Elementary Statistical Modeling for Applied Biostatistics

Jeffrey A. Walker

2018-09-10

Contents

Chapter 1

Statistical Modeling

More cynically, one could also well ask "Why has medicine not adopted frequentist inference, even though everyone presents P-values and hypothesis tests?" My answer is: Because frequentist inference, like Bayesian inference, is not taught. Instead everyone gets taught a misleading pseudo-frequentism: a set of rituals and misinterpretations caricaturing frequentist inference, leading to all kinds of misunderstandings. – Sander Greenland

We use statistics to learn from data with uncertainty. Traditional introductory textbooks in biostatistics implicitly or explicitly train students and researchers to "discover by p-value" using hypothesis tests (appendix xxx). Over the course of many chapters, the student is trained to use something like a dichotomous key to choose the correct "test" for the data at hand, compute a test statistic for their data, compute a p-value based on the test statistic, and compares the p-value to 0.05. Textbooks typically give very little guidance about what can be concluded if $p < 0.05$ or if $p > 0.05$, but many researchers conclude (incorrectly) they have "discovered" something if $p < 0.05$ but found "no effect" if $p > 0.05$.

Researchers learn almost nothing useful from a hypothesis test. If we are investigating the effects of an increasingly acidified ocean on coral growth, $p = 0.002$ may be evidence that pH affects growth, but, from everything we know about pH and cell biology, it would be absurd to conclude from any data that ocean acidification does not affect growth. Instead, we want to know the magnitude of the effect and our uncertainty in estimating this magnitude. We can use this magnitude and uncertainty to make predictions about the future of coral reefs, under different scenarios of ocean acidification. We can use the estimated effects and uncertainty to model the consquences of the effects of acidification on coral growth on fish production or carbon cycling.

The "discovery by p-value" strategy, or Null-Hypothesis Significance Testing (NHST), has been criticized by statisticians for many, many decades. Nevertheless, introductory biostatistics textbooks written by both biologists and statisticians continue to organize textbooks around a collection of hypothesis tests, with little emphasis on estimation and uncertainty.

1.1 Statistical modeling with linear models

This book is an introduction to the analysis of biological data using a statistical modeling approach. As an introduction, the focus will be linear models and extensions of the linear models including linear mixed models and generalized linear models. Here, I refer to all of these as "linear models" because all are a function of a linear predictor. Linear models are the engine behind many hypothesis tests but the emphasis in statistical modeling is estimation and uncertainty instead of test statistics and p-values. A modeling view of statistics is also more coherent than a dichotomous key strategy.

All students are familiar with the idea of a linear model from learning the equation of a line, which is

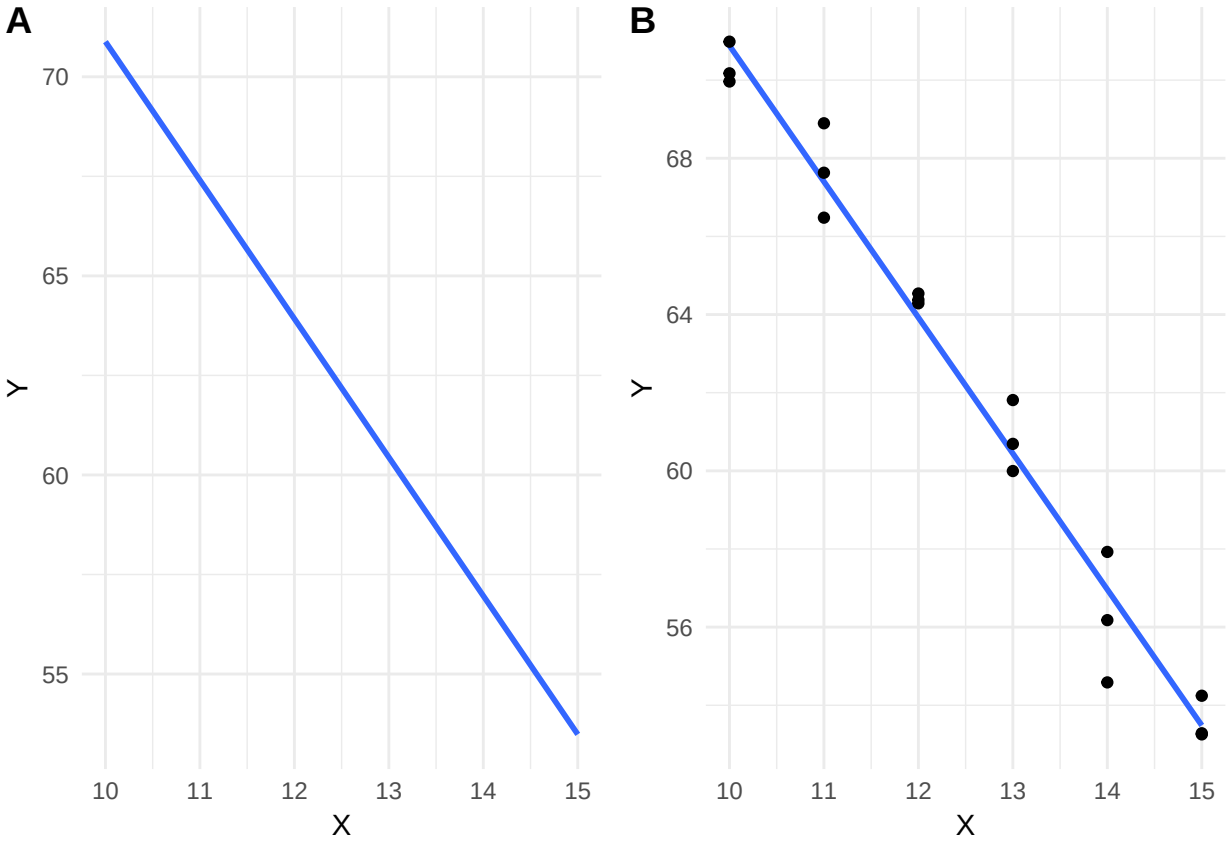


Figure 1.1: A line vs. a linear model. (A) the line $y=-3.48X + 105.7 is drawn. (B) A linear model fit to the data. The model coefficients are numerically equal to the slope and intercept of the line in A.

$$Y = mX + b \tag{1.1}$$

where m is the slope of the line and b is the Y-intercept. It is useful to think of equation (1.1) as a function that maps values of X to values of Y. Using this function, if we input some value of X, we always get the same value of Y as the output.

A linear model is a function, like that in equation (1.1), that is fit to a set of data, often to model a process that generated the data or something like the data. The line in Figure 1.1A is just that, a line, but the line in Figure 1.1B is a model of the data in Figure 1.1B. The basic structure of a linear model is

$$Y = \beta_0 + \beta_1 X + \varepsilon \tag{1.2}$$

A linear model has two parts: the "prediction" ($Y = \beta_0 + \beta_1 X$) and the "error" (ε). The prediction part looks like the equation for a line except that I've used β_0 for the intercept and β_1 for the slope and I've put the intercept term first. This re-labeling and re-arrangement make the notation for a linear model more flexible for more complicated linear models. For example $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$ is a model where Y is a function of two X variables.

As with the equation for a line, the prediction part of a linear model is a function that maps a value of X to a specific value of Y. This mapped value is the **expected value** given a specific input value of X. This is often written as $\mathrm{E}[Y|X]$. The error part of a linear model is a random variable that adds some random value to this expected value. Nothing about the model part of a linear model can predict its value.

The inputs to a linear model (the X variables) have many names including "independent variables," "predictor variables,", "explanatory variables," "treatment variables," and "covariates". The output of a linear model (the Y variable or variables if the model is multivariate) is the "dependent variable," "response," or "outcome." The β in the linear model are model **parameters** There can be additional parameters in more sophisticated models. The coefficients of the X in a linear model (β_1 in model (1.2)) are often called "the effects" (so β_1 is the effect of X_1).

Although a linear model is a model of a data-generating process, linear models are not typically used to actually generate any data. Instead, when we use a linear model to understand something about a real dataset, we think of our data as one realization of a process that generates data like ours. A linear model is a model of that process. That said, it is incredibly useful to use linear models to create fake datasets for at least two reasons: to probe our understanding of statistical modeling generally and, more specifically, to check that a model actually creates data like that in the real dataset that we are analyzing.

1.1.1 Linear models are used for prediction, explanation, and description

Researchers typically use linear models to understand relationships between one or more Y variables and one or more X variables. These relationships include

1. Descriptive modeling. Sometimes a researcher merely wants to describe the relationship between Y and a set of X variables, perhaps to discover patterns. For example, the arrival of a spring migrant bird (Y) as a function of sex (X_1) and age (X_2) might show that males and younger individuals arrive earlier. Importantly, if another X variable is added to the model (or one dropped), the coefficients, and therefore, the precise description, will change. That is, the interpretation of a coefficient as a descriptor is *conditional* on the other covariates (X variables) in the model. In a descriptive model, there is no implication of causal effects and the goal is not prediction. Nevertheless, it is very hard for humans to discuss a descriptive model without using causal language, which probably means that it is hard for us to think of these models as *mere description*. Like natural history, descriptive models are useful as patterns in want of an explanation, using more explicit causal models including experiments.

2. Predictive modeling. Predictive modeling is very common in applied research. For example, fisheries researchers might model the relationship between population density and habitat variables to predict

which subset of ponds in a region are most suitable for brook trout (*Salvelinus fontinalis*) reintroduction. The goal is to build a model with minimal prediction error, which is the error between predicted and actual values for a future sample. In predictive modeling, the X ("predictor") variables are largely instrumental – how these are related to Y is not a goal of the modeling, although sometimes an investigator may be interested in the relative importance among the X for predicting Y (for example, collecting the data may be time consuming, or expensive, or enviromentally destructive, so know which subset of X are most important for predicting Y is a useful strategy).

3. Explanatory (causal) modeling. Very often, researchers are explicitly interested in *how* the X variables are causally related to Y. The fisheries researchers that want to reintroduce trout may want to develop and manage a set of ponds to maintain healthy trout populations. This active management requires intervention to change habitat traits in a direction, and with a magnitude, to cause the desired response. This model is predictive – a specific change in X predicts a specific response in Y – because the coefficients of the model provide knowledge on how the system functions – how changes in the inputs *cause* change in the output. Causal interpretation of model coefficients requires a set of strong assumptions about the X variables in the model. These assumptions are typically met in **experimental designs** but not **observational designs**.

With observational designs, biologists are often not very explicit about which of these is the goal of the modeling and use a combination of descriptive, predictive, and causal language to describe and discuss results. Many papers read as if the researchers intend explanatory inference but because of norms within the biology community, mask this intention with "predictive" language. Here, I advocate embracing explicit, explanatory modeling by being very transparent about the model's goal and assumptions.

1.2 Model fitting

In order to use a linear model to describe, predict, or explain, we need to fit a model to data in order to estimate the parameters. If we fit model (1.3) to some data, the estimated parameters are the coefficients (b_0 and b_1) of the fit model

$$\mathrm{E}[Y|X] = b_0 + b_1 X \tag{1.3}$$

The left-hand side of equation (1.3) is the **conditional expectation** and is read as "the expectation of Y given X" or "the expected value of Y given X". Throughout this book, I use the greek β to refer to a theoretical, data-generating parameter and the roman "b" to refer its estimate.

The goal of descriptive and explanatory modeling is the estimate of the coefficients of the X variables and their uncertainty. The goal of predictive modeling is the estimate of predicted values, and their uncertainty, given specific values of X. These predicted values are the conditional expectations.

For the model fit to the data in Figure 1.1B, the coefficient of X is the slope of the line. Perhaps surprisingly, we can fit a model like equation (1.2) to data in which the X variable is categorical. A simple example is the experiment of antioxidants (vitamins C and E) on lifespan in Voles (Fig. 1.2). In this experiment, the X variable is categorical, with three **levels**: "Control", "Vitamin_E" and "Vitamin_C". Categorical X variables are often called **factors**. The trick to using a linear model with categorical X is to recode the factor levels into numbers – how this is done is explained in Chapter xxx. When the X variable is categorical, the coefficients of the X are *differences in group means*. The linear model fit to the vole data has two coefficients, one for Vitamin E and one for vitamin C. The estimate and uncertainty of the these two coefficients are shown in the top part of Figure 1.2. The bottom part shows the raw data, as well as the group (factor level) means and the uncertainty in the estimate of these means.

The simplest possible model that can be fit to the data is

$$\mathrm{E}[Y] = b_0 \tag{1.4}$$

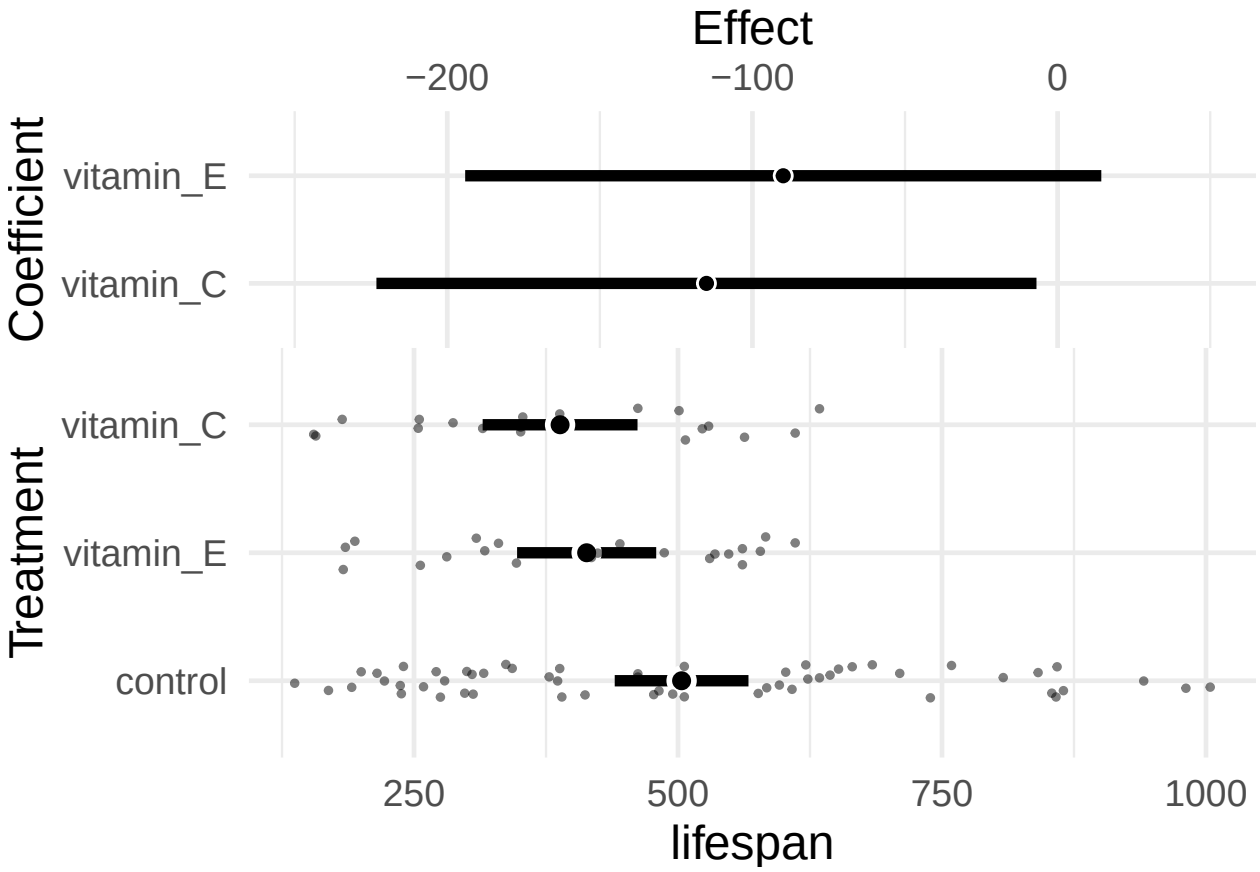


Figure 1.2: HarrellPlot of vole data.

which is simply the mean of Y, or, more specifically, the **unconditional mean** of Y, since its value is not conditional on any value of X.

1.2.1 "Statistical model" not "regression model"

Statistical modeling terminology can be confusing. The X variables in a statistical model may be quantitative (continuous or integers) or categorical (names or qualitative amounts) or some mix of the two. Linear models with all quantitative independent variables are often called "regression models." Linear models with all categorical independent variables are often called "ANOVA models." Linear models with a mix of quantitative and categorical variables are often called "ANCOVA models" if the focus is on one of the categorical X or "regression models" if there tend to be many independent variables. Other patterns occur. For example "ANCOVA models" often include interaction effects but "regression models" rarely do. To avoid thinking of statistical analysis as "regression vs. ANOVA", I will most often use the term "statistical model" for general usage, and use a more specific term only to emphasize something about the model in that particluar context.

1.3 Multilevel models

1.4 Linear models versus non-linear models

In this text, I use "linear model" for any model that is linear in the parameters, which means that the different components of the model are added together. Or, using the language of matrix algebra, the predictor is a simple dot product of the model matrix and the coefficients. For example, a cubic polynomial model

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \varepsilon \tag{1.5}$$

is a linear model, even though the function is non-linear, because the different components are added (or, using matrix algebra, the predictor is $\mathbf{X}\boldsymbol{\beta}$).

A generalized linear model (GLM) has the form $g(\mu_i) = \eta_i$ where η (the greek letter eta) is the linear predictor, which is linear in the parameters.

$$\eta = \mathbf{X}\boldsymbol{\beta} \tag{1.6}$$

Many sources do not consider a GLM to be a "linear model" but an "extension" of a linear model. Regardless, a GLM is linear in the parameters and here, I include GLMs under the "linear model" umbrella.

Non-linear models, in conrast to a GLM or classical linear model, are not linear in the parameters (the predictor is not a simple dot product of the model matrix and a vector of parameters). For example, the Michaelis-Menten model is a nonlinear model

$$Y = \frac{\beta_1 X}{\beta_2 + X} + \varepsilon \tag{1.7}$$

Appendix 1: Organization – R Projects and R Notebooks

1.5 Importing Packages

The R scripts you write will include functions in packages that are not included in Base R. These packages need to be downloaded from an internet server to your computer. You only need to do this once. But, each time you start a new R session, you will need to load a package using the `library()` function. Now is a good time to import packages that we will use

1. Open R Studio and choose the menu item "Tools" > "Install Packages"
2. In the "packages" input box, insert "ggplot2, data.table, emmeans, lme4, reshape2". Make sure that "install dependencies" is clicked before you click "Install"

Again, once these are installed, you don't need to do this again. You simply need to use the `library()` function at the start of a script.

1.6 Create an R Studio Project for this Class

1. Create a folder named "BIO_413"
2. Within this folder, create new folders named
    1. "notebooks" – this is where your R notebooks are stored
    2. "R" – this is where R scripts are stored
    3. "data" – this is where data that we download from public archives are stored
    4. "output" – this is where you will store fake data generated in this class
    5. "images" – this is where image files are stored
3. Open R Studio and click the menu item File > New Project…
4. Choose "Existing Directory" and navigate to your BIO_413 folder
5. Choose "Create Project"
6. Check that a file named "BIO_413.Rproj" is in your BIO_413 folder

1.7 R Notebooks

A typical statistical modeling project will consist of:

1. reading data from Excel or text (.csv or .txt) files
2. cleaning data
3. analysis
4. generating plots
5. generating tables

6. writing text to describe the project, the methods, the analysis, and the interpretation of the results (plots and tables)

The best practice for reproducible research is to have all six of these steps in your R Notebook. Too many research projects are not reproducible because the data were cleaned in Excel, and then different parts of the data were separately imported into a GUI statistics software for analysis, and then output from the statistics software was transcribed to Excel to make a table. And other parts of the analysis are used to create a plot in some plotting software. And then the tables and plots are pasted into Microsoft Word to create a report. Any change at any step in this process will require the researcher to remember all the downstream parts that are dependent on the change and to re-do an analysis, or a table, or a plot, etc. etc.

The goal with an R Studio Notebook is to explicitly link all this so that changes in earlier steps automatically flow into the later steps. So, at the end of a project, a researcher can choose "run all" from the menu and the data are read, cleaned, analyzed, ploted, tabled, and put into a report with the text.

This means that you have to think of the organization of the R code that your write in a Notebook. Your cannot simply append new code to the end of a script if something earlier (or above) is dependent on it. You need to go back up and insert the new code at some earlier (and meaningful) point.

For example, an R chunk generates 100 random normal values and then plots these with a histogram. This was the chunk that I wrote

```
x <- rnorm(n)
qplot(x)
```

When I ran the chunk, I got the error "Error in rnorm(n) : object n not found". I was using the function `rnorm()` to generate values but I hadn't assigned any value to `n` yet, so I got the error. To get this to work properly, I could have just typed `n <- 100` in the console and then re-run the script but I want it to work properly on a fresh run of the chunk (after quitting and re-opening R Studio) so I instead inserted `n <- 100` at the start of the chunk, like this:

```
n <- 100
x <- rnorm(n)
qplot(x)
```

1.7.1 Create an R Notebook for this Chapter

1. The top-left icon in R Studio is a little plus sign within a green circle. Click this and choose "R Notebook" from the pull-down menu.
2. Change the title of the notebook to "Notebook_01-organization"
3. Delete the default R Markdown text starting with "This is an [R Markdown]…"

Now write some text documenting which packages you installed.

1.7.2 Create a "setup" chunk

1. Click on the "Insert" menu on the right hand side of the script (R Markdown) pane and choose "R". This will insert an R code chunk into your R markdown document.
2. The first R chunk of a notebook should be a setup chunk. Name the chunk "setup"
3. load the libraries ggplot2 and data.table and click the chunk's run button (the green triangle to the right of the chunk)

```
library(ggplot2)
library(data.table)
```

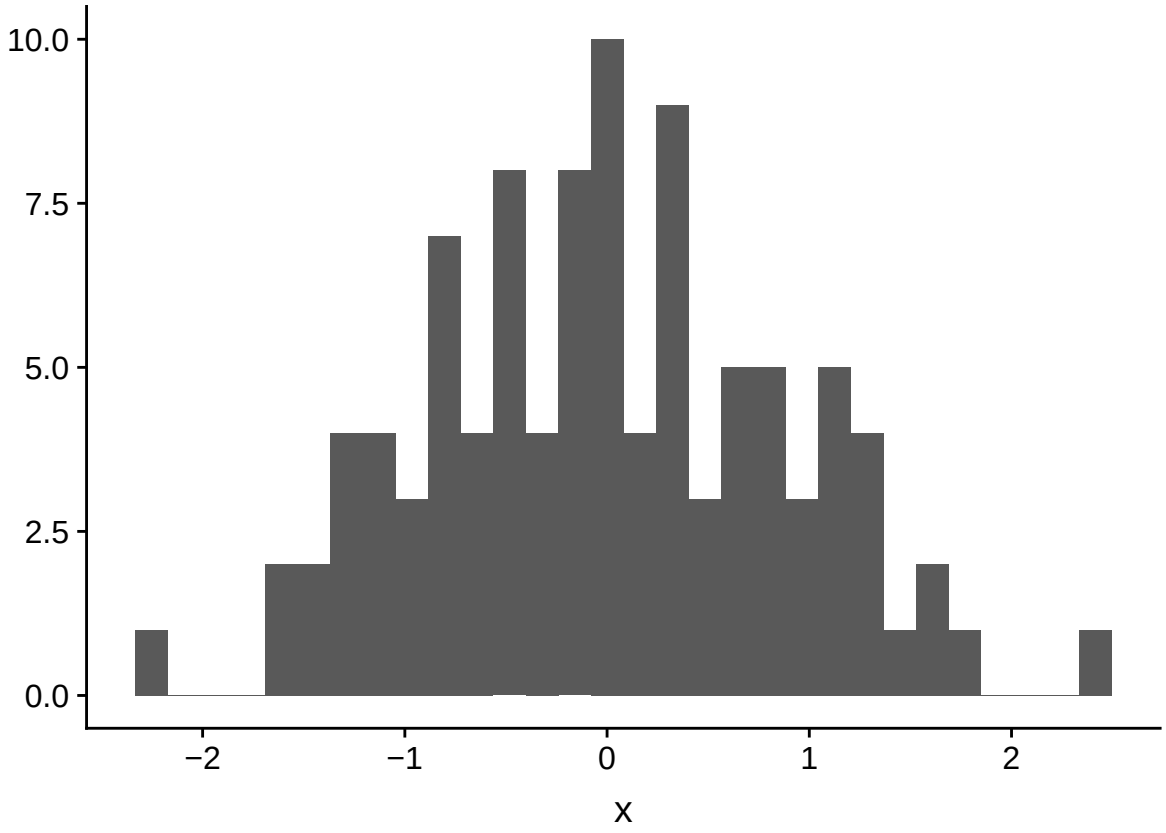
I added the chunk option "message=FALSE". Run your chunk with and without this as an option.

1.7.3 Create a "simple plot" chunk

4. Create a new chunk and label it "simple plot"
5. insert the following R script and then click the chunk's run button. Do you get a plot?

```
n <- 100
x <- rnorm(n)
qplot(x)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



1.7.4 Create more R chunks and explore options and play with R code

Appendix 2: Data – Reading, Writing, and Fake

1.8 Create new notebook for this chapter

Be sure to save the notebook in the "notebooks" folder of your BIO_413 project. Annotate your notebook with notes! Update it as you learn more! We will use data.table for importing text files in tab-delimited or comma-separated formats and the readxl package for importing excel files.

```
library(ggplot2)
library(ggpubr)
library(data.table)
library(readxl)
library(emmeans)
library(mvtnorm)

knitr::opts_chunk$set(fig.width=6, fig.height=4)
```

1.9 Importing Data

Throughout this book, we will download data from the Dryad Digital Repository, which is a major resource for increasing reproducibility in science. My own view is that *all data* should be archived on some public server (exceptions include data that are proprietary or contain sensitive information – such as human health measures).

The downloaded data will be inserted into the "data" folder. To access these data in an R script, the script needs to know "where to look" or the "address." This address is the **directory path**. The default path for an R notebook is the directory containing the notebook .Rmd file. This file should be in the "notebooks" folder within "BIO_413". The "BIO_413" Folder is the parent of the "notebooks" folder. It is also the parent of the "data" folder. To see any content within the "data" folder, the R script needs to tell R to move back (or up) the directory structure out of the "notebooks" folder into the parent "BIO_413" folder and then forward (or down) into the "data" folder. This is done with

```
data_path <- "../data"
```

The `..` moves the address (of where to read input or write output) back one step and `/data` moves the address forward into the "data" folder. This folder will eventually contains lots of data from Dryad Digital Repository.

1.9.1 Excel File

The Excel dataset is from an experiment on the growth response of zebra finch chicks to an incubation call that presumably signals "hot environment" to the embryos (Mariette, M.M. and Buchanan, K.L., 2016. Prenatal acoustic communication programs offspring for high posthatching temperatures in a songbird. Science, 353(6301), pp.812-814). The source file is from the Dryad Repository here:

file name: "allDatasetsMarietteBuchanan2016.xls"

source: https://datadryad.org//handle/10255/dryad.122315

Steps

1. Copy the title of the Dryad page, which is "Data from: Prenatal acoustic communication programs offspring for high post-hatching temperatures in a songbird"
2. Create a new folder within "data" and paste in the copied title as the folder name
3. Remove the colon from the name, so the folder name is "Data from Prenatal acoustic communication programs offspring for high post-hatching temperatures in a songbird"
4. Download the .xls file into this folder

A .xls file is an old (pre 2007) Microsoft Excel file type. It is a binary file and can only be opened into a readable format with specialized software. The more modern Excel file type is .xlsx, which contains within it multiple xml components. An xml file is a text file, and so contains readable content, but the content is xml code to display something. In general, I am a big advocate of archiving stuff as text files (manuscripts, data, scripts, blog posts) because these will *always* be readable by future software. Microsoft Excel is not likely to die anytime soon and software that can read .xls and especially .xlsx files (again, .xlsx files are text files) is even less likely to disappear but we can feel even more confident if data are archived as text files. That said, a single microsoft excel file with multiple sheets is an efficient method for distributing data and the readxl package provides excellent tools for reading different sheets of a single .xls or .xlsx file.

The code below uses the function `read_excel()` from the package readxl. More about the amazing power of this package is the tidyverse page and chapter 11 in the *R for Data Science* book.

```
data_folder <- "Data from Prenatal acoustic communication programs offspring for high post-hatching tem
filename <- "allDatasetsMarietteBuchanan2016.xls"
file_path <- paste(data_path, data_folder, filename, sep="/")
chick <- data.table(read_excel(file_path, sheet="nestlingMass"))
head(chick) # check -- are there headers? are there the correct number of columns?
```

```
##       chick ID brood ID brood composition sex rank in nest
## 1:    N1.10LF3  N1.10m3             mixed   F            2
## 2: N1.10noCut3  N1.10m3             mixed   M            4
## 3:    N1.10RB3  N1.10m3             mixed   F            2
## 4:    N1.10RF3  N1.10m3             mixed   F            5
## 5:    N1.12LB3  N1.12m3             mixed   F            3
## 6:    N1.12LF3  N1.12m3             mixed   F            1
##    playback treatment nest temperature above ambient
## 1:              treat                       4.289583
## 2:               cont                       4.289583
## 3:               cont                       4.289583
## 4:               cont                       4.289583
## 5:               cont                       3.972917
## 6:              treat                       3.972917
##    max daily temp hatch day mean max temp hatch to day2
## 1:                      17.4                    18.83333
## 2:                      19.0                    20.53333
## 3:                      17.4                    18.83333
## 4:                      19.0                    20.53333
```

```
## 5:                    29.0                    24.63333
## 6:                    25.1                    24.80000
##    mean max temp hatch to day10 mean max temp hatch to day13 hatching mass
## 1:                        22.70                     23.05714           0.7
## 2:                        24.53                     23.41429           0.6
## 3:                        22.70                     23.05714           0.7
## 4:                        24.53                     23.41429           0.6
## 5:                        22.85                     22.91429           0.7
## 6:                        23.35                     23.24286           0.6
##    day1 mass day2 mass day10 mass day13 mass day13 tarsus
## 1:       1.1       1.2         NA        9.8        14.11
## 2:       0.8       1.1         NA        9.1        12.90
## 3:       0.9       1.4         NA        9.3        13.60
## 4:       0.5       0.9         NA        7.7        13.06
## 5:         1       1.4        9.4       10.1        14.08
## 6:       0.9       1.4        8.1        9.6        13.46
```

NOTE

If you are getting errors when trying to read a file, it is probably a bug in the construction of the variable `file_path`, which is a string variable and the value has to be exactly match the directly path to the file you are trying to read. `file_path` is constructed by pasting together the variables `data_path`, `data_folder`, and `filename`. Type `file_path` into the console and look at the value. Then check

1. Spelling. Humans are very good at understanding misspelled words but the R language (or any computer language) is very literal. "../data" does not equal "./data" or "../ data" or "../data"
2. Capitalization. R is **case sensitive** (some programming languages are not). "../data" does not equal "../Data" or "../DATA".
3. is the file you are trying to read actually in the folder you are trying to read from?
4. is the notebook that you are writing in the folder "notebooks"? (the construction of `file_path` assumes that notebook is one folder deep within the project folder.

If the spelling or capitalization of any of these components is wrong, then `file_path` will be wrong. If there is any difference in any character in the string, then R will return an error. So spelling AND capitalization have to be perfect, not simply close. Humans are very good at understanding misspelled and OdDLy capitalized words but the R language (or any computer language) is very literal.

In this book, we will consistently uses the protocol for storing and retrieving downloaded files. The first three lines in the script above creates the directory path to the file. This path includes

1. data_path – the relative path into the folder "data" (relative to the location of the notebook file)
2. data_folder – the name of the folder within "data" containing the file
3. filename – the name of the file to read

These are all put together into a single path using the function `paste()`. Read about paste. It will be used repeatedly. The `read_excel(file_path, sheet="nestlingMass")` reads the nestlingMass sheet only. This function is embedded within the `data.table()` function and so is converted into a data.table. The data.table is assigned to the object "chick"

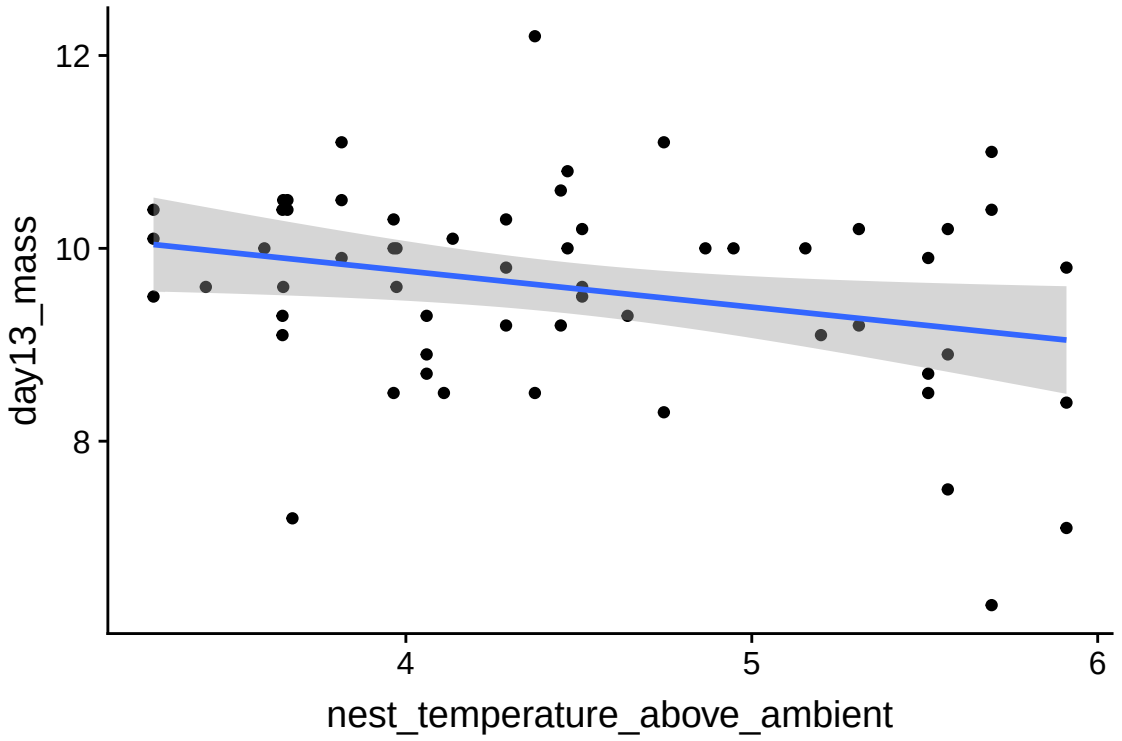
The `head(chick)` script simply displays the first few lines of the data.table. This is one way to check that the data were imported correctly. In this case, it is easy to see that the column names have spaces in them. It can sometimes be hard to work with column names with spaces and so this next line of code changes all spaces to an underscore

```
setnames(chick, old=colnames(chick), new=gsub(" ", "_", colnames(chick)))
```

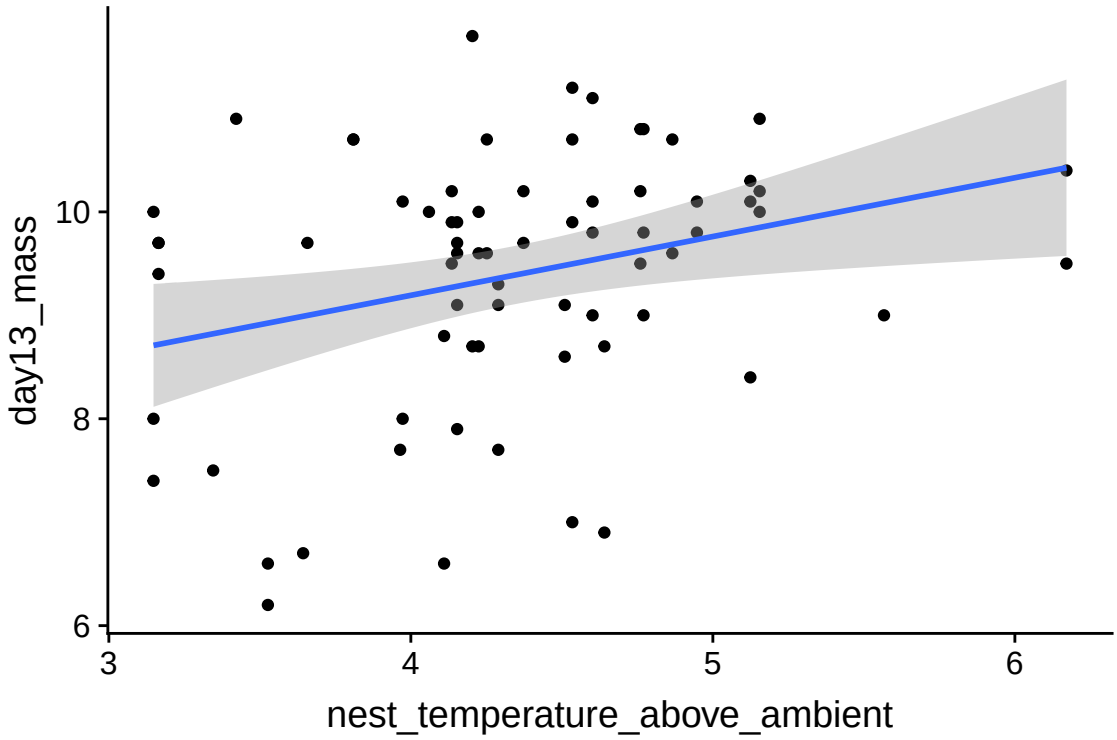
Resist the temptation to change the column names in the data file, which reduces reproducibility. Always increase reproducibility!

Just for fun, let's plot the data and reproduce Fig. 2A and B. We are using the `qplot` function, which is from the ggplot2 package. Two plots are made and only a subset of the rows are plotted in each (in A, the subset in which playback_treatment=="treat" and, in B, the subset in which playback_treatment=="cont"). This book uses the ggplot2 package extensively.

```
qplot(x=nest_temperature_above_ambient, y=day13_mass, data=chick[playback_treatment=="treat"]) +
  geom_smooth(method="lm")
```




```
qplot(x=nest_temperature_above_ambient, y=day13_mass, data=chick[playback_treatment=="cont"]) +
  geom_smooth(method="lm")
```



1.9.2 Text File

The example dataset comes from an experiment on the effect of neonicotinoid pesticides on bumble bee colony growth.

file name: "Whitehorn, O'Connor, Wackers, Goulson (2012) Data from 'Neonicotinoid pesticide reduces bumblebee colony growth and queen production'.csv.csv"

source: https://datadryad.org//resource/doi:10.5061/dryad.1805c973

Steps

1. Copy the title of the Dryad page, which is "Data from: Neonicotinoid pesticide reduces bumblebee colony growth and queen production"
2. Create a new folder within "data" and paste in the copied title as the folder name
3. Remove the colon from the name, so the folder name is "Data from Neonicotinoid pesticide reduces bumblebee colony growth and queen production"
4. Download the .csv file into this folder

A .csv file is a text file that is comma-delimted, which means that the entries of a row are separated by commas. A text file is readable by any text editor software and most other kinds of software. Datasets that are stored as text files are typically saved as either .csv (where the entries of a row are separated by commas) or .txt (where the entries are separated by tabs). The base R way to read a .csv file is using `read.csv`. The `read.table` function is more versatile, as the delimiter can be specified. The function `fread()` from the data.table package is fast, smart, and flexible. It is smart in the sense that it guesses what the delimter is. Unfortunately, because of spaces in the column labels for this file, fread guesses incorrectly (another reason why spaces in column labels should be avoided). To overcome this, the statement below specifies that the file contains a "header" (a line containing column labels)

```
data_folder <- "Data from Neonicotinoid pesticide reduces bumblebee colony growth and queen production"
filename <- "Whitehorn, O'Connor, Wackers, Goulson (2012) Data from 'Neonicotinoid pesticide reduces bu
file_path <- paste(data_path, data_folder, filename, sep="/")
```

```
bee <- fread(file_path, header=TRUE)
bee[, Treatment:=factor(Treatment, c("Control", "Low", "High"))]
head(bee)
```

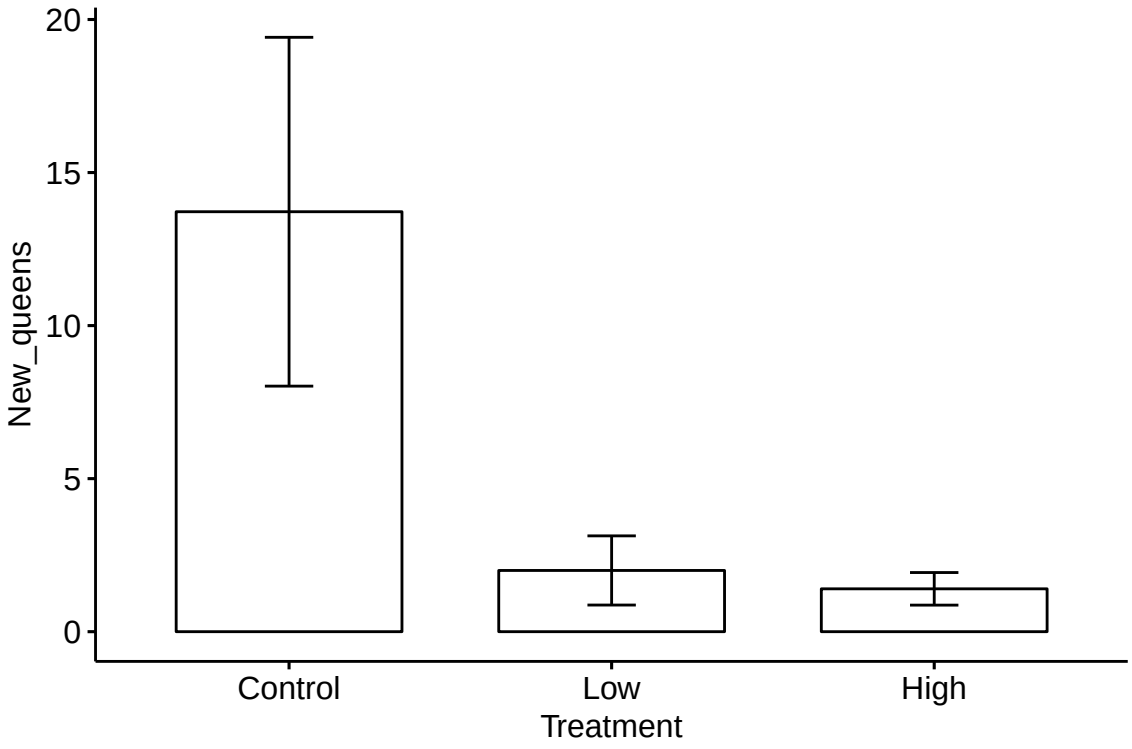
```
##    Treatment Nest ID No. workers      0      1      2   3   4    5    6
## 1:   Control      C1          13 712.95 748.30 800.57 865 966  997  850
## 2:   Control      C2          14 719.58 750.00 789.25 822 812  846  827
## 3:   Control      C3          17 704.92 736.31 767.99 837 976 1117 1050
## 4:   Control      C4          20 726.42 763.31 795.60 813 801  784   NA
## 5:   Control      C5          28 740.60 785.52 808.42 837 871  906  886
## 6:   Control      C6          15 727.10 751.90 774.80 807 847  859  827
##      7   8 V13 Workers left Males New queens Total unhatched pupae
## 1: 791 775  NA            2     0          1                    NA
## 2: 820 802  NA            6    15          0                    20
## 3: 866 808  NA            1     0          9                    NA
## 4:  NA  NA  NA            0     0          0                    12
## 5: 807 775  NA            3     0          0                    NA
## 6:  NA  NA  NA            0     0          0                   118
##    Queen pupae Empty cells
## 1:          NA          NA
## 2:           0         120
## 3:          NA          NA
## 4:           0          72
## 5:          NA          NA
## 6:          20         132
```

Here, as with the import of the Excel file, the first three lines create the directory path to the file. The treatment column is a factor variable containing three levels (Control, Low, and High). R automatically orders these alphabetically. For plotting and analysis, we might want a different order. For example, we want Control to be first in the order, since this is a natural "reference" level (what everything is compared to). And if we think of "Control" as no treatment, then it makes sense to have "Low" second in order and "Hight" last in order. The line `bee[, Treatment:=factor(Treatment, c("Control", "Low", "High"))]` re-orders these levels to this more meaningful order.

Again, there are spaces in the column names. **Here I'll leave it to you to change this**

Here is a reproduction of Fig 2.

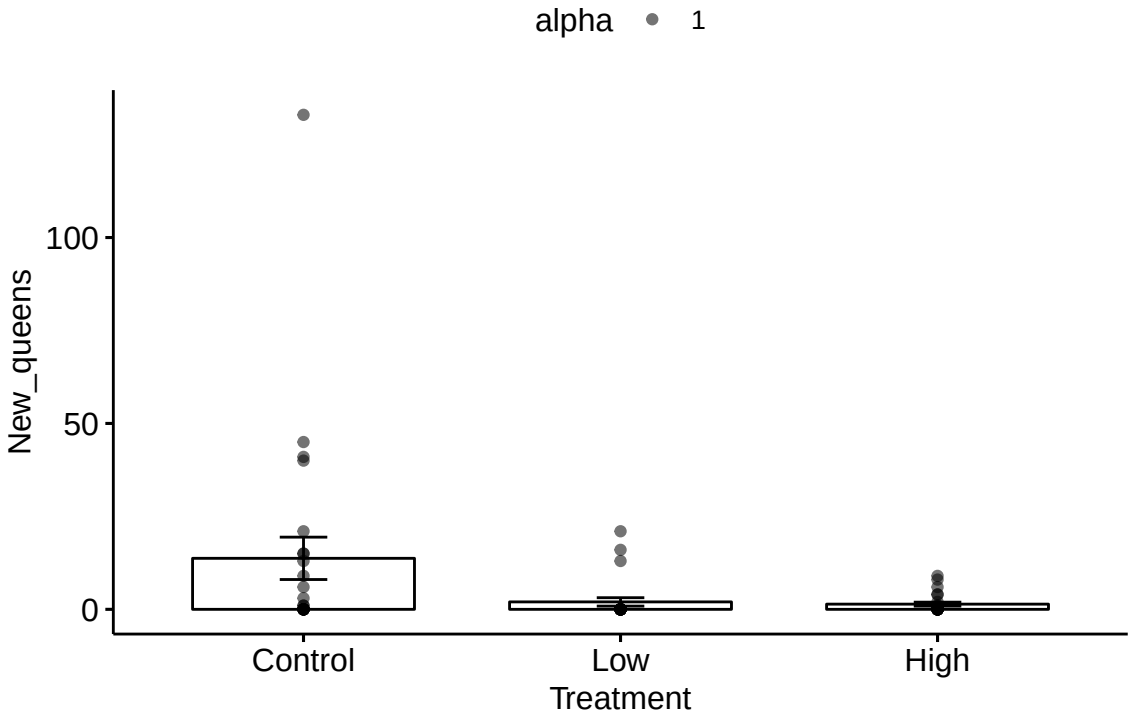
```
ggbarplot(data=bee, x="Treatment", y="New_queens", add = c("mean_se"))
```



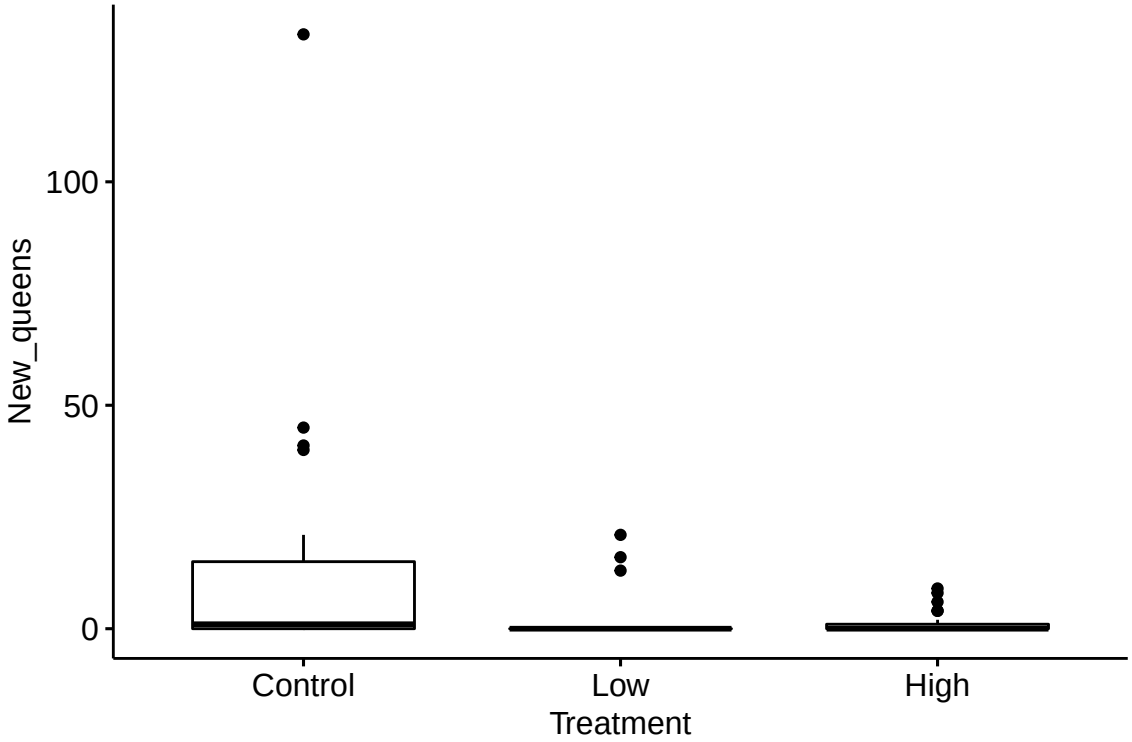
The plot suggests immediately some problems with the plot itself and the associated analysis. First, the y-axis is counts, which means that negative values are impossible. But the standard error bars look like they use standard errors computed from a model that allows infinetly large negative values, and the illustrated standard error bars imply that negative values exist. So these error bars are misleading. Second, it is good practice, especially if sample sizes are modest or small, to "show the data", which means, show the individual data points and not just a summary of the distribution.

Here are three alternative plots for exploratory purposes. The first simply "shows the data" but still uses the misleading standard error bars. The second uses a box plot. The last plots the means and 95% confidence intervals modeled with a GLM (generalized linear model) to account for the count data (the model used could be improved). Notice that the bar length above the mean is longer than the bar length below the mean (that is the interval is asymmetric about the mean). In order to stay focussed on importing data, I leave explanation of these plots and analysis to later chapters.

```
ggbarplot(data=bee, x="Treatment", y="New_queens", add = c("mean_se", "point"))
```

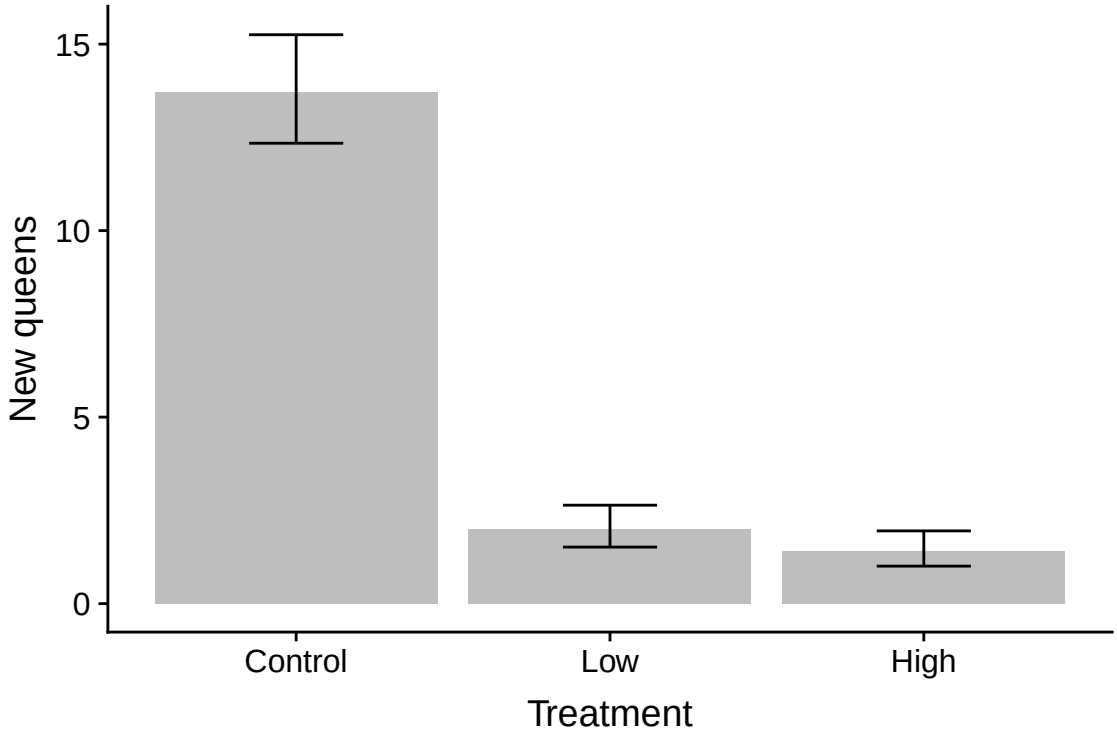



```
ggboxplot(data=bee, x="Treatment", y="New_queens")
```




```
fit.glm <- glm(New_queens ~ Treatment, data=bee, family=poisson())
means.glm <- emmeans(fit.glm, specs="Treatment", type = "response")
gg <- ggplot(data=data.frame(means.glm), aes(x=Treatment, y=rate)) +
  geom_col(fill="gray") +
  geom_errorbar(aes(x=Treatment, ymin=asymp.LCL, ymax=asymp.UCL), width=0.3) +
  ylab("New queens") +
```

```
  NULL
gg
```

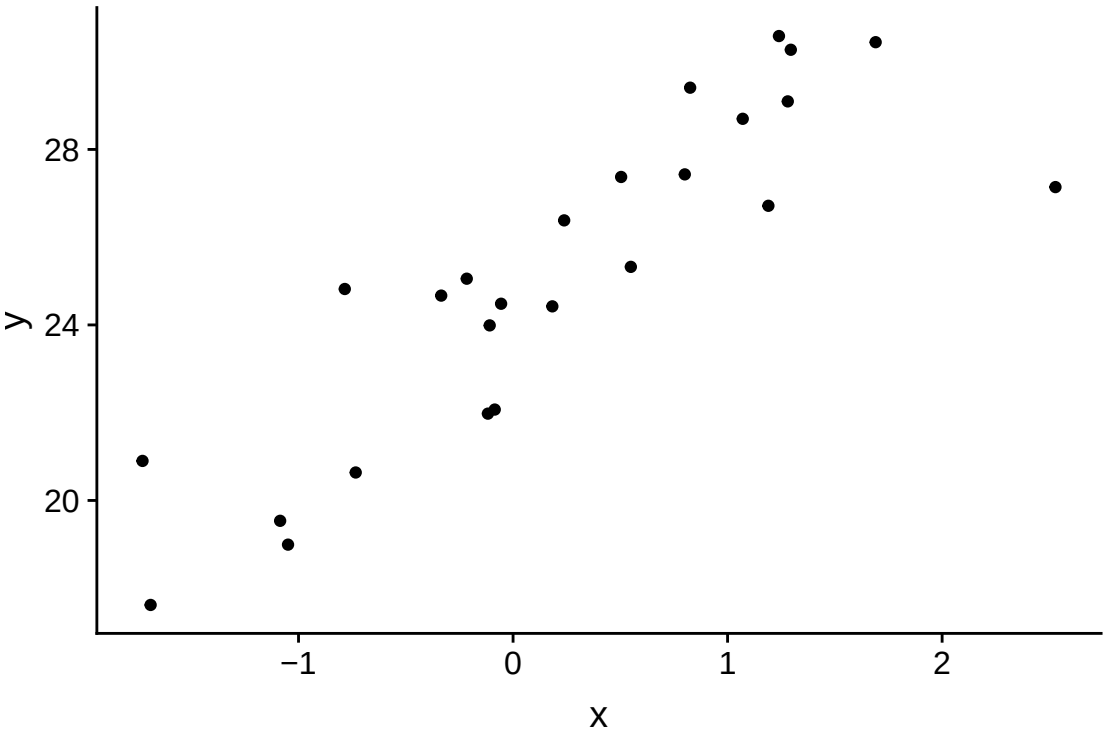


1.10 Creating Fake Data

1.10.1 Continuous X (fake observational data)

A very simple simulation of a regression model

```
n <- 25
beta_0 <- 25
beta_1 <- 3.4
sigma <- 2
x <- rnorm(n)
y <- beta_0 + beta_1*x + rnorm(n, sd=sigma)
qplot(x, y)
```




```
knitr::kable(coefficients(summary(lm(y ~ x))), digits=2)
```

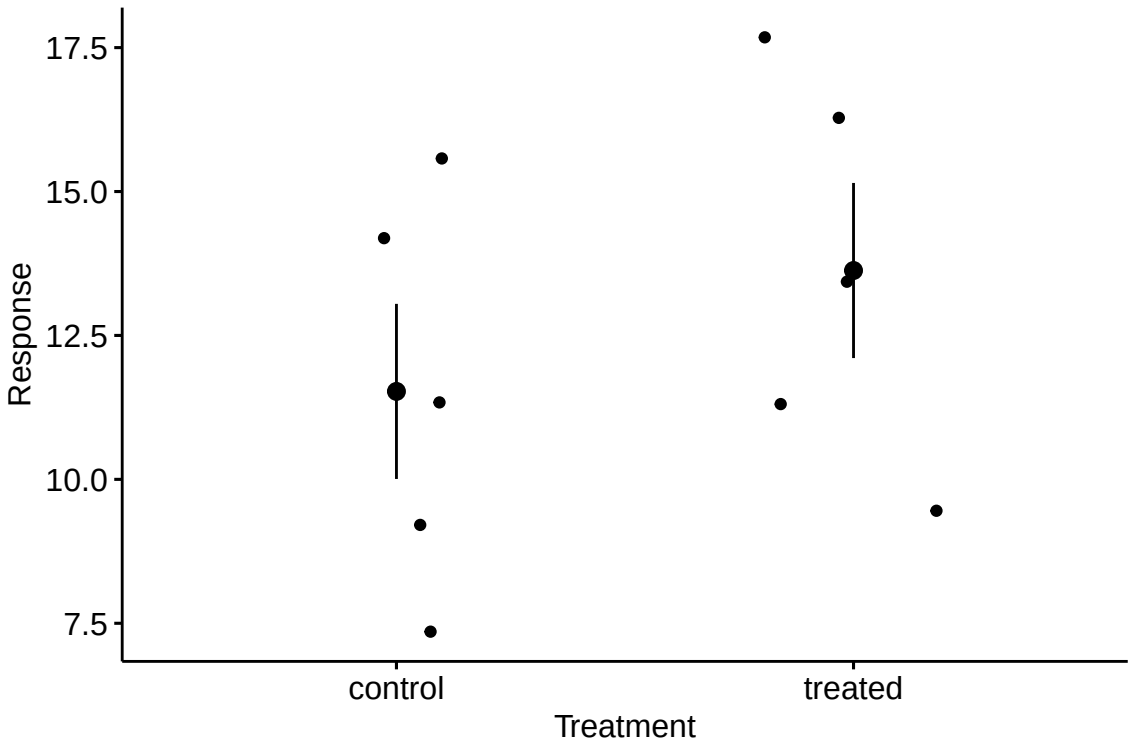
	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	24.46	0.39	62.43	0
x	3.05	0.37	8.25	0

The coefficient of x is the "Estimate". How close is the estimate? Run the simulation several times to look at the variation in the estimate – this will give you a sense of the uncertainty. Increase n and explore this uncertainty. Increase all the way up to $n = 10^5$. Commenting out the qplot line will make this exploration easier.

1.10.2 Categorical X (fake experimental data)

```
n <- 5

fake_data <- data.table(Treatment=rep(c("control", "treated"), each=n))
beta_0 <- 10.5 # mean of untreated
beta_1 <- 2.1 # difference in means (treated - untreated)
sigma <- 3 # the error standard deviation
# the Y variable ("Response") is a function of treatment. We use some matrix
# algebra to get this done.
# Turn the Treatment assignment into a model matrix. Take a peak at X!
X <- model.matrix(~ Treatment, fake_data)
# to make the math easier the coefficients are collected into a vector
beta <- c(beta_0, beta_1)
# you will see the formula Y=Xb many times. Here it is coded in R
fake_data[, Response:=X%*%beta + rnorm(n, sd=sigma)]
# plot it with a strip chart (often called a "dot plot")
ggstripchart(data=fake_data, x="Treatment", y="Response", add = c("mean_se"))
```




```
# fit using base R linear model function
fit <- lm(Response ~ Treatment, data=fake_data)
# display a pretty table of the coefficients
knitr::kable(coefficients(summary(fit)), digits=3)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	11.528	1.521	7.579	0.000
Treatmenttreated	2.100	2.151	0.976	0.358

Check that the intercept is close to beta_0 and the coefficient for Treatment is close to beta_1. This coefficient is the different in means between the treatment levels. It is the simulated effect. Again, change n. Good values are $n = 20$ and $n = 100$. Again, comment out the plot line to make exploration more efficient.

1.11 Saving Data

Let's save the fake data to the "Fake_Data" folder. In the "output" folder create a new folder named "week 01". Then set the path to the output folder:

```
output_path <- "../output" # out to parent directory than down into Fake_data
```

This could be done at the beginning of the notebook, especially if many output files are saved. Regardless, now complete the file_path with the specifics of this save.

```
data_folder <- "week 01"
filename <- "my_first_fake_data.txt"
file_path <- paste(output_path, data_folder, filename, sep="/")
write.table(fake_data, file_path, sep="\t", quote=FALSE)
```

We used `write.table()` to create a tab-delimited text file using `sep="\t"` to specify tabs to separate the row elements. "$\widehat{\ }$" is the standard character string for a tab. Check in your Fake_Data folder and open the file in a text editor.

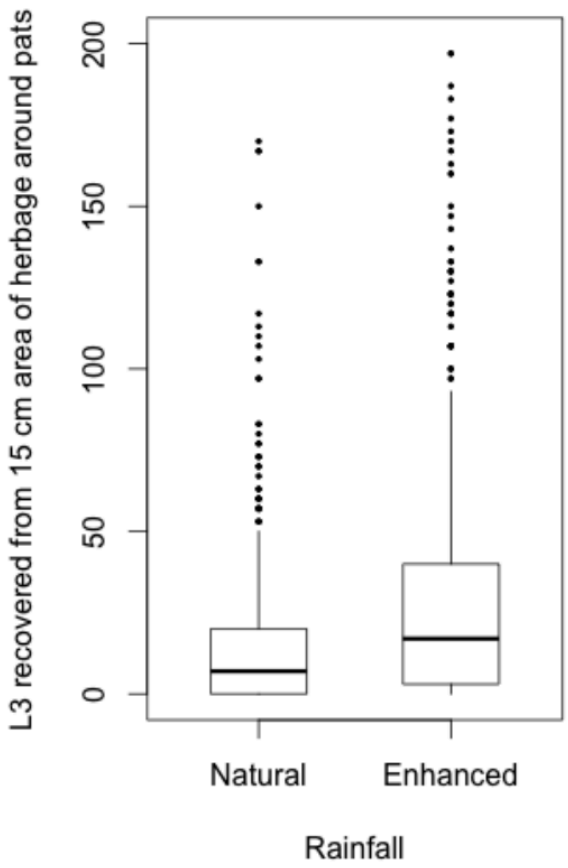


Figure 1.3: Fig. 1 from "Dung beetles reduce livestock…"

1.12 Problems

1. Download the dataset "data-Lodjak.et.al-2016-FuncEcol.xlsx" from the Dryad repository at https://datadryad.org/resource/doi:10.5061/dryad.rd01s. The .xlsx file presents the data cleanly but the trade-off is that the 1) multiple header rows, and 2) spaces in the header labels, 3) parentheses in the header labels make it more complex to import in a usable way. Import the data and plot Body Mass against Age (that is make Body Mass the "Y" variable and Age the "X" variable) using the qplot function. You should recode the column labels to remove spaces and parentheses using the setnames function.
2. Download the dataset "Results2015.txt" from the Dryad repository at https://datadryad.org//resource/doi:10.5061/dryad.65vk4. Try to reproduce Fig. 1. It's not easy. I've inserted the figure below.
3. (grad students only) Download and plot data from a Dryad Repository dataset of your choice.
4. (grad students only) Create fake experimental data with three treatment levels (control, lo_temp, high_temp). This will require three parameters: an intercept (beta_0), an effect of lo_temp (beta_1), and an effect of high_temp (beta_2). You should be able to plug and play from the script above even if you don't underdstand at this point what any of it is! Plot it as a strip chart, as above.

Appendix 3: Getting Started with R

1.13 Get your computer ready

1.13.1 Install R

R is the core software

Download R for your OS

1.13.2 Install R Studio

R Studio is a slick (very slick) GUI interface for developing R projects

Download R Studio Desktop

1.13.3 Resources for installing R and R Studio

On Windows

On a Mac

1.13.4 Install LaTeX

LaTeX ("la-tek") is necessary to use the pdf output of R Markdown.

On Windows

On a Mac

1.14 Start learning

1.14.1 Start with Data Camp Introduction to R

Data Camp: Introduction to R (free online course)

1.14.2 Then Move to Introduction to R Studio

R Studio Essentials, Programming Part 1 (Writing code in RStudio)

1.14.3 Develop your project with an R Studio Notebook

Getting Started with R Markdown

Introducing Notebooks with R Markdown

1.15 Getting Data into R

Getting your data into R

1.16 Additional R learning resources

Getting used to R, RStudio, and R Markdown

Link to list of R Studio webinars

Link to set of R package cheat sheets (amazing!)

Bookdown online books

1.17 Packages used extensively in this text

1. ggplot2
2. data.table
3. mvtnorm
4. lme4
5. nlme
6. emmeans
7. readxl
8. reshape2

Data Visualisation chapter from *R for Data Science*

Graphics for communication chapter from *R for Data Science*

Youtube: An Introduction to The data.table Package

Coursera: The data.table Package

Appendix 4: Online Resources for Getting Started with Linear Modeling in R

Regression Models for Data Science in R by Brian Caffo

Broadening Your Statistical Horizons: Generalized Linear Models and Multilevel Models by J. Legler and P. Roback

The Art of Data Science by Roger D. Peng and Elizabeth Matsui