Section 2

Your Name

13/12/2021

Contents

1. Introduction

Reproducible:

> A result is reproducible when the same analysis steps performed on the same dataset consistently produces the same answer.

---

2. RMarkdown

Have you ever used the internet?
The internet uses a language called HTML (Hypertext Markup Language). RMarkdown uses a similar language to link data, code, methods results, graphs and charts to produce an integrated formatted document. This tutorial was created in RMarkdown.

Have you ever completed a project and then realised there is an error in the data?
This probably took some time to fix if the graphs were created in Excel, the analysis in statistical software and the written text in Word. In RMarkdown you can create everything in the RMarkdown file. If you change the data and then run the RMarkdown file all the changes are done for you.

So how does this help reproducibility?
In a final Word document, publication or report where only the results are included, there is no visibility of the data, code or method. In order for research to be reproducible, to be able to check the accuracy and run it for our-self, we need to be able to understand how the result was formulated. In the RMarkdown file the names of the data sets, the manipulation of the data, the code and how the results are produced are all documented in the RMarkdown file. If the data sets are also shared on a shared repository, such as GitHub, you would even be able to create the entire report yourself by simply running the Rmarkdown file.

Now that you know how valuable RMarkdown is for reproducibility, we will work through some basic formatting features in RMarkdown.

First, install RMarkdown and open an RMarkdown file.

Instructions on how to install RMarkdown can be found here.

File/New File/RMarkdown

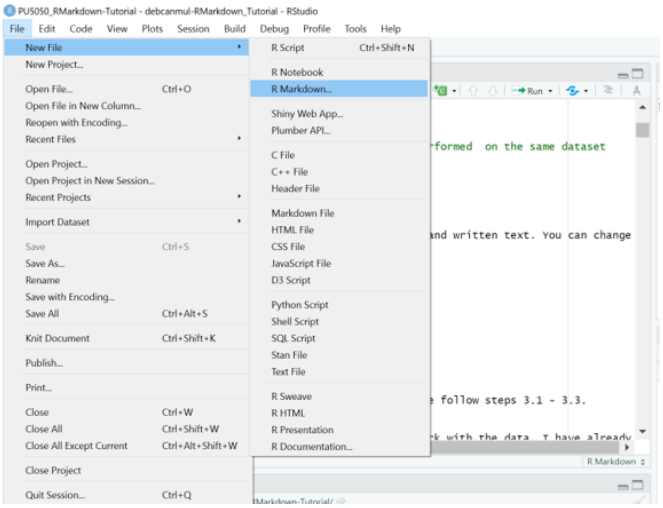


Figure 1: New RMarkdown File

We will work through each of these topics in turn:

- 2.1 RMarkdown File Header
- 2.2 Text Chunks
  - 2.2.1 Headers
  - 2.2.2 Lists
- 2.3 Code Chunks
- 2.4 Inline Code
- 2.5 Output File Type Specification
- 2.6 Generation of Output

2.1 RMarkdown File Header

What is an RMarkdown File Header?

A basic RMarkdown header is created by default when you create an RMarkdown file. It consists of the following text:

This header is called a YAML header. YAML stands for "YAML Ain't Markup Language". Yes, the terminology is rather odd. It consists of a title, author, date and output. You can change the default header by simply editing it in RMarkdown. However make sure it is in the same layout, indentation will change the output of the header. On a separate line start and end with —.

The YAML header can be changed to change the output of the document. In the basic header it is set to html_document. It can be changed to a pdf_document. We will talk about this in more detail in section 2.5. The font and font size for the entire document can also be changed by editing the YAML header. A contents page and a bibliography are also options.

2.2 Text Chunks

Text chunks are sections where you freely type text. This is anywhere below the YAML header, where you haven't added a code chunk, see section (2.3) for more details on code chunks. You can edit the text by using different syntax commands. We will look at headers and then lists.

2.2.1 Headers

To create a header in RMarkdown in a text chunk you have to start with a # and then a space and then type the name of the header. The biggest header will start with a single #, ## would create a slightly smaller header and so on.

```
### Large Header
#### Slightly smaller header
##### Even smaller header
```

The output would be as shown below:

Large Header

Slightly smaller header

Even smaller header

2.2.2 Lists

To create a list you can either use bullet points or a numbered list. To create bullet points start with * then a space, in front of the name of item in the list.

```
* Aberdeen
* Dundee
* Edinburgh
* Glasgow
* Inverness
```

The output would be as shown below:

- Aberdeen
- Dundee
- Edinburgh
- Glasgow
- Inverness

To create a numbered list you write with numbers in the front of the item in the list. Even if you don't number the items correctly from 1-5. If you number the list in any order RMarkdown will renumber them for you in ascending order starting from the top of the list.

```
1. Aberdeen
8. Dundee
3. Edinburgh
```

6. Glasgow
5. Inverness

The output would be as shown below:

1. Aberdeen
2. Dundee
3. Edinburgh
4. Glasgow
5. Inverness

To create a sublist in RMarkdown, you need to indent the sublist and use + in front of each item in the sublist.

1. Aberdeen
 + Aberdeen Art Gallery
 + Duthie Park
 + Aberdeen Maritime Museum

1. Aberdeen
    - Aberdeen Art Gallery
    - Duthie Park
    - Aberdeen Maritime Museum

2.3 Code Chunks

Code chunks are where you type all the code for your project. You can decide if you want all the code visible in your final report or if you want it hidden. You can also switch off error messages and decide if the output from the code chunk is to appear in your report or if it's just to be used within another part of your code in a different code chunk. You can change the settings for each code chunk using the code chunk cog.

Code chunks start with three back ticks, {r} and end with 3 back ticks "' on a separate line. You can use the shortcut **CTRL** + **ALT** + **I** to create a code chunk. The + means "and" and not the plus sign on the keyboard.

"'{r}

Your code goes in here...

"'

Code chunks in the RMarkdown file are highlighted in grey.

```
3+7
```

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

I set the code chunk cog to have the settings:

Output: Show code and output

You can add comments in code chunks by adding # to the front of the comment. This will be used in the next section to show you how to add inline code.

2.4 Inline Code

You may wish to add a value calculated from your data somewhere within a sentence in your text. In this case code chunks would not be suitable, but you can use inline code. Instead of 3 back ticks you use only 1 before and afer r.

```
#The total number of people waiting in the hospital for appointments are `r 3+7`
```

The total number of people in the waiting room are 10.

2.5 Output File Type Specification

Output file types were mentioned briefly in section 2.1, YAML header. We can change the output file type in the header from html_document to pdf_document. You can also change the file type to word documents (word_document), markdown files (md_document) and even different types of slides for presentations. If you want to take it even a step further you can create interactive documents. Just a word of warning some changes to the YAML header might not turn out the way you want them in different output file types.

2.6 Generation of Output

To create the chosen file type you have to Knit the data, code and text into the final document. If you want to create pdf documents you will have to install a package called tinytex.

---

3. Data Preparation

What is a fake dataset?
The data sets that were created for this tutorial are called fake data sets. They were created using code to generate random data specifically for this tutorial. If the data isn't real, why are fake data sets useful? The type of data for our scenario is health data. Health data cannot be openly shared due to confidentiality. If you were asked to find the answer to a similar question but with real data, you would have gained the knowledge and experience by using the fake data. Not only that, if I had shared the method with you, the code in the RMarkdown file, then very quickly with little manipulation you would have an answer.

You may also look at my code and think of improvements or find errors. This could all be fixed and edited before the real data was analysed. However, there are some limitations to fake data. They can't be used for analysis that requires a specific statistical test. The data was just randomly generated. There is a possible solution, synthetic datasets.

Synthetic datasets are a special type of fake dataset that are created to keep the statistical properties. Synthetic datasets can be time consuming to create. If you are also doing research on a rare disease in a small population, it is likely the patient could be identified. So great care must be taken, but the benefits of fake data sets and synthetic datasets are essential for the reproducibility of health data.

What is metadata?

Before we can start to prepare data we need to complete the follow steps 3.1 - 3.3.

3.1 Install the packages

First you want to install the packages you will need to work with the data. I have already installed the packages so for you, you would enter the following code.

- Start a new markdown file (2.2.1)
- Start a new code chunk (2.2.3)

Then type in the following code.

install.packages(tidyverse)
install.packages(here)

3.2 Read the packages into R

```
# These are the packages we will use to work with the data.
library(tidyverse)
library(here)
```

3.3 Read in the data sets

R-Software is case sensitive, so if you name an object with all lower case letters, R will only recognise the name in this form.

Explain objects and assignment operators

```
demographics<-read_csv(here("2021-12-02_PU5050_Assessment_Input/2021-12-02_PU5050_Demographics.csv"))
ImagingWaitTimes<-read_csv(here("2021-12-02_PU5050_Assessment_Input/2021-12-02_PU5050_ImagingWaitTimes.
```

3.4 Inspect the data

Now that we have the data loaded into R, using the object names demographics and ImagingWaitTimes, we can start to inspect the data. Let's look at demographics first..

```
# We will use a function called glimpse to look at the variable names within the data set
glimpse(demographics)
```

```
## Rows: 100
## Columns: 4
## $ chi  <dbl> 2865914370, 7429183650, 2041536789, 6705438219, 9598617423, 47351~
## $ dob  <chr> "04/06/1950", "14/08/1950", "16/03/1951", "11/07/1951", "26/10/19~
## $ sex  <chr> "female", "male", NA, "male", "female", "male", "female", "male",~
## $ city <chr> "Inverness", "Edinburgh", "Dundee", NA, "Glasgow", "Glasgow", "Gl~
```

We can see that the data has 4 columns: chi, dob, sex and city. Explain double/character

---

Another function to look at the data

```
head(demographics)
```

```
## # A tibble: 6 x 4
##          chi dob        sex    city
##        <dbl> <chr>      <chr>  <chr>
## 1 2865914370 04/06/1950 female Inverness
## 2 7429183650 14/08/1950 male   Edinburgh
## 3 2041536789 16/03/1951 <NA>   Dundee
## 4 6705438219 11/07/1951 male   <NA>
## 5 9598617423 26/10/1951 female Glasgow
## 6 4735162809 18/03/1954 male   Glasgow
```

Again we can see there are 4 groups and the types of data.

Now that we have an understanding of the variable names and type of variables we need to look in more detail for missing values, NA, and any unusual data values.

I'm going to show you first of all how to find the number of missing values in a single column.

```
demographics%>%
  filter(is.na(dob))%>%
summarise(n())
```

```
## # A tibble: 1 x 1
##   `n()`
##   <int>
## 1     2
```

This shows that within the column dob there are 2 missing values. We want to remove these and save the results to a new object. Let's call the filtered data filtered_dob.

```
filtered_dob<-demographics%>%
  filter(!is.na(dob))%>%
  view()
```

If we now check how many missing values there are in the dob column in filtered_dob.

```
filtered_dob%>%
  filter(is.na(dob))%>%
  summarise(n())
```

```
## # A tibble: 1 x 1
##   `n()`
##   <int>
## 1     0
```

We can see now that the missing values have been removed within the new filtered data. Now if we were to continue with this method we would have to work our way along one column at a time. We would check for the number of missing values, save the new filtered data to a new object, remove the missing values and finally check the values have been removed. However, there is a way we can check all the columns at the same time.

```
demographics%>%
  summarise_all(~sum(is.na(.)))
```

```
## # A tibble: 1 x 4
##     chi   dob   sex  city
##   <int> <int> <int> <int>
## 1     0     2     2     2
```

We have looked at the original demographics file and we can see there are 2 missing values in the columns dob, sex and city. Let's remove them and call the new object demo_filter

```
demo_filter<-demographics%>%
  na.omit()%>%
view()
```

Let's check that all the missing values have been removed.

```
#Remember the name of the new filtered data is demo_filter
demo_filter%>%
  summarise_all(~sum(is.na(.)))
```

```
## # A tibble: 1 x 4
##     chi   dob   sex  city
##   <int> <int> <int> <int>
## 1     0     0     0     0
```

We can see that all of the missing values have been removed.

Now we need to look for any unusual values within the data. As we mentioned earlier the dob variable is a character. We will first change this to a date variable.

```
#The new filtered data is demo_filter
demo_date<-demo_filter%>%
  mutate(dob=as.Date(dob,format("%d/%m/%Y")))%>%
  view()
```

Now I will find the youngest and oldest person in the demo_date.

```
#Youngest person
demo_date%>%
  summarise(min(dob))
```

```
## # A tibble: 1 x 1
##   `min(dob)`
##   <date>
## 1 1950-06-04
```

```r
#Oldest person
demo_date%>%
  summarise(max(dob))
```

```
## # A tibble: 1 x 1
##   `max(dob)`
##   <date>
## 1 2050-10-11
```

Now unless this person has traveled back in time, you can see there is an error with this date of birth.

This row will need to be removed from demo_date.

```r
# Filtered for dob before today's date
demo_date_filter<-demo_date%>%
  filter(dob<"2021-12-11")%>%
  view()
```

Our filtered data is now ready to be saved as a csv file

```r
write_csv(demo_date_filter,"2021-12_02_PU5050_Assessment_Output/2021-12-11_PU5050_Demo_Filtered.csv")
```

We are now going to look at the ImagingWaitTimes data.

```r
#Let's use the glimpse function that we used to inspect the demographics data
glimpse(ImagingWaitTimes)
```

```
## Rows: 100
## Columns: 5
## $ chi                       <dbl> 7154809632, 5986237140, 1520976843, 85623409~
## $ month                     <chr> "August", "September", "October", "August", ~
## $ DiagnosticTestType        <chr> "Imaging", "Imaging", "Imaging", "Imaging", ~
## $ DiagnosticTestDescription <chr> "Non-obstetric Ultrasound", "Computer Tomogr~
## $ WaitingDays               <dbl> 63, 60, 53, 35, 56, 44, 60, 31, 63, 62, 35, ~
```

We can see there are 5 columns, with variables, chi, month, DiagnosticTestType, DiagnosticTestDescription and WaitingDays.

```r
# The summary function will find the minimum, 1st Quartile, median, mean, 3rd quartile and maximum valu
summary(ImagingWaitTimes)
```

```
##       chi               month           DiagnosticTestType
##  Min.   :1.236e+09   Length:100         Length:100
##  1st Qu.:4.328e+09   Class :character   Class :character
##  Median :6.303e+09   Mode  :character   Mode  :character
##  Mean   :5.840e+09
##  3rd Qu.:7.350e+09
##  Max.   :9.841e+09
##  DiagnosticTestDescription  WaitingDays
```

```
##  Length:100                 Min.   :    -50
##  Class :character           1st Qu.:     33
##  Mode  :character           Median :     44
##                             Mean   :  32722
##                             3rd Qu.:     55
##                             Max.   :3267890
```

This function will provide more insight to numeric variables, but it shows you another way that you can quickly get a feel for the data in your dataset. We are particulary interested in the variable WaitingDays. You can see that the minimum number of days is -50. This is obviously an error and will need to be removed. Notice the maximum value is 3267890. We would hope people are not having to wait 8953.1 years for an imaging appointment! This will aslo have to be removed.

Let's filter the data so that the waiting days is not less than zero and not greater than 365 days. Then use the summary function to look at the new maximum and minimum values.

```
ImagWaitTimesFilter<-ImagingWaitTimes%>%
  filter(WaitingDays>0 & WaitingDays<100)%>%
  view()
```

We can see the minimum waiting time is 24 days and the maximum is 63 days. This seems more sensible. Now lets look for missing values.

```
ImagWaitTimesFilter%>%
  summarise_all(~sum(is.na(.)))%>%
   view()
```

```
WaitTime_Tidy<-ImagWaitTimesFilter%>%
  na.omit()%>%
  view()
```

```
WaitTime_Tidy%>%
summarise_all(~sum(is.na(.)))
```

```
## # A tibble: 1 x 5
##     chi month DiagnosticTestType DiagnosticTestDescription WaitingDays
##   <int> <int>              <int>                      <int>       <int>
## 1     0     0                  0                          0           0
```

The WaitTime_Tidy data has now been filtered for unusual values and the rows with missing values has been removed. We are now ready to save the sorted data.

```
write_csv(WaitTime_Tidy,"2021-12_02_PU5050_Assessment_Output/2021-12-12_PU5050_WaitTime_Tidy.csv")
```

We are now read to join the data sets. There are many ways to join datasets. You have to have a good understanding of the data and how you want it to join. In our scenario we want all of the waiting time details matched with the corresponding chi number from the demo_date_filter data.

```
join_demo_wait <- inner_join(WaitTime_Tidy,demo_date_filter, by = "chi")%>%
view()
# There are still two NA values in ImagWaitTimesFilter
```

```
# Check the NA values again once the issue has been fixed!!!
join_demo_wait%>%
  summarise_all(~sum(is.na(.)))
```

```
## # A tibble: 1 x 8
##     chi month DiagnosticTestTy~ DiagnosticTestDes~ WaitingDays   dob   sex  city
##   <int> <int>             <int>              <int>       <int> <int> <int> <int>
## 1     0     0                 0                  0           0     0     0     0
```

---

4. Data Visualisation

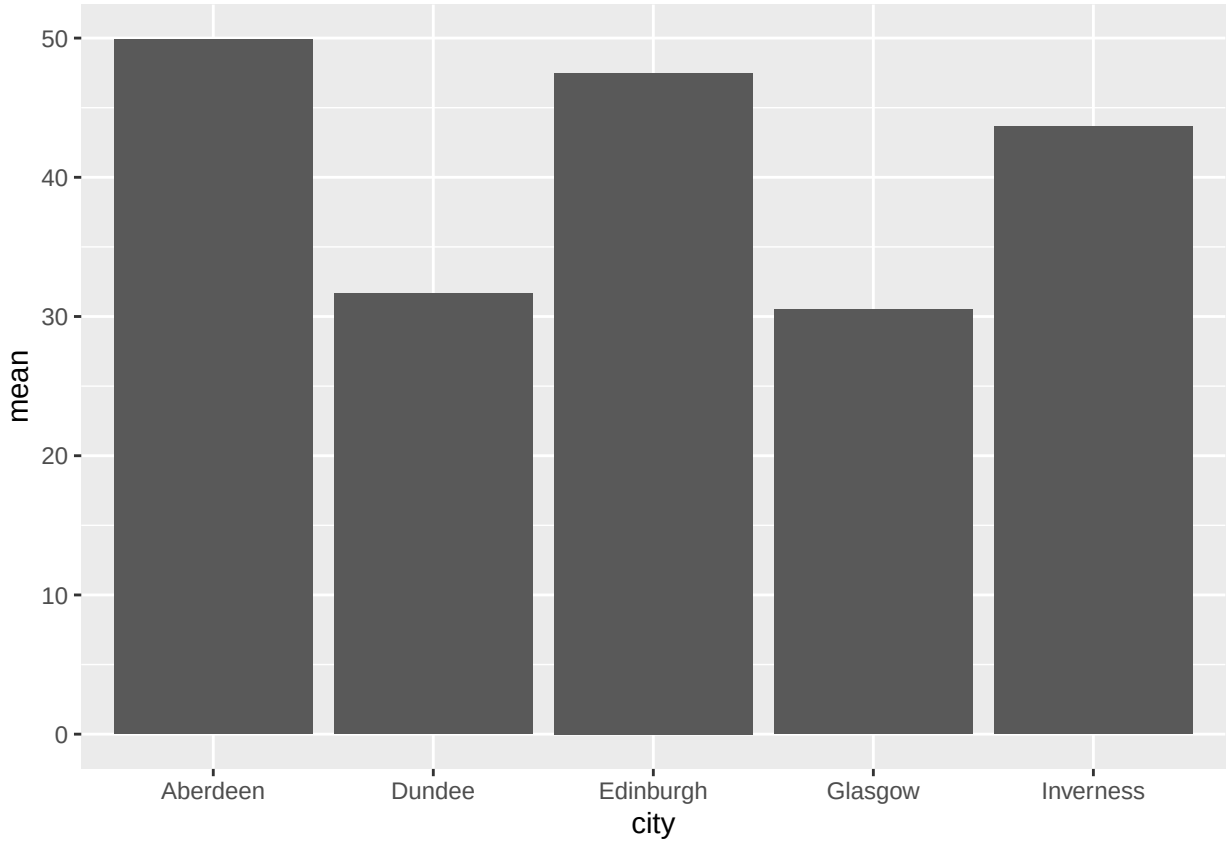
We are now ready to answer the question: What is the average number of Waiting Days, in each Scottish city, in October 2021?

The first thing to do is to decide what data it is we need to answer the question. We need the month, DiagnosticTestDescription, WaitingDays and City.

Instead of manipulating one part of the data at a time we will now make more use of the pipe operator and have multiple lines of code in one code chunk. Before we do this I will explain each of the function that we will use in the analysis.

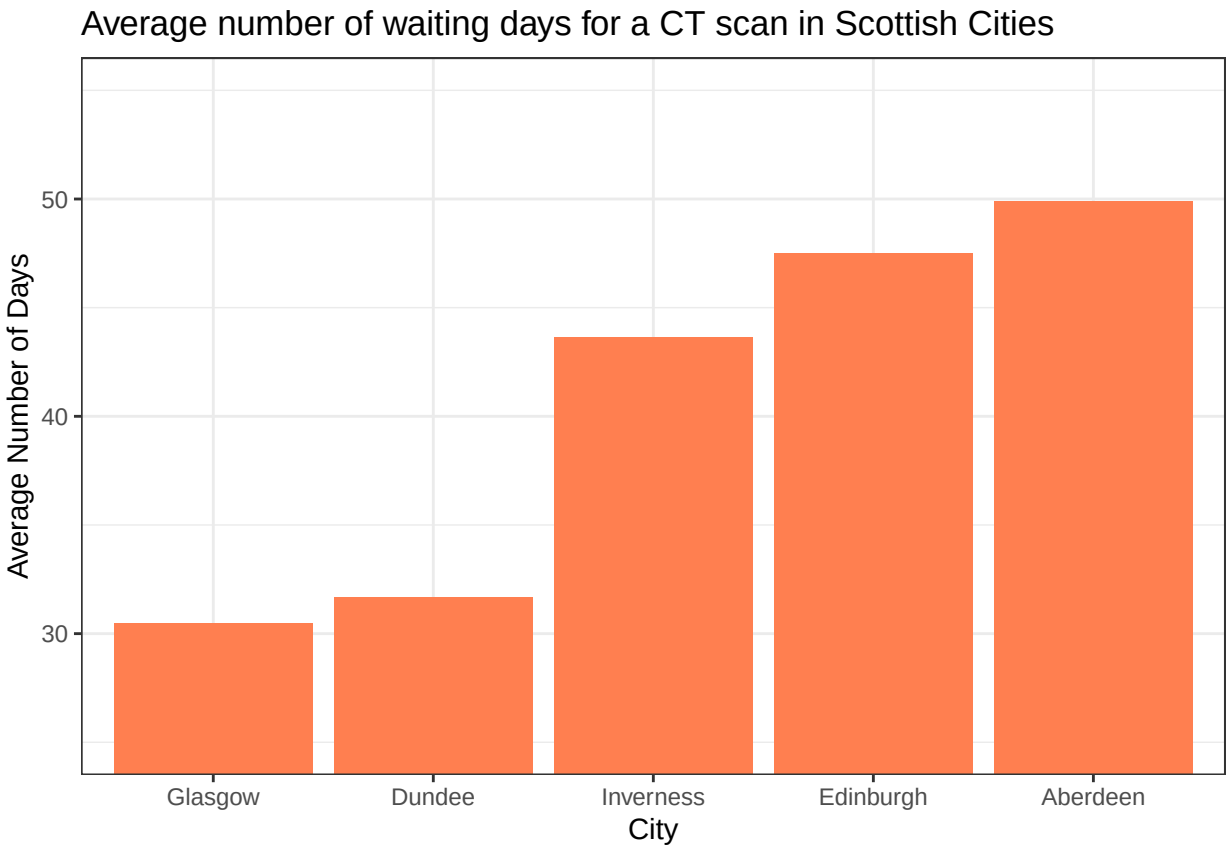
- select -
- filter -
- group_by -
- mutate -
- unique -
- ggplot -

```
#still got to remove the missing values
join_demo_wait%>%
  select(month,DiagnosticTestDescription,WaitingDays,city)%>%
  filter(DiagnosticTestDescription=="Computer Tomography")%>%
  filter(month=="October")%>%
  group_by(city)%>%
    mutate(mean=mean(WaitingDays))%>%
  select(city,mean)%>%
unique()%>%
ggplot()+
geom_col(aes(city,mean))
```




```
join_demo_wait%>%
  select(month,DiagnosticTestDescription,WaitingDays,city)%>%
  filter(DiagnosticTestDescription=="Computer Tomography")%>%
  filter(month=="October")%>%
  group_by(city)%>%
    mutate(mean=mean(WaitingDays))%>%
  select(city,mean)%>%
unique()%>%

ggplot(aes(fct_reorder(city,mean),mean))+
geom_col(fill="coral")+
   labs(x="City",y="Average Number of Days", title = "Average number of waiting days for a CT scan in S
  coord_cartesian(ylim=c(25,55))+
  theme_bw()
```



---

1. Data
2. Facets
3. Mapping
4. Scales
5. Coords
6. Geoms
7. Stats

   Themes

---

5. Summary and Reflection

What has been your experience using R for reproducible data analysis?

I found it challenging at first to understand the very basic concepts, such as, reading in data or how the file structures worked. This lack of understanding limited the reproducibility of my work. I kept revising the basics and gradually I began to improve my R programming skills. I realised very quickly I had to really think ahead about what it was I was trying to achieve. I had to ask myself constant questions;

- How is this dataset formatted?
- What functions can I use?

- Is this the expected outcome and if not, why?

The constant reflection is not enough will to improve the reproducibility, but all of the reflection must be documented for others to follow. This has also been a large challenge for me. The organisation of file structures and logically documenting my work is something I will have to practice.

- Reading lots of theory - really need practice
- Please reflect on any problems or issues you encountered, for example whilst creating and merging datasets, or using the R language.
  - Over thinking the basics
  - Not focusing on what I expect to happen
  - Not having a logical plan
  - Had to slow down
  - Couldn't visualise how everything connected ie branches
  - RNotebook
