid failing to rejecting the null hypothesis in the case your hypothesis was actually right.

4.8 Multiple tests

Many times in scientific papers you see the authors conduct many hypothesis tests in one analysis. This could either be because they test whether many independent variables are associated with changes in a dependent variable, or they might look at how one independent variable affects many dependent variables. An example of the later case would be a paper that looks at how elevated temperatures affect zebrafish, and they measure many different dependent variables to quantify the effect of temperature. For example they could measure growth rates, different aspects of behavior, multiple stress hormones, ect.. The danger with this is that if you conduct enough hypothesis tests you will frequently find 'statistically significant' results, even if the null hypothesis is true.

Problem 4.3 For example, assuming the null hypothesis is true, the probability of rejecting the null with an $alpha$ of 0.05, is 5%. Assuming the null hypothesis is true in each case, what is the probability of observing at least one 'statistically significant' result with an $alpha$ of 0.05, when you conduct 10 null hypothesis significance tests?

It is easier to answer this question if we calculate the probability of not finding the significant result in any of the 10 tests, which is 1 - P(1 or more significant test). The probability of not finding a significant result in one test is $1-alpha$, or in our case, 0.95. If we assume the null hypothesis is true, then the probability of not finding a significant result in one test should be independent of other tests. So to calculate the joint probability not finding a significant result in 10 tests, we just take the product of the individual probabilities:

```
alpha = 0.05
N_tests = 10
(1-alpha)^10
```

```
[1] 0.5987369
```

Thus if we conduct 10 tests, the probability that at least one of them will be significant even though the null hypothesis is true is about 40%. This means you should be skeptical of studies that report many tests, and only 1 or 2 marginally significant results, because you would expect to see results like this even when the null hypothesis true. This means that the probability of a reporting a false positive result in a paper increases with the number of hypothesis tests conducted. An even worse practice is to conduct many test and then only report or focus on the select few that were significant.

One way researchers attempt to reduce the probability of reporting a false positive in a paper is to lower the threshold for declaring a result significant with the number of tests conducted. The simplest of these corrections is called the **Bonferroni Correction** which simply divides $alpha$ by the number of hypothesis tests conducted in the analysis. So if I conducted 10 hypothesis tests, I would divide $alpha$ by 10, and only reject the null when the p value for a test is less than $alpha/10$, or 0.005.

4.9 Common pitfals with the use of NHST

Below I outline some of the most problematic uses of NHST.

4.9.1 Statistical significance does mean biological or practical significance

Just because an independent variable is "statistically significant" does not mean that the variable has any practical importance for the problem at hand. For example, if a study estimates reports that a drug reduces blood pressure by 0.5 mmHg (about 0.5% of a typical value) with a standard error of 0.1, this result is statistically significant, but the effect is so small that it is probably not worth prescribing the drug. Standard errors are inversely proportional to the square root of sample size, thus if we have very large sample sizes, we can detect smaller and smaller "effect sizes". With a large enough data set almost test will become significant, but this does not mean that the variable is practically important.

4.9.2 The null hypothesis is almost always false

This is closely related to another problem in that for most relevant biological questions the null hypothesis that the parameter of interest is exactly zero is almost certainly false. For example if we want to compare some variable of interest between two genotypes, or environments, or management policies the chance that there is exactly no difference between the groups, or a regression slope is exactly zero is probably very low. Thus whether or not we reject the null hypothesis is really just a matter of how much data we collect. This again emphasizes the need to pay attention to effect sizes and the practical significance of our parameter estimates.

4.9.3 Not staistically significant does not mean "no effect"

As shown in the section on false negatives, even when your hypothesis is true, you can fail to reject the null hypothesis. However it is very common for researcher to misinterpret 'non-significant' results to mean no effect. In fact, many studies are underpowered and don't have sample sizes large enough to reliably reject the null hypothesis even when their alternative hypothesis is true.

4.9.4 p-values do not tell you the probability your hypothesis is true

A p value tells you the probability of observing your data/parameter conditional on the null hypothesis being true. 1-p does not, however, represent the probability your hypothesis is true. For example, consider a case where we know that only one gene out of 10,000 genes in a genome controls a specific developmental disorder, but we don't know which gene it is. If we did a screening test for each gene, and then used NHST, we would expect about 500 cases where we reject the null hypothesis at an $alpha$ of 0.05. Yet we know that the hypothesis that a specific gene controls the developmental order is only true for 1 out all of the tests. Thus the our hypothesis is true only for one out of 500 significant tests.

4.9.5 Arbitrary cutoffs

One of my least favorite aspects of statistical significance is that results with p-values less then 0.05 tend to be treated as unambiguous established facts and results with p-values > 0.05 are no true effect. However nothing magical happens as p-values go from 0.0501 to 0.0499. These values are essentially identical, so it seems absurd to come to completely different conclusions for the two cases.

4.10 My advice

The two most important pieces of information you get when you fit a linear model are the values of the parameter estimate and their standard errors. From these two pieces of information you can very closely estimate 95% CI, which also will tell you the result of a NHST that the parameter = 0. In my opinion, p-values and NHST are entirely optional, while effect sizes and standard errors/CI are mandatory when reporting scientific results. Never just say, oh this variable is statistically significant, and nothing else. Remember it is the actual value of the estimated parameter that tells you the practical importance of your finding, and the standard error of a parameter value tells you the uncertainty of that estimate. Always be skeptical if you read in a paper or in the news that a variable is statistically significant when they don't report the effect size. Finally, try to avoid binary decisions. I think it is probably human nature to want to come to unambitious decisions in the face of uncertainty, and in science we also want our conclusions to be reached in an objective way. This helps explain the popularity of NHST in that it provides a way to come to binary conclusions in a way that is perceived as objective. However just because it is conventional practice to declare results with p< 0.05 "statistically significant and p > 0.05 not significant, does not change the fact that p = 0.0499 is effectively identical to p = 0.0501 and it is absurd to come to completely different conclusions for these two cases. It is better to think on a continuous scale of evidence against the null hypothesis.Finally, before you do a study, run simulations with plausible values for parameters and errors. See how large your sample sizes need to be to detect meaningful effects.

4.11 What you should know

1. What is a p-value? How to interpret p-values reported in the lm() output

2. You should be able to conduct a power analysis by simulation

3. You should know the relationship between 95% CI and NHST with an $alpha$ of 0.05

4. You should know that it is never OK just to say "variable X was statistically significant" and not report the effect size and standard error or confidence intervals.

5. You should be aware of how the probability of one or more false positives increases with the number of tests

5 Simulation methods

Previously you might have encountered various types of statistical methods to test hypothesis, such as the t-test the Chi-squared test. Or you might have learned a formula to compute the confidence interval of particular statistics such as the mean or the median. Such methods are part of *analytical statistics*, the branch of statistics that heavily relies on *analytical expressions* to compute, for example, test statistics, standard errors and confidence intervals. These analytical expressions are, however, only valid as long as the assumptions on which they are based are satisfied. For example, a common assumption in analytical statistics is that errors are independent and normally distributed. Whether or not such assumptions are satisfied in particular applications of analytical statistics is often unclear or even doubtful.

Luckily, with the rise in computational power has come the possibility to address many of the problems in applied statistics using an approach, referred to as Monte Carlo simulation, that does not require making the assumptions underlying analytical statistics. The concept of Monte Carlo simulation was developed by the mathematicians Stan Ulam and Nicholas Metropolis. During World War II Ulam and Metropolis were part of the Manhattan Project, the purpose of which was to develop an atomic weapon for the US. One question they had to solve was what the average distance is that a neutron would travel in a substance before it collided with an atomic nucleus. Faced with the situation that this question could not be answered using standard mathematics, Ulam realized that he could use simulations with random numbers to compute the average distance traveled. Just like in a casino game such as a roulette wheel, in such simulations numbers are generated at random and to estimate the probability of a specific outcome, one could play the game many, many times. The story goes that the technique is named after the Monte Carlo casino in Monaco, where Ulam's uncle used to gamble.

Nowadays Monte Carlo simulation techniques, or simulation techniques for short, are pivotal in many statistical studies. Simulation approaches can be used for different purposes:

- To estimate the ***accuracy*** or precision of sample statistics
- To perform ***significance tests***
- To validate models by using ***random subsets*** of a data set
- To analyse processes, for example ***to evaluate a particular experimental design*** or ***to estimate parameters***

All these approaches have a common theme: a particular process is simulated repeatedly very many times, every time with a new set of random numbers or a new random selection of a particular data set. These many repetitions provide a distribution of different outcomes, much like sampling a particular population many times provides a distribution of outcomes. The distribution of the simulation outcomes can be exploited to test hypothesis or compute confidence intervals.

Simulation methods in statistics encompass a suite of different approaches, each with their own names:

- **Bootstrapping and Jackknifing:** Bootstrapping is used to estimate the *sampling distribution* of a particular estimator by sampling with replacement from the original sample. This procedure allows for the estimation of standard errors and confidence intervals of a population parameter like a mean, median, or regression coefficient. Jackknifing is similar to bootstrapping and is used to estimate the bias and standard error (variance) of a statistic. The basic idea behind the jackknifing is to systematically recompute the estimate of a particular statistic (such as mean or median), leaving out one or more observations at a time from the sampled data set. Given a sample of size n, a distribution of a particular statistic can for example be created from all subsamples of size $n - 1$ obtained by omitting one observation. This new set of replicates of the statistic provides an estimate of the bias and the variance of the statistic.
- **Randomization or permutation tests:** These are methods used for hypothesis testing. For example, to test the significance of an observed statistic, such as a mean difference between 2 groups, the data is randomly shuffled and the statistic is computed anew for the shuffled data. Comparison of the originally observed value of the statistic with the distribution after reshuffling provides a measure of the likelihood that the observed value is significant or could be the outcome of a random process.
- **Cross-validation:** Cross-validation is a statistical method for validating a predictive model. Given a data set, a model is fit to only part of the data (the so-called training data). The fitted model is subsequently used to predict

the other part of the data set that was not used for fitting the model (the validation set). An overall measure of the prediction accuracy of the model can then be obtained from the fit of the model to the validation data.

- **Monte Carlo methods:** Monte Carlo methods, or Monte Carlo experiments, are used to repeatedly simulate a particular process, each time with different random values, to arrive at a distribution of possible outcomes of the process.

In the following we will discuss bootstrapping, hypothesis testing through randomization and Monte Carlo process simulation. However, before doing so and because all the methods rely on generating random numbers, we will first discuss the pseudo-randomness of random numbers.

5.1 Random numbers and how to generate them

Random numbers are crucial for any simulation method, but unfortunately true random numbers can only be generated by a physical process, such as throwing a dice or flipping a coin. The best a computer can do is to generate a sequence of ***pseudo-random numbers***, which are not random at all. In fact, the random numbers that can be generated by a computer program, whether it is R or any other program, are produced by an algorithm that necessarily repeats after a specific number of times. Different types of algorithms exist to generate pseudo-random numbers and R implements at least 7 different options, which you can explore by typing the command `?Random`. The default random number generator that R uses us called the *Mersenne-Twister*, which method repeats after exactly $2^{19937} - 1$ times.

In R the function `runif(n)` uses the selected random number generator to produces n different values between 0 and 1 (additional arguments `min` and `max`) can be supplied to the `runif()` function to change the range from which the values are selected). For example:

```
runif(6)
```

```
[1] 0.25699945 0.63990908 0.50690479 0.96627805 0.67166028 0.08302234
```

generates 6 values between 0 and 1 that at first sight look pretty random. The sequence is moreover hard to predict. Repeating the same call illustrates the randomness

```
runif(6)
```

```
[1] 0.8551046 0.4393758 0.5203980 0.9603904 0.5464044 0.5481606
```

and yields an entirely different sequence of 6 random values between 0 and 1.

However, the randomness is elusive, because the sequence of values that is generated is uniquely determined by the ***seed value*** of the random number generator. This seed value can be set using the function `set.seed()` in R. Let's repeat the previous two calls to `runif()`, while preceding each call with a call to `set.seed()` with the same value:

```
set.seed(seed = 567852)
runif(6)
```

```
[1] 0.1557190 0.6929711 0.4286529 0.6973091 0.1693208 0.9255609
```

Setting the seed value to 567852 is completely arbitrary here, we could have chosen any other value. But let's see what happens if we repeat the previous call:

```
set.seed(seed = 567852)
runif(6)
```

```
[1] 0.1557190 0.6929711 0.4286529 0.6973091 0.1693208 0.9255609
```

You can see that we obtain exactly the same random number sequence in the two calls, because we set the seed value of the random number generator to the same value. Similarly, if we set the seed value once and generate 2 sequences of 6 random numbers using 2 calls to the `runif()` function:

```
set.seed(seed = 567852)
runif(6)
```

```
[1] 0.1557190 0.6929711 0.4286529 0.6973091 0.1693208 0.9255609
```

```
runif(6)
```

```
[1] 0.06594427 0.03946674 0.84221252 0.74674796 0.49504277 0.12987301
```

we obtain the same sequence of 12 numbers as when the seed value would be set and the `runif()` is called only once to generate the 12 random numbers all at the same time:

```
set.seed(seed = 567852)
runif(12)
```

```
 [1] 0.15571901 0.69297114 0.42865290 0.69730914 0.16932079 0.92556089 0.06594427
 [8] 0.03946674 0.84221252 0.74674796 0.49504277 0.12987301
```

Notice how the last 6 numbers in this last sequence are indeed the same as the numbers generated by the call to `runif(6)` just before that.

A sequence of random numbers generated by a computer is therefore not random at all, but is *uniquely* determined by the seed value of the algorithm that generates the sequence of numbers. This unique relationship between the seed value of the random number generator and the sequence of random numbers can be usefully exploited if we want to repeat a particular simulation of a process with *exactly the same sequence of random numbers*.

5.2 Bootstrapping

Bootstrapping is a simulation-based method to estimate the uncertainty of a particular statistical quantity. In turn, such an estimate of the uncertainty of the statistic can be used to derive a confidence interval for the quantity. As opposed to other approaches to compute confidence intervals, bootstrapping does not make any assumption about the underlying distribution of the statistical quantity.

5.2.1 Confidence interval of the median

To illustrate bootstrapping we focus on computing the median of a particular data set. One of the data sets that is standard available in R is called `faithful`. This data set contains 272 observations on the duration of the eruptions for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA (see Figure 5.1) and associated with the eruption duration the waiting time till the next eruption.

Figure 5.1: The Old Faithful geyser in Yellowstone National Park, Wyoming, USA.

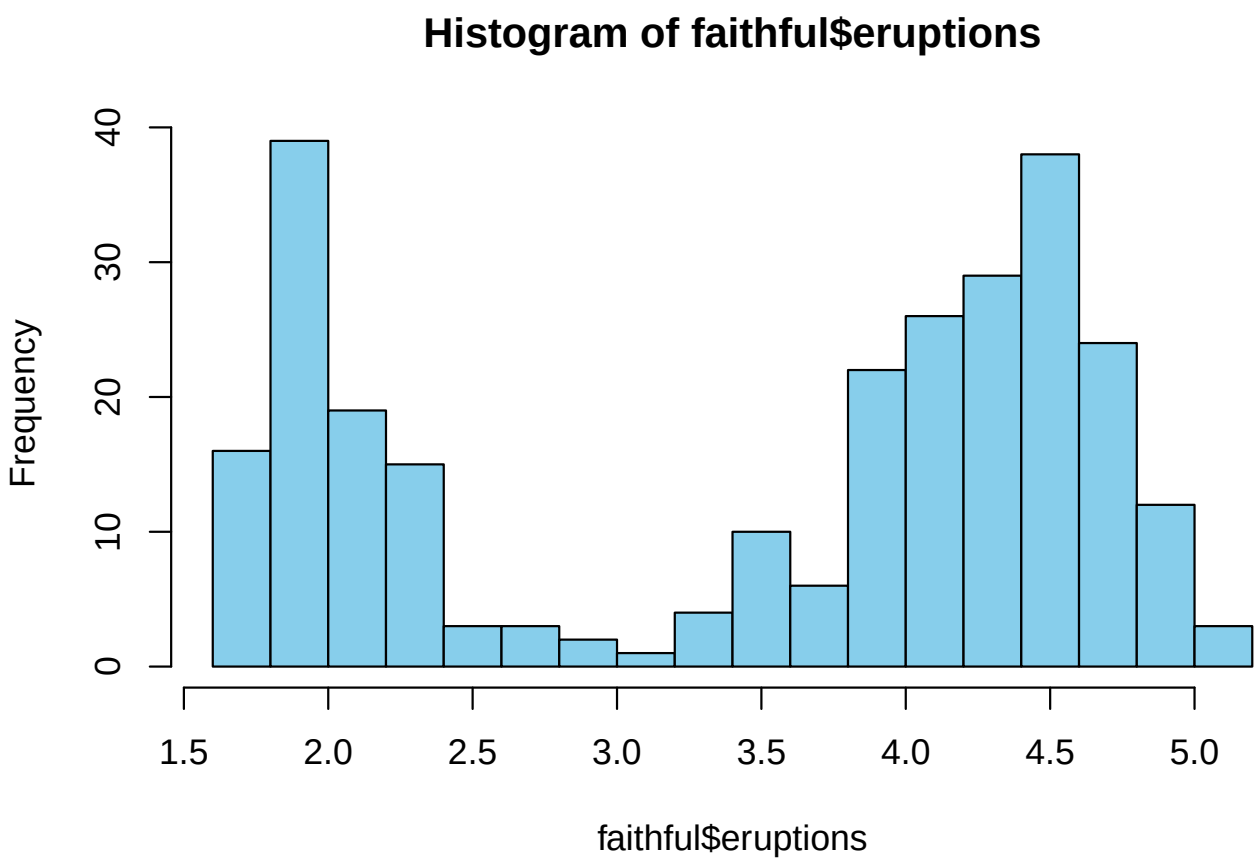
Here you see the first 8 lines of the data set (accessed by using `head(faithful, 8)`). The data set contains 2 columns, the first one called `eruptions` containing the values of the duration of an eruption (in minutes), while the second column called `waiting` contains the values of the waiting time till the next eruption occurs (also in minutes).

```
head(faithful, 8)
```

```
  eruptions waiting
1     3.600      79
2     1.800      54
3     3.333      74
4     2.283      62
5     4.533      85
6     2.883      55
7     4.700      88
8     3.600      85
```

Let's focus for the time being on the duration of the eruptions. Here is a histogram of the duration of the eruptions of Old Faithful:

```
hist(faithful$eruptions, breaks=20, col="skyblue")
```



The distribution is clearly bimodal with one peak around a duration of 2 minutes and a second peak around a duration of 4.5 minutes. The median value of the eruptions is 4:

```
median(faithful$eruptions)
```

```
[1] 4
```

Notice that this is the sample median. The question to answer is how we can compute a confidence interval for the median. Given that the distribution is bimodal, methods to compute confidence intervals that are based on an assumption of normally distributed data would obviously be inappropriate. Here bootstrapping is highly valuable and applicable.

The idea of bootstrapping is that we use the data set and treat it like it is a completely unbiased representation of all the eruption durations of Old Faithful. In other words, we adopt the data set not as a *sample* of a particular population of values but as the *entire population of values itself*. From this data set we then resample the same number of values as the data set itself contains, ***but we do this sampling with replacement***. The sampling with replacement is here crucial, because we would otherwise end up with a resampled, artificially constructed data set that is identical to the original one. Here are the two statements to construct a new, sampled data set from the Old Faithful data set:

```
nsample = nrow(faithful)                          # Determine the length of the data set
X = sample(faithful$eruptions, nsample, replace=T)    # Resample with replacement
```

For the resampled, artificially constructed data set we can again compute the median value, which may or may not be slightly different than the original value. To illustrate that we repeat the resampling and computation of the median 2 times:

```
median(X)
```

```
[1] 4.075
```

```
X = sample(faithful$eruptions, nsample, replace=T)    # Resample with replacement
median(X)
```

```
[1] 3.9665
```

```
X = sample(faithful$eruptions, nsample, replace=T)    # Resample with replacement
median(X)
```

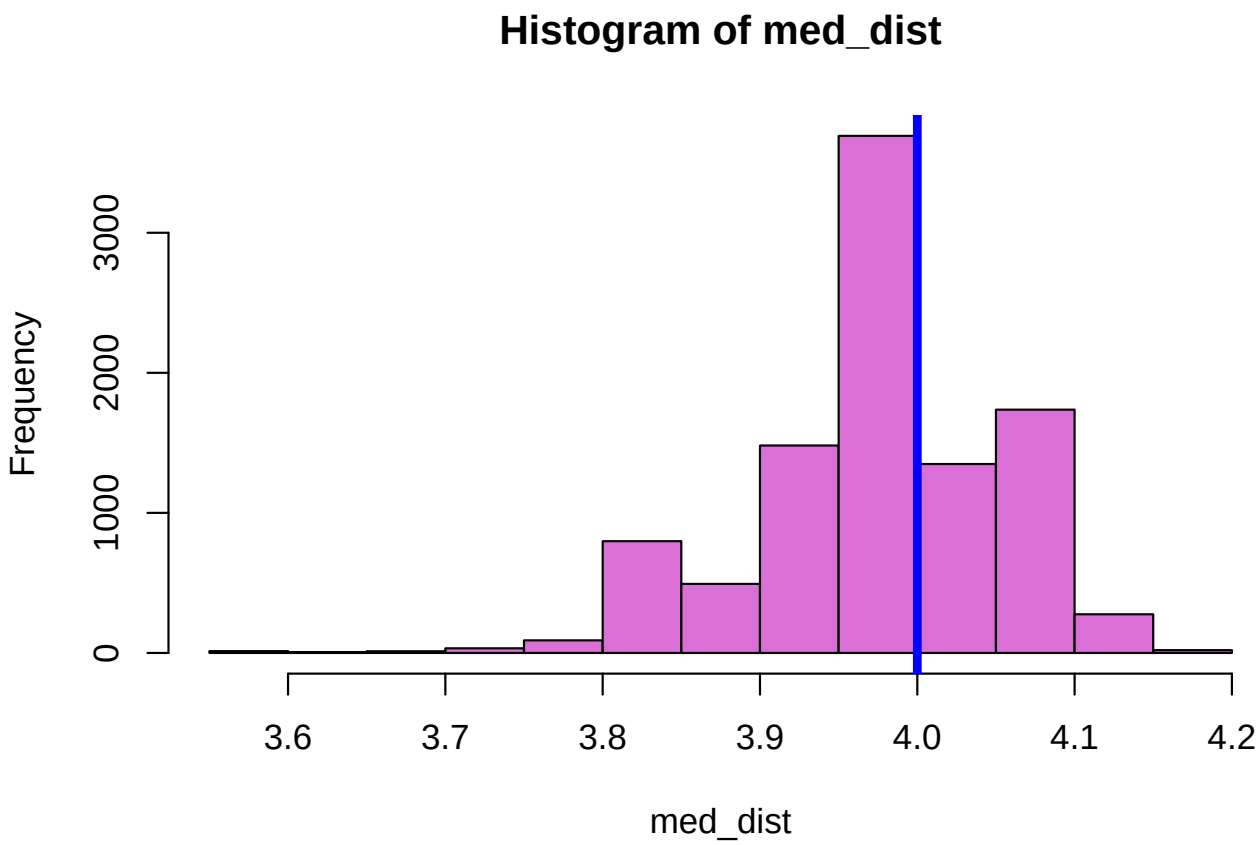
```
[1] 3.833
```

As you can see, the 3 resampled data sets all provide a different estimate for the median value of the duration. We can construct a distribution for these estimates of the median value by repeating the resampling with replacement and the computation of the median value not just 3 times, but very many times. In the code below, an empty vector `med_dist` is constructed of a length `N=10000` and a `for` loop is used to resample the original data set with replacement 10000 times. The median values of these resampled data set are stored in the elements `med_dist[i]` of the result vector.

```
nsample = nrow(faithful)
N = 10000                                          # N determines number of resamplings
med_dist = rep(NA, N)                              # Create empty vector for results
for(i in 1:N) {
  X = sample(faithful$eruptions, nsample, replace=T)    # Resample with replacement
  med_dist[i] = median(X)                          # Compute median for this sample
  }
```

The series of median values computed in these iterations (and stored in the variable `med_dist`) we refer to as the *bootstrap median values*. Now plot the distribution of these bootstrap median values and compare it with the median value of the original data set:

```
hist(med_dist, col="orchid")                       # Plot histogram of all results
abline(v=median(faithful$eruptions), col="blue",lwd=4)  # Plot the median of the data
```



This histogram shows that the bootstrap median values are approximately normally distributed. One approach to calculate a confidence interval for the median is to assume that indeed the bootstrap median values are normally distributed and use the techniques discussed in section 3.6 to estimate the confidence interval. A quick and dirty approach to estimate the confidence interval from the bootstrap median values is then to use the mean of their distribution and subtract or add to that mean value 2 times the standard deviation of their distribution. As also discussed in section 3.6, a more accurate estimate of the confidence interval can be obtained by using the quartile function `qnorm()` of the normal distribution. More specifically, multiplying the standard deviation of the bootstrap median values by `qnorm(0.025)` and `qnorm(0.975)`, respectively, and adding the result to the mean of the distribution gives the range of values that contains 95% of the median values that are obtained from the resampled data sets. The results in the following confidence interval for the median value of the eruption duration of Old Faithful:

```
mean(med_dist) + c(qnorm(0.025), qnorm(0.975)) * sd(med_dist)
```

```
[1] 3.830590 4.139324
```

One disadvantage of using this approach to estimate a confidence interval is that it requires again an assumption that the bootstrap median values are normally distributed. This type of bootstrap is also referred to as *Normal* bootstrapping as it depends on this assumption of normality. An alternative approach to calculate a confidence interval from the bootstrap median values is to sort all the bootstrap median values from low to high and then select the element $0.025 \cdot N$ (= 250 in the example where $N = 10000$) as lower bound of the confidence interval and the element $0.975 \cdot N$ (= 9750 in the example where $N = 10000$) as upper bound of the confidence interval on the grounds that 95% of the computed bootstrap median values fall in between these two limits. This approach to compute the confidence interval is referred to as *Percentile bootstrapping* because it uses the percentile of bootstrap median values that is below the lower threshold and above the upper threshold as determinant for the confidence interval. The result is the following:

```
med_dist_sorted = sort(med_dist)
c(med_dist_sorted[0.025 * length(med_dist_sorted)], med_dist_sorted[0.975 * length(med_dist_sorted)])
```

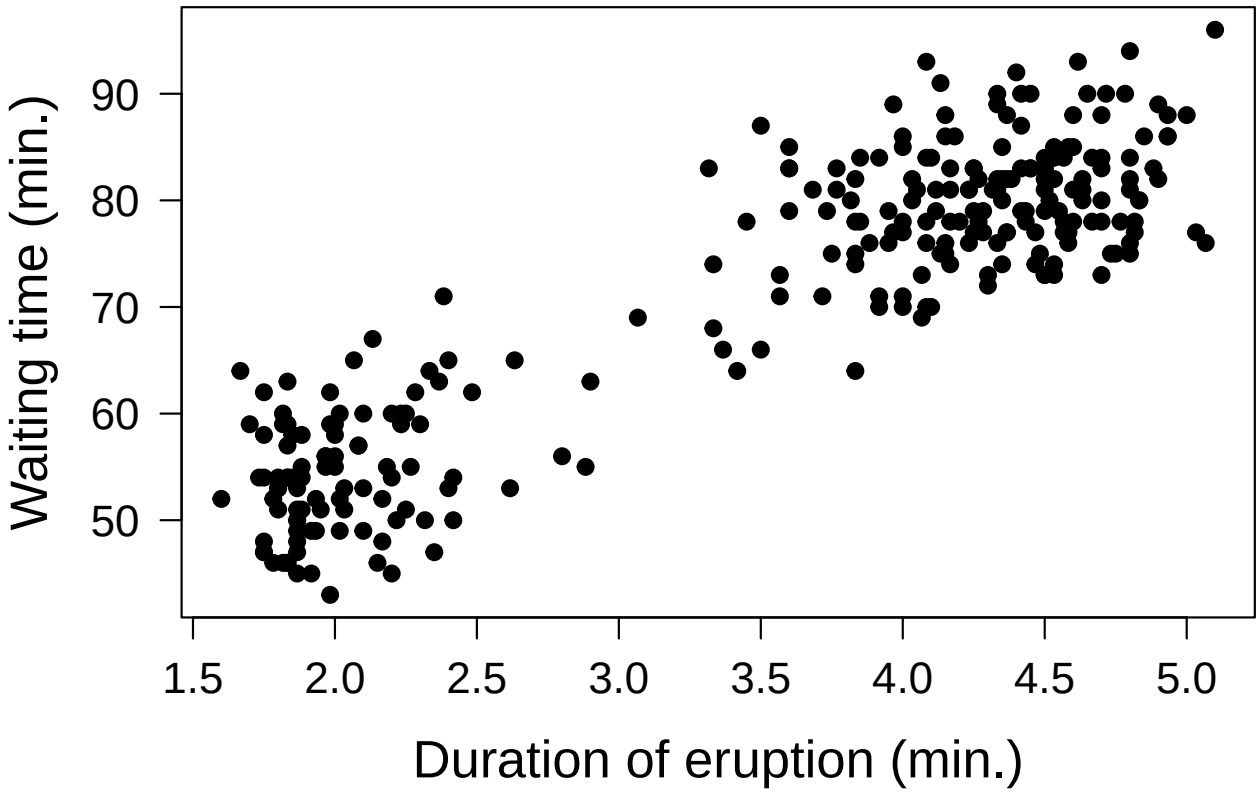
```
[1] 3.8330 4.1085
```

As can be seen the confidence interval estimated using *Percentile bootstrapping* is slightly narrower then the confidence interval estimated using *Normal bootstrapping*. *Percentile bootstrapping* is also preferable over *Normal bootstrapping* as it results in asymmetric confidence intervals. As can be seen from the last displayed results, the lower bound of the confidence interval is 0.167 below the median value computed from the Old Faithful geyser data (`median(faithful$eruptions) = 4`), while the upper bound is only 0.108 higher than this median.

5.2.2 Confidence interval for other statistics

As stated in the beginning bootstrapping can be used to estimate the confidence interval for any statistic. Consider, for example, the correlation between the duration of an eruption of Old Faithful and the waiting time till the next eruption of the geyser. Plotting these two variables against each other clearly shows that after an eruption that has lasted long, the waiting time till the next eruption is also long, as shown in the following plot:

```
par(mar = c(6,6,2,1))
plot(faithful, pch = 19, xlab = 'Duration of eruption (min.)', ylab = 'Waiting time (min.)', cex.lab =
    1.5, cex.axis = 1.25, yaxt = 'n')
axis(2, las = 2, cex.axis = 1.25)
```



We can now compute the correlation between these two variables using the function `cor()`:

```
cor(faithful)                                    # Correlation for the sample data
```

```
          eruptions   waiting
eruptions 1.0000000 0.9008112
waiting   0.9008112 1.0000000
```

The result of `cor()` is a 2×2 matrix with correlation coefficients. The diagonal elements of this 2×2 matrix equal 1 (implying perfect correlation) since any variable is of course perfectly correlated with itself. The two off-diagonal elements are equal as they both represent the correlation between the duration of an eruption and the waiting time till the next eruption. We can now compute a confidence interval for this correlation coefficient between the duration of an eruption and the waiting time till the next eruption using bootstrapping in much the same way as discussed in the previous subsection. However, because we are dealing here with *pairs* of data (a pair of a duration and a waiting time), there is a little twist: we have to resample the *rows* of the data set and keep the pair of duration and waiting time in each row always together. The following code lines first determine the number of rows (observation pairs) in the original data set (variable `nsample`) and then sample from all the row indices (`(1:nsample)`) with replacement as many times as their are observation pairs in the original sample (variable `nsample`). The resampled row indices (assigned to variable `rws`) are subsequently used to extract the random selection of paired observations on duration and waiting time from the original data set (variable `cur_sample`). Lastly, the correlation coefficient is computed for this bootstrap sample:

```
nsample = nrow(faithful)                         # Determine the number of rows in the data
rws = sample((1:nsample), nsample, replace=T)    # Resample row indices with replacement
cur_sample = faithful[rws,]                      # Construct the bootstrap sample
cor(cur_sample)[1,2]
```

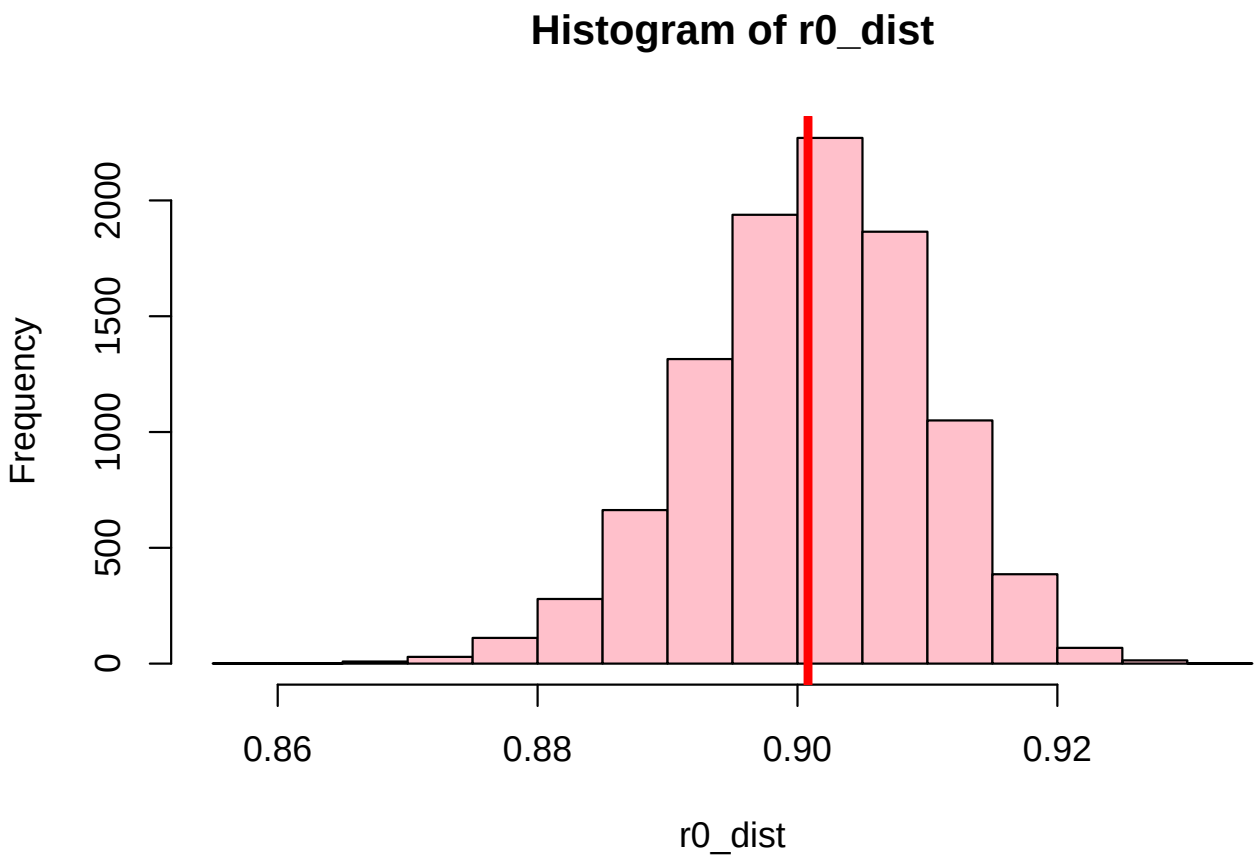
```
[1] 0.9024381
```

As before, this randomization process is repeated many times (here $N = 10000$ times) to compute a distribution of bootstrap correlation values. The resulting correlation values are stored in an array `r0_dist`:

```
nsample = nrow(faithful)
r0 = cor(faithful)[1,2]                          # Correlation for the sample data
N = 10000                                        # N determines number of resamplings
r0_dist = rep(NA, N)                             # Create empty vector for results
for(i in 1:N){
  rws = sample((1:nsample), nsample, replace=T)  # Resample row indices with replacement
  cur_sample = faithful[rws,]
  r0_dist[i] = cor(cur_sample)[1,2]              # Compute correlation for this sample
  }
```

Plotting the distribution of bootstrap correlation values reveals that this distribution again resembles a normal distribution:

```
hist(r0_dist, col="pink")
abline(v=r0, col="red",lwd=4)
```



Sorting the bootstrap correlation values from low to high, the percentile bootstrap method can be used to compute the confidence interval for the correlation coefficient:

```
r0_dist_sorted = sort(r0_dist)
c(r0_dist_sorted[0.025 * length(r0_dist_sorted)], r0_dist_sorted[0.975 * length(r0_dist_sorted)])
```

```
[1] 0.8820164 0.9169430
```

5.2.3 What you should know

1. Bootstrapping is a simulation-based method to investigate the distribution of a particular statistic.
2. The number of repetitions N controls the error of the simulation and hence the level of certainty / uncertainty in the distribution that is derived.
3. For very large N the distribution of the statistic is apprioximately normal
4. This distribution can be used to estimate a confidence interval for the statistic.
5. Despite many variations, all bootstrap approaches look as follows:
   - Obtain data sample $x_1, x_2, \ldots, x_N$ drawn from a distribution F.
   - Define u – the statistic computed from the sample (mean, median, etc).

- Sample $x_1^*, x_2^*, \ldots, x_n^*$ with replacement from the original data sample. Let it be F^* – the empirical (resampled) distribution.
- Repeat N times (N is bootstrap iterations)
- Compute u^* – the statistic calculated from each resample

6. Then the bootstrap principle says that:
   - The empirical distribution from bootstraps is approximately equal to distribution of sample data: $F^* \approx F$.
   - The variation of the statistic computed from the sample u is well-approximated by the variation of the statistic computed from each resample u^*. So, computed statistic u^* from bootstrapping is a good proxy for the statistic of sample data.
   - Sort the empirical distribution from bootstraps F^* from low to high and select those two elements from this sorted list that have 2.5% of all values below and above them, respectively (*Percential bootstrapping*).

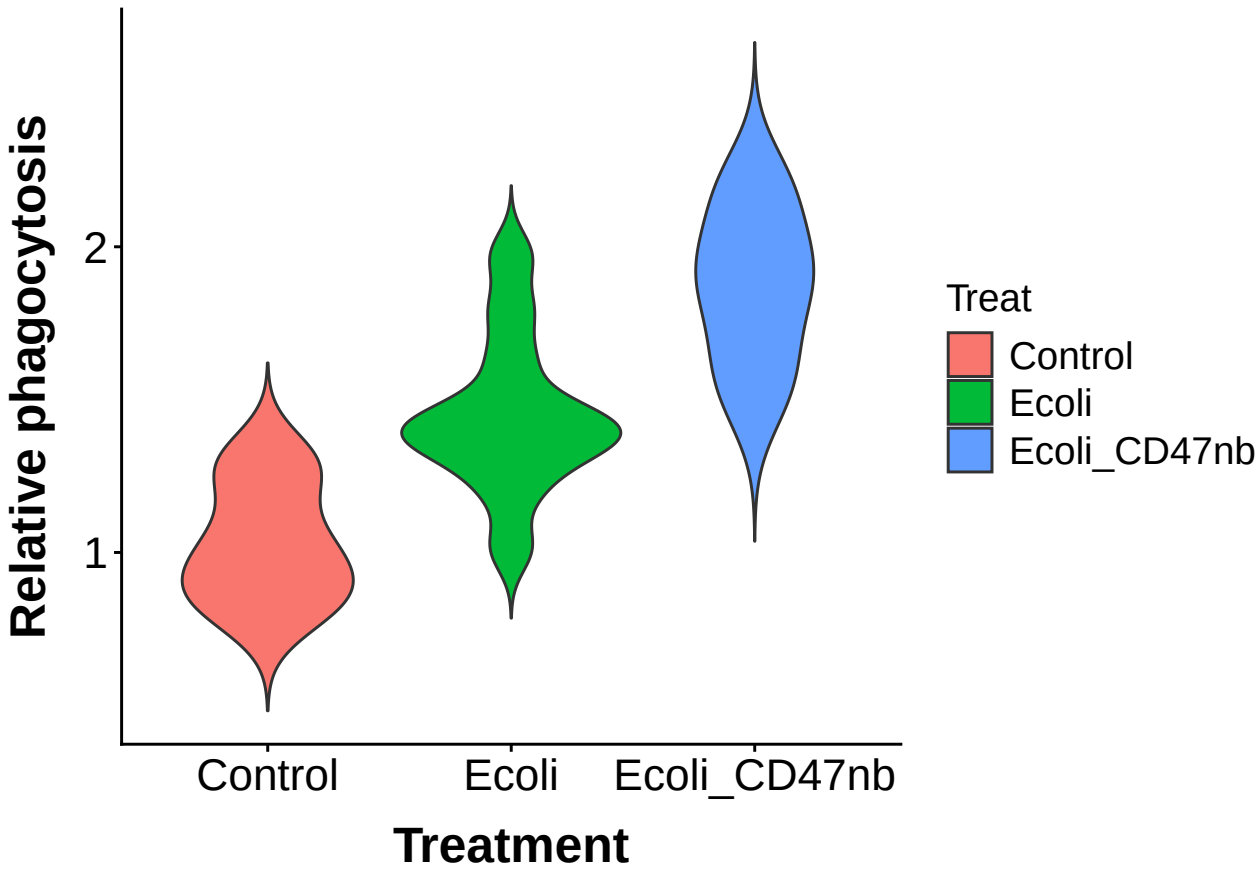
5.3 Hypothesis testing through randomization

A central element of bootstrapping is randomization: the construction of new (bootstrapped) data sets by randomly selecting values from the original data set, but always with replacement sampling. A similar randomization approach, but now *without* replacement, can also be used to test hypothesis. To illustrate this we use a data set presented in Chowdhury et al. (2019).

Many bacteria preferentially grow within tumors and can affect tumor growth by stimulating the immune system to phagocytize tumor cells. An emerging application of synthetic biology is to genetically engineer bacteria to trigger a strong and targeted immune response within a tumor, while avoiding toxicity that occurs with the delivery of therapeutic agents via conventional means. CD47 is an anti-phagocytic receptor overexpressed in some human cancers. Studies have shown that blocking CD47 can increase phagocytosis of cancer cells, however, directly delivering CD47 blocking antibodies can result in anemia due to expression of CD47 on red blood cells. In an experiment, Chowdhury et al. (2019) genetically engineered *E. coli* bacteria (that colonize tumors) to release an encoded nanobody antagonist of CD47 upon lysis. The researchers conducted an in vitro study to measure the ability of this genetically engineered bacteria to increase phagocytosis rates of tumor cells. Replicated culture plates containing macrophages and A20 tumor cells were treated with one of 3 randomly assigned treatments: Phosphate-buffered saline (a control), *E. coli* without the nanobody antagonist of CD47, and the genetically engineered *E. coli* with the CD47 antagonist. The researchers measured the rate of phagocytosis of tumor cells by immune cells before and after the treatments were applied and recorded the fold change in phagocytosis.

Below is a violin plot of the results, which shows for each of the 3 treatments the distribution of relative phagocytosis rate of tumor cells by immune cells:

```
library(ggplot2)
data <- read.csv("data/CD47_data.csv")
# Basic violin plot
p <- ggplot(data, aes(x=Treat, y=Rel_phagocytosis, fill = Treat)) +
       geom_violin(trim = FALSE) +
        labs(x="Treatment", y = "Relative phagocytosis")
p + theme_classic() + theme(axis.text=element_text(size=16),
                            axis.title=element_text(size=18,face="bold"),
                            legend.text=element_text(size=14),
                            legend.title = element_text(size=14),
                            axis.title.y = element_text(margin = margin(t = 0, r = 10, b = 0, l = 0)),
                            axis.title.x = element_text(margin = margin(t = 10, r = 0, b = 0, l = 0)))
```



The width of the violin for a particular treatment indicates the frequency with which a particular value of the phagocytosis rate occurs.

We will here use a randomization test to test whether or not the difference between the wild-type *E. coli* strain and the genetically engineered *E. coli* strain is significant. To this end, only select the data for the two treatments with *E. coli*, ignoring the control treatment:

```
# Exclude all rows of of which the Treat label is 'Control'
Ecoli2 = data[data$Treat != 'Control',]
Ecoli2
```

```
          Treat Rel_phagocytosis
12        Ecoli            1.014
13        Ecoli            1.196
14        Ecoli            1.293
15        Ecoli            1.356
16        Ecoli            1.355
17        Ecoli            1.418
18        Ecoli            1.402
19        Ecoli            1.432
20        Ecoli            1.491
21        Ecoli            1.632
22        Ecoli            1.794
23        Ecoli            1.971
24 Ecoli_CD47nb            1.456
25 Ecoli_CD47nb            1.565
26 Ecoli_CD47nb            1.590
```

```
27 Ecoli_CD47nb            1.704
28 Ecoli_CD47nb            1.817
29 Ecoli_CD47nb            1.951
30 Ecoli_CD47nb            1.951
31 Ecoli_CD47nb            1.853
32 Ecoli_CD47nb            1.979
33 Ecoli_CD47nb            2.170
34 Ecoli_CD47nb            2.161
35 Ecoli_CD47nb            2.249
```

The data can now be analysed with a linear model applied to categorical data, as discussed in chapter 1 and 2. Using the function `lm()` results in:

```
effect0 = lm(Rel_phagocytosis ~ Treat, Ecoli2)
summary(effect0)$coefficients
```

```
                   Estimate Std. Error   t value     Pr(>|t|)
(Intercept)       1.4461667 0.07390557 19.567764 2.102555e-15
TreatEcoli_CD47nb 0.4243333 0.10451826  4.059897 5.211791e-04
```

The fitting of the linear model provides an estimate equal to 0.42433 for the treatment effect of the CD47 genetic engineering. The function `lm()` furthermore provided estimates for the standard error of this treatment effect and the significance level of this treatment effect. This significance test is however based on using the t-distribution, an assumption which may or may not hold for the collected data. To avoid using the assumption that the treatment effect follows the t-distribution we can use a randomization approach to test the significance of the treatment effect of CD47 genetic engineering, as illustrated in this section.

To test the significance of the treatment effect of CD47 genetic engineering, we adopt as null hypothesis H_0 that CD47 genetic engineering of *E. coli* has ***no effect on the relative phagocytosis rate***. If H_0 holds, it implies there is *no relation* between the relative phagocytosis rate measured and the treatment. In other words, under H_0 every value for the phagocytosis rate contained in the data set is equally likely to belong to the *E. coli* treatment or to the CD47 genetically engineered *E. coli* treatment. If the null hypothesis H_0 holds, we can therefore randomly reshuffle the treatment labels of all the observations.

The following lines of code first make a copy of the original data set (`cur_sample`) and then samples from among the row indices ***without*** replacement. The resampled row indices are subsequently used to select the treatment labels at the corresponding rows. These resampled treatment labels are subsequently combined with the original values of the phagocytosis rate:

```
nsample = nrow(Ecoli2)
# Make an exact copy of the data
cur_sample = Ecoli2
# Reshuffle the labels by resampling WITHOUT replacement
cur_sample$Treat = sample(cur_sample$Treat, nsample, replace=F)
cur_sample
```

```
          Treat Rel_phagocytosis
12 Ecoli_CD47nb            1.014
13        Ecoli            1.196
14 Ecoli_CD47nb            1.293
15 Ecoli_CD47nb            1.356
```

```
16        Ecoli            1.355
17        Ecoli            1.418
18 Ecoli_CD47nb            1.402
19        Ecoli            1.432
20        Ecoli            1.491
21 Ecoli_CD47nb            1.632
22        Ecoli            1.794
23 Ecoli_CD47nb            1.971
24 Ecoli_CD47nb            1.456
25        Ecoli            1.565
26 Ecoli_CD47nb            1.590
27        Ecoli            1.704
28 Ecoli_CD47nb            1.817
29 Ecoli_CD47nb            1.951
30        Ecoli            1.951
31        Ecoli            1.853
32 Ecoli_CD47nb            1.979
33 Ecoli_CD47nb            2.170
34        Ecoli            2.161
35        Ecoli            2.249
```

Notice how as a result the labels `Ecoli` and `Ecoli_CD47nb` have been randomly reshuffled in the first column. We subsequently use the function `lm()` to fit a linear model to this reshuffled, randomization data set:

```
res = lm(Rel_phagocytosis ~ Treat, cur_sample)
summary(res)$coefficients
```

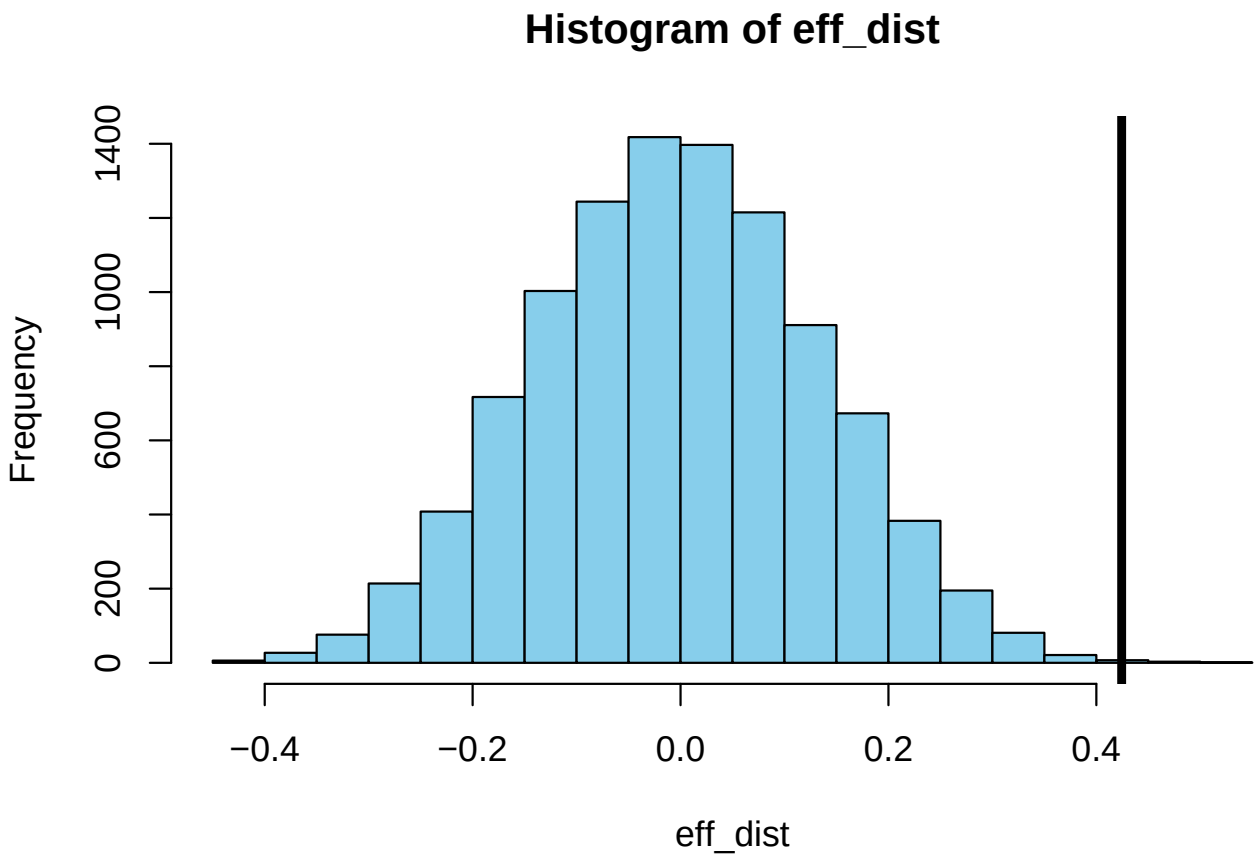
```
                    Estimate Std. Error    t value     Pr(>|t|)
(Intercept)       1.68075000 0.09751203 17.2363354 2.907787e-14
TreatEcoli_CD47nb -0.04483333 0.13790283 -0.3251081 7.481707e-01
```

The coefficient for the treatment effect is now quite different (-0.2913) than for the original data set. However, this value for the treatment effect is one of the possible values that can be observed *if the null hypothesis H_0 holds*. We can now repeat this reshuffling of the treatment labels and refitting of the linear model many times ($N = 10000$ in the example code below) to build up a distribution of possible values for the treatment effect under the assumption that the null hypothesis H_0 holds

```
nsample = nrow(Ecoli2)
N = 10000                                          # N determines number of resamplings
eff_dist = rep(NA, N)                              # Create empty vector for treatment effect
for(i in 1:N) {
  cur_sample = Ecoli2                              # Make exact copy
  cur_sample$Treat = sample(cur_sample$Treat, nsample, replace=F) # Reshuffle the labels!
  res = lm(formula = Rel_phagocytosis ~ Treat, data = cur_sample)
  eff_dist[i] = res$coefficients[2]                # Store the treatment effect for this sample
  }
```

If we now plot the distribution of these randomized values for the treatment effect and compare it with the observed value for the treatment effect we have to conclude that the observed difference between *E. coli* and the CD47 genetically engineered *E. coli* is very unlikely to result from chance alone (without differences between the two types).

```
hist(eff_dist, col="skyblue")
abline(v=effect0$coefficients[2], col="black",lwd=4)
```



In fact, if we simply count how many of the randomized treatment effect values are smaller than the treatment effect of the original data set, we see that more than 99.9% (9993 out of 10000) of the randomized treatment effect values are smaller. In other words, the probability that the null hypothesis H_0 holds is less than 0.001:

```
sum(eff_dist < effect0$coefficients[2])
```

```
[1] 9995
```

For such a randomization test of a particular H_0 hypothesis we are not restricted to using the `lm()` function to compute the treatment effect size. For example, we can also compute the difference between the *E. coli* and the CD47 genetically engineered *E. coli* using the two sample test statistic provided by the function `t.test()`. The function call `t.test(X, Y)` computes for two sets of observations X and Y the value of Welch's t-statistic, defined by the following formula:

$$t = \frac{\overline{X} - \overline{Y}}{\sqrt{s_{\bar{X}^2} + s_{\bar{Y}^2}}}$$

in which $\overline{X}$ and $\overline{Y}$ are the sample means of the data sets X and Y and $s_{\bar{X}}$ and $s_{\bar{Y}}$ are their standard errors, which are computed from the standard deviation of the sample data in the usual manner (see chapter 3.4):

$$s_{\bar{X}} = \frac{s_X}{\sqrt{n}}, \qquad s_{\bar{Y}} = \frac{s_Y}{\sqrt{n}}$$

In these formula n is the length of the data sets X and Y, which should be of equal length.

We can use the function `t.test()` to compute Welch's t-statistic for the data on the phagocytosis rate of wild-type *E. coli* and CD47 genetically engineered *E.coli*:

```
res0 = t.test(Ecoli2$Rel_phagocytosis[Ecoli2$Treat == 'Ecoli'],
              Ecoli2$Rel_phagocytosis[Ecoli2$Treat == 'Ecoli_CD47nb'])
res0
```

```
    Welch Two Sample t-test

data:  Ecoli2$Rel_phagocytosis[Ecoli2$Treat == "Ecoli"] and Ecoli2$Rel_phagocytosis[Ecoli2$Treat ==
    "Ecoli_CD47nb"]
t = -4.0599, df = 21.999, p-value = 0.0005212
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.6410914 -0.2075753
sample estimates:
mean of x mean of y
 1.446167  1.870500
```

The value of the t-statistic equals -4.0599 and the function `t.test()` estimates the probability that there is *no* difference between the two treatments (i.e. that the H_0 hypothesis holds) to equal 0.0005. However, the probability estimate depends on an assumption about the distribution of the computed t-statistic. More specifically, the assumption underlying this probability of 0.0005 is that the t-statistic follows a t-distribution with df = 21.999 degrees of freedom. We can however also use a randomization approach to estimate the probability that the observed value for the t-statistic of -4.0599 occurs under the H_0 hypothesis. This randomization approach would not require the assumption that the statistic follows the t-distribution.

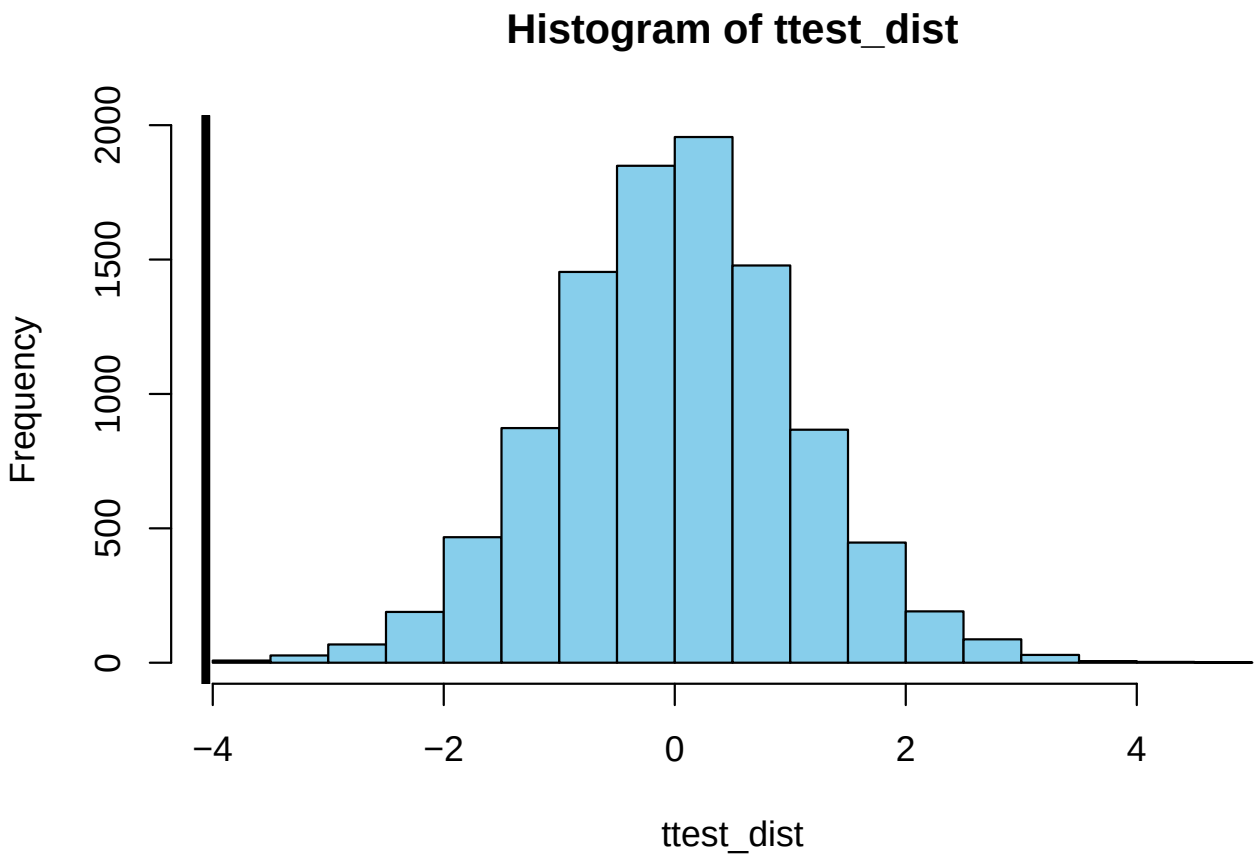
For the randomization approach to test the H_0 hypothesis we follow exactly the same procedure as discussed above when we were using the `lm()` function. Randomized samples are generated from among the row indices of the original data through sampling without replacement. The resampled row indices are subsequently used to select the treatment labels at the corresponding rows and combined with the original values of the phagocytosis rate. The value of the t-statistic is subsequently calculated for this resampled data set. By repeating this procedure many times (here $N = 10000$ times) we construct a distribution of the t-statistic under the assumption that the H_0 hypothesis is true.

```
nsample = nrow(Ecoli2)
# N determines number of resamplings
N = 10000
# Create empty vector for intercept
ttest_dist = rep(NA, N)
for(i in 1:N) {
  # Make exact copy
  cur_sample = Ecoli2
  # Reshuffle the labels by resampling WITHOUT replacement
  cur_sample$Treat = sample(cur_sample$Treat, nsample, replace=F)
  res = t.test(cur_sample$Rel_phagocytosis[cur_sample$Treat == 'Ecoli'],
               cur_sample$Rel_phagocytosis[cur_sample$Treat == 'Ecoli_CD47nb'])
  # Store the t-value of the randomized sample
  ttest_dist[i] = res$statistic
```

```
15  }
```

Plotting this distribution together with the value of the t-statistic that was calculated for the original, unshuffled data set, shows once again that the observed difference between the wild-type *E. coli* and CD47 genetically engineered *E.coli* treatment is very unlikely to be observed under the H_0 hypothesis.

```
hist(ttest_dist, col="skyblue")
abline(v=res0$statistic, col="black",lwd=4)
```



In fact, there is only a single value of the t-statistic in the distribution for the randomized data sets that is smaller than the value observed in the original data set, which suggest that the probability the H_0 hypothesis holds equals 0.0001 (1 out of 10000):

```
sum(ttest_dist < res0$statistic)
```

```
[1] 0
```

5.3.1 What you should know

1. Randomization or permutation tests use a simulation approach to test:
   - whether an observed difference between two groups of observations is significant, or
   - whether a correlation or association between two variables is significant
2. Randomization or permutation tests are based on the null hypothesis H_0 that there is ***no difference*** in the outcome between two groups or treatments or there is ***no correlation or association*** between two variables
3. Under the H_0 hypothesis
   - measured values can be randomly reshuffled between two groups or two treatments, or
   - the measured value of one particular variable can be paired with any of the values of the other variable
4. Repeating such a randomization of the data many times builds up a distribution of the test statistic (the treatment effect when comparing two groups or the correlation coefficient between two variables) under the assumption that the H_0 hypothesis holds.
5. Comparing the measured value of the statistic for the original data set with the distribution cnstructed using randomization provides an estimate for the likelihood for the data to occur under the H_0 hypothesis

5.4 Monte Carlo process simulation

Monte Carlo simulation is a very generic name referring to any type of stochastic simulation, usually of a particular process for which the outcome is not known. Monte Carlo simulation is then used to construct a distribution of possible outcomes of the process. In statistical analysis Monte Carlo simulations are especially useful to evaluate an experimental design or to estimate parameters.

5.4.1 Evaluating an experimental design: Blood samples from elephants

The use of Monte Carlo simulation for evaluating an experimental design will be illustrated with a hypothetical example, focusing on elephants living in the Kruger National park in South Africa. Assume that there are approximately 17000 elephants living in Kruger National park (which is not so hypothetical and about right), but that 8% of these elephants are carrier of an infection with the virus *Corona elephanticus* (this is hypothetical for sure). To assess the variability among different strains of the virus *Corona elephanticus* an ecologist wants to collect blood samples of 20 infected elephants. Elephants are captured at random and a fast test is used to determine whether or not the captured elephant is infected. For simplicity we will assume that the test has an accuracy of 100% (although accounting for a smaller accuracy is not too difficult). Every captured elephant is marked, so that the same elephant is not captured again.

The question to answer is now:

How many elephants does the ecologist have to sample to collect blood from 20 infected individuals?

To simulate the sampling process that the ecologist is going to follow we define variables for the total number of elephants (`Ntotal`), the total number of infected elephants (`NtotalInfected`) and the target of infected elephants (`Ntarget`). At the start of the ecologist's sampling campaign, no elephants are captured yet (`Ncaptured = 0`) and hence the number of infected elephants that have been captured also equals 0 (`Ninfected = 0`):

```
# Total number of elephants, the infection probability
# and the target number infected are fixed
Ntotal = 17000
NtotalInfected = 0.08 * Ntotal
Ntarget = 20

# Start with no individuals sampled or infected
Ncaptured = 0
Ninfected = 0
```

The sampling process carried out by the ecologist can now be simulated with a `repeat` loop (see the code below). The first step in this `repeat` loop is to randomly select a single individual from among the elephants that have not been captured yet (and are hence not marked as having been captured). This number of individuals is given by `Ntotal - Ncaptured` and the function `sample()` can be used to select a single individual from this collection. Once a single individual is selected the number of captured individuals is increased (`Ncaptured = Ncaptured + 1`). To determine whether the selected individual is infected or not we draw a single random number between 0 and 1 with a call to the function `runif()`. The fraction of infected individuals among all the individuals that have not been captured yet, equals the ratio of the number of not-captured, infected individuals and the total number of not-captured individuals (`(NtotalInfected - Ninfected) / (Ntotal - Ncaptured)`). If the random number generated with `runif(1)` is smaller than this fraction we assume that the newly captured individual is infected and hence the number of captured, infected individuals is increased (`Ninfected = Ninfected + 1`). The `repeat` loop is stopped using a `break` statement once the number of infected individuals captured equals the target (`(Ninfected >= Ntarget)`).

```
# Repeat the sampling until the target is reached
repeat{
  # Sample 1 individual at the time from the group
  # that has not been sampled yet
  indx = sample((1:(Ntotal - Ncaptured)), 1)
  Ncaptured = Ncaptured + 1

  # Test whether individual is infected
  # runif(1) generates 1 number between 0 and 1
  if (runif(1) < (NtotalInfected - Ninfected) / (Ntotal - Ncaptured)) {
    Ninfected = Ninfected + 1
  }

  # The break command jumps out of a loop
  if (Ninfected >= Ntarget) break
}
Ncaptured
```

```
[1] 203
```

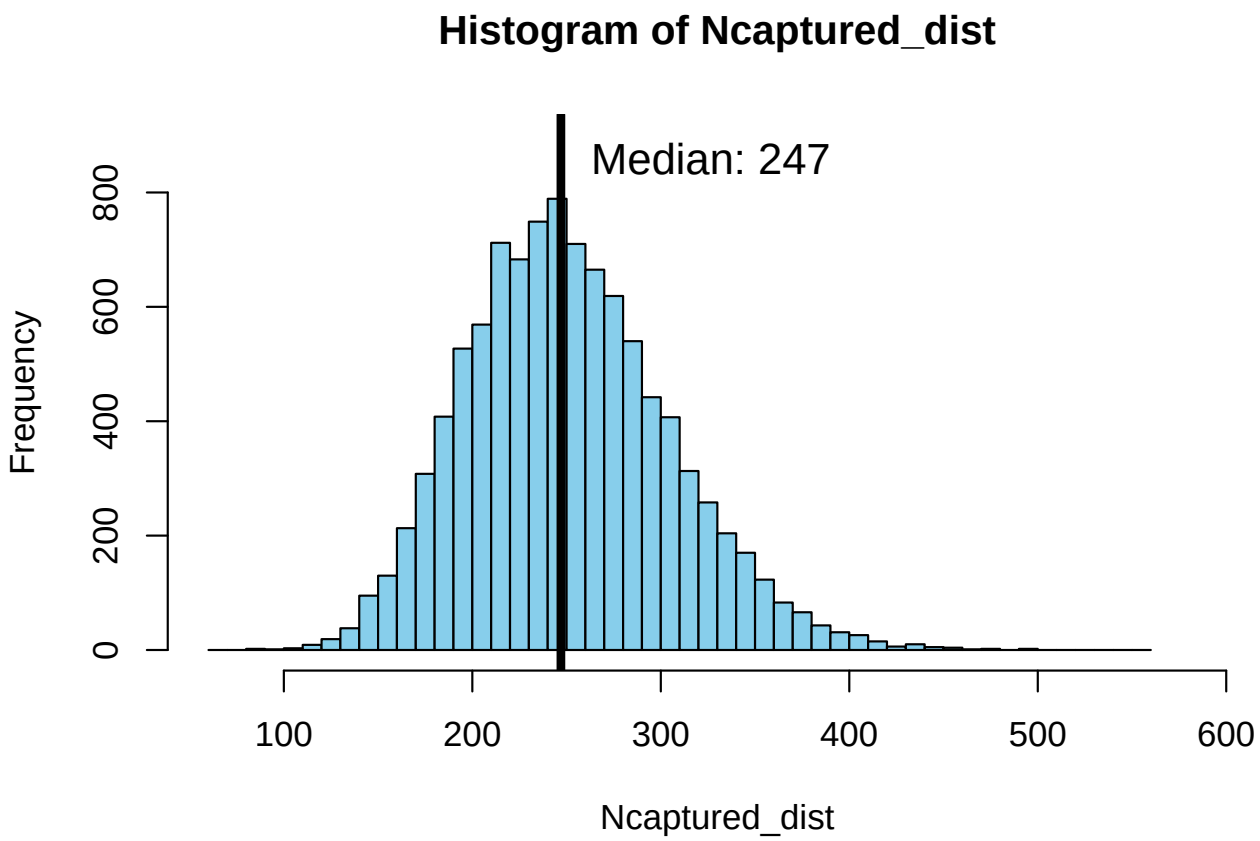
The above code section represents, however, a single realization of the process. In this realization only 203 elephants have to be captured to reach the target of 20 infected individuals, which is quite low. However, the sampling of individuals has a random component and this realization is therefore only one of many different realizations. In other cases we might be not so lucky and would need to capture far more elephants to reach the target of 20 infected individuals. We can now construct a distribution of possible realizations by repeating the above procedure many times a shown in the code section below (for $N = 10000$ times):

```
# Total number of elephants, the infection probability
# and the target number infected are fixed
Ntotal = 17000
NtotalInfected = 0.08 * Ntotal
Ntarget = 20

# N determines number of resamplings
N = 10000
# Create empty vector for results
Ncaptured_dist = rep(NA, N)
```

```
for(i in (1:N)) {

  # Start with no individuals sampled or infected
  Ncaptured  = 0
  Ninfected = 0

  # Repeat the sampling until the target is reached
  repeat{
    # Sample 1 individual at the time from the group
    # that has not been sampled yet
    indx = sample(Ntotal - Ncaptured, 1)
    Ncaptured = Ncaptured + 1

    # Test whether individual is infected
    # runif(1) generates 1 number between 0 and 1
    if (runif(1) < (NtotalInfected - Ninfected) / (Ntotal - Ncaptured)) {
      Ninfected = Ninfected + 1
    }

    # The break command jumps out of a loop
    if (Ninfected >= Ntarget) break
  }
  # Store the current result
  Ncaptured_dist[i] = Ncaptured
}
```

Plotting a histogram of these 10000 different realizations shows that the median value of the number of elephants that have to be captured is 246, as shown in the following plot:

```
hist(Ncaptured_dist, breaks = seq(60, 560, length.out = 51),
     xlim = c(60, 600), ylim = c(0, 900), col="skyblue")
abline(v=median(Ncaptured_dist), col="black",lwd=4)
text(median(Ncaptured_dist) + 5, 850, paste("Median:", median(Ncaptured_dist)),
     pos = 4, cex = 1.25)
```



This median value is close to the expectation that we could intuitively derive by computing $20/0.08 = 250$. However, the Monte Carlo simulation gives us insight into the distribution of possible outcomes and shows that sometimes we might only have to catch just over 100 elephants while in other scenarios it might take as many as 400 elephants to reach the target of 20 infected individuals. The real value of the Monte Carlo simulation approach is this insight into the distribution of possible outcomes.

5.4.2 Estimating parameters: Efficacy of the Pfizer vaccine

To illustrate the use of Monte Carlo simulation for estimation purposes, we will use the test results of a Covid-19 vaccine trial as an example. In a phase 3 trial of the Covid-19 vaccine produced by Pfizer, a randomized, double-blind test was carried out, in which individuals that signed up for the test were randomly given an injection with the vaccine or with a placebo. Neither the individuals receiving an injection nor the doctors administering them knew whether they had received a vaccine or not (hence the adjective "double-blind' '). A total of 43210 individuals received an injection in this phase 3 trial. The company Pfizer had determined that the trial would be stopped and evaluated when a total number of 170 participants in the trial had become infected with Covid-19 and hence tested positive. Participants in the trial were continuously monitored for Covid-19 infection.

After the completion of the trial, when 170 individuals had become infected with Covid-19, it turned out that 21380 individuals had received the vaccine ($N_v = 21380$), while 21830 individuals had received a placebo ($N_p = 21830$). Of the 170 individuals infected with Covid-19, 162 individuals belonged to the group that had received a placebo, while 8 individuals belonged to the group that had received the vaccine.

The question to answer with Monte Carlo simulation is:

What is the most likely value and the confidence interval of the probability that an individual that received the vaccine becomes infected with Covid-19, relative to the infection probability of individuals that did not receive the vaccine?

To start answering this question we set the infection probability of individuals that received a placebo equal to 1, while the infection probability of individuals that received a vaccine is set equal to some value, say $p = 0.04$. This probability that a vaccinated individual gets infected is therefore *relative to the infection probability of an unvaccinated*

individual. Moreover, the value of $p = 0.04$ is just an arbitrary choice, later the computations will be repeated for a range of different p values.

The code shown below computes a single simulated realization of the infection process that could have occurred within this trial for a relative infection probability of vaccinated individuals equal to $p = 0.04$. Check the code line by line while reading the explanation that follows.

In the code the variables `Nplacebo` and `Nvaccine` are set to the number of individuals that received a placebo and a vaccine, respectively, while the variable `Pinfect` is set to the relative infection probability of vaccinated people $p = 0.04$ and the variable `Ntarget` is set to the target value of 170 infected individuals which Pfizer had decided as the threshold value at which the trial would be stopped. The infection process is now simulated using a `repeat` loop that is stopped by means of a `break` statement if the total number of infected individuals equals this target value of 170. The infection process starts out without any infected individuals among the individuals that received a placebo (`NinfectP = 0`) and among the vaccinated people (`NinfectV = 0`).

Within the `repeat` loop as a first step the number of individuals is computed in the placebo and the vaccinated group that have not been infected yet by subtracting from the total number of individuals in these two groups the number of infected individuals in the two groups (`NnotillP = Nplacebo - NinfectP` and `NnotillV = Nvaccine - NinfectV`). The sum of these values gives the total number of individuals that is not infected yet (`Nnotill = NnotillP + NnotillV`). The function `sample()` is used to select the index of a single individual from this group of still uninfected individuals. If this randomly selected index is in the range between 1 and the value of the variable `NnotillP` the selected individual has received a placebo, if the index is larger than the value of the variable `NnotillP` the selected individual has received a vaccine. (As an aside: This choice of the implementation is arbitrary: for the same token we could have assumed that if the index of the randomly selected index is in the range between 1 and the value of the variable `NnotillV` the selected individual has received a *vaccine*, while if the index is larger than the value of the variable `NnotillV` the selected individual has received a placebo). If an individual is selected that has received a placebo it gets infected with probability 1 and the variable `NplaceboP` is increased by 1. For an individual that received a vaccine, the probability to get infected is smaller than 1 and hence we generate a random value between 0 and 1 using the function call `runif(1)`. Only if the result of this call is smaller than the infection probability `Pinfect` ($p = 0.04$) the vaccinated individual gets infected and the number of infected individuals that received a vaccine (`NinfectedV`) is increased by 1. This random sampling of individuals is continued until the total number of individuals that has become infected `NinfectP + NinfectV` reaches the target value of 170 individuals.

```
# Assume the total number of individuals receiving a placebo and a vaccine is fixed
Nplacebo = 21830
Nvaccine = 21380
Ntarget  = 170
NVobs    = 8

Pinfect  = 0.04

# Start with no individuals infected
NinfectP = 0
NinfectV = 0

# Repeat the sampling until the target is reached
repeat{

  # Sample 1 individual at the time from the group that has not fallen ill yet
  NnotillP = Nplacebo - NinfectP
  NnotillV = Nvaccine - NinfectV
  Nnotill  = NnotillP + NnotillV
  indx = sample((1:Nnotill), 1)

  # Test whether an individual received a placebo
  if (indx <= NnotillP) {
    NinfectP = NinfectP + 1
  } else {
```

```
    # Test whether individual that received vaccine gets infected
    if (runif(1) < Pinfect) {
      NinfectV = NinfectV + 1
    }
  }

  # The break command jumps out of a loop
  if ((NinfectP + NinfectV) >= Ntarget) break
}
```

```
# Print the current result
c(NinfectP, NinfectV)
```

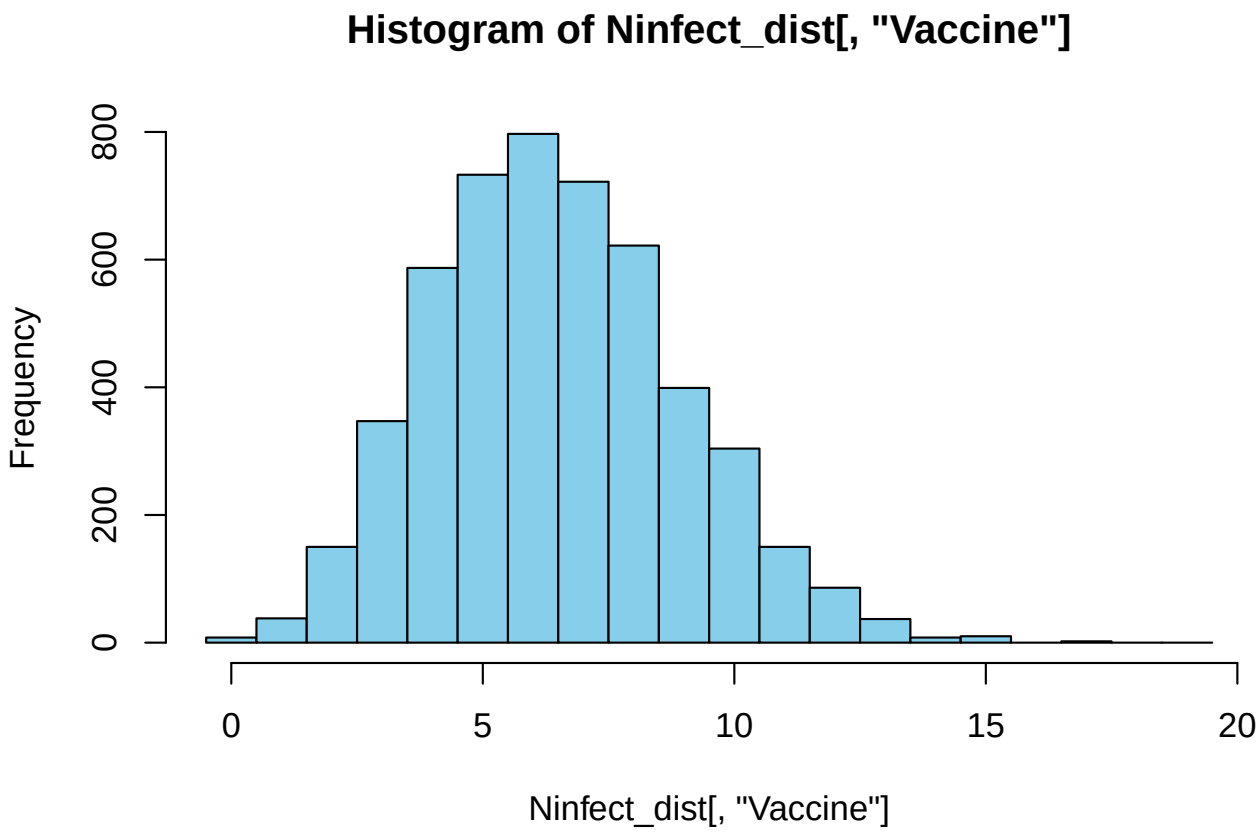
```
[1] 163   7
```

The above code section represents, however, a single realization of the infection process, resulting in 163 individuals getting infected that received a placebo and 7 individuals that received a vaccine. As in the previous subsection we can construct a distribution of the possible realizations by repeating the above procedure many times as shown in the code section below (for $N = 5000$ times). To store the results a matrix is constructed with two columns of empty values, in which the first column will hold the number of infected individuals that received a placebo and the second column the number of individuals that received a vaccine.

```
# Assume the total number of individuals receiving a placebo and a vaccine is fixed
Nplacebo = 21830
Nvaccine = 21380
Ntarget  = 170
NVobs    = 8

Pinfect  = 0.04

# N determines number of resamplings
N = 5000
# Create empty vector for results
Ninfect_dist = matrix(NA, nrow = N, ncol = 2)

for(i in (1:N)) {

  # Start with no individuals infected
  NinfectP = 0
  NinfectV = 0

  # Repeat the sampling until the target is reached
  repeat{

    # Sample 1 individual at the time from the group that has not fallen ill yet
    NnotillP = Nplacebo - NinfectP
    NnotillV = Nvaccine - NinfectV
    Nnotill  = NnotillP + NnotillV
    indx = sample((1:Nnotill), 1)

    # Test whether an individual received a placebo
    if (indx <= NnotillP) {
      NinfectP = NinfectP + 1
    } else {
      # Test whether individual that received vaccine gets infected
      if (runif(1) < Pinfect) {
        NinfectV = NinfectV + 1
      }
    }

    # The break command jumps out of a loop
    if ((NinfectP + NinfectV) >= Ntarget) break
  }

  # Store the current result
  Ninfect_dist[i,] = c(NinfectP, NinfectV)
}
colnames(Ninfect_dist) <- c("Placebo", "Vaccine")
```

To inspect the result we plot a histogram of the number of vaccinated individuals among the total group of 170 individuals that got infected observed in all these 5000 different realizations:

```
hist(Ninfect_dist[,"Vaccine"], breaks = seq(0, 20, length.out = 21) - 0.5, col="skyblue")
```



The histogram shows that for $p = 0.04$ most often only 6 vaccinated individuals would be among the total group of 170 infected individuals. To obtain an estimate for the relative probability of vaccinated individuals to get infected we can now repeat the procedure above for a range of different infection probability values $p = 0, 0.001, 0.002, \ldots, 0.099, 0.1$ and check for each value of p which fraction of the realizations resulted in 8 vaccinated individuals to become infected, as was observed in the real trial. By defining the variable `NVobs` equal to 8, the fraction of realizations that resulted in 8 vaccinated individuals to become infected can be calculated with the statement:

```
sum(Ninfect_dist[, "Vaccine"] == NVobs) / N
```

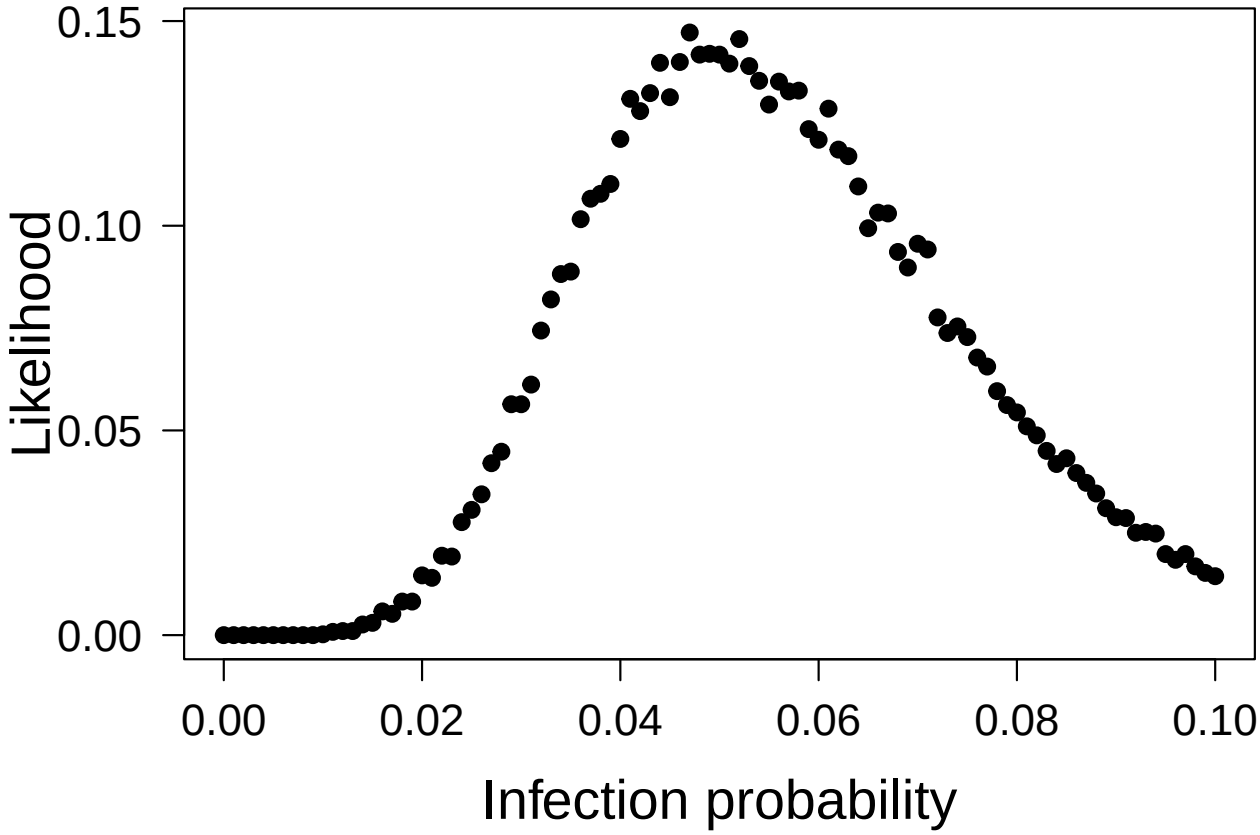
```
[1] 0.1244
```

The resulting value for this fraction is stored in the second column of a matrix `Pinfect`, the first column of which contains the value of the corresponding infection probability p. The following code carries out the computations for the range of infection probabilities $p = 0, 0.001, 0.002, \ldots, 0.099, 0.1$ (*this code section takes a very long time to complete, because it contains a* `repeat` *loop within a* `for` *loop that is in turn contained within another* `for` *loop. Be aware of that when trying to run the code yourself*).

```r
# Assume the total number of individuals receiving a placebo and a vaccine is fixed
Nplacebo = 21830
Nvaccine = 21380
Ntarget  = 170
NVobs    = 8

Plow = 0.0
Phigh = 0.1
Pnr = 101
Pinfect = matrix(NA, nrow = 101, ncol = 2)
Pinfect[,1] = seq(Plow, Phigh, length.out = Pnr)

# N determines number of resamplings
N = 5000
# Create empty vector for results
Ninfect_dist = matrix(NA, nrow = N, ncol = 2)

for (p in (1:Pnr)) {

  for(i in 1:N){

    # Start with no individuals infected
    NinfectP = 0
    NinfectV = 0

    # Repeat the sampling until the target is reached
    repeat{

      # Sample 1 individual at the time from the group that has not fallen ill yet
      NnotillP = Nplacebo - NinfectP
      NnotillV = Nvaccine - NinfectV
      Nnotill  = NnotillP + NnotillV
      indx = sample((1:Nnotill), 1)

      # Test whether an individual received a placebo
      if (indx <= NnotillP) {
        NinfectP = NinfectP + 1
      } else {
        # Test whether individual that received vaccine gets infected
        if (runif(1) < Pinfect[p , 1]) {
          NinfectV = NinfectV + 1
        }
      }

      # The break command jumps out of a loop
      if ((NinfectP + NinfectV) >= Ntarget) break
    }

    # Store the current result
    Ninfect_dist[i,] = c(NinfectP, NinfectV)
  }
  colnames(Ninfect_dist) <- c("Placebo", "Vaccine")

  # Store the current result
  Pinfect[p ,2] = sum(Ninfect_dist[, "Vaccine"] == NVobs) / N
}
```

The last step in the estimation process is to plot for each value of the infection probability p the fraction of realizations that resulted in exactly 8 vaccinated individuals to get infected:

```
par(mar = c(6,6,1,1))
plot(Pinfect, pch = 19, xlab ="Infection probability", ylab = "Likelihood", cex.lab = 1.6, yaxt = "n",
     cex.axis = 1.25)
axis(2, las = 2, cex.axis = 1.25)
```



This plot show for each value of the relative infection probability the likelihood of the outcome of 8 vaccinated individuals and 162 individuals from the placebo group to become infected, as was observed in the real trial. The maximum of the likelihood curve occurs at $p = 0.047$:

```
Pinfect[which.max(Pinfect[,2]), 1]
```

```
[1] 0.047
```

However, if we inspect the entire plot we also observe that the likelihood value varies quite a bit from one value of the infection probability p to the other, such that there is no clear single value with the highest likelihood. The relative infection probability of vaccinated individuals is therefore around 0.05 compared to unvaccinated individuals, but the simulation results are not sufficiently accurate to provide a more precise estimate of this probability. We can however estimate a confidence interval by checking for which values of the probability the likelihood of the outcome of 8 vaccinated individuals and 162 individuals from the placebo group to become infected is larger than or equal to 0.025.

```
c(min(Pinfect[(Pinfect[,2] >= 0.025), 1]), max(Pinfect[(Pinfect[,2] >= 0.025), 1]))
```

```
[1] 0.024 0.093
```

5.4.3 What you should know

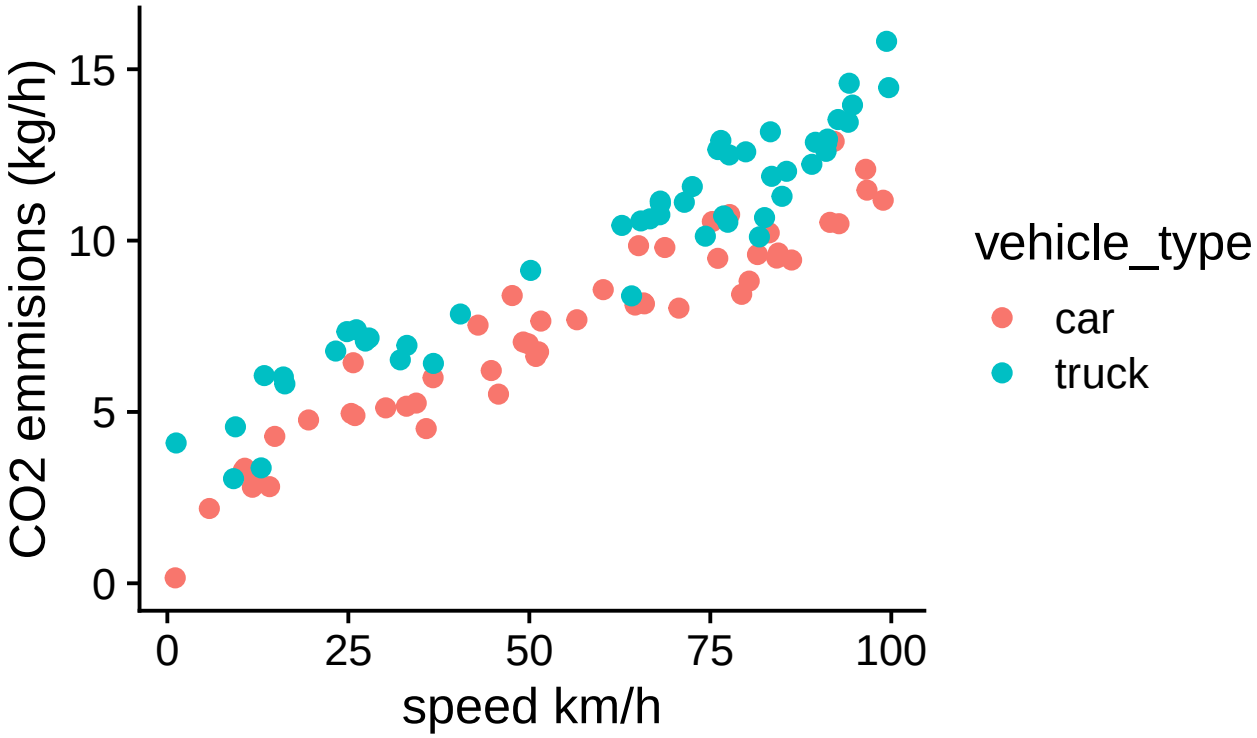
1. Monte Carlo simulations can be used to simulate any process, of which the outcome is influenced by some source of stochasticity
2. Repeating the simulation of the process many times, generates a distribution of possible outcomes
3. This distribution can in turn be used to analyze, for example, an experimental design or to estimate parameters, as illustrated by the two examples in this section
4. The application of Monte Carlo simulations is however not limited to evaluating experimental designs or estiamting parameters, the applicabilition of Monte Carlo simulation method is only limited by our own imagination

6 More complex models

6.1 More than one independent variable

In many cases, we are interested in modeling the dependent variable as a function of more than one independent variable. With general linear models this is easy to incorporate, we just add an additional term in the linear model for each new independent variable. For each new independent variable we also have an additional parameter associated with that variable. All the concepts we applied to single variable models about parameter estimation, standard errors, confidence intervals, and hypothesis tests largely apply to more complex models.

For example, in chapter 1, we introduced a data set for vehicle CO2 emissions that depended on the vehicle type and the speed of travel:



The simplest linear model we could use to describe this data set would be:

$CO2 = b0 + b1 * vehicle\_type + b2 * speed$

where "vehicle_type" is a dummy variable that is 0 for cars and 1 for trucks. We can fit this model to data using the lm function:

```
mod <- lm(CO2 ~ vehicle_type + speed,df)
summary(mod)
```

```

Call:
lm(formula = CO2 ~ vehicle_type + speed, data = df)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.95681 -0.51858  0.05663  0.53072  1.99293 

Coefficients:
```

```
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       1.954904   0.204214   9.573 1.12e-15 ***
vehicle_typetruck 1.936727   0.176809  10.954  < 2e-16 ***
speed             0.099942   0.003035  32.935  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8757 on 97 degrees of freedom
Multiple R-squared:  0.9319,    Adjusted R-squared:  0.9305
F-statistic: 664.1 on 2 and 97 DF,  p-value: < 2.2e-16
```

We can see in the coefficients table above that there are 3 rows of values, with each row corresponding to one of the parameters in our model. For each parameter, we get a parameter estimate, a standard error, a t-value (which is just the estimate divided by the standard error), and a p-value.

Problem 5.1 What are the predicted CO2 emissions of a truck traveling 0 km/h?

At 0 km/h we can ignore the effect of speed. We see that the intercept parameter, b_0 is 1.955 which is the predicted CO2 emissions of cars traveling 0 km/h. The expected difference in CO2 emissions of trucks compared, b_1, to cars is shown in the second row: 1.937. And thus the expected emissions of a truck traveling 0 km/h is $b_0 + b_1 = 3.892$.

Problem 5.2 What is the predicted CO2 emissions of a truck traveling 50 km/h? To answer this question we can just plug in the estimated values of parameters, along with the values of the independent variables we want to make our prediction for into the general linear model equation:

```
b0 = summary(mod)$coefficients[1,1]
b1 = summary(mod)$coefficients[2,1]
b2 = summary(mod)$coefficients[3,1]

vehicle_type = 1
speed = 50

pred = b0 + b1 * vehicle_type + b2 * speed
pred
```

```
[1] 8.888728
```

Problem 5.3 Describe the the estimated effect of speed on CO2 emissions and its uncertainty.

Vehicles traveling 1 km/h faster have an estimated 0.1 kg/h higher emissions. The standard error of this estimate is 0.003, thus we are approximately 95% confident the true population parameter lies between 0.0939 and 0.1059.

Also I didn't ask for it, but if you were going to do a null hypothesis test, for example, that the population parameter associated with speed = 0, we would see that the estimated parameter value is 33.5 standard errors away from zero (t-value). This alone tells us it would be extremely unlikely to observed this large of a value due to random chance alone (remember 95% of observations are expected to fall within 2 standard errors, and 99.7% within 3 SE). This intuition is confirmed if we look at the very small p-value (less than 2e-16, or 0.00000000000000022), which means we would essentially never see such a strong relationship due to chance alone. It is still important to keep in mind though, especially for observational datasets that just because you have a very low p-value does not imply causation. In this case, a mechanistic link between speed and CO2 emissions is clear, but this won't always be the case. For example consider the case where fast drivers also listen to loud music. If I fit a model with music volume instead of speed as the independent variable, we would also reject the null hypothesis that the relationship between music volume and CO2 emissions is unlikely to emerge by chance alone. However this would not mean CO2 emissions of a car will directly

increase in response to turning the volume up. It is important to remember when analyzing observational data that small p-values doesn't imply causation, because the observed relationship could always be due to an unmeasured causal variable that is correlated with the independent variable.

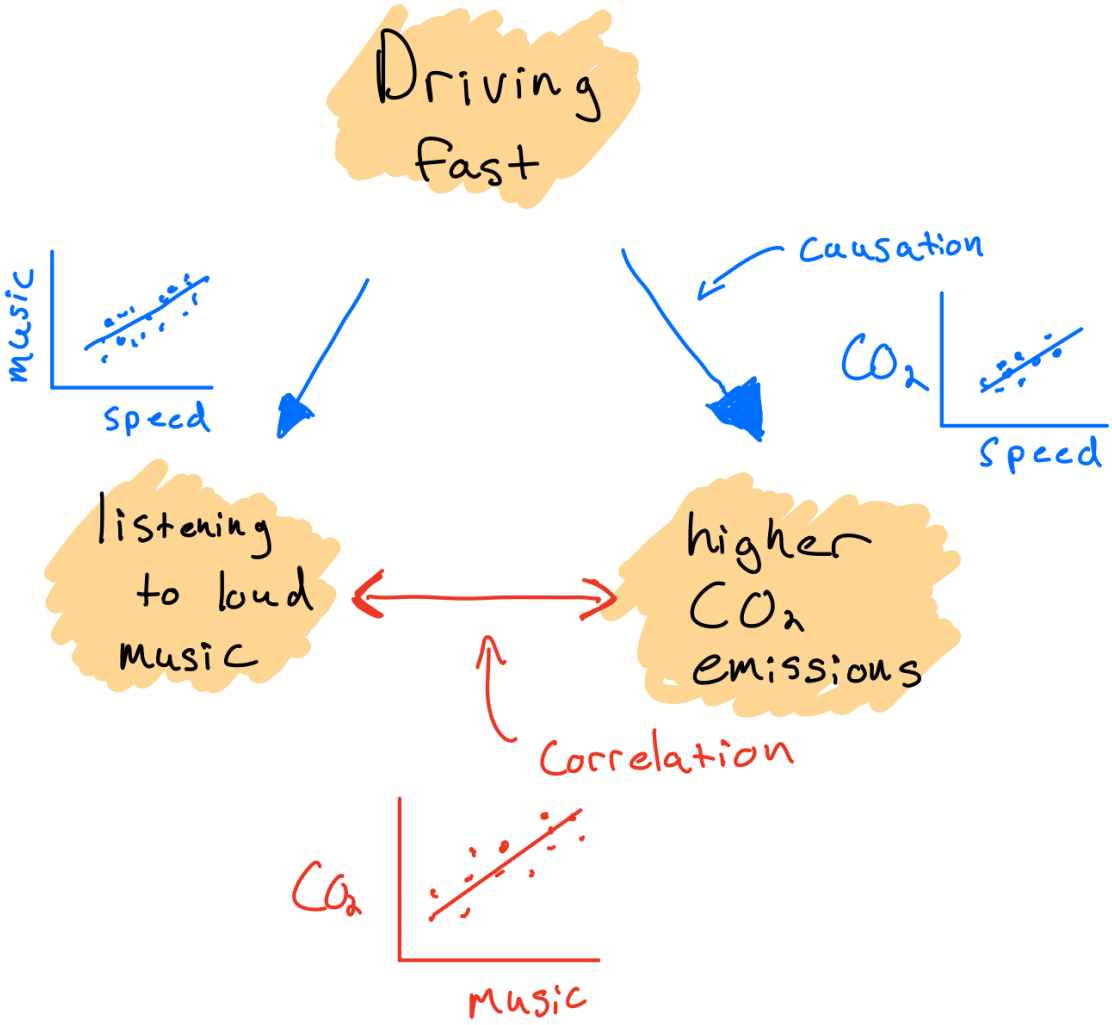
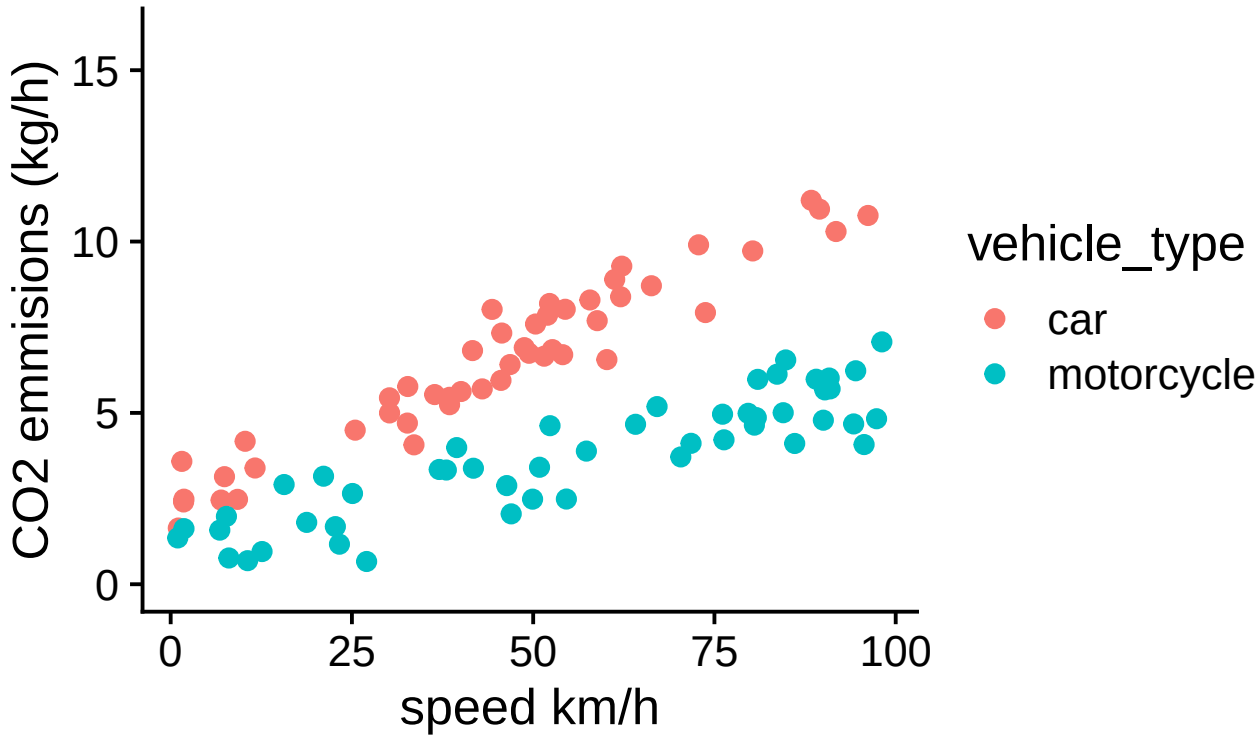


Figure 6.1: **Regression parameters do not necessarily represent causal effects**

6.2 Interactions

A key assumption of the CO2 emissions model was that the effect of speed on CO2 emissions did not depend on the vehicle type. Looking at the data this seems like a reasonable assumption, but it won't always be the case. For example, consider a similar dataset, but now comparing the CO2 emissions of cars and motorcycles:



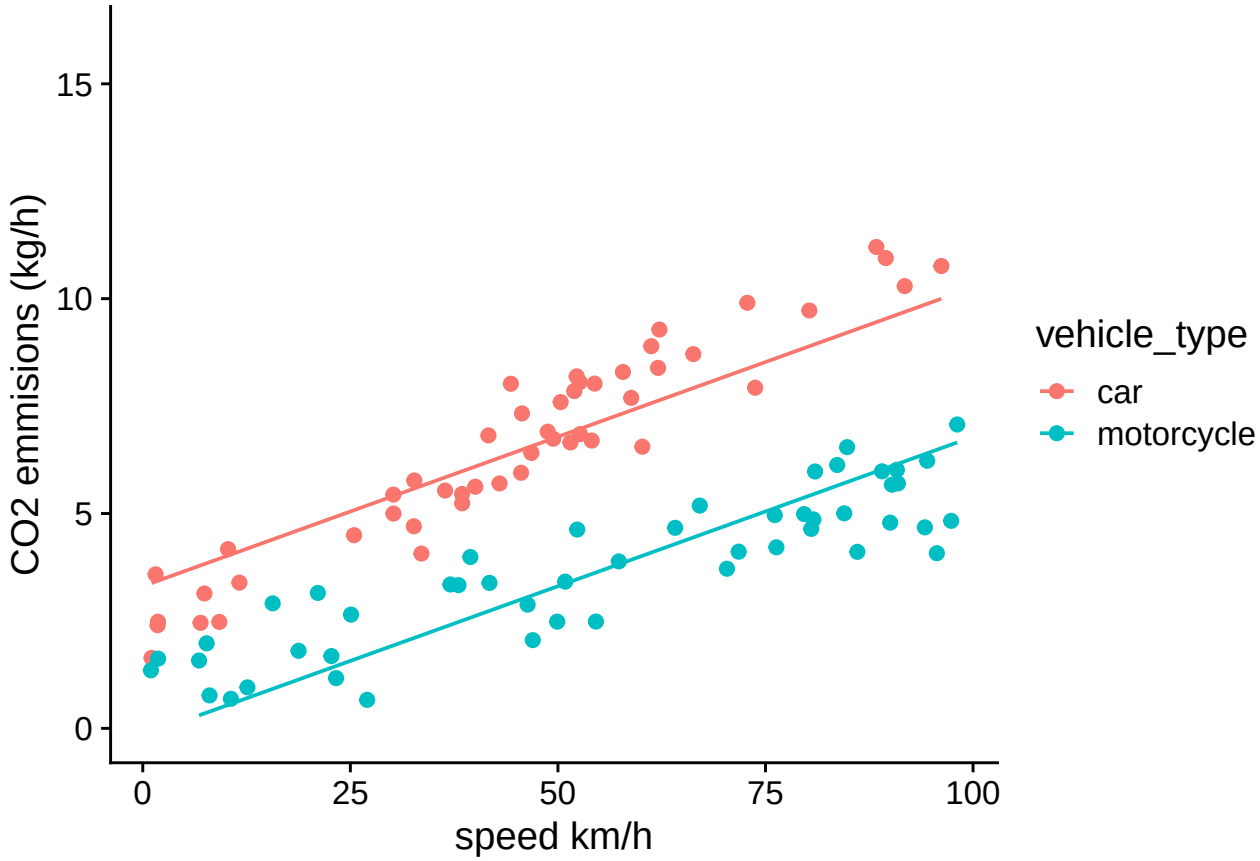
If we repeat the analysis assuming the relationship between speed and emissions is identical for cars and motorcycles:

```
mod<-lm(CO2 ~ vehicle_type + speed,df)
summary(mod)
```

```

Call:
lm(formula = CO2 ~ vehicle_type + speed, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-4.5741 -0.7047  0.0128  0.7954  1.9902

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)           3.308149   0.227935   14.51   <2e-16 ***
vehicle_typemotorcycle -3.477201   0.221368  -15.71   <2e-16 ***
speed                 0.069559   0.003825   18.18   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.087 on 97 degrees of freedom
Multiple R-squared:  0.834, Adjusted R-squared:  0.8305
F-statistic: 243.6 on 2 and 97 DF,  p-value: < 2.2e-16
```

We see again a strong association between speed and CO2 emissions, and that motorcycles have lower average CO2 emissions than cars. However, it is also always a good idea to visually inspect the model by plotting out the predictions of the model along with the data. We can do that by creating a new variable in the data frame that contains the predicted value of CO2 emissions for each observation, and then plotting out the model predictions along with the data.

```
b0 = summary(mod)$coefficients[1,1]
b1 = summary(mod)$coefficients[2,1]
b2 = summary(mod)$coefficients[3,1]

vehicle_type_dummy <- rep(0,nrow(df))
vehicle_type_dummy[df$vehicle_type == "motorcycle"] = 1
df$prediction<- b0 + b1* vehicle_type_dummy + b2* df$speed


ggplot(df,aes(x=speed,y=CO2,group=vehicle_type))+
  geom_point(aes(color=vehicle_type))+
  geom_line(aes(x=speed,y=prediction,color=vehicle_type))+
  theme_cowplot()+ylim(0,16)+xlab("speed km/h")+ylab("CO2 emmisions (kg/h)")
```



While the model fit looks pretty good, if you inspect carefully, you see that the residuals tends to show systematic bias. For example, for cars, most observed CO2 emissions are lower than the predicted value at low speeds, and above the predicted value at high speeds. The opposite pattern is true for motorcycles. This suggest that the relationship between CO2 emissions and speed might differ for cars vs. motorcycles. **In statistical models, when the effect of one independent variable depends on the value of another independent variable, we call it an interaction, or an interactive effect.**

In general we can add an interaction in our statistical model by creating a new variable that is the product of two independent variables:

$$\hat{y} = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_1 x_2$$

where b_3 captures the effect of the interaction.

Problem 5.3 Going back to the emissions example, lets assume x_1 is the dummy variables that is 0 for cars and 1 for motorcycles, and x_2 is speed. Based on this you should be able to work out the interpretation of each of parameter in the model. Hint: plug in the value of the dummy variable into the equation and remove terms that equal zero.

We can get the formula for cars by setting x_1 to 0, which eliminates the 2nd and 4th term in the linear model, such that:

$$\hat{y}(cars) = b_0 + b_2 x_2$$

From this we can see that b_0 is the expected CO2 emissions for cars at 0 km/h, and b_2 is the rate of increase in CO2 emissions with speed for cars.

We now have interpretations for two of the parameters in our linear model. We can identify the interpretation of the remaining two, but setting x_1 to 1, which results in:

$$\hat{y}(motorcycles) = b_0 + b_1 + (b_2 + b_3)x_2$$

We can see that when speed is 0, the predicted CO2 emissions of motorcycles is $b_0 + b_1$. Remembering the the predicted CO2 emissions of cars at 0 km/h is b_0, we can see that b_1 is the expected difference in emissions for motorcycles vs. cars at 0 km/hr. Similarly, we can see that emissions increase by $b_2 + b_3$ for each unit increase in speed for motorcycles. Because emissions increase by b_2 per unit speed for cars, we can interpret the parameter b_3 as the expected difference in slope of CO2 emissions with speed in motorcycles vs. cars.

We can fit a model with an interaction using the lm() function as follows:

```
mod<-lm(CO2 ~  vehicle_type + speed + vehicle_type:speed,df)
summary(mod)
```

```

Call:
lm(formula = CO2 ~ vehicle_type + speed + vehicle_type:speed,
    data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-3.3638 -0.5290  0.0588  0.5942  1.6177

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                 1.997425   0.244763   8.161 1.30e-12 ***
vehicle_typemotorcycle     -1.145352   0.344293  -3.327  0.00125 **
speed                       0.099353   0.004841  20.523  < 2e-16 ***
vehicle_typemotorcycle:speed -0.048403   0.006170  -7.844 6.05e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.853 on 96 degrees of freedom
Multiple R-squared:  0.8988,    Adjusted R-squared:  0.8957
F-statistic: 284.3 on 3 and 96 DF,  p-value: < 2.2e-16
```

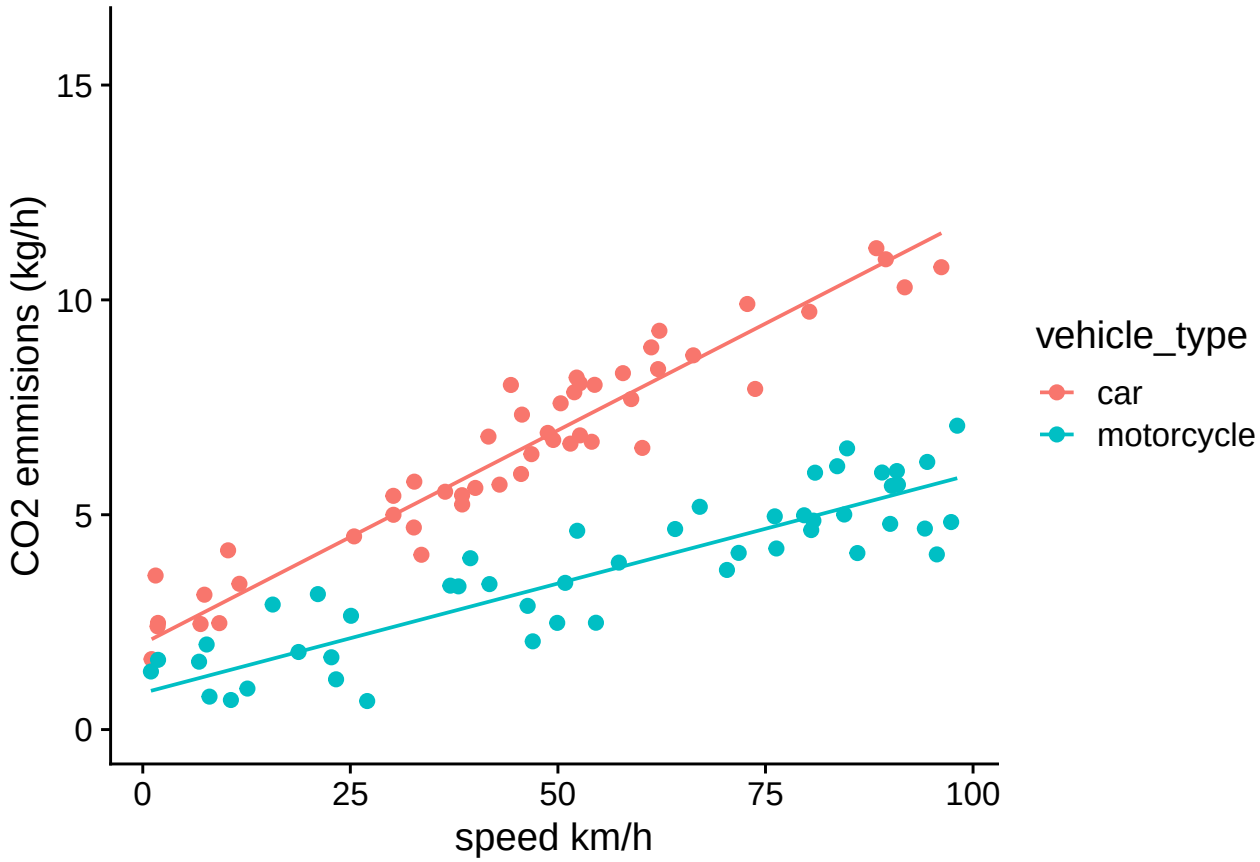
Problem 5.4 What is the estimated rate of increase in CO2 emissions with speed for motorcycles?

The last row in the coefficient table represents the estimated difference in the slope of CO2 emissions with speed for motorcycles vs. cars. We see that this value is negative which suggests that the rate of increase of CO2 emissions for motorcycles is lower than for cars, and based on the values of b_2 and b_3, would be approximately: 0.051.

We also can see that the standard error for the interaction parameter is small relative to the estimated value. This suggest that a model with different slopes for CO2 as a function speed for cars and motorcycles is more consistent with the data than a model that assumes the slopes are identical: $b_3 = 0$.

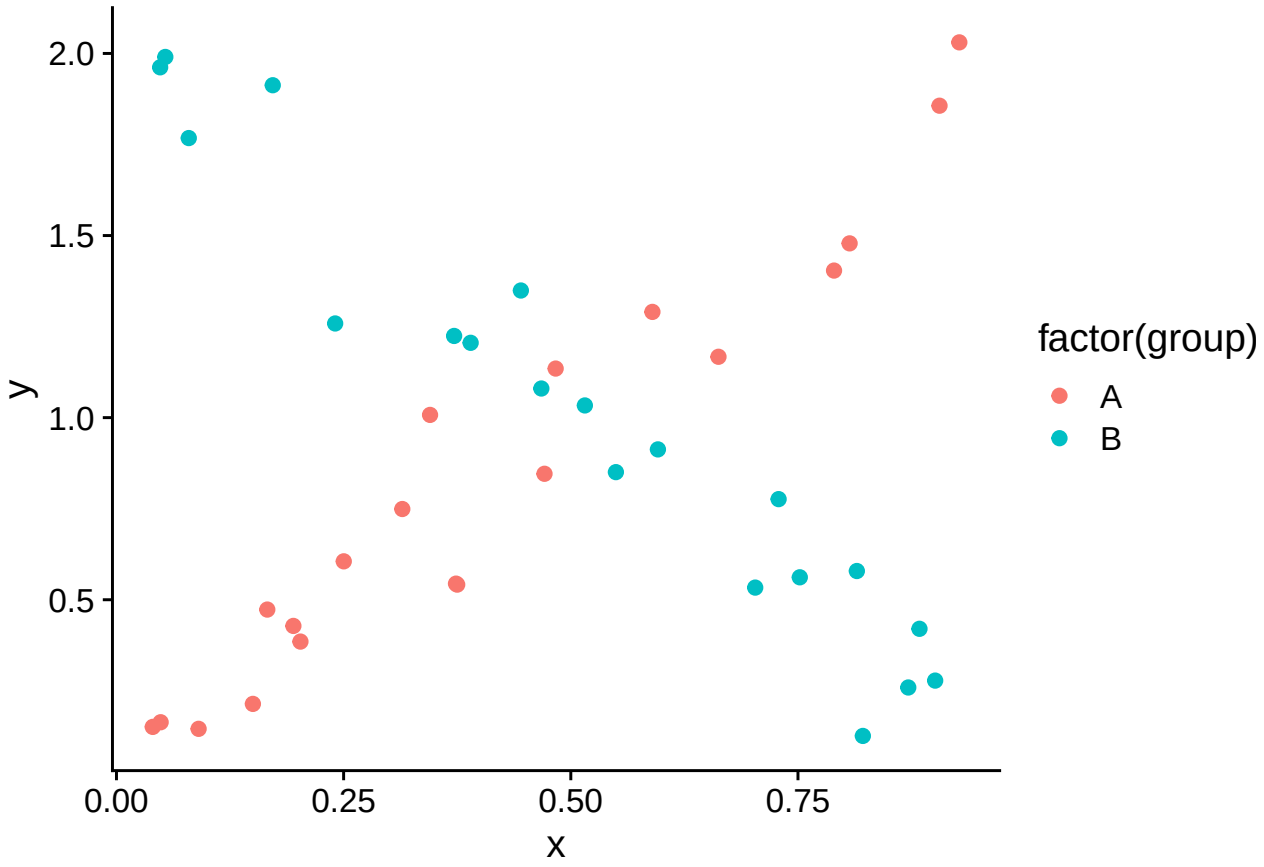
Finally, we can evaluate the fit of the model by plotting out its predictions along with the data:

```
b0 = summary(mod)$coefficients[1,1]
b1 = summary(mod)$coefficients[2,1]
b2 = summary(mod)$coefficients[3,1]
b3 = summary(mod)$coefficients[4,1]
vehicle_type_dummy <- rep(0,nrow(df))
vehicle_type_dummy[df$vehicle_type == "motorcycle"] = 1
df$prediction<- b0 + b1* vehicle_type_dummy + b2* df$speed +b3 * df$speed * vehicle_type_dummy


ggplot(df,aes(x=speed,y=CO2,group=vehicle_type))+
  geom_point(aes(color=vehicle_type))+
  geom_line(aes(x=speed,y=prediction,color=vehicle_type))+
  theme_cowplot()+ylim(0,16)+xlab("speed km/h")+ylab("CO2 emmisions (kg/h)")
```



Visually inspecting this model reveals that the residuals are no longer systemically biased as a function of speed. This further suggests that a model with an interaction term makes sense in this case.

One important thing to keep in mind is that you need to be careful interpreting the main effect of individual variables when there is an interaction. This is because the "effect" of one independent variable depends on the value of the other independent variable. For example if we had the following data:



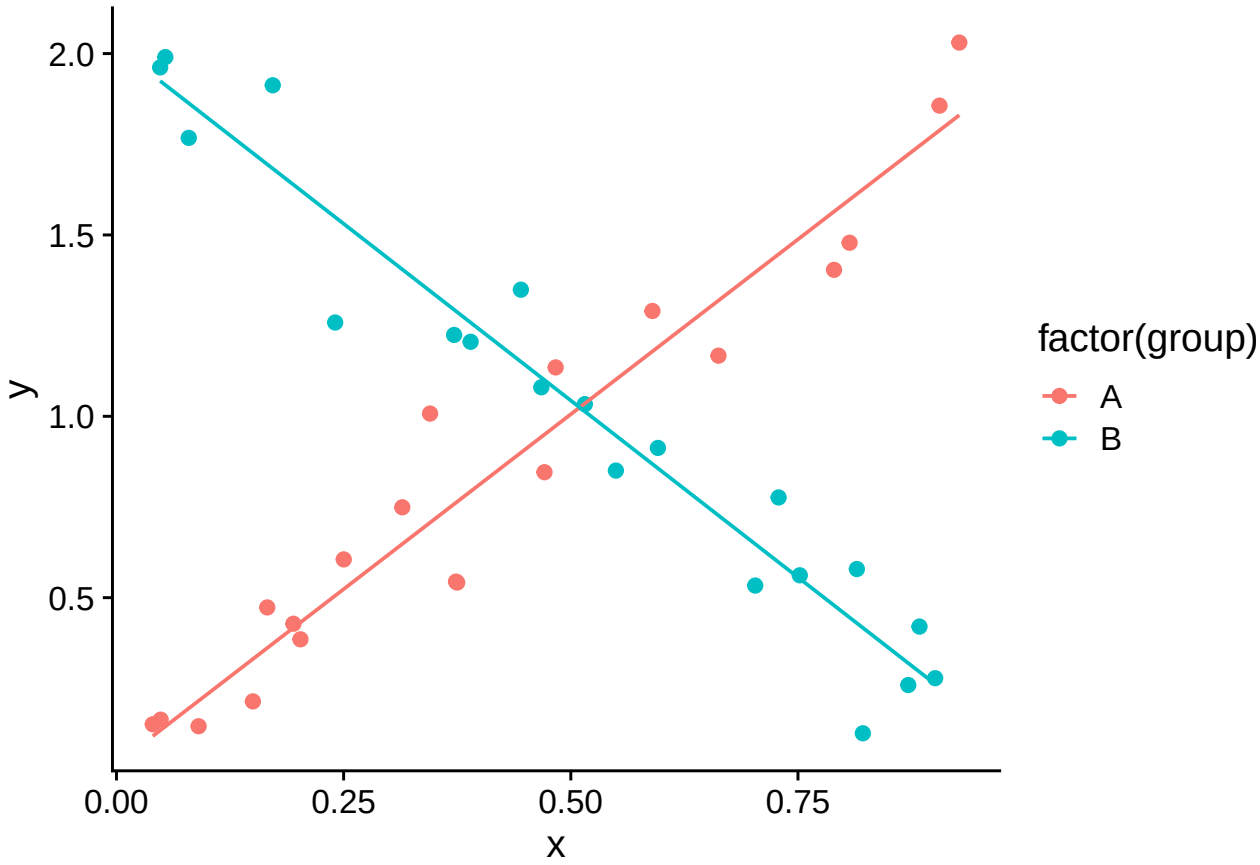
And we fit a linear model:

```
summary(lm(y~group*x,df))
```

```

Call:
lm(formula = y ~ group * x, data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-0.29079 -0.09822  0.01382  0.10375  0.30066

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.04117    0.05863   0.702    0.487
groupB       1.97709    0.09059  21.825   <2e-16 ***
x            1.92891    0.11846  16.283   <2e-16 ***
groupB:x    -3.87844    0.16625 -23.329   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1474 on 36 degrees of freedom
Multiple R-squared:  0.9394,    Adjusted R-squared:  0.9344
F-statistic: 186.1 on 3 and 36 DF,  p-value: < 2.2e-16
```

We see that the main effect of group B is positive (~2), which if there was no interaction would suggest that the value of y for group B is ~2 units higher that group A for a given level of x. However because there is a strong interaction, we can see in the plot that whether group B has a higher value of y than A depends on x. The interpretation of the main parameter is the estimated difference between B vs. A when x = 0. In general, when you have an interaction it is useful to plot out the predictions of the model to interpret model parameters:

```
mod<-(lm(y~group*x,df))
df$pred  <-predict(mod)
ggplot(df,aes(x=x,y=y,group=group))+
  geom_point(aes(col=factor(group)))+
  geom_line(aes(x=x,y=pred,col=factor(group)))+theme_cowplot()
```



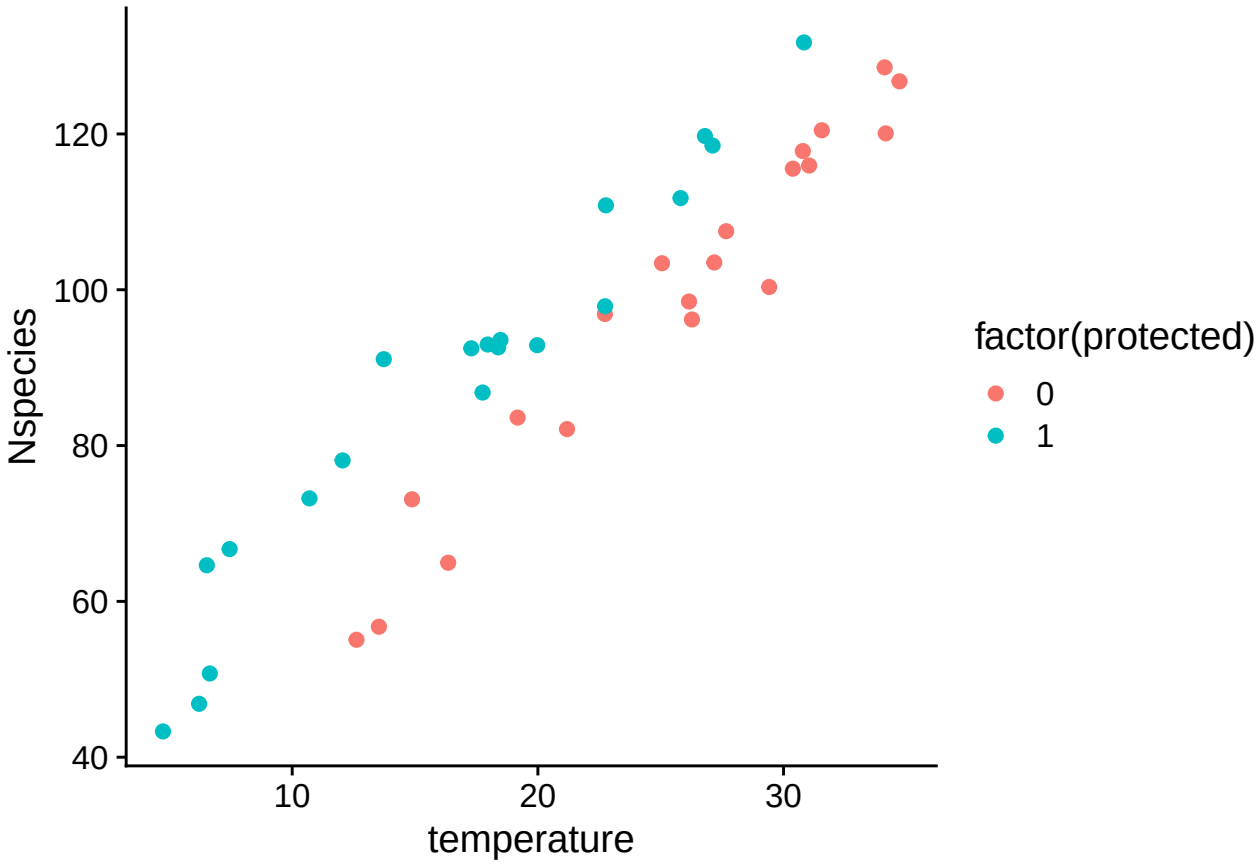
In this case we can see that the predicted value of y for each group depends on x. When x is 0, group B has a higher value of y, but the predicted value of y decreases with x for group B, while it increases with x for group A. Thus at higher values of x, group A has a higher predicted value of y than group B.

6.3 Controlling for a variable

Sometimes our hypothesis is related to the importance of a specific independent variable, but we know that other variables have strong impacts on the dependent variable. For example, we might be interested in whether there are more species of insects in protected (no industry and logging) forests vs. unprotected forests, but we also know that there tends to be more insects in warmer areas. Our primary interest is in the difference between protected and unprotected forests, but we think another variable (temperature) will cause a lot of variation in the dependent variable. If we don't account for this variable, it can 1) cause there to be a high residual error and thus high standard error of parameter estimates, and 2) if the level of the control variable is not balanced among the variable of interest (e.g. more protected forests in warmer locations) then it can lead to biased parameter estimates (e.g. because there

is some correlation between protection status and temperature. If we don't account for temperature, the influence of temperature will show up in the estimated affects of protection status).

Here we I illustrate these points with the following data:



If we just fit a model with only the variable of interest (protection status):

```
mod<-lm(Nspecies~protected,df)
summary(mod)
```

```

Call:
lm(formula = Nspecies ~ protected, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-44.512 -15.129   4.721  18.056  43.924

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   98.352      5.282   18.62   <2e-16 ***
protected    -10.530      7.470   -1.41    0.167
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 23.62 on 38 degrees of freedom
Multiple R-squared:  0.04969,   Adjusted R-squared:  0.02468
F-statistic: 1.987 on 1 and 38 DF,  p-value: 0.1668
```

We see first that the estimated effect of protecting a forest is negative (reduces the number of species), and second that there is a relatively large standard error for the parameter estimate. If we remember from Chapter 3 that the standard error of parameter estimates increases in proportion to the standard deviation of the residuals, it makes sense that our estimates of parameter values are imprecise.

However if we re-run the analysis with temperature in the model to account for variation in the number of species in an area due to temperature:

```
mod<-lm(Nspecies~protected+temperature,df)
summary(mod)
```

```

Call:
lm(formula = Nspecies ~ protected + temperature, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-10.152  -4.352   0.367   3.675  12.367

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  20.4147     3.2089   6.362 2.04e-07 ***
protected    16.2712     1.9928   8.165 8.45e-10 ***
temperature   3.0625     0.1168  26.223  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.41 on 37 degrees of freedom
Multiple R-squared:  0.9515,    Adjusted R-squared:  0.9489
F-statistic: 362.8 on 2 and 37 DF,  p-value: < 2.2e-16
```

Problem 5.5 We now see the estimated number of species is higher in protected vs. unprotected forests as opposed to being lower when temperature was not included in the model. Why caused this difference?

In this case we can see from the plot that there is a bit of correlation between temperature and protected status, in that temperatures in protected forests tended to be cooler on average than unprotected forests. Because forests with cooler temperatures also tend to have less species, when we fit a model without temperature, we actually estimated that protected forests have fewer species.

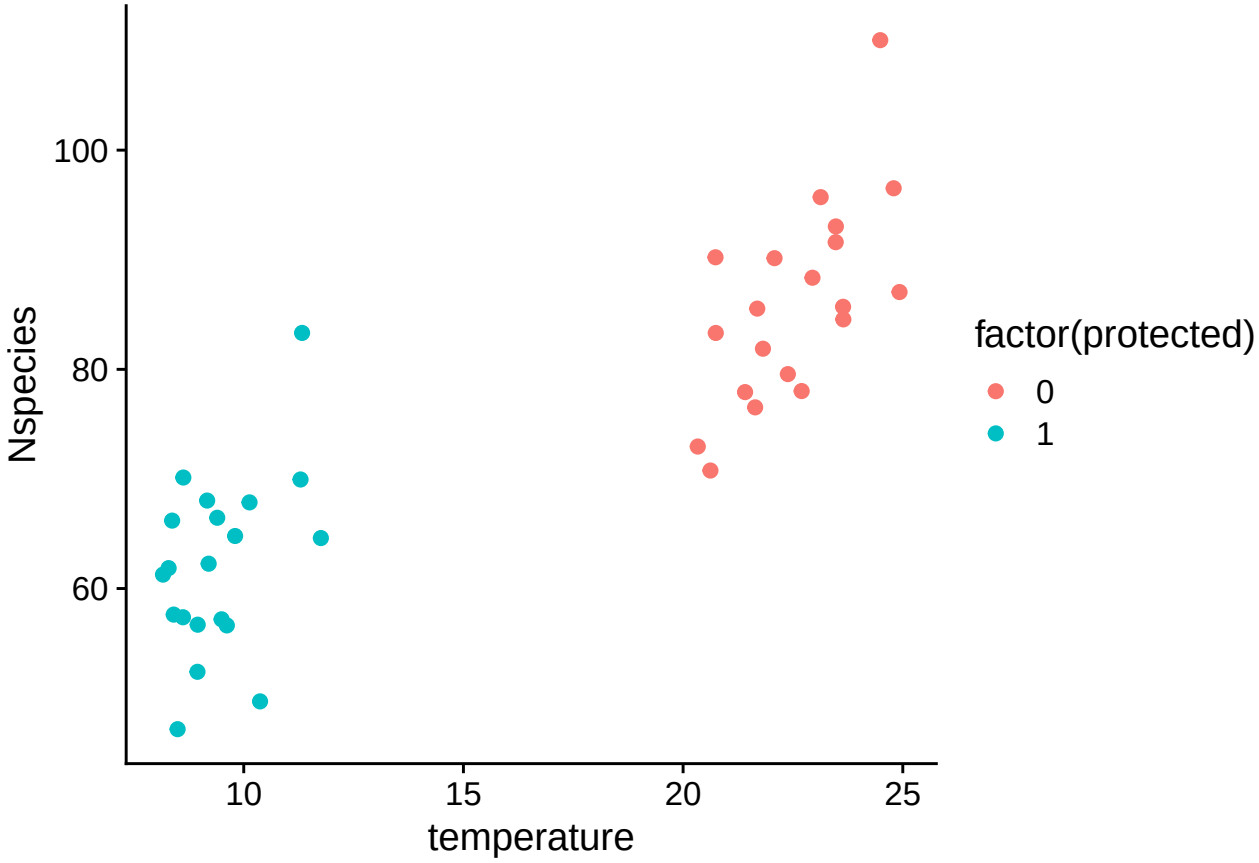
Problem 5.6 We can also see that the standard error associated with the effect of protection status is now much lower now that we have included temperature in the model. Why did including temperature as a independent variable reduce the the standard error for the parameter associated with protection status?

Because temperature appears to explain a large amount of variation in the number of species present in forests, including it in the statistical model reduced the standard residual error. Because standard errors of parameter estimates increase with the standard residual error, reducing the errors allows model parameters to be estimated more precisely.

In summary, we often want to control for variables to 1) reduce the residual errors and consequently increase the precision in our parameter estimates and 2) avoid bias in our estimates of parameters of interest that may occur when there is some correlation between our independent variable of interest and confounding variables.

6.4 Multicollinearity

The correlation between protected status and temperature was mild in the last example, in that there was still a fair amount of overlap in temperatures between protected and unprotected forests. However in other cases, the correlation can be more extreme. For example, if the data looked like this:



we now see that there is much stronger correlation between protection status and temperature. All the forests that are protected are from cooler climates than unprotected forests. The problem this creates is that it becomes difficult to estimate the parameters in the model because variation in the dependent variable could be attributed to either independent variable. For example, if we fit a model with either temperature or protection status alone, we would find strong associations for each independent variable:

Protection status model:

```
mod_protect<-lm(Nspecies~protected,df)
summary(mod_protect)
```

```

Call:
lm(formula = Nspecies ~ protected, data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-15.2060  -5.3933  -0.2404   4.6935  24.0627

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   85.972      1.948  44.125  < 2e-16 ***
```

```
protected    -23.896      2.755  -8.672 1.53e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.713 on 38 degrees of freedom
Multiple R-squared:  0.6643,    Adjusted R-squared:  0.6555
F-statistic: 75.21 on 1 and 38 DF,  p-value: 1.53e-10
```

Temperature model:

```
mod_temp<-lm(Nspecies~temperature,df)
summary(mod_temp)
```

```

Call:
lm(formula = Nspecies ~ temperature, data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-13.7106  -4.7581   0.4887   4.8336  19.9009

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  43.7757     3.1365   13.96  < 2e-16 ***
temperature   1.8932     0.1812   10.45 9.89e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.641 on 38 degrees of freedom
Multiple R-squared:  0.7419,    Adjusted R-squared:  0.7351
F-statistic: 109.2 on 1 and 38 DF,  p-value: 9.892e-13
```

But if we include a model with both variables:

```
mod_temp<-lm(Nspecies~ protected+ temperature,df)
summary(mod_temp)
```

```

Call:
lm(formula = Nspecies ~ protected + temperature, data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-16.1107  -5.7975   0.6444   4.2684  16.3863

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -2.5923    21.2914  -0.122 0.903753
protected    27.6398    12.5660   2.200 0.034163 *
temperature   3.9303     0.9421   4.172 0.000175 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.282 on 37 degrees of freedom
Multiple R-squared:  0.7717,    Adjusted R-squared:  0.7594
F-statistic: 62.54 on 2 and 37 DF,  p-value: 1.354e-12
```

We see that both terms are "statistically significant" but that there are quite large standard errors, especially for the parameter associated with protection status. In this case, even through both variables come out as "statistically significant" the data available probably is not sufficient to test the hypothesis that protected forests have more species than unprotected forests. This is because our linear model makes a lot of assumptions that are hard to asses. For example, our model assumes that the relationship between temperature and the number of species is the same for protected and unprotected areas, but given that the temperatures in protected and unprotected forests do not overlap, it is really difficult to asses this assumption. Our model also assumes the affect of temperature is linear, but because we only have a limited range of temperatures within each type of forest it is difficult to assess this assumption. Thus this is a case where even though the results of the statistical test are significant common sense and critical thinking suggests that we should be very cautious about interpreting the results of this test.

When two or more independent variables are highly correlated, we call this multicollinearity. Multicollinearity is very common in observational data sets, especially in ecological time series. The general problem is that when independent variables are highly correlated, it can be difficult to estimate the effects of individual independent variables. For example, consider that case where you doctor tells you that you need to improve your health, and prescribes you some medication. At the same time, you start eating better, and getting more regular sleep. After a few weeks you health improves. The question is, did you health improve because of the medication? Or eating better? Or getting more regular sleep? Or some combination of the three? In this example, it is difficult to tell because the 3 potential explanatory variables were all correlated. This is exactly the same problem when we have correlated independent variables, multicollinearity makes it hard to assess the relative importance of the independent variables. As a result when we fit models with correlated independent variables, the standard errors of the parameter estimates tend to be large. The more correlated the variables, the larger the standard errors will be for the parameters associated with those variables.

If we are trying to test hypotheses about the relative importance of two independent variables for some dependent variable of interest, and the two independent variables are highly correlated, then if we include both terms in the model, the standard errors of the parameter estimates will likely be large and perhaps overlap zero (the parameter associated with the independent variable could be positive or negative). In this case the data are not really sufficient to test our hypothesis, because there is not enough independent variation in the independent variables to be able to assess whether the dependent variable is more strongly associated with one or the other independent variable. In this case, the best option is to try to find more data (or do an experiment) where the independent variables are not correlated.

Sometimes our goal is prediction rather than inference. In this case we don't care as much about the parameter estimates, but rather how accurately our model predicts the dependent variable. In this case, multicollinearity is not problematic, because it generally does not affect the predictive performance of linear models (e.g. the residual standard error).

6.5 What you should know

1. How to write down equations for linear models with multiple independent variables

2. How to fit linear models with multiple independent variables or control variables, and interpret the output of models

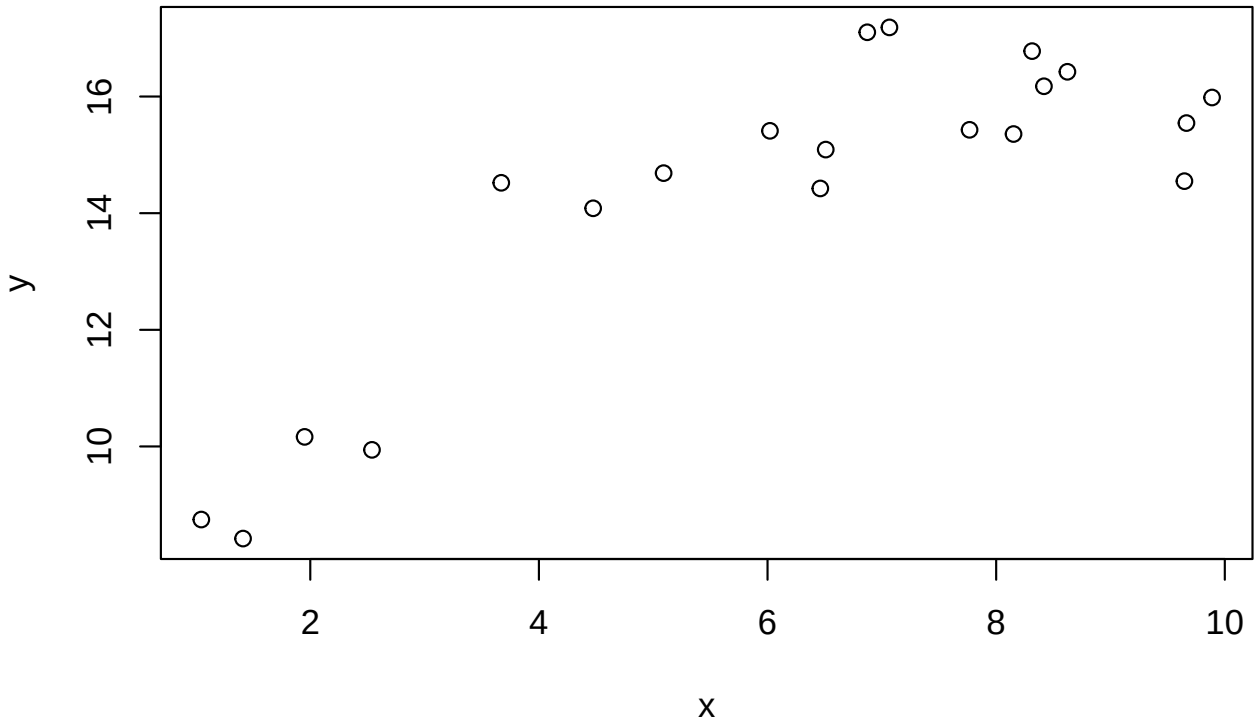
3. Understand and interpret interactive effects in statistical models

5. Understand what multicollinearity is, and how it affects the conclusions we can draw from analyses

7 Comparing models

7.1 Model complexity and overfitting

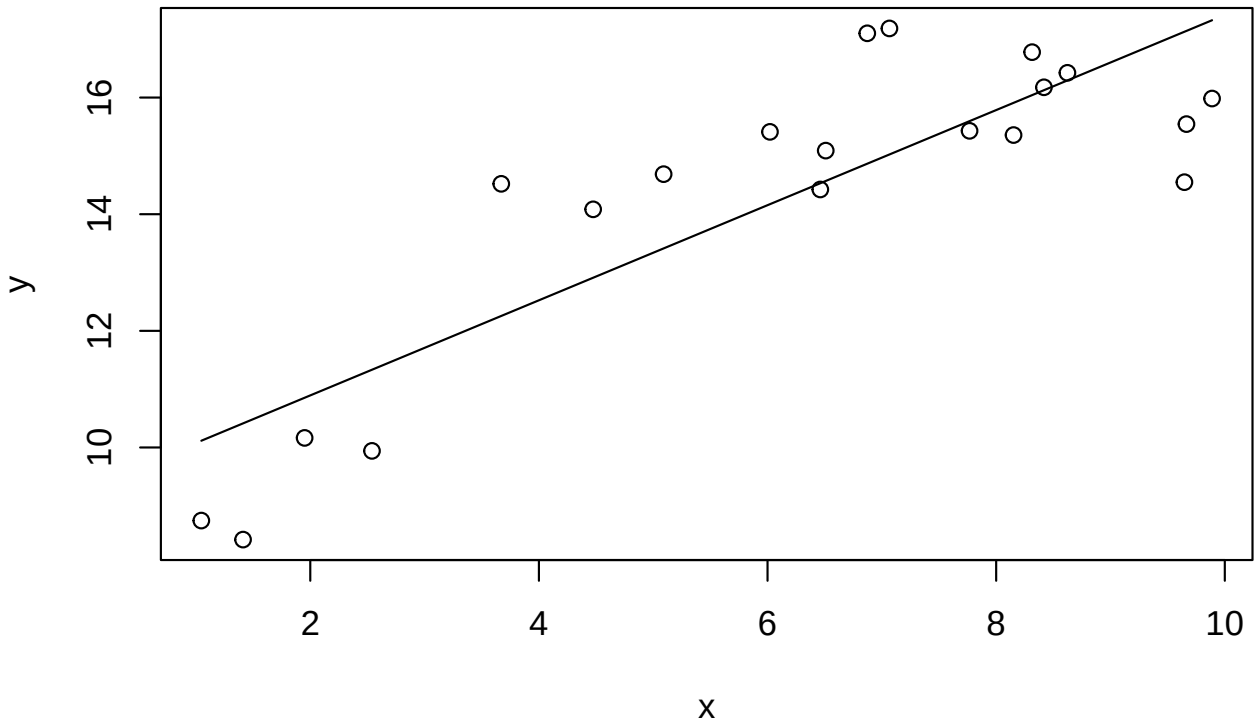
In general, if we add new terms to our model, it will only improve the fit. For example, consider the following data:



We can fit a linear model to this data and try to explain some of the variation in y.

```
Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-2.57936 -1.34904 -0.02879  1.18978  2.26550

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   9.2619     0.8160  11.351 1.23e-09 ***
x             0.8155     0.1206   6.762 2.46e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.487 on 18 degrees of freedom
Multiple R-squared:  0.7175,    Adjusted R-squared:  0.7018
F-statistic: 45.73 on 1 and 18 DF,  p-value: 2.459e-06
```

We see that x explains about 70% of the variation in y. However looking at the residuals we see evidence for a nonlinear relationship between x and y.



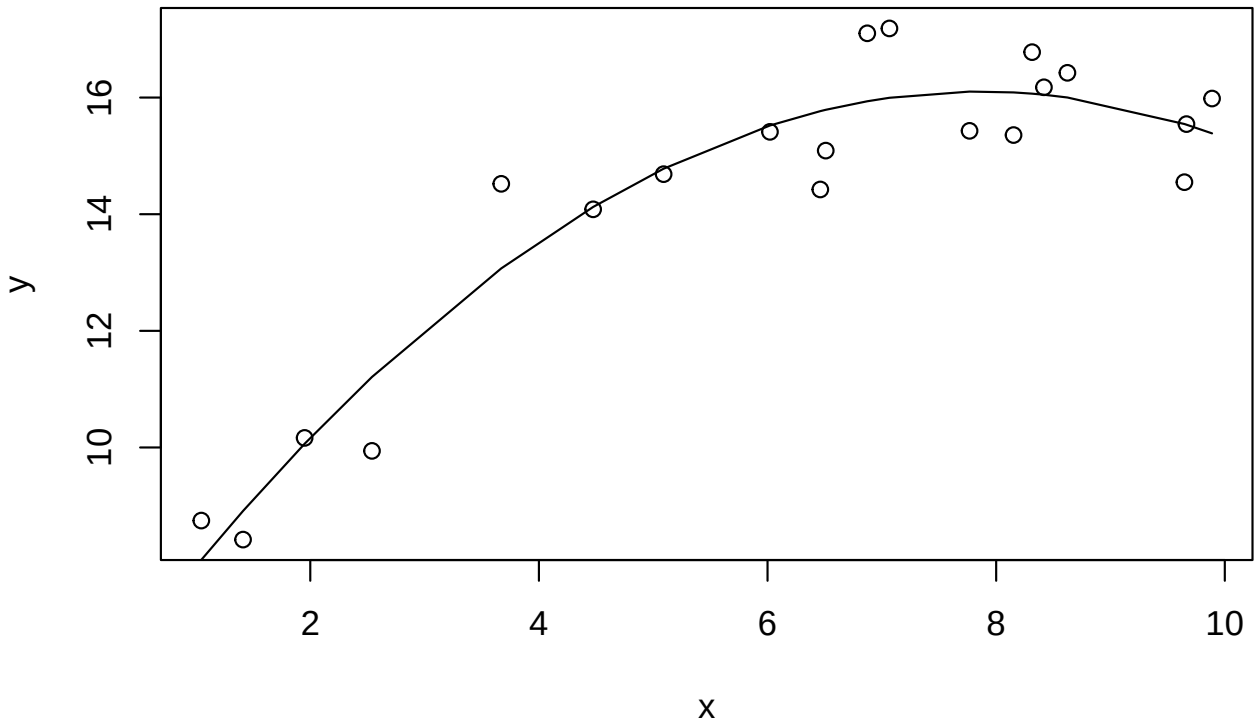
One simple way to account for this would be to add an quadratic term in the model, such that our model is:

$$\hat{y} = b_0 + b_1 x + b_2 x^2$$

We can fit this model in R:

```
Call:
lm(formula = y ~ x + I(x^2))

Residuals:
     Min       1Q   Median       3Q      Max
-1.34381 -0.67959 -0.01544  0.61630  1.44971

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.41370    0.78967   6.856 2.79e-06 ***
x            2.72222    0.32151   8.467 1.67e-07 ***
I(x^2)      -0.17331    0.02853  -6.075 1.24e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.859 on 17 degrees of freedom
Multiple R-squared:  0.9109,    Adjusted R-squared:  0.9004
F-statistic: 86.92 on 2 and 17 DF,  p-value: 1.183e-09
```

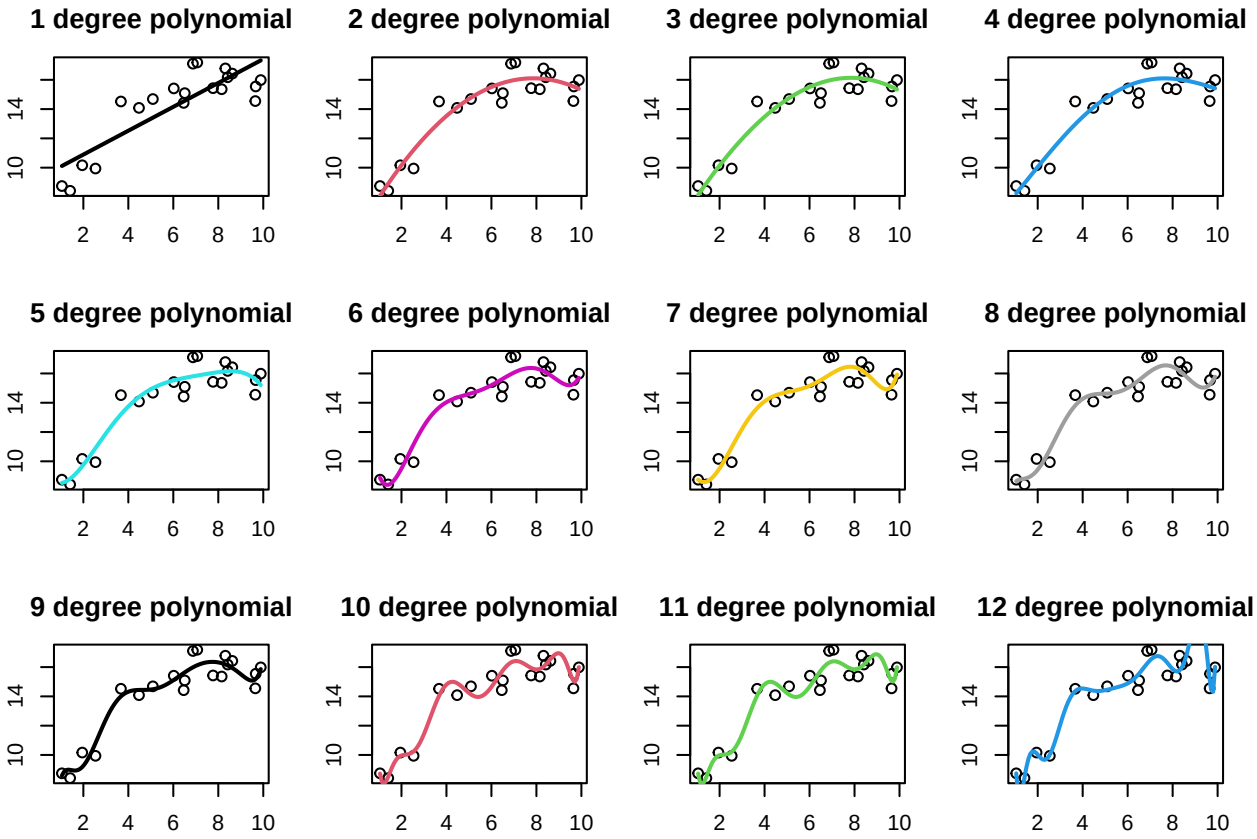
Looking at the coefficients table, we see that the quadratic coefficient is negative, which allows the model to "bend down" at higher values of x, as x^2 becomes » than x. We also see that this model explains even more variation y (~90%).



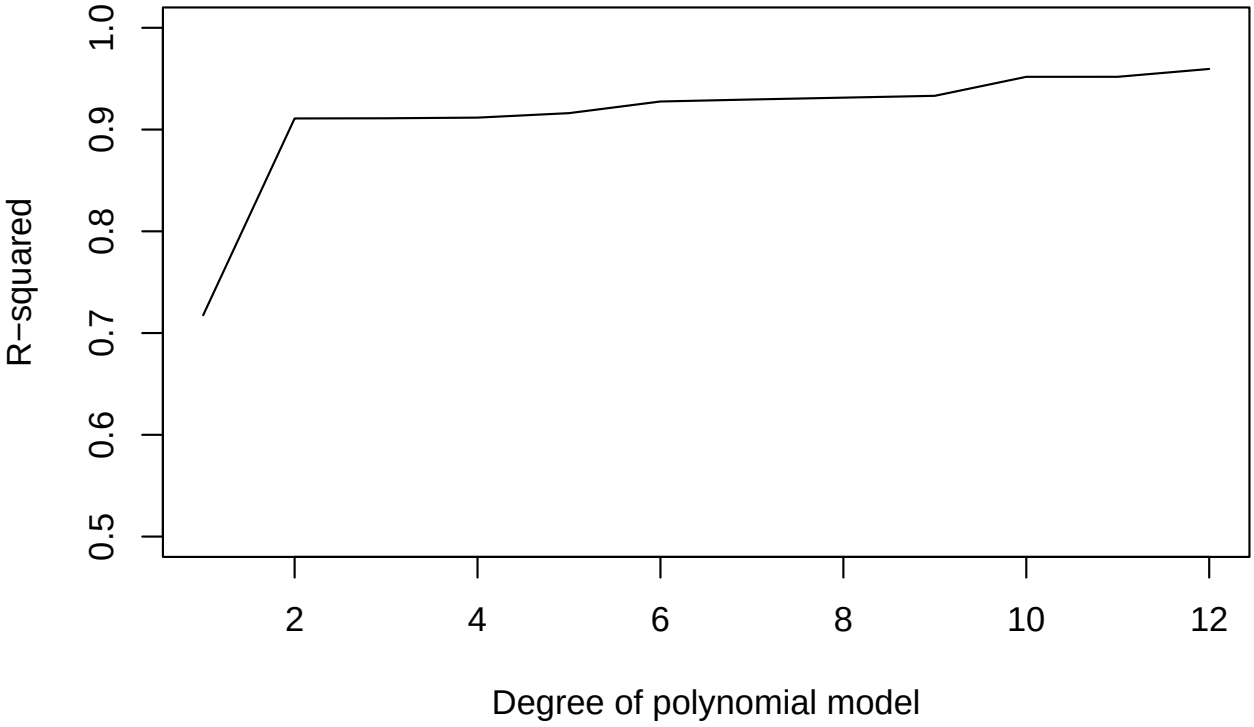
If we wanted we can keep adding higher order polynomial terms to our model:

$$\hat{y} = b_0 + b_1 x + b_2 x^2 + ... + b_n x^n$$

Below I plot out the model predictions for polynomial models as a function of the degree of the polynomial. y ~ x is a first order polynomial with two parameters, y ~ x +x^2 would be a 2nd degree polynomial with three parameters, and so on.



We can see that as our model gets more complex (more fitted parameters), it increasingly does a better job fitting the data, in the sense that the residuals get smaller as we add more terms to the model. Another way to visualize this is to plot out the R^2 as a function of the degree of the polynomial model in the model:



So more complex models will almost always be able to fit our data better. This holds not just for polynomial regression

models, but also models with multiple independent variables. Each new variable we add to our model will result in a better model fit (even if only slightly). However, the goal of fitting models to data isn't to just fit the data we have as well as possible, if that was the goal, we could just draw a line that goes through each data point. You probably already have an intuitive sense the the really complex models (e.g. degree 8 and higher) are perhaps overfitting the data, as the model predictions wiggle up and down to match the data we have as best as possible.

7.2 Cross-validation

While the goals of fitting statistical models are varied (see Advice for model selection), in general, we want our conclusions to generalize to new data. You probably already have an intuitive sense the the really complex models (e.g. degree 8 and higher) are likely overfitting the data we have and would not generalize if we had additional data. One way of assessing the ability of models to generalize to new data sets, is to use some of your data to fit the model, and then leave some of the data out to test the accuracy of the model predictions. This is called **cross-validation**. There are different ways to do cross-validation, but one simple version is called **leave-one-out-cross-validation** (LOOCV). The idea is that we leave out one data point, fit the model to the remaining data, and then evaluate how accurate the prediction was for the observation we left out. This gives us a way of assessing the **out-of-sample performance** of the model, where out-of-sample, means data the model was not fit with.

We can split out data in R as follows, where I use the sample function to randomly select one observation to be the test observation, or **test data**, and the remaining data is selected to fit the model (often referred to as the **training data**)

```
df<-data.frame(x,y)
test_row = sample(1:length(y),1)
test = df[test_row,]
train =df[-test_row,]
```

The next step is to fit the model to the training data

```
mod<-lm(y~x,train)
summary(mod)
```

```

Call:
lm(formula = y ~ x, data = train)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4330 -1.1601  0.1000  0.9896  2.1656

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   9.8691     0.8771   11.25 2.67e-09 ***
x             0.7373     0.1265    5.83 2.01e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.431 on 17 degrees of freedom
Multiple R-squared:  0.6666,    Adjusted R-squared:  0.647
F-statistic: 33.99 on 1 and 17 DF,  p-value: 2.009e-05
```

And then use this model to generate a prediction for the test observation:

```
mod<-lm(y~x,train)
predict(mod,test)
```

```
       2
10.91015
```

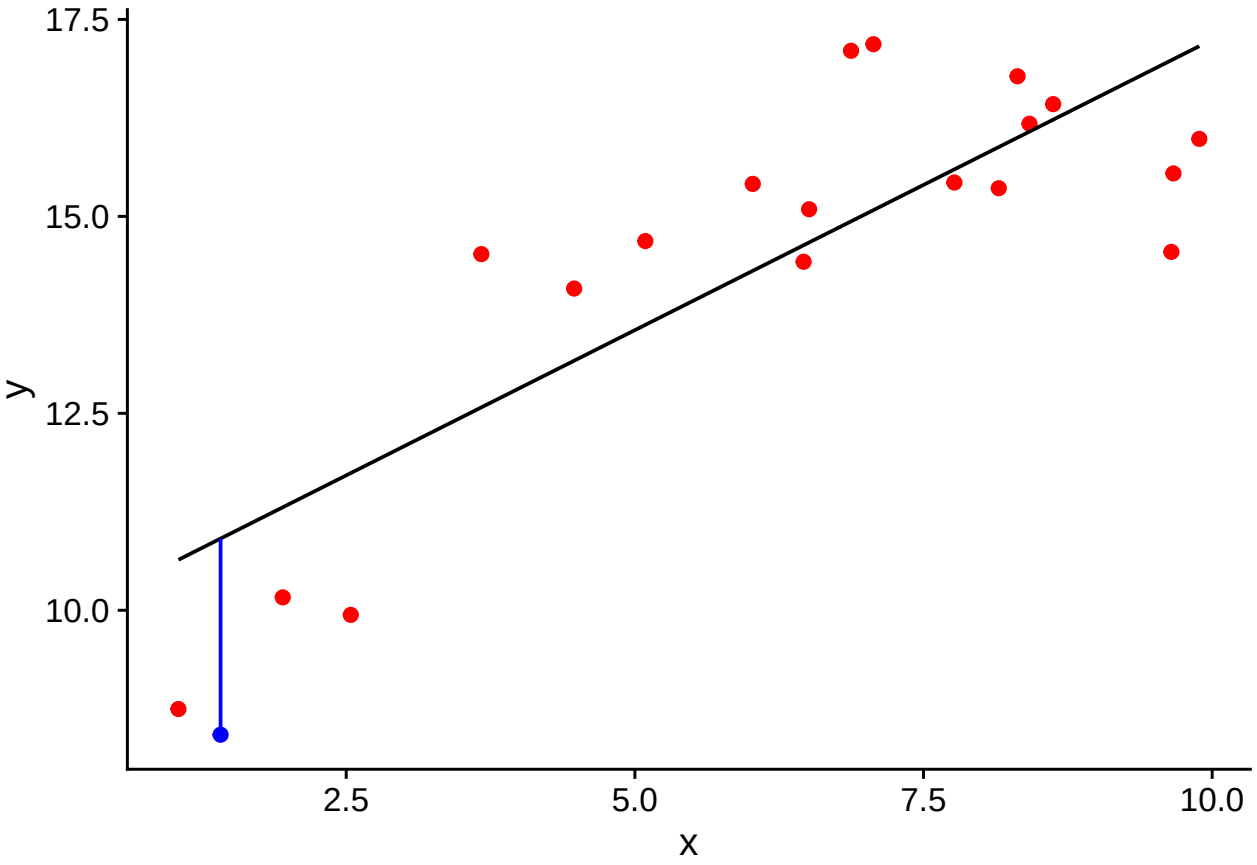
If we then compared the predicted value of y against the value of y for the test data, we see that our prediction was off by:

```
mod<-lm(y~x,train)
test$y - predict(mod,test)
```

```
        2
-2.489554
```

We can visualize this by plotting the training data in red, and the test data in blue. The blue vertical line depicts the size of the error for the testing dataset.

```
df$pred<-predict(mod,df)
test$pred<-predict(mod,test)

ggplot()+geom_point(data=train,aes(x=x,y=y),col='red')+
  geom_point(data=test,aes(x=x,y=y),col='blue')+
  geom_line(data=df,aes(x=x,y=pred))+
  geom_segment(data=test,aes(x=x,xend=x,y=y,yend=pred),col='blue')+
  theme_cowplot()
```



One problem with our method so far is that is our estimate of the prediction accuracy of the model for new data is quite variable, in that it depends a lot on which data point happened to be randomly selected for the test data. One way to reduce this variability is to repeat the analysis once for every data point in our data set. So if we have 20 data points, we fit the model 20 times, each time leaving our one data point of the training set. We can do this with a for loop:

```
df<-data.frame(x,y)
error<-rep(NA,nrow(df))
for (i in 1:nrow(df)){
  test = df[i,]
  train =df[-i,]
  mod<-lm(y~x,train)
  error[i]<-test$y - predict(mod,test)
}
```

In the code above, 'error' is a vector of the errors for each data point we left out. We can calculate various measures of goodness of fit, for example the standard deviation of the predicted errors:

```
Pred_SS_Error = sum(error^2)
sqrt(Pred_SS_Error/nrow(df))
```

```
[1] 1.591896
```

Or the predicted R^2:

```
Pred_SS_Error = sum(error^2)
SS_Total = sum((df$y-mean(df$y))^2)
Pred_R2 = 1 - Pred_SS_Error/SS_Total
Pred_R2
```

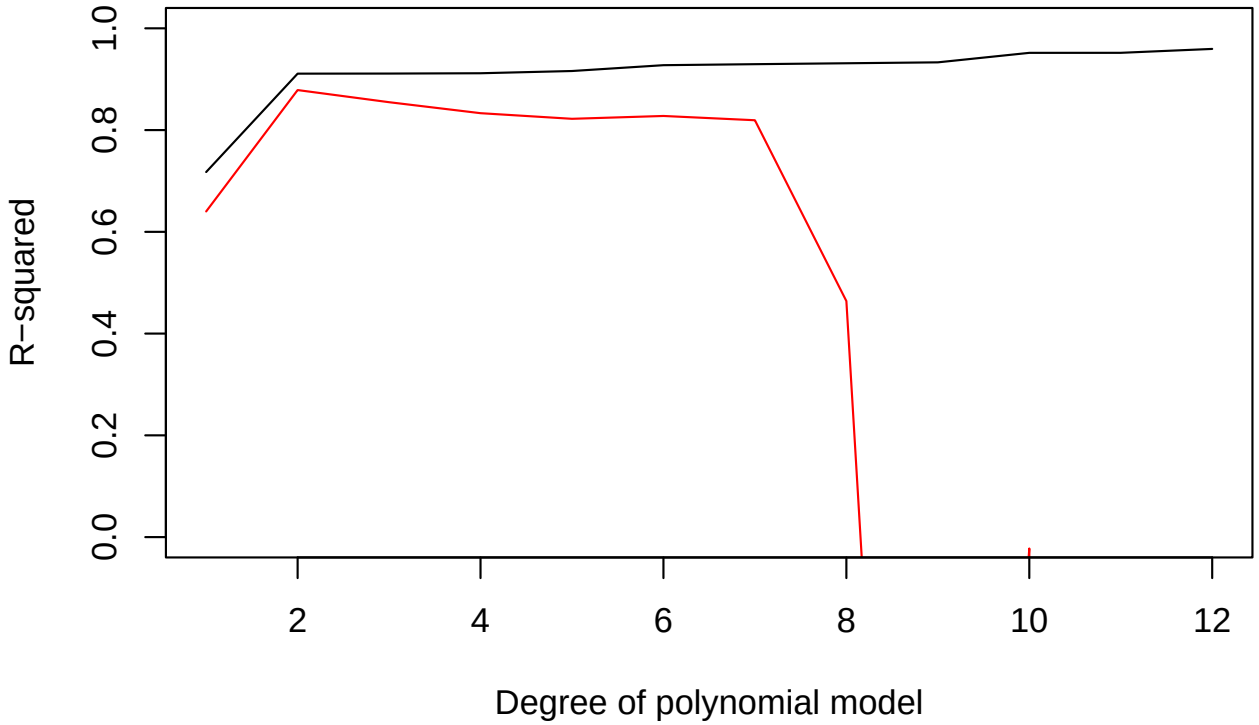
```
[1] 0.640084
```

We can use a nested for loop to repeat this analysis for increasingly complex polynomial models as follows:

```
Pred_R2<-rep(NA,12)
error<-rep(NA,nrow(df))
PredRMSE<-rep(NA,12)
for (j in 1:12){
  for (i in 1:nrow(df)){
    test = df[i,]
    train =df[-i,]
    mod<-lm(y~poly(x,j),train)
    error[i]<-test$y - predict(mod,test)
  }
  Pred_SS_Error = sum(error^2)
  PredRMSE[j] =  sqrt(Pred_SS_Error/nrow(df))
  Pred_R2[j] = 1 - Pred_SS_Error/SS_Total
}
```

If we then plot out the predicted R-squared (red line) we see that it peaks at a 2nd degree polynomial model. Furthermore, we see that the predicted R squared drops rapidly for polynomials of degree 8 or higher. This differs form the pattern we found when we looked R^2 on the whole dataset, where R^2 always improved with increasing model complexity (black line).

```
plot(1:12,Pred_R2,ylim=c(0,1),xlab="Degree of polynomial model",ylab="R-squared",col='red','l')
lines(1:12,R2)
```



So do we learn from leave-one-out-cross-validation? By repeatedly fitting our model to all but one data point and then testing against the remaining data point, we get a sense for how accurate the prediction of our model would be at predicting a new data point. You probably also noticed that the errors for the test data in LLOCV were larger than for the training data set. This is a very general result. When we assess goodness of fit on data we fit our model to, it will generally overestimate how well it would fit new observations. Thus if you want a more accurate estimate of how accurate your model predictions will be on new data, cross validation is a good choice as the goodness of fit estimates on the training data are generally biased to be to high, especially for models with many parameters.

In some cases, when you are interested in comparing many models, LOOCV can be used as a means of model selection, where models with lower LOOCV error are preferred over models with higher errors.

7.3 AIC

Another way of comparing models is with Akaike information criterion, AIC. The general idea of AIC is that you can compare different models by comparing their likelihood. This is the same likelihood that we used to find the parameters that best fit the data for a specific model with maximum likelihood estimation. We can also use likelihood to compare entirely different models (e.g. different independent variables). In general, when we add more parameters to the model, it will fit the data better, and thus the likelihood will increase as we add more terms. Thus in AIC there is a penalty that we add based on the number of parameters in the model.

AIC = - 2 LL + 2 p

where LL is the log-likelihood and p is the number of parameters in the model. If you remember from chapter 2, the likelihood is the joint probability density of the data given the parameters. To calculate the log-likelihood, we first need to calculate the predicted value of y for each observation, and the residual standard error (our estimate of sigma) for a model. So for the first order polynomial:

```
mod<-lm(y~x,df)
sigma = as.numeric(summary(mod)[6])
pdf = dnorm(df$y,predict(mod),sigma)
```

In the code above, pdf is a vector of the probability densities of each observation. They were calculated by providing dnorm() with the observed `df$y` and predicted `predict(mod)`, values of y, along with the estimate of the standard deviation of the errors, which is the residual standard error (the 6th element of the list of summary(mod)). To calculate the log-likelihood we just need to log the values in pdf, and sum them up. From this we then calculate the AIC score using the formula above:

```
  p = 2 # intercept and slope
  LL = sum(log(pdf))
  AIC = -2 * LL + 2 * p
  AIC
```

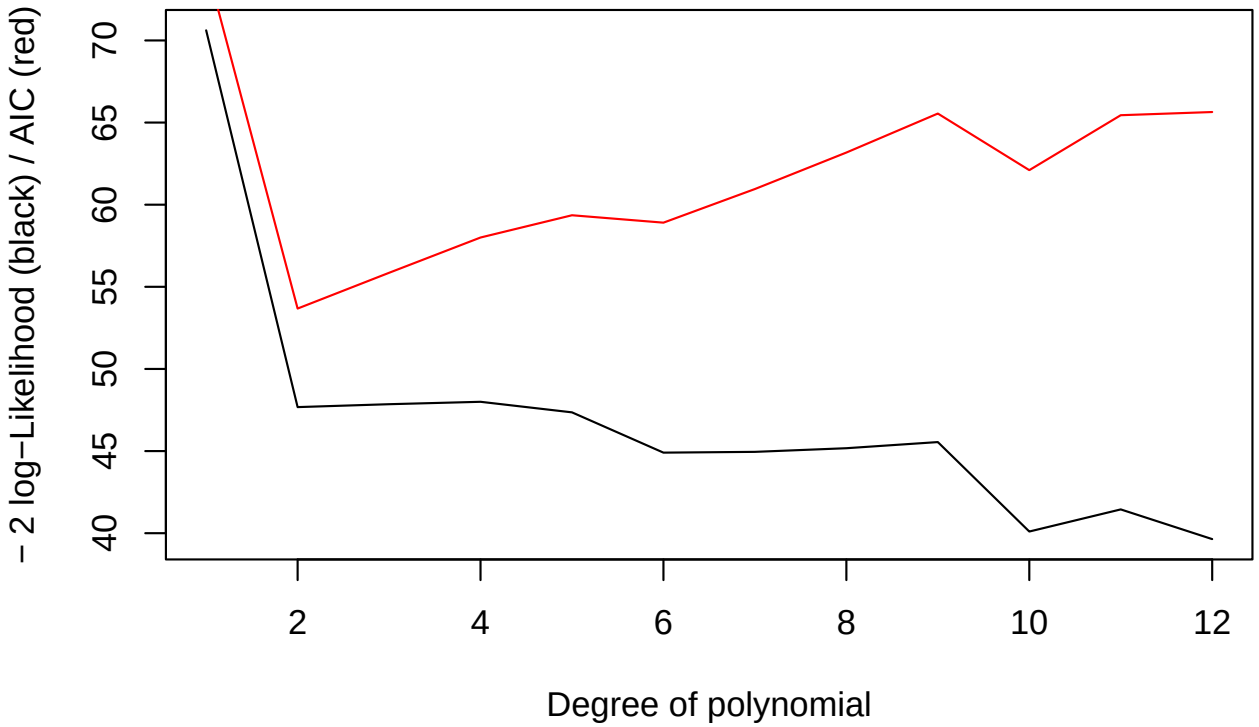
```
[1] 74.61517
```

Here we see that the AIC for this model is 74.615, which on its own doesn't tell us much. Really AIC scores are mostly useful as a relative metric of goodness of fit (releative to other models), with the model with the lowest AIC score being the preferred model. We can rerun the polynomial analysis but calculate AIC scores for each model with the following code:

```
LL<-rep(NA,12)
AIC<-rep(NA,12)

for (i in 1:12){
  mod<-lm(y~poly(x,i),df)
  sigma = as.numeric(summary(mod)[6])
  LL[i] = sum(log(dnorm(df$y,predict(mod),sigma)))
  AIC[i] = -2*LL[i] + 2*(i+1)
 }
```

If we plot out the -2 LL and AIC, we see the the negative log-likelihood keeps going down with model complexity (the likelihood goes up), but because of the parameter penalty, the AIC score is minimized at an intermediate complexities (2nd degree polynomial).

```
plot(1:12,-2*LL,'l',ylab="- 2 log-Likelihood (black) / AIC (red)",xlab="Degree of polynomial")
lines(1:12,AIC,col='red')
```



AIC is based off information theory, and AIC estimates the information lost by approximating the real world process that generated the data with a statistical model. The goal is to find the model that minimizes the information lost. We can't do this with certainty, but AIC seeks to estimate how much more or less information is lost by one model vs another. The relative probability of models losing less information than another can be calculate as:

$$\exp((AIC_{min} - AIC_i)/2)$$

where AIC_{min} is the model with the lowest AIC score in the set of models analyzed.

From these relative probabilities you can construct an AIC table, sorted by the AIC score (lowest to highest):

```
Nparms<-2:13
dfAIC<-data.frame(Nparms)


dfAIC$deltaAIC=min(AIC)-AIC
dfAIC$AICrelLike=exp(dfAIC$deltaAIC/2)
dfAIC$AICweight=dfAIC$AICrelLike/sum(dfAIC$AICrelLike)

dfAIC$AICweight
```

```
 [1] 1.730573e-05 6.092203e-01 2.052460e-01 7.013522e-02 3.560348e-02 4.460990e-02
 [7] 1.602717e-02 5.278258e-03 1.610839e-03 9.020612e-03 1.693373e-03 1.537539e-03
```

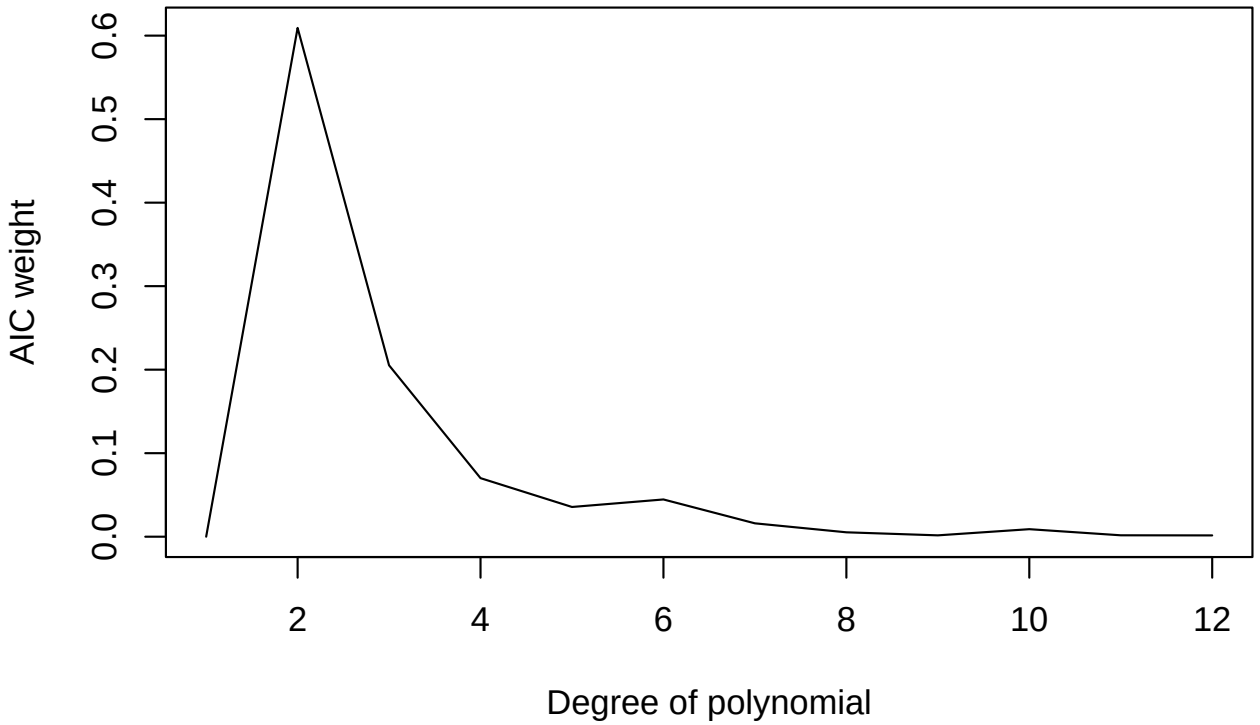
```
dfAICsort <- dfAIC[order(-dfAIC$AICweight),]
dfAICsort<-round(dfAICsort,3)
dfAICsort
```

```
   Nparms deltaAIC AICrelLike AICweight
2       3    0.000      1.000     0.609
3       4   -2.176      0.337     0.205
4       5   -4.324      0.115     0.070
6       7   -5.228      0.073     0.045
5       6   -5.679      0.058     0.036
7       8   -7.276      0.026     0.016
10     11   -8.425      0.015     0.009
8       9   -9.497      0.009     0.005
11     12  -11.771      0.003     0.002
9      10  -11.871      0.003     0.002
12     13  -11.964      0.003     0.002
1       2  -20.938      0.000     0.000
```

The first column indicates the number of paramters. The second column the difference in AIC score between the model with the minimum AIC score and the AIC score that that model (will be zero for the best model). The third column is the relative probability for each model (will be 1 for the best fitting model). The last column is the AIC weight for each model which is calculated by summing up the relative AIC probabilities for all models and dividing the the relative probabilities by the total so that the weights for all models sum to one. Thies weights can be interpreted as the probability a specific model out of all models considered is the model that minimizes information loss.

where AIC_{min} is the model with the lowest AIC score in the set of models analyzed. We can plot out the relative likelihood of the models as follows:

```
plot(1:12,dfAIC$AICweight,'l',ylab="AIC weight",xlab="Degree of polynomial")
lines(1:12,AIC,col='red')
```



There are other flavors of AIC-like model selection criteria (AICc, BIC). The generally only differ in how large the penalty is for each additional parameter.

We see with the polynomial example, that in terms of model selection, both LOOCV and AIC came to the same conclusion: the model with the lowest LOOCV prediction error also had the lowest AIC score. This will often be the case. I prefer to use LOOCV however because it is based off fewer assumptions and gives you a good estimate of how accurately your model will generate to new data.

In many cases there may be many models that perform roughly equally in either LOOCV or AIC. In these cases, it suggest that different models are roughly equally as consistent with the data, and that more data is needed to differentiate between competting models.

7.4 Advice for model selection

Often when analyzing data we have multiple models or hypotheses that we think could explain a variation in a particular variable and we are interested in comparing how consistent different models are with data. A question you probably have is what is the right way to compare models and select the most likely model or a set of most likely models to focus on. Unfortunately, there is no general answer to this question.

The problem is that we use statistical models for many different purposes (hypothesis testing or inference, data exploration, prediction) and in different contexts (analyzing data from an randomized controlled experiments vs. observational data). For each of these different purposes and contexts we might want to use different strategies to compare models. Below I will outline a couple different contexts we might want to compare models, and what you might want to consider when comparing models.

The only general advice I have about model comparison, is to come up with a plan of how you want to analyze your data before hand. What models do you want to compare? What confounding variables do you think you need to control for? Write down your plan ahead of time, and try to stick to it. If you realize in the middle of analysis that you want to do include a different variable, you can do it, but make sure you record you change your planned and note the reason why you changed your plan. It is good to keep a list of every model that you fit to data, along with the results for each model. When you write up your results, you should also state all the models you tested (and why you tested them). If you looked at a large number of models it may make sense to report the results from all the models in a supplementary table. Such a table might include the a list of which variables were included in the model, the coefficients for each parameter in the model, the standard errors for each coefficient, and one or more metrics of goodness of fit (e.g R^2).

7.4.1 Analysis of experiments

The most straightforward application of model comparisons is for the analysis of randomized controlled experiments. Here we have the advantage that if your experiment was conducted properly, the independent variables should not be correlated with one another. The most common type of experiment that looks at multiple variables is a factorial design. I would generally begin my analysis with the full model that includes each factor and its interaction (at least for a 2-way factorial experiment). If the confidence interval for the parameter associated with the interaction term overlaps with zero, this means that there is a lot of uncertainty about the sign of the interaction given the dataset. In this case it may make sense to look at a simpler model with just the main effects.

7.4.2 Exploratory data analysis

Goal: identify patterns and generate hypotheses

Priority: thoroughly search for patterns

Prior hypothesis: non required

Statistical tools: visualization, correlations, anything goes

Pitfalls: fooling yourself with overfitted models

Sometimes there are many possible variables of interest that might affect your dependent variable. A common example of this is you have data on species abundance or growth rates over time, and then a series of climate variables.

For example, you have daily temperature and precipitation data, but only annual estimate of the abundance of some species. You think that it is likely that temperature and precipitation affect the population dynamics of your species, but you aren't sure exactly which aspect of these variables is most important. For example is it the mean temperature in a year that matters? Or the maximum temperature? Or the minimum temperature? Or is it the variability in temperature? Or the temperature is September? For just this one variable, temperature, we could come up with hundreds of ways to aggregate the variable. In many contexts we find ourselves in the situation where the number of independent variables exceeds the number of data points. The risk with this type of analysis is that if we test many models, there is a good chance we might find a model that fits the data well just due to random chance. It is always important to keep this in mind when you are doing an exploratory analysis: the goal is to identify interesting patterns and generate hypotheses, but finding a strong relationship should not be taken as strong evidence for a hypothesis. Hypothesis testing must be conducted on a independent data set that wasn't used to generate the hypothesis. Testing hypotheses with the data used to generate them is much like noticing a coincidence, and then assuming it can't be a coincidence because the odds are too low. For example, it would be like finding out an acquaintance has the same birthday as you, and coming to the conclusion that it can't be a coincidence, because the probability that you share a birthday is 1/365 which is less than 0.05.

If you want to improve the chances that the patterns identified by an exploratory analysis aren't spurious, then it is good to limit the models you consider to those that seem plausible. If most of the models you test are implausible, there is a good chance the best models you identify in your analysis with be spurious correlations.

You can use whatever method you like to come up with model in an exploratory analysis. However it is a good idea keep track of how you do the exploratory analysis. For example, one strategy would be to decide on a list of independent variables, then look at the correlation coefficients between each independent variable and the dependent variable and only keep variables with a correlation coefficient above some threshold. Then with this limited set of variables you can look at more complex models (multiple terms or interactions). Alternatively you could use visualization (plots) to try to identify patterns in the data. It is harder to be systemic this way, but at least try to describe the processes of which variables you plotted, and how you went about looking for more complex models.

7.4.3 Inference: Testing hypotheses with observational datasets

Goal: test a hypothesis

Priority: avoid false positives

prior hypothesis: required

statistical tools: parameter estimates, CI, and NHST

pitfalls: doing an exploratory analysis, but then presenting results as a test of a hypothesis

In some cases, we already have a hypothesis or theoretical expectation we wish to test with data. This expectation can either come from a theoretical model or the scientific literature, for example the hypothesis could be based on a pattern identified by a previous exploratory study. In this case, we already have a strong expectation about what we should see in nature, and thus in this case we should really only be testing a very small number of models (ideally just one or two). In general, when our goal is inference, we are asking questions about parameters in our statistical model. For example, we might hypothesize that rainfall is a key variable limiting primary productivity in terrestrial environments, and then our hypothesis is that there should be a positive relationship between rainfall and primary productivity. Or our hypothesis might be that rainfall and sunlight determine primary productivity. In either case, we want to test only a limited set of models. If you can't limit yourself to a small set of models than you are probably doing an exploitative analysis rather than an inferential analysis.

When you test many different models, reporting a significant result as a test of a hypothesis is problematic because the probability of a false positive is high. If I look at the relationship between primary productivity and 20 different independent variables, chances are I will find a "significant" relationship even if no true relationship exists.

Another thing that is important to keep in mind when testing hypotheses, is whether the data at hand provide a strong test of the hypothesis. For example, say I hypothesized that warmer temperatures were causing a collapse of a

threatened fish population, and then fit a statistical model for population growth rate as a function of temperatures and found a significant effect. Is this strong evidence for my original hypothesis? Not necessarily. For example if there are other plausible mechanisms that could have resulted in the collapse of another fish population that are correlated with temperature (overfishing, habitat destruction), then really the data may not necessarily provide strong support for one hypothesis over the other. In this cases, critical thinking is very important. Are there reasonable alternative hypotheses that could explain the trend? If so it is good practice to try to think of other types of data that might discriminate between competing hypotheses, or to try to use experiments to rule out potential explanatory mechanisms.

7.4.4 Robustness checks

With hypotheses testing I personally like to see robustness checks. You have a main hypothesis you want to test using some data set. But in addition to this model, it is useful to check a handful of other reasonable models that could alternatively explain the data. These models should be treated differently then the main hypotheses that are the focus of you analysis, and should not be used for inference, but rather they are a check to see if alternative hypotheses could also explain the pattern you observed. If your original hypothesis compares well against other plausible hypotheses, or the effect you observe is robust to controlling for different variables, this suggest that the result you found is relatively robust, and at least cannot be easily explained by alternative hypotheses. For example my hypothesis is that Y increase with X, but then I also show that alternative hypotheses Y and Z cannot explain variation in Y, this provides additional support to my original hypothesis. Or if I get roughly the same estimated effect of X_1, even after controlling for X_2 and X_3, it suggests the estimated association between X_1 and Y is robust to controlling for other variables of interest.

7.4.5 Null hypothesis tests with large observation datasets are meaningless

Another word of warning related to hypothesis tests with large data sets: If you are analyzing a very large observational data set, there is a good chance that almost any variable would come out as statistically significant on its own. This is because there are generally complex patterns of correlation among variables, and if you have enough data, even small differences among groups will come out as significant. For example, you probably see a headline at least once a week about food X being good or bad for your health. However what we eat is correlated in complex ways with other aspects of our lives: our cultural backgrounds, education, income, etc.. For example if someone looked at a large dataset to test the hypothesis that the risk of getting covid is different for people that drink coffee, vs. don't drink coffee, they would almost certainly reject the null hypothesis. This probably not because coffee has any causal link to getting Covid, but rather that drinking coffee is associated with many other aspects of our lives that might have some influence of getting Covid. Now, if it turned out that people who drank coffee were much more likely to get covid (e.g. 4 times higher), and the result was robust to many controlling for many potential confounding variables (income, profession, ect.) then it might be worth looking further into whether there is a potential mechanistic link between coffee and Covid. However, if the effect size was relatively small and sensitive to which variables we control for, I wouldn't take rejecting the null hypothesis as strong evidence for a link between coffee and Covid.

7.4.6 Prediction

Goal: predict the dependent variable as accurately as possible

Priority: minimize prediction error

Prior hypothesis: not required, but can be help for picking variables

Statistical tools: AIC, out of sample validation

pitfalls: failure to validate predictions with out of sample data

Sometimes we just want to be able to predict the value of the dependent variable and we don't really care about what is in the model so long as it makes accurate predictions. For example, if I was a weather forecaster, I might not care about precisely estimating the relative importance of different independent variables for tomorrows temperature, I just want my forecasts to be as accurate as possible.

When the goal is prediction, analyses begin in a similar way to exploratory analyses. You might want to come up with a list of candidate models you want to test, which can either be based on an exploratory analysis of the data or informed by theory or the literature as to which variables might be useful for predicting the dependent variable of interest. Either AIC or LOOCV can be used for selecting which model to use to make your prediction. However one important difference, is that when your goal is prediction, and you want to truely asses the accuracy of the models predictions, you should leave some faction of the data for an additional level of cross validation. For example, I might use 70% of the data for model selection and leave 30% for testing. On the 70% data I can use either AIC or LOOCV to identify a preferred model that I will use for prediction, and then after I have decided on the model, I can test it's performance on the remaining 30% of the data. The performance on the out-of-sample data give you an estimate of how well you model might perform on new data. You might think that if you are already doing LOOCV, that you don't need an additional round of cross-validation. However if you are using LOOCV to compare the performance of a large set of models, the "best" model might have performed well in part due to random chance. So it is a good idea to have an additional level of cross-validation to assess the predictive performance of your model.

Appendix

8 Introduction to R

R is a programming language that is often used for biological data analysis. In the following tutorial, we will discuss some of the basics of working in R. You might already know how to do it, but for those of you who don't (or for whom it's been a while) it will probably help with the assignments.

8.1 Installing packages and working directories

Your R session is always stored in a working directory. Importing data sets is much easier when your R session is in the same working directory as the file you want to import. You can check and change the working directory the following ways:

```
# Find the current working directory
getwd()

# Change the current working directory to a specified location
setwd("C://file/path")
```

When you work in R (most likely in RStudio), you might have to import new packages. For most packages, you can do so in the following way:

```
# Install packages (for example 'ggplot2')
install.packages("ggplot2")

# Load package into memory
library(ggplot2)
```

8.2 Basics of R

8.2.1 Basic data types

R has 5 basic classes, which are important to understand as trying to manipulate the wrong way might lead to frustrating errors. Each data type is used to represent some type of info - numbers, strings, Boolean values, etc.

- logical (e.g., TRUE, FALSE)
- integer (e.g., 2L, as.integer(3))
- numeric (real or decimal) (e.g., 2, 15.5)
- complex (e.g., 1 + 4i, 1 + 0i)
- character (e.g., "a", "words", "names")

8.2.2 Basic data structures

R has 6 basic data structures, which are the vector, list, matrix, data frame, array and factor. Some of these require that all elements are of the same data type (vectors and matrices) while in others different data types can be combined (lists and data frames).

Vectors Vectors are the most common basic data structure in R and contain elements that are all of the same type and are single-dimensional. You can create a vector using the `c()` function. Vectors can contain all data types.

```
# Create a vector
vec = c(1,2,3,6,8,20)
```

```
vec
```

```
[1]  1  2  3  6  8 20
```

```
# Select the 4th element in the vector
vec[4]
```

```
[1] 6
```

Lists Lists are similar to vectors, but can contain different data types. To create a list, you use the `list()` function. Lists can contain other data structures as well as data types, even other lists. Lists are one-dimensional but can contain multidimensional data structures.

```
# Create a list
lst = list(1, c(1,2,3), "hello", FALSE, 5+4i, 5L, matrix(1:9, nrow = 3, ncol = 3))

lst
```

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

[[2]]
[1] 1 2 3

[[3]]
[1] "hello"

[[4]]
[1] FALSE

[[5]]
[1] 5+4i

[[6]]
[1] 5

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

```
# Select the 4th element in the list
lst[[4]]
```

```
[1] FALSE
```

```
# Select the 3nd element in the vector (2nd element in the list)
lst[[2]][3]
```

```
[1] 3
```

Matrices Matrices are two-dimensional data structures that contain elements of the same data type. If you make a matrix, the following syntax is used:

```
matrix(data, nrow, ncol, byrow, dimnames)
```

Where **data** defines the input values given as a vector, **nrow** and **ncol** define the number of rows and columns of the matrix, respectively. **byrow** tells the matrix to be constructed row-wise if TRUE (and is FALSE by default). **dimnames** is a list of row/column names.

```
# Create a matrix containing the numbers 1 to 9, with 3 rows and 3 columns
mat = matrix(c(1:9), nrow = 3, ncol = 3)

mat
```

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

```
# Selecting the '7' using matrix[i,j]
mat[1,3]
```

```
[1] 7
```

Data Frames Data frames are two-dimensional data structures that are essentially lists of same-length vectors. Data frames have the following constraints:

- They must have column-names and each row should have a unique name
- Each column should have the same number of items
- Each column should consist of items of the same type (a vector)
- Different columns can have different data types

```
# Example of a data frame
student_id <- c(1:5)
student_name <- c("raj", "jacob", "iqbal", "shawn", "hitesh")
student_rank <- c("third", "fifth", "second", "fourth", "first")
student.data <- data.frame(student_id , student_name, student_rank)

student.data
```

```
  student_id student_name student_rank
1          1          raj        third
2          2        jacob        fifth
3          3        iqbal       second
4          4        shawn       fourth
5          5       hitesh        first
```

```
# Indexing in a data frames is different than in matrices
student.data$student_id
```

```
[1] 1 2 3 4 5
```

Arrays Arrays are three-dimensional data structures, consisting of elements on the same data type. They are essentially stacked matrices. If you want to create an array, the following syntax is used:

```
array(data, dim, dimnames)
```

Where **data** defines the input values, **dim** is a vector containing the dimensions of the array and **dimnames** is a list contains the names of the rows, columns and matrices in the array.

```
arr = array(c(1:18), dim = c(2,3,3))
arr
```

```
, , 1

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

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

     [,1] [,2] [,3]
```

```
[1,]   13   15   17
[2,]   14   16   18
```

Factors Factors are vectors that can only store predefined values, and are useful for storing categorical data. Next to the elements, they also have **levels**, which is the set of allowed values for the elements.

```
fac = factor(c("green", "red", "green", "blue", "red"))

fac
```

```
[1] green red   green blue  red
Levels: blue green red
```

8.2.3 Comparisons

In R, you can use comparisons to check whether certain conditions are true. When the statement is correct, R returns `TRUE` and if the statement is not R returns `FALSE`. The logical operators `==` and `!=` can be used to compare all kinds of objects. For numeric objects, you can check if they are lesser or greater than other numeric values as well. The following comparison operators exist in R:

==	equal
!=	not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
|	or
!	not
%in%	in the set

These operations can be combined with `&` or `|`. `&`, meaning "and", is used when you want to check more than one comparison where ALL comparisons must return TRUE for the entire statement to be TRUE. `|`, meaning "or", is used when you want to check more than one comparison where ANY comparison returns TRUE.

```
1 == 1 & 2 != 5
```

```
[1] TRUE
```

```
2 != 3 & 5 == 6
```

```
[1] FALSE
```

```
1 == 2 | 3 != 2
```

```
[1] TRUE
```

```
"tracie" != "4" | "bill" != "joe"
```

```
[1] TRUE
```

"&" vs "&&" and "|" vs "||" When you want to use "and" in your code, there are options `&` and `&&`. The single `&` compares all elements in a vector and returns a logical vector. The double `&&` only compares the first element, even if it operates on vectors:

```
# Create a vector
x <- c(6,5,2,0,1,8)

# Compare every element in the vector
x > 1 & x < 5
```

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

```
# Compare the first element of the vector
x[1] > 1 && x[1] < 5
```

```
[1] FALSE
```

```
x[1] > 1 && x[1] < 8
```

```
[1] TRUE
```

One thing to note is that `&&` only works on single logical values, e.g., logical vectors of length 1 (like you would pass into an if condition), but `&` also works on vectors of length greater than 1. When using with `if` statements, it is thus better to use `&&` and make sure that both connected elements are single values of `TRUE` and `FALSE`.

The same thing goes for `|` and `||`.

Difference between '%in%' and 'in' The %in% operator is used to identify if an element belongs to a vector or data frame. The result is a logical value.

```
# Create values and vector
v1 = 3
v2 = 101
vec = c(1,2,3,4,5,6,7,8)

# Check if v1 is part of the vector
v1 %in% vec
```

```
[1] TRUE
```

```
# Check if v2 is a part of the vector
v2 %in% vec
```

```
[1] FALSE
```

Do not mistake %in% with in, which is used in for-loops to run through all the elements in the vector! (See the section about for-loops for an example).

8.2.4 TRUE and FALSE

R uses two logical values, TRUE and FALSE. They should be capitalized and not wrapped into quotes for R to recognize them. 0 corresponds to FALSE, while 1 corresponds TRUE. You can perform calculations using these logical values.

```
z = c(TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, FALSE, TRUE, FALSE)

# Calculates the number of times TRUE is in list z
sum(z)
```

```
[1] 4
```

```
# Multiplication shows that TRUE == 1 and FALSE == 0
z1 = z * -4
z1
```

```
[1] -4  0 -4  0 -4  0  0 -4  0
```

```
# The first element in z is TRUE, while it's not anymore after multiplication
isTRUE(z[1])
```

```
[1] TRUE
```

```
isTRUE(z1[1])
```

```
[1] FALSE
```

```
# Recreating a logical vector from z1 returns all non-zero values as TRUE
z2 = as.logical(z1)
z2
```

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

```
# And all those values are then returned back to 1's and 0's
z3 = as.numeric(z2)
z3
```

```
[1] 1 0 1 0 1 0 0 1 0
```

You can easily check which values in a vector meet a certain condition, using logical vectors. These logical vectors can be used to index any vector with the same length.

```
lengths = c(0.5, 0.7, 0.9, 1.5, 1.6, 0.4, 1.2, 3.0)

# Which lengths are > 1.0?
lengths > 1.0
```

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

```
# Which lengths are equal to 1.6?
lengths == 1.6
```

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

```
# What were the lengths that were above 1.0?
lengths[lengths > 1.0]
```

```
[1] 1.5 1.6 1.2 3.0
```

8.2.5 seq() and rep()

When creating a vector, you can do so by typing in all your values by hand, using `c(value1, value2, etc)`, but if your vector is very large this becomes very annoying. When your vector consists of a sequence with constant steps between the values, you can use `seq(from, to, by)`. When you want to make a vector consisting of the same value repeatedly, you can use `rep(value, times)`.

```
# Create a vector by hand
v1 = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
v1
```

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

```
# Create a regular sequence
v2 = seq(1, 6, 0.5)
v2
```

```
 [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
```

```
# Create a vector consisting of a repeated value
v3 = rep(5, 7)
v3
```

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

8.3 If-statements

You use `if` statements when you only want to perform an action if certain conditions are true. The syntax of the `if` statement is:

```
if (condition) {
do something
}
```

If the condition is `TRUE`, the statement gets executed. But if it's `FALSE`, nothing happens. The condition can be a logical or numeric vector, but only the first element will be checked. When the vector is numeric, only zero is taken as `FALSE`, the rest is `TRUE`.

The `if...else` statement has an additional action if the condition is `FALSE`. It is essentially the same as the `if` statement if you would state to do nothing when the condition is `FALSE`. The syntax of the `if...else` statement is shown below (**note**: the `else` must be in the same line as the closing braces of the `if` statement):

```
if (condition) {
do something
} else {
do something else
}
```

You could make a ladder of statements when there are more than two options. The syntax you would use in this case is:

```
if (condition1) {
do something
} else if (condition2) {
perform action 2
} else if (condition3) {
perform action 3
} else {
perform action 4
}
```

To recap:

- The essential characteristic of the `if` statement is that it helps us create a branching path in our code.
- Both the `if` and the `else` keywords in R are followed by curly brackets { }, which define code blocks.
- R does not run both, and it uses the comparison operator to decide which code block to run.

8.4 Loops

Programming loops are nothing more than an automatic version of a multi-step process in which you group the steps to be repeated, and perform them multiple times with only a few instructions.

There are two types of loops in programming. The first type consists of loops that execute for a prescribed number for times, controlled by a counter or index. These loops are `for`-loops. The second type consists of loops that are based on the verification of a logical condition. This condition is checked at the start or the end of the loop. These loops are `while` or `repeat` loops, respectively.

8.4.1 For-loops

`for`-loops specify the number of times a certain action needs to be performed. For example, if you want to perform an action for all values in a vector, using a `for`-loop saves you a lot of time. `for`-loops have the following general code:

```
for (every item in vector) {
    do something
    }
```

You can define a vector beforehand and use the specific values in the vector as 'item' (i).

```
a1 = seq(0.1, 0.8, 0.1)
for (i in a1) {
    print(i)
    }
```

```
[1] 0.1
[1] 0.2
[1] 0.3
[1] 0.4
[1] 0.5
[1] 0.6
[1] 0.7
[1] 0.8
```

When you need the 'item' to refer to the index of the values in stead of the real values (for example, when you want to use different data sets in one for-loop or you want to store the output in a new vector), you can define the for-loop-vector using 'for (i in start value : end value)', as is done below:

```
# Create a vector filled with random normal values
u1 = rnorm(n = 8)

# Initialize the vector containing the squared values
usq = 0

# Calculate squared number for all values in u1
for(i in 1:length(u1)) {
  # i-th element of `u1` squared into `i`-th position of `usq`
  usq[i] = u1[i] * u1[i]
  print(usq[i])
}
```

```
[1] 7.241391
[1] 0.3578144
[1] 2.596496
[1] 0.1913024
[1] 1.829583
[1] 0.2214347
[1] 0.2211333
[1] 0.1570606
```

```
print(usq)
```

```
[1] 7.2413906 0.3578144 2.5964963 0.1913024 1.8295825 0.2214347 0.2211333 0.1570606
```

Tip: when you get overwhelmed by all the loops, you can use the `print()` statement to see all the steps in the loop. 'print(i)' helps to make sense of which iteration has which result.

Nested for-loops Nested for loops are loops that contain other loops in their 'do something'-section. This comes in handy when you work with data sets and matrices, as they need two indices: one for their rows and one for their columns. An example of a nested for loop is shown below (remember that indexing is done as data[row, column]):

```r
# Create a 30 x 30 matrix (of 30 rows and 30 columns)
mymat = matrix(nrow=10, ncol=10)

# For each row and each column, assign values based on position
for (i in 1:nrow(mymat)) {
  for (j in 1:ncol(mymat)) {
    mymat[i,j] = i*j
  }
}

# Show the matrix
mymat
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1    2    3    4    5    6    7    8    9    10
 [2,]    2    4    6    8   10   12   14   16   18    20
 [3,]    3    6    9   12   15   18   21   24   27    30
 [4,]    4    8   12   16   20   24   28   32   36    40
 [5,]    5   10   15   20   25   30   35   40   45    50
 [6,]    6   12   18   24   30   36   42   48   54    60
 [7,]    7   14   21   28   35   42   49   56   63    70
 [8,]    8   16   24   32   40   48   56   64   72    80
 [9,]    9   18   27   36   45   54   63   72   81    90
[10,]   10   20   30   40   50   60   70   80   90   100
```

Note: Remember that every for-loop needs its own set of curly braces, each with its own block and governed by its own index.

8.4.2 Alternative loops

Sometimes, `for`-loops are not the best option, when you don't know or control the number of iterations, for instance. For example, when you need to count the number of clicks on a web page within a specified number of days or other unpredictable events. In these cased, you can use a `while`- or `repeat`-loop.

While loops `while`-loops often make use of a comparison between a control variable and a value, checking if the value is greater than the control variable, for instance. This check results in a logical response, either `TRUE` or `FALSE`. When the result is `FALSE`, the block of code is not executed and the program will continue with the code below the loop. If the result of the check is `TRUE`, the block of code gets executed. `while`-loops only stop running when the result of the check changes from `TRUE` to `FALSE`, so updating the control variable in the loop or adding a increment to a counter is needed for the loop to ever stop running. Note that the code will never run if the result is `FALSE` at the first check. An example of the `while`-loop is shown below.

```r
a = 1
while (a < 6) {
print(a / 5)
# a is updated every iteration
a = a + 1
}
```

```
[1] 0.2
[1] 0.4
```

```
[1] 0.6
[1] 0.8
[1] 1
```

Repeat loops and break The `repeat`-loop is similar to the `while`-loop, but its block of code is executed at least once, no matter the result of the condition. You could call this loop 'repeat until', as it is executed until the condition changes and the loop 'breaks'. When R encounters a `break`, it leaves the loop and continues with the code below it (if any). If the `break` is part of a nested loop, it will only exit the loop it is a part of. An example of the `repeat`-loop is shown below.

```
x = 1
repeat {
  print(x)
  x = x + 1
  if (x == 6){
    break
  }
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Note: be careful to specify the termination-condition when you use the `repeat`- loop, as you might end up in an infinite loop otherwise!

8.5 The runif() function

The `runif()` function generates random deviates of the uniform distribution (and has nothing to do with if-statements). `runif()` needs an input 'n', 'min' and 'max', which correspond to the number of values between the 'min' and 'max' values. It can be used to produce random numbers, and has fractional numbers as output.

The `runif()` function is useful when simulating probability problems, as show below.

```
# Make a vector of probabilities
x = runif(10, min=0, max=1)
# Check for every value in the vector x if it exceeds 0.6. If so, print that value
for (i in 1:length(x)) {
    if (x[i] > 0.6) {
    print(x[i])
    }
}
```

```
[1] 0.7431598
[1] 0.6412899
[1] 0.890269
[1] 0.7646008
[1] 0.83478
```

```
[1] 0.9935257
[1] 0.9099027
```

8.6 The sample() function

The `sample()` function draws a random sample of a specified size from a population of values, either with replacement or without replacement. The syntax of the `sample()` function is as follows:

```
sample(x, size, replace, prob)
```

Here `x` is a vector of values. You can use for example the range `(1:10)` for `x` if you want to draw a random sample from the numbers 1 to 10. `size` specifies how many values are to be drawn. `replace` can either take the value `TRUE` or `FALSE` depending on whether you want to sample with or without replacement. The argument `replace` is optional and takes as default value `FALSE`. **Note, however, that if you sample without replacement the maximum number of items that can be drawn equals the length of the first argument `x`.** Finally, the argument `prob` can be used to specify what the probability is to draw each of the elements of the vector `x`. This is also an optional argument, the default value is that each of the the elements in `x` has the same probability to be drawn.

The `sample()` function is useful when simulating chance processes. For example, we can simulate the process of tossing a coin 100 times with a single call to `sample()`:

```
# Possible outcomes are head (0) or tails (1)
values = c(0, 1)

# Toss a coin 100 times
outcomes = sample(values, 100, replace = TRUE)

# How many times did you throw tails?
sum(outcomes)
```

```
[1] 46
```

8.7 Data Frames and plotting data

8.7.1 Import data frames

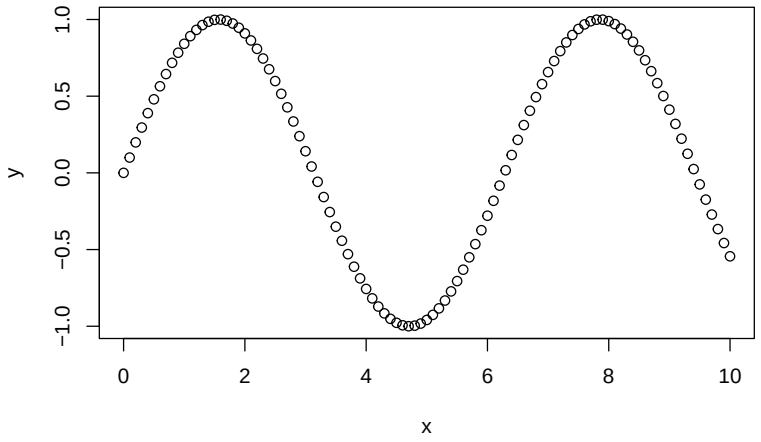
When you want to import a (.csv) data frame, you can do so using the following code. This is easiest when your working directory contains the file you want to import.

```
# Read specified .csv file
df = read.csv("filename.csv")

# You can write in a .csv file too
write.csv(df, "filename.csv")
```

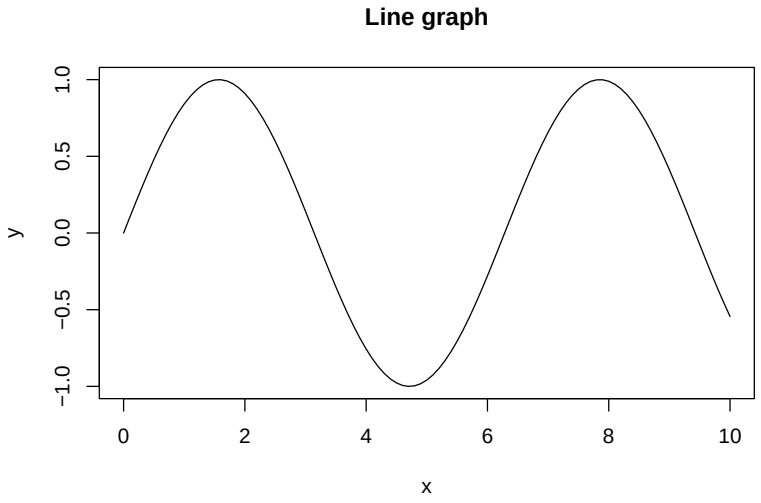
8.7.2 plot()-function

The simplest function functions to plot your data is the `plot` function. It allows you to create a plot based on two vectors (of the same length), a data frame, a matrix or other objects. The function has different arguments you can specify, which can be found when you run `?plot`.

```
# Generate data
x = seq(0, 10, by=0.1)
y = sin(x)

# Plot the data (without specifying anything)
plot(x, y)
```

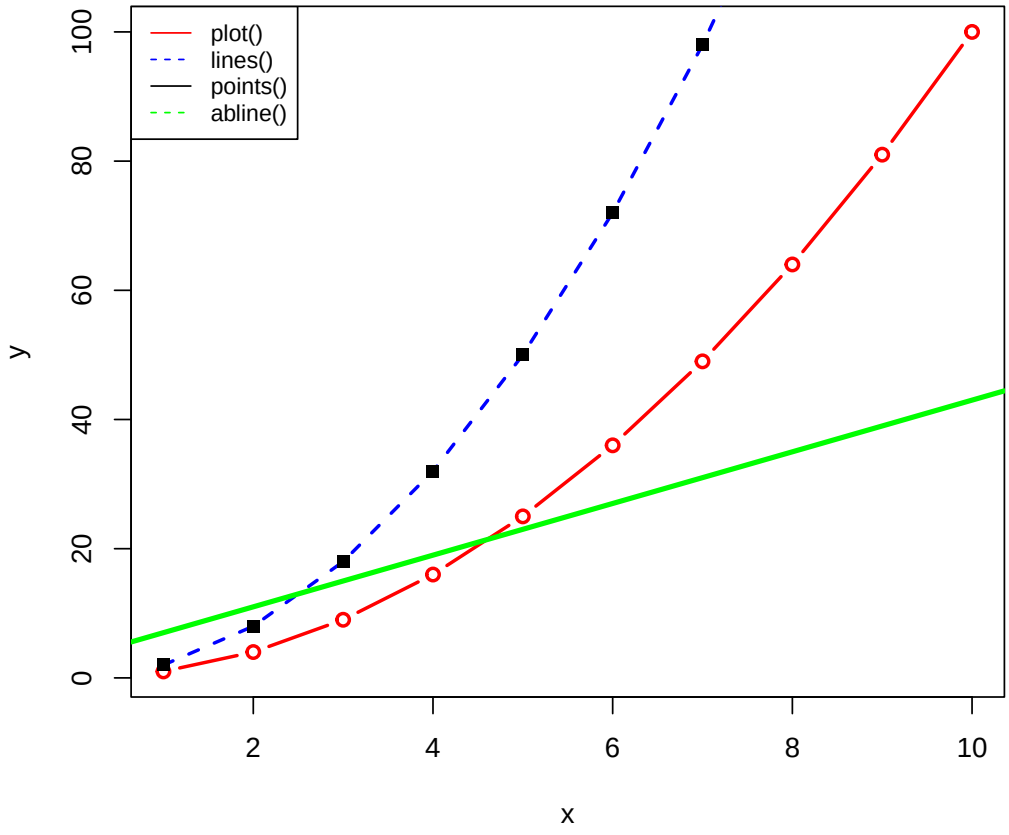



```
# Plot data using a different type and add a title:
plot(x, y, type = "l", main = "Line graph")
```



abline(), lines() and points() When you want to add an additional line to a graph made using `plot()`, you can so using either `abline()` or `lines()`. `abline()` creates a straight line using the intercept and the slope, while `lines()` uses a set of vectors to construct a line. `points()` also uses a set of vectors, but plots them as points rather than a line. Note that all require you use `plot()` first, as they cannot produce a plot on their own. An example is shown below (check the different style options such as lwd ('line width'), col ('color'), lty ('line type') and pch ('plotting symbols').

```
# Create some vectors
x = 1:10
y1 = x * x
y2 = 2 * y1

# Create plot of first line
plot(x, y1, type="b", col="red", xlab="x", ylab="y", lwd=2)

# Add second line (as well as points)
lines(x, y2, type="l", col="blue", lty=2, lwd=2)
points(x, y2, col="black", pch=15, lwd=3)

# Add a straight line specifying the intercept and slope
abline(3,4, col="green", lwd=3)

# Add a legend to the plot
legend("topleft", legend=c("plot()", "lines()", "points()", "abline()"),
col = c("red", "blue", "black", "green"), lty= 1:2, cex=0.8)
```



8.7.3 hist() function

The `hist()` function can be used to create and plot a histogram of results. The syntax of the `hist()` function is:

```
hist(x, breaks, freq, right, .....)
```

The 3 dots at the end of this statement means that there are more arguments that you can add, but that these are optional. For a full description of these optional arguments check out the help page on the `hist()` function using `?hist`

The first argument `x` of the `hist()` function is a vector of values that has to be sorted into bins. The arguments 'breaks' determines the bins. This argument `break` can be one of the following types:

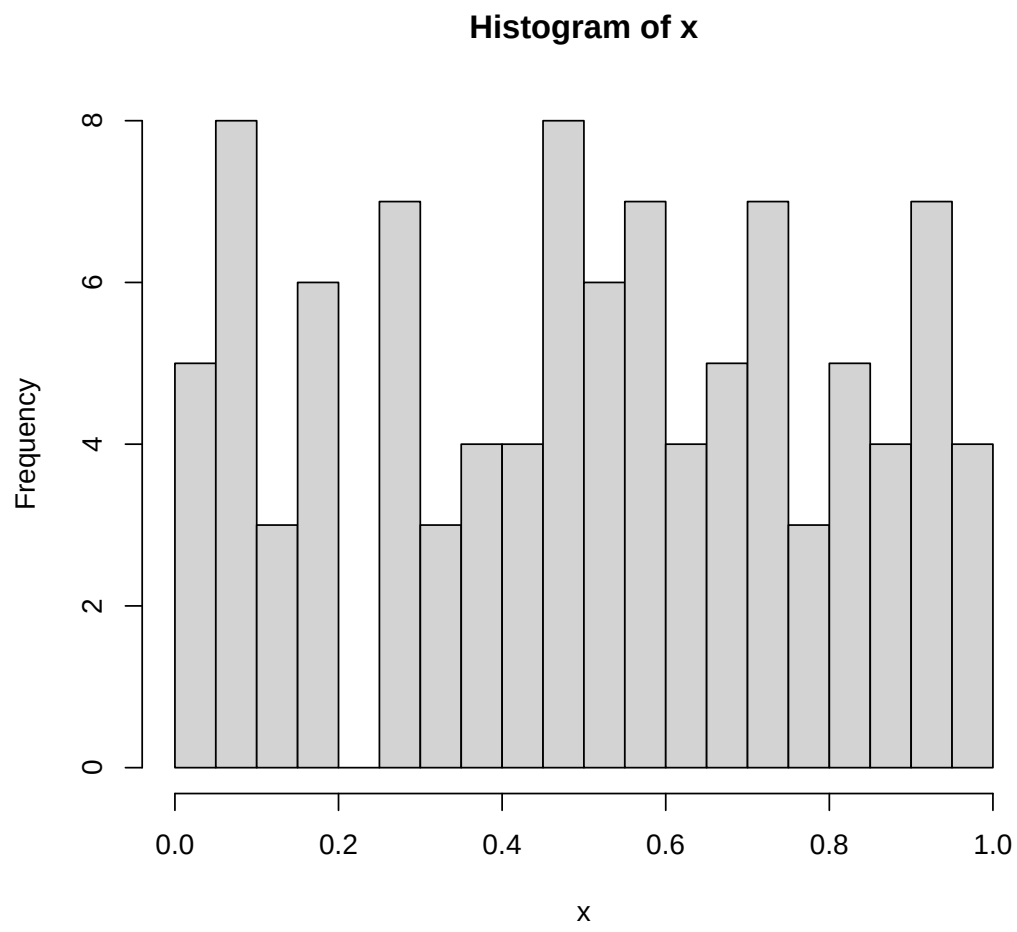
- a single number giving the number of cells for the histogram,
- a vector giving the breakpoints between histogram cells,

(More types of arguments are allowed for `breaks` but they are not so relevant for our purposes. See the help page `?hist` if you are interested).

The argument `freq` should be specified either `TRUE` or `FALSE`. If `freq` is `TRUE` the *number* of values in each bin is plotted, otherwise the *fraction* of all values in each bin is plotted. The default value of `freq` is `TRUE`.

The argument `right` should also be specified either `TRUE` or `FALSE`. If `right` is `TRUE` then a value a value in `x` that is exactly equal to right-most boundary of a bin is included in the bin (the histogram cells are right-closed (left open) intervals). The default value of `freq` is `TRUE`.

```
x = runif(100)
hist(x, breaks = 20, right = TRUE)
```



8.7.4 ggplot2()

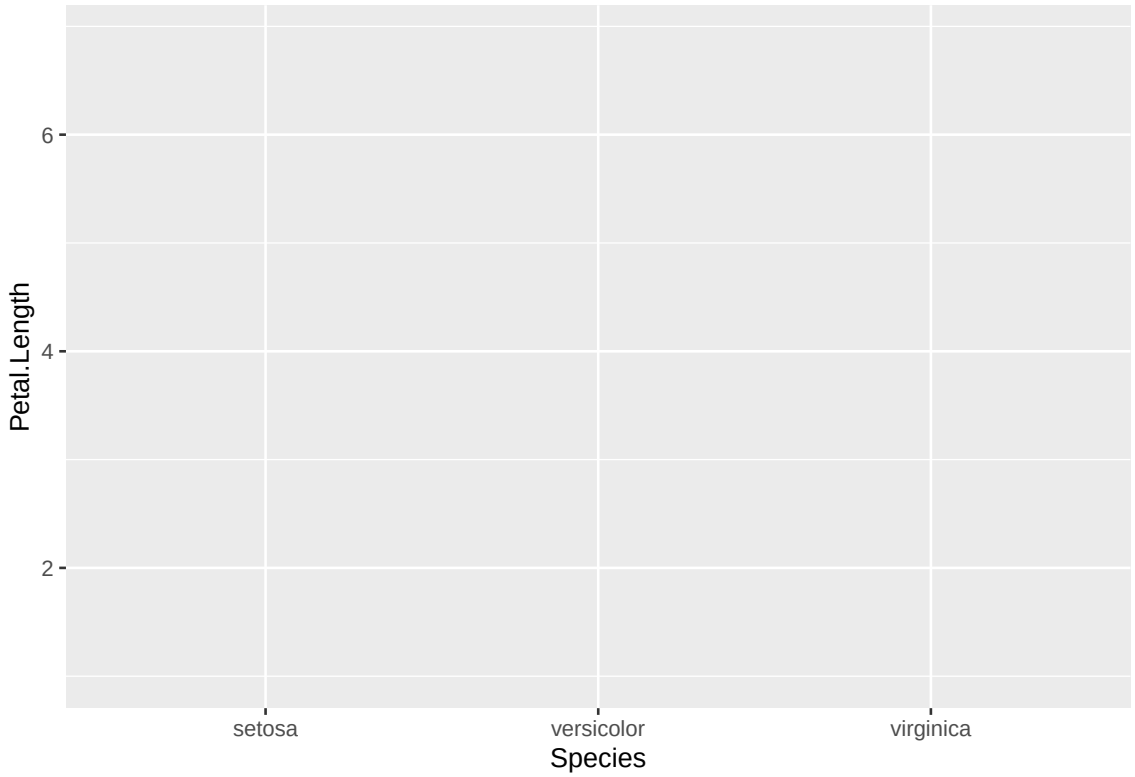
For more complex kinds of data, for example when you want to compare the data from different groups, it is easier to use `ggplot()` when plotting your data. `ggplot()` has many options, which we will not discuss here, but we will give you a short introduction with some examples.

The main difference between ggplot and basic plotting is that ggplot works with data frames and not individual vectors. When using ggplot, you first have to specify which data frame you're plotting. When your data frame consists of more than two columns (which is usually the case), you have to specify which columns to plot. This can be done the following way:

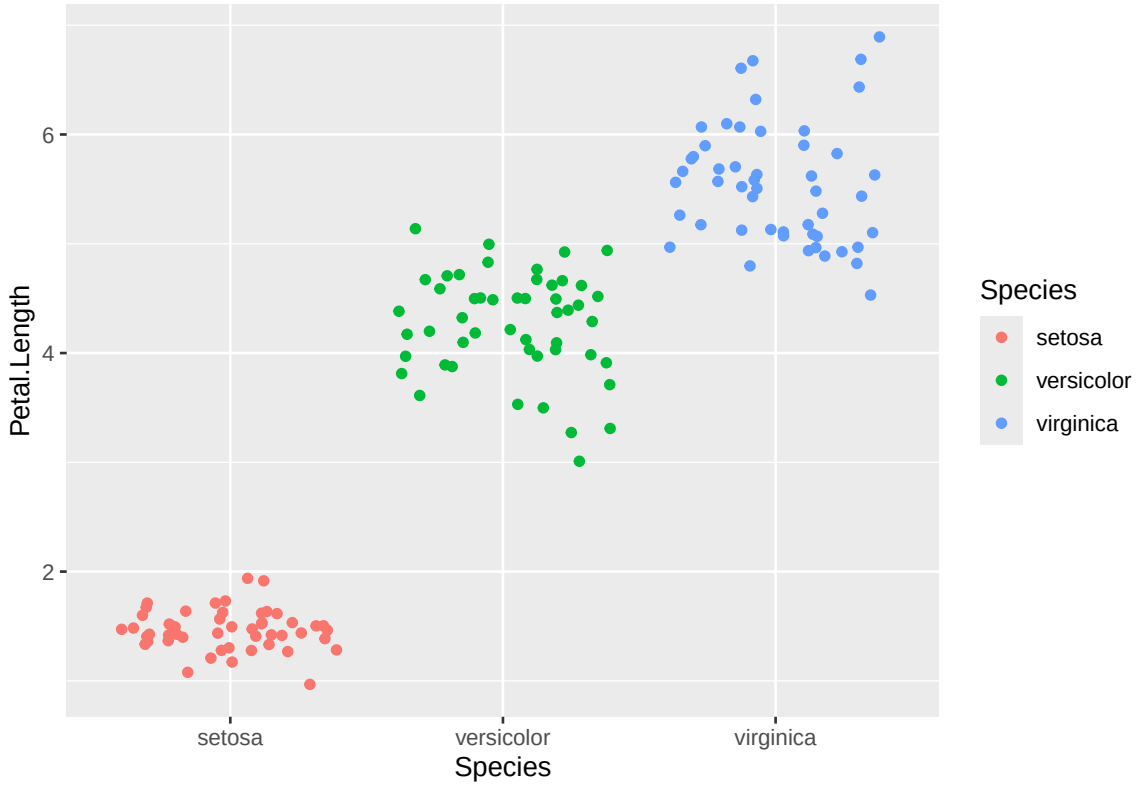
```
# Import the built-in data frame 'iris' and check its content
data("iris")
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

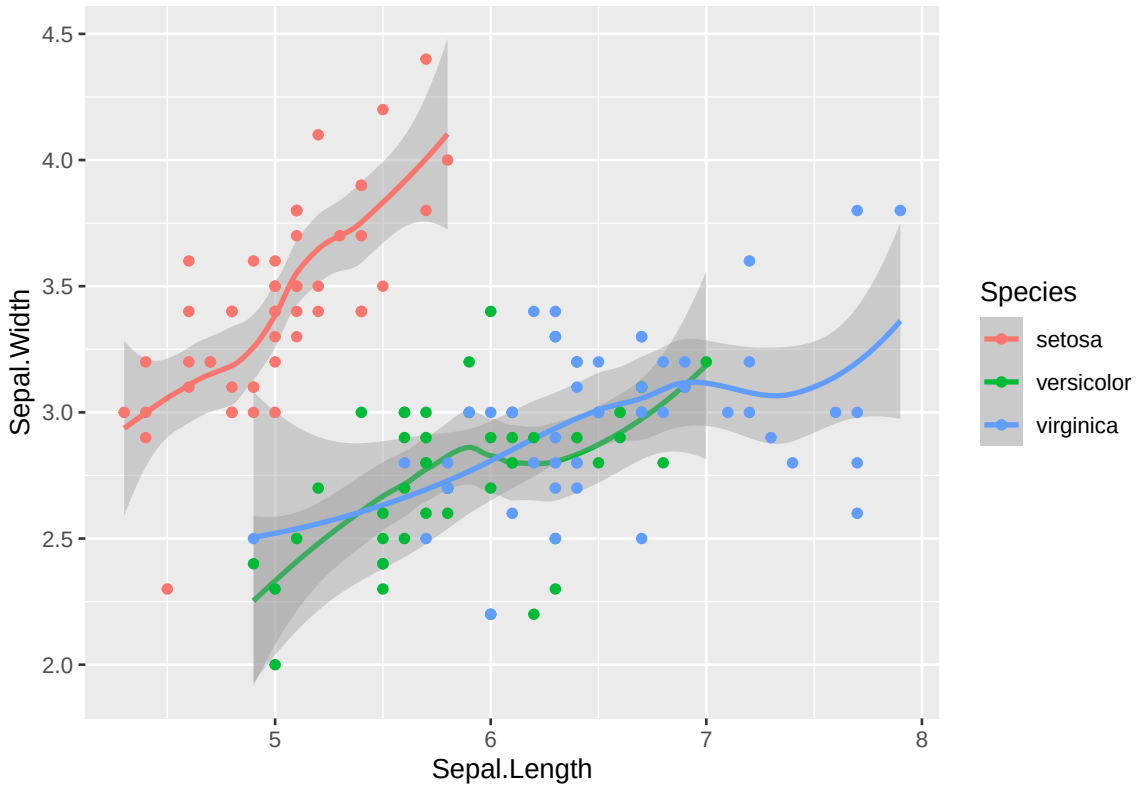
```
# Initiate plot (this does not show any data plotted, ggplot wants that specified)
ggplot(iris, aes(x=Species, y=Petal.Length))
```



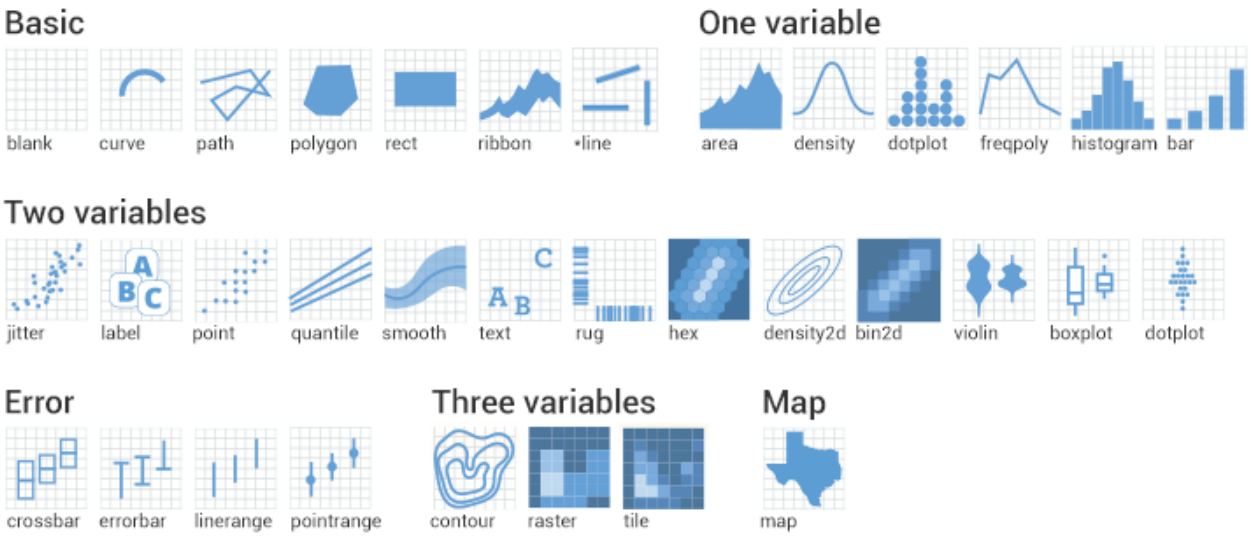
```
# To specify the type of plot, you need to use 'geom_...'
ggplot(iris, aes(x=Species, y=Petal.Length)) + geom_jitter(aes(color=Species))
```




```
# Using ggplot you can easily add new graphs to the plot
f = ggplot(iris)
g = f + geom_smooth(aes(x=Sepal.Length, y=Sepal.Width, color=Species))
g + geom_point(aes(x=Sepal.Length, y=Sepal.Width, color=Species))
```



In the `aes()` argument, you can specify your preferred X- and Y-axis, colour, size, shape and other aesthetic options. If you want to have the color, size etc fixed you need to specify it outside the `aes()`. Using `geom` you specify the type of plot you want. There are many options, which are shown in the figure below.



8.8 Exercises

8.8.1 Exercise 1

a. Use the `sample()` function to simulate an experiment in which you throw a six-sided dice 20 times.

b. Use the `hist()` function to plot a histogram of the results. How do the results compare with your expectation of the outcome of throwing a dice?

c. Repeat the above steps for throwing the dice 100 and 1000 times.

8.8.2 Exercise 2

a. Create a vector of the values 1 to 10, with steps of 0.5.

b. Using a for-loop randomly draw 20 "grades" from that vector and store them in a new vector. *Tip*: Use sampling with replacement!

c. For every value in the newly created vector, check if it is higher than or equal to 5.5, and store the result as "PASS" in a newly created vector. If the value is below 5.5, store "FAIL" to that vector.

d. Count the number of people that passed the course (tip: check out the "length"- and "which"-function).

8.8.3 Exercise 3

a. Import the built-in 'diamonds' data set and check its content (which is only available after loading the **ggplot2** package).

b. How many variables does the data set contain?

c. Create a scatter plot of the carat versus the price, where you also check the influence of the clarity.

d. Change the opacity of the points using the 'alpha'-function and add a smoothing trend to the plot.

e. Using ggplot, create a histogram of the carats where you check the influence of the cut.

f. Change the x limit to only see the carats between 0 and 2. Change the opacity of the bars as well.

8.8.4 Exercise 4

a. Create a vector containing 20 random numbers.

b. Use a repeat loop to print 10 sampled numbers from that vector. The loop should stop after 10 iterations.

Good luck with the assignments, and remember: when you don't know how to do something, google can help a lot! You should think about what you want to do specifically, and most of the time you can find the way to do it on Stack Overflow or another forum.

8.9 Sources used (and to check out if you want more information)

1. Datacamp (https://www.datacamp.com/tracks/data-scientist-with-r)
2. Codeacademy (https://www.codecademy.com/learn/paths/analyze-data-with-r)
3. Datamentor (https://www.datamentor.io/r-programming/)
4. YaRrr! The Pirates Guide to R (https://bookdown.org/ndphillips/YaRrr/)
5. Dataquest (https://www.dataquest.io/path/data-analyst-r/)
6. R-Statistics (http://r-statistics.co/)
7. Ggplot2 (https://ggplot2.tidyverse.org/reference/)
8. Techvidvan (https://techvidvan.com/tutorials/r-tutorial/)
9. DataScience Made Simple (https://www.datasciencemadesimple.com/r-tutorial-2/)
10. RaukR (https://nbisweden.github.io/RaukR-2019/ggplot/presentation/ggplot_presentation.html#1)

9 Introduction to plotting in R

For plotting data in R you have two main options. You can either use the plotting functions that come installed with base R (base R plotting), or use a speciallized plotting library for plotting called "ggplot", for which you need to install a package (ggplot2) in R. In general base R is easier to use for very simple plots, but as the plots get more complicated (for example you have multiple independent variables), using ggplot can save you some time. Feel free to use whichever tool you want throughout the class.

9.1 Introduction to Base R plotting

Base R plotting functions work directly with R's data structures, such as vectors and data frames, allowing for quick and easy creation of basic plots. Base R plotting is particularly useful for those who are new to R or those who need to create plots quickly without the overhead of learning additional packages. It serves as a solid foundation for data visualization.

9.2 Introduction to ggplot2

ggplot2 is a powerful and versatile plotting system in R, known for its ability to produce aesthetically pleasing and complex graphics with relatively simple commands. Developed by Hadley Wickham, it's based on the grammar of graphics—a coherent system for describing and building graphs. Here are some key concepts and features of ggplot2:

Layered Approach: ggplot2 uses a layered approach to build plots. You start by defining the data and mappings between variables to aesthetics (like color, size, and shape), and then you add layers of graphical objects (like points, lines, and bars).

Aesthetic Mappings: In ggplot2, you map data to visual properties (aesthetics) of geometric objects (geoms). For example, you might map the 'height' variable to the y-axis and 'gender' to the color aesthetic.

Geometric Objects (Geoms): These are the actual shapes or objects that are plotted. Common geoms include geom_point (for scatter plots), geom_line (for line plots), and geom_bar (for bar plots).

Statistical Transformations (Stats): These are used to transform data before it's plotted. For example, stat_summary can be used to plot summaries (like means) directly.

Scales: Scales map values in the data space to values in an aesthetic space, whether it be color, size, or shape. They also provide tools for formatting tick marks, labels, and legends.

Coordinate Systems: By default, ggplot2 uses Cartesian coordinates, but you can also use polar coordinates, or set up a map projection.

Faceting: Faceting allows you to create multiple plots based on a categorical variable. It's a powerful tool for breaking down complex data into manageable chunks.

Themes: ggplot2 allows extensive customization of plots through themes, where you can modify text, labels, legends, and other plot elements.

Here's a basic structure for a ggplot2 command:

```
ggplot(data = <DATA>, aes(x = <X_VARIABLE>, y = <Y_VARIABLE>)) + <GEOM_FUNCTION>(additional
parameters) + ... (other layers, scales, and theme adjustments)
```

9.3 Plotting continuous X and Y Data

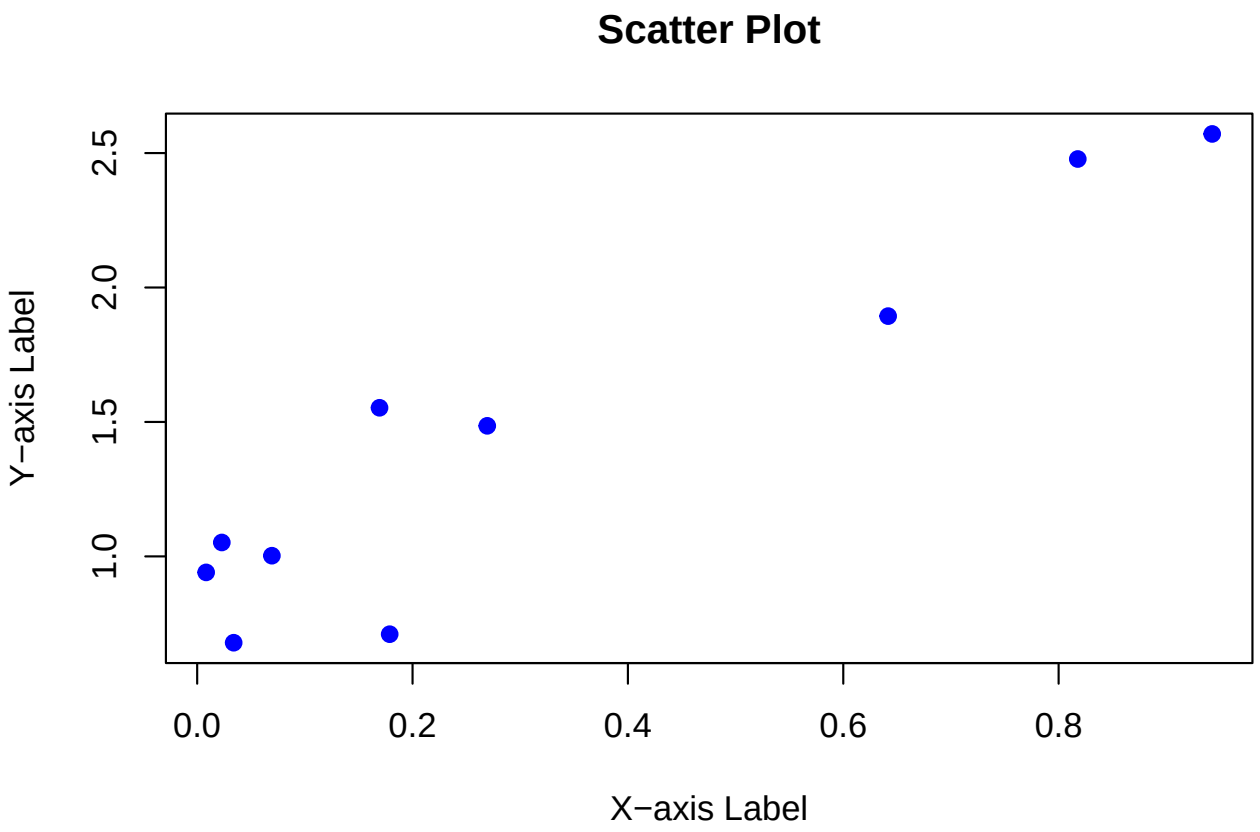
```r
# Sample Data
set.seed(12) # for reproducible data
x_data <- runif(10)
y_data <- 1 + 2 * x_data + rnorm(10,0,.5)

data = data.frame(x_data,y_data)
```

9.3.1 With Base R

Base R provides several functions for plotting continuous data, allowing you to create a variety of plots such as scatter plots, line plots, and more. Unlike ggplot2, base R graphics are generally less verbose and offer a straightforward approach for quick and simple plots. Here's an overview of plotting continuous X and Y data using base R:

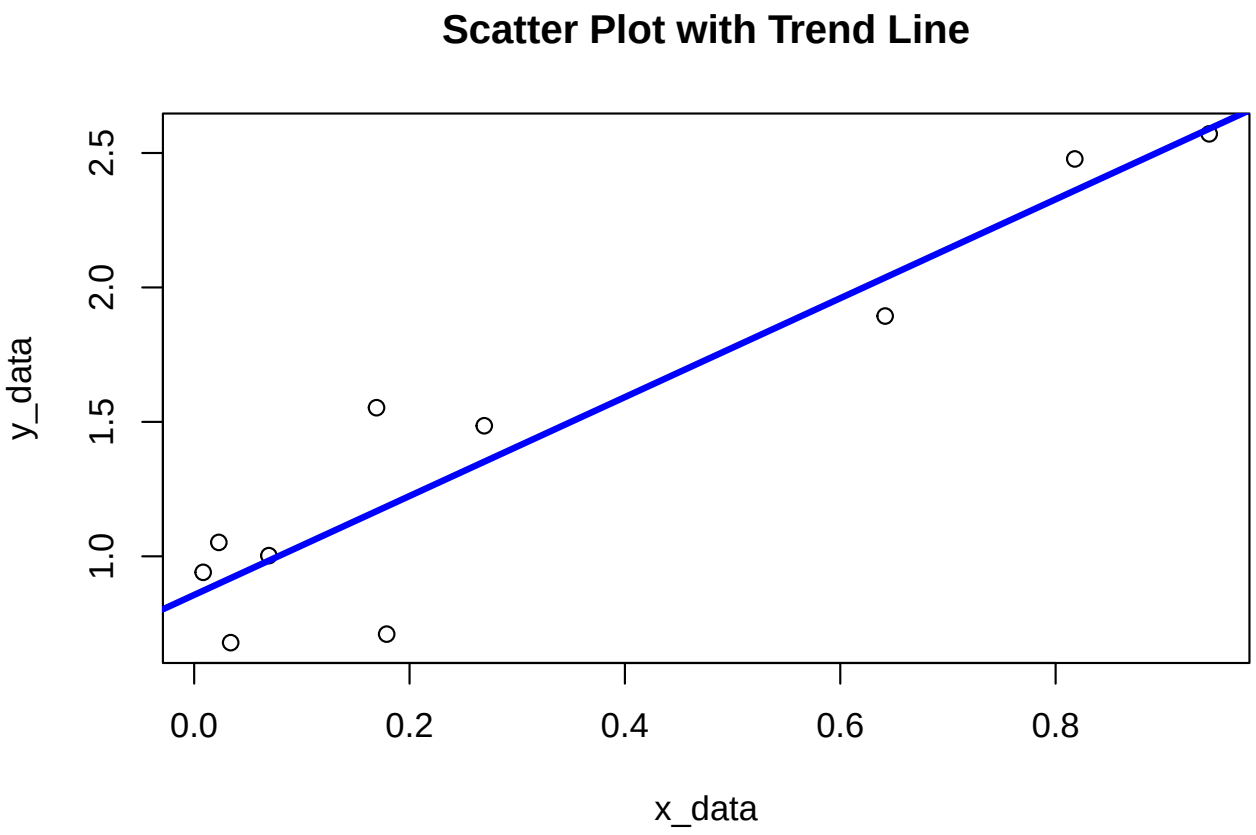
Scatter Plots: The plot() function is the most basic and versatile plotting function in base R. It creates a scatter plot by default, which is ideal for visualizing the relationship between two continuous variables.

```
# Scatter plot
plot(x_data, y_data,
     main = "Scatter Plot",
     xlab = "X-axis Label",
     ylab = "Y-axis Label",
     pch = 19, col = "blue")
```



Adding Trend Lines: You can add trend lines to your scatter plots using functions like abline() (for linear models), lines() (for non-linear models), or smooth.spline() (for smoothed lines).

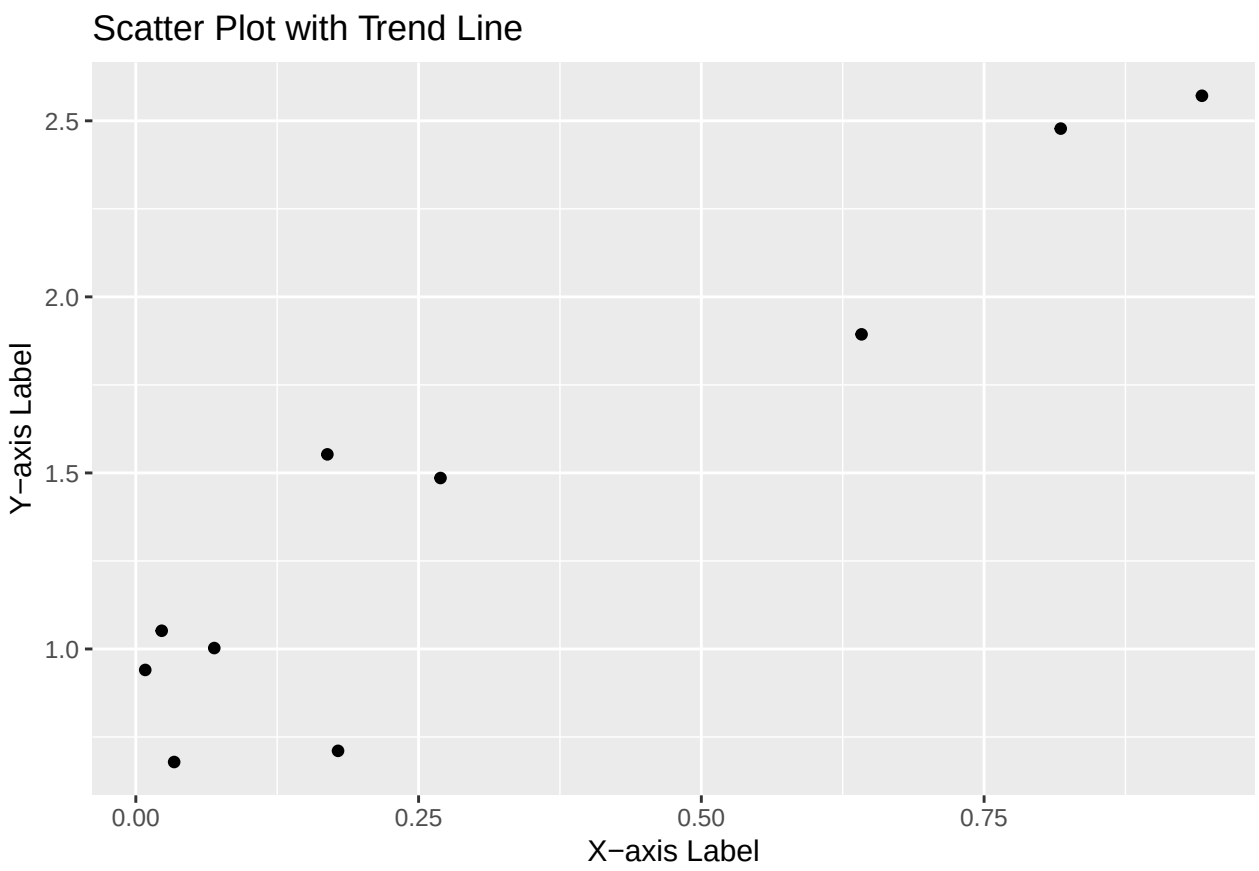
```
# Scatter plot with a linear trend line
plot(x_data, y_data, main = "Scatter Plot with Trend Line")
model <- lm(y_data ~ x_data)
abline(model, col = "blue", lwd=3)
```



9.3.2 With ggplot2

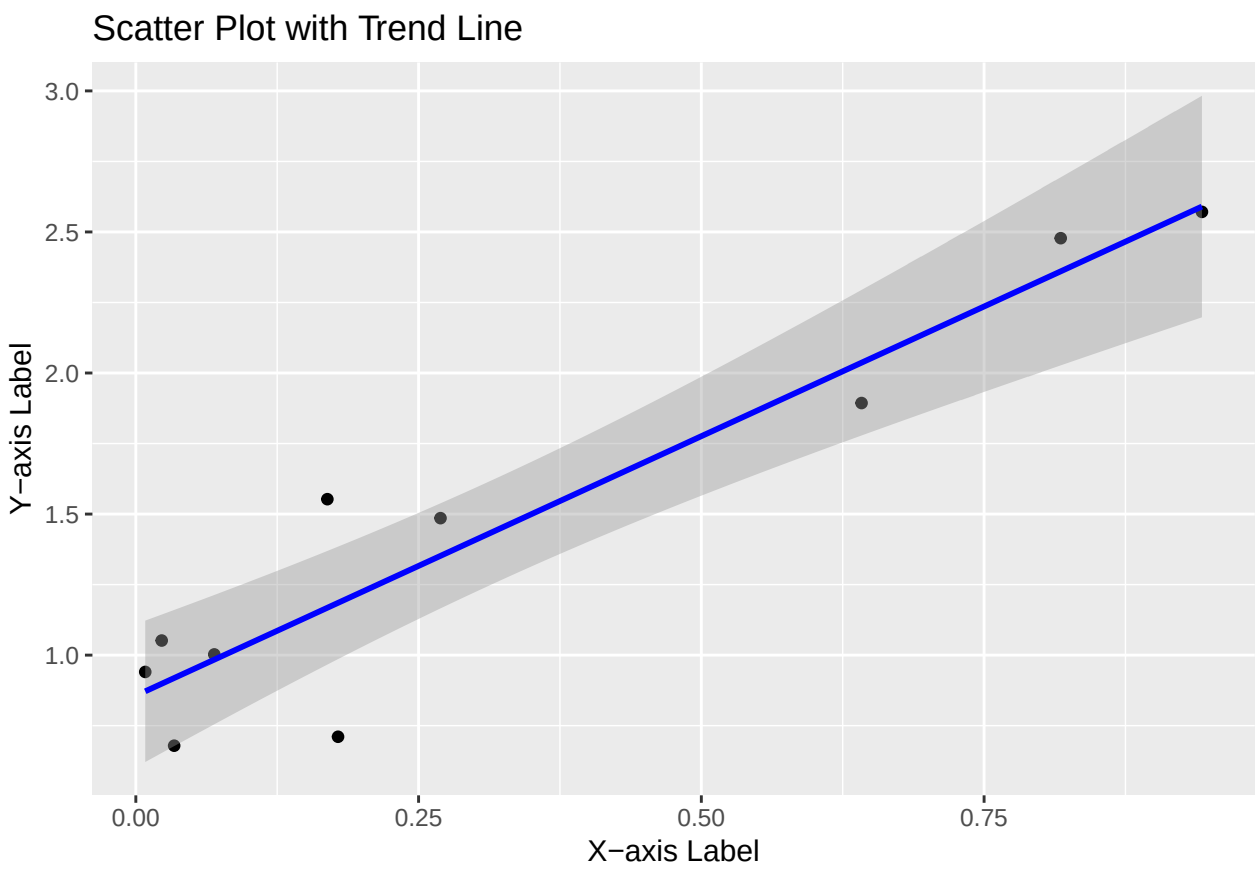
We can plot the relationship between two continous variables in ggplot as follows:

```
library(ggplot2)
# Scatter plot in ggplot2
ggplot(data, aes(x = x_data, y = y_data)) +
  geom_point() +
  labs(title = "Scatter Plot with Trend Line",
       x = "X-axis Label",
       y = "Y-axis Label")
```



We can also plot a line to represent the linear model that relates x to y:

```
# Scatter plot with a trend line in ggplot2
ggplot(data, aes(x = x_data, y = y_data)) +
  geom_point() +
  geom_smooth(method = "lm", color = "blue") + # 'lm' fits linear model to model y as function of x
  labs(title = "Scatter Plot with Trend Line",
       x = "X-axis Label",
       y = "Y-axis Label")
```



9.4 Plotting categorical X and continuous Y

This tutorial demonstrates how to plot data with categorical X-axis and continuous Y-axis using both base R and ggplot2.

9.4.1 Sample data

We'll create a sample dataset for demonstration purposes.

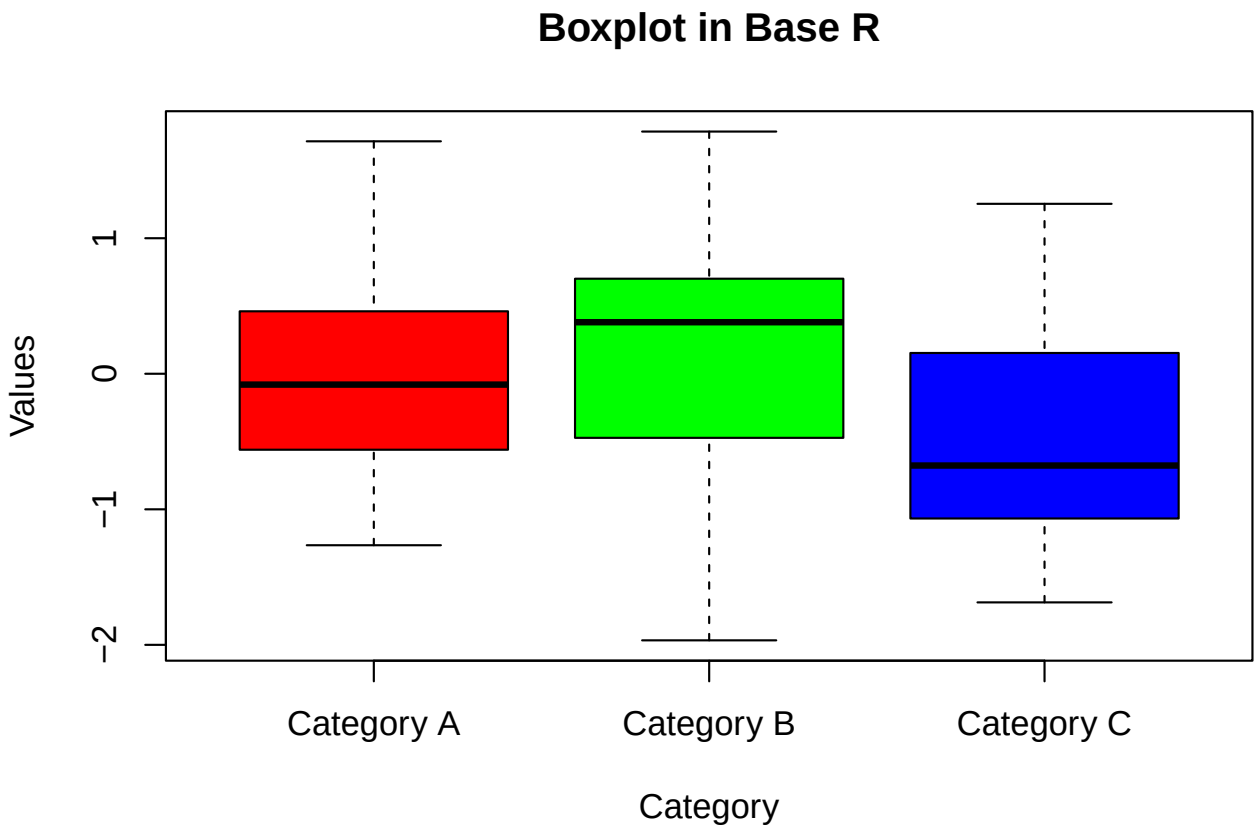
```
# Sample Data
set.seed(123) # for reproducible data
categories <- c("Category A", "Category B", "Category C")
values <- rnorm(30)
category <- factor(rep(categories, each = 10))

data <- data.frame(category, values)
```

9.4.2 With Base R

Using base R, we can create a boxplot, which is a common way to visualize data distribution across categories.

```
# Base R plot
boxplot(values ~ category, data = data,
        main = "Boxplot in Base R",
        xlab = "Category",
        ylab = "Values",
```

```
        col = rainbow(length(categories)))
```

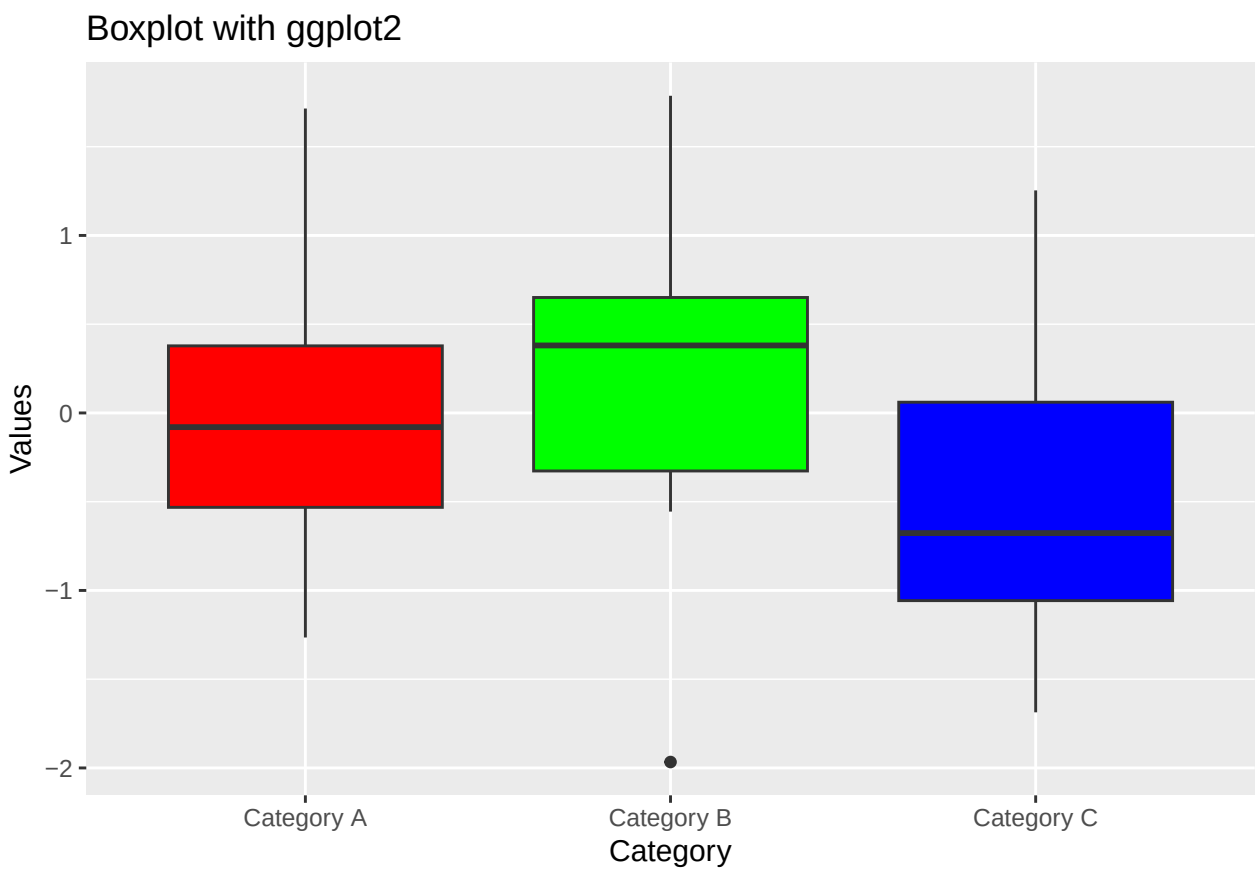


9.4.3 With ggplot2

Below I show I couple options for plotting data with categorical independent variables.

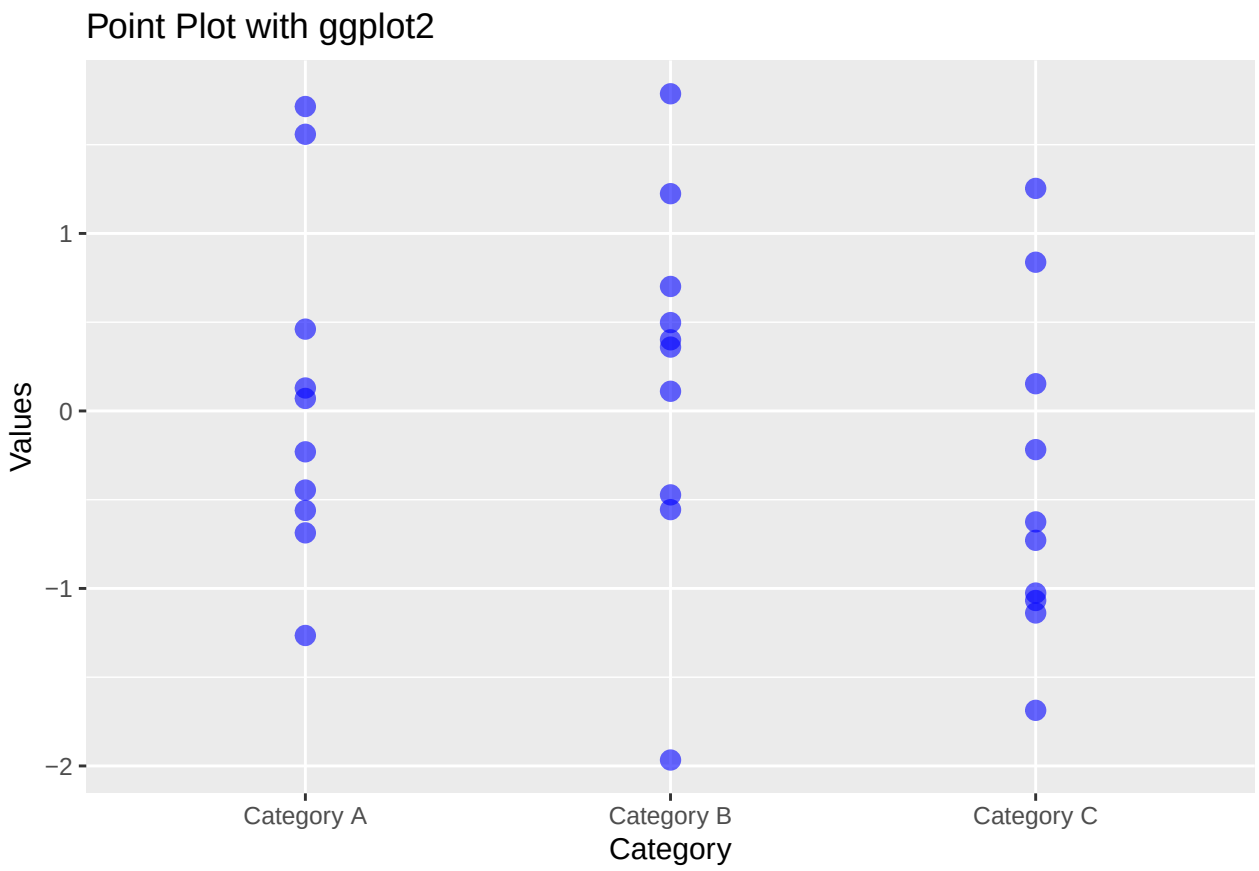
One option is to use a box plot, like in Base R:

```
# Loading the ggplot2 package
library(ggplot2)

# ggplot2 plot
ggplot(data, aes(x = category, y = values)) +
  geom_boxplot(fill = rainbow(length(categories))) +
  labs(title = "Boxplot with ggplot2",
       x = "Category",
       y = "Values")
```



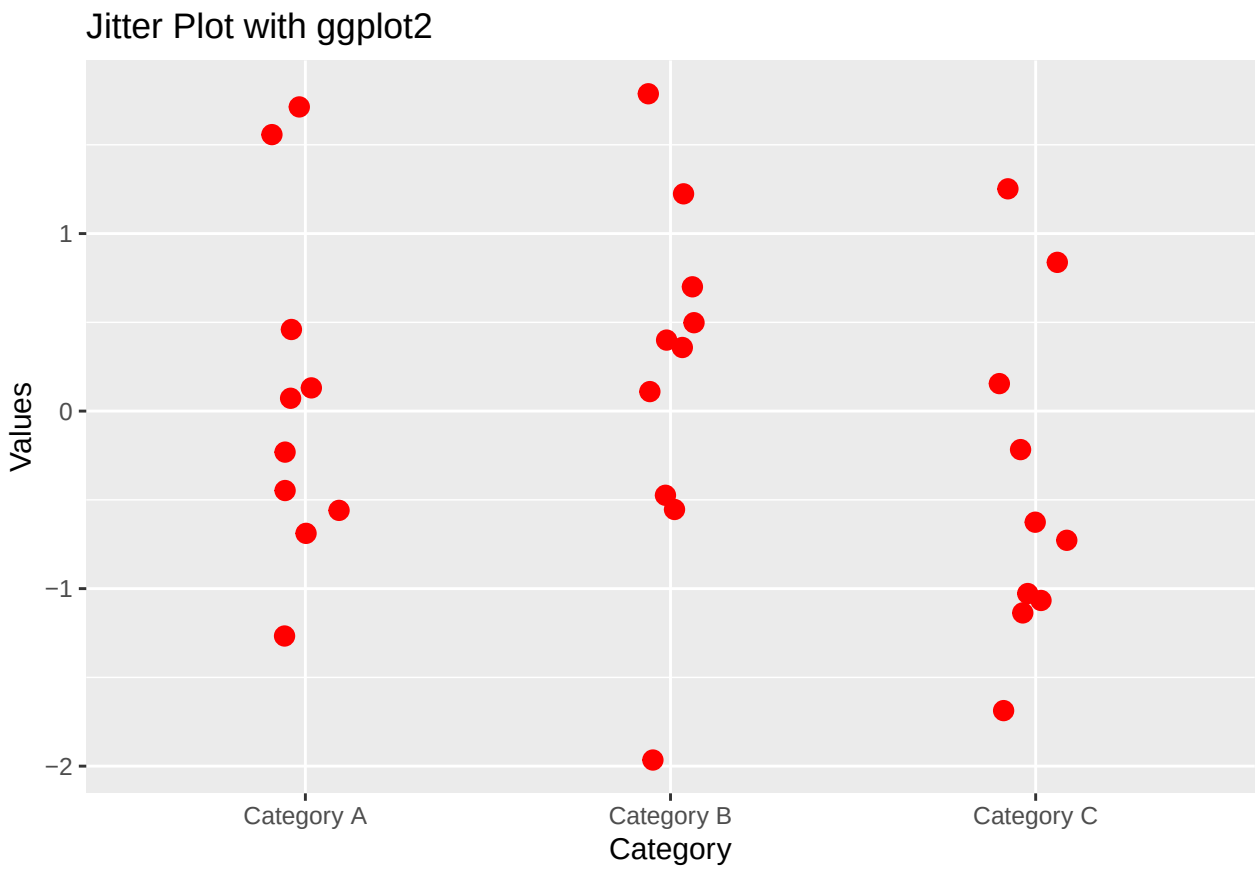
In addition to boxplots, ggplot2 allows for the plotting of individual data points. This can provide a more detailed view of the data distribution.

```
# ggplot2 plot for points
ggplot(data, aes(x = category, y = values)) +
  geom_point(color = "blue", size = 3, alpha = 0.6) +
  labs(title = "Point Plot with ggplot2",
       x = "Category",
       y = "Values")
```



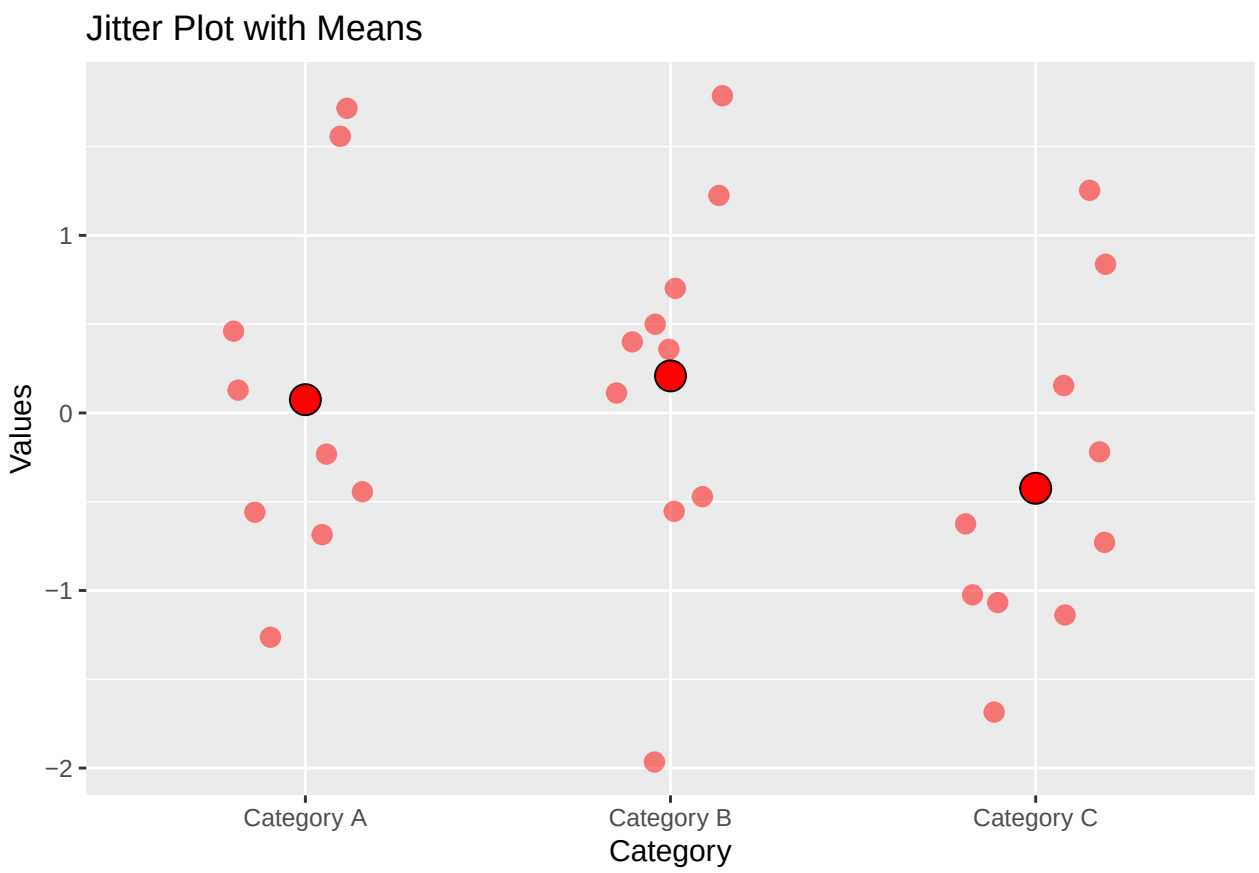
When there are multiple data points for each category, using jitter (adding a small amount of noise to the points) can help avoid overplotting and make the plot more informative.

```
# ggplot2 plot with jitter
ggplot(data, aes(x = category, y = values)) +
  geom_jitter(color = "red", size = 3, width = 0.1) +
  labs(title = "Jitter Plot with ggplot2",
       x = "Category",
       y = "Values")
```



We can also add additional graphic layers, for exaple plot the means for each group in addition to the individual data points:

```
# Calculate means
means <- aggregate(values ~ category, data = data, mean)

# ggplot2 plot with jitter and mean
ggplot(data, aes(x = category, y = values)) +
  geom_jitter(color = "red", size = 3, width = 0.2, alpha=.5) +
  geom_point(data = means, aes(x = category, y = values),
             fill = "red", size = 5,shape =21) +
  labs(title = "Jitter Plot with Means",
       x = "Category",
       y = "Values")
```



9.5 Data with a catagorical and continuous independent variable

Suppose you have a dataset with variables x_data, y_data, and a factor group which has two levels (e.g., "Group 1" and "Group 2").

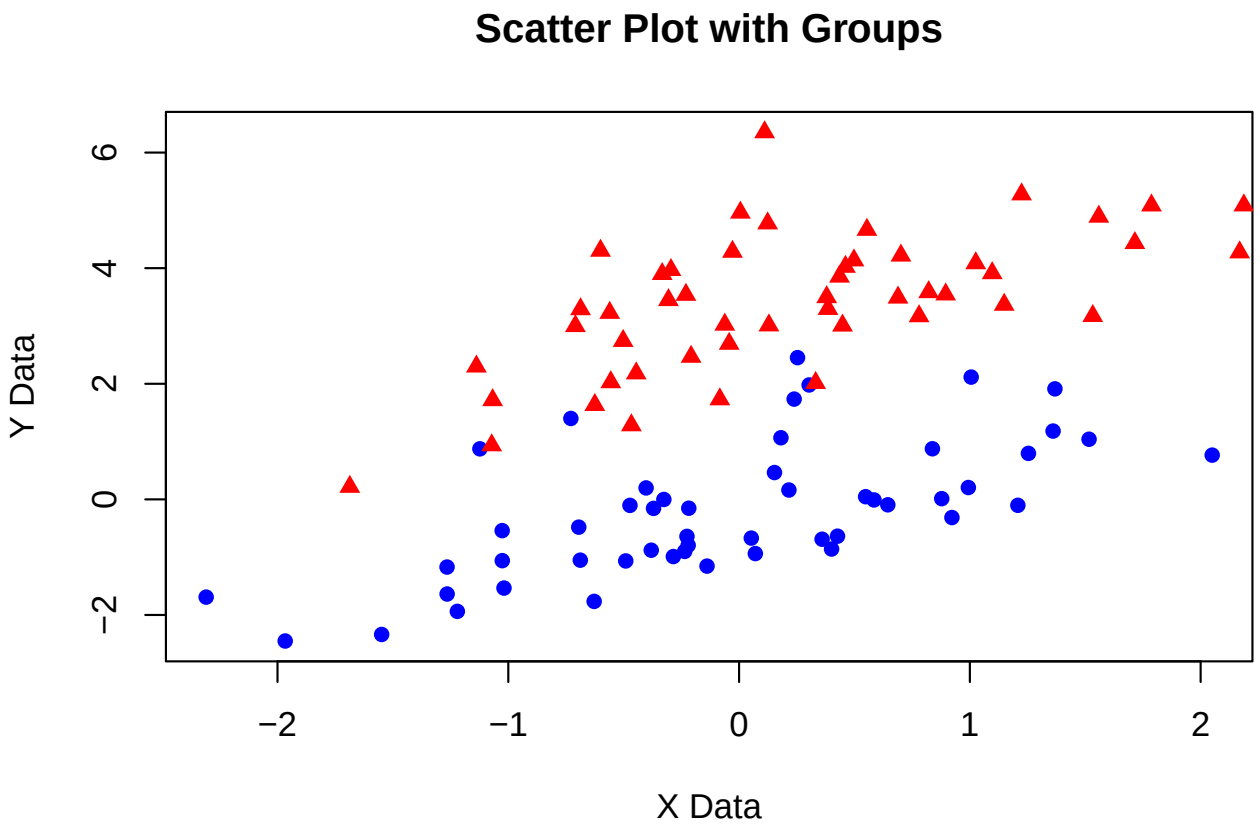
```
# Example dataset
set.seed(123) # For reproducible results
x_data <- rnorm(100)
group <- sample(c(0, 1), 100, replace = TRUE)

y_data <- 3 * group + 1 * x_data + rnorm(100)

data <- data.frame(x_data, y_data, group)
```

9.5.1 With Base R

In base R, you can use the plot() function and then add the points for the second group using points() or lines() for a different visual representation:

```
# Base R plot with groups
y_limits = c(min(y_data), max(y_data))

plot(y_data ~ x_data, data = data[data$group == 0,],
     col = "blue", pch = 16,
     main = "Scatter Plot with Groups",
     ylim = y_limits,
```

```
     xlab = "X Data", ylab = "Y Data")

points(y_data ~ x_data, data = data[data$group == 1,],
       col = "red", pch = 17)
```



In this code:

pch specifies the type of symbol used in the plot (e.g., circles, triangles).

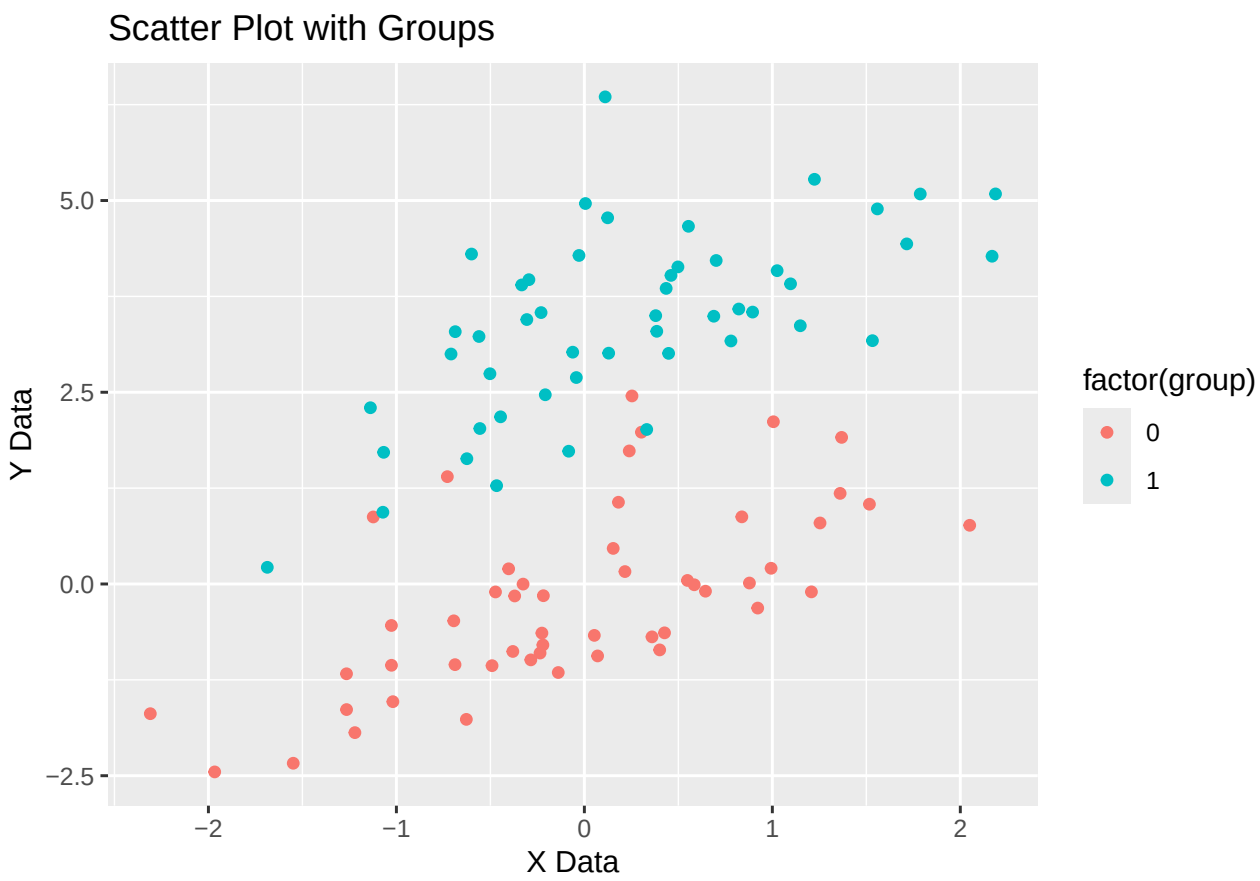
col changes the color of the points.

The firstplot()call plots data for "Group 1", and thenpoints()‘ adds data for “Group 2”.

9.5.2 With ggplot2

Using ggplot2, this becomes more straightforward, as ggplot2 automatically handles the grouping when you map the group variable to a visual property like color or shape:

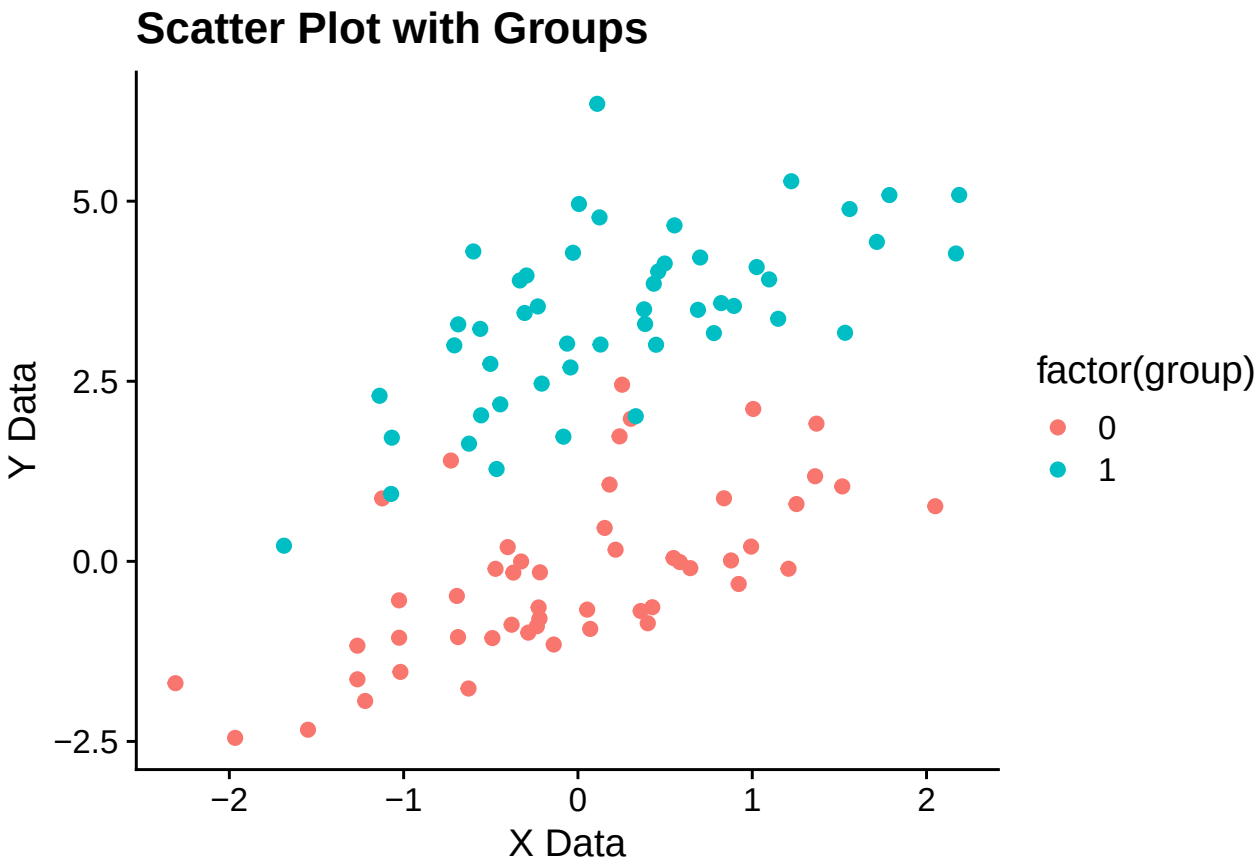
```
# ggplot2 plot with groups
library(ggplot2)

ggplot(data, aes(x = x_data, y = y_data, color = factor(group))) +
  geom_point() +
  labs(title = "Scatter Plot with Groups",
       x = "X Data", y = "Y Data")
```



9.6 Using different themes in ggplot2

If you don't like the grey background in default ggplot plots, you can experiment with different themes to create plots with different settings. I like to use theme_cowplot(). To use it you have to install the package (if you don't have it) and then import it.

```
# if you haven't installed cowplot yet, run the line of code below:
#install.packages("cowplot")

library(cowplot)

ggplot(data, aes(x = x_data, y = y_data, color = factor(group))) +
  geom_point() +
  labs(title = "Scatter Plot with Groups",
       x = "X Data", y = "Y Data") +
  theme_cowplot()
```


Scatter Plot with Groups
Y Data
X Data
5.0
2.5
0.0
−2.5
−2
−1
0
1
2
factor(group)
0
1

References

Chowdhury, S., Castro, S., Coker, C., Hinchliffe, T. E., Arpaia, N., and Danino, T. (2019). Programmable bacteria induce durable tumor regression and systemic antitumor immunity. *Nature Medicine*, 25(7):1057–1063.