Basic Statistics

A primer in basic statistics for BCB (Hons) 2018

AJ Smit and Robert Schlegel

2018-04-23

Contents

1 Introduction **7**
- 1.1 Venue, date and time . . . 7
- 1.2 Course outline . . . 7
- 1.3 About this Workshop . . . 8
- 1.4 This is biology: why more R coding? . . . 8
- 1.5 Installing R and RStudio . . . 9
- 1.6 Resources . . . 9
- 1.7 Style and code conventions . . . 9
- 1.8 Assessment and teaching philosophy . . . 10
- 1.9 About this document . . . 10

2 Types of data **12**
- 2.1 Data classes . . . 12
  - 2.1.1 Numerical data . . . 12
  - 2.1.2 Qualitative data . . . 13
  - 2.1.3 Binary data . . . 14
  - 2.1.4 Character values . . . 14
  - 2.1.5 Missing values . . . 14
  - 2.1.6 Complex numbers . . . 14
- 2.2 Viewing our data . . . 15
  - 2.2.1 From the Environment pane . . . 15
  - 2.2.2 `head()` and `tail()` . . . 15
  - 2.2.3 `colnames()` . . . 16
  - 2.2.4 `summary()` . . . 16

3 Descriptive statistics: central tendency and dispersion **18**
- 3.1 Samples and populations . . . 18
- 3.2 Measures of central tendency . . . 19
  - 3.2.1 The mean . . . 19
  - 3.2.2 The median . . . 21
  - 3.2.3 Skewness . . . 21
  - 3.2.4 Kurtosis . . . 22
- 3.3 Measures of variation and spread . . . 22
  - 3.3.1 The variance and standard deviation . . . 22
  - 3.3.2 Quantiles . . . 23
  - 3.3.3 The minimum, maximum and range . . . 24
  - 3.3.4 Covariance . . . 25
  - 3.3.5 Correlation . . . 25
- 3.4 Missing values . . . 25
- 3.5 Descriptive statistics by group . . . 25
  - 3.5.1 Groupwise summary statistics . . . 26

3.5.2 Displays of group summaries . . . 26
3.6 Exercises . . . 27
3.6.1 Exercise 1 . . . 27

4 Graphical data displays **29**
4.1 Qualitative data . . . 29
4.2 Continuous data . . . 31
4.2.1 Frequency distributions (histograms) . . . 31
4.2.2 Box plots . . . 33
4.2.3 Pairwise Scatter plots . . . 34
4.2.4 Bar graphs . . . 36
4.2.5 Density graphs . . . 36
4.2.6 Violin plots . . . 38
4.3 Exercises . . . 39
4.3.1 Exercise 1 . . . 39

5 Distributions **40**
5.1 Discrete distributions . . . 40
5.1.1 Bernoulli distribution . . . 40
5.1.2 Binomial distribution . . . 40
5.1.3 Negative binomial distribution . . . 41
5.1.4 Geometric distribution . . . 41
5.1.5 Poisson distribution . . . 41
5.2 Continuous distributions . . . 41
5.2.1 Normal distribution . . . 41
5.2.2 Uniform distribution . . . 42
5.2.3 Student T distribution . . . 42
5.2.4 Chi-squared distribution . . . 42
5.2.5 Exponential distribution . . . 42
5.2.6 F distribution . . . 43
5.2.7 Gamma distribution . . . 43
5.2.8 Beta distribution . . . 43
5.2.9 Paranormal distributions . . . 43
5.3 Finding one's data distribution . . . 43
5.4 Exercises . . . 48
5.4.1 Exercise 1 . . . 48

6 Inferences about one or two populations **49**
6.1 Assumptions . . . 50
6.1.1 Normality . . . 50
6.1.2 Homoscedasticity . . . 51
6.1.3 Two for one . . . 52
6.2 One-sample t-tests . . . 53
6.2.1 One-sided one-sample t-tests . . . 54
6.2.2 Two-sided one-sample t-tests . . . 55
6.3 Two-sample t-tests . . . 56
6.3.1 One-sided two-sample t-tests . . . 58
6.3.2 Two-sided two-sample t-tests . . . 58
6.4 Paired t-tests . . . 59
6.5 Comparison of two population proportions . . . 59
6.5.1 One-sample and two-sample tests . . . 59
6.5.2 One-sided and two-sided tests . . . 61

6.6 A t-test workflow . . . 62
6.6.1 Loading data . . . 62
6.6.2 Visualising data . . . 62
6.6.3 Formulating a hypothesis . . . 63
6.6.4 Choosing a test . . . 64
6.6.5 Checking assumptions . . . 64
6.6.6 Running an analysis . . . 65
6.6.7 Interpreting the results . . . 65
6.6.8 Drawing conclusions . . . 66
6.6.9 Going further . . . 66
6.7 Exercises . . . 66
6.7.1 Exercise 1 . . . 66
6.7.2 Exercise 2 . . . 66

7 ANOVA **67**
7.1 Remember the t-test . . . 68
7.2 ANOVA . . . 68
7.2.1 Single factor . . . 69
7.2.2 Multiple factors . . . 70
7.2.3 Examples . . . 73
7.3 Alternatives to ANOVA . . . 76
7.3.1 Wilcox rank sum test . . . 76
7.3.2 Kruskall-Wallis rank sum test . . . 77
7.3.3 The SA time data . . . 77
7.4 Exercises . . . 78
7.4.1 Exercise 1 . . . 78
7.4.2 Exercise 2 . . . 79
7.4.3 Exercise 3 . . . 79

8 Simple linear regressions **80**
8.1 The simple linear regression equation . . . 80
8.1.1 The intercept . . . 81
8.1.2 The regression coefficient . . . 82
8.1.3 A graph of the linear regression . . . 82
8.1.4 Predicting from the linear model . . . 82
8.1.5 The coefficient of determination, r^2 . . . 83
8.1.6 Significance test for linear regression . . . 84
8.1.7 Confidence interval for linear regression . . . 85
8.1.8 Prediction interval for linear regression . . . 85
8.1.9 Residual plot . . . 86
8.1.10 Standardised residual . . . 86
8.1.11 Normal probability plot of residuals . . . 86
8.2 Using an additional categorical variable . . . 86

9 Correlations **87**
9.1 Pearson correlation . . . 88
9.2 Spearman rank correlation . . . 89
9.3 Kendall rank correlation . . . 89
9.4 One panel visual . . . 90
9.5 Multiple panel visual . . . 91
9.6 Exercises . . . 91
9.6.1 Exercise 1 . . . 91

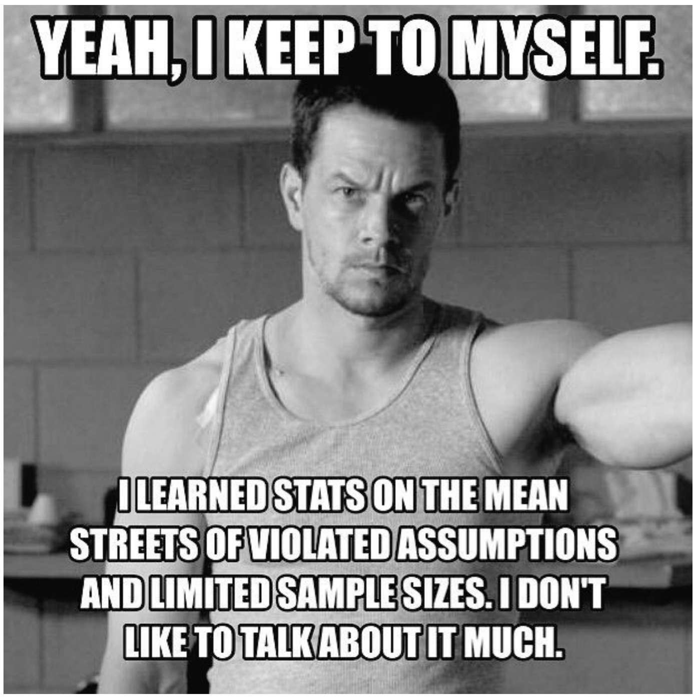
10 Confidence intervals **93**

11 Testing assumptions or: How I learned to stop worrying and transform the data **94**
11.1 The two main assumptions 94
11.1.1 Normality 94
11.1.2 Homoscedasticity 95
11.2 Transforming data 95
11.2.1 Log transform 95
11.2.2 95
11.3 Exercises 95
11.3.1 Exercise 1 95
11.3.2 Exercise 2 96

12 Linear mixed models **97**
12.1 Wilcox rank sum test 98
12.2 Kruskall-Wallis rank sum test 98
12.2.1 Single factor 98
12.2.2 Multiple factors 98
12.3 Generalised linear models 99
12.3.1 Sign Test 99
12.3.2 Wilcoxon Signed-Rank Test 99
12.3.3 Mann-Whitney-Wilcoxon Test 99
12.3.4 Kruskal-Wallis Test 99
12.3.5 Generalised linear models (GLM) 99
12.4 Exercises 99
12.5 Exercise 1 99

13 Chi-squared **100**

Preface


YEAH, I KEEP TO MYSELF.
I LEARNED STATS ON THE MEAN STREETS OF VIOLATED ASSUMPTIONS AND LIMITED SAMPLE SIZES. I DON'T LIKE TO TALK ABOUT IT MUCH.

This is a workshop about the practice of the basic statistics used by biologists, and not about the theory and mathematical underpinnings of the methods used. Each of the Chapters will cover a basic kind of statistical approach, and the main classes of data it applies to. Since much insight and understanding can be gained from visualising our data, we will also explore the main types of graphical summaries that best accompany the statistical methodologies. It is our intention to demonstrate how we go about analysing our data.

Prerequisites

A prerequisite for this course is a basic proficiency in using R (**?**). The necessary experience will have been gained from completing the Intro R Workshop: Data Manipulation, Analysis, and Graphing[1] Workshop that was part of your BCB Core Honours module (i.e. Biostatistics). You will also need a laptop with R and RStudio installed as per the instructions provided in that workshop. If you do not have a personal laptop, most computers in the 5th floor lab will be correctly set up for this purpose.

[1] https://robwschlegel.github.io/Intro_R_Workshop/

1 Introduction

"A scientist worthy of a lab coat should be able to make original discoveries while wearing a clown suit, or give a lecture in a high squeaky voice from inhaling helium. It is written nowhere in the math of probability theory that one may have no fun."

— Eliezer Yudkowsky

"Prediction is very difficult, especially about the future."

—- Niels Bohr

1.1 Venue, date and time

Basic Statistics is the second half of the BSc (Hons) Biostats core module, and will run from 12 April to 26 April 2018. This workshop will take place on Tuesdays from **13:00–17:00**, Thursdays from **10:40–17:00**, and Fridays from **08:30–17:00**. There will be an assignment due about six weeks after the end of this module, and it will provide the other half of the marks for the Biostats module. More on the assignment later.

1.2 Course outline

1. Introduction (this chapter)
2. Types of data
3. Descriptive statistics: Measures of location and dispersion
4. Representing data graphically
5. Distributions
6. One-sample and two-sample tests
7. Multi-sample (>2) tests
8. Linear regression
9. Correlation
10. Confidence intervals

11. Transforming data
12. Generalised linear model (GLM)
13. Chi square tests

The course content can broadly be classified into two parts: *Descriptive Statistics* and *Inferential Statistics*.

Descriptive statistics and their associated statistical and graphical data summaries will be covered in Chapters 3 and 4. In Chapter 5 we will introduce the concepts of data distributions, knowledge of which is required to select the most appropriate inferential statistical methods.

Chapters 6-15 are about inferential statistics. Inferential tests allow us to evaluate hypotheses within a framework of probabalistic theory, which helps us infer the nature of a "population" based on a smaller represenative set of samples. In partiucular, we can infer whether the property under scrutiny (arrived at by means of a designed experiment or a directed sampling programme) occured as a result of deterministic influences, or whether it is as a result of chance.

1.3 About this Workshop

The aim of this workshop is to guide you through the outline given above. The workshop focuses broadly (and unequally) on three groups of concepts:

- Data and distributions
- Descriptive statistics and graphics
- Inferential statistics

Data and distributions are unsurprisingly about the data itself. Here we will talk about the various kinds of data that we will encounter as biologists. In the second part we will describe the data using a combination of numerical and graphical summaries. Third, all of this culminates in trying to infer from a small subset (a sample) of subjects if the characteristics under scrutiny also hold true for the entire population. We may also ask questions about probailities, i.e. measuring the likelihood that an event will occur, or that an experiment has an outcome that is different from a situation where the influential factor(s) has no effect, or that some observation or outcome is non-random.

1.4 This is biology: why more R coding?

Please refer to the Intro R Workshop: Data Manipulation, Analysis and Graphing[1] for why we feel strongly that you use R (?) for the analyses that we will perform here. All of the reasons provided there are valid here too, but one reason perhaps more so than others—R and RStudio promote the principles of *reproducible research*, and in fact make it very easy to implement. We will focus on some of these principles throughout the workshop, and the assignments will require that you submit a fully functional working script, complete with all the notes, memos, examples, data, executable code, and output that will result from completing the course material.

What other options are there for analysing the kinds of data that we will encounter in biological research? Software packages like the ones you may be familiar with, such as Statistica and SPSS, are often used to perform many of these analyses. They are rather limited with regards to the full scope of modern statistical methods in use by biologists today, but many people still use these kinds of software as they provide the basic kinds analyses that still form the staple of the

[1]https://robwschlegel.github.io/Intro_R_Workshop/

biological and medical sciences. For the many reasons provided above, we prefer to use R as the *engine* within which to do our biological data analysis. R is used by academic statisticians the world over, and it is therefore an excellent choice for our purpose here.

1.5 Installing R and RStudio

We assume that you already have R installed on your computer, as all of you will have already completed the the Intro R Workshop. If you need a refresher, please refer to Intro R Workshop: Data Manipulation, Analysis and Graphing[2] for the installation instructions.

1.6 Resources

- New users should introduce themselves to the R ecosystem[3]
- A fancy interactive website that covers a wide range of basic statistics[4]
- An easy to follow walkthrough for a statistical analysis[5]
- Learn more about tidy statistical inference[6]
- A thorough journey through the philosophy of data visualisation[7]
- Google[8]
- Stack Overflow[9]

1.7 Style and code conventions

Early on, develop the habit of unambiguous and consistent style and formatting when writing your code, or anything else for that matter. Pay attention to detail and be pedantic. This will benefit your scientific writing in general. Although many R commands rely on precisely formatted statements (code blocks), style can nevertheless to *some extent* have a personal flavour to it. The key is *consistency*. In this book we use certain conventions to improve readability. We also use a consistent set of conventions to refer to code, and in particular to typed commands and package names.

- Package names are shown in a bold font over a grey box, *e.g.* **`tidyr`**.
- Functions are shown in normal font followed by parentheses and also over a grey box , *e.g.* `plot()`, or `summary()`.
- Other R objects, such as data, function arguments or variable names are again in normal font over a grey box, but without parentheses, *e.g.* `x` and `apples`.
- Sometimes we might directly specify the package that contains the function by using two colons, *e.g.* `dplyr::filter()`.
- Commands entered onto the R command line (console) and the output that is returned will be shown in a code block, which is a light grey background with code font. The commands entered start at the beginning of a line and the output it produces is preceded by `R>`, like so:

[2] https://robwschlegel.github.io/Intro_R_Workshop/
[3] fg2re.sellorm.com
[4] http://students.brown.edu/seeing-theory/
[5] http://www.sthda.com/english/wiki/unpaired-two-samples-t-test-in-r
[6] https://infer.netlify.com
[7] http://www.serialmentor.com/dataviz/
[8] www.google.com
[9] www.stackoverflow.com

```r
set.seed(666)
rnorm(n = 10, mean = 0, sd = 13)
```

```
R>  [1]   9.7930436  26.1866107  -4.6167480  26.3661820 -28.8193679
R>  [6]   9.8591503 -16.9804084 -10.4327544 -23.2991308  -0.5464219
```

Consult these resources for more about R code style :

- Google's R style guide[10]
- The tidyverse style guide[11]
- Hadley Wickham's advanced R style guide[12]

We may also insert maths expressions within the text, like this $f(k) = \binom{n}{k} p^k (1-p)^{n-k}$ or on their own, like this:

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

1.8 Assessment and teaching philosophy

Grades will be based on the aggregate performance across two group projects; the first group project was completed after the Intro R Workshop. The project for this workshop will represent 35% of the total grade for BioStatistics. The remaining 15% will come from daily participation. This will be assessed by the R scripts produced in class and follows these five criteria:

- The script has been uploaded to GitHub
- The script covers the content of the day
- The code runs without errors
- Proper style conventions have been observed
- Liberally commented

BONUS POINTS

- Additional analysis not performed in class
- Additional figure not created in class

In cases where students are borderline between lower and higher grades, a high level of participation in the class discussions and class in general will win the day for the higher grade.

The daily scripts are essential to understanding the material. Although they comprise only 15% of the final grade, performance on the projects is usually correlated with effort on the daily assignments.

Whereas plagiarism will not be tolerated, students ARE encouraged to work together to learn from one another and solve problems in a collaborative and collegial way.

1.9 About this document

This document, which as available as an HTML file that's viewable on a web browser of your choice (anything will do, but we discourage using Internet Explorer) and as a PDF (accessible from the link at the top of any of the website's pages) that may be printed, was prepared by the software tools available to R via RStudio. We use the package called `bookdown` that may

[10] https://google.github.io/styleguide/Rguide.xml
[11] http://style.tidyverse.org
[12] http://adv-r.had.co.nz/Style.html

be accessed and read about here[13] to produce this documentation. The entire source code to reproduce this book is available from my GitHub repo[14].

You will notice that this repository uses GitHub[15], and you are advised to set up your own repository for R scripts and all your data. We will touch on GitHub and the principles of reproducible research later, and GitHub forms a core ingredient of such a workflow.

[13] https://bookdown.org/yihui/bookdown/
[14] https://github.com/ajsmit/Basic_stats
[15] https://github.com

2 Types of data

“The plural of anecdote is not data.”

— Roger Brinner

In this chapter we will, firstly, look at the different kinds of biological and environmental data that are typically encountered by most biologists. The data seen here are not an exhaustive list of all the various types of data out there, but it should represent the bulk of our needs.

After we have become familiar with the different kinds of data, we will look at summaries of these data, which is generally required as the starting point for our analyses. After summarising the data in tables and so forth, we may want to produce graphical summaries to see broad patterns and trends; visual data representations, which complement the tabulated data, will be covered in a later chapter (Chapter 4). Both of these approaches form the basis of “exploratory data analysis.”

2.1 Data classes

In biology we will encounter many kinds of data, and depending on which kind, the type of statistical analysis will be decided.

2.1.1 Numerical data

Numerical data are quantitative in nature. They represent things that can be objectively counted, measured or claculated.

2.1.1.1 Nominal (discrete) data

Integer data (discrete numbers or whole numbers), such as counts. For example, family A may have 3 children and family B may have 1 child, neither may have 2.3 children. Integer data usually answer the question, “how many?” In R integer data are called `int` or `<int>`.

2.1.1.2 Continuous data

These usually represent measured "things," such as something's heat content (temperature, measured in degrees Celsius) or distance (measured in metres or similar), etc. They can be rational numbers including integers and fractions, but typically they have an infinite number of "steps" that depends on rounding (they can even be rounded to whole integers) or considerations such as measurement precision and accuracy. Often, continuous data have upper and lower bounds that depend on the characteristics of the phenomenon being studied or the measurement being taken. In R, continuous data are denoted `num` or `<dbl>`.

The kinds of summaries that lend themselves to continuous data are:

- Frequency distributions
- Relative frequency distributions
- Cumulative frequency distributions
- Bar graphs
- Box plots
- Scatter plots

2.1.1.3 Dates

Dates are a special class of continuous data, and there are many different representations of the date classes. This is a complex group of data, and we will not cover much of it in this course.

2.1.2 Qualitative data

Qualitative data may be well-defined categories or they may be subjective, and generally include descriptive words for classes (e.g. mineral, animal , plant) or rankings (e.g. good, better, best).

2.1.2.1 Categorical data

Because there are categories, the number of members belonging to each of the categories can be counted. For example, there are three red flowers, 66 purple flowers, and 13 yellow flowers. The categories cannot be ranked relative to each other; in the example just provided, for instance, no value judgement can be assigned to the different colours. It is not better to be red than it is to be purple. There are just fewer red flowers than purple ones. Contrast this to another kind of categorical data called "ordinal data" (see next). This class of data in an R dataframe (or in a "tibble") is indicated by `Factor` or `<fctr>`.

The kinds of summaries that lend themselves to categorical data are:

- Frequency distributions
- Relative frequency distributions
- Bar graphs
- Pie graphs (!!!)
- Category statistics

2.1.2.2 Ordinal data

This is a type of categorical data where the classes are ordered (a synonym is "ranked"), typically from low to high (or *vice versa*), but where the magnitude between the ordered classes cannot be precisely measured or quantified. In other words, the difference between them is somewhat subjective (i.e. it is qualitative rather than quantitative). These data are on an ordinal scale. The data may be entered as descriptive character strings (i.e. as words), or they may have been translated to an ordered vector of integers; for example, "1" for terrible, "2" for so-so, "3" for average, "4" for good and "5" for brilliant. Irrespective of how the data are present in the dataframe, computationally (for some calculations) they are treated as an ordered sequence of integers, but they are simultaneously treated as categories (say, where the number of responses that report "so-so" can be counted). Ordinal data usually answer questions such as, "how many categories can the phenomenon be divided into, and how does each category rank with respect to the others?" Columns containing this kind of data are named `Ord.factor` or `<ord>`.

2.1.3 Binary data

Right or wrong? True or false? Accept or reject? Black or white? Positive or negative? Good or bad? You get the idea… In other words, these are observations or responses that can take only one of two mutually exclusive outcomes. In R these are treated as "Logical" data that take the values of `TRUE` or `FALSE` (note the case). In R, and computing generally, logical data are often denoted with 1 for `TRUE` and 0 for `FALSE`. This class of data is indicated by `logi` or `<lgl>`.

2.1.4 Character values

As the name implies, these are not numbers. Rather, they are human words that have found their way into R for one reason or another. In biology we most commonly encounter character values when we have a list of things, such as sites or species. These values will often be used as categorical or ordinal data.

2.1.5 Missing values

Unfortunately, one of the most reliable aspects of any biological dataset is that it will contain some missing data. But how can something contain missing data? One could be forgiven for assuming that if the data are missing, then they obviously aren't contained in the dataset. TO better understand this concept we must think back to the principles of tidy data. Every observation must be in a row, and every column in that row must contain a value. The combination of multiple observations then makes up our matrix of data. Because data are therefore presented in a two-dimensional format, any missing values from an observation will need to have an empty place-holder to ensure the consistency of the matrix. These are waht we are referring to when we speak of "missing values". In R these appear as a `NA` in a dataframe and are slighlty lighter than the other values. These data are indicated in the Environment as `NA` and if a column contains only missing values it will be denoted as `<NA>`.

2.1.6 Complex numbers

"And if you gaze long enough into an abyss, the abyss will gaze back into you."

— Friedrich Nietzsche

In an attempt to allow the shreds of our sanity to remain stitched together we will end here with data types. But be warned, ye who enter, that below countless rocks, and around a legion of corners, lay in wait a myriad of complex data types. We will encounter many of these at the end of this course when we encounter modeling, but by then we will have learned a few techniques that will prepare us for the encounter.

2.2 Viewing our data

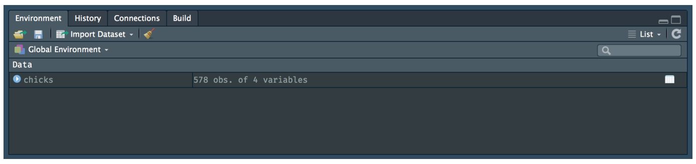
There are many ways of finding broad views of our data in R. The first few functions that we will look at were designed to simply scrutinise the contents of the tibbles, which is the "tidyverse" name for the general "container" that holds our data in the software's environment (i.e. in a block of the computer's memory dedicated to the R software). Whatever data are in R's environment will be seen in the "Environment" tab in the top right of RStudio's four panes.

2.2.1 From the Environment pane

The first way to see what's in the tibble is not really a function at all, but a convenient (and lazy) way of quickly seeing a few basic things about our data. Let us look at the `ChickWeight` data. Load it like so (you'll remember from the Intro R Workshop):

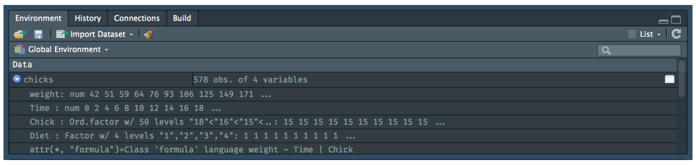
```r
# loads the tidyverse functions; it contains the 'as_tibble()' function
library(tidyverse)
# the 'ChickWeight' data are built into R;
# here we assign it as a tibble to an object named 'chicks'
chicks <- as_tibble(ChickWeight)
```

In the Environment pane, the object named `chicks` will now appear under the panel named Data. To the left of it is a small white arrow in a blue circular background. By default the arrow points to the right. Clicking on it causes it to point down, which denotes that the data contained within the tibble have become expanded. The names of the columns (more correctly called "variables") can now be seen. There you can see the variables `weight`, `Time`, `Chick` and `Diet`. The class of data they represent can be seen too: there's continuous data of class `num`, a variable of `Ord.factor`, and a categorical variable of class `Factor`. Beneath these there's a lot of attributes that denote some meta-data, which you may safely ignore for now.



```
Data
chicks                    578 obs. of 4 variables
```

Figure 2.1: What is in the Chicks data?



```
Data
chicks                    578 obs. of 4 variables
  weight: num 42 51 59 64 76 93 106 125 149 171 ...
  Time : num 0 2 4 6 8 10 12 14 16 18 ...
  Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15 15 15 15 15 15 ...
  Diet : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
  attr(*, "formula")=Class 'formula' language weight ~ Time | Chick
```

Figure 2.2: This is what is in the Chicks data.

2.2.2 `head()` and `tail()`

The `head()` and `tail()` functions simply display top and bottom portions of the tibble, and you may add the `n` argument and an integer to request that only a certain number of rows are returned; by default the top or bottom six rows are displayed.

There are various bits of additional information printed out. The display will change somewhat if there are many more variables than that which can comfortably fit within the width of the output window (typically the Console). The same kinds of information as was returned with the Environment pane expansion arrow are displayed, but the data class is now accompanied by an angle bracket (i.e. `<...>`) notation. For example, `num` in the Environment pane and `<dbl>` as per the `head()` or `tail()` methods are exactly the same: both denote continuous (or "double precision") data.

```
head(chicks)
```

```
R> # A tibble: 6 x 4
R>   weight  Time Chick Diet
R>    <dbl> <dbl> <ord> <fct>
R> 1    42.    0. 1     1
R> 2    51.    2. 1     1
R> 3    59.    4. 1     1
R> 4    64.    6. 1     1
R> 5    76.    8. 1     1
R> 6    93.   10. 1     1
```

```
tail(chicks, n = 2)
```

```
R> # A tibble: 2 x 4
R>   weight  Time Chick Diet
R>    <dbl> <dbl> <ord> <fct>
R> 1   264.   20. 50    4
R> 2   264.   21. 50    4
```

As an alternative to `head()`, you may also simply type the name of the object (here `chicks`) in the Console (or write it in the Source Editor if it is necessary to retain the function for future use) and the top portion of the tibble will be displayed, again trimmed to account for the width of the display.

2.2.3 `colnames()`

This function simply returns a listing of the variable (column) names.

```
colnames(chicks)
```

```
R> [1] "weight" "Time"   "Chick"  "Diet"
```

There is an equivalent function called `rownames()` that may be used to show the names of rows in your tibble, if these are present. Row names are generally discouraged, and we will refrain from using them here.

2.2.4 `summary()`

The next way to see the contents of the tibble is to apply the `summary()` function. Here we see something else. Some descriptive statistics that describe properties of the full set of data are now visible. These summary statistics condense each of the variables into numbers that describe some properties of the data within each column. You will already know the concepts of the "minimum," "median," "mean," and "maximum." These are displayed here.

```
summary(chicks)
```

```
R>      weight           Time           Chick     Diet
R>  Min.   : 35.0   Min.   : 0.00   13     : 12   1:220
```

```
R>  1st Qu.: 63.0   1st Qu.: 4.00   9      : 12   2:120
R>  Median :103.0   Median :10.00   20     : 12   3:120
R>  Mean   :121.8   Mean   :10.72   10     : 12   4:118
R>  3rd Qu.:163.8   3rd Qu.:16.00   17     : 12
R>  Max.   :373.0   Max.   :21.00   19     : 12
R>                                  (Other):506
```

This will serve well as an introduction to the next chapter, which is about descriptive statistics. What are they, and how do we calculate them?

3 Descriptive statistics: central tendency and dispersion

"I think it is much more interesting to live with uncertainty than to live with answers that might be wrong."

—- Richard Feynman

In this Chapter we will focus on basic descriptions of the data, and these initial forrays are built around measures of the central tendency of the data (the mean, median, mode) and the dispersion and variability of the data (standard deviations and their ilk). The materials covered in this and the next two chapters concern a broad discussion that will aid us in understanding our data better prior to analysing it, and before we can draw inference from it. In this work flow it emerges that descriptive statistics generally precede inferential statistics.

Let us now turn to some of the most commonly used descriptive statistics, and learn about how to calculate them.

3.1 Samples and populations

This is a simple toy example. In real life, however, our data will be available in a tibble (initially perhaps captured in MS Excel before importing it as a `.csv` file into R, where the tibble is created). To see how this can be done more realistically using actual data, let us turn to the ChickenWeight data, which, as before, we place in the object `chicks`. Recall the pipe operator (`%>%`, pronounced "then") that we introduced in the Intro R Workshop—we will use that here, throughout. Let us calculate the sample size:

To determine the sample size we can use the `length()` or `n()` functions; the latter is for use within **dplyr**'s `summarise()` method, and it is applied without writing anything inside of the `()`, like this:

```
# first load the tidyverse packages that has the pipe operator, %>%
library(tidyverse)
chicks <- as_tibble(ChickWeight)

# how many weights are available across all Diets and Times?
chicks %>%
  summarise(length = n())
```

```
R> # A tibble: 1 x 1
R>   length
R>    <int>
R> 1    578
```

```
# the same as
length(chicks$weight)
```

```
R> [1] 578
```

Note that this gives us the number of all of the chicks in the experiment, irrespective oif which diet they were given. It will make more sense to know how many chicks were assigned to each of the experimental diets they were raised on.

Task: Figure out the number of chicks in each of the feed categories.

3.2 Measures of central tendency

Table 3.1: Measures of central tendency.

Statistic	Function	Package
Mean	`mean()`	**base**
Median	`median()`	**base**
Skewness	`skewness()`	**e1071**
Kurtosis	`kurtosis()`	**e1071**

The measures of central tendency are also sometimes called “location” statistics. We have already seen summaries of the mean and the median when we called to `summary()` function on the `chicks` data in Chapter 2. Here we shall show you how they can be calculated using some built-in R functions.

3.2.1 The mean

The sample *mean* is the arithmetic average of the data, and it is calculated by summing all the data and dividing it by the sample size, n. The mean, $\bar{x}$, is calculated thus:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

where $x_1 + x_2 + \cdots + x_n$ are the observations and n is the number of observations in the sample.

In R one can quickly apply the `mean()` function to some data. Let us create a vector of arbitrary numbers using the "combine" function, `c()`, and then apply the function for the mean:

```
# combine a series of numbers into a vector;
# hint: use this function in the exercises that we will require from you
# later on...
dat1 <- c(23, 45, 23, 66, 13)
mean(dat1)
```

```
R> [1] 34
```

Below, we use another tidyverse package, **`dplyr`** and its `summarise()` function, whose purpose it is to *summarise* the entire column into one summary statistic, in this case the mean:

```
chicks %>%
  summarise(mean_wt = mean(weight))
```

```
R> # A tibble: 1 x 1
R>   mean_wt
R>     <dbl>
R> 1    122.
```

We can achieve the same using the more traditional syntax, which in some instances may be slightly less verbose, but less user-friendly, especially when multiple summary statistics are required (we shall later on how we can summarise a vector of data into multiple statistics). The traditional syntax is:

```
# the '$' operator is used to denote a variable inside of the tibble
mean(chicks$weight)
```

```
R> [1] 121.8183
```

> **Task:** How would you manually calculate the mean mass for the chicks? Do it now!

Notice above how the two approaches display the result differently: in the first instance, using `summarise()`, the answer is rounded to zero decimal places; in the second, it is displayed (here) at full precision. The precision of the answer that you require depends on the context of your study, so make sure that you use the appropriate number of significant digits. Using the `summarise()` approach again, here is how you can adjust the number of decimal places of the answer:

```
# the value printed in the HTML/PDF versions is incorrect;
# check in the console for correct output
chicks %>%
  summarise(mean_wt = round(mean(weight), 1))
```

```
R> # A tibble: 1 x 1
R>   mean_wt
R>     <dbl>
R> 1    122.
```

> **Task:** What happens when there are missing values (`NA`)? Consult the help file for the `mean()` function, discuss amongst yourselves, and then provide a demonstration to the class of how you would handle missing values. Hint: use the `c()` function to capture a series of data that you can then use to demonstrate your understanding.

At this point it might be useful to point out that the mean (or any function for that matter, even one that does not yet exist) can be programatically calculated. Let us demonstrate the principle by reproducing the mean function from the constituent parts:

```
chicks %>%
  summarise(mean_wt = sum(weight) / n())
```

```
R> # A tibble: 1 x 1
R>   mean_wt
R>     <dbl>
R> 1    122.
```

The mean is quite sensitive to the presence of outliers or extreme values in the data, and it is advised that its use be reserved for normally distributed data from which the extremes/outliers have been removed. When extreme values are indeed part of our data and not simply "noise," then we have to resort to a different measure of central tendency: the median.

> *Task:** In statistics, what do we mean with "noise"?

3.2.2 The median

The *median* can be calculated by "hand" (if you have a small enough amount of data) by arranging all the numbers in sequence from low to high, and then finding the middle value. If there are five numbers, say `5, 2, 6, 13, 1`, then you would arrange them from low to high, i.e. `1, 2, 5, 6, 13`. The middle number is `5`. This is the median. But there is no middle if we have an even number of values. What now? Take this example sequence of six integers (they may also be floating point numbers), which has already been ordered for your pleasure: `1, 2, 5, 6, 9, 13`. Find the middle two numbers (i.e. `5, 6`) and take the mean. It is `5.5`. That is the median.

Let us find the median for the weights of the chickens in the ChickWeight data:

```
chicks %>%
  summarise(med_wt = median(weight))
```

```
R> # A tibble: 1 x 1
R>   med_wt
R>    <dbl>
R> 1   103.
```

The median is therefore the value that separates the lower half of the sample data from the upper half. In normally distributed continuous data the median is equal to the mean. Comparable concepts to the median are the *1st* and *3rd quartiles*, which, respectively, separate the first quarter of the data from the last quarter—see later. The advantage of the median over the mean is that it is unaffected (i.e. not skewed) by extreme values or outliers, and it gives an idea of the typical value of the sample. The median is also used to provide a robust description of non-parametric data (see Chapter 4 for a discussion on normal data and other data distributions).

3.2.3 Skewness

Skewness is a measure of symmetry, and it is best understood by understanding the location of the median relative to the mean. A negative skewness indicates that the mean of the data is less than their median—the data distribution is left-skewed. A positive skewness results from data that have a mean that is larger than their median; these data have a right-skewed distribution.

```r
library(e1071)
skewness(faithful$eruptions)
```

```
R> [1] -0.4135498
```

Task: Is the distribution left or right skewed?

3.2.4 Kurtosis

Kurtosis describes the tail shape of the data's distribution. A normal distribution has zero kurtosis and thus the standard tail shape (mesokurtic). Negative kurtosis indicates data with a thin-tailed (platykurtic) distribution. Positive kurtosis indicates a fat-tailed distribution (leptokurtic).

```r
# library(e1071)
kurtosis(faithful$eruptions)
```

```
R> [1] -1.511605
```

3.3 Measures of variation and spread

Since the mean or median does not tell us everything there is to know about data, we will also have to determine some statistics that inform us about the variation (or spread or dispersal or inertia) around the central/mean value.

Table 3.2: Measures of variation and spread.

Statistic	Function
Variance	`var()`
Standard deviation	`sd()`
Minimum	`min()`
Maximum	`max()`
Range	`range()`
Quantile	`quantile()`
Covariance	`cov()`
Correlation	`cor()`

3.3.1 The variance and standard deviation

The variance and standard deviation are examples of interval estimates. The *sample variance*, S^2, may be calculated according to the following formula:

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

This reads: "the sum of the squared differences from the mean, divided by the sample size minus 1."

To get the *standard deviation*, S, we take the square root of the variance, i.e. $S = \sqrt{S^2}$.

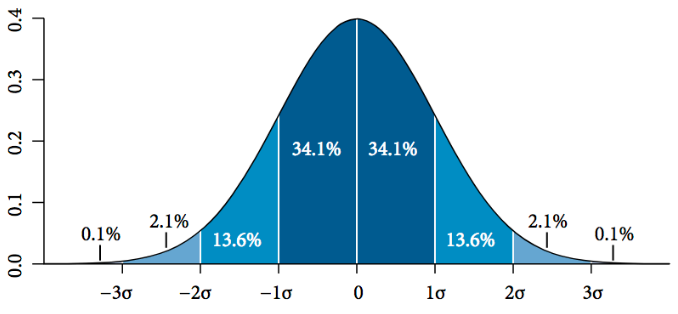
No need to plug these equations into MS Excel. Let us quickly calculate S in R. Again, we use the `chicks` data:

```
chicks %>%
  summarise(sd_wt = sd(weight))
```

```
R> # A tibble: 1 x 1
R>    sd_wt
R>    <dbl>
R> 1  71.1
```

The interpretation of the concepts of mean and median are fairly straight forward and intuitive. Not so for the measures of variance. What does S represent? Firstly, the unit of measurement of S is the same as that of $\bar{x}$ (but the variance doesn't share this characteristic). If temperature is measured in °C, then S also takes a unit of °C. Since S measures the dispersion *around* the mean, we write it as $\bar{x} \pm S$ (note that often the mean and standard deviation are written with the letters *mu* and *sigma*, respectively; i.e. $\mu \pm \sigma$). The smaller S the closer the sample data are to $\bar{x}$, and the larger the value is the further away they will spread out from $\bar{x}$. So, it tells us about the proportion of observations above and below $\bar{x}$. But what proportion? We invoke the the 68-95-99.7 rule: ~68% of the population (as represented by a random sample of n observations taken from the population) falls within $1S$ of $\bar{x}$ (i.e. ~34% below $\bar{x}$ and ~34% above $\bar{x}$); ~95% of the population falls within $2S$; and ~99.7% falls within $3S$.

\begin{figure}[h!]



\caption{The proportions of data representation by the standard deviation. Credit: wikipedia/Standard_deviation} \end{figure}

Like the mean, S is affected by extreme values and outliers, so before we attach S as a summary statistic to describe some data, we need to ensure that the data are in fact normally distributed. We will talk about how to do this in Chapter 6, where we will go over the numerous ways to check the assumption of normality. When the data are found to be non-normal, we need to find appropriate ways to express the spread of the data. Enter the quartiles.

3.3.2 Quantiles

A more forgiving approach (forgiving of the extremes, often called "robust") is to divide the distribution of ordered data into quarters, and find the points below which 25% (0.25, the first quartile), 50% (0.50, the median) and 75% (0.75, the third quartile) of the data are distributed. These are called *quartiles* (for "quarter;" not to be confused with *quantile*, which is a more general form of the function that can be used to divide the distribution into any arbitrary proportion from 0 to 1). In R we use the `quantile()` function to provide the quartiles; we demonstrate two approaches:

```
quantile(chicks$weight)
```

```
R>     0%    25%    50%    75%   100%
R>  35.00  63.00 103.00 163.75 373.00
```

```
chicks %>%
  summarise(min_wt = min(weight),
            qrt1_wt = quantile(weight, p = 0.25),
            med_wt = median(weight),
            qrt3_wt = median(weight, p = 0.75),
            max_wt = max(weight))
```

```
R> # A tibble: 1 x 5
R>   min_wt qrt1_wt med_wt qrt3_wt max_wt
R>    <dbl>   <dbl>  <dbl>   <dbl>  <dbl>
R> 1    35.     63.   103.     103.   373.
```

```
# note median(weight) is the same as quantile(weight, p = 0.5)
# in the summarise() implementation, above
```

> *Task:* What is different about the `quantile()` function that caused us to specify the calculation in the way in which we have done so above? You will have to consult the help file, read it, understand it, think about it, and experiment with the ideas. Take 15 minutes to figure it out and report back to the class.

3.3.3 The minimum, maximum and range

A description of the extent of the data can also be provided by the functions `min()`, `max()` and `range()`.

These statistics apply to data of any distribution, and not only to normal data. This if often the first place you want to start when looking at the data for the first time. We've seen above how to use `min()` and `max()`, so below we will quickly look at how to use `range()` in both the base R and tidy methods:

```
range(chicks$weight)
```

```
R> [1]  35 373
```

```
chicks %>%
  summarise(lower_wt = range(weight)[1],
            upper_wt = range(weight)[2])
```

```
R> # A tibble: 1 x 2
R>   lower_wt upper_wt
R>      <dbl>    <dbl>
R> 1      35.     373.
```

Note that `range()` actually gives us the minimum and maximum values, and not the difference between them. To find the range value properly we must be a bit more clever:

```
range(chicks$weight)[2] - range(chicks$weight)[1]
```

```
R> [1] 338
```

```
chicks %>%
  summarise(range_wt = range(weight)[2] - range(weight)[1])
```

```
R> # A tibble: 1 x 1
R>   range_wt
R>      <dbl>
R> 1     338.
```

3.3.4 Covariance

3.3.5 Correlation

The correlation coefficient of two matched (paired) variables is equal to their covariance divided by the product of their individual standard deviations. It is a normalised measurement of how linearly related the two variables are.

Graphical displays of correlations are provided by scatter plots as can be seen in Section X.

3.4 Missing values

As mentioned in Chapter 2, missing data are pervaise in the biological sciences. Happily for us, R is designed to handle these data easily. It is important to note here explicitly that all of the basic functions in R will by default *NOT* ignore missing data. This has been done so as to prevent the user from accidentally forgetting about the missing data and potentially making errors in later stages in an analysis. Therefore, we must explicitly tell R when we want it to ommit missing values from a calculation. Let's create a small vector of data to demonstrate this:

```
dat1 <- c(NA, 12, 76, 34, 23)

# Without telling R to ommit missing data
mean(dat1)
```

```
R> [1] NA
```

```
# Ommitting the missing data
mean(dat1, na.rm = TRUE)
```

```
R> [1] 36.25
```

Note that this argument, `na.rm = TRUE` may be used in all of the functions we have seen thus far in this chapter.

3.5 Descriptive statistics by group

Above we have revised the basic kinds of summary statistics, and how to calculate them. This is nice. But it can be more useful. The real reason why we might want to see the descriptive statistics is to facilitate comparisons between groups. In the chicks data we calculated the mean (etc.) for all the chickens, over all the diet groups to which they had been assigned (there are four factors, i.e. Diets 1 to 4), and over the entire duration of the experiment (the experiment lasted 21 days). It would be more useful to see what the weights are of the chickens in each of the four groups at the end of the experiment — we can compare means (± SD) and medians (± interquartile ranges, etc.), for instance. You'll notice now how the measures of central tendency is being combined with the measures of variability/range. Further, we can augment this statistical summary with many kinds of graphical summaries, which will be far more revealing of differences (if any) amongst groups. We will revise how to produce the group statistics and show a range of graphical displays.

3.5.1 Groupwise summary statistics

At this point you need to refer to Chapter 10[1] (Tidy data) and Chapter 11[2] (Tidier data) in the **Intro R Workshop** to remind yourself about in what format the data need to be before we can efficiently work with it. A hint: one observation in a row, and one variable per column. From this point, it is trivial to do the various data descriptions, visualisations, and analyses. Thankfully, the `chicks` data are already in this format.

So, what are the summary statistics for the chickens for each diet group at day 21?

```
grp_stat <- chicks %>%
  filter(Time == 21) %>%
  group_by(Diet, Time) %>%
  summarise(mean_wt = round(mean(weight, na.rm = TRUE), 2),
            med_wt = median(weight, na.rm = TRUE),
            sd_wt = round(sd(weight, na.rm = TRUE), 2),
            sum_wt = sum(weight),
            min_wt = min(weight),
            qrt1_wt = quantile(weight, p = 0.25),
            med_wt = median(weight),
            qrt3_wt = median(weight, p = 0.75),
            max_wt = max(weight),
            n_wt = n())
grp_stat
```

```
R> # A tibble: 4 x 11
R> # Groups:   Diet [?]
R>   Diet   Time mean_wt med_wt sd_wt sum_wt min_wt qrt1_wt qrt3_wt max_wt
R>   <fct> <dbl>   <dbl>  <dbl> <dbl>  <dbl>  <dbl>   <dbl>   <dbl>  <dbl>
R> 1 1       21.    178.   166.  58.7  2844.    96.    138.    166.   305.
R> 2 2       21.    215.   212.  78.1  2147.    74.    169.    212.   331.
R> 3 3       21.    270.   281.  71.6  2703.   147.    229.    281.   373.
R> 4 4       21.    239.   237.  43.4  2147.   196.    204.    237.   322.
R> # ... with 1 more variable: n_wt <int>
```

3.5.2 Displays of group summaries

There are several kinds of graphical displays for your data. We will show some which are able to display the spread of the raw data, the mean or median, as well as the appropriate accompanying indicators of variation around the mean or median.

```
library(ggpubr) # needed for arranging multi-panel plots
plt1 <- chicks %>%
  filter(Time == 21) %>%
  ggplot(aes(x = Diet, y = weight)) +
  geom_point(data = grp_stat, aes(x = Diet, y = mean_wt),
             col = "black", fill = "red", shape = 23, size = 3) +
```

[1] https://robwschlegel.github.io/Intro_R_Workshop/tidy.html
[2] https://robwschlegel.github.io/Intro_R_Workshop/tidier.html

```r
  geom_jitter(width = 0.05) + # geom_point() if jitter not required
  labs(y = "Chicken mass (g)") +
  theme_pubr()

plt2 <- ggplot(data = grp_stat, aes(x = Diet, y = mean_wt)) +
  geom_bar(position = position_dodge(), stat = "identity",
           col = NA, fill = "salmon") +
  geom_errorbar(aes(ymin = mean_wt - sd_wt, ymax = mean_wt + sd_wt),
                width = .2) +
  labs(y = "Chicken mass (g)") +
  theme_pubr()
# position_dodge() places bars side-by-side
# stat = "identity" prevents the default count from being plotted

# a description of the components of a boxplot is provided in the help file
# geom_boxplot()
plt3 <- chicks %>%
  filter(Time == 21) %>%
  ggplot(aes(x = Diet, y = weight)) +
  geom_boxplot(fill = "salmon") +
  geom_jitter(width = 0.05, fill = "white", col = "blue", shape = 21) +
  labs(y = "Chicken mass (g)") +
  theme_pubr()

plt4 <- chicks %>%
  filter(Time %in% c(10, 21)) %>%
  ggplot(aes(x = Diet, y = weight, fill = as.factor(Time))) +
  geom_boxplot() +
  geom_jitter(shape = 21, width = 0.1) +
  labs(y = "Chicken mass (g)", fill = "Time") +
  theme_pubr()

ggarrange(plt1, plt2, plt3, plt4, ncol = 2, nrow = 2, labels = "AUTO")
```

3.6 Exercises

3.6.1 Exercise 1

Notice how the data summary for chicken weights contained within `wt_summary` is very similar to the summary returned for `weight` when we apply `summary(chicks)`. Please use the `summarise()` approach and construct a data summary with exactly the same summary statistics for `weight` as that which `summary()` returns.

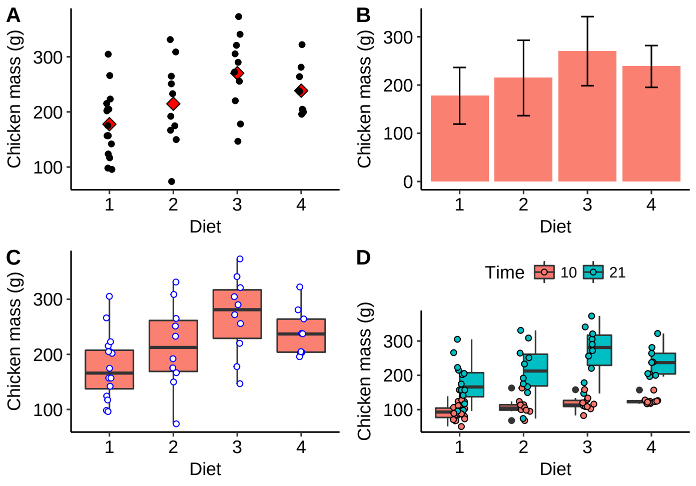


Figure 3.1: A) Scatterplot of the mean and raw chicken mass values. B) Bar graph of the chicken mass values, showing whiskers indicating 1 ±SD. C) Box and whisker plot of the chicken mass data.

4 Graphical data displays

```r
library(tidyverse)
library(ggpubr)
library(RColorBrewer)
library(ggthemes)
```

Here we shall provide examples of many kinds of graphical data summaries. We use **ggplot2** for these figures and will refrain from using the base R graphs. We shall provide examples of various themes so you can see what is available to use for your own plots. We also include various modifications of the default plots so you can get an idea of how to modify some of the plot characteristics, and we always make sure that our graphs are publication ready. The best way to learn is to work by example. Deeper understanding will emerge only from working through all of these example plots, and making your own changes here and there to see how your own modification will affect your graphs' appearance. Then find your own data and plot them. As always, liberally make use of the built-in help facility (the more you do, the easier it becomes, like riding a bicycle). Also, don't be shy to use Google.

4.1 Qualitative data

Qualitative data that describe group representivity to various categories are best presented as frequency distribution histograms (I interchangeably use histograms, frequency histograms, and frequency distribution histograms). Histograms apply to categorical data. Although it can be presented numerically in tabular form, one more frequently creates a bar or pie graph of the number of occurrences in a collection of non-overlapping classes or categories. Both the data and graphical displays will be demonstrated here.

The first case of a frequency distribution histogram is one that shows the raw counts per each of the categories that are represented in the data. The count within each of the categories (represented by a bar graph called a histogram) sums to the sample size, n. In the second case, we may want to report those data as proportions. Here we show the frequency proportion in a collection of non-overlapping categories. For example, we have a sample size of 12 ($n = 12$). In this sample, two are coloured blue, six red, and five purple. The relative proportions are $2/12 = 0.1666667$ blue, $6/12 = 0.5$ red, and $5/12 = 0.4166667$ purple. The important thing to note here is that the relative proportions sum to 1, i.e. $0.1666667 + 0.5 + 0.4166667 = 1$. These data may be presented as a table or as a graph.

Let us demonstrate the numerical and graphical summaries using the built-in `iris` data:

```
# the numerical summary produced by a piped series of functions;
# create a summary of the data (i.e. number of replicates per species)
# used for (A), (B) and (C), below
iris.cnt <- iris %>%
  count(Species) %>% # automagically creates a column, n, with the counts
  mutate(prop = n / sum(n)) # creates the relative proportion of each species
iris.cnt
```

```
R> # A tibble: 3 x 3
R>   Species        n  prop
R>   <fct>      <int> <dbl>
R> 1 setosa        50 0.333
R> 2 versicolor    50 0.333
R> 3 virginica     50 0.333
```

```
# a stacked bar graph with the cumulative sum of observations
plt1 <- ggplot(data = iris.cnt, aes(x = "", y = n, fill = Species)) +
  geom_bar(width = 1, stat = "identity") +
  labs(title = "Stacked bar graph", subtitle = "cumulative sum",
       x = NULL, y = "Count") +
  theme_pubclean() + scale_color_few() +
  scale_fill_few()

# a stacked bar graph with the relative proportions of observations
plt2 <- ggplot(data = iris.cnt, aes(x = "", y = prop, fill = Species)) +
  geom_bar(width = 1, stat = "identity") +
  scale_y_continuous(breaks = c(0.00, 0.33, 0.66, 1.00)) +
  labs(title = "Stacked bar graph", subtitle = "relative proportions",
       x = NULL, y = "Proportion") +
  theme_pubclean() + scale_color_few() +
  scale_fill_few()


# a basic pie chart
plt3 <- plt1 + coord_polar("y", start = 0) +
  labs(title = "Friends don't let...", subtitle = "...friends make pie charts",
       x = NULL, y = NULL) +
  scale_fill_brewer(palette = "Blues") +
  theme_minimal()
# if you seriously want a pie chart, rather use the base R function, `pie()`

# here now a bar graph...
# the default mapping of `geom_bar` is `stat = count`, which is a
# bar for each fo the categories (`Species`), with `count` along y
plt4 <- ggplot(data = iris, aes(x = Species, fill = Species)) +
```

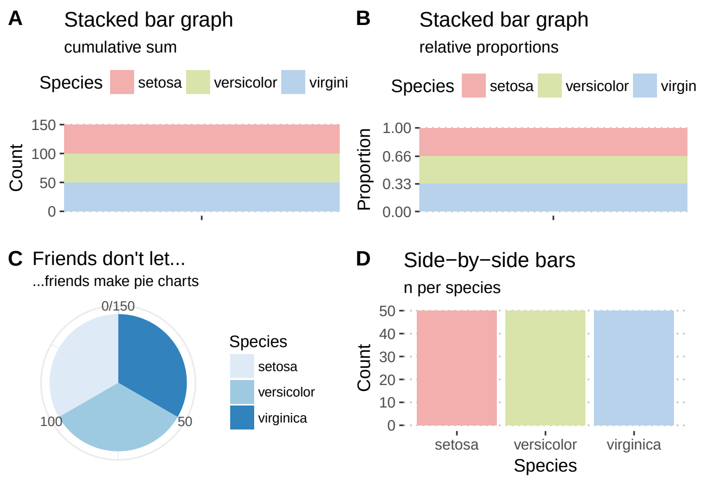


Figure 4.1: Examples of histograms for the built-in Iris data. A) A default frequency histogram showing the count of samples for each of the three species. B) A relative frequency histogram of the same data; here, the sum of counts of samples available for each of the three species is 1. C) A boring pie chart. D) A frequency histogram of raw data counts shown as a series of side-by-side bars.

```
  geom_bar(show.legend = FALSE) +
  labs(title = "Side-by-side bars", subtitle = "n per species", y = "Count") +
 theme_pubclean() + scale_color_few() +
  scale_fill_few()

ggarrange(plt1, plt2, plt3, plt4, nrow = 2, ncol = 2, labels = "AUTO")
```

4.2 Continuous data

4.2.1 Frequency distributions (histograms)

As with discrete data, we have a choice of absolute (Fig. 4.2A) and relative (Fig. 4.2 B-C) frequency histograms. There's also the empirical cumulative distribution function (ECDF) (Fig. 4.2 D) that uses relative proportions, but in this instance it is the relative proportion that each individual observation has towards the sample. Since the purpose of frequency histograms is to count the number of times something takes place or occurs within a category, what do we do when we are faced with continuous data where no categories are available? We can create our own categories, called *bins*. See the Old Faithful data, for example. The eruptions last between 1.6 and 5.1 minutes. So, we create intervals of time spanning these times, and within each count the number of times an event lasts as long as denoted by the intervals. Here we might choose intervals of 1-2 minutes, 2-3 minutes, 3-4 minutes, 4-5 minutes, and 5-6 minutes. The **ggplot2** `geom_histogram()` function automatically creates the bins, but we may specify our own. It is best to explain these principles by example (see Figure 4.2 A-D).

```
# a normal frequency histogram, with count along y
hist1 <- ggplot(data = faithful, aes(x = eruptions)) +
  geom_histogram(colour = "black", fill = "salmon", alpha = 0.6) +
  labs(title = "Old Faithful data",
       subtitle = "A vanilla frequency histogram",
       x = "Eruption duration (min)",
       y = "Count") + theme_pubclean()

# when the binwidth is 1, the density histogram *is* the relative
# frequency histogram
hist2 <- ggplot(data = faithful, aes(x = eruptions)) +
  geom_histogram(aes(y = ..density..),
                 position = 'identity', binwidth = 1,
                 colour = "black", fill = "salmon", alpha = 0.6) +
  labs(title = "Old Faithful data",
       subtitle = "Relative frequency histogram",
       x = "Eruption duration (min)",
       y = "Count") + theme_pubclean()


# if binwidth is something other than 1, the relative frequency in
# a histogram is ..density.. * binwidth
hist3 <- ggplot(data = faithful, aes(x = eruptions)) +
  geom_histogram(aes(y = 0.5 * ..density..),
                 position = 'identity', binwidth = 0.5,
                 colour = "black", fill = "salmon", alpha = 0.6) +
  labs(title = "Old Faithful data",
       subtitle = "Relative frequency histogram",
       x = "Eruption duration (min)",
       y = "Relative contribution") + theme_pubclean()

# ECDF
hist4 <- ggplot(data = faithful, aes(x = eruptions)) +
  stat_ecdf() +
  labs(title = "Old Faithful data",
       subtitle = "ECDF",
       x = "Eruption duration (min)",
       y = "Relative contribution") + theme_pubclean()

ggarrange(hist1, hist2, hist3, hist4, ncol = 2, nrow = 2, labels = "AUTO")
```

What if we have continuous data belonging with multiple categories? The `iris` data provide a nice set of measurements that we may use to demonstrate a grouped frequency histogram. These data are size measurements (cm) of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of three species of *Iris*. The species are *Iris setosa*, *I. versicolor*, and *I. virginica*.

```
# first we make long data
iris.long <- iris %>%
  gather(key = "variable", value = "size", -Species)

ggplot(data = iris.long, aes(x = size)) +
  geom_histogram(position = "dodge", # ommitting this creates a stacked histogram
                 colour = NA, bins = 20,
                 aes(fill = Species)) +
  facet_wrap(~variable) +
```

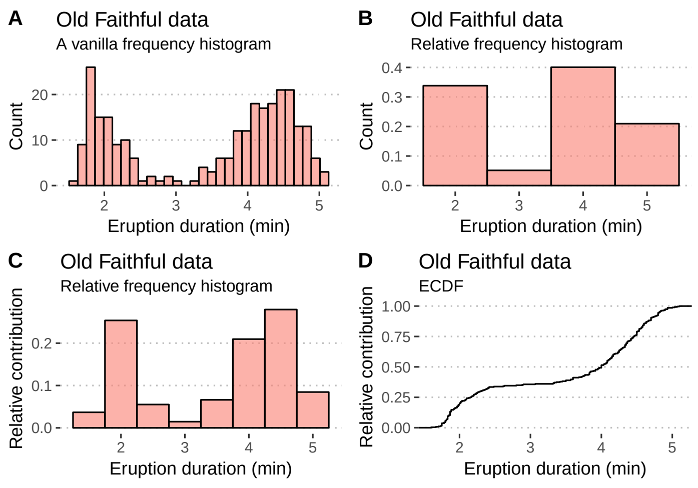


Figure 4.2: Example histograms for the Old Faithful data. A) A default frequency histogram with the count of eruption times falling within the specified bins. B) A relative frequency histogram with bins adjusted to a width of 1 minute intervals; here, the sum of counts within each of the four bins is 1. C) Another relative frequency histogram, but with the bins adjusted to each be 0.5 minute increments; again the sum of counts represented by each bin is equal to 1.

```
  labs(title = "Iris data",
       subtitle = "Grouped frequency histogram",
       x = "Size (mm)",
       y = "Count") +
  theme_pubclean()
```

4.2.2 Box plots

Box plots are sometimes called box-and-whisker plots. These graphs are a a graphical representation of the data based on its quartiles as well as its smallest and largest values. The keen eye can glance the "shape" of the data distribution; they provide an alternative view to that given by the frequency distribution. A variation of the basic box-and-whisker plot is to superimpose a jittered scatter plot of the raw data on each bar.

From the `geom_boxplot` documentation, which says it best (type `?geom_boxplot`):

"The lower and upper hinges correspond to the first and third quartiles (the 25th and 75th percentiles)."

"The upper whisker extends from the hinge to the largest value no further than 1.5 * IQR from the hinge (where IQR is the inter-quartile range, or distance between the first and third quartiles). The lower whisker extends from the hinge to the smallest value at most 1.5 * IQR of the hinge. Data beyond the end of the whiskers are called 'outlying' points and are plotted individually."

"In a notched box plot, the notches extend 1.58 * IQR / sqrt(n). This gives a roughly 95% confidence interval for comparing medians."

Here be examples:

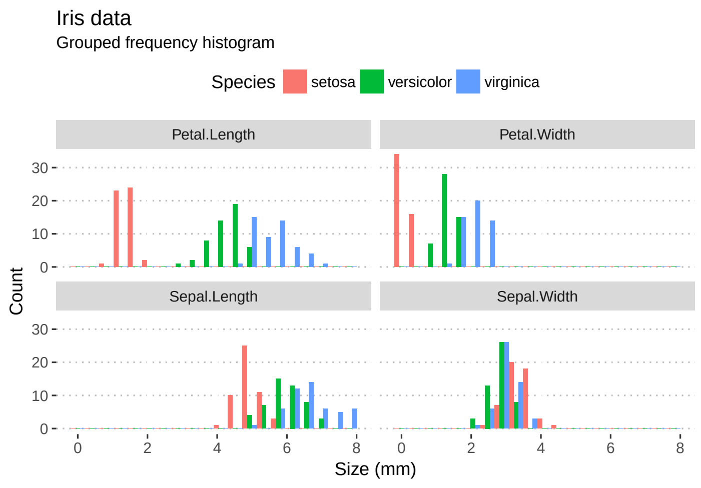


Figure 4.3: Panelled grouped histograms for the four Iris variables.

```r
plt1 <- ggplot(data = iris, aes(x = Species, y = Sepal.Length, fill = Species)) +
  geom_boxplot(show.legend = FALSE, notch = FALSE) + theme_pubclean() +
  labs(y = "Sepal length (mm)") +
  theme(axis.text.x = element_text(face = "italic"))

plt2 <- ggplot(data = iris.long, aes(x = Species, y = size)) +
  geom_boxplot(fill = "red", alpha = 0.4, notch = TRUE) +
  geom_jitter(width = 0.1, shape = 21, colour = "blue", fill = NA, alpha = 0.2) +
  facet_wrap(~variable, nrow = 1) +
  labs(y = "Size (mm)") + theme_pubclean() +
  theme(axis.text.x = element_text(face = "italic")) +
  theme(axis.ticks.length=unit(-0.25, "cm"), axis.ticks.margin=unit(0.5, "cm"))

ggarrange(plt1, plt2, nrow = 2, ncol = 1, labels = "AUTO")
```

Box-and-whisker plots have traditionally been used to display data that are not normally distributed, but I like to use them for any old data, even normal data. I prefer these over the old-fashioned bar graphs (as seen later in this section).

The **ggpubr** package provides many convenience functions for the drawing of publication quality graphs, many of which include summaries of pairwise comparisons (e.g. in t-tests and ANOVAs). Please see here[1] and here[2].

4.2.3 Pairwise Scatter plots

This graph shows the relationship between two (matched) continuous variables. The statistical strength of the relationship can be indicated by a correlation (no causal relationship implied as is the case here) or a regression (when a causal link of x on y is demonstrated).

```r
plt1 <- ggplot(data = iris, aes(x = Petal.Length, y = Petal.Width, colour = Species)) +
  geom_point() +
  labs(x = "Petal length (mm)", y = "Petal width (mm)") +
```

[1] http://www.sthda.com/english/articles/24-ggpubr-publication-ready-plots/
[2] http://www.sthda.com/english/rpkgs/ggpubr/

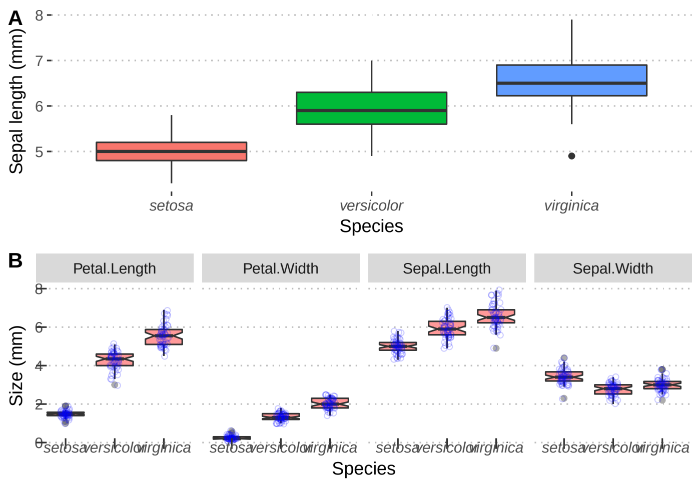


Figure 4.4: Examples of box plots made for the Iris data. A) A default box plot for one of the variables only. B) A panelled collection of box plots, one for each of the four variables, with a scatterplot to indicate the spread of the actual replicates.

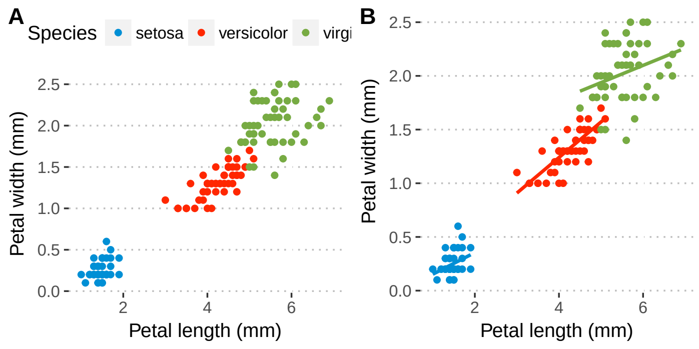


Figure 4.5: Examples of scatterplots made for the Iris data. A) A default scatter plot showing the relationship between petal length and width. B) The same as (A) but with a correlation line added.

```
  theme(legend.position = c(0.18, 0.85)) +
  scale_color_fivethirtyeight() +
  scale_fill_fivethirtyeight() +
  theme_pubclean()

plt2 <- ggplot(data = iris, aes(x = Petal.Length, y = Petal.Width, colour = Species)) +
  geom_point(show.legend = FALSE) +
  geom_smooth(method = "lm", se = FALSE, show.legend = FALSE) +
  scale_color_fivethirtyeight() +
  scale_fill_fivethirtyeight() +
  labs(x = "Petal length (mm)", y = "Petal width (mm)") +
  theme_pubclean()

ggarrange(plt1, plt2, ncol = 2, nrow = 1, labels = "AUTO")
```

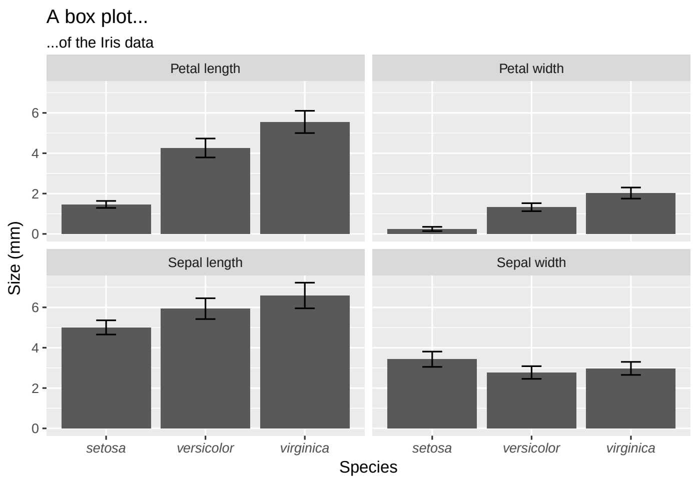


Figure 4.6: Box plots of the mean ± SD of the four Iris variables.

4.2.4 Bar graphs

Bar graphs display the mean plus/minus some measure of variation around the mean—typically the standard error or the standard deviation. The mean±SE and mean±SD are typically used for normally-distributed data. Here I provide an example bar graph for one of the Iris data set's variables:

```r
# first make nice labels for the facets because the default ones
# in the dataframe are not so nice; use the `labeller()` function
# to receive the new variable names defined here
facet.names <- c(Petal.Length = "Petal length",
                 Petal.Width = "Petal width",
                 Sepal.Length = "Sepal length",
                 Sepal.Width = "Sepal width")

# start with the `iris.long` long data that were produced above
# we create summaries of mean and SD and squirt it directly
# into the ggplot functions
iris.long %>%
  group_by(Species, variable) %>%
  summarise(mean.size = mean(size),
            sd.size = sd(size)) %>%
  ggplot(aes(x = Species, y = mean.size)) +
  geom_bar(stat = "identity") +
  geom_errorbar(aes(ymin = mean.size - sd.size, ymax = mean.size + sd.size), width = 0.2) +
  facet_wrap(~variable, labeller = labeller(variable = facet.names)) +
  labs(y = "Size (mm)", title = "A box plot...", subtitle = "...of the Iris data") +
  theme(axis.text.x = element_text(face = "italic"))
```

4.2.5 Density graphs

Often when we are displaying a distribution of data we are interested in the "shape" of the data more than the actual count of values in a specific category, as shown by a standard histogram. When one wishes to more organically visualise the frequency of values in a sample set a density graphs is used. These may also be thought of as smooth histograms. These work well with histograms and rug plots, as we may see in the figure below. It is important to note with density plots that they show the relative density of the distribution along the Y axis, and *not* the counts of the data. This can of course be changed, as seen below, but is not the default setting. Sometimes it can be informative to see how different the count and density distributions appear.

```
# a normal density graph
dens1 <- ggplot(data = faithful, aes(x = eruptions)) +
  geom_density(colour = "black", fill = "salmon", alpha = 0.6) +
  labs(title = "Old Faithful data",
       subtitle = "A vanilla density plot",
       x = "Eruption duration (min)",
       y = "Density") + theme_pubr()

# a density and rug plot combo
dens2 <- ggplot(data = faithful, aes(x = eruptions)) +
  geom_density(colour = "black", fill = "salmon", alpha = 0.6) +
  geom_rug(colour = "red") +
  labs(title = "Old Faithful data",
       subtitle = "A density and rug plot",
       x = "Eruption duration (min)",
       y = "Density") + theme_pubr()

# a relative frequency histogram overlayed with a density plot
dens3 <- ggplot(data = faithful, aes(x = eruptions)) +
  geom_histogram(aes(y = ..density..),
                 position = 'identity', binwidth = 1,
                 colour = "black", fill = "turquoise", alpha = 0.6) +
  geom_density(colour = "black", fill = "salmon", alpha = 0.6) +
  labs(title = "Old Faithful data",
       subtitle = "Relative frequency with density",
       x = "Eruption duration (min)",
       y = "Density") + theme_pubr()

# a normal frequency histogram with density overlayed
# note that the density curve must be adjusted by
# the number of data points times the bin width
dens4 <- ggplot(data = faithful, aes(x = eruptions)) +
  geom_histogram(aes(y = ..count..),
                 binwidth = 0.2, colour = "black", fill = "turquoise", alpha = 0.6) +
  geom_density(aes(y = ..density.. * nrow(datasets::faithful) * 0.2), position = "identity",
               colour = "black", fill = "salmon", alpha = 0.6) +
  labs(title = "Old Faithful data",
       subtitle = "Frequency with density",
       x = "Eruption duration (min)",
       y = "Count") + theme_pubr()

ggarrange(dens1, dens2, dens3, dens4, ncol = 2, nrow = 2, labels = "AUTO")
```

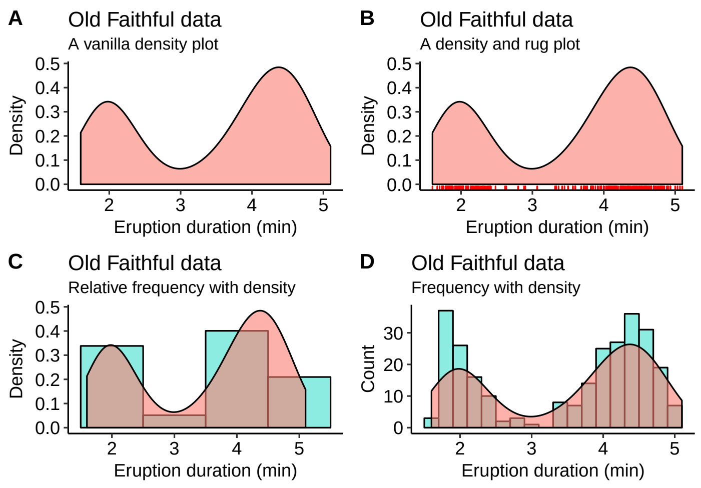


Figure 4.7: A bevy of density graphs option based on the iris data. A) A lone density graph. B) A density graph accompanied by a rug plot. C) A histogram with a density graph overlay. D) A ridge plot.

4.2.6 Violin plots

The density graph is not limited to it's use with histograms. We may combine this concept with box plots, too. These are known as violin plots and are very useful when we want to show the distribution of multiple categories of the same variable alongside one another. Violin plots may show the same information as box plots but take things one step further by allowing the shape of the boxplot to also show the distribution of the data within the sample set. We will use the `iris` data below to highlight the different types of violin plots one may use.

```r
# A basic violin plot
vio1 <- ggplot(data = iris, aes(x = Species, y = Sepal.Length, fill = Species)) +
  geom_violin() +
  theme_pubclean() + theme(legend.position = "none") +
  labs(title = "Iris data",
       subtitle = "Basic violin plot", y = "Sepal length (mm)") +
  theme(axis.text.x = element_text(face = "italic"))

# Aviolin plot showing the quartiles as lines
vio2 <- ggplot(data = iris, aes(x = Species, y = Sepal.Length, fill = Species)) +
  geom_violin(show.legend = FALSE, draw_quantiles = c(0.25, 0.5, 0.75)) +
  theme_pubclean() + theme(legend.position = "none") +
  labs(title = "Iris data",
       subtitle = "Violin plot with quartiles", y = "Sepal length (mm)") +
  theme(axis.text.x = element_text(face = "italic"))

# Box plots nested within violin plots
vio3 <- ggplot(data = iris, aes(x = Species, y = Sepal.Length, colour = Species)) +
  geom_violin(fill = "grey70") +
  geom_boxplot(width = 0.1, colour = "grey30", fill = "white") +
  theme_pubclean() + theme(legend.position = "none") +
  labs(title = "Iris data",
       subtitle = "Box plots nested within violin plots", y = "Sepal length (mm)") +
```

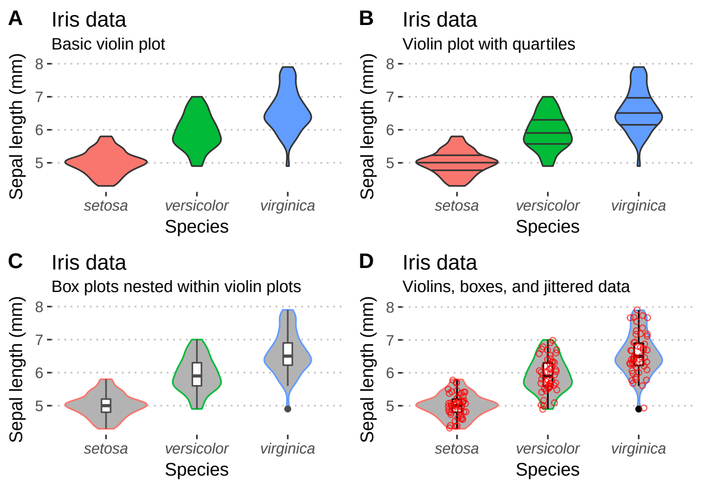


Figure 4.8: Variations of violin plots.

```
  theme(axis.text.x = element_text(face = "italic"))

# Boxes in violins with the raw data jittered about
vio4 <- ggplot(data = iris, aes(x = Species, y = Sepal.Length, colour = Species)) +
  geom_violin(fill = "grey70") +
  geom_boxplot(width = 0.1, colour = "black", fill = "white") +
  geom_jitter(shape = 1, width = 0.1, colour = "red", alpha = 0.7, fill = NA) +
  theme_pubclean() + theme(legend.position = "none") +
  labs(title = "Iris data",
       subtitle = "Violins, boxes, and jittered data", y = "Sepal length (mm)") +
  theme(axis.text.x = element_text(face = "italic"))

ggarrange(vio1, vio2, vio3, vio4, ncol = 2, nrow = 2, labels = "AUTO")
```

4.3 Exercises

4.3.1 Exercise 1

Choose a dataset, either one of the many built into R or one of your own, and create four distinctly different figures. Use `ggarrange()` to stitch them together in a meaningful way.

5 Distributions

Therefore, we must next learn about the different types of data distributions we are likely to encounter in the wild.

5.1 Discrete distributions

A discrete random variable has a finite or countable number of possible values. As the name suggests, it models integer data. Below we provide options to generate and visualise data belonging to several classes of discrete distributions. Later (Chapter 11) we will learn how to transform these data prior to performing the appropriate statistical analysis.

5.1.1 Bernoulli distribution

A Bernoulli random variable, x, takes the value 1 with probability p and the value 0 with probability $q = 1p$. It is used to represent data resulting from a *single* experiment with binary (yes or no; black or white; positive or negative; success or failure; dead or alive;) outcomes, such as a coin toss—there are only two options, heads or tails. Nothing else. Here, p represents the probability of the one outcome and q the probability of the other outcome. The distribution of the possible outcomes, x, is given by:

$$f(x;p) = \begin{cases} p, & \text{if } x = 1 \\ 1 - p, & \text{if } x = 0 \end{cases}$$

5.1.2 Binomial distribution

A binomial random variable, x, is the sum of n independent Bernoulli random variables with parameter p. This data distribution results from repeating identical experiments that produce a binary outcome with probability p a specified number of times, and choosing n samples at random. As such, it represents a collection of Bernoulli trials.

$$f(x; n, p) = \binom{n}{x} p^x (1-p)^{n-x}$$

5.1.3 Negative binomial distribution

A negative binomial random variable, x, counts the number of successes in a sequence of independent Bernoulli trials with probability p before r failures occur. This distribution could for example be used to predict the number of heads that result from a series of coin tosses before three tails are observed:

$$f(x; n, r, p) = \binom{x+r-1}{x} p^x (1-p)^r$$

where x is the number of successes, r is the number of failures, and p is the probability of success

5.1.4 Geometric distribution

A geometric random variable, x, represents the number of trials that are required to observe a single success. Each trial is independent and has success probability p. As an example, the geometric distribution is useful to model the number of times a die must be tossed in order for a six to be observed. It is given by:

$$f(x; p) = (1-p)^x p$$

5.1.5 Poisson distribution

A Poisson random variable, x, tallies the number of events occurring in a fixed interval of time or space, given that these events occur with an average rate λ. Poisson distributions can be used to model events such as meteor showers and or number of people entering a shopping mall. This equation describes the Poison distribution:

$$f(x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$$

5.2 Continuous distributions

5.2.1 Normal distribution

Another name for this kind of distribution is a Gaussian distribution. A random sample with a Gaussian distribution is normally distributed. These values are identically distributed and independent—we say they are independent and identically distributed random variables (i.i.d.), and they have an expected mean given by μ (or $\hat{x}$ in Chapter 3.2.1) and a finite variance given by σ^2 (or S^2 in Chapter 3.3.1); if the number of samples drawn from a population is sufficiently large, the estimated mean and SD will be indistinguishable from the population (as per the central limit theorem).

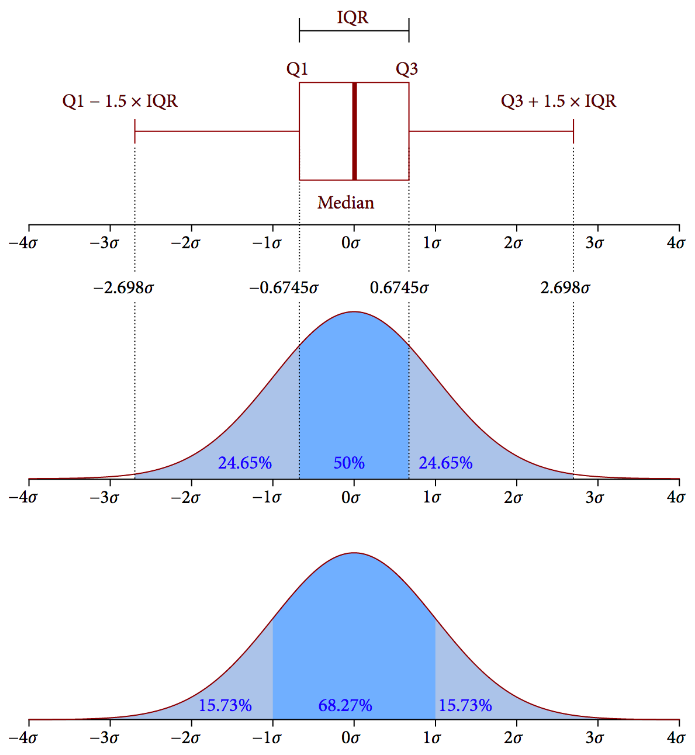


Figure 5.1: Boxplot and probability density function of a normal distribution. Credit: Wikipedia.

5.2.2 Uniform distribution

The continuous uniform distribution is sometime called a rectangular distribution. Simply, it states that all measurements of the same magnitude included with this distribution are equally probable. This is basically random numbers.

5.2.3 Student T distribution

This is a continuous probability distribution that arises when estimating the mean of a normally distributed population in situations where the sample size is small and population standard deviation is unknown. It is used in the statistical significance testing between the means of different sets of samples, and not much so in the modelling of natural phenomena.

5.2.4 Chi-squared distribution

Mostly used in hypothesis testing, but not to encapsulate the distribution of data drawn to represent natural phenomena.

5.2.5 Exponential distribution

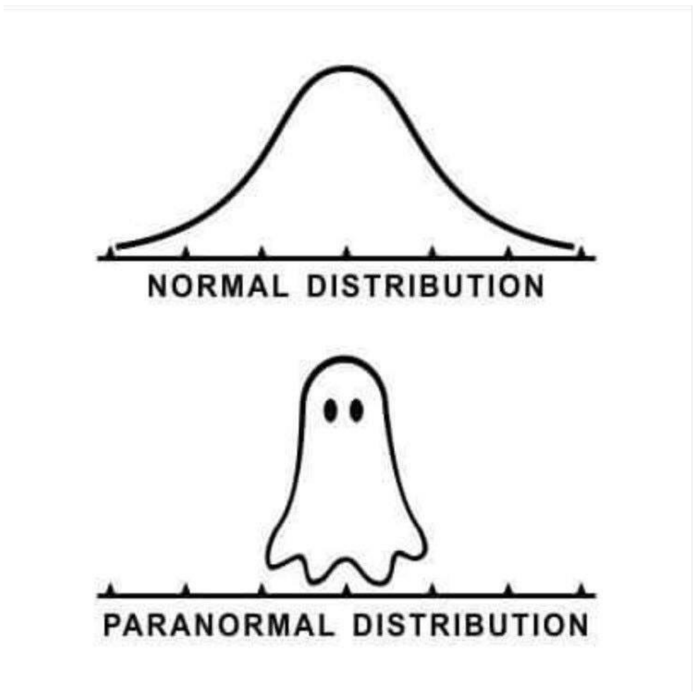
This is a probability distribution that describes the time between events in a Poisson point process, i.e., a process in which events occur continuously and independently at a constant average rate.

5.2.6 F distribution

5.2.7 Gamma distribution

5.2.8 Beta distribution

5.2.9 Paranormal distributions



5.3 Finding one's data distribution

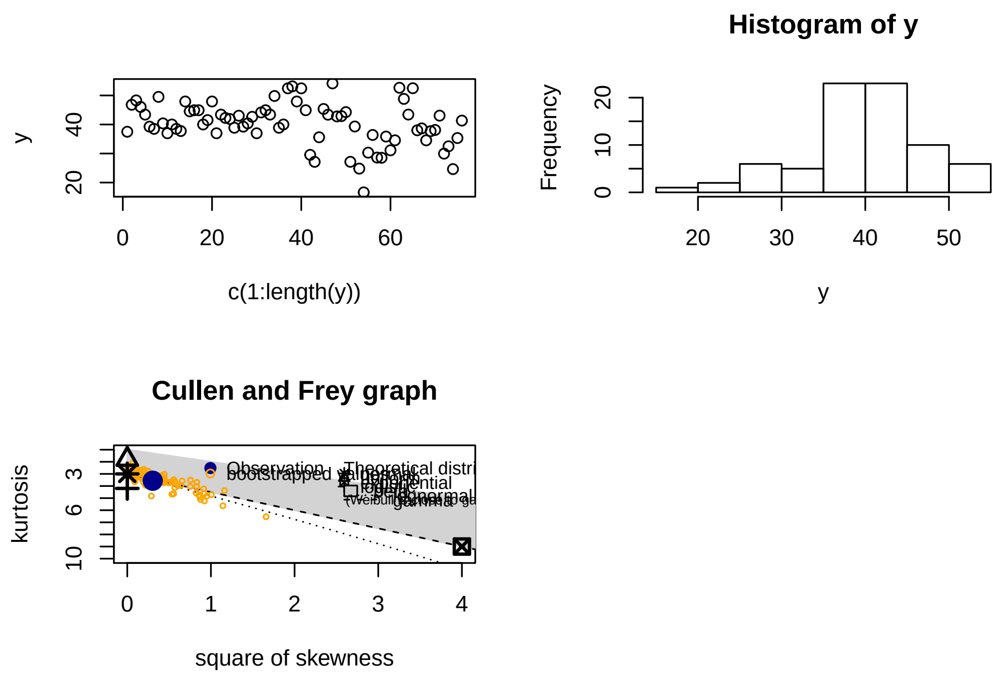
data belonging to a sample will never exactly follow a specific distribution, even when the test for normality says it does—there will always be a small probability that they are non-normal and is in fact better described by some other distribution. In other words, data are only *compatible* with a certain distribution, and one can never answer the question "Does my data follow the distribution xy exactly?" as simply as providing a yes/no answer. So what now? How does one find one's data distribution? We can use the *Cullen and Frey graph* function that lives in the **fitdistrplus** package. This graph tells us whether the skewness and kurtosis of our data are consistent with that of a particular distribution. We will demonstrate by generating various data distributions and testing them using the Cullen and Frey graph.

```
library(fitdistrplus)
```

```
## Loading required package: MASS

## Loading required package: survival

## Loading required package: methods
```

```r
library(logspline)

# Generate log-normal data
y <- c(37.50,46.79,48.30,46.04,43.40,39.25,38.49,49.51,40.38,36.98,40.00,
38.49,37.74,47.92,44.53,44.91,44.91,40.00,41.51,47.92,36.98,43.40,
42.26,41.89,38.87,43.02,39.25,40.38,42.64,36.98,44.15,44.91,43.40,
49.81,38.87,40.00,52.45,53.13,47.92,52.45,44.91,29.54,27.13,35.60,
45.34,43.37,54.15,42.77,42.88,44.26,27.14,39.31,24.80,16.62,30.30,
36.39,28.60,28.53,35.84,31.10,34.55,52.65,48.81,43.42,52.49,38.00,
38.65,34.54,37.70,38.11,43.05,29.95,32.48,24.63,35.33,41.34)

par(mfrow = c(2, 2))
plot(x = c(1:length(y)), y = y)
hist(y)
descdist(y, discrete = FALSE, boot = 100)
```

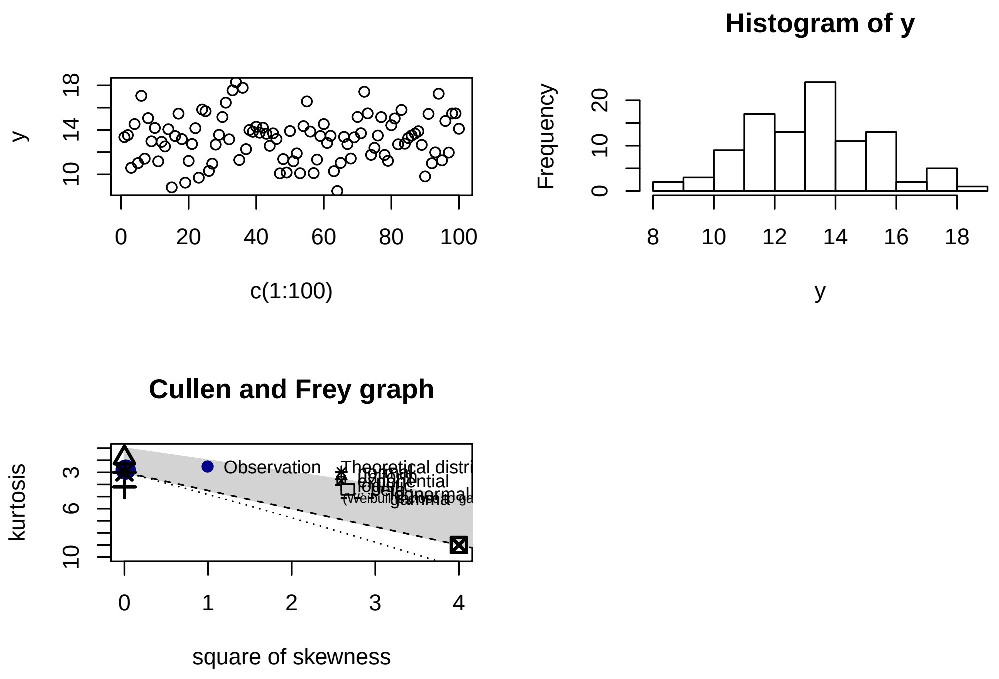
```
## summary statistics
## ------
## min:  16.62   max:  54.15
## median:  40.38
## mean:  40.28434
## estimated sd:  7.420034
## estimated skewness:  -0.551717
## estimated kurtosis:  3.565162
```



```r
# normally distributed data
y <- rnorm(100, 13, 2)
par(mfrow = c(2, 2))
plot(x = c(1:100), y = y)
hist(y)
```

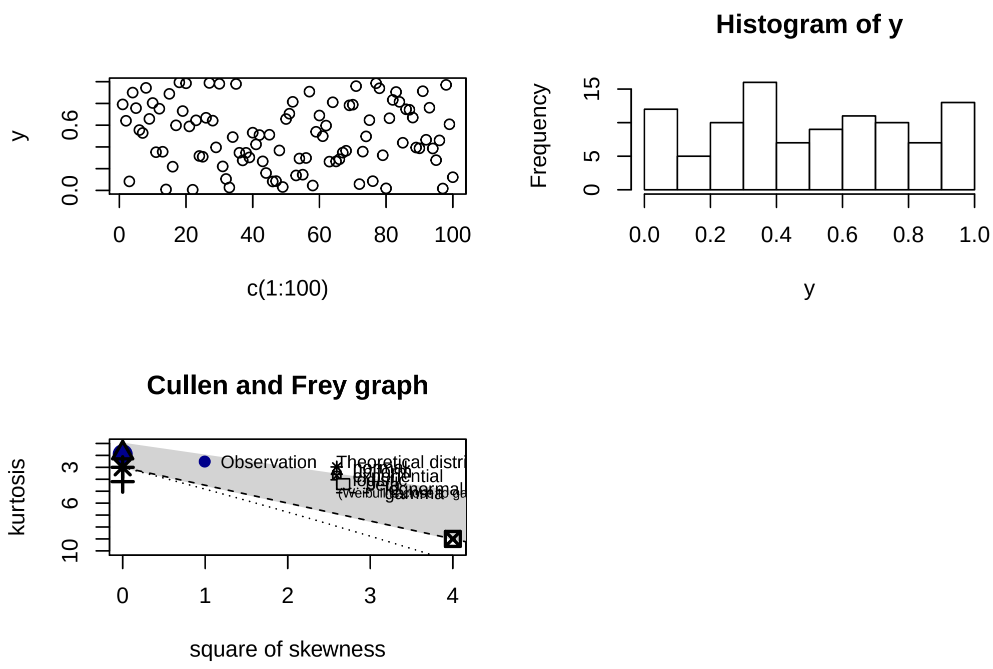
```
descdist(y, discrete = FALSE)
```

```
## summary statistics
## ------
## min:  8.503248   max:  18.29135
## median:  13.35981
## mean:  13.20023
## estimated sd:  2.085534
## estimated skewness:  0.1202547
## estimated kurtosis:  2.772078
```



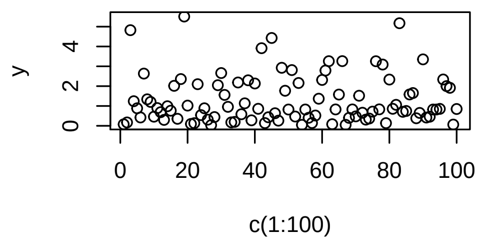
```
# uniformly distributed data
y <- runif(100)
par(mfrow = c(2, 2))
plot(x = c(1:100), y = y)
hist(y)
descdist(y, discrete = FALSE)
```

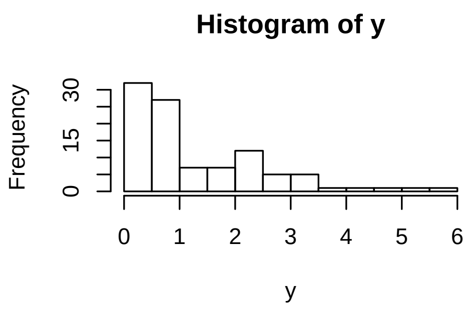
```
## summary statistics
## ------
## min:  0.005880884   max:  0.993874
## median:  0.5038945
## mean:  0.5082694
## estimated sd:  0.2943109
## estimated skewness:  -0.01172497
## estimated kurtosis:  1.898179
```

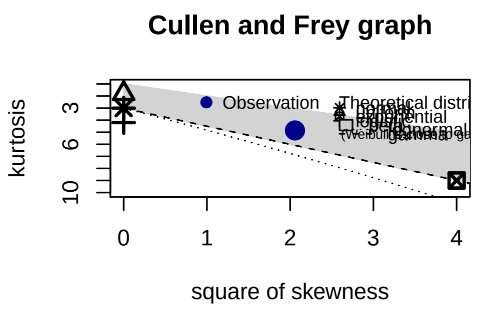


```r
# uniformly distributed data
y <- rexp(100, 0.7)
par(mfrow = c(2, 2))
plot(x = c(1:100), y = y)
hist(y)
descdist(y, discrete = FALSE)
```

```
## summary statistics
## ------
## min:  0.03435814   max:  5.51007
## median:  0.8386743
## mean:  1.295512
## estimated sd:  1.210403
## estimated skewness:  1.434539
## estimated kurtosis:  4.849173
```

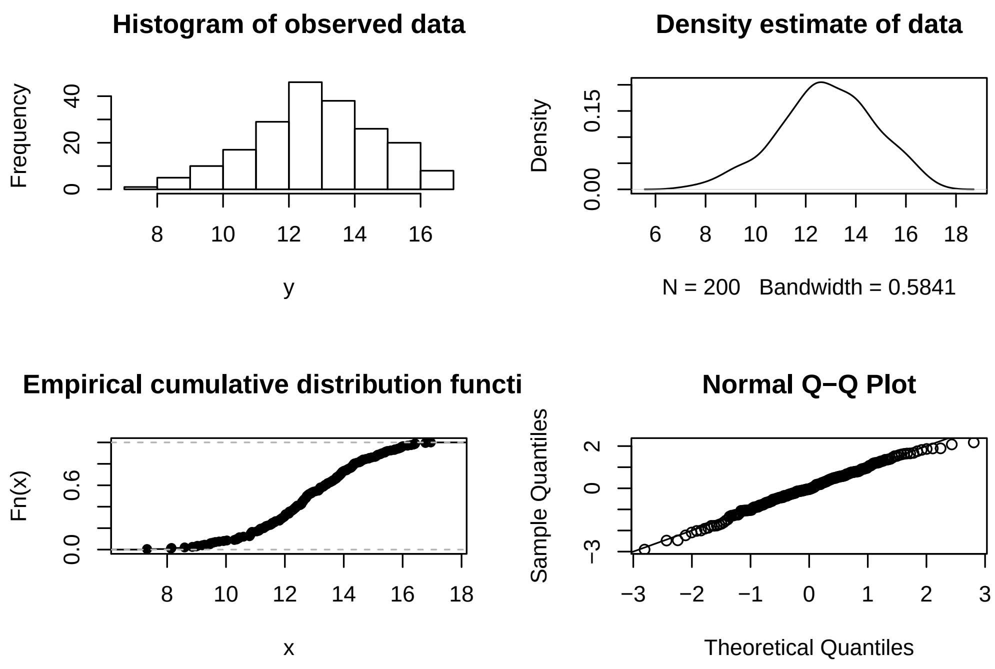






There is also a whole bunch of other approaches to use to try and identify the data distribution. Let us start with the gold standard first: normal data. We will demonstrate some visualisation approaches. The one that you already know is a basic histogram; it tells us something about the distribution's skewness, the tails, the mode(s) of the data, outliers, etc. Histograms can be compared to shapes associated with idealistic (simulated) distributions, as we will do here.

```r
y <-rnorm(n = 200, m = 13, sd = 2)
par(mfrow = c(2, 2))
# using some basic base graphics as ggplot2 is overkill;
# we can get a histogram using hist() statement
hist(y, main = "Histogram of observed data")
plot(density(y), main = "Density estimate of data")
plot(ecdf(y), main = "Empirical cumulative distribution function")
# standardise the data
z.norm <- (y - mean(y)) / sd(y)
# make a qqplot
qqnorm(z.norm)
# add a 45-degree reference line
abline(0, 1)
```



Above we have also added a diagonal line to the qqplot. If the sampled data come from the population with the chosen distribution, the points should fall approximately along this reference line. The greater the departure from this reference line, the greater the evidence for the conclusion that the data set have come from a population with a different distribution.

```
# curve(dnorm(100, m = 10, sd = 2), from = 0, to = 20, main = "Normal distribution")
# curve(dgamma(100, scale = 1.5, shape = 2), from = 0, to = 15, main = "Gamma distribution")
# curve(dweibull(100, scale = 2.5, shape = 1.5), from = 0, to = 15, main = "Weibull distribution")
```

5.4 Exercises

5.4.1 Exercise 1

Choose two different datasets and plot them as histograms with density curves overlayed. Label them with the distribution they appear to be and stitch them together with `ggarrange()`.

6 Inferences about one or two populations

```r
library(tidyverse)
library(plotly)
```

At the heart of many basic scientific inquiries is the simple question "Is A different from B?" The scientific notation for this question is:

- H0: Group A is not different from group B.
- H1: Group A is different from group B.

More formally, one would say:

1. $H_0 : \bar{A} = \bar{B}$ vs. the alternative hypothesis that $H_a : \bar{A} \neq \bar{B}$.
2. $H_0 : \bar{A} \leq \bar{B}$ vs. the alternative hypothesis that $H_a : \bar{A} > \bar{B}$.
3. $H_0 : \bar{A} \geq \bar{B}$ vs. the alternative hypothesis that $H_a : \bar{A} < \bar{B}$.

   NOTE: Hypothesis 1 is a two-sided t-test and hypotheses 2 and 3 are one-sided tests.

Biologists typically define the probability of one in twenty (0.05) as the cutoff level to reject the null hypothesis.

To answer this fundamental question one often uses a t-test. There are several variations of t-tests, depending on the nature of our samples and the type of question being asked:

- **One-sample t-tests:** only one sample set of data that we wish to compare against a known population mean:
  - one-sided one-sample t-tests
  - two-sided one-sample t-tests
- **Two-sample t-tests:** the means of two groups are compared against each other:
  - independent sample t-tests
    - one-sided two-sample t-tests

        * two-sided two-sample t-tests
    - paired sample t-tests
        * one-sided
        * two-sided

Before we cover each of these, we need to understand some of the assumptions behind t-tests. We shall cover that next.

6.1 Assumptions

Irrespective of the kind of t-test, we have to make a couple of important assumptions that are *not* guaranteed to be true. In fact, these assumptions are often violated because real data, especially biological data, are messy. In order to use a t-test to determine if a significant difference between two sample sets of data exists we must first establish that:

- the dependent variable must be continuous (i.e. it is measured at the interval or ratio level),
- the observations in the groups being compared are independent of each other,
- the data are **normally distributed**, and
- that the data are **homoscedastic**, and in particular, that there are no outliers.

6.1.1 Normality

Remember from Chapter 5 what a normal distribution is/looks like? Let's have a peek below to remind ourselves:

```
# Random normal data
set.seed(666)
r_dat <- data.frame(dat = c(rnorm(n = 1000, mean = 10, sd = 3),
                            rnorm(n = 1000, mean = 8, sd = 2)),
                    sample = c(rep("A", 1000), rep("B", 1000)))

# Create histogram
h <- ggplot(data = r_dat, aes(x = dat, fill = sample)) +
  geom_histogram(position = "dodge", binwidth = 1, alpha = 0.8) +
  geom_density(aes(y = 1*..count.., fill = sample), colour = NA, alpha = 0.4) +
  labs(x = "value")
h
```

Whereas histograms may be a pretty way to check the normality of our data, there is actually a statistical test for this, which is preferable to a visual inspection alone. But remember that you should *always* visualise your data before performing any statistics on them. To check the normality of the data we use the Shapiro-Wilk test. This test produces a W value and a p-value. We are only interested in the p-value as this is how we are to determine the normality of the data. The Shapiro–Wilk test tests the null hypothesis that a sample $x_1, ..., x_n$ comes from a normally distributed population. i.e. that the normality *does not* differ significantly from normal. If the p-value is **above** 0.05 we may assume the data to be normally distributed. In order to demonstrate what the output of `shapiro.test()` looks like we will run it on all of the random data we generated.

```
shapiro.test(r_dat$dat)
```

```
R>
```

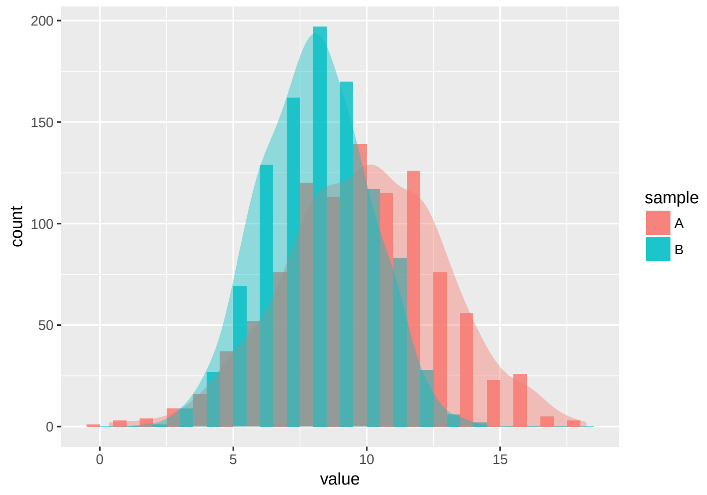


Figure 6.1: Interactive histogram showing two randomly generated normal distributions.

```
R>  Shapiro-Wilk normality test
R>
R> data:  r_dat$dat
R> W = 0.9942, p-value = 4.649e-07
```

Note that this shows that the data are *not* normally distributed. This is because we have incorrectly run this function simultaneously on two different samples of data. To perform this test correctly, and in the tidy way, we need to select only the second piece of information from the `shapiro.test()` output and ensure that it is presented as a numeric value:

```
# we use the square bracket notation to select only the p-value;
# had we used `[1]` we'd have gotten W
r_dat %>%
  group_by(sample) %>%
  summarise(norm_dat = as.numeric(shapiro.test(dat)[2]))
```

```
R> # A tibble: 2 x 2
R>   sample norm_dat
R>   <fct>     <dbl>
R> 1 A         0.375
R> 2 B         0.461
```

Now we see that our two sample sets are indeed normally distributed.

What if we find that the data are not normally distributed? Although there are many options, the easiest is to perform a Wilcoxon Rank Sum test, which is the non-parametric equivalent to a *t*-test (see Section X). Another option is to transform the data (Chapter 11).

6.1.2 Homoscedasticity

Besides requiring that our data are normally distributed, we must also ensured that they are homoscedastic. This word means that the scedasticity (variance) of things are homogeneous (similar). In practical terms this means that the variance of the samples we are comparing should not be more than two to four times greater than one another. In R, we use the function `var()` to check the variance in a sample:

```
r_dat %>%
  group_by(sample) %>%
  summarise(sample_var = var(dat))
```

```
R> # A tibble: 2 x 2
R>   sample sample_var
R>   <fct>       <dbl>
R> 1 A            8.72
R> 2 B            3.97
```

Above we see that the variance of our two samples are homoscedastic because the variance of one is not more than two to four times greater than the other.

What if our data are not equal in their variances? This is easier to fix as the solution is built right into the *t*-test function; all we need to do is to perform Welch Two Sample *t*-test (the default) in the `t.test()` function that we shall use below. If the variances are equal, we simply perform a normal Student's *t*-test by setting the argument `var.equal = TRUE` in the function call (see below).

6.1.3 Two for one

Because these two assumptions of normality and homoscedasticty are performed in tandem with one another, it is helpful to have a function that checks for both simultaneously. Below we see how just such a function would be written:

```
two_assum <- function(x) {
  x_var <- var(x)
  x_norm <- as.numeric(shapiro.test(x)[2])
  result <- c(x_var, x_norm)
  return(result)
}
```

To use our new function in a tidy way we use the following code:

```
r_dat %>%
  group_by(sample) %>%
  summarise(sample_var = two_assum(dat)[1],
            sample_norm = two_assum(dat)[2])
```

```
R> # A tibble: 2 x 3
R>   sample sample_var sample_norm
R>   <fct>       <dbl>       <dbl>
R> 1 A            8.72       0.375
R> 2 B            3.97       0.461
```

Do these data meet our assumptions? How do we know this?

Once we have tested our assumptions we may perform a *t*-test to ascertain whether or not our samples are significantly different from one another. The base R function for this is `t.test()`; however, by utilising the **ggpubr** package we gain access to `compare_means()`, which allows us to perform any sort of test that compares sample sets of data and outputs the results as a dataframe. We will see throughout this and the following chapters why this is so useful.

```
library(ggpubr)
```

6.2 One-sample *t*-tests

Generally when we use a t-test it will be a two-sample t-test (see below). Occasionally, however, we may have only one sample set of data that we wish to compare against a known population mean, which is generally denoted as μ, or `mu` in the function call to the t-test in R:

```
# create a single sample of random normal data
set.seed(666)
r_one <- data.frame(dat = rnorm(n = 20, mean = 20, sd = 5),
                    sample = "A")

# check normality
shapiro.test(r_one$dat)
```

```
R>
R>  Shapiro-Wilk normality test
R>
R> data:  r_one$dat
R> W = 0.94911, p-value = 0.3538
```

```
# No variance to compare
# ...

# compare random data against a population mean of 20
t.test(r_one$dat, mu = 20)
```

```
R>
R>  One Sample t-test
R>
R> data:  r_one$dat
R> t = 0.0048653, df = 19, p-value = 0.9962
R> alternative hypothesis: true mean is not equal to 20
R> 95 percent confidence interval:
R>  16.91306 23.10133
R> sample estimates:
R> mean of x
R>  20.00719
```

```
# compare random data against a population mean of 30
t.test(r_one$dat, mu = 30)
```

```
R>
R>  One Sample t-test
R>
R> data:  r_one$dat
R> t = -6.7596, df = 19, p-value = 1.858e-06
R> alternative hypothesis: true mean is not equal to 30
R> 95 percent confidence interval:
R>  16.91306 23.10133
R> sample estimates:
R> mean of x
R>  20.00719
```

What do the results of these two different tests show? Let's visualise these data to get a better

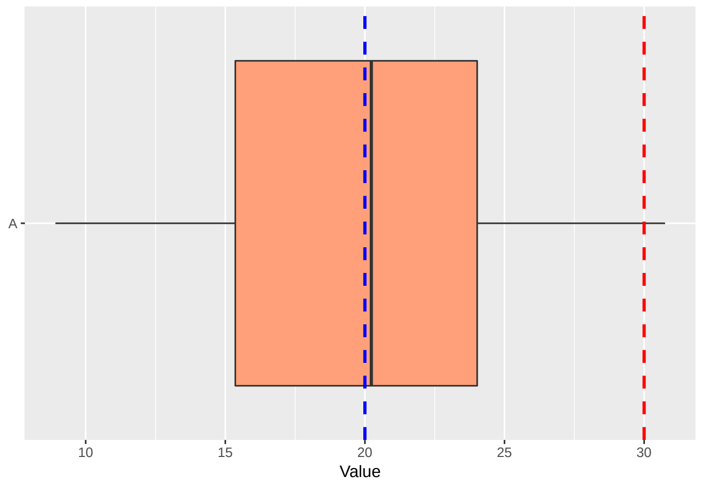


Figure 6.2: Boxplot of random normal data with. A hypothetical population mean of 20 is shown as a blue line, with the red line showing a mean of 30.

understanding.

```r
ggplot(data = r_one, aes(y = dat, x = sample)) +
  geom_boxplot(fill = "lightsalmon") +
  # population  mean (mu) = 20
  geom_hline(yintercept = 20, colour = "blue",
             size = 1, linetype = "dashed") +
  # population  mean (mu) = 30
  geom_hline(yintercept = 30, colour = "red",
             size = 1, linetype = "dashed") +
  labs(y = "Value", x = NULL) +
  coord_flip()
```

The boxplot above shows the distribution of our random data against two potential population means. Does this help now to illustrate the results of our one-sample *t*-tests?

6.2.1 One-sided one-sample *t*-tests

As we may remember from Chapter 5, a distribution has two tails. When we are testing for significance we are generally looking for a result that lays in the far end of either of these tails. Occasionally, however, we may want to know if the result lays specifically in one of the tails. Explicitly the leading or trailing tail. For example, is the mean value of our sample population significantly *greater than* the value μ? Or, is the mean value of our sample population significantly *less than* the value μ? To specify this in R we must add an argument as seen below:

```r
# check against the trailing tail
t.test(r_one$dat, mu = 30, alternative = "less")
```

```
R>
R>  One Sample t-test
R>
R> data:  r_one$dat
R> t = -6.7596, df = 19, p-value = 9.292e-07
R> alternative hypothesis: true mean is less than 30
```

```
R> 95 percent confidence interval:
R>      -Inf 22.56339
R> sample estimates:
R> mean of x
R>  20.00719
```

```
# check against the leading tail
t.test(r_one$dat, mu = 30, alternative = "greater")
```

```
R>
R>  One Sample t-test
R>
R> data:  r_one$dat
R> t = -6.7596, df = 19, p-value = 1
R> alternative hypothesis: true mean is greater than 30
R> 95 percent confidence interval:
R>  17.451    Inf
R> sample estimates:
R> mean of x
R>  20.00719
```

Are these the results we would have expected? Why does the second test not return a significant result?

TASK: Create a visualisation to graphically demonstrate the outcome of this *t*-test.

6.2.2 Two-sided one-sample *t*-tests

In R, the default setting for any comparison of means test is that it is two-sided so we do not need to state this explicitly. For the sake of thoroughness let's see how to do this below. Note that the results for the two following tests are identical:

```
# R assumes we want a to-sided test
t.test(r_one$dat, mu = 30)
```

```
R>
R>  One Sample t-test
R>
R> data:  r_one$dat
R> t = -6.7596, df = 19, p-value = 1.858e-06
R> alternative hypothesis: true mean is not equal to 30
R> 95 percent confidence interval:
R>  16.91306 23.10133
R> sample estimates:
R> mean of x
R>  20.00719
```

```
# but we can be explicit as we choose
t.test(r_one$dat, mu = 30, alternative = "two.sided")
```

```
R>
R>  One Sample t-test
R>
R> data:  r_one$dat
R> t = -6.7596, df = 19, p-value = 1.858e-06
```

```
R> alternative hypothesis: true mean is not equal to 30
R> 95 percent confidence interval:
R>  16.91306 23.10133
R> sample estimates:
R> mean of x
R>  20.00719
```

6.3 Two-sample *t*-tests

A two-sample *t*-test is used when we have samples from two different populations that we would like to compare against one another. This is the most common use of a *t*-test. Two sample *t*-tests can accommodate samples with equal variances or samples with unequal variances (as determined by the test for heteroscedasticity that we performed earlier).

In the case of samples that share the same variance we perform a classical *t*-test (otherwise known as Student's *t*-test); the equation to calculate the *t*-statistic is this one:

$$t = \frac{\bar{A} - \bar{B}}{\sqrt{\frac{S^2}{n} + \frac{S^2}{m}}}$$

$\bar{A}$ and $\bar{B}$ are the means for groups A and B, respectively; n and m are the sample sizes of the two sets of samples, respectively; and S^2 is the pooled variance, which is calculated as:

$$S^2 = \frac{(n-1)S_A^2 + (m-1)S_B^2}{n+m-2}$$

The degrees of freedom, $d.f.$, in the equation for the shared variance is $n_A + n_B - 2$.

When variances are unequal we perform the Welch's *t*-test; Welch's *t*-statistics is calculated as per this equation:

$$t = \frac{\bar{A} - \bar{B}}{\sqrt{\frac{S_A^2}{n} + \frac{S_B^2}{m}}}$$

Above, S_A and S_B are the variances of groups A and B, respectively (see Section X). The degrees of freedom to use with Welch's *t*-test is obtained using the Welch–Satterthwaite equation as:

$$d.f. = \frac{\left(\frac{S_A^2}{n} + \frac{S_B^2}{m}\right)^2}{\left(\frac{S_A^4}{n-1} + \frac{S_B^4}{m-1}\right)}$$

What do we do with this t-statistic? In the olden days we had to calculate the t-statistics and the $d.f.$ by hand. These two values, the $d.f.$ and t-value had to be read off a table of pre-calculated t-values, probabilities and degrees of freedom as in here[1]. Luckily, the *t*-test function nowadays does this all automagically. But if you are feeling nostalgic over times that you have sadly never experienced, please calculate the t-statistic and the $d.f.$ yourself and give the table a go.

[1] https://home.ubalt.edu/ntsbarsh/Business-stat/StatistialTables.pdf

Back to the present day and the wonders of modern technology. Let's generate some new random normal data and test to see if the data belonging to the two groups differ significantly from one-another. First, we apply the *t*-test function as usual:

```
# random normal data
set.seed(666)
r_two <- data.frame(dat = c(rnorm(n = 20, mean = 4, sd = 1),
                            rnorm(n = 20, mean = 5, sd = 1)),
                    sample = c(rep("A", 20), rep("B", 20)))

# perform t-test
# note how we set the `var.equal` argument to TRUE because we know
# our data has the same SD (they are simulated as such!)
t.test(dat ~ sample, data = r_two, var.equal = TRUE)
```

```
R>
R>  Two Sample t-test
R>
R> data:  dat by sample
R> t = -1.9544, df = 38, p-value = 0.05805
R> alternative hypothesis: true difference in means is not equal to 0
R> 95 percent confidence interval:
R>  -1.51699175  0.02670136
R> sample estimates:
R> mean in group A mean in group B
R>        4.001438        4.746584
```

```
# if the variances are not equal, simply set `var.equal` to false
# and a Welch's t-test will be performed
```

The first argument we see in `t.test()` is `dat ~ sample`. Usually in R when we see a ~ (tilde) we are creating what is known as a formula. A formula tells R how it should look for interactions between data and factors. For example `Y ~ X` reads: Y as a function of X. In our code above we see `dat ~ sample`. This means we are telling R that the *t*-test we want it to perform is when the `dat` column is a function of the `sample` column. In plain English we are dividing up the `dat` column into the two different samples we have, and then running a *t*-test on these samples. Another way of stating this is that the value of `dat` depends on the grouping it belong to (`A` or `B`). We will see this same formula notation cropping up later under ANOVAs, linear models, etc.

Now that we have seen the nitty gritty of how a t-test is meant to work, click here[2] to watch a visualisation that demonstrates how the relationships between two different sample sets (based on their mean and variance) influence the results.

TASK: Create a visualisation to graphically demonstrate the outcome of this *t*-test.

Now we do the same test using a convenient function that comes with the **ggpubr** package, called `compare_means()`, to perform the same *t*-test:

```
# perform t-test using `compare_means()`
# note that compare_means() takes the same arguments as t.test()
compare_means(dat ~ sample, data = r_two, method = "t.test", var.equal = TRUE)
```

```
R> # A tibble: 1 x 8
```

2

```
R>   .y.   group1 group2      p  p.adj p.format p.signif method
R>   <chr> <chr>  <chr>   <dbl>  <dbl> <chr>    <chr>    <chr>
R> 1 dat   A      B      0.0580 0.0580 0.058    ns       T-test
```

Note above that in order to tell `compare_means()` to perform a t-test we feed it the argument `method = "t.test"`. The output is similar to that of the familiar `t.test()` function that we used earlier, but the output is more abbreviated and less useful. Typically, the output of the t-tests that we need to report in the results sections of our papers include the t-statistic, the P-value, and the degrees of freedom, $d.f.$, and these are absent from the `compare_means()` function's output.

6.3.1 One-sided two-sample *t*-tests

Just as with the one-sample t-tests above, we may also specify which tail of the distribution we are interested in when we compare the means of our two samples. We do so by providing the same arguments as previously:

```
# is the mean of sample B smaller than that of sample A?
compare_means(dat ~ sample, data = r_two, method = "t.test", var.equal = TRUE, alternative = "less")
```

```
R> # A tibble: 1 x 8
R>   .y.   group1 group2     p p.adj p.format p.signif method
R>   <chr> <chr>  <chr>  <dbl> <dbl> <chr>    <chr>    <chr>
R> 1 dat   A      B      0.971 0.971 0.97     ns       T-test
```

```
# is the mean of sample B greater than that of sample A?
compare_means(dat ~ sample, data = r_two, method = "t.test", var.equal = TRUE, alternative = "greater")
```

```
R> # A tibble: 1 x 8
R>   .y.   group1 group2      p  p.adj p.format p.signif method
R>   <chr> <chr>  <chr>   <dbl>  <dbl> <chr>    <chr>    <chr>
R> 1 dat   A      B      0.0290 0.0290 0.029    *        T-test
```

What do these results show? Is this surprising?

6.3.2 Two-sided two-sample *t*-tests

Again, as stated above, the default setting in R for comparisons of means is that the test is two-sided. If one wants to state this explicitly it may be done as previously. Note that the results are identical.

```
# default settings
compare_means(dat ~ sample, data = r_two, method = "t.test")
```

```
R> # A tibble: 1 x 8
R>   .y.   group1 group2      p  p.adj p.format p.signif method
R>   <chr> <chr>  <chr>   <dbl>  <dbl> <chr>    <chr>    <chr>
R> 1 dat   A      B      0.0584 0.0584 0.058    ns       T-test
```

```
# explicitly state a two-sided test
compare_means(dat ~ sample, data = r_two, method = "t.test", alternative = "two.sided")
```

```
R> # A tibble: 1 x 8
R>   .y.   group1 group2      p  p.adj p.format p.signif method
R>   <chr> <chr>  <chr>   <dbl>  <dbl> <chr>    <chr>    <chr>
R> 1 dat   A      B      0.0584 0.0584 0.058    ns       T-test
```

6.4 Paired *t*-tests

Paired *t*-tests are done when comparing matched samples, and in other words, when our second assumption of *t*-tests is violated: the observations are independent of one another—in paired samples, clearly they are not independent. This test is also sometimes called a dependent sample *t*-test.

For example, we design a survey to determine if, in a group of 20 people, individuals' right arms differ in length from that of their left arms. For person A, we measure her right arm and her left arm. For person B we measure his right arm and his left arm. So we go all the way to person 20. A right arm belonging with one individual is always matched against a left arm in the same individual. The samples are paired so we use a paired *t*-test. Another example: we follow the growth of a sample of 20 piglets over three weeks to see if they weigh more after three weeks than they did at the start of the assessment period. We measure the first piglet, named Halal, at the start of the three week period and again after. We do the same for the second piglet, Kosher. And so it goes. Each piglet has a paired set of measurements, one before matched with one after. In both these examples the data in the two groups (left arm and right arm; or before and after) are not independent, so we need to account for this in the analysis. In practice, how do we perform such a *t*-test? Who can think of a dataset we've used in the past that we would use a paired *t*-test for?

```
compare_means(dat ~ sample, data = r_two, method = "t.test", paired = TRUE)
```

```
R> # A tibble: 1 x 8
R>   .y.   group1 group2      p  p.adj p.format p.signif method
R>   <chr> <chr>  <chr>   <dbl>  <dbl> <chr>    <chr>    <chr>
R> 1 dat   A      B      0.0391 0.0391 0.039    *        T-test
```

6.5 Comparison of two population proportions

All of the tests we covered above are designed to deal with continuous data, such as fish lengths or chlorophyll content. If we want to compare proportions (probabilities of success) of different samples against each other, or some known population mean, we need `prop.test()`. Let's create a dummy dataset to get a better idea of how this function works. Below we create some data showing the result of placing two different subjects, Jack and Jill, in separate sealed rooms for two hours (120 minutes). Once every minute a mosquito is let into the room before being extracted again. The columns `yes` and `no` show if the mosquito bit the subject during that one minute. Who says science can't be fun!

```
mosquito <- matrix(c(70, 85, 50, 35), ncol = 2)
colnames(mosquito) <- c("yes", "no")
rownames(mosquito) <- c("Jack", "Jill")
mosquito
```

```
R>      yes no
R> Jack  70 50
R> Jill  85 35
```

6.5.1 One-sample and two-sample tests

As with *t*-tests, proportion tests may also be based on one sample, or two. If we have only one sample we must specify the total number of trials as well as what the expected population probability of success is. Because these are individual values, and not matrices, we will show what this would look like without using any objects but will rather give each argument within `prop.test()` a single exact value. In the arguments within `prop.test()`, `x` denotes the number of successes recorded, `n` shows the total number of individual trials performed, and `p` is the expected probability. It is easiest to consider this as though it were a series of 100 coin tosses.

```r
# When the probability matches the population
prop.test(x = 45, n = 100, p = 0.5)
```

```
R> 
R>  1-sample proportions test with continuity correction
R> 
R> data:  45 out of 100, null probability 0.5
R> X-squared = 0.81, df = 1, p-value = 0.3681
R> alternative hypothesis: true p is not equal to 0.5
R> 95 percent confidence interval:
R>  0.3514281 0.5524574
R> sample estimates:
R>    p 
R> 0.45
```

```r
# When it doesn't
prop.test(x = 33, n = 100, p = 0.5)
```

```
R> 
R>  1-sample proportions test with continuity correction
R> 
R> data:  33 out of 100, null probability 0.5
R> X-squared = 10.89, df = 1, p-value = 0.0009668
R> alternative hypothesis: true p is not equal to 0.5
R> 95 percent confidence interval:
R>  0.2411558 0.4320901
R> sample estimates:
R>    p 
R> 0.33
```

If we have two samples that we would like to compare against one another we enter them into the function as follows:

```r
# NB: Note that the `mosquito` data are a matrix, NOT a data.frame
prop.test(mosquito)
```

```
R> 
R>  2-sample test for equality of proportions with continuity
R>  correction
R> 
R> data:  mosquito
R> X-squared = 3.5704, df = 1, p-value = 0.05882
R> alternative hypothesis: two.sided
R> 95 percent confidence interval:
R>  -0.253309811  0.003309811
R> sample estimates:
R>    prop 1    prop 2 
R> 0.5833333 0.7083333
```

Do mosquito's bite Jack and Jill at different proportions?

6.5.2 One-sided and two-sided tests

AS with all other tests that compare values, proportion tests may be specified as either one or two-sided. Just to be clear, the default setting for `prop.test()`, like everything else, is a two-sided test. See code below to confirm that the results are identical with or without the added argument:

```r
# Default
prop.test(mosquito)
```

```
R>
R>  2-sample test for equality of proportions with continuity
R>  correction
R>
R> data:  mosquito
R> X-squared = 3.5704, df = 1, p-value = 0.05882
R> alternative hypothesis: two.sided
R> 95 percent confidence interval:
R>  -0.253309811  0.003309811
R> sample estimates:
R>    prop 1    prop 2
R> 0.5833333 0.7083333
```

```r
# Explicitly state two-sided test
prop.test(mosquito, alternative = "two.sided")
```

```
R>
R>  2-sample test for equality of proportions with continuity
R>  correction
R>
R> data:  mosquito
R> X-squared = 3.5704, df = 1, p-value = 0.05882
R> alternative hypothesis: two.sided
R> 95 percent confidence interval:
R>  -0.253309811  0.003309811
R> sample estimates:
R>    prop 1    prop 2
R> 0.5833333 0.7083333
```

Should we want to specify only one of the tails to be considered, we do so precisely the same as with t-tests. Below are examples of what this code would look like:

```r
# Jack is bit less than Jill
prop.test(mosquito, alternative = "less")
```

```
R>
R>  2-sample test for equality of proportions with continuity
R>  correction
R>
R> data:  mosquito
R> X-squared = 3.5704, df = 1, p-value = 0.02941
R> alternative hypothesis: less
```

```
R> 95 percent confidence interval:
R>  -1.00000000 -0.01597923
R> sample estimates:
R>    prop 1    prop 2
R> 0.5833333 0.7083333
```

```
# Jack is bit more than Jill
prop.test(mosquito, alternative = "greater")
```

```
R>
R>  2-sample test for equality of proportions with continuity
R>  correction
R>
R> data:  mosquito
R> X-squared = 3.5704, df = 1, p-value = 0.9706
R> alternative hypothesis: greater
R> 95 percent confidence interval:
R>  -0.2340208  1.0000000
R> sample estimates:
R>    prop 1    prop 2
R> 0.5833333 0.7083333
```

Do these results differ from the two-sided test? What is different?

6.6 A *t*-test workflow

Now that we have seen how to compare the means of two sample sets of data, let's see what that complete workflow would look like in R. For this example we will use the `ecklonia` data from Intro R Workshop: Data Manipulation, Analysis and Graphing[3].

6.6.1 Loading data

Before we can run any analyses we will need to load our data. We are also going to convert these data from their wide format into a long format because this is more useful for the rest of our workflow.

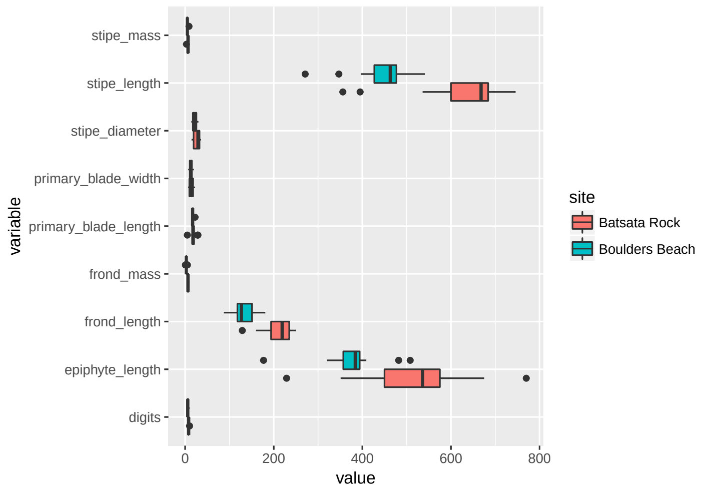
```
ecklonia <- read_csv("data/ecklonia.csv") %>%
  gather(key = "variable", value = "value", -species, -site, -ID)
```

6.6.2 Visualising data

With our data loaded, let's visualise them in order to ensure that these are indeed the data we are after. Visualising the data will also help us to formulate a hypothesis.

```
ggplot(data = ecklonia, aes(x = variable, y = value, fill = site)) +
  geom_boxplot() +
  coord_flip()
```

[3] https://robwschlegel.github.io/Intro_R_Workshop/



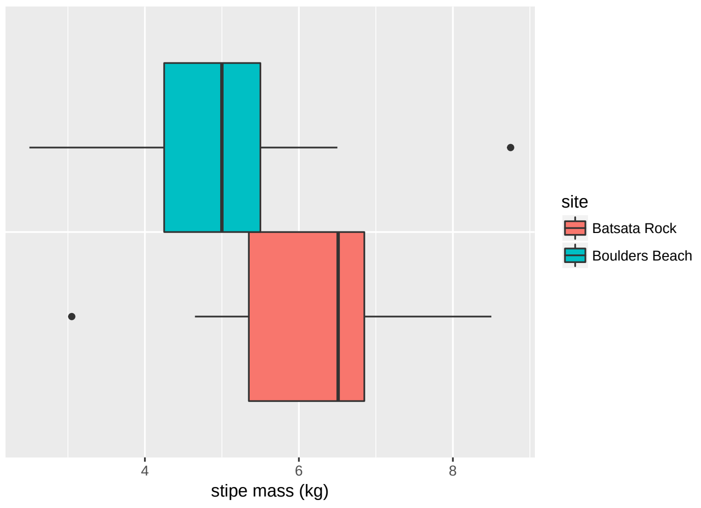
\begin{figure} \caption{Boxplots showing differences in morphometric properties of the kelp *Ecklonia maxima* at two sites in False Bay.} \end{figure}

The first thing we should notice from the figure above is that our different measurements are on very different scales. This makes comparing all of our data visually rather challenging. Even given this complication, one should readily be able to make out that the measurement values at Batsata Rock appear to be greater than at Boulders Beach. Within the framework of the scientific process, that is what we would call an "observation", and is the first step towards formulating a hypothesis. The next step is to refine our observation into a hypothesis. By what measurement are the kelps greater at one site than the other?

6.6.3 Formulating a hypothesis

Looking at the figure above it appears that for almost all measurements of length, Batsata Rock far exceeds that of Boulders Beach however, the stipe masses between the two sites appear to be more similar. Let's pull out just this variable and create a new boxplot.

```r
# filter the data
ecklonia_sub <- ecklonia %>%
  filter(variable == "stipe_mass")

# then create a new figure
ggplot(data = ecklonia_sub, aes(x = variable, y = value, fill = site)) +
  geom_boxplot() +
  coord_flip() +
  labs(y = "stipe mass (kg)", x = "") +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank())
```



\begin{figure} \caption{Boxplots showing the difference in stipe mass (kg) of the kelp *Ecklonia maxima* at two sites in False Bay.} \end{figure}

Now we have a more interesting comparison at hand. The question I think of when I look at these data is "Are the stipe masses at Batsata Rock greater than at Boulders Beach?". The hypothesis necessary to answer this question would look like this:

- H0: Stipe mass at Batsata Rock **is not** greater than at Boulders Beach.
- H1: Stipe mass at Batsata Rock **is** greater than at Boulders Beach.

Or more formally:

- $H_0 : \bar{A} \leq \bar{B}$
- $H_a : \bar{A} > \bar{B}$.

Which test must we use for this hypothesis?

6.6.4 Choosing a test

Before we can pick the correct statistical test for our hypothesis, we need to be clear on what it is we are asking. Starting with the data being used is usually a good first step. As we may see in the above figure, we have two sample sets that we are comparing. Therefore, unsurprisingly, we will likely be using a t-test. But we're not done yet. How is it that we are comparing these two sample sets? Remember from the examples above that there are multiple different ways to compare two sets of data. For our hypothesis we want to see if the stipe mass at Batsata Rock is **greater than** the stipe mass at Boulders Beach, not just that they are different. Because of this we will need a one-sided t-test. But wait, there's more! We've zeroed in on which sort of test would be appropriate for our hypothesis, but before we run it we need to check our assumptions.

6.6.5 Checking assumptions

In case we forgot, here are the assumptions for a t-test:

- the dependent variable must be continuous (i.e. it is measured at the interval or ratio level),
- the observations in the groups being compared are independent of each other,
- the data are **normally distributed**, and

- that the data are **homoscedastic**, and in particular, that there are no outliers.

We know that the first two assumptions are met because our data are measurements of mass at two different sites. Before we can run our one-sided *t*-test we must meet the last two assumptions. Lucky us, we have a function tat will do that automagically.

```
ecklonia_sub %>%
  group_by(site) %>%
  summarise(stipe_mass_var = two_assum(value)[1],
            stipe_mass_norm = two_assum(value)[2])
```

```
R> # A tibble: 2 x 3
R>   site           stipe_mass_var stipe_mass_norm
R>   <chr>                   <dbl>           <dbl>
R> 1 Batsata Rock             2.00           0.813
R> 2 Boulders Beach           2.64           0.527
```

Lovely. On to the next step.

6.6.6 Running an analysis

With our assumptions checked, we may now analyse our data. We'll see below how to do this with both of the functions we've learned in this chapter for comparing means of two sample sets.

```
# traditional output
t.test(value ~ site, data = ecklonia_sub, var.equal = TRUE, alternative = "greater")
```

```
R> 
R>  Two Sample t-test
R> 
R> data:  value by site
R> t = 1.8741, df = 24, p-value = 0.03657
R> alternative hypothesis: true difference in means is greater than 0
R> 95 percent confidence interval:
R>  0.09752735        Inf
R> sample estimates:
R>   mean in group Batsata Rock mean in group Boulders Beach 
R>                     6.116154                     4.996154
```

```
# dataframe output
compare_means(value ~ site, data = ecklonia_sub, method = "t.test", var.equal = TRUE, alternative = "greater")
```

```
R> # A tibble: 1 x 8
R>   .y.   group1         group2            p  p.adj p.format p.signif method
R>   <chr> <chr>          <chr>         <dbl>  <dbl> <chr>    <chr>    <chr> 
R> 1 value Boulders Beach Batsata Rock 0.0366 0.0366 0.037    *        T-test
```

6.6.7 Interpreting the results

We may reject the null hypothesis, that the stipe mass of kelps at Batsata Rock are not greater than at Boulders Beach, if our *t*-test returns a *p*-value ≤ 0.05. We must also pay attention to some of the other results from our *t*-test, specifically the *t*-value (t) and the degrees of freedom (df) as these are also needed when we are writing up our results. From all of the information above, we may accept the alternative hypothesis. But how do we write that up?

6.6.8 Drawing conclusions

There are many ways to present ones findings. Style, without *too much* flourish, is encouraged as long as certain necessary pieces of information are provided. The sentence below is a very minimalist example of how one may conclude this mini research project. A more thorough explanation would be desirable.

> The stipe mass (kg) of the kelp *Ecklonia maxima* was found to be significantly greater at Batsata Rock than at Boulders Beach ($p = 0.03$, $t = 1.87$, df = 24).

6.6.9 Going further

But why though? As is often the case in life, and science is no exception, answers to our questions just create even more questions! Why would the mass of kelp stipes at one locations in the same body of water and only a kilometre or so apart be significantly different? It looks like we are going to need to design a new experiment… Masters thesis anyone?

6.7 Exercises

6.7.1 Exercise 1

Find or create your own normally distributed data and think of a hypothesis you could use a t-test for. Write out the hypothesis, test it, and write a one sentence conclusion for it. Provide all of the code used to accomplish this.

6.7.2 Exercise 2

Do the same as Exercise 1, but for probability data.

7 ANOVA

Whole big books have been written about Analysis of Variance (ANOVA). Although there are many ANOVA experimental designs available, biologists are taught to pay special attention to the design of experiments, and generally make sure that the experiments are fully factorial (in the case of two-way or higher ANOVAs) and balanced. For this reason we will focus in this Introductory Statistics course on one-way and factorial ANOVAs only.

As *t*-tests, ANOVAs require that some assumptions are met:

- Normally distributed data
- Homogeneity of variances
- Independence of data
- In our case, we will encourage also that the data are balanced

If some of the above assumptions are violated, then your course of action is either to transform the data (if non-normal) or to use a generalised linear model (also when non-normal), or to use a linear mixed model (when the assumption on non-independence cannot be guaranteed). We will get to some of these methods in later chapters. Linked to the above, ANOVAs are also sensitive to the presence of outliers (see our earlier discussion about the mean and how it differs from the median), so we need to ensure that outliers are not present (they can be removed, and there are many ways of finding them and eliminating them). If outliers are an important feature of the data, then a non-parametric test can be used, or some other test that works well with extreme values can be applied.

Rather than talking about *t*-tests and ANOVAs as separate things, let us acknowledge that they are similar ways of asking the same question. That question being, are the means of these two or more things we want to compare different, or the same? At this stage it is important to note that the independent variable is categorical (i.e. a factor denoting two or more different treatments or sampling conditions) and that the dependent variable is continuous. You may perhaps be more familiar with this question when it is presented as a set of hypotheses.

H0: Group A is not different from group B.

H1: Group A is different from group B.

This is a scientific question in the simplest sense. Often, for basic inquiries such as that posed above, we need to see if one group differs significantly from another. The way in which we accomplish this is by looking at the mean and variance within a set of data compared against another similar set. In order to do so appropriately however we need to first assume that both sets of data are normally distributed, and that the variance found within each set of data is similar. These are the two primary assumptions we learned about in Chapter 6, and if they are met then we may use parametric tests. We will learn in Chapter 9 what we can do if these assumptions are not meant and we cannot adequately transform our data, meaning we will need to use non-parametric tests.

7.1 Remember the *t*-test

As you know, a *t*-test is used when we want to compare two different sample sets against one another. This is also known as a two-factor or two level test. When one wants to compare multiple (more than two) sample sets against one another an ANOVA is required (see below). Remember how to perform a *t*-test in R: we will revisit this test using the `chicks` data, but only for Diets 1 and 2 from day 21.

```r
# First grab the data
chicks <- as_tibble(ChickWeight)

# Then subset out only the sample sets to be compared
chicks_sub <- chicks %>%
  filter(Diet %in% c(1, 2), Time == 21)
```

Once we have filtered our data we may now perform the *t*-test. Traditionally this would be performed with `t.test()`, but recent developments in R have made any testing for the comparison of means more convenient by wrapping everything up into the one single function `compare_means()`. We may use only this one single function for many of the tests we will perform in this chapter as well as Chapter 9. To use `compare_means()` for a *t*-test we must simply specify this in the `method` argument, as seen below:

```r
compare_means(weight ~ Diet, data = chicks_sub, method = "t.test")
```

```
R> # A tibble: 1 x 8
R>   .y.    group1 group2     p p.adj p.format p.signif method
R>   <chr>  <chr>  <chr>  <dbl> <dbl> <chr>    <chr>    <chr>
R> 1 weight 1      2      0.218 0.218 0.22     ns       T-test
```

As one may recall from the previous chapter, whenever we want to give a formula to a function in R, we use the `~`. The formula used above, `weight ~ Diet`, reads in plain English as "weight as a function of diet". This is perhaps easier to understand as "Y as a function of X". This means that we are assuming whatever is to the left of the `~` is the dependant variable, and whatever is to the right is the independent variable. We then tell `compare_means()` to run a *t*-test on our `chicks_sub` dataframe and it does the rest. We see in the output above that this function gives us a rather tidy read-out of the information we require to test a potential hypothesis. Let's take a moment to look through the help file for this function and see what all of this means. Did the Diet 1 and 2 produce significantly fatter birds?

7.2 ANOVA

In the `chicks` data we have four diets, not only two as in the *t*-test example just performed. Why not then simply do a *t*-test multiple times, once for each pair of diets given to the chickens? The problem is that the chances of committing a Type I error increases as more multiple comparisons are done. So, the overall chance of rejecting the null hypothesis increases. Why? If one sets $\alpha = 0.05$ (the significance level below which the null hypothesis is no longer accepted), one will still reject the null hypothesis 5% of the time when it is in fact true (i.e. when there is no difference between the groups). When many pairwise comparisons are made, the probability of rejecting the null hypothesis at least once is higher because we take this 5% risk each time we repeat a t-test. In the case of the chicken diets, we would have to perform six t-tests, and the error rate would increase to slightly less than $6 \times 5\%$. If you insist in creating more work for yourself and do t-tests many times, one way to overcome the problem of committing Type I errors that stems from multiple comparisons is to apply a Bonferroni correction.

Or better still, we do an ANOVA that controls for these Type I errors so that it remains at 5%.

A suitable null hypothesis for our chicken weight data is:

$$H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4$$

where $\mu_{1...4}$ are the means of the four diets.

At this point I was very tempted to put many equations here, but I ommitted them for your sake. Let us turn to some examples.

7.2.1 Single factor

We continue with the chicken data. The t-test showed that Diets 1 and 2 resulted in the same chicken masses at the end of the experiment at Day 21. What about the other two diets? Our null hypothesis is that, at Day 21, $\mu_1 = \mu_2 = \mu_3 = \mu_4$. Is there a statistical difference between chickens fed these four diets, or do we retain the null hypothesis? The R function for an ANOVA is `aov()`. To look for significant differences between all four diets on the last day of sampling we use this one line of code:

```
chicks.aov1 <- aov(weight ~ Diet, data = filter(chicks, Time == 21))
summary(chicks.aov1)
```

```
R>             Df Sum Sq Mean Sq F value  Pr(>F)
R> Diet         3  57164   19055   4.655 0.00686 **
R> Residuals   41 167839    4094
R> ---
R> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Task: What does the outcome say about the chicken masses? Which ones are different from each other?

Task: Devise a graphical display of this outcome.

If this seems too easy to be true, it's because we aren't quite done yet. You could use your graphical display to eyeball where the significant differences are, or we can turn to a more "precise" approach. The next step one could take is to run a Tukey HSD test on the results of the ANOVA by wrapping `tukeyHSD()` around `aov()`:

```
TukeyHSD(chicks.aov1)
```

```
R>   Tukey multiple comparisons of means
R>     95% family-wise confidence level
R>
R> Fit: aov(formula = weight ~ Diet, data = filter(chicks, Time == 21))
```

```
R> 
R> $Diet
R>          diff        lwr       upr     p adj
R> 2-1  36.95000  -32.11064 106.01064 0.4868095
R> 3-1  92.55000   23.48936 161.61064 0.0046959
R> 4-1  60.80556  -10.57710 132.18821 0.1192661
R> 3-2  55.60000  -21.01591 132.21591 0.2263918
R> 4-2  23.85556  -54.85981 102.57092 0.8486781
R> 4-3 -31.74444 -110.45981  46.97092 0.7036249
```

The output of `tukeyHSD()` shows us that pairwise comparisons of all of the groups we are comparing.

> **Task:** Look at the help file for this function to better understand what the output means.
>
> **Task:** How does one interpret the results? What does this tell us about the effect that that different diets has on the chicken weights at Day 21?
>
> **Task:** Figure out a way to plot the Tukey HSD outcomes.
>
> **Task:** Why does the ANOVA return a significant result, but the Tukey test shows that not all of the groups are significantly different from one another?
>
> **Task:** Produce a graphical display of the Tukey HSD result.

Now that we've seen how to perform a single factor ANOVA, let's watch some animations that highlight how certain aspects of our data may affect our results.

- When the sample size[1] changes
- When the mean[2] of one sample changes
- When the variance[3] of one sample increases

7.2.2 Multiple factors

What if we have multiple grouping variables, and not just one? In the case of the chicken data, there is also time that seems to be having an effect.

> **Task:** How is time having an effect?
>
> **Task:** What hypotheses can we construct around time?

Let us look at some variations around questions concerning time. We might ask, at a particular time step, are there differences amongst the effect due to diet on chicken mass? Let's see when diets are starting the have an effect by examining the outcomes at times 0, 2, 10, and 21:

```r
summary(aov(weight ~ Diet, data = filter(chicks, Time %in% c(0))))
```

```
R>             Df Sum Sq Mean Sq F value Pr(>F)
R> Diet         3   4.32   1.440   1.132  0.346
R> Residuals   46  58.50   1.272
```

```r
summary(aov(weight ~ Diet, data = filter(chicks, Time %in% c(2))))
```

[1] https://raw.githubusercontent.com/ajsmit/Basic_stats/master/figures/aov_n_slide.avi
[2] https://raw.githubusercontent.com/ajsmit/Basic_stats/master/figures/aov_mean_slide.avi
[3] https://raw.githubusercontent.com/ajsmit/Basic_stats/master/figures/aov_sd_slide.avi

```
R>             Df Sum Sq Mean Sq F value  Pr(>F)   
R> Diet         3  158.4   52.81   4.781 0.00555 **
R> Residuals   46  508.1   11.05                   
R> ---
R> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(aov(weight ~ Diet, data = filter(chicks, Time %in% c(10))))
```

```
R>             Df Sum Sq Mean Sq F value   Pr(>F)    
R> Diet         3   8314    2771    6.46 0.000989 ***
R> Residuals   45  19304     429                     
R> ---
R> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(aov(weight ~ Diet, data = filter(chicks, Time %in% c(21))))
```

```
R>             Df Sum Sq Mean Sq F value  Pr(>F)   
R> Diet         3  57164   19055   4.655 0.00686 **
R> Residuals   41 167839    4094                   
R> ---
R> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Task: What do you conclude from the above series of ANOVAs?

Task: What problem is associated with running multiple tests in the way that we have done here?

Or we may ask, regardless of diet (i.e. disregarding the effect of diet by clumping all chickens together), is time having an effect?

```
chicks.aov2 <- aov(weight ~ as.factor(Time), data = filter(chicks, Time %in% c(0, 2, 10, 21)))
summary(chicks.aov2)
```

```
R>                  Df Sum Sq Mean Sq F value Pr(>F)    
R> as.factor(Time)   3 939259  313086   234.8 <2e-16 ***
R> Residuals       190 253352    1333                   
R> ---
R> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Task: What do you conclude from the above ANOVA?

Or, to save ourselves a lot of time and reduce the coding effort, we may simply run a two-way ANOVA and look at the effects of `Diet` and `Time` simultaneously. To specify the different factors we put them in our formula and separate them with a `+`:

```
summary(aov(weight ~ Diet + as.factor(Time), data = filter(chicks, Time %in% c(0, 21))))
```

```
R>                 Df Sum Sq Mean Sq F value  Pr(>F)    
R> Diet             3  39595   13198   5.987 0.00091 ***
R> as.factor(Time)  1 734353  734353 333.120 < 2e-16 ***
R> Residuals       90 198402    2204                    
R> ---
R> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Task: What question are we asking with the above line of code? What is the answer? Also, why did we wrap `Time` in `as.factor()`?

It is also possible to look at what the interaction effect between grouping variables (i.e. in this case the effect of time on diet—does the effect of time depend on which diet we are looking at?), and not just within the individual grouping variables. To do this we replace the + in our formula with *:

```
summary(aov(weight ~ Diet * as.factor(Time), data = filter(chicks, Time %in% c(4, 21))))
```

```
R>                      Df Sum Sq Mean Sq F value   Pr(>F)    
R> Diet                  3  40914   13638   6.968 0.000298 ***
R> as.factor(Time)       1 582221  582221 297.472  < 2e-16 ***
R> Diet:as.factor(Time)  3  25530    8510   4.348 0.006684 ** 
R> Residuals            86 168322    1957                     
R> ---
R> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Task: How do these results differ from the previous set?

One may also run a post-hoc Tukey test on these results the same as for a single factor ANOVA:

```
TukeyHSD(aov(weight ~ Diet * as.factor(Time), data = filter(chicks, Time %in% c(20, 21))))
```

```
R>   Tukey multiple comparisons of means
R>     95% family-wise confidence level
R> 
R> Fit: aov(formula = weight ~ Diet * as.factor(Time), data = filter(chicks, Time %in% c(20, 21)))
R> 
R> $Diet
R>          diff        lwr       upr     p adj
R> 2-1  36.18030  -9.301330  81.66194 0.1663037
R> 3-1  90.63030  45.148670 136.11194 0.0000075
R> 4-1  62.25253  15.223937 109.28111 0.0045092
R> 3-2  54.45000   3.696023 105.20398 0.0305957
R> 4-2  26.07222 -26.072532  78.21698 0.5586643
R> 4-3 -28.37778 -80.522532  23.76698 0.4863940
R> 
R> $`as.factor(Time)`
R>           diff       lwr      upr     p adj
R> 21-20 8.088223 -17.44017 33.61661 0.5303164
R> 
R> $`Diet:as.factor(Time)`
R>                  diff        lwr        upr     p adj
R> 2:20-1:20  35.188235  -40.67378 111.050253 0.8347209
R> 3:20-1:20  88.488235   12.62622 164.350253 0.0111136
R> 4:20-1:20  63.477124  -14.99365 141.947897 0.2035951
R> 1:21-1:20   7.338235  -58.96573  73.642198 0.9999703
R> 2:21-1:20  44.288235  -31.57378 120.150253 0.6116081
R> 3:21-1:20  99.888235   24.02622 175.750253 0.0023872
R> 4:21-1:20  68.143791  -10.32698 146.614563 0.1371181
R> 3:20-2:20  53.300000  -31.82987 138.429869 0.5234263
R> 4:20-2:20  28.288889  -59.17374 115.751515 0.9723470
R> 1:21-2:20 -27.850000 -104.58503  48.885027 0.9486212
R> 2:21-2:20   9.100000  -76.02987  94.229869 0.9999766
R> 3:21-2:20  64.700000  -20.42987 149.829869 0.2732059
R> 4:21-2:20  32.955556  -54.50707 120.418182 0.9377007
R> 4:20-3:20 -25.011111 -112.47374  62.451515 0.9862822
R> 1:21-3:20 -81.150000 -157.88503  -4.414973 0.0305283
```

```
R> 2:21-3:20 -44.200000 -129.32987  40.929869 0.7402877
R> 3:21-3:20  11.400000  -73.72987  96.529869 0.9998919
R> 4:21-3:20 -20.344444 -107.80707  67.118182 0.9960548
R> 1:21-4:20 -56.138889 -135.45396  23.176184 0.3619622
R> 2:21-4:20 -19.188889 -106.65152  68.273738 0.9972631
R> 3:21-4:20  36.411111  -51.05152 123.873738 0.8984019
R> 4:21-4:20   4.666667  -85.06809  94.401428 0.9999998
R> 2:21-1:21  36.950000  -39.78503 113.685027 0.8067041
R> 3:21-1:21  92.550000   15.81497 169.285027 0.0075185
R> 4:21-1:21  60.805556  -18.50952 140.120628 0.2629945
R> 3:21-2:21  55.600000  -29.52987 140.729869 0.4679025
R> 4:21-2:21  23.855556  -63.60707 111.318182 0.9896157
R> 4:21-3:21 -31.744444 -119.20707  55.718182 0.9486128
```

Task: Jikes! That's a massive amount of results. What does all of this mean, and why is it so verbose?

7.2.2.1 *About interaction terms*

7.2.3 Examples

7.2.3.1 *Snakes!*

These data could be analysed by a two-way ANOVA without replication, or a repeated measures ANOVA. Here I will analyse it by using a two-way ANOVA without replication.

Place and Abramson (2008) placed diamondback rattlesnakes (*Crotalus atrox*) in a "rattlebox," a box with a lid that would slide open and shut every 5 minutes. At first, the snake would rattle its tail each time the box opened. After a while, the snake would become habituated to the box opening and stop rattling its tail. They counted the number of box openings until a snake stopped rattling; fewer box openings means the snake was more quickly habituated. They repeated this experiment on each snake on four successive days, which is treated as an influential variable here. Place and Abramson (2008) used 10 snakes, but some of them never became habituated; to simplify this example, data from the six snakes that did become habituated on each day are used.

First, we read in the data, making sure to convert the column named `day` to a factor. Why? Because ANOVAs work with factor independent variables, while `day` as it is encoded by default is in fact a continuous variable.

```r
snakes <- read_csv("data/snakes.csv")
snakes$day = as.factor(snakes$day)
```

The first thing we do is to create some summaries of the data. Refer to the summary statistics Chapter.

```r
snakes.summary <- snakes %>%
  group_by(day, snake) %>%
  summarise(mean_openings = mean(openings),
            sd_openings = sd(openings)) %>%
  ungroup()
snakes.summary
```

```
R> # A tibble: 24 x 4
R>    day   snake mean_openings sd_openings
```

```
R>    <fct> <chr>         <dbl>       <dbl>
R>   1 1     D1               85.          NA
R>   2 1     D11              40.          NA
R>   3 1     D12              65.          NA
R>   4 1     D3              107.          NA
R>   5 1     D5               61.          NA
R>   6 1     D8               22.          NA
R>   7 2     D1               58.          NA
R>   8 2     D11              45.          NA
R>   9 2     D12              27.          NA
R> 10 2     D3               51.          NA
R> # ... with 14 more rows
```

Task: Something seems… off. What's going on here? Please explain this outcome.

To fix this problem, let us ignore the grouping by both `snake` and `day`.

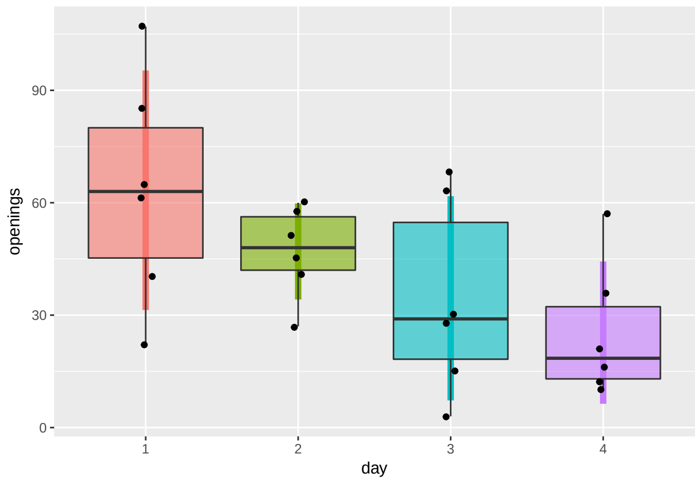
```
snakes.summary <- snakes %>%
  group_by(day) %>%
  summarise(mean_openings = mean(openings),
            sd_openings = sd(openings)) %>%
  ungroup()
snakes.summary
```

```
R> # A tibble: 4 x 3
R>   day   mean_openings sd_openings
R>   <fct>         <dbl>       <dbl>
R> 1 1              63.3        30.5
R> 2 2              47.0        12.2
R> 3 3              34.5        26.0
R> 4 4              25.3        18.1
```

```
library(Rmisc)
snakes.summary2 <- summarySE(data = snakes, measurevar = "openings", groupvars = c("day"))
```

Now we turn to some visual data summaries.

```
ggplot(data = snakes, aes(x = day, y = openings)) +
  geom_segment(data = snakes.summary2, aes(x = day, xend = day, y = openings - ci, yend = openings + ci, colour =
              size = 2.0, linetype = "solid", show.legend = F) +
  geom_boxplot(aes(fill = day), alpha = 0.6, show.legend = F) +
  geom_jitter(width = 0.05)
```



What are our null hypotheses?

1. H0: There is no difference between snakes with respect to the number of openings at which they habituate.
2. H0: There is no difference between days in terms of the number of openings at which the snakes habituate.

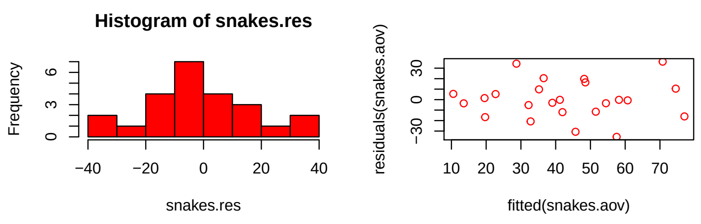
Fit the ANOVA model to test these hypotheses:

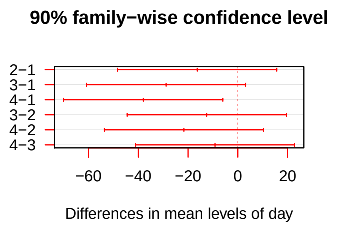
```
snakes.aov <- aov(openings ~ day + snake, data = snakes)
summary(snakes.aov)
```

```
R>              Df Sum Sq Mean Sq F value Pr(>F)
R> day           3   4878  1625.9   3.320 0.0487 *
R> snake         5   3042   608.4   1.242 0.3382
R> Residuals    15   7346   489.7
R> ---
R> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Now we need to test of the assumptions hold true (i.e. erros are normally distributed and heteroscedastic). Also, where are the differences?

```
par(mfrow = c(2, 2))
# Checking assumptions...
# make a histogram of the residuals;
# they must be normal
snakes.res <- residuals(snakes.aov)
hist(snakes.res, col = "red")

# make a plot of residuals and the fitted values;
# # they must be normal and homoscedastic
plot(fitted(snakes.aov), residuals(snakes.aov), col = "red")

snakes.tukey <- TukeyHSD(snakes.aov, which = "day", conf.level = 0.90)
plot(snakes.tukey, las = 1, col = "red")
```





7.3 Alternatives to ANOVA

In the first main section of this chapter we learned how to test hypotheses based on the comparisons of means between sets of data when we were able to meet our two base assumptions. These parametric tests are preferred over non-parametric tests because they are more robust. However, when we simply aren't able to meet these assumptions we must not despair. Non-parametric tests are still useful. In this chapter we will learn how to run non-parametric tests for two sample and multiple sample datasets. To start, let's load our libraries and `chicks` data if we have not already.

```
# First activate libraries
library(tidyverse)
library(ggpubr)

# Then load data
chicks <- as_tibble(ChickWeight)
```

With our libraries and data loaded, let's find a day in which at least one of our assumptions are violated.

```
# Then check for failing assumptions
chicks %>%
  filter(Time == 0) %>%
  group_by(Diet) %>%
  summarise(norm_wt = as.numeric(shapiro.test(weight)[2]),
            var_wt = var(weight))
```

```
R>       norm_wt   var_wt
R> 1 0.000221918 1.282041
```

7.3.1 Wilcox rank sum test

The non-parametric version of a t-test is a Wilcox rank sum test. To perform this test in R we may again use `compare_means()` and specify the test we want:

```
compare_means(weight ~ Diet, data = filter(chicks, Time == 0, Diet %in% c(1, 2)), method = "wilcox.test")
```

```
R> # A tibble: 1 x 8
R>   .y.    group1 group2     p p.adj p.format p.signif method
R>   <chr>  <chr>  <chr>  <dbl> <dbl> <chr>    <chr>    <chr>
R> 1 weight 1      2      0.235 0.235 0.23     ns       Wilcoxon
```

What do our results show?

7.3.2 Kruskall-Wallis rank sum test

7.3.2.1 *Single factor*

The non-parametric version of an ANOVA is a Kruskall-Wallis rank sum test. As you may have by now surmised, this may be done with `compare_means()` as seen below:

```r
compare_means(weight ~ Diet, data = filter(chicks, Time == 0), method = "kruskal.test")
```

```
R> # A tibble: 1 x 6
R>   .y.        p p.adj p.format p.signif method
R>   <chr>  <dbl> <dbl> <chr>    <chr>    <chr>
R> 1 weight 0.475 0.475 0.48     ns       Kruskal-Wallis
```

As with the ANOVA, this first step with the Kruskall-Wallis test is not the last. We must again run a post-hoc test on our results. This time we will need to use `pgirmess::kruskalmc()`, which means we will need to load a new library.

```r
library(pgirmess)

kruskalmc(weight ~ Diet, data = filter(chicks, Time == 0))
```

```
R> Multiple comparison test after Kruskal-Wallis
R> p.value: 0.05
R> Comparisons
R>     obs.dif critical.dif difference
R> 1-2    6.95     14.89506      FALSE
R> 1-3    6.90     14.89506      FALSE
R> 1-4    4.15     14.89506      FALSE
R> 2-3    0.05     17.19933      FALSE
R> 2-4    2.80     17.19933      FALSE
R> 3-4    2.75     17.19933      FALSE
```

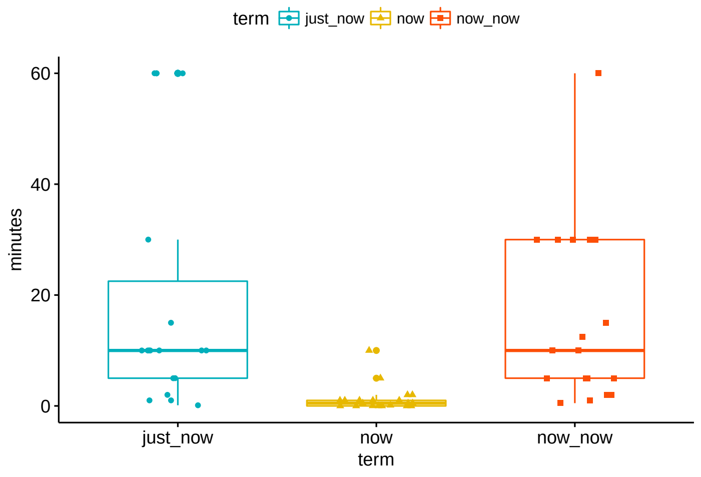
Let's consult the help file for `kruskalmc()` to understand what this print-out means.

7.3.2.2 *Multiple factors*

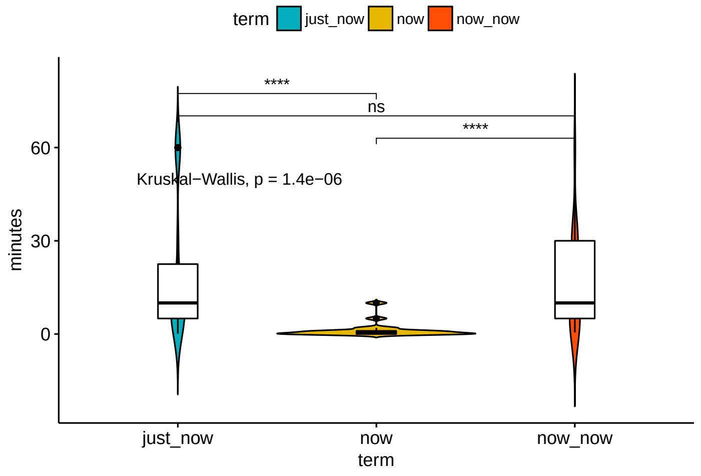
The water becomes murky quickly when one wants to perform multiple factor non-parametric comparison of means tests. To that end, we will not cover the few existing methods here. Rather, one should avoid the necessity for these types of tests when designing an experiment.

7.3.3 The SA time data

```r
sa_time <- as_tibble(read_csv("data/SA_time.csv", col_types = list(col_double(), col_double(), col_double())))
sa_time_long <- sa_time %>%
  gather(key = "term", value = "minutes") %>%
  filter(minutes < 300) %>%
  mutate(term = as.factor(term))

my_comparisons <- list( c("now", "now_now"), c("now_now", "just_now"), c("now", "just_now") )

ggboxplot(sa_time_long, x = "term", y = "minutes",
          color = "term", palette = c("#00AFBB", "#E7B800", "#FC4E07"),
          add = "jitter", shape = "term")
```



```r
ggviolin(sa_time_long, x = "term", y = "minutes", fill = "term",
         palette = c("#00AFBB", "#E7B800", "#FC4E07"),
         add = "boxplot", add.params = list(fill = "white")) +
  stat_compare_means(comparisons = my_comparisons, label = "p.signif") + # Add significance levels
  stat_compare_means(label.y = 50)                                       # Add global the p-value
```



7.4 Exercises

7.4.1 Exercise 1

Here is bunch of data for pigs raised on different diets. The experiment is similar to the chicken one. Does feed type have an effect on the mass of pigs at the end of the experiment?

```r
# enter the mass at the end of the experiment
feed_1 <- c(60.8, 57.0, 65.0, 58.6, 61.7)
feed_2 <- c(68.7, 67.7, 74.0, 66.3, 69.8)
feed_3 <- c(102.6, 102.1, 100.2, 96.5)
feed_4 <- c(87.9, 84.2, 83.1, 85.7, 90.3)

# make a dataframe
bacon <- as.tibble(data.frame(
  feed = c(
  rep("Feed 1", length(feed_1)),
  rep("Feed 2", length(feed_2)),
  rep("Feed 3", length(feed_3)),
  rep("Feed 4", length(feed_4))
  ),
  mass = c(feed_1, feed_2, feed_3, feed_4)
  ))
```

7.4.2 Exercise 2

Construct suitable null and alternative hypotheses for the built-in `ToothGrowth` data, and test your hypotheses using an ANOVA.

```r
teeth <- datasets::ToothGrowth
```

7.4.3 Exercise 3

Find or generate your own data that lend themselves to being analysed by a two-way ANOVA. Generate suitable hypotheses about your data, and analyse it. Supplement your analysis by providing a suitable descriptive statistical summary and graph(s) of your data.

8 Simple linear regressions

```r
library(tidyverse)
```

Regressions test the statistical significance of the *dependence* of one continuous variable on one or many independent continuous variables.

8.1 The simple linear regression equation

The linear regression equation is already known to you. It is:

$$y_n = \beta \cdot x_n + \alpha + \epsilon$$

Coefficients are parameters (statistics) that describe two properties of the linear line that best fit a scatter plot between a dependent variable and the independent variable. The dependent variable, $y_{1..n}$, may also be called the response variable, and the independent variable, $x_{1..n}$, the predictor. The regression model consists of an *intercept term*, α, that describes where the fitted line starts and intercepts with the *y*-axis, and the *slope*, β, of the line. The amount of variation not explained by a linear relationship of y on x is termed the residual variation, or simply the residual or the error term, and in the above equation it is indicated by ϵ.

The regression parameters α and β are determined by *minimising the error sum of squares* of the error term, ϵ. It allows us to predict new fitted values of y based on values of x. The error sum of squares is calculated by:

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

To see an animation demonstrating this click here[1].

[1] https://raw.githubusercontent.com/ajsmit/Basic_stats/master/figures/lm_rotate.avi

We will demonstrate the principle behind a simple linear regression by using the built-in dataset `faithful`. According to the R help file for the data, the dataset describes the "Waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA."

```
head(faithful)
```

```
R>   eruptions waiting
R> 1     3.600      79
R> 2     1.800      54
R> 3     3.333      74
R> 4     2.283      62
R> 5     4.533      85
R> 6     2.883      55
```

In this dataset there are two columns: the first, `eruptions`, denotes the duration of the eruption (in minutes), and the second, `waiting`, is the waiting time (also in minutes) until the next eruptions. Its linear regression model can be expressed as:

$$eruption_n = \beta \cdot waiting_n + \alpha + \epsilon$$

Here we fit the model in R. When we perform a linear regression in R, it will output the model and the coefficients:

```
eruption.lm <- lm(eruptions ~ waiting, data = faithful)
summary(eruption.lm)
```

```
R> 
R> Call:
R> lm(formula = eruptions ~ waiting, data = faithful)
R> 
R> Residuals:
R>      Min       1Q   Median       3Q      Max 
R> -1.29917 -0.37689  0.03508  0.34909  1.19329 
R> 
R> Coefficients:
R>              Estimate Std. Error t value Pr(>|t|)    
R> (Intercept) -1.874016   0.160143  -11.70   <2e-16 ***
R> waiting      0.075628   0.002219   34.09   <2e-16 ***
R> ---
R> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
R> 
R> Residual standard error: 0.4965 on 270 degrees of freedom
R> Multiple R-squared:  0.8115, Adjusted R-squared:  0.8108 
R> F-statistic:  1162 on 1 and 270 DF,  p-value: < 2.2e-16
```

8.1.1 The intercept

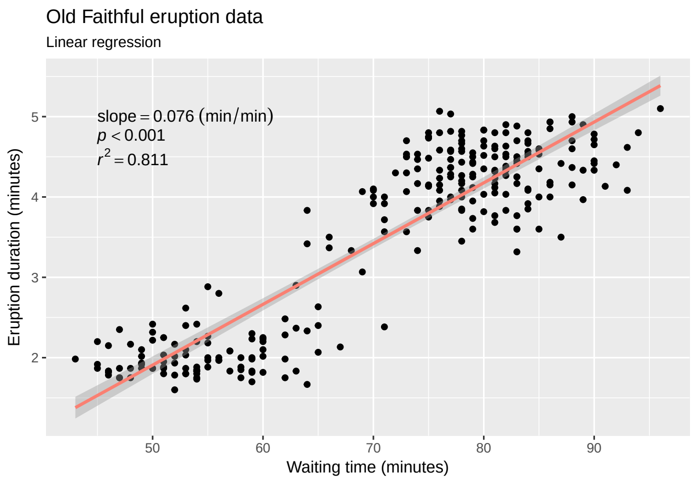
The intercept is the best estimate of the starting point of the fitted line on the lefthand side of the graph. You will notice that there is also an estimate for the standard error of the estimate for the intercept.

8.1.2 The regression coefficient

The interpretation of the regression coefficient is simple. For every one unit of change in the independent variable (here waiting time) there is a corresponding change in the dependent variable (here the duration of the eruption). This is the *slope* or *gradient*, and it may be positive or negative. In the example the coefficient of determination of the line is denoted by the value 0.076 min.min^{-1} in the column termed `Estimate` and in the row called `waiting` (the latter name will of course depend on the name of the response column in your dataset). The coefficient of determination multiplies the response variable to produce a prediction of the response based on the slope of the relationship between the response and the predictor. It tells us how much one unit in change of the independent variable *determines* the corresponding change in the response variable. There is also a standard error for the estimate.

8.1.3 A graph of the linear regression

```r
slope <- round(eruption.lm$coef[2], 3)
# p.val <- round(coefficients(summary(eruption.lm))[2, 4], 3) # it approx. 0, so...
p.val = 0.001
r2 <- round(summary(eruption.lm)$r.squared, 3)

ggplot(data = faithful, aes(x = waiting, y = eruptions)) +
  geom_point() +
  annotate("text", x = 45, y = 5, label = paste0("slope == ", slope, "~(min/min)"), parse = TRUE, hjust = 0) +
  annotate("text", x = 45, y = 4.75, label = paste0("italic(p) < ", p.val), parse = TRUE, hjust = 0) +
  annotate("text", x = 45, y = 4.5, label = paste0("italic(r)^2 == ", r2), parse = TRUE, hjust = 0) +
  stat_smooth(method = "lm", colour = "salmon") +
  labs(title = "Old Faithful eruption data",
       subtitle = "Linear regression",
       x = "Waiting time (minutes)",
       y = "Eruption duration (minutes)")
```



8.1.4 Predicting from the linear model

Knowing α and β allows us to predict what the eruption duration will be for a certain amount of waiting. Since the slope of the line is positive we can expect that the longer the waiting time is between eruptions the longer the eruption would be. But how can we quantify this? We start by extracting the coefficients (both the intercept and the regression coefficient). Then we use these values to reassemble the regression equation that we have written out above (i.e., $eruption_n = \beta \cdot waiting_n + \alpha + \epsilon$). Here's how:

```
# use the accessor function to grab the coefficients:
erupt.coef <- coefficients(eruption.lm)
erupt.coef
```

```
R> (Intercept)     waiting
R> -1.87401599  0.07562795
```

```
# how long would an eruption last of we waited, say, 80 minutes?
waiting <- 80

# the first and second coef. can be accessed using the
# square bracket notation:
erupt.pred <- erupt.coef[1] + (erupt.coef[2] * waiting)
erupt.pred # the unit is minutes
```

```
R> (Intercept)
R>     4.17622
```

The prediction is that, given a waiting time of 80 minutes since the previous eruption, the next eruption will last 4.176 minutes.

There is another way to do this. The `predict()` function takes a dataframe of values for which we want to predict the duration of the eruption and returns a vector with the waiting times:

```
pred.val <- data.frame(waiting = c(60, 80, 100))
predict(eruption.lm, pred.val) # returns waiting time in minutes
```

```
R>        1        2        3
R> 2.663661 4.176220 5.688779
```

8.1.5 The coefficient of determination, r^2

The coefficient of determination, the r^2, of a linear model is the quotient of the variances of the fitted values, $\hat{y}_i$, and observed values, y_i, of the dependent variable. If the mean of the dependent variable is $\bar{y}$, then the r^2 is:

$$r^2 = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2}$$

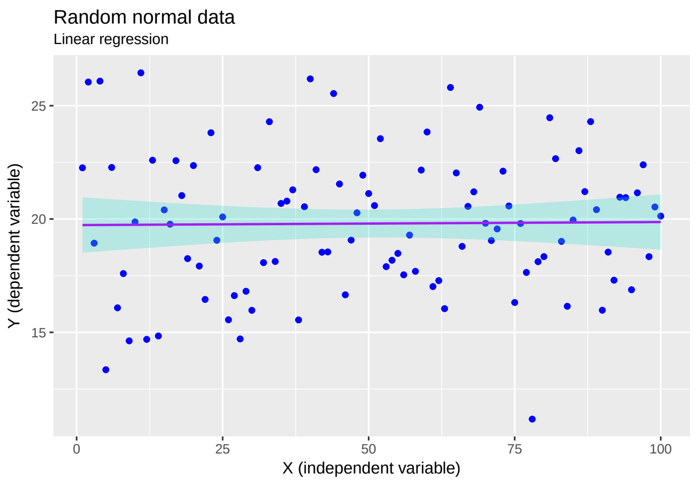
In our Old Faithful example, the coefficient of determination is returned together with the summary of the `eruption.lm` object, but it may also be extracted as:

```
summary(eruption.lm)$r.squared
```

```
R> [1] 0.8114608
```

What does the r^2 tell us? It tells us the "fraction of variance explained by the model" (from the `summary.lm()` help file). In other words it is the proportion of variation in the dispersion (variance) of the measured dependent variable, y, that can be predicted from the measured independent variable, x (or variables in the case of multiple regressions). It gives us an indication of how well the observed outcome variable is predicted by the observed influential variable, and in the case of a simple linear regression, the geometric relationship of y on x is a straight line. r^2 can take values from 0 to 1: a value of 0 tells us that there is absolutely no relationship between the two, whilst a value of 1 shows that there is a perfect fit and a scatter of points to denote the y vs. x relationship will all fall perfectly on a straight line.

```r
library(tidyverse)
n <- 100
set.seed(666)
rand.df <- data.frame(x = seq(1:n),
                      y = rnorm(n = n, mean = 20, sd = 3))
ggplot(data = rand.df, aes(x = x, y = y)) +
  geom_point(colour = "blue") +
  stat_smooth(method = "lm", colour = "purple", size = 0.75, fill = "turquoise", alpha = 0.3) +
  labs(title = "Random normal data",
       subtitle = "Linear regression",
       x = "X (independent variable)",
       y = "Y (dependent variable)")
```



Regressions may take on any relationship, not only a linear one. For example, there are parabolic, hyperbolic, logistic, exponential, etc. relationships of y on x, and here, too, does r^2 tell us the same thing. If we assume that the samples were representatively drawn from a population (i.e. the sample fully captures the relationship of y on x that is present in the entire population), the r^2 will represent the relationship in the population too.

In the case of our Old Faithful data, the r^2 is 0.811, meaning that the proportion of variance explained is 81.1%; the remaining 18.9% is not (yet) accounted for by the linear relationship. Adding more predictors into the regression (i.e. a multiple regression) might consume some of the unexplained variance and increase the overall r^2.

8.1.6 Significance test for linear regression

There are several hypothesis tests associated with a simple linear regression. All of them assume that the residual error, ϵ, in the linear regression model is independent of x (i.e. nothing about the structure of the error term can be inferred based on a knowledge of x), is normally distributed, with zero mean and constant variance. We say the residuals are *i.i.d.* (independent and identically distributed, which is a fancy way of saying they are random).

We can decide whether there is any significant relationship (slope) of y on x by testing the null hypothesis that $\beta = 0$. Rejecting the null hypothesis causes the alternate hypothesis of $\beta \neq 0$ to be accepted. This test is automatically performed when fitting a linear model in R and asking for a summary of the regression object, but it is insightful and important to know that the test is simply a one-sample t-test. In the regression summary the probability associated with this test is given in the `Coefficients` table in the column called `Pr(>|t|)`.

In the Old Faithful data, the p-value associated with `waiting` is less than 0.05 and we therefore reject the null hypothesis that $\beta = 0$. So, there is a significant linear relationship of eruption duration on the waiting time between eruptions.

> **Question:** Note that there is also a hypothesis test in the `(Intercept)` row. What does this do?

8.1.7 Confidence interval for linear regression

Again we have to observe the assumption of *i.i.d.* as before. For a given value of x, the 95% confidence interval around the mean of the *observed* dependent variable, $\bar{y}$, can be obtained as follows:

```
pred.val <- data.frame(waiting = c(80))
predict(eruption.lm, pred.val, interval = "confidence")
```

```
R>       fit      lwr      upr
R> 1 4.17622 4.104848 4.247592
```

So, the 95% confidence interval of the mean eruption duration for the waiting time of 80 minutes is between 4.105 and 4.248 minutes.

8.1.8 Prediction interval for linear regression

Observe that ϵ is *i.i.d.* For a given value of x, the interval estimate of the *future* dependent variable, y, is called the prediction interval. The way we do this is similar to finding the confidence interval:

```
pred.val <- data.frame(waiting = c(80))
predict(eruption.lm, pred.val, interval = "prediction")
```

```
R>       fit      lwr      upr
R> 1 4.17622 3.196089 5.156351
```

The intervals are wider. The difference between confidence and prediction intervals is subtle and requires some philosophical consideration. In practice, if you use these intervals to make inferences about the population from which the samples were drawn, use the prediction intervals. If you instead want to describe the samples which you have taken, use the confidence intervals.

8.1.9 Residual plot

8.1.10 Standardised residual

8.1.11 Normal probability plot of residuals

8.2 Using an additional categorical variable

- When you use a categorical variable, in R the intercept represents the default position for a given value in the categorical column. Every other value then gets a modifier to the base prediction.

9 Correlations

A correlation is performed when we want to investigate potential relationships between variables from the same sample. This does not mean that one variable explains the other, we arrive at that through the use of regression, as seen in Chapter 8. Like all statistical tests, correlation requires a series of assumptions as well:

- pair-wise data
- absence of outliers
- linearity
- normality of distribution
- homoscedasticity
- level (type) of measurement
  - Continuous data (Pearson correlation)
  - Ordinal data (Spearman correlation)

In order to investigate correlations in biological data lets load the `ecklonia` dataset.

```r
# Load libraries
library(tidyverse)
library(ggpubr)
library(corrplot)

# Load data
ecklonia <- read_csv("data/ecklonia.csv")
```

We will also create a subsetted version of our data by removing all of the categorical variables. If we have a dataframe where each column represents pair-wise continuous/ordinal measurements with all of the other columns we may very quickly and easily perform a much wider range of correlation analyses.

```r
ecklonia_sub <- ecklonia %>%
  select(-species, - site, - ID)
```

9.1 Pearson correlation

When the values we are comparing are continuous, we may use a Pearson test. This is the default and so requires little work on our end. The resulting statistic from this test is known as the correlation coefficient.

```
# Perform correlation analysis on two specific variables
  # Note that we do not need the final two arguments in this function to be stated
  # as they are the defaut settings.
  # They are only shown here to illustrate that they exist.
cor.test(x = ecklonia$stipe_length, ecklonia$frond_length,
    use = "everything", method = "pearson")
```

```
R>
R>  Pearson's product-moment correlation
R>
R> data:  ecklonia$stipe_length and ecklonia$frond_length
R> t = 4.2182, df = 24, p-value = 0.0003032
R> alternative hypothesis: true correlation is not equal to 0
R> 95 percent confidence interval:
R>  0.3548169 0.8300525
R> sample estimates:
R>       cor
R> 0.6524911
```

Above we have tested the correlation between the length of *Ecklonia maxima* stipes and the length of their fronds. A perfect positive (negative) relationship would produce a value of 1 (-1), whereas no relationship would produce a value of 0. The result above, `cor = 0.65` is relatively strong. That being said, should our dataset contain multiple variables, as `ecklonia` does, we may investigate all of the correlations simultaneously. Remember that in order to do so we want to ensure that we may perform the same test on each of our paired variables. In this case we will use `ecklonia_sub` as we know that it contains only continuous data and so are appropriate for use with a Pearson test. By default R will use all of the data we give it and perform a Pearson test so we do not need to specify any further arguments. Note however that this will only output the correlation coefficients, and does not produce a full test of each correlation. This will however be useful for us to have just now.

```
ecklonia_pearson <- cor(ecklonia_sub)
ecklonia_pearson
```

```
R>                      stipe_length stipe_diameter frond_length     digits
R> stipe_length            1.0000000      0.5853577   0.65249111 0.24273818
R> stipe_diameter          0.5853577      1.0000000   0.39032107 0.24098021
R> frond_length            0.6524911      0.3903211   1.00000000 0.35604257
R> digits                  0.2427382      0.2409802   0.35604257 1.00000000
R> primary_blade_width     0.3413692      0.8269196   0.28307223 0.13954003
R> primary_blade_length    0.1330329      0.3176688  -0.01584175 0.09784266
R> stipe_mass              0.5768102      0.8226091   0.39471926 0.06958775
R> frond_mass              0.5124551      0.5066763   0.56762418 0.28162375
R> epiphyte_length         0.6056152      0.5372470   0.61489806 0.04634642
R>                      primary_blade_width primary_blade_length stipe_mass
R> stipe_length                   0.3413692           0.13303292 0.57681018
R> stipe_diameter                 0.8269196           0.31766879 0.82260910
```

```
R> frond_length                  0.2830722         -0.01584175 0.39471926
R> digits                        0.1395400          0.09784266 0.06958775
R> primary_blade_width           1.0000000          0.33931061 0.82745162
R> primary_blade_length          0.3393106          1.00000000 0.16175948
R> stipe_mass                    0.8274516          0.16175948 1.00000000
R> frond_mass                    0.3630630          0.14729132 0.47195130
R> epiphyte_length               0.4119969          0.26051399 0.50790877
R>                      frond_mass epiphyte_length
R> stipe_length          0.5124551      0.60561520
R> stipe_diameter        0.5066763      0.53724704
R> frond_length          0.5676242      0.61489806
R> digits                0.2816237      0.04634642
R> primary_blade_width   0.3630630      0.41199691
R> primary_blade_length  0.1472913      0.26051399
R> stipe_mass            0.4719513      0.50790877
R> frond_mass            1.0000000      0.43920289
R> epiphyte_length       0.4392029      1.00000000
```

Task: What does the outcome of this test show?

9.2 Spearman rank correlation

When the data we want to compare are not continuous, but rather ordinal, we will need to use a Spearman test. This is not often a test one uses in biology because we tend to want to compare continuous data within categories. In the code below we will add a column of ordinal data to our `ecklonia` data to so that we may look at this test.

```
# Create ordinal data
ecklonia$length <- as.numeric(cut((ecklonia$stipe_length+ecklonia$frond_length), breaks = 3))

# Run test on any variable
cor.test(ecklonia$length, ecklonia$digits)
```

```
R> 
R>  Pearson's product-moment correlation
R> 
R> data:  ecklonia$length and ecklonia$digits
R> t = 1.4979, df = 24, p-value = 0.1472
R> alternative hypothesis: true correlation is not equal to 0
R> 95 percent confidence interval:
R>  -0.1070908  0.6105880
R> sample estimates:
R>     cor
R> 0.29239
```

Task: How else might we use this?

9.3 Kendall rank correlation

This test will work for both continuous and ordinal data. A sort of dealers choice of correlation tests. It's main purpose is to allow us to perform a correlation on non-normally distributed data.

Let's look at the normality of our `ecklonia` variables and pull out those that are not normal in order to see how the results of this test may differ from our Pearson tests.

```
ecklonia_norm <- ecklonia_sub %>%
  gather(key = "variable") %>%
  group_by(variable) %>%
  summarise(variable_norm = as.numeric(shapiro.test(value)[2]))
ecklonia_norm
```

```
R> # A tibble: 9 x 2
R>   variable             variable_norm
R>   <chr>                        <dbl>
R> 1 digits                     0.0671
R> 2 epiphyte_length            0.626
R> 3 frond_length               0.202
R> 4 frond_mass                 0.277
R> 5 primary_blade_length       0.00393
R> 6 primary_blade_width        0.314
R> 7 stipe_diameter             0.170
R> 8 stipe_length               0.213
R> 9 stipe_mass                 0.817
```

From this analysis we may see that the values for primary blade length are not normally distributed. In order to make up for this violation of our assumption of normality we may use the Spearman test.

```
cor.test(ecklonia$primary_blade_length, ecklonia$primary_blade_width, method = "kendall")
```

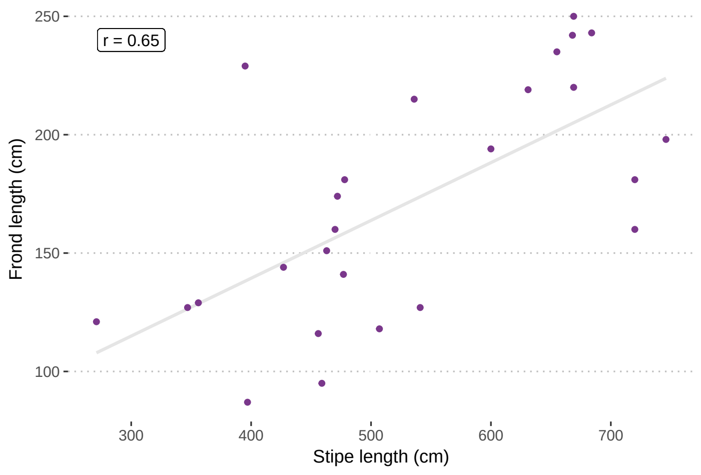
```
R>
R>  Kendall's rank correlation tau
R>
R> data:  ecklonia$primary_blade_length and ecklonia$primary_blade_width
R> z = 2.3601, p-value = 0.01827
R> alternative hypothesis: true tau is not equal to 0
R> sample estimates:
R>       tau
R> 0.3426171
```

9.4 One panel visual

As is the case with everything else we have learned thus far, a good visualisation can go a long way to help us understand what the statistics are doing. Below we visualise the stipe length to frond length relationship.

```
# Calculate Pearson r beforehand for plotting
r_print <- paste0("r = ",
                  round(cor(x = ecklonia$stipe_length, ecklonia$frond_length),2))

# Then create a single panel showing one correlation
ggplot(data = ecklonia, aes(x = stipe_length, y = frond_length)) +
  geom_smooth(method = "lm", colour = "grey90", se = F) +
  geom_point(colour = "mediumorchid4") +
  geom_label(x = 300, y = 240, label = r_print) +
```

```r
  labs(x = "Stipe length (cm)", y = "Frond length (cm)") +
  theme_pubclean()
```



\begin{figure} \caption{Scatterplot showing relationship between *Ecklonia maxima* stipe length (cm) and frond length (cm). The correlation coefficient (Pearson r) is shown in the top left corner. Note that the grey line running through the middle is a fitted linear model and is not generating the correlation value. Rather it is included to help visually demonstrate the strength of the relationship.} \end{figure}

Just by eye-balling this scatterplot it should be clear that these data tend to increase at a roughly similar rate. Our Pearson r value is an indication of what that is.

9.5 Multiple panel visual

But why stop at one panel? It is relatively straightforward to quickly plot correlation results for all of our variables in one go. In order to show which variables correlate most with which other variables all at once, without creating chaos, we will create what is known as a heat map. This visualisation uses a range of colours, usually blue to red, to demonstrate where more of something is. In this case, we use it to show where more correlation is occurring between morphometric properties of the kelp *Ecklonia maxima*.

```r
corrplot(ecklonia_pearson, method = "circle")
```

> **Task:** What does the series of dark blue circles through the middle of this plot mean?
>
> < **Task:** Which morphometric properties correlate best/worst?

Slightly more involved, but much more useful, is the generation of a heat map using **`ggplot2`**.

9.6 Exercises

9.6.1 Exercise 1

Produce a heat map using **`ggplot2`**.

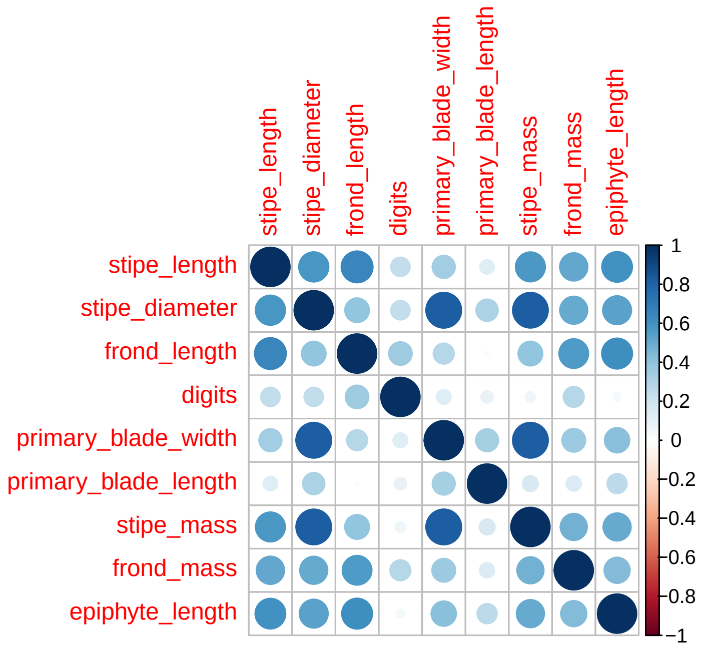


Figure 9.1: Correlation plot showing the strength of all correlations between all variables as a scale from red (negative) to blue (positive).

10 Confidence intervals

A confidence interval (CI) tells us within what range we may be certain to find the true mean from which any sample has been taken. To calculate a confidence interval requires three things:

```
# First we set the sample mean
sample_mean <- 10

# Then the sample standard deviation
sample_sd <- 2

# Then the number of samples
sample_n <- 20
```

Once we know these things we must then use the following formula to calculate th CI:

```
error <- qnorm(0.975)*sample_sd/sqrt(sample_n)
```

With the CI known, we then substract/add it from the sample mean in order to find the upper and lower limits:

```
lower <- sample_mean-error
upper <- sample_mean+error
lower
```

```
R> [1] 9.123477
```

```
upper
```

```
R> [1] 10.87652
```

11

Testing assumptions or: How I learned to stop worrying and transform the data

In this chapter we will learn how to test for the two most common assumptions we will make in the biological sciences.

Our tests for these two assumptions fail often with real data. Therefore, we must often identify the way in which our data are distributed so we may better decide how to transform them in an attempt to coerce them into a format that will pass the assumptions.

Before we begin, let's go ahead and activate our packages and load our data.

```
library(tidyverse)
chicks <- as_tibble(ChickWeight)
```

11.1 The two main assumptions

11.1.1 Normality

The quickest method of testing the normality of a variable is with the Shapiro-Wilk normality test. This will return two values, a W score and a *p*-value. FOr the purposes of this course we may safely ignore the W score and focus on the *p*-value. When $p >= 0.05$ we may assume that the data are normally distributed. If $p < 0.05$ then the data are not normally distrubted. Let's look at all of the `chicks` data without filtering it:

```
shapiro.test(chicks$weight)
```

```
R>
R>  Shapiro-Wilk normality test
```

```
R> 
R> data:  chicks$weight
R> W = 0.90866, p-value < 2.2e-16
```

Are these data normally distributed? How do we know? Now let's filter the data based on the different diets for only the weights taken on the last day (21):

```
chicks %>%
  filter(Time == 21) %>%
  group_by(Diet) %>%
  summarise(norm_wt = as.numeric(shapiro.test(weight)[2]))
```

```
R> # A tibble: 4 x 2
R>   Diet  norm_wt
R>   <fct>   <dbl>
R> 1 1       0.591
R> 2 2       0.949
R> 3 3       0.895
R> 4 4       0.186
```

How about now?

11.1.2 Homoscedasticity

Here we need no fancy test. We must simply calculate the variance of the variables we want to use and see that they are not more than 3 – 4 times greater than one another.

```
chicks %>%
  filter(Time == 21) %>%
  group_by(Diet) %>%
  summarise(var_wt = var(weight))
```

```
R> # A tibble: 4 x 2
R>   Diet  var_wt
R>   <fct>  <dbl>
R> 1 1      3446.
R> 2 2      6106.
R> 3 3      5130.
R> 4 4      1879.
```

Well, do these data pass the two main assumptions?

11.2 Transforming data

11.2.1 Log transform

11.2.2

11.3 Exercises

11.3.1 Exercise 1

Find one of the days of measurement where the chicken weights do not pass the assumptions of normality, and another day (not day 21!) in which they do.

11.3.2 Exercise 2

Transform the data so that they may pass the assumptions.

12 Linear mixed models

In the previous chapter we learned how to test hypotheses based on the comparions of means between sets of data when we were able to meet our two base assumptions. These parametric tests are preferred over non-parametric tests because they are more robust. However, when we simply aren't able to meet these assumptions we must not despair. Non-parametric tests are still useful. In this chapter we will learn how to run non-parametirc tests for two sample and multiple sample datasets. To start, let's load our libraries and `chicks` data if we have not already.

```
# First activate libraries
library(tidyverse)
library(ggpubr)

# Then load data
chicks <- as_tibble(ChickWeight)
```

With our libraries and data loaded, let's find a day in which at least one of our assumptions are violated.

```
# Then check for failing assumptions
chicks %>%
  filter(Time == 0) %>%
  group_by(Diet) %>%
  summarise(norm_wt = as.numeric(shapiro.test(weight)[2]),
            var_wt = var(weight))
```

```
R> # A tibble: 4 x 3
R>   Diet  norm_wt var_wt
R>   <fct>   <dbl>  <dbl>
R> 1 1     0.0138   0.989
R> 2 2     0.138    2.23
R> 3 3     0.00527  1.07
R> 4 4     0.0739   1.11
```

12.1 Wilcox rank sum test

The non-parametric version of a t-test is a Wilcox rank sum test. To perform this test in R we may again use `compare_means()` and specify the test we want:

```
compare_means(weight ~ Diet, data = filter(chicks, Time == 0, Diet %in% c(1, 2)), method = "wilcox.test")
```

```
R> # A tibble: 1 x 8
R>   .y.    group1 group2     p p.adj p.format p.signif method
R>   <chr>  <chr>  <chr>  <dbl> <dbl> <chr>    <chr>    <chr>
R> 1 weight 1      2      0.235 0.235 0.23     ns       Wilcoxon
```

What do our results show?

12.2 Kruskall-Wallis rank sum test

12.2.1 Single factor

The non-parametric version of an ANOVA is a Kruskall-Wallis rank sum test. As you may have by now surmised, this may be done with `compare_means()` as seen below:

```
compare_means(weight ~ Diet, data = filter(chicks, Time == 0), method = "kruskal.test")
```

```
R> # A tibble: 1 x 6
R>   .y.        p p.adj p.format p.signif method
R>   <chr>  <dbl> <dbl> <chr>    <chr>    <chr>
R> 1 weight 0.475 0.475 0.48     ns       Kruskal-Wallis
```

As with the ANOVA, this first step with the Kruskall-Wallis test is not the last. We must again run a post-hoc test on our results. This time we will need to use `pgirmess::kruskalmc()`, which means we will need to load a new library.

```
library(pgirmess)

kruskalmc(weight ~ Diet, data = filter(chicks, Time == 0))
```

```
R> Multiple comparison test after Kruskal-Wallis
R> p.value: 0.05
R> Comparisons
R>     obs.dif critical.dif difference
R> 1-2    6.95     14.89506      FALSE
R> 1-3    6.90     14.89506      FALSE
R> 1-4    4.15     14.89506      FALSE
R> 2-3    0.05     17.19933      FALSE
R> 2-4    2.80     17.19933      FALSE
R> 3-4    2.75     17.19933      FALSE
```

Let's consult the help file for `kruskalmc()` to understand what this print-out means.

12.2.2 Multiple factors

The water becomes murky quickly when one wants to perform mutliple factor non-parametric comparison of means tests. TO that end, we will not cover the few existing methods here. Rather, one should avoid the necessity for these types of tests when designing an experiment.

12.3 Generalised linear models

12.3.1 Sign Test

12.3.2 Wilcoxon Signed-Rank Test

12.3.3 Mann-Whitney-Wilcoxon Test

12.3.4 Kruskal-Wallis Test

12.3.5 Generalised linear models (GLM)

12.4 Exercises

12.5 Exercise 1

Write out the hypotheses that we tested for in this chapter and answer them based on the results we produced in class.

13 Chi-squared

A chi-squared test is used when one wants to see if there is a realtionship between count data of two or more factors.

```r
x <- c(A = 20, B = 15, C = 25)
chisq.test(x)
```

```
R> 
R>  Chi-squared test for given probabilities
R> 
R> data:  x
R> X-squared = 2.5, df = 2, p-value = 0.2865
```