Data Introduction 2 - Covid 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 and scaling techniques (Sections 1.3 and 1.5)
- produce line charts using ggplot (Section 1.4)
- learn how to use maps to illustrate geographic differences in the pandemic (Sections 1.7 and 1.8)
- use data designed to capture the severity of policy responses (Section 2)
- and think about the principles of evaluating the effectiveness of government policy (Sections 2.1 to 2.3)

The CORE-ECON Covid-19 Collection contains a more detailed version of this example.

1. Some exploratory data analysis

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

1.1 Getting started

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

```r
#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
library(stargazer)     # for nice regression output
```

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")))
```

```
## Response [https://opendata.ecdc.europa.eu/covid19/casedistribution/csv/]
##   Date: 2021-02-13 18:11
##   Status: 200
##   Content-Type: application/octet-stream
##   Size: 680 kB
## <ON DISK>  C:\Users\msassrb2\AppData\Local\Temp\RtmpKGCCW4\file4db47cce4a7d.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 (6 January 2021).

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':    10647 obs. of  10 variables:
##  $ dateRep                                        : Factor w/ 58 levels "01/02/2021","01/06/2020",..
##  $ year_week                                      : Factor w/ 58 levels "2020-01","2020-02",..: 58 5
##  $ cases_weekly                                   : int  238 267 713 557 675 902 1994 740 1757 1672
##  $ deaths_weekly                                  : int  8 16 43 45 71 60 88 111 71 137 ...
##  $ countriesAndTerritories                        : Factor w/ 215 levels "Afghanistan",..: 1 1 1 1 1
##  $ geoId                                          : Factor w/ 214 levels "AD","AE","AF",..: 3 3 3 3
##  $ countryterritoryCode                           : Factor w/ 214 levels "","ABW","AFG",..: 3 3 3 3
##  $ popData2019                                    : int  38041757 38041757 38041757 38041757 3804175
##  $ continentExp                                   : Factor w/ 6 levels "Africa","America",..: 3 3 3
##  $ notification_rate_per_100000_population_14.days: num  1.33 2.58 3.34 3.24 4.15 7.61 7.19 6.56 9.0
```

Some of the variables have obvious meaning, such as `day`, `month`, `year`, `countriesAndTerritories` and `popData2019`, the latter giving the population of the respective country in 2019. `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 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 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")
```

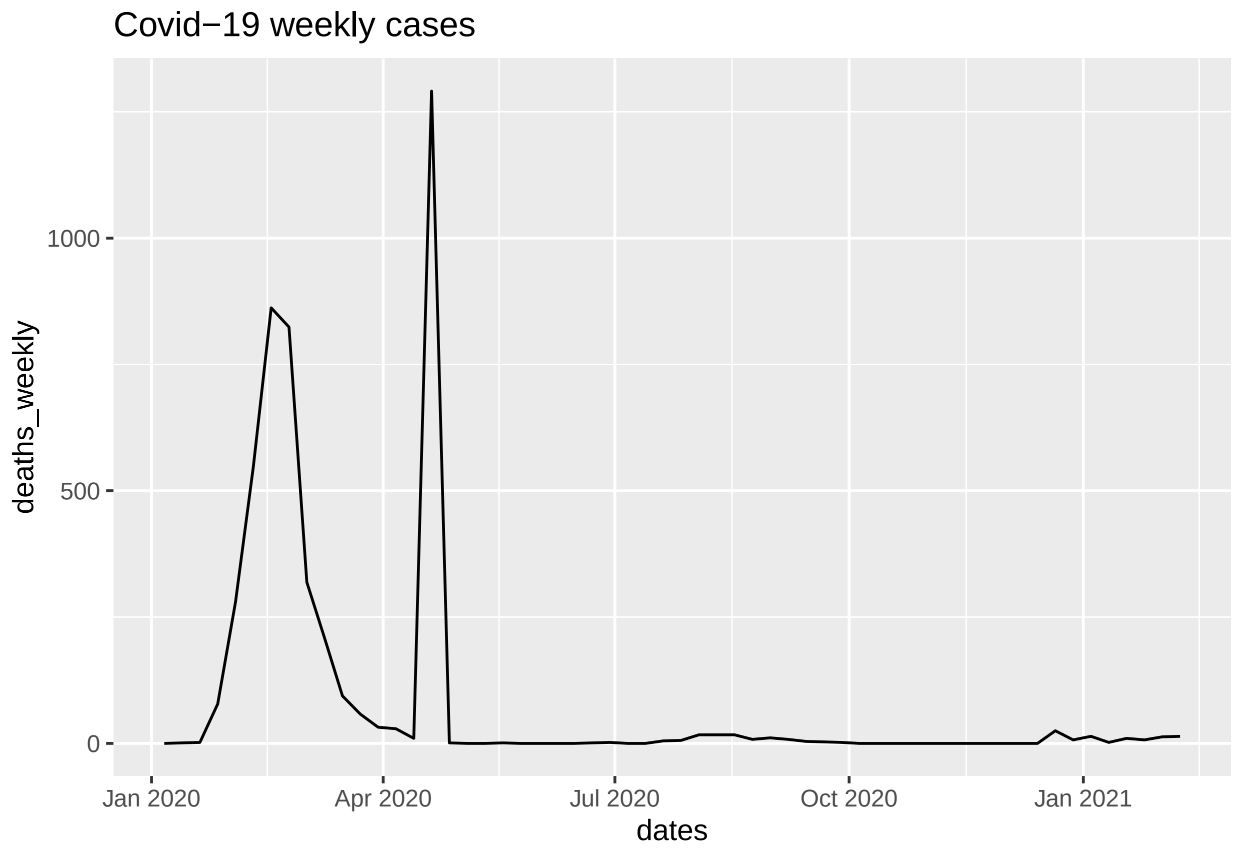
Last, but most importantly, there are two variables called `cases_weekly` and `deaths_weekly`. This inicates that the dataset is organised as a weekly dataset. Each data-entry (each row) represents the data for a particular week in a particular country.

1.4 Plotting line charts for some countries

Let's create some charts to describe the development of the pandemic in different countries. These are weekly case and death data. Let's pick one country and plot information on this country through time. We chose China, where this particular virus was first identified

We use the `ggplot` function, which produces very nice charts. We will create the chart and save it as the object `g1`, then display it by just calling `g1`. We will use the subset function to select data from China only (`subset(data, country == "China")`).

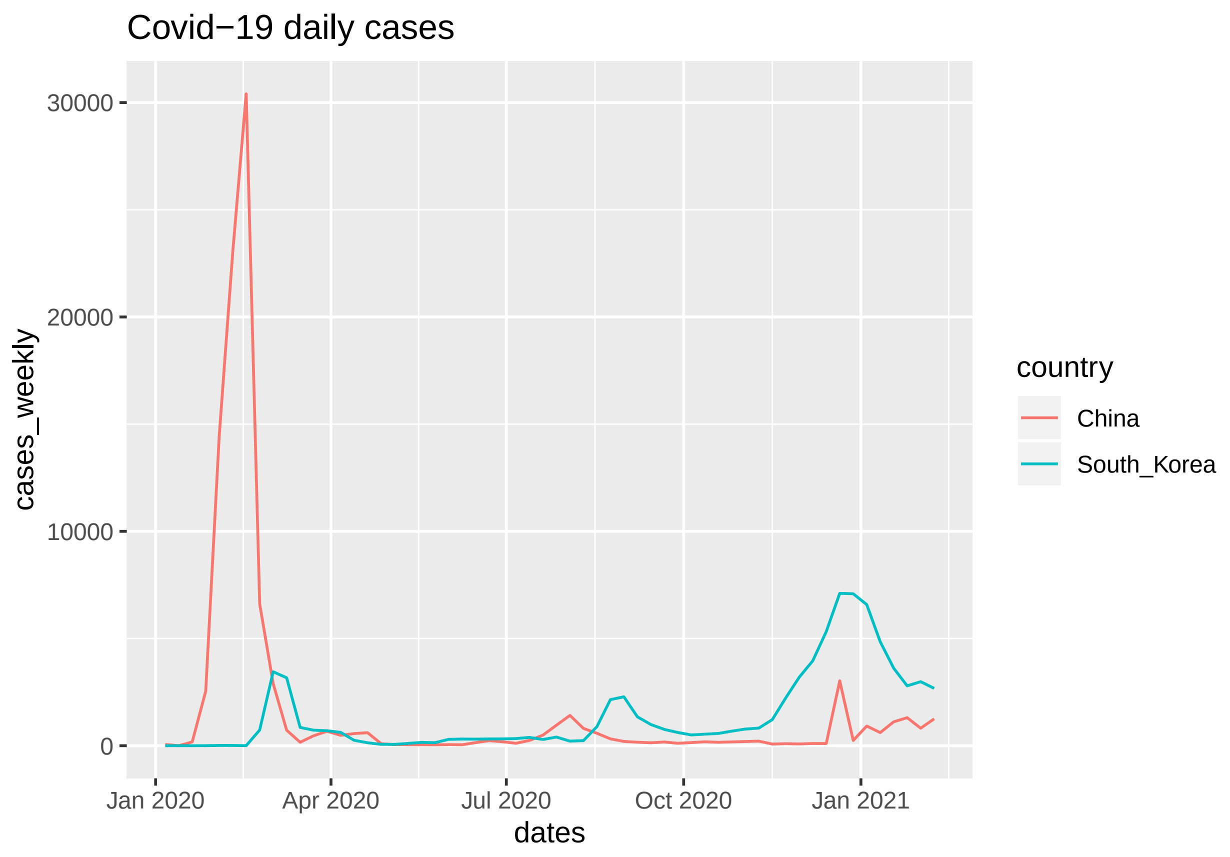
```
g1 <- ggplot(subset(data, country == "China"), aes(x=dates,y=deaths_weekly)) +
      geom_line() +
      ggtitle("Covid-19 weekly cases")
g1
```



Before continuing we may also want to highlight a particularity in these data. You can see that in the middle of April there is one week (20 April 2020) on which almost 1,300 deaths have been reported, vastly larger than the numbers in the weeks before and after. And in fact, the day before there were only 13 reported deaths. It turns out that this apparent outlier is the result of some revision of previously published statistic. In that week China declared that around 1,300 deats, which were previously not attributed to Covid, should be attributed https://www.livescience.com/wuhan-coronavirus-death-toll-revised.html.

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

```r
sel_countries <- c("China", "South_Korea")
g2 <- ggplot(subset(data, country %in% sel_countries),
             aes(x=dates,y=cases_weekly, color = country)) +
      geom_line() +
      ggtitle("Covid-19 daily cases")
g2
```

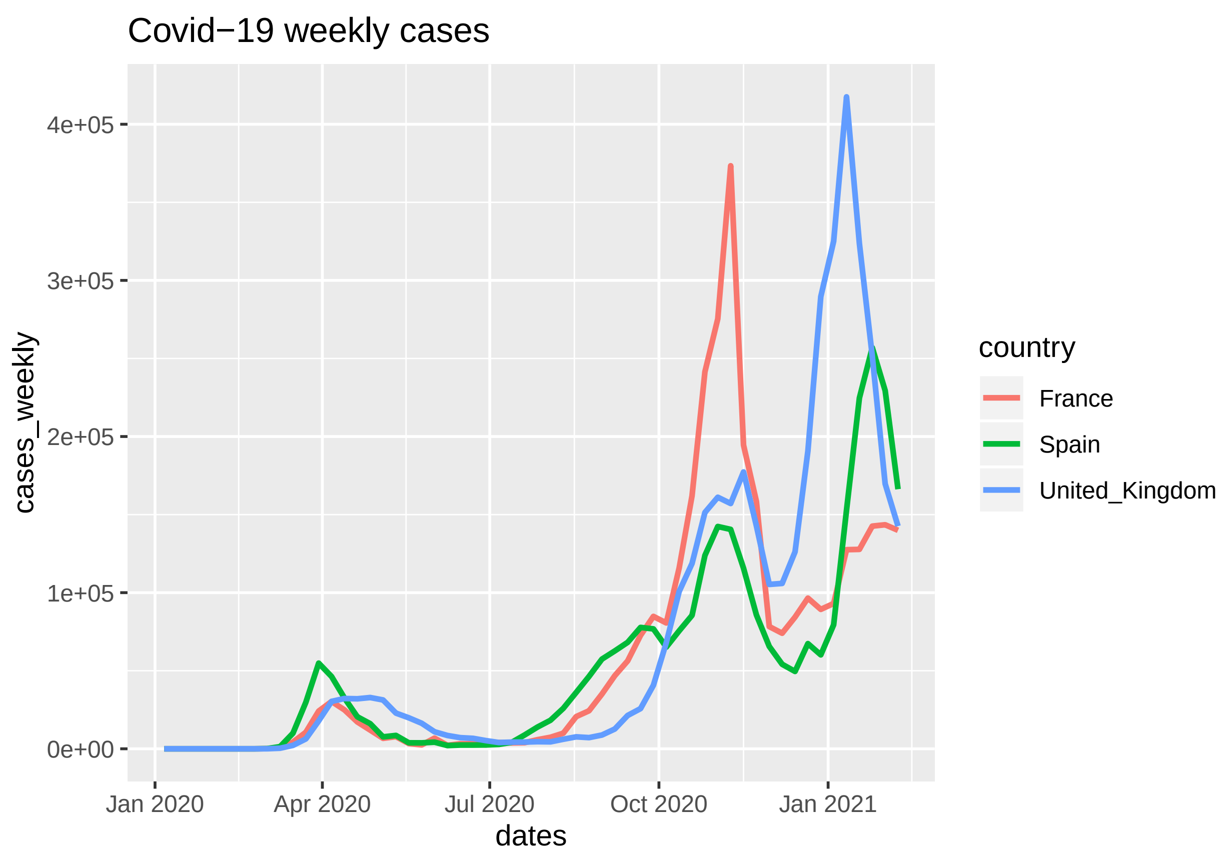


Here you can see the much-praised ability by South Korea (https://tinyco.re/4913845) to suppress the numbers of infections effectively. However, you might argue that it is difficult to directly compare the outcomes of countries with different population sizes and different knowledge about the virus (China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

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

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



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().

It is important to understand that the number of newly identified cases is also a function of the testing effort in a particular week. This is particularly important when looking at the above figure. During the first wave, testing for Covid-19 was mainly done for symptomatic patients and their contacts. In other words it was very targeted. In the second half of 2020, testing became much more widespread and the higher numbers of confirmed cases in the second wave are mainly explained by this difference in testing regime.

Read more: • This https://tinyco.re/4319550, written on 1 April 2020, compares testing strategies adopted by various countries. • https://tinyco.re/6395215, written on 4 April 2020, explains how differences in testing strategies makes comparisons across countries difficult.

1.5 Per-capita statistics, other standardisations and some graphical representations

It was mentioned earlier that comparing raw numbers of reported Covid cases between countries is not so instructive when comparing countries or regions with different population sizes. A coomonly used population adjusted measure is the number of weekly cases per 100,000 population, sometimes called the incidence rate (e.g. https://coronavirus.jhu.edu/map.html).

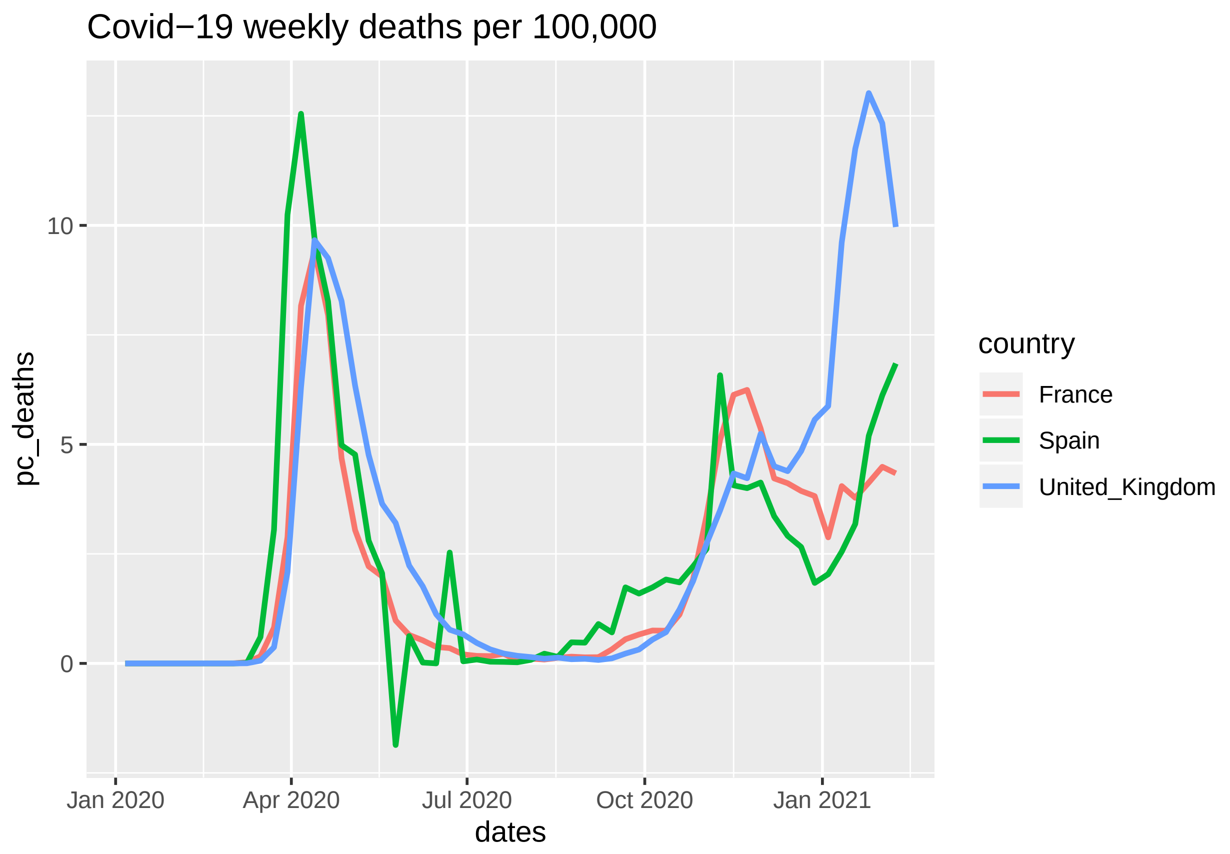
In order to adjust the weekly case numbers we use the 2019 population data (`popData2019`) which are included in the dataset. While each week has its own population entry, you should recognise that this population information does not actually change.

We will now create new variables `pc_cases` and `pc_deats` which report the number of confirmed new cases or deaths per 100,000 people for a 7 day period.

```
data <- data %>%
          mutate(pc_cases = (cases_weekly/popData2019)*100000,
                 pc_deaths = (deaths_weekly/popData2019)*100000)
```

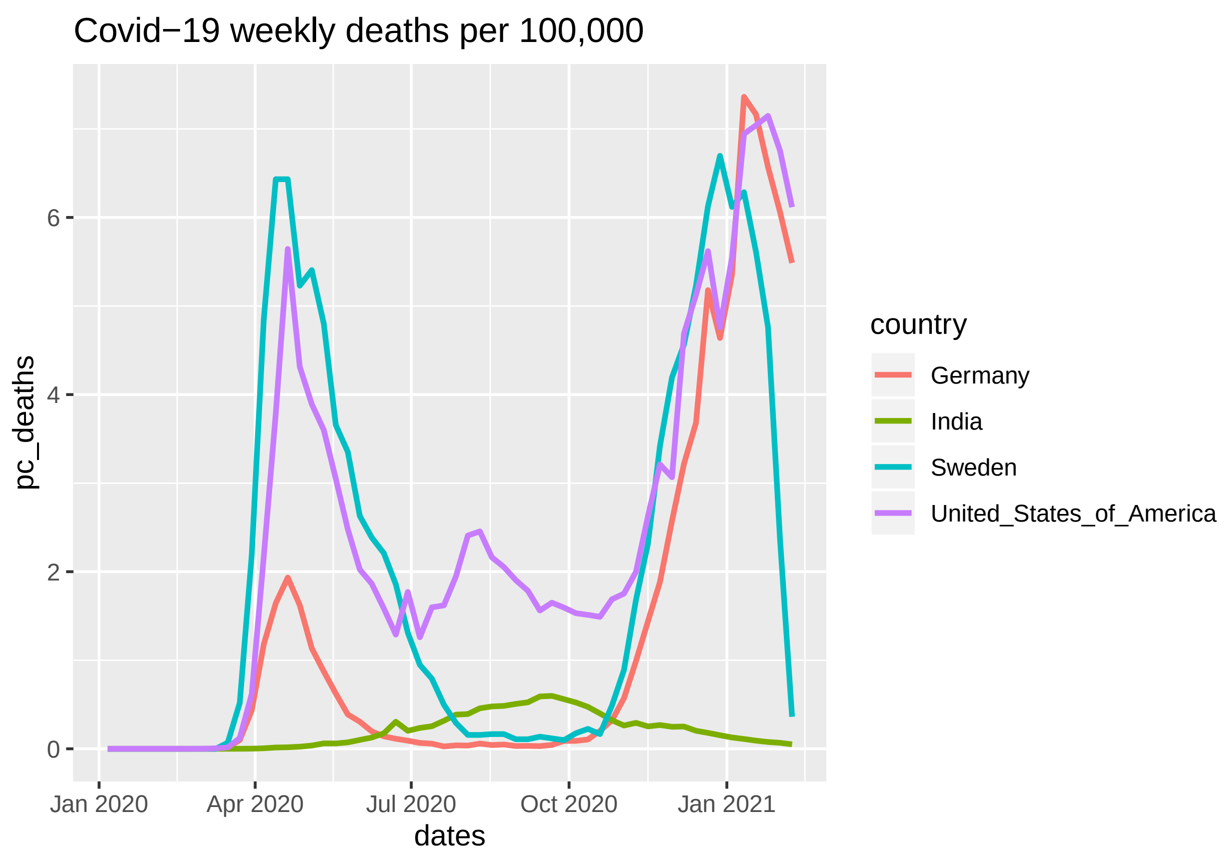
With

```
sel_countries <- c("Spain", "France", "United_Kingdom")
g5 <- ggplot(subset(data, country %in% sel_countries),
             aes(x=dates,y=pc_deaths, color = country)) +
      geom_line(size = 1) +   # size controls the line thickness
      ggtitle("Covid-19 weekly deaths per 100,000")
g5
```



This makes it obvious that Spain was hit harder by the first wave in Spring 2020 and that the UK had a more difficult Winter 20/21. More importantly, these population-adjusted data allows us to compare countries of very different size. For instance we can look at the India, US, Sweden and Germany.

```
sel_countries <- c("United_States_of_America", "Germany", "Sweden", "India")
g5 <- ggplot(subset(data, country %in% sel_countries),
             aes(x=dates,y=pc_deaths, color = country)) +
      geom_line(size = 1) +   # size controls the line thickness
      ggtitle("Covid-19 weekly deaths per 100,000")
g5
```



Is it the case that countries with larger population density find it more difficult to control the spread of Covid? This is a very fair question to ask. In order to answer this question we need to import data on the land area size of countries. We already have population data and then we can easily calculate a measure for the population density.

The data for the country size are available from the following csv file: `CountryIndicators.csv`. The data in this file have been downloaded from the https://www.gapminder.org/data/ (this also imports two further country indicators, Health Expenditure and GDP per capita which we will use Later, from the World Health Organisation). I made your life easy by ensuring that the country names match as much as possible. For instance in Gapminder the UK is "United Kingdom" and in the Covid data it is called "United_Kingdom". We need the names to be identical in order for the following code to correctly merge the country size into our datafile. This may require some manual checking (which I have done for you now).

Another reason why you need to check the country names is that there are countries or territories which may or may not be universally recognised as countries. Perhaps you realised that at the beginning of the code I changed the variable name from `countriesAndTerritories` to `country` just for simplicity. But as the name indicates, `data` contains territories which are not universally recognised as independent countries (e.g. Anguilla, which is a British Overseas Territory or Taiwan, which is only recognised by currently 15 countries as an independent state and otherwise is treated as a part of the People's Republic of China - see https://en.wikipedia.org/wiki/Political_status_of_Taiwan). So data can be quite political and it is for this reason that you will often see economic statistics recorded for countries and terretories. Renaming this variable here to `countries` is really only meant to make our coding life easier and not as a political statement.

The file "LandArea.csv" also imports the variable 'ContinentCode" which indicates which continent a country belongs to .

After downlaoding the list of countries and their respective sizes (in column `Land_Area_sqkm`) we can merge this information into `data`. Both `data` and `landarea` contain a column `country` and that is what R will use to match information. If you want to merge on columns which do not have the same name you can do this by using the `by.x` and `by.y` options in the `merge` function. Use `?merge` to look at the help pages.

```
countryInd <- read_csv("CountryIndicators.csv",na = "#N/A")
data<- merge(data,countryInd,,all.x=TRUE)
```

We can now calculate the poppulation density

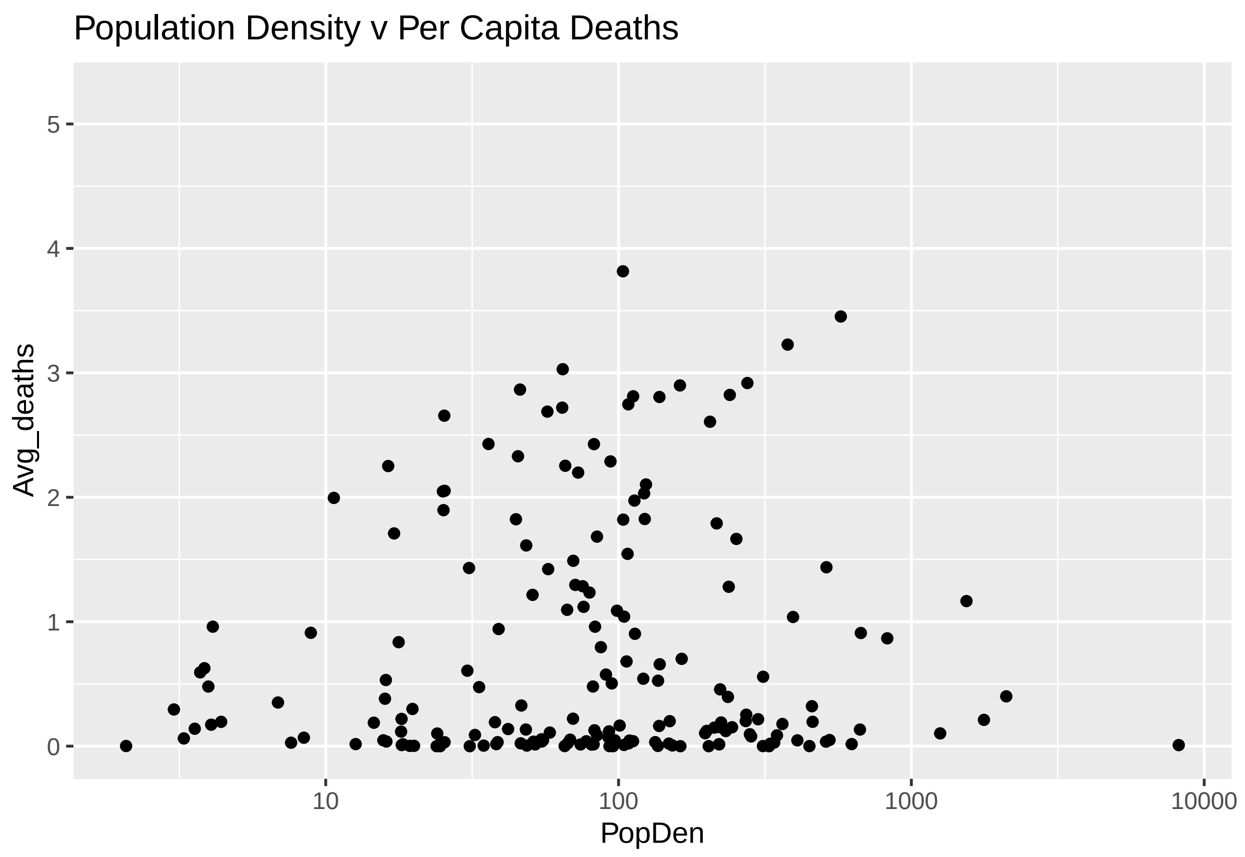
```
data <- data %>% mutate(popdens = popData2019/Land_Area_sqkm)  # calculate population density
```

We can now calculate a new summary table for per-capita cases and include the population density.

```
table3 <- data %>% group_by(country) %>% # groups by Country
            summarise(Avg_cases = mean(pc_cases,na.rm = TRUE),
                      Avg_deaths = mean(pc_deaths,na.rm = TRUE),
                      PopDen = mean(popdens))
```

This table has one row per country and three additional variables, `Avg_cases`, `Avg_deaths` and `PopDen`. We can now use these data to create a scatterplot using density average

```
ggplot(table3,aes(PopDen,Avg_deaths)) +
  geom_point() +
  scale_x_log10() +
  ggtitle("Population Density v Per Capita Deaths")
```



This uses the typical `ggplot` architecture. If you want to figure out what `scale_x_log10()` does, look at the Figure if you take this out.

Looking at the plot it is not immediatly obvious whether there is a clear relationship between these variables.

Just looking at the overall population density may be sliightly too simplistic as it matters how bunched up in a country a population lives.

1.6 Correlations

An important statistic which is used to measure the strength of a relationship is the correlation coefficient.

Is there a relationship between `PopDen` and `Avg_deaths`?

$$Corr_{PopDen,Avg\_deaths} = \frac{Cov(PopDen, Avg\_deaths)}{s_{PopDen}\ s_{Avg\_deaths}}$$

Correlations are in the $[-1, 1]$ interval. They are standardised covariances. Ensure you revise how to calculate sample s.d. and covariances by hand! R does it using the `cor` function.

```
cor(table3$PopDen, table3$Avg_deaths,use = "complete.obs")
```

```
## [1] -0.06936992
```

So if at all, there is a negative relationship but close to 0.

1.7 Plotting maps of COVID-19 data

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

For Sections 1.6 and 1.7, 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")
```

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 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 https://tinyco.re/1848888, written by Robin Lovelace, Jakub Nowosad, and Jannes Muenchow.

We want to make a similar map as in Figure 1.X, 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"
```

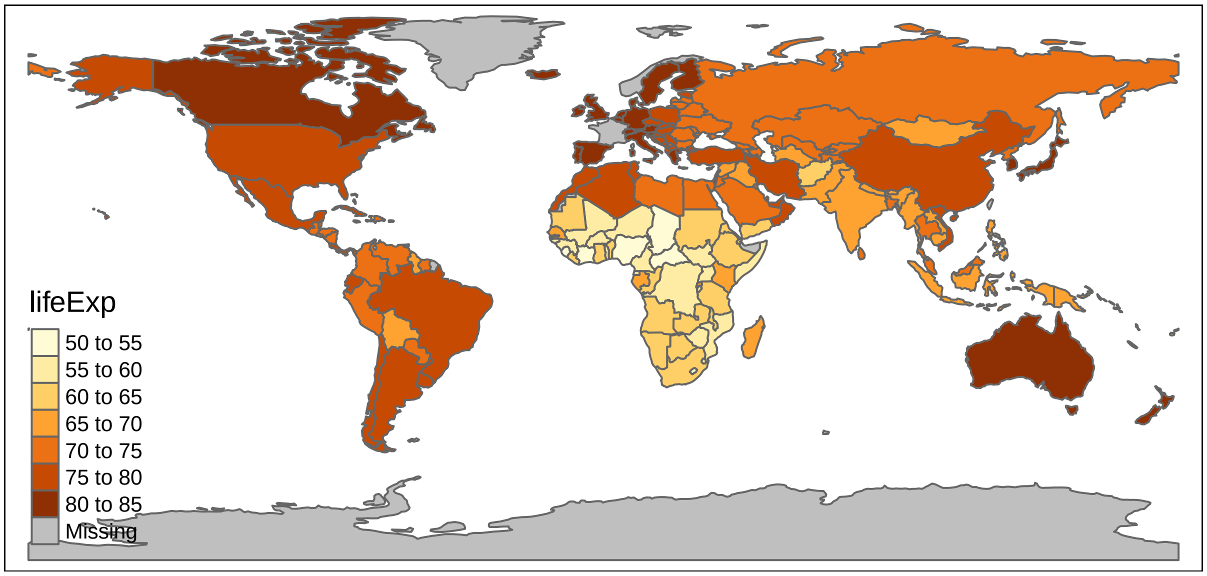


Figure 1: Figure 1.X: World Map indicating Life expectancy

```
##   ..$ 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 co
##   .. ..$ 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"
```

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`.

```r
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 14 December 2020 ("2020-12-14").

```r
temp_mergein <- data %>% filter(dates == "2020-12-14") %>%
                         select(geoId, cases_weekly,
                                pc_cases, deaths_weekly, pc_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"’
```

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 == "2020-12-14") %>%
                          dplyr::select(geoId, cases_weekly,
                                        pc_cases, deaths_weekly, pc_deaths)
```

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)   # turn geoId into character
temp_mergein$geoId[temp_mergein$geoId == "UK"] <- "GB"

# Alternativly you could leave geoId a factor var and change UK level name to GB
#levels(temp_mergein$geoId)[levels(temp_mergein$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 at the core of the mapping operation we need to insert it back into the `m2` file 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 and plot the weekly, per 100,000 population, number of casualties.

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

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 = "pc_deaths", n=10) +  # n = 10 controls the number of categories
      tm_style("col_blind")
```

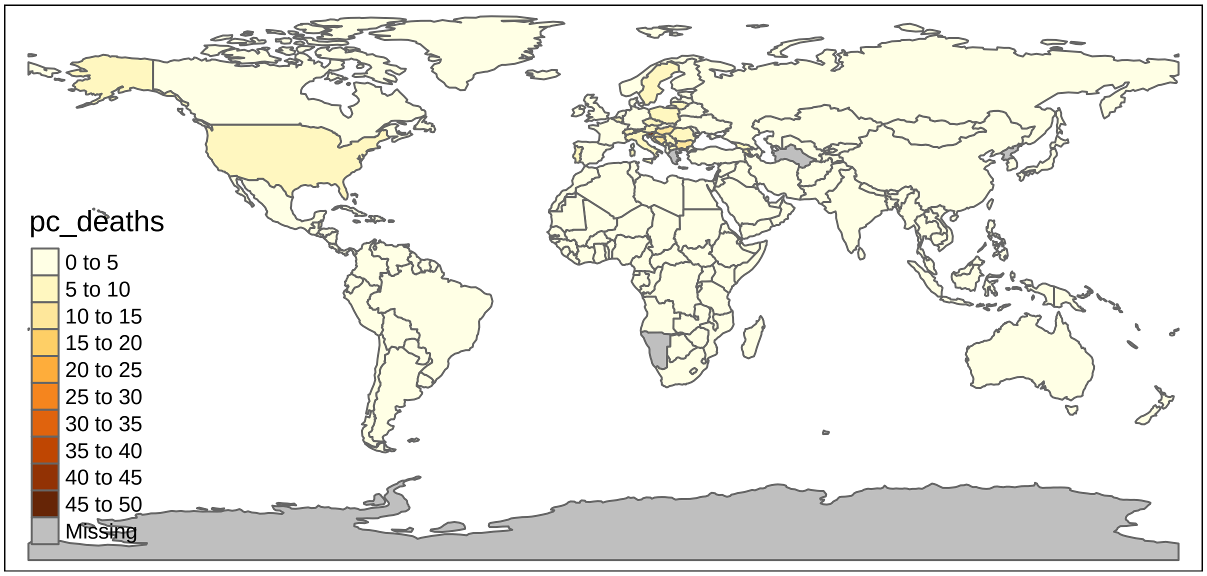


Figure 2: Figure 1.X: Weekly number of deaths per 100,000

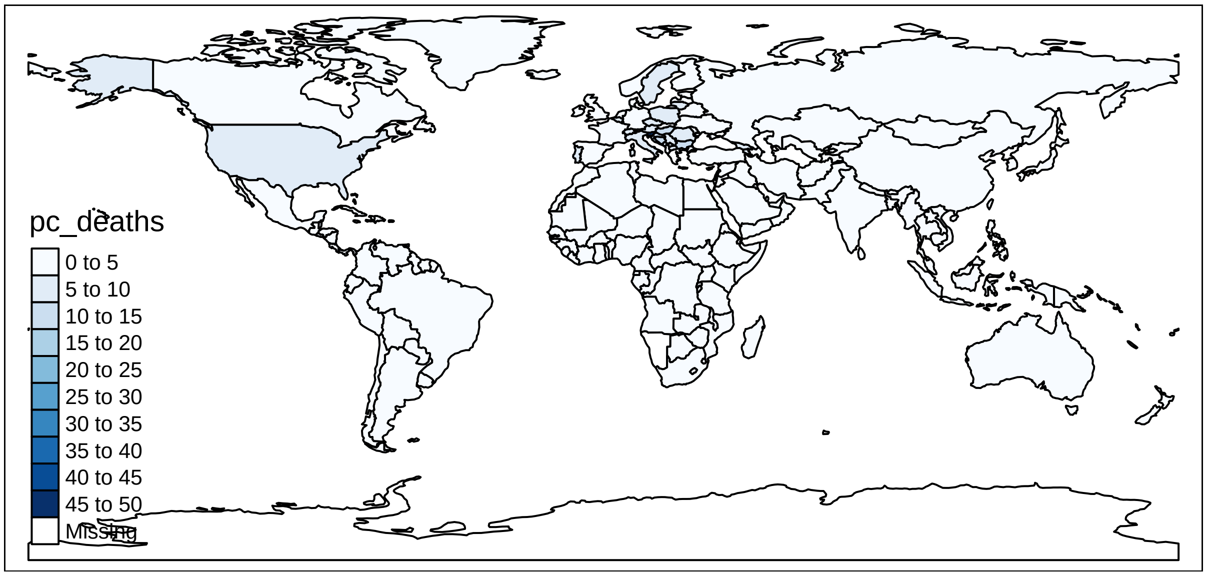


Figure 3: Figure 1.X: Weekly number of deaths per 100,000

Or you can change the categories, add a background and change the legend title.

```r
breaksm <- c(1,2,3,4,5,10,50)  # controls the breakpoints for categories
m2 +
  tm_polygons(col = "pc_deaths", breaks=breaksm, title = "Weekly deaths\n Jan 2021") +
  tm_layout(bg.color = "lightblue")
```

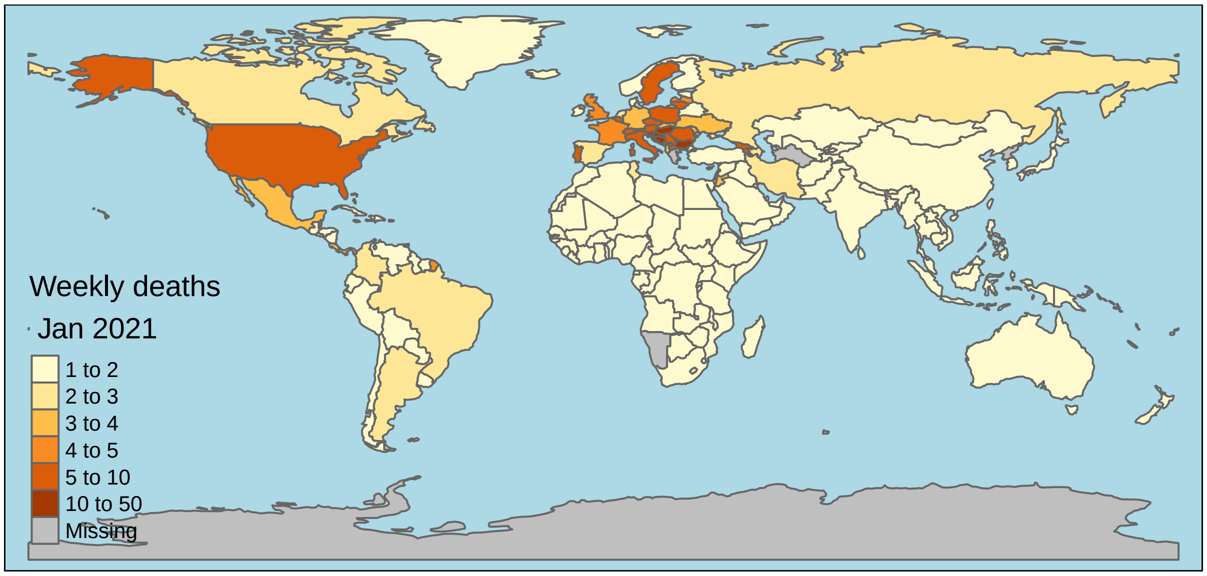


Figure 4: Figure 1.X: Weekly number of deaths per 100,000

2 Hypothesis Testing

In order to practice some hypothesis testing we will look at the data as aggregated across continents (`continentExp`). Here we select all observations from the first week of February 2021 and then average data across all countries in a particular continent.

```r
seldate <- "2021-02-01"   # Set the date you wnat to look at
table4 <- data %>%   filter(dates == seldate) %>%
              group_by(continentExp) %>%
              summarise(Avg_cases = mean(pc_cases, na.rm = TRUE),
                        Avg_deaths = mean(pc_deaths, na.rm = TRUE),
                        n = n()) %>% print()
```

```
## # A tibble: 5 x 4
##   continentExp Avg_cases Avg_deaths     n
##   <fct>            <dbl>      <dbl> <int>
## 1 Africa             23.5      0.825    55
## 2 America            89.2      1.41     49
## 3 Asia               53.1      0.566    42
## 4 Europe            180.       4.64     55
## 5 Oceania            16.0      0.149    13
```

The question we will be asking is whether the differences between continents are statistically significant.

We start by just picking out the

```r
test_data_EU <- data %>%
  filter(continentExp == "Europe") %>%     # pick European data
  filter(dates == seldate)    # pick the date
mean_EU <- mean(test_data_EU$pc_cases,rm.na = TRUE)

test_data_AM <- data %>%
  filter(continentExp == "America") %>%     # pick European data
  filter(dates == seldate)    # pick the date
mean_AM <- mean(test_data_AM$pc_cases,rm.na = TRUE)

sample_diff <- mean_EU - mean_AM
paste("mean_EU =", round(mean_EU,1),", mean_A =", round(mean_AM,1))
```

```
## [1] "mean_EU = 180.1 , mean_A = 89.2"
```

```r
paste("sample_diff =", round(sample_diff,1))
```

```
## [1] "sample_diff = 90.9"
```

We want to know whether this difference is statistically and/or economically significant?

We apply a two-sample t-test (comparing the means in two samples) using the `t.test` function.

```r
t.test(test_data_EU$pc_cases,test_data_AM$pc_cases, mu=0)  # testing that mu = 0
```

```
##
##  Welch Two Sample t-test
##
## data:  test_data_EU$pc_cases and test_data_AM$pc_cases
## t = 3.3495, df = 92.555, p-value = 0.001173
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   37.00215 144.78208
## sample estimates:
## mean of x mean of y
## 180.09998  89.20786
```

The p-value is very small and hence it is very unlikely that this difference would have arisen by chance if the null hypothesis WAS correct.

We can try the same for comparing Africa and Asia.

```r
test_data_AF <- data %>%
  filter(continentExp == "Africa") %>%     # pick European data
  filter(dates == "2021-02-01")    # pick the date

test_data_AS <- data %>%
  filter(continentExp == "Asia") %>%     # pick European data
  filter(dates == "2021-02-01")    # pick the date
```

```
t.test(test_data_AF$pc_cases,test_data_AS$pc_cases, mu=0)   # testing that mu = 0
```

```
##
##  Welch Two Sample t-test
##
## data:  test_data_AF$pc_cases and test_data_AS$pc_cases
## t = -1.7137, df = 52.492, p-value = 0.09249
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -64.25237   5.05260
## sample estimates:
## mean of x mean of y
##  23.48184  53.08173
```

Here the difference is less obvious. The p-value is just below 10%. So there is actually a substantial (i.e. around 10%) probability that the difference we see (the difference in means is around 30), could be the result of random variation.

3 Inference in Regression Analysis

Previously we learned how to run a regression in R. Running a regression is really the easiest part of the whole business. Much more difficult is to perform inference and to interpret the results correctly. Much of the unit is about the latter and that is teh most difficult aspect. Here we will touch on the task of performing statistical inference in a regression model.

Let's create new dataset which contains for every country:

- the average `pc_deaths`,
- the continent (`continentExp`),
- the population density data (`PopDen`).
- the GDP per capita (`GDPpc`,2018, in US$), from the https://apps.who.int/nha/database
- Current Health Expenditure (`HealthExp`) as % GDP, 2018, from the https://apps.who.int/nha/database

`table3` already contains `pc_deaths` and `PopDen` and has one line for each country.

```
names(table3)
```

```
## [1] "country"    "Avg_cases"  "Avg_deaths" "PopDen"
```

We need to merge in the other info from `data`. We first select the variables from the `data` datframe. After just selecting these variables and the country name (`dplyr::select(country ...)`) we still have weekly observations for each country. However, none of the selected info actually changes through the weeks. The call of `unique()` ensures that per country we just keep one row/observation.

For some countries/territories we do not have any health data and by applying `drop_na` we remove all such incomplete observations.

```
mergecont <- data %>% dplyr::select(country,continentExp, GDPpc, HealthExp) %>%
                     unique() %>% # this reduces each country to one line
                     drop_na  # this drops all countries which have incomplete information
table3 <- merge(table3,mergecont) # merges in continent information
table3 <- table3 %>% mutate(GDPpc = GDPpc/1000) # convert pc GDP into units of $1,000
```

We are now left with 183 countries/territories. Recall, one observation here is one country.

Now we run a regresison of the average `pc_deaths` (`Avg_deaths` in `table3`) against a constant only.

$Avg\_deaths_i = \alpha + u_i$

```r
mod1 <- lm(Avg_deaths~1,data=table3)
```

The new object `mod1` ontains a whole host of regression output. You can see what elements it has by checking `names(mod1)`. The nicest way to display regression results is by using the `stargazer` function.

```
##
## ===============================================
##                         Dependent variable:
##                     ---------------------------
##                             Avg_deaths
## -----------------------------------------------
## Constant                     0.747***
##                               (0.071)
##
## -----------------------------------------------
## Observations                    176
## R2                             0.000
## Adjusted R2                    0.000
## Residual Std. Error      0.940 (df = 175)
## ===============================================
## Note:               *p<0.1; **p<0.05; ***p<0.01
```

Check out this page on the ECLR webpage to learn more details about how to access particular elements of a regression (like residuals, fitted values or coefficients).

When you run a regression with only a constant then the sample estimate of the coefficient α is the sample mean of the dependent variable, here `Avg_deaths`. So on average, countries had an average rate of deaths of just under 1 per 100,000 per week due to Covid-19. (Note that in this average all countries have the same weight).

The information provided by the regression output allows you to perform a hypothesis test on the population mean.

$t - test = \widehat{\alpha}/se_{\widehat{\alpha}}$

which can be used to test the null hypothesis $H_0 : \alpha = 0$ against $H_A : \alpha \neq 0$. If you want to test a hypothesis against another null hypothesis, for example $H_0 : \alpha = 1$ against $H_A : \alpha \neq 1$ then you would use the following t-test statistic

$t - test = (\widehat{\alpha} - 1)/se_{\widehat{\alpha}}$

On most occasions you would want too estimate a regression which actually includes explanatory variables (one or more). Let's illustrate this for the following regression relationship, which includes the GDP per capita ($GDPpc$) as an explanatory variable.

$Avg\_deaths_i = \alpha + \beta\ GDPpc_i + u_i$

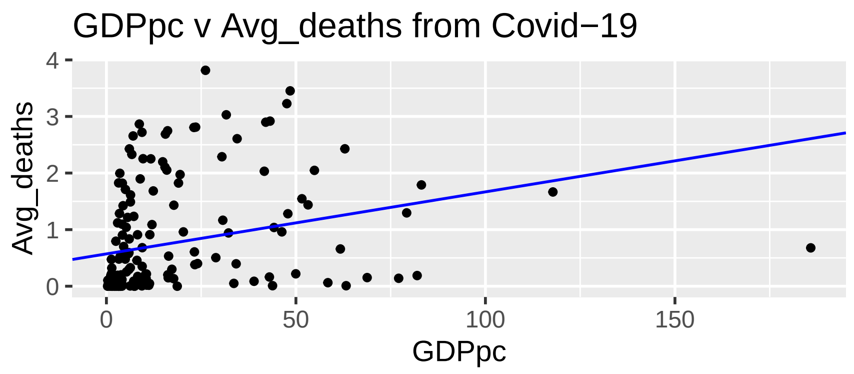
```r
mod2 <- lm(Avg_deaths~GDPpc,data=table3)
stargazer(mod2, type="text")
```

```
##
## ===============================================
##                         Dependent variable:
##                     ---------------------------
##                             Avg_deaths
## -----------------------------------------------
## GDPpc                        0.011***
##                               (0.003)
```

```
##
## Constant                          0.571***
##                                    (0.082)
##
## -----------------------------------------------
## Observations                        176
## R2                                 0.079
## Adjusted R2                        0.074
## Residual Std. Error          0.905 (df = 174)
## F Statistic               14.910*** (df = 1; 174)
## ===============================================
## Note:                  *p<0.1; **p<0.05; ***p<0.01
```

For a simple regression (only one explanatory variable) it actually possible to represent the result of this regression graphically.

```
ggplot(table3, aes(x=GDPpc, y=Avg_deaths)) +
    labs(x = "GDPpc", y = "Avg_deaths") +
    geom_point(size = 1.0) +
    geom_abline(intercept = mod2$coefficients[1],
                slope = mod2$coefficients[2], col = "blue")+
    ggtitle("GDPpc v Avg_deaths from Covid-19")
```



Here you can see a scatter plot of the data in a `GDPpc-Avg_deaths` diagram. Superimposed on that scatter plot is the line of best fit or the estimated regression line. You can see in the code above that we use the estimated coefficients `intercept = mod2$coefficients[1], slope = mod2$coefficients[2],` to create this line.

Additional explanatory variables can easily be includes as demonstrated in the next line of code.

```
mod3 <- lm(Avg_deaths~GDPpc+HealthExp,data=table3)
stargazer(mod3,type = "text")
```

```
##
## ===============================================
##                         Dependent variable:
##                     ---------------------------
##                             Avg_deaths
## -----------------------------------------------
## GDPpc                             0.008***
##                                    (0.003)
##
## HealthExp                         0.100***
##                                    (0.024)
```

```
##
## Constant                          -0.036
##                                   (0.167)
##
## -----------------------------------------------
## Observations                        176
## R2                                 0.161
## Adjusted R2                        0.151
## Residual Std. Error      0.866 (df = 173)
## F Statistic           16.565*** (df = 2; 173)
## ===============================================
## Note:               *p<0.1; **p<0.05; ***p<0.01
```

Would you have expected the coefficient to `HealthExp` to be statistically significant (possible yes!) and would you have expected it to be negative (possibly not!). May it be that countries that spend more on Health deal less well with the Covid-19 pandemic? Or could it be that these explanatory variables are not exogenous and that the estimated coefficients are biased?

It is important to understand a couple of properties of estimated regresison models. When we think about regression models we have to make assumptions about the error terms u_i. In particular we need to assume thay are uncorrelated to the explanatory variables. Note that these error terms are unobserved! What a regresison produces are estimated residuals $\hat{u}_i$, but do note that these are not the same as the unobserved error terms u_i.

In fact it is a property of an OLS regresison that the estimated error terms, the residuals $\hat{u}_i$, are in fact uncorrelated with the explanatory variables. Let is demonstrate this.

```
cor(mod3$residuals,table3$GDPpc)
```

```
## [1] -1.069519e-17
```

```
cor(mod3$residuals,table3$HealthExp)
```

```
## [1] 5.087362e-17
```

Note the meaning of of `e-17`. 3.4e-17 basically means $3.4 \cdot 10^{-17}$. So this is a very small number; basically equal to 0.

The important point here is that we may well suspect that `GDPpc` and `HealthExp` are not exogenous, i.e correlated with u. However, as we have just established is that both explanatory variables are actually uncorrelated with $\hat{u}$. This implies that we cannot use $\hat{u}$ to test properties of u.