Biostats with R

Aditya Srinivasulu

November 16, 2019

What is R*?*

Defined by the R Project for Statistical Computing as a language and environment for statistical computing and graphics,[1] R is an open-source free language primarily designed to be used to perform statistics.

It is a suite of tools and facilities for data manipulation, analysis, and visualisation. The three main features of R are:

- It is an effective data handling and manipulation facility
- It is a modular collection of functions and packages that allow processing and analysis
- It is a simple, readable, yet effective and fully-functioning programming language

In recent times, R has seen a surge in popularity. Rexer, a company which handle big data analysis and also conduct annual surveys of data scientists worldwide, found in 2015 that R is the most common primary-use statistical package, and this has remained the case since.[2]

[1] R Core Team, *R: A Language and Environment for Statistical Computing* (Vienna, Austria: R Foundation for Statistical Computing, 2016), `https://www.R-project.org/`.

The running citations in this document are formatted according to the Chicago Manual of Style (17th Edition) because of restrictions with LaTeX. However, I have also included a bibliography at the end of the document in the Harvard (Cite Them Right) format, which should be easier to read.

[2] Rexer, "Data Science Survey," 2015, `https://www.rexeranalytics.com/data-science-survey.html`.

Starting off in R

To begin working in R, we have to first install R and RStudio, a helpful Integrated Development Environment (IDE, essentially a software that makes it easier to work with a programming language).

Installing R

First, head to the Comprehensive R Archive Network (CRAN) website[3] and click the link to download the appropriate version of R for your operating system.

On Windows:

Ensure you download the base version!

When the file is downloaded, find it in the Downloads section of your browser, and click it to open. Alternatively, find it in the Downloads folder of your computer, and double-click it to open.

Follow the default instructions and complete the installation.

On Mac:

Download the .pkg file

[3] `https://cloud.r-project.org/`

When the file is downloaded, find it in the Downloads section of your browser, and click it to open. Alternatively, find it in the Downloads folder of your computer, and double-click it to open.

Follow the default instructions and complete the installation.

On Linux:

Follow Jason French's guide[4] on how to install R on Linux.

[4] `https://www.jason-french.com/blog/2013/03/11/installing-r-in-linux/`

Installing RStudio

Once you've installed R, head to the RStudio download page[5] and scroll down till you find the list of "Installers for Supported Platforms" for RStudio Desktop. Download the appropriate installer and run it, keeping everything to default.

[5] `https://rstudio.com/products/rstudio/download/`

'Speaking' in R

Objects

R treats data in the form of objects, called **vectors** (we will refer to them as objects, however). These objects can have values or data assigned to them.

R can be used as a complex calculator:

```
1 + 26 * (152 / 3)
```

```
## [1] 1318.333
```

And you can create new objects using the assignment symbol[6], `<-`

[6] The shortcut in RStudio for this is `Alt + -`

```
object <- 2+2
data <- "This object contains some data"
```

You can always remember that `<-` means `gets`. So `object <- value` is read as `object gets value`.

You can call objects that you have assigned data to by typing in their name:

```
object
```

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

```
data
```

```
## [1] "This object contains some data"
```

Object names can be anything as long as they start with a letter and have no special characters in them except `.` and `_`. I personally use a mix of `snake_case_object_names` and `CamelCaseObjectNames` but it's good to be consistent.

Also, very importantly: **R is case-sensitive**. In the words of Hadley Wickham, there is an implied contract between you and R - it will do all the tedious computation for you, but in return you must be completely precise in your instructions.[7]

[7] Hadley Wickham and Garrett Grolemund, *R for Data Science* (California: O'Reilly Media, 2016).

Functions

R interprets, manipulates, and analyses data using **functions**. Functions are typically in the format `function(object, arguments)` where `function` is the name of the function (e.g. `mean()`, `sum()`), `object` is the name of the object which holds the data in question, and `arguments` are any arguments (little tweaky settings which fine-tune the working of the function to your specifications) that the function requires to… function.

For instance, `rnorm` is a function that generates a random normal distribution. It doesn't need an object, but it does need three arguments: `n` for the sample size, `mean` for the mean of the distribution, and `sd` for the standard deviation.

```
rnorm(n = 4, mean = 3.4, sd = 1)
```

```
## [1] 3.869706 4.971227 3.838481 4.367909
```

However, in order to work with the results of a function downstream, we must store it in an object. We use the assigner `<-` for that. Remember that the assigner is read as `gets`.

```
normal <- rnorm(n = 300, mean = 24, sd = 6)
```

After storing data in an object, we can analyse it further - let's use the `mean()` and `sd()` functions to find the mean and sd of our random normal distribution. Then, let's use the `boxplot()` function to make a simple boxplot of our distribution.

```
mean(normal)
```

```
## [1] 24.41798
```

```
sd(normal)
```

```
## [1] 5.532878
```

```
boxplot(normal)
```

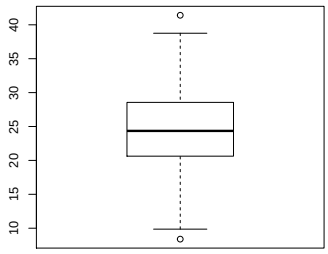


Figure 1: A simple box plot using the 'base' graphics in R

We'll get to visualising data in a bit.

Packages

Functions and public-assessible data are collected together into packages, which are freely available online (either on CRAN or GitHub, a popular software development website).

In order to use functions in a particular package, you must install it first using `install.packages('PackageName').^[*Nb.* We are using double quotation marks with`install.packages()`and none with`library()`']`

Once the package is installed, you have to let R know you want to use it by running `library(PackageName).

If you want to uninstall a package, you can use `remove.packages('PackageName')`.

An Ode to the Pipe

In recent times, there has been a passionate emphasis in the data science community on **tidy** data. Tidy data is data which is organised in a specific structure, which allows each column to hold a variable (`col1, col2, col3` correspond to `x1, x2, x3`, etc.), and each row represents a data-point, or an observation. This way, variables can be selected, and observations can be filtered with ease.

The `tidyverse` suite of packages, spearheaded by Hadley Wickham, has been a massive influence in the turn towards tidy data, and one of the key packages in the tidyverse is `dplyr` - its primary function is data wrangling.

`dplyr` uses a particular function called the pipe function `%>%`[8]. `%>%` is read as `then`, and it places whatever is to the left of it as the first argument of what is on the right of it.

[8] You can input it in RStudio by using Shift + Ctrl + M or Shift + Cmd + M

For instance, `x %>% y()`, read as `x, then y of x` is the same as `y(x)`. Further, `x %>% y(arg)` is the same as `y(x, arg)` where `arg` is one or a set of arguments passed to the function y.

The power of this function comes when you're nesting functions within other functions - it allows you to read R like English sentences, and is really handy.

For example, if we want to generate a random normal distribution of 100 points (using `rnorm()`), get the absolute value for each point (using `abs()`), log them (using `log()`), then get the mean (using `mean()`), we could do it in three ways:

The step-by-step approach:

```
r <- rnorm(n = 100)
a <- abs(r)
l <- log(a)
mean(l)
```

```
## [1] -0.4954771
```

This approach is really tedious and tends to clutter up the environment with unnecessary ephemeral objects.

The nested functions approach:

```
mean(log(abs(rnorm(n = 100))))
```

```
## [1] -0.5561881
```

While this is concise and doesn't generate any objects, it is also very confusing to read, as you have to read the function from the inside out. This approach is effective to an extent, but difficult to interpret when you have a massive script of many lines of code.

The pipeline (or dplyr) approach:

```
library(dplyr)
rnorm(n = 100) %>%
  abs() %>%
  log() %>%
  mean()
```

```
## [1] -0.5713404
```

This approach is so much more readable! It literally reads like an imperative English sentence:

generate rnorm with n = 100, *then* get abs of that, *then* get log of that, *then* histogram that

The Workflow in R - Reading or Generating Data

Importing data from files

R can read data from all major file formats (including .csv, .txt, .tsv, .xls, .sav) and preserves them in the environment as data frames.

For example, the function read.csv is used to read a csv file. We'll read a publicly available csv, but you can specify a path to your own file (more on that later).

`head()` is a function that displays the head (i.e. the first few rows, usually 6) of the data. This is to save space and make sure we don't display huge datasets in the handout!

I know this is the wrong kind of biostats (it's biometric statistics), but that's on purpose!

```
biostats <- read.csv('https://people.sc.fsu.edu/~jburkardt/data/csv/biostats.csv',
                     header = T)
head(biostats)
```

```
##   Name      Sex Age Height..in. Weight..lbs.
## 1 Alex        M  41          74          170
## 2 Bert        M  42          68          166
```

```
## 3 Carl      M  32          70          155
## 4 Dave      M  39          72          167
## 5 Elly      F  30          66          124
## 6 Fran      F  33          66          115
```

To import local data (i.e. files stored on your computer), you can nest the `choose.files()` function inside the function that reads the data. For example, if I want to read a .tsv file:

```
read.table(choose.files(), sep = '\t', header = T)
#the sep argument defines the separating character
#in this case it is a tab, signified by \t
```

This will generate a dialog box, which you can use to select files.

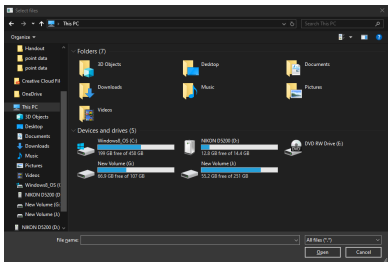


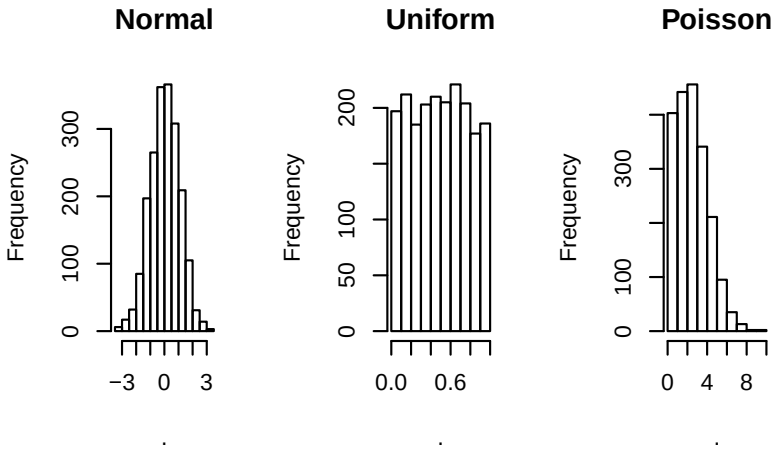
Figure 2: A choose.files() dialog box

Generating random data

R can also generate random[9] data with little to no effort. There are many functions available in base R[10] which you can use to generate data, including `rnorm` for a random normal distribution, `runif` for a random uniform distribution, `rpois` for a random poisson distribution, etc.

Let's experiment with generating some distributions:

```
par(mfrow = c(1,3))
rnorm(n = 2000) %>% hist(main = "Normal")
runif(n = 2000) %>% hist(main = "Uniform")
rpois(n = 2000, lambda = 3) %>% hist(main = "Poisson")
```



[9] Actually, as the data is generated based on an algorithm, it is only pseudo-random. However, the algorithm is set to a near-random seed every time it is run, so the data generated is effectively random.

[10] base refers to functions that exist in R without depending on any other packages

The Workflow in R - Data Wrangling

Remember how the core tenet of tidy data is that all the data must be accessible? This means that each row must correspond to a variable,

and each observation to a column.

Now if we were to look at some untidy data (the `language_diversity` dataset included within the `untidydata` package):

```
library(untidydata)
df <- language_diversity %>%
  data.frame()
head(df, 12)
```

```
##    Continent      Country Measurement Value
## 1     Africa      Algeria       Langs    18
## 2     Africa       Angola       Langs    42
## 3    Oceania    Australia       Langs   234
## 4       Asia   Bangladesh       Langs    37
## 5     Africa        Benin       Langs    52
## 6   Americas      Bolivia       Langs    38
## 7     Africa     Botswana       Langs    27
## 8   Americas       Brazil       Langs   209
## 9     Africa Burkina Faso       Langs    75
## 10    Africa          CAR       Langs    94
## 11      Asia     Cambodia       Langs    18
## 12    Africa     Cameroon       Langs   275
```

We can use the `str` function to ask R for the structure of this data frame.

```
str(df)
```

```
## 'data.frame':    444 obs. of  4 variables:
##  $ Continent  : chr  "Africa" "Africa" "Oceania" "Asia" ...
##  $ Country    : chr  "Algeria" "Angola" "Australia" "Bangladesh" ...
##  $ Measurement: chr  "Langs" "Langs" "Langs" "Langs" ...
##  $ Value      : num  18 42 234 37 52 38 27 209 75 94 ...
```

If we were to inspect the whole dataset (using `View(df)`), we would be able to see that there are 4 variables: Continent, Country, Measurement, and Value. The messiness in this data comes from the fact that Country has a repetition for each value in Measurement.

Country	Measurement	Value
Algeria	Langs	x
Algeria	Area	y
Algeria	Population	z
Bhutan	Langs	a
Bhutan	Area	b
Bhutan	Population	c

This is long data that doesn't really say much. Further, there are variables hidden inside the Measurement column, which we can extract and spread out into a wider dataframe. We will use `spread()` from the package `tidyr` for this. Spread takes a 'key', which is the column to split into multiple columns, and a 'value', which is a column or a set of columns to get data from while spreading, and places it all into a dataframe.

```
library(tidyr)
spreaded <- df %>%
  spread(key = Measurement, value = Value)
head(spreaded)
```

```
##   Continent      Country    Area Langs  MGS Population Stations  Std
## 1    Africa      Algeria 2381741    18 6.60      25660      102 2.29
## 2    Africa       Angola 1246700    42 6.22      10303       50 1.87
## 3    Africa        Benin  112622    52 7.14       4889        7 0.99
## 4    Africa     Botswana  581730    27 4.60       1348       10 1.69
## 5    Africa Burkina Faso  274000    75 5.17       9242        6 1.07
## 6    Africa     Cameroon  475422   275 9.17      12239       35 1.75
```

Suddenly, the data is a lot more clean and usable. Each measurement is split into different columns, and now we can work with the columns as we would expect. For example, we can filter out records from Africa and get the mean Area.

```
spreaded %>%
  select(Continent, Area) %>%
  filter(Continent == 'Africa') %>%
  summarise(mean = mean(Area))
```

```
##       mean
## 1 789654.2
```

Can you figure out what exactly we have done in this line of code?

The process of data wrangling and tidying in R is complicated, and there is no real catch-all. A great place to start is learning the core principles behind tidy data,[11] and exploring the `tidyverse` packages.

[11] Hadley Wickham, "Tidy Data," *Journal of Statistical Software* 59, no. 10 (2014), https://doi.org/10.18637/jss.v059.i10.

The Workflow in R - Exploration

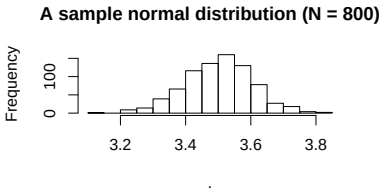
Usually, when you have a large dataset densely packed with information, it's not easy to find patterns in it. This is where data exploration comes in: through understanding the distribution, measuring sameness (central tendency) and differentness (variance within data), looking for patterns in our data through visualisation, and perhaps building models to explore the data more deeply.

Distribution

Going by the Central Limit Theorem of probability,[12] which states that as the number of independent random samples and variables with unbiased measurement increase, any dataset tends towards normality. So we can assume that with a large enough sample size, data collected in the field will usually tend towards a normal distribution.

Normal distributions can be generated[13] using `rnorm()`, specifying N, mean, and sd.

```
rnorm(n = 800, mean = 3.5, sd = 0.1) %>%
  hist(main = "A sample normal distribution (N = 800)")
```



As normality is an assumption for parametric tests, we have to test for normality by checking the data in several ways. In order to do this, we visualise data (using boxplots, histograms, and/or Quantile-Quantile plots), and conduct a frequentist test. The Shapiro-Wilk test has the best power for a given significance,[14] and so we will use that.

Let's test the normality of the `Weight..lbs` variable in the `biostats` dataset from earlier. The function `par(mfrow = c(x,y)` splits the plotting window into x rows and y columns. Here, I will use 1 row and 3 columns.

```
par(mfrow = c(1,3))

#Boxplot
boxplot(biostats$Weight..lbs.)

#Histogram
hist(biostats$Weight..lbs.)

#QQPlot
qqnorm(biostats$Weight..lbs.)
qqline(biostats$Weight..lbs.)
```

We can see that while the data does not have many samples (`n = 18`), it does not show massive divergence from the normal Q-Q line, and so we *might* be safe in assuming normality. However, we should confirm this by conducting a Shapiro-Wilk test - this is done in R using `shapiro.test()`.

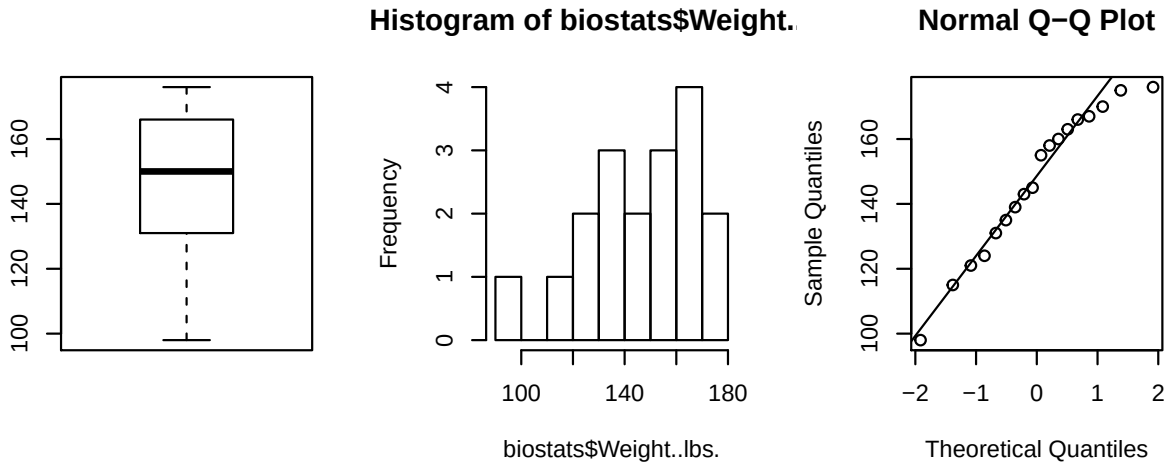
```
shapiro.test(biostats$Weight..lbs.)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  biostats$Weight..lbs.
## W = 0.94659, p-value = 0.374
```

[12] Patrick Billingsley, *Probability and Measure*, 3rd ed. (Hoboken, NJ: John Wiley & Sons, 1995).

[13] see *Generating random data*

[14] Nornadiah Razali and Yap Bee Wah, "Power Comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling Tests," *Journal of Statistical Modeling and Analytics* 2, no. 1 (2011): 21–33.



By the way that the Shapiro-Wilk test works, a non-significant p-value is interpreted as the data being normal[15]. So this confirms that the data is, in fact, normal.

Can you think of sampling situations where the data you collect can be assumed to be normal?

However, other distributions obviously exist. When dealing with some kinds of natural data, it is quite common to encounter a **Poisson** distribution - i.e. one that is inflated in the lower values[16]

Usually, count data tends to fall into Poisson distributions, as it is more likely for you to get lower count values than higher - think about it, is it more likely to spot 3 tigers in one survey or 36?

In order to test for a Poisson distribution, we can again use visualisation - this time a histogram would be enough to show the low-value inflation. Let's test the `Area` variable in the `spreaded` dataset from earlier. I am also going to scale the values in `Area` by percentage, by dividing each by the maximum value, and then multiplying that with 100.

```
spreaded %>%
  transmute(percarea = (Area/max(Area))*100) %>%
  #This function mutates the data as specified
  pull() %>%
  #This function 'pulls' the transmuted data into a vector
  hist(main = "Area")
```

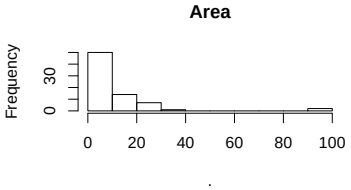


Figure 3: This is what a Poisson distribution usually looks like. Note the higher frequencies in the lower values

However, for a frequentist test, we have to test one of the assumptions of a Poisson distribution: that **the mean and the variance of the data are equal**. For this, we will use functions `mean()` and `var()`, and see if the difference between them is 0. If the variance is greater or

[15] The null hypothesis for a Shapiro-Wilk test is that the data is normal, and as we have a non-significant p-value (i.e. > 0.05), we have no reason to reject the null hypothesis.

[16] While this is true, the actual definition and workings of a Poisson distribution and function are a lot more complex and depend on things like the inter-observation 'time' and the effective 'rate' of sampling, which are summarised into a λ parameter. This is far beyond the scope of this handout, but I recommend reading up about it on Wikipedia.

lesser than the mean, then this is a special case called over- or under-dispersion, and we won't be looking into that for now[17].

[17] There are many approaches to modelling overdispersed or underdispersed Poisson data; you could use a generalised linear model with a negative binomial error distribution to account for overdispersion, for instance.

```
mean(spreaded$Area)
```

```
## [1] 880698.2
```

```
var(spreaded$Area)
```

```
## [1] 1.957758e+12
```

```
mean(spreaded$Area) - var(spreaded$Area)
```

```
## [1] -1.957757e+12
```

There is an *enormous* amount of variance in the data[18], and we can safely say that it is overdispersed. However, we won't be getting into what you should do about overdispersed Poisson data (or normal Poisson data for that matter) for now.

[18] If you're not familiar with exponent notation in computers, 1.95e+12 signifies 1.95×10^{12}. This is massive variance.

Just know that not all data in the natural world is normally distributed, and you assume normality at your own peril! Always test, test, test, like a good chef.

Sameness and differentness

As much as knowing how our data is distributed helps, we also need summarise or describe it somehow. Measures of central tendency and differentness are great for that. We won't be going into depth about the definitions of these measures, but R is capable of calculating many such summary statistics:

```
mean(spreaded$MGS) #Mean
```

```
## [1] 7.028919
```

```
median(spreaded$MGS) #Median
```

```
## [1] 7.355
```

```
sd(spreaded$MGS) #Standard Deviation
```

```
## [1] 3.135325
```

```
var(spreaded$MGS) #Variance
```

```
## [1] 9.830262
```

The function `c()` is a simple concatenator (in fact, the c stands for concatenate). It puts the elements inside it together as a vector. For instance, 'c(x, y, z) would return a vector {x, y, z}.

```
c(min(spreaded$MGS), max(spreaded$MGS)) %>%
  print() #Minimum and maximum
```

```
## [1]  0 12
```

To calculate standard error, however, we have to use the estimation formula $s_x = \frac{s}{\sqrt{n}}$ where s is the standard deviation of the sample, and n is the sample size.

```
sd(spreaded$MGS) / sqrt(length(spreaded$MGS))
```

```
## [1] 0.3644741
```

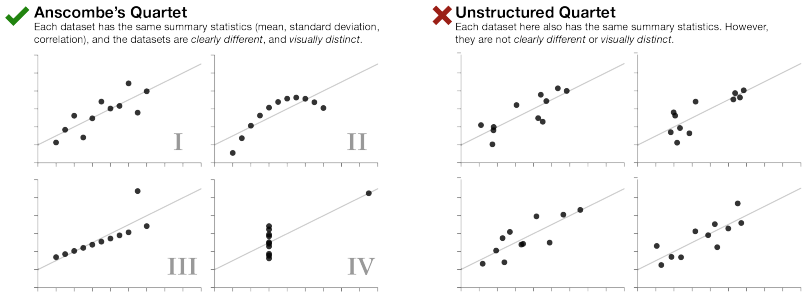
Visualisation

> Figures often beguile me, particularly when I have the arranging of them myself; in which case the remark attributed to Disraeli would often apply with justice and force: 'There are three kinds of lies: lies, damned lies, and statistics.' - Mark Twain

Finally, we come to one of the most important (and exciting) parts of the process of data exploration: visualisation. This is the process by which we plot our data to look for patterns in it.

Visualisation is a very powerful tool, because it is such a subjective, human experience. When you describe, summarise, or model data, the algorithms that do that are based around a set of assumptions. Because of these predefined assumptions, it's fundamentally impossible for the model to act in a way you didn't expect it to, or show patterns in the data that truly surprise you. However, with the right kind of visualisation, the data has the potential to surprise you. This is why it is never good to trust summary statistics or modelling alone.

A great example of the power and the importance of viz is Anscombe's Quartet.[19] It is a collection of four datasets with the same summary statistics, but radically different distributons when visualised.



A modern-day reinterpretation of the Anscombe Quartet is the Datasaurus Dozen,[20] aptly named for the clever girl hidden in there.

There are three ways to visualise data in R - `base` graphics, `lattice`, and `ggplot2`. We won't be touching on `lattice` as I personally think the package is not worth the effort of learning it. However, `base` and `ggplot2` are great in their own ways.

[19] Francis John Anscombe, "Graphs in Statistical Analysis," *American Statistician* 27, no. 1 (1973): 17–21, `https://doi.org/10.1080/00031305.1973.10478966`.

[20] Justin Matejka and George Fitzmaurice, "Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics Through Simulated Annealing," *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017, 1290–4, `https://doi.org/10.1145/3025453.3025912`.

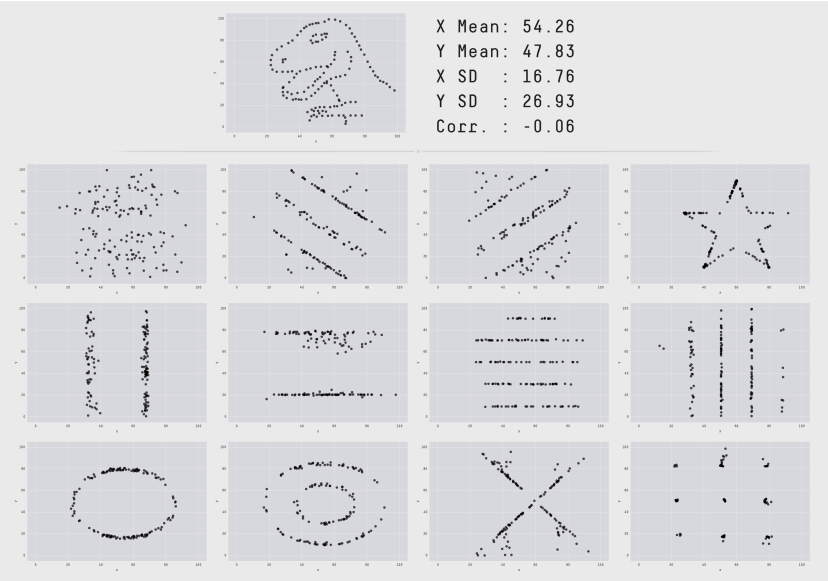


Figure 4: The Datasaurus Dozen: rawr data, am I right?

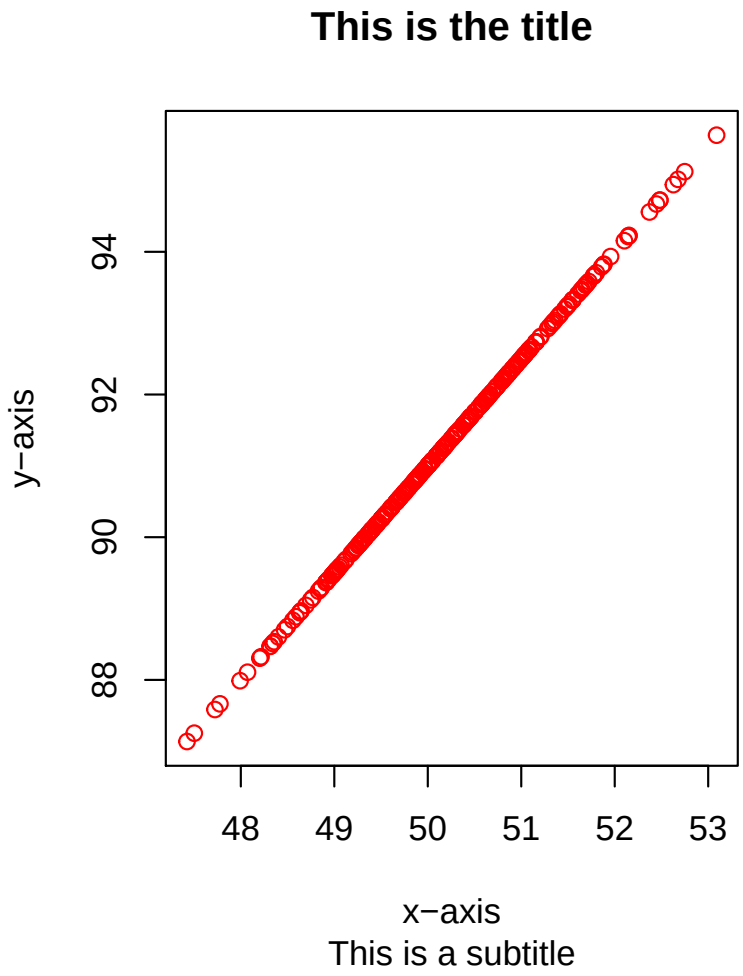
base

The graphics capabilities of `base` are simple, and most people start plotting in R using functions from it, e.g. `hist()`, `boxplot()`, and the catch-all `plot()`. The main advantage of `base` graphics is that they are included within the R architecture by default, which means you don't have to install or load any additional packages. However, this is really not a decisive advantage as modern computers have more than enough RAM to be able to load `ggplot2`, which is the best graphics system in R.

`base` graphics are usually simplistic and basic, with little customisability. All the plots up until now have been `base`-based, and they have been quite uncomplicated, and also in a way inelegant. Further, the scope of customisability in `base` graphics is limited.

```
x <- rnorm(300, 50, 1)
y <- 16 + (1.5 * x)

plot(x, y,
     main = "This is the title",
     xlab = "x-axis",
     ylab = "y-axis",
     sub = "This is a subtitle",
     col = 'red')
```



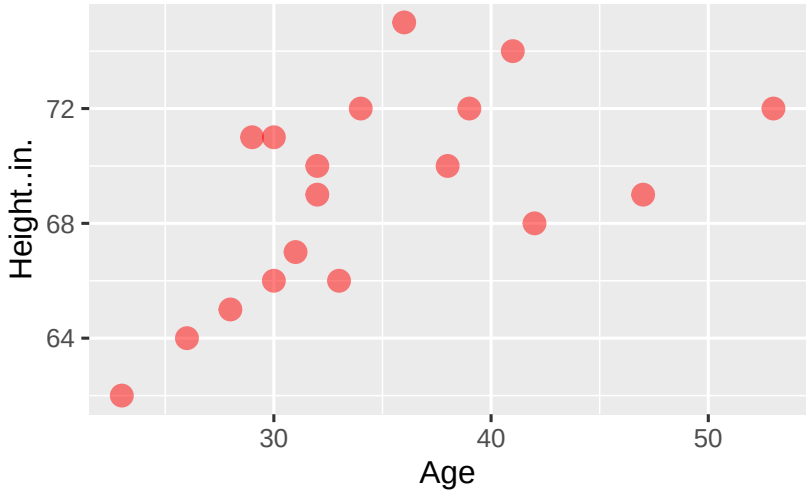
ggplot2

ggplot2, developed by Hadley Wickham,[21] is an incredibly powerful graphics engine, and can be used in R by loading the package ggplot2. The way it works is very similar to the ideology behind dplyr, in that graphics are written with a particular grammar. This makes it so that plots can be built in terms of layers, as opposed to the very one-step pen-and-paper plotting in base. Additionally, plots can be stored in objects, to which we can append more information.

I'll demonstrate the modularity and power of ggplot2 using the biostats data from earlier.

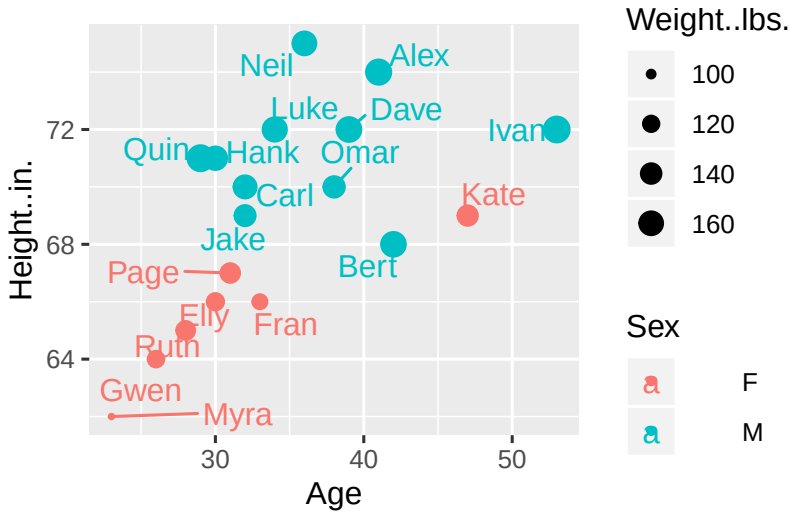
[21] "ggplot2: Elegant Graphics for Data Analysis," *Journal of Statistical Software* 35, no. 1 (2010).

```
library(ggplot2)
plot <- ggplot(biostats, aes(x = Age, y = Height..in.,
    shape = Sex, colour = Sex))
plot + geom_point(colour = "red", size = 5,
    alpha = 0.5, shape = 20)
```



We could choose to size the points by the Weight variable and colour them by Sex, and then label each point using `ggrepel`:

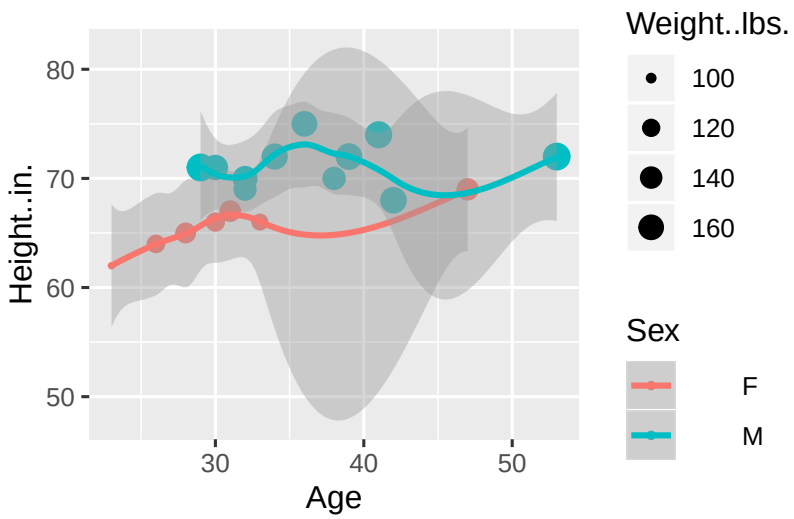
```
library(ggrepel)
plot +
  geom_point(aes(size = Weight..lbs.), shape = 20) +
  geom_text_repel(label = biostats$Name)
```



Maybe not the labels. What if I want to draw a loess (locally estimated scatterplot smoothing) line through each group in the data?

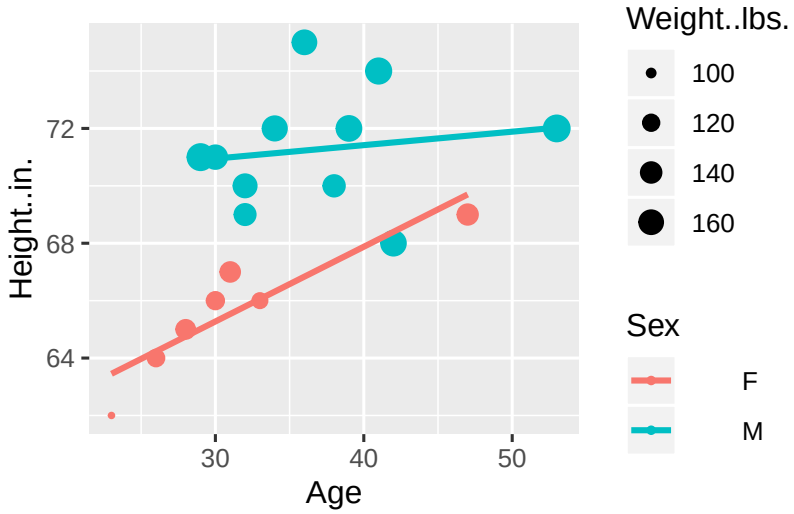
```
plot +
  geom_point(aes(size = Weight..lbs.), shape = 20) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



The dark area represents the 95% CI of the data. However, I don't really like it, and I don't like the loess line either. I can plot linear regression lines for Males and Females separately, and then disable the CI shading:

```
plot +
  geom_point(aes(size = Weight..lbs.), shape = 20) +
  geom_smooth(method = lm, se = FALSE)
```



So much nicer. As you can see, `ggplot2` is based around the idea of specifying a particular set of data and parameters to plot (using the `ggplot` function), then adding (using `+`) any number of elements (`geom_`, `stat_`, etc.) to the plot, with the ability to modify them or remove them at any point, without affecting the main plot itself.

`ggplot2` is the industry standard in publication-quality figures these days, and for good reason. With enough practice, you'll be making beautiful viz like this:

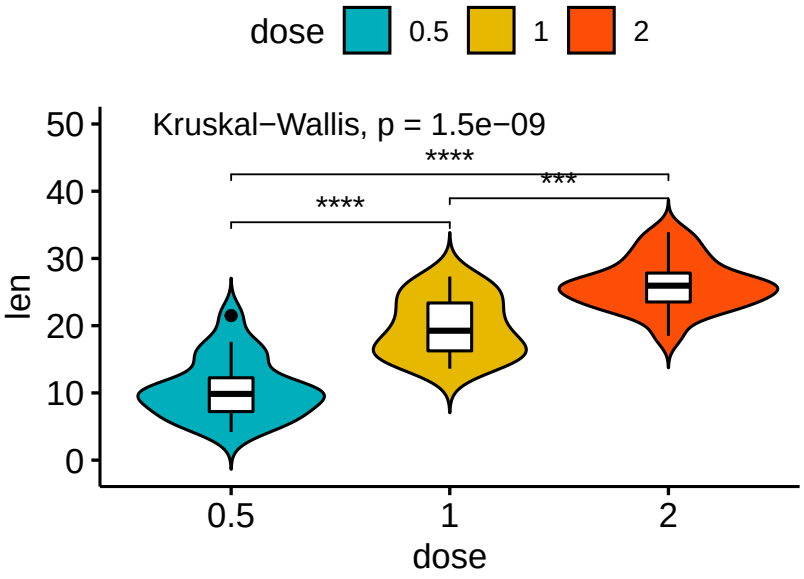
```
library(ggpubr)
data("ToothGrowth")
df <- ToothGrowth

my_comparisons <- list(c("0.5", "1"),
                       c("1", "2"),
                       c("0.5", "2"))

ggviolin(df, x = "dose", y = "len", fill = "dose",
         palette = c("#00AFBB", "#E7B800", "#FC4E07"),
         add = "boxplot",
         add.params = list(fill = "white")) +

    stat_compare_means(comparisons = my_comparisons,
                       label = "p.signif") +

    stat_compare_means(label.y = 50)
```



It would be intensely complicated to try to do this in `base`, so much so that I don't really want to attempt it!

In conclusion, always visualise your data while exploring it. In the eternal words of Forrest Gump, *data is like a box of chocolates - you never know what you're gonna get.*

Modelling

We won't be working with modelling as an exploratory tool right now - for that, I very much recommend reading Hadley Wickham's works on R and its applications in data science, particularly 'R for Data Science.'[22] Further, you can also read Matthew J C Crump's[23] introductory textbook to stats, which is based almost entirely around the capabilities of R.

However, we will be looking at hypothesis testing, as many of us will be using statistics for these purposes. We will run through 3 basic models: a paired 2 sample t-test, a Kruskal-Wallis test (with 3 groups)[24], and a basic linear regression. We won't be going through the mathematics behind these statistical tests - you can read Crump for a very detailed explanation of most common statistical tests.

[22] Wickham and Grolemund, *R for Data Science*.

[23] *Answering Questions with Data (Textbook): Introductory Statistics for Psychology Students*, 2019, `https://doi.org/10.17605/OSF.IO/JZE52`.

[24] As a non-parametric alternative to a one-way ANOVA

Paired 2-sample t-test

First, let's understand the data. This data is the result of an experiment where captive dung beetle activity was measured (in scaled units of walking activity). These beetles were subjected to two different temperatures: 15 °C and 20 °C. We want to figure out if there is a significant difference in the activity levels of these beetles between the two temperature treatments.

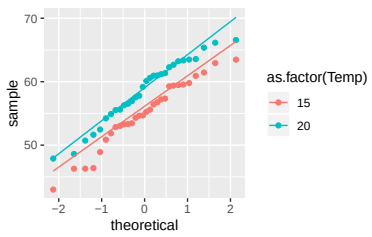
First, we'll read in the data, and then test the assumptions of a paired t-test: normality of the response variable using `shapiro.test`, and homogeneity of variances[25] between the two samples using `var.test`.

[25] Homoskedasticity

```
data <- read.table("https://git.io/JeoIk",
                   header = T, sep = '\t') %>%
  data.frame()
head(data)
```

```
##   Temp Activity
## 1   15 63.47816
## 2   15 46.38115
## 3   15 59.28270
## 4   15 61.44609
## 5   15 53.03586
## 6   15 59.48380
```

```
#A simple Q-Q plot in ggplot2, which allows me to
#simultaneously plot activity values for both temperatures
ggplot(data, aes(sample = Activity, colour = as.factor(Temp))) +
  stat_qq() +
  stat_qq_line() +
  scale_fill_manual(values=c("#E69F00", "#56B4E9"))
```




```
shapiro.test(data$Activity)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  data$Activity
## W = 0.97725, p-value = 0.3235
```

The Q-Q plots for both variables show slight divergences from the expected Q-Q line. However, this is probably down to the small sample size, and as the Shapiro-Wilk test is non-significant, we have no reason to reject the null hypothesis, and so we can assume that the data is normal.

R can be given 'formulae', which look like `x ~ y`, where `x` is the response variable, `y` is the predictor variable, and the symbol `~` can be read as "depends on".

```
#we're specifying that Temp is a factor
var.test(data$Activity ~ as.factor(data$Temp))
```

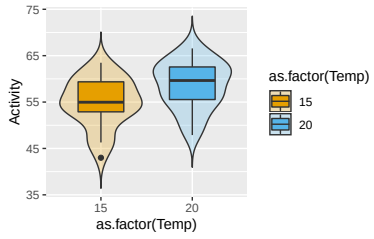
```
##
##  F test to compare two variances
##
## data:  data$Activity by as.factor(data$Temp)
## F = 1.0406, num df = 29, denom df = 29, p-value = 0.9153
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.4953027 2.1863572
## sample estimates:
## ratio of variances
##           1.040629
```

As the F test is also non-significant, we can assume that there is no significant difference between the variances of the two groups. So the assumption of homoskedasticity holds.

Let's draw up box-and-violin plots[26] for the two groups using `ggplot2`, and see if we can see any difference between them.

[26] For those unfamiliar, violin plots show the probability density of the data as the width of the violin. The wider the violin is, the more data is present at that value.

```
ggplot(data, aes(x=as.factor(Temp), y=Activity,
                 fill=as.factor(Temp))) +
  geom_violin(trim = FALSE, alpha = 0.25) +
  geom_boxplot(width = 0.5) +
  scale_fill_manual(values=c("#E69F00", "#56B4E9"))
```



It's clear that the activity at 15 °C is less than that at 20 °C. So we're expecting a negative t-statistic[27].

[27] as the 'observed' value (15 °C) is less than the 'expected' value (20 °C). See: Wikipedia/Student's t-test

Now that we've tested the assumptions and also visualised the data to look for patterns, all that remains is to run the t-test. We can set `alternative = 'less'` to refine the test by making it one-sided.

```
t.test(data$Activity ~ as.factor(data$Temp), paired = TRUE,
                                             alternative = 'less')
```

```
##
##  Paired t-test
##
## data:  data$Activity by as.factor(data$Temp)
## t = -7.5534, df = 29, p-value = 1.256e-08
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##       -Inf -2.831638
## sample estimates:
## mean of the differences
##               -3.653491
```

That's it - the t-test is significant, which means there's a significant difference in the activity levels between 15 and 20 °C. Further, the t-statistic itself is negative, which means that there is a decrease in the activity level as it gets warmer.

Kruskal-Wallis test

For this, we will be using a dataset of ant richness sampled between three types of leaf litter density - low, medium, and high. The richness is actually a count of individuals collected in pitfall traps at sample sites - can you remember what distribution we expect count data to fit?

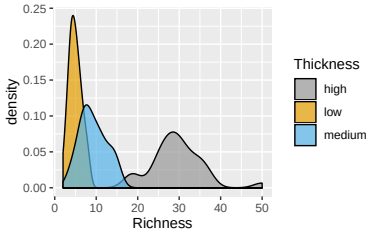
Figure 5: A pitfall trap in dense jungle. (c) Maurice Leponce.

```
df <- read.table("https://git.io/Jeoqq", header = TRUE, sep = '\t') %>%
  data.frame()

head(df)
```

```
##   Thickness Richness
## 1       low        4
## 2       low        3
## 3       low        5
## 4       low        2
## 5       low        5
## 6       low        5
```

```
ggplot(df, aes(x = Richness, fill = Thickness)) +
  geom_density(alpha = 0.7) +
  scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))
```



It's quite clear that the data follows a Poisson distribution. It's also quite clear that the variances within the groups are very different. This breaks both the assumptions of One-way ANOVA, and so we need to use a non-parametric alternative, the Kruskal-Wallis test.

```
kruskal.test(Richness ~ as.factor(Thickness), data = df)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  Richness by as.factor(Thickness)
## Kruskal-Wallis chi-squared = 71.984, df = 2, p-value = 2.339e-16
```

The test comes out significant, which means that there is a significant change in the richness between the three thickness types. We can further investigate this by doing pairwise comparisons of means - usually this is done by using a pairwise t-test, but as our analysis is non-parametric, we will use a pairwise Wilcoxon signed-rank test, correcting the p-value by the Bonferroni method[28].

[28] see Wikipedia/Multiple comparisons problem

```
pairwise.wilcox.test(df$Richness, as.factor(df$Thickness),
                     p.adjust.method = 'bonferroni')
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test
##
## data:  df$Richness and as.factor(df$Thickness)
##
##        high    low
## low    8.0e-11 -
## medium 8.6e-11 3.8e-07
##
## P value adjustment method: bonferroni
```

We can see that there are significant pairwise differences between each pair of groups. So we can say that leaf litter density has a significant effect on ant richness.

Linear regression

Many of the complex models you will encounter in day-to-day statistics are based around the core concepts underlying linear (or ordinary least squares).

A linear model is computed by plotting the data in space, and then finding the best-fit line that passes through the data in such a way that it has the minimum values of the squares of the residuals (i.e. the distance between the points and the line). This way, the line is said to fit the data well, and so the slope and intercept of the line describe the linear model[29].

[29] Always remember $y = \alpha + \beta(x)$.

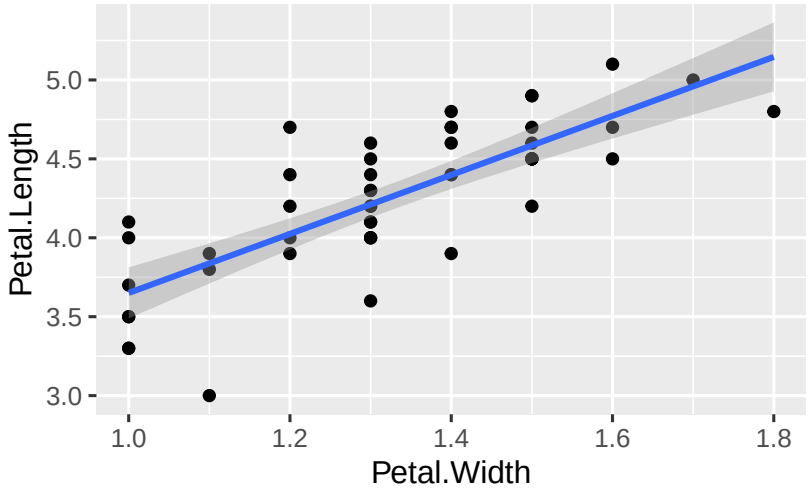
Let's do a quick regression. The data we will use is `iris`, a collection of measurements (Sepal Length, Sepal Width, Petal Length, Petal Width) of flower characters in three species of the genus *Iris*: *I setosa*, *I versicolor*, and *I virginica*. We will look only at *I versicolor* data, and try to see if there is a relationship between the Petal Width and Petal Length, to see if we can predict how long a petal will be if we know how wide it is.

This will take a little bit of wrangling with `dplyr`, after which we can plot the data in an xy scatter plot, with the best fit line running through it[30]

[30] `ggplot2` is capable of computing the best fit linear model line.

```
versicolor <- iris %>%
          filter(Species == "versicolor") %>%
          select(Petal.Width, Petal.Length)

ggplot(versicolor, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point() +
  geom_smooth(method = 'lm')
```



The line shows a clear positive correlation, which means we might see a significant result in our regression.

```
lm <- lm(Petal.Length ~ Petal.Width, data = versicolor)
summary(lm)
```

```
##
## Call:
## lm(formula = Petal.Length ~ Petal.Width, data = versicolor)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.8375 -0.1441 -0.0114  0.1984  0.6755
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.7813     0.2838   6.276 9.48e-08 ***
## Petal.Width   1.8693     0.2117   8.828 1.27e-11 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2931 on 48 degrees of freedom
## Multiple R-squared:  0.6188, Adjusted R-squared:  0.6109
## F-statistic: 77.93 on 1 and 48 DF,  p-value: 1.272e-11
```

Firstly, if we diagnose the regression:

- the p-value shows that the model is significant.
- the adjusted R^2 is higher than 0.5, which means the model is performing better than random.
- the residual standard error is not extremely large.

The model performed moderately well.

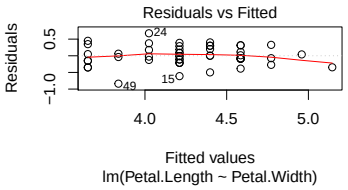
Secondly, we can interpret the results - as the β coefficient of petal width is significant, we can infer that it is possible to predict petal length from petal width. Also, as the coefficient is positive, we can infer that an increase in petal width leads to an increase in petal length.

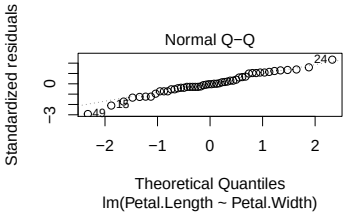
However, we also have to diagnose the residuals, to ensure that the regression fulfils its main assumptions: normality of residuals, homogeneity of variance in the residuals, and no overly influential data points. We can do this visually, by graphing the residuals[31]:

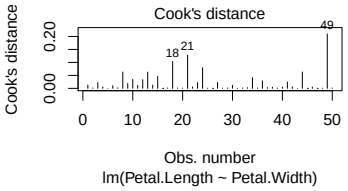
```
plot(lm, which = c(1,2,4))
```

While not incredible, this model fits the assumptions. The normality of the residuals is observed by the Normal QQ plot; the homogeneity of variance is seen by there not being any clear patterns or deviations in the Residuals vs Fitted plot; and the lack of overly influential points is seen in the Residuals vs Leverage plot, as there don't seem to be any points which have reached a leverage of 3 times the mean leverage (the soft cutoff for an outlier).

[31] I'm using `base` for this, because the `ggplot2` plotter for residuals doesn't like PDFs







Multivariate Modelling

Multiple regression

Now that we have looked at basic univariate models in R, let's touch on some multivariate models too. For instance, linear regression can be extended into multiple regression easily - we can test if there's a significant effect of Sepal Width, Sepal Length, and Petal Width on Petal Length, and if there's a significant difference in Petal Length between the three species of *Iris* - all in one model.

We won't be plotting or diagnosing the model, as this is just a demonstration of multiple regression.

```
df_iris <- iris %>% na.omit() %>% data.frame()
lm2 <- lm(df_iris$Petal.Length ~ df_iris$Sepal.Width +
                                 df_iris$Sepal.Length +
                                 df_iris$Petal.Width +
                                 df_iris$Species)
summary(lm2)
```

```
##
## Call:
## lm(formula = df_iris$Petal.Length ~ df_iris$Sepal.Width + df_iris$Sepal.Length +
##     df_iris$Petal.Width + df_iris$Species)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.78396 -0.15708  0.00193  0.14730  0.65418
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                -1.11099    0.26987  -4.117 6.45e-05 ***
## df_iris$Sepal.Width        -0.18052    0.08036  -2.246   0.0262 *
## df_iris$Sepal.Length        0.60801    0.05024  12.101  < 2e-16 ***
## df_iris$Petal.Width         0.60222    0.12144   4.959 1.97e-06 ***
## df_iris$Speciesversicolor   1.46337    0.17345   8.437 3.14e-14 ***
## df_iris$Speciesvirginica    1.97422    0.24480   8.065 2.60e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2627 on 144 degrees of freedom
## Multiple R-squared:  0.9786, Adjusted R-squared:  0.9778
## F-statistic:  1317 on 5 and 144 DF,  p-value: < 2.2e-16
```

So from this model we can infer:

- Sepal length and petal width are significant predictors of petal length,

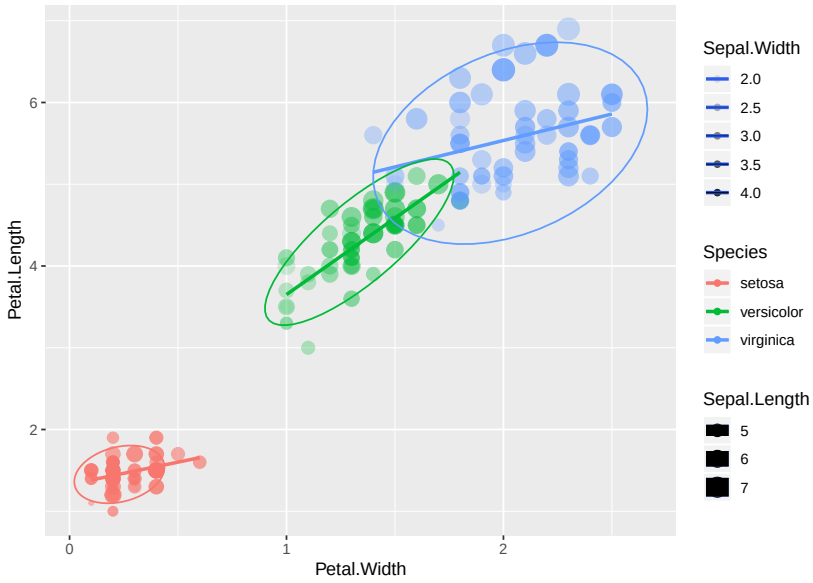
and they both have a positive relationship with it, i.e. an increase in sepal length and petal width leads to an increase in petal length.

- Sepal width is also a significant predictor of petal length, but it has a small effect, and a negative relationship with it.
- There is a significant difference in petal lengths between the three species - *versicolor* and *virginica* have larger petal lengths than *setosa*.

That was easy! This is where R comes out stronger than most of its competitors - expanding and increasing the scale of your analysis is incredibly easy. I could very easily conduct large scale analyses on big data orders of magnitude larger than this dataset, and the functions R uses would scale perfectly across them.

As a final note to multiple regression - we can also make a complicated plot which is really hard to read. This is not really the best way to visualise multiple regression - the best way to do it would be to make a box or violin plot with comparisons of means. Don't try this at home!

```
g <- ggplot(df_iris, aes(x = Petal.Width, y = Petal.Length,
                         size = Sepal.Length,
                         alpha = Sepal.Width,
                         colour = Species)) +
     geom_point() +
     geom_smooth(method = 'lm', se = FALSE) +
     stat_ellipse()
g
```



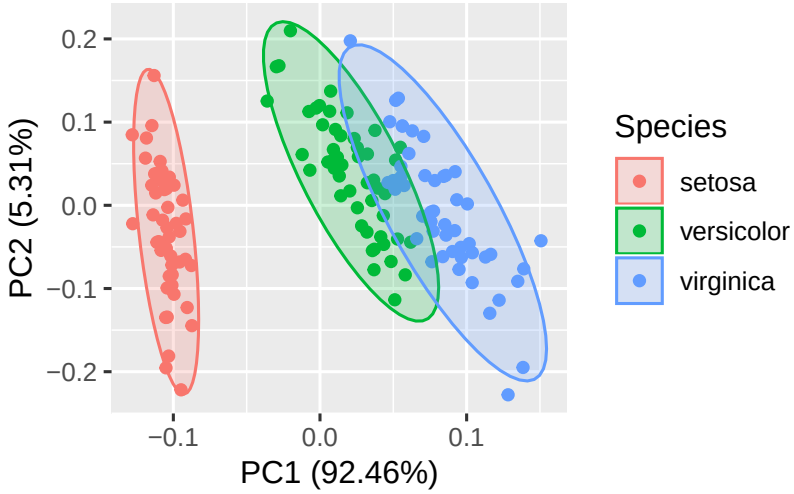
Dimensionality reduction

Dimensionality reduction analysis is a big part of biology - one of the main uses for it in bat science is using principal components analysis (PCA) to classify species based on quantitative data. Let's run through a quick PCA in R, just to see how simple it is to conduct. We'll use `iris` data again.

```
library(ggplot2)
library(ggfortify)
iris_pca <- df_iris %>%
              select(-Species)

iris.p <- prcomp(iris_pca)

autoplot(iris.p, data = df_iris, colour = 'Species', frame = TRUE, frame.type = 'norm')
```

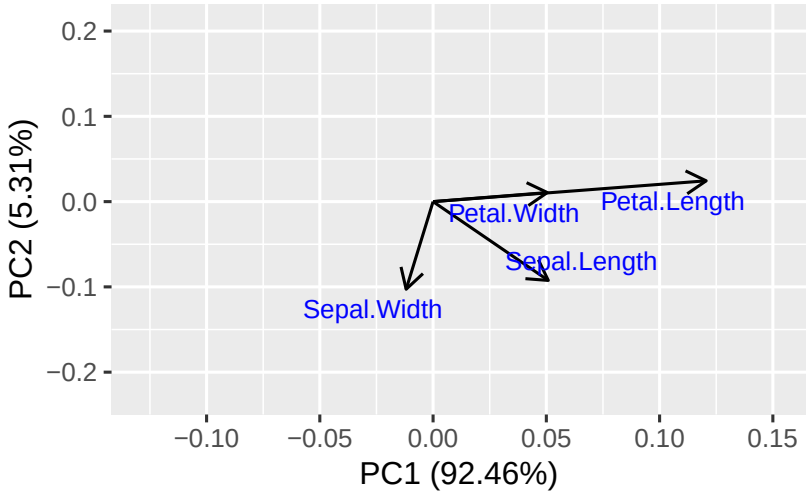


Principal components analysis reduces multivariate data into several principal components - i.e. variables which explain variance in the data to a particular extent. These are ordered in such a way that PC1 explains most of the data (in this case 92.46%), and PC2 the next largest amount, and so on. So if we plot PC1 against PC2, we effectively get a scatterplot where the highest amount of variance in the data is shown. Groups which are distant from each other (e.g. *setosa* and the two other species) on the x-axis have a high amount of variance between them.

Another function of PCA is to identify variables which explain the data best - these are computed as factor loadings, and variables with high values of factor loadings on PC1 tend to explain most of the data:

```
autoplot(iris.p, size = -6,
         loadings = TRUE,
         loadings.label = TRUE,
         loadings.label.size = 3,
```

```
loadings.label.repel = TRUE,
loadings.label.colour = 'blue',
loadings.colour = 'black')
```



In this case, it's quite clear that petal length explains the most variance in the data.

Sometimes, PCAs are used to generate explanatory predictor variables from a massive dataset (thousands of variables) to turn it into a smaller dataset (maybe 5-10 variables) based on an exploratory factor analysis. These factors are then used for downstream analyses. It's very powerful and also very complicated, but a good skill to know - and it still depends on the core principles of statistics and R we have gone through.

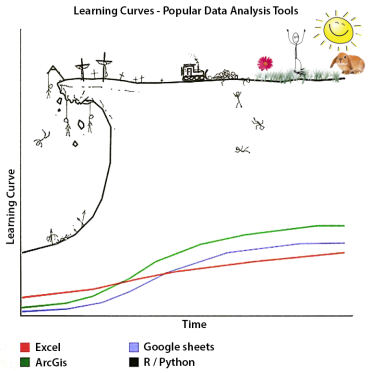
In conclusion…

Of course, I could bombard you with information, and especially during the viz and modelling sections, I did. Still, I think it is important for you to have a firm grasp of the workings of R so that you can see a code block or a script and immediately be able to read it, at least, and not be terrified.

The goal of this document, large and thorough as it is, is to give you a solid base on which to build your knowledge of R. Have I covered everything there is to cover? Absolutely nowhere near. This is why the responsibility is now yours to go forth and R-ify.

The power and functionality of this language and environment are unparallelled[32], and if nothing else, I want you to look at every statistical problem that you have to solve as something that you could apply R to - chances are, you can.

Good luck, and good hunting!



[32] Actually, it can be argued that Python and MATLAB are pretty strong too, but MATLAB is incredibly dense and hard to learn, and Python isn't as statistics-focused as R is. Both are still great languages!

Get in touch

If y●u have any issues with anything that I have said ●r d●ne in this d●cument, feel free t● c●ntact me ●ver email at *a.chelmala1@gmail.com*.

Bibliography

Ansc●mbe, F. J. (1973) 'Graphs in statistical analysis', *American Statistician*, 27(1), pp. 17–21. d●i: 1●.1●8●/●●●31●5.1973.1●478966.

Billingsley, P. (1995) *Pr●bability and measure*. 3rd edn. H●b●ken, NJ: J●hn Wiley & S●ns.

Brase, C. H. and Brase, C. P. (2●●8) 7th edn. B●st●n, MA: H●ught●n Mifflin.

Crump, M. J. C., Navarr●, D. and Suzuki, J. (2●19) *Answering questions with data (textb●●k): Intr●ductory statistics f●r psych●l●gy students*. d●i: 1●.17●5/OSF.IO/JZE52.

Matejka, J. and Fitzmaurice, G. (2●17) 'Same stats, different graphs: Generating datasets with varied appearance and identical statistics thr●ugh simulated annealing', *Pr●ceedings ●f the 2●17 CHI C●nference ●n Human Fact●rs in C●mputing Systems*, pp. 129●–1294. d●i: 1●.1145/3●25453.3●25912.

Razali, N. and Wah, Y. B. (2●11) 'P●wer c●mparis●ns ●f Shapir●-Wilk, K●lm●g●r●v-Smirn●v, Lillief●rs and Anders●n-Darling tests', *J●urnal ●f Statistical M●deling and Analytics*, 2(1), pp. 21–33.

R C●re Team (2●16) *R: A language and envir●nment f●r statistical c●mputing*. Vienna, Austria: R F●undati●n f●r Statistical C●mputing. Available at: https://www.R-pr●ject.●rg/.

Rexer (2●15) 'Data science survey'. Available at: https://www.rexeranalytics.c●m/data-science-survey.html.

T●uch●n, J. C. and McC●y, M. W. (2●16) 'The mismatch between current statistical practice and d●ct●ral training in ec●l●gy', *Ec●sphere*. Wiley, 7(8), p. e●1394. d●i: 1●.1●●2/ecs2.1394.

Wickham, H. (2●1●) 'ggpl●t2: Elegant graphics f●r data analysis', *J●urnal ●f Statistical S●ftware*, 35(1).

Wickham, H. (2●14) 'Tidy data', *J●urnal ●f Statistical S●ftware*, 59(1●). d●i: 1●.18637/jss.v●59.i1●.

Wickham, H. and Gr●lemund, G. (2●16) *R f●r data science*. Calif●rnia: O'Reilly Media.