Understanding the COVID-19 pandemic

A data-driven approach using the sars2pack R package

Vincent Carey Charles Morefield John Mallery Sean Davis

2020-04-02

Contents

Part I

Introduction

1 Motivation

The COVID-19 pandemic is ongoing. The situation on the ground is changing daily as captured by data reported around the world. The sars2pack package aims to:

- Provide timely, computable, easily accessible data for research, policy-making, and educational purposes.
- Promote easy computational experimentation with COVID-19 data
- Serve as a source of documentation and education for available COVID-19 analysis and visualization approaches.
- House recipes for regularly updated data products such as spreadsheets and maps for use by non-R-savvy data consumers.
- Collect interesting data stories along with code as data science training resources for the many biomedical researchers who cannot currently perform experiments

```
## Warning: package 'BiocStyle' was built under R version 3.6.2
```

```
## Warning: package 'tibble' was built under R version 3.6.2
```

2 Quick start

2.1 Installation

```
BiocManager::install('seandavi/sars2pack')
```

2.2 COVID-19 resources in this package

The COVID-19 data in this package are, right now, focused toward time-series descriptions of confirmed cases, deaths, testing, and recovered cases. **There is no requirement that this remain the case**. Contributions of additional data resources or simple accessor functions will only add to our abilities to use data science and modeling to understand COVID-19.

Request for help: I would be more than happy to accept help with defining new data resources. Consider a pull request (or an issue for non-programmer types).

2.2.1 Epidemic time-series data

- JHU : global deaths, confirmed cases, and recovered time series data; *does not include fine-level United States data.* See `jhu_data()`.

- New York Times : United states state and county level deaths, confirmed cases time series. See `nytimes_county_data()` and `nytimes_state_data`.
- USAFacts : Alternative United states state and county level deaths and confirmed cases time series

2.3 Additional resources described in this vignette

-

3 Epidemic time series data

Usage of each of the time series datasets follows a similar pattern.

1. Fetch a tidy `tbl_df` using a function such as `jhu_data()`
2. In the resulting `tbl_df`, the columns `date` (of type `date`) and `count` of type `numeric` columns are standard.
3. Additional columns describe locations, subsets of data (such as `confirmed`, `deaths`, `recovered`) and vary from dataset to dataset.

Regardless of the original format of the data, the `sars2pack` datasets are presented as tidy data to facilitate `dplyr`, `ggplot`, and other fluid analysis approaches to apply directly.

3.1 Access data

This section briefly introduces how to access the data resources in this package. Note that many of the functions below **require a network connection** to get updated data.

3.1.1 JHU Dataset

```
jhu = jhu_data()
class(jhu)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
dim(jhu)
```

```
## [1] 38332     7
```

Column names include:

```
colnames(jhu)
```

```
## [1] "ProvinceState" "CountryRegion" "Lat"           "Long"
## [5] "date"          "count"         "subset"
```

And a very small subset of the data.

```
head(jhu,3)
```

```
## # A tibble: 3 x 7
##   ProvinceState CountryRegion   Lat  Long date       count subset
##   <chr>         <chr>         <dbl> <dbl> <date>     <dbl> <chr>
## 1 <NA>          Afghanistan      33    65 2020-01-22     0 confirmed
## 2 <NA>          Afghanistan      33    65 2020-01-23     0 confirmed
## 3 <NA>          Afghanistan      33    65 2020-01-24     0 confirmed
```

3.1.2 USAFacts Dataset

```
usa_facts = usa_facts_data()
class(usa_facts)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
dim(usa_facts)
```

```
## [1] 434792      7
```

Column names include:

```
colnames(usa_facts)
```

```
## [1] "county_fips" "county"      "state"       "state_fips"  "subset"
## [6] "date"        "count"
```

And a very small subset of the data.

```
head(usa_facts,3)
```

```
## # A tibble: 3 x 7
##   county_fips county                state state_fips subset    date       count
##         <dbl> <chr>                 <chr>      <dbl> <chr>     <date>     <dbl>
## 1           0 Statewide Unallocated AL             1 confirmed 2020-01-22     0
## 2           0 Statewide Unallocated AL             1 confirmed 2020-01-23     0
## 3           0 Statewide Unallocated AL             1 confirmed 2020-01-24     0
```

3.1.3 NYTimes datasets

```
nytimes_state = nytimes_state_data()
class(nytimes_state)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
dim(nytimes_state)
```

```
## [1] 3658    5
```

Column names include:

```
colnames(nytimes_state)
```

```
## [1] "date"   "state"  "fips"   "count"  "subset"
```

And a very small subset of the data.

```
head(nytimes_state,3)
```

```
## # A tibble: 3 x 5
##   date       state      fips  count subset
##   <date>     <chr>      <chr> <dbl> <chr>
## 1 2020-01-21 Washington 00053     1 confirmed
## 2 2020-01-22 Washington 00053     1 confirmed
## 3 2020-01-23 Washington 00053     1 confirmed
```

```
nytimes_county = nytimes_county_data()
class(nytimes_county)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
dim(nytimes_county)
```

```
## [1] 66502     6
```

```
colnames(nytimes_county)
```

```
## [1] "date"   "county" "state"  "fips"   "count"  "subset"
```

4 Use cases

4.1 Basic data exploration

In this section, we will be using a combination of [dplyr] and [ggplot2] to explore the COVID-19 global data from JHU. For details on this dataset, see the help using `?jhu_data`.

The next line of code will do a (set of) network calls to fetch the most up-to-date dataset from the JHU github repository.

```
jhu = jhu_data()
head(jhu,3)
```

```
## # A tibble: 3 x 7
##   ProvinceState CountryRegion   Lat  Long date       count subset
##   <chr>         <chr>         <dbl> <dbl> <date>     <dbl> <chr>
## 1 <NA>          Afghanistan      33    65 2020-01-22     0 confirmed
## 2 <NA>          Afghanistan      33    65 2020-01-23     0 confirmed
## 3 <NA>          Afghanistan      33    65 2020-01-24     0 confirmed
```

We now want to ask a series of questions about the dataset.

- **How many records are in the dataset?**

```
nrow(jhu)
```

```
## [1] 38332
```

- **How many different countries are represented?**

```
length(unique(jhu$CountryRegion))
```

```
## [1] 181
```

Most records have no listing for `ProvinceState` column. Let's look at a few of those to get an idea of what is there when not empty:

- **What is in the `ProvinceState` column?**

To answer this question, we will be using `dplyr`, so some familiarity with that package will be helpful to follow this code.

```
jhu %>%
    dplyr::filter(!is.na(ProvinceState)) %>%
    dplyr::select(ProvinceState, CountryRegion) %>%
    unique() %>%
    head(10)
```

```
## # A tibble: 10 x 2
##    ProvinceState               CountryRegion
```

```
##    <chr>                          <chr>
##  1 Australian Capital Territory Australia
##  2 New South Wales                Australia
##  3 Northern Territory             Australia
##  4 Queensland                     Australia
##  5 South Australia                Australia
##  6 Tasmania                       Australia
##  7 Victoria                       Australia
##  8 Western Australia              Australia
##  9 Alberta                        Canada
## 10 British Columbia               Canada
```

We still have not looked at the most valuable information, the `date` and `count` columns in any detail.

- **What is the current count of confirmed cases by country, ordered by highest count down?**

There is a lot to unpack in the next code block, but the results are quite useful. We will use the DT package to make the dataset searchable and sortable.

```
library(DT)
latest_jhu_data = jhu %>%
    dplyr::filter(subset=='confirmed' & is.na(ProvinceState)) %>%
    dplyr::group_by(CountryRegion) %>%
    dplyr::slice(which.max(date)) %>%
    dplyr::arrange(desc(count))
DT::datatable(latest_jhu_data, rownames=FALSE)
```

Show 10 entries Search:

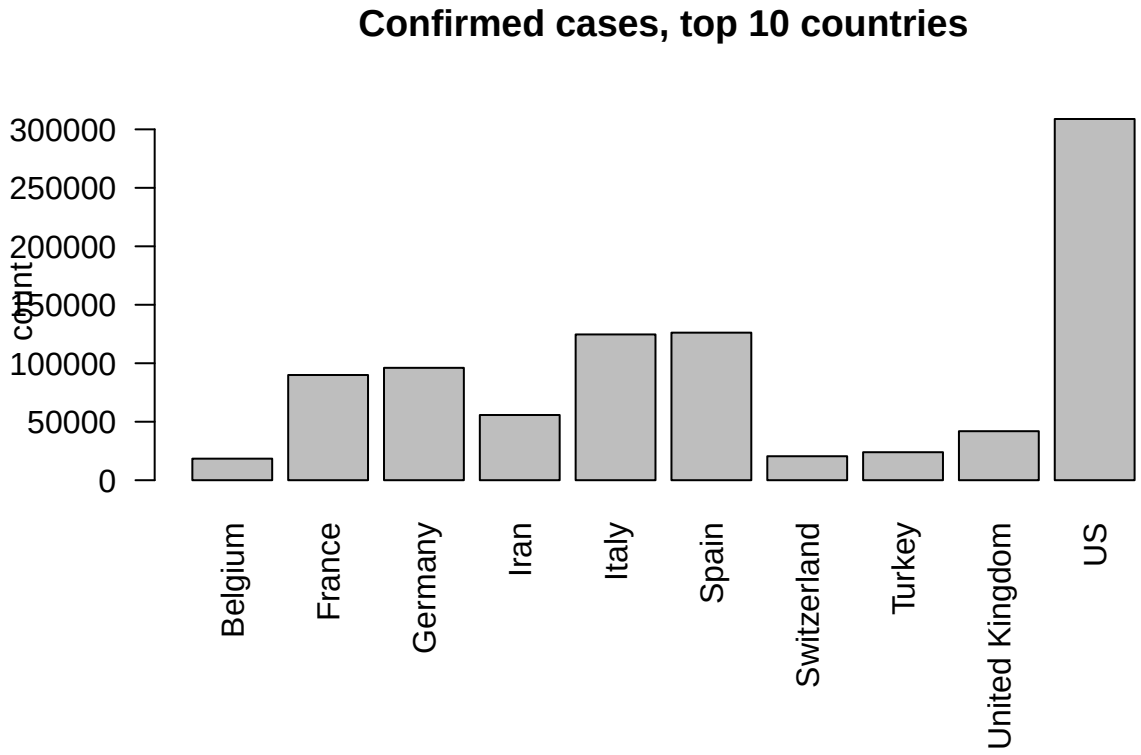
ProvinceState	CountryRegion	Lat	Long	date	count	subset
	US	37.0902	-95.7129	2020-04-04	308850	confirmed
	Spain	40	-4	2020-04-04	126168	confirmed
	Italy	43	12	2020-04-04	124632	confirmed
	Germany	51	9	2020-04-04	96092	confirmed
	France	46.2276	2.2137	2020-04-04	89953	confirmed
	Iran	32	53	2020-04-04	55743	confirmed
	United Kingdom	55.3781	-3.436	2020-04-04	41903	confirmed
	Turkey	38.9637	35.2433	2020-04-04	23934	confirmed
	Switzerland	46.8182	8.2275	2020-04-04	20505	confirmed
	Belgium	50.8333	4	2020-04-04	18431	confirmed

Showing 1 to 10 of 178 entries Previous 1 2 3 4 5 … 18 Next

Note: I included a little `is.na` in the filtering above to remove records where country data are split out over subparts. We revisit this below.

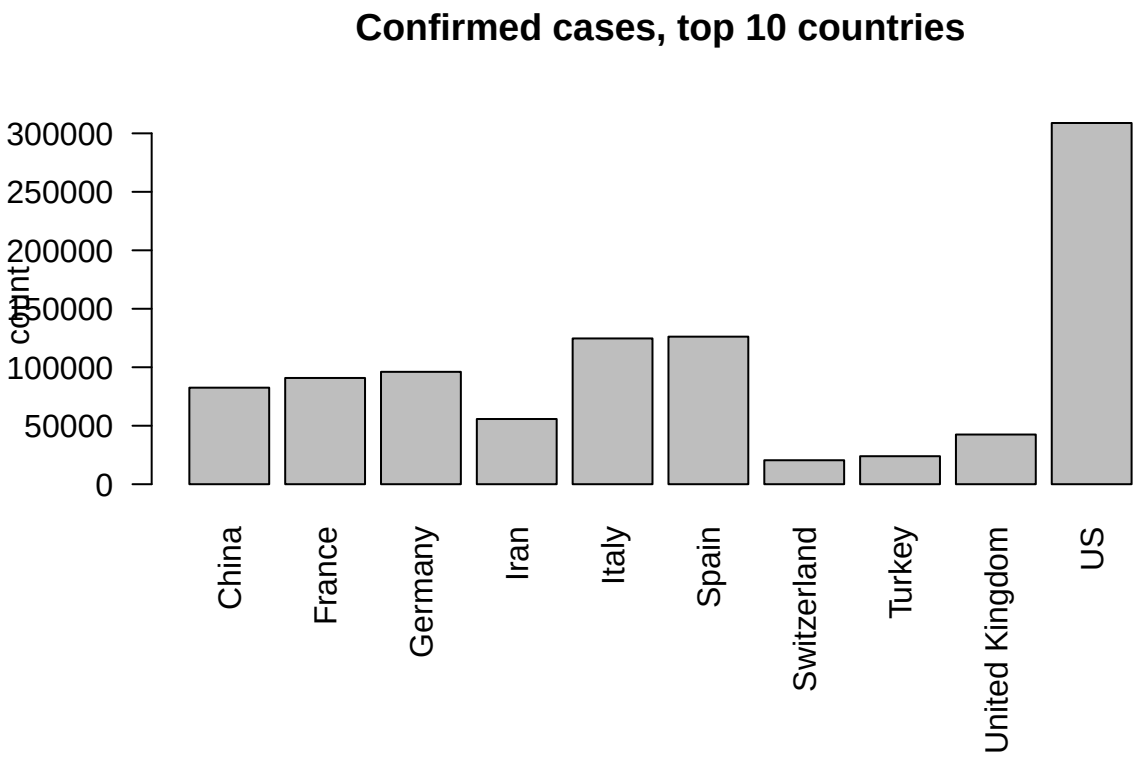
The data here could be usefully displayed as a graph as well.

```
par(las=2, mar=c(8,5,5,1))
barplot(count ~ CountryRegion, xlab = '',
        data=head(latest_jhu_data,10),
        main='Confirmed cases, top 10 countries')
```



We note here that China is not shown. That is because we limited the data to only rows that had empty ProvinceState records. To add those records back in, we sum all the China rows (and those of other countries like Australia, etc.) by country and then perform similar work to produce a final plot.

```
latest_jhu_data = jhu %>%
    dplyr::filter(subset=='confirmed') %>%
    dplyr::select(-c(ProvinceState,Lat,Long)) %>%
    dplyr::group_by(CountryRegion,date) %>%
    dplyr::summarize(count = sum(count)) %>%
    dplyr::slice(which.max(date)) %>%
    dplyr::arrange(desc(count))
par(las=2, mar=c(8,5,5,1))
barplot(count ~ CountryRegion, xlab = '',
        data=head(latest_jhu_data,10),
        main='Confirmed cases, top 10 countries')
```

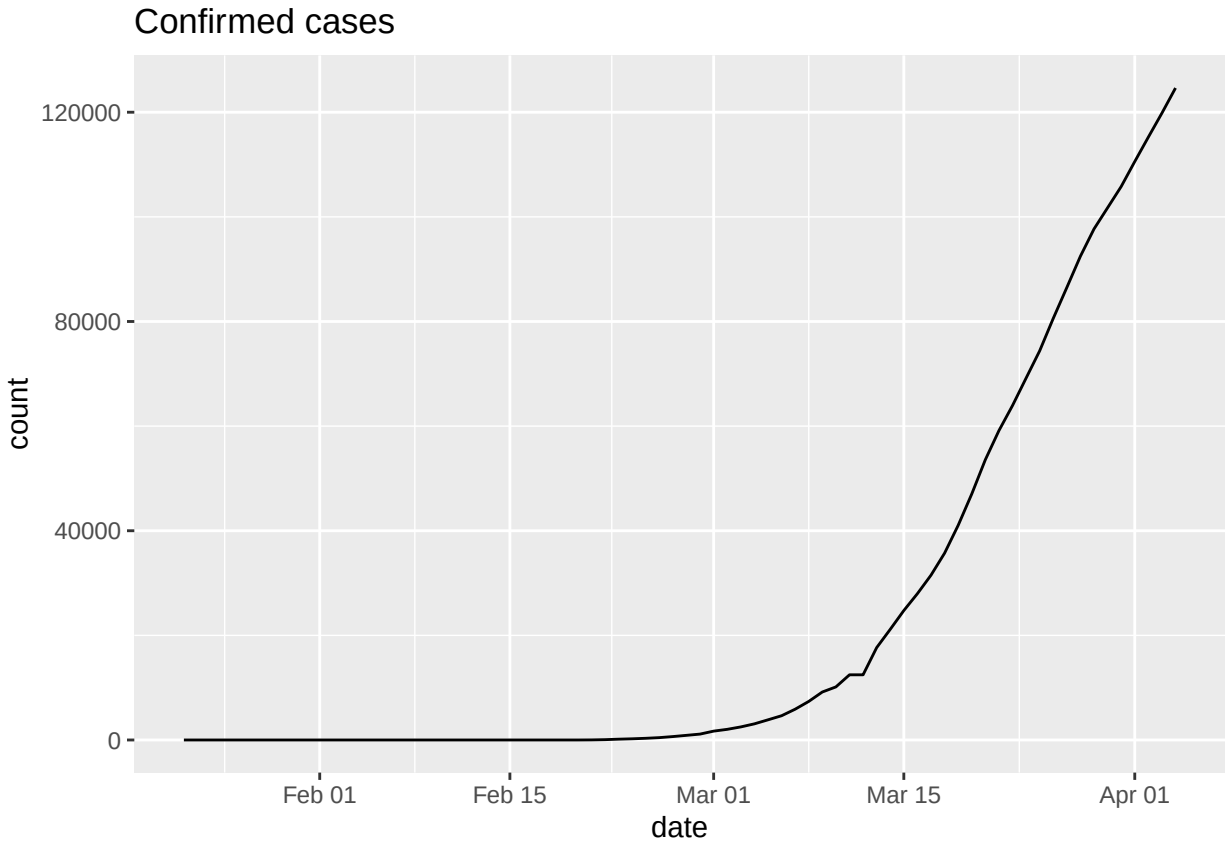


4.2 Visualize time series data

Up to now, we have ignored the time series aspects of the data and have sliced the dataset by country. In this section, we will be using dplyr and ggplot2 to visualize disease infection and deaths over time.

- **How have the cases in Italy changed over time?**

```
library(ggplot2)
italy_cc_ts = jhu %>%
    dplyr::filter(CountryRegion == 'Italy' & subset=='confirmed')
ggplot(italy_cc_ts,aes(x=date, y=count)) +
    geom_line() +
    ggtitle('Confirmed cases')
```



- **How do the confirmed cases in China, US, Italy, Spain, Germany, and Russia compare over time?**

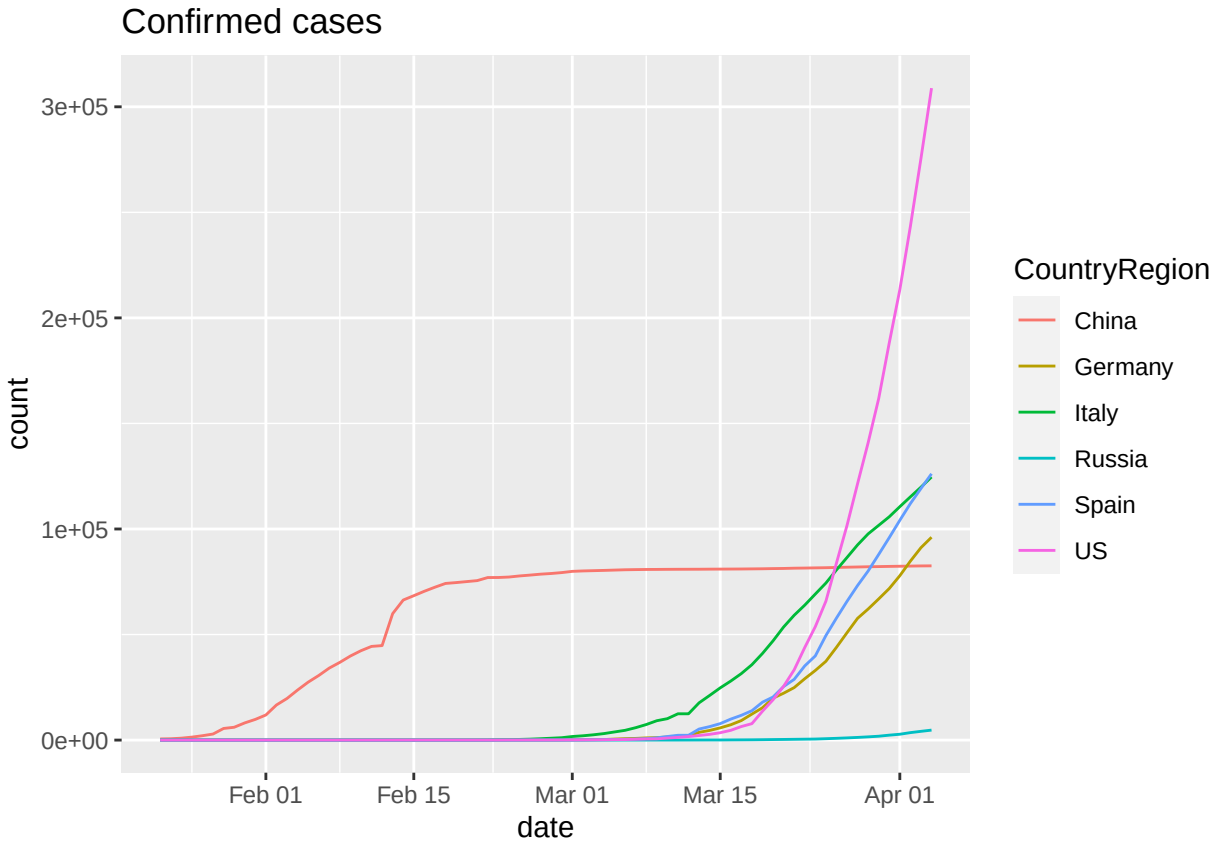
We have to play the same game of summing all values by country and date. Here, we filter the countries to be in a list of countries.

```
countries_of_interest = c('China','US','Italy','Spain','Germany','Russia')
library(ggplot2)
cc_ts = jhu %>%
    dplyr::group_by(CountryRegion,date) %>%
    dplyr::filter(CountryRegion %in% countries_of_interest & subset=='confirmed') %>%
    dplyr::summarize(count = sum(count))
head(cc_ts)
```

```
## # A tibble: 6 x 3
## # Groups:   CountryRegion [1]
##   CountryRegion date       count
##   <chr>         <date>     <dbl>
## 1 China         2020-01-22   548
## 2 China         2020-01-23   643
## 3 China         2020-01-24   920
## 4 China         2020-01-25  1406
## 5 China         2020-01-26  2075
## 6 China         2020-01-27  2877
```

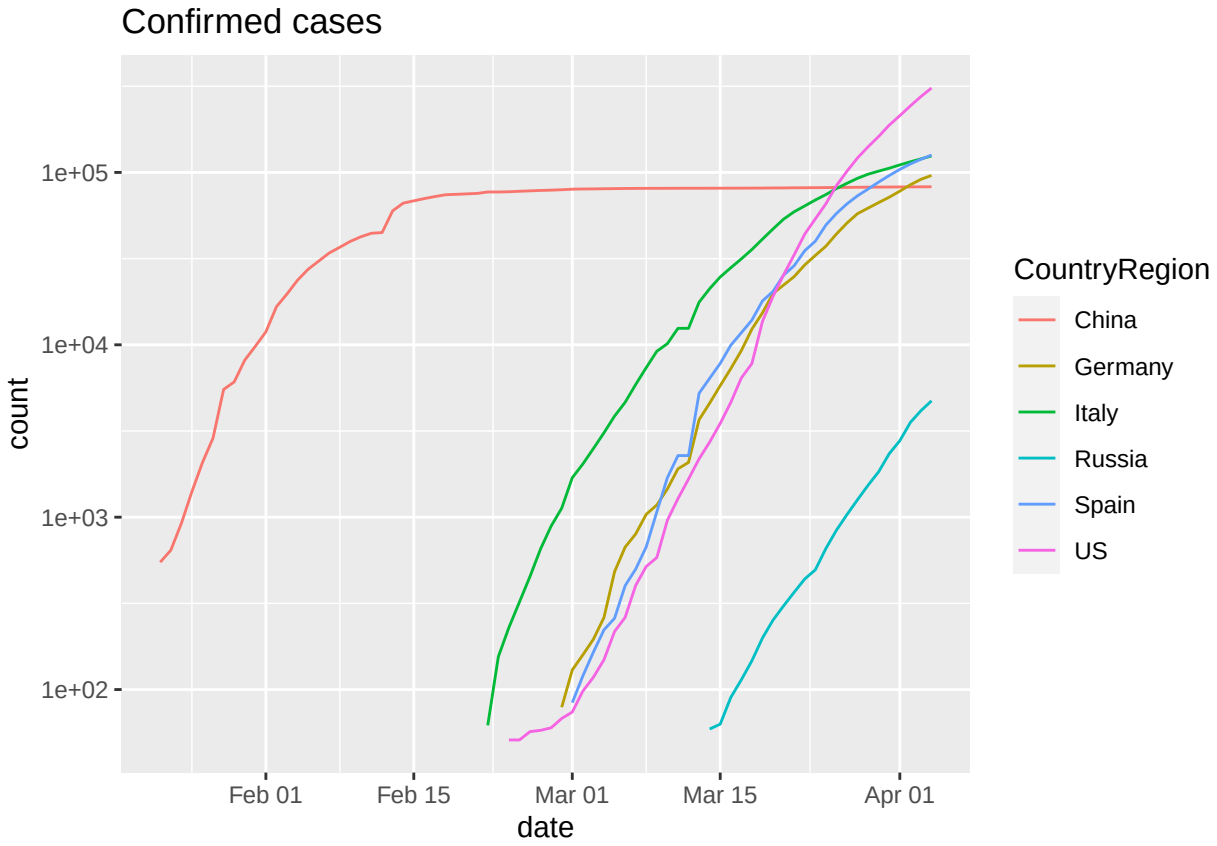
To make the plot, we use the ggplot2 grouping and coloring to provide curves for each country on the same axis.

```
ggplot(cc_ts,aes(x=date, y=count, group=CountryRegion)) +
    geom_line(aes(color=CountryRegion)) +
    ggtitle('Confirmed cases')
```



Changing to log scale can give a sense of the "exponentialness" of these data. Also, to remove zeros from the data (which cause problems when taking logs), we can filter data to include only values >=50. Note that ggplot2 will "do the right thing".

```
cc_ts %>%
    dplyr::filter(count>=50) %>%
    ggplot(aes(x=date, y=count, group=CountryRegion)) +
    geom_line(aes(color=CountryRegion)) +
    ggtitle('Confirmed cases') +
    scale_y_log10()
```



Consider the following questions based on the figure:

- What does the slope of the lines in this plot represent?
- What is the difference between China and other countries? What does this difference mean in terms of how the disease is spreading?
- What does each
- Pick an arbitrary level on the y-axis and look at the dates associated with each country's curve with respect to that level. What do differences along the x-axis tell us about where the countries are with respect to disease process?

5 Modeling Replication rate

5.1 Background

TODO: PARAPHRASE!!!!!

5.1.1 What is R_0?

R_0 is pronounced "R naught." It's a mathematical term that indicates how contagious an infectious disease is. It's also referred to as the reproduction number. As an infection spreads to new people, it reproduces itself.

R_0 tells you the average number of people who will catch a disease from one contagious person. It specifically applies to a population of people who were previously free of infection and haven't been vaccinated. If a disease has an R_0 of 18, a person who has the disease will transmit it to an average of 18 other people, as long as no one has been vaccinated against it or is already immune to it in their community.

5.1.2 What do R_0 values mean?

Three possibilities exist for the potential spread or decline of a disease, depending on its R_0 value:

- If R_0 is less than 1, each existing infection causes less than one new infection. In this case, the disease will decline and eventually die out.
- If R_0 equals 1, each existing infection causes one new infection. The disease will stay alive and stable, but there won't be an outbreak or an epidemic.
- If R_0 is more than 1, each existing infection causes more than one new infection. The disease will spread between people, and there may be an outbreak or epidemic.

Importantly, a disease's R_0 value only applies when everyone in a population is completely vulnerable to the disease. This means:

- no one has been vaccinated
- no one has had the disease before
- there's no way to control the spread of the disease

This combination of conditions is rare nowadays thanks to advances in medicine. Many diseases that were deadly in the past can now be contained and sometimes cured. For example, in 1918 there was a worldwide outbreak of the swine flu that killed 50 million people. According to a review article published in BMC Medicine, the R_0 value of the 1918 pandemic was estimated to be between 1.4 and 2.8. But when the swine flu, or H1N1 virus, came back in 2009, its R_0 value was between 1.4 and 1.6, report researchers in the journal Science. The existence of vaccines and antiviral drugs made the 2009 outbreak much less deadly.

5.1.3 How is the R_0 of a disease calculated?

The following factors are taken into account to calculate the R_0 of a disease:

- *Infectious period*: Some diseases are contagious for longer periods than others. For example, according to the Centers for Disease Control and Prevention, adults with the flu are typically contagious for up to eight days, while children can be contagious for up to two weeks. The longer the infectious period of a disease, the more likely an infected person is to spread the disease to other people. A long period of infectiousness will contribute to a higher R_0 value.
- *Contact rate*: If a person who's infected with a contagious disease comes into contact with many people who aren't infected or vaccinated, the disease will spread more quickly. If that person remains at home, in a hospital, or otherwise quarantined while they're contagious, the disease will spread more slowly. A high contact rate will contribute to a higher R_0 value.
- *Mode of transmission*: The diseases that spread most quickly and easily are the ones that can travel through the air, such as the flu or measles. Physical contact with an infected person isn't necessary for the transmission of such conditions. You can catch the flu from breathing near someone who has the flu, even if you never touch them.

In contrast, diseases that are transmitted through bodily fluids, such as Ebola or HIV, aren't as easy to catch or spread. This is because you need to come into contact with infected blood, saliva, or other bodily fluids to contract them. Airborne illnesses tend to have a higher R_0 value than those spread through contact.

5.2 Simulated epidemic model

Following code conveyed by John Mallery, we have the following approach for estimating R_0 using a single realization of an epidemic simulation.

Note that there can be failures of `estimate.R` for certain inputs. We are working on that.

```
library(R0)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
# Generating an epidemic with given parameters
mGT <- generation.time("gamma", c(3,1.5))
set.seed(5432)  # always initialize when simulating!
mEpid <- sim.epid(epid.nb=1, GT=mGT, epid.length=30,
     family="poisson", R0=1.67, peak.value=500)
mEpid <- mEpid[,1]
# Running estimations
est <- estimate.R(epid=mEpid, GT=mGT, methods=c("EG","ML","TD"), begin=1, end=30)
```

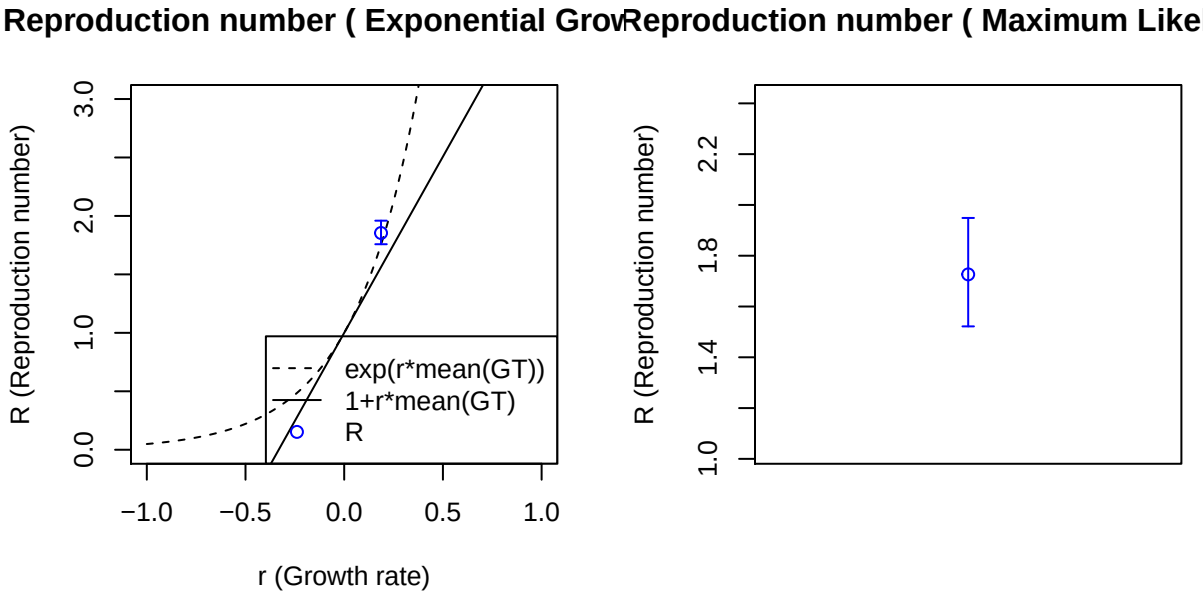
```
## Waiting for profiling to be done...
```

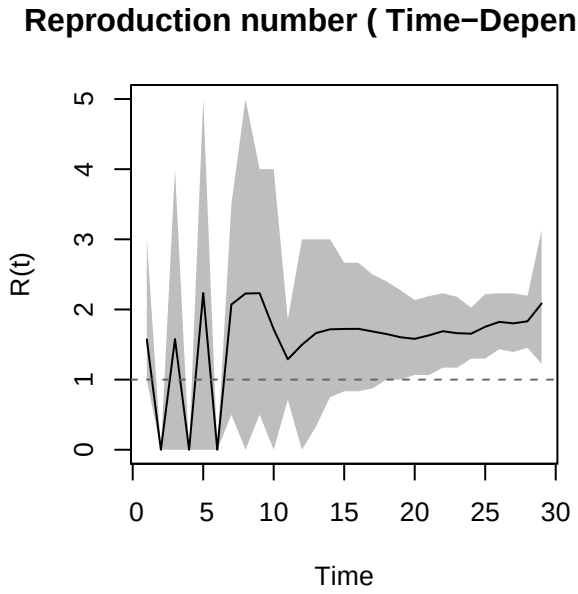
```
## Warning in est.R0.TD(epid = c(1, 0, 1, 0, 1, 0, 2, 1, 2, 1, 7, 2, 3, 4, :
## Simulations may take several minutes.
```

```
## Warning in est.R0.TD(epid = c(1, 0, 1, 0, 1, 0, 2, 1, 2, 1, 7, 2, 3, 4, : Using
## initial incidence as initial number of cases.
```

We modified the plotting function in *R0* which was calling `dev.new` too often. Use `plot2`.

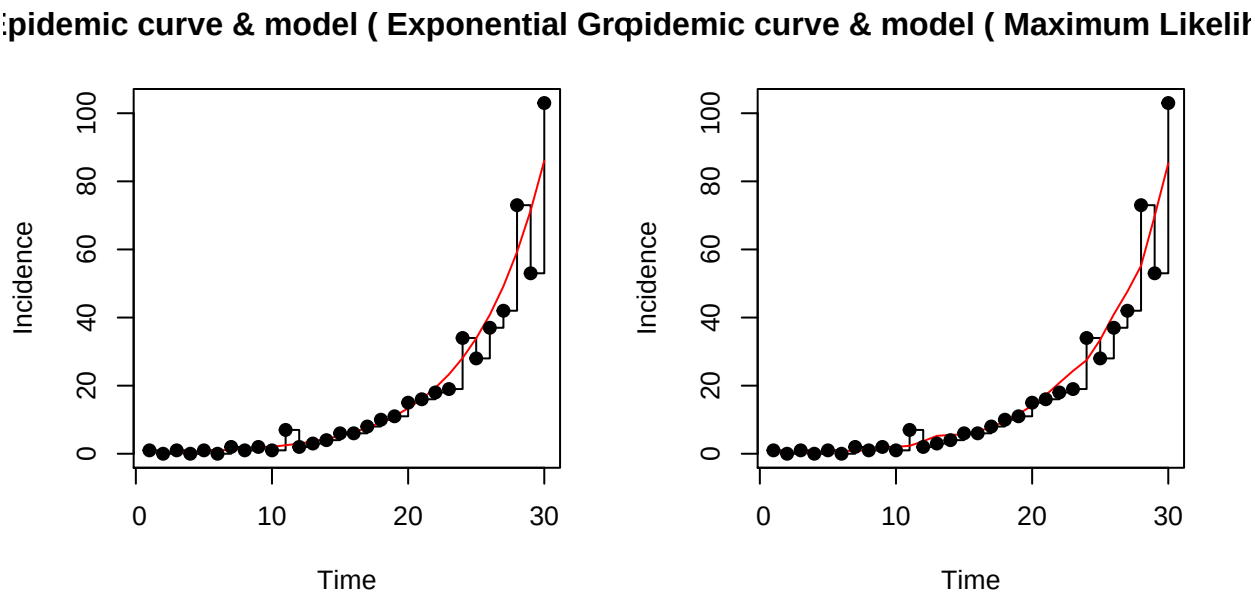
```
par(mfrow=c(2,2))
plot2(est)
```

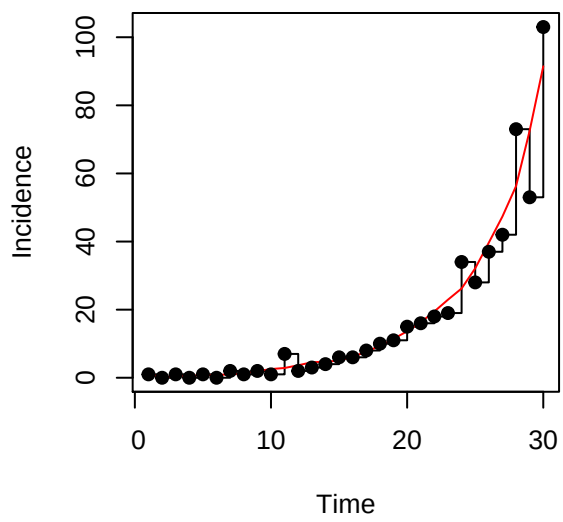




The plotfit2 function is also useful. These fits look identical but they are not.

```
par(mfrow=c(2,2))
plotfit2(est)
```





5.3 Real data examples

Now we extract information from the time-series table and obtain estimates of R_0 under exponential growth.

5.3.1 Hubei Province

We are able to use exponential growth and time-dependent models with this data, using generation time model from a recent Annals of Internal Medicine paper.

The incidence data probably need smoothing, and the time-dependent model has unreasonable fluctuations.

```
dates = lubridate::as_date(mdy(names(mar19df)[-c(1:4)]))
hubdat = as.numeric(get_series(province="Hubei", country="China",
```

```
    dataset=sars2pack::mar19df))
names(hubdat) = dates
mGT <- generation.time("gamma", c(5.8, 0.95)) # from DOI 10.7326/M20-0504
mGT <- generation.time("gamma", c(3.96, 4.75)) # from DOI 10.7326/M20-0504
hubdat.filt = trim_leading_values(c(hubdat[1], diff(hubdat)))
est.EG <- estimate.R(epid=hubdat.filt, GT=mGT,
    methods=c("EG", "TD"), begin=1L, end=as.integer(length(hubdat.filt)))
```

```
## Waiting for profiling to be done...
```

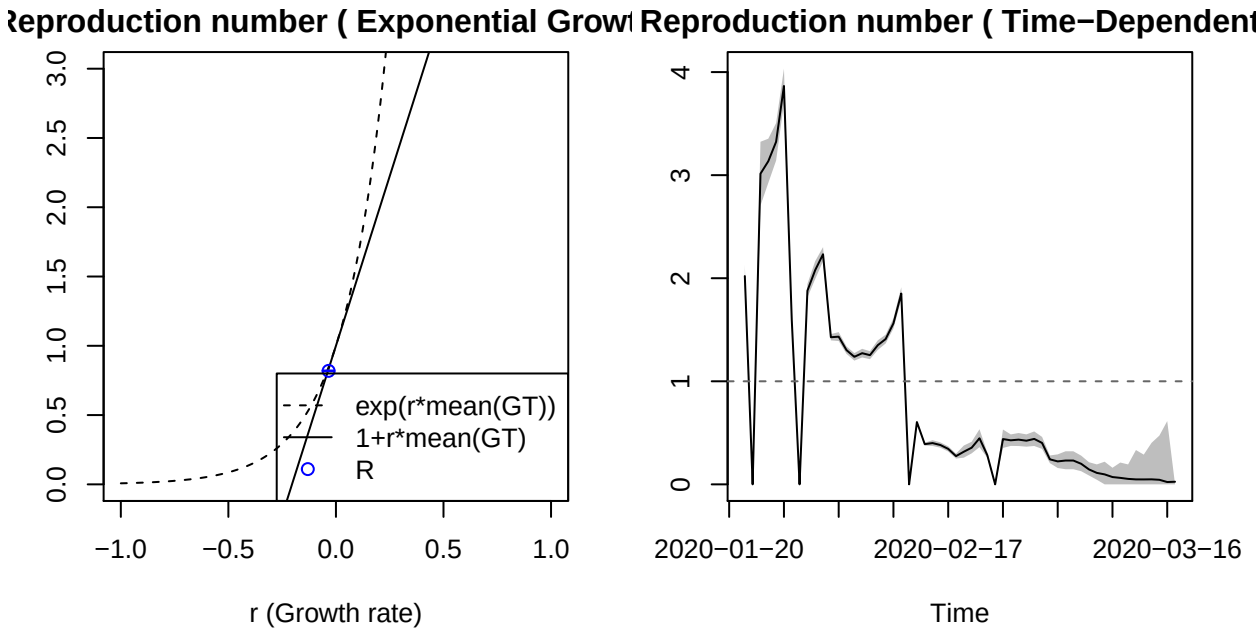
```
## Warning in est.R0.TD(epid = c(`2020-01-22` = 444, `2020-01-23` = 0, `2020-01-24`
## = 105, : Simulations may take several minutes.
```

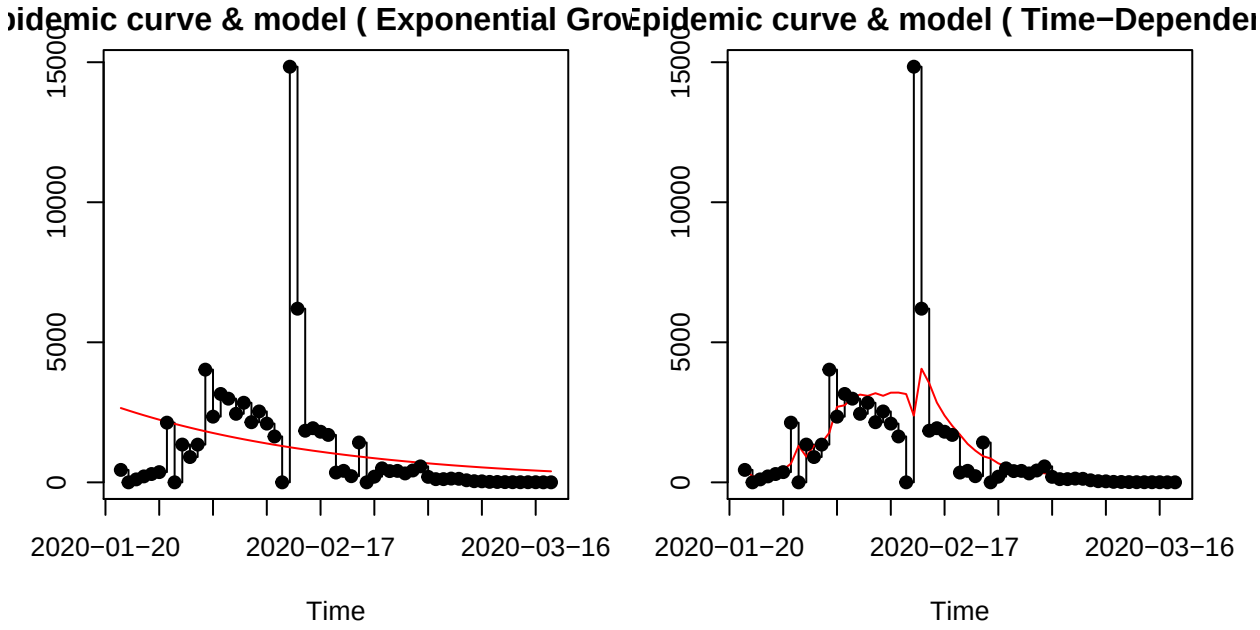
```
## Warning in est.R0.TD(epid = c(`2020-01-22` = 444, `2020-01-23` = 0, `2020-01-24`
## = 105, : Using initial incidence as initial number of cases.
```

```
est.EG
```

```
## Reproduction number estimate using  Exponential Growth  method.
## R :  0.8190473[ 0.8164334 , 0.821658 ]
##
## Reproduction number estimate using  Time-Dependent  method.
## 2.020789 0 3.0142 3.134995 3.32356 3.865543 1.596743 0 1.878637 2.079345 ...
```

```
par(mfrow=c(2,2), mar=c(5,3,2,2))
plot2(est.EG)
plotfit2(est.EG)
```





5.3.2 Italy

For Italy, only the EG model seems to work, with the Annals of Internal Medicine generation time model. It fits the data reasonably well, but the data seems to include a reporting gap.

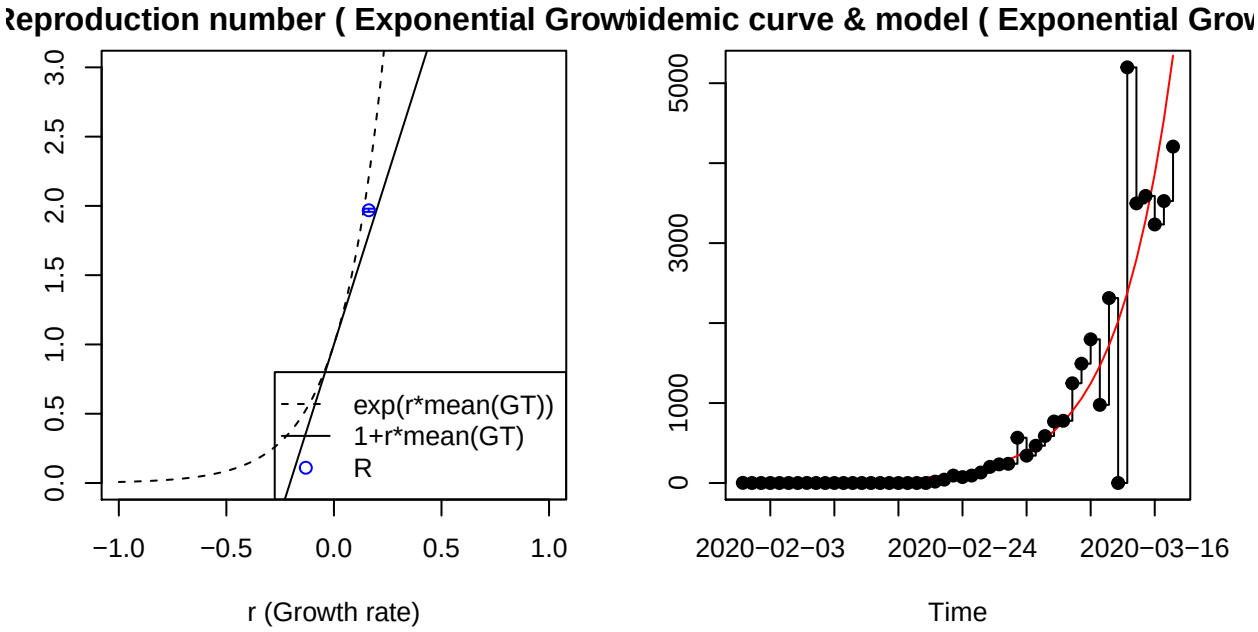
```
itdat = as.numeric(get_series(province="", country="Italy",
    dataset=sars2pack::mar19df))
names(itdat) = dates
itdat.filt = trim_leading_values(c(itdat[1], diff(itdat)))
est.EG <- estimate.R(epid=itdat.filt, GT=mGT,
    methods=c("EG"), begin=1L, end=as.integer(length(itdat.filt)))
```

```
## Waiting for profiling to be done...
```

```
est.EG
```

```
## Reproduction number estimate using  Exponential Growth  method.
## R :  1.968466[ 1.957161 , 1.979874 ]
```

```
par(mfrow=c(2,2), mar=c(5,3,2,2))
plot2(est.EG, main="Italy")
plotfit2(est.EG, main="Italy")
```



5.3.3 New York City

```
nyt = nytimes_county_data() %>%
    dplyr::filter(county=='New York City' & subset=='confirmed') %>%
    dplyr::arrange(date)
nytdat = nyt$count
# do we need to chop zeros off? Seems like not.
nytdat.filt = c(nytdat[1], diff(nytdat))
est <- estimate.R(epid=nytdat.filt, GT=mGT,
                  methods=c("EG","TD","ML"), begin=1L, end=as.integer(length(nytdat.filt)))
```

We can also use the package *EpiEstim* to perform time-dependent R_0 calculations.

```
library(EpiEstim)
```

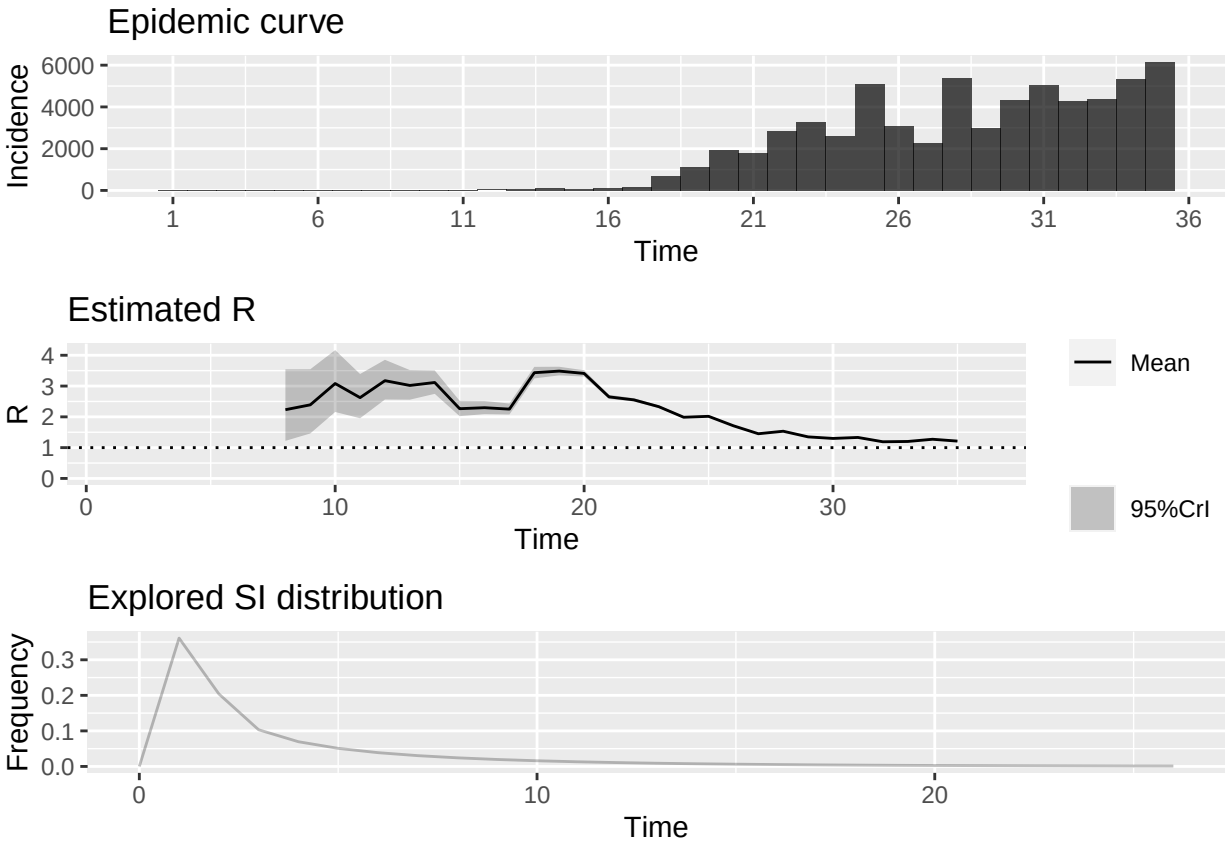
```
##
## Attaching package: 'EpiEstim'
```

```
## The following object is masked from 'package:sars2pack':
##
##     estimate_R
```

```
epiestim = EpiEstim::estimate_R(nytdat.filt, method = "parametric_si",
                                config = EpiEstim::make_config(list(
                                    mean_si = 3.96, std_si = 4.75)))
```

```
## Default config will estimate R on weekly sliding windows.
##     To change this change the t_start and t_end arguments.
```

```
plot(epiestim)
```




```
## TableGrob (3 x 1) "arrange": 3 grobs
##       z     cells    name           grob
## incid 1 (1-1,1-1) arrange gtable[layout]
## R     2 (2-2,1-1) arrange gtable[layout]
## SI    3 (3-3,1-1) arrange gtable[layout]
```

6 European datasets

```
library(knitr)
knitr::opts_chunk$set(message=FALSE)
```

```
library(sars2pack)
library(eurostat)
library(dplyr)
library(tmap)
library(sf)
```

```
eucov = eu_data_cache_data()
```

```
geodata = get_eurostat_geospatial(output_class = "sf",
                                  resolution = "60",
                                  nuts_level = 2,
```

```
                                    year=2016)
## The names here need to be initialized to NULL
## for the join below to working
## See https://github.com/r-spatial/sf/issues/1177#issuecomment-541858742
head(names(geodata$geometry))
```

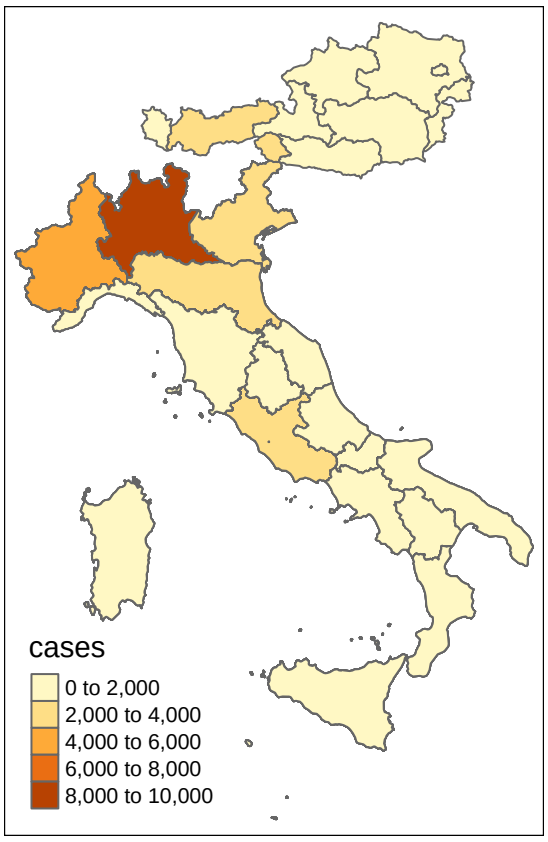
```
## [1] "0" "1" "2" "3" "4" "5"
```

```
names(geodata$geometry)=NULL
```

```
x = left_join(geodata, eucov, by=c('NUTS_NAME'='nuts_2'))
```

```
tmap::tm_shape(geodata) + tm_polygons()
```

```
x %>% filter(date=='2020-04-01') %>% tmap::tm_shape() + tm_polygons(col='cases')
```



7 Epicurves

```
library(sars2pack)
```

```
## Loading required package: R0
## Loading required package: MASS
## Loading required package: sf
## Linking to GEOS 3.7.2, GDAL 2.4.2, PROJ 5.2.0
```

```
library(DT)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following object is masked from 'package:MASS':
##
##     select
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

7.1 Worldwide datasets

```
jhu = jhu_data()
DT::datatable(jhu[sample(1:nrow(jhu),500),] %>%
              dplyr::arrange(CountryRegion, date),
              extensions = 'Scroller', options = list(scrollX=TRUE))
```

Show 10 entries Search:

	ProvinceState	CountryRegion	Lat	Long	date	count	subset
1		Afghanistan	33	65	2020-02-05	0	deaths
2		Afghanistan	33	65	2020-02-11	0	deaths
3		Afghanistan	33	65	2020-03-10	0	deaths
4		Afghanistan	33	65	2020-03-21	0	deaths
5		Afghanistan	33	65	2020-03-25	84	confirmed
6		Albania	41.1533	20.1683	2020-02-01	0	deaths
7		Albania	41.1533	20.1683	2020-03-23	4	deaths
8		Albania	41.1533	20.1683	2020-03-31	243	confirmed
9		Algeria	28.0339	1.6596	2020-02-04	0	deaths
10		Algeria	28.0339	1.6596	2020-03-20	90	confirmed

Showing 1 to 10 of 500 entries Previous 1 2 3 4 5 … 50 Next

7.2 European datasets

```
eucov = eu_data_cache_data()
DT::datatable(eucov[sample(1:nrow(eucov),500),] %>%
              dplyr::arrange(country, date),
              extensions = 'Scroller', options = list(scrollX=TRUE))
```

Show 10 entries Search:

	country	nuts_2	cases	recovered	deaths	hospitalized	intensive_care	datetime	date
1	AT	Wien	50					2020-03-11T15:00:00Z	2020-03-1
2	AT	Kärnten	3					2020-03-12T15:00:00Z	2020-03-1
3	AT	Burgenland	9	0	0			2020-03-14T08:00:00Z	2020-03-1
4	AT	Salzburg	39	0	0			2020-03-15T08:00:00Z	2020-03-1
5	AT	Niederösterreich	216	2	0			2020-03-17T15:00:00Z	2020-03-1
6	AT	Tirol	382	2	0			2020-03-18T15:00:00Z	2020-03-1
7	AT	Oberösterreich	707	0	0			2020-03-23T15:00:00Z	2020-03-2
8	AT	Niederösterreich	512	2	0			2020-03-23T08:00:00Z	2020-03-2
9	AT	Vorarlberg	354	0	0			2020-03-24T08:00:00Z	2020-03-2
10	AT	Salzburg	493	0	0			2020-03-25T08:00:00Z	2020-03-2

Showing 1 to 10 of 500 entries Previous 1 2 3 4 5 … 50 Next

7.3 United States datasets

7.3.1 USAFacts

```
usa_facts = usa_facts_data()
DT::datatable(usa_facts[sample(1:nrow(usa_facts),500),] %>%
              dplyr::arrange(state, county, date, subset),
              extensions = 'Scroller', options = list(scrollX=TRUE))
```

Show 10 entries Search:

	county_fips	county	state	state_fips	subset	date	count
1	2050	Bethel Census Area	AK	2	deaths	2020-02-28	0
2	2100	Haines Borough	AK	2	confirmed	2020-01-26	0
3	2164	Lake and Peninsula Borough	AK	2	confirmed	2020-02-01	0
4	1013	Butler County	AL	1	confirmed	2020-01-31	0
5	1045	Dale County	AL	1	deaths	2020-02-29	0
6	1047	Dallas County	AL	1	deaths	2020-02-03	0
7	1051	Elmore County	AL	1	confirmed	2020-03-11	0
8	1051	Elmore County	AL	1	confirmed	2020-03-22	6
9	1087	Macon County	AL	1	deaths	2020-02-28	0
10	1101	Montgomery County	AL	1	deaths	2020-03-08	0

Showing 1 to 10 of 500 entries Previous 1 2 3 4 5 … 50 Next

7.3.2 New York Times

```
nytimes = nytimes_county_data()
DT::datatable(nytimes[sample(1:nrow(nytimes),500),] %>%
              dplyr::arrange(state, county, date, subset),
              extensions = 'Scroller', options = list(scrollX=TRUE))
```

Show 10 entries Search:

	date	county	state	fips	count	subset
1	2020-04-04	Chilton	Alabama	01021	0	deaths
2	2020-04-01	DeKalb	Alabama	01049	0	deaths
3	2020-03-21	Jefferson	Alabama	01073	61	confirmed
4	2020-03-26	Marshall	Alabama	01095	0	deaths
5	2020-04-01	Shelby	Alabama	01117	89	confirmed
6	2020-03-19	St. Clair	Alabama	01115	1	confirmed
7	2020-03-24	Talladega	Alabama	01121	2	confirmed
8	2020-03-27	Talladega	Alabama	01121	0	deaths
9	2020-03-30	Walker	Alabama	01127	0	deaths
10	2020-03-29	Anchorage	Alaska	02020	59	confirmed

Showing 1 to 10 of 500 entries Previous 1 2 3 4 5 … 50 Next

8 Map visualizations

```
library(tmap)
library(dplyr)
library(sars2pack)
library(htmltools)
library(htmlwidgets)
```

```
ejhu = enriched_jhu_data()
```

```
glimpse(ejhu)
```

```
## Rows: 38,332
## Columns: 20
## $ name           <chr> "Afghanistan", "Afghanistan", "Afghanistan", "Afghan...
## $ topLevelDomain <list> [".af", ".af", ".af", ".af", ".af", ".af", ".af", "...
## $ alpha2Code     <chr> "AF", "AF", "AF", "AF", "AF", "AF", "AF", "AF", "AF"...
## $ alpha3Code     <chr> "AFG", "AFG", "AFG", "AFG", "AFG", "AFG", "AFG", "AF...
## $ capital        <chr> "Kabul", "Kabul", "Kabul", "Kabul", "Kabul", "Kabul"...
## $ region         <chr> "Asia", "Asia", "Asia", "Asia", "Asia", "Asia", "Asi...
## $ subregion      <chr> "Southern Asia", "Southern Asia", "Southern Asia", "...
## $ population     <int> 27657145, 27657145, 27657145, 27657145, 27657145, 27...
## $ area           <dbl> 652230, 652230, 652230, 652230, 652230, 652230, 6522...
## $ gini           <dbl> 27.8, 27.8, 27.8, 27.8, 27.8, 27.8, 27.8, 27.8, 27.8...
## $ borders        <list> [<"IRN", "PAK", "TKM", "UZB", "TJK", "CHN">, <"IRN"...
## $ numericCode    <chr> "004", "004", "004", "004", "004", "004", "004", "00...
## $ cioc           <chr> "AFG", "AFG", "AFG", "AFG", "AFG", "AFG", "AFG", "AF...
```

```
## $ ProvinceState  <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ CountryRegion  <chr> "Afghanistan", "Afghanistan", "Afghanistan", "Afghan...
## $ Lat            <dbl> 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, ...
## $ Long           <dbl> 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, ...
## $ date           <date> 2020-01-22, 2020-01-23, 2020-01-24, 2020-01-25, 202...
## $ count          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ subset         <chr> "confirmed", "confirmed", "confirmed", "confirmed", ...
```

We need a description of the regions of the world.

```
data(World)
```

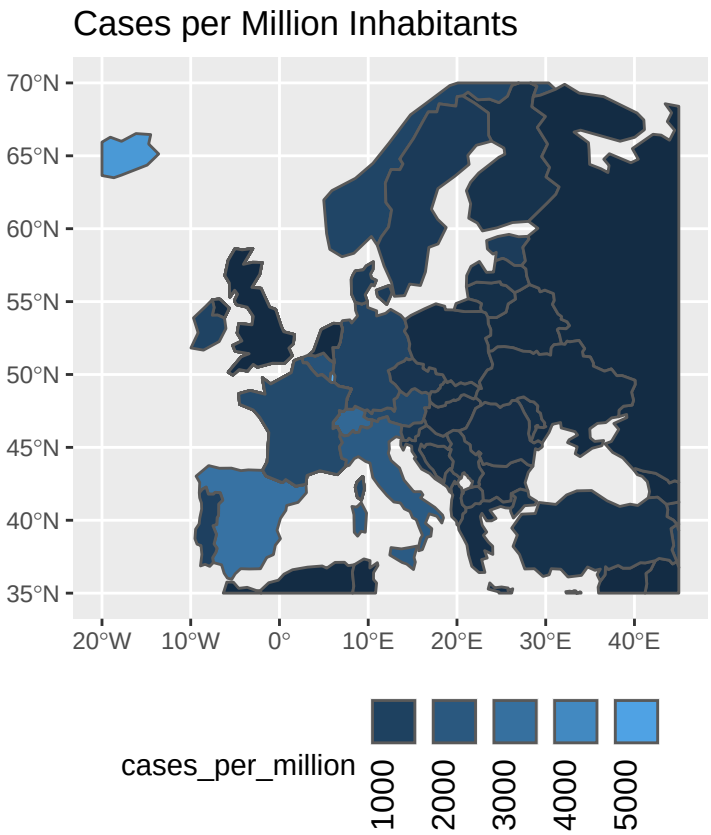
The `World` object has a column, `geometry`, that describes the shape of each country in the `World` dataset. Join the `ejhu` data.frame with the `World` data using `dplyr` join as normal.

```
geo_ejhu = World %>%
    dplyr::left_join(ejhu, by = c('iso_a3' = 'alpha3Code'))
```

```
w2 = geo_ejhu %>%
    filter(!is.na(date) & subset=='confirmed') %>%
    group_by(iso_a3) %>%
    filter(date==max(date)) %>%
    mutate(cases_per_million = 1000000*count/pop_est)
```

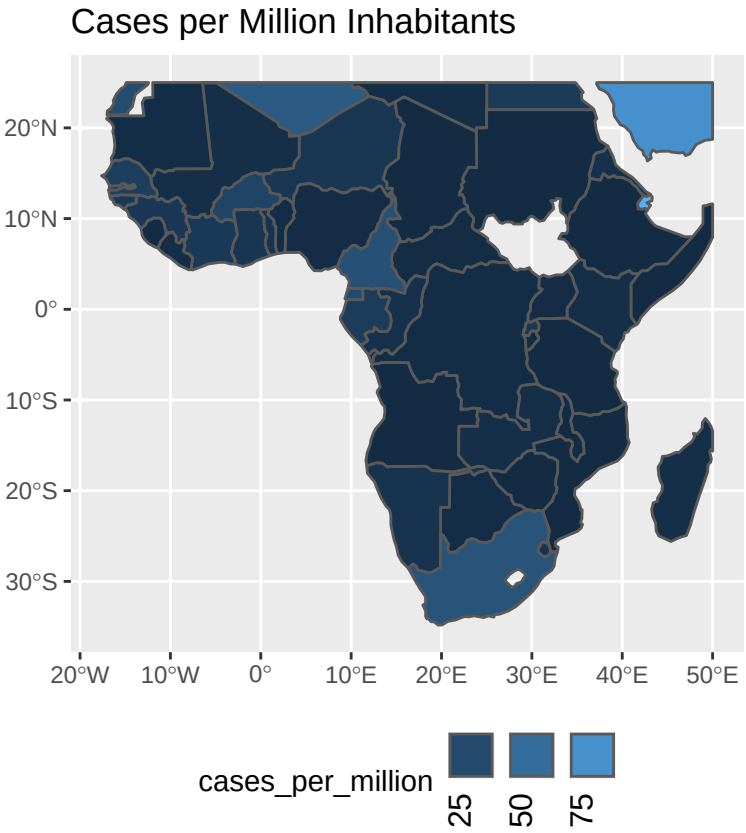
The R package `ggplot2` has geospatial plotting capabilities built in for geospatial `simple features (sf)` data types. In this first plot, we focus in on Europe.

```
library(ggplot2)
# transform to lat/long coordinates
st_transform(w2, crs=4326) %>%
# Crop to europe (rough, by hand)
    st_crop(xmin=-20,xmax=45,ymin=35,ymax=70) %>%
ggplot() +
    geom_sf(aes(fill=cases_per_million)) +
    scale_fill_continuous(
        guide=guide_legend(label.theme = element_text(angle = 90),
                           label.position='bottom')
        ) +
    labs(title='Cases per Million Inhabitants') +
    theme(legend.position='bottom')
```



Another plot, but now for Africa.

```
library(ggplot2)
# transform to lat/long coordinates
st_transform(w2, crs=4326) %>%
# Crop to europe (rough, by hand)
    st_crop(xmin=-20,xmax=50,ymin=-60,ymax=25) %>%
ggplot() +
    geom_sf(aes(fill=cases_per_million)) +
    scale_fill_continuous(
        guide=guide_legend(label.theme = element_text(angle = 90),
                           label.position='bottom')
        ) +
    labs(title='Cases per Million Inhabitants') +
    theme(legend.position='bottom')
```



8.1 Interactive maps

The following will not produce a plot when run non-interactively. However, pasting this into your R session will result in an interactive plot with multiple "layers" that you can choose to visualize different quantitative variables on the map. Zooming also works as expected.

```
tmap_mode('view')
## geo_ejhu %>%
##     filter(!is.na(date) & subset=='confirmed') %>%
##     group_by(iso_a3) %>%
##     filter(date==max(date)) %>%
##     tm_shape() +
##         tm_polygons(col='count')
w2 = geo_ejhu %>%
    filter(!is.na(date) & subset=='confirmed') %>%
    group_by(iso_a3) %>%
    filter(date==max(date)) %>%
    mutate(cases_per_million = 1000000*count/pop_est) %>%
    filter(region == 'Africa')
m = tm_shape(w2,id='name.x', name=c('cases_per_million'),popup=c('pop_est')) +
    tm_polygons(c('Cases Per Million' = 'cases_per_million','Cases' = 'count',"Well-being index"='well_
                selected='cases_per_million',
                border.alpha = 0.5,
                alpha=0.6,
                popup.vars=c('Cases Per Million'='cases_per_million',
                             'Confirmed Cases'  ='count',
                             'Population'       ='pop_est',
```

```
                            'gini'              ='gini',
                            'Life Expectancy'   ='life_exp')) +
    tm_facets(as.layers = TRUE)
tmap_save(m, filename='abc.html')
```

8.2 United States

```
library(ggplot2)
library(tigris)
library(tidycensus)
library(plotly)
library(sf)
```

```
county_geom = tidycensus::county_laea
nyt_counties = nytimes_county_data()
full_map = county_geom %>%
    left_join(
        nyt_counties %>%
        group_by(fips) %>%
        filter(date==max(date) & count>0 & subset=='confirmed'), by=c('GEOID'='fips')) %>%
    mutate(mid=sf::st_centroid(geometry))
z = ggplot(full_map, aes(label=county)) +
    geom_sf(aes(geometry=geometry),color='grey85') +
    geom_sf(aes(geometry=mid, size=count, color=count), alpha=0.5, show.legend = "point") +
    scale_color_gradient2(midpoint=5500, low="lightblue", mid="orange",high="red", space ="Lab" ) +
    scale_size(range=c(1,10))
ggplotly(z)
```

A static plot as a png:

```
z
```

Alternatively, produce a PDF of the same plot.

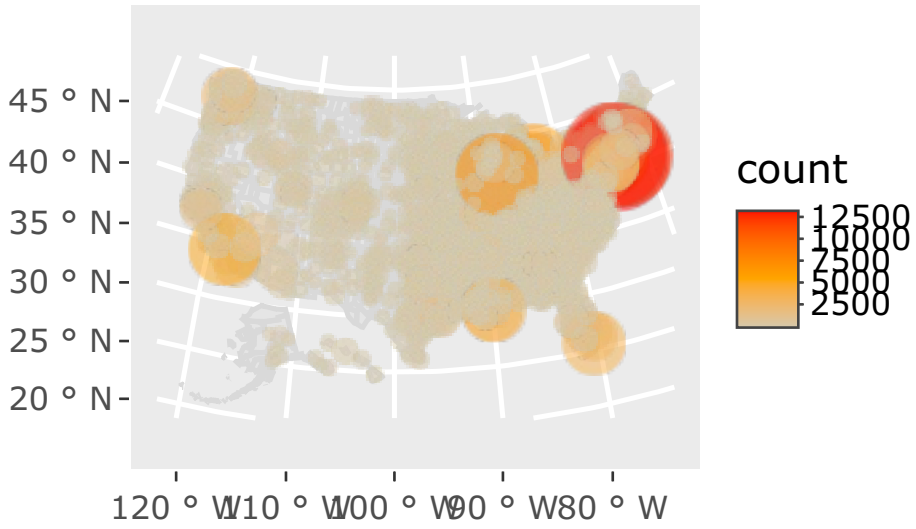


Figure 1: United States confirmed cases by County with interactive plotly library. Click and drag to zoom in to a region of interest.

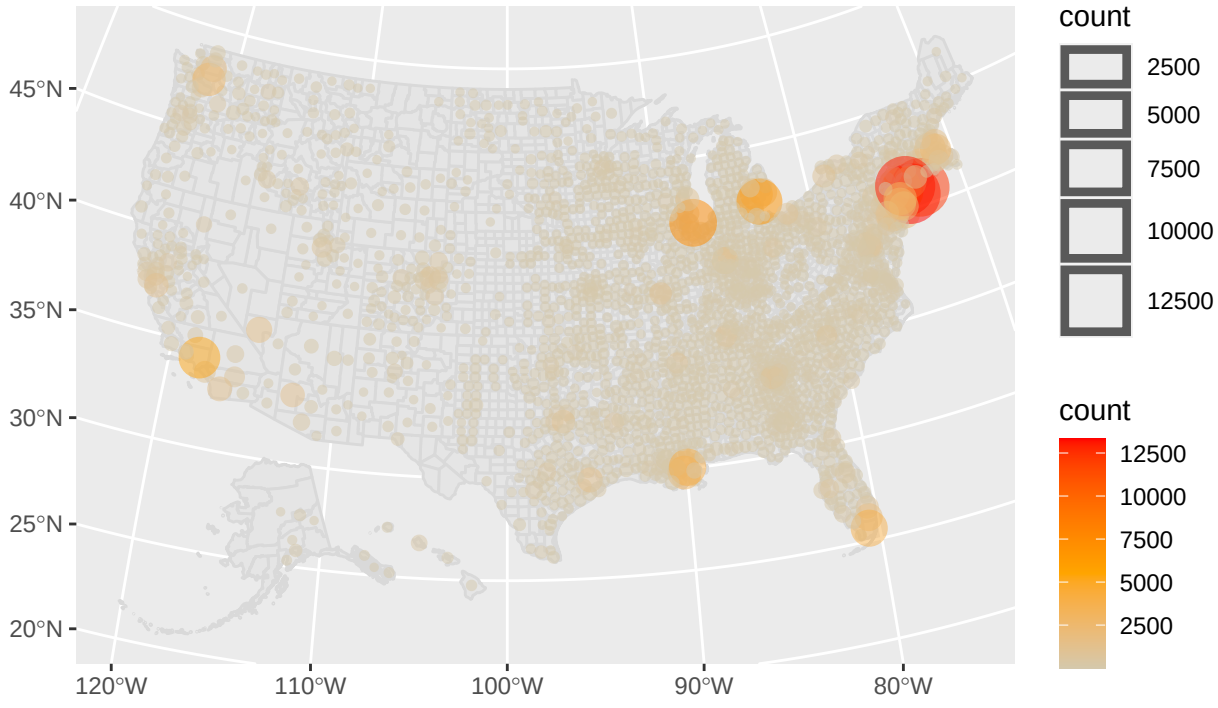


Figure 2: United States confirmed cases by County as a static graphic.

```
pdf('us_county_numbers.pdf', width=11, height=8)
print(z)
dev.off()
```

```
## pdf
##   2
```

Origins of this package

John C. Mallery conveyed code of Charles Morefield to harvest COVID-19 time series data. Vince Carey then started an R package/github repo to manage the relevant code. The package was named `sars2pack` in hopes of avoiding name conflict with many other packages while remaining descriptive and focused.

Summary

9 Data resource details

9.1 Epidemic datasets

- https://github.com/beoutbreakprepared/nCoV2019

10

10.1 World Bank

10.2 United States Census

10.3 Eurostat

10.4 Other resources

- https://cengel.github.io/gearup2016/SULdataAccess.html g