# na-notebook

2024-08-19

```
## [1] "is_na"  "signal" "issue"
```

## How Different NAs Arise and Strategies for Handling Them

### NAs from merging

First let's start with discussing the most common type of missing values that appeared in the context of my auxiliary signal project. When working with multiple signals each signal will likely begin recording at different times. In other words each signal's first data point $t_0$ will differ on the absolute time scale. As a result, when calling `epix_merge()` to combine multiple signals, the signals that started recording at a later point in time will have missing values for the time periods where the other signals were already recording. Here's a quick example

Say signal A reads with the values (11, 12, 14, 15, 16) with its first recorded data point being October 10th. Signal B reads (20, 21, 22, 23, 24) with its first recorded data point being 2 days later on October 12th. And lastly signal C reads (30, 32, 33, 34) with its first recorded data point being 3 days after that on October 15th. Then when we merge all of the data points together it will look like this:

| Time | Signal A | Signal B | Signal C |
|---|---|---|---|
| $t_0$ | 11 | NA | NA |
| $t_1$ | 12 | NA | NA |
| $t_2$ | 13 | 20 | NA |
| $t_3$ | 14 | 21 | NA |
| $t_4$ | 15 | 22 | NA |
| $t_5$ | NA | 23 | 30 |
| $t_6$ | NA | 24 | 32 |
| $t_7$ | NA | NA | 33 |
| $t_8$ | NA | NA | 34 |

Where $t_0$ would be October 10th.

Now let's take a look at some real data, and see how the same issue arises when combining Google Search data with doctor visits data for the states California, Florida, New York and Texas.

```
# Gathering the Data
print(getwd())
```

```
## [1] "/Users/ryannayebi/Downloads/delphi_project/epiprocess/vignettes"
```

```
states <- "ca,fl,ny,tx"
today <- 20240725

# s05 is Anosmia, Dysgeusia, Ageusia
s05_smoothed_search <- pub_covidcast(
  source = "google-symptoms",
  signals = "s05_smoothed_search",
```

```
  time_type = "day",
  geo_type = "state",
  time_values = epirange(20200220, today),
  geo_values = states,
  issues = epirange(20201130, today)
) %>%
  select(geo_value, time_value, version = issue, search_smoothed = value) %>%
  as_epi_archive()

smoothed_out_dr <- pub_covidcast(
  source = "doctor-visits",
  signals = "smoothed_adj_cli",
  time_type = "day",
  geo_type = "state",
  time_values = epirange(20201109, today),
  geo_values = states,
  issues = epirange(20200429, today)
) %>%
  select(geo_value, time_value, version = issue, out_visits_smoothed_dr = value) %>%
  as_epi_archive()
```

Let's verify that there aren't any NAs in the signals prior to merging. *Note we will be looking into the latest version of the data first*

```
latest <- function(x) {
  epix_as_of(x, max_version = max(x$versions_end))
}

print(any(is.na(latest(s05_smoothed_search)$search_smoothed)))
```

```
## [1] FALSE
```

```
print(any(is.na(latest(smoothed_out_dr)$out_visits_smoothed_dr)))
```

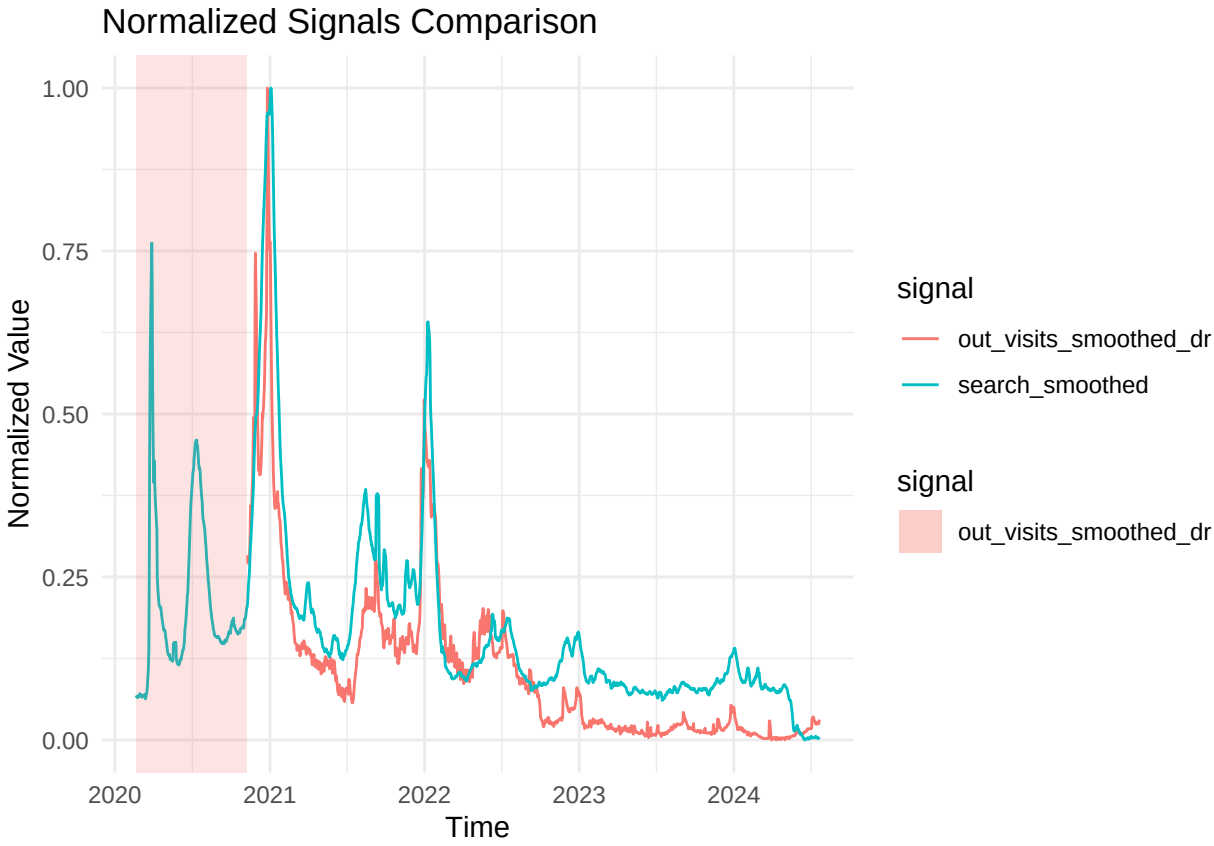
```
## [1] FALSE
```

```
# combining the signals
aux_signal <- epix_merge(s05_smoothed_search, smoothed_out_dr)
```

Now using the `plot_signals` function from my previous notebook for visualization, let's examine what this auxiliary signals looks like

```
aux_signal_latest_ca <- aux_signal %>%
  latest() %>%
  filter(geo_value == "ca")
plot_signals(aux_signal_latest_ca)
```

```
## Signal: search_smoothed
##   No gaps (no NAs found).
## Signal: out_visits_smoothed_dr
##   Gap: 2020-02-20 to 2020-11-08
```




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

Here we can now see that because of the mismatch between the starting recording dates we have lots of NAs at the beginning of the `out_visits_smoothed_dr` signal. Because of the fact that there is a large chunk of NAs at the very beginning of the data set, imputation isn't feasible for this region. This is because imputation works best in settings where there are relatively small holes within the data and you can use prior knowledge to predict the missingness of the data. Most of the time we can simply remove this larger chunk of missing data and focus on where both of signals exist. In this case, this would be the first available time where`out_visits_smoothed_dr` exits.

However, NAs can arise in different settings as well. For example if we look at this signal, NAs are more sparse and spread throughout the data. And this isn't caused by any merging actions, because we are only observing one individual signal. These can arise from things such as privacy issues, a noisy signal recording, or corrupted data.

(Perhaps omit the code to find the example)

```
cov_nas <- readRDS("cov_nas.rds") # simulated data, not real
paste0("geo_value: ", unique(cov_nas$geo_value))
```

```
## [1] "geo_value: tx"
```

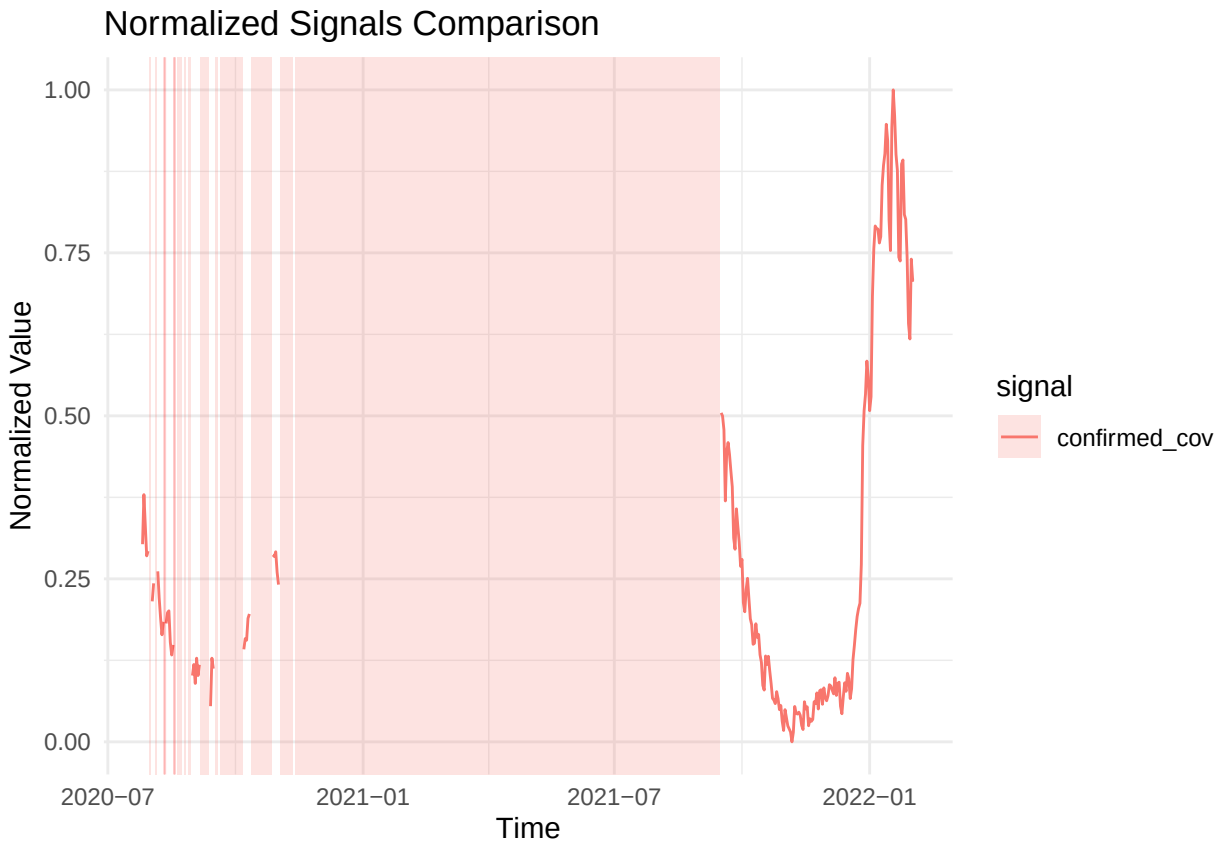
```
paste0("version: ", "2022-02-04")
```

```
## [1] "version: 2022-02-04"
```

```
plot_signals(complete(cov_nas))
```

```
## Signal: confirmed_cov
##   Gap: 2020-07-31 to 2020-08-01
##   Gap: 2020-08-04 to 2020-08-05
```

```
##   Gap: 2020-08-11 to 2020-08-11
##   Gap: 2020-08-18 to 2020-08-18
##   Gap: 2020-08-20 to 2020-08-23
##   Gap: 2020-08-25 to 2020-08-26
##   Gap: 2020-08-28 to 2020-08-30
##   Gap: 2020-09-06 to 2020-09-12
##   Gap: 2020-09-16 to 2020-09-18
##   Gap: 2020-09-20 to 2020-10-06
##   Gap: 2020-10-12 to 2020-10-27
##   Gap: 2020-11-02 to 2020-11-11
##   Gap: 2020-11-13 to 2021-09-15
```




```
## [1] 13
```

As a side note, notice how we use the `complete()` function here from the `tidyr` package. It is commonly used as a way to standardize our data as it ensures that all combinations of specified keys such as the time value and geos are present, even if this results in some missing values. If certain combinations are missing in the original data, `complete()` fills them in with rows containing NA values.

Here is small example:

```
data <- tibble(
  geo = c("CA", "CA", "NY"),
  time = as.Date(c("2020-01-01", "2020-01-03", "2020-01-01")),
  value = c(10, 20, 30)
)

data_completed <- data %>%
  complete(
```

```
    geo,
    time = seq.Date(from = min(time), to = max(time), by = "day")
  )

print(data_completed)
```

```
## # A tibble: 6 x 3
##   geo   time       value
##   <chr> <date>     <dbl>
## 1 CA    2020-01-01    10
## 2 CA    2020-01-02    NA
## 3 CA    2020-01-03    20
## 4 NY    2020-01-01    30
## 5 NY    2020-01-02    NA
## 6 NY    2020-01-03    NA
```

Returning back, as we can see from both the plot and the output of our `plot_signals()` function that there are many gaps of NA values (13 in total), many of which with only a few days missing, but one gap with almost a year of missing values. For now we will focus on the gaps that are relatively small. These NAs have likely arisen due to reporting delays or inconsistencies in data collection from the hhs source.

The process of filling in these NA values are known as imputation. Typically when imputing missing values, we use prior data in an attempt to predict what the signal would have been (there are exceptions as we will see later with the Google data).

There are two different time scales we can impute on when analyzing `epi_archive`s. Imputation in time and imputation in version. Imputation in time fills in missing values of the signal based on previous values from previous times points of that signal. On the other hand, because our signals have multiple versions of the same signal, which arise from correcting previous recordings of the data, we can use these previous versions of the data and fill in the missing value for a day based on the value from previous versions. This is known as imputation in version. We holding constant the same day, but looking back at previous values that this day had in other versions of the data.

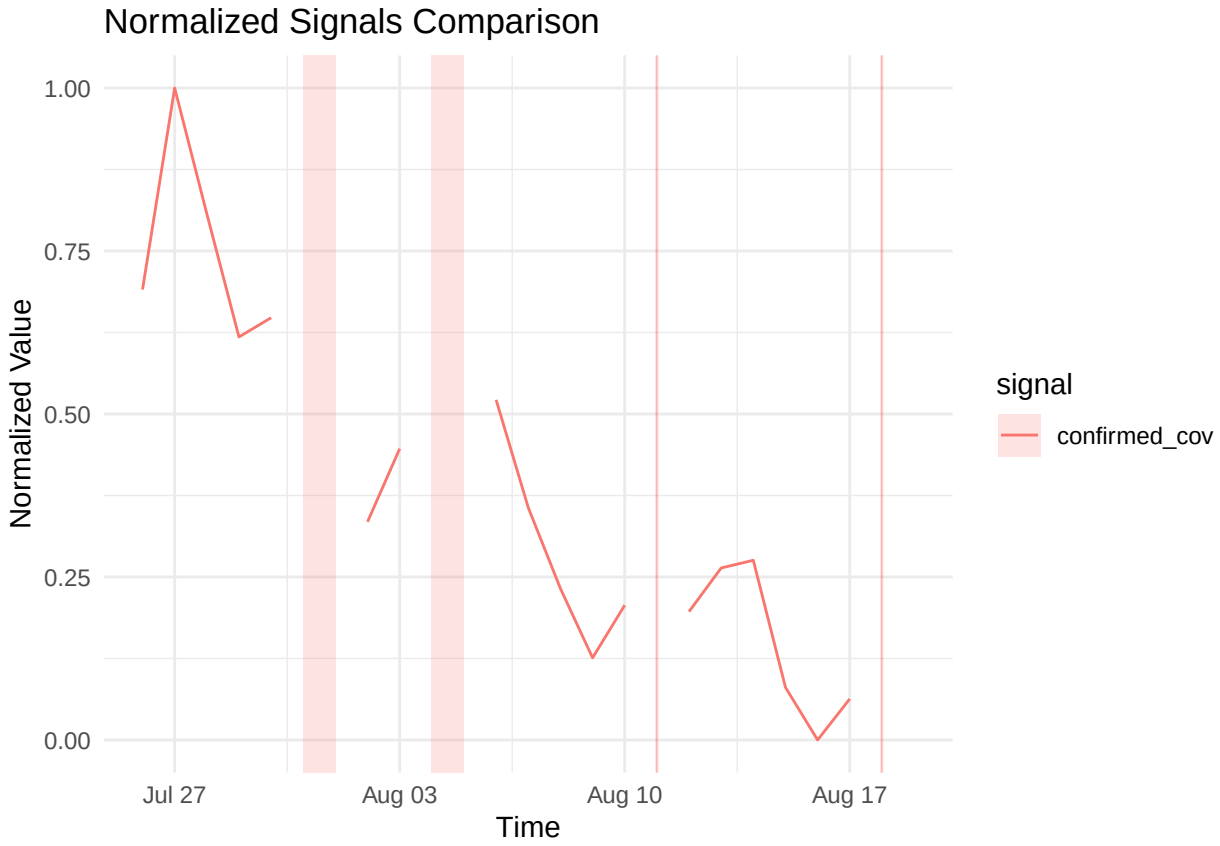
To deal with these smaller NA gaps we will impute in time (as opposed to in version). Two of the most common and more simpler imputation schemes are the last observation carried forward (LOCF) method and a moving average imputation method. As suggested in the name of the method, to do LOCF imputation you simply use the previous available value for the missing value. Here is the corresponding code for LOCF.

```
impute_locf <- function(data) {
  data %>%
    group_by(geo_value) %>%
    mutate(across(where(is.numeric), ~ zoo::na.locf(.x, na.rm = FALSE), .names = "{.col}_locf")) %>%
    ungroup()
}
```

Now let's apply this to the first few weeks of our data, where we have a lot of smaller gaps.

```
cov_nas_subset <- cov_nas %>% filter(time_value <= as.Date("2020-08-19"))
plot_signals(cov_nas_subset)
```

```
## Signal: confirmed_cov
##   Gap: 2020-07-31 to 2020-08-01
##   Gap: 2020-08-04 to 2020-08-05
##   Gap: 2020-08-11 to 2020-08-11
##   Gap: 2020-08-18 to 2020-08-18
```



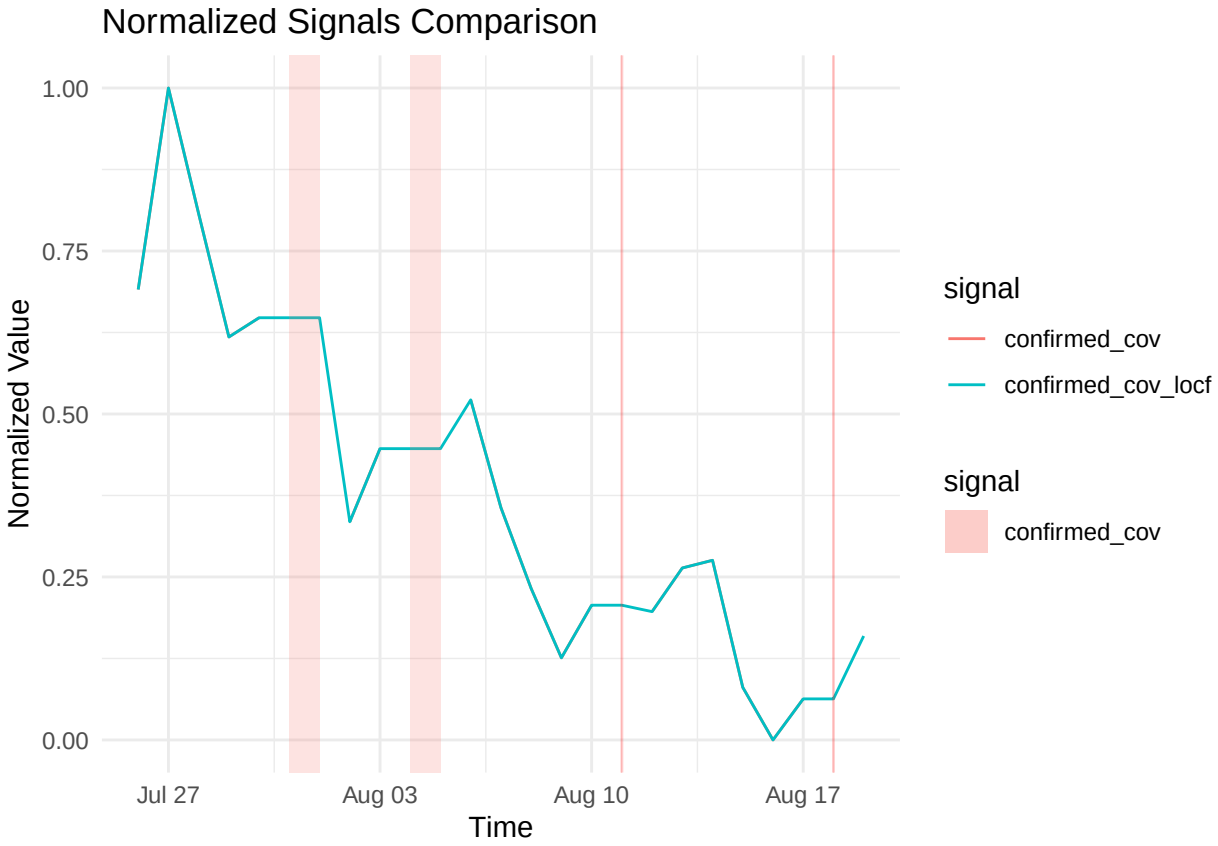

```
## [1] 4
```

Now let's apply the LOCF function that we wrote to our data and visualize it. We will plot it so you can see how the NAs were filled in. The red background is where the values were NA previosuly prior to the correction.

```
cov_nas_subset_locf <- cov_nas_subset %>% impute_locf()
plot_signals(cov_nas_subset_locf)
```

```
## Warning: There were 2 warnings in `summarize()`.
## The first warning was:
## i In argument: `start = min(time_value)`.
## Caused by warning in `min.default()`:
## ! no non-missing arguments to min; returning Inf
## i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

```
## Signal: confirmed_cov
##   Gap: 2020-07-31 to 2020-08-01
##   Gap: 2020-08-04 to 2020-08-05
##   Gap: 2020-08-11 to 2020-08-11
##   Gap: 2020-08-18 to 2020-08-18
## Signal: confirmed_cov_locf
##   No gaps (no NAs found).
```




```
## [1] 4
```

No more NAs in our final signal! But this a little bit of a naive approach. Rather than simply using the previous time step, perhaps we can utilize multiple previous values to come up with a prediction for our missing data using a moving average.

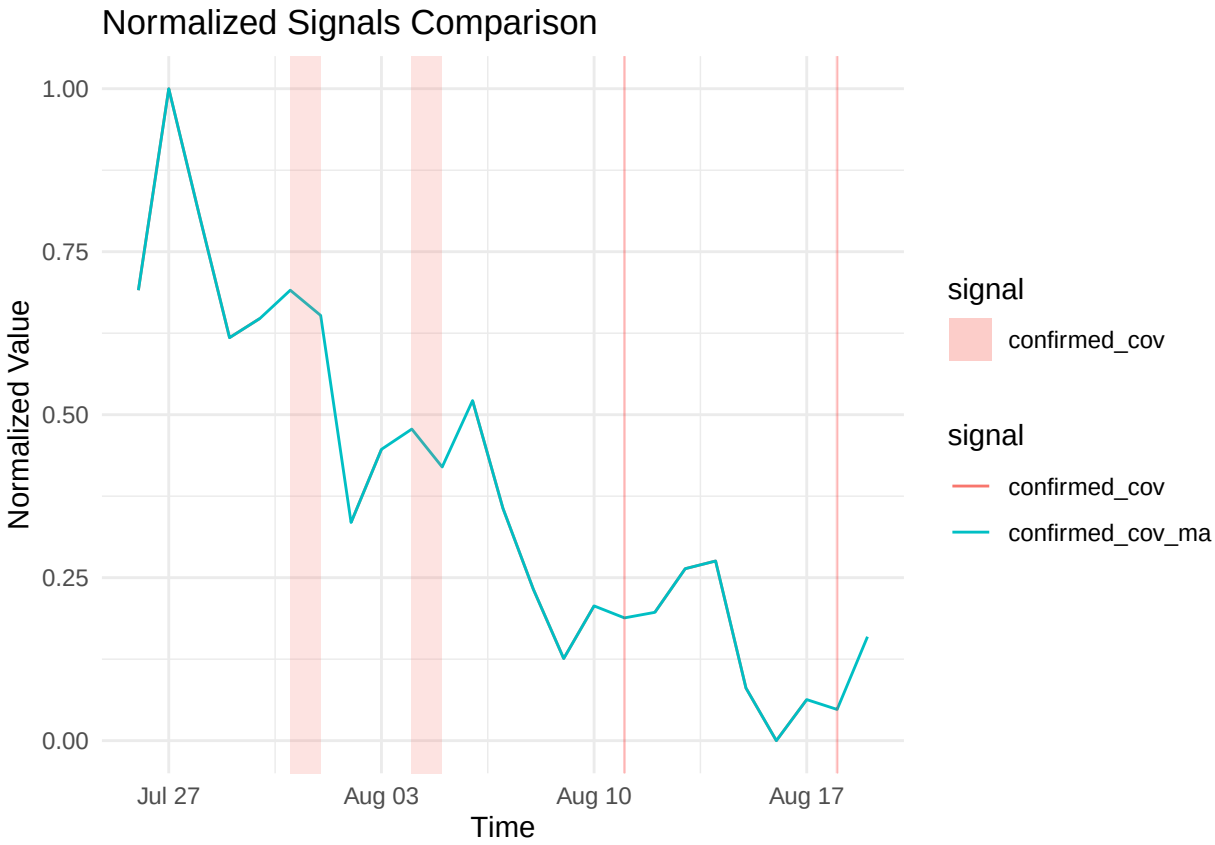
```
# *** note using an older version of epi_slide
impute_moving_average <- function(col, window_size = 3) {
  n <- length(col)
  for (i in seq(window_size, n - 1)) {
    curr_sum <- sum(col[max(1, i - window_size + 1):i])
    average <- curr_sum / window_size
    if (col[i + 1] %>% is.na()) {
      col[i + 1] <- average
    }
  }
  col
}
```

We will plot it so you can see how the NAs were filled in. The red background is where the values were NA previously prior to the correction.

```
cov_nas_subset_ma <- cov_nas_subset
cov_nas_subset_ma$confirmed_cov_ma <- cov_nas_subset_ma$confirmed_cov %>% impute_moving_average()
plot_signals(cov_nas_subset_ma)
```

```
## Signal: confirmed_cov
##   Gap: 2020-07-31 to 2020-08-01
##   Gap: 2020-08-04 to 2020-08-05
```

```
##   Gap: 2020-08-11 to 2020-08-11
##   Gap: 2020-08-18 to 2020-08-18
## Signal: confirmed_cov_ma
##   No gaps (no NAs found).
```




```
## [1] 4
```

We can confirm this is taking the average as looking at the original data we have:

```
cov_nas_subset
```

```
## # A tibble: 25 x 3
##    time_value geo_value confirmed_cov
##    <date>     <chr>              <dbl>
##  1 2020-07-26 tx                   885
##  2 2020-07-27 tx                  1042
##  3 2020-07-28 tx                   944
##  4 2020-07-29 tx                   848
##  5 2020-07-30 tx                   863
##  6 2020-07-31 tx                    NA
##  7 2020-08-01 tx                    NA
##  8 2020-08-02 tx                   704
##  9 2020-08-03 tx                   761
## 10 2020-08-04 tx                    NA
## # i 15 more rows
```

Let's do this by hand to verify that our results are correct, by taking the previous 3 values (window size specified) and computing the average

```r
jul_31 <- (944 + 848 + 863) / 3
aug_01 <- (848 + 863 + jul_31) / 3
aug_04 <- (aug_01 + 704 + 761) / 3

paste0("Value for Jul 31st: ", jul_31)
```

```
## [1] "Value for Jul 31st: 885"
```

```r
paste0("Value for Aug 1st: ", aug_01)
```

```
## [1] "Value for Aug 1st: 865.333333333333"
```

```r
paste0("Value for Aug 4th: ", aug_04)
```

```
## [1] "Value for Aug 4th: 776.777777777778"
```

```r
cov_nas_subset_ma
```

```
## # A tibble: 25 x 4
##    time_value geo_value confirmed_cov confirmed_cov_ma
##    <date>     <chr>             <dbl>            <dbl>
##  1 2020-07-26 tx                  885              885
##  2 2020-07-27 tx                 1042             1042
##  3 2020-07-28 tx                  944              944
##  4 2020-07-29 tx                  848              848
##  5 2020-07-30 tx                  863              863
##  6 2020-07-31 tx                   NA              885
##  7 2020-08-01 tx                   NA              865.
##  8 2020-08-02 tx                  704              704
##  9 2020-08-03 tx                  761              761
## 10 2020-08-04 tx                   NA              777.
## # i 15 more rows
```

Note that we can do this exact same appraoch also using multiple calls to `epi_slide_mean()`. Because we have to continually update the average each time we update an NA, especially for NAs that occur within the window range.

```r
cov_nas_subset_epi_ma <- cov_nas_subset %>% as_epi_df()
first <- TRUE
curr_pass <- cov_nas_subset_epi_ma %>% epi_slide_mean(confirmed_cov, before = 3, na.rm = TRUE)

for (i in 1:length(cov_nas_subset_epi_ma$confirmed_cov)) {
  curr_val <- cov_nas_subset_epi_ma$confirmed_cov[i]
  if (is.na(curr_val)) {
    cov_nas_subset_epi_ma$confirmed_cov[i] <- curr_pass$slide_value_confirmed_cov[i]
    curr_pass <- cov_nas_subset_epi_ma %>% epi_slide_mean(confirmed_cov, before = 3, na.rm = TRUE)
  }
}
cov_nas_subset_epi_ma
```

```
## An `epi_df` object, 25 x 3 with metadata:
## * geo_type  = state
## * time_type = day
## * as_of     = 2024-12-10 22:49:26.488079
##
## # A tibble: 25 x 3
##    geo_value time_value confirmed_cov
```

```
##    <chr>     <date>             <dbl>
##  1 tx        2020-07-26          885
##  2 tx        2020-07-27         1042
##  3 tx        2020-07-28          944
##  4 tx        2020-07-29          848
##  5 tx        2020-07-30          863
##  6 tx        2020-07-31          885
##  7 tx        2020-08-01          865.
##  8 tx        2020-08-02          704
##  9 tx        2020-08-03          761
## 10 tx        2020-08-04          777.
## # i 15 more rows
```

And as we can see we get the same results.

The next step is linear interpolation. Here we can think of this as using linear regression to be able to connect the dots between our NA points.

```r
# my implementation base R
# first pass create a list of tuples that are end points for each NA
# second passes goes through and runs the regressions between each end point
linear_interpolate_two_pass <- function(values) {
  interpolated_values <- values

  # first pass
  na_gaps <- list()
  in_gap <- FALSE
  for (i in seq_along(values)) {
    if (is.na(values[i])) {
      if (!in_gap) {
        # start of NA gap
        start <- i - 1
        in_gap <- TRUE
      }
    } else {
      if (in_gap) {
        # end of NA gap
        end <- i
        na_gaps <- append(na_gaps, list(c(start, end)))
        in_gap <- FALSE
      }
    }
  }

  # second pass
  for (gap in na_gaps) {
    start <- gap[1]
    end <- gap[2]
    if (start > 0 && end <= length(values)) {
      y1 <- values[start]
      y2 <- values[end]
      interpolated_section <- seq(y1, y2, length.out = (end - start + 1))
      interpolated_values[(start + 1):(end - 1)] <- interpolated_section[-c(1, length(interpolated_secti
    }
  }
```

```
  return(interpolated_values)
}
```

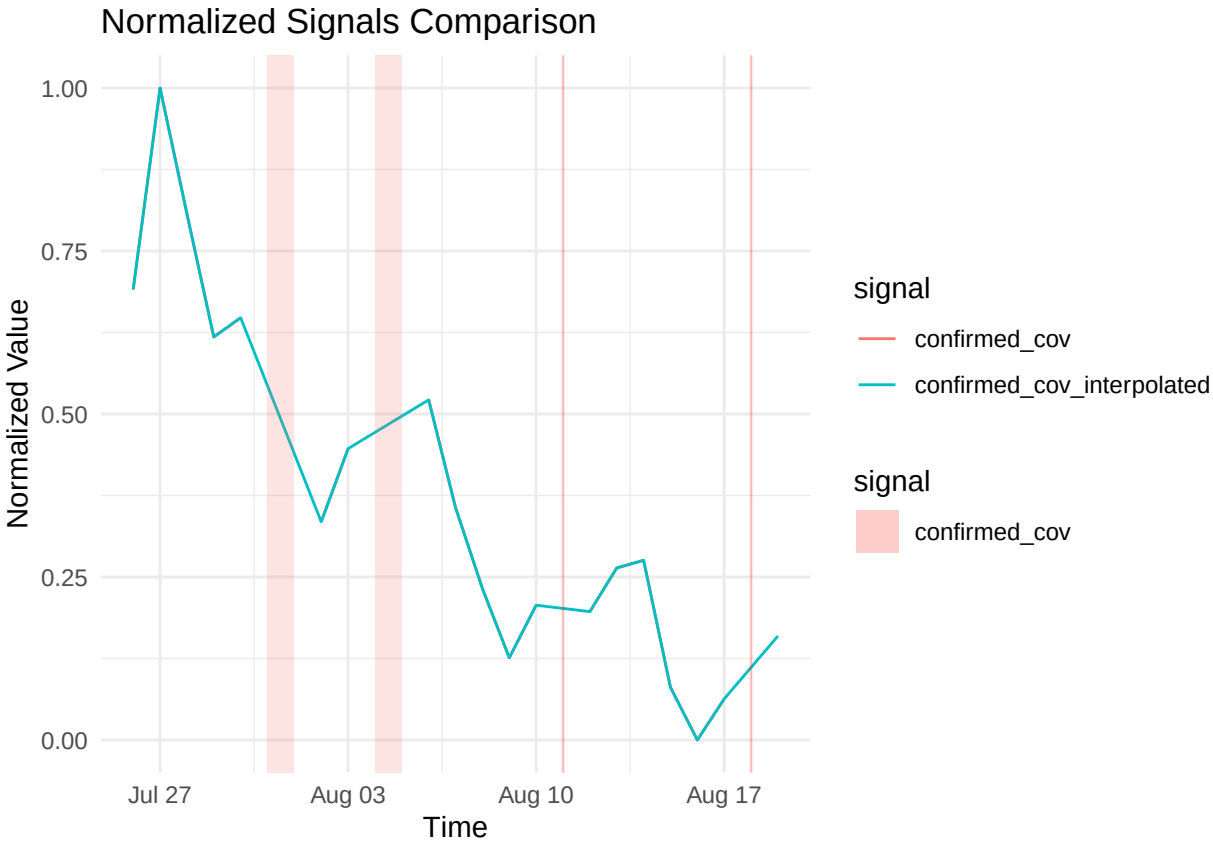
```
cov_nas_subset_linear_int <- cov_nas_subset
cov_nas_subset_linear_int$confirmed_cov_interpolated <- linear_interpolate_two_pass(cov_nas_subset$confi
cov_nas_subset_linear_int
```

```
## # A tibble: 25 x 4
##    time_value geo_value confirmed_cov confirmed_cov_interpolated
##    <date>     <chr>             <dbl>                      <dbl>
##  1 2020-07-26 tx                  885                        885
##  2 2020-07-27 tx                 1042                       1042
##  3 2020-07-28 tx                  944                        944
##  4 2020-07-29 tx                  848                        848
##  5 2020-07-30 tx                  863                        863
##  6 2020-07-31 tx                   NA                        810
##  7 2020-08-01 tx                   NA                        757
##  8 2020-08-02 tx                  704                        704
##  9 2020-08-03 tx                  761                        761
## 10 2020-08-04 tx                   NA                        774.
## # i 15 more rows
```

We can also plot it so you can see how the NAs were filled in. The red background is where the values were NA previously prior to the correction.

```
plot_signals(cov_nas_subset_linear_int)
```

```
## Signal: confirmed_cov
##   Gap: 2020-07-31 to 2020-08-01
##   Gap: 2020-08-04 to 2020-08-05
##   Gap: 2020-08-11 to 2020-08-11
##   Gap: 2020-08-18 to 2020-08-18
## Signal: confirmed_cov_interpolated
##   No gaps (no NAs found).
```




```
## [1] 4
```

As we discussed earlier. These approaches can also be done in version rather than in time. We will show how to do the LOCF method in version to get a sense of how this works. The benefit of doing imputation in version is that we can fill in larger gaps of data that might exist in the time domain, but don't necessarily exist in the version domain.

Prior to this section, we were just looking at one snapshot of a signal (as an `epi_df`). But the beauty of the `epi_archive` data structure is that we have a history of snapshots. In the context of `epi_archive`s, latency is an important concept relating to the delay in reporting of the data generating process.

Let's consider that we have two signals $x$ and $y$, one with a higher latency than the other. To simplify things let's only observe these signals at a time $t = 9$. On a table it may look something like this

| geo | $t$ | version | $x$ |
|---|---|---|---|
| CA | 9 | 10 | 14 |
| CA | 9 | 15 | 18 |
| CA | 9 | 20 | 19 |

| geo | $t$ | version | $y$ |
|---|---|---|---|
| CA | 9 | 10 | 3 |
| CA | 9 | 12 | 6 |
| CA | 9 | 15 | 7 |
| CA | 9 | 18 | 9 |
| CA | 9 | 20 | 9 |

Then when we call `epix_merge()` on these two tables we would get the following table

| geo | $t$ | version | $x$ | $y$ |
|---|---|---|---|---|
| CA | 9 | 10 | 14 | 3 |
| CA | 9 | 12 | NA | 6 |
| CA | 9 | 15 | 18 | 7 |
| CA | 9 | 18 | NA | 9 |
| CA | 9 | 20 | 19 | 9 |

And now we get these gaps in version, because of these differences in latency frequencies. However, in version it's almost always the case that our values are between the two end points. So in this case $x_9$ at version 12 will be between 14 and 18.

Now we can use the same imputation methods we used when imputing in time but for imputing in version. Linear interpolation or a moving average could be used here to cover up the NA gaps in a more continuous nature to help cover up the discretezation that real world data reporting does to a continuous data generating process. However, to avoid over complicating a rather simple problem and sticking to Occam's razor, it's most common that we use simple LOCF here and use the most recent version available in place of the NA.

Doing LOCF here would result in the new column $x_{\text{LOCF}}$

| geo | $t$ | version | $x$ | $x_{\text{LOCF}}$ | $y$ |
|---|---|---|---|---|---|
| CA | 9 | 10 | 14 | 14 | 3 |
| CA | 9 | 12 | NA | **14** | 6 |
| CA | 9 | 15 | 18 | 18 | 7 |
| CA | 9 | 18 | NA | **18** | 9 |
| CA | 9 | 20 | 19 | 19 | 9 |

Here's how you would do this in practice. As seen on the `epix_merge()` documentation it has the parameter `sync`. Here to do this LOCF in version we would set `sync = locf`.

```
example_archive_x <- tribble(
  ~geo_value, ~time_value, ~version, ~x,
  "CA", "2000-01-09", "2000-01-10", 14,
  "CA", "2000-01-09", "2000-01-15", 18,
  "CA", "2000-01-09", "2000-01-20", 19
) %>%
  mutate(
    time_value = as.Date(time_value),
    version = as.Date(version)
  ) %>%
  as_epi_archive(compactify = FALSE)

example_archive_y <- tribble(
  ~geo_value, ~time_value, ~version, ~y,
  "CA", "2000-01-09", "2000-01-10", 3,
  "CA", "2000-01-09", "2000-01-12", 6,
  "CA", "2000-01-09", "2000-01-15", 7,
  "CA", "2000-01-09", "2000-01-18", 9,
  "CA", "2000-01-09", "2000-01-20", 9
) %>%
  mutate(
```

```
    time_value = as.Date(time_value),
    version = as.Date(version)
  ) %>%
  as_epi_archive(compactify = FALSE)
```

```
epix_merge(example_archive_x, example_archive_y, sync = "locf")
```

```
## > An `epi_archive` object, with metadata:
## i Min/max time values: 2000-01-09 / 2000-01-09
## i First/last version with update: 2000-01-10 / 2000-01-20
## i Versions end: 2000-01-20
## i A preview of the table (5 rows x 5 columns):
##    geo_value time_value    version  x y
## 1:        CA 2000-01-09 2000-01-10 14 3
## 2:        CA 2000-01-09 2000-01-12 14 6
## 3:        CA 2000-01-09 2000-01-15 18 7
## 4:        CA 2000-01-09 2000-01-18 18 9
## 5:        CA 2000-01-09 2000-01-20 19 9
```