Introduction to Power BI

Contents

Introduction

These are course notes for the "Introduction to Power BI" course given by the Monash Bioinformatics Platform[1] for the Monash Data Fluency[2] initiative. Our teaching style is based on the style of The Carpentries[3].

- PDF version for printing[4]
- ZIP of data files used in this workshop[5]

During this workshop we will be using Power BI Desktop installed on your computer. There are several ways to download Power BI Desktop, depending on which system you use.

1. Windows User

- Power BI website You can download Power BI Desktop from the website and install it as an application on your computer. Monash machine, My software, contact eSolutions to gain admin access(link to eSolution)

After the setup process, you will be able to see the following Start Screen.

- Windows Store Or you can visit Windows Store to get the Power BI Desktop app and install it. Note that the system requirements is Windows 10 version 14393.0 or higher.

- Power BI service You can also download it from the Power BI service by clicking the download button in the upper right and selecting Power BI Desktop. To use Power BI service, you may need a Microsoft account.

2. Mac User

Power BI Desktop is not available on Macs. There are two main options. Dual BootCamp The first is to run a Windows session on your Mac via BootCamp

[1]https://www.monash.edu/researchinfrastructure/bioinformatics
[2]https://monashdatafluency.github.io/
[3]https://carpentries.org/
[4]https://monashdatafluency.github.io/powerbi/powerbi-intro.pdf
[5]https://monashdatafluency.github.io/powerbi-intro/powerbi-files.zip

or something similar. This is probably a longer term solution until Microsoft release a Mac version. MoVE: TODO

After installing Power BI Desktop, you can sign up for Power BI using your Monash account here By signing in the Power BI Desktop, you will be able to save your work and later publish it to the Power BI service.

3. Linux Users

TODO

Data

Please download the data file here for the course.

Source code

This book was created in R using the `rmarkdown` and `bookdown` packages!

- GitHub page[6]

Authors and copyright

This course is developed for the Monash Data Fluency Team.

[6]https://github.com/MonashDataFluency/r-intro-2
[7]http://creativecommons.org/licenses/by/4.0/
[8]https://www.gapminder.org

Chapter 1

Introduction to PowerBI

1.1 Overview of Power BI

Microsoft Power BI is a collection of apps, software services and connectors that come together to turn the unrelated data into visually impressive and interactive insights. Power BI can work with the simplest of data sources like Microsoft Excel and the more complicated ones like a collection of cloud-based or on-premises hybrid Data warehouses. Power BI has the capabilities to easily connect to your data sources, visualise and share and publish your findings with anyone and everyone.

Power BI can be simple and fast enough to connect to an Excel workbook or a local database or it can be robust and enterprise-grade, ready for extensive modeling and real time analytics and also for custom development. Hence, it can be a personal report and vis tool but can also act as the analytics and decision engine behind group projects, divisions, or entire corporations.

1.2 The parts of Power BI

Power BI constitutes of a Microsoft Windows desktop application called Power BI Desktop, an online SaaS (Software as a Service) called Power BI Service and a mobile Power BI apps that can be accessed from Windows phones and tablets, and are also available on Apple iOS and Google Android devices.

These three elements— **Desktop**, the **service**, and **Mobile** apps - are the backbone of the Power BI system and lets users create, share and consume the actionable insights in the most effective way.

1.3 Use of Power BI and roles

The use of Power BI could depend a lot on the role that you are in. For example: if you are the stakeholder of a project, then you might want to use **Power BI**

Service or the Mobile **app** to have a glance at how the business is performing. But on the other hand, if you are a developer, you would be using **Power BI Desktop** extensively and then publish Power BI desktop reports to the Power BI Service.

In the upcoming modules we would be discussing about these three components - **Desktop**, **Service** and **Mobile** apps - in more detail.

1.4 Power BI Flow

In the most general way, the flow starts at the Power BI Desktop, where a report is created. This created report can be published to the Power BI Service and finally shared so that the users can use it from the Mobile apps.

Its not always the case that this flow happens, but more often or not it is. We will stick to this flow for this entire tutorial to help learn the different aspects of Power BI.

1.5 Use Power BI:

The **common** flow of activity in Power BI looks like this: 1. Bring data into Power BI Desktop, and create a report. 2. Publish to the Power BI service, where you can create new visualizations or build dashboards. 3. Share dashboards with others, especially people who are on the go. 4. View and interact with shared dashboards and reports in Power BI Mobile apps.

As mentioned earlier, depending on the user role, the user might spend its most of the time in one of the three components than the other.

1.6 Building blocks of Power BI:

The basic building blocks in Power BI are: * Visualizations

- Datasets
- Reports
- Dashboards
- Tiles

1.6.1 Visualizations

A visualization is a representation of data in a visual format. It could be a line chart, a bar graph, a color coded map or anything interesting to present the data.

–Picture of a final visualisation–

Visualizations can be simple as a number representing something significant or it could be quite complex like multiple stacked chart showing the proportion users participating in a survey. The prime idea of visualisation is to show the data in a way that it tells the story that's lying underneath it. Like its said, a picture says a thousand words.

1.6.2 Datasets:

A **dataset** is a collection of data that Power BI uses to create its visualizations. You can have a simple dataset that's based on a single table from a Microsoft Excel workbook, similar to what's shown in the following image.

–Picture of Dataset–

Dataset can also be a combination of many different sources, which can be filtered using Power BI and combine into one to use.

For eg: One of the data could be the countries and its central location in the form of Latitude and Longitude and other data could be the demographics of the countries like, population, GDP etc. Power BI can combine these two data and make one dataset out of it to be used for visualizations.

An important feature of Power BI is the ability of it to connect to various data sources using its connectors. Whether the data you want is in Excel or a Microsoft SQL Server database, in Azure or Oracle, or in a service like Facebook, Salesforce, or MailChimp, Power BI has built-in data connectors that let you easily connect to that data, filter it if necessary, and bring it into your dataset.

After you have a dataset, you can begin creating visualizations that show different portions of it in different ways, and gain insights based on what you see. That's where reports come in.

1.6.3 Reports:

In Power BI, a **report** is a collection of visualizations that appear together on one or more pages. A report in Power BI is a collection of items that are related to each other. The following image shows a report that you would be creating by the end of the session. You can also create reports in the Power BI service.

– Picture of report–

Reports let us create many visualizations and possibly on multiple pages based on the way the developer wants to tell the story.

1.6.4 Dashboards:

A Power BI dashboard is a collection of visuals from a single page that you can share with others. Often, it's a selected group of visuals that provide quick insight into the data or story you're trying to present.

A dashboard must fit on a single page, often called a canvas (the canvas is the blank backdrop in Power BI Desktop or the service, where you put visualizations). Think of it like the canvas that an artist or painter uses—a workspace where you create, combine, and rework interesting and compelling visuals. You can share dashboards with other users or groups, who can then interact with your dashboards when they're in the Power BI service or on their mobile device.

1.6.5 Tiles:

TODO

1.7 Power BI Services:

1.7.1 Overview of Power BI Desktop

Power BI Desktop is a free application for PCs that lets you gather, transform, and visualize your data. In this module, you'll learn how to find and collect data from different sources and how to clean or transform it. You'll also learn tricks to make data-gathering easier. Power BI Desktop and the Power BI Service work together. You can create your reports and dashboards in Power BI Desktop, and then publish them to the Power BI Service for others to consume.

–Picture of desktop view–

1. **Ribbon** - Displays common tasks that are associated with reports and visualizations.
2. **Report view, or canvas** - Where visualizations are created and arranged. You can switch between **Report**, **Data**, and **Model** views by selecting the icons in the left column.
3. **Pages tab** - Located along the bottom of the page, this area is where you would select or add a report page.
4. **Visualizations pane** - Where you can change visualizations, customize colors or axes, apply filters, drag fields, and more.
5. **Fields pane** - Where query elements and filters can be dragged onto the **Report** view or dragged to the **Filters** area of the Visualizations pane.

Chapter 2

Import dataset and modelling (preprocessing)

Power BI can connect to a whole range of data sources, right from Excel sheets, Local databases to several Cloud services. Currently, over 60 different cloud services have specific connectors to help you connect with generic sources through XML, CSV, text, and ODBC. Let us start connecting to one of the data sources. For today we will be working on `gap_minder_map.csv` file.

2.1 Importing data into Power BI Desktop

Power BI Desktop has a "**Get Data**" button from the ribbon on the "**Home**" tab. In Power BI, there are all sorts of different data sources available. Select a source to establish a connection. Depending on your selection, you will be asked to find the source on your computer or network, or be prompted to sign in to a service to authenticate your request.

As our first step to import the dataset/file into Power BI, we click on the Get Data icon on the ribbon of Home tab.

–picture–

Once we select this, we go ahead and select the "**CSV option**" under the "**file**" subheading.

–picture–

Then browse the file and select the necessary CSV file. Press on "**Connect**" to have a quick preview of the file. Once we click on "**load**", Power BI will successfully import the file. Any errors will then pop up ready to be analysed and fixed.

–picture–

–picture–

Clicking on "**View errors**" will enable us to check the detected errors right away. Alternatively, you close the pop up and click "**Edit queries**" to check for any errors. This will query and list the errors in the data.

2.2 Dealing with errors

2.2.1 Check for "controlled" errors

Once queried, we can see the controlled errors in the data. These errors can be analysed by clicking on them. Every error has a brief description to it saying what might've gone wrong.

2.2.2 Change datatype of column

One of the most common errors is the detection of the datatype of a particular column. While loading the data, Power BI guesses the column type based on the data it sees. If needed the datatypes of the columns can be changed to something relevant. Double click on the "**datatype**" icon on top of the column, select the "**datatype**" and click on "**Replace current step**".

Any changes to the data needs to be done under the "**Data file**" listed under "**Other Queries**". Once necessary changes are made, it is important to refresh and check if the change was applied. We can do this by clicking on the "**Refresh Preview**" button on the ribbon.

Once all the required changes are done, we can close and apply going back to the main Power BI Desktop interface.

Make sure you see all the column names in the data on the bottom right corner.

2.2.3 Replacing null values

Data can have missing values for a number of reasons. This missing data is represented as `null` in the data. A lot of times it is important to deal with such values and fix or remove them.

In the "**Data**" section on the left sidebar, data can be viewed. By applying a filter to a particular column, the null values can be analysed. To replace any null values, we can go back to the "**Query Editor**" and use the "Replace values" option.

Replacing missing values is not always a direct operation. Most of the times, the missing values must be carefully analyzed and values need to be computed based on several factors. This computational procedure can be programmed with DAX. Let us cancel this particular step in the "Applied steps" by clicking on the red crossmark right next to "**Replaced Value**"'.

2.2.4 Challenge 1: Replace missing values

2.3 'Applied steps' in modelling data

2.3.1 Renaming columns

Columns from raw data can be difficult to read or meaningless. Renaming the columns in your query to a meaningful name will make it easier for you and your audience to understand your data. This will often save you trouble in the future when it comes to working and presenting the data.

There are two ways to rename the columns in Power BI. Right-clicking on the header of the column gives you a menu of functions that you can do to the column. Select "**Rename**" to rename the column. You can also click on the column and then click on the "**Transform**" tab, from here you are presented a variety of transformation functions for the column. From here you can click on "**Rename**".

Example: - Right-click on the "**name**" column header, click on "**Rename**" and rename the column to "**Country**".

- Click on the header for "**life_exp**"'. Click on the "**Transform**" tab and click rename. Rename the column to "**Life expectancy**".

2.3.2 Add and remove column

2.3.2.1 Removing columns

Often when dealing with raw data you will find columns that are meaningless or unsuitable for your analysis. You can remove these columns in Query to eliminate clutter and streamline the data set making it easier to work with.

There are two ways to remove columns in Power BI. Like in renaming a column, you can right-click the header, which will present a menu with the option to "**Remove**"'. You can also click on the column, click on the "**Home**" tab and then click "**Remove Columns**".

Example: You may not need the `g77` and `oecd` information. - Right-click on the `g77` column header, click on "**Remove**"

- Left-click on `oecd` column header, click on "**Home**" tab, click "**Remove Column**"

2.3.2.2 Adding columns

Just as you would remove unsuitable data from your queries, you may need to add new columns to your data. There are a variety of options in Power BI to add different columns. Click on the "**Add Column**" tab to see the ways you can add a column.

Example: - Add or subtract a year to the `year` column to fix any widespread issues.

- Click on `gdp_percap`, click on the "**Add Column**" tab, click duplicate to create a separate column to run calculations on.

2.3.3 Challenge 2:

perform simple mathematical operation and add this as a new column Let's say you want to calculate total Gross Domestic Products (GDP) per country. Total GDP = Population x GDP per capita. Working with the new `gdp_percap` column you created in the previous example, create a new column with Total GDP per country.

Chapter 3

Plotting with ggplot2

We already saw some of R's built in plotting facilities with the function `plot`. A more recent and much more powerful plotting library is `ggplot2`. `ggplot2` is another mini-language within R, a language for creating plots. It implements ideas from a book called "The Grammar of Graphics"[1]. The syntax can be a little strange, but there are plenty of examples in the online documentation[2].

`ggplot2` is part of the Tidyverse, so loadinging the `tidyverse` package will load `ggplot2`.

```
library(tidyverse)
```

We continue with the Gapminder dataset, which we loaded with:

```
geo <- read_csv("r-intro-2-files/geo.csv")
gap <- read_csv("r-intro-2-files/gap-minder.csv")
gap_geo <- left_join(gap, geo, by="name")
```

3.1 Elements of a ggplot

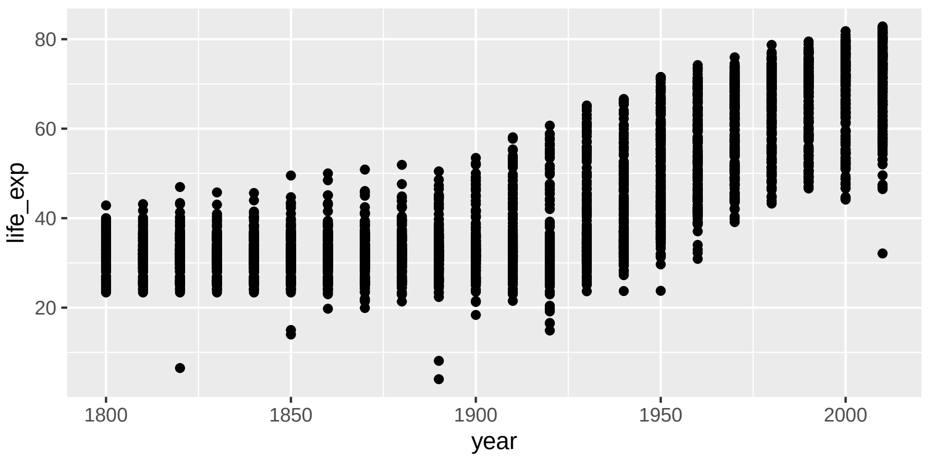
Producing a plot with `ggplot2`, we must give three things:

1. A data frame containing our data.
2. How the columns of the data frame can be translated into positions, colors, sizes, and shapes of graphical elements ("aesthetics").
3. The actual graphical elements to display ("geometric objects").

Let's make our first ggplot.

[1]https://www.amazon.com/Grammar-Graphics-Statistics-Computing/dp/0387245448
[2]http://ggplot2.tidyverse.org/reference/

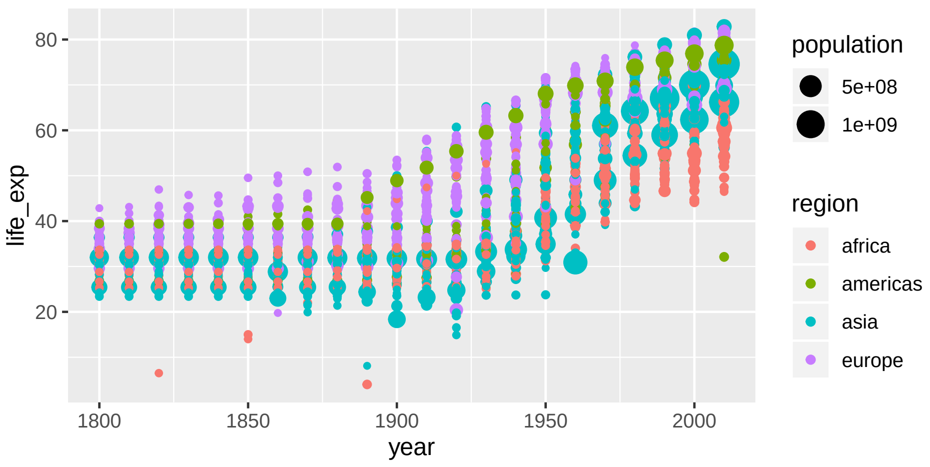
```r
ggplot(gap_geo, aes(x=year, y=life_exp)) +
    geom_point()
```



The call to `ggplot` and `aes` sets up the basics of how we are going to represent the various columns of the data frame. `aes` defines the "aesthetics", which is how columns of the data frame map to graphical attributes such as x and y position, color, size, etc. `aes` is another example of magic "non-standard evaluation", arguments to `aes` may refer to columns of the data frame directly. We then literally add layers of graphics ("geoms") to this.

Further aesthetics can be used. Any aesthetic can be either numeric or categorical, an appropriate scale will be used.

```r
ggplot(gap_geo, aes(x=year, y=life_exp, color=region, size=population)) +
    geom_point()
```



3.1.1 Challenge: make a ggplot

This R code will get the data from the year 2010:

```
gap2010 <- filter(gap_geo, year == 2010)
```

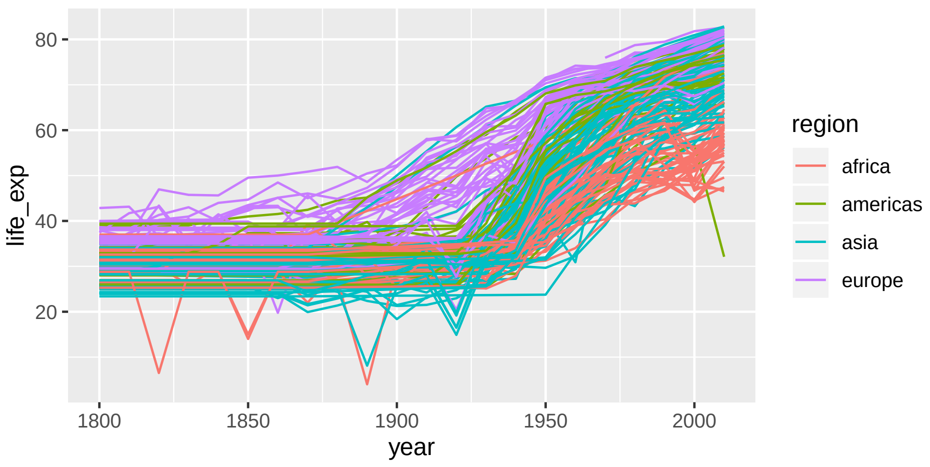
Create a ggplot of this with:

- `gdp_percap` as x.
- `life_exp` as y.
- `population` as the size.
- `region` as the color.

3.2 Further geoms

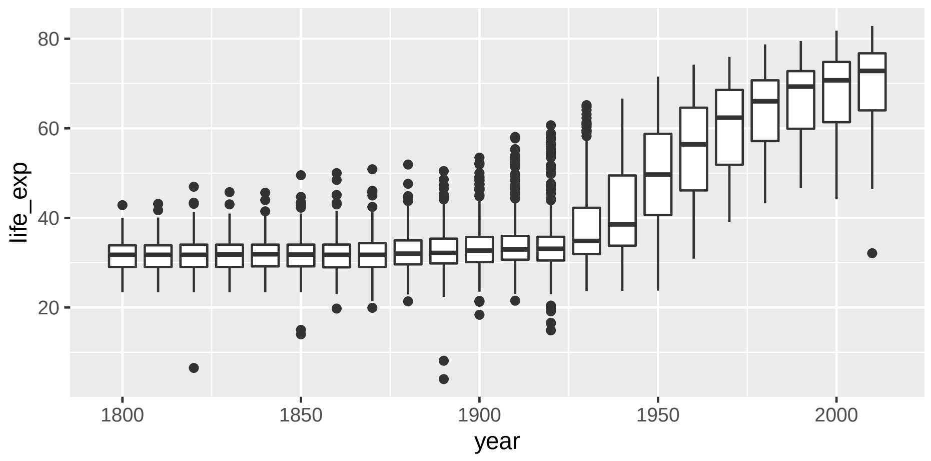
To draw lines, we need to use a "group" aesthetic.

```
ggplot(gap_geo, aes(x=year, y=life_exp, group=name, color=region)) +
    geom_line()
```



A wide variety of geoms are available. Here we show Tukey box-plots. Note again the use of the "group" aesthetic, without this ggplot will just show one big box-plot.

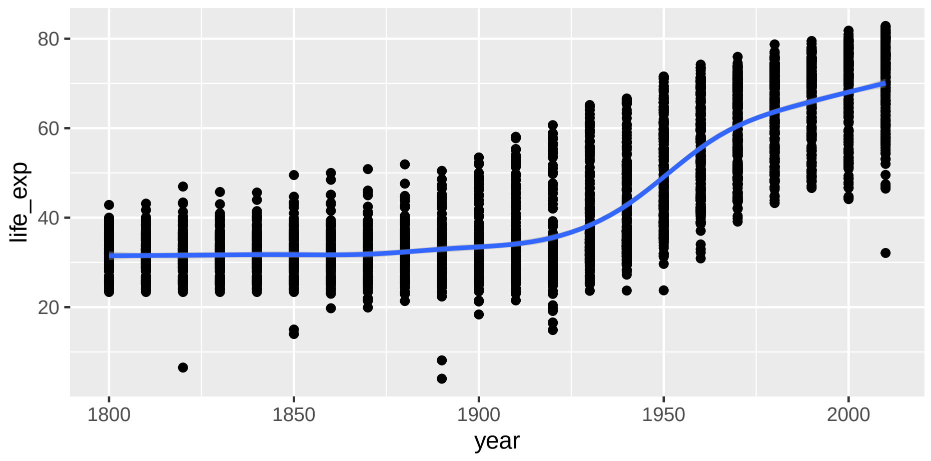
```
ggplot(gap_geo, aes(x=year, y=life_exp, group=year)) +
    geom_boxplot()
```



`geom_smooth` can be used to show trends.

```
ggplot(gap_geo, aes(x=year, y=life_exp)) +
    geom_point() +
    geom_smooth()
```

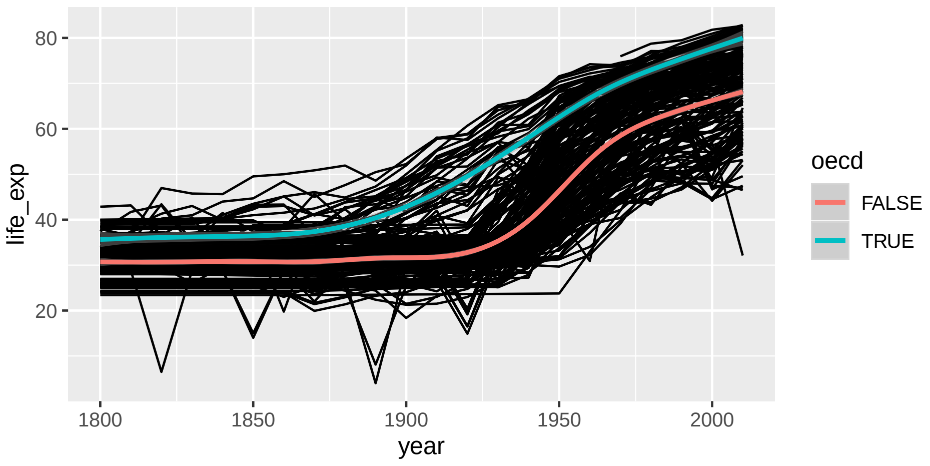
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Aesthetics can be specified globally in `ggplot`, or as the first argument to individual geoms. Here, the "group" is applied only to draw the lines, and "color" is used to produce multiple trend lines:

```
ggplot(gap_geo, aes(x=year, y=life_exp)) +
    geom_line(aes(group=name)) +
    geom_smooth(aes(color=oecd))
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

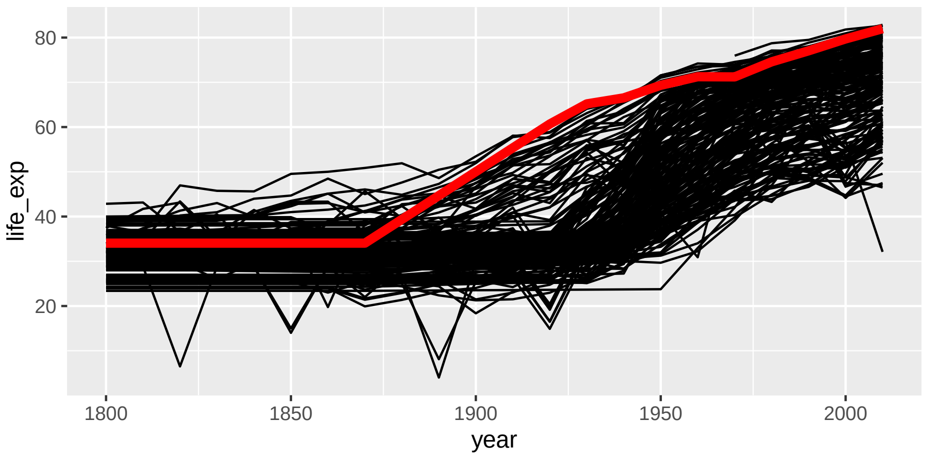


3.3 Highlighting subsets

Geoms can be added that use a different data frame, using the `data=` argument.

```r
gap_australia <- filter(gap_geo, name == "Australia")

ggplot(gap_geo, aes(x=year, y=life_exp, group=name)) +
    geom_line() +
    geom_line(data=gap_australia, color="red", size=2)
```

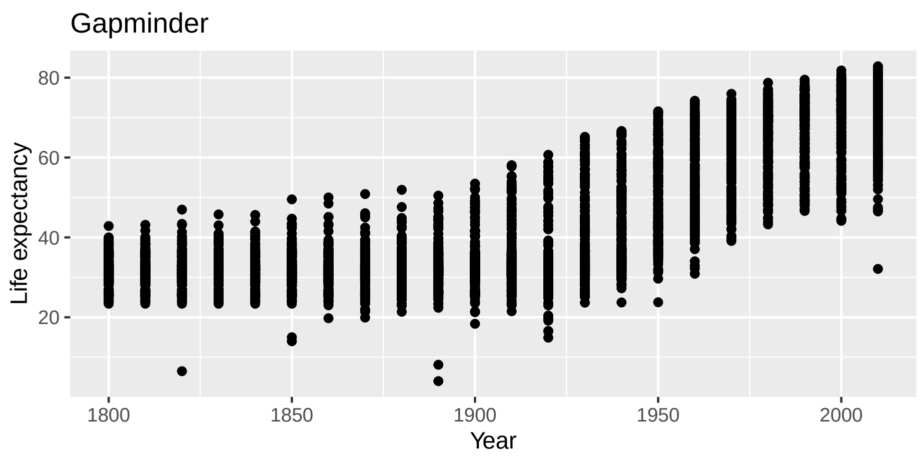


Notice also that the second `geom_line` has some further arguments controlling its appearance. These are **not** aesthetics, they are not a mapping of data to appearance, but rather a direct specification of the appearance. There isn't an associated scale as when color was an aesthetic.

3.4 Fine-tuning a plot

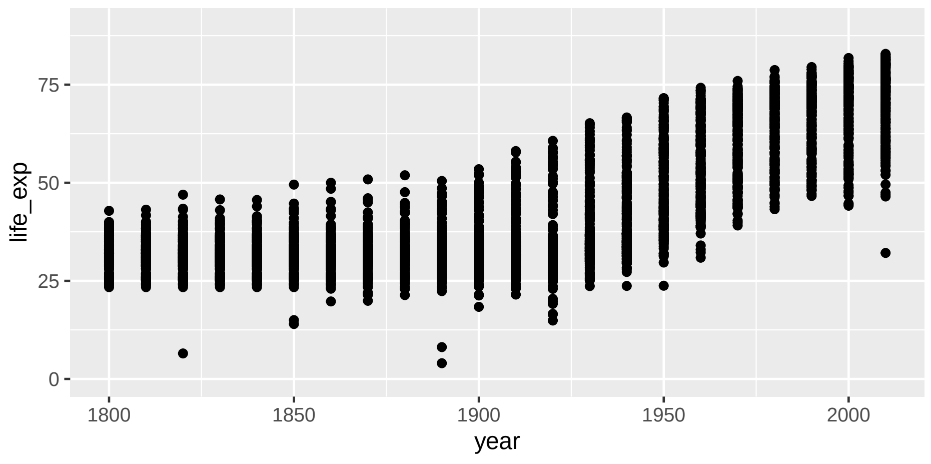
Adding `labs` to a ggplot adjusts the labels given to the axes and legends. A plot title can also be specified.

```
ggplot(gap_geo, aes(x=year, y=life_exp)) +
    geom_point() +
    labs(x="Year", y="Life expectancy", title="Gapminder")
```



`coord_cartesian` can be used to set the limits of the x and y axes. Suppose we want our y-axis to start at zero.

```
ggplot(gap_geo, aes(x=year, y=life_exp)) +
    geom_point() +
    coord_cartesian(ylim=c(0,90))
```



Type `scale_` and press the tab key. You will see functions giving fine-grained controls over various scales (x, y, color, etc). These allow transformations (eg log10), and manually specified breaks (labelled values). Very fine grained control is possible over the appearance of ggplots, see the ggplot2 documentation for details and further examples.

3.4.1 Challenge: refine your ggplot

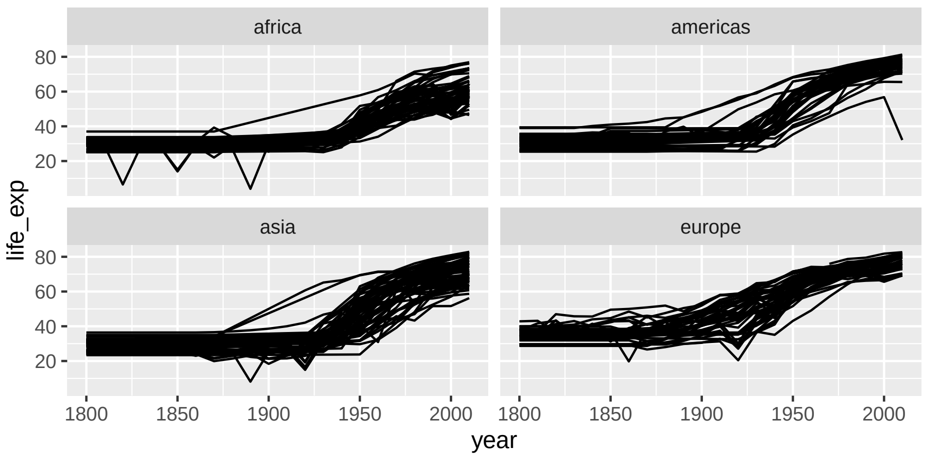
Continuing with your scatter-plot of the 2010 data, add axis labels to your plot.

Give your x axis a log scale by adding `scale_x_log10()`.

3.5 Faceting

Faceting lets us quickly produce a collection of small plots. The plots all have the same scales and the eye can easily compare them.

```r
ggplot(gap_geo, aes(x=year, y=life_exp, group=name)) +
    geom_line() +
    facet_wrap(~ region)
```



Note the use of `~`, which we've not seen before. `~` syntax is used in R to specify dependence on some set of variables, for example when specifying a linear model. Here the information in each plot is dependent on the continent.

3.5.1 Challenge: facet your ggplot

Let's return again to your scatter-plot of the 2010 data.

Adjust your plot to now show data from all years, with each year shown in a separate facet, using `facet_wrap(~ year)`.

Advanced: Highlight Australia in your plot.

3.6 Saving ggplots

The act of plotting a ggplot is actually triggered when it is printed. In an interactive session we are automatically printing each value we calculate, but if you are using it with a programming construct such as a for loop or function you might need to explcitly `print( )` the plot.

Ggplots can be saved using `ggsave`.

```
# Plot created but not shown.
p <- ggplot(gap_geo, aes(x=year, y=life_exp)) + geom_point()

# Only when we try to look at the value p is it shown
p

# Alternatively, we can explicitly print it
print(p)

# To save to a file
ggsave("test.png", p)


# This is an alternative method that works with "base R" plots as well:
png("test.png")
print(p)
dev.off()
```

3.6.1 Tip about sizing

Figures in papers tend to be quite small. This means text must be proportionately larger than we usually show on screen. Dots should also be proportionately larger, and lines proportionately thicker. The way to achieve this using `ggsave` is to specify a small width and height, given in inches. To ensure the output also has good resolution, specify a high dots-per-inch, or use a vector-graphics format such as PDF or SVG.

```
ggsave("test2.png", p, width=3, height=3, dpi=600)
```

Chapter 4

Summarizing data

Having loaded and thoroughly explored a data set, we are ready to distill it down to concise conclusions. At its simplest, this involves calculating summary statistics like counts, means, and standard deviations. Beyond this is the fitting of models, and hypothesis testing and confidence interval calculation. R has a huge number of packages devoted to these tasks and this is a large part of its appeal, but is beyond the scope of today.

Loading the data as before, if you have not already done so:

```r
library(tidyverse)

geo <- read_csv("r-intro-2-files/geo.csv")
gap <- read_csv("r-intro-2-files/gap-minder.csv")
gap_geo <- left_join(gap, geo, by="name")
```

4.1 Summary functions

R has a variety of functions for summarizing a vector, including: `sum`, `mean`, `min`, `max`, `median`, `sd`.

```r
mean( c(1,2,3,4) )
```

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

We can use these on the Gapminder data.

```r
gap2010 <- filter(gap_geo, year == 2010)
sum(gap2010$population)
```

```
## [1] 6949495061
```

```
mean(gap2010$life_exp)
```

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

4.2 Missing values

Why did `mean` fail? The reason is that `life_exp` contains missing values (`NA`).

```
gap2010$life_exp
```

```
##   [1] 56.20 76.31 76.55 82.66 60.08 76.85 75.82 73.34 81.98 80.50 69.13
##  [12] 73.79 76.03 70.39 76.68 70.43 79.98 71.38 61.82 72.13 71.64 76.75
##  [23] 57.06 74.19 77.08 73.86 57.89 57.73 66.12 57.25 81.29 72.45 47.48
##  [34] 56.49 79.12 74.59 76.44 65.93 57.53 60.43 80.40 56.34 76.33 78.39
##  [45] 79.88 77.47 79.49 63.69 73.04 74.60 76.72 70.52 74.11 60.93 61.66
##  [56] 76.00 61.30 65.28 80.00 81.42 62.86 65.55 72.82 80.09 62.16 80.41
##  [67] 71.34 71.25 57.99 55.65 65.49 32.11 71.58 82.61 74.52 82.03 66.20
##  [78] 69.90 74.45 67.24 80.38 81.42 81.69 74.66 82.85 75.78 68.37 62.76
##  [89] 60.73 70.10 80.13 78.20 68.45 63.80 73.06 79.85 46.50 60.77 76.10
## [100]    NA 73.17 81.35 74.01 60.84 53.07 74.46 77.91 59.46 80.28 63.72
## [111] 68.23 73.42 75.47 65.38 69.74    NA 66.18 76.36 73.55 54.48 66.84
## [122] 58.60    NA 68.26 80.73 80.90 77.36 58.78 60.53 81.04 76.09 65.33
## [133]    NA 77.85 58.70 74.07 77.92 69.03 76.30 79.84 79.52 73.66 69.24
## [144] 64.59    NA 75.48 71.64 71.46    NA 68.91 75.13 64.01 74.65 73.38
## [155] 55.05 82.69 75.52 79.45 61.71 53.13 54.27 81.94 74.42 66.29 70.32
## [166] 46.98 81.52 82.21 76.15 79.19 69.61 59.30 76.57 71.10 58.74 69.86
## [177] 72.56 76.89 78.21 67.94    NA 56.81 70.41 76.51 80.34 78.74 76.36
## [188] 68.77 63.02 75.41 72.27 73.07 67.51 52.02 49.57 58.13
```

R will not ignore these unless we explicitly tell it to with `na.rm=TRUE`.

```
mean(gap2010$life_exp, na.rm=TRUE)
```

```
## [1] 70.34005
```

Ideally we should also use `weighted.mean` here, to take population into account.

```
weighted.mean(gap2010$life_exp, gap2010$population, na.rm=TRUE)
```

```
## [1] 70.96192
```

`NA` is a special value. If we try to calculate with `NA`, the result is `NA`

```
NA + 1
```

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

`is.na` can be used to detect `NA` values, or `na.omit` can be used to directly remove rows of a data frame containing them.

```
is.na( c(1,2,NA,3) )
```

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

```
cleaned <- filter(gap2010, !is.na(life_exp))
weighted.mean(cleaned$life_exp, cleaned$population)
```

```
## [1] 70.96192
```

4.3 Grouped summaries

The `summarize` function in `dplyr` allows summary functions to be applied to data frames.

```
summarize(gap2010, mean_life_exp=weighted.mean(life_exp, population, na.rm=TRUE))
```

```
## # A tibble: 1 x 1
##   mean_life_exp
##           <dbl>
## 1          71.0
```

So far unremarkable, but `summarize` comes into its own when the `group_by` "adjective" is used.

```
summarize(
    group_by(gap_geo, year),
    mean_life_exp=weighted.mean(life_exp, population, na.rm=TRUE))
```

```
## # A tibble: 22 x 2
##     year mean_life_exp
##    <dbl>         <dbl>
##  1  1800          30.9
##  2  1810          31.1
##  3  1820          31.2
##  4  1830          31.4
##  5  1840          31.4
##  6  1850          31.6
##  7  1860          30.3
##  8  1870          31.5
##  9  1880          32.0
## 10  1890          32.5
## # ... with 12 more rows
```

Challenge: summarizing

What is the total population for each year? Plot the result.

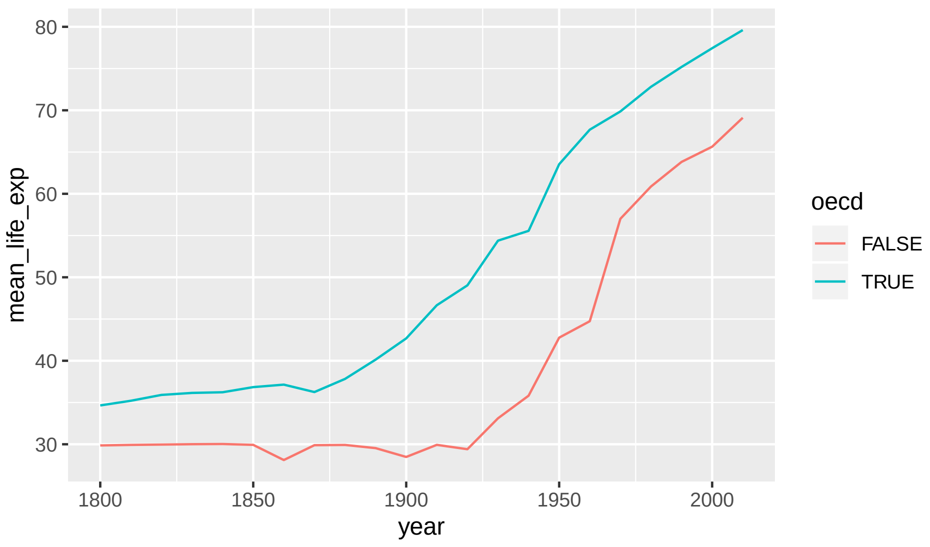
Advanced: What is the total GDP for each year? For this you will first need to calculate GDP per capita times the population of each country.

`group_by` can be used to group by multiple columns, much like `count`. We can use this to see how the rest of the world is catching up to OECD nations in terms of life expectancy.

```
result <- summarize(
    group_by(gap_geo,year,oecd),
    mean_life_exp=weighted.mean(life_exp, population, na.rm=TRUE))
result
```

```
## # A tibble: 44 x 3
## # Groups:   year [22]
##     year oecd  mean_life_exp
##    <dbl> <lgl>         <dbl>
##  1  1800 FALSE          29.9
##  2  1800 TRUE           34.7
##  3  1810 FALSE          29.9
##  4  1810 TRUE           35.2
##  5  1820 FALSE          30.0
##  6  1820 TRUE           35.9
##  7  1830 FALSE          30.0
##  8  1830 TRUE           36.2
##  9  1840 FALSE          30.0
## 10  1840 TRUE           36.2
## # ... with 34 more rows
```

```
ggplot(result, aes(x=year,y=mean_life_exp,color=oecd)) + geom_line()
```



A similar plot could be produced using `geom_smooth`. Differences here are that we have full control over the summarization process so we were able to use the exact summarization method we want (`weighted.mean` for each year), and we have access to the resulting numeric data as well as the plot. We have reduced a large data set down to a smaller one that distills out one of the stories present in this data. However the earlier visualization and exploration activity using `ggplot2` was essential. It gave us an idea of what sort of variability was present in the data, and any unexpected issues the data might have.

4.4 t-test

We will finish this section by demonstrating a t-test. The main point of this section is to give a flavour of how statistical tests work in R, rather than the details of what a t-test does.

Has life expectancy increased from 2000 to 2010?

```
gap2000 <- filter(gap_geo, year == 2000)
gap2010 <- filter(gap_geo, year == 2010)

t.test(gap2010$life_exp, gap2000$life_exp)
```

```
##
##  Welch Two Sample t-test
##
## data:  gap2010$life_exp and gap2000$life_exp
## t = 3.0341, df = 374.98, p-value = 0.002581
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1.023455 4.792947
## sample estimates:
## mean of x mean of y
##  70.34005  67.43185
```

Statistical routines often have many ways to tweak the details of their operation. These are specified by further arguments to the function call, to override the default behaviour. By default, `t.test` performs an unpaired t-test, but these are repeated observations of the same countries. We can specify `paired=TRUE` to `t.test` to perform a paired sample t-test and gain some statistical power. Check this by looking at the help page with `?t.test`.

It's important to first check that both data frames are in the same order.

```
all(gap2000$name == gap2010$name)
```

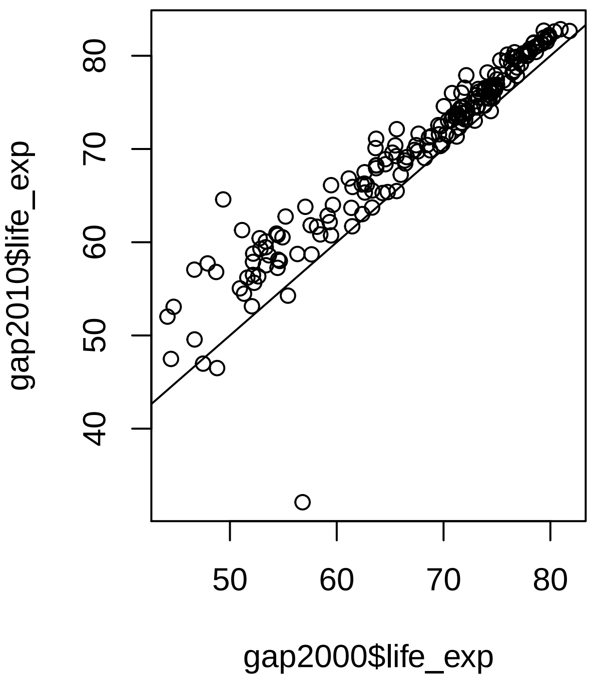
```
## [1] TRUE
```

```
t.test(gap2010$life_exp, gap2000$life_exp, paired=TRUE)
```

```
##
##  Paired t-test
##
## data:  gap2010$life_exp and gap2000$life_exp
## t = 13.371, df = 188, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  2.479153 3.337249
## sample estimates:
## mean of the differences
##                2.908201
```

When performing a statistical test, it's good practice to visualize the data to make sure there is nothing funny going on.

```
plot(gap2000$life_exp, gap2010$life_exp)
abline(0,1)
```



This is a visual confirmation of the t-test result. If there were no difference between the years then points would lie approximately evenly above and below the diagonal line, which is clearly not the case. However the outlier may warrant investigation.

Chapter 5

Thinking in R

The result of a t-test is actually a value we can manipulate further. Two functions help us here. `class` gives the "public face" of a value, and `typeof` gives its underlying type, the way R thinks of it internally. For example numbers are "numeric" and have some representation in computer memory, either "integer" for whole numbers only, or "double" which can hold fractional numbers (stored in memory in a base-2 version of scientific notation).

```
class(42)
```

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

```
typeof(42)
```

```
## [1] "double"
```

Let's look at the result of a t-test:

```
result <- t.test(gap2010$life_exp, gap2000$life_exp, paired=TRUE)

class(result)
```

```
## [1] "htest"
```

```
typeof(result)
```

```
## [1] "list"
```

```
names(result)
```

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

```
result$p.value
```

```
## [1] 4.301261e-29
```

In R, a t-test is just another function returning just another type of data, so it can also be a building block. The value it returns is a special type of vector called a “list”, but with a public face that presents itself nicely. This is a common pattern in R. Besides printing to the console nicely, this public face may alter the behaviour of generic functions such as `plot` and `summary`.

Similarly a data frame is a list of vectors that is able to present itself nicely.

5.1 Lists

Lists are vectors that can hold anything as elements (even other lists!). It’s possible to create lists with the `list` function. This becomes especially useful once you get into the programming side of R. For example writing your own function that needs to return multiple values, it could do so in the form of a list.

```
mylist <- list(hello=c("Hello","world"), numbers=c(1,2,3,4))
mylist
```

```
## $hello
## [1] "Hello" "world"
##
## $numbers
## [1] 1 2 3 4
```

```
class(mylist)
```

```
## [1] "list"
```

```
typeof(mylist)
```

```
## [1] "list"
```

```
names(mylist)
```

```
## [1] "hello"   "numbers"
```

Accessing lists can be done by name with `$` or by position with `[[ ]]`.

```
mylist$hello
```

```
## [1] "Hello" "world"
```

```
mylist[[2]]
```

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

5.2 Other types not covered here

Matrices are another tabular data type. These come up when doing more mathematical tasks in R. They are also commonly used in bioinformatics, for example to represent RNA-Seq count data. A matrix, as compared to a data frame:

- contains only one type of data, usually numeric (rather than different types in different columns).
- commonly has `rownames` as well as `colnames`. (Base R data frames can have `rownames` too, but it is easier to have any unique identifier as a normal column instead.)
- has individual cells as the unit of observation (rather than rows).

Matrices can be created using `as.matrix` from a data frame, `matrix` from a single vector, or using `rbind` or `cbind` with several vectors.

You may also encounter "S4 objects", especially if you use Bioconductor[1] packages. The syntax for using these is different again, and uses `@` to access elements.

5.3 Programming

Once you have a useful data analysis, you may want to do it again with different data. You may have some task that needs to be done many times over. This is where programming comes in:

- Writing your own functions[2].
- For-loops[3] to do things multiple times.
- If-statements[4] to make decisions.

The "R for Data Science" book[5] is an excellent source to learn more. Monash Data Fluency "Programming and Tidy data analysis in R" course[6] also covers this.

[1]http://bioconductor.org/
[2]http://r4ds.had.co.nz/functions.html
[3]http://r4ds.had.co.nz/iteration.html
[4]http://r4ds.had.co.nz/functions.html#conditional-execution
[5]http://r4ds.had.co.nz/
[6]https://monashdatafluency.github.io/r-progtidy/

Chapter 6

Next steps

6.1 Deepen your understanding

Our number one recommendation is to read the book "R for Data Science"[1] by Garrett Grolemund and Hadley Wickham.

Also, statistical tasks such as model fitting, hypothesis testing, confidence interval calculation, and prediction are a large part of R, and one we haven't demonstrated fully today. Linear models, and the linear model formula syntax `~`, are core to much of what R has to offer statistically. Many statistical techniques take linear models as their starting point, including limma[2] for differential gene expression, `glm` for logistic regression (etc), survival analysis with `coxph`, and mixed models to characterize variation within populations.

- "Statistical Models in S" by J.M. Chambers and T.J. Hastie is the primary reference for this, although there are some small differences between R and its predecessor S.
- "An Introduction to Statistical Learning"[3] by G. James, D. Witten, T. Hastie and R. Tibshirani can be seen as further development of the ideas in "Statistical Models in S", and is available online. It has more of a machine learning than a statistics flavour to it (the distinction is fuzzy!).
- "Modern Applied Statistics with S" by W.N. Venable and B.D. Ripley is a well respected reference covering R and S.
- "Linear Models with R" and "Extending the Linear Model with R" by J. Faraway[4] cover linear models, with many practical examples.

[1]http://r4ds.had.co.nz/
[2]https://bioconductor.org/packages/release/bioc/html/limma.html
[3]http://www-bcf.usc.edu/~gareth/ISL/
[4]http://www.maths.bath.ac.uk/~jjf23/

6.2 Expand your vocabulary

Have a look at these cheat sheets to see what is possible with R.

- RStudio's collection of cheat sheets[5] cover newer packages in R.
- An old-school cheat sheet[6] for dinosaurs and people wishing to go deeper.
- A Bioconductor cheat sheet[7] for biological data.

The R Manuals[8] are the place to look if you need a precise definition of how R behaves.

6.3 Join the community

Join the Data Fluency community at Monash[9].

- Mailing list for workshop and event announcements.
- Slack for discussion.
- Monthly seminars on Data Science topics.
- Drop-in sessions on Friday afternoon.

Meetups in Melbourne:

- MelbURN[10]
- R-Ladies[11]

The Carpentries[12] run intensive two day workshops on scientific computing and data science topics worldwide. The style of this present workshop is very much based on theirs. For bioinformatics, COMBINE[13] is an Australian student and early career researcher organization, and runs Carpentries workshops and similar.

[5]https://www.rstudio.com/resources/cheatsheets/
[6]https://cran.r-project.org/doc/contrib/Short-refcard.pdf
[7]https://github.com/mikelove/bioc-refcard/blob/master/README.Rmd
[8]https://cran.r-project.org/manuals.html
[9]https://www.monash.edu/data-fluency
[10]https://www.meetup.com/en-AU/MelbURN-Melbourne-Users-of-R-Network/
[11]https://www.meetup.com/en-AU/R-Ladies-Melbourne/
[12]https://carpentries.org/
[13]https://combine.org.au/