**Introduction**

The coronavirus pandemic is a global phenomenon that will affect the lives of the majority of the world’s population for years to come.  
Impacts range from physical distancing measures [already affecting more than half of Earth’s population](https://en.wikipedia.org/wiki/Social_distancing#2019%E2%80%932020_coronavirus_pandemic) and knock-on impacts such as [changes in air quality](https://www.esa.int/Applications/Observing_the_Earth/Copernicus/Sentinel-5P/Coronavirus_lockdown_leading_to_drop_in_pollution_across_Europe) to potentially life threatening illness, with peer reviewed estimates of [infection fatality rates](https://www.thelancet.com/pdfs/journals/laninf/PIIS1473-3099(20)30243-7.pdf) showing the disease disproportionately affects the elderly and people with underlying health conditions.

Like other global phenomena such as climate change, the impacts of the pandemic vary greatly by geographic location, with effective and early implementation of physical distancing measures and effective contact tracing associated with lower death rates, according to preliminary [research](https://www.medrxiv.org/content/10.1101/2020.03.10.20033738v3), as illustrated in the animation below (source: [Washington Post](https://www.washingtonpost.com/graphics/2020/world/corona-simulator/)).

This article demonstrates how to download and map open data on the evolving coronavirus pandemic, using reproducible R code.  
The aim is not to provide scientific analysis of the data, but to demonstrate how ‘open science’ enables public access to important international datasets.  
It also provides an opportunity to demonstrate how techniques taught in [*Geocomputation with R*](https://geocompr.robinlovelace.net/) can be applied to real-world datasets.

Before undertaking geographic analysis of ‘rate’ data, such as the number Covid-19 infections per unit area, it is worth acknowledging caveats at the outset.  
Simple graphics of complex phenomena can be misleading.  
This is well-illustrated in the figure below by [Will Geary](https://github.com/willgeary), which shows how the ecological fallacy can affect interpretations of geographical analysis of areal units such countries that we will be using in this research.

The post is intended more as a taster of geographic visualisation in R than as a gateway to scientific analysis of Covid-19 data.  
See resources such as the [eRum2020 CovidR](https://2020.erum.io/covidr-contest/) contest and [lists](https://towardsdatascience.com/top-5-r-resources-on-covid-19-coronavirus-1d4c8df6d85f) of online [resources](https://github.com/HDRUK/covid-19) for pointers on how to usefully contribute to data-driven efforts to tackle the crisis.

**Set-up**

To reproduce the results presented in this article you will need to have an R installation with up-to-date versions of the following packages installed and loaded.  
(See the [Installing R on Ubuntu](https://geocompr.github.io/post/2020/installing-r-spatial-ubuntu/) article for more on setting up your computer to work with R).

library(sf)

## Linking to GEOS 3.8.0, GDAL 3.0.4, PROJ 7.0.0

library(tmap)

library(dplyr)

**Getting international Covid-19 data**

To get data on official Covid-19 statistics, we will use the [COVID19](https://github.com/covid19datahub/COVID19) R package.

This package provides daily updated data on a variety of variables related to the coronavirus pandemic at national, regional and city levels.  
Install it as follows:

install.packages("COVID19")

After the package is installed, you can get up-to-date country-level data as follows:

d = COVID19::covid19()

To minimise dependencies for reproducing the results in this article, we uploaded a copy of the data, which can be downloaded as follows (re-run the code above to get up-to-date data):

d = readr::read\_csv("<https://git.io/covid-19-2020-04-23>")

class(d)

## [1] "spec\_tbl\_df" "tbl\_df" "tbl" "data.frame"

The previous code chunk read a .csv file from online and confirmed, we have loaded a data frame (we will see how to join this with geographic data in the next section).  
We can get a sense of the contents of the data as follows:

ncol(d)

## [1] 24

nrow(d)

## [1] 17572

names(d)

## [1] "id" "date" "deaths" "confirmed"

## [5] "tests" "recovered" "hosp" "icu"

## [9] "vent" "driving" "walking" "transit"

## [13] "country" "state" "city" "lat"

## [17] "lng" "pop" "pop\_14" "pop\_15\_64"

## [21] "pop\_65" "pop\_age" "pop\_density" "pop\_death\_rate"

**Getting geographic data**

We will use a dataset representing countries worldwide from the rnaturalearth package.  
Assuming you have the package installed you can get the geographic data as follows (see the subsequent code chunk if not):

world\_rnatural = rnaturalearth::ne\_download(returnclass = "sf")

# names(world\_iso) # variables available

world\_iso = world\_rnatural %>%

select(NAME\_LONG, ISO\_A3\_EH, POP\_EST, GDP\_MD\_EST, CONTINENT)

The result of the previous code block, an object representing the world and containing only variables of interest, was uploaded to GitHub and can be loaded from a GeoJSON file as follows:

world\_iso = sf::read\_sf("<https://git.io/JfICT>")

To see what’s in the world\_iso object we can plot it, with the default setting in sf showing the geographic distribution of each variable:

plot(world\_iso)

**Transforming geographic data**

An issue with the result from a data visualisation perspective is that this unprojected visualisation distorts the world: countries such as Greenland at high latitudes appear bigger than the actually are.  
To overcome this issue we will project the object as follows (see Chapter [6](https://geocompr.robinlovelace.net/reproj-geo-data.html) of Geocomputation with R and a recent article on the [r-spatial website](https://www.r-spatial.org/r/2020/03/17/wkt.html) for more on coordinate systems):

world\_projected = world\_iso %>%

st\_transform("+proj=moll")

We can plot just the geometry of the updated object as follows, noting that the result is projected in a way that preserves the true area of countries (noting also that all projections introduce distortions):

plot(st\_geometry(world\_projected))

**Attribute joins**

As outlined in Chapter [3](https://geocompr.robinlovelace.net/attr.html#vector-attribute-joining) of *Geocomputation with R*, attribute joins can be used to add additional variables to geographic data via a ‘key variable’ shared between the geographic and non-geographic objects.  
In this case the shared variables are ISO\_A3\_EH in the geographic object and id in the Covid-19 dataset d.  
We will be concise and call the dataset resulting from this join operation w.

w = dplyr::left\_join(world\_projected, d, by = c("ISO\_A3\_EH"= "id"))

class(w)

## [1] "sf" "tbl\_df" "tbl" "data.frame"

nrow(w)

## [1] 14919

**Calculating area**

The package sf provides a wide range of functions for calculating geographic variables such as object centroid, bounding boxes, lengths and, as demonstrated below, area.  
We use this area data to calculate the population density of each country as follows:

w$Area\_km = as.numeric(st\_area(w)) / 1e6

w$`Pop/km2` = as.numeric(w$POP\_EST) / w$Area\_km

**Plotting international Covid-19 data for a single day**

The class of w shows that it has geometries for each row.  
Notice that it has many more rows of data than the original world object: geometries are repeated for every year.  
This is not an efficient way to store data, as it means lots of duplicate geometries.  
On a small dataset that doesn’t matter, but it’s something to be aware of.  
To check that the join has worked, we will take a subset of rows representing the global situation yesterday relative to the date of data access:

w\_yesterday = w %>%

filter(date == max(date, na.rm = T) - 1)

plot(w\_yesterday)

The plot method for sf objects is fast and flexible, as documented in [sf’s *Plotting Simple Features* vignette](https://r-spatial.github.io/sf/articles/sf5.html), which can be accessed with vignette("sf5") from the R console.  
We can set the breaks to better show the difference between countries with no reported deaths and countries with few reported deaths as follows:

plot(w\_yesterday["deaths"])

b = c(0, 10, 100, 1000, 10000, 100000)

plot(w\_yesterday["deaths"], breaks = b)

To plot the other Covid-19 variables, reporting number of confirmed cases, number of tests and number of people who have recovered, we can subset the relevant variables and pipe the result to the plot() function (noting the [caveat](https://t.co/NyIrKHQSUy) that code containing pipes may be hard to debug) as follows:

w\_yesterday %>%

dplyr::select(deaths, confirmed, tests, recovered) %>%

plot()

**Making maps with tmap**

The [mapping chapter](https://geocompr.robinlovelace.net/adv-map.html) of *Geocomputation with R* shows how the tmap package enables publication-quality maps to be created with concise and relatively commands, such as:

tm\_shape(w\_yesterday) +

tm\_polygons(c("deaths", "recovered"))

We can modify the palette and scale as follows:

tm\_shape(w\_yesterday) +

tm\_polygons(

c("deaths", "recovered"),

palette = "viridis",

style = "log10\_pretty"

)

The map can be further improved by adding graticules representing the curvature of the Earth, created as follows:

g = st\_graticule(w\_yesterday)

It’s also worth moving the legend:

tm\_shape(g) +

tm\_lines(col = "grey") +

tm\_shape(w\_yesterday) +

tm\_polygons(

c("deaths", "recovered"),

palette = "viridis",

style = "log10\_pretty"

) +

tm\_layout(legend.position = c(0.01, 0.25))

A problem with choropleth maps is that they can under-represent small areas.  
To overcome this issue we can use dot size instead of color to represent number:

tm\_shape(g) +

tm\_lines(col = "grey") +

tm\_shape(w\_yesterday) +

tm\_polygons() +

tm\_layout(legend.position = c(0.01, 0)) +

tm\_shape(w\_yesterday) +

tm\_dots(

col = c("red", "green"),

size = c("deaths", "recovered"),

palette = "viridis"

)

One question I have here: make the size legend have style = "log10\_pretty" also?

**Making animated maps**

The animation at the beginning of this article shows how dynamic graphics can communicate change effectively.  
Animated maps are therefore useful for showing evolving geographic phenomena, such as the spread of Covid-19 worldwide.  
As covered in section [8.3](https://geocompr.robinlovelace.net/adv-map.html#animated-maps) of *Geocomputation with R*, animated maps can be created with tmap by extending the tm\_facet() functionality.  
So let’s start by creating a facetted map showing the total number of deaths on the first day of each month in our data:

w$Date = as.character(w$date)

tm\_shape(g) +

tm\_lines(col = "grey") +

tm\_shape(w\_yesterday) +

tm\_polygons(

"Pop/km2",

palette = "viridis",

style = "log10\_pretty",

n = 3

) +

tm\_shape(w %>% filter(grepl(pattern = "01$", date))) +

tm\_dots(size = "deaths", col = "red") +

tm\_facets("Date", nrow = 1, free.scales.fill = FALSE) +

tm\_layout(

legend.outside.position = "bottom",

legend.stack = "horizontal"

)

To create an animated map, following instructions in Chapter [8](https://geocompr.robinlovelace.net/adv-map.html#animated-maps) of *Geocomputation with R*, we need to make some small changes to the code above:

m = tm\_shape(g) +

tm\_lines(col = "grey") +

tm\_shape(w\_yesterday) +

tm\_polygons(

"Pop/km2",

palette = "viridis",

style = "log10\_pretty",

n = 3

) +

tm\_shape(w %>% filter(grepl(pattern = "01$", date))) +

tm\_dots(size = "deaths", col = "red") +

tm\_facets(along = "Date", free.coords = FALSE) +

tm\_layout(legend.outside = TRUE)

tmap\_animation(m, "covid-19-animated-map-test.gif", width = 800)

browseURL("covid-19-animated-map-test.gif")

We made an animated map!  
The first version is rarely the best though, and the map above clearly could benefit from some adjustments before we plot the results for the whole dataset:

w$Date = paste0("Total deaths from 22nd January 2020 to ", w$date)

m = tm\_shape(g) +

tm\_lines(col = "grey") +

tm\_shape(w) +

tm\_polygons(

"Pop/km2",

palette = "viridis",

style = "log10\_pretty",

lwd = 0.5

) +

tm\_shape(w) +

tm\_dots(size = "deaths", col = "red") +

tm\_facets(along = "Date", free.coords = FALSE) +

tm\_layout(

main.title.size = 0.5,

legend.outside = TRUE

)

tmap\_animation(m, "covid-19-animated-map.gif", width = 1400, height = 600)

browseURL("covid-19-animated-map.gif")

**Conclusion**

This article has demonstrated how to work with and map geographic data using the free and open source statistical programming language R.  
It demonstrates that by representing analysis in code, research can be made reproducible and more accessible to others, encouraging transparent and open science.  
This has multiple advantages, from education and citizen engagement with the evidence to increased trust in the evidence on which important, life-or-death, decisions are made.

For further details on geographic data analysis in R in general, we recommend checkout out in-depth materials such as [*Geocomputation with R*](https://www.amazon.co.uk/Geocomputation-Chapman-Hall-Robin-Lovelace/dp/1138304514) and the in-progress open source book [*Spatial Data Science*](https://keen-swartz-3146c4.netlify.app/).

Session info

devtools::session\_info()

## ─ Session info ───────────────────────────────────────────────────────────────

## setting value

## version R version 3.6.3 (2020-02-29)

## os Ubuntu 18.04.4 LTS

## system x86\_64, linux-gnu

## ui X11

## language (EN)

## collate C.UTF-8

## ctype C.UTF-8

## tz UTC

## date 2020-04-25

##

## ─ Packages ───────────────────────────────────────────────────────────────────

## package \* version date lib source

## abind 1.4-5 2016-07-21 [1] CRAN (R 3.6.3)

## assertthat 0.2.1 2019-03-21 [1] CRAN (R 3.6.3)

## backports 1.1.6 2020-04-05 [1] CRAN (R 3.6.3)

## base64enc 0.1-3 2015-07-28 [1] CRAN (R 3.6.3)

## blogdown 0.18.1 2020-04-25 [1] Github (rstudio/blogdown@dbd9ca1)

## bookdown 0.18.1 2020-04-22 [1] Github (rstudio/bookdown@cd97d40)

## callr 3.4.3 2020-03-28 [1] CRAN (R 3.6.3)

## class 7.3-15 2019-01-01 [2] CRAN (R 3.6.3)

## classInt 0.4-3 2020-04-07 [1] CRAN (R 3.6.3)

## cli 2.0.2 2020-02-28 [1] CRAN (R 3.6.3)

## codetools 0.2-16 2018-12-24 [2] CRAN (R 3.6.3)

## colorspace 1.4-1 2019-03-18 [1] CRAN (R 3.6.3)

## crayon 1.3.4 2017-09-16 [1] CRAN (R 3.6.3)

## crosstalk 1.1.0.1 2020-03-13 [1] CRAN (R 3.6.3)

## curl 4.3 2019-12-02 [1] CRAN (R 3.6.3)

## DBI 1.1.0 2019-12-15 [1] CRAN (R 3.6.3)

## desc 1.2.0 2018-05-01 [1] CRAN (R 3.6.3)

## devtools 2.3.0 2020-04-10 [1] CRAN (R 3.6.3)

## dichromat 2.0-0 2013-01-24 [1] CRAN (R 3.6.3)

## digest 0.6.25 2020-02-23 [1] CRAN (R 3.6.3)

## dplyr \* 0.8.5 2020-03-07 [1] CRAN (R 3.6.3)

## e1071 1.7-3 2019-11-26 [1] CRAN (R 3.6.3)

## ellipsis 0.3.0 2019-09-20 [1] CRAN (R 3.6.3)

## evaluate 0.14 2019-05-28 [1] CRAN (R 3.6.3)

## fansi 0.4.1 2020-01-08 [1] CRAN (R 3.6.3)

## fs 1.4.1 2020-04-04 [1] CRAN (R 3.6.3)

## glue 1.4.0 2020-04-03 [1] CRAN (R 3.6.3)

## hms 0.5.3 2020-01-08 [1] CRAN (R 3.6.3)

## htmltools 0.4.0 2019-10-04 [1] CRAN (R 3.6.3)

## htmlwidgets 1.5.1 2019-10-08 [1] CRAN (R 3.6.3)

## KernSmooth 2.23-16 2019-10-15 [2] CRAN (R 3.6.3)

## knitr 1.28 2020-02-06 [1] CRAN (R 3.6.3)

## lattice 0.20-38 2018-11-04 [2] CRAN (R 3.6.3)

## leafem 0.1.1 2020-04-05 [1] CRAN (R 3.6.3)

## leaflet 2.0.3 2019-11-16 [1] CRAN (R 3.6.3)

## leafsync 0.1.0 2019-03-05 [1] CRAN (R 3.6.3)

## lifecycle 0.2.0 2020-03-06 [1] CRAN (R 3.6.3)

## lwgeom 0.2-3 2020-04-12 [1] CRAN (R 3.6.3)

## magrittr 1.5 2014-11-22 [1] CRAN (R 3.6.3)

## memoise 1.1.0 2017-04-21 [1] CRAN (R 3.6.3)

## munsell 0.5.0 2018-06-12 [1] CRAN (R 3.6.3)

## pillar 1.4.3 2019-12-20 [1] CRAN (R 3.6.3)

## pkgbuild 1.0.6 2019-10-09 [1] CRAN (R 3.6.3)

## pkgconfig 2.0.3 2019-09-22 [1] CRAN (R 3.6.3)

## pkgload 1.0.2 2018-10-29 [1] CRAN (R 3.6.3)

## png 0.1-7 2013-12-03 [1] CRAN (R 3.6.3)

## prettyunits 1.1.1 2020-01-24 [1] CRAN (R 3.6.3)

## processx 3.4.2 2020-02-09 [1] CRAN (R 3.6.3)

## ps 1.3.2 2020-02-13 [1] CRAN (R 3.6.3)

## purrr 0.3.4 2020-04-17 [1] CRAN (R 3.6.3)

## R6 2.4.1 2019-11-12 [1] CRAN (R 3.6.3)

## raster 3.1-5 2020-04-19 [1] CRAN (R 3.6.3)

## RColorBrewer 1.1-2 2014-12-07 [1] CRAN (R 3.6.3)

## Rcpp 1.0.4.6 2020-04-09 [1] CRAN (R 3.6.3)

## readr 1.3.1 2018-12-21 [1] CRAN (R 3.6.3)

## remotes 2.1.1 2020-02-15 [1] CRAN (R 3.6.3)

## rlang 0.4.5 2020-03-01 [1] CRAN (R 3.6.3)

## rmarkdown 2.1 2020-01-20 [1] CRAN (R 3.6.3)

## rprojroot 1.3-2 2018-01-03 [1] CRAN (R 3.6.3)

## scales 1.1.0 2019-11-18 [1] CRAN (R 3.6.3)

## sessioninfo 1.1.1 2018-11-05 [1] CRAN (R 3.6.3)

## sf \* 0.9-2 2020-04-14 [1] CRAN (R 3.6.3)

## sp 1.4-1 2020-02-28 [1] CRAN (R 3.6.3)

## stars 0.4-1 2020-04-07 [1] CRAN (R 3.6.3)

## stringi 1.4.6 2020-02-17 [1] CRAN (R 3.6.3)

## stringr 1.4.0 2019-02-10 [1] CRAN (R 3.6.3)

## testthat 2.3.2 2020-03-02 [1] CRAN (R 3.6.3)

## tibble 3.0.1 2020-04-20 [1] CRAN (R 3.6.3)

## tidyselect 1.0.0 2020-01-27 [1] CRAN (R 3.6.3)

## tmap \* 3.0 2020-04-22 [1] Github (mtennekes/tmap@4a9249a)

## tmaptools 3.0-1 2020-04-22 [1] Github (mtennekes/tmaptools@adbf1da)

## units 0.6-6 2020-03-16 [1] CRAN (R 3.6.3)

## usethis 1.6.0 2020-04-09 [1] CRAN (R 3.6.3)

## vctrs 0.2.4 2020-03-10 [1] CRAN (R 3.6.3)

## viridisLite 0.3.0 2018-02-01 [1] CRAN (R 3.6.3)

## withr 2.2.0 2020-04-20 [1] CRAN (R 3.6.3)

## xfun 0.13 2020-04-13 [1] CRAN (R 3.6.3)

## XML 3.99-0.3 2020-01-20 [1] CRAN (R 3.6.3)

## yaml 2.2.1 2020-02-01 [1] CRAN (R 3.6.3)

##

## [1] /home/runner/work/\_temp/Library

## [2] /opt/R/3.6.3/lib/R/library