Assignment4

Richard McCormick

February 20th, 2022

In this assignment we will learn how to estimate the instantaneous reproduction number, R_t, from incidence data. Luckily, there is an R package called *EpiEstim* that has functions to do this estimation for us. However, to learn the concept more deeply, we will do a thorough analysis to test the ability of the R package to accurately estimate R_t. Note: we will also use the *ggplot* package for plotting in this assignment. *ggplot* is part of the "tidyverse" family of packages, and we'll need a few of these packages for this assignment.

There is a common work-flow in statistics and data science to make sure that an analysis is working as one expects, with no bugs or issues. To summarize the process:

1. Use fixed (i.e., "known") parameter values to simulate a data set from the "generative" model (i.e., the model of interest, for which you want to estimate parameters).
2. Use the statistical analysis to estimate the parameters using the synthetic data.
3. Validate that the statistical analysis was able to accurately and precisely estimate the "known" parameters that were used to generate the synthetic data in the first place.

This routine shows that we the statistical analysis can accurately estimate model parameters from data in an ideal situation (i.e., using a synthetic, unbiased data set). We then have high confidence that the statistical analysis will work to estimate parameters from real data sets, for which we don't know the true parameter values.

In our case, we want to test the *EpiEstim* package to validate that it can accurately and precisely estimate R_t from incidence data. Therefore we will complete the following procedure:

1. We know that $R_t = R_0 S_{(t)} = \frac{\beta}{\gamma} S_{(t)}$ for the SIR model without demography. First, we will generate a hypothetical pattern of R_t over time, one that we might see in real life, but in this case we know the exact values of R_t. Then, we will use the SIR model to generate a synthetic data set of incidence, by allowing R_t to change each day in the model based on our simulated pattern of R_t over time.
2. We will then use the *EpiEstim* package to estimate R_t per day from the synthetic incidence data set.
3. We will visually validate that the *EpiEstim* package accurately estimated our known pattern of R_t over time.

This is a fairly complicated simulation, and I will therefore provide almost all of the code. Your primary task is to learn and understand the code. Your final task will require you to estimate R_t using the *EpiEstim* package from a real data set of your choosing.

```r
# Install any of these packages that you don't have already
library(tidyverse)
library(deSolve)
library(lubridate)
library(EpiEstim)

options(dplyr.width = Inf, dplyr.print_max = 1e9)
```

Task 1

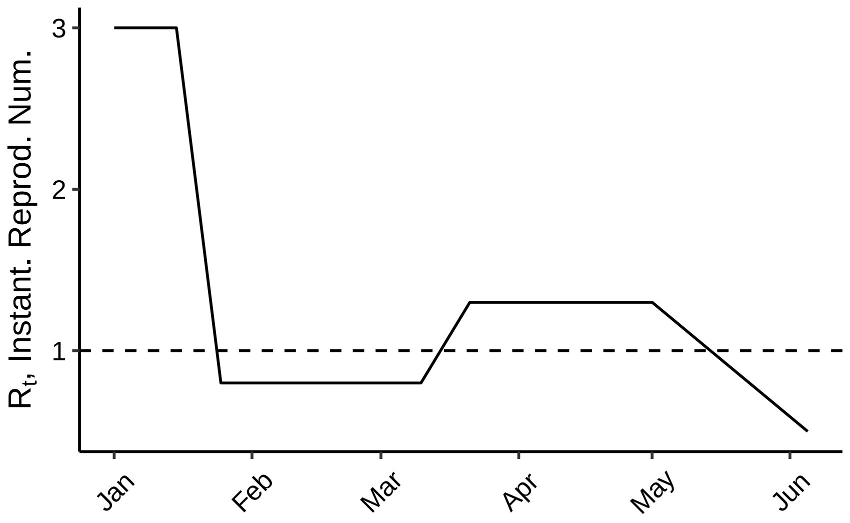
Your primary task is to understand the code that I provide to simulate the incidence data set from a "known" pattern of R_t changing over time, which we will simulate below.

Task 1.1 (20 points)

Annotate all of the code that I provide for you in this assignment (i.e., line by line). Your annotations should be detailed enough for me to know that you understand the code completely, but you don't have to go overboard here. If you don't know what a function is doing, use the *R* help manual. I recommend reading all of the code before trying to figure out what each line means separately. *Make sure that when you render this file to PDF, your code annotations are all visible and do not run over the page width.*

First, we will create a simulated pattern of R_t over time. These are our "known" values of R_t, which we will later try to estimate using the *EpiEstim* package.

```r
# Create the incidence time windows (six total)
n_windows = 6

# Create array of start dates
start_dates = c(mdy("1-1-20"),
                mdy("1-16-20"),
                mdy("1-26-20"),
                mdy("3-11-20"),
                mdy("3-22-20"),
                mdy("5-2-20"))

# Create array of end dates
end_dates =   c(mdy("1-15-20"),
                mdy("1-25-20"),
                mdy("3-10-20"),
                mdy("3-21-20"),
                mdy("5-1-20"),
                mdy("6-5-20"))

# Create array of R naught values, 1 for each window
changing_Rt = c(3.0, 0.8, 0.8, 1.3, 1.3, 0.5)

# Create empty array to store R values
Rt_seq = NULL

# Create array to store R values for each date window
Rt_seq[1:(end_dates[1] - start_dates[1] + 1)] = changing_Rt[1]

# Loop through the windows
for(i in 2:n_windows){

  # Create empty variable and arrays for storage
  Rt_temp_seq = NULL
  Rt_temp = NULL

  # If the current R value is different from previous R value
  if(changing_Rt[i] != changing_Rt[i-1]){

    # Find the difference between the two values
    Rt_diff = changing_Rt[i-1] - changing_Rt[i]
    # Advance the current day by one
    n_days = as.numeric(end_dates[i] - start_dates[i] + 1)
    # Find the slope as a quotient of the new R value and current day
    Rt_slope = - Rt_diff / n_days

    # Loop through each day
    for(j in 1:n_days){
      # Fill previously created array with each day's previous R value,
      # plus change for the current day.
      Rt_temp_seq[j] = changing_Rt[i-1] + Rt_slope*j
    }

  }else{
    # If the current R value is the same as the previous day's,
    # fill array value with current day's R value.
    n_days = as.numeric(end_dates[i] - start_dates[i] + 1)
    Rt_temp_seq = rep(changing_Rt[i], times = n_days)
  }

  # Combine the temp seq with the existing seq
  Rt_seq = c(Rt_seq, Rt_temp_seq)

}

# Create sequence of dates within interval, 1 day each apart
date_seq = seq.Date(start_dates[1], end_dates[n_windows], by = "1 day")
# New dataframe, one column is Rt values, one column is dates
Rt_seq_df = data.frame(Rt_seq, date_seq)
# Create sequence of 1 month intervals in the given dates
date_breaks = seq(range(date_seq)[1],
                  range(date_seq)[2],
                  by = "1 month")

# Graph the estimated Rt value over the given dates
ggplot(Rt_seq_df) +
  geom_path(aes(x = date_seq, y = Rt_seq)) +
  scale_x_date(breaks = date_breaks, date_labels = "%b") +
  labs(x="", y=expression(R[t]*", Instant. Reprod. Num.")) +
  geom_hline(yintercept = 1, linetype = 2) +
  theme_classic()+
  theme(
    axis.text = element_text(size = 10, color = "black"),
    axis.title = element_text(size = 12, color = "black"),
    axis.text.x = element_text(angle = 45, vjust = 0.5)
  )
```



R_t, Instant. Reprod. Num.
3
2
1
Jan
Feb
Mar
Apr
May
Jun

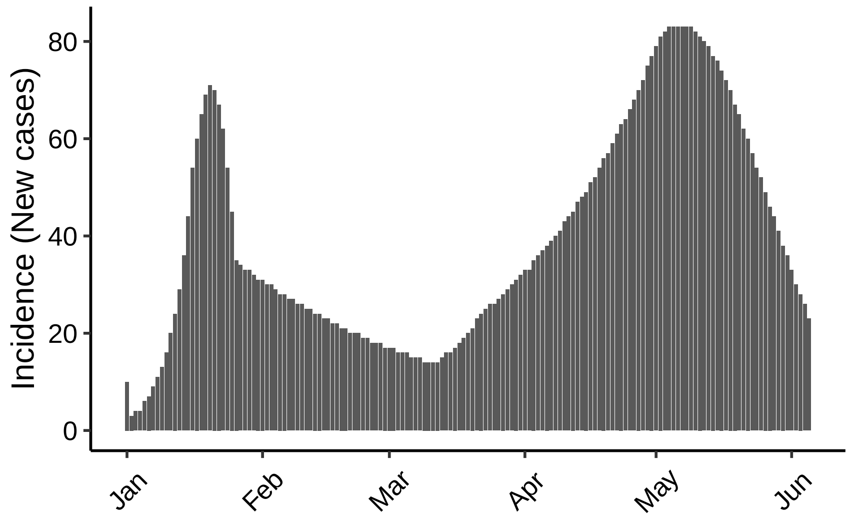
We will now build some useful functions for our data simulation:

```
#########################
# FUNCTIONS
#########################

# Standard SIR ODE function without reproduction and death
# Function 1
SIR_ode = function(t, y, params){
    with(as.list(c(y, params)), {

        dydt = rep(0, 3)

        dydt[1] = -beta * y[1] * y[2]
        dydt[2] = beta * y[1] * y[2] - gamma * y[2]
        dydt[3] = gamma * y[2]

        return(list(dydt))

    })
}

# Function 2
# One step function to solve given ODE, specifically SIR. Pump out dataframe
solve_ODE_1step = function(inits, params){

    out_temp = ode(y = inits,
                   times = time_ode,
                   func = SIR_ode,
                   method="ode45",
                   parms = params)

    colnames(out_temp) = c("time", "S", "I", "R")
    out_temp = data.frame(out_temp)

    return(out_temp)

}

# Function 3
# Find a beta value at time T given susceptible, Rt, and gamma variables
beta_calc = function(S_val, Rt_val, gamma){

    beta_t = Rt_val * gamma / S_val

    return(beta_t)

}
```

Next we will simulate the incidence data, using the above pattern of R_t and our helper functions. Essentially, we are solving the SIR model one day at a time, and each day, the value of the transmission rate, β_t, changes, depending on the value of R_t.

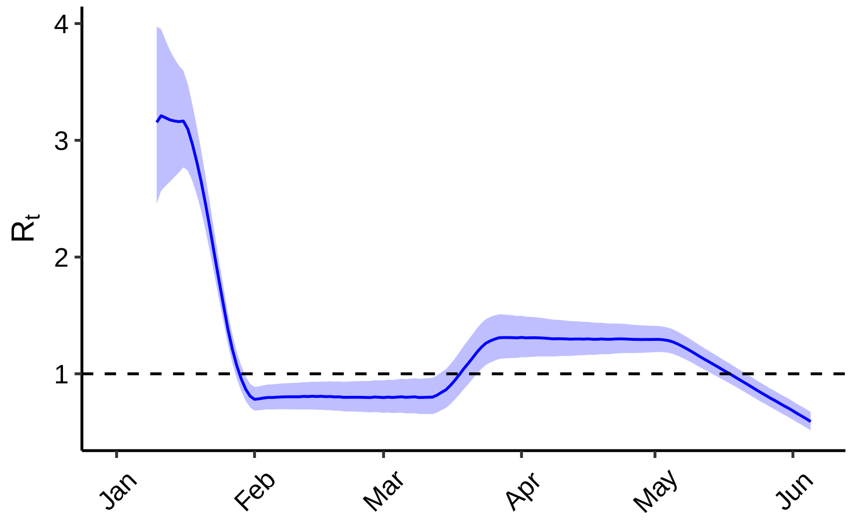
```
# Size of hypothetical host population
n_pop = 10000
```

```
#Number of days is equal to number of Rt vals
n_day = length(Rt_seq)
t_min = 0
t_max = n_day

# Create a vector of size equal to number of days
incidence_obs = vector(mode="numeric", length = n_day)

# Establish variables for viral reproduction
avg_infect_period = 10.0
gamma = 1 / avg_infect_period

# Create empty storage with intervals of size 1
time_ode = seq(0, 1, by = 1)

for(i in t_min:(t_max-1)){

  #if at first loop
  if(i == t_min){

    #establish S, I, R variables
    I0 = 0.001
    S0 = 1 - I0
    R0 = 0
    #fill storage dataframe
    incidence_obs[i+1] = floor(n_pop*I0)

  }else{
    #calculate fraction of the population infected per day by taking initial
    #susceptible population and subtracting current number of susceptible
    frac_inf_per_day = S0 - df_temp$S[nrow(df_temp)]
    #find number of people currently infected by multiplying population size
    #by the fraction infected
    incidence_obs[i+1] = floor(n_pop*frac_inf_per_day)

    #set S, I, R vars to most current values
    S0 = df_temp$S[nrow(df_temp)]
    I0 = df_temp$I[nrow(df_temp)]
    R0 = df_temp$R[nrow(df_temp)]
  }
  #set current S, I, R vars into init vector
  inits_temp = c(S0, I0, R0)
  #set time equal to current day
  t_temp = i

  #solve for beta using most recent values
  beta_temp = beta_calc(S_val = S0,
                        Rt_val = Rt_seq[i+1],
                        gamma = gamma)

  #create params and store beta, gamma (most recent)
  params_temp = c(beta = beta_temp,
                  gamma = gamma)
  #update df_temp by solving SIR ODE with most recent variables
  df_temp = solve_ODE_1step(inits = inits_temp,
                            params = params_temp)

  #increments time by 1 and updates temp dataframe
  time_ode_fix = seq(t_temp, t_temp+1, by = 1)
  df_temp$time = time_ode_fix

  #if on day 1, set out equal to newly solved dataframe
  if(i == t_min){
    out = df_temp
  }else{
    # if not on day 1, merge new data with existing data
    out = rbind(out, df_temp)
  }

}

# store dates and new incidents in dataframe
incidence_df = data.frame(incidence_obs, date_seq)

#graph dataframe
ggplot(incidence_df) +
  geom_col(aes(x = date_seq, y = incidence_obs)) +
  scale_x_date(breaks = date_breaks, date_labels = "%b") +
  labs(x="", y="Incidence (New cases)") +
  theme_classic()+
  theme(
    axis.text = element_text(size = 10, color = "black"),
    axis.title = element_text(size = 12, color = "black"),
    axis.text.x = element_text(angle = 45, vjust = 0.5)
  )
```



Finally, we will use the *EpiEstim* package to estimate the R_t per day from these incidence data that we just simulated using the SIR model.

```r
# Note: EpiEstim requires a specific format for the data frame.
Rt_cases_df =
  incidence_df %>%
  rename(I = incidence_obs, dates = date_seq)

# Below configures the specifications for the R_t estimation
# We will estimate R_t across 7 day moving windows
# mean_si is the mean of the serial interval distribution
# std_si is the standard deviation of the serial interval ditribution

#establish variables
window_length = 7
n_add = window_length - 1
n_dates = nrow(Rt_cases_df)
t_start = seq(2, n_dates - n_add)
t_end = t_start + n_add

#establish list of parameters for use later on
config_list =
  EpiEstim::make_config(
    list(
      t_start = t_start,
      t_end = t_end,
      mean_si = avg_infect_period,
      std_si = avg_infect_period)
  )

#solve for weekly value of Rt. feed in previously established params
weekly_Rt =
  EpiEstim::estimate_R(
    Rt_cases_df,
    method="parametric_si",
    config = config_list
  )

#pull Rt value out of solved dataframe, assign to new dataframe
Rt_estim_df = weekly_Rt$R
#establish a buffer at the beginning of the dates in new dataframe
Rt_estim_df$dates = Rt_cases_df$dates[(n_add + 2):n_dates]

#set start and end point for dates
min_date = min(Rt_cases_df$dates)
max_date = max(Rt_cases_df$dates)

#Graph estimated Rt dataframe
ggplot(Rt_estim_df) +
  geom_path(aes(x = dates, y = `Median(R)` ),
            color = "blue") +
  geom_ribbon(aes(x = dates,
                  ymin = `Quantile.0.025(R)`,
                  ymax = `Quantile.0.975(R)`),
              fill = "blue", alpha = 0.25) +
  geom_hline(yintercept = 1, linetype = 2) +
  labs(x = "", y = expression(R[t])) +
  scale_x_date(limits = c(min_date, max_date)) +
  theme_classic() +
  theme(
    axis.text = element_text(size = 10, color = "black"),
    axis.title = element_text(size = 12, color = "black"),
    axis.text.x = element_text(angle = 45, vjust = 0.5)
  )
```

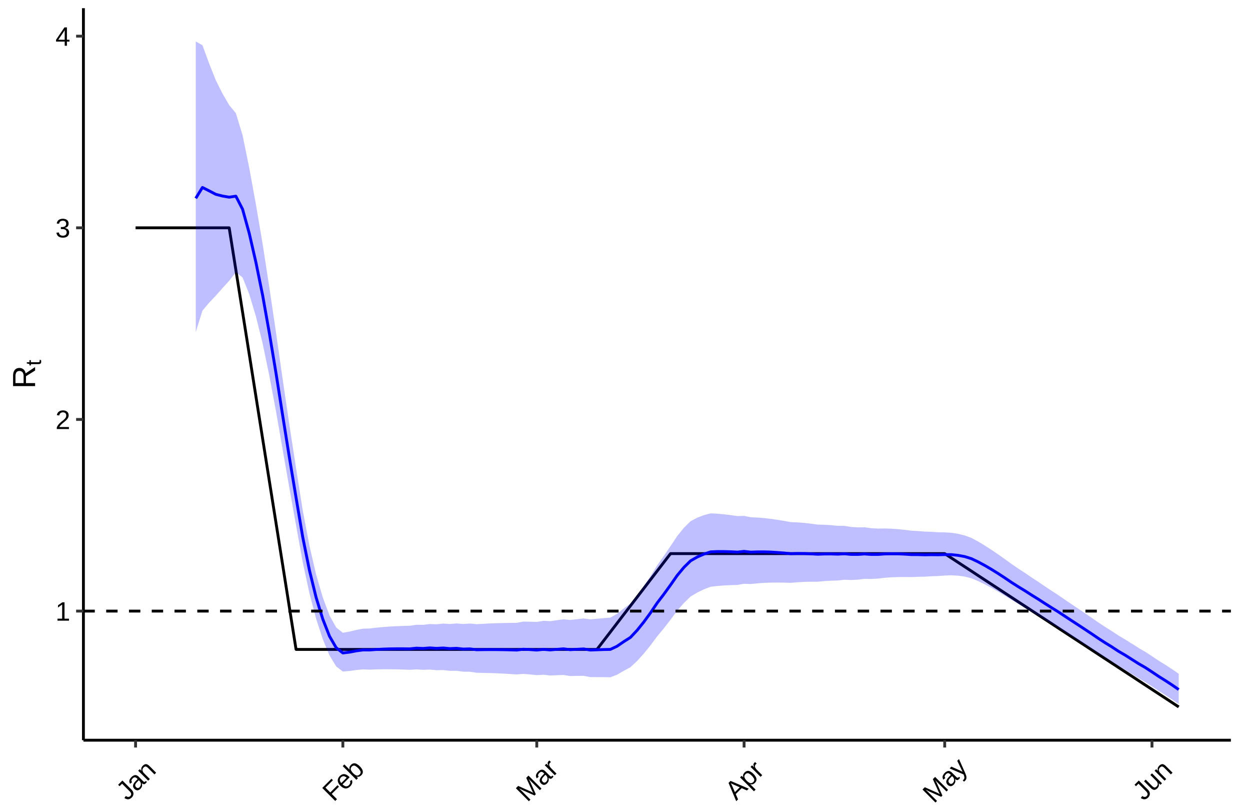


Task 1.2 (10 points)

Using *ggplot*, visually demonstrate that the *EpiEstim* package accurately estimated our known pattern of R_t. In other words, on the same plot, show the estimated R_t and the "known" values of R_t that we originally simulated.

```
ggp <- ggplot() +
  geom_path(data = Rt_seq_df, aes(x = date_seq, y = Rt_seq)) +
  geom_path(data = Rt_estim_df, aes(x = dates, y = `Median(R)` ),
            color = "blue") +
  geom_ribbon(data = Rt_estim_df, aes(x = dates,
                  ymin = `Quantile.0.025(R)`,
                  ymax = `Quantile.0.975(R)`),
              fill = "blue", alpha = 0.25) +
  geom_hline(yintercept = 1, linetype = 2) +
  labs(x = "", y = expression(R[t])) +
  scale_x_date(limits = c(min_date, max_date)) +
  theme_classic() +
  theme(
    axis.text = element_text(size = 10, color = "black"),
    axis.title = element_text(size = 12, color = "black"),
    axis.text.x = element_text(angle = 45, vjust = 0.5)
  )

ggp
```

```
## Warning: Removed 2 row(s) containing missing values (geom_path).
```



Task 2 (30 points)

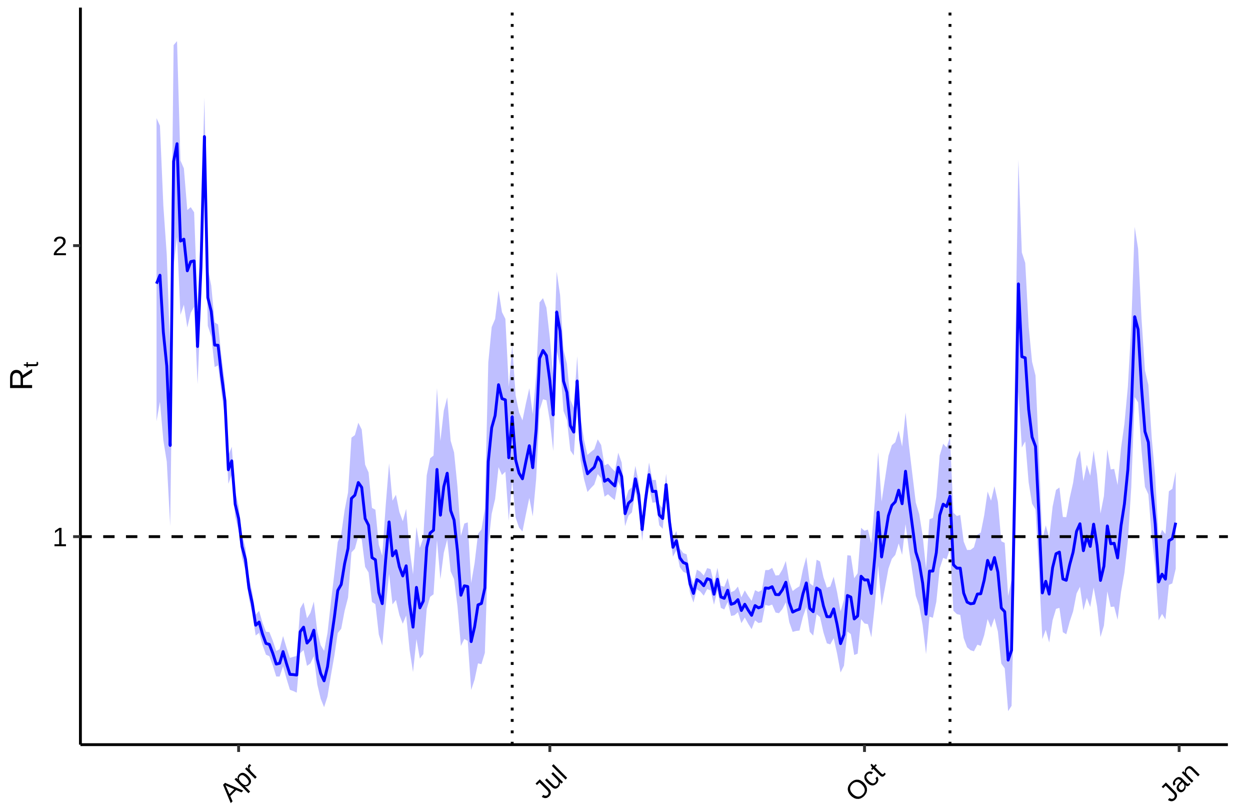
Your final task is to find an incidence data set for SARS-CoV-2 from any country or territory (e.g., a US State) and use the *EpiEstim* package to estimate R_t over the data set.

Complete the following:

1. Find your data set (lots of online resources for this) and make sure you get it in into the correct format for *EpiEstim*. Note that no dates can be missing from the data set. If you have missing dates, just put 0 for the incidence.
2. Restrict the data set from March 2020 to end of December 2020, before the COVID-19 vaccine became broadly available.
3. Use *EpiEstim* to estimate R_t from the data set. For the mean serial interval distribution of SARS-CoV-2, use 3.6 days, and for the standard deviation of the serial interval, use 3.0 days.
4. Create a graph of the pattern, as above.
5. Write a few sentences describing the pattern. Specifically, relate the implementation of one or two known public health interventions (and their implementation dates) to the pattern of R_t. You don't need to mention all possible interventions, but do some research to comment on one or two obvious ones.

```
# Data taken from: https://github.com/owid/covid-19-data/tree/master/public/data
# March - December 2020
# Austrailia ('straya)

readfile <- read.csv("C:\\Users\\richard\\Documents\\School\\INF 414\\owid-covid-data-auz.csv")

# Note: EpiEstim requires a specific format for the data frame.
Rt_cases_df =
  readfile %>%
  rename(I = new_cases, dates = date)

readfile$date <- mdy(readfile$date)

# Below configures the specifications for the R_t estimation
# We will estimate R_t across 7 day moving windows
# mean_si is the mean of the serial interval distribution
# std_si is the standard deviation of the serial interval ditribution
window_length = 7
n_add = window_length - 1
n_dates = nrow(Rt_cases_df)
t_start = seq(2, n_dates - n_add)
t_end = t_start + n_add

config_list =
  EpiEstim::make_config(
    list(
      t_start = t_start,
      t_end = t_end,
      mean_si = 3.6,
      std_si = 3.0)
  )

Rt_cases_df$dates <- mdy(Rt_cases_df$dates)
```

```r
weekly_Rt =
  EpiEstim::estimate_R(
    Rt_cases_df,
    method="parametric_si",
    config = config_list
  )

Rt_estim_df = weekly_Rt$R
Rt_estim_df$dates = Rt_cases_df$dates[(n_add + 2):n_dates]

min_date = min(Rt_cases_df$dates)
max_date = max(Rt_cases_df$dates)

ggplot(Rt_estim_df) +
  geom_path(aes(x = dates, y = `Median(R)` ),
            color = "blue") +
  geom_ribbon(aes(x = dates,
                  ymin = `Quantile.0.025(R)`,
                  ymax = `Quantile.0.975(R)`),
              fill = "blue", alpha = 0.25) +
  geom_hline(yintercept = 1, linetype = 2) +
  # date 112 is June 20th, 2020 first date of restrictions implemented
  geom_vline(xintercept=as.numeric(Rt_cases_df$dates[112]), linetype=3) +
  # date 240 is October 27th, 2020 end date of restrictions implemented
  geom_vline(xintercept=as.numeric(Rt_cases_df$dates[240]), linetype=3) +
  labs(x = "", y = expression(R[t])) +
  scale_x_date(limits = c(min_date, max_date)) +
  theme_classic() +
  theme(
    axis.text = element_text(size = 10, color = "black"),
    axis.title = element_text(size = 12, color = "black"),
    axis.text.x = element_text(angle = 45, vjust = 0.5)
  )
```



This model of R_t is generated from data taken from Austrailia, between March and December of 2020. The first vertical marked line shows the date at which the government implemented restrictions on non-essential travel. From this date, R_t briefly increases, coinciding with the massive summer peak of cases. Afterwards begins a steady downward trend, even becoming negative for several months, coinciding with the downside of the summer peak.

The second vertical line shows the date at which the government ended travel restrictions. The R_t value remains mostly at or below 1, and despite several sharp spikes in R_t, the number of new cases recorded in country remained consistantly low for the rest of the year, suggesting that the travel restrictions were effective.