Ewen Harrison and Riinu Ots

HealthyR: R for healthcare data analysis

Never trust a data scientist - they are always plotting.

Contents

List of Tables

List of Figures

Preface

Version 0.3.1

Contributors: Riinu Ots, Ewen Harrison, Tom Drake, Peter Hall, Kenneth McLean.

Why read this book

> We are drowning in information but starved for knowledge. John Naisbitt

In this age of information, the manipulation, analysis and interpretation of data has become paramount. Nowhere more so than in the delivery of healthcare. From the understanding of disease and the development of new treatments, to the diagnosis and management of individual patients, the use of data and technology is now an integral part of the business of healthcare.

Those working in healthcare interact daily with data, often without realising it. The conversion of this avalanche of information to

useful knowledge is essential for high quality patient care. An important part of this information revolution is the opportunity for everybody to become involved in data analysis. This democratisation of data analysis is driven in part by the open source software movement – no longer do we require expensive specialised software to do this.

The statistical programming language, R, is firmly at the heart of this!

This book will take an individual with little or no experience in data analysis all the way through to performing sophisticated analyses. We emphasise the importance of understanding the underlying data with liberal use of plotting, rather than relying on opaque and possibly poorly understand statistical tests. There are numerous examples included that can be adapted for your own data, together with our own R packages with easy-to-use functions.

We have a lot of fun teaching this course and focus on making the material as accessible as possible. We banish equations in favour of code and use examples rather than lengthy explanations. We are grateful to the many individuals and students who have helped refine these and welcome suggestions and bug reports via https://github.com/SurgicalInformatics.

Ewen Harrison and Riinu Ots

August 2019

Structure of the book

Chapters **??** introduces a new topic, and …

Software information and conventions

I used the **knitr** package (Xie, 2015) and the **bookdown** package (Xie, 2018) to compile my book. My R session information is shown below:

```
xfun::session_info()
```

```
## R version 3.4.4 (2018-03-15)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.5 LTS
##
## Locale:
##   LC_CTYPE=en_GB.UTF-8
##   LC_NUMERIC=C
##   LC_TIME=en_GB.UTF-8
##   LC_COLLATE=en_GB.UTF-8
##   LC_MONETARY=en_GB.UTF-8
##   LC_MESSAGES=en_GB.UTF-8
##   LC_PAPER=en_GB.UTF-8
##   LC_NAME=C
##   LC_ADDRESS=C
##   LC_TELEPHONE=C
##   LC_MEASUREMENT=en_GB.UTF-8
##   LC_IDENTIFICATION=C
##
## Package version:
##   base64enc_0.1.3  bookdown_0.7     compiler_3.4.4
##   digest_0.6.20    evaluate_0.13    glue_1.3.1
##   graphics_3.4.4   grDevices_3.4.4  highr_0.8
##   htmltools_0.3.6  jsonlite_1.6     knitr_1.22
##   magrittr_1.5     markdown_0.9     methods_3.4.4
##   mime_0.6         Rcpp_1.0.1       rmarkdown_1.12.4
##   stats_3.4.4      stringi_1.4.3    stringr_1.4.0
##   tinytex_0.11     tools_3.4.4      utils_3.4.4
```

```
##   xfun_0.6          yaml_2.2.0
```

Package names are in bold text (e.g., **rmarkdown**), and inline code and filenames are formatted in a typewriter font (e.g., `knitr::knit('foo.Rmd')`). Function names are followed by parentheses (e.g., `bookdown::render_book()`).

Acknowledgments

A lot of people helped me when I was writing the book.

Frida Gomam
on the Mars

Installation

- Download R

https://www.r-project.org/

- Install RStudio

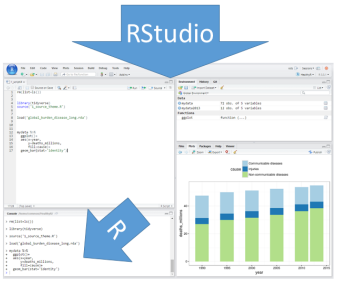
https://www.rstudio.com/products/rstudio/

- Install packages (copy these lines into the Console in RStudio):

```
install.packages("tidyverse")

install.packages("gapminder")

install.packages("gmodels")

install.packages("Hmisc")
```

```
install.packages("devtools")

devtools::install_github("ewenharrison/finalfit")

install.packages("pROC")

install.packages("survminer")
```

When working with data, don't copy or type code directly into the Console. We will only be using the Console for viewing output, warnings, and errors (and installing packages as in the previous section). All code should be in a script and executed (=Run) using Control+Enter (line or section) or Control+Shift+Enter (whole script). Make sure you are always working in a project (the right-top corner of your RStudio interface should say "HealthyR").



Part I

Data wrangling and visualisation

1

Your first R plots

In this session, we will create five beautiful and colourful barplots in less than an hour. Do not worry about understanding every single word or symbol (e.g. the pipe - `%>%`) in the R code you are about to see. The purpose of this session is merely to

- gain familiarity with the RStudio interface:
  - to know what a script looks like,
  - what is the Environment tab,
  - where do your plots appear.

1.1 Data

Load the example dataset which is already saved as an R-Data file (recognisable by the file extension .rda or .RData):

```
library(ggplot2)

source("1_source_theme.R")

load("global_burden_disease_long.rda")
```

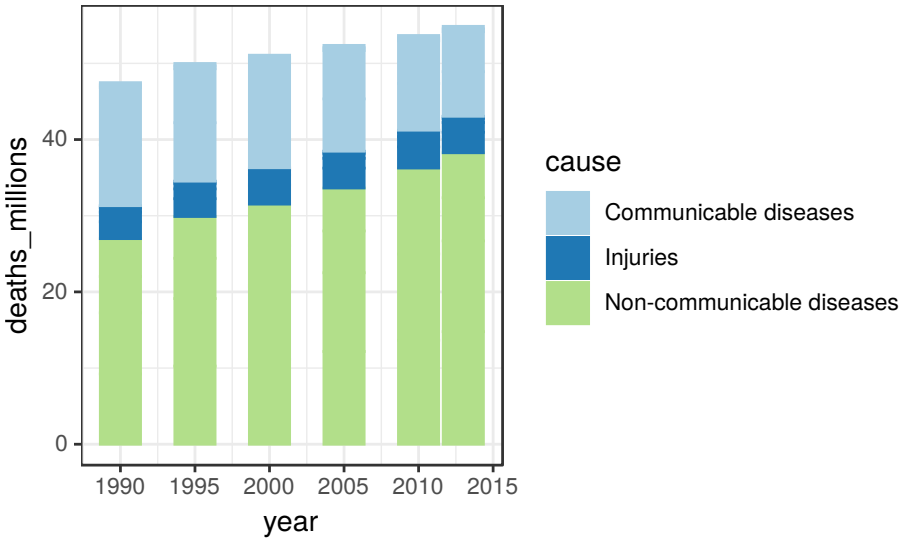
After loading the datasets, investigate your Environment tab (top-right). You will see two things listed: `mydata` and `mydata2013`, which is a subset of mydata.

Click on the name `mydata` and it will pop up next to where your

script is. Clicking on the blue button is not as useful (in this session), but it doesn't do any harm either. Try it.

1.2 First plot

```
mydata %>% #press Control-Shift-M to insert this symbol (pipe)
  ggplot(aes(x      = year,
             y      = deaths_millions,
             fill   = cause,
             colour = cause)) +
  geom_col()
```



`ggplot()` stands for **grammar of graphics plot** - a user friendly yet flexible alternative to `plot()`.

`aes()` stands for **aesthetics** - things we can see.

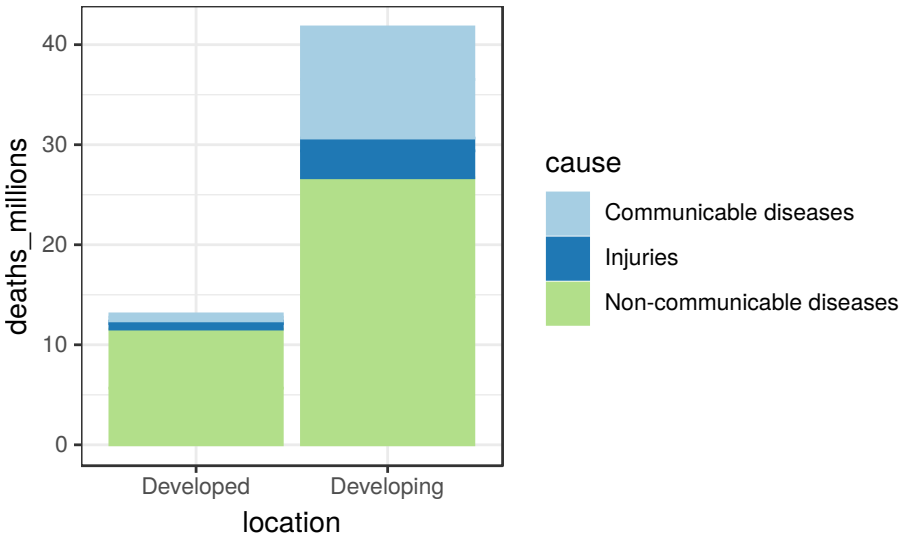
`geom_()` stands for **geometric**.

1.2.1 Question

Why are there two closing brackets - `))` - after the last aesthetic (colour)?

1.2.2 Exercise

Plot the number of deaths in Developed and Developing countries for the year 2013:

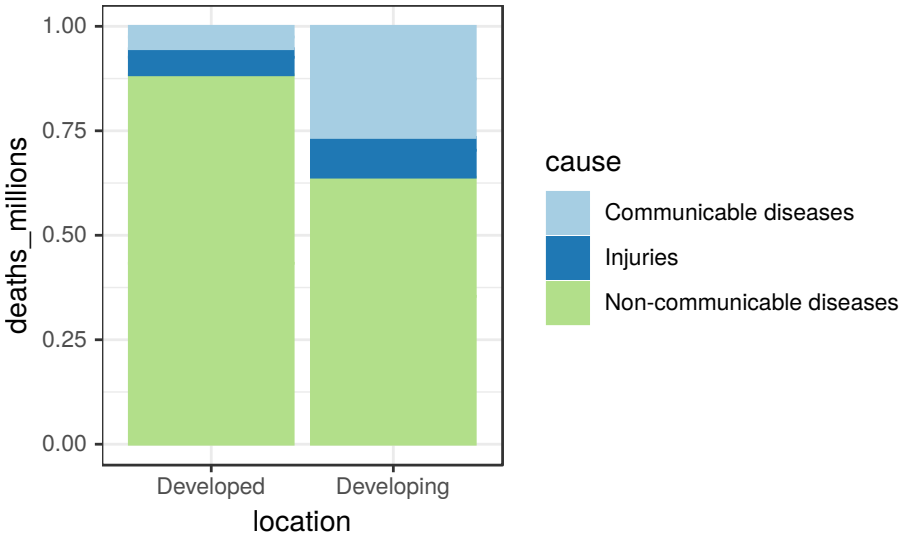


1.3 Comparing bars of different height

1.3.1 Stretch each bar to 100%

`position="fill"` stretches the bars to show relative contributions:

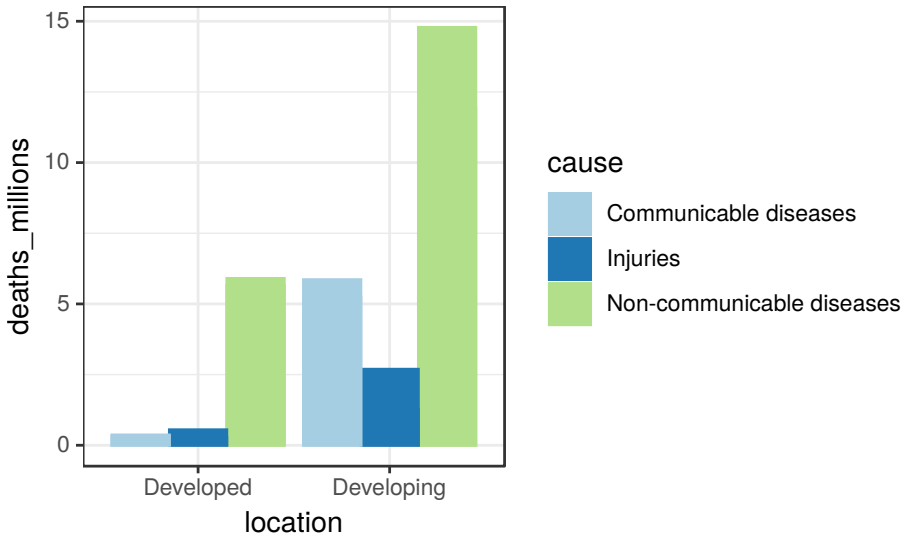
```
mydata2013 %>%
  ggplot(aes(x      = location,
             y      = deaths_millions,
             fill   = cause,
             colour = cause)) +
  geom_col(position = "fill")
```



1.3.2 Plot each bar next to each other

`position="dodge"` puts the different causes next to each rather (the default is `position="stack"`):

```
mydata2013 %>%
  ggplot(aes(x      = location,
             y      = deaths_millions,
             fill   = cause,
             colour = cause)) +
  geom_col(position = "dodge")
```

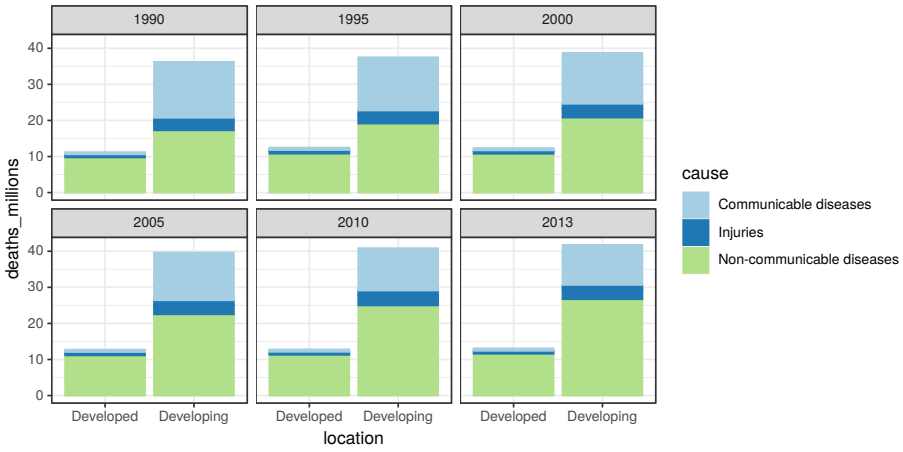


1.4 Facets (panels)

Going back to the dataframe with all years (1990 – 2015), add `facet_wrap(~year)` to plot all years at once:

```
mydata %>%
  ggplot(aes(x      = location,
             y      = deaths_millions,
             fill   = cause,
             colour = cause)) +
```

```
  geom_col() +
  facet_wrap(~year)
```

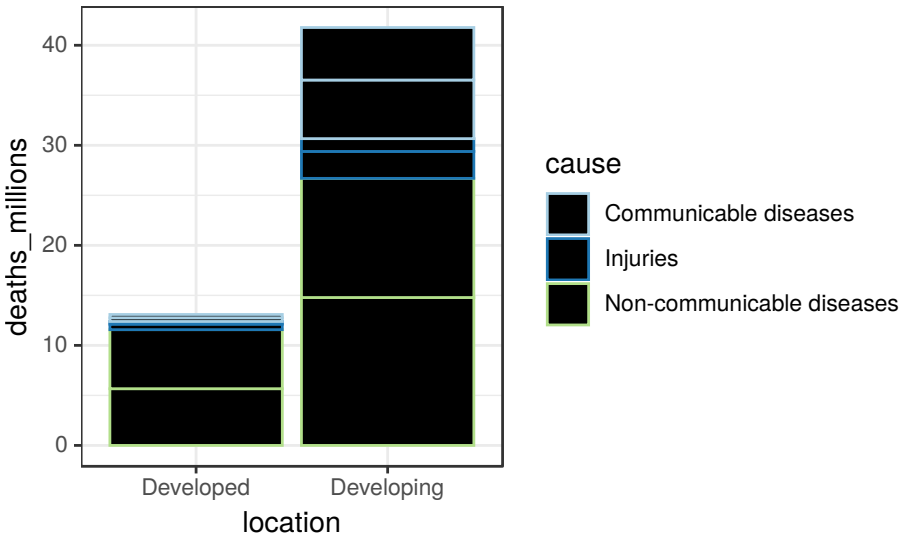


1.5 Extra: using aethetics outside of the aes()

1.5.1 Setting a constant fill

Using the `mydata2013` example again, what does the addition of `fill = "black"` in this code do? Note that putting the `ggplot(aes())` code all on one line does not affect the result.

```
mydata2013 %>%
  ggplot(aes(x = location, y = deaths_millions, fill = cause, colour = cause
  geom_col(fill = "black")
```



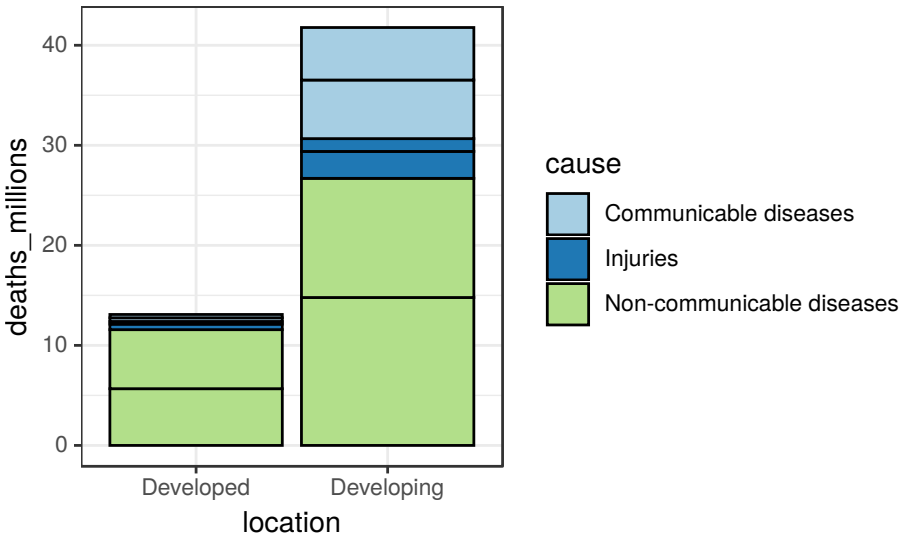
Setting aesthetics (x, y, fill, colour, etc.) outside of `aes()` sets them to a constant value. R can recognise of a lot of colour names, e.g., try "cornflowerblue", "firebrick", or just "red", "green", "blue", etc. For a full list, search Google for "Colours in R". R also knows HEX codes, e.g. `fill = "#fec3fc"` is pink.

1.5.2 Exercise

What is the difference between colour and fill in the context of a barplot?

Hint: Use `colour = "black"` instead of `fill = "black"` to investigate what `ggplot()` thinks a colour is.

```
mydata2013 %>%
  ggplot(aes(x = location, y = deaths_millions, fill = cause, colour = cause
  geom_col(colour = "black")
```



1.5.3 Exercise

Why are some of the words in our code quoted (e.g. `fill = "black"`) whereas others are not (e.g. `x = location`)?

1.6 Two geoms for barplots: `geom_bar()` or `geom_col()`

Both `geom_bar()` and `geom_col()` create barplots. If you:

- Want to visualise the count of different lines in a dataset - use geom_bar()
  - For example, if you are using a patient-level dataset (each line is a patient record): `mydata %>% ggplot(aes(x = sex)) + geom_bar()`
- Your dataset is already summarised - use geom_col()
  - For example, in the GBD dataset we use here, each line already includes a summarised value (`deaths_millions`)

If you have used R before you might have come across `geom_bar(stat = "identity")` which is the same as `geom_col()`.

1.7 Solutions

1.2.1: There is a double closing bracket because `aes()` is wrapped inside `ggplot()` - `ggplot(aes())`.

1.2.2:

```
mydata2013 %>%
  ggplot(aes(x      = location,
             y      = deaths_millions,
             fill   = cause,
             colour = cause)) +
  geom_col()
```

1.5.2:

On a barplot, the colour aesthetic outlines the fill. In a later session we will see, however, that for points and lines, colour is the main aesthetic to define.

1.5.3:

Words in quotes are generally something set to a constant value (e.g. make all outlines black, rather than colour them based on the cause they are representing). Unquoted words are generally variables (or functions). If the word "function" just threw you, Google "Jesse Maegan: What the h*ck is a function"

2

R Basics

The aim of this module is to familiarise you with how R works. We will read in data and start basic manipulations. We will be working with a shorter version of the Global Burden of Disease dataset that we met earlier.

Throughout this course, don't copy or type code directly into the Console. We will only be using the Console for viewing output, warnings, and errors. All code should be in a script and executed (=run) using Ctrl+Enter (line or section) or Ctrl+Shift+Enter (whole script). Make sure you are always working in a project (the right-top corner of your RStudio interface should say "HealthyR").

2.1 Getting help

RStudio has a built in Help tab. To use the Help tab, click your cursor on something in your code (e.g. `read_csv()`) and press F1. This will show you the definition and some examples. However, the Help tab is only useful if you already know what you are looking for but can't remember exactly how it works. For finding help on things you have not used before, it is best to Google it. R has about 2 million users so someone somewhere has had the same question or problem.

TABLE 2.1: Example of a table, including missing values - denoted NA (Not applicable/Not available).

id	sex	var1	var2	var3
1	Male	4	NA	2
2	Female	1	4	1
3	Female	2	5	NA
4	Male	3	NA	NA

2.2 Objects and functions

The two fundamental concepts to understand about statistical programming are objects and functions. As usual, in this book, we prefer introducing new concepts using specific examples first. And then define things in general terms after examples.

The most common data object you will be working with is a table - so something with rows and columns. It should be regular, e.g., the made-up example in Table 2.1.

Now, regular does not mean it can't have missing values. Missing values are denoted `NA` that stands for either `Not available` or `Not applicable`. In same contexts, these things can have a different meaning, For example:

Since `var2` is `NA` for all Male subjects, it may mean "Not applicable", i.e. something that can only be measured in females. Whereas in `var3`, `NA` is more likely to mean "Not available" so real missing data, e.g. lost to follow-up. We will come back to handling missing values later, but let's return to introducing objects and functions.

A table can live anywhere: on paper, in a Spreadsheet, in an SQL database, or it can live in R session's Environment. And yes, R sessions are as fun as they sound, almost as fun as, e.g., music sessions. We usually initiate and interface R using RStudio, but everything we talk about here (objects, functions, sessions, enviroment) also

work when RStudio is not available, but R is. This can be the case if you are working on a supercomputer that can only serve the R Console, and not an RStudio IDE (reminder from first chapter: Integrated Development Environment). So, regualarly shaped data in rows and columns is called table when it lives outside R. Once you read it into R (import it), we call it a tibble. When you are in one of your very cool R sessions and read in a table, it goes into this session's Envionment. There used to be an older version of tables in R - they are called data frames. In most cases, `data frames` and `tibbles` work interchangeably (and both are R objects), but `tibbles` are newer and better. Another great alterntative to base R's `data frames` are `data tables`. In this book, and for most of our day-to-day work these days, we only use tibbles.

So, the tibble `mydata` example of an object that lives in the Environment of your R Session. A function that you may want to apply on numeric data is `mean()`. R functions always have brackets after their name. This is for two reasons. First, to easily differentiate them from objects - which don't have brackets after their name. Second, and more important, we can put arguments in these brackets. Arguments can also be thought of as input, and in data analysis, the most common input for a function is data: we need to give the R function `mean()` some data to average. It does not make sense (nor will it work) to feed it the whole tibble that has multiple columns, including patient IDs and a categorical variable (`sex`). To quickly extract a single column, we use the `$` symbol, like this:

```
mydata$var1
```

```
## [1] 4 1 2 3
```

You can ignore the `## [1]` at the beginning of the extracted values - this is something that becomes more useful when printing multiple lines of data, the number in the square brackets keeps count on how many values we are seeing.

We can then use `mydata$var1` as the first argument of `mean()` by putting it inside its brackets:

```
mean(mydata$var1)
```

```
## [1] 2.5
```

Which tells us that the mean of `var1` (4, 1, 2, 3) is 2.5. In this example, `mydata$var1` is the first and only argument to `mean()`. But what happens if we try to calculate the average value of `var2` (NA, 4, 5, NA)?

```
mean(mydata$var2)
```

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

We get an `NA` ("Not applicable"). We would expect to see an `NA` if we tried to, for example, calculate the average of `sex`:

```
mean(mydata$sex)
```

```
## Warning in mean.default(mydata$sex): argument is not numeric or logical:
## returning NA
```

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

In fact, R then gives as us a pretty clear Error (our first Error in this book!) saying it can't compute the mean of an argument that is not numeric or logical. The sentence actually reads pretty fun, as if R was saying it was not logical to calculate the mean of something that is not numeric. But what R is actually saying that it is happy to calculate the mean of two types of variables: numerics or logicals which is a data type with just two potential values TRUE/FALSE. We will come back to data types shortly.

So `mean(mydata$var2)` does not return an Error, but it also doesn't return the mean of the numeric values included in this column. That is because the column includes missing values (`NA`s),

and R does not want to average over NAs implicitly. It is being cautious - what if you didn't know there were missing values for some patients? If you wanted to compare the means of `var1` and `var2` without any further filtering, you would be comparing samples of different size. Which might be fine if the sample sizes are sufficiently representative and the values are missing at random. Therefore, if you decide to ignore the NAs and want to calculate the mean anyway, you can do so by adding another argument to `mean()`:

```
mean(mydata$var2, na.rm = TRUE)
```

```
## [1] 4.5
```

Adding `na.rm = TRUE` tells are that you are happy for it to calculate the mean of any existing values (but to remove - `na.rm` - the `NA` values). This 'removal' only means excluded from the calculation, it does not affect the actual tibble (`mydata`) holding the columns and rows. R is case sensitive, so it has to be `na.rm`, not `NA.rm` etc. There is no need to try to memorise how the arguments of functions are exactly spelled - this is what the Help tab (press `F1` when the cursor is on the name of the function) can remind you of. Information about functions (Help) is built into R, so an internet connection is not required for this.

Make sure to always separate arguments with commas, or R will give you an error of `Error: unexpected symbol`.

Finally, some functions do not need any arguments to work. A good example is the `Sys.time()` which returns the current time and date. This is very useful when using R to generate and update reports automatically. Including this on the output document means you as well as your colleagues can always be clear on when the results were last updated.

```
Sys.time()
```

```
## [1] "2019-07-17 11:36:37 BST"
```

To summarise, objects and functions work hand in hand. Objects are both an input as well as the output of a function (what the function returns). The data values input into a function are usually its first argument, further arguments can be used to specify a functions behaviour. When we say "the function returns", we are referring to its output. The returned object can be different to its input object. In our `mean()` examples, the input object was a column (`mydata$var1`: 4, 1, 2, 3), whereas the output was a single value: 2.5.

2.3 Working with Objects

It's sometimes difficult to appreciate how coding works without trying it first. These exercises will show you how R works.

We'll first create an object and call it `a`, we will give the object `a` a value of 1. In R the equals `=` sign tells R to give the object on the left of the sign the value of whatever is on the right of the sign.

```
a = 1
```

In your environment panel, you should see `a` appear under the `Values` section.

Now, lets create `b` and give it a value of 2.

```
b = 2
```

Lets now add `a` and `b` together to create the object `c`

```
c = a + b

# Print the value of c to the Console

c # should return the number 3
```

```
## [1] 3
```

All of R is just an extension of this: applying more complex functions (calculations) across more complex objects.

It's important to appreciate that objects can be more than just single numbers too. They can be entire spreadsheets, which in R are known as `tibbles` (or in base R: `data frames`).

Note that many people use `<-` instead of `=`. They mean the same thing in R. `=` and `<-` save what is on the right into the name on the left. There is also a left-to-right operator: `->`.

2.3.1 Exercise

Create 3 new variables, `d`, `e`, `f` with values 6, 7, 8 using the different assignment operators.

```
d  = 6
e <- 7
8 -> f
```

2.4 Loading data

Before we load a new dataset, we should clear our experiments from the previous section. Restart R by pressing Ctrl+Shift+F10 or Select Section -> Restart R from the menu above.

Now the environment is clear, lets load in the data:

```
library(tidyverse) # Tidyverse is the package which contains some of the co

mydata = read_csv("global_burden_disease_short.csv")
```

But how can we look at the data we just loaded? How do we know which variables it contains? Hint: the Environment tab.

2.4.1 Exercise

Answer these question about your data:

1. At present, how many variables are there?
2. How many deaths were there from communicable diseases in 1990? Hint: clicking on columns when Viewing a tibble orders it.

2.4.2 Other ways to investigate objects

In most cases, you can rely on the Environment tab to see how many variables you have. If, however, the dataset you are using is too big to easily navigate within, you might need to use `names(mydata)`, `head(mydata)`, or `str(mydata)`.

Furthermore, we can select a single column using the dollar sign: `$`.

So if we type:

```
mydata$deaths
```

```
##  [1] 16149409 26993493  4325788 15449045 29897069  4639869 14775502
##  [8] 31521934  4776852 13890709 33637815  4833919 12431802 36259550
## [15]  4970846 11809640 38267197  4786929
```

R will give us all the data for that variable.

2.4.3 Exercise

Image source: *https://cran.r-project.org/web/packages/magrittr/vignettes/magrittr.html*

Re-write `names(mydata)` and `head(mydata)` using the pipe (`%>%`). Use the keyboard shortcut `Ctrl+Shift+M` to insert it.

2.4.4 Exercise

How many unique values does the `cause` variable have? Hint: `mydata$cause` piped into `unique()` piped into `length()`.

2.5 Operators

Operators are symbols in R Code that tell R how to handle different pieces of data or objects.

Here are the main operators:

`=, <-, ==, <, >, <=, >=`

Some of these perform a test on data. A good example of this is the '==' operator.

This tells R to compare two things and ask if they are equal. If they are equal R will return 'TRUE', if not R will return 'FALSE'.

On your R cheat sheet, you can see what the others do. Here is a reminder:

Symbol	What does	Example	Example result
= or <-	assigns	`x = 2`	the value of x is now 2
==	Equal?	`x == 2`	TRUE
!=	Not equal?	`x != 1`	TRUE
<	Less than	`x < 2`	FALSE
>	Greater than	`x > 1`	TRUE
<=	Less than or equal to	`x <= 2`	TRUE
>=	Greater than or equal to	`x >= 1`	TRUE
%>%	sends data into a function	`x %>% print()`	2
::	indicates package	`dplyr::count()`	`count()` fn. from the `dplyr` p
->	assigns	`2 -> x`	the value of x is now 2
&	AND	`x > 1 & x < 3`	TRUE
|	OR	`x > 3 | x == 3`	TRUE
%in%	is value in list	`x %in% c(1,2,3)`	TRUE
$	select a column	`mydata$year`	1990,1996,...
c()	combines values	`c(1, 2)`	1, 2
#	comment	`#Riinu changed this`	ignored by R

For example, if we wanted to select the years in the Global Burden of disease study after 2000 (and including 2000) we could type the following:

```
mydata %>%
  filter(year >= 2000)
```

To save this as a new object we would then write:

```
mydata_out = mydata %>%
  filter(year >= 2000)

# Or we could write

mydata %>%
  filter(year >= 2000) -> mydata_out
```

How would you change the above code to only include years greater than 2000 (so not including 2000 itself too)? Hint: look at the table of operators above (also in your HealthyR QuickStart Sheet).

2.5.1 Exercise

Modify the above example to filter for only year 2000, not all years greater than 2000. Save it into a variable called `mydata_year2000`.

2.5.2 Exercise

Let's practice this and combine multiple selections together.

This '|' means OR and '&' means AND.

From `mydata`, select the lines where year is either 1990 or 2013 and cause is "Communicable diseases":

```
new_data_selection = mydata %>%
  filter( (year == 1990 | year == 2013) & cause == "Communicable diseases")

# Or we can get rid of the extra brackets around the years
# by moving cause into a new filter on a new line:

new_data_selection = mydata %>%
  filter(year == 1990 | year == 2013) %>%
  filter(cause == "Communicable diseases")
```

2.6 Types of variables

consider structuring as per here: https://finalfit.org/articles/data_prep.html

Like many other types of statistical software, R needs to know the variable type of each column. The main types are:

2.6.1 Characters

Characters (sometimes referred to as *strings* or *character strings*) in R are letters, words, or even whole sentences (an example of this may be free text comments). We can specify these using the `as.character()` function. Characters are displayed in-between `""` (or `''`).

2.6.2 Factors

Factors are fussy characters. Factors are fussy because they have something called levels. Levels are all the unique values this variable could take - e.g. like when we looked at `mydata$cause %>% unique()`. Using factors rather than just characters can be useful because:

- The values factor levels can take is fixed. For example, if the levels of your column called `sex` are "Male" and "Female" and you try to add a new patient where sex is called just "F" you will get a warning from R. If `sex` was a character column rather than a factor R would have no problem with this and you would end up with "Male", "Female", and "F" in your column.
- Levels have an order. When we plotted the different causes of death in the last session, R ordered them alphabetically (because `cause` was a character rather than a factor). But if you want to use a non-alphabetical order, e.g. "Communicable diseases"-"Non-communicable diseases"-"Injuries", we need make `cause`

into a factor. Making a character column into a factor enables us to define and change the order of the levels. Furthermore, there are useful tools such as `fct_inorder` or `fct_infreq` that can order factor levels for us.

These can be huge benefits, especially as a lot of medical data analyses include comparing different risks to a reference level. Nevertheless, the fussiness of factors can sometimes be unhelpful or even frustrating. For example, if you really did want to add a new level to your `gender` column (e.g., "Prefer not to say") you will either have to convert the column to a character, add it, and convert it back to a factor, or use `fct_expand` to add the level and then add your new line.

2.6.2.1 Exercise

Temporarily type `fct_inorder` anywhere in your script, then press F1. Read the **Description** in the Help tab and discuss with your neighbour how `fct_inorder` and `fct_infreq` would order your factor levels.

2.6.3 Numbers

Self-explanatory! These are numbers. In R, we specify these using the `as.numeric()` function. Numbers without decimal places are sometimes called integers. Click on the blue arrow in front of `mydata` in the Environment tab and see that `year` is an `int` (integer) whereas `deaths` is a `num` (numeric).

2.6.4 Specifying variable types

```
as.character(mydata$cause)

as.numeric(mydata$year)
```

```
factor(mydata$year)

#Lets save the cause as a factor

mydata$cause = factor(mydata$cause)

#Now lets print it out

mydata$cause
```

2.6.5 Exercise

Change the order of the levels in `mydata$cause` so that "Non-communicable diseases" come before "Injuries". Hint: use F1 to investigate examples of how `fct_relevel()` works.

2.7 Importing data

For historical reasons, R's default functions (e.g. `read.csv()` or `data.frame()`) convert all characters to factors automatically (for more on this see forcats.tidyverse.org[1]. But it is usually more convenient to deal with characters and convert some of the columns to factors when necessary.

Base R:

```
mydata = read.csv("global_burden_disease_short.csv", stringsAsFactors = FALS
```

The tidyverse version, `read_csv()`, has `stringsAsFactors` set to

[1] http://forcats.tidyverse.org

FALSE by default (and it is a lot faster than `read.csv()` when reading in large datasets).

Tidyverse:

```
mydata = read_csv("global_burden_disease_short.csv")
```

```
## Parsed with column specification:
## cols(
##   cause = col_character(),
##   year = col_double(),
##   deaths = col_double()
## )
```

You can use the "Import Dataset" button in the Environment tab to get the code for importing data from Excel, SPSS, SAS, or Stata.

2.8 Adding columns to dataframes

If we wanted to add in a new column or variable to our data, we can simply use the dollar sign '$' to create a new variable inside a pre-existing piece of data:

```
mydata$new = 1

mydata$new2 = 1:18
```

Run these lines and click on `mydata` in the Environment tab to check this worked as expected.

Conversely, if we want to delete a specific variable or column we can use the 'NULL' function, or alternatively ask R to `select()` the data without the new variable included.

```
mydata$new = NULL

mydata = mydata %>%
  select(-new2)
```

We can make new variables using calculations based on variables in the data too.

The mutate function is useful here. All you have to specify within the mutate function is the name of the variable (this can be new or pre-existing) and where the new data should come from.

There are two equivalent ways of defining new columns based on a calculation with a previous column:

mutate formally introduced in later chapter. Need to think how best to present this in book.

```
# First option

mydata$years_from_1990 = mydata$year - 1990
mydata$deaths_millions = mydata$deaths/1000000

# Second option (mutate() function)

mydata = mydata %>%
  mutate(years_from_1990 = year-1990,
         deaths_millions = deaths/1000000)
```

Throughout this course we will be using both of these ways to create or modify columns. The first option (using the `$`) can look neater when changing a single variable, but when combining multiple ones you will end up repeating `mydata$`. `mutate()` removes the duplication, but it does add a new line and brackets.

2.9 Rounding numbers

We can use `round()` to round the new variables to create integers.

2.9.1 Exercise

Round the new column `deaths_millions` to no decimals:

```
##  [1] 16 27  4 15 30  5 15 32  5 14 34  5 12 36  5 12 38  5
```

- How would you round it to 2 decimals? Hint: use F1 to investigate `round()`.
- What do `ceiling()` and `floor()` do? Hint: sometimes you want to round a number up or down.

2.10 The combine function: c()

The combine function combines several values: `c()`

The combine function can be used with numbers or characters (like words or letters):

```
examplelist = c("Red", "Yellow", "Green", "Blue")

# Ask R to print it by executing it on its own line

examplelist
```

```
## [1] "Red"    "Yellow" "Green"  "Blue"
```

2.10.1 Exercise

There are 18 lines (observations) in mydata. Create a new variable using `c()` with 18 values (numbers, words, whichever you like, e.g. like we created `examplelist`). Then add it as new column to `mydata$newlist`. Advanced version: do this using a combination of `rep()` and `c()`.

2.11 The paste() function

The paste() function is used to paste several words or numbers into one character variable/sentence.

In the paste function we need to specify what we would like to combine, and what should separate the components. By default, the separation is a space, but we can change this using the sep = option within the paste function.

So, for example if we wanted to make a sentence:

```
#
#paste("Edinburgh", "is", "Great")

# Lets add in full stops


paste("Edinburgh", "is", "Great", sep = ".")
```

```
## [1] "Edinburgh.is.Great"
```

```
# separator needs to go in "" as it is a character


# If we really like Edinburgh

#paste("Edinburgh", "is", "Great", sep = "!")

# If we want to make it one word

#paste("Edinburgh", "is", "Great", sep = "") # no separator (still need the
```

We can also join two different variables together using paste():

```
paste("Year is", mydata$year)
```

```
##  [1] "Year is 1990" "Year is 1990" "Year is 1990" "Year is 1995"
##  [5] "Year is 1995" "Year is 1995" "Year is 2000" "Year is 2000"
##  [9] "Year is 2000" "Year is 2005" "Year is 2005" "Year is 2005"
## [13] "Year is 2010" "Year is 2010" "Year is 2010" "Year is 2013"
## [17] "Year is 2013" "Year is 2013"
```

2.11.1 Exercise

Fix this code:

Hint: Think about characters and quotes!

```
paste(Today is, Sys.Date() )
```

2.12 Combining two dataframes

For combining dataframes based on shared variables we use the joins: `left_join()`, `right_join()`, `inner_join()`, or `full_join()`. Let's split some of the variables in `mydata` between two new dataframes: `first_data` and `second_data`. For demonstrating the difference between the different joins, we will only include a subset (first 6 rows) of the dataset in `second_data`:

```
first_data  = select(mydata, year, cause, deaths_millions)
second_data = select(mydata, year, cause, deaths_millions) %>% slice(1:6)

# change the order of rows in first_data to demosntrate the join does not r
first_data = arrange(first_data, deaths_millions)
```

```
combined_left  =  left_join(first_data, second_data)
combined_right = right_join(first_data, second_data)
combined_inner = inner_join(first_data, second_data)
combined_full  =  full_join(first_data, second_data)
```

Those who have used R before, or those who come across older scripts will have seen `merge()` instead of the joins. `merge()` works similarly to joins, but instead of having the four options defined clearly at the front, you would have had to use the `all = FALSE, all.x = all, all.y = all` arguments.

2.12.1 Exercise

Investigate the four new dataframes called `combined_` using the Environment tab and discuss how the different joins (left, right, inner, full) work.

2.13 The `summary()` function

In R, the `summary()` function provides a quick way of summarising both data or the results of statistical tests.

Lets get a quick summary of all the variables inside the Global Burden of Disease dataset. It will work for whole datasets and single variables too.

```
mydata %>% summary()
```

```
##    cause           year       deaths      years_from_1990
## Length:18       Min.   :1990   Min.   : 4325788   Min.   : 0.00
## Class :character  1st Qu.:1995   1st Qu.: 4868151   1st Qu.: 5.00
## Mode  :character  Median :2002   Median :14333106   Median :12.50
##                Mean   :2002   Mean   :17189854   Mean   :12.17
##                3rd Qu.:2010   3rd Qu.:29171175   3rd Qu.:20.00
##                Max.   :2013   Max.   :38267197   Max.   :23.00
##  deaths_millions
##  Min.   : 4.00
##  1st Qu.: 5.00
##  Median :14.50
##  Mean   :17.22
##  3rd Qu.:29.25
##  Max.   :38.00
```

This even works on statistical tests (we will learn more about these later):

```
# lm stands for linear model
lm(deaths ~ year, data = mydata) %>% summary()
```

```
##
## Call:
## lm(formula = deaths ~ year, data = mydata)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -13480641 -11791203  -2889909  12818624  19999627
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -181988644  736014812  -0.247    0.808
## year             99482     367606   0.271    0.790
##
```

```
## Residual standard error: 12590000 on 16 degrees of freedom
## Multiple R-squared:  0.004556,  Adjusted R-squared:  -0.05766
## F-statistic: 0.07324 on 1 and 16 DF,  p-value: 0.7901
```

2.13.1 When pipe sends data to the wrong place

Note that our usual way of doing things with the pipe would not work here:

```
mydata %>%
  lm(deaths ~ year) %>%
  summary()
```

This is because the pipe tries to send data into the first place of the function (first argument), but `lm()` wants the formula (`deaths ~ year`) first, then the dataframe. We can bypass this using `data = .` to tell the pipe where to put mydata:

```
mydata %>%
  lm(deaths ~ year, data = .) %>%
  summary()
```

2.13.2 Exercise

Try adding a new variable called `death_over_10m` which indicates whether there were more than 10 million deaths for a cause. The new variable should take the form 'Yes' or 'No'.

Then make it into a factor.

Then use `summary()` to find out about it!

```
mydata = mydata %>%
  mutate(death_over_10m = ifelse(deaths >= 10000000, "Yes", "No")) # Using i
```

```
mydata$death_over_10m = as.factor(mydata$death_over_10m)

mydata$death_over_10m %>% summary()
```

```
##  No Yes
##   6  12
```

2.14 Extra: Creating a dataframe from scratch

It is rare that you will need to create a data frame by hand as most of the time you will be reading in a data from a .csv or similar. But in some cases (e.g. when creating special labels for a plot) it might be useful, so this is how to create one:

```
patient_id = paste0("ID", 1:10)
sex        = rep(c("Female", "Male"), 5)
age        = 18:27

newdata = data_frame(patient_id, sex, age)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
# same as

newdata      = data_frame(
  patient_id = paste0("ID", 1:10), #note the commas
  sex        = rep(c("Female", "Male"), 5),
  age        = 18:27
)
```

If we used `data.frame()` instead of `data_frame()`, all our char-

acter variables (`patient_id`, `sex`) would become factors automatically. This might make sense for `sex`, but it doesn't for `patient_id`.

2.14.1 Exercise

Create a new dataframe called `my_dataframe` that looks like this:

Hint: Use the functions `paste0()`, `seq()` and `rep()`

```
## # A tibble: 10 x 3
##    patient_id   age sex
##    <chr>      <dbl> <chr>
##  1 ID11          15 Male
##  2 ID12          20 Male
##  3 ID13          25 Male
##  4 ID14          30 Male
##  5 ID15          35 Male
##  6 ID16          40 Female
##  7 ID17          45 Female
##  8 ID18          50 Female
##  9 ID19          55 Female
## 10 ID20          60 Female
```

2.15 Solutions

2.5.3

```
mydata %>% names()
mydata %>% head()
mydata %>% str()
```

2.5.4

```
mydata$cause %>% unique() %>% length()
```

```
## [1] 3
```

2.6.2

```
mydata_year2000 = mydata %>%
  filter(year == 2000)
```

2.7.5

```
mydata$cause %>% fct_relevel("Injuries", after = 1)
```

2.10.1

```
mydata$deaths_millions = round(mydata$deaths_millions)

# or
mydata$deaths_millions = mydata$deaths_millions %>% round()
```

2.11.1

```
examplelist = c("Red", "Yellow", "Green", "Blue",
                "Red", "Yellow", "Green", "Blue",
                "Red", "Yellow", "Green", "Blue",
                "Red", "Yellow", "Green", "Blue",
                "Green", "Blue")

#Let's see what we've made by using print

mydata$newlist = examplelist


# using rep()
```

```
examplelist2 = rep(c("Green", "Red"), 9)
```

2.12.1

```
paste("Today is", Sys.Date())
```

2.15.1

```
my_dataframe = data_frame(
  patient_id = paste0("ID", 11:20),
  age        = seq(15, 60, 5),
  sex        = c( rep("Male", 5), rep("Female", 5))
)
```

3

Summarising data

In this session we will get to know our three best friends for summarising data: `group_by()`, `summarise()`, and `mutate()`.

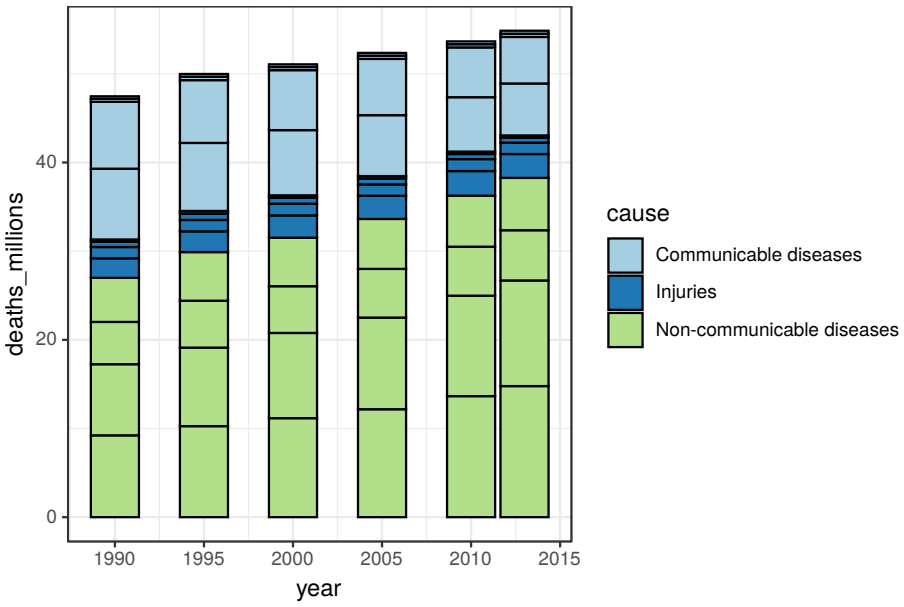
3.1 Data

In Session 2, we used a very condensed version of the Global Burden of Disease data. We are now going back to a longer one and we will learn how to summarise it ourselves.

```
source("healthyr_theme.R")
load("global_burden_disease_long.rda")
```

We were already using this longer dataset in Session 1, but with `colour=cause` to hide the fact that the total deaths in each year was made up of 12 groups of data (as the black lines on the bars indicate):

```
mydata %>%
    ggplot(aes(x = year, y = deaths_millions, fill = cause))+
    geom_col(colour = "black")
```

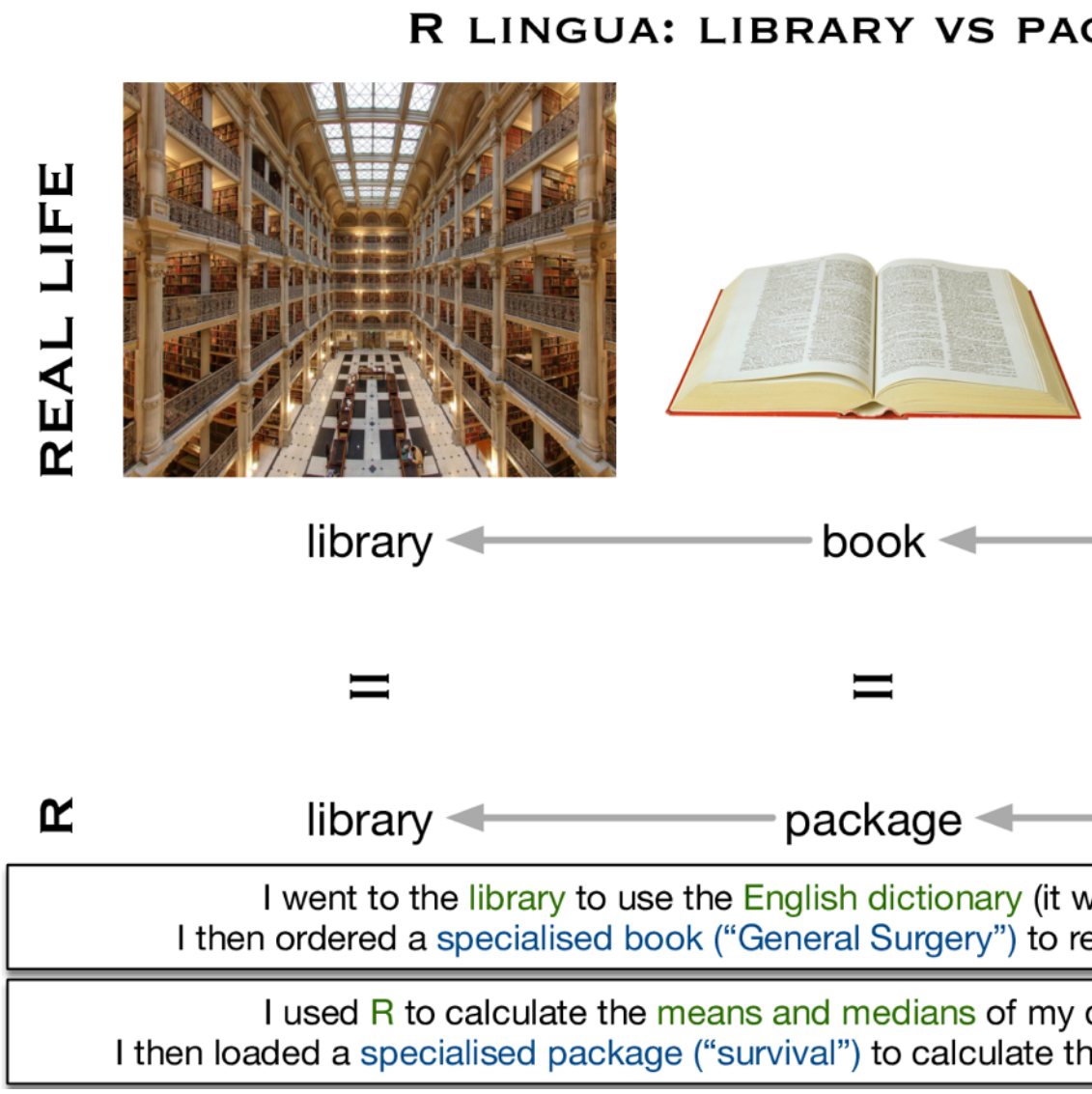



```
mydata %>%
    filter(year == 1990)
```

```
##    location               cause    sex year deaths_millions
## 1  Developing Non-communicable diseases   Male 1990      9.2277141
## 2  Developing Non-communicable diseases Female 1990      8.0242455
## 3   Developed Non-communicable diseases   Male 1990      4.7692902
## 4   Developed Non-communicable diseases Female 1990      4.9722431
## 5  Developing             Injuries   Male 1990      2.2039625
## 6  Developing             Injuries Female 1990      1.2698308
## 7   Developed             Injuries   Male 1990      0.5941184
## 8   Developed             Injuries Female 1990      0.2578759
## 9  Developing    Communicable diseases   Male 1990      7.9819728
## 10 Developing    Communicable diseases Female 1990      7.5416376
## 11  Developed    Communicable diseases   Male 1990      0.3387820
## 12  Developed    Communicable diseases Female 1990      0.2870169
```

3.2 Tidyverse packages: ggplot2, dplyr, tidyr, etc.

Most of the functions introduced in this session come from the tidyverse family (`http://tidyverse.org/`), rather than Base R. Including `library(tidyverse)` in your script loads a list of packages: ggplot2, dplyr, tidry, forcats, etc.




```
library(tidyverse)
```

3.3 Basic functions for summarising data

You can always pick a column and ask R to give you the `sum()`, `mean()`, `min()`, `max()`, etc. for it:

```
mydata$deaths_millions %>% sum()
```

```
## [1] 309.4174
```

```
mydata$deaths_millions %>% mean()
```

```
## [1] 4.297463
```

But if you want to get the total number of deaths for each `year` (or `cause`, or `sex`, whichever grouping variables you have in your dataset) you can use `group_by()` and `summarise()` that make subgroup analysis very convenient and efficient.

3.4 Subgroup analysis: `group_by()` and `summarise()`

The `group_by()` function tells R that you are about to perform subgroup analysis on your data. It retains information about your groupings and calculations are applied on each group separately. To go back to summarising the whole dataset again use `ungroup()`. Note that `summarise()` is different to the `summary()` function we used in Session 2.

With `summarise()`, we can calculate the total number of deaths per year:

```
mydata %>%
    group_by(year) %>%
    summarise(total_per_year = sum(deaths_millions)) ->
    summary_data1


mydata %>%
    group_by(year, cause) %>%
    summarise(total_per_cause = sum(deaths_millions)) ->
    summary_data2
```

- `summary_data1` includes the total number of deaths per year.
- `summary_data2` includes the number of deaths per cause per year.

year	total_per_year
1990	47
1995	50
2000	51
2005	52
2010	54
2013	55

year	cause	total_per_cause
1990	Communicable diseases	16
1990	Injuries	4
1990	Non-communicable diseases	27
1995	Communicable diseases	15
1995	Injuries	5
1995	Non-communicable diseases	30

... remaining years omitted from printing.

3.4.1 Exercise

Compare the sizes - number of rows (observations) and number of columns (variables) - of `mydata`, `summary_data1`, and `summary_data2` (in the Environment tab).

- Convince yourself that for 1990, deaths by the three causes (`summary_data2`) add up to total deaths per year (`summary_data1`).
- `summary_data2` has exactly 3 times as many rows as `summary_data1`. Why?
- `mydata` has 5 variables, whereas the summarised dataframes have 2 and 3. Which variables got dropped? Why?

3.4.2 Exercise

For each cause, calculate its percentage to total deaths in each year.

Hint: Use `full_join()` on `summary_data1` and `summary_data2`.

Solution:

```
alldata = full_join(summary_data1, summary_data2)
```

```
## Joining, by = "year"
```

```
alldata$percentage = 100*alldata$total_per_cause/alldata$total_per_year %>% 
```

`round()` defaults to 0 digits. If you want to round to a specified number of decimal places, use, e.g., round(digits = 2).

TABLE 3.1: alldata

year	total_per_year	cause	total_per_cause	percentage
1990	47	Communicable diseases	16	34
1990	47	Injuries	4	9
1990	47	Non-communicable diseases	27	57
1995	50	Communicable diseases	15	31
1995	50	Injuries	5	9
1995	50	Non-communicable diseases	30	60

3.5 `mutate()`

Mutate works similarly to `summarise()` (as in it respects groupings set with `group_by()`), but it adds a new column into the original data. `summarise()`, on the other hand, condenses the data into a minimal table that only includes the variables specifically asked for.

3.5.1 Exercise

Investigate these examples to learn how `summarise()` and `mutate()` differ.

```
summarise_example = mydata %>%
    summarise(total_deaths = sum(deaths_millions))

mutate_example = mydata %>%
    mutate(total_deaths = sum(deaths_millions))
```

```
mutate_example %>%
  slice(1:5) %>%
```

TABLE 3.2: summarise example

total_deaths
309

TABLE 3.3: mutate_example

location	cause	sex	year	deaths_millions	total_deaths
Developing	Non-communicable diseases	Male	1990	9	309
Developing	Non-communicable diseases	Female	1990	8	309
Developed	Non-communicable diseases	Male	1990	5	309
Developed	Non-communicable diseases	Female	1990	5	309
Developing	Non-communicable diseases	Male	1995	10	309

```
knitr::kable(digits = 0,
             booktabs = TRUE,
             caption = "mutate\\_example",
             align = "c")
```

You should see that `mutate()` adds the same total number (309) to every line in the dataframe.

3.5.2 Optional advanced exercise

Based on what we just observed on how `mutate()` adds a value to each row, can you think of a way to redo **Exercise 3.4.2** without using a join? Hint: instead of creating `summary_data1` (total deaths per year) as a separate dataframe which we then merge with `summary_data2` (total deaths for all causes per year), we can use `mutate()` to add `total_per_year` to each row.

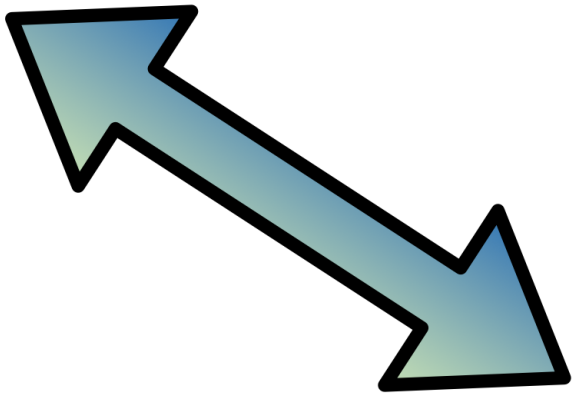
```
mydata %>%
    group_by(year, cause) %>%
```

```
    summarise(total_per_cause = sum(deaths_millions)) %>%
    group_by(year) %>%
    mutate(total_per_year = sum(total_per_cause)) %>%
    mutate(percentage = 100*total_per_cause/total_per_year) -> alldata
```

3.6 Wide vs long: `spread()` and `gather()`

Developed countries:	1990	1995	2000	2005	20
Communicable diseases	0.6	0.7	0.7	0.7	
Injuries	0.9	1	0.9	0.9	
Non-communicable diseases	9.7	10.8	10.7	11.1	1

Developing countries:	1990	1995	2000	2005	20
Communicable diseases	15.5	14.8	14.1	13.2	1
Injuries	3.5	3.6	3.8	3.9	
Non-communicable diseases	17.3	19.1	20.8	22.5	



locatio
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo
Develo

3.6.1 Wide format

Although having data in the long format is very convenient for R, for publication tables, it makes sense to spread some of the values out into columns:

```
alldata %>%
    mutate(percentage = paste0(round(percentage, 2), "%")) %>%
    select(year, cause, percentage) %>%
    spread(cause, percentage)
```

```
## # A tibble: 6 x 4
## # Groups:   year [6]
##   year `Communicable diseases` Injuries `Non-communicable diseases`
##   <int> <chr>                   <chr>    <chr>
## 1  1990 34.02%                  9.11%    56.87%
## 2  1995 30.91%                  9.28%    59.81%
## 3  2000 28.93%                  9.35%    61.72%
## 4  2005 26.53%                  9.23%    64.24%
## 5  2010 23.17%                  9.26%    67.57%
## 6  2013 21.53%                  8.73%    69.75%
```

- `select()` pick the variables you want to keep. Try running the lines until `spread()` to see how it works.

3.6.2 Exercise

Calculate the percentage of male and female deaths for each year. Spread it to a human readable form:

Hints:

- create `summary_data3` that includes a variable called `total_per_sex`
- merge `summary_data1` and `summary_data3` into a new data frame
- calculate the percentage of `total_per_sex` to `total_per_year`
- round, add % labels

- spread

Solution:

```
mydata %>%
    group_by(year) %>%
    summarise(total_per_year = sum(deaths_millions)) ->
    summary_data1

mydata %>%
    group_by(year, sex) %>%
    summarise(total_per_sex = sum(deaths_millions)) ->
    summary_data3

alldata = full_join(summary_data1, summary_data3)
```

```
## Joining, by = "year"
```

```
result_spread = alldata %>%
  mutate(percentage = round(100*total_per_sex/total_per_year, 0)) %>%
  mutate(percentage = paste0(percentage, "%")) %>%
  select(year, sex, percentage) %>%
  spread(sex, percentage)

result_spread
```

```
## # A tibble: 6 x 3
##    year Female Male
##   <int> <chr>  <chr>
## 1  1990 47%    53%
## 2  1995 47%    53%
## 3  2000 46%    54%
## 4  2005 46%    54%
## 5  2010 46%    54%
## 6  2013 45%    55%
```

And save it into a csv file using `write_csv()`:

```
write_csv(result_spread, "gbd_genders_summarised.csv")
```

You can open a csv file with Excel and copy the table into Word or PowerPoint for presenting.

3.6.3 Long format

The opposite of `spread()` is `gather()`:

- The first argument is a name for the column that will include columns gathered from the wide columns (in this example, `Male` and `Female` are gathered into `sex`).
- The second argument is a name for the column that will include the values from the wide-format columns (the values from `Male` and `Female` are gathered into `percentage`).
- Any columns that already are condensed (e.g. year was in one column, not spread out like in the pre-course example) must be included with a negative (i.e. -year).

```
result_spread %>%
  gather(sex, percentage, -year)
```

```
## # A tibble: 12 x 3
##     year sex    percentage
##    <int> <chr>  <chr>
##  1  1990 Female 47%
##  2  1995 Female 47%
##  3  2000 Female 46%
##  4  2005 Female 46%
##  5  2010 Female 46%
##  6  2013 Female 45%
##  7  1990 Male   53%
##  8  1995 Male   53%
##  9  2000 Male   54%
## 10  2005 Male   54%
```

```
## 11  2010 Male   54%
## 12  2013 Male   55%
```

3.6.4 Exercise

Test what happens when you

- Change the order of sex and percentage:

```
result_spread %>%
  gather(percentage, sex, -year)
```

Turns out in the above example, `percentage` and `sex` were just label you assigned to the gathered columns. It could be anything, e.g.:

```
result_spread %>%
  gather(`look-I-gathered-sex`, `values-Are-Here`, -year)
```

- What happens if we omit `-year`:

```
result_spread %>%
  gather(sex, percentage)
```

`-year` was telling R we don't want the year column to be gathered together with Male and Female, we want to keep it as it is.

3.7 Sorting: `arrange()`

To reorder data ascendingly or descendingly, `use arrange()`:

```
mydata %>%
    group_by(year) %>%
    summarise(total = sum(deaths_millions))  %>%
    arrange(-year) # reorder after summarise()
```

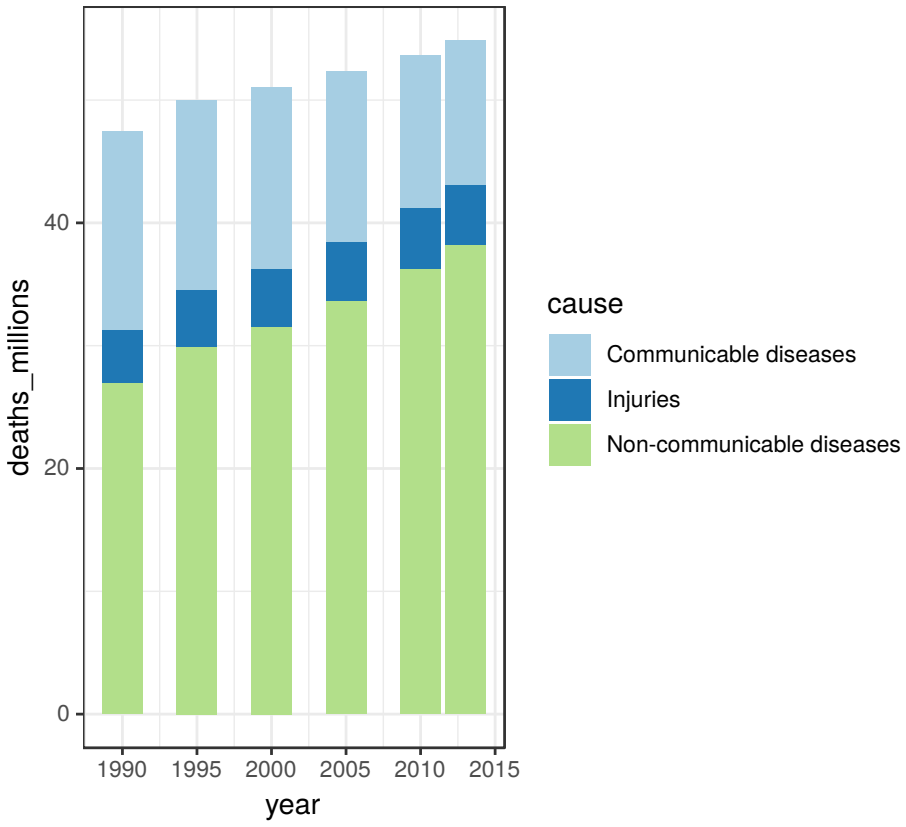
3.8 Factor handling

We talked about the pros and cons of working with factors in Session 2. Overall, they are extremely useful for the type of analyses done in medical research.

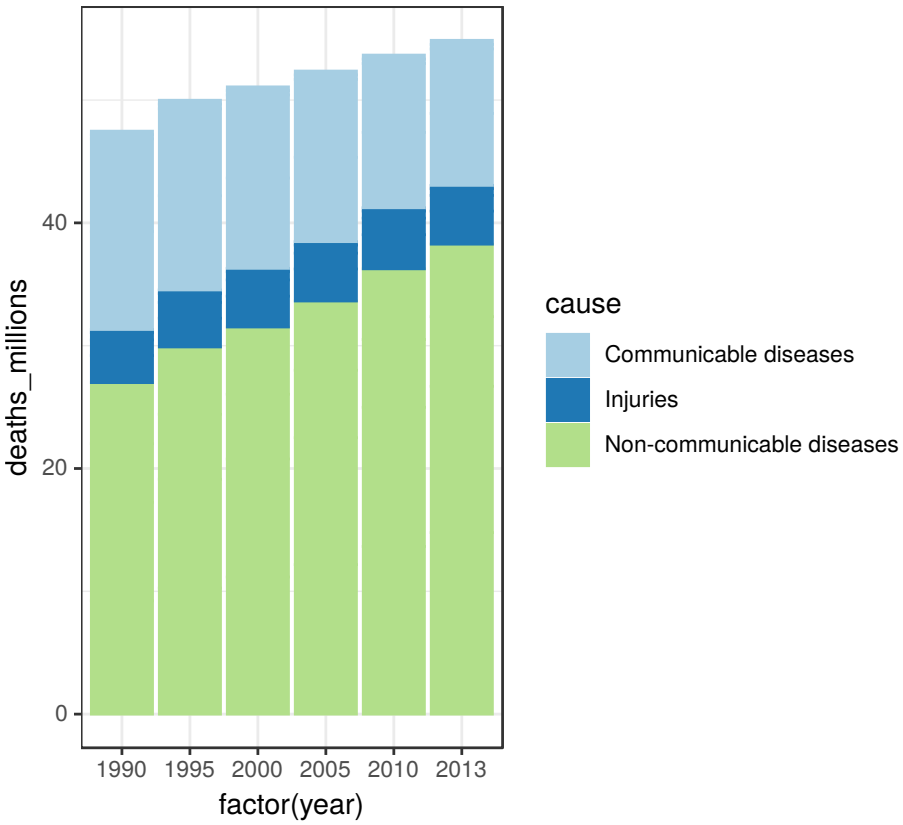
3.8.1 Exercise

Explain how and why these two plots are different.

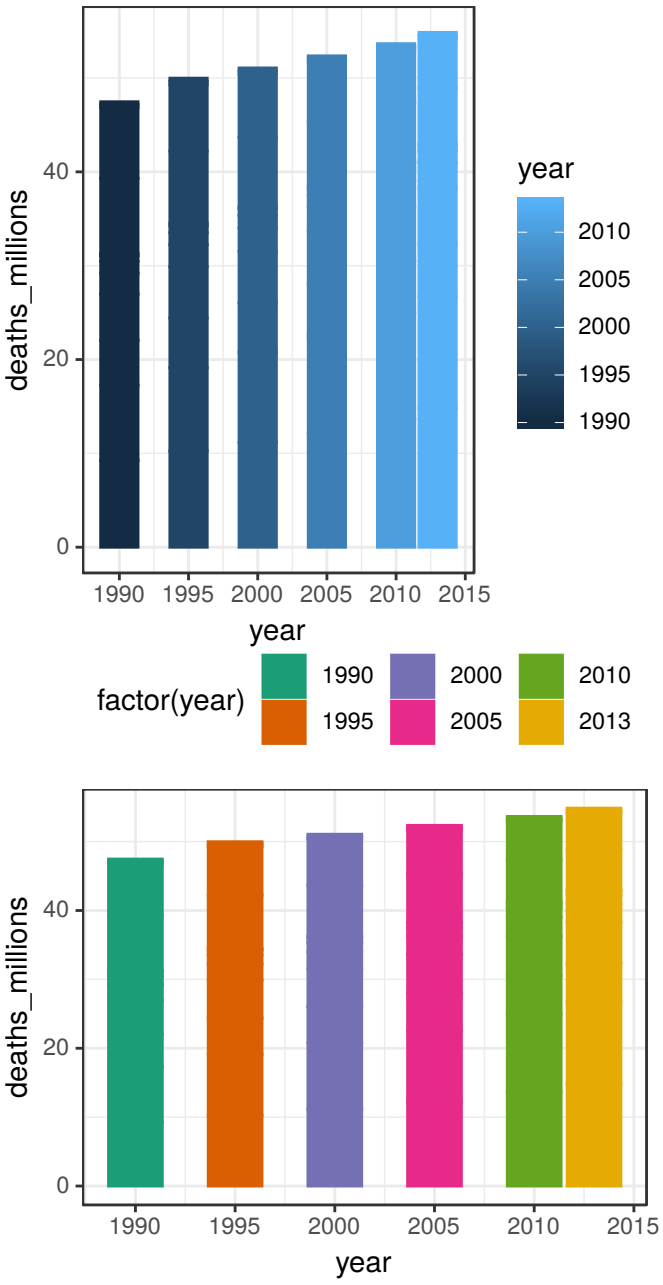
```
mydata %>%
    ggplot(aes(x = year, y = deaths_millions, fill = cause))+
    geom_col()
```




```
mydata %>%
    ggplot(aes(x = factor(year), y = deaths_millions, fill = cause, colour =
    geom_col()
```



What about these?



These illustrate why it might sometimes be useful to use numbers as factors - on the second one we have used `fill = factor(year)`

as the fill, so each year gets a distinct colour, rather than a gradual palette.

3.8.2 `fct_collapse()` - grouping levels together

```
mydata$cause  %>%
    fct_collapse("Non-communicable and injuries" = c("Non-communicable disea
    mydata$cause2

mydata$cause %>% levels()
```

```
## [1] "Communicable diseases"     "Injuries"
## [3] "Non-communicable diseases"
```

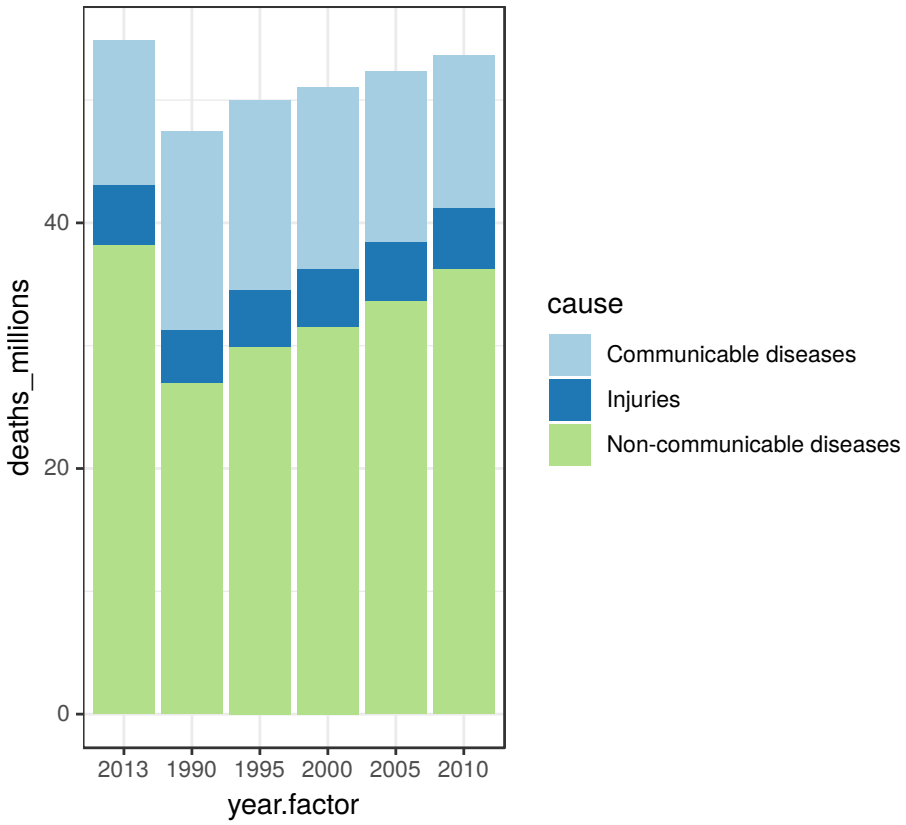
```
mydata$cause2 %>% levels()
```

```
## [1] "Communicable diseases"      "Non-communicable and injuries"
```

3.8.3 `fct_relevel()` - change the order of levels

Another reason to sometimes make a numeric variable into a factor is that we can then reorder it for the plot:

```
mydata$year %>%
  factor() %>%
    fct_relevel("2013") -> #brings 2013 to the front
    mydata$year.factor

source("1_source_theme.R")

mydata %>%
    ggplot(aes(x=year.factor, y=deaths_millions, fill=cause))+
    geom_col()
```



3.8.4 `fct_recode()` - rename levels

```
mydata$cause %>%
    levels()  # levels() lists the factor levels of a column
```

```
## [1] "Communicable diseases"     "Injuries"
## [3] "Non-communicable diseases"
```

```
mydata$cause %>%
    fct_recode("Deaths from injury" = "Injuries") %>%
    levels()
```

```
## [1] "Communicable diseases"     "Deaths from injury"
## [3] "Non-communicable diseases"
```

3.8.5 Converting factors to numbers

MUST REMEMBER: factor needs to become `as.character()` before converting to numeric or date! Factors are actually stored as labelled integers (so like number codes), only the function `as.character()` will turn a factor back into a collated format which can then be converted into a number or date.

3.8.6 Exercise

Investigate the two examples converting the `year.factor` variable back to a number.

```
mydata$year.factor
```

```
##  [1] 1990 1990 1990 1990 1995 1995 1995 1995 2000 2000 2000 2000 2005 2005
## [15] 2005 2005 2010 2010 2010 2010 2013 2013 2013 2013 1990 1990 1990 1990
## [29] 1995 1995 1995 1995 2000 2000 2000 2000 2005 2005 2005 2005 2010 2010
## [43] 2010 2010 2013 2013 2013 2013 1990 1990 1990 1990 1995 1995 1995 1995
## [57] 2000 2000 2000 2000 2005 2005 2005 2005 2010 2010 2010 2010 2013 2013
## [71] 2013 2013
## Levels: 2013 1990 1995 2000 2005 2010
```

```
mydata$year.factor %>%
    as.numeric()
```

```
##  [1] 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 6 6 6 6 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4
## [36] 4 5 5 5 5 6 6 6 6 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 6 6 6 6 1 1
## [71] 1 1
```

```
mydata$year.factor %>%
```

```
    as.character() %>%
    as.numeric()
```

```
##  [1] 1990 1990 1990 1990 1995 1995 1995 1995 2000 2000 2000 2000 2005 2005
## [15] 2005 2005 2010 2010 2010 2010 2013 2013 2013 2013 1990 1990 1990 1990
## [29] 1995 1995 1995 1995 2000 2000 2000 2000 2005 2005 2005 2005 2010 2010
## [43] 2010 2010 2013 2013 2013 2013 1990 1990 1990 1990 1995 1995 1995 1995
## [57] 2000 2000 2000 2000 2005 2005 2005 2005 2010 2010 2010 2010 2013 2013
## [71] 2013 2013
```

3.9 Long Exercise

This exercise includes multiple steps, combining all of the above.

First, create a new script called "2_long_exercise.R". Then Restart your R session, add `library(tidyverse)` and load `"global_burden_disease_long.rda"`.

- Calculate the total number of deaths in Developed and Developing countries. Hint: use `group_by(location)` and `summarise(new-column-name = sum(variable-to-sum))`.
- Calculate the total number of deaths in Developed and Developing countries and for men and women. Hint: this is as easy as adding `, sex` to `group_by()`.
- Filter for 1990.
- `spread()` the `location` column.

```
## # A tibble: 2 x 3
##   sex    Developed Developing
##   <fct>      <dbl>      <dbl>
## 1 Female      5.52       16.8
## 2 Male        5.70       19.4
```

3.10 Extra: formatting a table for publication

Creating a publication table with both the total numbers and percentages (in brackets) + using `formatC()` to retain trailing zeros:

```
# Let's use alldata from Exercise 5.2:

mydata %>%
    group_by(year, cause) %>%
    summarise(total_per_cause = sum(deaths_millions)) %>%
```

```
    group_by(year) %>%
    mutate(total_per_year = sum(total_per_cause)) %>%
    mutate(percentage = 100*total_per_cause/total_per_year) -> alldata

alldata %>%
    mutate(total_percentage =
                    paste0(round(total_per_cause, 1)  %>% formatC(1, format
                           " (", round(percentage, 1) %>% formatC(1, format
                           "%)"
                           )
                    ) %>%
    select(year, cause, total_percentage) %>%
    spread(cause, total_percentage)
```

```
## # A tibble: 6 x 4
## # Groups:   year [6]
##   year `Communicable diseases` Injuries  `Non-communicable diseases`
##   <int> <chr>                    <chr>      <chr>
## 1  1990 16.1 (34.0%)          4.3 (9.1%) 27.0 (56.9%)
## 2  1995 15.4 (30.9%)          4.6 (9.3%) 29.9 (59.8%)
## 3  2000 14.8 (28.9%)          4.8 (9.4%) 31.5 (61.7%)
## 4  2005 13.9 (26.5%)          4.8 (9.2%) 33.6 (64.2%)
## 5  2010 12.4 (23.2%)          5.0 (9.3%) 36.3 (67.6%)
## 6  2013 11.8 (21.5%)          4.8 (8.7%) 38.3 (69.7%)
```

3.11 Solution: Long Exercise

```
mydata %>%
  filter(year == 1990) %>%
  group_by(location, sex) %>%
```

```
  summarise(total_deaths = sum(deaths_millions)) %>%
  spread(location, total_deaths)
```

4

Different types of plots

4.1 Data

We will be using the gapminder dataset:

```
library(tidyverse)
library(gapminder)

mydata = gapminder

summary(mydata)
```

```
##       country       continent      year         lifeExp
## Afghanistan: 12   Africa  :624   Min.   :1952   Min.   :23.60
## Albania    : 12   Americas:300   1st Qu.:1966   1st Qu.:48.20
## Algeria    : 12   Asia    :396   Median :1980   Median :60.71
## Angola     : 12   Europe  :360   Mean   :1980   Mean   :59.47
## Argentina  : 12   Oceania : 24   3rd Qu.:1993   3rd Qu.:70.85
## Australia  : 12                  Max.   :2007   Max.   :82.60
##  (Other)    :1632
##       pop              gdpPercap
##  Min.   :6.001e+04   Min.   :   241.2
##  1st Qu.:2.794e+06   1st Qu.:  1202.1
##  Median :7.024e+06   Median :  3531.8
##  Mean   :2.960e+07   Mean   :  7215.3
##  3rd Qu.:1.959e+07   3rd Qu.:  9325.5
##  Max.   :1.319e+09   Max.   :113523.1
##
```

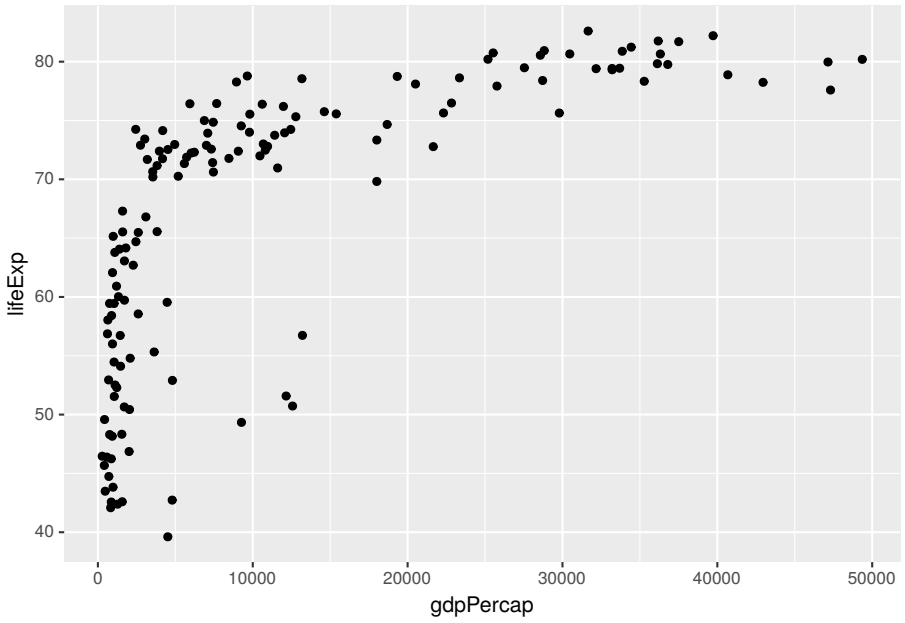
```
mydata$year %>% unique()
```

```
##  [1] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002 2007
```

4.2 Scatter plots/bubble plots - geom_point()

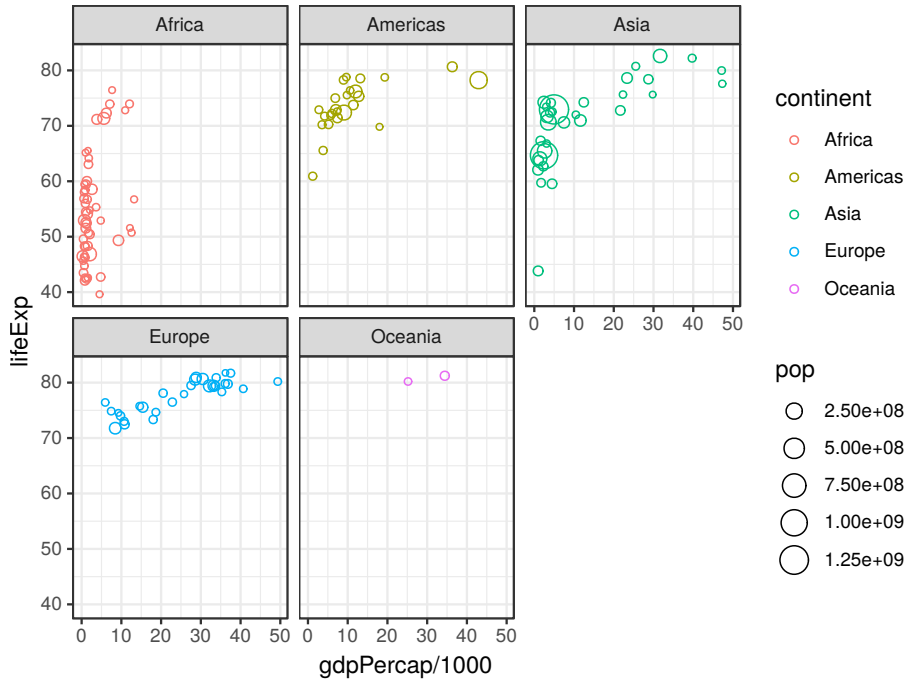
Plot life expectancy against GDP per capita (`x = gdpPercap, y=lifeExp`) at year 2007:

```
mydata %>%
  filter(year == 2007) %>%
  ggplot(aes(x = gdpPercap, y=lifeExp)) +
  geom_point()
```



4.2.1 Exercise

Follow the step-by-step instructions to transform the grey plot just above into this:

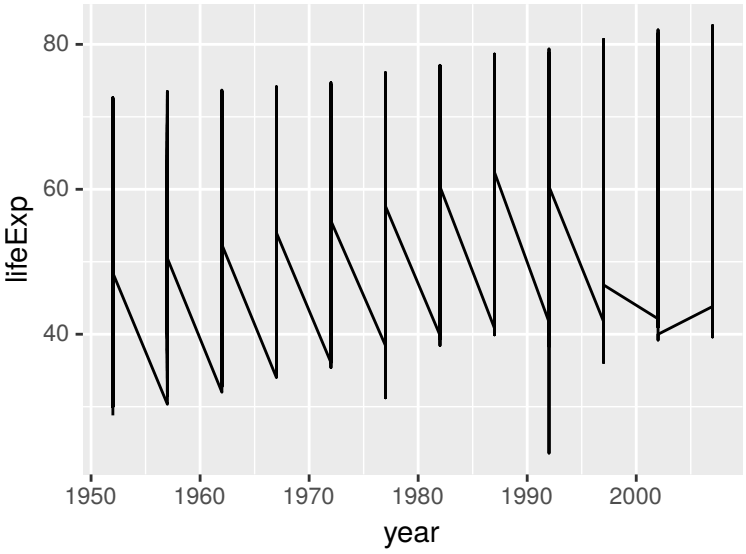


- Add points: `geom_point()`
  - Change point type: `shape = 1` (or any number from your Quickstart Sheet) inside the `geom_point()`
- Colour each country point by its continent: `colour=continent` to aes()
- Size each country point by its population: `size=pop` to aes()
- Put the country points of each continent on a separate panel: `+ facet_wrap(~continent)`
- Make the background white: `+ theme_bw()`

4.3 Line chart/timeplot - `geom_line()`

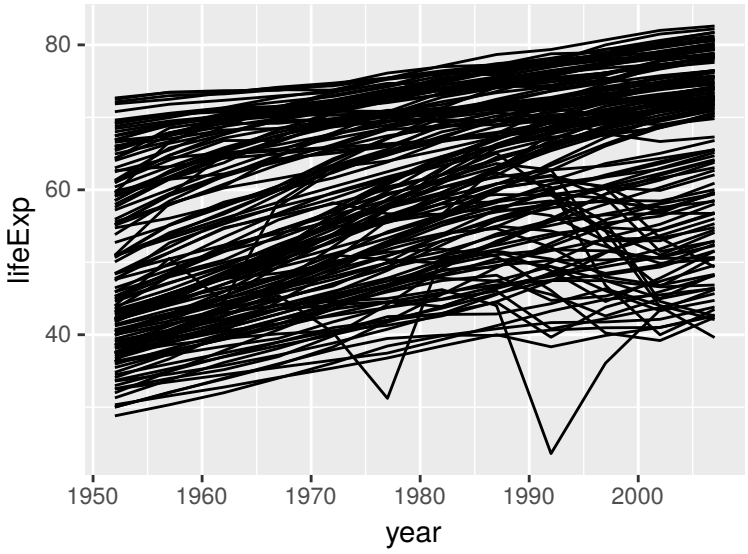
Plot life expectancy against year (`x = year, y=lifeExp`), add `geom_line()`:

```
mydata %>%
  ggplot(aes(x = year, y=lifeExp)) +
  geom_line()
```



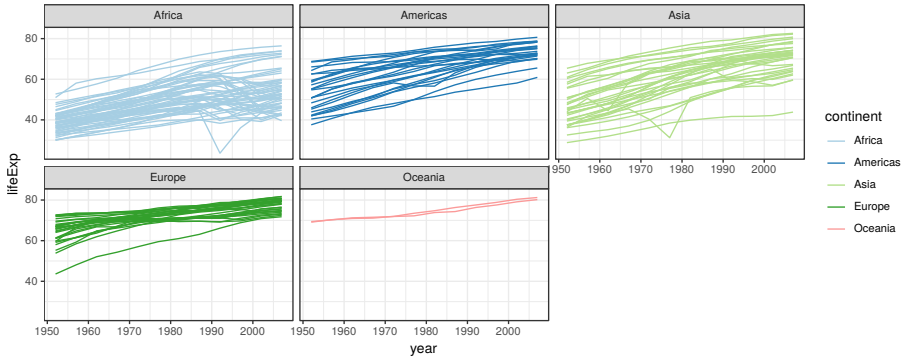
The reason you now see this weird zig-zag is that, using the above code, R does not know you want a connected line for each country. Specify how you want data points grouped to lines: `group = country` in `aes()`:

```
mydata %>%
  ggplot(aes(x = year, y=lifeExp, group = country)) +
  geom_line()
```



4.3.1 Exercise

Follow the step-by-step instructions to transform the grey plot just above into this:



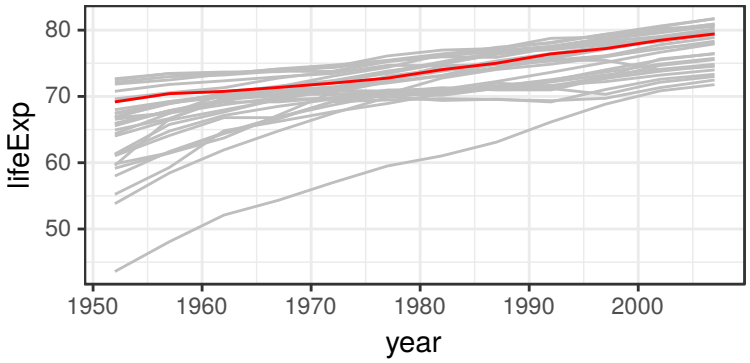
- Colour lines by continents: `colour=continent` to `aes()`
- *Similarly to what we did in `geom_point()`, you can even size the line thicknesses by each country's population: `size=pop` to `aes()`*
- Continents on separate panels: `+ facet_wrap(~continent)`
- Make the background white: `+ theme_bw()`

- Use a nicer colour scheme: `+ scale_colour_brewer(palette = "Paired")`

4.3.2 Advanced example

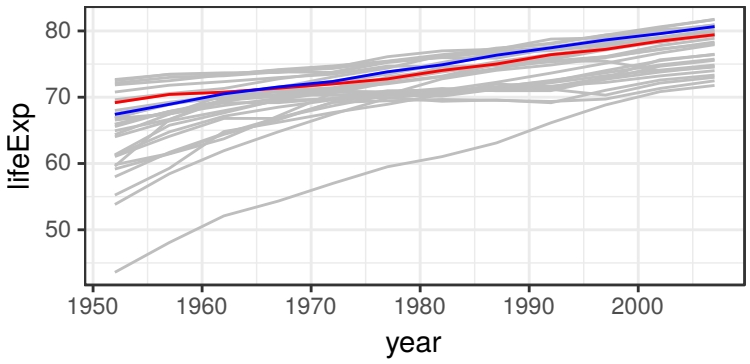
For European countries only (`filter(continent == "Europe") %>%`), plot life expectancy over time in grey colour for all countries, then add United Kingdom as a red line:

```
mydata %>%
  filter(continent == "Europe") %>% #Europe only
  ggplot(aes(x = year, y=lifeExp, group = country)) +
  geom_line(colour = "grey") +
  theme_bw() +
  geom_line(data = filter(mydata, country == "United Kingdom"), colour = "re
```



4.3.3 Advanced Exercise

As previous, but add a line for France in blue:

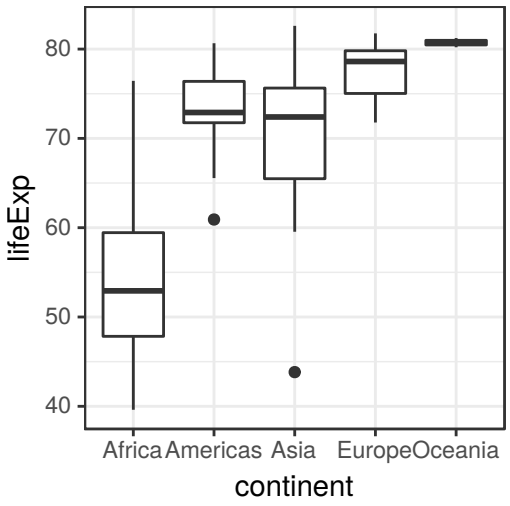


4.4 Box-plot - geom_boxplot()

Plot the distribution of life expectancies within each continent at year 2007:

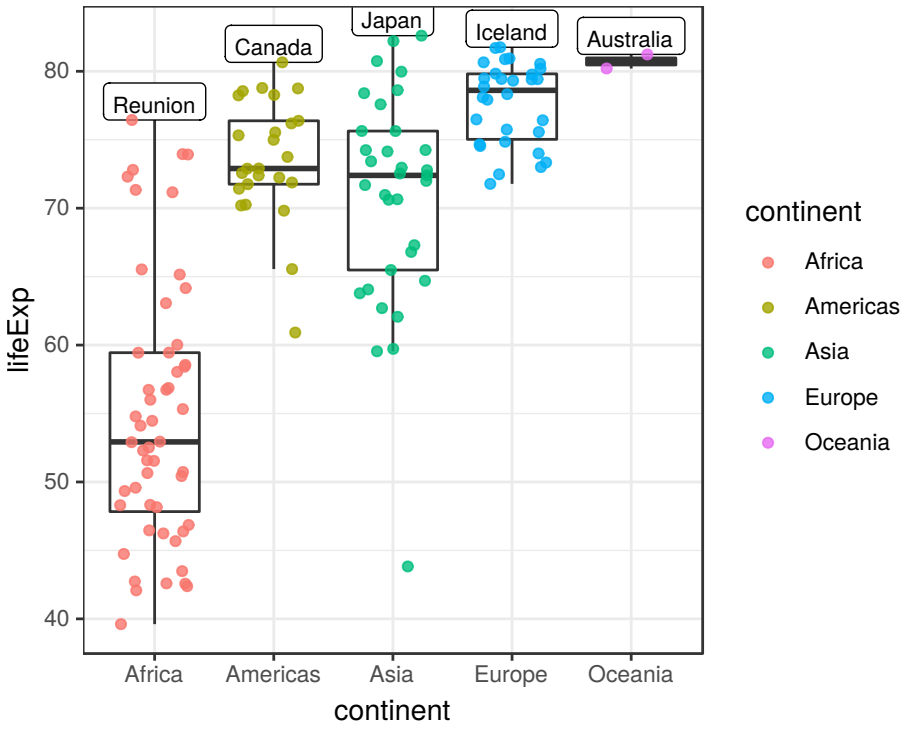
- filter(year == 2007) %>%
- x = continent, y = lifeExp
- + geom_boxplot()

```
mydata %>%
  filter(year == 2007) %>%
  ggplot(aes(x = continent, y = lifeExp)) +
  geom_boxplot() +
  theme_bw()
```



4.4.1 Exercise

Add individual (country) points on top of the box plot:



Hint: Use `geom_jitter()` instead of `geom_point()` to reduce overlap by spreading the points horizontally. Include the `width=0.3` option to reduce the width of the jitter.

Optional:

Include text labels for the highest life expectancy country of each continent.

Hint 1 Create a separate dataframe called `label_data` with the maximum countries for each continent:

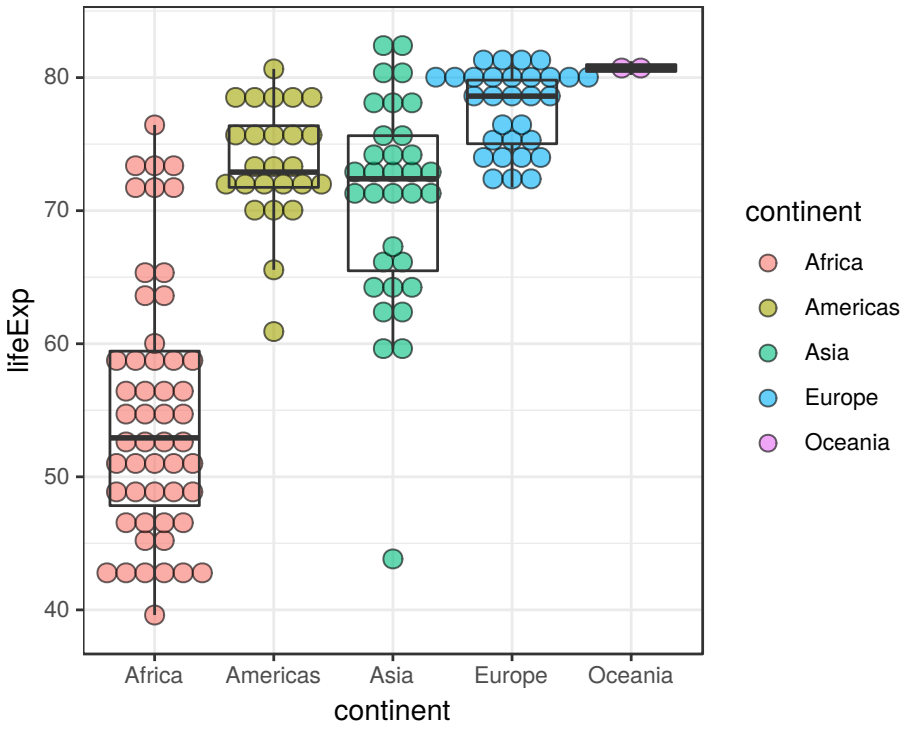
```
label_data = mydata %>%
  filter(year == max(year)) %>% # same as year == 2007
  group_by(continent) %>%
  filter(lifeExp == max(lifeExp) )
```

Hint 2 Add `geom_label()` with appropriate `aes()`:

```
+ geom_label(data = label_data, aes(label=country), vjust = 0)
```

4.4.2 Dot-plot - `geom_dotplot()`

```
geom_dotplot(aes(fill=continent), binaxis = 'y',
stackdir = 'center', alpha=0.6)
```

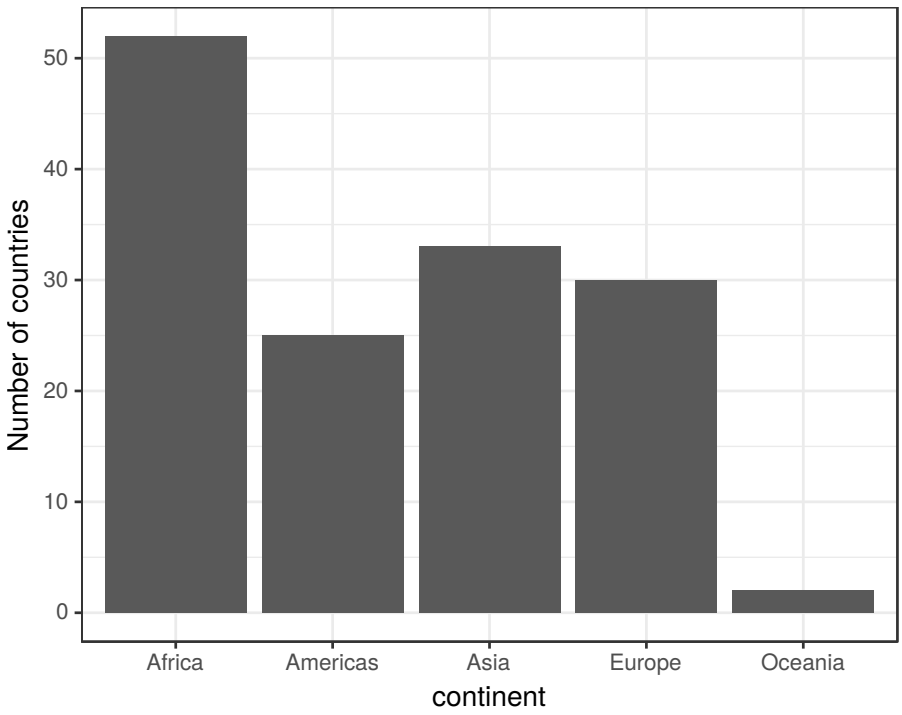


4.5 Barplot - `geom_bar()` and `geom_col()`

In the first module, we plotted barplots from already summarised data (using the `geom_col`), but `geom_bar()` is perfectly happy to count up data for you. For example, we can plot the number of countries in each continent without summarising the data beforehand:

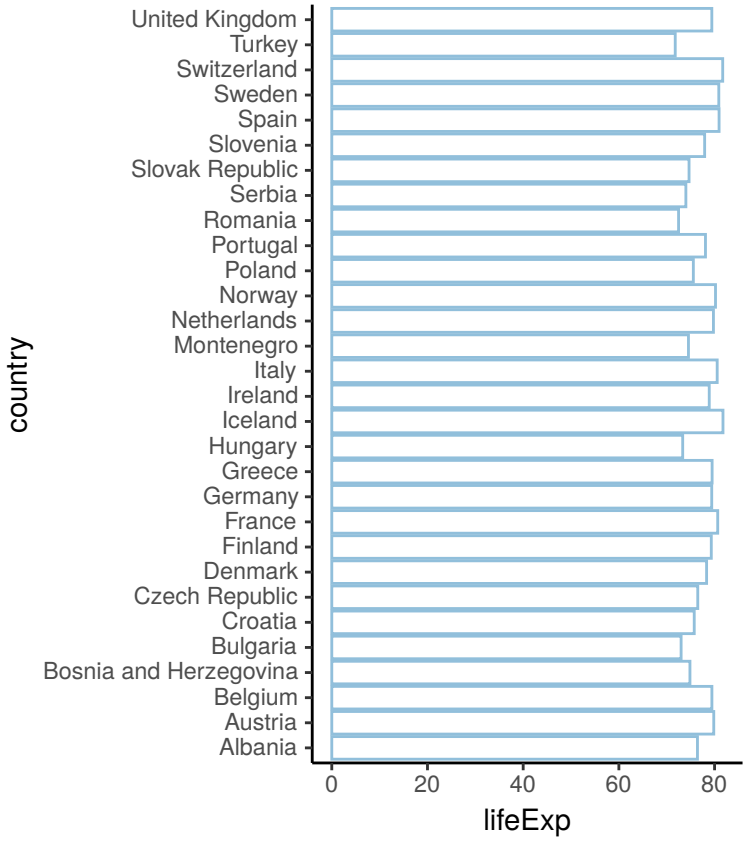
```
mydata %>%
  filter(year == 2007) %>%
  ggplot(aes(x = continent)) +
  geom_bar() +
```

```
  ylab("Number of countries") +
  theme_bw()
```



4.5.1 Exercise

Create this barplot of life expectancies in European countries (year 2007). Hint: `coord_flip()` makes the bars horizontal, `fill = NA` makes them empty, have a look at your QuickStar sheet for different themes.



4.6 All other types of plots

These are just some of the main ones, see this gallery for more options: http://www.r-graph-gallery.com/portfolio/ggplot2-package/

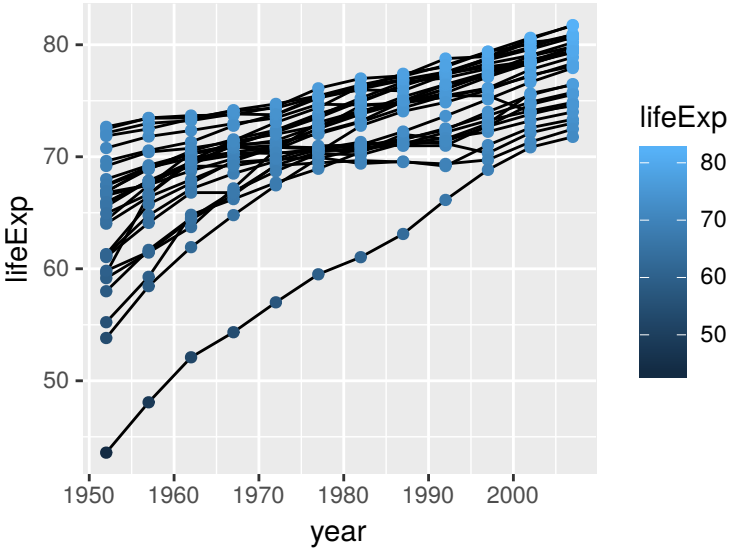
And the `ggplot()` documentation: http://docs.ggplot2.org/

Remember that you can always combine different types of plots - i.e. add lines or points on bars, etc.

4.7 Specifying `aes()` variables

The `aes()` variables wrapped inside `ggplot()` will be taken into account by all geoms. If you put `aes(colour = lifeExp)` inside `geom_point()`, only points will be coloured:

```
mydata %>%
  filter(continent == "Europe") %>%
  ggplot(aes(x = year, y = lifeExp, group = country)) +
  geom_line() +
  geom_point(aes(colour = lifeExp))
```

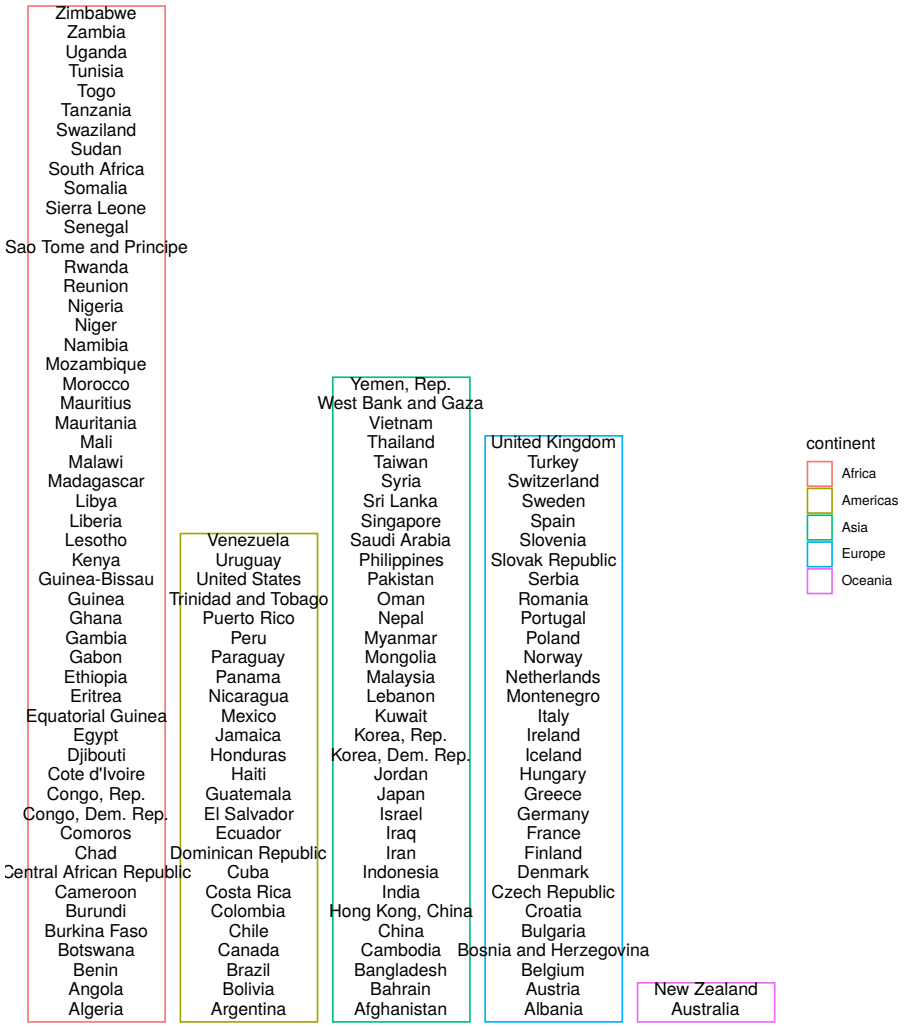


4.8 Extra: Optional exercises

4.8.1 Exercise

Make this:

```
mydata$dummy = 1  # create a column called "dummy" that includes number 1 f

mydata2007 = mydata %>%
  filter(year==max(year)) %>%
  group_by(continent) %>%
  mutate(country_number = cumsum(dummy))  # create a column called "country_
  # is a cumulative sum of the number of countries before it - basically in


mydata2007 %>%
  ggplot(aes(x = continent)) +
  geom_bar(aes(colour=continent), fill = NA) +
  geom_text(aes(y = country_number, label=country), size=4, vjust=1, colour=
  theme_void()
```

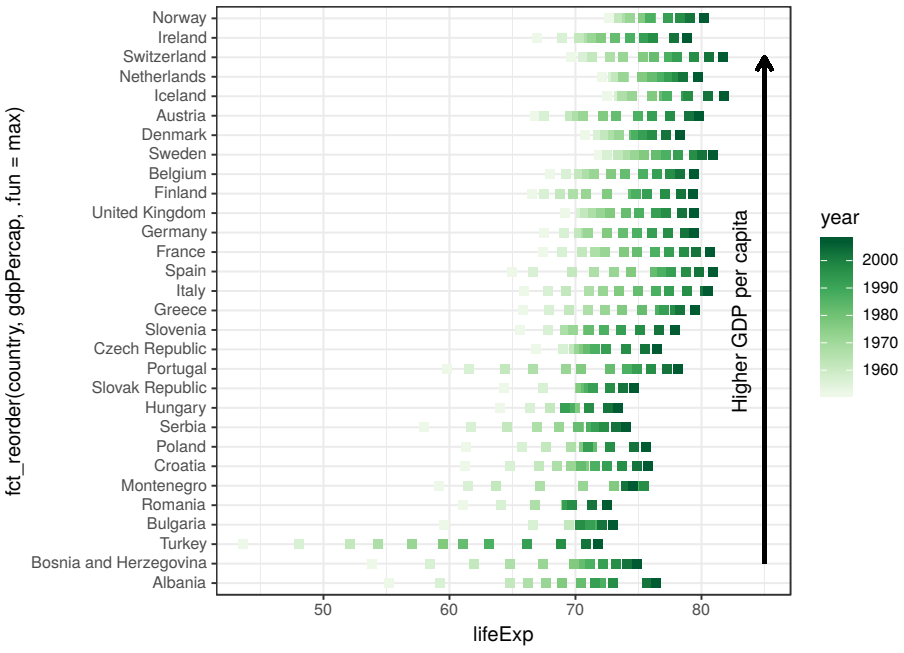

Zimbabwe
Zambia
Uganda
Tunisia
Togo
Tanzania
Swaziland
Sudan
South Africa
Somalia
Sierra Leone
Senegal
Sao Tome and Principe
Rwanda
Reunion
Nigeria
Niger
Namibia
Mozambique
Morocco
Mauritius
Mauritania
Mali
Malawi
Madagascar
Libya
Liberia
Lesotho
Kenya
Guinea-Bissau
Guinea
Ghana
Gambia
Gabon
Ethiopia
Eritrea
Equatorial Guinea
Egypt
Djibouti
Cote d'Ivoire
Congo, Rep.
Congo, Dem. Rep.
Comoros
Chad
Central African Republic
Cameroon
Burundi
Burkina Faso
Botswana
Benin
Angola
Algeria
Venezuela
Uruguay
United States
Trinidad and Tobago
Puerto Rico
Peru
Paraguay
Panama
Nicaragua
Mexico
Jamaica
Honduras
Haiti
Guatemala
El Salvador
Ecuador
Dominican Republic
Cuba
Costa Rica
Colombia
Chile
Canada
Brazil
Bolivia
Argentina
Yemen, Rep.
West Bank and Gaza
Vietnam
Thailand
Taiwan
Syria
Sri Lanka
Singapore
Saudi Arabia
Philippines
Pakistan
Oman
Nepal
Myanmar
Mongolia
Malaysia
Lebanon
Kuwait
Korea, Rep.
Korea, Dem. Rep.
Jordan
Japan
Israel
Iraq
Iran
Indonesia
India
Hong Kong, China
China
Cambodia
Bangladesh
Bahrain
Afghanistan
United Kingdom
Turkey
Switzerland
Sweden
Spain
Slovenia
Slovak Republic
Serbia
Romania
Portugal
Poland
Norway
Netherlands
Montenegro
Italy
Ireland
Iceland
Hungary
Greece
Germany
France
Finland
Denmark
Czech Republic
Croatia
Bulgaria
Bosnia and Herzegovina
Belgium
Austria
Albania
New Zealand
Australia
continent
Africa
Americas
Asia
Europe
Oceania

4.8.2 Exercise

Make this:

Hints: `coord_flip()`, `scale_color_gradient(...)`, `geom_segment(...)`, `annotate("text", ...)`

```
mydata %>%
  filter(continent == "Europe") %>%
  ggplot(aes(y = fct_reorder(country, gdpPercap, .fun=max), x=lifeExp, colou
  geom_point(shape = 15, size = 2) +
  theme_bw() +
  scale_colour_distiller(palette = "Greens", direction = 1) +
  geom_segment(aes(yend = "Switzerland", x = 85, y = "Bosnia and Herzegovina
                   colour = "black", size=1,
                   arrow = arrow(length = unit(0.3, "cm"))) +
  annotate("text", y = "Greece", x=83, label = "Higher GDP per capita", angl
```



4.9 Solutions

4.2.1

```
mydata %>%
  filter(year == 2007) %>%
  ggplot(  aes(x = gdpPercap/1000, #divide by 1000 to tidy the x-axis
               y=lifeExp,
               colour=continent,
               size=pop)) +
  geom_point(shape = 1) +
  facet_wrap(~continent) +
  theme_bw()
```

4.3.1

```
mydata %>%
  ggplot(  aes(x = year, y=lifeExp, group = country, colour=continent)) +
  geom_line() +
  facet_wrap(~continent) +
  theme_bw() +
  scale_colour_brewer(palette = "Paired")
```

which

Add `+ geom_line(data = filter(mydata, country == "France"), colour = "blue")`

4.4.1

```
mydata %>%
  filter(year == 2007) %>%
  ggplot(aes(x = continent, y = lifeExp)) +
  geom_boxplot(outlier.shape = NA) +
```

```
  geom_jitter(aes(colour=continent), width=0.3, alpha=0.8) + #width defaults
  theme_bw()
```

```
mydata %>%
  filter(year == 2007) %>%
  ggplot(aes(x = continent, y = lifeExp)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(aes(colour=continent), width=0.3, alpha=0.8)
  theme_bw()
```

4.5.1

```
mydata %>%
  filter(year == 2007) %>%
  filter(continent == "Europe") %>%
  ggplot(aes(x = country, y = lifeExp)) +
  geom_col(colour = "#91bfdb", fill = NA) +
  coord_flip() +
  theme_classic()
```

5

Fine tuning plots

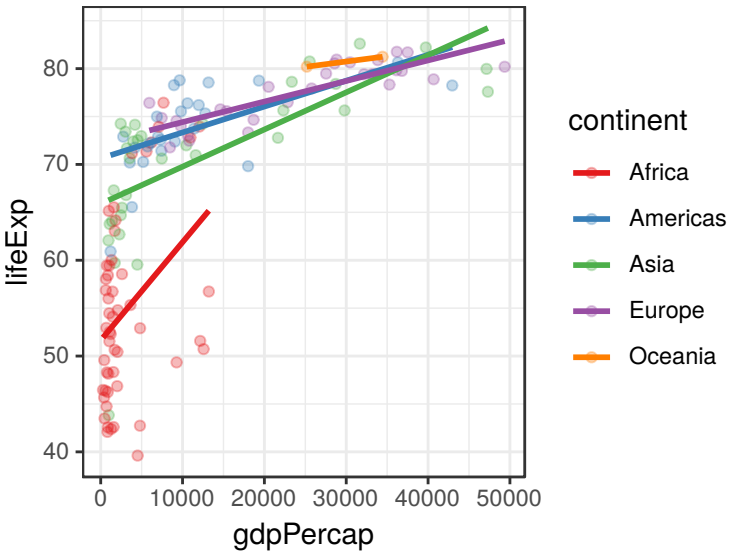
5.1 Data and initial plot

We can save a `ggplot()` object into a variable (usually called `p` but can be any name). This then appears in the Environment tab. To plot it it needs to be recalled on a separate line. Saving a plot into a variable allows us to modify it later (e.g., `p + theme_bw()`).

```
library(gapminder)
library(tidyverse)

mydata = gapminder

mydata$year %>% unique()
```

```
##  [1] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002 2007
```

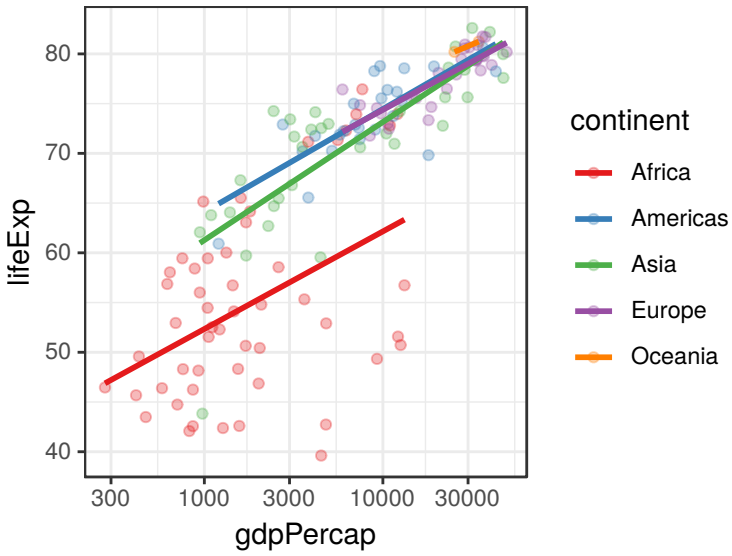
```
p = mydata %>%
  filter(year == 2007) %>%
  group_by(continent, year) %>%
  ggplot(aes(y = lifeExp, x = gdpPercap, colour = continent)) +
  geom_point(alpha = 0.3) +
  theme_bw() +
  geom_smooth(method = "lm", se = FALSE) +
  scale_colour_brewer(palette = "Set1")

p
```



5.2 Scales

5.2.1 Logarithmic

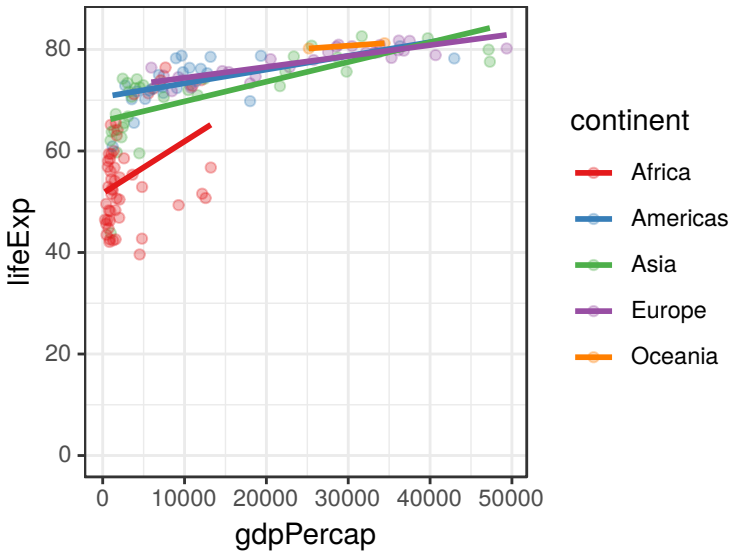
```
p +
  scale_x_log10()
```



5.2.2 Expand limits

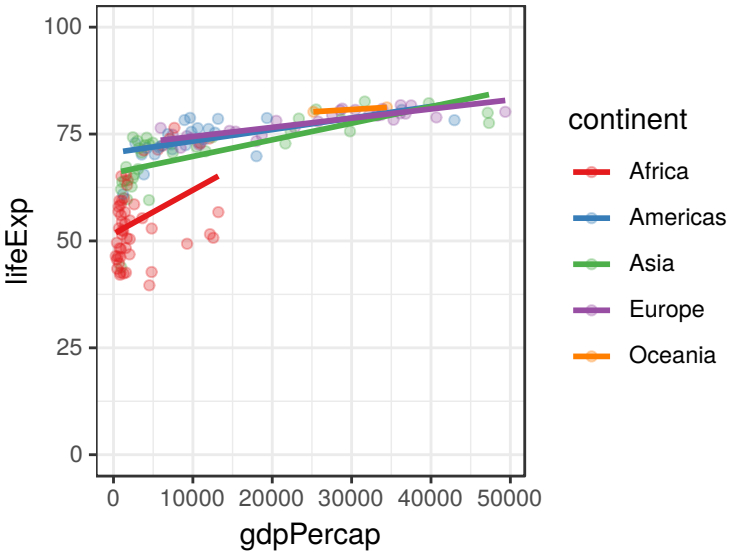
Specify the value you want to be included:

```
p +
  expand_limits(y = 0)
```



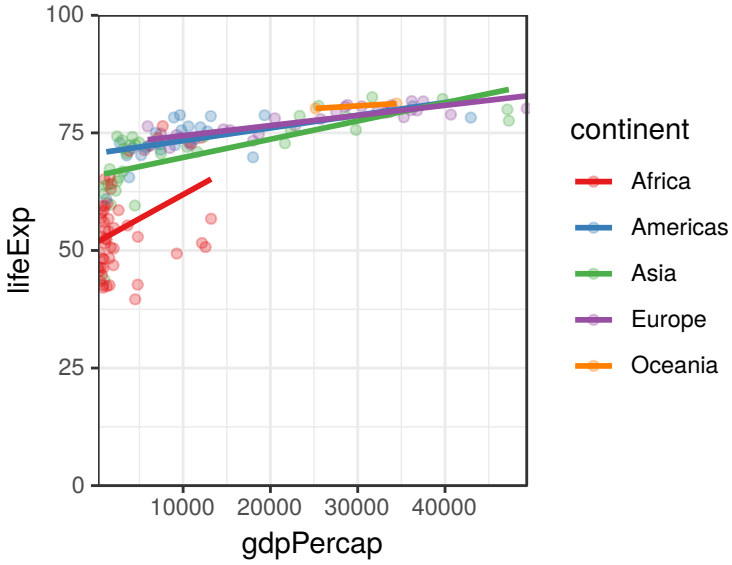
Or two:

```
p +
  expand_limits(y = c(0, 100))
```



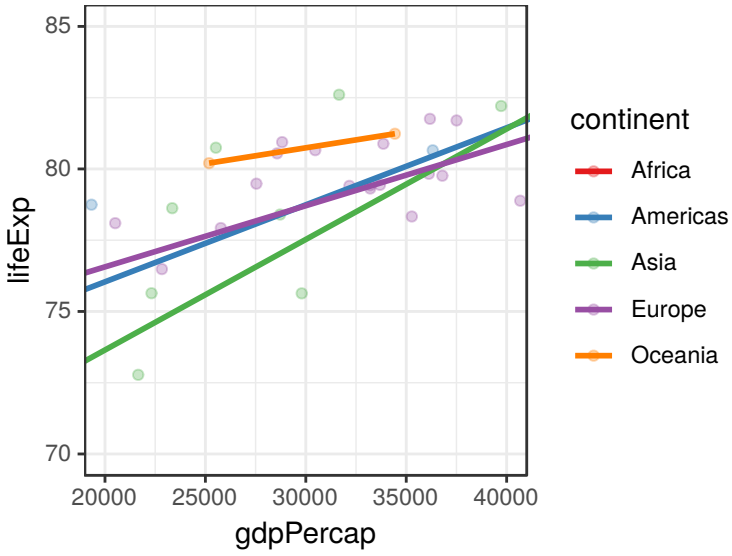
By default, `ggplot()` adds some padding around the included area (see how the scale doesn't start from 0, but slightly before). You can remove this padding with the expand option:

```
p +
  expand_limits(y = c(0, 100)) +
  coord_cartesian(expand = FALSE)
```



5.2.3 Zoom in

```
p +
  coord_cartesian(ylim = c(70, 85), xlim = c(20000, 40000))
```



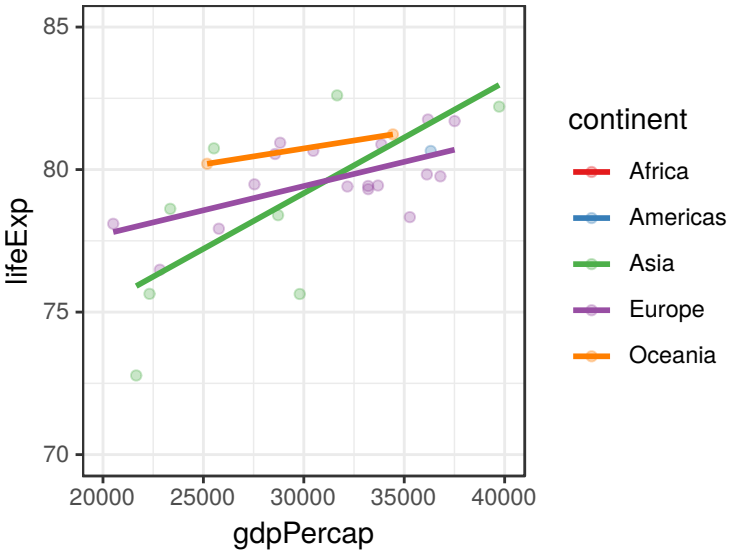
5.2.4 Exercise

How is this one different to the previous?

```
p +
  scale_y_continuous(limits = c(70, 85)) +
  scale_x_continuous(limits = c(20000, 40000))
```

```
## Warning: Removed 114 rows containing non-finite values (stat_smooth).
```

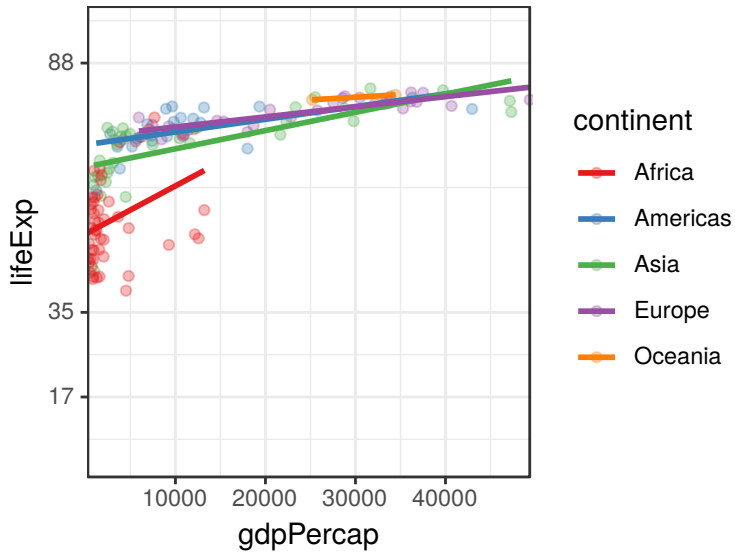
```
## Warning: Removed 114 rows containing missing values (geom_point).
```



Answer: the first one zooms in, still retaining information about the excluded points when calculating the linear regression lines. The second one removes the data (as the warnings say), calculating the linear regression lines only for the visible points.

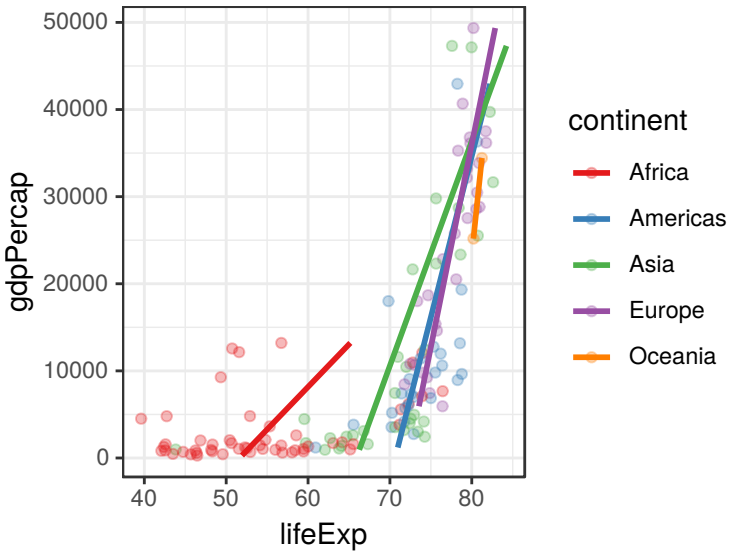
5.2.5 Axis ticks

```
p +
  coord_cartesian(ylim = c(0, 100), expand = 0) +
  scale_y_continuous(breaks = c(17, 35, 88))
```



5.2.6 Swap the axes

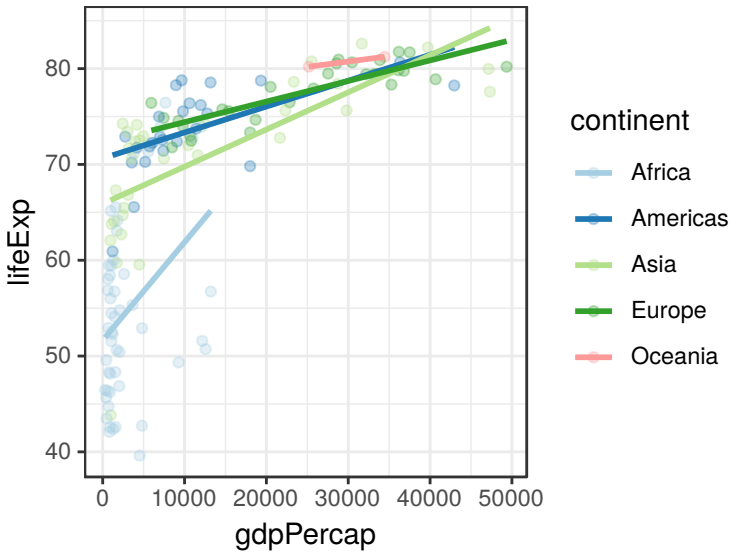
```
p +
  coord_flip()
```



5.3 Colours

5.3.1 Using the Brewer palettes:

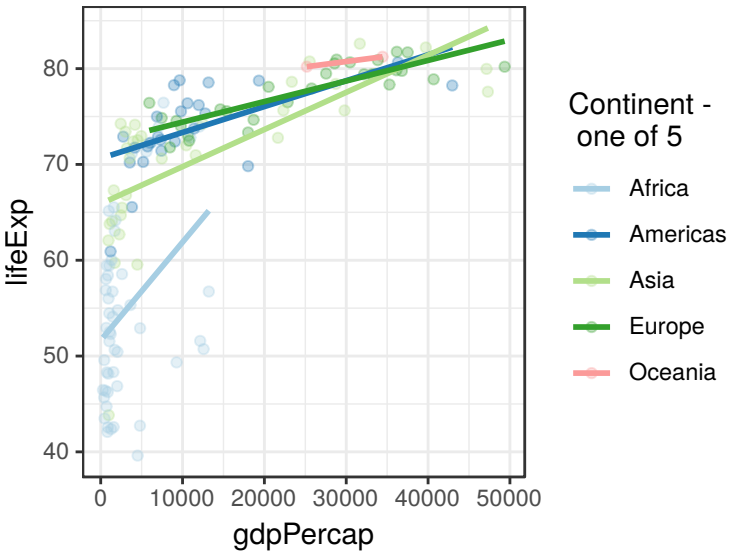
```
p +
  scale_color_brewer(palette = "Paired")
```



5.3.2 Legend title

`scale_colour_brewer()` is also a conventient place to change the legend title:

```
p +
  scale_color_brewer("Continent - \n one of 5", palette = "Paired")
```

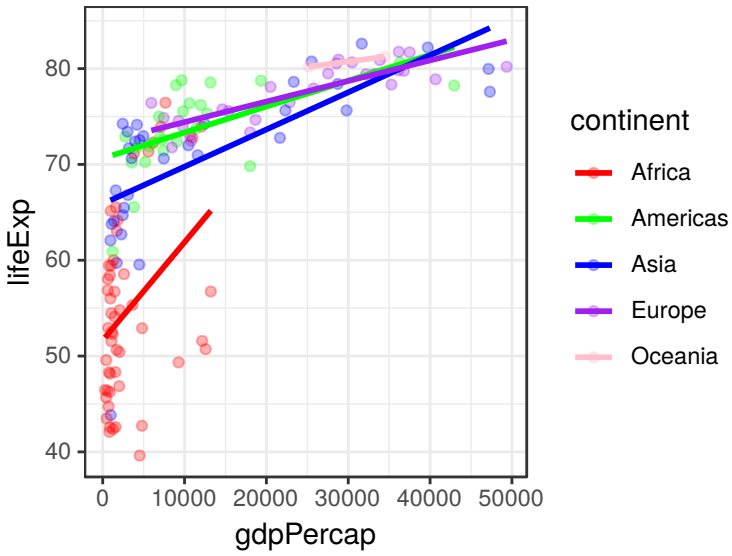


Note the \n inside the new legend title - new line.

5.3.3 Choosing colours manually

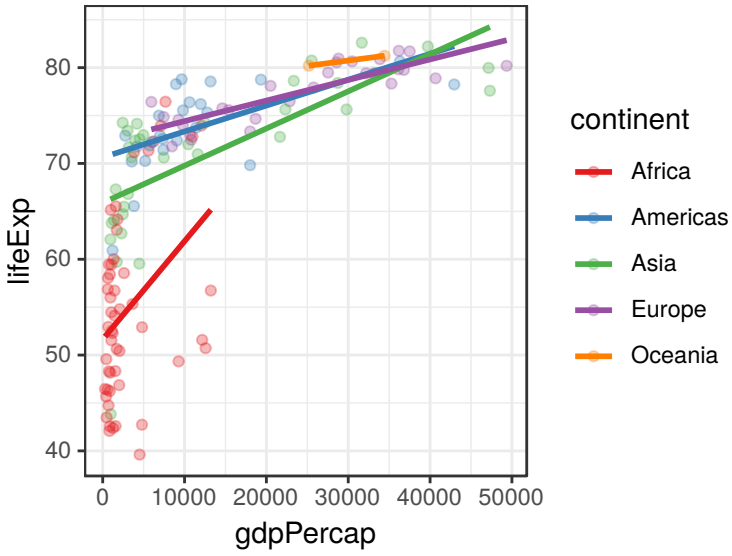
Use words:

```
p +
  scale_color_manual(values = c("red", "green", "blue", "purple", "pink"))
```



Or HEX codes (either from http://colorbrewer2.org/ or any other resource):

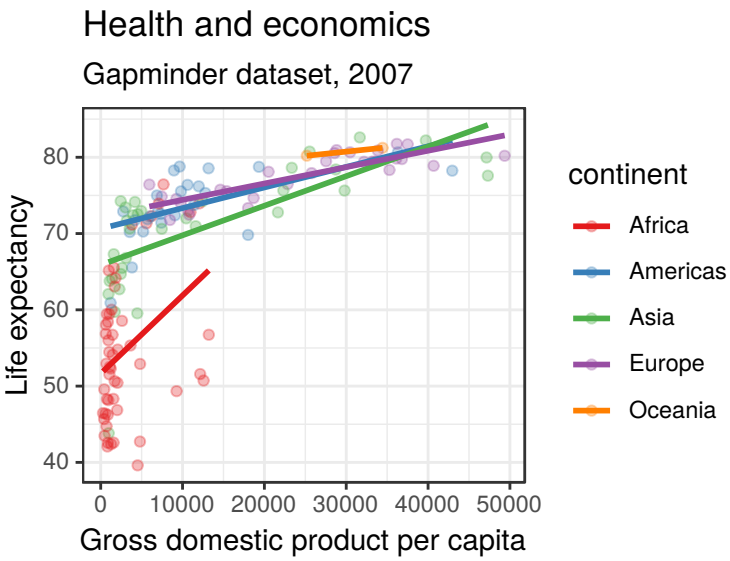
```
p +
  scale_color_manual(values = c("#e41a1c", "#377eb8", "#4daf4a", "#984ea3",
```



Note that http://colorbrewer2.org/ also has options for *Colourblind safe* and *Print friendly*.

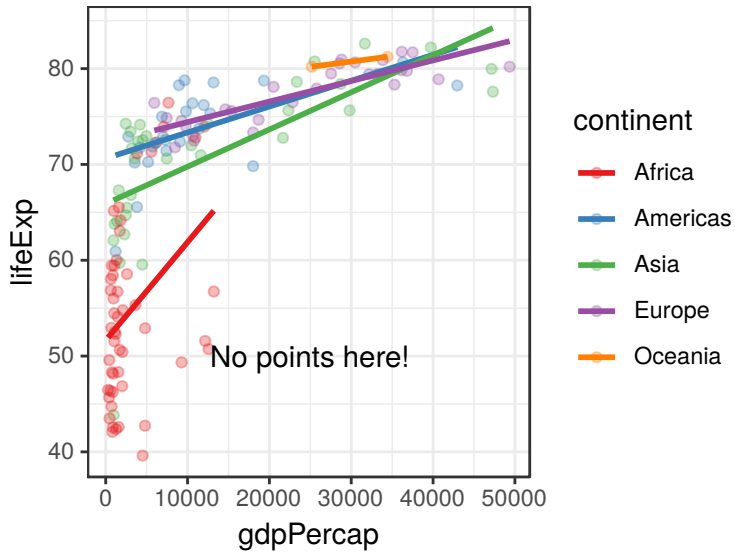
5.4 Titles and labels

```
p +
  labs(x = "Gross domestic product per capita",
       y = "Life expectancy",
       title = "Health and economics",
       subtitle = "Gapminder dataset, 2007")
```

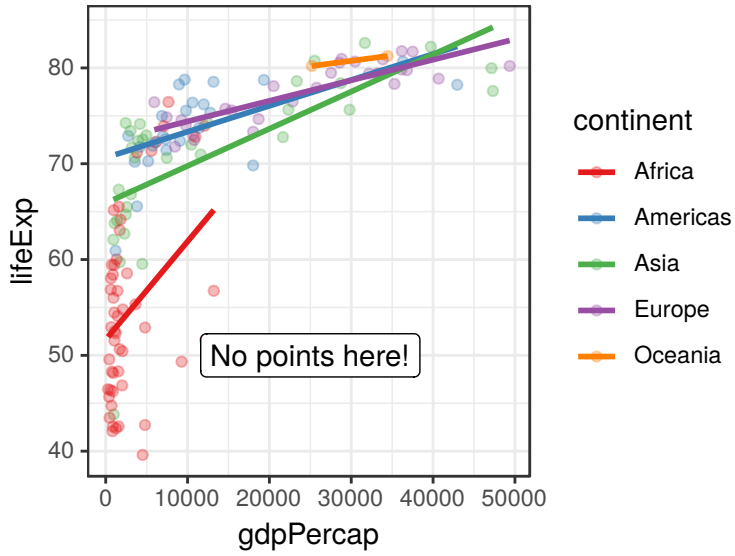


5.4.1 Annotation

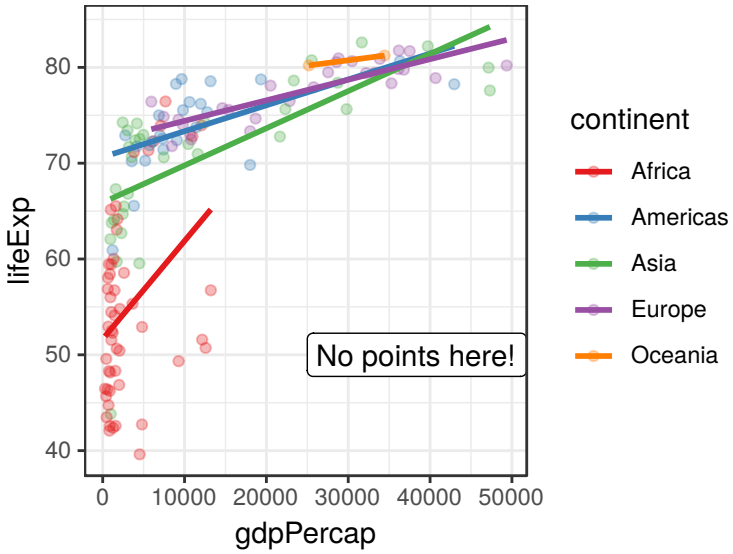
```
p +
  annotate("text",
           x = 25000,
           y = 50,
           label = "No points here!")
```




```
p +
  annotate("label",
           x = 25000,
           y = 50,
           label = "No points here!")
```



```
p +
  annotate("label",
           x = 25000,
           y = 50,
           label = "No points here!",
           hjust = 0)
```

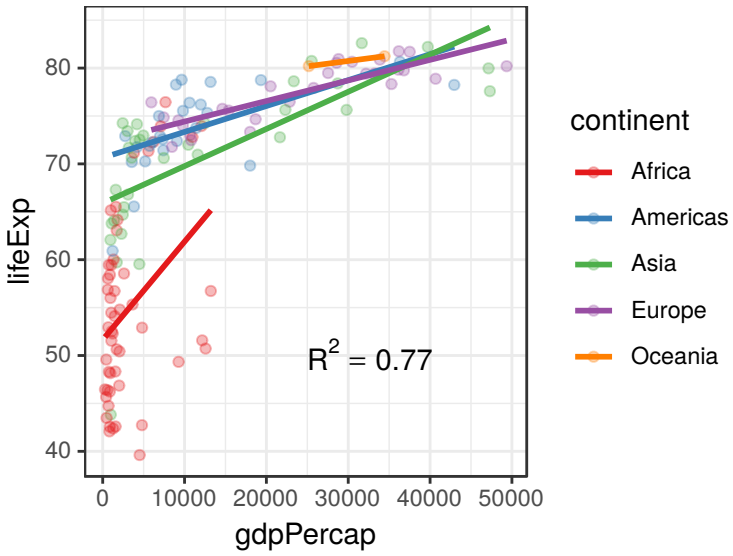


`hjust` stand for horizontal justification. It's default value is 0.5 (see how the label was centered at 25,000 - our chosen x location), 0 means the label goes to the right from 25,000, 1 would make it end at 25,000.

5.4.2 Annotation with a superscript and a variable

```
fit_glance = data.frame(r.squared = 0.7693465)


plot_rsquared = paste0(
  "R^2 == ",
```

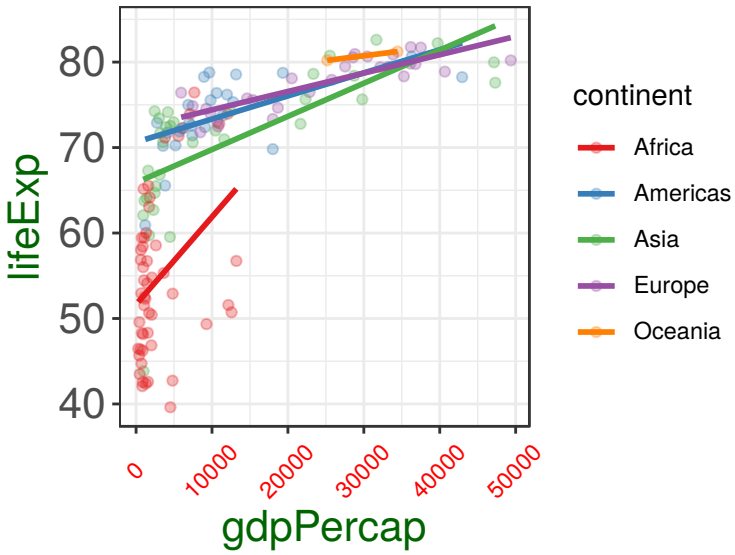
```
  fit_glance$r.squared %>% round(2))


p +
  annotate("text",
           x = 25000,
           y = 50,
           label = plot_rsquared, parse = TRUE,
           hjust = 0)
```



5.5 Text size

```
p +
  theme(axis.text.y = element_text(size = 16),
        axis.text.x = element_text(colour = "red", angle = 45, vjust = 0.5),
```

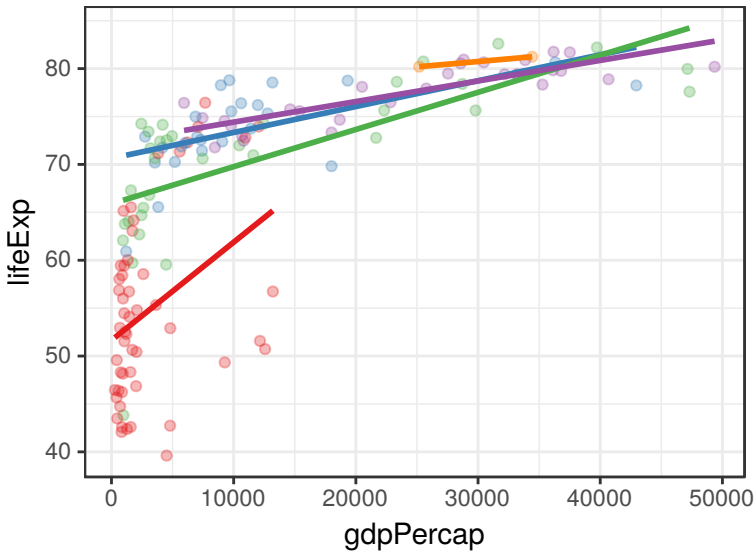
```
        axis.title = element_text(size = 16, colour = "darkgreen")
        )
```



5.5.1 Legend position

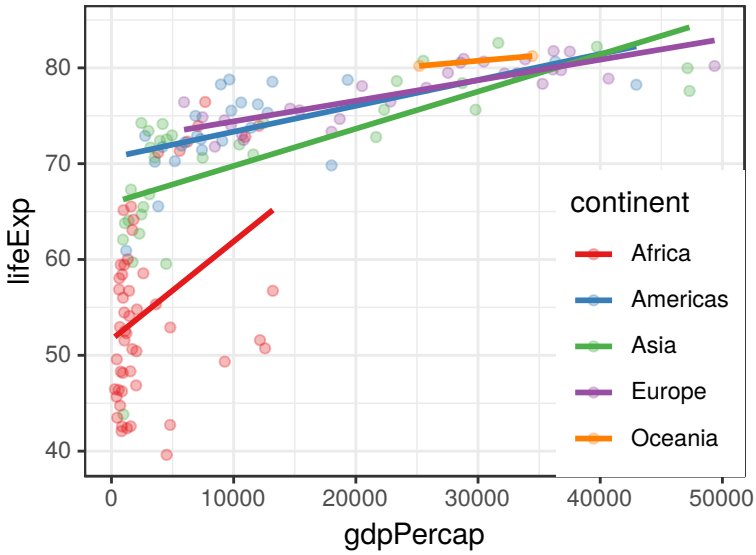
Use the following words: `"right"`, `"left"`, `"top"`, `"bottom"`, or `"none"` to remove the legend.

```
p +
  theme(legend.position = "none")
```

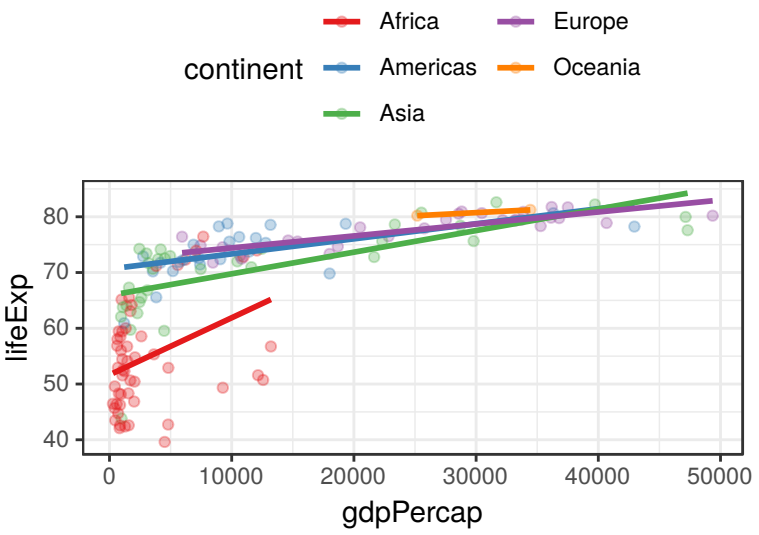


Or use relative coordinates (0–1) to give it an -y location:

```
p +
  theme(legend.position       = c(1,0),
        legend.justification = c(1,0)) #bottom-right corner
```



```
p +
  theme(legend.position = "top") +
  guides(colour = guide_legend(ncol = 2))
```



5.6 Saving your plot

```
ggsave(p, file = "my_saved_plot.png", width = 5, height = 4)
```

Part II

Data analysis

6

Tests for continuous outcome variables

6.1 Load data

This session we will be using the gapminder dataset as in Session 4.

```
library(tidyverse)
library(gapminder)
library(broom)

mydata = gapminder
```

Consider adding `ff_glimpse()`

The first step of choosing the right statistical test is determining the type of variable you have.

Lets first have a look at some of our available data:

```
mydata$continent %>% unique() # categorical
```

```
## [1] Asia     Europe   Africa   Americas Oceania
## Levels: Africa Americas Asia Europe Oceania
```

```
mydata$year %>% unique() # categorical
```

```
##  [1] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002 2007
```

```
mydata$lifeExp %>% head() # continuous
```

```
## [1] 28.801 30.332 31.997 34.020 36.088 38.438
```

6.2 T-test

A *t*-test is used to compare the means of two groups of continuous variables.

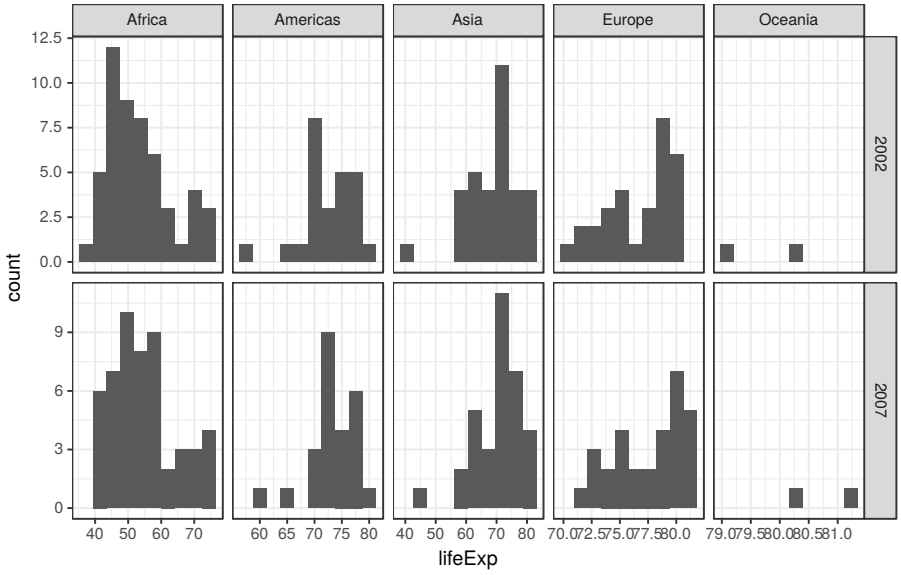
6.2.1 Plotting

Before you perform any statistical tests, you should always plot your data first to determine whether these have a "normal" distribution.

- Histograms should form a symmetrical "bell-shaped curve".
- Q-Q plots should fall along the 45 degree line.
- Box-plots should be symmetrical and have few outliers.

6.2.2 Histogram for each continent

```
theme_set(theme_bw())

mydata %>%
    filter(year %in% c(2002, 2007)) %>%
    ggplot(aes(x = lifeExp)) +
    geom_histogram(bins = 10) +
    facet_grid(year~continent, scales = "free")
```

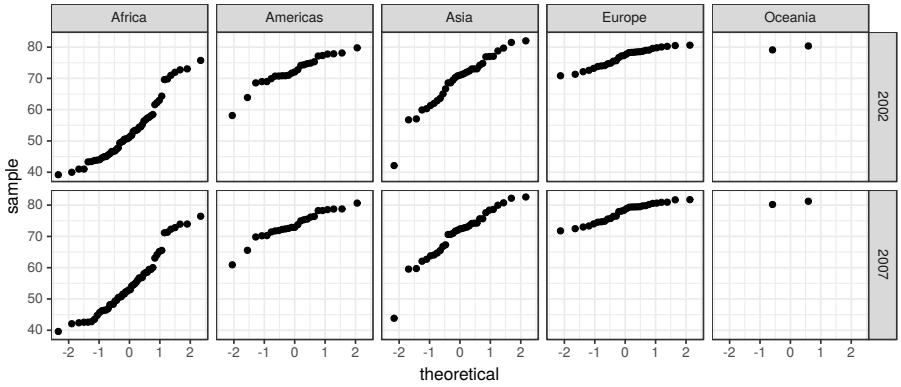


6.2.3 Q-Q plot for each continent

add what a q-q plot is

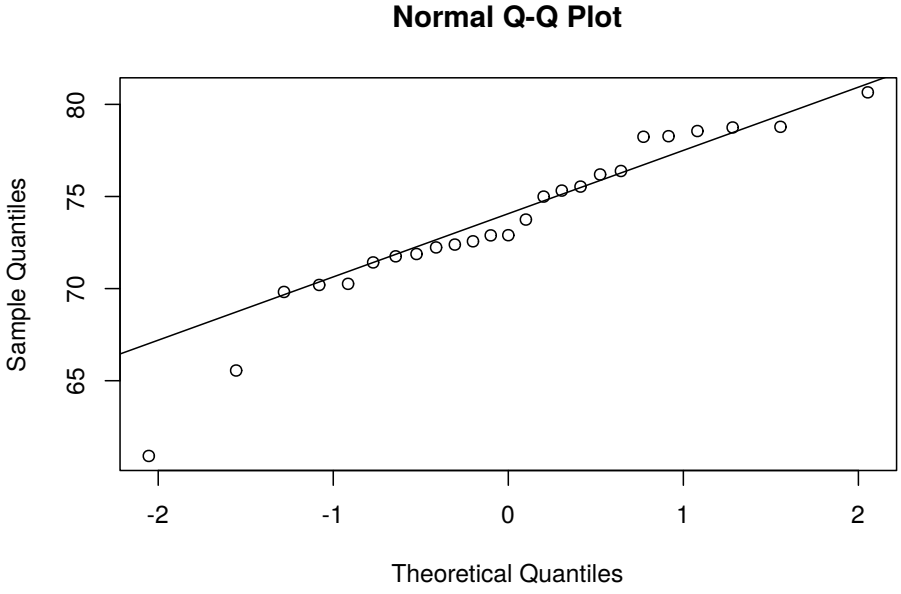
With `ggplot()`, we can draw a Q-Q plot for each subgroup very efficiently:

```
mydata %>%
  filter(year %in% c(2002, 2007)) %>%
  ggplot(aes(sample = lifeExp)) +
  geom_point(stat = "qq") +
  facet_grid(year~continent)
```



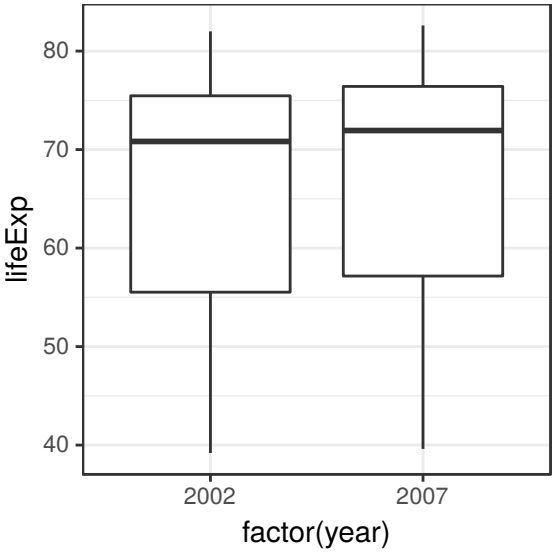
Or we could save a subset of the data (e.g., "Americas" and year 2007 only) into a new variable (`subdata`) and use base R to draw a single Q-Q plot with less code:

```
mydata %>%
  filter(year == 2007) %>%
  filter(continent == "Americas") -> subdata

qqnorm(subdata$lifeExp)
qqline(subdata$lifeExp)
```



6.2.4 Boxplot of 2 years

```
mydata %>%
    filter(year %in% c(2002, 2007)) %>%
    ggplot(aes(x = factor(year), y=lifeExp)) +  # show that x = year errors
    geom_boxplot()                              # needs to be factor(year)
```



6.2.5 Exercise

Make a histogram, Q-Q plot, and a box-plot for the life expectancy for a continent of your choice, but for all years. Does the data appear normally distributed?

6.3 Two-sample *t*-tests

Lets perform a *t*-test on the "Americas" data as it appears normally distributed. We are savings the results of our *t*-test into a

variable called t.result, but you can call it whatever you like (e.g. `myttest`).

```
mydata %>%
  filter(year %in% c(2002, 2007)) %>%
  filter(continent == "Americas") -> test.data

t.test(lifeExp~year, data=test.data)
```

```
##
##  Welch Two Sample t-test
##
## data:  lifeExp by year
## t = -0.90692, df = 47.713, p-value = 0.369
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.816017  1.443857
## sample estimates:
## mean in group 2002 mean in group 2007
##           72.42204           73.60812
```

```
mydata %>%
  filter(year %in% c(2002, 2007)) %>%
  filter(continent == "Americas") %>%
  t.test(lifeExp~year, data = .) -> t.result

t.result
```

```
##
##  Welch Two Sample t-test
##
## data:  lifeExp by year
## t = -0.90692, df = 47.713, p-value = 0.369
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.816017  1.443857
```

```
## sample estimates:
## mean in group 2002 mean in group 2007
##           72.42204           73.60812
```

6.3.1 T-test output

However, that output isn't in a useful format, let's investigate the output of the function `t.test()`.

```
names(t.result)
```

```
## [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
## [6] "null.value"  "alternative" "method"      "data.name"
```

```
str(t.result) # or click on the blue button in the Environment tab
```

```
## List of 9
##  $ statistic  : Named num -0.907
##   ..- attr(*, "names")= chr "t"
##  $ parameter  : Named num 47.7
##   ..- attr(*, "names")= chr "df"
##  $ p.value    : num 0.369
##  $ conf.int   : atomic [1:2] -3.82 1.44
##   ..- attr(*, "conf.level")= num 0.95
##  $ estimate   : Named num [1:2] 72.4 73.6
##  ..- attr(*, "names")= chr [1:2] "mean in group 2002" "mean in group 2007"
##  $ null.value : Named num 0
##   ..- attr(*, "names")= chr "difference in means"
##  $ alternative: chr "two.sided"
##  $ method     : chr "Welch Two Sample t-test"
##  $ data.name  : chr "lifeExp by year"
##  - attr(*, "class")= chr "htest"
```

The structure of R's `t.test()` result looks a bit overwhelming. Fortunately, the `tidy()` function from `library(broom)` puts it into a neat data frame for us:

```
t.result <- tidy(t.result) # broom package puts it all in a data frame
```

Try clicking on it in the Environment tab.

Thus, now we understand the output structure we can extract any result.

```
t.result$p.value
```

```
## [1] 0.3690064
```

6.3.2 Exercise

1. Select any 2 years in any continent and perform a *t*-test to determine whether the life expectancy is signficantly different. Remember to plot your data first.
2. Extract only the p-value from your `t.test()` output.

6.4 One sample *t*-tests

However, we don't always want to compare 2 groups or sometimes we don't have the data to be able to.

Let's investigate whether the mean life expectancy in each continent significant different to 77 years in 2007.

```
mydata %>%
  filter(year==2007, continent=='Europe') -> subdata

# Standard one-sample t-test
t.test(subdata$lifeExp, mu=77)
```

```
##
##  One Sample t-test
##
## data:  subdata$lifeExp
## t = 1.1922, df = 29, p-value = 0.2428
## alternative hypothesis: true mean is not equal to 77
## 95 percent confidence interval:
##  76.53592 78.76128
## sample estimates:
## mean of x
##   77.6486
```

6.4.1 Exercise

1. Select a different year, different continent, and different age to compare with mean life expectancy.
2. Replace mu=77 with mu=0 (the default value). How does this affect your result?

6.5 ANOVA

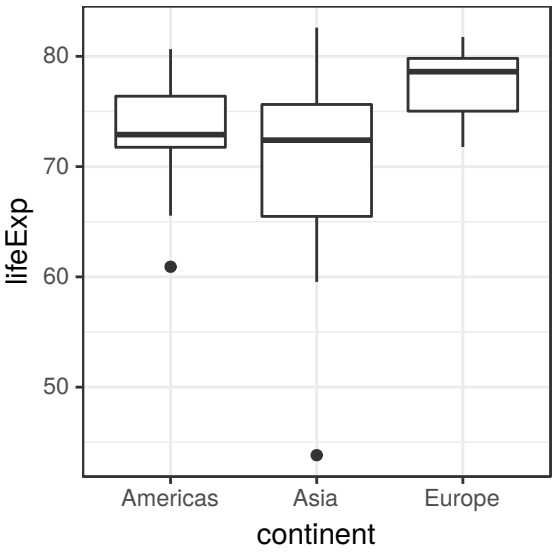
In some cases, we may also want to test more than two groups to see if they are signficantly different.

6.5.1 Plotting

For example, lets plot the life expectancy in 2007 accross 3 continents.

```
mydata %>%
    filter(year == 2007) %>%
    filter(continent %in% c("Americas", "Europe", "Asia")) %>%
```

```
    ggplot(aes(x = continent, y=lifeExp)) +
    geom_boxplot()
```



6.5.2 Analysis

ANOVA tests are useful for testing for the presence of signficant differences between more than two groups or variables.

```
mydata %>%
  filter(year == 2007) %>%
  filter(continent %in% c("Americas", "Europe", "Asia")) -> subdata

fit = aov(lifeExp~continent, data = subdata)

summary(fit)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## continent    2  755.6   377.8   11.63 3.42e-05 ***
## Residuals   85 2760.3    32.5
## ---
```

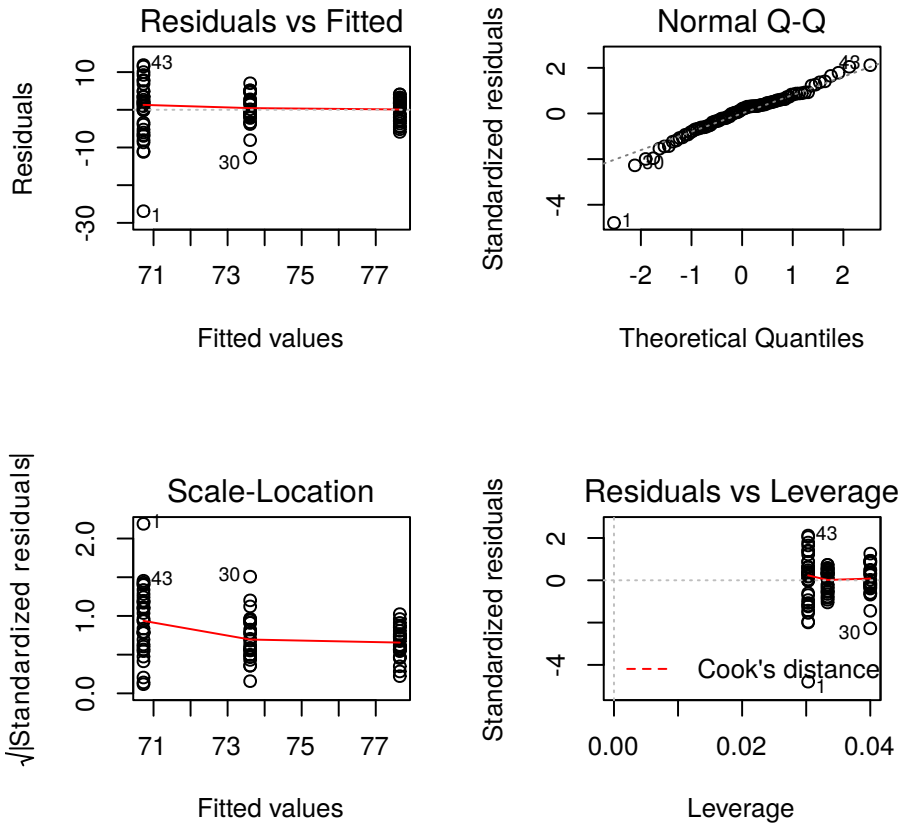
```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
mydata %>%
  filter(year == 2007) %>%
  filter(continent %in% c("Americas", "Europe", "Asia")) %>%
  aov(lifeExp~continent, data = .) %>%
    tidy()
```

```
## # A tibble: 2 x 6
##   term         df sumsq meansq statistic    p.value
##   <chr>     <dbl> <dbl>  <dbl>     <dbl>      <dbl>
## 1 continent     2  756.  378.       11.6  0.0000342
## 2 Residuals    85 2760.   32.5      NA   NA
```

6.5.3 Check assumptions

```
par(mfrow=c(2, 2)) # 4 plots in 2 x 2 grid
plot(fit)
```



6.5.4 Perform pairwise tests

The ANOVA test was significant, indicating that there is a signficant difference in the mean life expectancy across those continents.

But which continents are significantly different, and can we quantify this difference as a p-value?

```
mydata %>%
  filter(year == 2007) %>%
  filter(continent %in% c("Americas", "Europe", "Asia")) -> subdata

pairwise.t.test(subdata$lifeExp, subdata$continent)
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  subdata$lifeExp and subdata$continent
##
##        Americas Asia
## Asia   0.060    -
## Europe 0.021    1.9e-05
##
## P value adjustment method: holm
```

```
# or equivalently, without saving the subset in a separate variable:
# sending it into the test using pipes only

mydata %>%
  filter(year == 2007) %>%
  filter(continent %in% c("Americas", "Europe", "Asia")) %>%
  pairwise.t.test(.$lifeExp, .$continent, data=.) %>%
  tidy()
```

```
## # A tibble: 3 x 3
##   group1 group2     p.value
##   <chr>  <chr>        <dbl>
## 1 Asia   Americas 0.0601
## 2 Europe Americas 0.0209
## 3 Europe Asia     0.0000191
```

F1 for help to see options for `pairwise.t.test()`.

6.5.5 Top tip: the cut() function

A great way of easily converting a continuous variable to a categorical variable is to use the `cut()` function.

```
pop_quantiles = quantile(mydata$pop)
```

```
mydata %>%
    mutate(pop.factor = cut(pop, breaks=pop_quantiles)) -> mydata
```

6.5.6 Exercise

When we used `cut()` to divide country populations into quantiles, the labels it assigned were not very neat:

```
mydata$pop.factor %>% levels()
```

```
## [1] "(6e+04,2.79e+06]"   "(2.79e+06,7.02e+06]" "(7.02e+06,1.96e+07]"
## [4] "(1.96e+07,1.32e+09]"
```

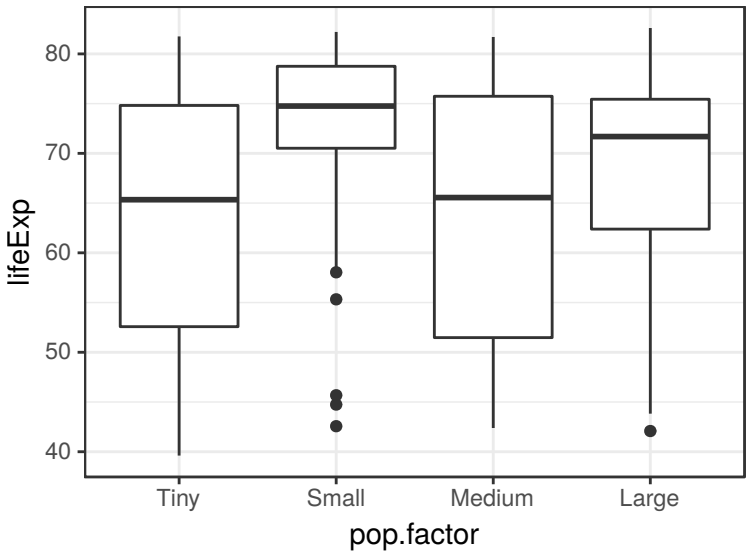
Use `fct_recode()` to change them to something nicer, e.g., "Tiny", "Small", "Medium", "Large":

```
mydata$pop.factor %>%
  fct_recode("Tiny"   = "(6e+04,2.79e+06]",
             "Small"  = "(2.79e+06,7.02e+06]",
             "Medium" = "(7.02e+06,1.96e+07]",
             "Large"  = "(1.96e+07,1.32e+09]") -> mydata$pop.factor
```

6.5.7 Exercise

Perform ANOVA to test for a difference in mean life expectancy by country population factor (`mydata$pop.factor`). Remember to plot data first

```
mydata %>%
    filter(year == 2007) %>%
    ggplot(aes(x=pop.factor, y=lifeExp))+
    geom_boxplot()
```




```
mydata %>%
    filter(year == 2007) %>%
    aov(.$lifeExp ~ .$pop.factor, data=.) %>%
    summary()
```

```
##                Df Sum Sq Mean Sq F value Pr(>F)
## .$pop.factor    3   1160   386.6   2.751 0.0451 *
## Residuals     138  19392   140.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

6.6 Non-parametric data

If your data is not parametric (i.e. not normally distributed), then the usual t-test is invalid. In this case there are 2 options:

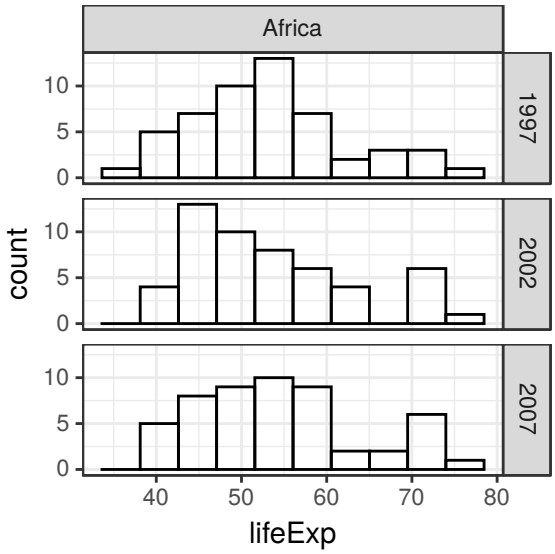
1. Non-parametric statistical tests.

2. "Transform" the data to fit a normal distribution (*not covered here*) so that a t-test can be used.

6.6.1 Plotting

Lets plot the life expectancy within Africa in 1997, 2002, and 2007.

```
# African data is not normally distributed
mydata %>%
  filter(year %in% c(1997, 2002, 2007)) %>%
  filter(continent == "Africa") %>%
  ggplot(aes(x = lifeExp)) +
  geom_histogram(bins = 10, fill=NA, colour='black') +
  facet_grid(year~continent)
```




```
mydata %>%
  filter(year %in% c(1997, 2002, 2007)) %>%
  filter(continent == "Africa") %>%
  group_by(year) %>%
  summarise(avg = mean(lifeExp), med = median(lifeExp))
```

```
## # A tibble: 3 x 3
##    year   avg   med
##   <int> <dbl> <dbl>
## 1  1997  53.6  52.8
## 2  2002  53.3  51.2
## 3  2007  54.8  52.9
```

6.6.2 Exercise: Non-parametric testing

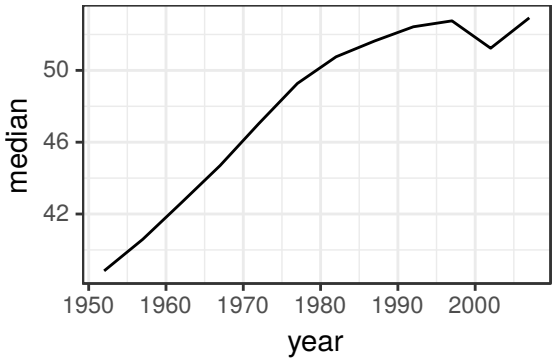
Mann-Whitney U test is also called the Wilcoxon rank sum test (note the Wilcoxon signed rank test is for paried data).

Is there a significant increase in the life expectencies for African countries between 1992 and 2007? How about 1982 and 2007?

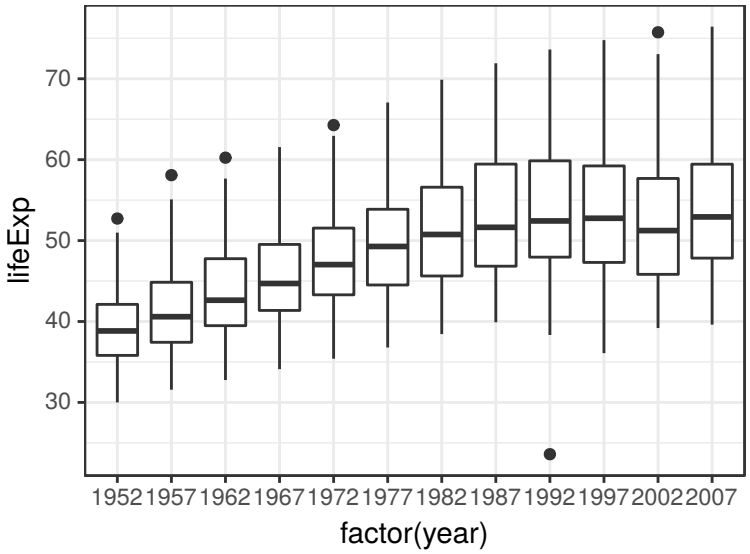
```
mydata$year %>%  unique()
```

```
##  [1] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002 2007
```

```
mydata %>%
  filter(continent == "Africa") %>%
  group_by(year) %>%
  summarise(mean = mean(lifeExp), median = median(lifeExp)) %>%
  ggplot(aes(x = year, y = median)) +
      geom_line()
```



```
mydata %>%
  filter(continent == "Africa") %>%
  ggplot(aes(x = factor(year), y=lifeExp)) + #demonstrate that needs to be
  geom_boxplot()
```




```
mydata %>%
  filter(year %in% c(1992, 2007)) %>%
  filter(continent == "Africa") %>%
  wilcox.test(lifeExp~year, data=.)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  lifeExp by year
## W = 1314, p-value = 0.8074
## alternative hypothesis: true location shift is not equal to 0
```

6.7 Solutions

5.2.2

```
mydata %>%
  filter(continent == "Europe") %>%
  ggplot(aes(x = lifeExp)) +
  geom_histogram() +
  facet_wrap(~year)

mydata %>%
  filter(continent == "Europe") %>%
  ggplot(aes(sample = lifeExp)) +
  geom_point(stat = "qq") +
  facet_wrap(~year)

mydata %>%
  filter(continent == "Europe") %>%
  ggplot(aes(y = lifeExp, x = factor(year))) +
  geom_boxplot()
```

6.8 Advanced example

This is a complex but useful example which shows you the power of the syntax. Here multiple *t*-tests are performed and reported with just a few lines of code.

Performing *t*-tests across all continents at once:

```
mydata %>%
  filter(year %in% c(1997, 2007)) %>%
```

```
  group_by(continent) %>%
    do(
        tidy(
            t.test(lifeExp~year, data=.)
        )
    )
```

```
## # A tibble: 5 x 11
## # Groups:   continent [5]
##   continent estimate estimate1 estimate2 statistic p.value parameter
##   <fct>        <dbl>     <dbl>     <dbl>     <dbl>   <dbl>     <dbl>
## 1 Africa      -1.21      53.6      54.8    -0.657  0.513      102.
## 2 Americas    -2.46      71.2      73.6    -1.86   0.0690      47.6
## 3 Asia        -2.71      68.0      70.7    -1.37   0.175       64.0
## 4 Europe      -2.14      75.5      77.6    -2.73   0.00842     57.9
## 5 Oceania     -2.53      78.2      80.7    -3.08   0.0965       1.91
## # ... with 4 more variables: conf.low <dbl>, conf.high <dbl>,
## #   method <chr>, alternative <chr>
```

7

Linear regression

7.1 Data

We will be using the same gapminder dataset as in the last two sessions.

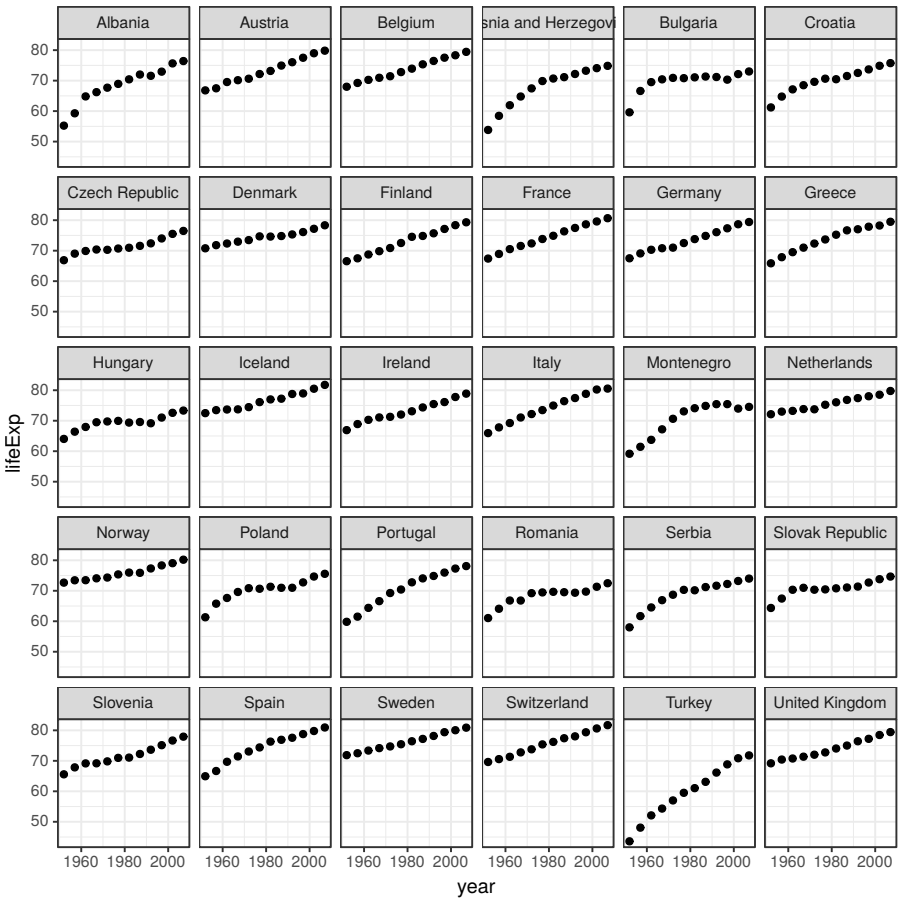
```
library(tidyverse)
library(gapminder) # dataset
library(lubridate) # handles dates
library(broom)     # transforms statistical output to data frame

mydata = gapminder
```

7.2 Plotting

Let's plot the life expectancies in European countries over the past 60 years:

```
mydata %>%
  filter(continent == "Europe") %>%
  ggplot(aes(x = year, y = lifeExp)) +
  geom_point() +
  facet_wrap(~country) +
```

```
  theme_bw() +
  scale_x_continuous(breaks = c(1960, 1980, 2000))
```



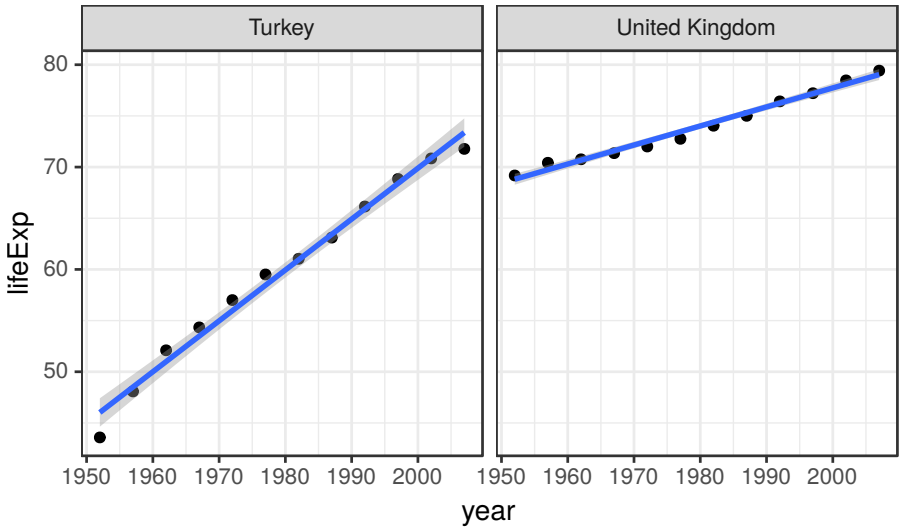
7.2.1 Exercise

Save the above filter into a new variable called `eurodata`:

```
eurodata = mydata %>%
  filter(continent == "Europe")
```

7.2.2 Exercise

Create the same plot as above (life expectancy over time), but for just Turkey and the United Kingdom, and add linear regression lines. Hint: use `+ geom_smooth(method = "lm")` for the lines. `lm()` stands for linear model.



7.3 Simple linear regression

As you can see, `ggplot()` is very happy to run and plot linear regression for us. To access the results, however, we should save the full results of the linear regression models into variables in our Environment. We can then investigate the intercepts and the slope coefficients (linear increase per year):

```
fit_uk = mydata %>%
  filter(country == "United Kingdom") %>%
  lm(lifeExp~year, data = .)  # the data=. argument is necessary
```

```
fit_turkey = mydata %>%
  filter(country == "Turkey") %>%
  lm(lifeExp~year, data = .)


fit_uk$coefficients

fit_turkey$coefficients
```

```
##  (Intercept)         year
## -294.1965876    0.1859657
##  (Intercept)         year
## -924.5898865    0.4972399
```

7.3.1 Exercise

To make the intercepts more meaningful, add a new column called `year_from1952` and redo `fit_turkey` and `fit_uk` using `year_from1952` instead of `year`.

```
mydata$year_from1952 = mydata$year - 1952

fit_uk = mydata %>%
  filter(country == "United Kingdom") %>%
  lm(lifeExp~year_from1952, data = .)


fit_turkey = mydata %>%
  filter(country == "Turkey") %>%
  lm(lifeExp~year_from1952, data = .)


fit_uk$coefficients
```

```
fit_turkey$coefficients
```

```
##   (Intercept) year_from1952
##    68.8085256     0.1859657
##   (Intercept) year_from1952
##    46.0223205     0.4972399
```

7.3.2 Model information: `summary()`, `tidy()` ,`glance()`

Accessing all other information about our regression model:

```
fit_uk %>% summary()
```

```
##
## Call:
## lm(formula = lifeExp ~ year_from1952, data = .)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.69767 -0.31962  0.06642  0.36601  0.68165
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  68.808526   0.240079  286.61  < 2e-16 ***
## year_from1952  0.185966   0.007394   25.15 2.26e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4421 on 10 degrees of freedom
## Multiple R-squared:  0.9844, Adjusted R-squared:  0.9829
## F-statistic: 632.5 on 1 and 10 DF,  p-value: 2.262e-10
```

```
fit_uk %>% tidy()
```

```
## # A tibble: 2 x 5
##   term         estimate std.error statistic  p.value
##   <chr>           <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)     68.8     0.240       287.  6.58e-21
## 2 year_from1952    0.186   0.00739      25.1 2.26e-10
```

```
fit_uk %>% glance()
```

```
## # A tibble: 1 x 11
##  r.squared adj.r.squared sigma statistic  p.value    df logLik  AIC  BIC
##      <dbl>         <dbl> <dbl>     <dbl>    <dbl> <int>  <dbl> <dbl> <dbl>
## 1    0.984         0.983 0.442     633. 2.26e-10     2  -6.14  18.3  19.7
## # ... with 2 more variables: deviance <dbl>, df.residual <int>
```

7.4 If you are new to linear regression

See these interactive Shiny apps provided by RStudio:

https://gallery.shinyapps.io/simple_regression/

https://gallery.shinyapps.io/multi_regression/

(`library(shiny)` is an R package for making your output interactive)

7.4.1 Exercise - Residuals

Open the first Shiny app ("Simple regression"). Move the sliders until the red lines (residuals*) turn green - this means you've made the line fit the points as well as possible. Look at the intercept and slope - discuss with your neighbour or a tutor what these numbers mean and how they affect the straight line on the plot.

*Residual is how far away each point (observation) is from the

linear regression line. (In this example it's the linear regression line, but residuals are relevant in many other contexts as well.)

7.5 Multiple linear regression

Multiple linear regression includes more than one predictor variable. There are a few ways to include more variables, depending on whether they should share the intercept and how they interact:

Simple linear regression (exactly one predictor variable):

```
myfit = lm(lifeExp~year, data=eurodata)
```

Multiple linear regression (additive):

```
myfit = lm(lifeExp~year+country, data=eurodata)
```

Multiple linear regression (all interactions):

```
myfit = lm(lifeExp~year*country, data=eurodata)
```

These examples of multiple regression include two variables: `year` and `country`, but we could include more by just adding them with `+`.

7.5.1 Exercise

Open the second Shiny app ("Multiple regression") and see how:

- In simple regression, there is only one intercept and slope for the whole dataset.
- Using the additive model (`lm(formula = y ~ x + group`) the two lines (one for each group) have different intercepts but the same slope. However, the `lm()` summary seems to only include one line called "(Intercept)", how to find the intercept for the second group of points?

- Using the interactive model (`lm(formula = y ~ x*group)`) the two lines have different intercepts and different slopes.

7.5.2 Exercise

Convince yourself that using an fully interactive multivariable model is similar to running several separate simple linear regression models. Remember that we calculate the life expectancy in 1952 (intercept) and improvement per year (slope) for Turkey and the United Kingdom:

```
fit_uk %>%
  tidy() %>%
  mutate(estimate = round(estimate, 2)) %>%
  select(term, estimate)
```

```
## # A tibble: 2 x 2
##   term          estimate
##   <chr>            <dbl>
## 1 (Intercept)      68.8
## 2 year_from1952     0.19
```

```
fit_turkey %>%
  tidy() %>%
  mutate(estimate = round(estimate, 2)) %>%
  select(term, estimate)
```

```
## # A tibble: 2 x 2
##   term          estimate
##   <chr>            <dbl>
## 1 (Intercept)       46.0
## 2 year_from1952      0.5
```

(The lines `tidy()`, `mutate()`, and `select()` are only included for neater presentation here, you can use `summary()` instead.)

We can do this together using `year_from1952*country` in the `lm()`:

```
mydata %>%
  filter(country %in% c("Turkey", "United Kingdom")) %>%
  lm(lifeExp ~ year_from1952*country, data = .)   %>%
  tidy() %>%
  mutate(estimate = round(estimate, 2)) %>%
  select(term, estimate)
```

```
## # A tibble: 4 x 2
##   term                                  estimate
##   <chr>                                    <dbl>
## 1 (Intercept)                              46.0
## 2 year_from1952                             0.5
## 3 countryUnited Kingdom                    22.8
## 4 year_from1952:countryUnited Kingdom      -0.31
```

Now. It may seem like R has omitted Turkey but the values for Turkey are actually in the Intercept = 46.02 and in year_from1952 = 0.50. Can you make out the intercept and slope for the UK? Are they the same as in the simple linear regression model?

7.5.3 Exercise

Add a third country (e.g. "Portugal") to `filter(country %in% c("Turkey", "United Kingdom"))` in the above example. Do the results change?

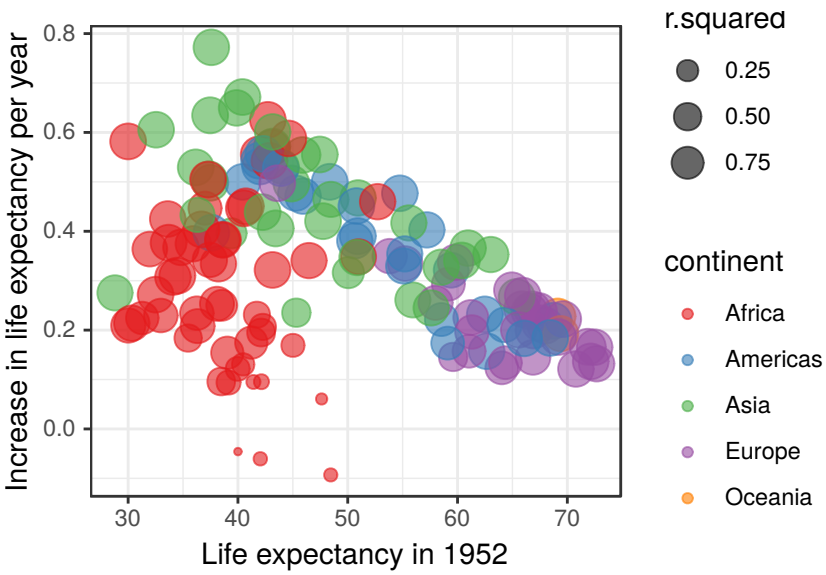
7.5.4 Optional (Advanced) Exercise

Run separate linear regression models for every country in the dataset at the same time and putting it all in two neat dataframes (one for the coefficients, one for the summary statistics):

```
linfit_coefficients = mydata %>%
  group_by(country) %>%
  do(
    tidy(
      lm(lifeExp~year, data=.)
    )
  )


linfit_overall = mydata %>%
  group_by(country) %>%
  do(
    glance(
      lm(lifeExp~year, data=.)
    )
  )
```

Plot the linear regression estimate (improvement per year between 1952 – 2007), size the points by their r-squared values, and colour the points by continent (hint: you will have to join `mydata`, `linfit_coefficients %>% filter(term == "year")`, and `linfit_overall`):

```
mydata %>%
  filter(year == 1952) %>%
  full_join(linfit_coefficients %>% filter(term == "year"), by = "country")
  full_join(linfit_overall, by = "country") %>%
  ggplot(aes(x = lifeExp, y = estimate, colour = continent, size = r.squared
  geom_point(alpha = 0.6) +
  theme_bw() +
  scale_colour_brewer(palette = "Set1") +
  ylab("Increase in life expectancy per year") +
  xlab("Life expectancy in 1952")
```



7.6 Very advanced example

Or you can do the above in a nested tibble/data frame:

```
nested_linreg = mydata %>%
  group_by(country) %>%
  nest() %>%
  mutate(model = purrr::map(data, ~ lm(lifeExp ~ year, data = .)))
```

7.7 Solutions

6.2.2

```
mydata %>%
  filter(country %in% c("United Kingdom", "Turkey") ) %>%
  ggplot(aes(x = year.formatted, y = lifeExp)) +
  geom_point() +
  facet_wrap(~country) +
  theme_bw() +
  geom_smooth(method = "lm")
```

6.5.3

```
mydata %>%
  filter(country %in% c("Turkey", "United Kingdom", "Portugal")) %>%
  lm(lifeExp ~ year_from1952*country, data = .)   %>%
  tidy() %>%
  mutate(estimate = round(estimate, 2)) %>%
  select(term, estimate)
```

Overall, the estimates for Turkey and the UK do not change, but Portugal becomes the reference (alphabetically first) to which you can subtract or add the relevant lines for Turkey and the UK.

8

Tests for categorical variables

8.1 Data

We are now changing to a new dataset, `melanoma`. Click on `mydata` in your environment and have a look at the values - you'll see that categorical variables are coded as numbers, rather than text. You will need to recode these numbers into proper factors.

```
library(tidyverse)
library(finalfit)
library(broom)
mydata = boot::melanoma
```

8.1.1 Recap on factors

Press `F1` on `boot::melanoma` to see its description. Use the information from help to change the numbers into proper factors (e.g. 0 - female, 1 - male).

```
mydata$status %>%
  factor() %>%
  fct_recode("Died"  = "1",
             "Alive" = "2",
             "Died - other causes" = "3") %>%
  fct_relevel("Alive") -> # move Alive to front (first factor level)
  mydata$status.factor    # so odds ratio will be relative to that
```

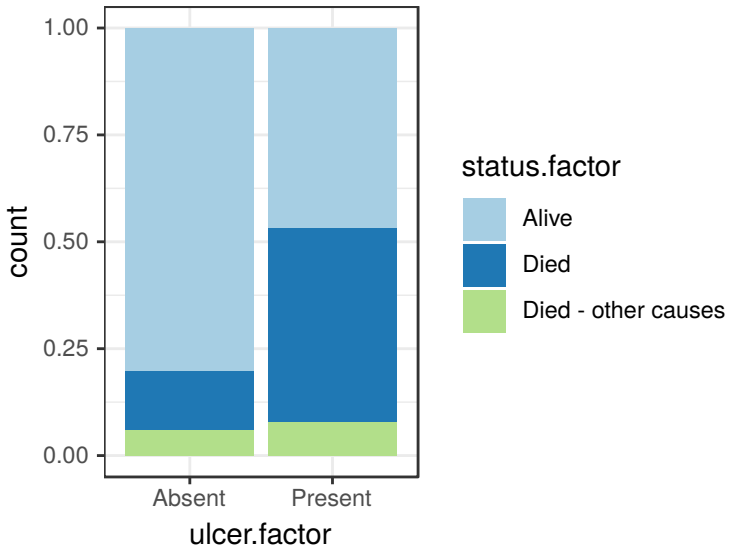
```
mydata$sex %>%
  factor() %>%
  fct_recode("Female" = "0",
             "Male" = "1") ->
  mydata$sex.factor

mydata$ulcer %>%
  factor() %>%
  fct_recode("Present" = "1",
             "Absent"  = "0") ->
  mydata$ulcer.factor

#the cut() function makes a continuous variable into a categorical variable
mydata$age %>%
  cut(breaks = c(4,20,40,60,95), include.lowest=TRUE) ->
  mydata$age.factor
```

8.2 Chi-squared test / Fisher's exact test

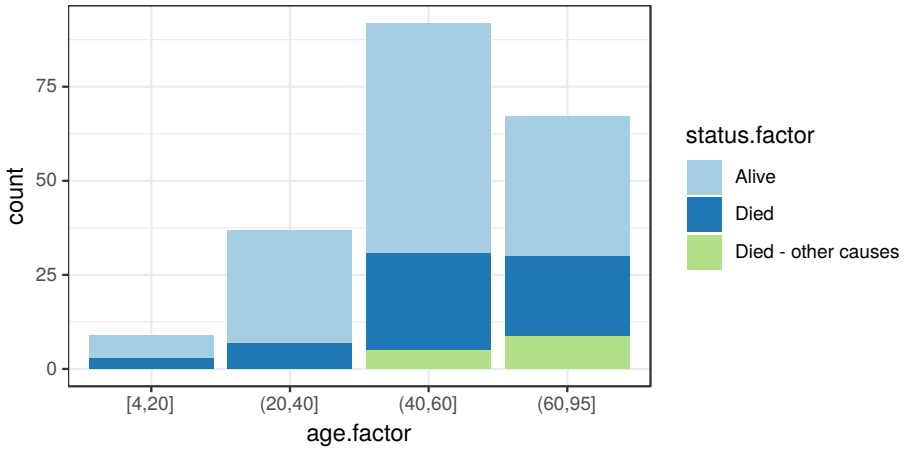
8.2.1 Plotting

Always plot new data first!

```
mydata %>%
    ggplot(aes(x = ulcer.factor, fill=status.factor)) +
      geom_bar(position = "fill") +
      theme_bw() +
      scale_fill_brewer(palette = "Paired")
```

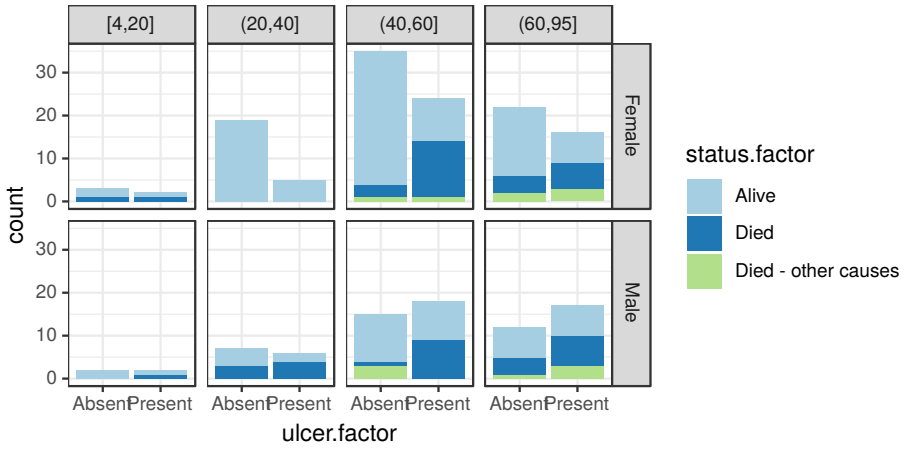



```
mydata %>%
  ggplot(aes(x = age.factor, fill = status.factor)) +
    geom_bar() +
    theme_bw() +
    scale_fill_brewer(palette = "Paired")
```




```
mydata %>%
  ggplot(aes(x = ulcer.factor, fill=status.factor)) +
```

```
    geom_bar() +
    theme_bw() +
    scale_fill_brewer(palette = "Paired") +
    facet_grid(sex.factor~age.factor)
```



8.3 Analysis

8.3.1 Using base R

First lets group together those that 'died of another cause' with those 'alive', to give a disease-specific mortality variable (`fct_collapse()` will help us).

```
mydata$status.factor %>%
    fct_collapse("Alive" = c("Alive", "Died - other causes")) ->
  mydata$status.factor
```

Let's test mortality against sex.

```
table(mydata$status.factor, mydata$sex.factor)
```

```
##
##         Female Male
##   Alive     98   50
##   Died      28   29
```

```
chisq.test(mydata$status.factor, mydata$sex.factor)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  mydata$status.factor and mydata$sex.factor
## X-squared = 4.3803, df = 1, p-value = 0.03636
```

Note that `chisq.test()` defaults to the Yates' continuity correction.

It is fine to use this, but if you have a particular need not to, turn if off with `chisq.test(mydata$status.factor, mydata$sex.factor, correct=FALSE)`.

8.3.2 Using `CrossTable`

This gives lots of useful information. It is readable in R and has lots of options, including Fisher's exact test. It is not that easy to extract results.

```
library(gmodels)
# F1 CrossTable to see options
CrossTable(mydata$status.factor, mydata$sex.factor, chisq=TRUE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  205
##
##
##                      | mydata$sex.factor
## mydata$status.factor |   Female |    Male | Row Total |
## ---------------------|----------|----------|----------|
##              Alive |       98 |       50 |      148 |
##                    |    0.544 |    0.868 |          |
##                    |    0.662 |    0.338 |    0.722 |
##                    |    0.778 |    0.633 |          |
##                    |    0.478 |    0.244 |          |
## ---------------------|----------|----------|----------|
##               Died |       28 |       29 |       57 |
##                    |    1.412 |    2.253 |          |
##                    |    0.491 |    0.509 |    0.278 |
##                    |    0.222 |    0.367 |          |
##                    |    0.137 |    0.141 |          |
## ---------------------|----------|----------|----------|
##       Column Total |      126 |       79 |      205 |
##                    |    0.615 |    0.385 |          |
```

```
## --------------------|----------|----------|----------|
##
##
## Statistics for All Table Factors
##
##
## Pearson's Chi-squared test
## ------------------------------------------------------------
## Chi^2 =  5.076334     d.f. =  1     p =  0.0242546
##
## Pearson's Chi-squared test with Yates' continuity correction
## ------------------------------------------------------------
## Chi^2 =  4.380312     d.f. =  1     p =  0.03635633
##
##
```

8.3.3 Exercise

Use the 3 methods (`table`, `chisq.test`, `CrossTable`) to test `status.factor` against `ulcer.factor`.

```
table(mydata$status.factor, mydata$ulcer.factor)
chisq.test(mydata$status.factor, mydata$ulcer.factor)
```

Using `CrossTable`

```
CrossTable(mydata$status.factor, mydata$ulcer.factor, chisq=TRUE)
```

8.3.4 Fisher's exact test

An assumption of the chi-squared test is that the 'expected cell count' is greater than 5. If it is less than 5 the test becomes unreliable and the Fisher's exact test is recommended.

Run the following code.

```
library(gmodels)
CrossTable(mydata$status.factor, mydata$age.factor, expected=TRUE, chisq=TRU
```

```
## Warning in chisq.test(t, correct = FALSE, ...): Chi-squared approximation
## may be incorrect
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |              Expected N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  205
##
##
##                      | mydata$age.factor
## mydata$status.factor |   [4,20] |  (20,40] |  (40,60] |  (60,95] | Row Total
## ---------------------|----------|----------|----------|----------|----
##            Alive |      6 |     30 |     66 |     46 |    148 |
##                  |   6.498 |  26.712 |  66.420 |  48.371 |        |
##                  |   0.038 |   0.405 |   0.003 |   0.116 |        |
##                  |   0.041 |   0.203 |   0.446 |   0.311 |   0.722 |
##                  |   0.667 |   0.811 |   0.717 |   0.687 |        |
##                  |   0.029 |   0.146 |   0.322 |   0.224 |        |
## ---------------------|----------|----------|----------|----------|----
##             Died |      3 |      7 |     26 |     21 |     57 |
##                  |   2.502 |  10.288 |  25.580 |  18.629 |        |
##                  |   0.099 |   1.051 |   0.007 |   0.302 |        |
##                  |   0.053 |   0.123 |   0.456 |   0.368 |   0.278 |
```

```
##               |    0.333 |    0.189 |    0.283 |    0.313 |        |
##               |    0.015 |    0.034 |    0.127 |    0.102 |        |
## --------------------|----------|----------|----------|----------|----
##      Column Total |        9 |       37 |       92 |       67 |      205 |
##               |    0.044 |    0.180 |    0.449 |    0.327 |        |
## --------------------|----------|----------|----------|----------|----
##
##
## Statistics for All Table Factors
##
##
## Pearson's Chi-squared test
## ------------------------------------------------------------
## Chi^2 =  2.019848     d.f. =  3     p =  0.5682975
##
##
##
```

Why does it give a warning? Run it a second time including `fisher=TRUE`.

```
library(gmodels)
CrossTable(mydata$status.factor, mydata$age.factor, expected=TRUE, chisq=TRU
```

```
## Warning in chisq.test(t, correct = FALSE, ...): Chi-squared approximation
## may be incorrect
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |              Expected N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
```

```
##
##
## Total Observations in Table:  205
##
##
##                       | mydata$age.factor
## mydata$status.factor |   [4,20] |  (20,40] |  (40,60] |  (60,95] | Row Total
## ---------------------|----------|----------|----------|----------|----
##            Alive |      6 |     30 |     66 |     46 |    148 |
##                  |   6.498 |  26.712 |  66.420 |  48.371 |        |
##                  |   0.038 |   0.405 |   0.003 |   0.116 |        |
##                  |   0.041 |   0.203 |   0.446 |   0.311 |   0.722 |
##                  |   0.667 |   0.811 |   0.717 |   0.687 |        |
##                  |   0.029 |   0.146 |   0.322 |   0.224 |        |
## ---------------------|----------|----------|----------|----------|----
##             Died |      3 |      7 |     26 |     21 |     57 |
##                  |   2.502 |  10.288 |  25.580 |  18.629 |        |
##                  |   0.099 |   1.051 |   0.007 |   0.302 |        |
##                  |   0.053 |   0.123 |   0.456 |   0.368 |   0.278 |
##                  |   0.333 |   0.189 |   0.283 |   0.313 |        |
##                  |   0.015 |   0.034 |   0.127 |   0.102 |        |
## ---------------------|----------|----------|----------|----------|----
##       Column Total |      9 |     37 |     92 |     67 |    205 |
##                  |   0.044 |   0.180 |   0.449 |   0.327 |        |
## ---------------------|----------|----------|----------|----------|----
##
##
## Statistics for All Table Factors
##
##
## Pearson's Chi-squared test
## ------------------------------------------------------------
## Chi^2 =  2.019848     d.f. =  3     p =  0.5682975
##
##
##
```

8.4 Summarising multiple factors (optional)

`CrossTable` is useful for summarising single variables. We often want to summarise more than one factor or continuous variable against our `dependent` variable of interest. Think of Table 1 in a journal article.

8.5 Summarising factors with `library(finalfit)`

This is our own package which we have written and maintain. It contains functions to summarise data for publication tables and figures, and to easily run regression analyses. We specify a `dependent` or outcome variable, and a set of `explanatory` or predictor varaibles.

```
library(finalfit)
mydata %>%
  summary_factorlist(dependent = "status.factor",
                     explanatory = c("sex.factor", "ulcer.factor", "age.facto
                     p = TRUE,
                     column = TRUE)
```

```
## Warning in chisq.test(tab, correct = FALSE): Chi-squared approximation may
## be incorrect
```

```
##          label  levels     Alive      Died      p
## 5   sex.factor  Female 98 (66.2) 28 (49.1)  0.024
## 6                 Male 50 (33.8) 29 (50.9)
## 7 ulcer.factor  Absent 99 (66.9) 16 (28.1) <0.001
## 8              Present 49 (33.1) 41 (71.9)
## 1   age.factor  [4,20]   6 (4.1)   3 (5.3)  0.568
## 2              (20,40] 30 (20.3)  7 (12.3)
```

```
## 3              (40,60] 66 (44.6) 26 (45.6)
## 4              (60,95] 46 (31.1) 21 (36.8)
```

8.5.1 Summarising factors with `library(tidyverse)`

8.5.2 Example

`Tidyverse` gives the flexibility and power to examine millions of rows of your data any way you wish. The following are intended as an extension to what you have already done. These demonstrate some more advanced approaches to combining `tidy` functions.

```
# Calculate number of patients in each group
counted_data = mydata %>%
  count(ulcer.factor, status.factor)

# Add the total number of people in each status group
counted_data2  = counted_data %>%
  group_by(status.factor) %>%
  mutate(total = sum(n))
```

```
# Calculate the percentage of n to total
counted_data3 = counted_data2 %>%
  mutate(percentage = round(100*n/total, 1))
```

Create a combined columns of both `n` and `percentage` using `paste()` to add brackets around the percentage.

```
counted_data4  = counted_data3 %>%
  mutate(count_perc = paste0(n, " (", percentage, ")"))
```

Or combine everything together without the intermediate `counted_data` breaks.

```
mydata %>%
  count(ulcer.factor, status.factor) %>%
  group_by(status.factor) %>%
  mutate(total = sum(n)) %>%
  mutate(percentage = round(100*n/total, 1)) %>%
  mutate(count_perc = paste0(n, " (", percentage, ")")) %>%
  select(-total, -n, -percentage) %>%
  spread(status.factor, count_perc)
```

```
## # A tibble: 2 x 3
##   ulcer.factor Alive     Died
##   <fct>        <chr>     <chr>
## 1 Absent       99 (66.9) 16 (28.1)
## 2 Present      49 (33.1) 41 (71.9)
```

8.5.3 Exercise

By changing one and only one word at a time in the above block (the "Combine everything together" section)

Reproduce this:

```
##   age.factor     Alive      Died
## 1     [4,20]   6 (4.1)   3 (5.3)
## 2    (20,40] 30 (20.3)  7 (12.3)
## 3    (40,60] 66 (44.6) 26 (45.6)
## 4    (60,95] 46 (31.1) 21 (36.8)
```

And then this:

```
##   sex.factor     Alive      Died
## 1     Female 98 (66.2) 28 (49.1)
## 2       Male 50 (33.8) 29 (50.9)
```

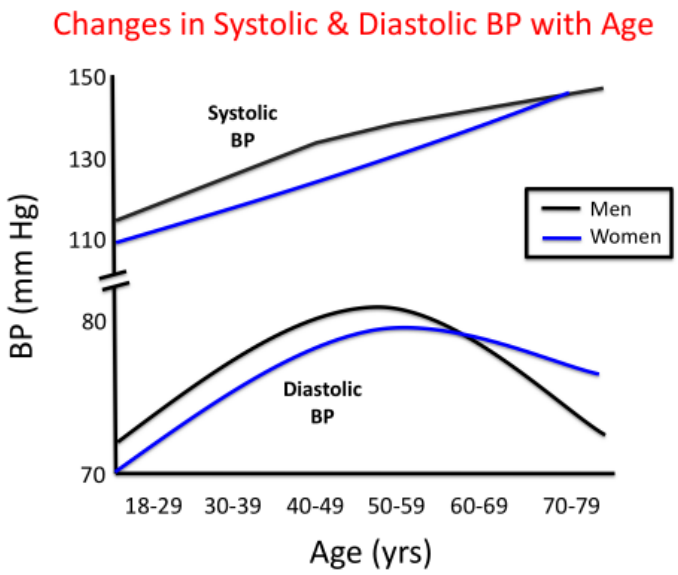
Solution: The only thing you need to change is the first variable in `count()`, e.g., `count(age.factor, ....`

9

Logistic regression

9.1 What is Logistic Regression?

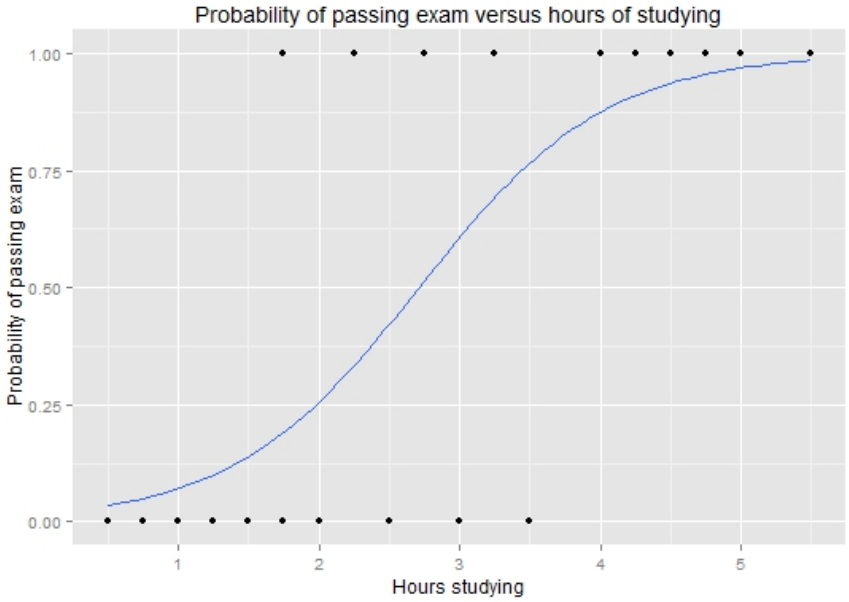
As we have seen in previously, regression analysis is a statistical process for estimating the relationships between variables. For instance, we may try to predict the blood pressure of a group of patients based on their age. As age and blood pressure are on a continuous scale, this is an example of linear regression.



Adapted from: JNC7 & Burt et al (1995) Hypertension 23:305-313

Logistic regression is an extension of this, where the variable being predicted is *categorical*. We will deal with binary logistic regres-

sion, where the variable being predicted has two levels, e.g. yes or no, 0 or 1. In healthcare, this is usually done for an event (like death) occurring or not occurring. Logistic regression can tell us the probability of the outcome occuring.



Logistic regression lets you adjust for the effects of confounding factors on an outcome. When you read a paper that says it has adjusted for confounding factors, this is the usual method which is used.

Adjusting for confounding factors allows us to isolate the true effect of a variable upon an outcome. For example, if we wanted to know the effects of smoking on deaths from heart attacks, we would need to also control for things like sex and diabetes, as we know they contribute towards heart attacks too.

Although in binary logistic regression the outcome must have two levels, the predictor variables (also known as the explanatory variables) can be either continuous or categorical.

Logistic regression can be performed to examine the influence of

one predictor variable, which is known as a univariable analysis. Or multiple predictor variables, known as a multivariable analysis.

9.2 Definitions

Dependent variable (in clinical research usually synonymous to **outcome**) - is what we are trying to explain, i.e. we are trying to identify the factors associated with a particular outcome. In binomial logistic regression, the dependent variable has exactly two levels (e.g. "Died" or "Alive", "Yes - Complications" or "No Complications", "Cured" or "Not Cured", etc.).

Explanatory variables (also known as **predictors**, **confounding** variables, or **"adjusted for"**) - patient-level information, usually including demographics (age, gender) as well as clinical information (disease stage, tumour type). Explanatory variables can be categorical as well as continuous, and categorical variables can have more than two levels.

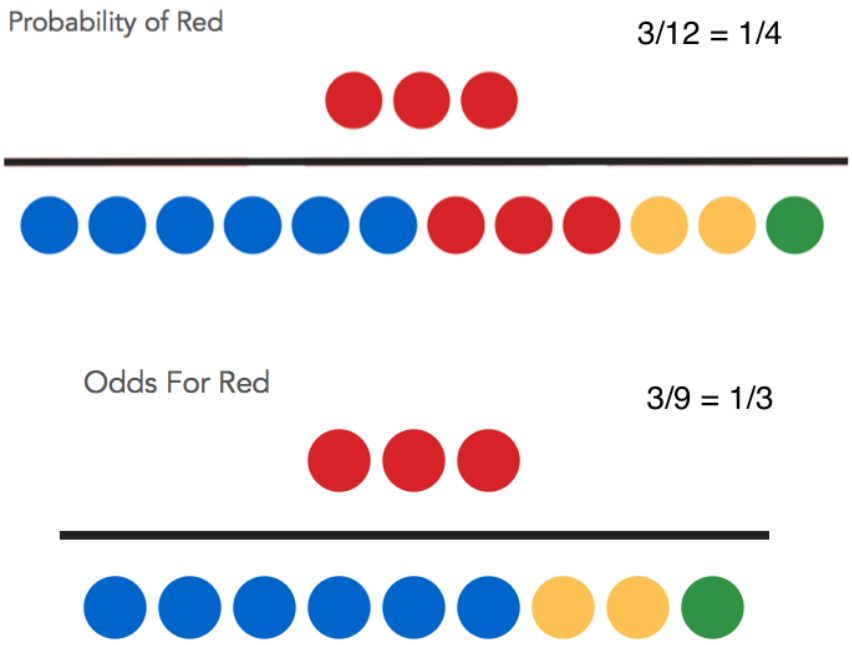
Univariable - analysis with only one Explanatory variable.

Multivariable - analysis with more than one Explanatory variable. Synonymous to "adjusted".

(**Multivariate** - technically means more than one **Dependent variable** (we will not discuss this type of analysis), but very often used interchangeably with **Multivariable**.)

9.3 Odds and probabilities

Odds and probabilities can get confusing so let's get them straight:



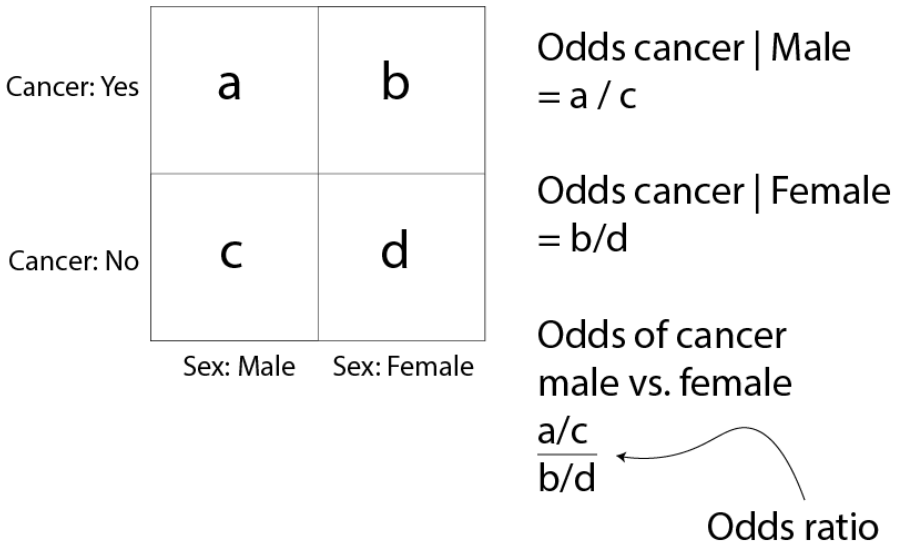
Odds and probabilities can be interconverted. For example, if the odds of a patient dying from a disease are `9 to 1` then the probability of death (also known as risk) is 10%. Odds of `1 to 1` equal 50%.

$Odds = \frac{p}{1-p}$, where p is the probability of the outcome occuring (or the circle being red).

Look at the numbers and convince yourself that this works.

9.3.1 Odds ratios

For a given categorical explanatory variable (e.g. gender), the likelihood of an outcome/dependent occuring (e.g cancer) can be expressed in a ratio of odds or odds ratio, e.g. the odds of men developing cancer is 2-times that of females, odds ratio = 2.0.



An alternative is a ratio of probabilites, called a risk ratio or relative risk. Odds ratios have useful mathematical characteristics and are the main expression of results in logistic regression analysis.

9.4 Melanoma dataset

Malignant melanoma is a cancer of the skin. It is agressive and highly invasive, making it difficult to treat.

It's classically divided into 4 stages of severity, based upon the depth of the tumour:

- Stage I: <0.5 mm depth
- Stage II: 0.5 to 1.0 mm depth
- Stage III: 1.0 to 4.0 mm depth
- Stage IV: > 4.0 mm depth

This will be important in our analysis as we will creating a new variable based upon this.

Using logistic regression, we will investigate factors associated with death from malignant melanoma.

9.4.1 Doing logistic regression in R

There are a few different ways of doing logistic regression in R. The `glm()` function is probably the most common and most flexible one to use. (`glm` stands for `generalised linear model`.)

Within the `glm()` function there are several `options` in the function we must define to make R run a logistic regression.

`data` - you must define the dataframe to be used in the regression.

`family` - this tells R to treat the analysis as a logisitic regression. For our purposes, `family` will always be `"binomial"` (as binary data follow this distribution).

`x ~ a + b + c` - this is the formula for the logistic regression, with `x` being the outcome and `a`, `b` and `c` being predictor variables.

Note the outcome is separated from the rest of the formula and

sits on the left hand side of a `~`. The confounding variables are on the right side, separated by a `+` sign.

The final `glm()` function takes the following form:

```
glm(x ~ a + b + c + d, data = data, family = "binomial")
```

9.5 Setting up your data

The most important step to ensure a good basis to start from is to ensure your variables are well structured and your outcome variable has exactly two outcomes.

We will need to make sure our outcome variables and predictor variables (the ones we want to adjust for) are suitably prepared.

In this example, the outcome variable called `status.factor` describes whether patients died or not and will be our (dependent) variable of interest.

9.5.1 Worked Example

```
library(tidyverse)

load("melanoma_factored.rda")
#Load in data from the previous session
```

Here `status.factor` has three levels: `Died`, `Died - other causes` and `Alive`. This is not useful for us, as logistic regression requires outcomes to be binary (exactly two levels).

We want to find out which variables predict death from melanoma. So we should create a new factor variable, `died_melanoma.factor`. This will have two outcomes, `Yes` (did die from melanoma) or `No` (did not die from melanoma).

```
mydata$status.factor %>%
  fct_collapse("Yes" = c("Died"),
               "No"  = c("Alive", "Died - other causes")) ->
  mydata$died_melanoma.factor

mydata$died_melanoma.factor %>% levels()
```

```
## [1] "No"  "Yes"
```

9.6 Creating categories

Now that we have set up our outcome variable, we should ensure our predictor variables are prepared too.

Remember the stages of melanoma? This is an important predictor of melanoma Mortality based upon the scientific literature.

We should take this into account in our model.

9.6.1 Exercise

Create a new variable called `stage.factor` to encompass the stages of melanoma based upon the thickness. In this data, the `thickness` variable is measured in millimetres too.

```
#the cut() function makes a continuous variable into a categorical variable
mydata$thickness %>%
    cut(breaks = c(0,0.5,1,4, max(mydata$thickness, na.rm=T)),
            include.lowest = T) ->
mydata$stage.factor

mydata$stage.factor %>% levels()
```

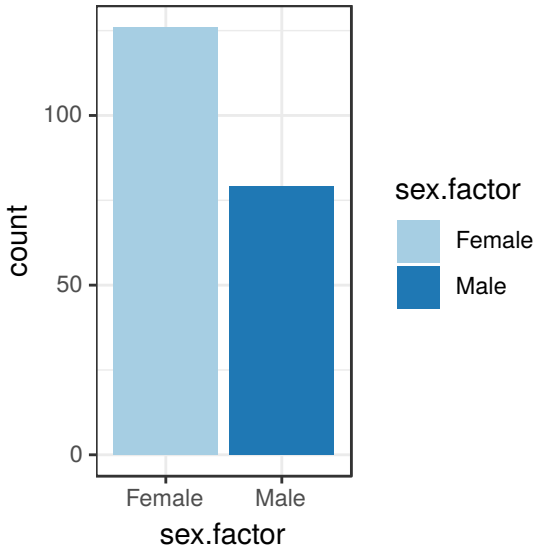
```
## [1] "[0,0.5]"  "(0.5,1]"  "(1,4]"    "(4,17.4]"
```

```
mydata$stage.factor %>%
  fct_recode("Stage I"   = "[0,0.5]",
             "Stage II"  = "(0.5,1]",
             "Stage III" = "(1,4]",
             "Stage IV"  = "(4,17.4]"
    ) -> mydata$stage.factor

mydata$stage.factor %>% levels()
```

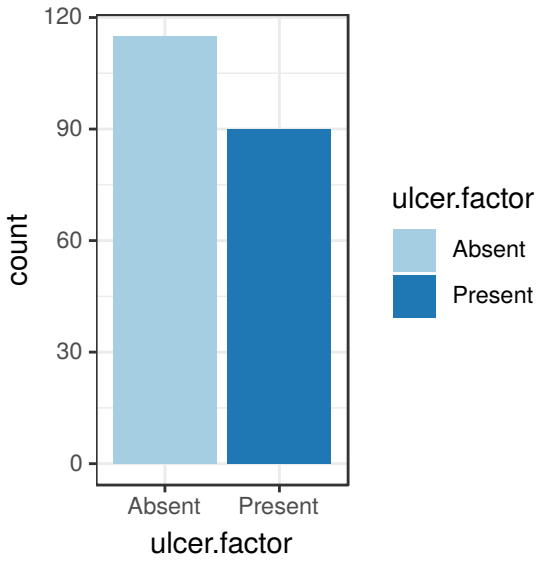
```
## [1] "Stage I"   "Stage II"  "Stage III" "Stage IV"
```

9.6.2 Always plot your data first!

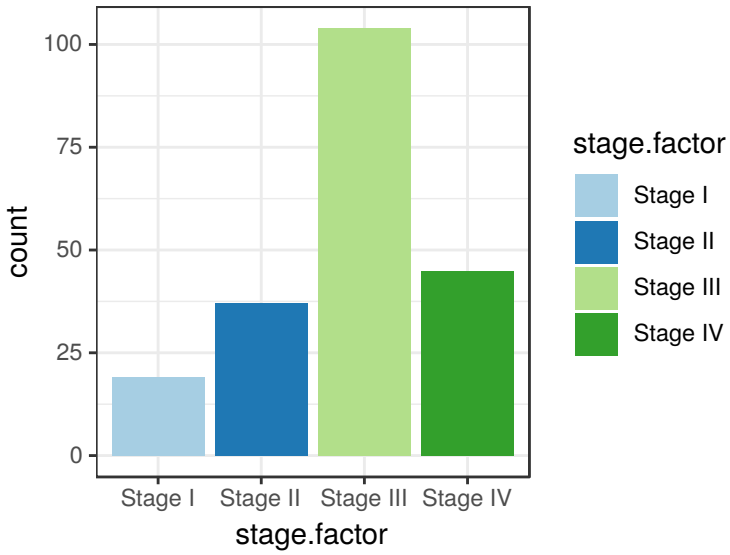
```
source("1_source_theme.R")

mydata %>%
  ggplot(aes(x = sex.factor)) +
  geom_bar(aes(fill = sex.factor))
```



```
mydata %>%
  ggplot(aes(x = ulcer.factor)) +
  geom_bar(aes(fill = ulcer.factor))
```

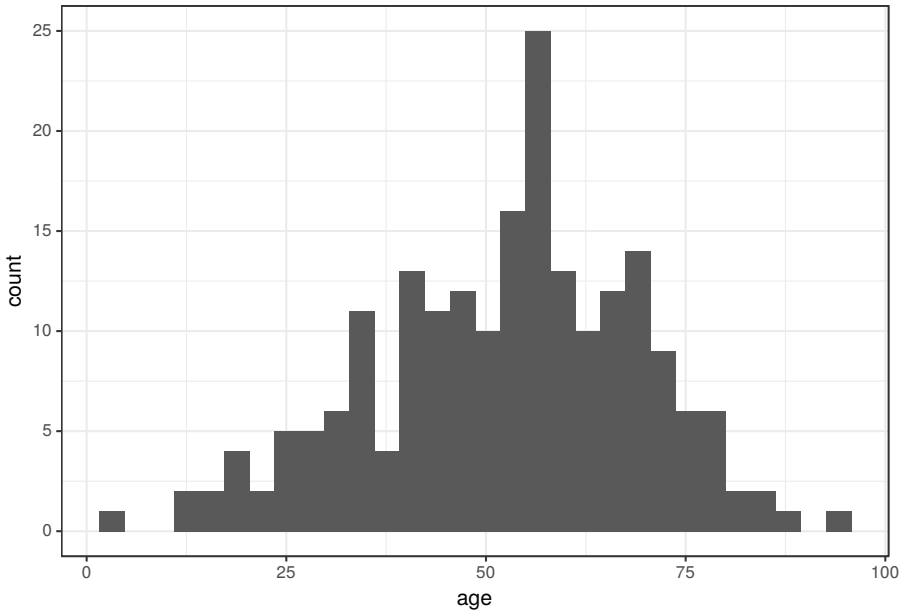



```
mydata %>%
  ggplot(aes(x = stage.factor)) +
  geom_bar(aes(fill = stage.factor))
```




```
mydata %>%
  ggplot(aes(x = age)) +
  geom_histogram(aes(fill = age))
```

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



Now we are ready for some modelling!

9.7 Basic: One explanatory variable (predictor)

Lets find out what the influence of each predictor/confounding variable is on mortality from melanoma, which may help inform a more complicated regression, with multiple predictors/confounders.

We'll start with whether the patient was male or female:

9.7.1 Worked example

First we need to create a regression model using `glm()`. We will then summarise it using `summary()`

Note, we need to use the `family` option. Specifying `'binomial'` in `family` tells `glm()` to switch to logistic regression.

```
#Create a model

glm(died_melanoma.factor ~ sex.factor, data = mydata, family = "binomial")
```

```
##
## Call:  glm(formula = died_melanoma.factor ~ sex.factor, family = "binomial",
##     data = mydata)
##
## Coefficients:
##    (Intercept)  sex.factorMale
##         -1.253           0.708
##
## Degrees of Freedom: 204 Total (i.e. Null);  203 Residual
## Null Deviance:       242.4
## Residual Deviance: 237.4     AIC: 241.4
```

```
model1 = glm(died_melanoma.factor ~ sex.factor, data = mydata, family = "bin

summary(model1)
```

```
##
## Call:
## glm(formula = died_melanoma.factor ~ sex.factor, family = "binomial",
##     data = mydata)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.9565  -0.7090  -0.7090   1.4157   1.7344
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -1.2528     0.2143  -5.846 5.03e-09 ***
## sex.factorMale   0.7080     0.3169   2.235   0.0254 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 242.35  on 204  degrees of freedom
## Residual deviance: 237.35  on 203  degrees of freedom
## AIC: 241.35
##
## Number of Fisher Scoring iterations: 4
```

Now we have created the model - fantastic!

But this doesn't mean a lot to humans reading a paper - or us in fact.

The estimate output of `summary(model_1)` represents the logarithm of the odds ratio. The odds ratio would be a lot easier to understand.

Therefore, to sort that out we should exponentiate the output of the model! The `exp()` function will do this.

```
exp(model1$coefficients)
```

```
##    (Intercept) sex.factorMale
##      0.2857143      2.0300000
```

This gives us an odds ratio of 2.03 for males. That is to say, males are twice as likely to die from melanoma than females.

Now a confidence interval might be handy. As this will be the logarithm of the confidence interval, we should exponentiate it to make it understandable.

```
exp(confint(model1))
```

```
## Waiting for profiling to be done...
```

```
##                     2.5 %    97.5 %
## (Intercept)    0.1843592 0.4284939
## sex.factorMale 1.0914854 3.7938450
```

The 2.5% is the lower bound and the 97.5% is the upper bound of the 95% confidence interval.

So we can therefore say that being male doubles your chances of dying from melanoma with an Odds Ratio of 2.03 (95% confidence interval of 1.09 to 3.79)

9.7.2 Exercise

Repeat this for all the variables contained within the data, particulary:

`stage.factor`, `age`, `ulcer.factor`, `thickness` and `age.factor`.

Write their odds ratios and 95% confidence intervals down for the next section!

Congratulations on building your first regression model in R!

9.8 Finalfit package

We have developed our `finalfit` package to help with advanced regression modelling. We will introduce it here, but not go into detail.

See www.finalfit.org for more information and updates.

9.9 Summarise a list of variables by another variable

We can use the `finalfit` package to summarise a list of variables by another variable. This is very useful for "Table 1" in many studies.

```
library(finalfit)
dependent   = "died_melanoma.factor"
explanatory = c("age", "sex.factor")

table_result = mydata %>%
  summary_factorlist(dependent, explanatory, p = TRUE)
```

label	levels	No	Yes	p
age	Mean (SD)	51.5 (16.1)	55.1 (17.9)	0.189
sex.factor	Female	98 (77.8)	28 (22.2)	0.024
	Male	50 (63.3)	29 (36.7)	

9.10 `finalfit` function for logistic regression

We can then use the `finalfit` function to run a logistic regression analysis with similar syntax.

```
dependent   = "died_melanoma.factor"
explanatory = c("sex.factor")

model2 = mydata %>%
  finalfit(dependent, explanatory)
```

Dependent: died_melanoma.factor		No	Yes	OR (univariabl
sex.factor	Female	98 (66.2)	28 (49.1)	
	Male	50 (33.8)	29 (50.9)	2.03 (1.09-3.79, p=0.025

9.11 Adjusting for multiple variables in R

Your first models only included one variable. It's time to scale them up.

Multivariable models take multiple variables and estimates how each variable predicts an event. It adjusts for the effects of each one, so you end up with a model that calculates the adjusted effect estimate (i.e. the odds ratio), upon an outcome.

When you see the term 'adjusted' in scientific papers, this is what it means.

9.11.1 Worked Example

Lets adjust for `age` (as a continuous variable), `sex.factor` and `stage.factor`. Then output them as odds ratios.

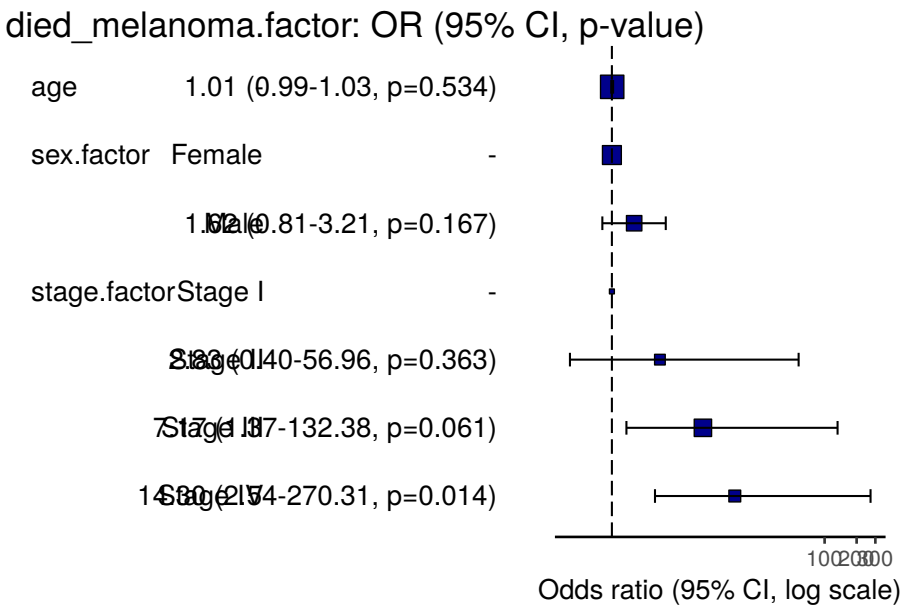
```
dependent   = "died_melanoma.factor"
explanatory = c("age", "sex.factor", "stage.factor")

model3 = mydata %>%
  finalfit(dependent, explanatory)
```

Dependent: died_melanoma.factor		No	Yes	OR
age	Mean (SD)	51.5 (16.1)	55.1 (17.9)	1.01 (0.99-1.
sex.factor	Female	98 (66.2)	28 (49.1)	
	Male	50 (33.8)	29 (50.9)	2.03 (1.09-3.
stage.factor	Stage I	18 (12.2)	1 (1.8)	
	Stage II	32 (21.6)	5 (8.8)	2.81 (0.41-56.
	Stage III	75 (50.7)	29 (50.9)	6.96 (1.34-128.
	Stage IV	23 (15.5)	22 (38.6)	17.22 (3.13-322.

```
or_plot(mydata, dependent, explanatory)
```

```
## Waiting for profiling to be done...
## Waiting for profiling to be done...
## Waiting for profiling to be done...
```

```
## Warning: Removed 2 rows containing missing values (geom_errorbarh).
```



When we enter age into regression models, the effect estimate is provided in terms of per unit increase. So in this case it's expressed in terms of an odds ratio per year increase (i.e. for every year in age gained odds of death increases by 1.02).

9.11.2 Exercise

Create a regression that includes `ulcer.factor`.

9.12 Advanced: Fitting the best model

Now we have our preliminary model. We could leave it there.

However, when you publish research, you are often asked to supply a measure of how well the model fitted the data.

There are different approaches to model fitting. Come to our course HealthyR-Advanced: Practical Logistic Regression. At this we describe use of the Akaike Information Criterion (AIC) and the C-statistic.

The C-statistic describes discrimination and anything over 0.60 is considered good. The closer to 1.00 the C-statistic is, the better the fit.

The AIC measure model fit with lower values indicating better fit.

These metrics are available here:

```
mydata %>%
  finalfit(dependent, explanatory, metrics=TRUE)
```

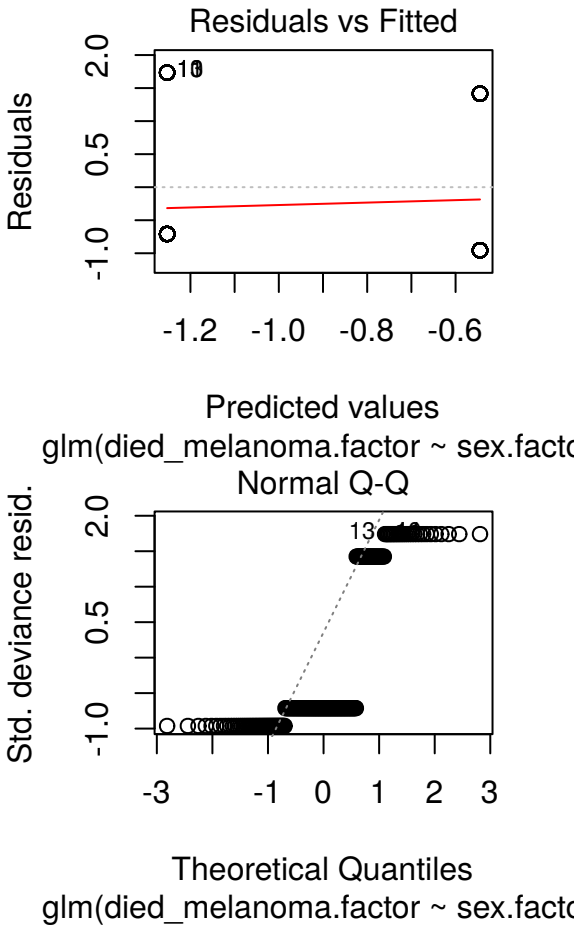
```
## Waiting for profiling to be done...
## Waiting for profiling to be done...
## Waiting for profiling to be done...
## Waiting for profiling to be done...

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## [[1]]
##  Dependent: died_melanoma.factor              No      Yes
## 1                    age Mean (SD) 51.5 (16.1) 55.1 (17.9)
## 2             sex.factor   Female  98 (66.2)  28 (49.1)
## 3                          Male  50 (33.8)  29 (50.9)
## 4            stage.factor  Stage I  18 (12.2)   1 (1.8)
## 5                        Stage II  32 (21.6)   5 (8.8)
```

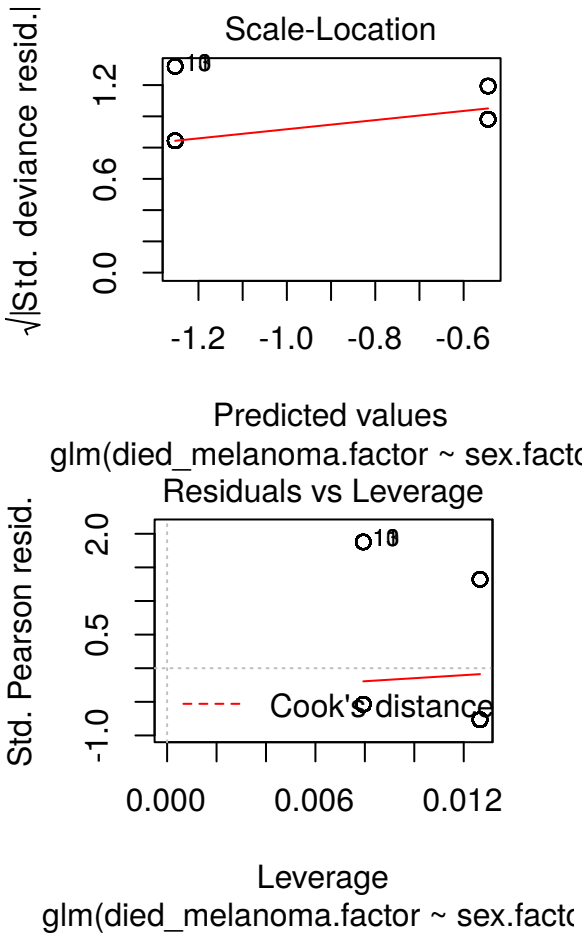
```
## 6                       Stage III  75 (50.7)  29 (50.9)
## 7                        Stage IV  23 (15.5)  22 (38.6)
##           OR (univariable)         OR (multivariable)
## 1   1.01 (0.99-1.03, p=0.163)   1.01 (0.99-1.03, p=0.534)
## 2                             -                         -
## 3   2.03 (1.09-3.79, p=0.025)   1.62 (0.81-3.21, p=0.167)
## 4                             -                         -
## 5  2.81 (0.41-56.12, p=0.362)  2.83 (0.40-56.96, p=0.363)
## 6  6.96 (1.34-128.04, p=0.065) 7.17 (1.37-132.38, p=0.061)
## 7 17.22 (3.13-322.85, p=0.008) 14.30 (2.54-270.31, p=0.014)
##
## [[2]]
## [1] "Number in dataframe = 205, Number in model = 205, Missing = 0, AIC = 232.3
```

9.12.1 Extra material: Diagnostics plots

While outwith the objectives of this course, diagnostic plots for `glm` models can be produced by:

```
plot(model1)
```


Residuals vs Fitted
Residuals
2.0
0.5
-1.0
-1.2
-1.0
-0.8
-0.6
Predicted values
glm(died_melanoma.factor ~ sex.factc
Normal Q-Q
Std. deviance resid.
2.0
0.5
-1.0
-3
-1
0
1
2
3
Theoretical Quantiles
glm(died_melanoma.factor ~ sex.factc


Scale-Location
√|Std. deviance resid.|
0.0
0.6
1.2
-1.2
-1.0
-0.8
-0.6
Predicted values
glm(died_melanoma.factor ~ sex.factc
Residuals vs Leverage
Std. Pearson resid.
-1.0
0.5
2.0
Cook's distance
0.000
0.006
0.012
Leverage
glm(died_melanoma.factor ~ sex.factc

10

Time-to-event data and survival

10.1 Data

The `boot::melanoma` dataset was introduced in chapter 7.

In the previous session, we used logistic regression to investigate death by calculating odds ratios for different factors at a single point in time.

```
library(tidyverse)
library(broom)
library(survival)
library(survminer)
mydata = boot::melanoma

mydata$status %>%
    factor() %>%
    fct_recode("Died" = "1",
                          "Alive" = "2",
                          "Died - other causes" = "3") %>%
    fct_relevel("Alive") -> # move Alive to front (first factor level)
    mydata$status.factor    # so OR will be relative to that

mydata$sex %>%
    factor() %>%
    fct_recode("Female" = "0",
                          "Male"   = "1") ->
    mydata$sex.factor
```

```
mydata$ulcer %>%
    factor() %>%
    fct_recode("Present" = "1",
                          "Absent" = "0") ->
    mydata$ulcer.factor

mydata$age %>%
    cut(breaks = c(4,20,40,60,95), include.lowest=TRUE) ->
    mydata$age.factor
```

10.2 Kaplan-Meier survival estimator

The Kaplan-Meier (KM) survival estimator is a non-parametric statistic used to estimate the survival function from time-to-event data.

'Time' is time from event to last known status. This status could be the event, for instance death. Or could be when the patient was last seen, for instance at a clinic. In this circumstance the patient is considered 'censored'.

```
survival_object = Surv(mydata$time, mydata$status.factor == "Died")

# It is often useful to convert days into years
survival_object = Surv(mydata$time/365, mydata$status.factor == "Died")

# Investigate this:
head(survival_object) # + marks censoring in this case "Died of other cause
# Or that the follow-up ended and the patient is censored.
```

```
## [1] 0.02739726+ 0.08219178+ 0.09589041+ 0.27123288+ 0.50684932  0.55890411
```

10.2.1 KM analysis for whole cohort

10.2.2 Model

The survival object is the first step to performing univariable and multivariable survival analyses. A univariable model can then be fitted.

If you want to plot survival stratified by a single grouping variable, you can substitute "survival_object ~ 1" by "survival_object ~ factor"

```
# For all patients
my_survfit = survfit(survival_object ~ 1, data = mydata)
my_survfit # 205 patients, 57 events
```

```
## Call: survfit(formula = survival_object ~ 1, data = mydata)
##
##       n  events  median 0.95LCL 0.95UCL
##     205      57      NA      NA      NA
```

10.2.3 Life table

A life table is the tabular form of a KM plot, which you may be familiar with. It shows survival as a proportion, together with confidence limits. The whole table is shown with, `summary(my_survfit)`.

```
summary(my_survfit, times = c(0, 1, 2, 3, 4, 5))
```

```
## Call: survfit(formula = survival_object ~ 1, data = mydata)
##
##  time n.risk n.event survival std.err lower 95% CI upper 95% CI
##    0   205      0   1.000  0.0000      1.000      1.000
##    1   193      6   0.970  0.0120      0.947      0.994
##    2   183      9   0.925  0.0187      0.889      0.962
```

```
##    3   167      15   0.849  0.0255       0.800       0.900
##    4   160       6   0.818  0.0274       0.766       0.874
##    5   122       9   0.769  0.0303       0.712       0.831
```

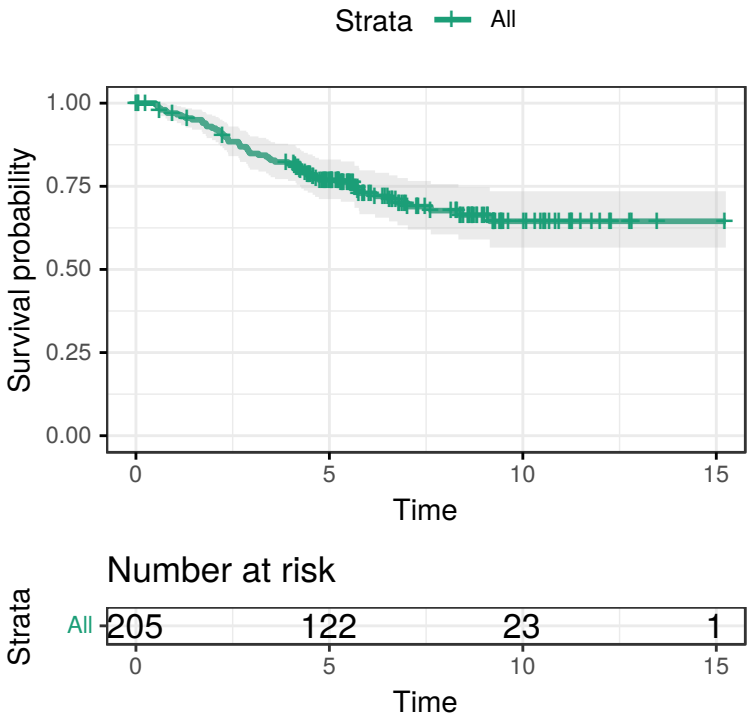
```
# 5 year survival is 77%

# Help is at hand
help(summary.survfit)
```

10.2.4 KM plot

A KM plot can easily be generated using the `survminer` package.

For more information on how the survminer package draws this plot, or how to modify it: http://www.sthda.com/english/wiki/survminer-r-package-survival-data-analysis-and-visualization and https://github.com/kassambara/survminer

```
library(survminer)
my_survplot = ggsurvplot(my_survfit, data = mydata,
           risk.table = TRUE,
           ggtheme = theme_bw(),
           palette = 'Dark2',
           conf.int = TRUE,
           pval=FALSE)
my_survplot
```

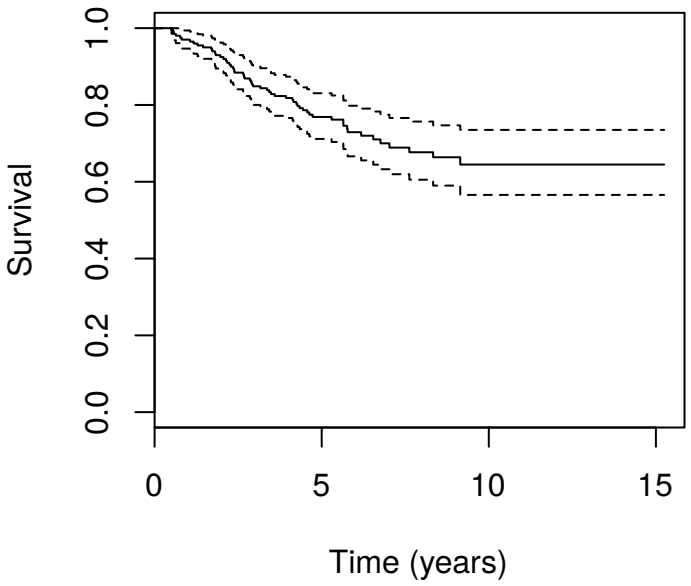



```
# Note can also take `ggplot()` options.
my_survplot$plot +
    annotate('text', x = 5, y = 0.25, label='Whole cohort')
```

Here is an alternative plot in base R to compare. Not only does this produce a more basic survival plot, but tailoring the plot can be more difficult to achieve.

Furthermore, appending a life table ('risk.table') alongside the plot can also be difficult, yet this is essential for interpretation.

```
plot(my_survfit, mark.time=FALSE, conf.int=TRUE,
         xlab="Time (years)", ylab="Survival")
```

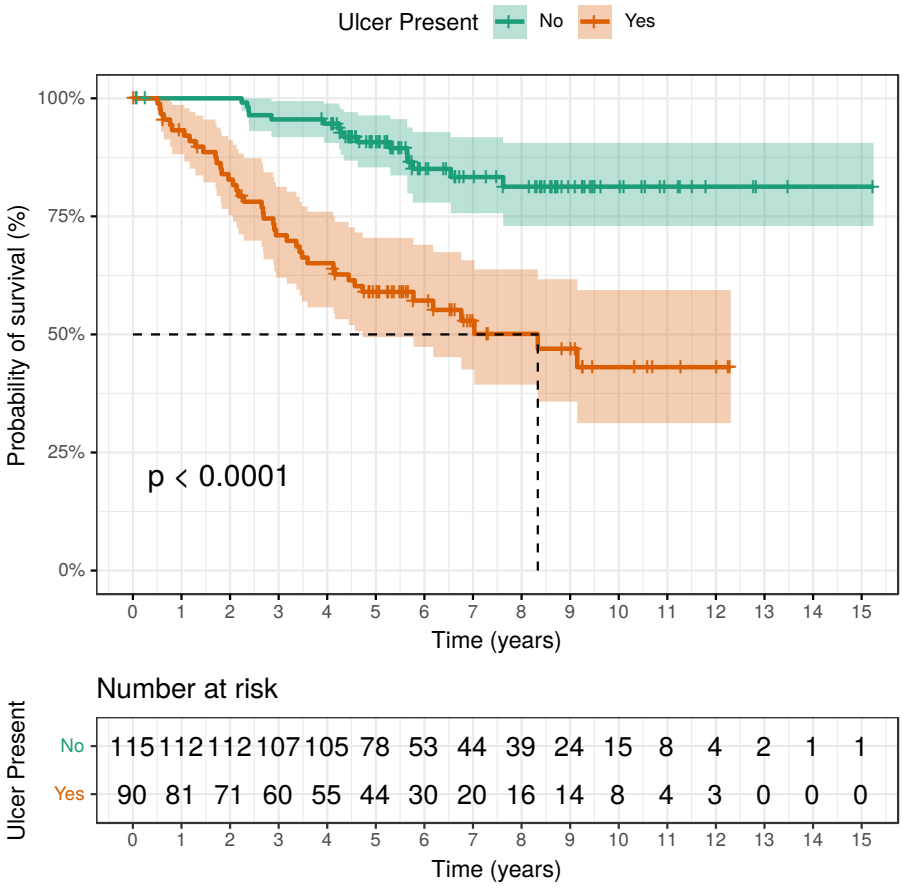


10.2.5 Exercise

Using the above scripts, perform a univariable Kaplan Meier analysis to determine if `ulcer.factor` influences overall survival. Hint: `survival_object ~ ulcer.factor`.

Try modifying the plot produced (see Help for ggsurvplot). For example:

- Add in a medial survival lines: `surv.median.line="hv"`
- Alter the plot legend: `legend.title = "Ulcer Present", legend.labs = c("No", "Yes")`
- Change the y-axis to a percentage: `ylab = "Probability of survival (%)", surv.scale = "percent"`
- Display follow-up up to 10 years, and change the scale to 1 year: `xlim = c(0,10), break.time.by = 1)`



10.2.6 Log-rank test

Two KM survival curves can be compared using the log-rank test. Note survival curves can also be compared using a Wilcoxon test that may be appropriate in some circumstances.

This can easily be performed in `library(survival)` using the function `survdiff()`.

```
survdiff(survival_object ~ ulcer.factor, data = mydata)
```

```
## Call:
## survdiff(formula = survival_object ~ ulcer.factor, data = mydata)
```

```
##
##                 N Observed Expected (O-E)^2/E (O-E)^2/V
## ulcer.factor=Absent  115     16    35.8    10.9    29.6
## ulcer.factor=Present  90     41    21.2    18.5    29.6
##
##  Chisq= 29.6  on 1 degrees of freedom, p= 5e-08
```

Is there a signficiant difference between survival curves?

10.3 Cox proportional hazard regression

10.3.1 Model

Multivariable survival analysis can be complex with parametric and semi-parametric methods available. The latter is performed using a Cox proportional hazard regression analysis.

```
# Note several variables are now introduced into the model.
# Variables should be selected carefully based on published methods.

my_hazard = coxph(survival_object~sex.factor+ulcer.factor+age.factor, data=m
summary(my_hazard)
```

```
## Call:
## coxph(formula = survival_object ~ sex.factor + ulcer.factor +
##     age.factor, data = mydata)
##
##   n= 205, number of events= 57
##
##                    coef exp(coef) se(coef)     z Pr(>|z|)
## sex.factorMale     0.48249   1.62011  0.26835  1.798   0.0722 .
## ulcer.factorPresent  1.38972   4.01372  0.29772  4.668 3.04e-06 ***
## age.factor(20,40]  -0.40628   0.66613  0.69339 -0.586   0.5579
## age.factor(40,60]  -0.04513   0.95588  0.61334 -0.074   0.9414
```

```
## age.factor(60,95]   0.17889  1.19588 0.62160  0.288  0.7735
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##                exp(coef) exp(-coef) lower .95 upper .95
## sex.factorMale      1.6201    0.6172   0.9575    2.741
## ulcer.factorPresent   4.0137    0.2491   2.2394    7.194
## age.factor(20,40]    0.6661    1.5012   0.1711    2.593
## age.factor(40,60]    0.9559    1.0462   0.2873    3.180
## age.factor(60,95]    1.1959    0.8362   0.3537    4.044
##
## Concordance= 0.735  (se = 0.04 )
## Rsquare= 0.153   (max possible= 0.937 )
## Likelihood ratio test= 34.08  on 5 df,   p=2e-06
## Wald test            = 30.19  on 5 df,   p=1e-05
## Score (logrank) test = 35.21  on 5 df,   p=1e-06
```

```
library(broom)
tidy(my_hazard)
```

```
## # A tibble: 5 x 7
##   term         estimate std.error statistic  p.value conf.low conf.high
##   <chr>           <dbl>    <dbl>    <dbl>    <dbl>    <dbl>     <dbl>
## 1 sex.factorMale   0.482    0.268    1.80    7.22e-2 -0.0435    1.01
## 2 ulcer.factorPr~  1.39     0.298    4.67    3.04e-6  0.806     1.97
## 3 age.factor(20,~ -0.406    0.693   -0.586   5.58e-1 -1.77      0.953
## 4 age.factor(40,~ -0.0451   0.613   -0.0736  9.41e-1 -1.25      1.16
## 5 age.factor(60,~  0.179    0.622    0.288   7.74e-1 -1.04      1.40
```

The interpretation of the results of model fitting are beyond the aims of this course. The exponentiated coefficient (`exp(coef)`) represents the hazard ratio. Therefore, patients with ulcers are 4-times more likely to die at any given time than those without ulcers.

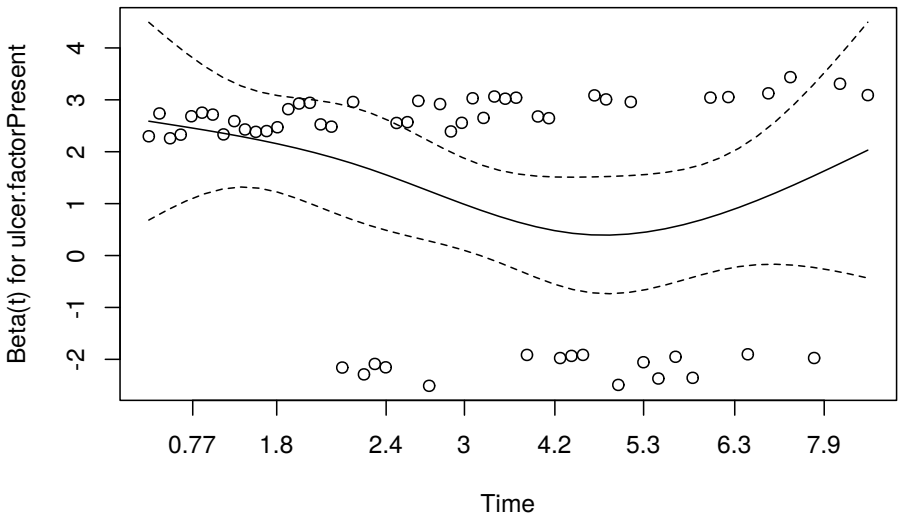
10.3.2 Assumptions

The CPH model presumes 'constant hazards'. That means that the risk associated with any given variable (like ulcer status) shouldn't get worse or better over time. This can be checked.

```
ph = cox.zph(my_hazard)
ph
```

```
##                         rho chisq      p
## sex.factorMale      -0.104 0.647 0.4212
## ulcer.factorPresent -0.238 3.135 0.0766
## age.factor(20,40]    0.110 0.716 0.3976
## age.factor(40,60]    0.194 2.222 0.1361
## age.factor(60,95]    0.146 1.257 0.2622
## GLOBAL                  NA 6.949 0.2244
```

```
# GLOBAL shows no overall violation of assumptions.
# Ulcer.status is borderline significant

# Plot Schoenfield residuals to evaluate PH
plot(ph, var=2) # ulcer.status is variable 2
```



```
# help(plot.cox.zph)
```

Hazard decreases a little between 2 and 5 years, but is acceptable.

10.3.3 Exercise

Create a new CPH model, but now include the variable `thickness` as a variable. How would you interpret the output? Is it an independent predictor of overall survival in this model? Are CPH assumptions maintained?

10.4 Dates in R

10.4.1 Converting dates to survival time

In the melanoma example dataset, we already had the time in a convenient format for survial analysis - survival time in days since the operation. This section shows how to convert dates into "days from event". First we will generate a dummy operation date and censoring date based on the melanoma data.

```
library(lubridate)
first_date = ymd("1966-01-01")           # let's create made-up dates for th
last_date = first_date + days(nrow(mydata)-1) # assume tone every day from 1
operation_date = seq(from = first_date, to = last_date, by = "1 day") # crea

mydata$operation_date = operation_date # add the created sequence to melanom
```

Now we will to create a 'censoring' date by adding `time` from the melanoma dataset to our made up operation date.

Remember the censoring date is either when an event occurred (e.g. death) or the last known alive status of the patient.

```
mydata = mydata %>%
  mutate(censoring_date = operation_date + days(time))

# (Same as doing:):
mydata$censoring_date = mydata$operation_date + days(mydata$time)
```

Now consider if we only had the `operation date` and `censoring date`. We want to create the `time` variable.

```
mydata = mydata %>%
  mutate(time_days = censoring_date - operation_date)
```

The `Surv()` function expects a number (`numeric` variable), rather than a `date` object, so we'll convert it:

```
# Surv(mydata$time_days, mydata$status==1) # this doesn't work

mydata %>%
  mutate(time_days_numeric = as.numeric(time_days))  ->
  mydata

survival_object = Surv(mydata$time_days_numeric, mydata$status.factor == "Di
```

10.5 Solutions

9.2.2

```
# Fit survival model
my_survfit.solution = survfit(survival_object ~ ulcer.factor, data = mydata)

# Show results
```

```
my_survfit.solution
summary(my_survfit.solution, times=c(0,1,2,3,4,5))

# Plot results
my_survplot.solution = ggsurvplot(my_survfit.solution,
                          data = mydata,
                          palette = 'Dark2',
                          risk.table = TRUE,
                          ggtheme = theme_bw(),
                          conf.int = TRUE,
                          pval=TRUE,

                          # Add in a medial survival line.
                          surv.median.line="hv",

                          # Alter the plot legend (change the names)
                          legend.title = "Ulcer Present",
                          legend.labs = c("No", "Yes"),

                          # Change the y-axis to a percentage
                          ylab = "Probability of survival (%)",
                          surv.scale = "percent",

                          # Display follow-up up to 10 years, and change the
                          xlab = "Time (years)",
                          # present narrower X axis, but not affect survival
                          xlim = c(0,10),
                          # break X axis in time intervals by 1 year
                          break.time.by = 1)

my_survplot.solution
```

9.3.3

```
# Fit model
my_hazard = coxph(survival_object~sex.factor+ulcer.factor+age.factor+thickne
summary(my_hazard)

# Melanoma thickness has a HR 1.12 (1.04 to 1.21).
# This is interpretted as a 12% increase in the
# risk of death at any time for each 1 mm increase in thickness.

# Check assumptions
ph = cox.zph(my_hazard)
ph
# GLOBAL shows no overall violation of assumptions.
# Plot Schoenfield residuals to evaluate PH
plot(ph, var=6)
```

Part III

Workflow

11

Notebooks and markdown

12

Missing data

13

Encryption

14

Exporting tables and plots

15

RStudio settings, good practise

15.1 Starting with a blank canvas

In the first session we loaded some data that we then plotted. When we import data, R stores it and displays it in the Environment tab.

It's good practice to restart R before commencing new work. This is to avoid accidentally using the wrong data or functions stored in the environment.
Restarting R only takes a second!

- Restart R (Ctrl+Shift+F10 or select it from Session -> Restart R).

RStudio has a default setting that is no longer considered best practice. You should do this once:

- Go to Tools -> Global Options -> General and set "Save .RData on exit" to Never. This does not mean you can't or shouldn't save your work in .RData files. But it is best to do it consciously and load exactly what you need to load, rather than letting R always save and load everything for you, as this could also include broken data or objects.

Bibliography

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2018). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.7.15.

Index