Cases and test positivity in Victoria

I wanted to understand these issues better, particularly in the context of the recent marked increase in Covid-19 cases in Melbourne, Victoria, Australia where I live. Let's start with looking at confirmed case numbers and test positivity rates. The Guardian compiles various announcements, media releases, dashboards and the like from the different state authorities around Australia into a tidy Google sheet. Thank you The Guardian. The data are presented as cumulative totals of tests and of cases whenever data comes in – so they are *event* data, with between zero and four observation events per day per state. This needs a bit of tidying to turn into daily increases in cases and tests.

Here's the number of cases in Victoria, with and without a modest adjustment for test positivity as set out in my earlier post:

The history here will be familiar for Australians, but for other newcomers, the first burst of infections was driven nearly entirely by international travellers. There was a very low level of community transmission at that point, and the social distancing, test and trace, and lockdown measures pursued by governments at both Federal and State level were effective in suppressing the cases down to single digits. Then, since the second half of June, there has been a big increase in cases, this time locally contracted.

The adjustment I'm making for positivity (the aqua-blue line) is very mild – I'm multiplying the confirmed cases by positivity to the power of 0.1 and then scaling the result so that the cases at the *lowest* test positivity are left as the originally were. I'm not using the power of 0.5 (ie square root) that I used in my previous post in the US context and that has been taken up by some other analysts. I don't believe testing bottlenecks are as much of an issue here as in the US; I think our testing regime comes much closer to the ideal of a census of cases, or at least a relatively constant proportion of cases allowing for a big but relatively stable proportion of asymptomatic cases below the radar. To see why, here's the time series of test positivity in Victoria:

That uptick is an issue, but it's clearly a different order of magnitude of challenge to that faced in the US, where several locations have seen 20% of tests returning positive or higher (even 50% – one in two tests returning positive!), a major flashing red light that many many cases remain undiagnosed. The bottom line – there's a lot of testing going on here in Victoria at the moment, I'm not sure it's a major bottleneck for counting cases, and I only want to make a small adjustment for the positivity rate which is so close to zero for a lot of the period of interest.

Here's the code to download that data from The Guardian, estimate daily changes, model and smooth the test positivity rate and estimate an adjusted rate. It also sets us up for our next look at the issue of dealing with the delay, and with the downwards bias in recent days' counts resulting from how we deal with the delay.

Post continues after R code

```
library(tidyverse)
library(googlesheets4)
library(janitor)
library(scales)
library(mgcv)
library(EpiNow2) # remotes::install_github("epiforecasts/EpiNow2")
library(frs)     # removes::install_github("ellisp/frs-r-package/pkg")
library(patchwork)
library(glue)
library(surveillance) # for backprojNP()

#-----------------the Victoria data-------------

url <- "https://docs.google.com/spreadsheets/d/1q5gdePANXci8enuiS4oHUJxcxC13d6bjMRSicakychE/edit#gid=1437767505"

gd_orig <- read_sheet(url)
```

```
d <- gd_orig %>%
  clean_names() %>%
  filter(state == "VIC") %>%
  # deal with problem of multiple observations some days:
  mutate(date = as.Date(date)) %>%
  group_by(date) %>%
  summarise(tests_conducted_total = max(tests_conducted_total, na.rm =
TRUE),
            cumulative_case_count = max(cumulative_case_count, na.rm =
TRUE)) %>%
  mutate(tests_conducted_total  = ifelse(tests_conducted_total < 0,
NA, tests_conducted_total),
         cumulative_case_count = ifelse(cumulative_case_count < 0, NA,
cumulative_case_count)) %>%
  ungroup() %>%
  # correct one typo, missing a zero
  mutate(tests_conducted_total = ifelse(date == as.Date("2020-07-10"),
1068000, tests_conducted_total)) %>%
  # remove two bad dates
  filter(!date %in% as.Date(c("2020-06-06", "2020-06-07"))) %>%
  mutate(test_increase = c(tests_conducted_total[1],
diff(tests_conducted_total)),
         confirm = c(cumulative_case_count[1],
diff(cumulative_case_count)),
         pos_raw = pmin(1, confirm / test_increase)) %>%
  complete(date = seq.Date(min(date), max(date), by="day"),
           fill = list(confirm = 0)) %>%
  mutate(numeric_date = as.numeric(date),
         positivity = pos_raw) %>%
  filter(date > as.Date("2020-02-01")) %>%
  fill(positivity, .direction = "downup") %>%
# I don't believe the sqrt "corrected" cases helped here so have a
much more modest 0.1.
# But first we need to model positivity to smooth it, as it's far too
spiky otherwise:
  mutate(ps1 = fitted(gam(positivity ~ s(numeric_date), data = .,
family = "quasipoisson")),
         ps2 = fitted(loess(positivity ~ numeric_date, data = ., span
= 0.1)),
         cases_corrected = confirm * ps1 ^ 0.1 / min(ps1 ^ 0.1)) %>%
  ungroup() %>%
  mutate(smoothed_confirm = fitted(loess(confirm ~ numeric_date, data
= ., span = 0.1)))


the_caption <- "Data gathered by The Guardian; analysis by
http://freerangestats.info"

# Positivity plot:
d %>%
  ggplot(aes(x = date)) +
  geom_point(aes(y = pos_raw)) +
  geom_line(aes(y = ps2)) +
  scale_y_continuous(label = percent_format(accuracy = 1)) +
  labs(x = "", y = "Test positivity",
       title = "Positive test rates for COVID-19 in Melbourne,
```

```
Victoria",
       caption = the_caption)

# Case numbers plot
d %>%
  select(date, cases_corrected, confirm) %>%
  gather(variable, value, -date) %>%
  mutate(variable = case_when(
    variable == "confirm" ~ "Recorded cases",
    variable == "cases_corrected" ~ "With small adjustment for test
positivity"
  )) %>%
  ggplot(aes(x = date, y = value, colour = variable)) +
  geom_point()  +
  geom_smooth(se = FALSE, span = 0.07) +
  labs(x = "", y = "Number of new cases per day",
       colour = "",
       caption = the_caption,
       title = "Covid-19 cases per day in Melbourne, Victoria",
       subtitle = "With and without a small adjustment for test
positivity. No adjustment for delay.")
```

Convolution and Deconvolution

One of the obvious challenges for estimating effective reproduction number () is the delay between infection and becoming a confirmed case. The Gostic et al paper looked at different ways of doing this. They found that with poor information, an imperfect but not-too-bad method is just to left-shift the confirmed cases by subtracting the estimated average delay from infection to reporting. A better method takes into account the distribution of that delay – typically a Poisson or negative binomial random variable of counts of days. However, a common approach to do this by subtracting delays drawn from that distribution is noticeably worse than simply subtracting the mean. In the words of Gostic et al:

> “One method infers each individual’s time of infection by subtracting a sample from the delay distribution from each observation time. This is mathematically equivalent to convolving the observation time series with the reversed delay distribution. However, convolution is not the correct inverse operation and adds spurious variance to the imputed incidence curve. The delay distribution has the effect of spreading out infections incident on a particular day across many days of observation; subtracting the delay distribution from the already blurred observations spreads them out further. Instead, deconvolution is needed. In direct analogy with image processing, the subtraction operation blurs, whereas the proper deconvolution sharpens”

To be clear, it’s not just armchair epidemiologists who are wrongly using convolution backwards in time here, but specialists performing Covid-19 surveillance or research for their professional roles. So this paper does a great service in pointing out how it makes things worse. I think that at least one high profile dashboard of estimates has modified its method based on the findings in this paper already. Self-correcting science at work!

Recovering unobserved past infections with toy data

To see this issue in action I made myself two functions `blur()` and `sharpen()` which respectively do the simple convolution and deconvolution tasks described above in a deterministic way. The job of `blur()` is to delay a vector of original cases in accordance with a given distribution. In effect, this spreads out the original vector over a greater (and delayed) time period. The job of `sharpen()` is to reverse this process – to recover an original vector of observations that, if blurred, would create the original (unobserved) incidence counts.

I tested these two functions with a super-simple set of six original incidence counts, the vector `4, 6, 8, 9, 7, 5`. I blurred these into the future in accordance with expected values of a Poisson distribution with a

mean of 3 days. Then I used `sharpen()` to recover the original values, which it does with near perfect success:

My blur() and sharpen() functions (which are completely deterministic and not fit in my opinion for dealing with real-life random data) are in the `frs` R package where I store miscellaneous stuff for this blog.

Here's he code that makes those toy example original, delayed and recovered time series:

```
#------------------understanding convolution----------------------

x <- c(4,6,8,9,7,5)
pmf <- dpois(0:14, lambda = 3)
pmf <- pmf / sum(pmf)

# create a lagged version of x, with lags determined by a known
probability mass function:
y <- blur(x, pmf, scale_pmf = TRUE)
# recover the original version of x, given its blurred version
# and the original probabilities of delays of various lags:
recovered <- sharpen(y, pmf)


p_conv <- tibble(original_x = x, position = 1:length(x)) %>%
  right_join(recovered, by = "position") %>%
  gather(variable, value, -position) %>%
  mutate(variable = case_when(
    variable == "original_x" ~ "Original (unobserved) values",
    variable == "x" ~ "Original values recovered",
    variable == "y" ~ "Values after a delay"
   )) %>%
  filter(position > -1) %>%
  ggplot(aes(x = position, y = value, colour = variable)) +
  geom_point() +
  geom_line() +
  theme(legend.position = "right") +
  labs(title = "Convolution and deconvolution demonstrated with
simulated data",
       colour = "",
       x = "Time",
       y = "Value")
```

… and for completeness, here are the definitions of my `blur()` and `sharpen()` functions. Most of the versions of these I've seen in R, Python and Stan use loops, but the particular nature of the "multiply every value of a vector by the probabilities for various lags" operation suggested to me it would be simpler to write as a join of two tables. Maybe I spend too much time with databases. I did think at one point in writing these that I seemed to be re-inventing matrix multiplication, and I'm sure there's a better way than what I've got; but the complications of forcing the recovered vector of original infections to match the observed vector were too much for me to come up with a more elegant approach in the hour or so time budget I had for this bit of the exercise.

```
#' One-dimensional convolution
#'
#' @param x a vector of counts or other numbers
#' @param pmf a vector of probabilities to delay x at various lags.
First value should be for lag 0,
#' second for lag 1, etc.
#' @param warnings whether to show warnings when the resulting vector
does not add up to the
#' same sum as the original
```

```
#' @param scale_pmf whether or not to scale pmf so it adds exactly to
one
#' @details \code{blur} and \code{sharpen} are deterministic single
dimensional convolution functions
#' for simple convolution by a lagged probability mass function
representing the proportion of original
#' cases that are delayed at various lags. They are for illustrative /
toy purposes and probably should
#' not be used for actual analysis.
#' @return A vector of length equal to the length of x plus length of
pmf minus 1
#' @export
#' @importFrom dplyr left_join
#' @examples
#' x <- c(4,6,8,9,7,5)
#' pmf <- c(0, 0.3, 0.5, 0.2)
#' blur(x, pmf)
blur <- function(x, pmf, warnings = TRUE, scale_pmf = FALSE){

  if(!class(x) %in% c("integer", "numeric") || !is.null(dim(x))){
    stop("x should be a numeric vector of numbers")
  }

  if(!"numeric" %in% class(pmf) || !is.null(dim(pmf))){
    stop("pmf should be a numeric vector of probabilities")
  }

  if(scale_pmf){
    pmf <- pmf / sum(pmf)
  }

  cvd <- data.frame(pmf = pmf, link = 1, lag = 0:(length(pmf) - 1))
  orig <- data.frame(x = x, link = 1)
  orig$position <- 1:nrow(orig)
  combined <- dplyr::left_join(orig, cvd, by = "link")
  combined$z <- with(combined, x * pmf)
  combined$new_position <- with(combined, position + lag)
  y <- aggregate(combined$z, list(combined$new_position), sum)$x

  if(sum(x)!= sum(y) & warnings){
    warning("Something went wrong with blur; result did not sum up to
original")
  }

  return(y)
}

#' Single dimensional deconvolution
#'
#' @param y a vector of values to be deconvolved
#' @param pmf a vector of probabilities for the original convolution
that created y
#' @param warnings passed through to blur
#' @param digits how many digits to round the deconvolved values to.
If NULL no rounding occurs.
#' @details \code{blur} and \code{sharpen} are deterministic single
dimensional convolution functions
```

```
#' for simple convolution by a lagged probability mass function
representing the proportion of original
#' cases that are delayed at various lags. They are for illustrative /
toy purposes and probably should
#' not be used for actual analysis.
#'
#' Use \code{\link[surveillance]{backprojNP}} for a better, maximum
likelihood approach to recovering
#' an unseen set of original cases that result in the observations.
#'
#' \code{sharpen} is the inverse of \code{blur}; it seeks to recover
an original vector that, when blurred
#' via \code{pmf}, would produce the actual observations.
#' @return a data frame with columns for x (the inferred original
values what were convolved to y),
#' y (which will be padded out with some extra zeroes), and position
(which is the numbering of the
#' row relative to the original ordering of y; so position = 1 refers
to the first value of y)
#' @seealso \code{\link[surveillance]{backprojNP}}.
#' @export
#' @examples
#' x <- c(4,6,8,9,7,5)
#' pmf <- c(0, 0.3, 0.5, 0.2)
#' # create a convolved version of x:
#' y <- blur(x, pmf)
#' # recover the original version of x, given its blurred version
#' # and the original convolution probabilities:
#' sharpen(y, pmf)
sharpen <- function(y, pmf, warnings = FALSE, digits = NULL){
  y2 <- c(rep(0, length(pmf)), y)
  starter_x <- c(y, rep(0, length(pmf)))

  fn <- function(x){
    x <- x / sum(x) * sum(y2)
    d <- sqrt(sum((blur(x, pmf, warnings = warnings, scale_pmf =
TRUE)[1:length(x)] - y2) ^ 2))
    return(d)
  }

  op_res <- optim(starter_x, fn, lower = 0, method = "L-BFGS-B")

  x <- op_res$par
  x <- x / sum(x) * sum(y2)
  if(!is.null(digits)){
    x <- round(x, digits = digits)
  }

  output <- data.frame(x, y = y2)
  output$position <- seq(from = length(y) - nrow(output) + 1, to =
length(y), by = 1)
  return(output)

}
```

A big limitation on these toy `blur()` and `sharpen()` functions is that they assume not only that we know the probability mass for each possible day of lagging, but that the process is deterministic. Of course, this isn't the case. The `backprojNP()` function in the `surveillance` R package estimates those unobserved

latent series on the assumption that the observations come from a Poisson process. Here's how that looks when we apply it to my toy data:

This uses a method first developed in the context of understand the spread of HIV, which has a particularly long and uncertain delay between infection and confirmation of a case.

It doesn't do as well at recovering the original as my `sharpen()` did, because `sharpen()` has the luxury of knowing the blurring took place deterministically. With these small numbers that makes as big difference. But `backprojNP()` does an ok job at recovering at least some of the original structure. It's a *bit* sharper than the blurred observations. Here's the code applying `backprojNP()` to my toy data (mostly this is plotting data; I stuck to base graphics to give me a bit more control over the legend):

```
# Make a "surveillance time series" of our toy observations
y_sts <- sts(y)
# Back-propagate to get an estimate of the original, assuming our
observations
# are a Poisson process
x2 <- backprojNP(y_sts, pmf)

par(family = "Roboto")
plot(x2, xaxis.labelFormat = NULL,
   legend=NULL, lwd=c(1,1,3), lty=c(1,1, 1),
   col = c("grey80", "grey90", "red"),
   main="", bty = "l",
   ylim = c(0, 10),
   xlab = "Time",
   ylab = "Infections")
title("Recovering / sharpening unobserved infections with simulated
toy data and \nsurveillance::backprojNP()",
        adj = 0, family = "Sarala", font.main = 1)
points(1:6, x, col = "orange", cex = 4, pch = "-")

legend(15, 10, legend = c("Observed", "Back-propagated", "Original"),
         pch = c(15, NA, NA),
         lty = c(0, 1, 1),
         lwd = c(0, 3, 3),
         col = c("grey80", "red", "orange"),
         pt.cex = c(2, 2, 2),
         cex = 0.8,
         bty = "n")

# note the actual estimates are in x2@upperbound, which adds up to 39,
same as the original y
```

Not only does `backprojNP()` do a better job at coping with real-life random data, it's much faster than my `sharpen()` function. So I should emphasise again that `frs::sharpen()` is here purely for illustrative purposes, to help me get my head around how this whole single dimensional deconvolution thing works.

Recovering unobserved past infections with toy data

OK, so how does this go when we apply it to real Covid-19 data? A quick google suggests that the delay from infection to symptoms is approximately a Poisson distribution with a mean of 6 days. The time from symptoms to becoming a confirmed case is unfortunately very much unknown, and also is likely to change over time (for example, as queues for testing lengthen or shorten, and testing regimes become more or less aggressive). I took a guess for illustrative purposes that it is going to be a Poisson distribution with a mean of about 3 days, shifted one day to the right (because it takes at least one day to get one's test results). As the sum of two Poisson distributions is another Poisson distribution, this means we can model the overall delay from (unobserved) infection to confirmation as a Poisson distribution with mean of 9 days, plus one day for good luck. In the code below the probability of delay by various lags is defined on this basis in the

`pmf_covid` vector.

Here's what this looks like when I apply it to the Victorian data:

Overall the curve looks like how we'd expect – earlier than the observed cases, and a little steeper (not as much steeper as I expected though – in the course of writing this blog over a few days I noted this curve can change noticeably as individual data points come in, so it should be treated with caution – see later for a better chart).

Note how this chart really draws attention to the final challenge we'll be looking at – the right truncation of the data. That massive drop in the aqua-blue line is very misleading, because it's based on there being zero confirmed cases from tomorrow onwards! We've got a way to go yet before we want to turn this into estimates of effective reproduction number, but we're on track.

Here's the code for that bit of analysis:

```
pmf_covid <- c(0, dpois(0:20, lambda = 9))

# takes a few minutes
bp_covid <- backprojNP(sts(d$confirm), pmf_covid)

sharpened <- tibble(recovered_x = bp_covid@upperbound) %>%
  mutate(position = 1:n())

p_adj_conv <- d %>%
  mutate(position = 1:n()) %>%
  left_join(sharpened, by = "position") %>%
  select(date, recovered_x, confirm)  %>%
  mutate(recovered_x = replace_na(recovered_x, 0)) %>%
  gather(variable, value, -date) %>%
  mutate(variable = case_when(
    variable == "confirm" ~ "Confirmed cases",
    variable == "recovered_x" ~ "Estimated original cases accounting
for delay"
  )) %>%
  ggplot(aes(x = date, y = value, colour = variable)) +
  geom_line(size = 1.5) +
  labs(title = "Back-projection of Victorian Covid-19 infections",
       subtitle = str_wrap("Non-parametric back-projection of
incidence cases assuming
       average delay of 10 days between infection and observation,
using methods in
                           Becker et al (1991). No correction for
right truncation of data,
                           so the last 15 days will be badly biased
downwards.", 100),
       x = "",
       y = "Number of infections",
       colour = "")

print(p_adj_conv1)
```

Adjust for testing, delay and right truncation all at once

Clearly the problem of adjusting for the delay is going to need a more careful evidence base than my casual guess at a Poisson variable with mean of 9 days displaced one day to the right; we'll need an effective way to manage the smoothing of our inferred original infection estimates; we need a way of estimating the unobserved cases; and a method for dealing with the "right truncation" ie the absence of data that can be mapped to infections today, yesterday, and other recent days.

Following one of the footnotes in Gostic et al (referring to “Many statistical methods are available to adjust for right truncation in epidemiological data”) led me to the very excellent `EpiNow2` R package, which tries to do most of this simultaneously. It wraps a Bayesian Gaussian process model implemented in Stan, which is exactly the way I think this problem should be approached. Their documentation helpfully includes instructions on using their package for real-time estimation of Covid-19, exactly my problem. It even includes literature-based estimates of the key characteristics of the appropriate distributions to use for estimating and combining different types of lags (generation, incubation and reporting).

`EpiNow2` is one of the tools developed and used by the team behind epiforecasts.io – Sebastian Funk and others at the London School of Hygiene and Tropical Medicine. The guts of the software is this Stan program.

While it’s still under development, it’s very impressive. Hats off to the team. Here’s the full citation, noting a fair bit of cross-over with both the epiforecasts.io team and the Gostic et al paper I started this blog with a link to:

```
Sam Abbott, Joel Hellewell, Robin Thompson, Katelyn Gostic, Katharine Sherratt,
 Sophie Meakin, James Munday, Nikos Bosse and Sebastian Funk (2020). EpiNow2:
 Estimate Realtime Case Counts and Time-varying Epidemiological Parameters.
 R package version 0.3.0.
```

Here’s what I get when I apply their approach out-of-the-box to Victoria’s case counts:

As can be seen, several sets of estimates are returned. Panel B is important – it shows the estimated actual infection counts (the curving ribbon, which reflects a credibility interval) in contrast to the reported confirmed cases (the grey columns). It makes it easy to see the broadening uncertainty as we get to today and even a bit into the future; and also how the estimated infections precede the confirmed cases.

From panel B, the estimates of instantaneous reproduction number follow in straightforward fashion, as seen in the bottom panel. I’ve opted to only show the estimates of from late April onwards because prior to that the cases were dominated by international arrivals. While methods exist to estimate reproduction number appropriately in this circumstance, I’m not sufficiently interested in that bit of history (*months* ago now…) to go to the effort to do it.

Of course, this chart is cautiously good news for Victorians. The brown ‘nowcasting’ segment of the plot shows a decided downwards trend in estimated infections, timing closely with the extra shutdown and distancing measures brought in just under two weeks ago. And it shows the best estimate of today’s effective reproduction number to be (just) less than 1. Let’s hope that stays the case.

That first fit was with the confirmed case numbers directly from The Guardian. If I instead apply them to the numbers after my mild correction for test positivity, we see a similar picture. Recent estimated case numbers are higher because of the higher test positivity, but the estimate of today is similar – still just below 1.0, and on its way down (albeit with lots of uncertainty).

That uncertainty issue is actual one of my main motivators for writing this blog. The fact is, we don’t know what’s going to happen here. This is definitely one of those cases when one of the most useful things a forecast or nowcast can do is highlight the range of possibilities that are consistent with the data we have. And critically, what happens in the future – that big blue credibility interval in the last couple of charts – depends on actual people’s actual decisions and actions.

> “The fault, dear Brutus, is not in our stars, but in ourselves”

That reminds me, a Shakespearean blog post really is on the way.

So the situation in Melbourne is clearly on a knife edge. Today’s numbers (Saturday 18 July) are good (in fact exactly what and when would be hoped if the suppression strategy is doing its job), but we know not to pay too much attention to one point in these noisy processes. In a week’s time, any number of cases per day from zero to more than 1,000 (the chart is cut off well below its top point) is possible. Let’s remember those big, uncertain prediction intervals; not be too confident about which way things are headed; and do our best to point them in the right direction by our own behaviour. Stay safe out there! Practice social distancing, stay at home as much as possible, wear a mask when going out, and pay attention to your local public health

experts. New Zealanders are excluded the first three of those four things, because they did the fourth one reasonably well.

Anyway, here's the code for those final bits of analysis. Running the estimation processes took several hours each on my laptop:

```
#------------Estimating R with EpiNow2---------------------
# Various delay/lag distributions as per the Covid-19 examples in the
EpiNow2 documentation.

reporting_delay <- EpiNow2::bootstrapped_dist_fit(rlnorm(100, log(6),
1))
## Set max allowed delay to 30 days to truncate computation
reporting_delay$max <- 30

generation_time <- list(mean = EpiNow2::covid_generation_times[1,
]$mean,
                        mean_sd = EpiNow2::covid_generation_times[1,
]$mean_sd,
                        sd = EpiNow2::covid_generation_times[1, ]$sd,
                        sd_sd = EpiNow2::covid_generation_times[1,
]$sd_sd,
                        max = 30)

incubation_period <- list(mean = EpiNow2::covid_incubation_period[1,
]$mean,
                          mean_sd = EpiNow2::covid_incubation_
period[1, ]$mean_sd,
                          sd = EpiNow2::covid_incubation_period[1,
]$sd,
                          sd_sd = EpiNow2::covid_incubation_period[1,
]$sd_sd,
                          max = 30)


#---------Based on original data-------------

estimates <- EpiNow2::epinow(reported_cases = d,
                             generation_time = generation_time,
                             delays = list(incubation_period,
reporting_delay),
                             horizon = 7, samples = 3000, warmup =
600,
                             cores = 4, chains = 4, verbose = TRUE,
                             adapt_delta = 0.95)

# Function for doing some mild polishing to the default plot from
epinow();
# assumes existence in global environment of a ggplot2 theme called
my_theme,
# defined previously, and takes the output of epinow() as its main
argument:
my_plot_estimates <- function(estimates, extra_title = ""){
  my_theme <- my_theme +
    theme(axis.text.x = element_text(angle = 45, size = 8, hjust = 1))

  p <- estimates$plots$summary
```

```
  p1 <- p$patches$plots[[1]] +
    scale_y_continuous(label = comma_format(accuracy = 1)) +
    my_theme +
    theme(legend.position = "none",
          panel.grid.minor = element_blank()) +
    coord_cartesian(ylim = c(0, 1000)) +
    labs(title = glue("Estimated infections based on confirmed
cases{extra_title}"),
         x = "") +
    scale_x_date(date_breaks = "1 week", date_labels = "%d %b", limits
= range(p$data$date))

  p2 <- p$patches$plots[[2]] +
    scale_y_continuous(label = comma_format(accuracy = 1)) +
    my_theme +
    theme(legend.position = "none",
          panel.grid.minor = element_blank()) +
    coord_cartesian(ylim = c(0, 1000)) +
    labs(title = glue("Estimated infections taking delay{extra_title}
into account"),
         x = "") +
    scale_x_date(date_breaks = "1 week", date_labels = "%d %b", limits
= range(p$data$date))

  p3 <- p$data %>%
    filter(date > as.Date("2020-04-20")) %>%
    ggplot(aes(x = date, y = median, fill = type)) +
    my_theme +
    geom_hline(yintercept = 1, colour = "steelblue") +
    geom_ribbon(aes(ymin = lower, ymax = upper), alpha = 0.5) +
    geom_ribbon(aes(ymin = bottom, ymax = top), alpha = 0.1) +
    geom_line(aes(colour = type)) +
    theme(legend.position = "none",
          panel.grid.minor = element_blank()) +
    ggplot2::scale_fill_brewer(palette = "Dark2") +
    labs(title = glue("Effective Reproduction Number, correcting for
both delay and right truncation{extra_title}"),
         y = bquote("Estimated" ~ R[t]),
         x = "")  +
    scale_x_date(date_breaks = "1 week", date_labels = "%d %b", limits
= range(p$data$date))

  pc <- p1 + p2 + p3 + plot_layout(ncol = 1)

  return(pc)
}


my_plot_estimates(estimates)

#---------Based on positivity-adjusted-------------

d2 <- select(d, date, cases_corrected) %>%
  mutate(confirm = round(cases_corrected) )

estimates2 <- EpiNow2::epinow(reported_cases = d2,
                              generation_time = generation_time,
```

```
                              delays = list(incubation_period,
reporting_delay),
                              horizon = 7, samples = 3000, warmup =
600,
                              cores = 4, chains = 4, verbose = TRUE,
                              adapt_delta = 0.95)


my_plot_estimates(estimates2, extra_title = " and positivity")
```