Basic analysis of the SARS-CoV-2 (Covid-19) pandemic

Louis Boulanger

December 19, 2020

Contents

1 Foreword

In order to process this computational document, you will need to install:

- **Emacs** 25.0 or greater (no guarantees on previous versions of Emacs)
- **Python** 3.6.0 or greater
- **R** 3.4

```python
import sys
if sys.version_info.major < 3 or sys.version_info.minor < 6:
    print("Please use Python 3.6 (or higher)!")
```

```elisp
(unless (featurep 'ob-python)
  (print "Please activate python in org-babel
  ↪   (org-babel-do-lnaguages)!"))
```

```elisp
(unless (featurep 'ob-R)
  (print "Please activate R in org-babel
  ↪   (org-babel-do-lnaguages)!"))
```

2 Introduction

The goal of this document is to provide an analysis of the Coronavirus pandemic numbers, in particular the number of cases for a select number of countries since the beginning of the pandemic. The data is provided by the John Hopkins University Center for Systems Science and Engineering (JHU CSSE), and freely available on GitHub. The analysis focuses primarily on the `time_series_covid19_confirmed_global.csv` file, containing time series for the confirmed cases for each state/province of each affected country. Following the data pre-processing, the analysis will show the evolution of the cases in:

- Belgium
- China (all provinces except Hong-Kong)
- Hong-Kong
- France (except DOM/TOMs)
- Germany
- Iran
- Italy
- Japan
- South Korea
- The Netherlands (except colonies)

- Portugal
- Spain
- United Kingdom (except colonies)
- United States of America

3 Data pre-processing

The data containing the amount of confirmed cases of Covid-19 is taken from the aforementioned JHU CSSE GitHub repository; specifically, the file used is: `https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv` . According to the information on the repository, the data is updated every day at 23:59 UTC. The file contains the following fields:

llX Column name	Type	Description
Province/State	Text (can be empty)	The state or province of a country, if any.
Country/Region	Text	A country or region affected by the Covid-19 pandemic.
Lat	Floating number	The latitude of the general location of the country or region
Long	Floating number	The longitude of the general location of the country or region
<mm/dd/yy>	Integer	The number of cases for the specified country/province on the specified day

Each of the columns corresponding for the data of a date are written in the American date format of month/day/year.

3.1 Downloading the data

In order to save time and resources, the data is downloaded only if:

- the file has not been downloaded before,
- or if the file is obsolete, as in the last day recorded is in the past.

Particular care must be taken for the row containing "Korea, South": since the document uses commas to separate the fields, we need to make sure that we don't separate the country name into two separate fields.

```
from urllib.request import urlopen
import datetime

temp_file_name = 'data.csv'

# Downloads the data from GitHub
def download_data():
    data = urlopen(data_url).read()
    with open(temp_file_name, 'wb') as f:
        f.write(data)

# Tries to read data from the local file and returns the
↪   content
# parsed as a series of lines
def read_data():
    try:
        with open(temp_file_name, 'r') as f:
            data = f.read()
            lines = data.split('\n')
            table = [line.replace("\"Korea, South\"", "South
            ↪   Korea").split(',') for line in lines]
            return table[:-2] # Removing the empty last line
    except IOError as e:
        raise e

# Decides whether or not to download the file from GitHub
↪   based on the
# presence of a local file and the last recorded date in the
↪   local
# file
def try_download_data():
    data = None
    try:
        data = read_data()
        last_date = datetime.datetime.strptime(data[0][-1],
        ↪   "%m/%d/%y")
        today = datetime.datetime.today()
        if today - last_date > datetime.timedelta(day=1):
            print("Data obsolete, downloading new data...")
            download_data()
            data = read_data()
    except IOError:
        download_data()
        data = read_data()
    finally:
        return data

data = try_download_data()
```

Let's print the first five lines for the last two dates.

Province/State	Country/Region	Lat	Long	1/22/20	12/14/20	12/15/20
	Afghanistan	33.93911	67.709953	0	48718	48952
	Albania	41.1533	20.1683	0	49191	50000
	Algeria	28.0339	1.6596	0	92597	93065
	Andorra	42.5063	1.5218	0	7382	7382

3.2 Checking for missing data

The data is generated automatically, but it's never too prudent to check if some data is malformed or missing. We can assume that the first 4 rows, containing information about the countries and provinces, are correct, since they are the key of the real data.

```
valid_data = []

valid_data.append(data[0])
for row in data[1:]:
    missing = any([value == '' for value in row[4:]])
    if missing:
        print(row)
    else:
        valid_data.append(row)
```

3.3 Extracting the relevant information

As mentioned in the introduction, we only care for a few countries in the list; we will filter the rows in which we are not interested. There is also little use for the `Long` and `Lat` columns, so we will drop them.

We also need to group the different Chinese provinces into one row, and add the values together, while counting Hong Kong as a separate country.

```
target_countries = [
    [None, "Belgium"],
    ["Hong Kong", None],
    ["Hong Kong", "China"], # China without Hong Kong
    [None, "France"],
    [None, "Germany"],
    [None, "Iran"],
    [None, "Italy"],
    [None, "Japan"],
    [None, "South Korea"],
    [None, "Netherlands"],
    [None, "United Kingdom"],
    [None, "US"]
]

def is_target_country(province, country):
    specific_province = lambda t, p, c: t[0] == p and t[1] is
    ↪  None
    without_specific_province = lambda t, p, c: t[0] is not
    ↪  None and t[0] != p and t[1] == c
    without_provinces = lambda t, p, c: t[0] is None and p ==
    ↪  "" and t[1] == c
    check = lambda t, p, c: specific_province(t, p, c) or
    ↪  without_specific_province(t, p, c) or
    ↪  without_provinces(t, p, c)
    res = [check(target, province, country) for target in
    ↪  target_countries]
    return any(res)

extracted_data = []
extracted_data.append([data[0][1]] + data[0][4:])

for row in valid_data[1:]:
    if "Korea" in row[1]:
        print(row[1])
    if is_target_country(row[0], row[1]):
        # print(row[0])
        if row[0] == "Hong Kong":
            extracted_data.append(["Hong Kong"] + row[4:])
        elif row[1] == "China":
            try:
                idx = [row[0] for row in
                ↪  extracted_data].index("China")
                extracted_data[idx][1:] = [int(a) + int(b)
                ↪  for a, b in zip(extracted_data[-1][1:],
                ↪  row[4:])]
            except ValueError:
                extracted_data.append(["China"] + row[4:])
        else:
            extracted_data.append([row[1]] + row[4:])
```

Let's look at the last five days of the countries we selected.

Country/Region	12/11/20	12/12/20	12/13/20	12/14/20	12/15/20
Belgium	600397	603159	608137	609211	611422
China	8673	8742	8838	8920	9018
Hong Kong	7377	7446	7541	7623	7721
France	2350923	2350793	2376228	2379291	2390419
Germany	1314309	1336101	1350810	1357261	1391086
Iran	1092407	1100818	1108269	1115770	1123474
Italy	1805873	1825775	1843712	1855737	1870576
Japan	175310	178272	180639	182311	184752
South Korea	41736	42766	43484	44364	45442
Netherlands	594523	603603	613487	621944	628577
US	15913292	16134237	16325615	16518420	16716777
United Kingdom	1809455	1830956	1849403	1869666	1888116

3.4 Date conversion

The dates are currently expressed using the American format `mm/dd/yy`. We will need to convert them into proper dates in order to analyze the data further.

```
extracted_data[0][1:] = [datetime.datetime.strptime(date,
↪  "%m/%d/%y") for date in extracted_data[0][1:]]
```

Now, let's take a look at the last two days again:

Country/Region	2020-12-14 00:00:00	2020-12-15 00:00:00
Belgium	609211	611422
China	8920	9018
Hong Kong	7623	7721
France	2379291	2390419
Germany	1357261	1391086
Iran	1115770	1123474
Italy	1855737	1870576
Japan	182311	184752
South Korea	44364	45442
Netherlands	621944	628577
US	16518420	16716777
United Kingdom	1869666	1888116

3.5 Conversion into a regular table

Right now, each date is represented as a column; we will flip the table and have the dates as rows, and the countries as columns.

```
flipped_data = [[str(row[i]) for row in extracted_data] for i
↪  in range(0, len(extracted_data[0]))]
flipped_data[0][0] = "Date"
flipped_data[0] = [s.replace(" ", "") for s in
↪  flipped_data[0]]
```

Let's look at the data for a few countries now:

Date	Belgium	China	HongKong
2020-12-11 00:00:00	600397	8673	7377
2020-12-12 00:00:00	603159	8742	7446
2020-12-13 00:00:00	608137	8838	7541
2020-12-14 00:00:00	609211	8920	7623
2020-12-15 00:00:00	611422	9018	7721

This format is much more usable now.

3.6 Transferring the data from Python to R

We will switch from Python to R for the analysis, since R is a much better tool than Python for that. We will use org-mode's data exchange utility in order to transfer the data.

```
[flipped_data[0], None] + flipped_data[1:]
```

In R, we get the data in the form of a data-frame, and the strings must be converted.

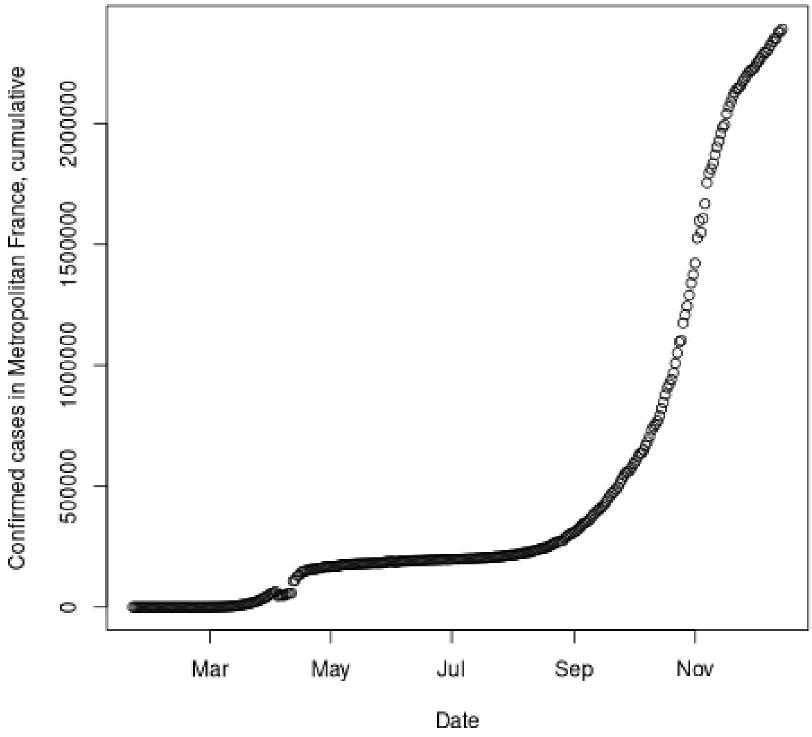
```
data$Date <- as.Date(data$Date)
summary(data)
```

```
     Date               Belgium           China         HongKong
Min.   :2020-01-22   Min.   :     0   Min.   :  10   Min.   :   0
1st Qu.:2020-04-13   1st Qu.: 30589   1st Qu.:2276   1st Qu.:1009
Median :2020-07-04   Median : 61838   Median :2527   Median :1258
Mean   :2020-07-04   Mean   :132884   Mean   :3903   Mean   :2682
3rd Qu.:2020-09-24   3rd Qu.:108768   3rd Qu.:6338   3rd Qu.:5056
Max.   :2020-12-15   Max.   :611422   Max.   :9018   Max.   :7721
    France           Germany             Iran              Italy
Min.   :      0   Min.   :      0   Min.   :      0   Min.   :      0
1st Qu.: 110836   1st Qu.: 130072   1st Qu.:  73303   1st Qu.: 159516
Median : 197994   Median : 197198   Median : 237878   Median : 241419
Mean   : 494998   Mean   : 278940   Mean   : 301025   Mean   : 362680
3rd Qu.: 513732   3rd Qu.: 281346   3rd Qu.: 436319   3rd Qu.: 304323
Max.   :2390419   Max.   :1391086   Max.   :1123474   Max.   :1870576
    Japan          SouthKorea     Netherlands           US
Min.   :     2   Min.   :    1   Min.   :     0   Min.   :       1
1st Qu.:  7773   1st Qu.:10537   1st Qu.: 26551   1st Qu.:  585518
Median : 19461   Median :13091   Median : 50548   Median : 2833290
Mean   : 45932   Mean   :15651   Mean   :118150   Mean   : 4329085
3rd Qu.: 80490   3rd Qu.:23455   3rd Qu.:103141   3rd Qu.: 6972152
Max.   :184752   Max.   :45442   Max.   :628577   Max.   :16716777
UnitedKingdom
Min.   :      0
1st Qu.:  97068
Median : 284900
Mean   : 420031
3rd Qu.: 416363
Max.   :1888116
```

3.7 Quick analysis of the data

Now, we can inspect the data and look at the curve for a country, for example, France.

```
plot(data[,'Date'], data[,'France'], xlab="Date",
     ylab="Confirmed cases in Metropolitan France,
      ↪  cumulative")
```



4 Comparative analysis of the accumulated cases in the selected countries

4.1 Linear scale

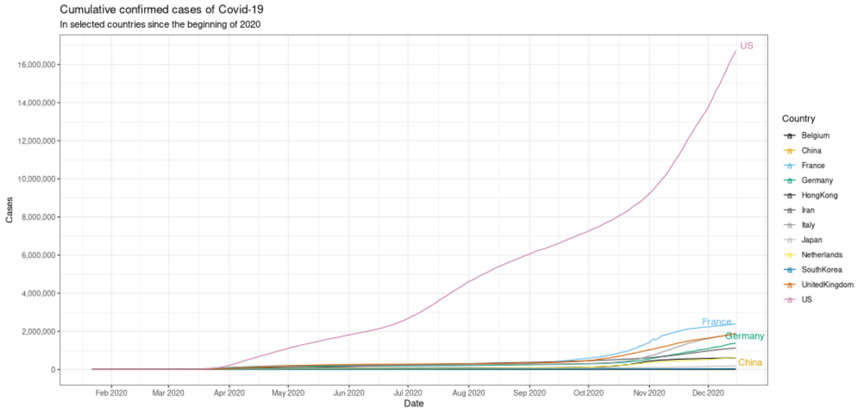
Let's build a graph showing the confirmed cases for all of the selected countries, on the same graph. The goal of such a graphic is to compare the countries, determine outliers from a glance, and see the general shape of the phenomenon across different places in the world. An *interactive* graph might be easier to read and parse, such as the excellent ones in the Financial Times website.

```
library(tidyverse)
library(ggrepel)
library(scales)

last_date <- data %>%
  gather(Country, Cases, Belgium:UnitedKingdom) %>%
  dplyr::filter(Date == tail(data$Date, 1), Country %in%
   ↪  c("US", "France", "Germany", "China"))

# Color-blind friendly palette taken from
# https://bconnelly.net/posts/creating_colorblind-friendly_fi⌋
 ↪  gures/
# with added grayscale values
palette <- c("#000000", "#E69F00", "#56B4E9", "#009E73",
             "#292929", "#555555", "#999999", "#BBBBBB",
             "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

data %>%
  gather(Country, Cases, Belgium:UnitedKingdom) %>%
  ggplot(aes(x=Date, y=Cases, colour=Country)) +
  geom_line() +
  scale_x_date(breaks = pretty_breaks(8), labels =
   ↪  date_format("%b %Y")) +
  scale_color_manual(values=palette) +
  scale_y_continuous(labels = comma_format(), breaks =
   ↪  pretty_breaks(8)) +
  geom_text_repel(data=last_date, aes(label = Country)) +
  ggtitle("Cumulative confirmed cases of Covid-19",
   ↪  subtitle="In selected countries since the beginning of
   ↪  2020") +
  theme_bw()
```



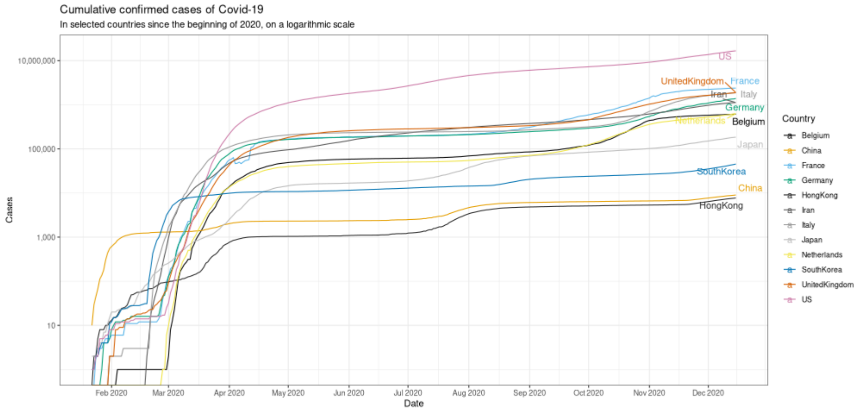
This graph is dominated by the spike of the confirmed cases in the United States, which outweighs the other countries'. It is also clear that the second wave, which started for the selected countries in mid-October of 2020, was significantly larger than the first wave.

4.2 Logarithmic scale

Let's now take a look at the same graph, but with a logarithmic scale.

```
last_date <- data %>%
  gather(Country, Cases, Belgium:UnitedKingdom) %>%
  dplyr::filter(Date == tail(data$Date, 1))


data %>%
  gather(Country, Cases, Belgium:UnitedKingdom) %>%
  ggplot(aes(x=Date, y=Cases, colour=Country)) +
  geom_line() +
  scale_x_date(breaks = pretty_breaks(8), labels =
   ↪  date_format("%b %Y")) +
  scale_color_manual(values=palette) +
  scale_y_continuous(trans="log10", labels = comma_format()) +
  geom_text_repel(data=last_date, aes(label = Country)) +
  ggtitle("Cumulative confirmed cases of Covid-19",
   ↪  subtitle="In selected countries since the beginning of
   ↪  2020, on a logarithmic scale") +
  theme_bw()
```



This graph shows in better details the different waves of infection among the selected countries. We can clearly see how the pandemic started in China, starting in January, and was mitigated in mid-February; while the other countries experienced the rapid growth of the cases in March. The period of international lockdown during the spring and summer is visible, as well the different waves of infections later in the year.