Covid 19 - some Challenges - some Data

Ralf Becker, The University of Manchester

Introduction

This lab has been written in the first week of April 2020.

In this lab we will investigate some of the data which can help inform issues around the Covid 19 Pandemic.

Worldwide there is a huge effort being undertaken by specialists of all colours to understand and reduce the threat of this pandemic and of course there are huge efforts undertaken by many specialists to help us all live under these new conditions.

Here we will illuminate some of the challenges and questions to which people with data skills can contribute.

There are a number of great place to start any such journey:

- Coronavirus Resource Center at the John Hopkins University. There you can find a collation of data but also articles on the topic. have to achieve the following tasks/learning outcomes:
- Our World in Data has a dedicated Covid-19 page where they review some of the latest numbers. A particularly interesting element of this page is that they provide a discussion of why they use the daily updates provided by the European Centre for Disease Control, ECDC, rather than other sources. This is a particularly good case of careful datawork.
- The data competition site Kaggle has a dedicated Covid-19 section of challenges.

In fact we start with the latter site. At the time of writing the structure their data challenges into three categories. "Use Natural Language Processing to answer key questions from the scientific literature", "Forecast COVID-19 cases and fatalities to help understand what drives transmission rates", "Curate COVID-19 related datasets to further research" and "Use exploratory analysis to answer research questions that support frontline responders".

Exploring the latter of these you will find that 12 particular tasks were posed:

- Which populations are at risk of contracting COVID-19?
- What is the incidence of infection with coronavirus among cancer patients?
- Which patient populations pass away from COVID-19?
- Are hospital resources being diverted from providing oncology care to support the COVID-19 response?
- How is the implementation of existing strategies affecting the rates of COVID-19 infection?
- Which populations have contracted COVID-19 and require ventilators?
- Which populations have contracted COVID-19 who require the ICU?
- What is the change in turnaround time for routine lab values for oncology patients?
- Which populations of clinicians are most likely to contract COVID-19?
- Which populations assessed should stay home and which should see an HCP?
- Which populations of clinicians and patients require protective equipment?
- How are patterns of care changing for current patients (i.e. cancer patients)?

Quoting from the Kaggle website: "The tasks associated with this dataset were developed and evaluated by global frontline healthcare providers, hospitals, suppliers, and policy makers. They represent key research questions where insights developed by the Kaggle community can be most impactful in the areas of at-risk population evaluation and capacity management."

This should give you a good idea of what type of questions data analysis can contribute to. You should check on the Kaggle website and check out the respective notebooks people contributed. Most of them are written in Python, but some are in R. But even looking at the Python ones is instructive.

This document is written for people who have had some first experiences with R (it is not suitable for total novices) and want to learn some new techniques. In particular you will be able to learn how to

- import Covid-19 data directly from the European Centre for Disease Control
- perform some basic data cleaning techniques
- produce some line graphs in `ggplot`
- create smoothed versions of volatile daily series.
- produce maps illustrating the regional spread of the Covid-19 cases using the `tmap` package

In addition to this we will end this computer lab by importing some travel data (flight connections) and provide a very basic illustration of how such data can be used in a model which attempts to anticipate the spread of a pandemic.

Some exploratory data analysis

Let's do some exploratory analysis using a dataset published by the ECDC. This dataset is used by the Our World in Data page and is also part of the dataset in the Kaggle challenge.

Preparing your workfile

We add the basic libraries needed for this work. Later we will load some additional packes (to produce maps).

```
library(sets)          # used for some set operations
library(forecast)      # used for some data smoothing
library(readxl)        # enable the read_excel function
library(tidyverse)     # for almost all data handling tasks
library(ggplot2)       # plotting toolbox
```

Data Upload

Very helpfully the ECDC webpage through which you can download the data provides an R script which allows you to download the most current dataset. This is the script replicated in the next code block. You could of course download the datset to your computer and then upload the excel or csv file, but here the ECDC has build a direct pipeline into their data.

```
#these libraries need to be loaded
library(utils)
library(httr)

#download the dataset from the ECDC website to a local temporary file
GET("https://opendata.ecdc.europa.eu/covid19/casedistribution/csv",
    authenticate(":", ":", type="ntlm"),
    write_disk(tf <- tempfile(fileext = ".csv")))
```

```
## Response [https://opendata.ecdc.europa.eu/covid19/casedistribution/csv/]
##   Date: 2020-04-09 15:15
##   Status: 200
##   Content-Type: application/octet-stream
##   Size: 497 kB
```

```
## <ON DISK>  C:\Users\msassrb2\AppData\Local\Temp\RtmpS60riX\file162043837a0f.csv
```

```
#read the Dataset sheet into "R". The dataset will be called "data".
data <- read.csv(tf)
```

** By uploading data in this way you will always get up to date data. This means that, if you replicate this code, you are having more recet data than the ones used during the writing process of this document. The data interpretations should therefore be read as valid on 9 April 2020.**

Some data cleaning

Let's look at the structure of this dataset. We want to make sure we understand all the variables and give them sensible names we want to work with.

```
str(data)
```

```
## 'data.frame':    9717 obs. of  10 variables:
##  $ dateRep                : Factor w/ 101 levels "01/01/2020","01/02/2020",..: 36 32 28 24 20 16 12 8
##  $ day                    : int  9 8 7 6 5 4 3 2 1 31 ...
##  $ month                  : int  4 4 4 4 4 4 4 4 4 3 ...
##  $ year                   : int  2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 ...
##  $ cases                  : int  56 30 38 29 35 0 43 26 25 27 ...
##  $ deaths                 : int  3 4 0 2 1 0 0 0 0 0 ...
##  $ countriesAndTerritories: Factor w/ 205 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ geoId                  : Factor w/ 204 levels "AD","AE","AF",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ countryterritoryCode   : Factor w/ 202 levels "","ABW","AFG",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ popData2018            : int  37172386 37172386 37172386 37172386 37172386 37172386 37172386 3717
```

Some of the variables have obvious meaning, sich as `day`, `month`, `year`, `countriesAndTerritories` and `popData2018`, the latter giving the population of the respective country in 2018. `geoId` and `countryterritoryCode` are common appreviations for the respective country.

For starters we want to shorten the name of `countriesAndTerritories` to `country` and `countryterritoryCode` to `countryCode` and `dateRep` to `dates`.

```
names(data)[names(data) == "countriesAndTerritories"] <- "country"
names(data)[names(data) == "countryterritoryCode"] <- "countryCode"
names(data)[names(data) == "dateRep"] <- "dates"
```

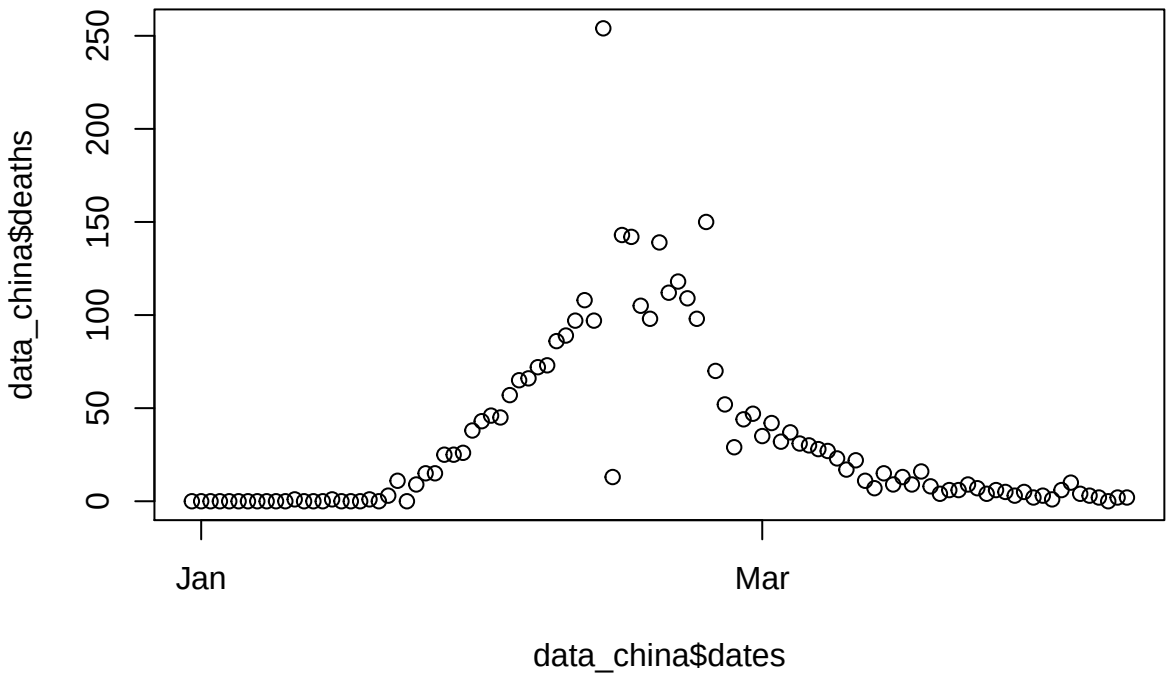
The first variable is a factor variable which includes the entire date string. At this stage R does not recognise this as a date. So let's change this as it will be useful for R to know that this variable represents a date. Dates are of the format 24/01/2020. In the `format` option we specify this format (`format = "%d/%m/%Y"`) such that R knows how to translate to dates (see this page to understand how to specify the format string).

```
data$dates <- as.Date(as.character(data$dates),format = "%d/%m/%Y")
```

Last, but most importantly, there are two variables called `cases` and `deaths`. These are daily data. So without further explanation it is not obvious whether these are the cumulative data (e.g. all the Covid-19 cases which have been identified in a country by a certain day, or whether these are the cases identified on that particular day). You could either go back to the data source to find an explanation or we could use our understanding of the problem at hand. The two series should look very different.

To investigate this let's look at one country in particular, say China, where this particular virus was first identified. We use the plot function which provides the standard build-in R plotting facility. Later we will look at using `ggplot` to produce nicer plots.

```
data_china <- data %>%  filter(country == "China")
plot(data_china$dates,data_china$deaths)  # specifies variable for x and y axis
```



We can clearly see that after an increase in the numbers of fatalities in China in January and February we see a decrease in numbers in March. If these were cumulative numbers we would not see a decrease. So these are the deaths which occured on a particular day. You can find the same for cases.

Before continuing we may also want to highlight a particularity in these data. You can see that in the middle of March there is one day (13 Feb 2020) on which almost 100 more deaths have been reported than at any other day. And in fact, the day before there were only 13 reported deaths. We will later see similar variability in other countries. The reason for this being a combination of definitial changes, but also being due to the fact that there may be daily variation in the underlying testing activity. Therefore any short term variation in identified cases is most likely a function of variation in testing activity masking a more steady development o fthe underlying cases.

Knowing that we are talking about daily statistics for new confirmed infections and daily deaths, we may also want to calculate the accumulated infections and deaths. This is achieved with the `cumsum` (cumulative sum) command. This takes a vector of data and keeps adding these up.

To illustrate what this command does, let's use an example.

```
test <- c(0,0,2,4,9,2)
cumsum(test)
```

```
## [1]  0  0  2  6 15 17
```

Before we apply this to our data in `data`, we need to make sure that we only accumulate by country (`group_by(country)`) and that the data are arranged by date (`arrange(dates)`) before we apply the `cumsum` function.

```
data <- data %>% group_by(country) %>%
          arrange(dates) %>%
```

```
        mutate(c_cases = cumsum(cases), c_deaths = cumsum(deaths)) %>%
        ungroup() # this undoes the grouping as we need the ungrouped data going ahead
```

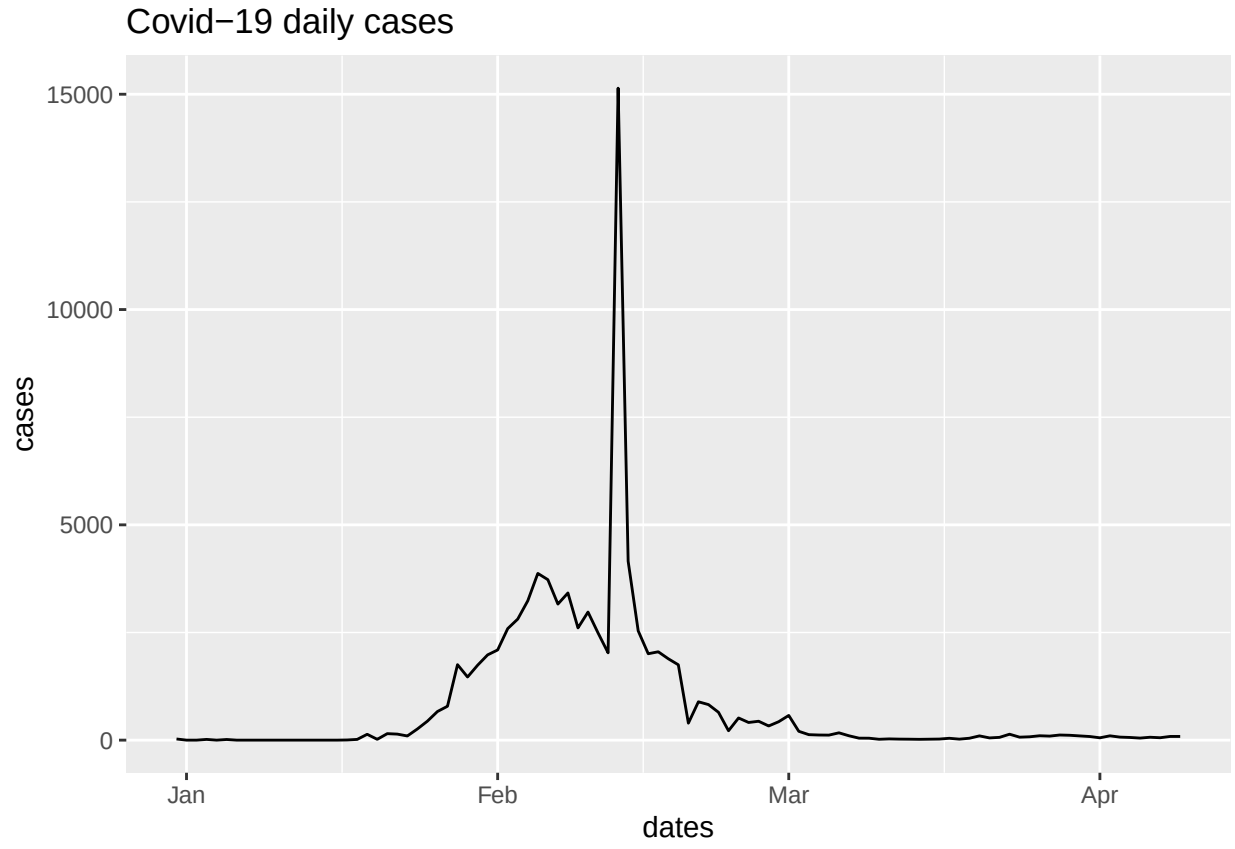
In the variables `c_cases` and `c_deaths` we now have the accumulated cases and deaths by country. At the end we undid the `group_by` by calling `ungroup()`.

Some country graphs

Let's create some nicer graphs to describe the development of the pandemic in different countries. We initially continue with the Chinese data.

First we replicate the above Figure but using the `ggplot` function which produces much nicer graphs. We first crreate the graph and save it in `g1` and eventually print it by just calling `g1`.

```
g1 <- ggplot(subset(data, country == "China"), aes(x=dates,y=cases)) +
      geom_line() +
      ggtitle("Covid-19 daily cases")
g1
```



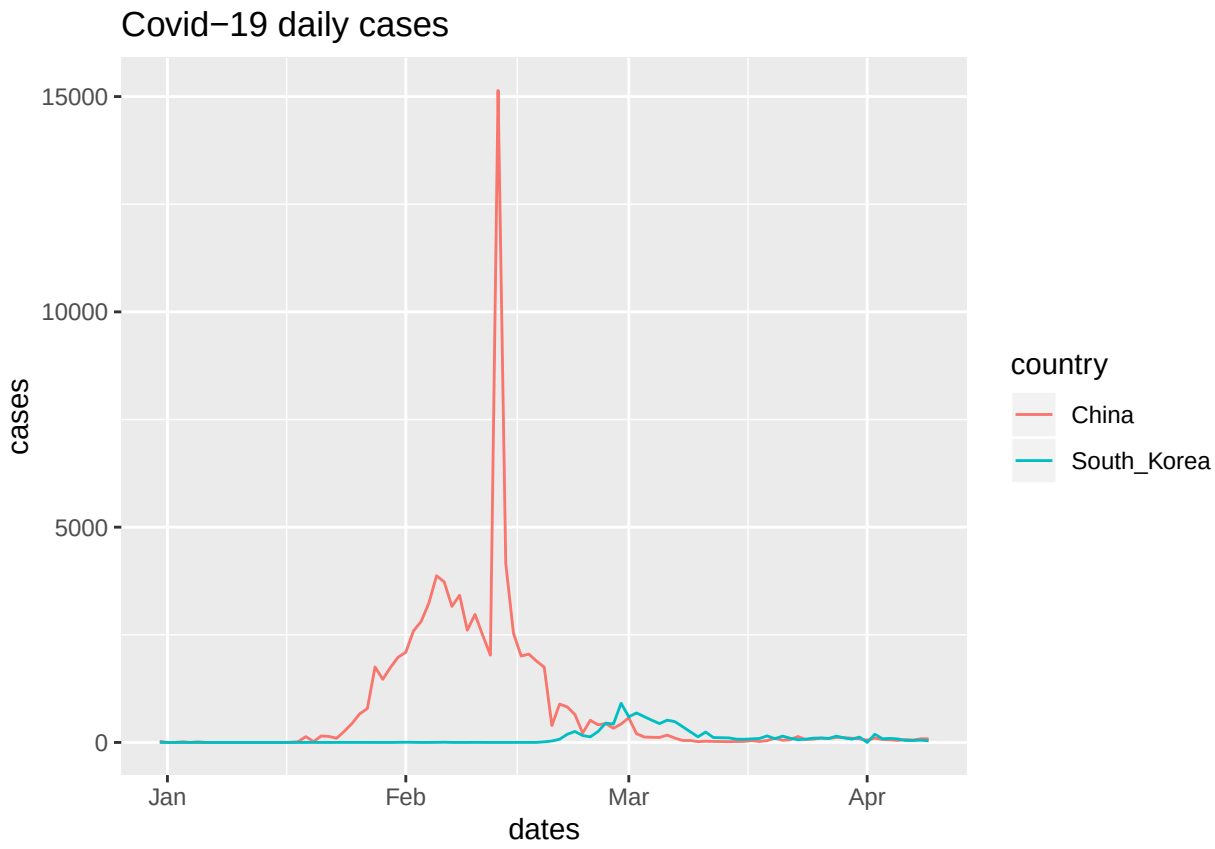
As you can see we didn't create a special Chinese dataset first, but we used the subset function instead.

You can see a huge spike in this dataseries. It turns out that this irregularity is the result of changes in data definitions as reported, for instance, in this CNBC article.

Let's overlay the daily cases for two countries, say China and South Korea.

```
sel_countries <- c("China", "South_Korea")
g2 <- ggplot(subset(data, country %in% sel_countries),
```

```
            aes(x=dates,y=cases, color = country)) +
      geom_line() +
      ggtitle("Covid-19 daily cases")
g2
```

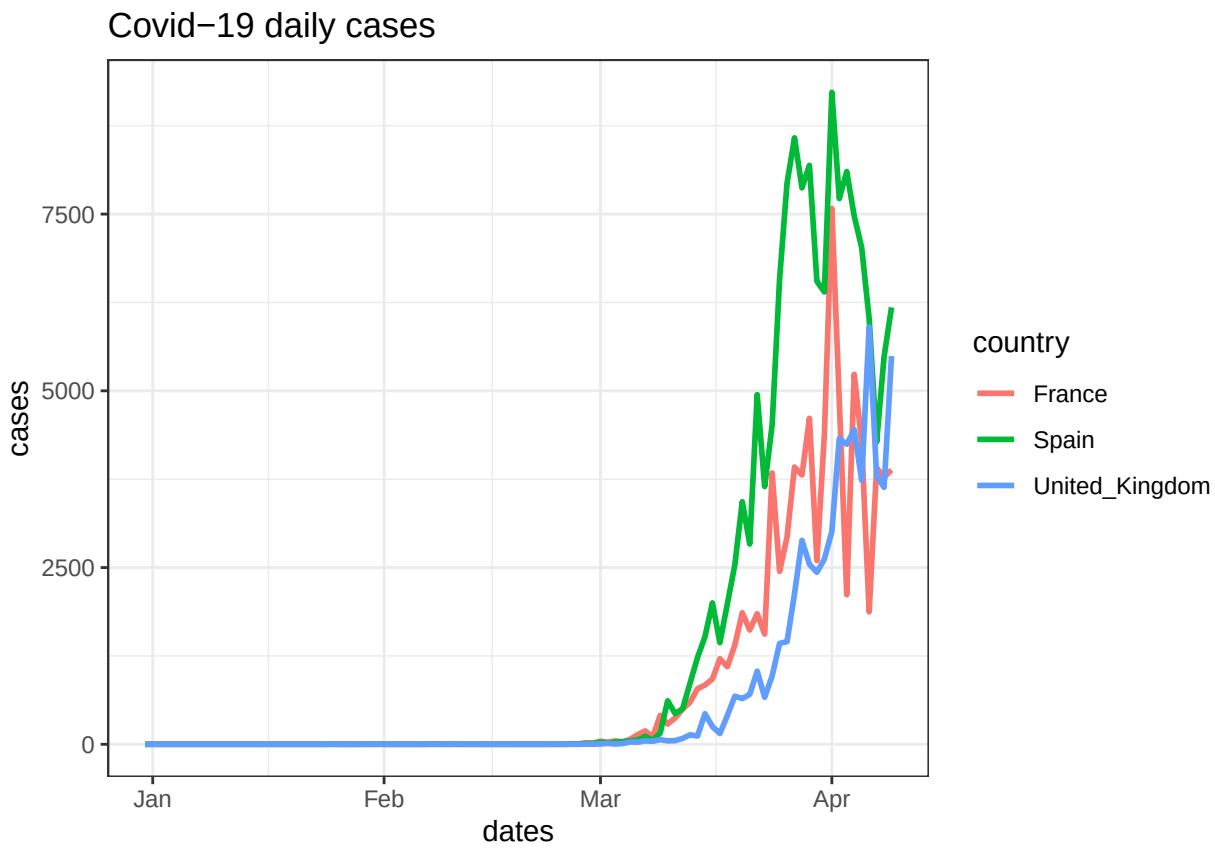


Here you can see the much praised ability by South Korea to suppress the numbers of infections effectively. However, you might argue that

> Explore: Look at graphs for different pairs or even trios of countries? Challenge: How do these look like if you were to look at per capita infection rates?

Now we look at some European countries.

```
sel_countries <- c("Spain", "France", "United_Kingdom")
g3 <- ggplot(subset(data, country %in% sel_countries),
            aes(x=dates,y=cases, color = country)) +
      geom_line(size = 1) +   # size controls the line thickness
      ggtitle("Covid-19 daily cases") +
      theme_bw()

g3
```



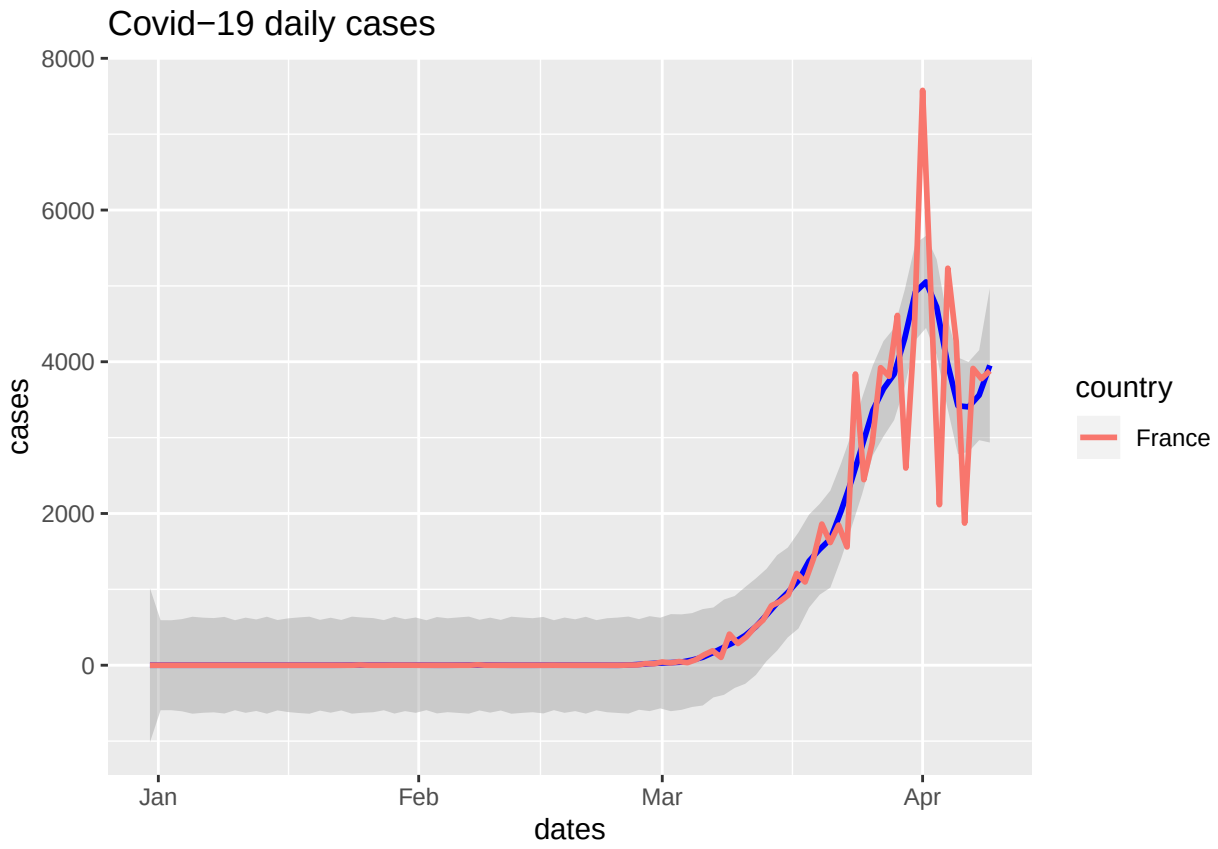
Explore: Check out the `ggplot` cheatsheet to see some of the many ways in which you can customise graps. In particular see what happens if you replace the last line `g3` with `g3 + theme_bw()` or `g3 + theme_dark()`.

You can see that especially the French data are very variable. It is important to understand that the number of newly identified cases is also a function of the testing effort on a particular day and if that is variable then we may well expect a huge variability. In addition to daily variability in one country in testing activity it is important to acknowledge that different countries may have quite different policies in place. For instance it has been well publicised that different countries have different strategies on testing (e.g. Financial Times, 2 April 2010, or FiveThirtyEight, 4 April 2020).

In fact there is so much variation that it was difficult to separate the signal from the noise. As this is a typical issue with high frequency data we often see a smoothed version of these data. `ggplot` has a build-in function which allows you to see smoothed versions of the data, `geom_smooth(method = "loess")`. This averages data in some local (time) window. We will not discuss the details of the algorithm here.

We start by focusing on the French data and add the smoothed version.

```
sel_countries <- c("France")
g4 <- ggplot(subset(data, country %in% sel_countries),
             aes(x=dates,y=cases, color = country)) +
      geom_smooth(method = "loess", span = 0.1,color = "blue") + # smoothed data
      geom_line(size = 1) +   # size controls the line thickness
      ggtitle("Covid-19 daily cases")
g4
```

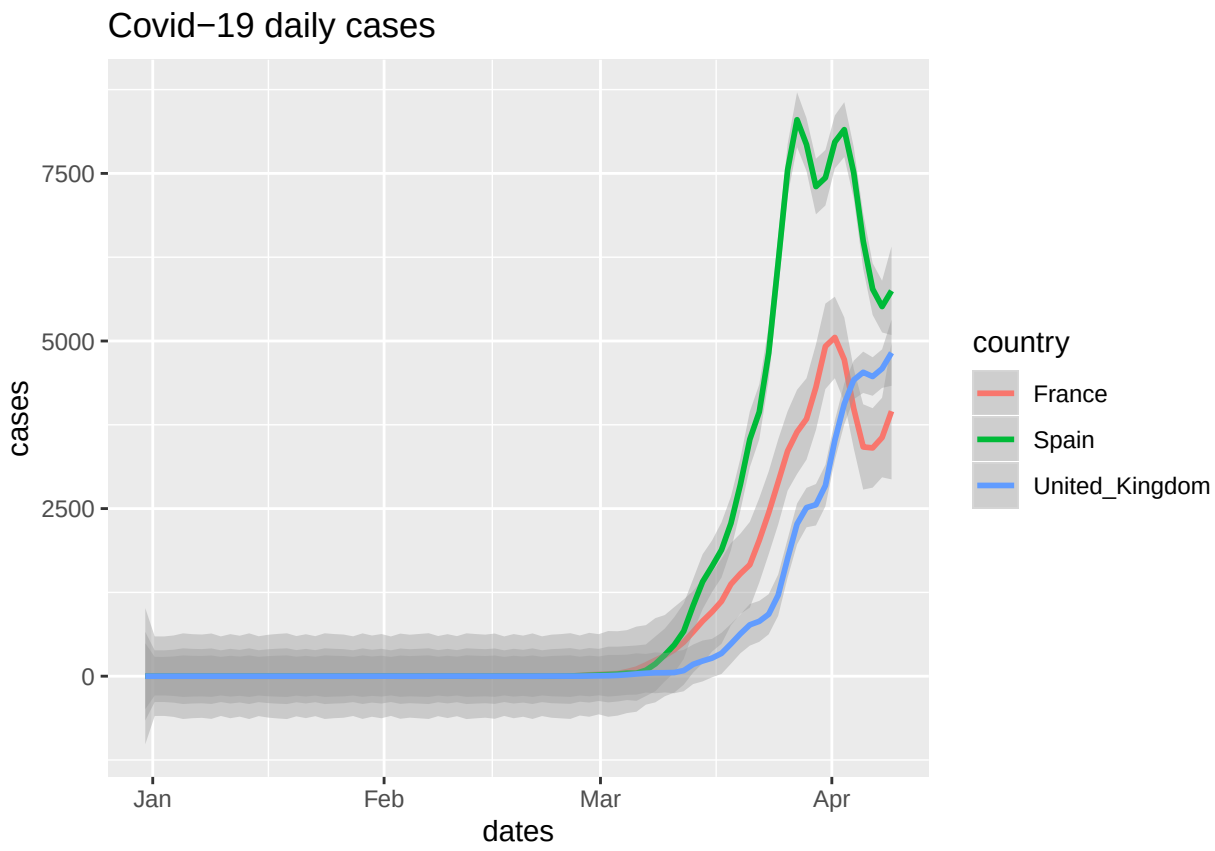


You can see that the smoothed version (in blue) clearly shows an early exponential trend and the most recent decline (as of 7/4/2020).

Explore: Change the values of the `span` option to figure out how it changes the results.

Let's look at the three countries again and only show the smoothed versions.

```
sel_countries <- c("Spain", "France", "United_Kingdom")
g5 <- ggplot(subset(data, country %in% sel_countries),
             aes(x=dates,y=cases, color = country)) +
      geom_smooth(method = "loess", span = 0.1) + # smoothed data
      ggtitle("Covid-19 daily cases")
g5
```



The grey areas around the smoothed data versions are confidence intervals. They recognise that the data are noisy and it is impossible to precisely get the underlying trend. As the French data are noisier than those of the UK and Spain the grey error bands are somewhat wider than the others, indicating larger uncertainty about the trend.

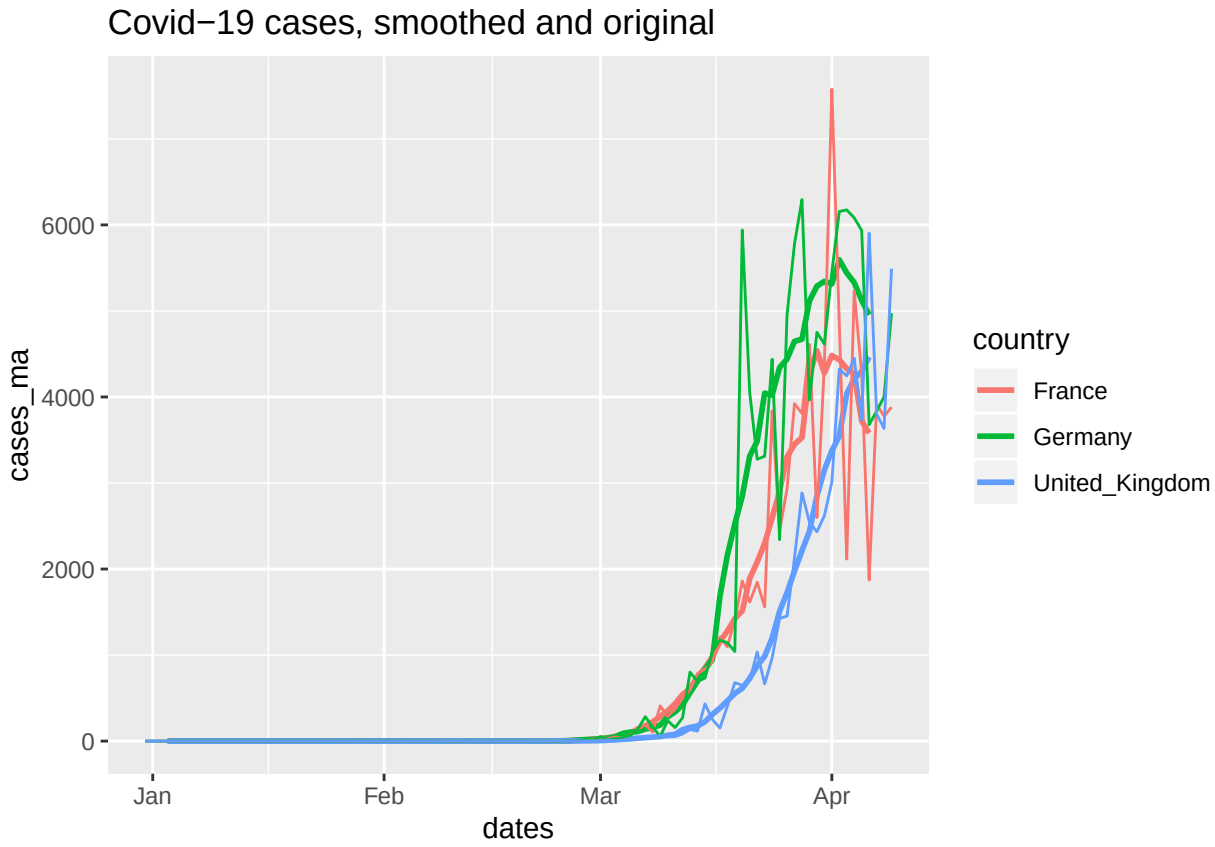
Smoothing can be useful. Here for instance we can perhaps see that France and Spain may have reached the peak of new registered cases, where, at the time of writing (7 April 2020), the UK numbers were still on a clearly positive trend.

In the example above the smoothing was done by and inside the `ggplot` function. You can also produce the moving average "manually". This is useful if you need the moving average for some further analysis (as we will later). Here we will use the `forecast` package which you can use to calculate a moving average. Here we create a new variable `cases_ma = ma(cases,order = 7)` which calculates a 7 day moving average. By way of example, take the 10th of March. Instead of reporting the cases figure for the 10th of March we are now using 7 observations from 7th of March to 13th of March (i.e. centered around the 10th of March) and `cases_ma` will then have, for the 10th of March the average of these 7 days.

```
data <- data %>% group_by(country) %>%
          arrange(dates) %>%
          mutate(obs = n()) %>%
          filter(obs > 20) %>%
          mutate(cases_ma = ma(cases,order = 7),
                 deaths_ma = ma(deaths,order = 7)) %>%
          ungroup()
```

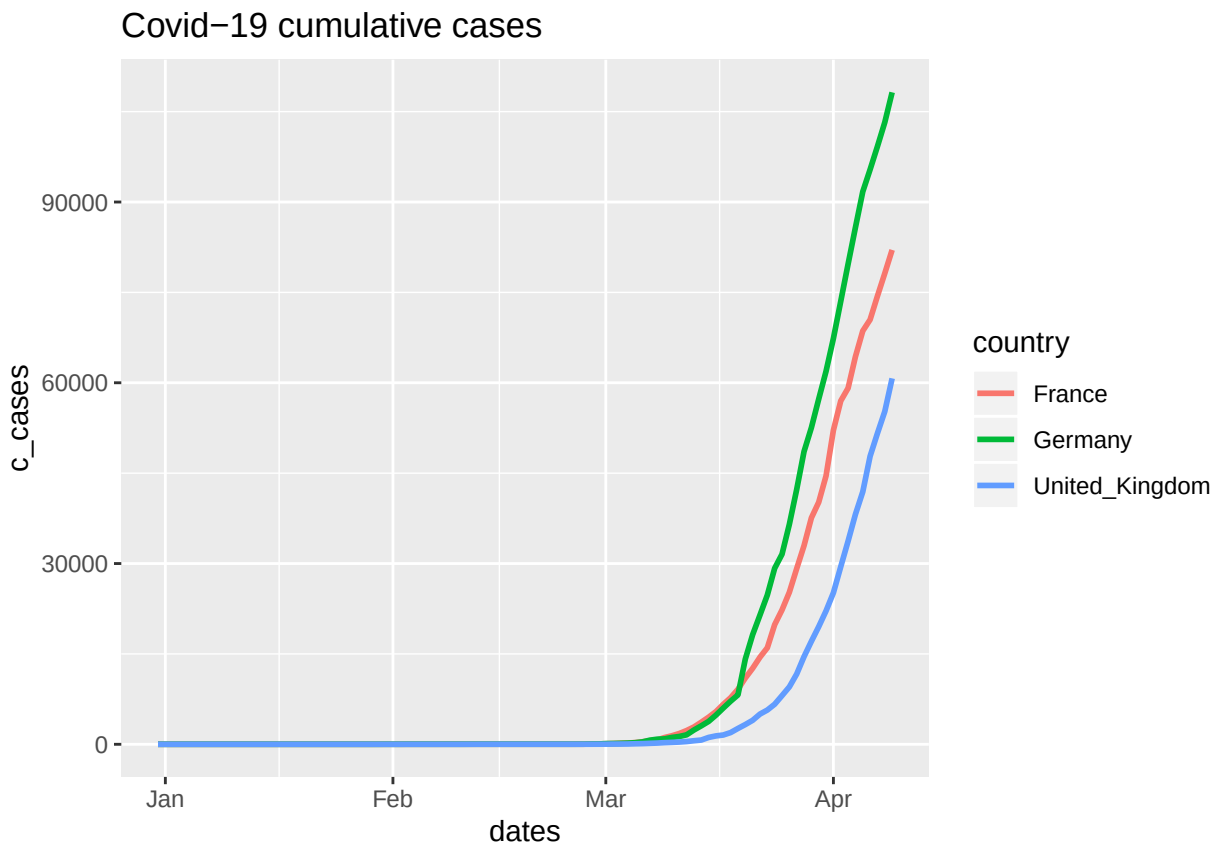
We can then plot this smoothed series.

```
sel_countries <- c("Germany", "France", "United_Kingdom")
g5a <- ggplot(subset(data, country %in% sel_countries),
              aes(x=dates,y=cases_ma, color = country)) +
      geom_line(size = 1) +
      geom_line(aes(x=dates,y=cases, color = country), size = 0.5) +
      ggtitle("Covid-19 cases, smoothed and original")
g5a
```



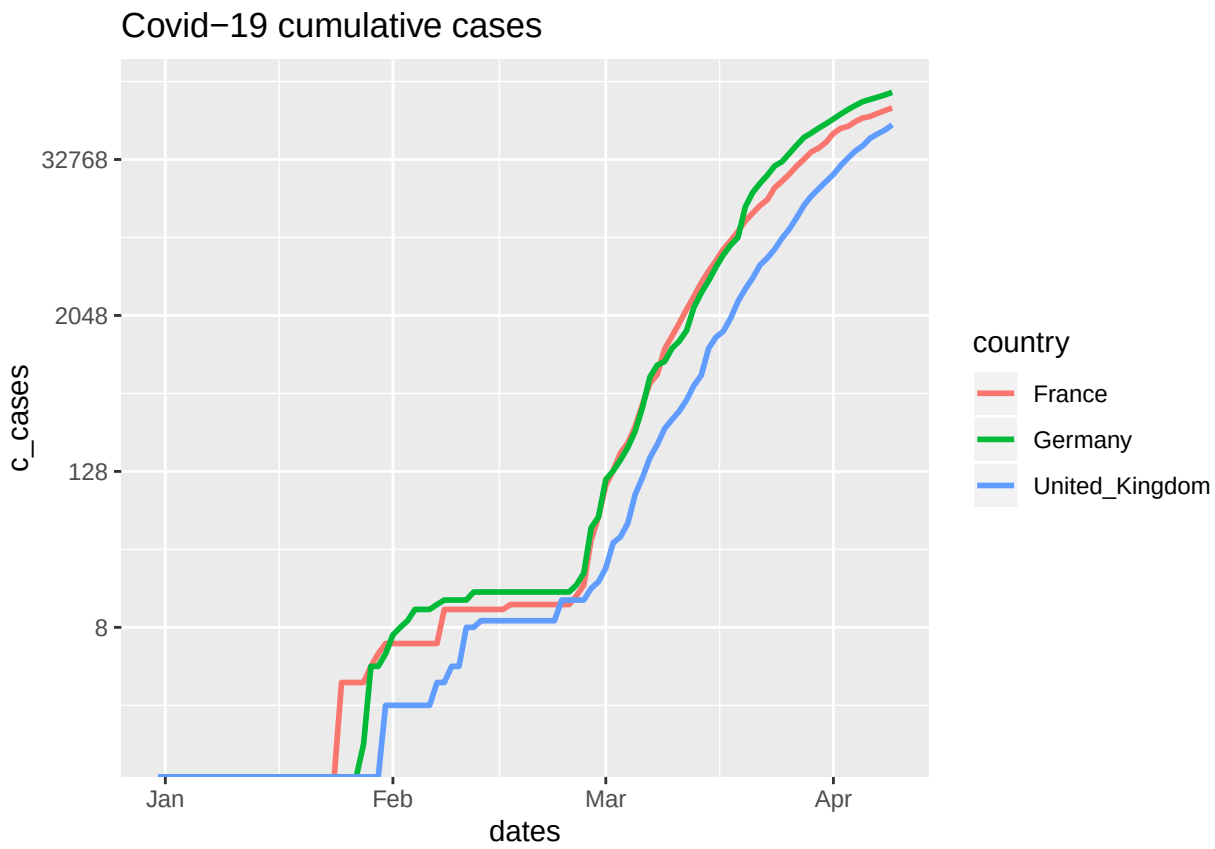
Let's also look at the cumulative case numbers.

```
sel_countries <- c("Germany", "France", "United_Kingdom")
g6 <- ggplot(subset(data, country %in% sel_countries),
              aes(x=dates,y=c_cases, color = country)) +
      geom_line(size = 1) +
      ggtitle("Covid-19 cumulative cases")
g6
```



You can clearly see the exponential growth in the number of cases. Sometimes you will see these graphs on a logarithmic scale.

```
sel_countries <- c("Germany", "France", "United_Kingdom")
g7 <- ggplot(subset(data, country %in% sel_countries),
             aes(x=dates,y=c_cases, color = country)) +
      geom_line(size = 1) +
      scale_y_continuous(trans='log2') +
      ggtitle("Covid-19 cumulative cases")
g7
```



From this you can clearly see some early dynamics which is not visible from the natural scale. But looking at the logarithmic scale takes away the dramatic impression the natural scale shows.

Some country maps

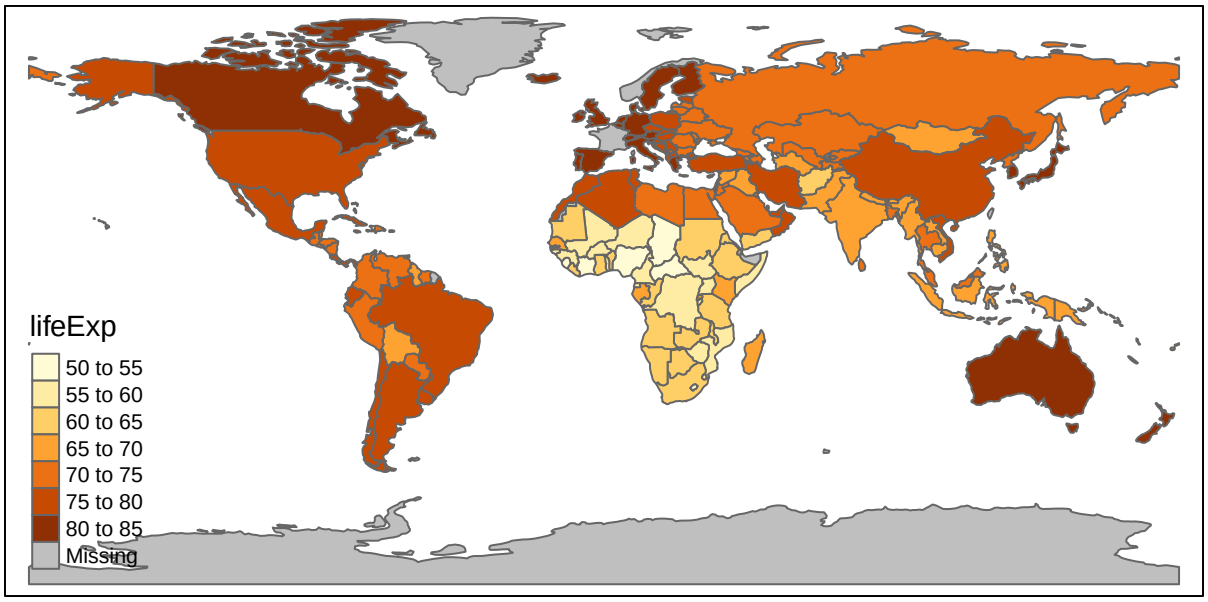
Maps are a great tool to illustrate the geographic distribution of any statistics.

We need a few more packages at this stage.

```
library(sf)
library(raster)
library(spData)
library(tmap)
```

Let's create a map first and then we will find out how to manipulate the map to display what we want.

```
# Add fill and border layers to world shape
tm_shape(world) + tm_polygons(col = "lifeExp")
```



Wow, one line of code and you get a world map which shows which countries have the highest and lowest life expectation. Amazing! What did the code do? We saved the map in a new object `m1`. We used the `tmap` package (Check out Geocomputation with R by Robin Loveace for an introduction into this package and geocomputing in general). It has a range of map information at its fingertips and `tm_shape(world)` basically called up shape information on the world's countries with some basic information. Then we add the information which should determine the colors and borders (`+ tm_polygons(col = "lifeExp")`). Clearly we will want to replace the information on life expectatncy with information on Covid-19 cases. To do this we first need to understand where tmaps stores the data on life expectatncy.

Hence we need to slightly deconstruct the above code

```
m2 <- tm_shape(world)
str(m2)
```

```
## List of 1
##  $ tm_shape:List of 12
##   ..$ shp_name   : chr "world"
##   ..$ shp        :Classes 'sf', 'tbl_df', 'tbl' and 'data.frame':    177 obs. of  11 variables:
##   .. ..$ iso_a2   : chr [1:177] "FJ" "TZ" "EH" "CA" ...
##   .. ..$ name_long: chr [1:177] "Fiji" "Tanzania" "Western Sahara" "Canada" ...
##   .. ..$ continent: chr [1:177] "Oceania" "Africa" "Africa" "North America" ...
##   .. ..$ region_un: chr [1:177] "Oceania" "Africa" "Africa" "Americas" ...
##   .. ..$ subregion: chr [1:177] "Melanesia" "Eastern Africa" "Northern Africa" "Northern America" ..
##   .. ..$ type     : chr [1:177] "Sovereign country" "Sovereign country" "Indeterminate" "Sovereign c
##   .. ..$ area_km2 : num [1:177] 19290 932746 96271 10036043 9510744 ...
##   .. ..$ pop      : num [1:177] 8.86e+05 5.22e+07 NA 3.55e+07 3.19e+08 ...
##   .. ..$ lifeExp  : num [1:177] 70 64.2 NA 82 78.8 ...
```

```
##   .. ..$ gdpPercap: num [1:177] 8222 2402 NA 43079 51922 ...
##   .. ..$ geom     :sfc_MULTIPOLYGON of length 177; first list element: List of 3
##   .. .. ..$ :List of 1
##   .. .. .. ..$ : num [1:8, 1:2] 180 180 179 179 179 ...
##   .. .. ..$ :List of 1
##   .. .. .. ..$ : num [1:9, 1:2] 178 178 179 179 178 ...
##   .. .. ..$ :List of 1
##   .. .. .. ..$ : num [1:5, 1:2] -180 -180 -180 -180 -180 ...
##   .. .. ..- attr(*, "class")= chr [1:3] "XY" "MULTIPOLYGON" "sfg"
##   .. ..- attr(*, "sf_column")= chr "geom"
##   .. ..- attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA NA NA NA NA NA
##   .. .. ..- attr(*, "names")= chr [1:10] "iso_a2" "name_long" "continent" "region_un" ...
##   ..$ name       : NULL
##   ..$ is.master  : logi NA
##   ..$ projection : NULL
##   ..$ bbox       : NULL
##   ..$ unit       : NULL
##   ..$ simplify   : num 1
##   ..$ point.per  : logi NA
##   ..$ line.center: chr "midpoint"
##   ..$ filter     : NULL
##   ..$ check_shape: logi TRUE
##  - attr(*, "class")= chr "tmap"
```

So, this looks complicated. `m2` is a list with one element called `sm_shape`. That in turn is a list with 12 elements. Think of drawers inside drawers. Importantly one of these elements (drawers inside the drawer called `tm_shape`) is called `shp` and this contains information on the respective countries.

Let's just look at that to understand what it looks like.

```
temp <- m2$tm_shape$shp
names(temp)
```

```
##  [1] "iso_a2"    "name_long" "continent" "region_un" "subregion" "type"
##  [7] "area_km2"  "pop"       "lifeExp"   "gdpPercap" "geom"
```

This is really a traditional data frame with country specific information and you can see that one of the pieces of information here is life expectation (`lifeExp`). This is where `tmap` got the info from. We will insert into this spreadsheet the information on cases and then use that to display the data. Importantly, `iso_a2` is a variable with a country appreviation. As we have this information also in our `data` set we will use that to merge the data.

We start by extracting the information we want to merge into `temp` from our original dataset `data`. Here the data for all countries on the 4 April 2020.

```
temp_mergein <- data %>% filter(dates == "2020-04-04") %>%
                         select(geoId, cases, c_cases, deaths, c_deaths)
```

When you run this you are likely to get the following error message

```
Error in (function (classes, fdef, mtable)  :
  unable to find an inherited method for function 'select' for signature '"tbl_df"'
```

I had to google the error message and found out that the select function stops working like this as the `raster` function also has a `select` function (type `?select` into the command window to see this). When we loaded the `raster` package the `select` command started pointing to the `select` function in the `raster` package rather than the select function of the `dplyr` package (automatically loaded with the `tidyverse`) which is really the function we want.

These are the issues which arise with an open source software where many people contribute different packages, like the `tidyverse` and the `raster` package and no one institution ensures that they are not using the same name for functions. In fact, look at the notices you ignored after loading the `raster` package. Most likely you will find something like "Attaching package: 'raster'. The following object is masked from 'package:dplyr': select". This could all have bee avoided by loading the `tidyverse` package after the `raster` package, but then I wouldn't have had the opportunities to expose you to one of the quirks you will encounter when you work with R.

So when we want to run the above command we have to tell R that we want the select function of the dplyr package (`dlyr::select`).

```
temp_mergein <- data %>% filter(dates == "2020-04-04") %>%
                         dplyr::select(geoId, cases, c_cases, cases_ma,
                                       deaths, c_deaths, deaths_ma)
```

We only selected the actual variables we are interested in and the `geoId` variable as this will be the variable which will be matched with `iso_a2` in the shape file.

As it turns out the country code for the United Kingdom in `data` (and hence in `temp_mergein`) is `UK` and in the shape file it is `GB`. So we need to change on of them to enable a match. If you didn't do this you would find that the map shows missing data for the United Kingdom.

```
temp_mergein$geoId <- as.character(temp_mergein$geoId)
temp_mergein$geoId[temp_mergein$geoId == "UK"] <- "GB"
```

Now we will `merge` this info into `m2` such that we have our COVID-19 data available to map. We specify the respective variables used to match the data (`by.x = "iso_a2", by.y = "geoId"`) and also ensure that we keep all of our original country info, even if there is no COVID-19 information (`all.x = TRUE`).

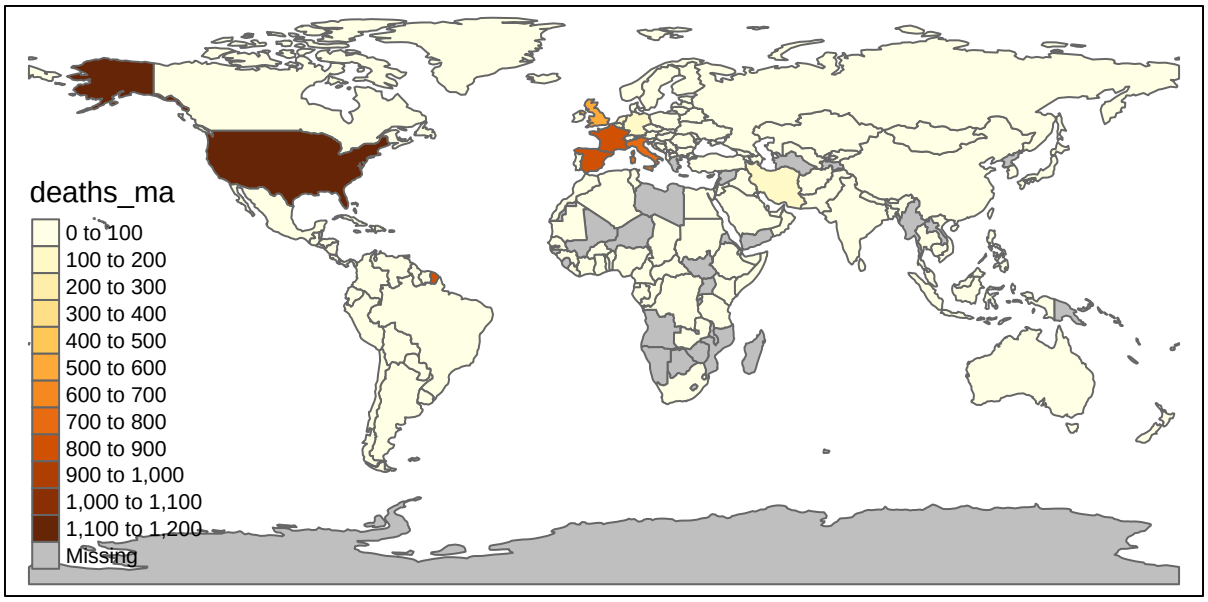
```
temp <- merge(temp, temp_mergein, by.x = "iso_a2", by.y = "geoId", all.x = TRUE)
```

Now that we manipulated the datafile at the core of the mapping operation we need to insert it back into `m2`, into exactly the same spot where we found that datafile in the first place (`m2$tm_shape$shp`).

```
m2$tm_shape$shp <- temp
```

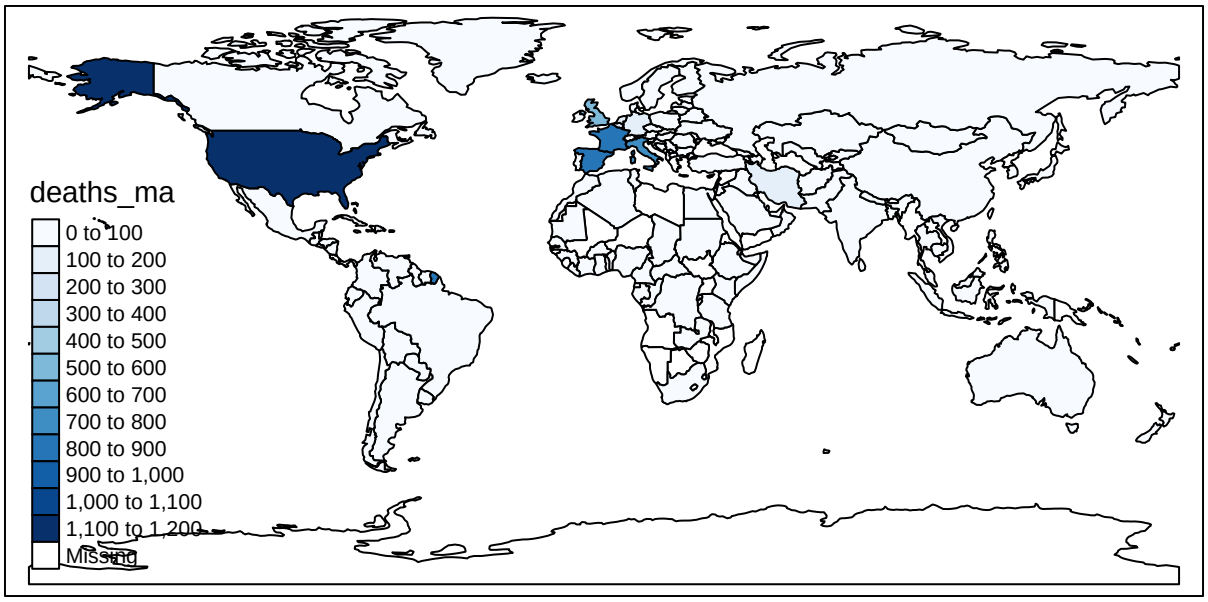
Now we can return to the mapping code displaying the smoothed number of casualties.

```
# Add polygons layer to world shape
m2 + tm_polygons(col = "deaths_ma", n=10)  # n = 10 controls the number of categories
```



There is all sorts of things you can change about these maps. For instance the colour scheme using the `tm_style` function.

```
m2 + tm_polygons(col = "deaths_ma", n=10) +  # n = 10 controls the number of categories
      tm_style("col_blind")
```



More useful maps

The above maps were created for one particular day ("04/04/2020"). Could we create multiple maps for each day in 2020 so that we can visualise the spread of the pandemic? The answer is yes. Let's start by creating the base world map again.

```
m3 <- tm_shape(world) # create a new shape file from scratch
temp3 <- m3$tm_shape$shp # extract the data frame with the data
```

First we need to ensure that we insert the daily information into `m3` just as we inserted the info for the one day. What we will do is to select one day a week, starting with the 15th of Jan 2020. This is done in `date_sel` using the sequence (`seq` function). Then we select all the information available for these dates from `data`.

```
# prepare the data from data
date_sel <- seq(as.Date("2020-01-15"), as.Date("2020-04-04"), 7)
temp_mergein <- data %>% filter(dates %in% date_sel) %>%
                  dplyr::select(geoId, dates, cases, c_cases, cases_ma,
                                deaths, c_deaths, deaths_ma) %>%
                  arrange(geoId,dates)
temp_mergein$geoId <- as.character(temp_mergein$geoId)
temp_mergein$geoId[temp_mergein$geoId == "UK"] <- "GB"
```

So far we extracted all our data, just for more dates. We are now also keeping the `dates` variable.

Now we need to merge the new data into `temp3`.

```
temp3 <- merge(temp3, temp_mergein, by.x = "iso_a2", by.y = "geoId", all.x = TRUE)
temp3$dates <- as.character(temp3$dates)  # tmaps doesn't like date formats I think
temp3 <- temp3 %>%  filter(!is.na(dates)) # delete countries with no data
```

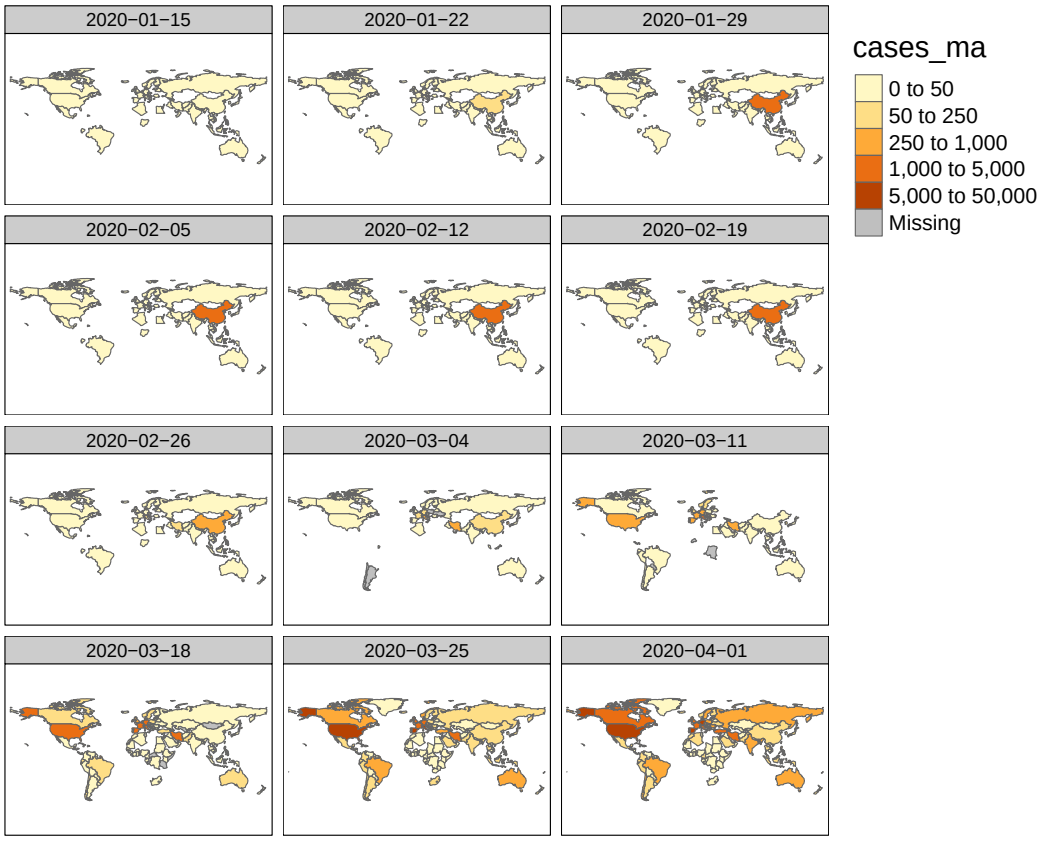
Here we turned the dates into the character format (as `tmap` seemed to dislike date formats) and we now delete countries with no available observations as these would otherwise introduce an extra `na` date.

This updated data frame is now inserted back into the shape file.

```
m3$tm_shape$shp <- temp3
```

Now we need to instruct `tmap` to produce the maps, but now a separate map for every day. The `tmap` command which is useful here is `tm_facets(by = "dates", free.coords = FALSE)`. The `by = "dates"` option selects the `dates` variable to be the facet variable, i.e. the variable for which we calculate separate maps for every possible outcome (i.e. day).

```
covid_fac <- m3 + tm_polygons(col = "cases_ma",
                              style = "fixed", # use fixed categories
                              breaks = c(0,50,250,1000,5000,50000)) + # set category boundaries
                  tm_facets(by = "dates", free.coords = FALSE)
covid_fac
```



The result of this calculation was saved in `covid_fac`. As you can see, we do not have an observation from every country on all days. If we are missing an observation then, for the particular, day the country will not appear on the map.

> Explore: Use `?tm_polygons` to learn about some of the adjustments you can undertake for thes graphs. In particular experiment with the way in which the colours are defined. Replace, in the

above commands, `style = "fixed", breaks = c(0,50,250,1000,5000,50000)` with `style = "cont"` or `style = "log10_pretty"` and judge whether any way to set the colour scale is equally useful.

Correlate Covid-19 data with other indicators

In this section we will link some other data to these Covid-19 data. In the Kaggle data competition you will find a range of dataset which have been proposed to provide potentially useful insights into how to best handle the pandemic. Amongst these are:

- number of international tourism arrivals
- percentage of population with access to hand-washing facilities
- number of hospital beds
- number of nurses and midwives

The first dataset mainly as it may allow us to find a proxy for how the pandemic spreads around the world. The others as they relate to how well a country may either prevent the spread of the virus (handwashing facilities) and how well their health system may be able to cope with any significant outbreak.

You may want to look at the submissions to the Kaggle competition to see how some of these have been used.

International travel

Let's load some international travel data. We can get some data on flight connection information (from 2014, no later info is available) from the Open Flights website. Find the "routes.dat" datafile which is a csv file without column headings which is why we need to add the column names in the import process.

- Airline 2-letter (IATA) or 3-letter (ICAO) code of the airline.
- Airline ID Unique OpenFlights identifier for airline (see Airline).
- Source airport 3-letter (IATA) or 4-letter (ICAO) code of the source airport.
- Source airport ID Unique OpenFlights identifier for source airport (see Airport)
- Destination airport 3-letter (IATA) or 4-letter (ICAO) code of the destination airport.
- Destination airport ID Unique OpenFlights identifier for destination airport (see Airport)
- Codeshare "Y" if this flight is a codeshare (that is, not operated by Airline, but another carrier), empty otherwise.
- Stops Number of stops on this flight ("0" for direct)
- Equipment 3-letter codes for plane type(s) generally used on this flight, separated by spaces

```
data_routes <- read_csv("routes.dat",
                        col_names = c("Airline","Airline.ID","Source.Airport","Source.Airport.ID",
                                      "Destination.Airport","Destination.Airport.ID","Codeshare",
                                      "Stops","Equipment"))
```

This could be potentially useful as it would allow us to get information on travel routes which capture the potential routes of transmission. To make this useful we will have to add the country information for the airports. The airport.dat file is also available from Open Flights.

```
data_airports <- read_csv("airports.dat",
                        col_names = c("Airport.ID","Airport","Airport.City","Country",
                                      "IATA.ID","ICAO.ID","Lat","Long","Altitude","TZ", "DST",
                                      "TZ2","Type","Source"))
```

Let's explore the data by looking at all the routes connecting Manchester, UK, with Paris, France.

We need to find the `IATA.ID` for Manchester and Paris airports.

```
data_airports %>% filter(Airport.City == "Manchester")
```

```
## # A tibble: 2 x 14
##   Airport.ID Airport Airport.City Country IATA.ID ICAO.ID   Lat  Long Altitude
##        <dbl> <chr>   <chr>        <chr>   <chr>   <chr>   <dbl> <dbl>    <dbl>
## 1        478 Manche~ Manchester   United~ "MAN"   EGCC     53.4 -2.27      257
## 2       9860 City A~ Manchester   United~ "\\N"   EGCB     53.5 -2.39       73
## # ... with 5 more variables: TZ <dbl>, DST <chr>, TZ2 <chr>, Type <chr>,
## #   Source <chr>
```

```
data_airports %>% filter(Airport.City == "Paris")
```

```
## # A tibble: 4 x 14
##   Airport.ID Airport Airport.City Country IATA.ID ICAO.ID   Lat   Long Altitude
##        <dbl> <chr>   <chr>        <chr>   <chr>   <chr>   <dbl>  <dbl>    <dbl>
## 1       1380 Paris-~ Paris        France  LBG     LFPB     49.0   2.44      218
## 2       1382 Charle~ Paris        France  CDG     LFPG     49.0   2.55      392
## 3       1386 Paris-~ Paris        France  ORY     LFPO     48.7   2.38      291
## 4      11095 Cox Fi~ Paris        United~ PRX     KPRX     33.6 -95.5       547
## # ... with 5 more variables: TZ <dbl>, DST <chr>, TZ2 <chr>, Type <chr>,
## #   Source <chr>
```

So, `MAN` and `CDG` (for Charles de Gaulle). Let's find the route information (recall that we now need to use `dplyr::select`).

```
data_routes %>% filter(Source.Airport == "MAN" & Destination.Airport == "CDG") %>%
  dplyr::select(Airline, Source.Airport, Destination.Airport, Codeshare)
```

```
## # A tibble: 4 x 4
##   Airline Source.Airport Destination.Airport Codeshare
##   <chr>   <chr>          <chr>               <chr>
## 1 AF      MAN            CDG                 <NA>
## 2 AZ      MAN            CDG                 Y
## 3 BE      MAN            CDG                 <NA>
## 4 LS      MAN            CDG                 <NA>
```

So in 2014 there were connections by 4 airlines between Manchester and Paris (Charles de Gaulle Airport). What is not obvious from these data is how many daily flights were offered by a particular airline. And really we would be interested in the passenger capacity as well. Also, the Allitalia flight (`AZ`) was not a real flight but codeshared with Air France (`AF`).

We will want to calculate some information which indicates the degree of connectedness between countries as we think that this may give a proxy for how likely it is that a virus may spread from one country to another.

We start by removing codeshare connections from the data base (we are keeping those where the entry is `NA`).

```
data_routes <- data_routes %>%  filter(is.na(Codeshare))
```

Now we are adding source and destination country information to `data_routes`. We prepare two airport files with only the `IATA.ID`, which will be matched to `Source.Airport` and `Destimation.Airport` in `data_routes`, and the country. We create two to make the merging process more straightforward.

```
airports_s_merge <- data_airports %>% dplyr::select(IATA.ID, Country)
names(airports_s_merge) <- c("Source.Airport", "Source.Country")

airports_d_merge <- data_airports %>% dplyr::select(IATA.ID, Country)
names(airports_d_merge) <- c("Destination.Airport", "Destination.Country")
```

```
data_routes <- merge(data_routes,airports_s_merge)
data_routes <- merge(data_routes,airports_d_merge)
```

If we are now looking again at our Manchester to Paris connections we get

```
data_routes %>% filter(Source.Airport == "MAN" & Destination.Airport == "CDG") %>%
  dplyr::select(Airline, Source.Airport, Source.Country,
                Destination.Airport, Destination.Country)
```

```
##   Airline Source.Airport Source.Country Destination.Airport Destination.Country
## 1      LS            MAN United Kingdom                 CDG              France
## 2      AF            MAN United Kingdom                 CDG              France
## 3      BE            MAN United Kingdom                 CDG              France
```

Now we can accumulate the number of connections between countries.

```
connections <- data_routes %>% group_by(Source.Country, Destination.Country) %>%
                summarise(connect = n()) %>% arrange(Source.Country,-connect)

connections_CH <- connections %>% filter(Source.Country == "China")
head(connections_CH)
```

```
## # A tibble: 6 x 3
## # Groups:   Source.Country [1]
##   Source.Country Destination.Country connect
##   <chr>          <chr>                 <int>
## 1 China          China                  5698
## 2 China          Taiwan                  164
## 3 China          South Korea             136
## 4 China          Hong Kong               110
## 5 China          Japan                    89
## 6 China          Thailand                 71
```

Here we can see that the most connection to the outside have been to other South-East Asian countries.

Perhaps we can even check to which countries the city of Wuhan is best connected

```
data_airports %>% filter(Airport.City == "Wuhan")
```

```
## # A tibble: 1 x 14
##   Airport.ID Airport Airport.City Country IATA.ID ICAO.ID   Lat  Long Altitude
##        <dbl> <chr>   <chr>        <chr>   <chr>   <chr>   <dbl> <dbl>    <dbl>
## 1       3376 Wuhan ~ Wuhan        China   WUH     ZHHH     30.8  114.      113
## # ... with 5 more variables: TZ <dbl>, DST <chr>, TZ2 <chr>, Type <chr>,
## #   Source <chr>
```

So the Airport code is `WUH`.

```
connections_WUH <- data_routes %>% filter(Source.Airport == "WUH") %>%
                   group_by(Source.Airport, Destination.Country) %>%
                   summarise(connect = n()) %>%
                   arrange(-connect)  # orders in decreasing order

head(connections_WUH)
```

```
## # A tibble: 6 x 3
## # Groups:   Source.Airport [1]
##   Source.Airport Destination.Country connect
##   <chr>          <chr>                 <int>
```

```
## 1 WUH            China                142
## 2 WUH            Taiwan                 6
## 3 WUH            Hong Kong              4
## 4 WUH            South Korea            3
## 5 WUH            Thailand               3
## 6 WUH            Singapore              2
```

So the picture is pretty much the same as for China as a whole. **But remember that the connection data are from 2014.**

Predicting the spread

Perhaps we can build a prediction tool of how the pandemic spreads. Basically we are building a transition matrix. We are using the moving average of daily cases (`cases_ma`) and combine them with the measure of strength of travel connection to think about where it is likely that the virus would spread. There are of course many other factors which play into the spread of a virus. But here we will abstract from these. Even acknowledging this, it is apparent that travel connections are deemed to be important as many countries have now introduced severe travel restrictions. This also implies that using travel routes to anticipate the spread is likelly to onlly be useful in the early stages of a pandemic when such restrictions are not yet in place.

Let's look at all the countries in our sample for which we have any virus data (in `data`) and any international flight connection data in `connections`.

```
countries_c <- unique(connections$Source.Country) # countries in connection
length(countries_c)
```

```
## [1] 223
```

```
countries_d <- unique(as.character(data$country)) # countries in data, convert from factor to chr
length(countries_d)
```

```
## [1] 164
```

```
cou_list <- intersect(countries_c,countries_d)
length(cou_list)
```

```
## [1] 127
```

`cou_list` now contains 127 countries. These are 37 fewer countries than we have virus data for. It may well be that some of these are because countries in the two datasets have different spelling. To check whether this is likely we will look at the set of 37 countries which have been excluded from the `countries_d` set.

```
setdiff(countries_d,cou_list)
```

```
##  [1] "Cases_on_an_international_conveyance_Japan"
##  [2] "Czechia"
##  [3] "Dominican_Republic"
##  [4] "Monaco"
##  [5] "New_Zealand"
##  [6] "North_Macedonia"
##  [7] "San_Marino"
##  [8] "South_Korea"
##  [9] "Sri_Lanka"
## [10] "United_Arab_Emirates"
## [11] "United_Kingdom"
## [12] "United_States_of_America"
## [13] "Andorra"
## [14] "Saudi_Arabia"
```

```
## [15] "Liechtenstein"
## [16] "Bosnia_and_Herzegovina"
## [17] "Palestine"
## [18] "South_Africa"
## [19] "Costa_Rica"
## [20] "Holy_See"
## [21] "Brunei_Darussalam"
## [22] "Burkina_Faso"
## [23] "Democratic_Republic_of_the_Congo"
## [24] "Cote_dIvoire"
## [25] "Trinidad_and_Tobago"
## [26] "Antigua_and_Barbuda"
## [27] "Equatorial_Guinea"
## [28] "Eswatini"
## [29] "Saint_Lucia"
## [30] "Central_African_Republic"
## [31] "Congo"
## [32] "Kosovo"
## [33] "United_Republic_of_Tanzania"
## [34] "El_Salvador"
## [35] "French_Polynesia"
## [36] "Cayman_Islands"
## [37] "Faroe_Islands"
```

Amongst these are a number of large and important countries which we would certainly want to take account off. For these countries we need to equalise the spellings in the `connections` and `data` datasets and then repeat the exercise which identifies the set of common countries. We will change the names in the `connections` dataset to fit the names in the `data` dataset.

```
connections[connections=="Burkina Faso"]<-"Burkina_Faso"
connections[connections=="Bosnia and Herzegovina"]<-"Bosnia_and_Herzegovina"
connections[connections=="Czech Republic"]<-"Czechia"
connections[connections=="Cote d'Ivoire"]<-"Cote_dIvoire"
connections[connections=="Tanzania"]<-"United_Republic_of_Tanzania"
connections[connections=="El Salvador"]<-"El_Salvador"
connections[connections=="Trinidad and Tobago"]<-"Trinidad_and_Tobago"
connections[connections=="Cote dIvoire"]<-"Cote_dIvoire"
connections[connections=="Brunei"]<-"Brunei_Darussalam"
connections[connections=="Costa Rica"]<-"Costa_Rica"
connections[connections=="Sri Lanka"]<-"Sri_Lanka"
connections[connections=="Saudi Arabia"]<-"Saudi_Arabia"
connections[connections=="South Africa"]<-"South_Africa"
connections[connections=="South Korea"]<-"South_Korea"
connections[connections=="United Kingdom"]<-"United_Kingdom"
connections[connections=="United States"]<-"United_States_of_America"
connections[connections=="New Zealand"]<-"New_Zealand"
```

```
countries_c <- unique(connections$Source.Country) # countries in connection
length(countries_c)
```

```
## [1] 223
```

```
countries_d <- unique(as.character(data$country)) # countries in data, convert from factor to chr
length(countries_d)
```

```
## [1] 164
```

```
cou_list <- intersect(countries_c,countries_d)
length(cou_list)
```

```
## [1] 143
```

In `countries` we now have a list of alphabeticaly ordered countries for which we will proceed with our analysis.

Let's consider the state of affairs on the same twelve days we used earlier to display the spread of the virus.

```
date_sel <- seq(as.Date("2020-01-15"), as.Date("2020-04-04"), 7)
```

First we will select all observations for the countries in `cou_list` and the dates in `date_sel`. Then we will organise the data such that the dates are variable names (using the `spread` function)

```
cou_list <- as.factor(cou_list)
spread <- data %>%  dplyr::filter(country %in% cou_list & dates %in% date_sel)  # only countries in cou
spread <- spread %>% dplyr::select(dates,country, geoId, cases_ma) %>%
                        spread(dates,cases_ma)

spread[is.na(spread)] <- 0 # Replace all misssing information with 0s
spread$country <- as.character(spread$country)
```

Have a look at the spread dataframe to understand what it looks like!

We proceed as follows. We take the data on current infections and multiply the number with the number of connections. We do this in a double loop. For every (receiving) country we will calculate the potential contribution of receiving the virus (cases in the source country x number of connections). All of these contributons are then summed up. **The resulting number has no direct interpretation, but the larger the number the more likely that the virus will travel into that country**.

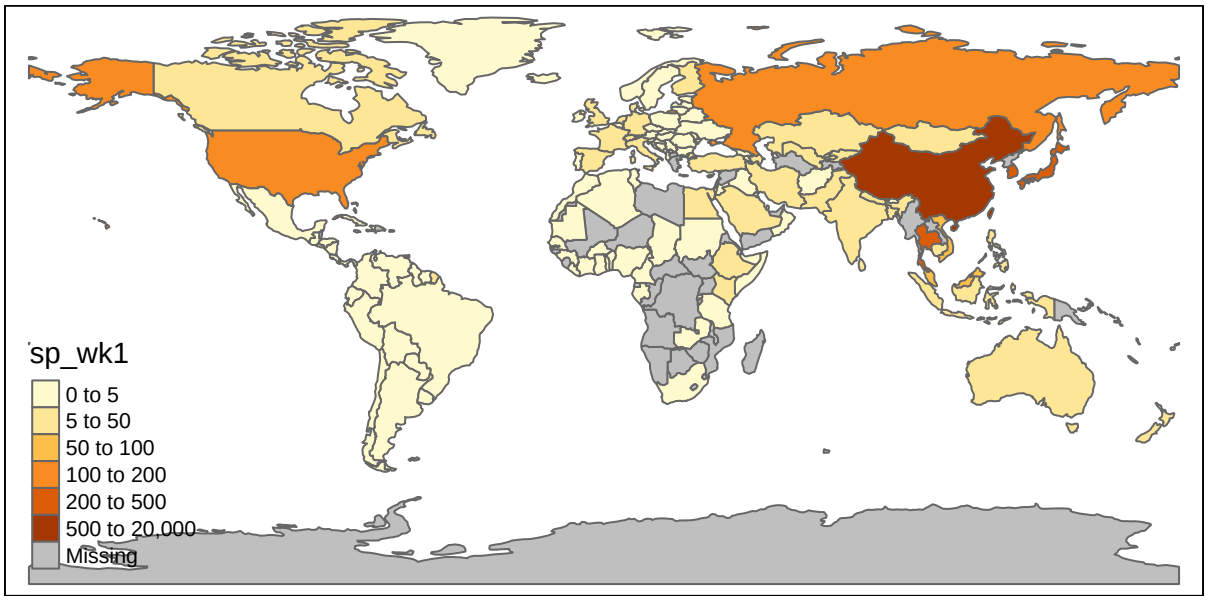
```
spread$sp_wk1 <- 0

for (i in cou_list) { # loop for destimation country

  trans_fac <- 0  # we will accumulate the transmission factors here

  for (j in cou_list) { # loop for source country
    # pick the connection strength
    conn <- connections$connect[connections$Source.Country == j &
                                  connections$Destination.Country == i]
    if (length(conn) == 0) {conn <- 0}   # ensure 0 if no connection

    # pick current cases in source country (1st date in col 2)
    # and calculate transmission factor from country j to country i
    supp <- spread[spread$country == j,as.character(date_sel[1])] * conn

    trans_fac <- trans_fac + supp   # summing up the transmission factors
  }

  # save the accumulated transmission to country i
  # trans_fac is a list (with one element), unlist extracts the actual value only
  spread$sp_wk1[spread$country == i] <- unlist(trans_fac)
}
```

The way in which this approach was implemented is a very inefficient way (using nested loops). Computationally loops tend to take up a lot of computing time and mathematically more pleasing and computationally

more efficient ways are available using matrix algebra. However, implementing these here would merely distract from the purpose of illustrtaing a principle.

We will now plot the resulting information on a map basically replicating the earlier mapping operations.

```
m4 <- tm_shape(world)
temp <- m4$tm_shape$shp
temp_mergein <- spread %>% dplyr::select(geoId,sp_wk1)
temp_mergein$geoId <- as.character(temp_mergein$geoId)
temp_mergein$geoId[temp_mergein$geoId == "UK"] <- "GB"
temp <- merge(temp, temp_mergein, by.x = "iso_a2", by.y = "geoId", all.x = TRUE)
m4$tm_shape$shp <- temp
m4 + tm_polygons(col = "sp_wk1",
                 style = "fixed",
                 breaks = c(0,5,50,100,200,500,20000))
```



In that first week the only country with cases was China. The region which, using air transport as the only transmission mechanism, is most likely to be affected next, is South-East Asia, in particular Japan, South Korea and Taiwan which have the closest travel connections with China.

Let's look at the ten top countries or terretories susceptible from this analysis.

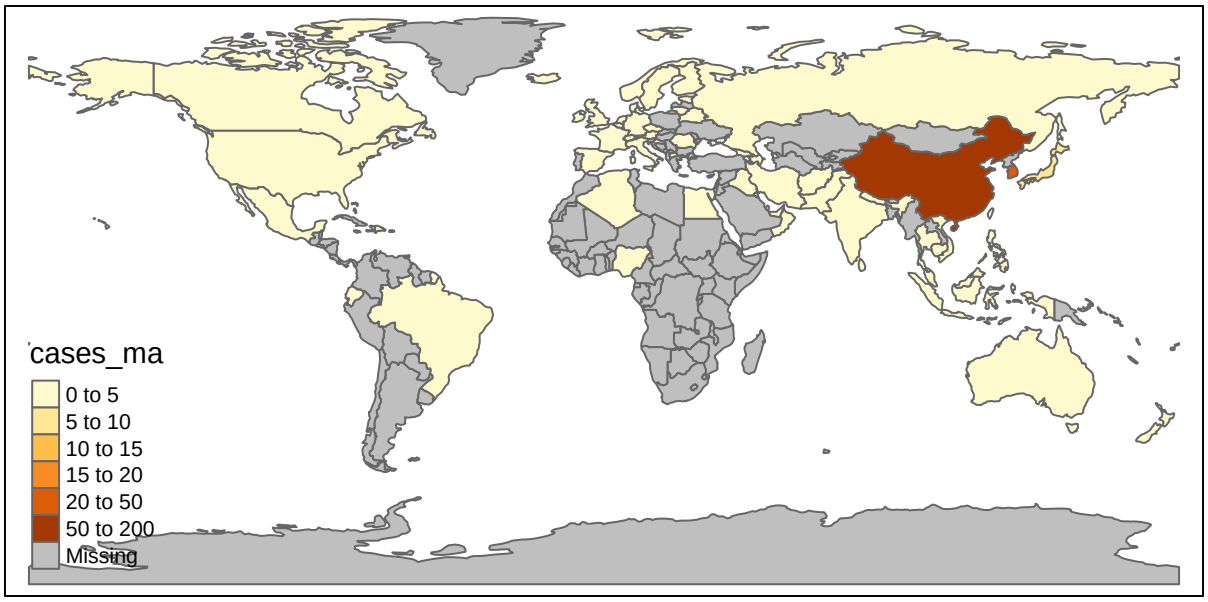
```
# name_long is country/terretory name
temp %>% arrange(-sp_wk1) %>%  dplyr::select(name_long, sp_wk1)
```

```
## Simple feature collection with 177 features and 2 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -180 ymin: -90 xmax: 180 ymax: 83.64513
```

```
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
## First 10 features:
##             name_long      sp_wk1                       geometry
## 1               China 17126.85714 MULTIPOLYGON (((109.4752 18...
## 2              Taiwan   502.28571 MULTIPOLYGON (((121.7778 24...
## 3   Republic of Korea   423.57143 MULTIPOLYGON (((126.1748 37...
## 4               Japan   341.57143 MULTIPOLYGON (((141.8846 39...
## 5            Thailand   256.42857 MULTIPOLYGON (((105.2188 14...
## 6  Russian Federation   139.57143 MULTIPOLYGON (((178.7253 71...
## 7       United States   108.28571 MULTIPOLYGON (((-122.84 49,...
## 8            Malaysia    85.85714 MULTIPOLYGON (((100.0858 6....
## 9             Vietnam    67.85714 MULTIPOLYGON (((104.3343 10...
## 10          Australia    48.57143 MULTIPOLYGON (((147.6893 -4...
```

Now we will compare this to the actual spread of infections five weeks later (`dates == date_sel[6]`, 19 Feb 2020, the first week when notable numbers of cases outside China were identified).

```
m5 <- tm_shape(world)
temp <- m5$tm_shape$shp
temp_mergein <- data %>% filter(dates == date_sel[6]) %>%
                         dplyr::select(geoId, cases, c_cases, cases_ma,
                                       deaths, c_deaths, deaths_ma)
temp_mergein$geoId <- as.character(temp_mergein$geoId)
temp_mergein$geoId[temp_mergein$geoId == "UK"] <- "GB"
temp <- merge(temp, temp_mergein, by.x = "iso_a2", by.y = "geoId", all.x = TRUE)
m5$tm_shape$shp <- temp
m5 + tm_polygons(col = "cases_ma",
                 style = "fixed",
                 breaks = c(0,5,10,15,20,50,200))
```



And let's look at the 10 countries with the 10 highest smoothed infections on the 19 Feb 2020.

```
temp %>% arrange(-cases_ma) %>%  dplyr::select(name_long, cases_ma)
```

```
## Simple feature collection with 177 features and 2 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -180 ymin: -90 xmax: 180 ymax: 83.64513
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
## First 10 features:
##               name_long     cases_ma                       geometry
## 1                 China 1401.4285714 MULTIPOLYGON (((109.4752 18...
## 2     Republic of Korea   45.2857143 MULTIPOLYGON (((126.1748 37...
## 3                 Japan    9.5714286 MULTIPOLYGON (((141.8846 39...
## 4         United States    2.8571429 MULTIPOLYGON (((-122.84 49,...
## 5                  Iran    2.5714286 MULTIPOLYGON (((48.56797 29...
## 6                 Italy    2.0000000 MULTIPOLYGON (((10.4427 46....
## 7                Taiwan    1.1428571 MULTIPOLYGON (((121.7778 24...
## 8             Australia    0.8571429 MULTIPOLYGON (((147.6893 -4...
## 9  United Arab Emirates    0.4285714 MULTIPOLYGON (((51.57952 24...
## 10               Canada    0.1428571 MULTIPOLYGON (((-122.84 49,...
```

Indeed the three highest (after China) all apear in the above list. Remember, also, that these are identified infections not the actual numbers which are unknown. Altogether five of these countries appear in the above list of top 10 countries.

This was, of course, a very simplified analysis. But perhaps it was powerful enough to realise that using such

data as transport links can be a powerful input into any analysis of how a pandemic can spread. This is also apparent from a literature review of mathematical models of how a global pandemic spreads (Walters et al., 2018, Modelling the global spread of diseases: A review of current practice and capability, Epidemics, 25) which identifies that all such models make use of some information on transport links.

You can also see that such models have potential to simulate the effect of travel restrictions. You could for instance set an element in the `connections` data to 0 to emulate the reduction of air links to 0. Clearly there are many more things to consider, in particular for the later spread of a pandemic when many countries implement multiple policies (school closures, stay at home orders etc.) which any model would need to take into account.