coreecon
Economics for a changing world

COVID-19: Some challenges, some data

Ralf Becker, University of Manchester
Eileen Tipoe, University of Oxford/CORE

Note to instructors

This project was written by the authors of the ebook *Doing Economics* (https://tinyco.re/1425213). *Doing Economics* aims to help its users to develop important data handling and data analysis skills by exposing them to policy-relevant data.

The COVID-19 pandemic is one of the most relevant issues in today's society. While the skills at the forefront of coping with and understanding this pandemic are those of nurses, carers, cleaners, pharmacists and doctors in the health system, researchers in the field of biochemistry researching for medication and vaccines and frontline staff in supermarkets and their supply chain, it is important to understand that all of these professions, in the background, are supported by people with excellent data skills. They need not be economists, but many economists do have the data skills required to make significant contributions in extraordinary times like these.

In this project, we help readers get to grips with some of the global data on the COVID-19 pandemic. Readers can import up-to-date case and fatality data, clean the dataset, understand some of the shortcomings of available data, and visualise the data in charts and maps. We also use some data on international flight routes to illustrate how the pandemic could initially spread along transport routes.

Throughout the project, readers can pick up important data skills. In times like this, more than ever, it pays to have a good understanding of data and what it can/cannot measure.

Table of Contents

Note:
1. This project requires some basic knowledge of R and is aimed at users who want to learn new techniques. Complete beginners can refer to Project 1 in *Doing Economics* (https://tinyco.re/1425810) for help with installing and getting started in R.
2. This project was written in the first week of April 2020, using data available at that time. **Your results and charts will look slightly different if you are using more updated data.**

Introduction

Worldwide, there is a huge effort being undertaken by specialists of all fields to understand and reduce the impact of this pandemic, as well as devising measures to help us all live under these new conditions.

In this project we will investigate some of the data related to the COVID-19 pandemic, focusing on the challenges and questions that this data can help to answer.

Learning objectives

In this project, you will:

- import data directly from websites (Section 1.2)
- perform some basic data cleaning techniques (Section 1.3)
- produce line charts using `ggplot` (Section 1.4)
- create smoothed versions of daily time series data (Section 1.4)
- produce maps illustrating the global spread of COVID-19 cases (Sections 1.5-1.6)
- use international flight data to identify potential transmission routes (Section 2.1)
- make a simple model to predict the spread of a pandemic (Section 2.2)

1. Some exploratory data analysis

Let's do some exploratory analysis using a dataset published by the European Centre for Disease Control (ECDC) (https://tinyco.re/4826169).

1.1 Getting started

For Sections 1.2 to 1.4, you will need the following packages, which we will install and import now:

```
install.packages(c("sets", "forecast", "readxl", "tidyverse", "ggplot2", "utils",
"httr"))
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
library(utils).        # for reading data into R # for reading data into R
library(httr)          # for downloading data from a URL
```

1.2 Import the data into R

Very helpfully, the ECDC webpage that contains the data (https://tinyco.re/7709786) provides an R script (shown in the next code block) that allows you to download the most current dataset. You could download the dataset to your computer and then import it into R instead, but here the ECDC has built a direct pipeline into their data.

After running the code below, a datafile called **data** will appear in your environment. It will contain up-to-date case and fatality data.

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

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

Note: By uploading data in this way you will always get the latest data. This means that, if you replicate this code, you will have more recent data than the data used when this project was written (9 April 2020).

1.3 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 can work with.

```
str(data)

## 'data.frame':    9922 obs. of  10 variables:
##  $ dateRep                : Factor w/ 102 levels "01/01/2020","01/02/2020",..:
40 36 32 28 24 20 16 12 8 4 ...
##  $ day                    : int  10 9 8 7 6 5 4 3 2 1 ...
##  $ month                  : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ year                   : int  2020 2020 2020 2020 2020 2020 2020 2020 2020 2
020 ...
##  $ cases                  : int  61 56 30 38 29 35 0 43 26 25 ...
##  $ deaths                 : int  1 3 4 0 2 1 0 0 0 0 ...
##  $ countriesAndTerritories: Factor w/ 206 levels "Afghanistan",..: 1 1 1 1 1 1
1 1 1 1 ...
##  $ geoId                  : Factor w/ 205 levels "AD","AE","AF",..: 3 3 3 3 3 3
3 3 3 3 ...
##  $ countryterritoryCode   : Factor w/ 203 levels "","ABW","AFG",..: 3 3 3 3 3 3
3 3 3 3 ...
##  $ popData2018            : int  37172386 37172386 37172386 37172386 37172386 3
7172386 37172386 37172386 37172386 37172386 ...
```

It is obvious what some of the variables mean, such as **day**, **month**, **year**, **countriesAndTerritories** and **popData2018** (the population of the respective country in 2018). **geoId** and **countryterritoryCode** are common abbreviations for the respective country.

First, we will 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 variable **dates** is currently a factor variable, but we want R to know that each string represents a date. Dates are of the format day/month/year e.g. 24/01/2020. In the **format** option we specify this format (**format = "%d/%m/%Y"**) so R can correctly translate the strings to dates

(see this page (https://tinyco.re/8606284)) to understand how to correctly specify the `format` option for other date formats).

```
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, but from the dataset alone it is not obvious whether these variables are cumulative (i.e. all the COVID-19 cases identified in a country up to that day), or only the cases identified on that particular day. You could either go back to the data source to find an explanation or we could plot the data and use our understanding of what the data should look like.

To investigate this issue, let's pick one country – we chose China, where this particular virus was first identified. We use the `plot` function, which is already built into R. Later we will use `ggplot` to produce nicer charts.

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

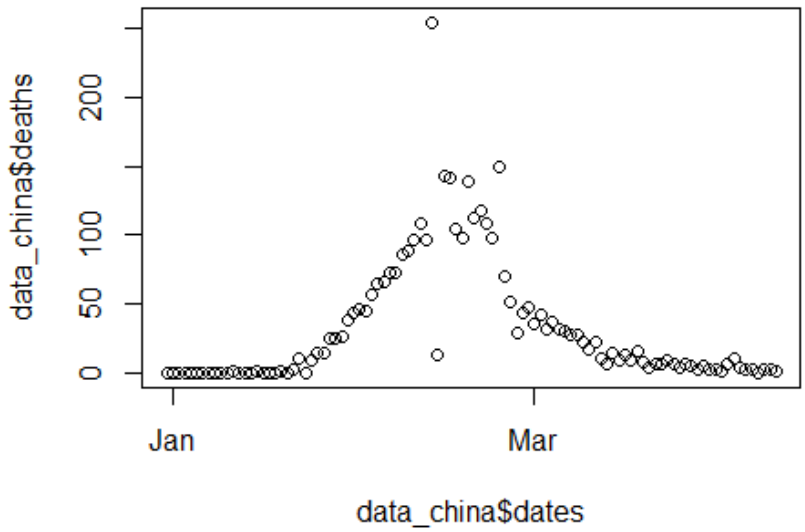

Figure 1.1 Using `plot`: Deaths in China of people who tested positive for coronavirus (1 January 2020 - 7 April 2020).

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 data, it would not be possible for the numbers to decrease. So, these must be the deaths which occurred on a particular day. You can find the same conclusion is true for cases.

Before continuing we may also want to highlight a particular feature of these variables. You can see that in the middle of March there is one day (13 Feb 2020) in 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 variation is a combination of the way that a positive COVID-19 case was defined[1], and the fact that there may be daily variation in the underlying testing activity. Therefore, any short-term variation in identified

[1] As detailed in this CNBC article (https://tinyco.re/8737043), on February 12, Chinese authorities revised national guidelines regarding what should be counted as a COVID-19 case.

cases is most likely a function of variation in testing activity, which masks a steadier development of the underlying cases.

We may also want to use our daily data to calculate the accumulated infections and deaths. This is achieved with the `cumsum` function (short for 'cumulative sum'). 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
```

You can see that each number in the sequence is the sum of all the preceding numbers (including itself), for example, we got the fifth number, 15, by adding 0, 0, 2, 4, and 9.

Before we apply the `cumsum` function to our dataset, we need to make sure that we cumulate separately by country (`group_by(country)`) and that the data are arranged by date (`arrange(dates)`).

We will store the accumulated cases and deaths by country in the variables `c_cases` and `c_deaths`. Finally, we use `ungroup()` to undo the `group_by` function (we will need the ungrouped data later).

```
data <- data %>% group_by(country) %>%
          arrange(dates) %>%
          mutate(c_cases = cumsum(cases), c_deaths = cumsum(deaths)) %>%
          ungroup()
```

1.4 Plotting line charts for some countries

Let's create some charts to describe the development of the pandemic in different countries. We initially continue with the data for China.

First, we replicate Figure 1.1 using the `ggplot` function, which produces much nicer charts. We will create the chart and save it as the object `g1`, then display it by just calling `g1`. Unlike in Section 1.3, we will use the `subset` function instead of creating a separate dataset containing only the data for China.

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

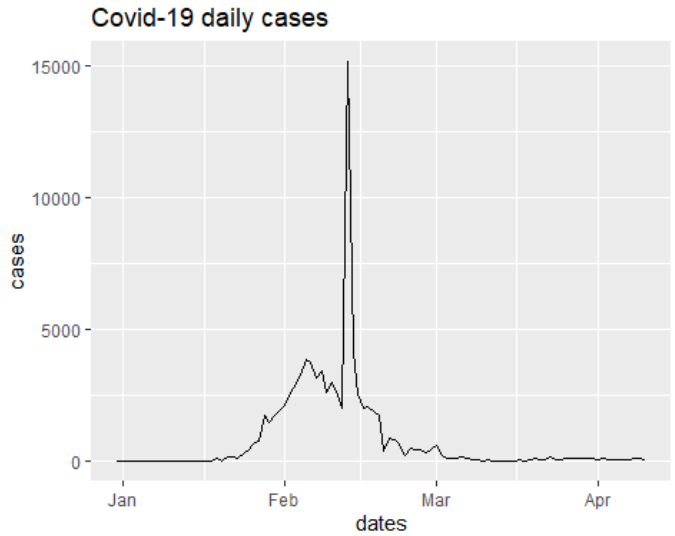

Figure 1.2 Using `ggplot`: Daily coronavirus cases identified in China (1 January 2020 - 7 April 2020).

Let's overlay the daily cases for two countries. We chose China and South Korea, but you can change the code accordingly for countries you are interested in.

```
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
```

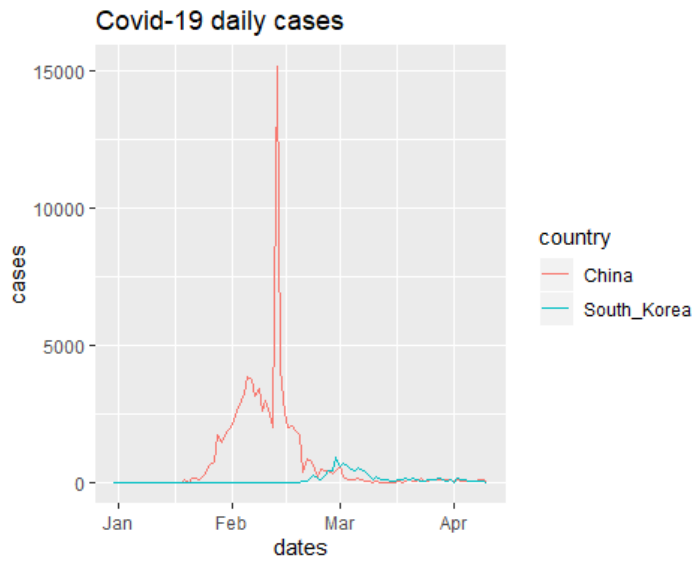

Figure 1.3 Daily coronavirus cases identified in China and South Korea (1 January 2020 - 7 April 2020).

Here you can see the much-praised ability by South Korea (https://tinyco.re/4913845) to suppress the numbers of infections effectively. However, you might argue that it is difficult to directly compare the outcomes of countries with different population sizes and different knowledge about the virus (China undertook virus containment measures at a time when very little was known about the virus).

Explore the data: Redo Figure 1.3, but for two or three different countries of your choice.

Extension: Redo Figure 1.3, but with the number of cases per capita on the vertical axis.

Now we look at some European countries (Spain, France, and the UK).

```
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
```

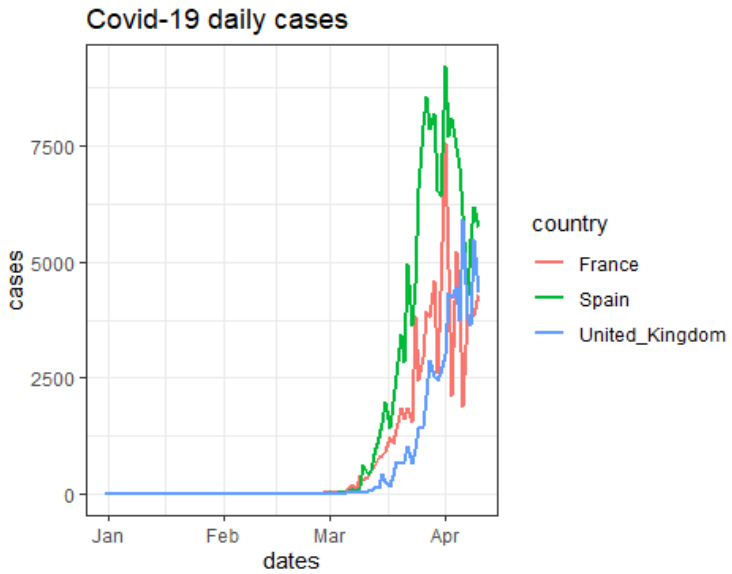

Figure 1.4 Daily coronavirus cases identified in France, Spain, and the UK (1 January 2020 - 7 April 2020).

Explore the data: Check out the `ggplot` cheatsheet (https://tinyco.re/8940854) to see some of the many ways in which you can customise graphs. In particular, see what happens if you replace the last line in the code block above (`g3`) *with* `g3 + theme_bw()` *or* `g3 + theme_dark()`.

You can see that the data are very variable, especially the time series for France. It is important to understand that the number of newly identified cases also depends on the number of tests conducted or test results received on a particular day, and if that varies from day to day, then we would expect a huge variability in reported cases. In addition to daily variability in a particular country's testing activity, different countries may have quite different strategies on testing.

Read more:

- This iNews article (https://tinyco.re/4319550), written on 1 April 2020, compares testing strategies adopted by various countries.
- Nate Silver's article (https://tinyco.re/6395215), written on 4 April 2020, explains how differences in testing strategies makes comparisons across countries difficult.

In fact, the amount of daily variation makes it difficult to see longer-term trends. This issue is common with high frequency data, and we often address it by "smoothing" the data. `ggplot` has a built-in function called `geom_smooth` which allows you to see smoothed versions of the data. We will not discuss the details of the exact method used, but it essentially averages data over some local (time) window.

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

```
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
```

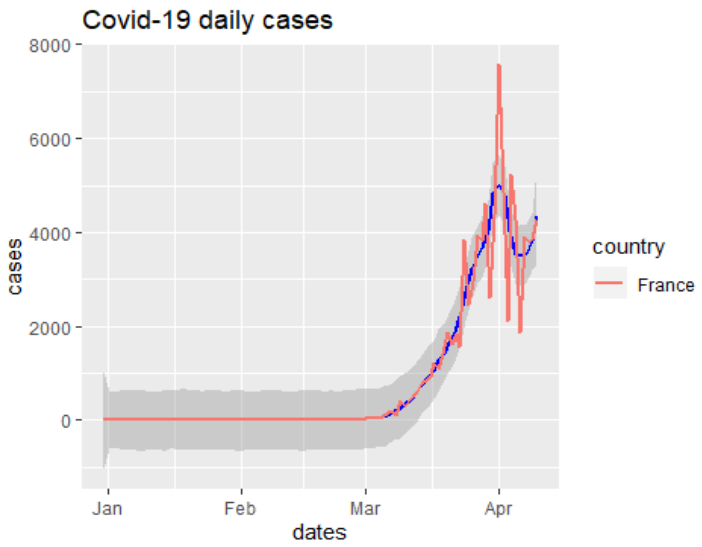

Figure 1.5 Coronavirus cases identified in France: Actual and smoothed time series (1 January 2020 - 7 April 2020).

You can see that the smoothed version (in blue) clearly shows an early exponential trend but some evidence of a recent decline (as of 7 April 2020).

> *Explore the data:* Change the values of the `span` option in the code block above, and see how the resulting chart differs. Can you figure out how the `span` option works?

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
```

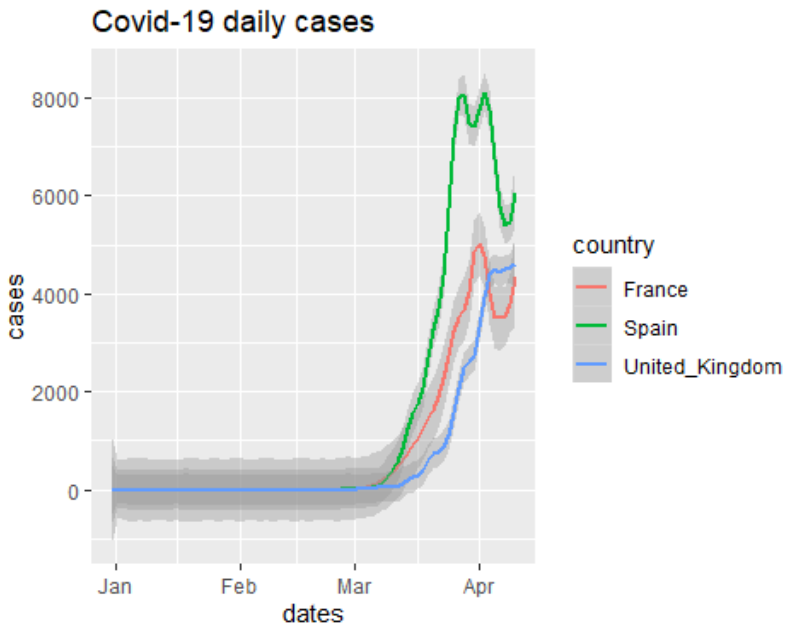

Figure 1.6 Coronavirus cases identified in France, Spain, and the UK: Actual and smoothed time series (1 January 2020 - 7 April 2020).

The grey areas around the smoothed data versions are confidence intervals, in recognition of the fact that the data are "noisy" so there will be some uncertainty about the underlying trend. As the French data are "noisier" than those of the UK and Spain the grey confidence intervals are somewhat wider than the others, indicating larger uncertainty about the trend.

Smoothing can be useful. Here we can perhaps infer that France and Spain may have reached the peak of new registered cases, whereas at the time of writing (7 April 2020), the number of UK cases was still increasing.

In the example above, the smoothing was done by the `ggplot` function, but 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 to calculate a moving average.

First, we create a new variable using the `ma` *function* (`cases_ma = ma(cases,order = 7)`), which calculates a 7-day moving average. To illustrate, suppose the date is March 10. Instead of reporting the number of cases for March 10, we take the average over 7 observations (days) – March 7 to March 13 (i.e. centered around March 10) - and recording this value March 10 in `cases_ma`.

```
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 time 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
```

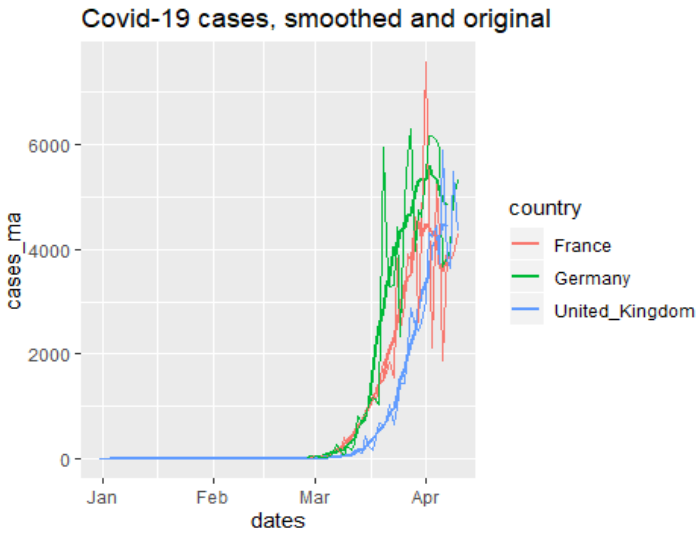

Figure 1.7 Coronavirus cases identified in France, Spain, and the UK: Actual time series and moving averages (1 January 2020 - 7 April 2020).

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
```

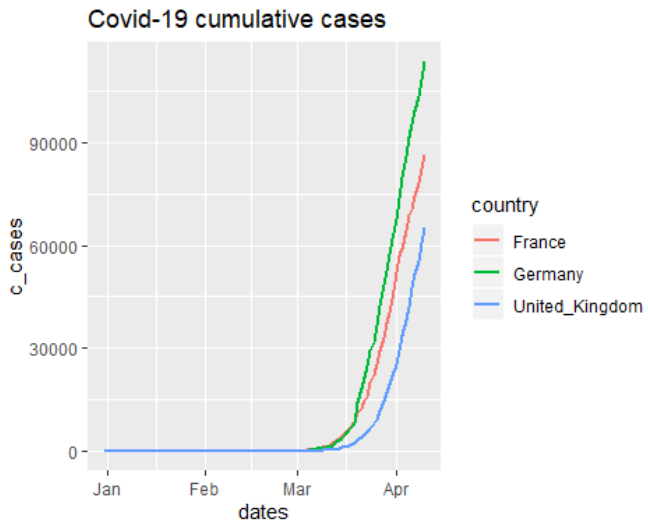

Figure 1.8 Cumulative coronavirus cases identified in France, Spain, and the UK (1 January 2020 - 7 April 2020).

You can clearly see the exponential growth in the number of cases. Sometimes you will see these graphs plotted using a logarithmic (ratio) 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
```

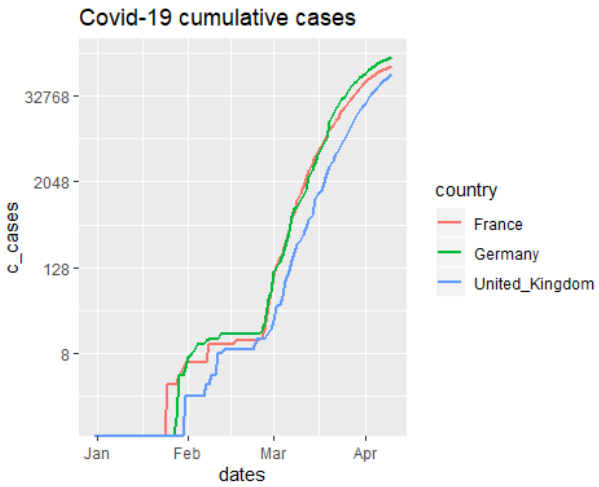

Figure 1.9 Cumulative coronavirus cases identified in France, Spain, and the UK, using a ratio scale (1 January 2020 - 7 April 2020).

In Figure 1.9, you can clearly see some early dynamics that are not visible in Figure 1.8, but using the ratio scale somewhat reduces the dramatic impression that Figure 1.8 shows.

> *Read more:* To learn more about ratio scales and how to interpret them, see Unit 1.4 of *Economy, Society, and Public Policy* (https://tinyco.re/8137587).

1.5 Plotting maps of daily COVID-19 data

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

For Sections 1.5 and 1.6, you will need the following packages for drawing maps, which we will install and import now:

```
install.packages(c("sf", "raster", "spData", "tmap"))
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")
```

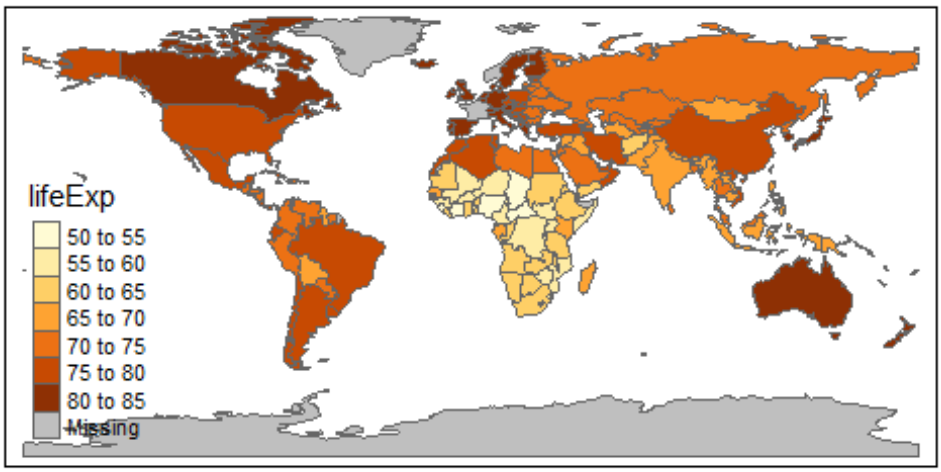

Figure 1.10 Life expectancy around the world (2018).

Wow, one line of code and you get a world map which shows which countries have the highest and lowest life expectancy. Amazing!

What did the code do? We used the `tmap` package, which has a range of built-in map information, and `tm_shape(world)`, which contains shape information (details of the boundaries) of the world's countries. Shape information is essential for drawing maps. Then we specify the variable that determines the colors and borders (`+ tm_polygons(col = "lifeExp")`).

> *Read more:* To learn more about geocomputing and the `tmap` package, check out Geocomputation with R (https://tinyco.re/1848888) written by Robin Lovelace, Jakub Nowosad, and Jannes Muenchow.

We want to make a similar map as in Figure 1.10, but showing information on COVID-19 cases instead. We will first deconstruct the code above to understand where `tmap` stores the data on life expectancy.

```
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. o
f  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" "Indete
rminate" "Sovereign country" ...
##   .. ..$ 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" "regi
on_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"
```

The output above looks complicated. `m2` is a list with one element called `sm_shape`, which in turn is a list with 12 elements. Importantly one of these elements, called `shp`, contains information on the respective countries.

Let's look at the element `shp` to understand what it looks like. We will save it as the object `temp`.

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

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

`shp` is a "standard" dataframe with country-specific information, and you can see that one of the variables is life expectancy (`lifeExp`). This is where `tmap` got the info from. We will insert the information on cases into this dataframe and then use that to display the data. `iso_a2` is a variable with country abbreviations. As we have this information also in our dataset (`data`) we will use country abbreviations to merge the data.

We start by extracting the information we want to merge into `temp` from our original dataset (`data`). As an example, we will use data for all countries on 4 April 2020.

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

When you run the code above, 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"'
```

A Google search reveals that this issue arose because the `select` function appears in two different packages we loaded (`raster` and `tidyverse`) - type `?select` into the command window to see this problem. In these cases, R chooses the function from the package loaded last (`raster` in this case), whereas we wanted the function from the `dplyr` package (automatically loaded with the `tidyverse`).

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 there isn't an external institution that ensures people do not use the same name for different functions. In fact, look at the notices in your R console that you ignored after loading the `raster` package. Most likely you will find a message similar to: **"Attaching package: 'raster'. The following object is masked from 'package:dplyr': select"**. This problem could have been avoided by loading the `tidyverse` package after the `raster` package (this is 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 from the `dplyr` package (`dplyr::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 variables we are interested in, and the `geoId` variable which we will match with `iso_a2` in the shape file.

As it turns out, the country code for the UK in `data` (and hence in `temp_mergein`) is `UK` and in the shape file it is `GB`. We need to change one of them to enable data for the UK to be matched correctly, otherwise the map would show missing data for the UK.

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

Now we will use the `merge` function to add this info into `m2` so our COVID-19 data is 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 information, even if it is unrelated to COVID-19 (`all.x = TRUE`).

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

Now that we manipulated the datafile required to make maps, 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 plot a map 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 categor
ies
```

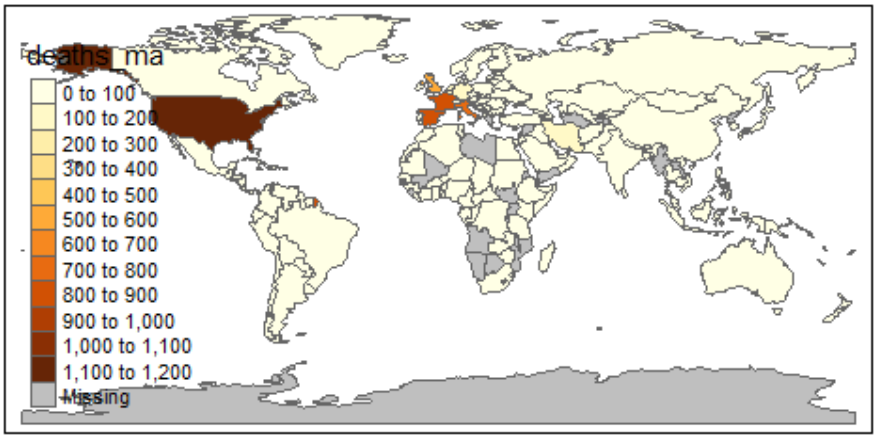

Figure 1.11 Worldwide deaths from COVID-19 (4 April 2020).

There are many ways you can customize your maps. For instance, the `tm_style` function allows you to change the colour scheme.

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

Figure 1.12 Worldwide deaths from COVID-19, alternative colour scheme (4 April 2020).

1.6 Plotting maps showing the spread of the pandemic

The maps in Section 1.5 were created for one particular day (4 April 2020), but we can create multiple maps for each day in 2020 so that we can visualise the spread of the pandemic. 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 our original dataset (`data`).

```
# prepare the data from original dataset
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 the data as in Section 1.5, but for more dates. We are now also keeping the **dates** variable. Now we need to merge the new data into **temp3**. To do so, we will turn the dates into the character format (**tmap** does not work well with date formats) and remove the countries with no available observations, as these would introduce an extra date (recorded as **na**).

```
temp3 <- merge(temp3, temp_mergein, by.x = "iso_a2", by.y = "geoId", all.x = TRUE)
temp3$dates <- as.character(temp3$dates)  # convert dates to character variables (
date formats do not work well with tmaps)
temp3 <- temp3 %>%  filter(!is.na(dates)) # delete countries with no data
```

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 make 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 that determines when a separate map should be plotted (one map per day).

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

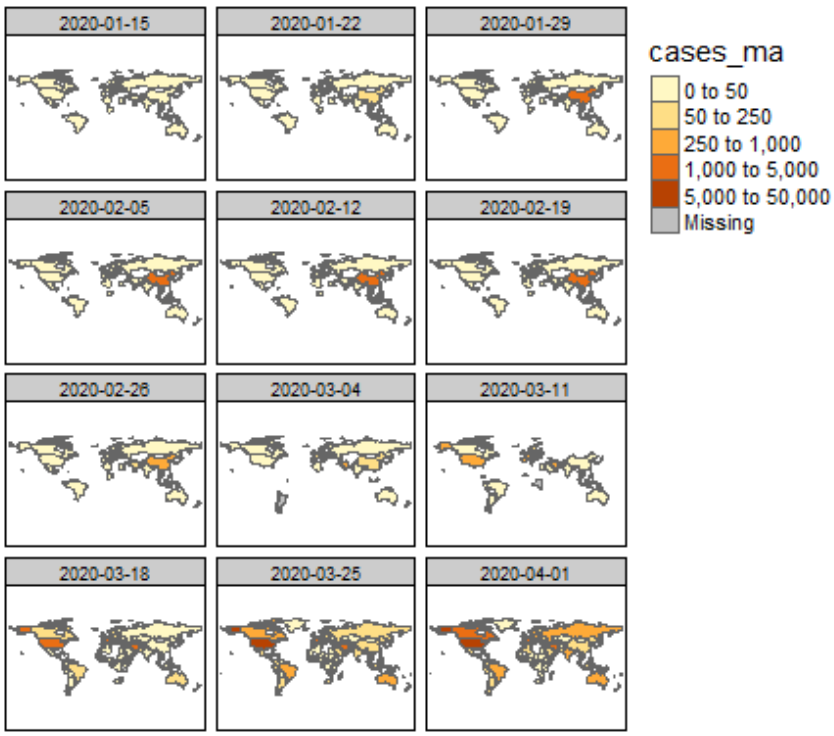

Figure 1.13 Worldwide deaths from COVID-19 (15 January 2020 – 1 April 2020).

The output 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 that particular day, the country will not appear on the map.

Explore the data: Use `?tm_polygons` to learn about some of the adjustments you can make to these graphs. For example, experiment with the colour scale - in the code block above, replace `style = "fixed", breaks = c(0,50,250,1000,5000,50000)` with `style = "cont"` *or* `style = "log10_pretty"`, and see which colour scale works best in this context.

2. Linking Covid-19 data with travel route data

In this section, we will link some other data to our COVID-19 data from Section 1. We will focus on how the virus might spread from one country to another through international travel routes (flights).

Explore the data: In the Kaggle data competition website (https://tinyco.re/9244615), you will find a range of other datasets which could provide useful insights into managing 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

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

2.1 Using international travel route data to identify potential transmission routes

Let's load some international travel data. We can get some data on flight connection information from the Open Flights website (https://tinyco.re/8457387). The "routes.dat" datafile is a CSV file without column headings, which is why we need to add the following 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 the airline listed, but another carrier), empty otherwise.
- Stops: Number of stops on this flight ("0" for direct flights).
- Equipment: 3-letter codes for plane type(s) generally used on this flight, separated by spaces.

Note that this dataset is from 2014 (no later data is available).

```
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 dataset could be useful as it provides information on travel routes, which also could be virus transmission routes. We will add country-specific information for these airports, taken from the "airport.dat" file on the Open Flights website (https://tinyco.re/8457387).

```
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 first need to find the airport ID (**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** is the airport code for Manchester and **CDG** (for Charles de Gaulle) is the airport code for Paris. Let's find the route information (recall from Section 1.6 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>
```

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, or what the passenger capacity of these flights were. Also, note that the Alitalia flight (**AZ**) was codeshared with Air France (**AF**).

We will want to calculate some measure that indicates the degree of connectedness between countries, which could give a proxy for how likely it is that a virus may spread from one country to another. To avoid double-counting flights, we start by removing codeshare connections from the dataset (we are keeping those where the entry is **NA**), and save this subset of data as **data_routes**.

```
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 country variable and ***IATA.ID***, which will be matched to **Source.Airport** and **Destination.Airport** in **data_routes**. We create two files to make the merging process more straightforward. These data come from **data_airports**.

```
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 look again at our Manchester to Paris connections, we get the following:

```
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 add up 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 connections from China to other countries have been to South-East Asian countries.

We can even check the countries that the city of Wuhan is most connected to. First, getting the airport code for Wuhan:

```
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>
```

The airport code is **WUH**.

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

The results are similar to those we got for China as a whole. **However, remember that the connection data are from 2014.**

2.2 Predicting the initial spread of COVID-19

We can use the flight data from Section 2.1 to build a prediction tool that could form part of a more complex model of how the pandemic might spread. Real prediction tools involve many other elements (see the *Read more* box at the end of this section for a review of fully thought-out models), but here we illustrate how one of the elements in a fully-developed prediction tool could work.

We will use the moving average of daily cases (**cases_ma**) and combine them with the measure of strength of travel connection from Section 2.1 to assess where it is likely that the virus would spread. There are, of course, many other factors which affect the spread of a virus. Even if we cannot incorporate all of these factors into our prediction tool, it is apparent that travel connections are an important factor as many countries have now introduced severe travel restrictions. These travel restrictions also imply that using flight routes to predict the spread of the virus is likely to only 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 our original dataset from Section 1, **data**) as well as international flight connection data (in our dataset from Section 2.1, **connections**).

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

## [1] 223

countries_d <- unique(as.character(data$country)) # countries in virus dataset, co
nverting country names from factor to chr variable
length(countries_d)

## [1] 171

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

## [1] 130
```

`cou_list`, the list of countries with both virus and flight data, contains 130 countries – 41 fewer than the list of countries we have virus data for. It is possible that in some cases, countries in the two datasets have different spelling. To check whether this issue is likely, we will look at the set of 41 countries which have been excluded from the `countries_d` subset.

```
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"
## [38] "Cape_Verde"
## [39] "Isle_of_Man"
## [40] "New_Caledonia"
## [41] "Papua_New_Guinea"
```

Amongst these are a number of large and important countries which we would certainly want to take account of. For these countries we need to make the spelling in the **connections** and **data** datasets consistent, and then rerun the code that 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 flight connection
dataset
length(countries_c)

## [1] 223

countries_d <- unique(as.character(data$country)) # countries in virus dataset, co
nverting country names from factor to chr variable
length(countries_d)

## [1] 171

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

## [1] 146
```

Now we have 146 countries, 16 more than before. In **countries** we now have a list of alphabetically-ordered countries which we will use for the rest of our analysis.

Let's consider the situation for the same twelve days we used in Section 1.6 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 use the **spread** function to reformat the data so that the dates are variable names, and save this output as the dataframe called **spread**.

```
cou_list <- as.factor(cou_list)
spread <- data %>%  dplyr::filter(country %in% cou_list & dates %in% date_sel)  #
```

```
only countries in cou_list
spread <- spread %>% dplyr::select(dates,country, geoId, cases_ma) %>%
                        spread(dates,cases_ma)

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

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

To create a measure of how likely the virus is to spread into a particular country, we multiply the number of current infections with the number of connections. We use a double loop in R to do this: for every (receiving) country we will calculate the potential risk of receiving the virus (cases in the source country × number of connections). All of these risks 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 destination 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)
}
```

Note that there are other ways of calculating this measure e.g. using matrix algebra.

We will now plot the resulting information on a map, following the mapping operations used in Section 1.6.

```
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))
```

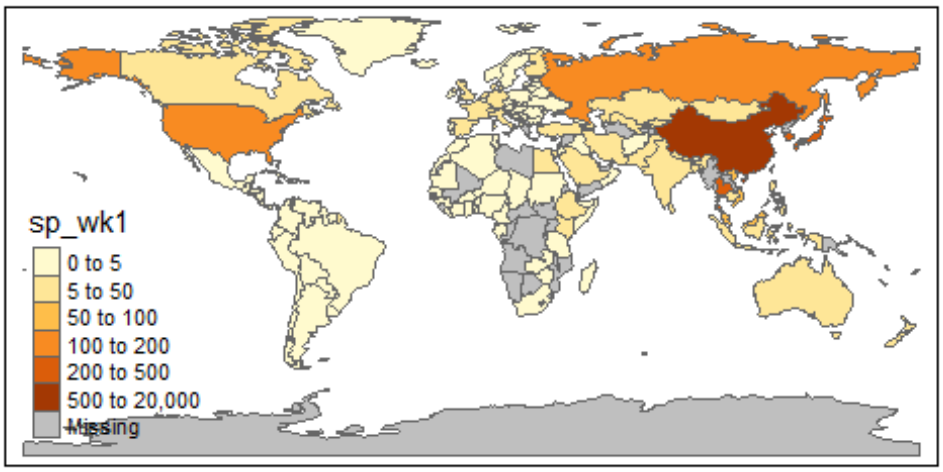

Figure 1.14 Predicted susceptibility to COVID-19 worldwide, after first week of cases in China.

In that first week the only country with cases was China. The region which is most likely to be affected next, assuming that air transport is the only transmission mechanism, 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 countries or territories that we predict would be most susceptible to receiving the virus.

```
# name_long is country/territory 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 our predictions to the actual spread of infections five weeks later (`dates == date_sel[6]`) – specifically smoothed data on 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))
```

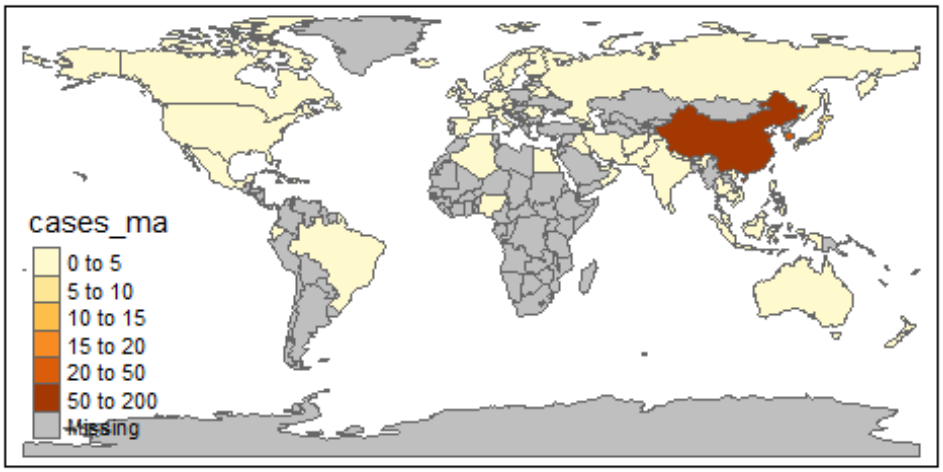

Figure 1.15 Number of COVID-19 cases worldwide (19 February 2020).

Let's look at the ten countries with the highest number of cases (smoothed) on 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,...
```

Altogether, five of the countries we predicted would be most susceptible to receiving the virus appear in the list of countries with the highest number of reported cases. Remember, that these are identified infections, not the actual numbers (which are unknown).

This analysis was, of course, very simplified, but a key insight is that using data on transport links is important for analysing how a pandemic can spread.

You can also see that models have potential to simulate the effect of travel restrictions. For example, to assess the effect of a travel ban, you could set an element in the `connections` data to 0. Of course, there are many more factors to consider when modelling virus transmission, in particular during the later spread of a pandemic, when many countries implement multiple policies (such as school closures and stay at home orders).

Read more: To learn more about how global pandemics can be modelled mathematically, check out this literature review: Walters et al., 2018, Modelling the global spread of diseases: A review of current practice and capability, Epidemics, 25 (https://tinyco.re/8847944). All models discussed in this article incorporate some information about transport links.

Find out more

There are a number of great places to find data on COVID-19:

- The Coronavirus Resource Center (https://tinyco.re/7053991), run by the John Hopkins University, publishes data as well as articles on coronavirus.
- Our World in Data (https://tinyco.re/8868909) has a dedicated COVID-19 page where they review some of the latest data. 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) (https://tinyco.re/4826169), rather than other sources. This is a particularly good case of careful data analysis.
- The data competition site Kaggle (https://tinyco.re/9244615) has a dedicated section for COVID-19-related challenges. In their data challenge entitled "Use exploratory analysis to answer research questions that support frontline responders" (https://tinyco.re/9692781), you can see the particular questions posed and how other people have used data to answer them. Some Kaggle users have contributed notebooks (https://tinyco.re/2784324), which contain their code and data analysis. Most of the entries are written in Python, but some are in R ("Rmd") format. Looking at the Python ones can still give you a sense of the kinds of data analysis that are possible.

If you are interested in using data to investigate pressing policy issues, the *Doing Economics* ebook (https://tinyco.re/1425213) contains a range of projects similar to this one, which you can complete in R, Excel, or Google Sheets.