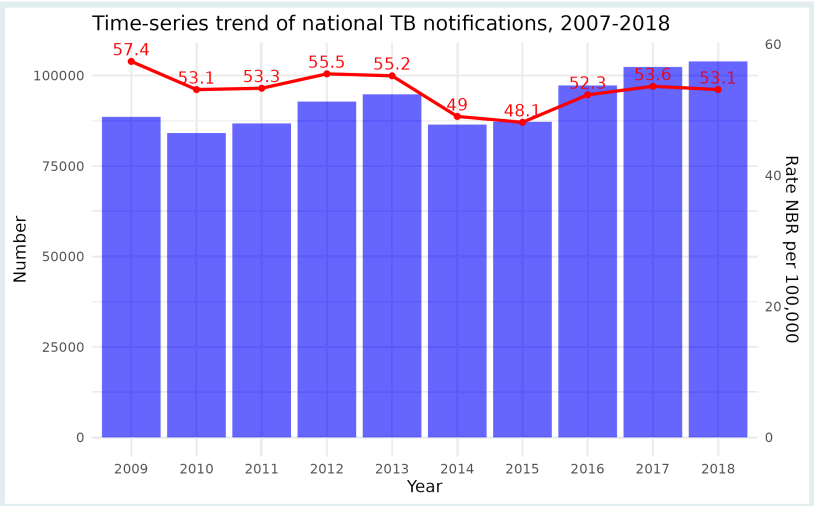
Epidemiological Time Series Visualization

Introduction

By analyzing time series data—observations made sequentially over time—epidemiologists can spot trends and patterns in disease outbreaks, and inform decision-making for improved health outcomes. In this lesson, we explore using ggplot and the tidyverse to visualize time series data and effectively communicate insights.



Learning Objectives

By the end of this lesson you will be able to:

- Reshape time series data for plotting with pivot_longer()
- Create line graphs in ggplot2 mapping time to x and values to y
- Enhance line graph aesthetics with techniques like custom labels, color palettes, annotations
- Visualize confidence intervals with geom_ribbon()
- Highlight patterns in noisy data using smoothing and aggregation
- Compare time series with distinct scales using dual axes and sec_axis()

Packages

Install and load the necessary packages with the following code chunk:

```
# Load packages
if(!require(pacman)) install.packages("pacman")
pacman::p_load(dplyr, ggplot2, tidyr, lubridate, outbreaks, scales, ggrepel,
         ggthemes, zoo, here)
options(scipen=999)
```

Setting `options(scipen = 999)` prevents the use of scientific notation in our plots, making long numbers easier to read and interpret.

Intro to Line Graphs for Time Series Data

To get started with visualizing time series data, we'll examine the dynamics of tuberculosis (TB) notifications in Australia over time, comparing notifications in urban and rural areas. The source dataset can be accessed here

Notifications are a technical term for the number of cases of a disease that are reported to public health authorities.

Data Preparation: Aggregating and Pivoting

Let's start by loading and inspecting the data:

```
tb_data_aus <- read_csv(here::here("data/aus_tb_notifs.csv"))
head(tb_data_aus)
```

```
## # A tibble: 6 × 3
##   period rural urban
##   <chr>  <dbl> <dbl>
## 1 1993Q1     6    51
## 2 1993Q2    11    52
## 3 1993Q3    13    67
## 4 1993Q4    14    82
## 5 1994Q1    16    63
## 6 1994Q2    15    65
```

This dataset includes the columns `period` (time in quarterly format, e.g., '1993Q1'), `rural` (cases in rural areas), and `urban` (cases in urban areas).

We would like to visualize the number of *annual* TB notifications in urban and rural areas, but the data is currently in a quarterly format. So, we need to aggregate the data by year.

Let's start by extracting the year from the `period` column. We do this using the `str_sub()` function from the `stringr` package:

```
tb_data_aus %>%
  mutate(year = str_sub(period, 1, 4)) %>%
  # convert back to numeric
  mutate(year = as.numeric(year))
```

```
## # A tibble: 120 × 4
##    period rural urban  year
##    <chr>  <dbl> <dbl> <dbl>
##  1 1993Q1     6    51  1993
##  2 1993Q2    11    52  1993
##  3 1993Q3    13    67  1993
##  4 1993Q4    14    82  1993
##  5 1994Q1    16    63  1994
##  6 1994Q2    15    65  1994
##  7 1994Q3    18    60  1994
##  8 1994Q4    25    71  1994
##  9 1995Q1     7    77  1995
## 10 1995Q2     9    52  1995
## # ℹ 110 more rows
```

The `str_sub()` function takes three arguments: the string we want to extract from, the starting position, and the ending position. In this case, we want to extract the first

four characters from the `period` column, which correspond to the year.

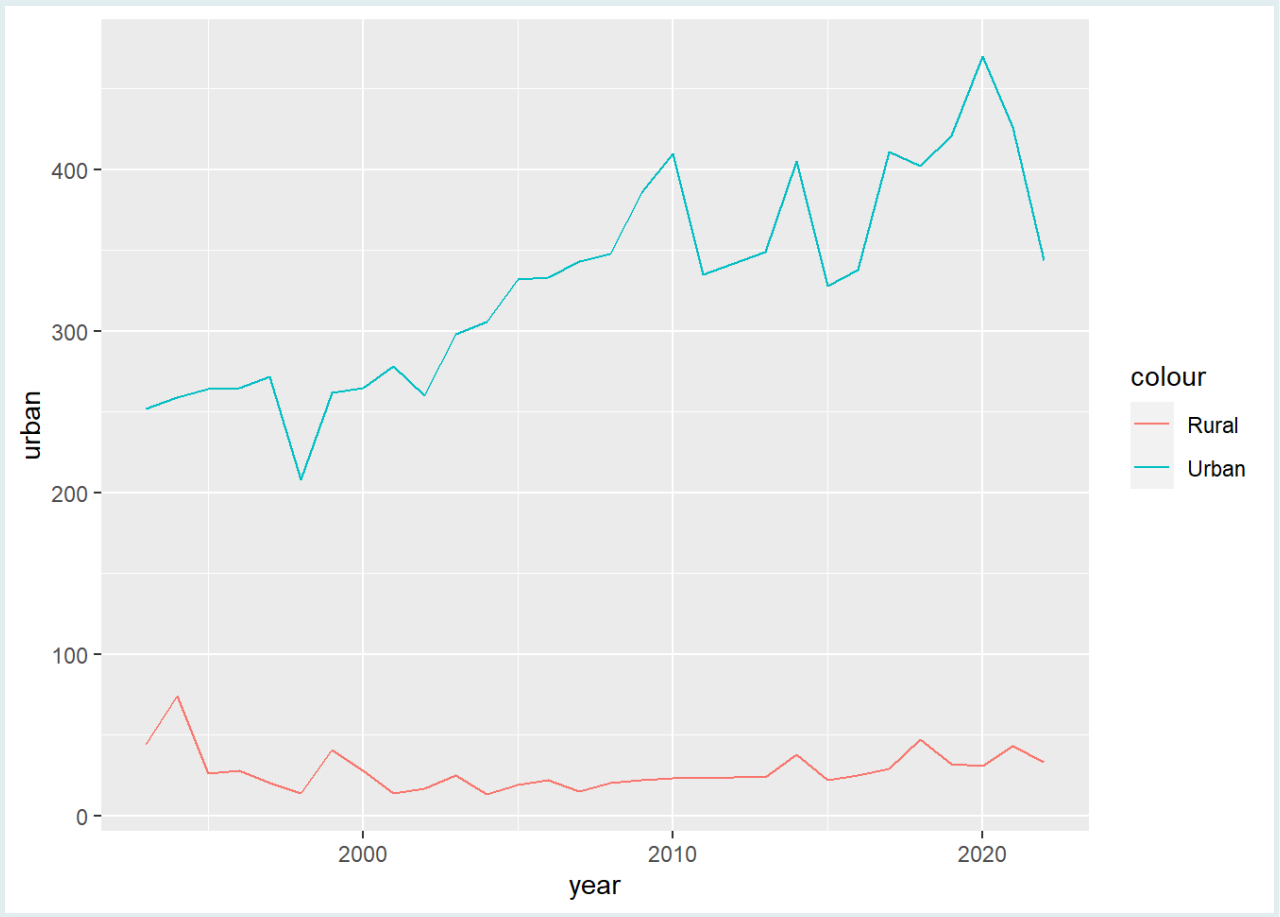
Now, let's aggregate the data by year. We can do this using the `group_by()` and `summarise()` functions:

```
annual_data_aus <- tb_data_aus %>%
  mutate(year = str_sub(period, 1, 4)) %>%
  mutate(year = as.numeric(year)) %>%
  # group by year
  group_by(year) %>%
  # sum the number of cases in each year
  summarise(rural = sum(rural),
            urban = sum(urban))
annual_data_aus
```

```
## # A tibble: 30 × 3
##     year rural urban
##    <dbl> <dbl> <dbl>
##  1  1993    44   252
##  2  1994    74   259
##  3  1995    26   264
##  4  1996    28   265
##  5  1997    20   272
##  6  1998    14   208
##  7  1999    41   262
##  8  2000    28   265
##  9  2001    14   278
## 10  2002    17   260
## # ℹ 20 more rows
```

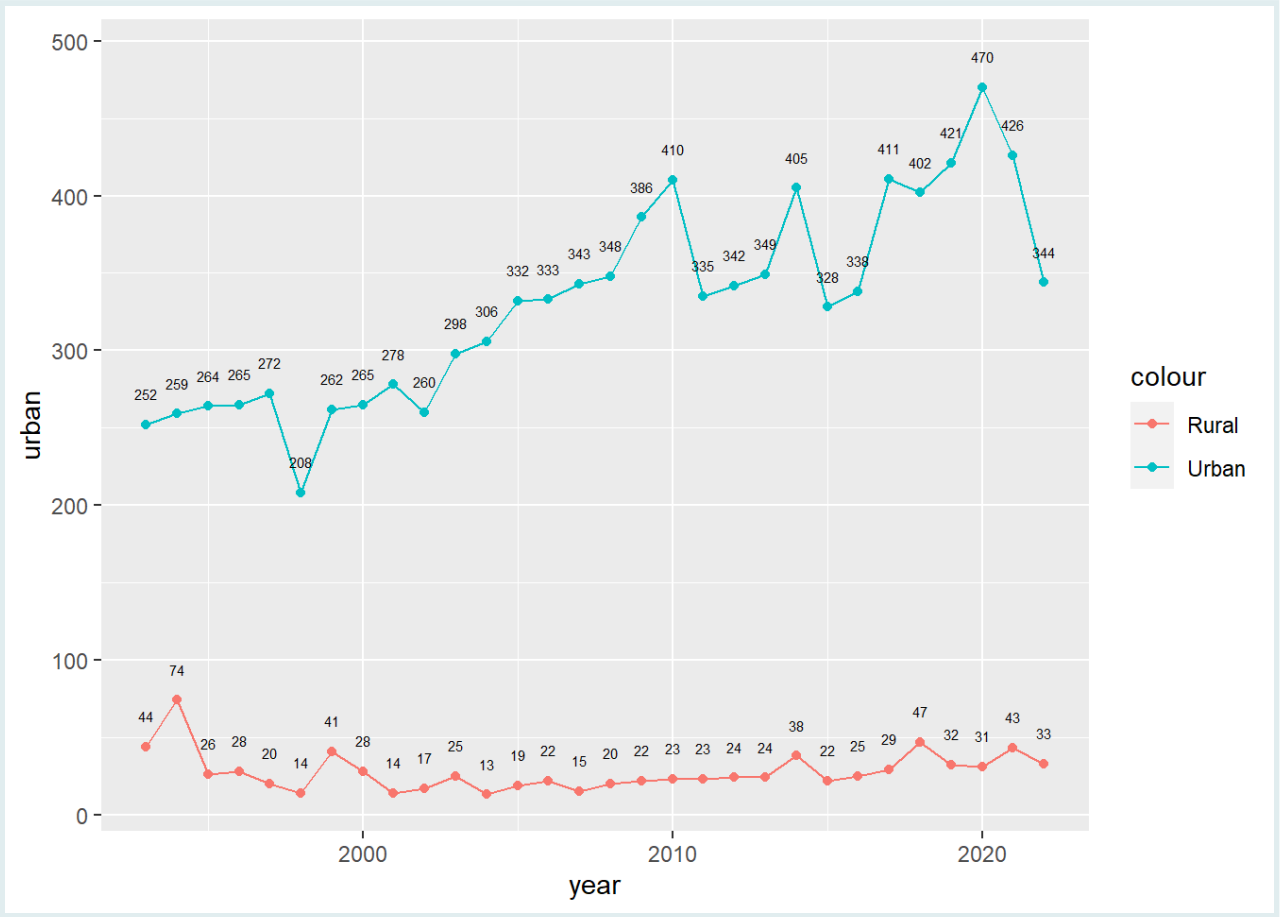
Now that we seem to have the data in the format we want, let's make an initial line plot:

```
ggplot(annual_data_aus, aes(x = year)) +
  geom_line(aes(y = urban, colour = "Urban")) +
  geom_line(aes(y = rural, colour = "Rural"))
```



This is an informative plot, however, there is some unnecessary code duplication, though you may not yet realize it. This will become clearer if we try to add additional geoms, such as points, or text:

```
ggplot(annual_data_aus, aes(x = year)) +
  geom_line(aes(y = urban, colour = "Urban")) +
  geom_line(aes(y = rural, colour = "Rural")) +
  geom_point(aes(y = urban, colour = "Urban")) +
  geom_point(aes(y = rural, colour = "Rural")) +
  geom_text(aes(y = urban, label = urban), size = 2, nudge_y = 20) +
  geom_text(aes(y = rural, label = rural), size = 2, nudge_y = 20)
```



As you can see, we have to repeat the same lines of code for each geom. This is not only tedious, but also makes the code more difficult to read and interpret. If we had more than two categories, as often happens, it would be even more cumbersome.

Fortunately, there is a better way. We can use the `pivot_longer()` function from the {tidyr} package to reshape the data into a format that is more suitable for plotting:

```
# Using tidyr's `pivot_longer` to reshape the data
annual_data_aus %>%
  pivot_longer(cols = c("urban", "rural"))
```

```
## # A tibble: 60 × 3
##     year name  value
##    <dbl> <chr> <dbl>
##  1  1993 urban   252
##  2  1993 rural    44
##  3  1994 urban   259
##  4  1994 rural    74
##  5  1995 urban   264
##  6  1995 rural    26
##  7  1996 urban   265
##  8  1996 rural    28
##  9  1997 urban   272
```

```
## 10  1997 rural    20
## # ℹ 50 more rows
```

The code above has converted the data from a “wide” format to a “long” format. This is a more suitable format for plotting, as it allows us to map a specific column to the `colour` aesthetic.

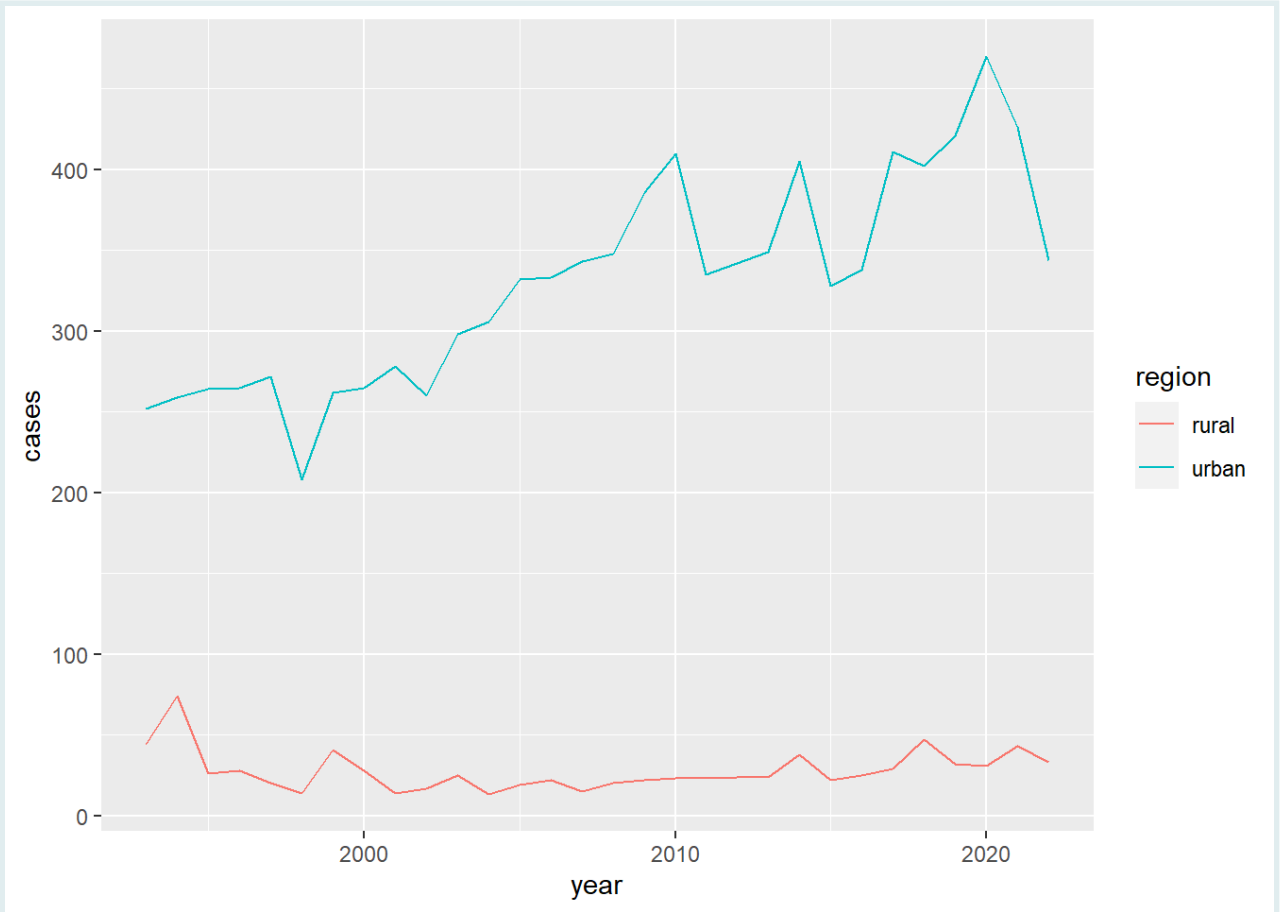
Before we plot this long dataset, let’s rename the columns to make them more informative:

```
aus_long <- annual_data_aus %>%
  pivot_longer(cols = c("urban", "rural")) %>%
  rename(region = name, cases = value)
```

A Basic Grouped Line Graph

We’re ready to plot the data again. We map the colour and group aesthetics to the `region` column, which contains the two categories of interest: urban and rural.

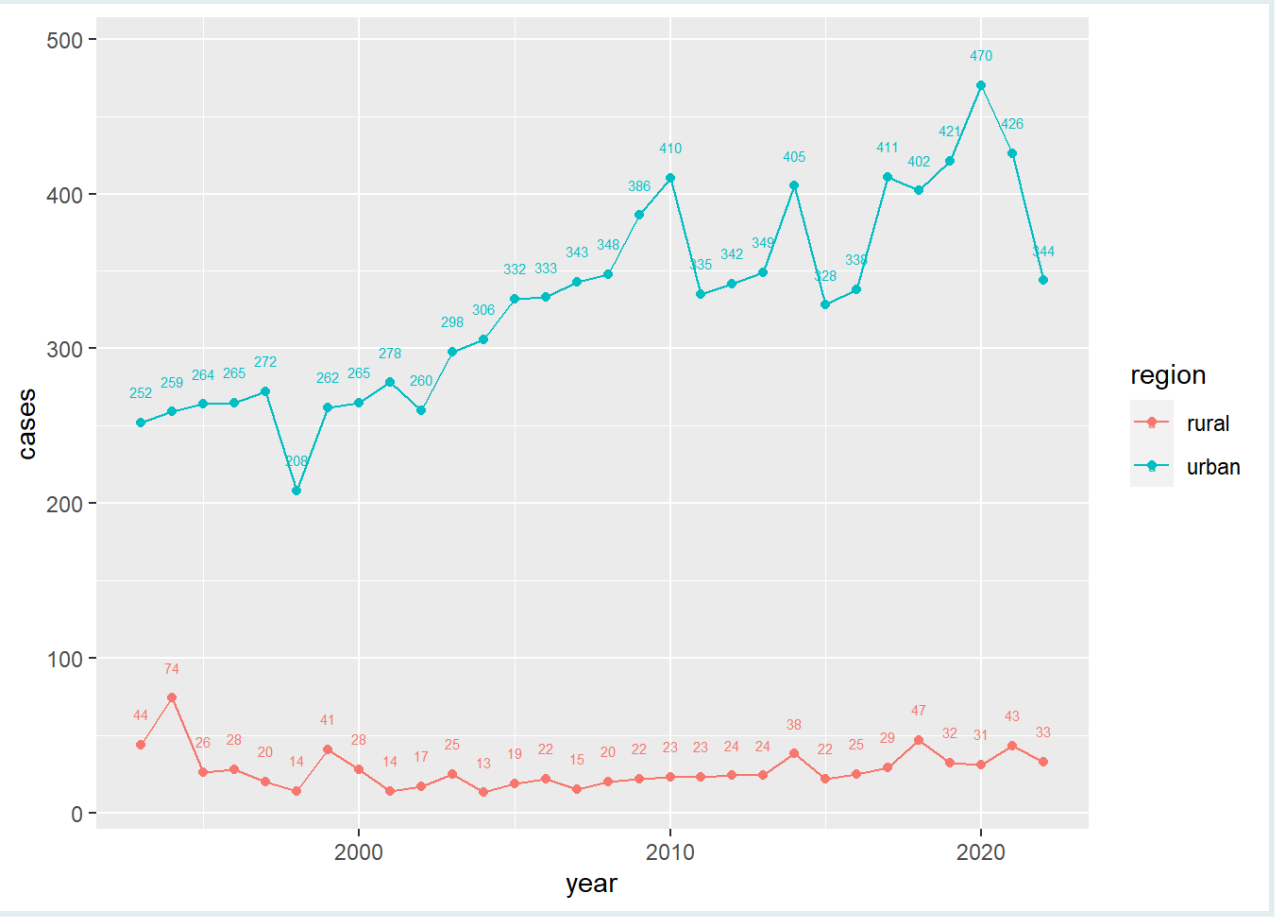
```
ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +
  geom_line()
```



The plotting code is now more concise, thanks to the pivoting operation executed previously.

We can now also add points and text labels with significantly less code:

```
ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +
  geom_line() +
  geom_point() +
  geom_text(aes(label = cases), size = 2, nudge_y = 20)
```



Great! We now have a clear view of of trends in annual TB case notifications in rural and urban areas over time. However, there are still some aesthetic improvements we can make; we will cover these in the next section.

Q: Reshaping and Plotting TB Data

Consider the Benin dataset shown below, which contains information about bacteriologically confirmed and clinically diagnosed TB cases for several years in Benin. (The data was sourced from a paper here

```
tb_data_benin <- read_csv(here("data/benin_tb_notifs.csv"))
tb_data_benin
```

```
## # A tibble: 15 × 3
##     year new_clindx new_labconf
##    <dbl>      <dbl>       <dbl>
##  1  2000        289        2280
##  2  2001        286        2289
##  3  2002        338        2428
##  4  2003        350        2449
##  5  2004        330        2577
##  6  2005        346        2731
##  7  2006        261        2950
##  8  2007        294        2755
##  9  2008        239        2983
## 10  2009        279        2950
## 11  2010        307        2958
## 12  2011        346        3326
## 13  2012        277        3086
## 14  2013        285        3219
## 15  2014        318        3062
```

Reshape the dataset using `pivot_longer()`, then create a plot with two lines, one for each type of TB case diagnosis. Add points and text labels to the plot.

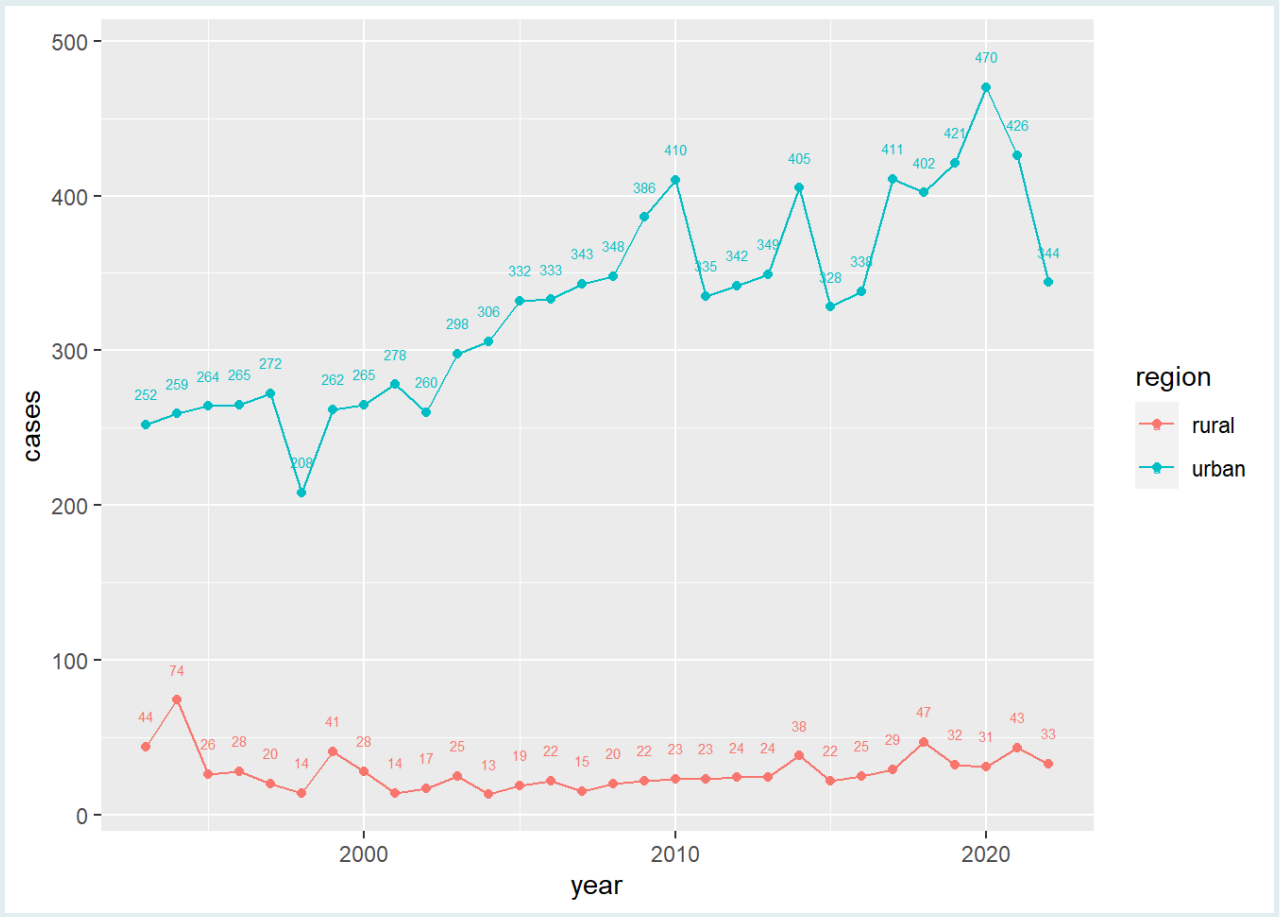
Aesthetic Improvements to Line Graphs

In this section, we will focus on improving the aesthetics of time series line graphs to enhance their clarity and visual appeal.

Reducing Label Frequency

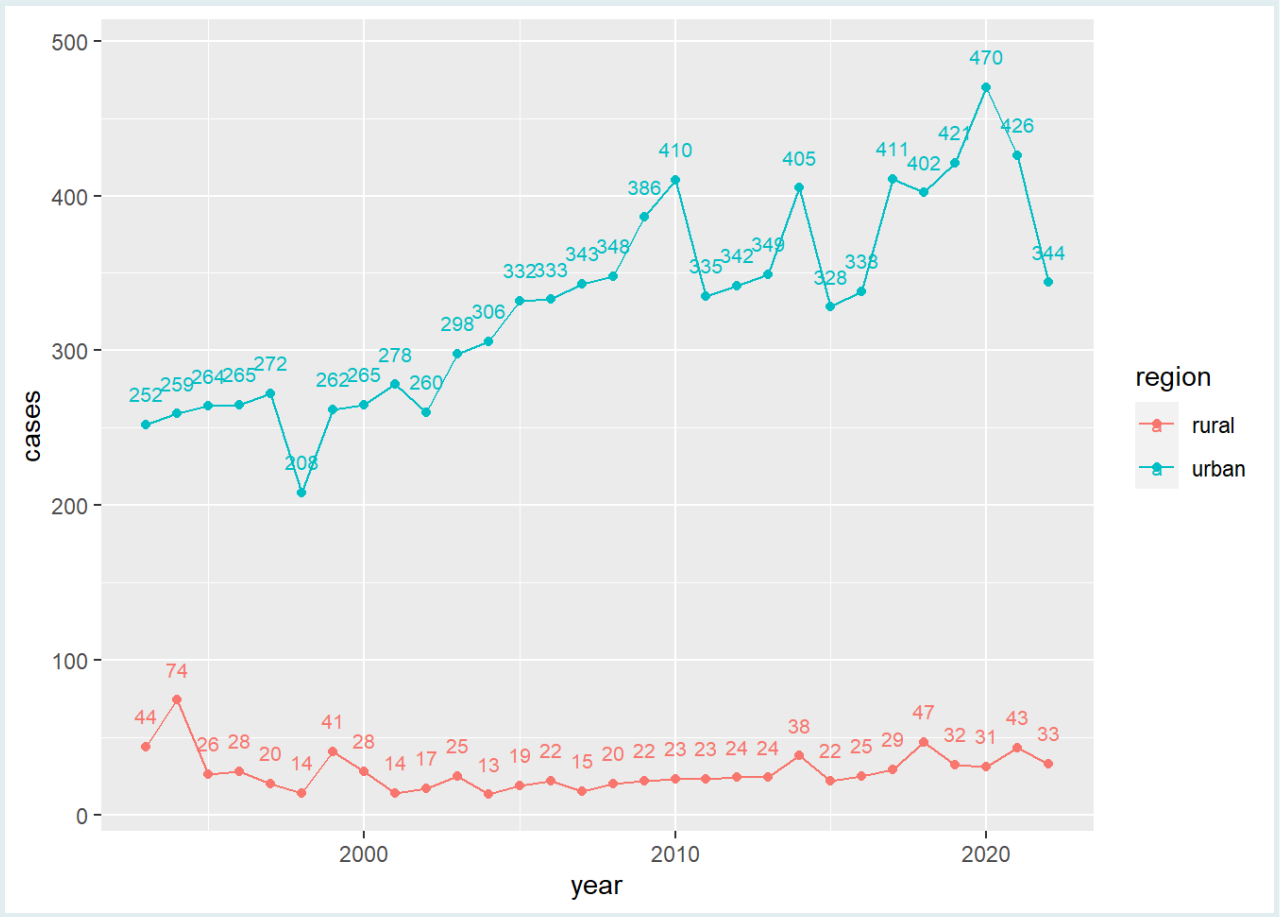
Where we last left off, we had a plot that looked like this:

```
ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +
  geom_line() +
  geom_point() +
  geom_text(aes(label = cases), size = 2, nudge_y = 20)
```



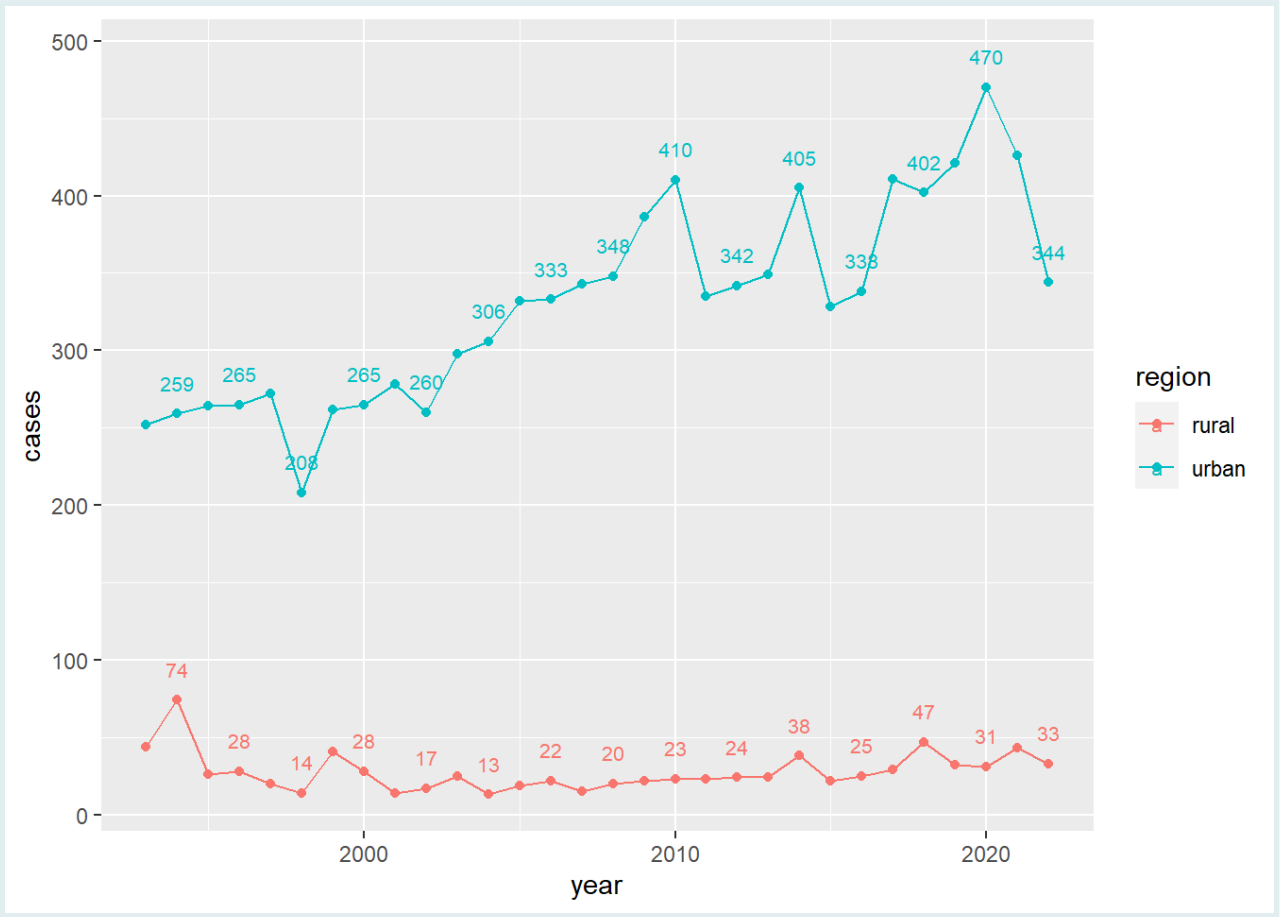
One problem with this plot is that the text labels are a bit too small. Such tiny labels are not ideal for a public-facing plot, as they are difficult to read. However, if we increase the label size, the labels will start to overlap, as shown below:

```
ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +
  geom_line() +
  geom_point() +
  geom_text(aes(label = cases), size = 2.8, nudge_y = 20)
```



To avoid this clutter, a handy technique is to display labels for only certain years. To do this, we can give a custom dataset to the `geom_text()` function. In this case, we will create a dataset that contains only the even years:

```
even_years <- aus_long %>%
  filter(year %% 2 == 0) # Keep only years that are multiples of 2

ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +
  geom_line() +
  geom_point() +
  geom_text(data = even_years, aes(label = cases),
            size = 2.8, nudge_y = 20)
```



Great, now we have larger labels and they do not overlap.

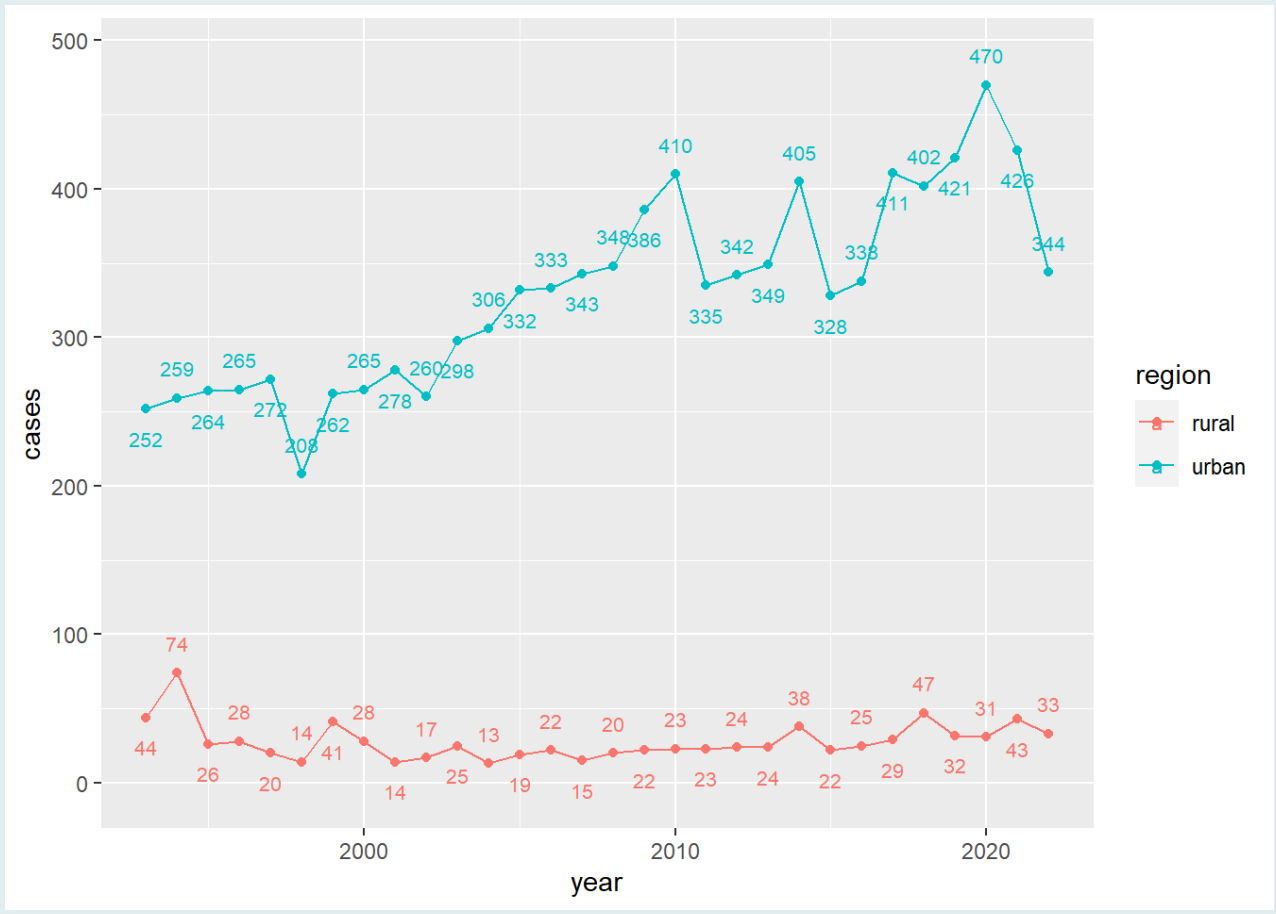
Alternating Labels

While the plot above is an improvement, it would be even better if we could display the labels for *all* years. We can do this by displaying the labels for the even years above the data points, and the labels for the odd years below the data points.

Including many data points (within reason) in your plots is helpful for public health officials; as they can pull quick numbers from the plot when trying to make decisions, without needing to look at the reference datasets.

To address this, let's create a filtered dataset for odd years, and then use `geom_text()` twice, once for each filtered dataset.

```r
odd_years <- aus_long %>%
  filter(year %% 2 != 0) # Keep only years that are NOT multiples of 2

ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +
  geom_line() +
  geom_point() +
  geom_text(data = even_years, aes(label = cases),
            nudge_y = 20, size = 2.8) +
  geom_text(data = odd_years, aes(label = cases),
            nudge_y = -20, size = 2.8)
```



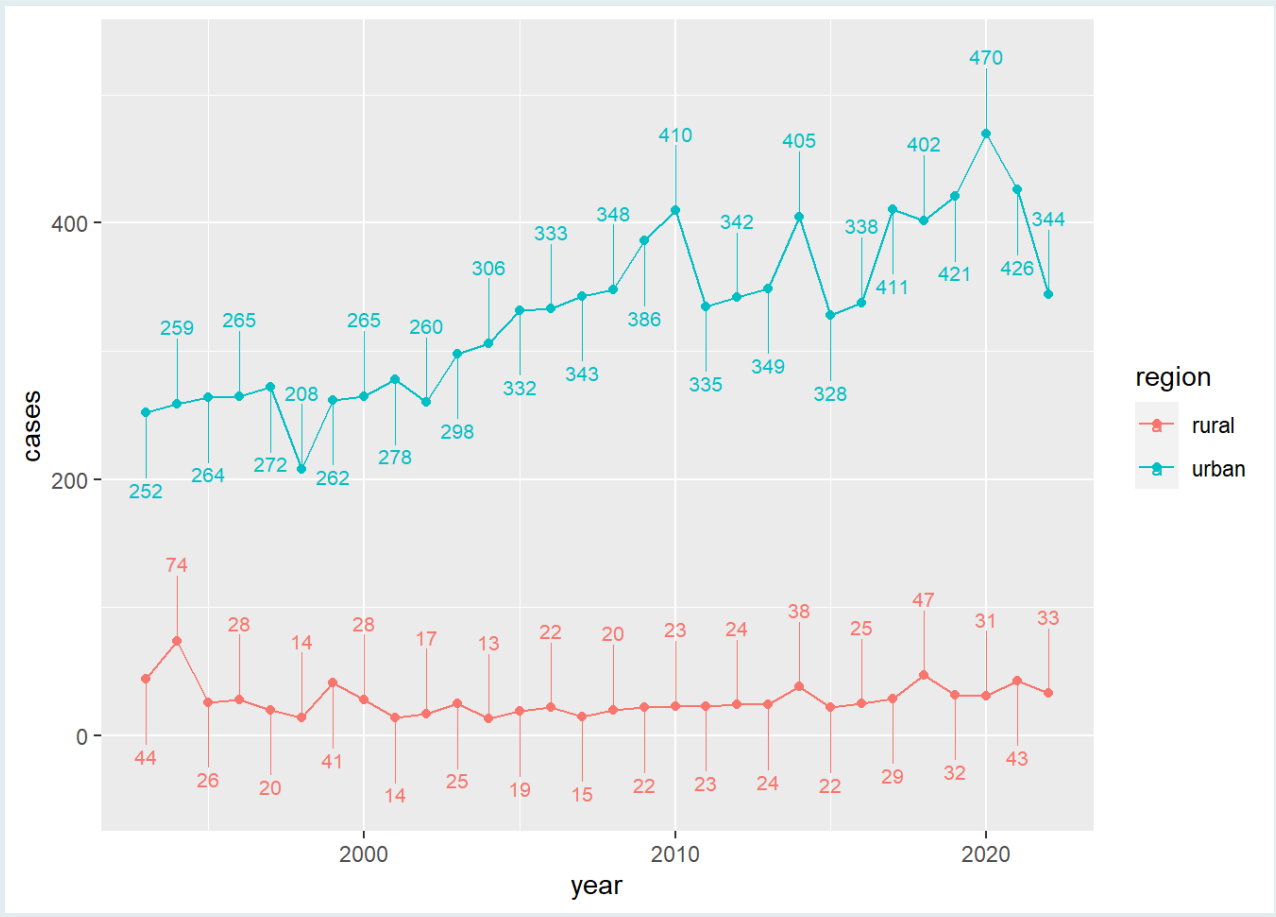
`ggrepel::geom_text_repel()`

The plot above is clear, but there is still some overlap between the labels and the line.

To further enhance clarity, we can use the `geom_text_repel()` from the {ggrepel} package.

This function nudges individual labels to prevent overlap, and connects labels to their data points with lines, making it easier to see which label corresponds to which data point, and allowing us to increase the distance between the labels and the data points.

```
ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +
  geom_line() +
  geom_point() +
  geom_text_repel(data = even_years, aes(label = cases),
                  nudge_y = 60, size = 2.8, segment.size = 0.1) +
  geom_text_repel(data = odd_years, aes(label = cases),
                  nudge_y = -60, size = 2.8, segment.size = 0.1)
```



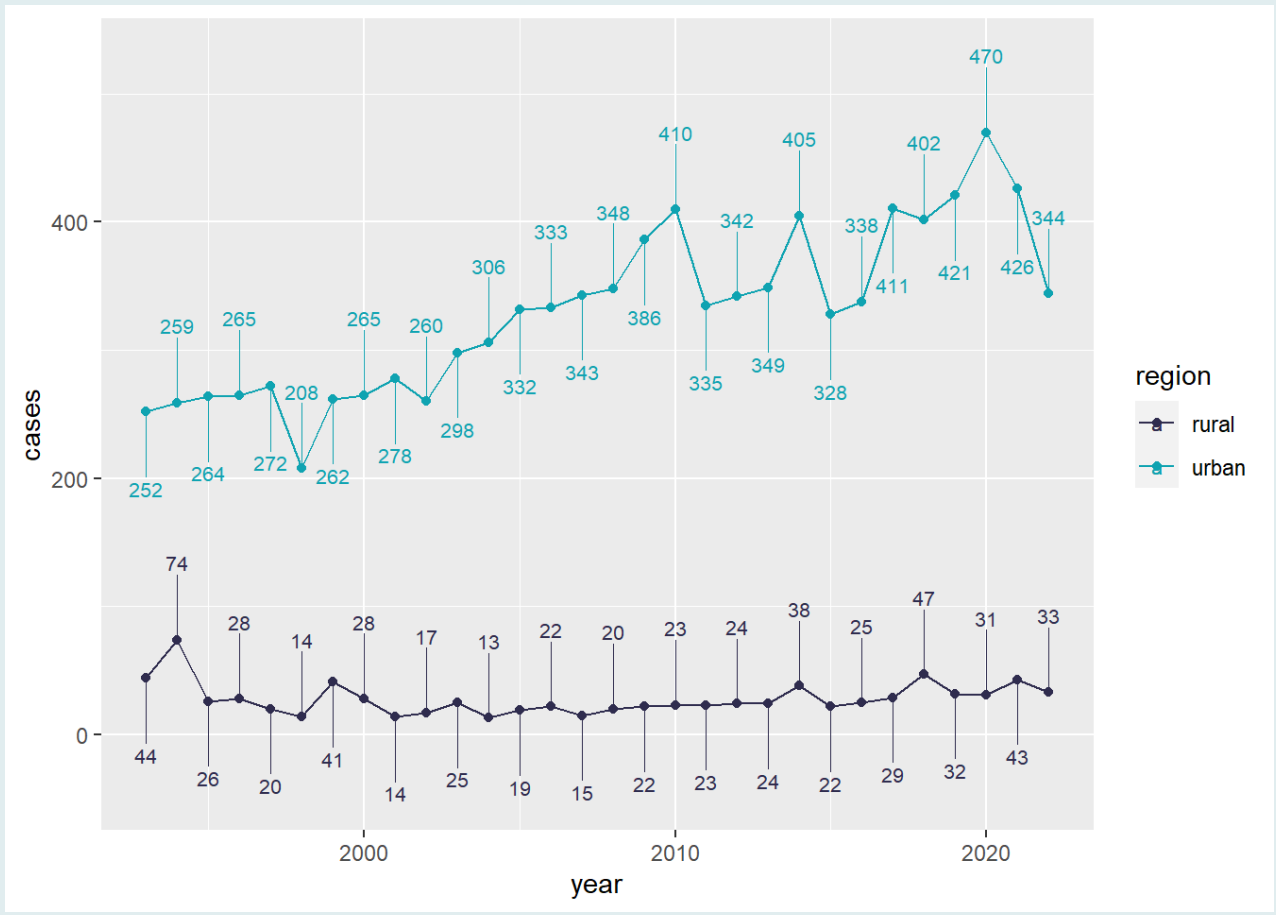
As you can see, the function `geom_text_repel()` takes basically the same arguments as `geom_text()`. The extra argument, `segment.size`, controls the width of the lines connecting the labels to the data points.

Customizing the Color Palette

It is often useful to customize the color palette of your plots, so that they match, for example, the color scheme of your organization.

We can customize the colors of the lines using the `scale_color_manual()` function. Below, we specify two colors, one for each region:

```
ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +
  geom_line() +
  geom_point() +
  geom_text_repel(data = even_years, aes(label = cases),
                  nudge_y = 60, size = 2.8, segment.size = 0.1) +
  geom_text_repel(data = odd_years, aes(label = cases),
                  nudge_y = -60, size = 2.8, segment.size = 0.1) +
  scale_color_manual(values = c("urban" = "#0fa3b1",
                                "rural" = "#2F2C4E"))
```

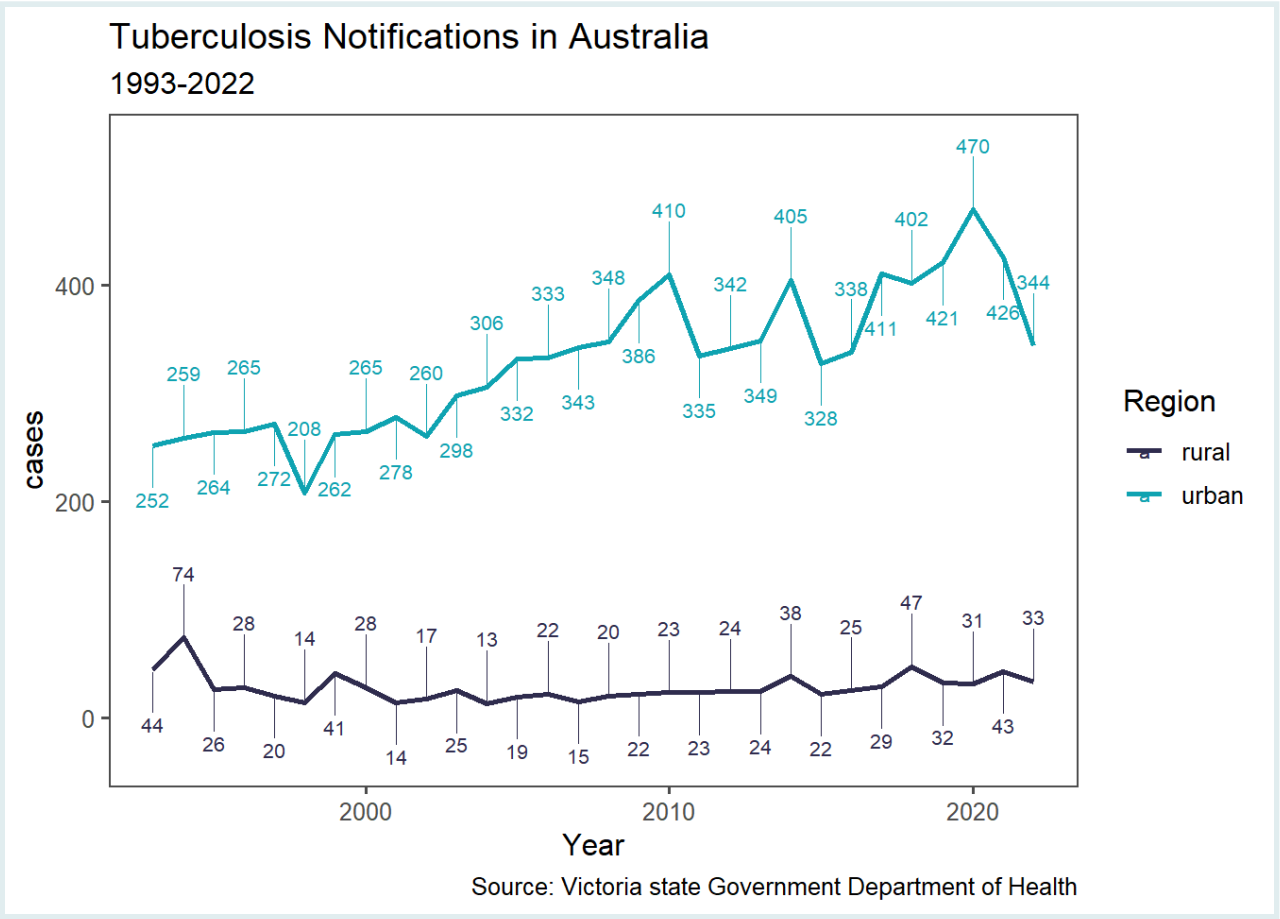


Success!

Adding Plot Annotations

Finally, let's add a set of finish touches. We'll annotate the plot with appropriate titles, axis labels, and captions, and modify the theme:

```
ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +
  geom_line(linewidth = 1) +
  geom_text_repel(data = even_years, aes(label = cases),
                  nudge_y = 60, size = 2.8, segment.size = 0.08) +
  geom_text_repel(data = odd_years, aes(label = cases),
                  nudge_y = -50, size = 2.8, segment.size = 0.08) +
  scale_color_manual(values = c("urban" = "#0fa3b1", "rural" = "#2F2C4E")) +
  labs(title = "Tuberculosis Notifications in Australia",
       subtitle = "1993-2022",
       caption = "Source: Victoria state Government Department of Health",
       x = "Year",
       color = "Region") +
  ggthemes::theme_few() +
  theme(legend.position = "right")
```



This covers some options for improving line chart aesthetics! Feel free to further tweak and adjust visuals based on your specific analysis needs.

RECAP

We've transformed our plot into a visually appealing, easy-to-read representation of TB notification trends in Australia. We balanced the need for detailed information with a clear presentation, making our plot both informative and accessible.

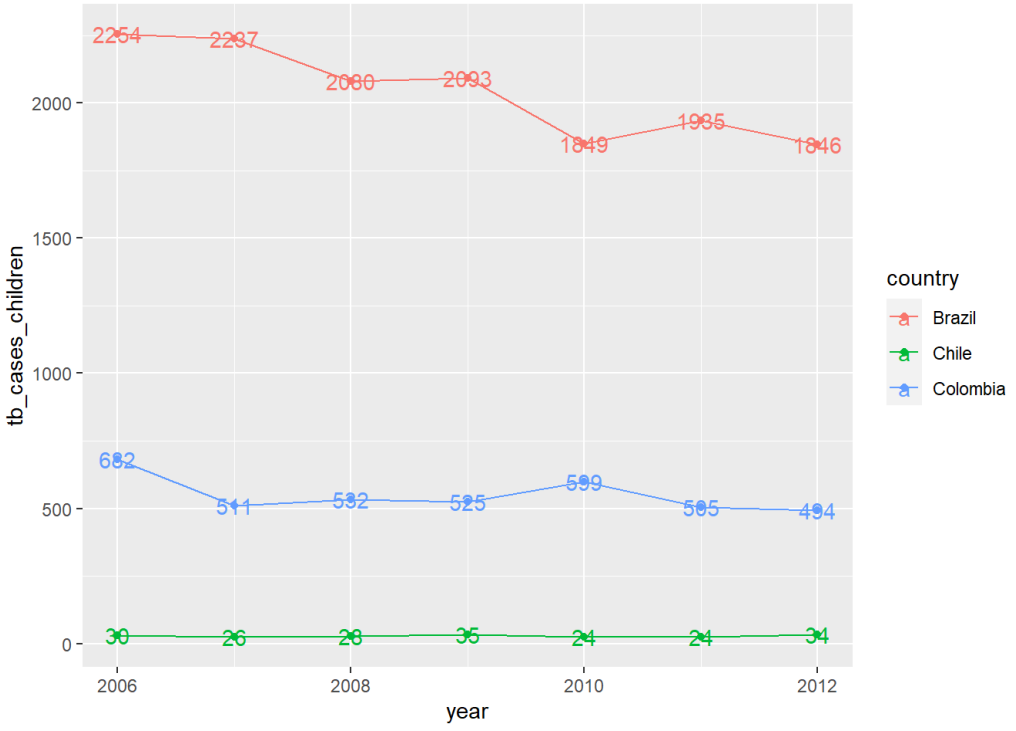
Q: Aesthetic improvements

Consider the following plot, which shows the number of child TB cases in three countries over time:

```
tb_child_cases_southam <- tidyr::who2 %>%
  transmute(country, year,
            tb_cases_children = sp_m_014 + sp_f_014 + sn_m_014 +
        sn_f_014) %>%
  filter(country %in% c("Brazil", "Colombia", "Chile")) %>%
  filter(!is.na(tb_cases_children))

tb_child_cases_southam %>%
  ggplot(aes(x = year, y = tb_cases_children, color = country))
        +
  geom_line() +
  geom_point() +
  geom_text(aes(label = tb_cases_children))
```

PRACTICE

(in RMD)



Build on this plot, implementing the following improvements:

- Set the `geom_text` labels to alternate above and below the lines, similar to the example we saw above.

- Use the following color palette `c("#212738", "#F97068", "#067BC2")`
- Apply `theme_classic()`
- Add a title, subtitle, and caption to provide context and information about the data. (You can type `?tidyr::who2` into the console to learn more about the data source.)

Plotting Confidence Intervals with `geom_ribbon()`

In time series visualizations, it is often important to plot confidence intervals to indicate the level of uncertainty in your data.

We will demonstrate how to do this using a dataset on new HIV infections in Brazil, which includes estimated numbers for male and female cases along with confidence intervals. The dataset is sourced from the World Health Organization (WHO) and can be accessed here.

Let's start by loading and inspecting the dataset:

```
hiv_data_brazil <-
  rio::import(here("data/new_hiv_infections_gho.xlsx"),
                                sheet = "Brazil") %>%
  as_tibble() %>%
  janitor::clean_names()
hiv_data_brazil
```

```
## # A tibble: 89 × 5
##    continent country  year sex        new_hiv_cases
##    <chr>     <chr>   <dbl> <chr>      <chr>
##  1 Americas  Brazil   2022 Female     13 000 [12 000 - 15 0…
##  2 Americas  Brazil   2022 Male       37 000 [35 000 - 40 0…
##  3 Americas  Brazil   2022 Both sexes 51 000 [47 000 - 54 0…
##  4 Americas  Brazil   2021 Female     14 000 [12 000 - 15 0…
##  5 Americas  Brazil   2021 Male       37 000 [34 000 - 39 0…
##  6 Americas  Brazil   2021 Both sexes 50 000 [47 000 - 53 0…
##  7 Americas  Brazil   2020 Female     14 000 [13 000 - 15 0…
##  8 Americas  Brazil   2020 Male       35 000 [32 000 - 37 0…
##  9 Americas  Brazil   2020 Both sexes 49 000 [46 000 - 52 0…
## 10 Americas  Brazil   2019 Female     14 000 [13 000 - 15 0…
## # ℹ 79 more rows
```

We can see that the `new_hiv_cases` column contains both the number of cases and the corresponding confidence intervals in square brackets. This format cannot be directly used for plotting, so we will need to extract them into pure numeric forms.

First, to separate these values, we can use the `separate()` function from the {tidyr} package:

```
hiv_data_brazil %>%
  separate(new_hiv_cases,
           into = c("cases", "cases_lower", "cases_upper"),
           sep = "\\[|-")
```

```
## # A tibble: 89 × 7
##    continent country  year sex        cases     cases_lower
##    <chr>     <chr>   <dbl> <chr>      <chr>     <chr>
##  1 Americas  Brazil   2022 Female     "13 000 " "12 000 "
##  2 Americas  Brazil   2022 Male       "37 000 " "35 000 "
##  3 Americas  Brazil   2022 Both sexes "51 000 " "47 000 "
##  4 Americas  Brazil   2021 Female     "14 000 " "12 000 "
##  5 Americas  Brazil   2021 Male       "37 000 " "34 000 "
##  6 Americas  Brazil   2021 Both sexes "50 000 " "47 000 "
##  7 Americas  Brazil   2020 Female     "14 000 " "13 000 "
##  8 Americas  Brazil   2020 Male       "35 000 " "32 000 "
##  9 Americas  Brazil   2020 Both sexes "49 000 " "46 000 "
## 10 Americas  Brazil   2019 Female     "14 000 " "13 000 "
## # ℹ 79 more rows
## # ℹ 1 more variable: cases_upper <chr>
```

In the code above, we split the `new_hiv_cases` column into three new columns: `cases`, `cases_lower`, and `cases_upper`. We use the `[` and `-` as separators. The double backslash `\\` is used to escape the square bracket, which has a special meaning in regular expressions. And the `|` is used to indicate that either the `[` or the `-` can be used as a separator.

Large Language Models like ChatGPT are excellent at regular expression understanding. If you're ever stuck with code like `sep = "\\[|-"` and want to understand what it does, you can ask ChatGPT to explain it to you. And if you need to generate such expressions yourself, you can ask ChatGPT to generate them for you.

Next, we need to convert these string values into numeric values, removing any non-numeric characters.

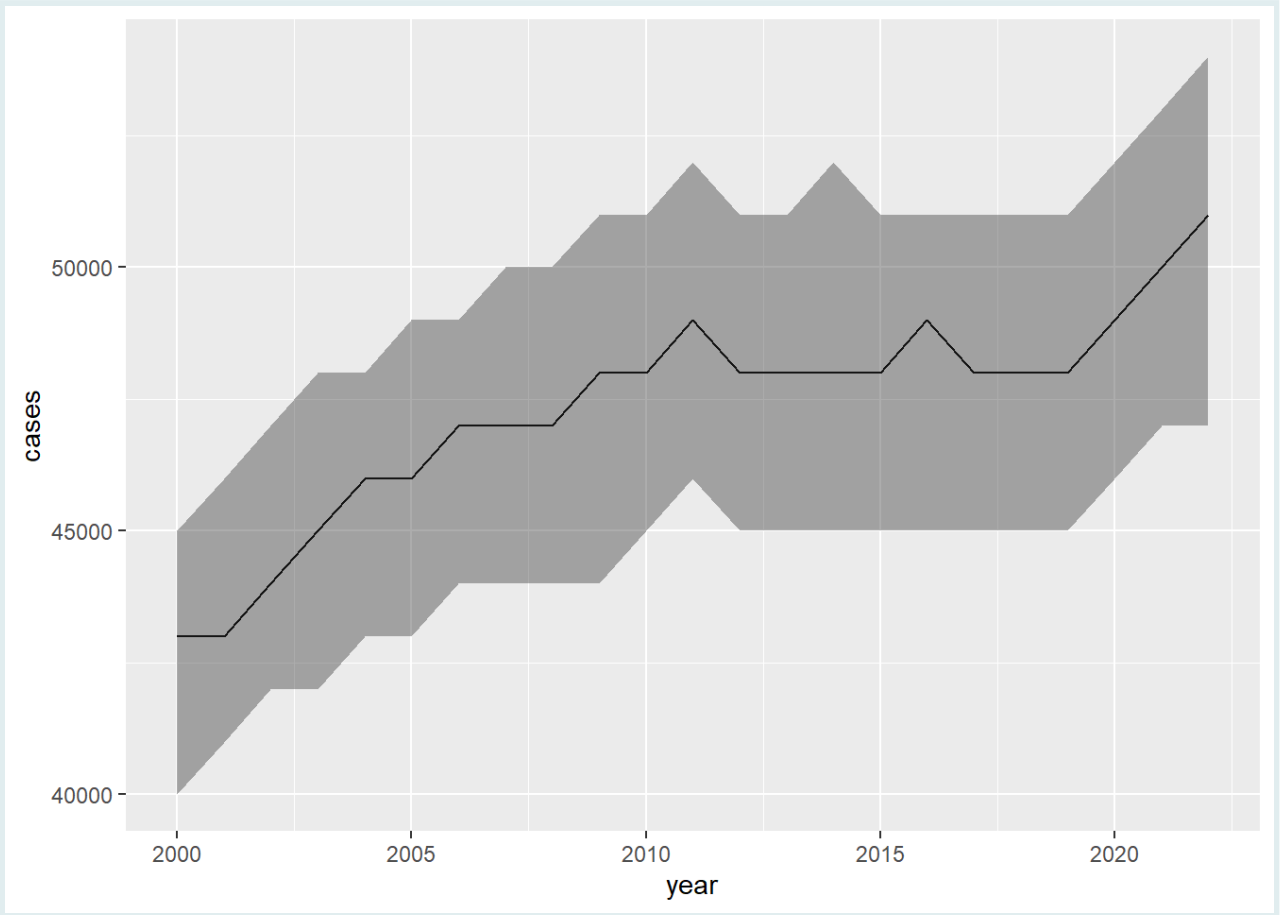
```
hiv_data_brazil_clean <-
  hiv_data_brazil %>%
  separate(new_hiv_cases,
           into = c("cases", "cases_lower", "cases_upper"),
           sep = "\\[|-") %>%
  mutate(across(c("cases", "cases_lower", "cases_upper"),
                ~ str_replace_all(.x, "[^0-9]", "") %>%
                  as.numeric()))

hiv_data_brazil_clean
```

```
## # A tibble: 89 × 7
##    continent country  year sex        cases cases_lower
##    <chr>     <chr>   <dbl> <chr>      <dbl>       <dbl>
##  1 Americas  Brazil   2022 Female     13000       12000
##  2 Americas  Brazil   2022 Male       37000       35000
##  3 Americas  Brazil   2022 Both sexes 51000       47000
##  4 Americas  Brazil   2021 Female     14000       12000
##  5 Americas  Brazil   2021 Male       37000       34000
##  6 Americas  Brazil   2021 Both sexes 50000       47000
##  7 Americas  Brazil   2020 Female     14000       13000
##  8 Americas  Brazil   2020 Male       35000       32000
##  9 Americas  Brazil   2020 Both sexes 49000       46000
## 10 Americas  Brazil   2019 Female     14000       13000
## # ℹ 79 more rows
## # ℹ 1 more variable: cases_upper <dbl>
```

The code above looks complex, but essentially, it cleans the data by keeping only numeric characters and then converts these numbers to actual numeric values. See our lesson on the `across()` function for further details.

We're finally ready to plot the data. We'll use ggplot's `geom_ribbon()` to display the confidence intervals:

```
hiv_data_brazil_clean %>%
  filter(sex == "Both sexes") %>%
  ggplot(aes(x = year, y = cases)) +
  geom_line() +
  geom_ribbon(aes(ymin = cases_lower, ymax = cases_upper), alpha = 0.4)
```



The `geom_ribbon()` function takes the `x` and `y` aesthetics like `geom_line()`, but it also takes in `ymin` and `ymax` aesthetics, to determine the vertical extent of the ribbon. We also set the transparency of the ribbon using the `alpha` argument.

We can create a separate ribbon for men and women to compare their infection trends.

```
hiv_data_brazil_clean %>%
  filter(sex != "Both sexes") %>%
  ggplot(aes(x = year, y = cases, color = sex, fill = sex)) +
  geom_line() +
  geom_ribbon(aes(ymin = cases_lower, ymax = cases_upper), alpha = 0.4)
```

Notably HIV infection rates among women have been falling in recent years, but those among men have been rising.

Q: Plotting confidence intervals

Consider the following dataset that shows the number of annual malaria cases in Kenya and Nigeria. The data is sourced from the WHO Global Health Observatory data repository and can be accessed here.

```
nig_ken_mal <- read_csv("data/nigeria_kenya_malaria.csv")
nig_ken_mal
```

```
## # A tibble: 44 × 3
##    country  year malaria_cases
##    <chr>   <dbl> <chr>
##  1 Kenya    2021 3 419 698 (2 478 000 to 4 641 000)
##  2 Nigeria  2021 65 399 501 (47 520 000 to 89 000 000)
##  3 Kenya    2020 3 302 189 (2 385 000 to 4 466 000)
```

```
##  4 Nigeria  2020 65 133 759 (46 800 000 to 88 650 000)
##  5 Kenya    2019 3 037 541 (2 168 000 to 4 130 000)
##  6 Nigeria  2019 61 379 283 (47 440 000 to 78 810 000)
##  7 Kenya    2018 3 068 062 (2 148 000 to 4 260 000)
##  8 Nigeria  2018 59 652 248 (46 930 000 to 75 230 000)
##  9 Kenya    2017 3 155 636 (2 217 000 to 4 375 000)
## 10 Nigeria  2017 57 869 533 (45 870 000 to 72 050 000)
## # ℹ 34 more rows
```

Write code to extract the confidence intervals from the "malaria_cases" column and create a plot with confidence intervals using `geom_ribbon()`. Use a different color for each country.

Smoothing Noisy Trends

When analyzing time series data, it is common for daily or granular measurements to show a lot of noise and variability, and this can hide the important trends we are actually interested in. Smoothing techniques can help highlight these trends and patterns. We'll explore several techniques for this in the sections below.

First though, let's do some data preparation!

Creating an Incidence Table from a Linelist

Consider the following linelist of pediatric malaria admissions in four hospitals in Mozambique (Data source):

```
mal <-
  rio::import(here("data/pediatric_malaria_data_joao_2021.xlsx")) %>%
  as_tibble() %>%
  mutate(date_positive_test = as.Date(date_positive_test)) %>%
  # Keep data from 2019-2020
  filter(date_positive_test >= as.Date("2019-01-01"),
         date_positive_test <= as.Date("2020-12-31"))
mal
```

```
## # A tibble: 20,939 × 4
##    date_positive_test neighbourhood sex   age
##    <date>             <chr>         <chr> <chr>
##  1 2020-01-22         25 de Junho 1 M     6-11 meses
##  2 2020-01-22         Chicueu       F     5-14 anos
##  3 2020-01-22         Mussessa      M     5-14 anos
##  4 2020-01-22         Nhamizara     M     12-23 meses
##  5 2020-01-22         Nhamizara     F     12-23 meses
##  6 2020-01-22         Unidade       F     5-14 anos
##  7 2020-01-22         Bapua         F     12-23 meses
```

```
##  8 2020-01-22          Bapua          F      5-14 anos
##  9 2020-01-22          7 de Abril     M      12-23 meses
## 10 2020-01-22          Nhanguzue      F      5-14 anos
## # ℹ 20,929 more rows
```

Each row corresponds to a single malaria case, and the `date_positive_test` column indicates the date when the child tested positive for malaria.

To get a count of cases per day—that is, an incidence table—we can simply use `count()` to aggregate the cases by date of positive test:

```
mal %>%
  count(date_positive_test, name = "cases")
```

```
## # A tibble: 235 × 2
##    date_positive_test cases
##    <date>             <int>
##  1 2019-01-01            67
##  2 2019-01-02           120
##  3 2019-01-03           112
##  4 2019-01-04           203
##  5 2019-01-05            85
##  6 2019-01-07           115
##  7 2019-01-08           196
##  8 2019-01-10            89
##  9 2019-01-11            55
## 10 2019-01-12            69
## # ℹ 225 more rows
```

There are many dates missing though—days when no children were admitted. To create a complete incidence table, we should use `complete()` to insert missing dates and then fill in the missing values with 0:

```
mal_notif_count <- mal %>%
  count(date_positive_test, name = "cases") %>%
  complete(date_positive_test = seq.Date(min(date_positive_test),
                                         max(date_positive_test),
                                         by = "day"),
           fill = list(cases = 0))

mal_notif_count
```

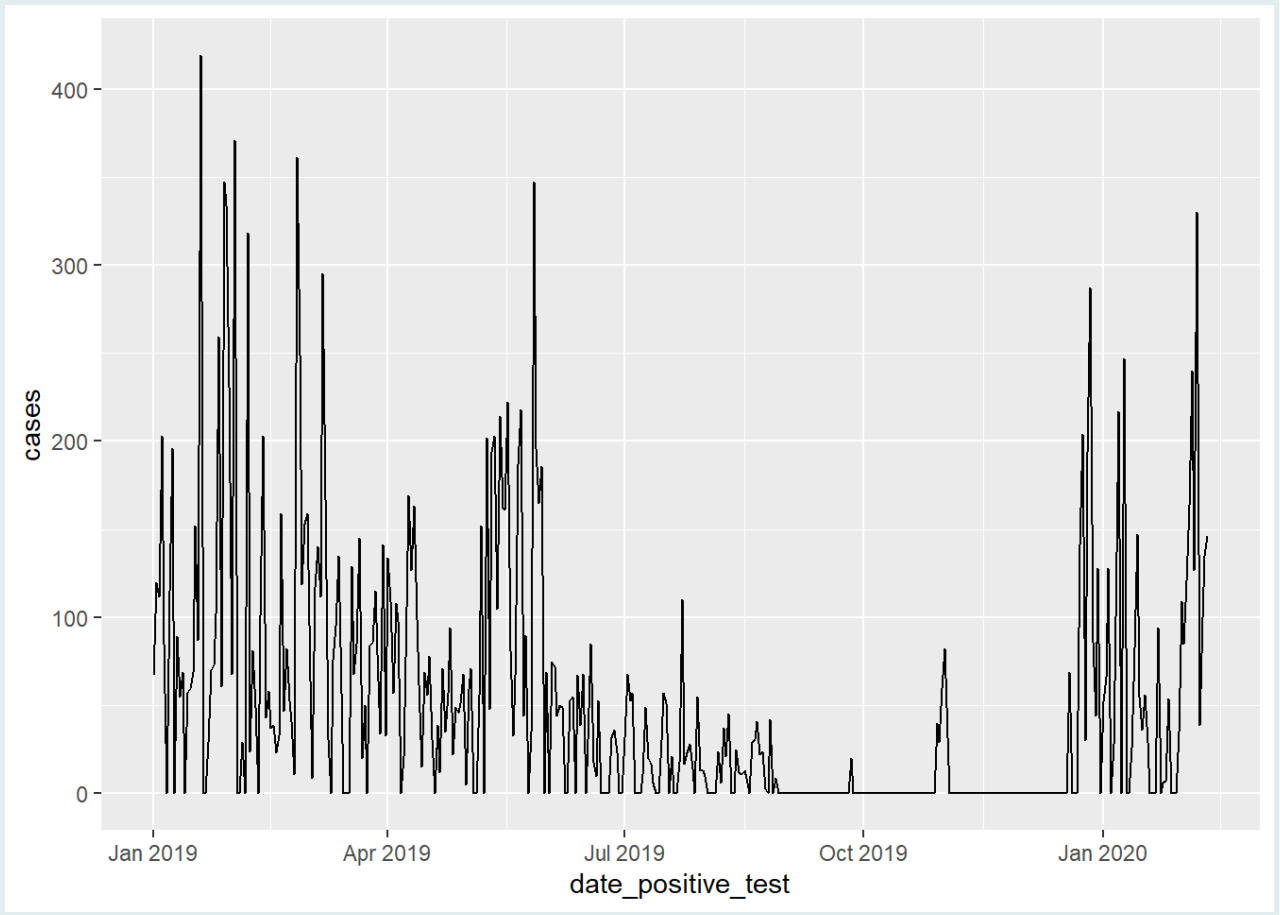
```
## # A tibble: 406 × 2
##    date_positive_test cases
##    <date>             <int>
##  1 2019-01-01            67
##  2 2019-01-02           120
##  3 2019-01-03           112
##  4 2019-01-04           203
##  5 2019-01-05            85
##  6 2019-01-06             0
```

```
##  7 2019-01-07           115
##  8 2019-01-08           196
##  9 2019-01-09             0
## 10 2019-01-10            89
## # ℹ 396 more rows
```

Now we have a complete incidence table with the number of cases on 406 consecutive days.

We can now plot the data to see the overall trend:

```
# Create a basic epicurve using ggplot2
ggplot(mal_notif_count, aes(x = date_positive_test, y = cases)) +
  geom_line()
```

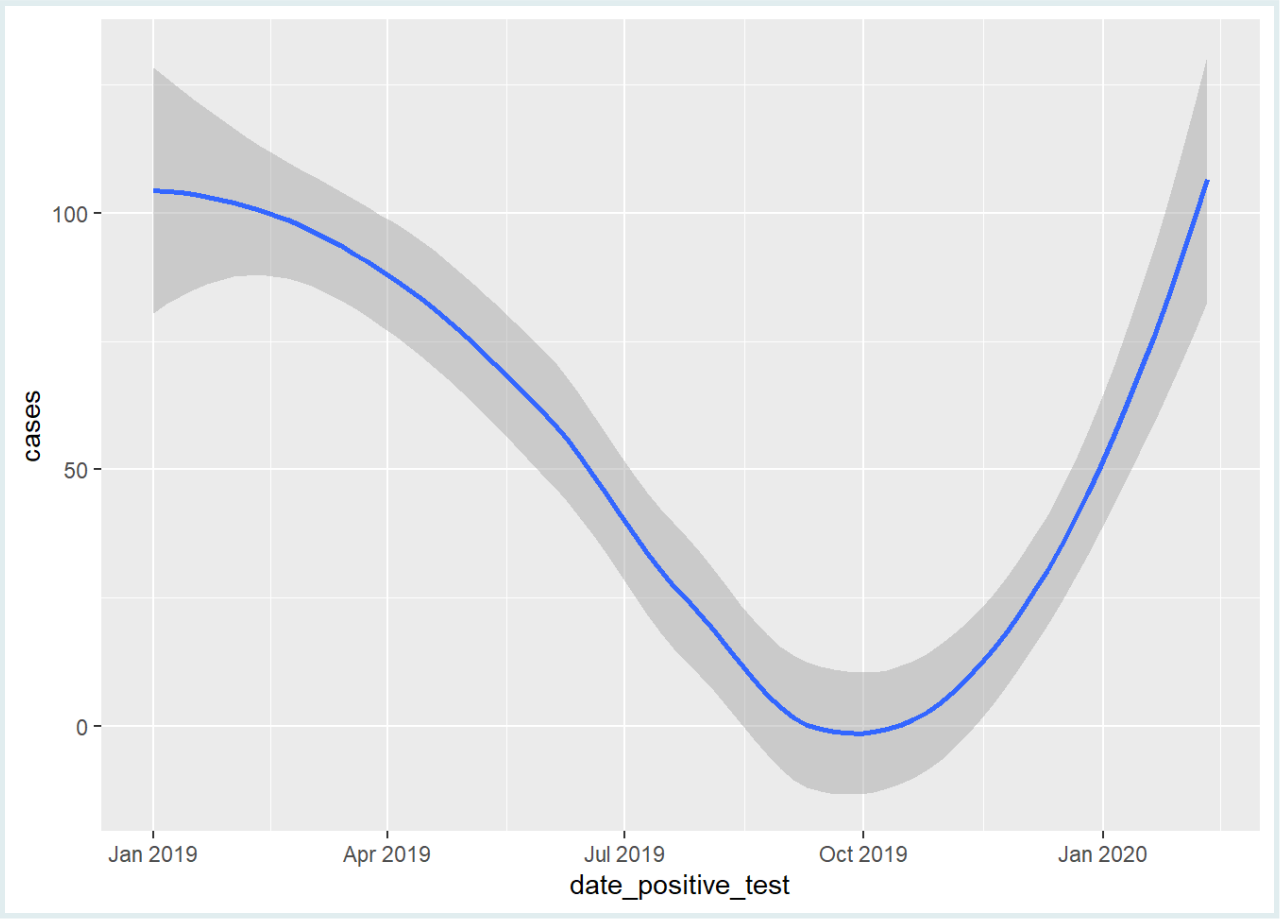


We have a valid epicurve, but as you may notice, the daily variability makes it hard to see the overall trend. Let's smooth things out.

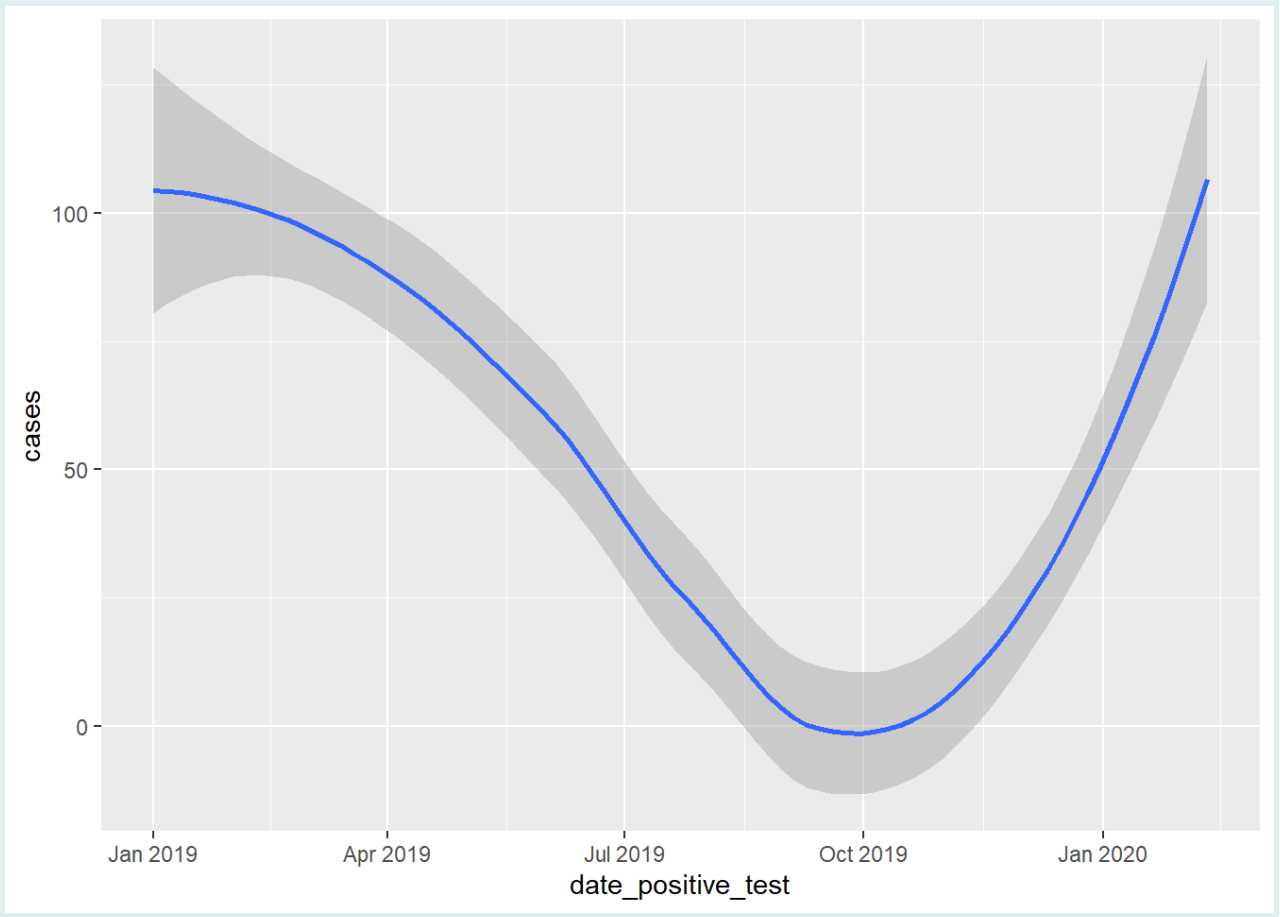
Smoothing with `geom_smooth()`

One option for smoothing is the `geom_smooth()` function, which can perform local regression with `loess` to smooth out the time series. Let's try it out:

```
ggplot(mal_notif_count, aes(x = date_positive_test, y = cases)) +
  geom_smooth()
```



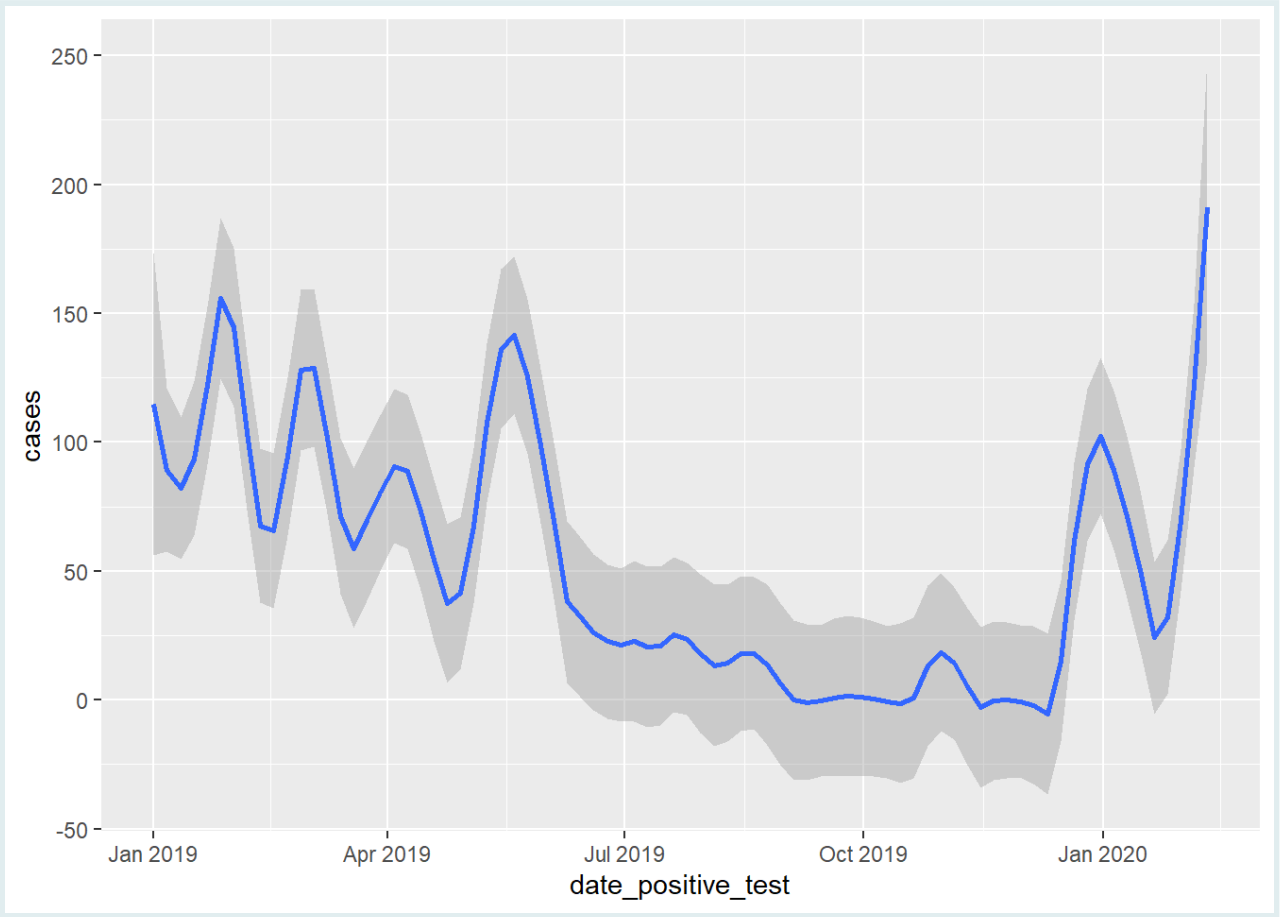

```
# Or we can specify the method explicitly
ggplot(mal_notif_count, aes(x = date_positive_test, y = cases)) +
  geom_smooth(method = "loess")
```



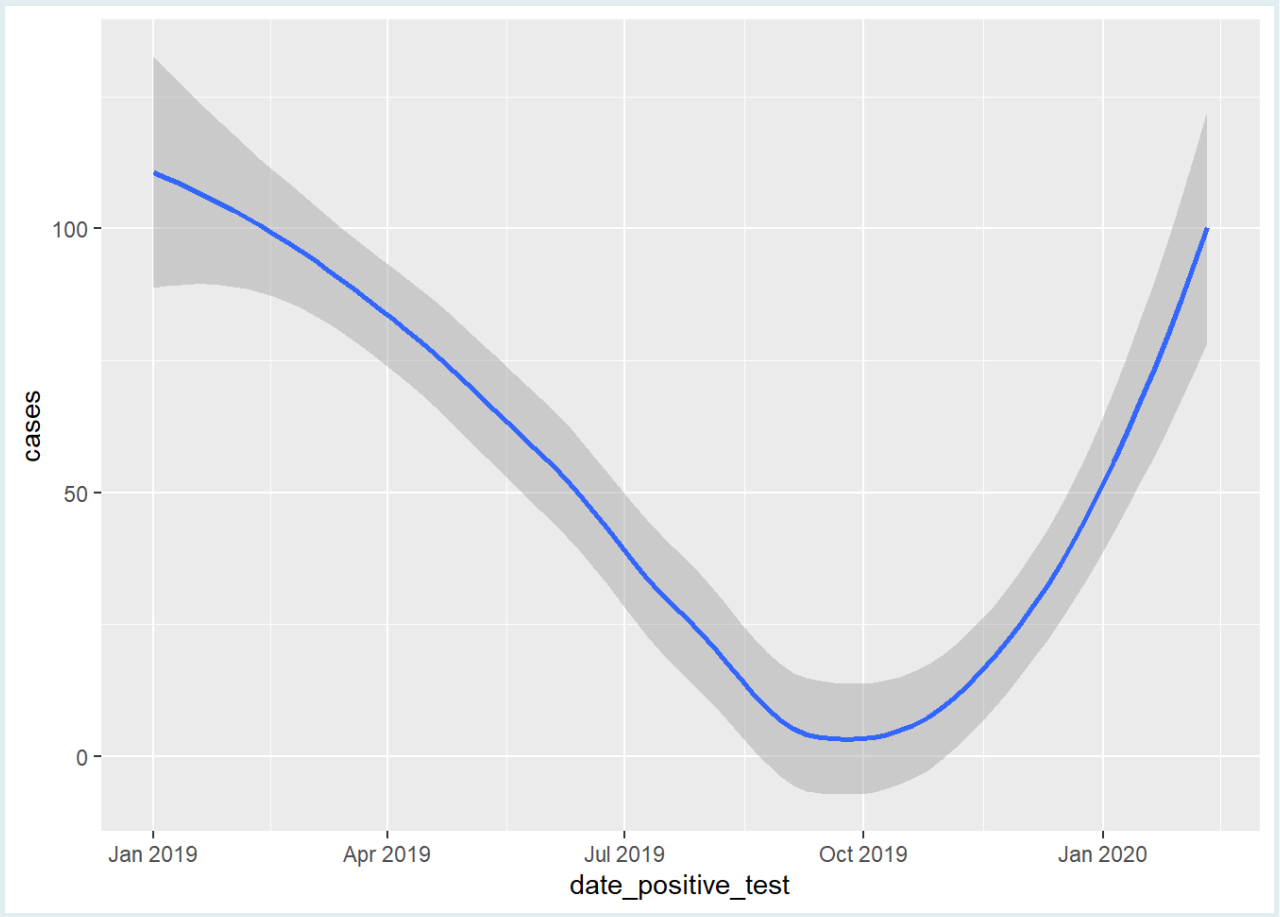
The `loess` methods, which stands for locally weighted scatterplot smoothing, fits a smooth curve to the data by calculating weighted averages for nearby points.

You can adjust the sensitivity of the smoothing by modifying the `span` argument. A span of 0.1 will result in a more sensitive smoothing, while a span of 0.9 will result in a less sensitive smoothing.

```
# Adjust the sensitivity of the smoothing
ggplot(mal_notif_count, aes(x = date_positive_test, y = cases)) +
  geom_smooth(method = "loess", span = 0.1)
```




```
ggplot(mal_notif_count, aes(x = date_positive_test, y = cases)) +
  geom_smooth(method = "loess", span = 0.9)
```



Smoothing by Aggregating

Another way to smooth data is by aggregating it—grouping the data into larger time intervals and calculating summary statistics for each interval.

We already saw this at the start of the lesson, when we aggregated quarterly data to yearly data.

Let's apply it again, this time aggregating daily malaria incidence to monthly incidence. To do this, we employ the `floor_date()` function from the `lubridate` package to round the dates down to the nearest month:

```
mal_notif_count %>%
  mutate(month = floor_date(date_positive_test, unit = "month"))
```

```
## # A tibble: 406 × 3
##    date_positive_test cases month
##    <date>             <int> <date>
##  1 2019-01-01            67 2019-01-01
##  2 2019-01-02           120 2019-01-01
##  3 2019-01-03           112 2019-01-01
##  4 2019-01-04           203 2019-01-01
##  5 2019-01-05            85 2019-01-01
```

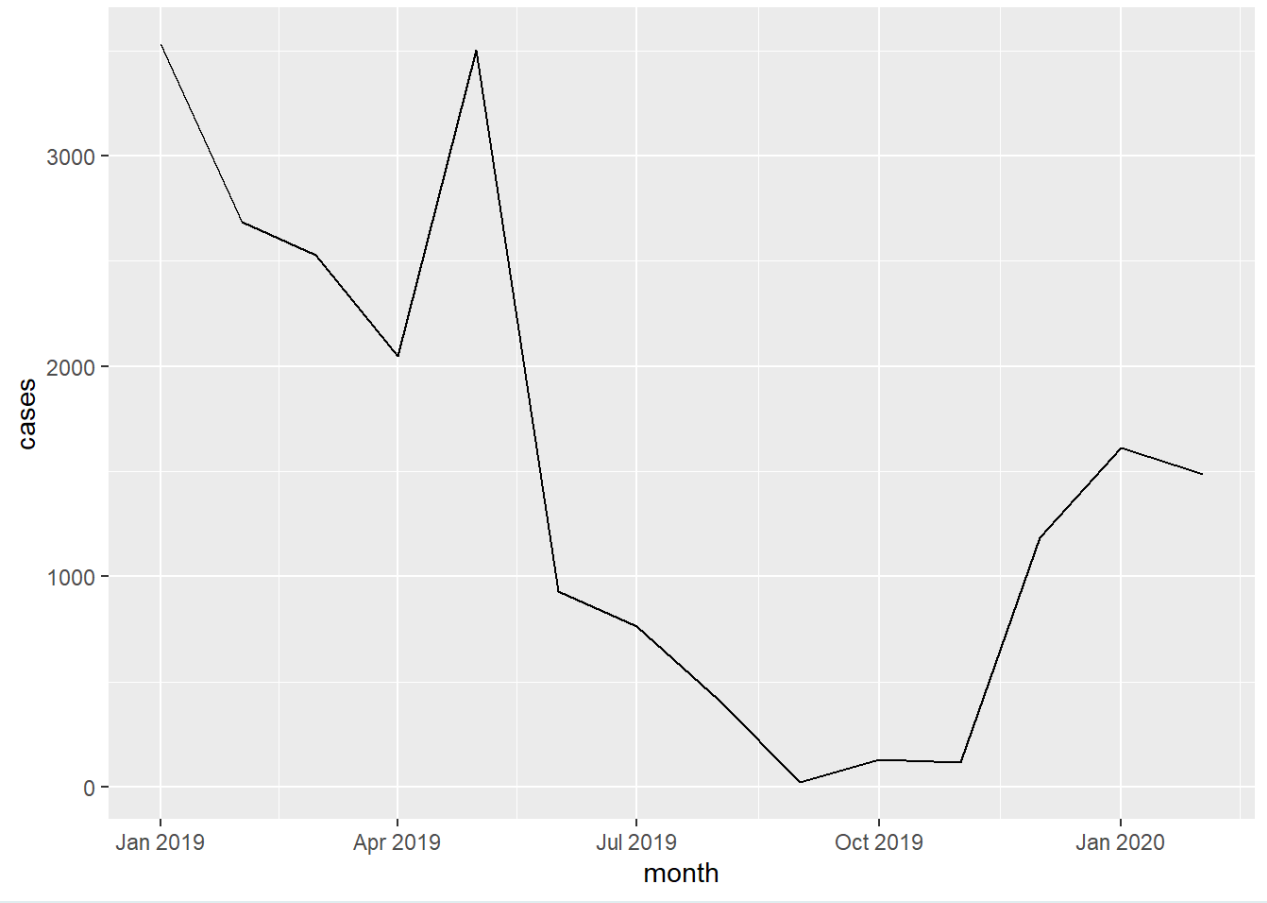
```
##  6 2019-01-06             0 2019-01-01
##  7 2019-01-07           115 2019-01-01
##  8 2019-01-08           196 2019-01-01
##  9 2019-01-09             0 2019-01-01
## 10 2019-01-10            89 2019-01-01
## # ℹ 396 more rows
```

We can then use `group_by()` and `summarize()` to calculate the total number of cases per month:

```
mal_monthly <-
  mal_notif_count %>%
  mutate(month = floor_date(date_positive_test, unit = "month")) %>%
  group_by(month) %>%
  summarize(cases = sum(cases))
```

This gives us a monthly incidence table, which we can plot to see the overall trend:

```
ggplot(mal_monthly, aes(x = month, y = cases)) +
  geom_line()
```



Voila! A much clearer picture.

PRACTICE

Q: Smoothing HIV Death Data in Colombia

Consider this dataset of individuals who died from HIV in Colombia between 2010 and 2016, sourced from this URL.

```
colom_hiv_deaths <-
  read_csv(here("data/colombia_hiv_deaths_2010_to_2016.csv")) %>%
  mutate(date_death = ymd(paste(death_year, death_month,
         death_day, sep = "-")))
colom_hiv_deaths
```

```
## # A tibble: 445 × 26
##    municipality_type death_location  birth_date birth_year
##    <chr>             <chr>           <date>          <dbl>
##  1 Municipal head    Hospital/clinic 1956-05-26       1956
##  2 Municipal head    Hospital/clinic 1983-10-10       1983
##  3 Municipal head    Hospital/clinic 1967-11-22       1967
##  4 Municipal head    Home/address    1964-03-14       1964
##  5 Municipal head    Hospital/clinic 1960-06-27       1960
##  6 Municipal head    Hospital/clinic 1982-03-23       1982
##  7 Municipal head    Hospital/clinic 1964-12-09       1964
##  8 Municipal head    Hospital/clinic 1975-01-15       1975
##  9 Municipal head    Hospital/clinic 1988-02-15       1988
## 10 Municipal head    Hospital/clinic NA                 NA
## # ℹ 435 more rows
## # ℹ 22 more variables: birth_month <chr>, …
```

Using the steps taught above:

1. Create a table that counts HIV-related deaths per month.
2. Plot an epicurve of the deaths per month
3. Apply `geom_smooth` to the epicurve for a smoother visualization. Ensure you choose an appropriate span for smoothing.

Smoothing with Rolling Averages

Another technique to smooth noisy time series data is to calculate **rolling averages**. This takes the average of a fixed number of points, centered around each data point.

The `rollmean()` function from the {zoo} package will be your primary work-horse for calculating rolling means.

Let's apply a 14 day rolling average to smooth our daily malaria case data:

```
mal_notif_count <- mal_notif_count %>%
  mutate(roll_cases = rollmean(cases, k = 14, fill = NA))
mal_notif_count
```

```
## # A tibble: 406 × 3
##    date_positive_test cases roll_cases
##    <date>             <int>      <dbl>
##  1 2019-01-01            67       NA
##  2 2019-01-02           120       NA
##  3 2019-01-03           112       NA
##  4 2019-01-04           203       NA
##  5 2019-01-05            85       NA
##  6 2019-01-06             0       NA
##  7 2019-01-07           115       83.4
##  8 2019-01-08           196       82.9
##  9 2019-01-09             0       79.3
## 10 2019-01-10            89       82.1
## # ℹ 396 more rows
```

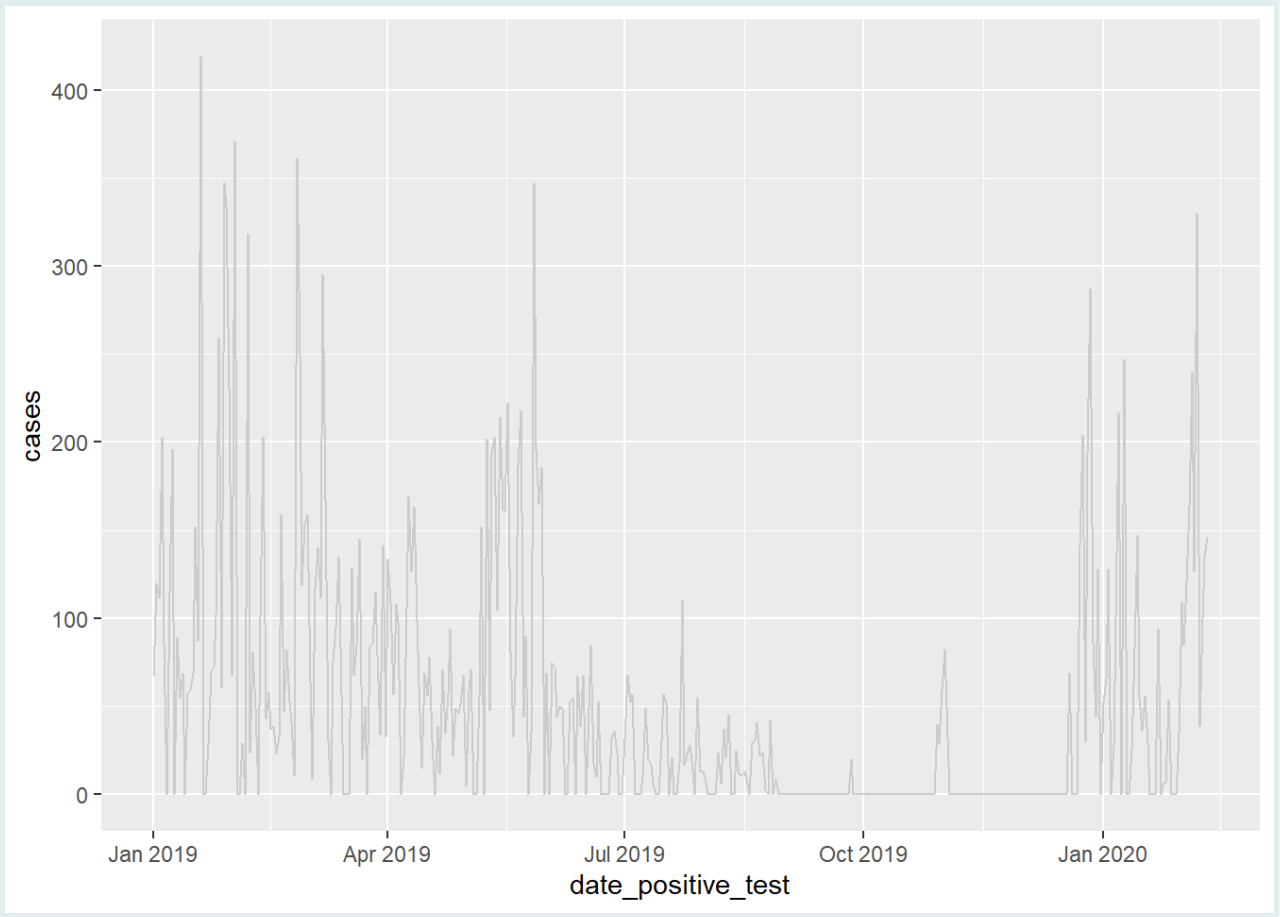
The key arguments are:

- `x`: The time series to smooth
- `k`: The number of points before and after to average
- `fill`: How to handle missing data within each window

This calculates the 14-day moving average, leaving missing data as NA. Notice that the first 6 days are NA, since there are not enough points to average over (with a `k` of 14, we need 7 days before and 7 days after each point to calculate the rolling average.

Let's plot it:

```
mal_notif_count %>%
  ggplot(aes(x = date_positive_test, y = cases)) +
  geom_line(color = "gray80")
```



Commonly, you will be asked to plot a rolling average of the *past* 1 or 2 weeks. For this, you must set the `align` argument to `"right"`:

```
mal_notif_count_right <-
  mal_notif_count %>%
  mutate(roll_cases = rollmean(cases, k = 14, fill = NA, align = "right"))

head(mal_notif_count_right, 15)
```

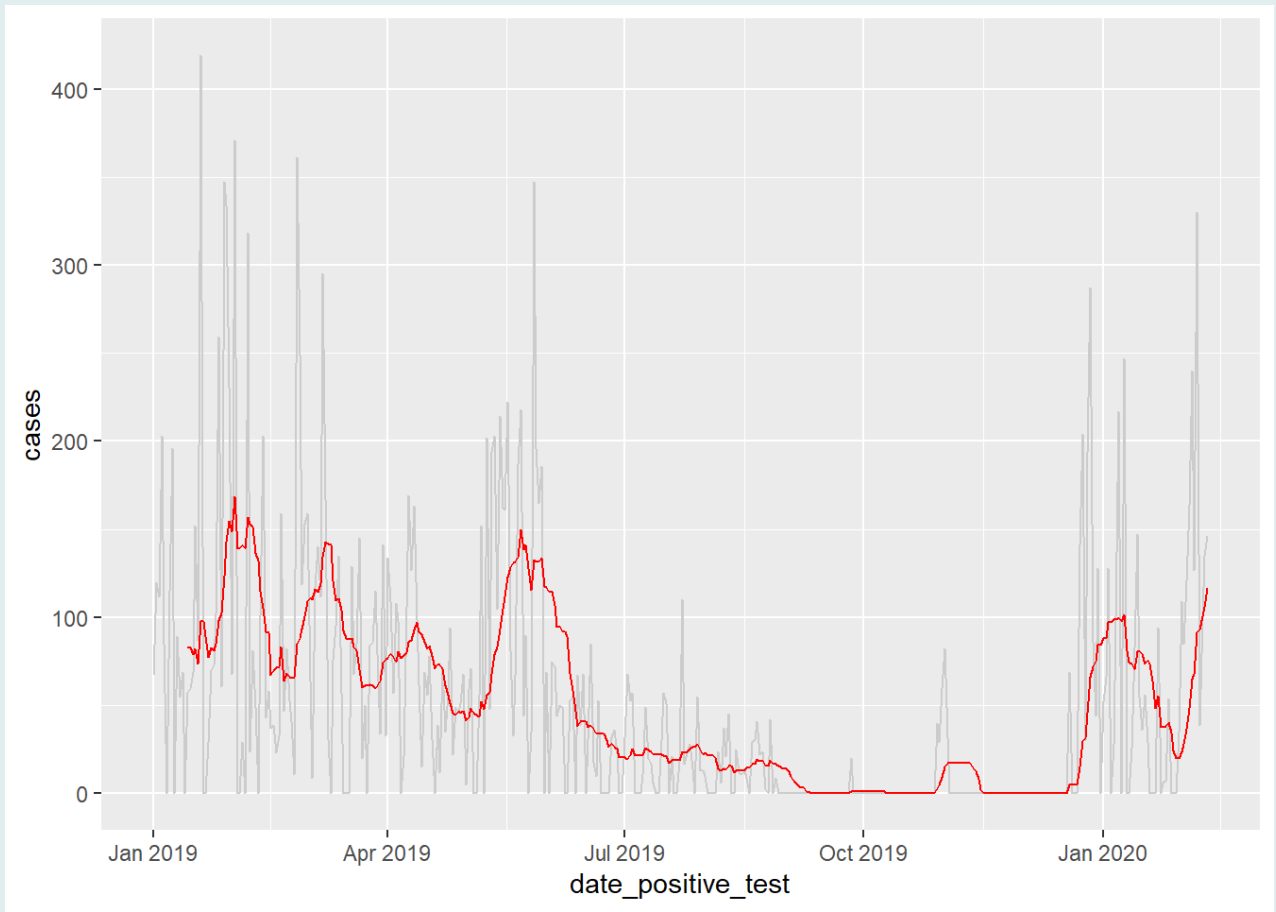
```
## # A tibble: 15 × 3
##    date_positive_test cases roll_cases
##    <date>             <int>      <dbl>
##  1 2019-01-01            67         NA
##  2 2019-01-02           120         NA
##  3 2019-01-03           112         NA
##  4 2019-01-04           203         NA
##  5 2019-01-05            85         NA
##  6 2019-01-06             0         NA
##  7 2019-01-07           115         NA
##  8 2019-01-08           196         NA
##  9 2019-01-09             0         NA
## 10 2019-01-10            89         NA
## 11 2019-01-11            55         NA
## 12 2019-01-12            69         NA
## 13 2019-01-13             0         NA
```

```
## 14 2019-01-14            57        83.4
## 15 2019-01-15            60        82.9
```

Notice that now the first 13 days are NA, since there are not enough points to average over. This is because we are calculating the average of the *past* 14 days, and the first 13 days do not have 14 days before them.

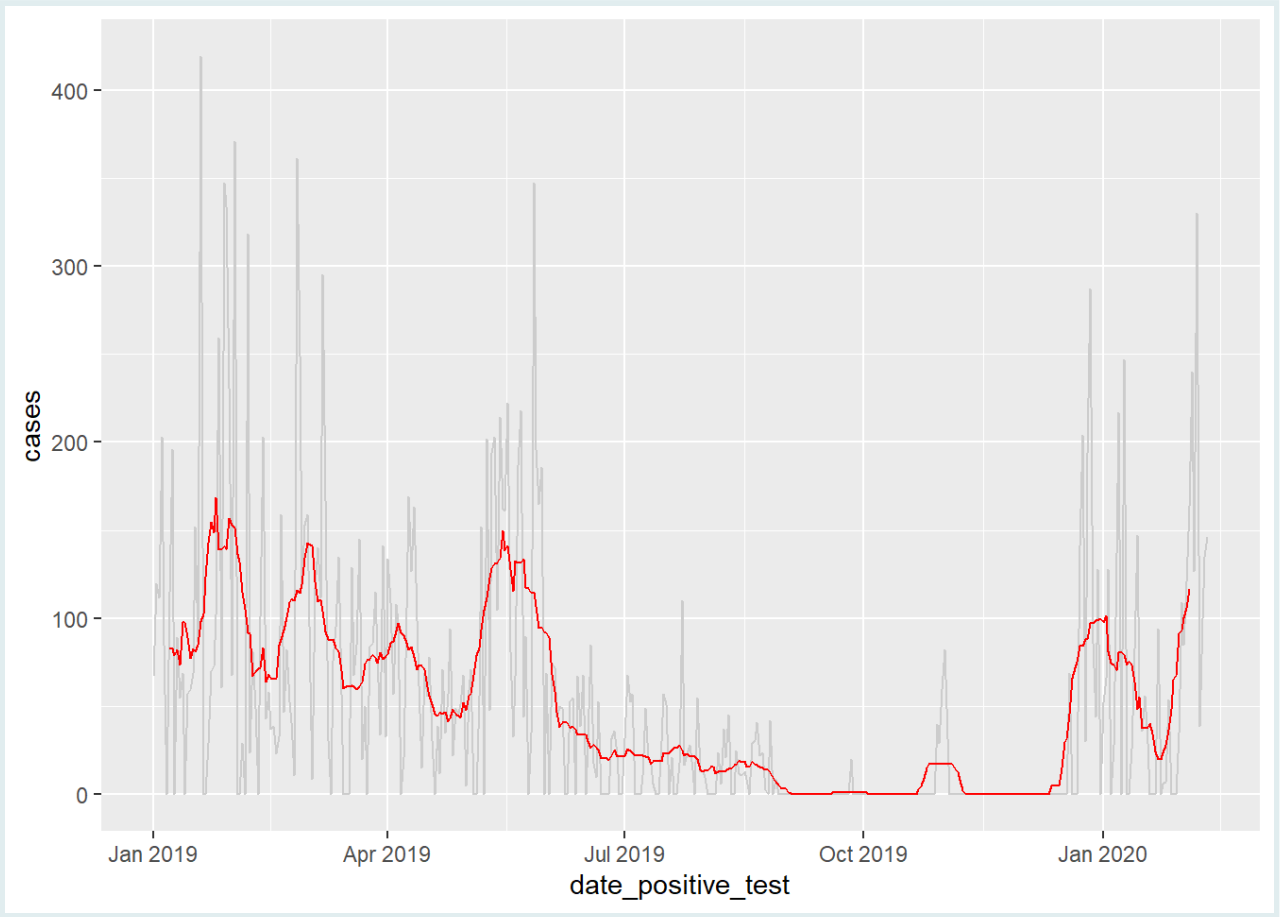
The output does not change much in this case:

```
ggplot(mal_notif_count_right, aes(x = date_positive_test, y = cases)) +
  geom_line(color = "gray80") +
  geom_line(aes(y = roll_cases), color = "red")
```



Finally, let's plot both the original and smoothed data:

```
mal_notif_count %>%
  ggplot(aes(x = date_positive_test, y = cases)) +
  geom_line(color = "gray80") +
  geom_line(aes(y = roll_cases), color = "red")
```



In summary, the `rollmean()` function lets us easily compute a rolling average over a fixed window to smooth and highlight patterns in noisy time series data.

Q: Smoothing with rolling averages

Consider again the dataset of HIV patient deaths in Colombia:

```
colom_hiv_deaths
```

```
## # A tibble: 445 × 26
##    municipality_type death_location  birth_date birth_year
##    <chr>             <chr>           <date>          <dbl>
##  1 Municipal head    Hospital/clinic 1956-05-26       1956
##  2 Municipal head    Hospital/clinic 1983-10-10       1983
##  3 Municipal head    Hospital/clinic 1967-11-22       1967
##  4 Municipal head    Home/address    1964-03-14       1964
##  5 Municipal head    Hospital/clinic 1960-06-27       1960
##  6 Municipal head    Hospital/clinic 1982-03-23       1982
##  7 Municipal head    Hospital/clinic 1964-12-09       1964
##  8 Municipal head    Hospital/clinic 1975-01-15       1975
```

```
##  9 Municipal head    Hospital/clinic 1988-02-15       1988
## 10 Municipal head    Hospital/clinic NA                 NA
## # ℹ 435 more rows
## # ℹ 22 more variables: birth_month <chr>, …
```

The following code calculates the number of deaths per day:

```
colom_hiv_deaths_per_day <-
  colom_hiv_deaths %>%
  group_by(date_death) %>%
  summarize(deaths = n()) %>%
  complete(date_death = seq.Date(min(date_death),
                                 max(date_death),
                                 by = "day"),
           fill = list(deaths = 0))

colom_hiv_deaths_per_day
```


```
## # A tibble: 2,543 × 2
##    date_death deaths
##    <date>      <int>
##  1 2010-01-05      1
##  2 2010-01-06      0
##  3 2010-01-07      0
##  4 2010-01-08      0
##  5 2010-01-09      1
##  6 2010-01-10      0
##  7 2010-01-11      1
##  8 2010-01-12      0
##  9 2010-01-13      0
## 10 2010-01-14      0
## # ℹ 2,533 more rows
```

Your task is to create a new column that calculates the rolling average of deaths per day over a 14-day period. Then, plot this rolling average alongside the raw data.

Secondary Axes

Understanding the Concept of a Secondary Y-Axis

A secondary y-axis is helpful when visualizing two different measures with distinct scales on the same plot. This approach is useful when the variables have different units or magnitudes, making direct comparison on a single scale challenging.

While some data visualization experts caution against using secondary axes, public health decision-makers often appreciate these plots.

Creating a Plot with a Secondary Y-Axis

Let's demonstrate how to create a plot with a secondary y-axis using our dataset of malaria notifications:

Step 1: Create Cumulative Case Counts

First, we'll aggregate our malaria data to calculate cumulative case counts.

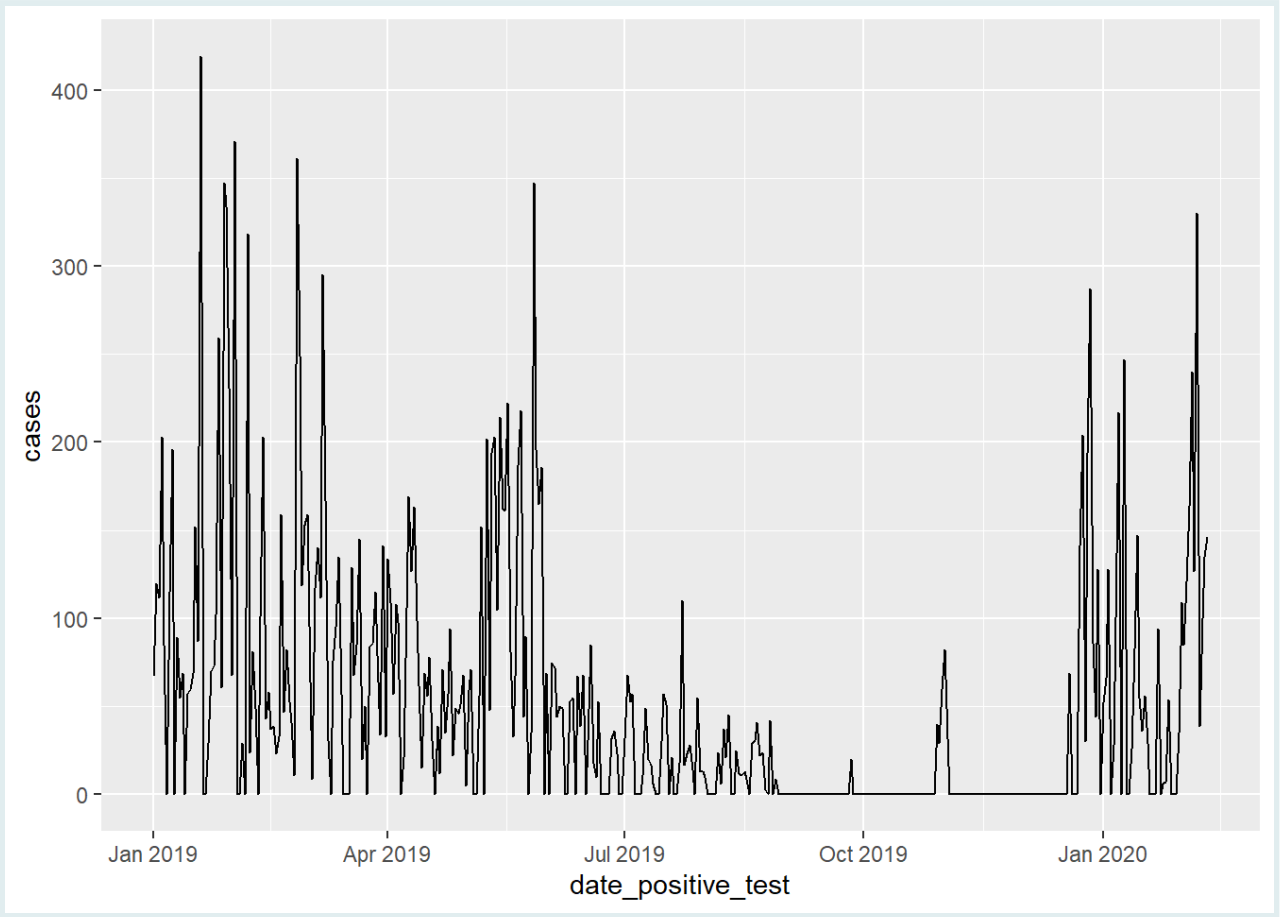
```
mal_notif_count_cumul <-
  mal_notif_count %>%
  mutate(cumul_cases = cumsum(cases))

mal_notif_count_cumul
```

```
## # A tibble: 406 × 4
##    date_positive_test cases roll_cases cumul_cases
##    <date>             <int>      <dbl>       <int>
##  1 2019-01-01            67       NA            67
##  2 2019-01-02           120       NA           187
##  3 2019-01-03           112       NA           299
##  4 2019-01-04           203       NA           502
##  5 2019-01-05            85       NA           587
##  6 2019-01-06             0       NA           587
##  7 2019-01-07           115       83.4         702
##  8 2019-01-08           196       82.9         898
##  9 2019-01-09             0       79.3         898
## 10 2019-01-10            89       82.1         987
## # ℹ 396 more rows
```

Step 2: Identifying the Need for a Secondary Y-Axis

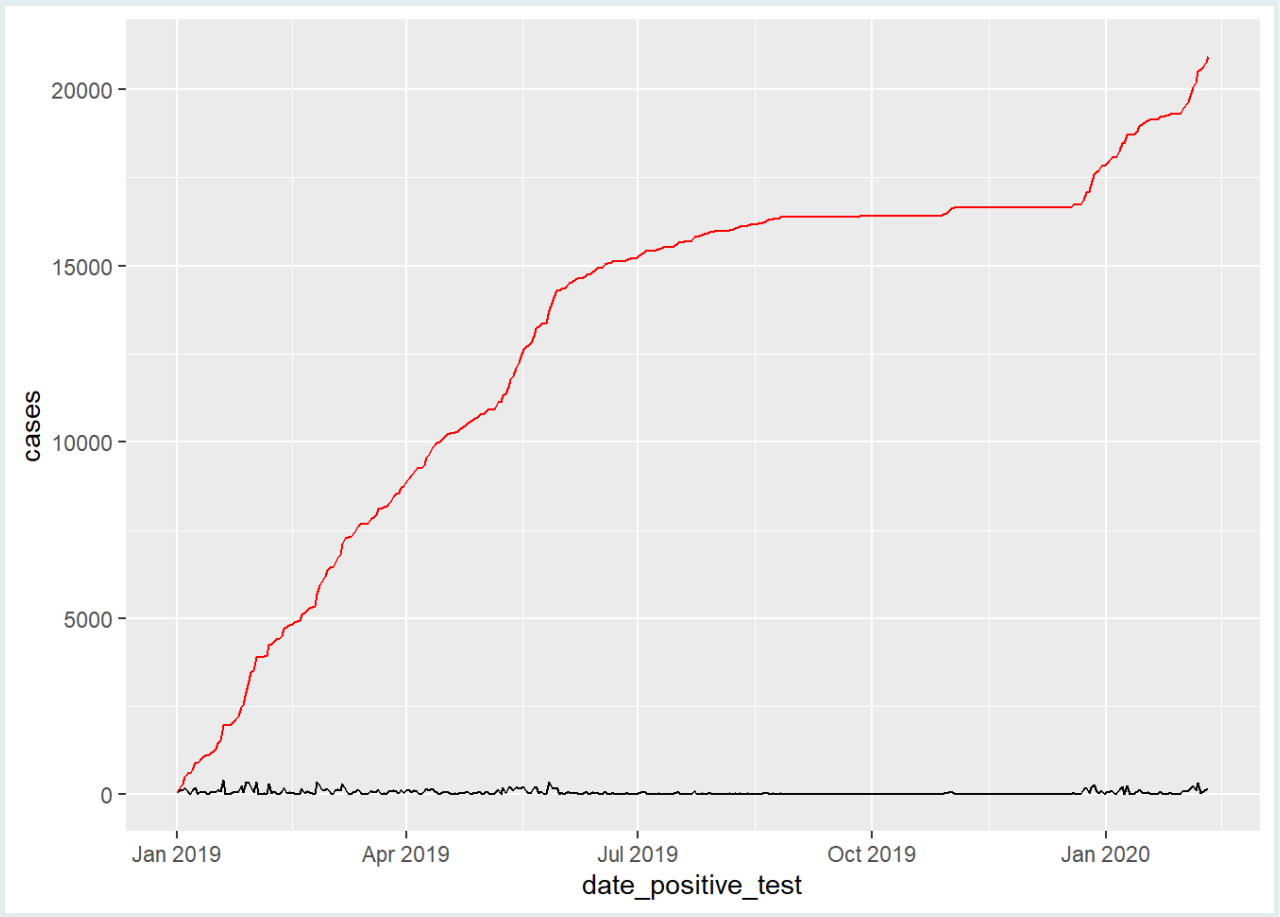
Now, we can start plotting. First, we plot just the daily cases:

```
# Plotting total malaria cases
ggplot(mal_notif_count_cumul, aes(x = date_positive_test)) +
  geom_line(aes(y = cases))
```



If we try to add cumulative cases on the same y-axis, the daily cases will be dwarfed and their magnitude will hard to read due to the much larger scale of cumulative data:

```
# Adding cumulative malaria cases to the plot
ggplot(mal_notif_count_cumul, aes(x = date_positive_test)) +
  geom_line(aes(y = cases)) +
  geom_line(aes(y = cumul_cases), color = "red")
```



To effectively display both sets of data, we must introduce a secondary y-axis.

Step 3: Calculating and Applying the Scale Factor

Before adding a secondary axis, we need to determine a *scale factor* by comparing the ranges of cases and cumulative cases.

The scale factor is typically the ratio of the maximum values of the two datasets. Let's see what the maximum values are for each variable:

```
max(mal_notif_count_cumul$cases)
```

```
## [1] 419
```

```
max(mal_notif_count_cumul$cumul_cases)
```

```
## [1] 20939
```

With a maximum or around 20000 for the cumulative cases, and about 400 for the daily cases, we can see that the cumulative cases are about 50 times larger than the

daily cases, so our scale factor will be about 50.

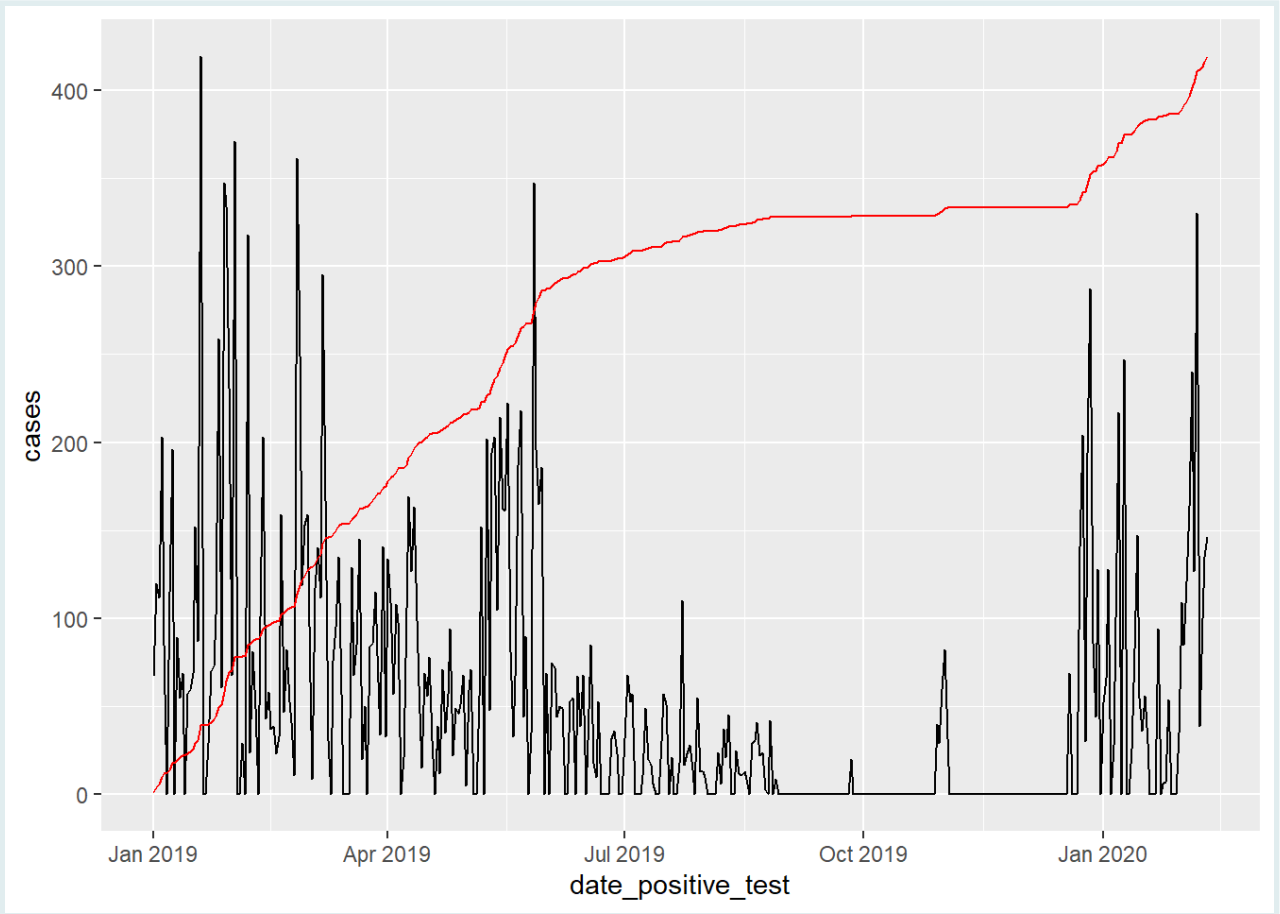
More precisely, the scale factor will be:

```
max(mal_notif_count_cumul$cumul_cases) / max(mal_notif_count_cumul$cases)
```

```
## [1] 49.97375
```

We'll need to divide the cumulative cases by this ratio to force the two variables to be on a similar scale:

```
ggplot(mal_notif_count_cumul, aes(x = date_positive_test)) +
  geom_line(aes(y = cases)) +
  geom_line(aes(y = cumul_cases / 49.97), color = "red") # divide by scale
        factor
```



Great! Now we can see both sets of data clearly on the same plot, and their maximum points are aligned. However, the y-axis is no longer relevant for the red cumulative cases line. We need to add a secondary y-axis for this.

SIDE NOTE

Normally, you would assign the scale factor to a variable, and then use that variable in the `geom_line()` function. In this case, we're typing it out directly in the function for easier understanding.

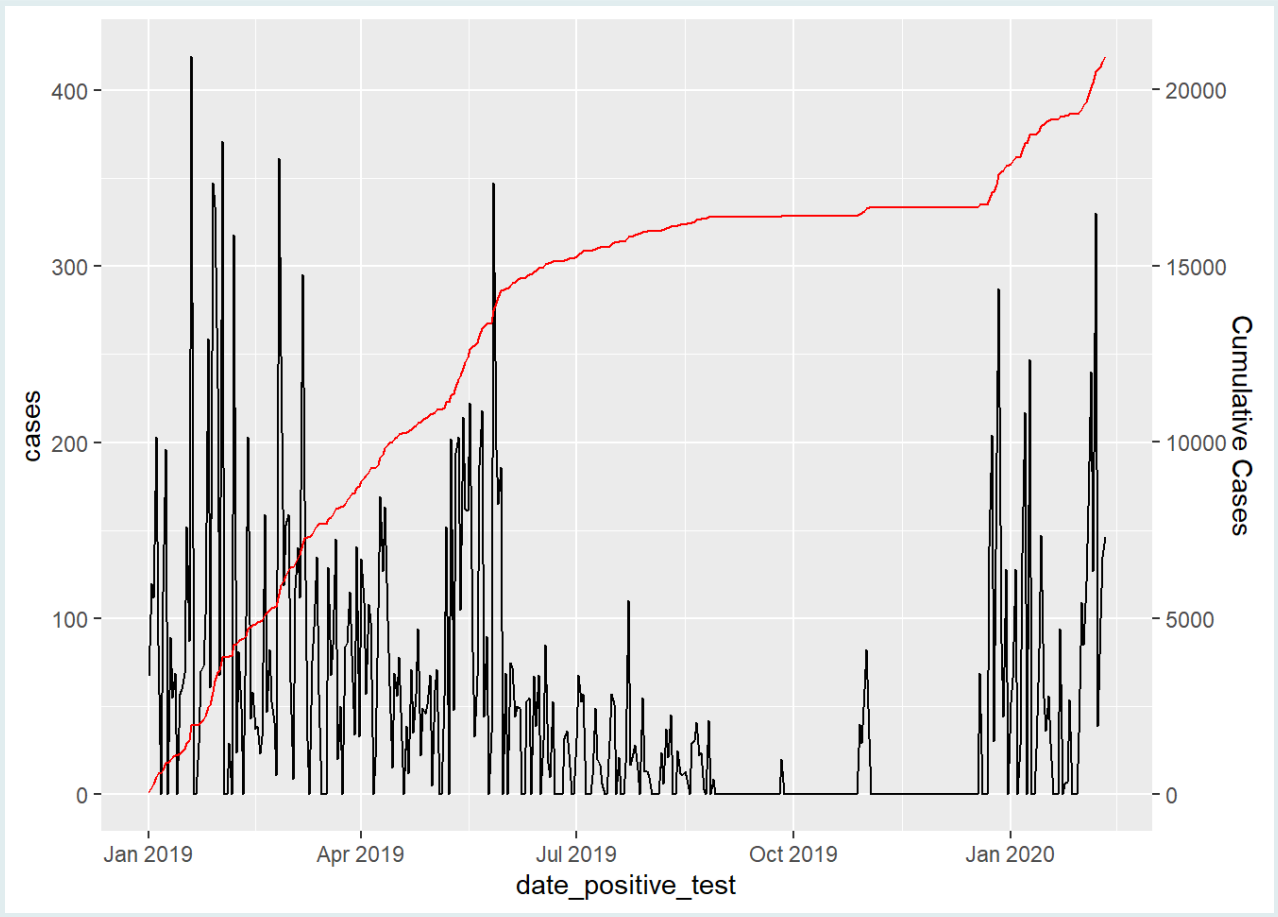
Step 4: Adding the Secondary Y-Axis

We'll use the `sec_axis()` function from {ggplot2}. The key arguments are `trans`, which indicates how much to multiply or divide the original y axis, and `name`, which specifies the name of the secondary axis.

In our case, we want the secondary axis to be about 49.97 times larger than the original axis, so we'll use `trans = ~ .x * 49.97`. (The `~` symbol is a special operator that tells R to treat the expression that follows it as a function, whose input is indicated by the `.x` symbol.)

Let's implement this:

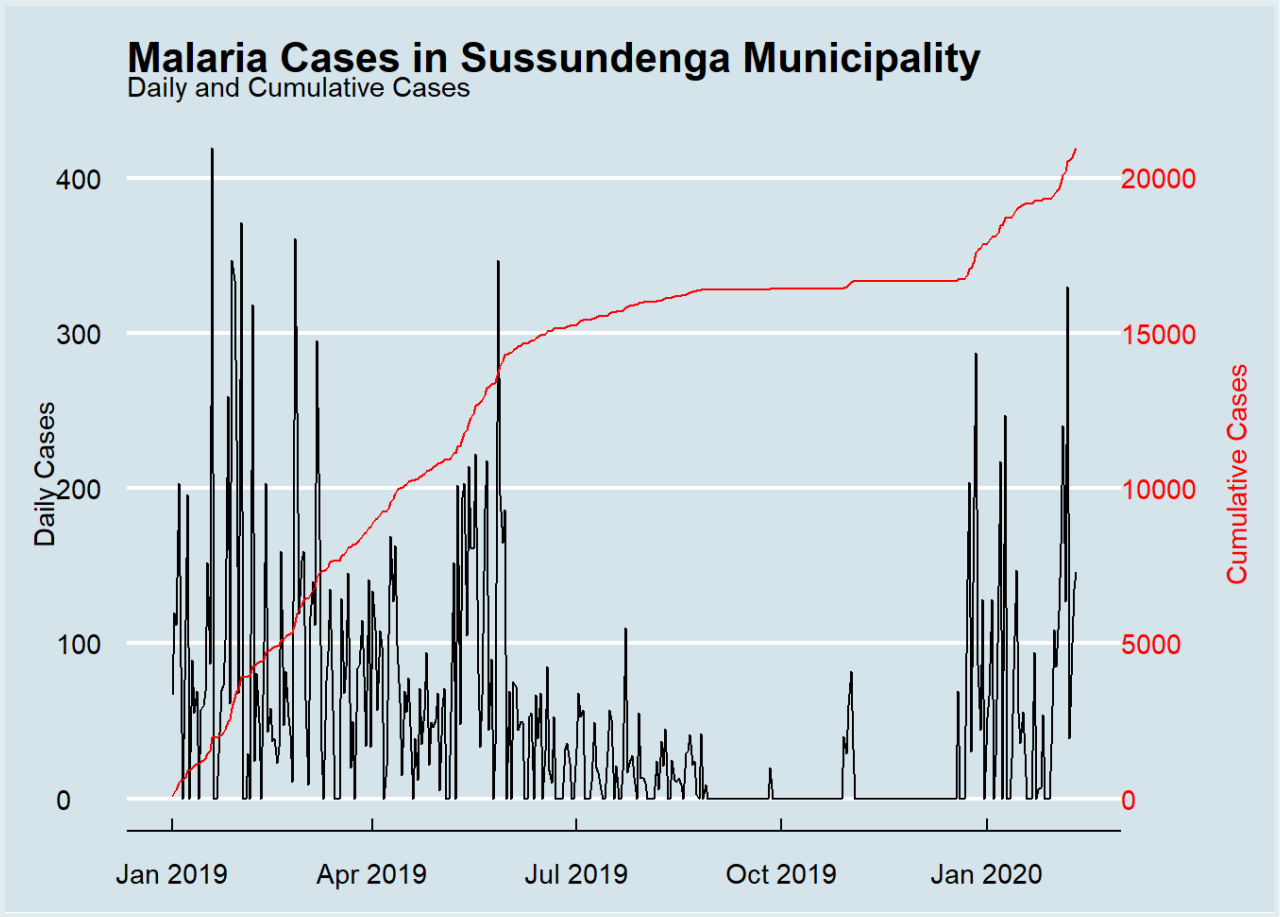
```
# Add a secondary y-axis
ggplot(mal_notif_count_cumul, aes(x = date_positive_test)) +
  geom_line(aes(y = cases)) +
  geom_line(aes(y = cumul_cases / 49.97), color = "red") +
  scale_y_continuous(sec.axis = sec_axis(trans = ~ .x * 49.97,
                                         name = "Cumulative Cases"))
```



Step 5: Enhancing Plot Readability

To improve readability, we'll make the secondary axis labels red, matching the color of the cumulative cases line, and we'll add some additional formatting to the plot:

```
# Finalizing the plot with color-coordinated axes
ggplot(mal_notif_count_cumul, aes(x = date_positive_test)) +
  geom_line(aes(y = cases)) +
  geom_line(aes(y = cumul_cases / 49.97), color = "red") +
  scale_y_continuous(
    name = "Daily Cases",
    sec.axis = sec_axis(~ . * 49.97, name = "Cumulative Cases")
  ) +
  labs(title = "Malaria Cases in Sussundenga Municipality",
       subtitle = "Daily and Cumulative Cases",
       x = NULL) +
  theme_economist() +
  theme(axis.text.y.right = element_text(color = "red"),
        axis.title.y.right = element_text(color = "red"))
```



All done! We've successfully added a secondary y-axis to a plot, enabling the comparison of two datasets with different scales in a single visualization.

Q: Secondary axes

Revisit the dataset `colom_hiv_deaths_per_day`.

PRACTICE (in RMD)

```
colom_hiv_deaths_per_day
```

```
## # A tibble: 2,543 × 2
##    date_death deaths
##    <date>      <int>
##  1 2010-01-05      1
##  2 2010-01-06      0
##  3 2010-01-07      0
##  4 2010-01-08      0
##  5 2010-01-09      1
##  6 2010-01-10      0
##  7 2010-01-11      1
##  8 2010-01-12      0
```

```
##  9 2010-01-13      0
## 10 2010-01-14      0
## # ℹ 2,533 more rows
```

Your task is to create a plot with two y-axes: one for the daily deaths and another for the cumulative deaths in Colombia.

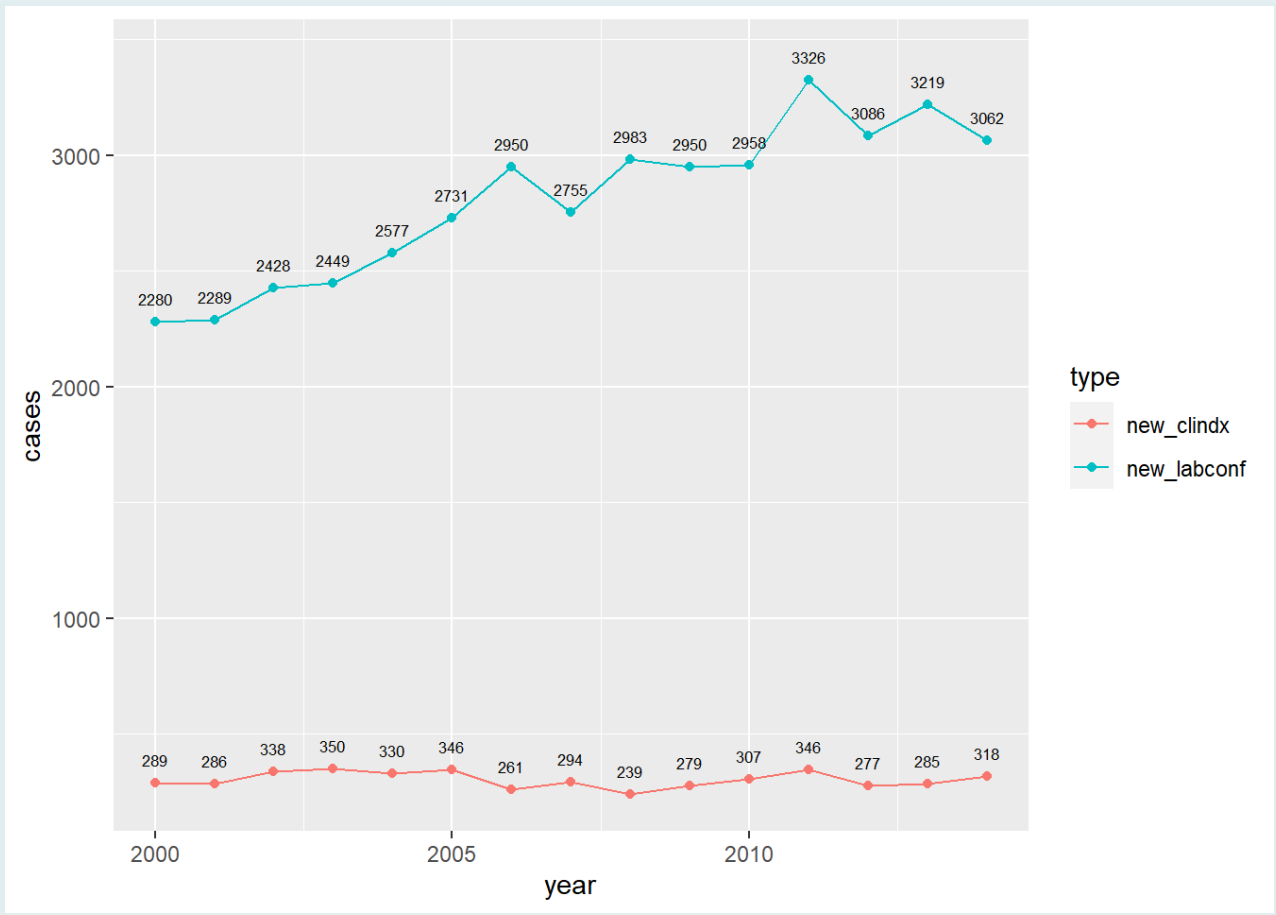
Wrap up

In this lesson, you developed key skills for wrangling, visualizing, and enhancing time series data to uncover and communicate meaningful trends over time. These skills will come in handy as you continue to explore and analyze time series data in R.

Answer Key

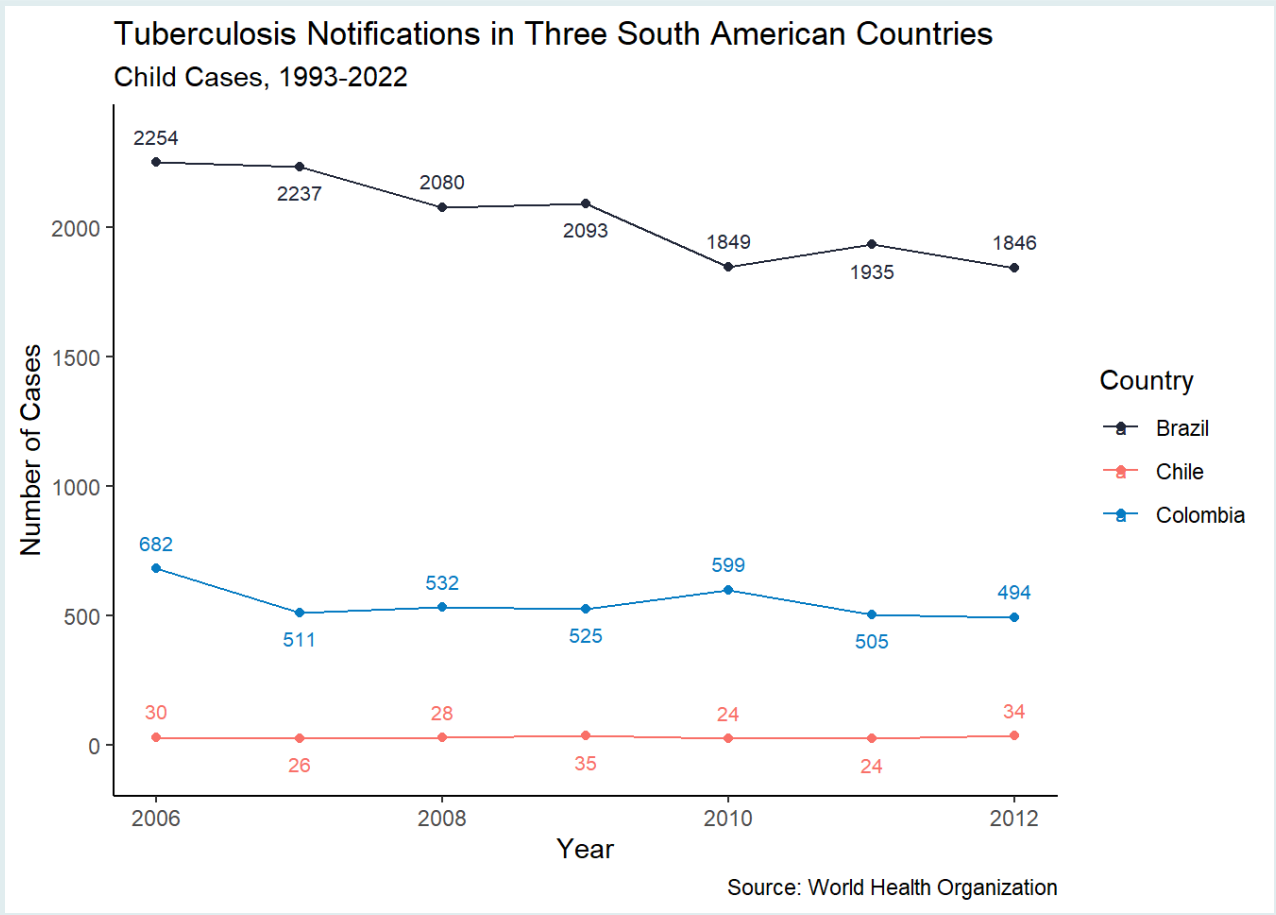
Q: Reshaping and Plotting TB Data

```
tb_benin_long <- tb_data_benin %>%
  pivot_longer(cols = c("new_clindx", "new_labconf")) %>%
  rename(type = name, cases = value)

ggplot(tb_benin_long, aes(x = year, y = cases, colour = type, group = type)) +
  geom_line() +
  geom_point() +
  geom_text(aes(label = cases), size = 2.2, nudge_y = 100, color = "black")
```



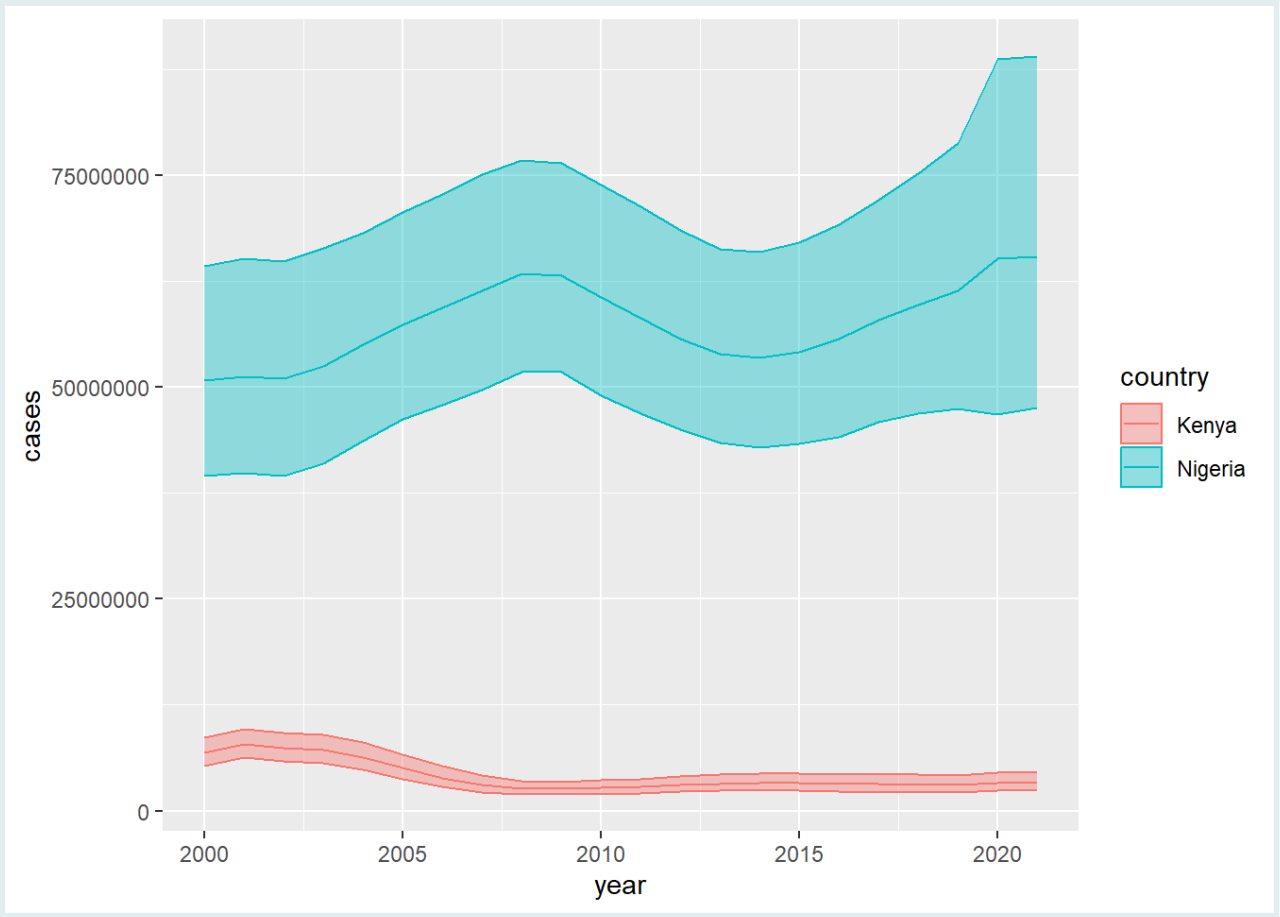
Q: Aesthetic improvements {.unlisted .unnumbered}

```
even_years_southam <- tb_child_cases_southam %>%
  filter(year %% 2 == 0) # Keep only years that are multiples of 2

odd_years_southam <- tb_child_cases_southam %>%
  filter(year %% 2 == 1) # Keep only years that are not multiples of 2


tb_child_cases_southam %>%
  ggplot(aes(x = year, y = tb_cases_children, color = country)) +
  geom_line() +
  geom_point() +
  geom_text(data = even_years_southam, aes(label = tb_cases_children),
            nudge_y = 100, size = 2.8) +
  geom_text(data = odd_years_southam, aes(label = tb_cases_children),
            nudge_y = -100, size = 2.8) +
  scale_color_manual(values = c("#212738", "#F97068", "#067BC2")) +
  labs(title = "Tuberculosis Notifications in Three South American Countries",
       subtitle = "Child Cases, 1993-2022",
       caption = "Source: World Health Organization",
       x = "Year",
       y = "Number of Cases",
       color = "Country") +
  theme_classic()
```



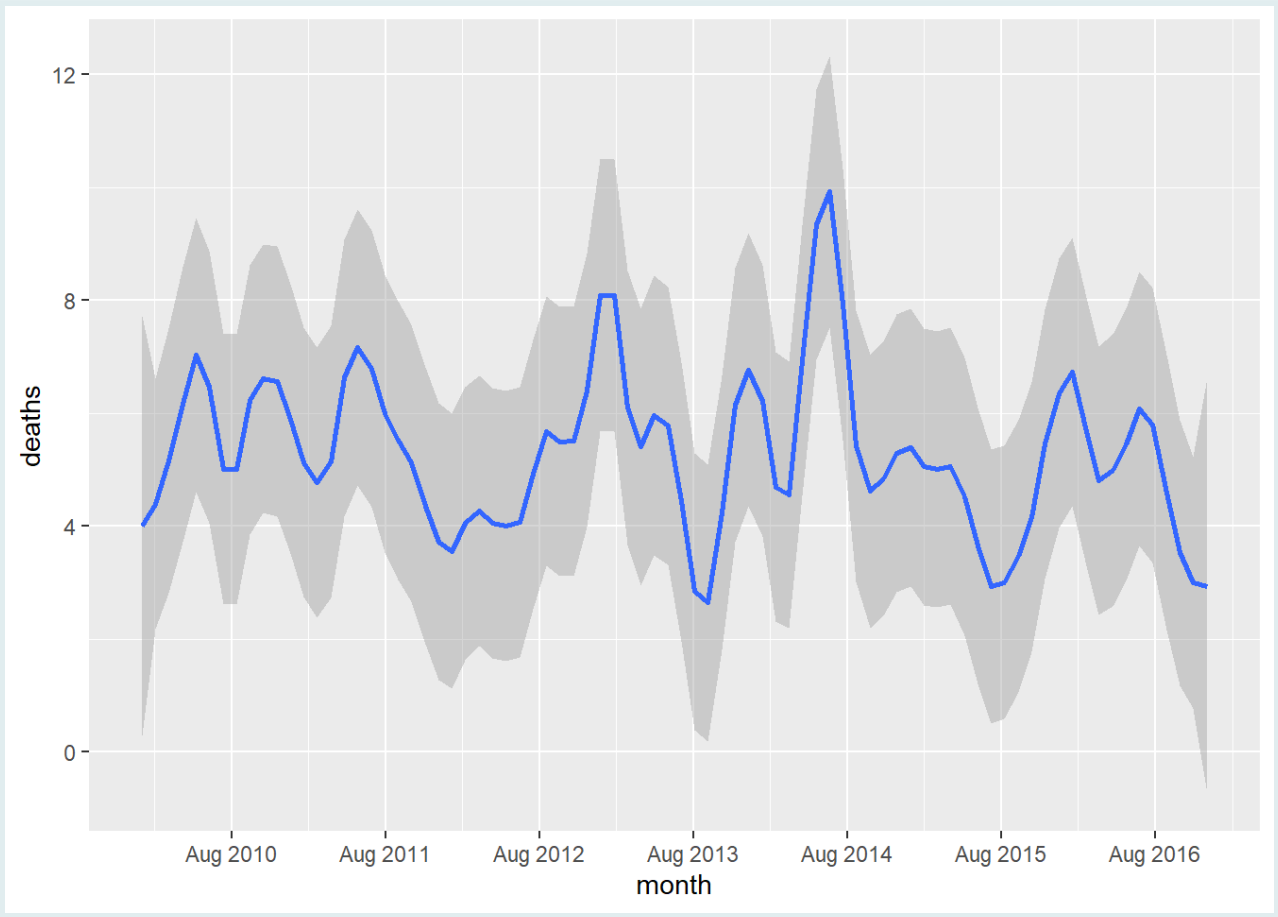
Q: Plotting confidence intervals

```
nig_ken_mal %>%
  separate(malaria_cases,
           into = c("cases", "cases_lower", "cases_upper"),
           sep = "\\(|to") %>%
  mutate(across(c("cases", "cases_lower", "cases_upper"),
                ~ str_replace_all(.x, "[^0-9]", "") %>%
                  as.numeric()
                )) %>%
  ggplot(aes(x = year, y = cases, color = country, fill = country)) +
  geom_line() +
  geom_ribbon(aes(ymin = cases_lower, ymax = cases_upper), alpha = 0.4)
```



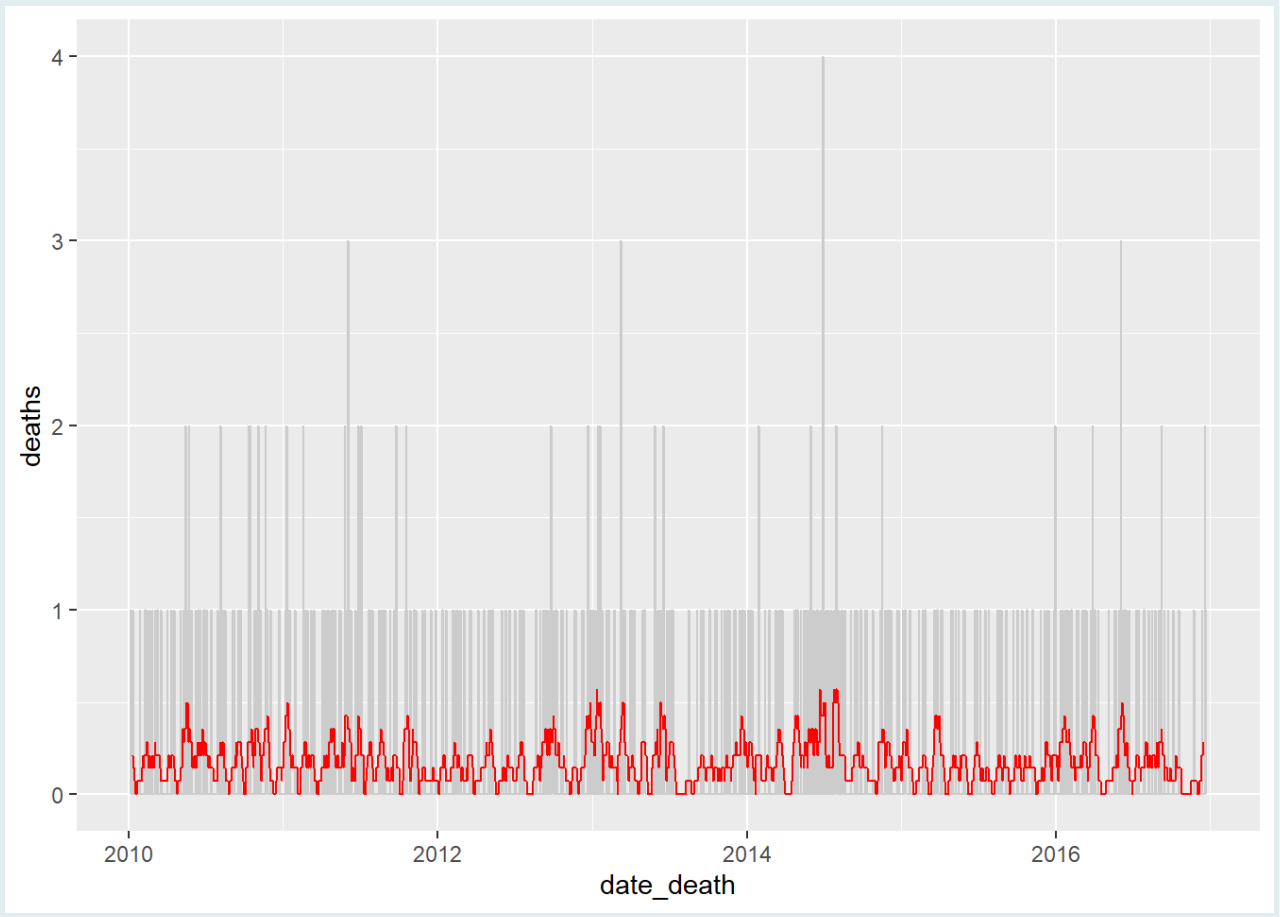
Q: Smoothing HIV Death Data in Colombia

```
hiv_monthly_deaths_table <-
  colom_hiv_deaths %>%
  # Aggregate data to count deaths per month
  mutate(month = floor_date(date_death, unit = "month")) %>%
  group_by(month) %>%
  summarize(deaths = n())

# Create the epicurve
ggplot(hiv_monthly_deaths_table, aes(x = month, y = deaths)) +
  # Apply smoothing to the curve
  geom_smooth(method = "loess", span = 0.1) +
  scale_x_date(date_breaks = "12 months", date_labels = "%b %Y")
```



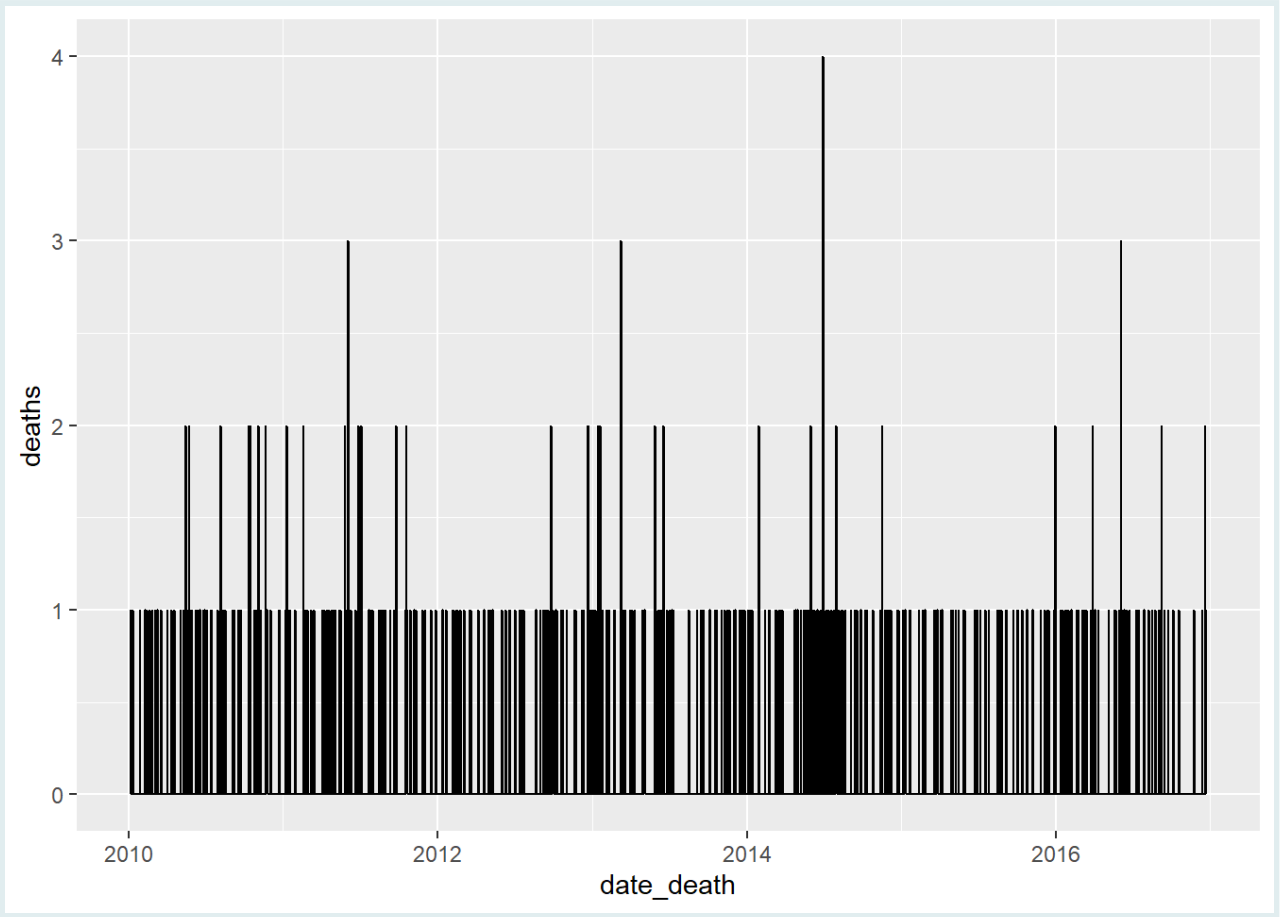
Q: Smoothing with rolling averages

```
colom_hiv_deaths_per_day %>%
  mutate(roll_deaths = rollmean(deaths, k = 14, fill = NA)) %>%
  ggplot(aes(x = date_death, y = deaths)) +
  geom_line(color = "gray80") +
  geom_line(aes(y = roll_deaths), color = "red")
```

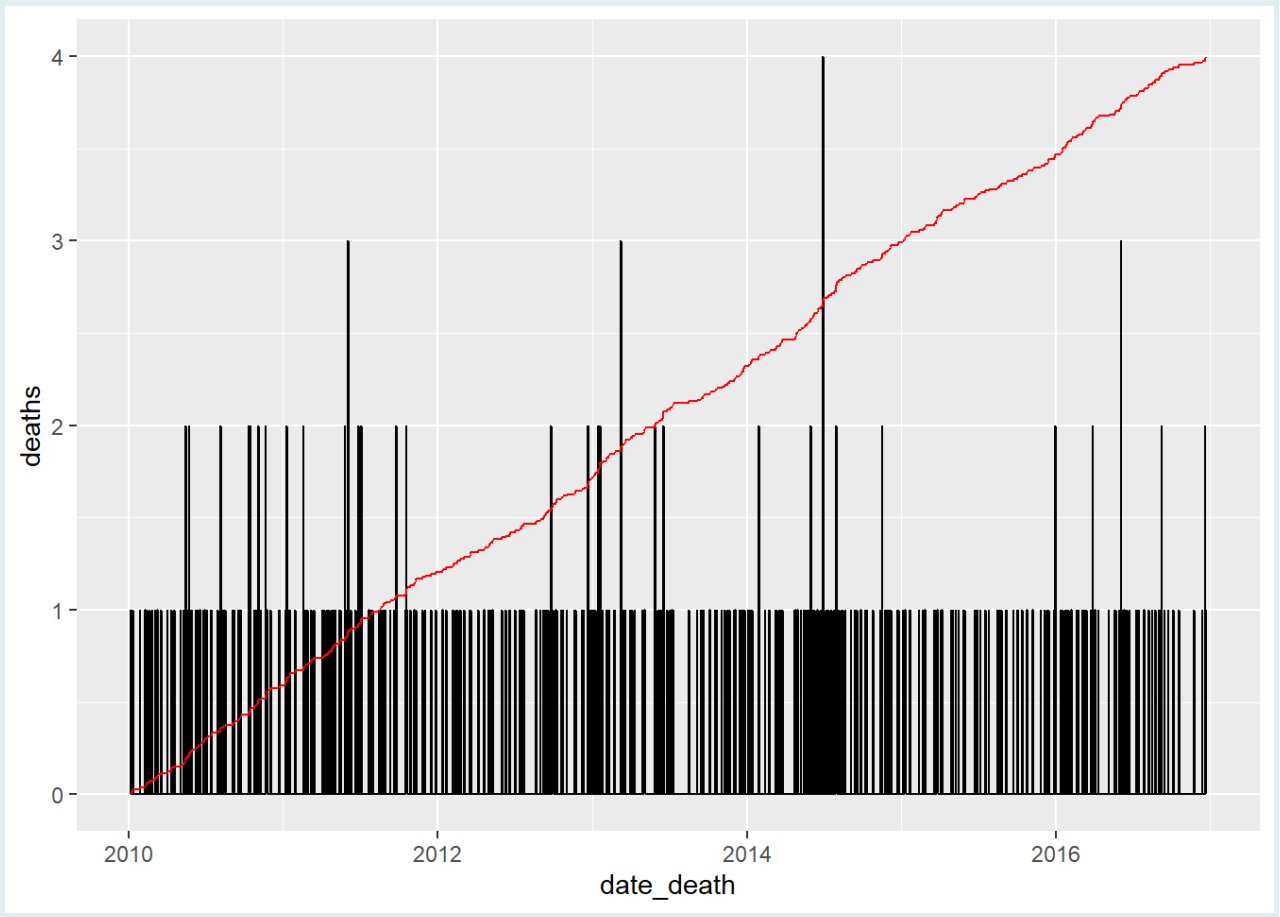


Q: Secondary axes

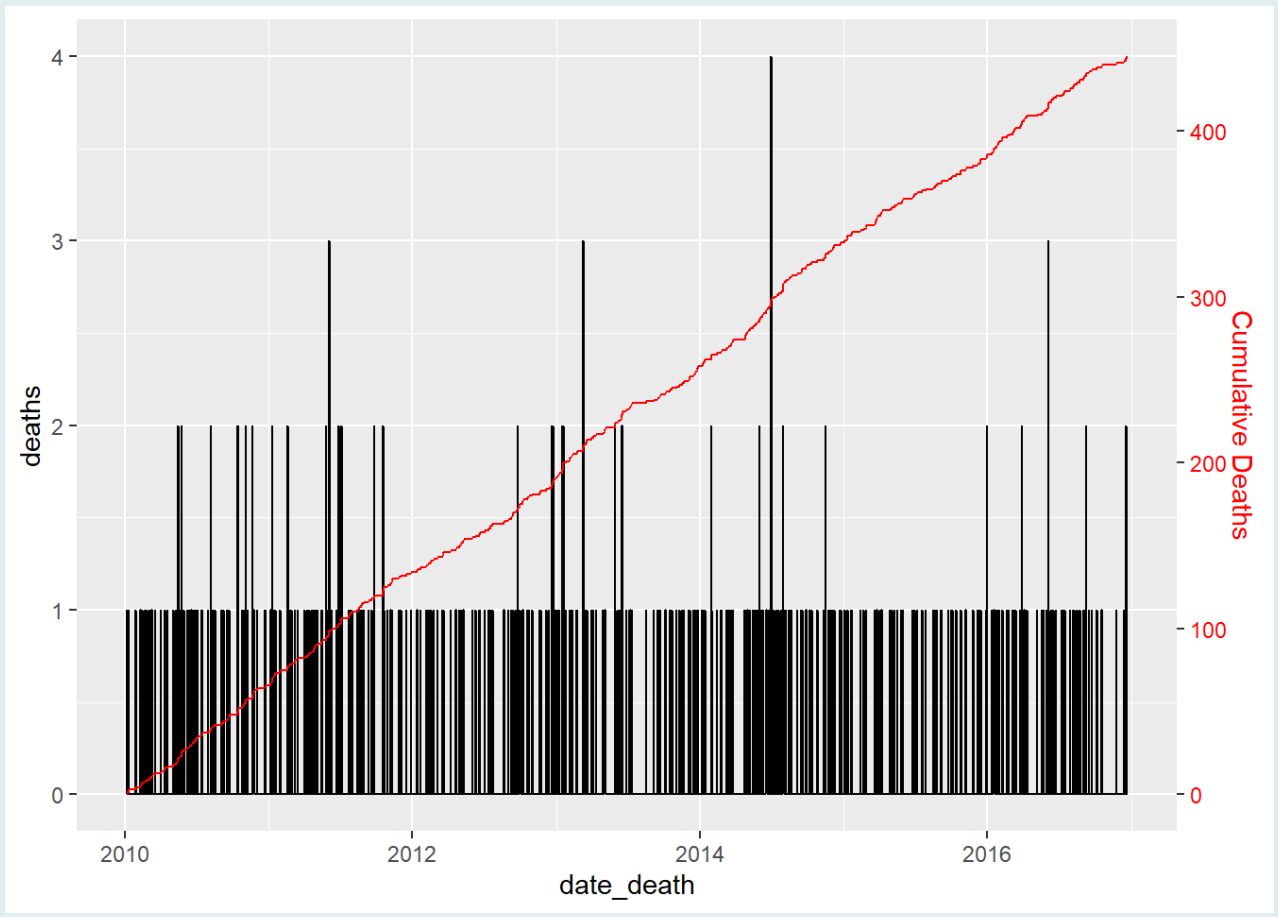
```r
# Step 1: Calculate cumulative deaths
colom_hiv_deaths_cumul <- colom_hiv_deaths_per_day %>%
  mutate(cum_deaths = cumsum(deaths))

# Step 2: Plot daily deaths
ggplot(colom_hiv_deaths_cumul, aes(x = date_death)) +
  geom_line(aes(y = deaths))
```




```r
# Step 3: Calculate scale factor
scale_factor <- max(colom_hiv_deaths_cumul$cum_deaths) /
        max(colom_hiv_deaths_cumul$deaths)

# Step 4: Add cumulative deaths to the plot
ggplot(colom_hiv_deaths_cumul, aes(x = date_death)) +
  geom_line(aes(y = deaths)) +
  geom_line(aes(y = cum_deaths / scale_factor), color = "red")
```




```
# Step 5: Add secondary y-axis
ggplot(colom_hiv_deaths_cumul, aes(x = date_death)) +
  geom_line(aes(y = deaths)) +
  geom_line(aes(y = cum_deaths / scale_factor), color = "red") +
  scale_y_continuous(sec.axis = sec_axis(trans = ~ .x * scale_factor, name =
        "Cumulative Deaths")) +
    theme(axis.text.y.right = element_text(color = "red"),
        axis.title.y.right = element_text(color = "red"))
```



Contributors

The following team members contributed to this lesson:

IMAD EL BADISY

Data Science Education Officer
Deeply interested in health data

JOY VAZ

R Developer and Instructor, the GRAPH Network
Loves doing science and teaching science

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement