COVID-19 Data Report

A. Lenters

Overview

Welcome to my analysis of the Johns Hopkins COVID-19 dataset.[1] In this report, I will build models to predict the number of new deaths attributed to COVID-19 in a particular country/region, based on new case counts from earlier weeks. After tidying the data, I will be using tidyverse's many-models paradigm to easily fit one model per location.

Contents

Note to peer graders

You can find my data source overview in Wrangle Data, my missing data check in Data Summary, my plots in Exploratory Data Analysis/Modelling, my model fits in Modelling, and my discussion of bias in Conclusion.

[1] Sourced from: https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data

Setup

Setup ▶ Packages

I have not used any unusual packages in this analysis, but users may still want to upgrade to the latest versions of `tidyverse` and `knitr` for optimal performance, by running `install.packages(c("tidyverse", "knitr"))` in the console.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(knitr)

set.seed(1)

options(
  "pillar.print_min" = 4,
  "pillar.print_max" = 4
)
```

Setup ▶ User definitions

Here, users of this report can set the locations and backup locations of the files used in this analysis. The URLs that have been provided are publicly accessible, so no changes should be necessary.

```
global_cases_url <- paste0("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/",
                           "master/csse_covid_19_data/csse_covid_19_time_series/",
                           "time_series_covid19_confirmed_global.csv")

global_cases_url_bak <- paste0("https://raw.githubusercontent.com/alenters/dsaaf-cvd/",
                               "main/backup_data/",
                               "time_series_covid19_confirmed_global.csv")

global_deaths_url <- paste0("https://raw.githubusercontent.com/CSSEGISandData/",
                            "COVID-19/master/csse_covid_19_data/",
                            "csse_covid_19_time_series/",
                            "time_series_covid19_deaths_global.csv")

global_deaths_url_bak <- paste0("https://raw.githubusercontent.com/alenters/",
                                "dsaaf-cvd/main/backup_data/",
                                "time_series_covid19_deaths_global.csv")

global_pop_url <- paste0("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/",
                         "master/csse_covid_19_data/UID_ISO_FIPS_LookUp_Table.csv")

global_pop_url_bak <- paste0("https://raw.githubusercontent.com/alenters/dsaaf-cvd/",
```

```
                              "main/backup_data/UID_ISO_FIPS_LookUp_Table.csv")

us_cases_url <- paste0("https://github.com/CSSEGISandData/COVID-19/raw/master/",
                       "csse_covid_19_data/csse_covid_19_time_series/",
                       "time_series_covid19_confirmed_US.csv")

us_cases_url_bak <- paste0("https://github.com/alenters/dsaaf-cvd/raw/main/",
                           "backup_data/time_series_covid19_confirmed_US.csv")

us_deaths_url <- paste0("https://github.com/CSSEGISandData/COVID-19/raw/master/",
                        "csse_covid_19_data/csse_covid_19_time_series/",
                        "time_series_covid19_deaths_US.csv")

us_deaths_url_bak <- paste0("https://github.com/alenters/dsaaf-cvd/raw/main/",
                            "backup_data/time_series_covid19_deaths_US.csv")
```

Setup ▶ Fixed definitions

Below are constants I use in my analysis, which you can optionally adjust. But the document should knit on any machine with the values I have set.

```
data_format_str <- "\\d{1,2}\\/\\d{1,2}\\/\\d{2}"
sample_size <- 16
min_datapoints <- 100
train_end_date <- ymd("2022-01-01")
colors <- c("actuals" = "#88cccc", "predictions" = "red")

suggested_theme_base_size <- switch (
    knitr::pandoc_to(),
    latex = 9,
    html = 11,
    revealjs = 14,
    10 # conservative default
)
```

Verbs

All verbs used in the analysis will be created in this section. I have given them self-explanatory names, so you can skip this section and proceed to Wrangle Data, and return here to look at details as necessary.

Verbs ▶ Import & Tidy

```
get_file <- function(fn, url, url_bak) {
  tryCatch(fn(url),
    error = function(w) {
      warning(rlang::englue(
        "Could not read file from {url}. Reverting to backup @ {url_bak}."
      ))
      fn(url_bak)
    }
  )
}
```

```
read_us_cases <- function(url) {
  read_csv(url,
    col_types = cols(
      UID = col_skip(),
      iso2 = col_skip(),
      iso3 = col_skip(),
      code3 = col_skip(),
      FIPS = col_skip(),
      Admin2 = col_skip(),
      Province_State = col_character(),
      Country_Region = col_character(),
      Lat = col_skip(),
      Long_ = col_skip(),
      Combined_Key = col_skip(),
      .default = col_integer()
    )
  ) %>%
    pivot_longer(
      cols = matches(data_format_str),
      names_to = "date",
      values_to = "cases_county_level"
    ) %>%
    mutate(date = mdy(date)) %>%
    unite(combined_key, Province_State:Country_Region,
      sep = ", ",
      na.rm = TRUE
    ) %>%
    mutate(combined_key = as.factor(combined_key)) %>%
    group_by(combined_key, date) %>%
    summarize(
      cases = sum(cases_county_level),
      .groups = "drop"
    ) %>%
    select(combined_key, date, cases)
}
```

```
get_us_cases <- function() {
  get_file(read_us_cases, us_cases_url, us_cases_url_bak)
}
```

```
read_us_deaths <- function(url) {
  read_csv(url,
    col_types = cols(
      UID = col_skip(),
      iso2 = col_skip(),
      iso3 = col_skip(),
      code3 = col_skip(),
      FIPS = col_skip(),
      Admin2 = col_skip(),
      Province_State = col_character(),
      Country_Region = col_character(),
      Lat = col_skip(),
      Long_ = col_skip(),
      Combined_Key = col_skip(),
      Population = col_integer(),
      .default = col_integer()
    )
  ) %>%
    pivot_longer(
      cols = matches(data_format_str),
      names_to = "date",
      values_to = "deaths_county_level"
    ) %>%
    mutate(date = mdy(date)) %>%
    unite(combined_key, Province_State:Country_Region,
      sep = ", ",
      na.rm = TRUE
    ) %>%
    mutate(combined_key = as.factor(combined_key)) %>%
    group_by(combined_key, date) %>%
    summarize(
      deaths = sum(deaths_county_level),
      population = sum(Population),
      .groups = "drop"
    ) %>%
    select(combined_key, date, deaths, population)
}

get_us_deaths <- function() {
  get_file(read_us_deaths, us_deaths_url, us_deaths_url_bak)
}
```

```
read_global_cases <- function(url) {
  read_csv(url,
    col_types = cols(
      `Province/State` = col_character(),
      `Country/Region` = col_character(),
      Lat = col_skip(),
      Long = col_skip(),
      .default = col_integer()
```

```
    )
  ) %>%
    pivot_longer(
      cols = matches(data_format_str),
      names_to = "date",
      values_to = "cases"
    ) %>%
    mutate(date = mdy(date)) %>%
    unite(combined_key, `Province/State`:`Country/Region`,
      sep = ", ",
      na.rm = TRUE
    ) %>%
    select(combined_key, date, cases)
}

get_global_cases <- function() {
  get_file(read_global_cases, global_cases_url, global_cases_url_bak)
}
```

```
read_global_deaths <- function(url) {
  read_csv(url,
    col_types = cols(
      `Province/State` = col_character(),
      `Country/Region` = col_character(),
      Lat = col_skip(),
      Long = col_skip(),
      .default = col_integer()
    )
  ) %>%
    pivot_longer(
      cols = matches(data_format_str),
      names_to = "date",
      values_to = "deaths"
    ) %>%
    mutate(date = mdy(date)) %>%
    unite(combined_key, `Province/State`:`Country/Region`,
      sep = ", ",
      na.rm = TRUE
    ) %>%
    select(combined_key, date, deaths)
}

get_global_deaths <- function() {
  get_file(read_global_deaths, global_deaths_url, global_deaths_url_bak)
}
```

```
read_global_population <- function(url) {
  read_csv(url,
    col_types = cols_only(
      Combined_Key = col_character(),
      Population = col_integer()
    )
  ) %>%
    rename(
```

```
      combined_key = Combined_Key,
      population = Population
    ) %>%
    select(combined_key, population)
}

get_global_population <- function() {
  get_file(read_global_population, global_pop_url, global_pop_url_bak)
}
```

```
has_na <- function(df) {
  sum(is.na(df)) > 0
}

count_duplicates <- function(df) {
  df %>%
    tally(name = "copy_count") %>%
    group_by(copy_count) %>%
    tally()
}
```

Verbs ▶ Transform

```
roll_up <- function(df, ..., date_unit = "week") {
  df %>%
    mutate(date = floor_date(date, unit = date_unit)) %>%
    group_by(pick(where(~ !is.numeric(.x)))) %>%
    summarize(..., .groups = "drop")
}
```

```
add_lags <- function(df, .col, .group_by, .order_by, ...,
                     lags = 0:9,
                     prefix = "lag_") {
  lag_tibble <- lags %>%
    set_names(map(lags, ~ paste0(prefix, .x))) %>%
    as_tibble_row()
  list_cbind(list(df, lag_tibble)) %>%
    group_by(pick({{ .group_by }})) %>%
    arrange(pick({{ .order_by }})) %>%
    mutate(across(
      starts_with(prefix), ~ lag({{ .col }}, n = .x[[1]], default = 0)
    )) %>%
    ungroup()
}
```

```
add_difference_col <- function(df, .col, new_col_name, .group_by, .order_by) {
  df %>%
    group_by(pick({{ .group_by }})) %>%
    arrange(pick({{ .order_by }}), .by_group = TRUE) %>%
    mutate({{ new_col_name }} := {{ .col }} - lag({{ .col }}, default = 0)) %>%
    ungroup()
}
```

Verbs ▶ Model

```
fit_model_per_location <- function(df, .group_by, model_function) {
  df %>%
    group_by(pick({{ .group_by }})) %>%
    nest() %>%
    mutate(model = map(data, model_function))
}
```

```
get_model_pred_info <- function(fit_nested_df) {
  fit_nested_df %>%
    mutate(
      fits = map(model, broom::glance),
      pred = map2(model, data, ~ predict(.x, .y, type = "response"))
    ) %>%
    unnest(c(fits, data, pred)) %>%
    mutate(
      resid = pred - new_deaths,
      gof = 1 - deviance / null.deviance
    )
}
```

```
get_model_fit_info <- function(fit_nested_df) {
  fit_nested_df %>%
    mutate(coeffs = map(model, broom::tidy)) %>%
    unnest(c(coeffs)) %>%
    mutate(sig_label = case_when(
      p.value <= 0.001 ~ "***",
      p.value <= 0.01 ~ "**",
      p.value <= 0.05 ~ "*",
      p.value <= 0.1 ~ ".",
      .default = ""
    ))
}
```

Verbs ▶ Communicate

```
present_table <- function(df, ..., n = 4, caption = "") {
  knitr::kable(head(df,
                    n = n),
               align = "l",
               caption = caption)
}
```

```
plot_model_eval <- function(.model) {
  par(mfrow = c(2, 2))
  plot(.model)
}
```

```
plot_model_coeffs <- function(df, .location_col, .gof_col, ...,
                              show_text = TRUE,
                              theme_base_size = suggested_theme_base_size) {
  ret <- df %>%
    ggplot(aes(x = term,
               y = reorder({{ .location_col }}, {{ .gof_col }}),
```

```
                fill = estimate)) +
    geom_tile() +
    geom_text(aes(label = sig_label), color = "#333333") +
    scale_x_discrete(guide = guide_axis(angle = 90), position = "top") +
    scale_fill_gradient2(low = "blue", mid = "white", high = "red", midpoint = 0) +
    theme_minimal(base_size = theme_base_size) +
    theme(legend.position = "bottom") +
    labs(
      title = "Looking for patterns in model coefficients and p-values",
      x = "Model feature",
      y = "Model",
      fill = "Feature coefficient",
      caption = "Feature p-value: < 0.001 = ***\n< 0.01 = **\n< 0.05 = *\n< 0.1 = ."
    )

  if (!show_text) {
    ret <- ret +
      theme(axis.text.y = element_blank())
  }
  ret
}
```

Wrangle Data

I will now use the verbs I created in Verbs to import, tidy, transform and visualize the data, in preparation for modelling.

Wrangle ▶ Data Source

The data we were asked to analyze for this assignment is a COVID-19 dataset from Johns Hopkins.[2] It contains 5 main files of interest:

- Global cases: `time_series_covid19_confirmed_global.csv`
- US cases: `time_series_covid19_confirmed_US.csv`
- Global deaths: `time_series_covid19_deaths_global.csv`
- US deaths: `time_series_covid19_deaths_US.csv`
- Global population: `UID_ISO_FIPS_LookUp_Table.csv`

This analysis was initially designed to be able to work with US and Global data simultaneously. However, the US files are significantly larger than the Global files, so, in order to speed up knitting time, I will use only the Global files. Additionally, I have no need for population data in the analysis I plan to run, so I will omit that as well.

The files I will use in this analysis are therefore:

- Global cases: `time_series_covid19_confirmed_global.csv`
- Global deaths: `time_series_covid19_deaths_global.csv`

These files collate daily counts of COVID-19 `cases` and `deaths` from governments and health authorities around the world. Although the raw data has daily granularity, I will be transforming it to weekly for use in my model.

Wrangle ▶ Import Global Cases

Let's import the global cases data. I will immediately roll it up to weekly granularity, to make it a more manageable size:

```
global_cases <- get_global_cases() %>%
  roll_up(
    cases = max(cases),
    date_unit = "week"
  )
```

We end up with a dataframe like this:

```
sample_n(global_cases, 4) %>%
  present_table(caption = "Four randomly-sampled rows from `global_cases`")
```

Table 1: Four randomly-sampled rows from `global_cases`

combined_key	date	cases
MS Zaandam	2022-04-03	9
Turks and Caicos Islands, United Kingdom	2020-03-29	6
Benin	2022-03-20	26952
Eritrea	2021-10-10	6773

Let's look at the data graphically:

[2]Sourced from: https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data

```
plot(global_cases)
```

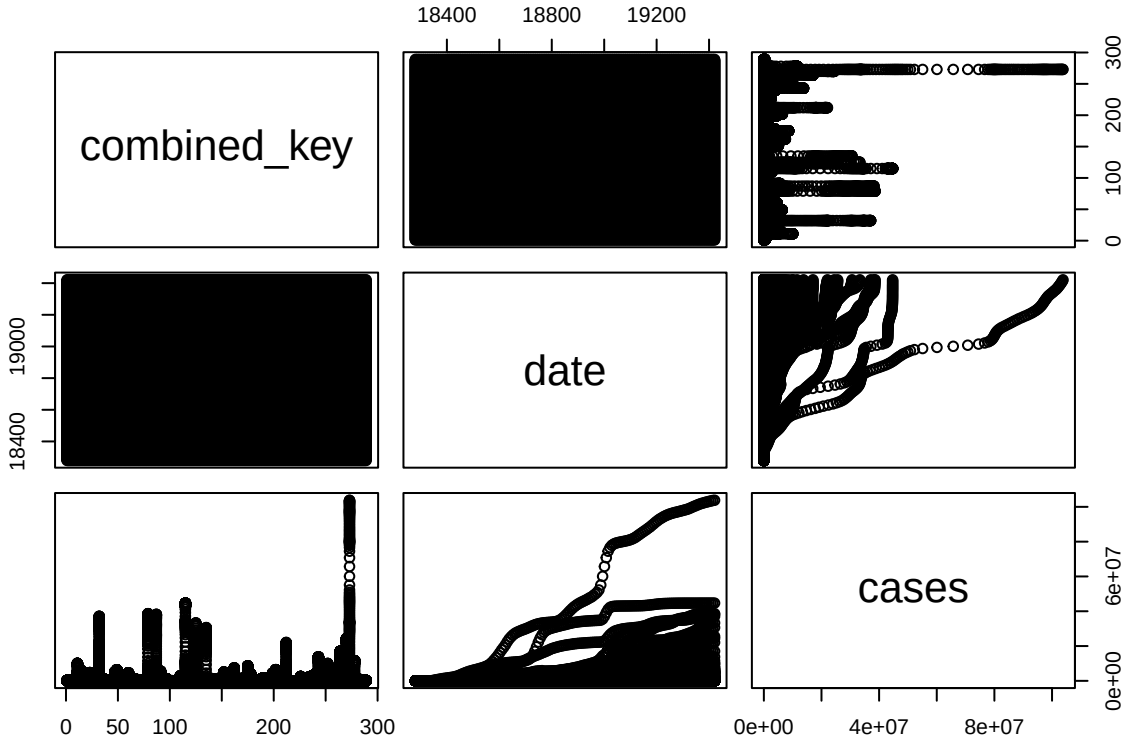

Figure 1: Output of `plot()` for `global_cases`.

Note that `cases` are monotonically increasing, i.e. the metric reported in the files is actually **cumulative** cases, which is why I used `max()` to perform the aggragation when rolling up from daily to weekly. I want to work with incremental cases, so I will need to address this.

Wrangle ▶ Import Global Deaths

Now, let's repeat the above process to import the global deaths data:

```
global_deaths <- get_global_deaths() %>%
  roll_up(
    deaths = max(deaths),
    date_unit = "week"
  )
```

We end up with a dataframe like this:

```
sample_n(global_deaths, 4) %>%
  present_table(caption = "Four randomly-sampled rows from `global_deaths`")
```

Table 2: Four randomly-sampled rows from `global_deaths`

combined_key	date	deaths
Maldives	2021-08-01	222
Peru	2022-10-30	217069
Gansu, China	2022-06-19	2
Congo (Brazzaville)	2020-08-02	58

Graphical view:

```
plot(global_deaths)
```

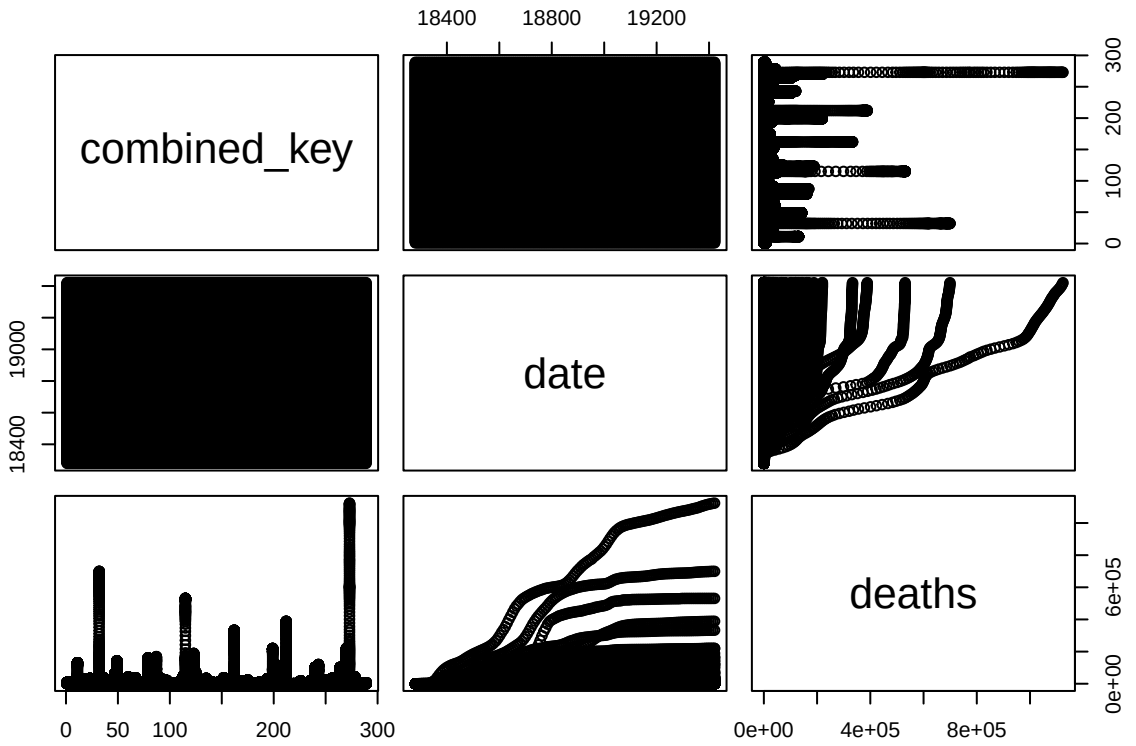

Figure 2: Output of `plot()` for `global_deaths`.

Global deaths are also cumulative.

Wrangle ▶ Tidy

Before doing any further work with the dataframes, I will check each one to ensure that it is tidy.

The three tidiness requirements:

For a tibble to be tidy, three requirements should be met:

1. Each variable has its own column
2. Each observation has its own row
3. Each value has its own cell

The book *R For Data Science* notes[3] that if two of these are true the third is also guaranteed to be true, so I will check the first two.

Tidy ▶ Does each variable have its own column?

Short answer: **yes**.

This was handled during import, by pivoting all of the date columns (that were making each file extremely wide) into a `location, date, value` format that stores the timeseries of cases and deaths for each location.

[3] https://r4ds.had.co.nz/tidy-data.html

```
glimpse(global_cases)
```

```
## Rows: 47,396
## Columns: 3
## $ combined_key <chr> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan~
## $ date         <date> 2020-01-19, 2020-01-26, 2020-02-02, 2020-02-09, 2020-02-~
## $ cases        <int> 0, 0, 0, 0, 0, 5, 8, 14, 26, 106, 270, 521, 908, 1330, 24~
```

```
glimpse(global_deaths)
```

```
## Rows: 47,396
## Columns: 3
## $ combined_key <chr> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan~
## $ date         <date> 2020-01-19, 2020-01-26, 2020-02-02, 2020-02-09, 2020-02-~
## $ deaths       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 5, 15, 30, 43, 72, 105, 168~
```

No variable is spread across multiple columns, and no column contains multiple variables, so this requirement is met.

Tidy ▶ Does each observation have its own row?

Short answer: **yes**.

The main thing to look for here is duplicate observations in a single location/date. I created a `count_duplicates()` verb to perform the duplicate-checking. Let's use `purrr::map()` to check all dataframes simultaneously, and `purrr::list_rbind()` to return the results in a dataframe.

```
list(
  global_cases = global_cases %>% group_by(combined_key, date),
  global_deaths = global_deaths %>% group_by(combined_key, date)
) %>%
  map(count_duplicates) %>%
  list_rbind(names_to = "dataframe") %>%
  present_table(n = Inf,
                caption = "Confirming that no value of `copy_count` exceeds 1.")
```

Table 3: Confirming that no value of `copy_count` exceeds 1.

dataframe	copy_count	n
global_cases	1	47396
global_deaths	1	47396

If there were duplicates in any of the dataframes, `count_duplicates()` would have reported values `> 1` in `copy_count`. Therefore, there are no duplicates in the data.

We have confirmed that at least two of the three tidiness requirements are met, so the data is tidy.

Wrangle ▶ Join Global dataframes

Let's join `cases` and `deaths` information together to make a single dataframe, and then add some new columns that I will need for modelling. This would arguably be even tidier than having the information spread across two different dataframes.

`global_cases` and `global_deaths` should contain exactly the same rows, differing only on whether they contain a `cases` or `deaths` column. Therefore, I will join them using `inner_join()`, with `relationship =`

"one-to-one" and unmatched = "error". This will cause the code to halt if either of the dataframes is missing row, relative to the other.

```
global <- global_cases %>%
  inner_join(global_deaths,
    by = join_by(combined_key, date),
    relationship = "one-to-one",
    unmatched = "error")
```

There were no errors, so the inner join worked perfectly, and we now have a single dataframe called global to use in the rest of the analysis.

Data Summary

Prior to modelling, I want to convert the `cases` and `deaths` columns from cumulative to incremental, and I want to add lag columns. Before doing any of this, it would be a good idea to check for missing values and fix any we find. Missing values will affect plots and calculations.

```
global %>%
  summary()
```

```
##  combined_key            date                cases                deaths
##  Length:47396       Min.   :2020-01-19   Min.   :        0   Min.   :      0
##  Class :character   1st Qu.:2020-10-30   1st Qu.:      688   1st Qu.:      3
##  Mode  :character   Median :2021-08-11   Median :    14599   Median :    152
##                     Mean   :2021-08-11   Mean   :   965616   Mean   :  13428
##                     3rd Qu.:2022-05-23   3rd Qu.:   229892   3rd Qu.:   3052
##                     Max.   :2023-03-05   Max.   :103802702   Max.   :1123836
```

According to the summary, there are no missing values in the `global` dataframe. (They would have their own row in the summary if they existed.) We can run this check to throw an error if missing values creep into the data in the future.

```
if(global %>% has_na()) {
  stop("NA values found in dataset. Please fix.")
}
```

Transform

Now that the raw data is as tidy as can be, let's move on to transforming it to suit our modelling needs.

Transform ▶ Cumulative to incremental

We saw earlier that the `cases` and `deaths` data provided is cumulative, but for my analysis, I would like to work with incremental values. I will use a differencing transform to convert from cumulative to incremental.

I will create two incremental columns called `new_cases` and `new_deaths`, using the `add_difference_col()` verb, which internally makes use of `dplyr`'s `lag()` function.

```
global <- global %>%
  add_difference_col(cases, "new_cases", combined_key, date) %>%
  add_difference_col(deaths, "new_deaths", combined_key, date)
```

Let's pick a random location and view the results for it:

```
random_location <- global %>%
  filter(deaths > 0) %>%
  select(combined_key) %>%
  sample_n(1) %>%
  deframe()
```

```
global %>%
  filter(combined_key == random_location) %>%
  filter(deaths > 0) %>%
  arrange(date) %>%
  present_table(n = 10,
                caption = paste("Post-differencing. Ten consecutive rows from",
                                "`global`, for a random location."))
```

Table 4: Post-differencing. Ten consecutive rows from `global`, for a random location.

combined_key	date	cases	deaths	new_cases	new_deaths
New Zealand	2020-03-29	950	1	499	1
New Zealand	2020-04-05	1312	4	362	3
New Zealand	2020-04-12	1422	12	110	8
New Zealand	2020-04-19	1470	18	48	6
New Zealand	2020-04-26	1487	20	17	2
New Zealand	2020-05-03	1494	21	7	1
New Zealand	2020-05-10	1499	22	5	1
New Zealand	2020-05-17	1504	22	5	0
New Zealand	2020-05-24	1504	23	0	1
New Zealand	2020-05-31	1504	23	0	0

Transform ▶ Add lags

The main features I want to use for predicting weekly `deaths` are the weekly `cases` from past weeks. These can be obtained using `dplyr`'s `lag()` function. I wrote the verb `add_lags()` to add multiple lag columns simultaneously. It takes care of things like grouping by location and sorting by date, to ensure that `lag()` obtains the correct numbers for each row in the dataframe.

```
global <- global %>%
  add_lags(new_cases, combined_key, date,
```

```
            lags = 0:8,
            prefix = "new_cases_lag_")
```

Let's view the result for the same random location as before, but with the table transposed to make it easier to read. I've also restricted the date range to March 2020, for the same reason.

```
global %>%
  filter(combined_key == random_location) %>%
  filter(date >= ymd("2020-03-01") & date < ymd("2020-04-01")) %>%
  arrange(date) %>%
  t() %>%
  present_table(n = Inf,
                caption = "Post-lagging, for a random location.")
```

Table 5: Post-lagging, for a random location.

combined_key	New Zealand	New Zealand	New Zealand	New Zealand	New Zealand
date	2020-03-01	2020-03-08	2020-03-15	2020-03-22	2020-03-29
cases	5	6	52	451	950
deaths	0	0	0	0	1
new_cases	4	1	46	399	499
new_deaths	0	0	0	0	1
new_cases_lag_0	4	1	46	399	499
new_cases_lag_1	1	4	1	46	399
new_cases_lag_2	0	1	4	1	46
new_cases_lag_3	0	0	1	4	1
new_cases_lag_4	0	0	0	1	4
new_cases_lag_5	0	0	0	0	1
new_cases_lag_6	0	0	0	0	0
new_cases_lag_7	0	0	0	0	0
new_cases_lag_8	0	0	0	0	0

As expected, lagging by `n` shifts values by `n` time periods (weeks).

Transform ▶ Missing value recheck

Let's rerun our check for missing values, to ensure that our transformations didn't insert missing values in any of the new columns.

```
if(global %>% has_na()) {
  stop("NA values found in dataset. Please fix.")
}
```

Exploratory Data Analysis

Let's visually check for a relationship between new_cases and new_deaths in various locations.

I will be plotting on a log-log scale, so I will filter for values `> 0`.

```
global %>%
  pivot_longer(starts_with("new_cases_lag_"),
    names_to = "lag_amount",
    values_to = "val"
  ) %>%
  filter(val > 0 & new_deaths > 0) %>%
  ggplot(aes(x = val, y = new_deaths)) +
  geom_point(aes(color = combined_key), alpha = 0.4) +
  geom_smooth(method = "lm", color = "black", alpha = 0.5) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10") +
  theme_minimal() +
  theme(legend.position = "none", aspect.ratio = 1) +
  facet_wrap(~lag_amount, ncol = 3) +
  labs(
    title = "Do previous weeks' case numbers predict deaths?",
    subtitle = "Visually, a 2-week lag looks like a sweet spot",
    x = "Lagged new cases (log scale)",
    y = "New deaths (log scale)"
  )
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

From a purely visual analysis, `log(new_deaths)` seems to have the best correlation with `log(new_cases)` when lagged by 0-2 weeks. Longer lags cause the correlation to weaken. But it would be interesting to try to formalize this relationship using a model.

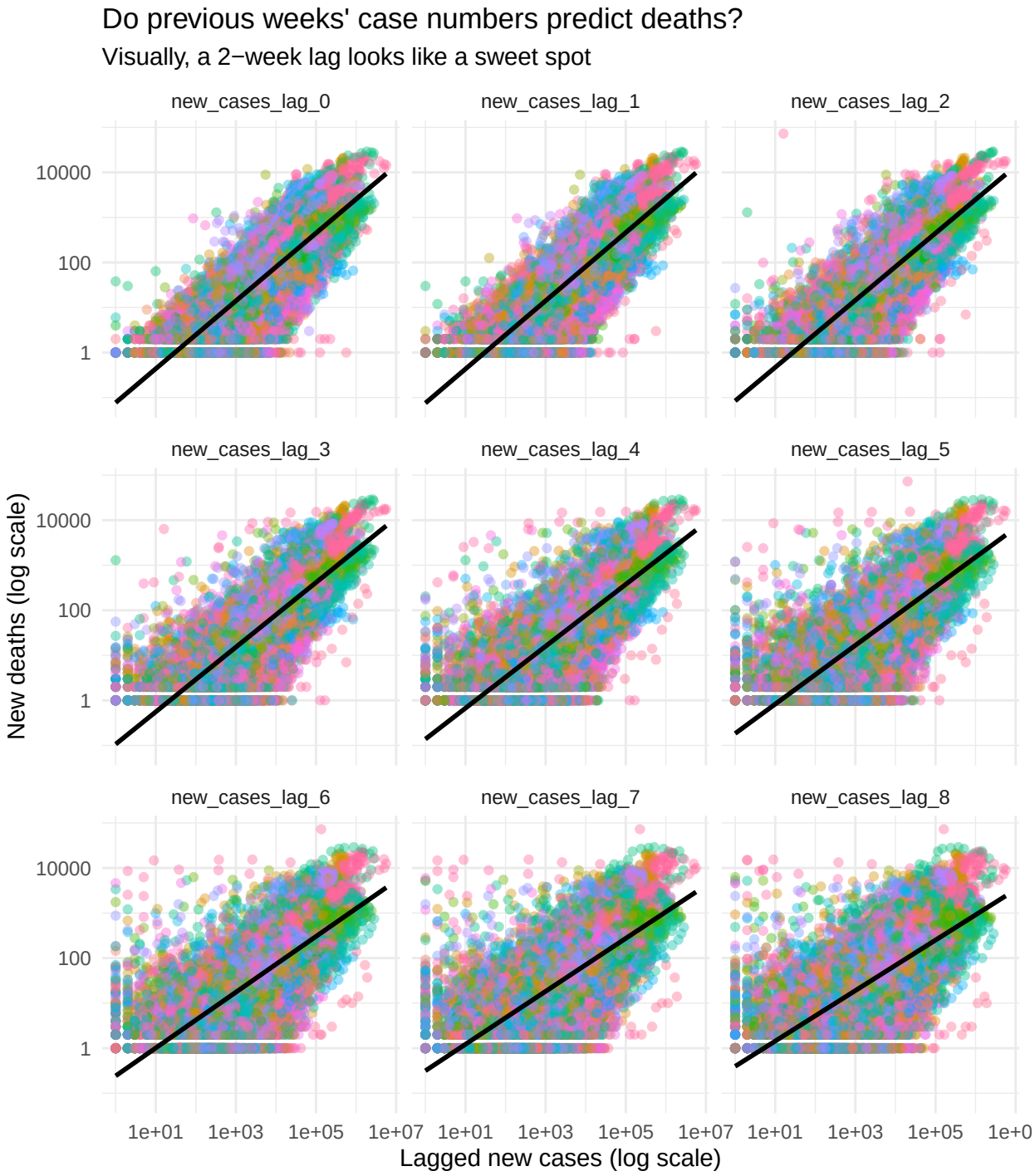

Figure 3: Relationship between weekly new cases and weekly new deaths, at various lags

Modelling

Model ▶ Filtering

Let's first apply a bit of filtering to `global`, to exclude any values that won't work in the models. I will exclude all rows with non-positive values, and I will also exclude locations that have fewer than 100 remaining points in the dataset.

(There are a few negative values in the dataframe due to the differencing transform, resulting from errors in the raw data where a location's timeseries doesn't increase monotonically like it should have. Originally, I was using convex optimization to smooth the curves out prior to differencing, but it was slow to run, so instead I am just excluding problematic rows.)

```
global_filtered <-
  global %>%
  filter(if_all(contains("deaths") | contains("cases"), ~ .x > 0))

has_enough_points <- global_filtered %>%
  count(combined_key) %>%
  filter(n >= min_datapoints) %>%
  select(combined_key)

global_filtered <- global_filtered %>%
  inner_join(has_enough_points, by = join_by(combined_key))
```

Model ▶ Sample locations

Let's randomly select a small number of locations to plot during model evaluation, to make the figures more manageable. Random selection (vs. hardcoding a list, picking the largest locations, etc.) prevents bias from sneaking into the analysis via my choice of locations.

```
locations <- global_filtered %>%
  select(combined_key) %>%
  distinct() %>%
  sample_n(sample_size)
```

Model ▶ Define model

This is an important step: deciding the form of the model and what features to put in it. I have gone with a poisson model since we are dealing with count data (and I wanted to learn how to use it!)

And because the various lag columns are probably highly correlated, and I want my model coefficients to be interpretable, I will select just a handful of them to use as features. My results will be only as good as my feature selection. Future work could include a more systematic way to engineer/select/decorrelate features, such as `PCA`.

Note that I restrict training data to dates before `train_end_date`, which was defined above as 2022-01-01.

```
mod <- function(df) {
  glm(new_deaths ~ log(new_cases_lag_0) +
                   log(new_cases_lag_2) +
                   log(new_cases_lag_4) +
                   log(new_cases_lag_6),
    data = df %>% filter(date < train_end_date),
    family = poisson(link = "log"),
    na.action = na.exclude
```

```
  )
}
```

Model ▶ Apply "many models" pattern to create per-location models

The "many models" pattern was taught in Chapter 25[4] of the first edition of *R For Data Science*. It allows us to fit many models, while remaining in the context of a dataframe (rather than, for example, breaking out a for loop.)

I wrote the verbs `fit_model_per_location()`, `get_model_pred_info()`, and `get_model_fit_info()` for:

1. performing the fitting,
2. calculating the predictions, residuals, and goodness of fit for each model,
3. extracting each model's coefficients and their significance values

```
(model_fits <- fit_model_per_location(global_filtered, combined_key, mod))
```

```
## # A tibble: 109 x 3
## # Groups:   combined_key [109]
##   combined_key data                model
##   <chr>        <list>              <list>
## 1 Japan        <tibble [156 x 14]> <glm>
## 2 Korea, South <tibble [155 x 14]> <glm>
## 3 US           <tibble [156 x 14]> <glm>
## 4 Thailand     <tibble [114 x 14]> <glm>
## # i 105 more rows
```

```
(model_preds <- get_model_pred_info(model_fits))
```

```
## # A tibble: 14,251 x 27
## # Groups:   combined_key [109]
##   combined_key date       cases deaths new_cases new_deaths new_cases_lag_0
##   <chr>        <date>     <int>  <int>     <int>      <int>           <int>
## 1 Japan        2020-03-15  1059     36       264         14             264
## 2 Japan        2020-03-22  1728     56       669         20             669
## 3 Japan        2020-03-29  3525     87      1797         31            1797
## 4 Japan        2020-04-05  6951    138      3426         51            3426
## # i 14,247 more rows
## # i 20 more variables: new_cases_lag_1 <int>, new_cases_lag_2 <int>,
## #   new_cases_lag_3 <int>, new_cases_lag_4 <int>, new_cases_lag_5 <int>,
## #   new_cases_lag_6 <int>, new_cases_lag_7 <int>, new_cases_lag_8 <int>,
## #   model <list>, null.deviance <dbl>, df.null <int>, logLik <dbl>, AIC <dbl>,
## #   BIC <dbl>, deviance <dbl>, df.residual <int>, nobs <int>, pred <dbl>,
## #   resid <dbl>, gof <dbl>
```

```
(model_fit_info <- get_model_fit_info(model_fits))
```

```
## # A tibble: 545 x 9
## # Groups:   combined_key [109]
##   combined_key data     model  term        estimate std.error statistic  p.value
##   <chr>        <list>   <list> <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 Japan        <tibble> <glm>  (Intercept)   0.221     0.0665      3.32 8.87e- 4
## 2 Japan        <tibble> <glm>  log(new_ca~  -0.0539    0.0197     -2.74 6.09e- 3
## 3 Japan        <tibble> <glm>  log(new_ca~   0.395     0.0399      9.90 4.17e-23
## 4 Japan        <tibble> <glm>  log(new_ca~   0.224     0.0394      5.68 1.32e- 8
```

[4] https://r4ds.had.co.nz/many-models.html

```
## # i 541 more rows
## # i 1 more variable: sig_label <chr>
```

Model ▶ Goodness of Fit

I will use Cohen's[5] R_L^2 to measure goodness of fit for my poisson model. It is bounded between 0 and 1, similar to R^2. Let's plot a histogram of this value for each of the many models we fit:

```
model_preds %>%
  ggplot(aes(x = gof)) +
  geom_histogram() +
  theme_minimal(base_size = suggested_theme_base_size) +
  labs(
    title = "Model fits well in most locations",
    x = "Goodness of fit",
    y = "Model count"
  )
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

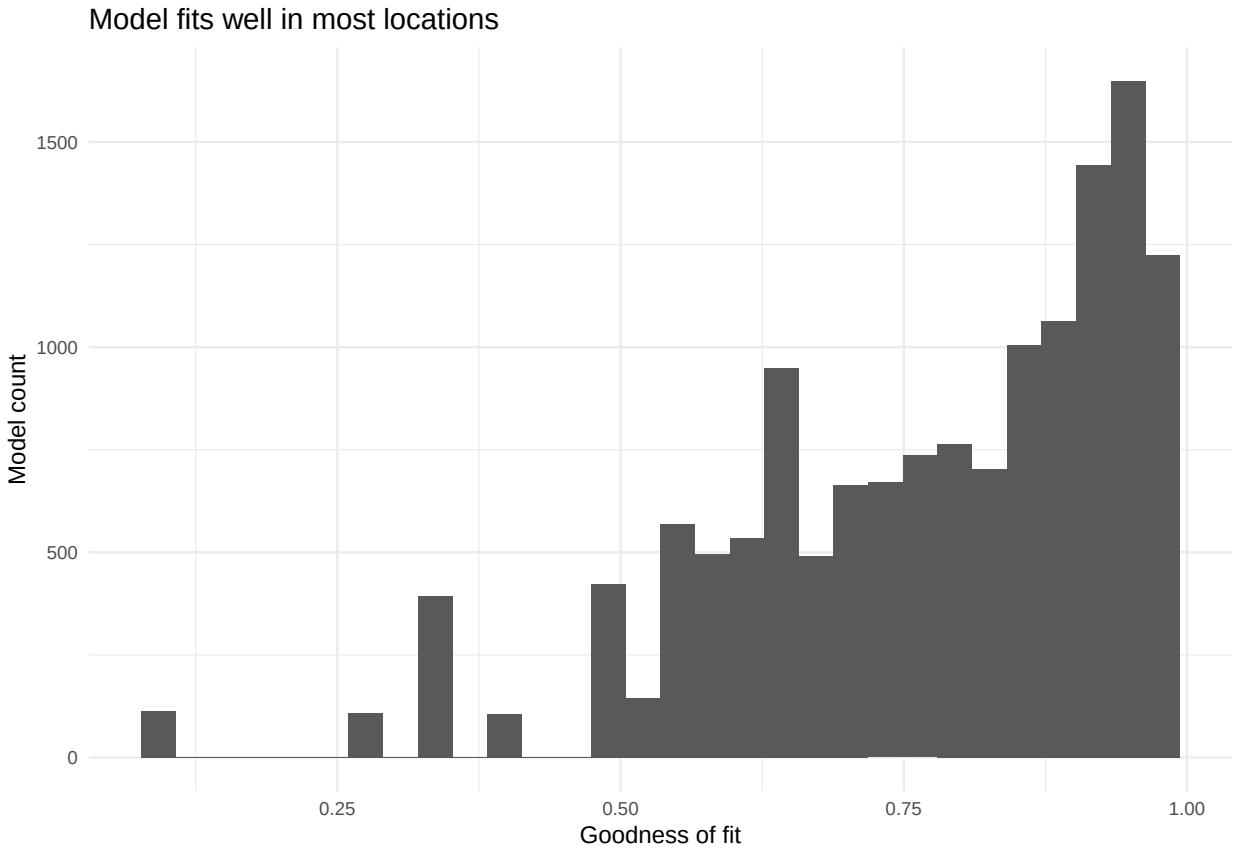

Figure 4: Goodness of fit for the many models.

The histogram shows that most models have a decently high goodness of fit.

[5]https://en.wikipedia.org/wiki/Pseudo-R-squared

Model ▶ Model coeffs

I was curious to know if there is any pattern across locations in terms of which `cases` lags the model relies on to make predictions (2 weeks ago, 4 weeks ago, etc). We can use a heatmap to look for patterns. Because there are so many locations in the data, I'll first sample a handful, and then show the entire set (which will be unreadable, but will give an overall sense of the pattern).

The number of *s indicate significance level (more is better).

```
model_fit_info %>%
  inner_join(locations,
             by = join_by(combined_key)) %>%
  inner_join(model_preds %>%
               select(combined_key, gof) %>%
               distinct(),
             by = join_by(combined_key)) %>%
  filter(term != "(Intercept)") %>%
  plot_model_coeffs(combined_key, gof)
```

```
model_fit_info %>%
  inner_join(model_preds %>%
               select(combined_key, gof) %>%
               distinct(),
             by = join_by(combined_key)) %>%
  filter(term != "(Intercept)") %>%
  plot_model_coeffs(combined_key, gof, show_text = FALSE)
```

As suspected from our earlier visual analysis, across most of the models, the feature `log(new_cases_lag_2)` has the largest positive coefficient.

Model ▶ Predictive performance

Let's see if a model trained up to some cutoff date (2022-01-01) can make good future predictions about number of new deaths, given information about number of new cases in recent weeks.

```
n_cols <- switch (
    knitr::pandoc_to(),
    latex = 4,
    html = 3,
    3 # default
)

model_preds %>%
  inner_join(locations, by = join_by(combined_key)) %>%
  ggplot(aes(x = date)) +
  geom_line(aes(y = new_deaths, color = "actuals"),
            alpha = 0.8) +
  geom_line(aes(y = pred, color = "predictions"),
            linewidth = 0.7,
            alpha = 0.8) +
  geom_vline(
    xintercept = train_end_date,
    color = "orange",
    linetype = "dashed",
    linewidth = 1.2
  ) +
```

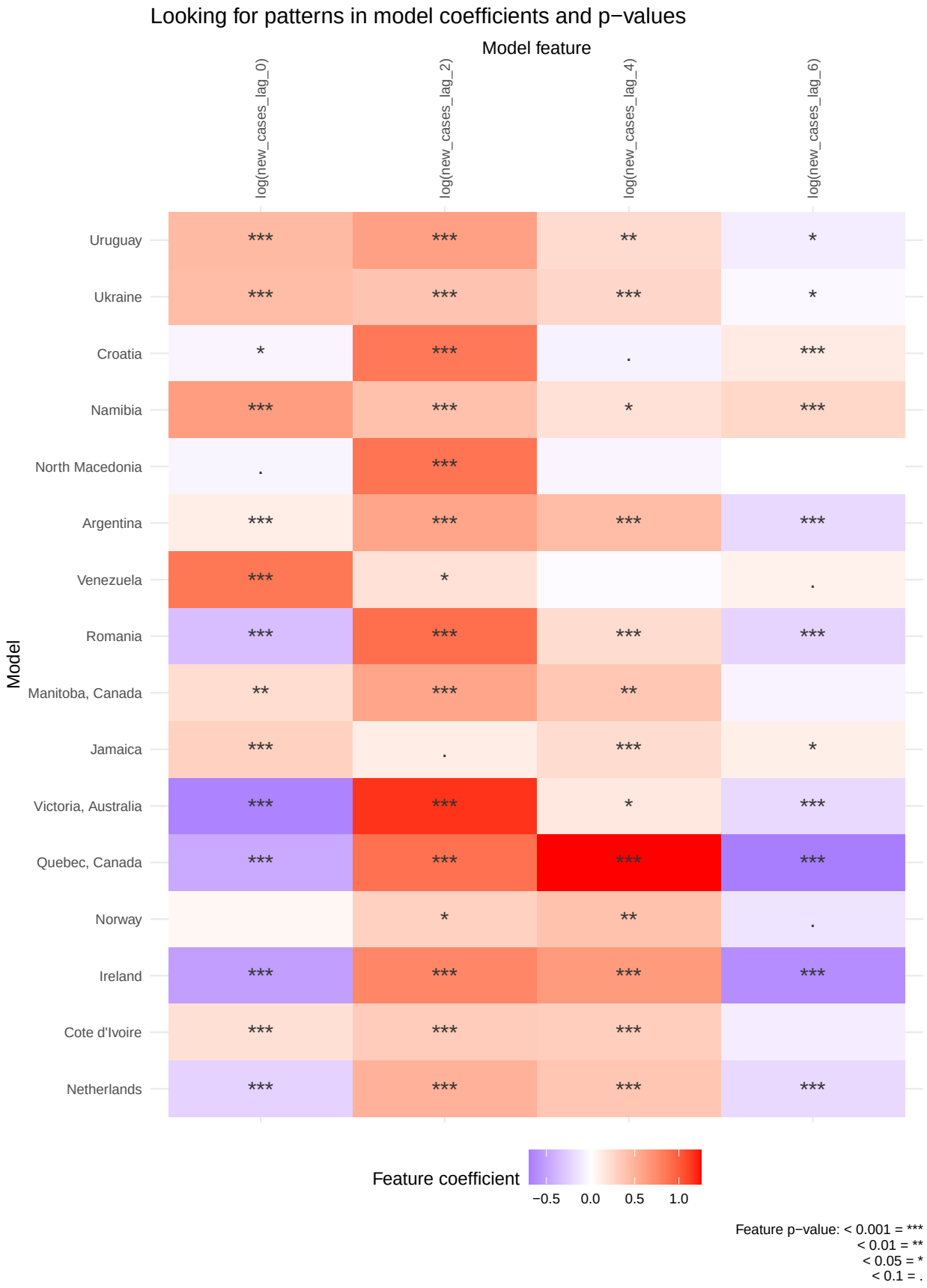

Figure 5: Model feature coefficients and significance, select locations.

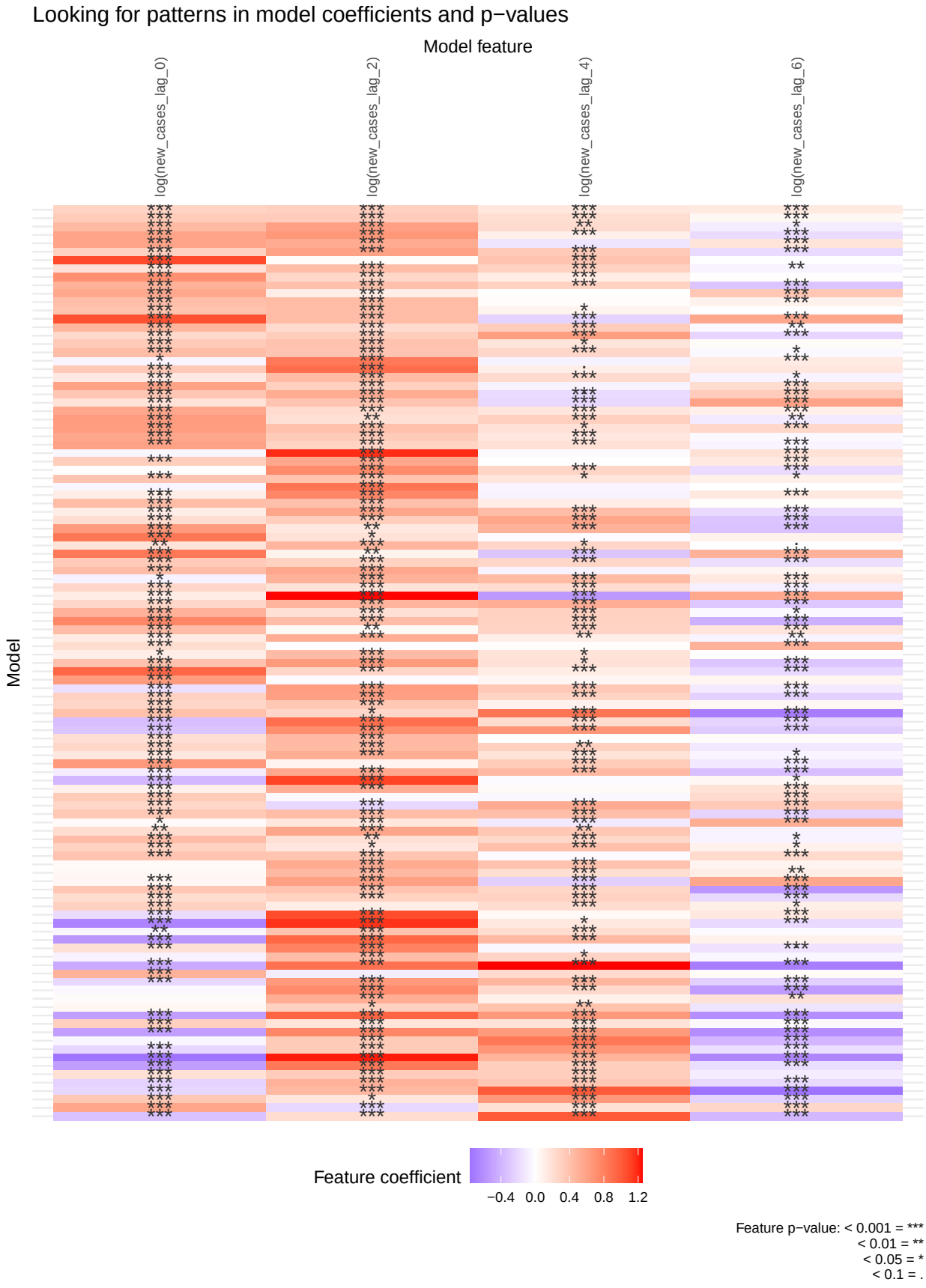

Figure 6: Model feature coefficients and significance, all locations.

```
  theme_minimal(base_size = suggested_theme_base_size) +
  theme(legend.position = "top") +
  theme(aspect.ratio = 1) +
  scale_color_manual(values = colors) +
  scale_x_date(breaks = as.Date(c("2021-01-01", "2023-01-01"))) +
  scale_y_continuous(trans = "log10") +
  facet_wrap(~combined_key, scales = "free", ncol = n_cols) +
  labs(
    title = "Predictive performance",
    subtitle = paste("Per-location models fit on data up to",
                     as.character(train_end_date)),
    caption = paste0("Prior to ", as.character(train_end_date),
                     ", red-teal correspondence shows models' capacity to fit ",
                     "the training data.\n After ", as.character(train_end_date),
                     ", red-teal correspondence shows models' performance on ",
                     "unseen data.\n Each model has the same form, so it is ",
                     "interesting to consider the performance\n similarities and ",
                     "differences between locations.", sep = ""),
    x = "Date",
    y = "Incremental deaths (actual or predicted)"
  )
```

This model evaluation could be formalized numerically, but as a starting place I like being able to see the models' performance graphically for a random sample of locations. See my takeaways in Conclusion.

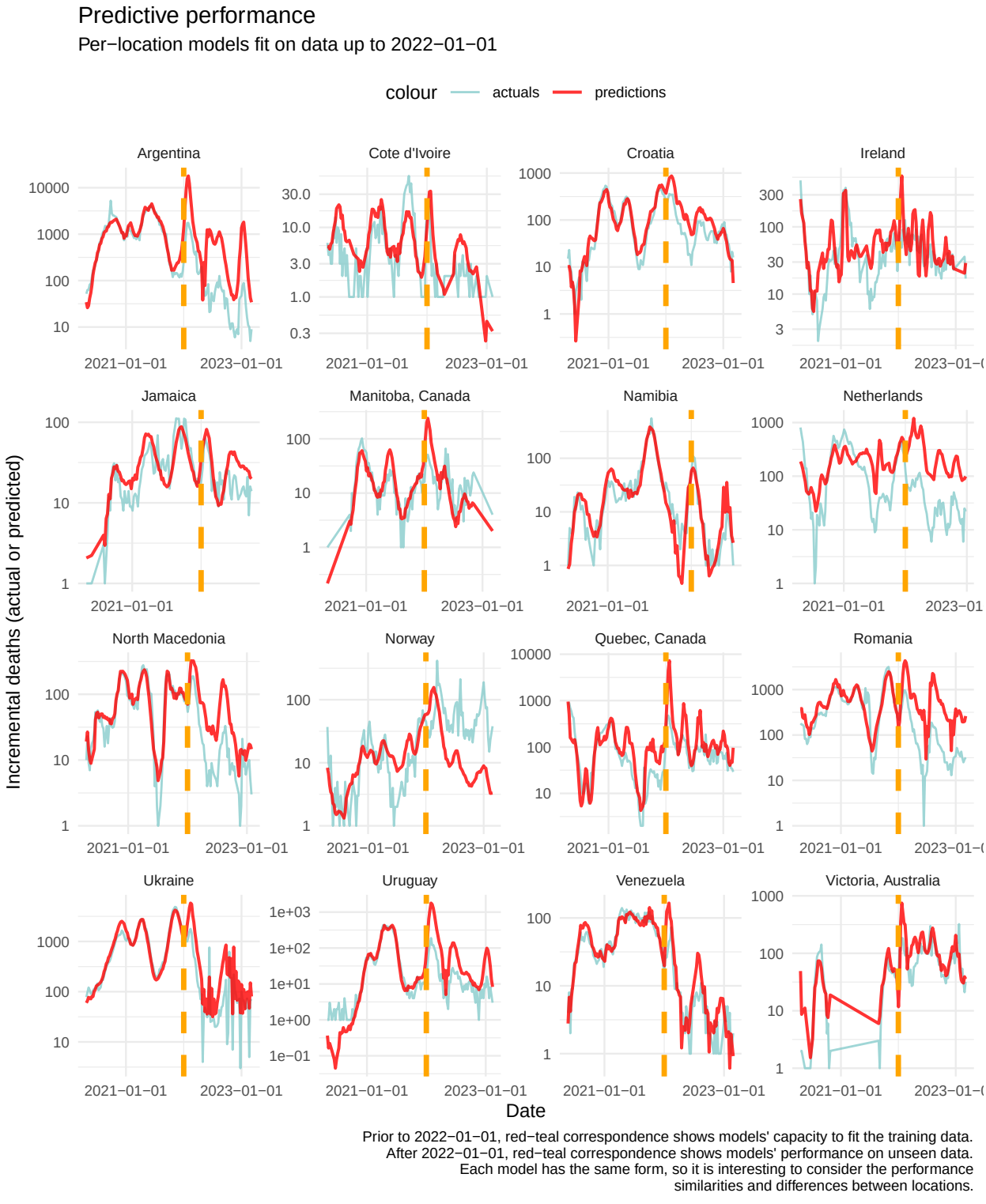

Figure 7: Model predictive performance visual analysis, select locations.

Conclusion

Conclusion ▶ Model findings

In this analysis, we have seen that in general, the number of `new_cases` from 2 weeks ago are more predictive of the number of `new_deaths` than `new_cases` from 0, 4 or 6 weeks ago.

A visual inspection of the location models' predictive performance shows that my modelling approach worked better in some locations than others. There could be numerous explanations for this, which could be investigated in follow-on studies.

Next steps

For modelling, next steps could include a more systematic approach to feature selection/decorrelation, or experimenting with other model forms.

Model evaluation could be formalized numerically. Locations could even be clustered based on their model coefficients (which features are most predictive of `deaths`) or their model predictive performance scores. Since there are so many locations, clustering them would make it easier to analyze and discuss them.

Conclusion ▶ Bias

It is important to consider the biases that may have influenced our work and modelling outcomes. I will focus on two potential sources of bias: personal bias and data bias.

Bias ▶ Personal

An area in which my personal biases influenced my work is in my modelling decisions, including:

- The problem I chose to focus on.
- How I decided to go about solving it (many models, largely because I had read about it and wanted to try it out).
- The features I chose to use.

Mitigation One way to prevent personal biases from having an effect on one's work is to choose an evaluation procedure ahead of time, and let that procedure guide you towards the correct model. Future work could involve selecting some evaluation metric, such as RMSE, and trying out numerous models to see which one performs best (though an added complexity in this case is that we are modelling many locations simultaneously, and each location could suit a different model). Before proceeding down this path, I think it would be a good idea to use something like `tidymodels` to set up a modelling pipeline. The more efficiently I can build, train and compare models, the more likely I am to find the one(s) that are best for the data, rather than being guided by my personal modelling interests/learning objectives.

A second way I prevented my personal biases from influencing my work was to use random sampling when choosing which locations' data to look at or plot. In the past, I might have picked a location that was interesting or familiar to me, which is a form of bias.

Bias ▶ Data

Data bias is very important to consider here. As indicated on the Johns Hopkins COVID-19 Data Repository main page[6], each government supplied its own data via official websites. Each country/regional healthcare system was also responsible for its own testing and treatment procedures. I can think of many ways this could have influenced the data:

- Test availability may have been more constrained in some locations than others, impacting the `cases` numbers.

[6] https://github.com/CSSEGISandData/COVID-19

- Medical treatments for COVID-19 may have varied by location, resulting in different death rates.
- Instructions regarding when to attribute COVID-19 as the cause of death may have varied by location.
- Environmental reasons: differences in strains and seasons could have impacted populations' rates of transmission, illness and death.
- The extent to which statistical modelling was used to generate the cases and/or deaths data in each location may also have had an impact.

All of these could also change over time, further complicating the picture.

Mitigation The concern about COVID-19 `cases` and `deaths` measurements varying by location and date was partially mitigated by the fact that I chose to model each location separately. However, it is still important to keep these differences in mind, and use them to temper our interpretations of the models. For example, if one location's model underpredicts `deaths` (compared with other models), it could be a modelling issue, it could have something to do with the quality of that location's COVID-19 treatment procedures, or, it could be due to a change in that location's testing policy causing reported `cases` to be lower than expected. The main point to keep in mind is that it is difficult to make generalizations when working with such a heterogeneous dataset.

Appendix

Appendix A ► Session Info

```
sessionInfo()
```

```
## R version 4.3.2 (2023-10-31)
## Platform: x86_64-apple-darwin20 (64-bit)
## Running under: macOS Monterey 12.7.3
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRlapack.dylib;  LAPACK v
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Edmonton
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] knitr_1.45      lubridate_1.9.3 forcats_1.0.0   stringr_1.5.1
##  [5] dplyr_1.1.4     purrr_1.0.2     readr_2.1.5     tidyr_1.3.1
##  [9] tibble_3.2.1    ggplot2_3.5.0   tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] utf8_1.2.4        generics_0.1.3    lattice_0.22-6    stringi_1.8.3
##  [5] hms_1.1.3         digest_0.6.34     magrittr_2.0.3    evaluate_0.23
##  [9] grid_4.3.2        timechange_0.3.0  fastmap_1.1.1     Matrix_1.6-1.1
## [13] backports_1.4.1   mgcv_1.9-0        fansi_1.0.6       scales_1.3.0
## [17] cli_3.6.2         rlang_1.1.3       crayon_1.5.2      bit64_4.0.5
## [21] munsell_0.5.0     splines_4.3.2     withr_3.0.0       yaml_2.3.8
## [25] tools_4.3.2       parallel_4.3.2    tzdb_0.4.0        colorspace_2.1-0
## [29] broom_1.0.5       curl_5.2.0        vctrs_0.6.5       R6_2.5.1
## [33] lifecycle_1.0.4   bit_4.0.5         vroom_1.6.5       pkgconfig_2.0.3
## [37] pillar_1.9.0      gtable_0.3.4      glue_1.7.0        xfun_0.43
## [41] tidyselect_1.2.0  highr_0.10        rstudioapi_0.15.0 farver_2.1.1
## [45] htmltools_0.5.7   nlme_3.1-163      labeling_0.4.3    rmarkdown_2.25
## [49] compiler_4.3.2
```