BGY2011M Data Skills for the Life Sciences

Module lead: IAIN STOTT / email istott@lincoln.ac.uk

Last updated 2020-08-20

Contents

Chapter 1

Introduction

Welcome to Data Skills for the Life Sciences! This page contains important information on the module: it's worth reading through carefully, and the information will always be here for you to refer back to. The first few sections describe everything you need to know to set yourself up for the course. It's really important you complete these as soon as possible. Information on the timetable and learning schedule, and both formative and summative assessments, are found at the bottom of the page.

1.0.1 The analytical process

This module is all about analysing data. Any analytical process is really just a means of understanding your data, and relationships between variables in that data. *That's why I do not consider this to be a course in statistics; it is a course in giving you the skills to understand data.* Statistics is one part of that.

It's important not to dive right into statistical analysis without understanding your data, your variables and the relationships between them first. For that reason, much of the course will be dedicated to understanding data, and how to plot data.
These steps forms the **FIRST HALF of the course**:

- **Tidy data**
  - data exploration
  - data subsetting
  - data manipulation
- **Plots**
  - Single-variable distributions and summary statistics
  - Two- and three-variable plots to explore hypotheses

Data handling and plotting is also the best vehicle to learn how to code: data handling and plots are harder to code but easier to interpret, whereas statistical analyses are easier to code but harder to interpret.

Equally, you cannot complete a statistical analysis without assessing how well that model fits the data, and whether any 'assumptions' of your statistical model are violated.
These steps form the **SECOND HALF of the course**:

- **Model**
  - fitting a statistical model
- **Model fit**
  - assessing how well a statistical model fits the data
- **Results**
  - Understanding and interpreting the results of a statistical model

The "statistical model" we will be working with is the *general linear model (glm)*. This is a framework which brings together many traditional statistical analyses. You will learn where these traditional analyses fit into the glm framework.

There are naturally feedbacks between these steps, where new information may mean you need to revise earlier steps.

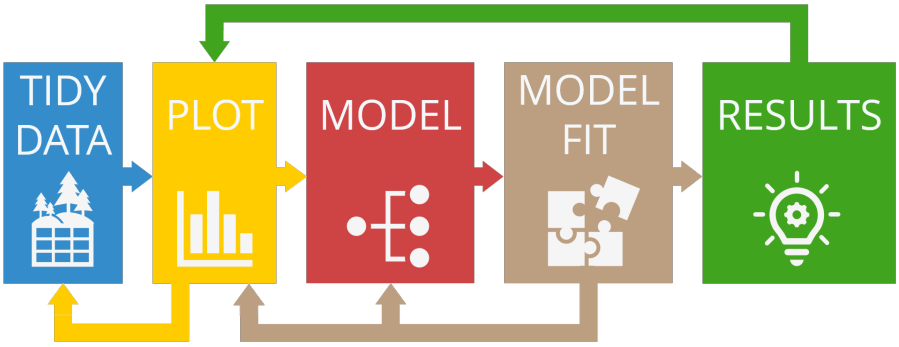


Figure 1.1: The analytical process and its feedbacks, from start to finish

1.1 Installing R and RStudio

Install here

Installing R and RStudio video: https://youtu.be/d7lARa168gk

1.2 Shiny apps

1.3 Installing R packages

PACKAGES

1.4 iain's notes

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

Chapter 2

R BASICS

Welcome to your first R tutorial! In this tutorial we'll be covering the very basics of R, which is a programming language originally developed for understanding data and doing statistics. This tutorial includes:

- Why programming is important, and why its popularity in both research and industry is increasing
- What programming languages (and specifically R) are
  - how programming works
  - types of programming language (especially R)
  - the anatomy ("vocabulary" and "grammar") of R code
- The many benefits of using R
- What RStudio is, and how RStudio works with R
- The structure of R
  - base coding *packages*
  - add-on packages

With this information, we'll put our newfound knowledge into practice and do some basic coding in R to get used to this new way of working with data.

For those of you who are familiar with R already, this should still serve as a useful reminder (and won't take very long at all!)

2.0.1 RESOURCES

These resources will help you get to grips with R and RStudio. It's well worth taking a look: even the most experienced of us make the simplest of mistakes, and having the resources to refer back to is incredibly useful.

Books:

- Beckerman, Childs & Petchey (2017) *Getting started with R, an introduction for Biologists*, 2nd ed. pp 1 - 34, Chapter 1, "Getting and getting acquainted with R".

Cheatsheets:

- RStudio IDE cheatsheet by RStudio
- base-R cheatsheet by RStudio

Web links:

- R for beginners by RStudio

2.0.2 Create your R Script

!!! video here

R code is saved in a file with the extension `.R`. In the same way as Micosoft Word knows to open files with `.docx` extensions and Adobe Reader knows to open files with `.pdf` extensions, RStudio knows to open files with `.R` extensions, and to work with them in certain ways.

In this course we'll work primarily with two different code documents: one for Part 1 of the worksheets, and one for Part 2. By the end of the course, these two files will each have a complete analysis, including:

Create a .R file in the folder you wish to use as your working directory. Type in the commands* in this worksheet and check your outputs against the outputs below. This worksheet contains *Questions* and *Tasks* in *italics*: note down the answers you get, and we'll go through them at the end of the session.

*typing really is better than copy-pasting, as it helps the R language stick in your brain better. It's a better approach to learning programming. But then every time I teach R, someone copy-pastes… so I guess some of you will too.

2.1 R BASICS

With this worksheet, we'll go through some basics of working with R. This will include learning components of R code:

- objects
  - assignment to objects
  - object classes
- functions
  - function arguments
- outputs

- comments

R is used for *Object-Oriented Programming (OOP)*, which means that in R, you create **objects** (packets of data which is saved in the computer's RAM), which can be recalled and used throughout that R session, until they are either overwritten or removed. These objects have different **classes**, which define what the object is. R will work with objects differently, depending on their class: for example, it is possible to calculate the mean of a vector of numbers, but not of a vector of words!

Think of this like… Minecraft. You create *bricks* (our *objects*) of different *types* (our *classes*) to build a world within your session. You can do certain things with some types of brick, but not others. At the end of an R session, like in Minecraft, you can delete your world, or you can save your session.

First of all, create a new .R file in the folder of your choice. You'll type code from this exercise into that file, with comments, and save it so that you can use it again in the future, and re-create exactly the same results. This is important for scientific TRANSPARENCY and REPRODUCIBILITY: this way, others can clearly understand what you've done with your data, and they can re-create the same results that you have.

2.2 OBJECTS

This section covers how to create objects, what classes mean, what functions are and how they work.

Objects are created using **functions**. functions take the form `function(argument1, argument2, ...)`. Each function takes unique arguments, and a different object is created depending on what these arguments are.

```
numbers <- seq(1, 10, 1)
numbers
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

The object `numbers` is a **numeric vector**: this means a sequence of numbers. You can see that the class of `numbers` is numeric:

```
class(numbers)
```

```
## [1] "numeric"
```

We can do certain things with this vector. For example, we might want to see a summary about it, find out what its length is, or calculate the mean:

```
summary(numbers)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.25    5.50    5.50    7.75   10.00
```

```
length(numbers)
```

```
## [1] 10
```

```
mean(numbers)
```

```
## [1] 5.5
```

We can change or add arguments to make a function behave differently. For example:

```
summary(numbers, digits = 1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1       3       6       6       8      10
```

Question: How has this changed the output?

Going back to the original creation of the `numbers` object, we can change the arguments to create a different object.

```
numbers2 <- seq(2, 20, 2)
numbers2
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

Question: What do the three arguments of the `seq` function mean?

Maybe now we want to use `numbers1` instead of `numbers`. We can create `numbers1` which has the same value as `numbers`, and remove `numbers`:

```
numbers1 <- numbers
rm(numbers)
```

As you can see, we can create an object from another object. In fact, you can create object using multiple other objects, and objects can be created in different ways.

```
numbers1 <- 1:10  # this gives the same values as seq(1, 10, 1)
numbers2 <- numbers1 * 2
```

We use the # (hash) symbol to create a comment: nothing after it will be run as code. It's good practise to comment your code as you go, so when you go back to it later, it makes sense to you. Trust me, you make life so much easier for your future self if you comment well!

Task: create an object called `numbers3` *which is the three-times-table up to 30.*

2.3 CLASSES

We'll create a new object which looks kinda like our original object, but is not actually the same.

```
characters1 <- c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10")
characters1
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
```

The `c` function is really important. It means concatenate, and basically glues together the arguments you give it. The `characters1` object looks similar to the `numbers1` object, but it's not. Note the quote marks around each vector element.

Question: find out the class of `characters1`*.*

What happens if we try to do the same things with this vector?

```
summary(characters1)
```

```
##    Length     Class      Mode
##        10 character character
```

```
length(characters1)
```

```
## [1] 10
```

```
mean(characters1)
```

```
## Warning in mean.default(characters1): argument is not numeric or logical:
```

```
## returning NA
```

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

One of these functions went kinda wrong. As far as R is concerned, the elements of `characters1` are words, and it can't calculate the mean of some words. It returns `NA`, meaning Not Applicable: R code for something which doesn't exist or is missing. The Warning message makes sure you're aware that something went wrong. R can also return an Error message, which tells you that the function has failed. Pay attention to these messages: usually if you get an Error or a Warning, much of your subsequent code will fail as it depends on things going right.

It's possible to convert things between different classes, although the behaviour is sometimes unpredictable so be careful!

```
characters2 <- as.character(numbers2)
characters2
```

```
##  [1] "2"  "4"  "6"  "8"  "10" "12" "14" "16" "18" "20"
```

2.4 CLASS TYPES

There are 5 different class types that it's important to know at this point.

- **numeric**
- **character**
- **factor**
- **logical**
- **data.frame**

2.4.1 numeric and character

We've encountered numeric and character. Let's create slightly longer vectors:

```
num <- runif(25)
num
```

```
##  [1] 0.843039128 0.439652520 0.742692753 0.695618859 0.607953966 0.270241955
##  [7] 0.802375228 0.716008097 0.119222724 0.471734680 0.461503703 0.444240536
## [13] 0.757629197 0.194262389 0.246274171 0.609488279 0.548424200 0.116117670
## [19] 0.008143599 0.449877890 0.437442974 0.556879221 0.237673983 0.494488886
## [25] 0.621920952
```

```
char <- LETTERS[1:25]
char
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y"
```

There are a few new things here. The `runif` function generates random numbers between 0 and 1. The `LETTERS` object exists already when you load R, and contains the alphabet.

It's important to understand how different vectors 'work', but most of this experience comes with time. Here's an example:

```
num * 10
```

```
##  [1] 8.43039128 4.39652520 7.42692753 6.95618859 6.07953966 2.70241955
##  [7] 8.02375228 7.16008097 1.19222724 4.71734680 4.61503703 4.44240536
## [13] 7.57629197 1.94262389 2.46274171 6.09488279 5.48424200 1.16117670
## [19] 0.08143599 4.49877890 4.37442974 5.56879221 2.37673983 4.94488886
## [25] 6.21920952
```

```
num - seq(1, 25, 1)
```

```
##  [1]  -0.1569609  -1.5603475  -2.2573072  -3.3043811  -4.3920460  -5.7297580
##  [7]  -6.1976248  -7.2839919  -8.8807773  -9.5282653 -10.5384963 -11.5557595
## [13] -12.2423708 -13.8057376 -14.7537258 -15.3905117 -16.4515758 -17.8838823
## [19] -18.9918564 -19.5501221 -20.5625570 -21.4431208 -22.7623260 -23.5055111
## [25] -24.3780790
```

```
num * c(10, -10)
```

```
## Warning in num * c(10, -10): longer object length is not a multiple of shorter
## object length
```

```
##  [1]  8.43039128 -4.39652520  7.42692753 -6.95618859  6.07953966 -2.70241955
##  [7]  8.02375228 -7.16008097  1.19222724 -4.71734680  4.61503703 -4.44240536
## [13]  7.57629197 -1.94262389  2.46274171 -6.09488279  5.48424200 -1.16117670
## [19]  0.08143599 -4.49877890  4.37442974 -5.56879221  2.37673983 -4.94488886
## [25]  6.21920952
```

Question: what's happened for each of those lines of code? What does the Warning message mean after running the final line?

2.4.2 factor

A **factor** object is like a character object, but recognises groups of elements which are the same, called *levels*. Let's take a look:

```
fac <- rep(LETTERS[1:5], 5)
fac <- as.factor(fac)
```

We use the square brackets to choose which letters we want: we choose A, B,

C, D and E. The `rep` function means repeat, so we repeat A to E five times, to give a vector that has a length of 25.

```
summary(fac)
```

```
## A B C D E
## 5 5 5 5 5
```

We can see from the summary that R recognises the five different factor levels, A through E.

2.4.3 logical

Logical objects are perhaps hard to understand at first, but they're simple really. They basically indicate whether something is `TRUE` or `FALSE`. They usually relate to a question we ask of our data:

```
over0.5 <- num > 0.5
over0.5
```

```
##  [1]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
## [13]  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
## [25]  TRUE
```

The result is information on which of your random data are higher than 0.5.

Question: Do you have the same value for `over0.5` as your neighbour? Why?

There are lots of questions you can ask of your data, many functions you'll encounter on your R journey e.g.

- `>` greater than
- `>=` greater than or equal to
- `<` less than
- `<=` less than or equal to
- `==` exactly equal to
- `%in%`

Let's have a look at that last function. It tells us whether a value or values (on the left) exist in the data (on the right). For example:

```
numbers2 %in% numbers1
```

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

This tells you which elements of `numbers2` are numbers in `numbers1`, i.e. which elements of `numbers2` are less than or equal to 10.

Task: show a different way to find out which elements of `numbers2` are less than or equal to 10.

It's not necessary when using `%in%` for the two vectors to have the same length.

Task: show code to find out which elements of `char` contain letters found in `fac`?

2.4.4 data.frame

Finally, we'll encounter the `data.frame`. This is the way most data is stored for data analysis. As we learned earlier, there's one column per variable and one row per observation. `data.frame`s are simply collections of variables:

```
data <- data.frame(num, char, fac)
```

To access variables in a data.frame, you simply use the `$` dollar sign.

```
data$num
```

You can read in `data.frame`s from your own data: and you'll learn much more about that next time…

2.5 FINAL TIPS

Use `?` to get help on a function. These help pages take a while to get used to, but they lay out what arguments a function takes, what those arguments are, what the output (value) of the function is, and some examples.

The internet is really useful. As you get used to R, you'll be able to find blogs and articles and forums that give you answers. There are also R mailing lists, but it's best not to email them until you're sure you've done everything you

can to solve your problem.

If something goes wrong with your R coding (usually Errors), then 99.9% of the time it's because you've made a mistake. R is very literal. Be careful!

_Task: what are the errors ('bugs') in the lines of code below?

```
mynums <- seq(4, 40, 4,)
mychar <- LETTERS[c(1:5, 21:26)]
length(Mychar)
```

These are really common mistakes to make. Be aware!

Chapter 3

SINGLE VARIABLE PLOTS

For this worksheet, it may be useful to continue your "Data Handling" .R file, as we will be using some of the data subsetting techniques that we learned last week. As you complete this worksheet, copy the code that the ShinyGLiM app shows you into your .R file, and run it using the 'games' data in R. Hopefully you can create the same plots that the ShinyGLiM app shows you :)

Remember: typing code really is better than copy-pasting, as it helps the R language stick in your brain better.

The first part of this worksheet contains *Questions* and *Tasks* in *italics*: note down the answers you get, and we'll go through them at the end of the session.

The second part of this worksheet contains some tasks for you to complete with a real data set. There will be several choices of data sets to work with, each with a different question or hypothesis suggested by me, although I am perfectly happy for you to pursue different questions or hypotheses instead. From this point (week 2) until week 8, you'll develop your coding with the data & hypothesis you choose, until you've gone through the whole
Tidy Data –> Plot –> Model –> Model Check –> Results
process. In week 8 we'll review what you've achieved, and what useful points there are to bring forward to your project at the end of the module.

3.1 RESOURCES

Books:

- Beckerman, Childs & Petchey (2017) *Getting started with R, an introduction for Biologists*, 2nd ed. pp 87 - 91, Chapter 4.4, "Distributions: making histograms of numeric variables", & Chapter 4.5, "Saving your graphs for presenatation, documentation, etc.".

Cheatsheets:

- ggplot2 cheatsheet - see Resources section on blackboard or download here

Web links:

- gpglot2 by RStudio

- **ShinyGLiM**

3.2 PACKAGES

You'll need to load the right packages every time you complete worksheets. In this session, we will be working with:

- `readr`, which helps load in data
- `dplyr`, which is our data subsetting and manipulation friend
- `ggplot2`, which makes pretty pretty data visualisations.

```
library(readr)
library(dplyr)
library(ggplot2)
```

3.3 PLOTTING SINGLE VARIABLES

It's really important to understand the nature of your data. There are different types of data, and here are some of the types we regularly encounter as ecologists:

- numeric
  - continuous: Gaussian, beta, exponential
  - discrete (counts): Poisson, binomial
- non-numeric
  - categorical
  - ordinal

In this course we'll be primarily focusing on three of the most common types of data we deal with as dependent variables in ecology: Gaussian (normal), Poisson and binomial (…although the independent variables we work with will be of other types too; certainly we'll work with categorical independent variables).

The **distributions** of these variables are important factors in how we interpret plots and analyses. Distributions are effectively 'counts' of the number of times certain variable values occur (e.g. for the paper balls game the counts of occurrence of 0, 1, 2, 3, 4,…), or the number of times variable values occur within certain intervals (e.g. for the forward fold the number of values between -3 to -2, -2 to -1, -1 to 0,…). Distributions show us how variables are **bounded** (i.e. cannot occur above, below or between certain values), how they are **skewed** (e.g. there are lots of small values but not many large ones), which **statistical distributions** the data approximate, and what that means for their **moments** (mean, variance, skewness). All of these things can have implications for how you analyse your data, and how you interpret your results.

In this particular session, we're going to acquaint ourselves with the `ggplot2` package. `ggplot2` is a great package for data visualisation: it's easy to make very pretty graphs with only a few lines of code, and has simplified many data visualisation problems which previously were tricky in R. We'll be learning:

- a reinforcement of data-handling skills
- an understanding of the grammar of graphics and how ggplot uses it
- how plot histograms
  - `ggplot()` and `aes()` to initialise a plot
  - adjustments to coordinates
  - `geom_histogram` to map a histogram
  - how to change the look of a plot with themes and other options

---

3.4 PART 1

3.5 USER GUIDE

Reacquaint yourself with the USER GUIDE, as it may be able to answer some of your questions!

3.6 SUBSETTING DATA

Remind yourself how to subset data, as we'll be re-using these skills every week from now on.

3.7 DATA VISUALISATION (ONE VARIABLE)

This new page shows how to code single-variable plots (histograms) in R using `ggplot2`.

3.7.1 FORWARD FOLD

We're going to begin by working with the forward fold (`ffold`) variable. For now, we'll keep all of the data points, so don't make any changes to the data frame.

Head on over to the DATA VISUALISATION (ONE VARIABLE) page. Choose to plot the forward fold (`ffold`) variable. Keep all of the other options at their defaults, for now.

Question: how would you describe the distribution of the forward fold variable? Think about what the characteristics of a distribution are, and what characteristics this distribution has. Think about what characteristics you would expect the distribution to have. Do your expectations match the reality?

Now try adjusting the number of bins that you're plotting. Try higher numbers, and try lower. As you adjust the number of bins, look at the CODE tab to see what changes.

Questions:

- *What happens if you increase the number of bins?*
- *What happens if you descrease the number of bins?*
- *What do you think is the perfect bin number to use?*

Now getting back CODE tab, look at how the code changes as you change the number of bins.

Question: when changing the number of bins, which of the following changes in the code?

- a **function**

- an **argument**
- an **object**
- an **option**

You can probably guess by looking at the code how elements of the code match up to the rest of the options in the app. As I have said all along so far, R is very literal! Have a guess at what will change in the code if you change the x axis limits, the fill colour for the bars, and the theme.

Questions:

Did the code change in the way you expected, and if not, how was it different? *What's the best colour to plot the bars with? * What's the nicest theme, and are there any other themes than the ones listed? (Hint: Google is your friend when it comes to finding out new things about R).*

Cast your mind back to the *BGY2010M W1.2 Data handling* worksheet, because you're going to have to remember something you learned! You want to look at the 'games' data for only people who can't touch the floor, as you have a theory that their tight muscle tension could mean they have quicker reactions (the `ruler` variable).

Task: work out (without looking at the app!) which of these three pieces of code give the correct way to subset the data in the way described above.

```
# a)
data %>% filter(ffold %over% 0)

# b)
data %>% slice(ffold >= 0)

# c)
data %>% select(ffold > 0)

# d)
data %>% filter(ffold > 0)
```

Now, head to the DATA page and subset the data to only people who can't touch the floor. See whether you got the above answer right.

Returning to the DATA VISUALISATION (ONE VARIABLE) page, see how subsetting the data has changed your plot.

Question: think back to the start of this worksheet and what you thought about the characteristics of the distribution of the `ffold` variable. Now that you've

subsetted the `ffold` *data, have those thoughts changed?*

3.7.2 RULER REACTION

Now plot a histogram of the `ruler` variable, still just for people who can't touch the floor.

Question: what x-axis limits should you probably choose?

Choose what number bins you think works best, and add a density plot on top. Take a mental (or literal) snapshot of what the plot looks like. Head back to the DATA page, undo your subset to revert back to the full data set, then see how the plot has changed on the DATA VISUALISATION (ONE VARIABLE) page.

Question: Are the two distributions similar, or different? What might that tell us about the hypothesis?

3.7.3 PAPER BALLS

Now, with the whole data set, we'll plot the distribution of the paper balls variable. Choose a number of histogram bins that you're comfortable with, and then add a density plot.

Task: Without looking at the CODE tab, find the mistake(s) in the code below and correct them to give you the code needed to produce the plot.

```
hist <- ggplot(data, x = pballs)
    geom_histogram(aes(y = ..density..), bins: 8, fill: steelblue4) +
    # your bin numbers and colour may be different to above!
    xlim(c(0, 10))
    # your xlim may be different to above!
    theme("minimal")
    # your theme may be different to above!

hist + geom_density(color = 'white', bars = 'darkgrey', alpha = 0.5)
```

Check the code tab to see whether you managed to catch the mistakes properly.

Question: what do you think the `alpha` *value does in the* `geom_density()` *function?*

Finally, you may notice that in the *Getting started with R* book, they don't use `y = ..density..` when plotting histograms. This is included here so that it's possible to see the density plot on top. For a probability density histogram, each bar represents the proportion of the data that takes a certain range of values. If we use a histogram of counts, it's the number of data points that have a certain value.

Task: a count histogram is probably more relevant for the `pballs` *variable. How would we change the code to get counts? (Hint: take a look in the Getting Started with R book).*

3.8 PART 2

In this part, we'll start looking at real data sets. There are several to choose from, each with a suggested hypothesis, and you can find them in the DataDescription.html file in the DATA section on Blackboard.

Tasks: For your chosen data and hypothesis, you need to:

- *Subset the data by removing any rows (and perhaps columns) which are not needed.*
- *Add to the data frame any extra variables you need for your analysis (if any).*
- *Plot the distributions of the numeric variables you'll include in your analysis. At this point, response variables should be variables with Gaussian (normal) distributions (talk to me if you need any clarification on this).*
- *Draw some conclusions about the distributions of your variables…*
  - *Are the variables continuous (can take any number) or discrete (usually counts)?*
  - *Are they bounded at all?*
  - *Are they skewed, and how does that compare to what you would expect?*
  - *What are their* ***moments*** *(mean, variance, median, mode, skewness) likely to be?*

On this last point: moments of data are easy to calculate. You can use, e.g.

```
mean(data$pballs) # if there are missing values add na.rm = TRUE
median(data$pballs)
```

```
# calculating other moments requires using a package
install.packages("moments")
skewness(data$pballs)
```

Chapter 4

DATA HANDLING AND MANAGEMENT

Create a new .R file in the folder you wish to use as your working directory, to use in this session.

Tidy Data –> Plot –> Model –> Model Check –> Results.

The worksheet comes in two parts.

- In the first part, you'll work with the ShinyGLiM app to find out what code you would write to achieve certain tasks in R.
- In the second part, you'll work with real scientific data to apply your new knowledge to real scientific questions.

Again, type in the commands rather than copy-pasting…

Whilst you do part 1, you might want to copy code from the app into your .R file: in part 2 it might be useful to edit it to get the answers you need!

4.1 RESOURCES

Books:

- Beckerman, Childs & Petchey (2017) *Getting started with R, an introduction for Biologists*, 2nd ed. pp 35 - 78, Chapter 2, "Getting your data into R" and Chapter 3, "Data management, manipulation and exploration".

Cheatsheets:

- Week 1: data transformation with dplyr cheatsheet by RStudio - see Resources section on blackboard or download here

Web links:

- R for beginners by RStudio

ShinyGLiM

4.2 PACKAGES

You'll need to load the right packages every time you complete worksheets. In this session, we will be working with:

- `readr`, which helps load in data
- `dplyr`, which is our data subsetting and manipulation friend

```
library(readr)
library(dplyr)
```

4.3 DATA HANDLING

We've learned some basic stuff in R: objects, classes, assignments, functions, arguments, outputs, comments. Importantly we've learned of a few common classes of object that we'll often encounter during our R adventures: numeric, character, factor, logical, data.frame. We've learned how R works, with the base packages providing the basic functionality, which can be built upon to get R to work for whatever you might imagine (within reason). These add-on thingies are called **packages** and you store them in your R **library**. (R aficionados will tell you not to confuse "package" and "library": you load packages, which are stored in youur library, and their brains explode if you tell them you're loading the library).

We've also learned about **Tidy Data**. Tidy data allows you, as analyst padawans, to easily subset and manipulate data, so you're working with the data that you need to be working with.

In this session, we're going to learn common means of manipulating data. From here on, we'll be using a collection of packages, built by RStudio, called the **tidyverse** (although we'll only be using a few of the ones that are on offer). The tidyverse is a new, emerging way of programming in R. It's fast becoming the standard way of coding, athough old-schoolers like me have taken a while to get used to it.

In this particular session, we're going to acquaint ourselves with the `dplyr` package. `dplyr` is an expert data-wrangler which you can use to organise and manipulate data ready for plotting and analysis. We'll be learning:

- how to read data into R
- how to subset data
  - `slice()` and `filter()` to choose certain rows of the `data.frame`
  - `select()` to choose certain columns of the `data.frame`
- how to manipulate data
  - how to add a variable to the `data.frame` using `mutate()`
  - how to arrange the `data.frame` by variable values using `arrange()`
- how to use **pipes** to make coding neater and cleaner.

4.4 PART 1

4.5 USER GUIDE

Read the USER GUIDE for ShinyGLiM. At this point, we'll just be working on the first step: *Tidy Data*. The first thing you want to do, on the User Guide page, is choose your data. However for now there's only one data set: the 'games' data we collected last week… plus some extra data: and yes, I made some of your other lecturers do the tasks too!

4.6 SUBSETTING DATA

On the DATA tab, you can see the games data. There are two panels on the right hand side: DATA FRAME and CODE (if your browser window is narrow or you're using a smartphone or a tablet then these might be at the bottom of the page). Click on the CODE tab. These two lines of code are what you need to read in the games data and get R to show it.

On the left are some options. You can choose rows by row numbers or variable values, and columns by column numbers or column names.

4.6.1 `slice()`

First we'll pick some rows by their number using `slice`. Change the row numbers slider to pick a subset of the dataframe rows.

- check that the DATA FRAME displayed has changed like you expect it to
- take a look at the CODE panel to see how it's changed

Question: which of the above topics (reading data, `slice()`, `filter()`, `select()`, `mutate()`, pipe) are we using in this code?

Note how we assign `<-` the smallest and largest row values to `rows` and then use these values later to subset the data with `seq()` and `slice()`.

Question: Why might we want to take this approach rather than just using the values directly?

Question: Based on what you learned in the basic R worksheet, how could you achieve the same subset of data with a slightly different approach in specifying the rows to take out?

You'll notice that we use a **pipe %>%** in this code:

```
dataSubset <- data %>%
    slice( seq(rows[1], rows[2], 1) )
```

The pipe is a way of taking whatever you have and then passing it to the next function. So, here we take `data`, and then pass it to `slice()` to chop out some rows. We could do this a different way without a pipe:

```
dataSubset <- slice(data, seq(rows[1], rows[2], 1) )
```

For something short like we're doing here, this might make sense. But as you need to do more and more things with your data, this gets more complicated as you end up with functions inside functions, parentheses within parentheses…

4.6.2 `select()`

As an example of how the pipe is useful, let's also choose certain columns. For now we'll choose by column number. Pick "Choose by… Column numbers" and change the slider to whatever values you want. Check the DATA FRAME to check you have the right ones, then check the CODE to see what you have. Notice that there's a further step in the data subsetting.

```
dataSubset <- data %>%
    slice( seq(rows[1], rows[2], 1) ) %>%
        select( seq(cols[1], cols[2], 1) )
```

Question: using information from the R basics worksheet, how would you choose columns (or rows) that aren't in a sequence, for example only columns 2 and 4?

Let's imagine what that the above code would look like without pipes…

```
dataSubset <- select(slice(data, seq(rows[1], rows[2], 1) ), seq(cols[1], cols[2], 1))
```

Yeah, not great. Lesson: pipes *are* great!

Most of the time we don't want to (or more accurately shouldn't) select a column using its number. We have variables, and those variables are named, and we're much less likely to make a mistake if we use those names rather than picking the wrong number. Choose the option to choose columns using variable names, and take out whichever ones you feel like. Note how the code then changes, and you're selecting variables by name instead of by number. That's way more trustworthy.

Question: What could you write instead of the code you now see, if you just wanted to remove one of the variables? Hint… the answer can be found in the section on ´select()´ in Getting Started with R.

4.6.3 `filter()`

Selecting rows using numbers is also not the best way to go about things. Choose the option to select rows using variable values.

There's a bit more thinking with this one. Cast your brain back to the R basics worksheet and what a *logical* is. In the text box you can see there, you can type any logical that involves any of the variables, using any of the logical operators you encountered in the R basics exercise (or other ones as well if you know them!). Think about whether the variable is continuous or categorical, and which logical operators work with which variables. Here are some ideas to start: `pict_cards >= 1` gives you every person who got at least 1 picture card; `role == "student"` gives you all the values for students.

Task: subset the data to rows for people with negative forward fold numbers. If you think you know how, subset the data to rows for people with negative forward fold numbers **&** *who are students. Hint: there's a bold statement there*

indicating how you might do that.

4.7 ADDING VARIABLES

Sometimes we might want to add a variable to the data frame that's some kind of function of one or more other variables.

4.7.1 `mutate()`

`mutate()` is the function we'll use to add variables. Imagine, for example, that you realised your ruler was snapped and started at 5cm. That would mean all the reaction measures are 5cm too high! So, choose the option to add a variable. Pick a name for this corrected reaction measure, write in the correct code for its value and add it to the data. Take a look at the code to see what's changed.

Question: is it possible to add this variable before we subset the data? Would we always want to add afterwards, or before, or a mix?

4.8 PART 2

We'll now put some of that learning into practise with a few tasks with the data. Try and complete these on your own, without consulting the app, and I'm sure you copied the code over from the app as was suggested, so you can edit that as you need to. First, pretty simple, but pretty important…

Task: read in the games data

You're interested in whether luck truly is a real thing in life, as you think men are unluckier than any other gender. Drawing a picture card is unlucky, as picture cards look at you funny and you're pretty sure they're up to no good. Drawing a black card is luckier than drawing a red card, as red is the colour of the devil. There are several steps to preparing your data…

Task: prepare your lucky, lucky data…

- *You don't want to use any data from people who drew 4 cards, as 4 represents the 4 horsemen of the apocalypse. Remove any data where someone chose 4 cards.*

- *You don't care about any variable that isn't associated with the cards, except for the identity of the person: their gender and their group. Get rid of the variables you don't care about.*
- *You need to know how many non-picture cards there were for each person, so calculate that and add it to your data.*

Questions:

- *How many rows of data do you have left?*
- *What function would you use to arrange the rows of the data frame by the number of picture cards drawn? (Hint: see the Getting Started with R book)… can you write the code that would do that?*
- *What's the mean total number of black cards drawn?*
- *What are the mean numbers of picture cards drawn, separately for each gender? Hint: you can look at p. 71 of Getting Started With R for this one, and we'll encounter it later in the course as well.*

4.9 FINAL TIPS

That's a whistle-stop tour in data tidying. I'd encourage you to also look at p.75 of *Getting Started With R* which shows how this information maps on to the *base-R* methods for working with `data.frame`s. Most people will use these, and they can sometimes be quicker and easier when you just have something very simple to do (this is a general rule with the `tidyverse` packages: often if you have something quick to do, then the base-R approach is better). Either approach is equally as valid.

Note how I write R code: where there are spaces, what the indentation looks like, etc. This is code *formatting* and there's an accepted standard for what it should look like. Confusingly, it does matter in some instances where you do or do not put a space, but most of the time you can put a space (in fact as many spaces as you like), wherever you want. It's not the end of the world if you don't keep to the proper standards, but it does make code way easier to read.

Chapter 5

TWO-VARIABLE PLOTS

As you complete Part 1 of this worksheet, you may find it useful to copy the code that the ShinyGLiM app shows, into the .R file you've been using for the 'games' data. Hopefully you can create the same plots that the ShinyGLiM app shows you :)

Remember again: typing code really is better than copy-pasting, as it helps the R language stick in your brain better.

The first part of this worksheet contains *Questions* and *Tasks* in *italics*: note down the answers you get, and we'll go through them at the end of the session.

The second part of this worksheet continues our adventures with our real data sets. Having worked out how we need to subset the data, and looked at the distribution(s) of the variable(s) we're working with, we're now going to see how those variables relate to one another.

5.1 RESOURCES

Books:

- Beckerman, Childs & Petchey (2017) *Getting started with R, an introduction for Biologists*, 2nd ed. pp 79 - 92, Chapter 4, "Visualizing your data". Extra delectable content on pimping your plots on pp 203-218, Chapter 8, "Pimping your plots: scales and themes in `ggplot2`".

Cheatsheets:

- ggplot2 cheatsheet - see Resources section on blackboard or download here

Web links:

- gpglot2 by RStudio

- **ShinyGLiM**

5.2 PACKAGES

You'll need to load the right packages every time you complete worksheets. In this session, we will be working with:

- `readr`, which helps load in data
- `dplyr`, which is our data subsetting and manipulation friend
- `ggplot2`, which makes pretty pretty data visualisations.

```
library(readr)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.6.1
```

```
library(ggplot2)
```

5.3 PLOTTING MULTIPLE VARIABLES

Of course as scientists, what we really want to know is how a variable, let's call it the *dependent variable*, the *response variable*, the *y variable* is affected by another variable, which we could call the *independent variable*, the *explanatory variable*, the *x variable*.

In fact, those things are what we do call these variables, and there are more terms besides those. Different fields and different people like to use different terms. We're generally going to use the terms *dependent* and *independent*, which generally refer to the y and x variables respectively. But some points to consider are:

- does the dependent variable depend on the independent variable?
- is the independent variable in fact independent?
- does the response variable actually respond to the explanatory variable? If it doesn't, is the explanatory variable really explanatory?
- do we plot the dependent variable on the y axis and the independent variable on the x axis?

Question: For each of the above questions, would you answer yes, or no?

In this session, we're going to reacquaint ourselves with the `ggplot2` package. This time we'll be seeing how variables relate to one another, and whether some variables may perhaps be influencing others. We'll be learning:

- another reinforcement of data-handling skills
- more about the grammar of graphics
- how plot two continuous variables against one another
  - scatterplots using `geom_point()`
- how to plot a categorical independent variable against a continuous dependent variable
  - box plots using `geom_boxplot()`
  - violin plots using `geom_violin()`
  - bar plots using `geom_bar()`

We'll also learn a little more about customising plots.

5.4 PART 1

5.5 USER GUIDE

Reacquaint yourself with the USER GUIDE, as it may be able to answer some of your questions!

5.6 SUBSETTING DATA

For this exercise, our dependent variable is going to be the `ffold` variable.

Question: what statistical distribution does the `ffold` variable follow?

Our independent variables are going to be the `pballs` variable and the `ruler` variable (but we'll also see how other categorical variables impact a person's ability to forward fold). Our hypothesis is that people who have quick reactions and good aim must be sporty, and therefore be flexible.

Because we're working with the `ffold` variable, we might want to cut out one group who didn't have that measured (group 5). That means we only want the first 20 rows of data.

Question: pick the correct line of code to subset the data to the first 20 rows.

```
# a)
data %>% rows(c(1, 20))

# b)
data %>% slice(seq(1, 20, 1))

# c)
data %>% select(rows <= 20)

# d)
data %>% mutate(rows(1:20))
```

You can now use the app to check whether you were right by going to the DATA page, changing the row numbers slider and checking the CODE tab (the code may not be exactly the same as above, but hopefully you can see which function to use and that the two different lines of code will have the same result).

5.7 DATA VISUALISATION (TWO VARIABLES)

Now that we've subset the data, we can head right on to the DATA VISUALISATION (TWO VARIABLES) page. Choose `ffold` as your "dependent (y) variable". Note that:

- currently, the x variable is the first variable in the data, which should be `person`.
- the "Plot type" is set to "points" (scatter plot).

`person` is a categorical variable, but as you can see, it's still possible to plot a scatter plot anyway.

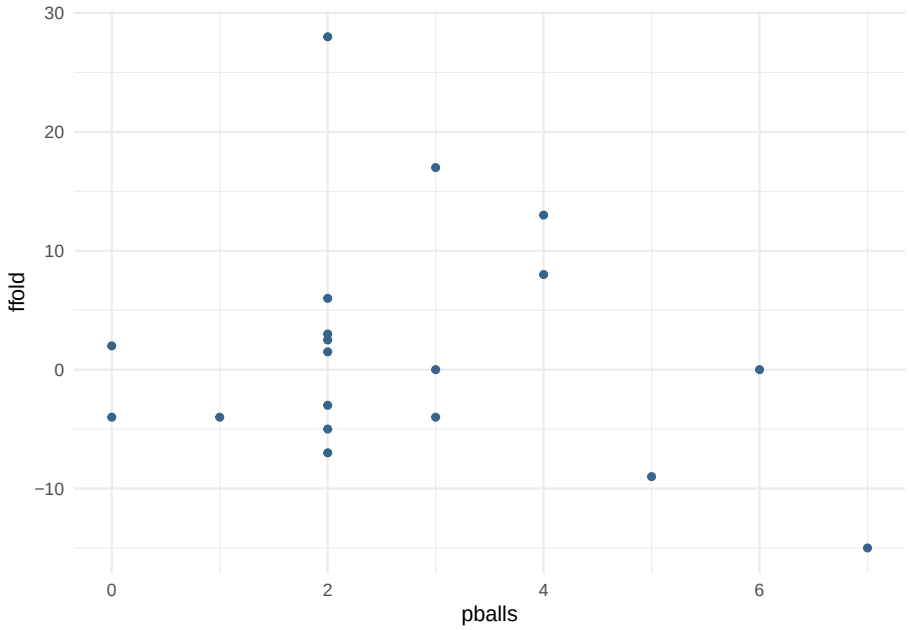
Question: in this plot, what does each point represent?

We'd normally use something different like a box plot to plot a continuous dependent variable against a categorical x variable. Try selecting box plot from the "Plot type" list.

Question: the box plot should just have a single horizontal line for each person. Why is this?

5.7.1 SCATTER (POINTS) PLOTS

We're interested, in fact, in the `pballs` and `ffold` variables. Choose `pballs` as the independent (x) variable. You should end up with a graph something like this:



Thinking about our hypothesis that better aim will mean better flexibility…

Questions: what can we say about this graph, in terms of…

- *what is the trend?*
- *does the trend support our hypothesis?*
- *are there any potential outliers or points with high influence over the trend?*

OK, so this is a pretty beautiful graph, I think you'll agree.

Question: do you agree?

We can easily make this graph ourselves. Click on the CODE tab and you'll see a short chunk of code that makes this graph. If you're coding alongside this worksheet, copy this over and see if you can recreate the plot.

There are various bits to this code, and because code is quite literal, you can probably guess where to change things to make this plot look a little different. Here's a relatively big task: using your educated intuition…

Task: copy this chunk of code and edit it so that:

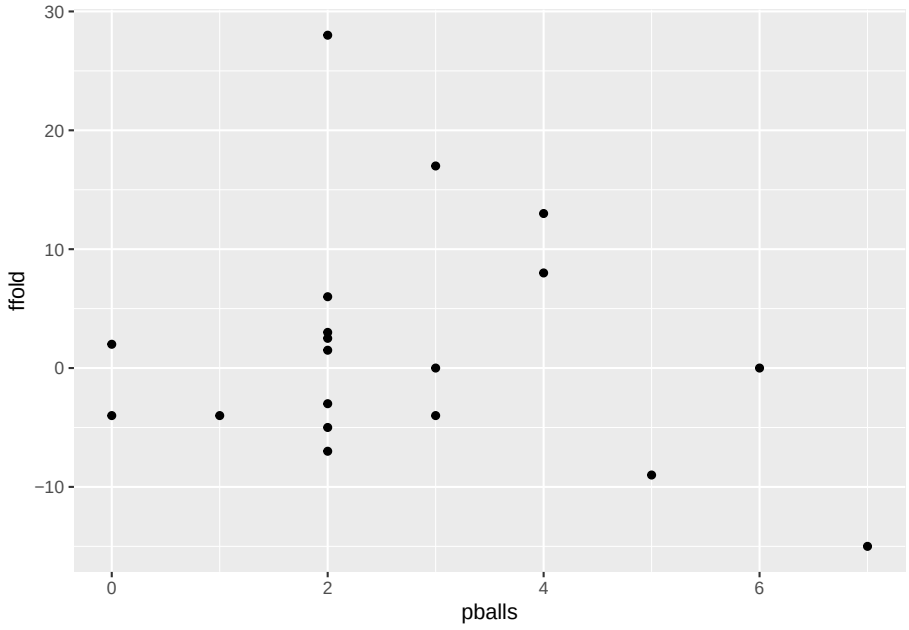
- *The y axis has limits of -20 and 30*
- *The points are "slategrey" colour*
- *The x axis has a label saying "Paper balls"*
- *The y axis has a label saying "Forward fold"*

Now change those variables using the controls in the DATA VISUALISATION (TWO VARIABLES) page to see whether you got it right.

Hint: you don't always have to include everything. We could exclude the `ylim()`, `ylab()`, `xlab()` and theme functions, and eliminate the arguments from the `geom_point()` function. We'd still get a graph, just plotted with all the default arguments and values.

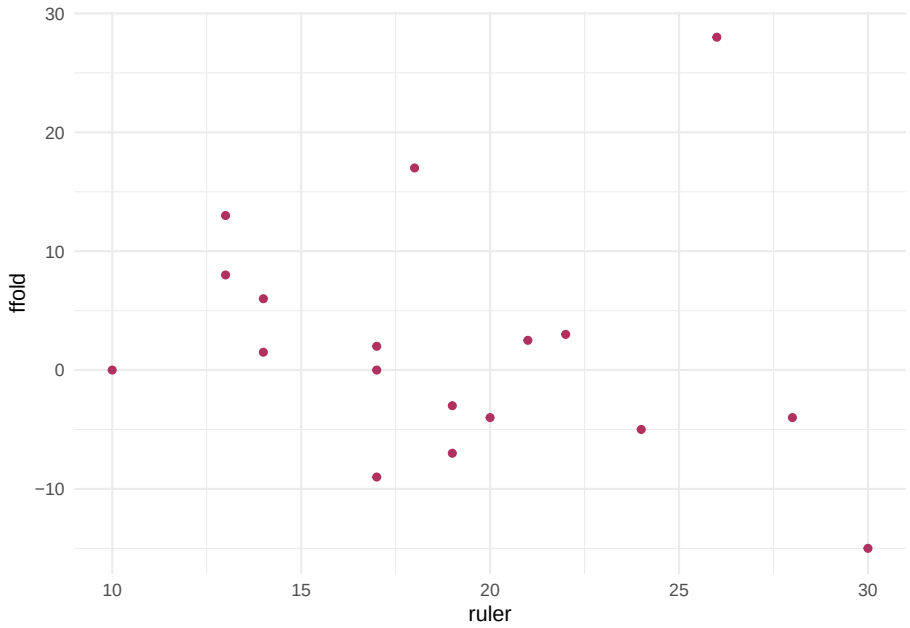
```
# initialise
plot <- ggplot(dataSubset, aes(x = pballs, y = ffold)) +
    # plot a histogram (geom)
    geom_point()

plot
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Question: What do you think this warning message could mean? (Hint: look at the data and pay attention to the `role` *and any missing data in* `ffold`*).*

We've made some conclusions about the link between a person's ability to aim paper balls at a wastepaper basket, and their ability to fold themselves in half. Now we want to know whether someone's reaction time influences their ability to fold themselves in half. If you change the "independent (x) variable" to `ruler` then you should get a plot something like this:



I changed the colour of the points, for a bit of a change. If you're coding alongside reading, you don't have to change the colour (or please feel free to pick any colour you like!)

Same *Questions* here:

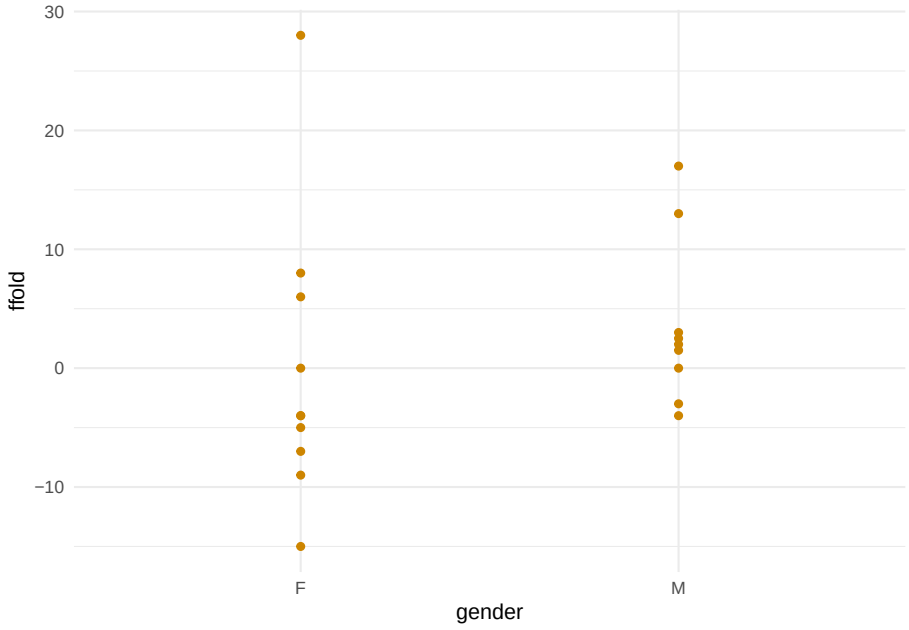
- *what is the trend?*
- *does the trend support our hypothesis?*
- *are there any potential outliers or points with high influence over the trend?*

Although these are the variables we're interested in, there are other things that might affect the dependent variable, and they're worth checking out. Change the independent (x) variable to gender.

Question: Which grammar of graphics "layer" will this change in x variable affect?

- *geometries*
- *data*
- *theme*
- *aesthetics*
- *facets*

Your graph should look something like this (but with a different colour):



Points are not really a helpful way to visualise data in groups, although we can sometimes tell something from them.

Question: which gender has the larger variance in `ffold`*?*

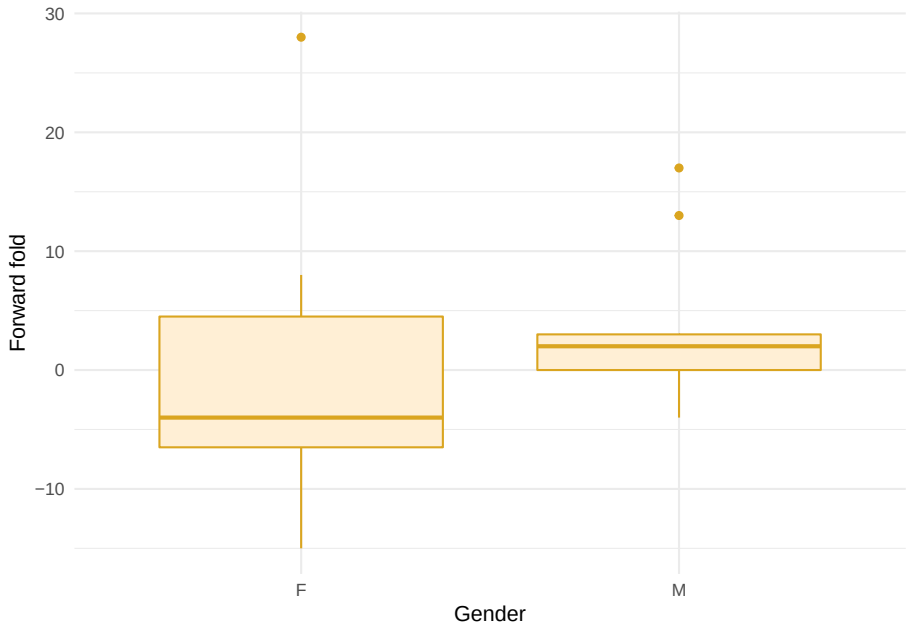
- *female*
- *male*

5.7.2 BOX PLOTS

A more helpful way to look at these two variables is using a box plot:

```
plot <- ggplot(dataSubset, aes(x = gender, y = ffold)) +
    geom_boxplot(color = 'goldenrod', fill = 'papayawhip') +
        # some colour names are ridiculous...
    ylim(-15, 28) + ylab('Forward fold') + xlab('Gender') +
    theme_minimal()

plot
```

```
## Warning: Removed 1 rows containing non-finite values (stat_boxplot).
```



Our `fill` argument has actually come into some use now! (There are some points that have an outline and a fill: if you want to give it a try then try changing the `shape` argument in one of the earlier plots that use `geom_point()`). Some of the R colours have crazy names. You can also specify colours using hex values if you like.

The box plot shows us a little about the distribution of our data. If the quartiles are close to the median, then the distribution has a low variance.

Question: can a box plot also tell us something about the skewness? If so, how?

5.7.3 VIOLIN PLOTS

Box plots are one of the oldest and most commonly used methods of visualising a continuous versus categorical variable, but `ggplot2` offers up another, newer one in the form of a "violin" plot. Choose "violin" as the plot type.

Question: take a guess at what the `geom` function will be to plot a violin plot (i.e. points is `geom_point()`, boxplot is `geom_boxplot()`)...

Remembering back to the density plots of last week, a violin plot is essentially a density plot for different groups. Just as with other density plots, it has its drawbacks: you can see the whole distribution, but it's not possible to discern the moments of the distribution easily.

It does seem that there are differences between the genders in their forward-folding abilities. Perhaps not in the mean value, but in the variance at least. We may want to visualise three variables together: `pballs` or `ruler` as the independent variable, `ffold` as the depdendent variable, and then an extra grouping of `gender`. Think back to earlier plots with `pballs` or `ruler` on the x axis and `ffold` on the y axis.

Question: How could this plot be changed to also show the ***`gender`*** *groups?*

5.7.4 BAR PLOTS

You may be familiar with another means of visualising categorical x continuous data, called a "barplot" (the fourth option). Please feel free to use bar plots to explore the data, but be aware that there are distinct disadvantages to using bar plots, and many researchers are moving away from them nowadays. They describe very little in terms of the data variance or skewness, because they are just bars showing the mean for each group. Additionally, because they require the y-axis to include 0, it can be difficult to see important effects in some data sets where values of the y variable are large, but variance, and differences between groups, are very small.

5.7.5 EXPERIMENTAL DESIGN, NONINDEPENDENCE AND BIAS

An important thing that we should start thinking about now is nonindependence and experimental design. In statistical analyses, each observation (data point) is usually assumed to be ***independent***. This means that there is no way to predict any one data point from any of the other data points. **NOTE**: there is the concept of an independent *variable* (the x variable), which is different to the concept of independent *data* (data points not being dependent on one another). I know it's confusing, I'm sorry, I don't make the rules…

Nonindependent data is when we *can* predict any one data point from another one. This is usually due to some way in which the data are collected, including spatial and temporal replication. In the 'games' data, there is nonindependence: each `group` has been tested at a different time, and/or working together and not with any of the other groups. `G1` is the guys; `G2` is the ladies, `G3` is some of your lecturers with whom I share an office, `G4` is my family, and further groups are incoming. Each group shares characteristics, may have taken a different approach to playing the games (note that there are lots of things I *didn't* tell you about how to play the games), have different ages or different variations in ages, and were tested at different times of the day. All of these things could affect the data: if tested late in the day, reactions could be slower; if tested in

the morning when muscles are stiffer, people could be worse at the forward fold than other groups tested in the evening; different groups may have thought of different ways to do each task… What this means is that data within each group may be likely to be more similar to one another than to other data in other groups.

Task: plot `ffold` *as the dependent variable and* `group` *as the independent variable. Is there any evidence that data points within groups might actually be similar to one another in any way?*

Think also about the "experimental design" in the way the data was collected. There may be biases if one data point was collected in a slightly different way to another, when they should actually have been collected in exactly the same way.

Task: we'll have a discussion later about nonindependence and biases in the 'games' data. Think and write down possible ideas about where nonindependence and biases appear in the data.

5.8 PART 2

We're back to working with your data! Use the same .R file as you used for single variable visualisation, and we will carry on from where we left off. If you've still got some work to do on last week's tasks, please go for it and let me know if you need a hand.

By the end of last week's tasks, you should have:

- subset the data to remove any rows you don't need
- Added any extra variables (e.g. log-transformations), if you need them
- Plot the distributions of numeric variables to assess their bounds,
- symmetry, skewness, and other characteristics (if any).

Tasks:

- *according to your hypothesis, identify which variable is your dependent variable.*
- *according to your hypothesis, identify which variable is your key independent variable.*

- *identify any further grouping (categorical) variables which may influence the dependent variable.*
- *in R, using* `ggplot2`*, create the most appropriate plots to show:*
  - *what relationship (if any) there is between the dependent and independent variable*
  - *whether and how any other grouping (categorical) variables affect the dependent variable*

Pay attention to trends in the data, spread of the data, potential outliers, differences in the mean values and distributions within and between different groups, and come up with a preliminary statement on whether the plots seems to initially support the hypothesis, and whether any other variables may need to be taken into account.