Introduction to R for Disease Surveillance and Outbreak Forecasting: Day 3

Designing data tables

Michael Wimberly, Dawn Nekorchuk and Andrea Hess,
Department of Geography and Environmental Sustainability, University of Oklahoma

October 24 2018, Bahir Dar, Ethiopia

Contents

1 Introduction

Today you will get an introduction to creating and manipulating data. We begin to more fully explore two more of packages of the tidyverse: tibble, dplyr. The tibble package makes creating and viewing data frames easier, while the dplyr packages provides powerful functions for manipulating data frames. You will master those concepts to the point where you can work effectively with almost any dataset. You will learn some useful new tools for working with single tables, including summarizing data by groups, organizing data so that it is *tidy*, and dealing with both explicit and implicit missing values. Next, you will learn to combine datasets using functions that work no two or more tables.

2 Data transformations

You now have a solid background in plotting data with ggplot. But what happens when your data is not already in the correct format for plotting? Or what if you want to plot only a subset of your data?

In this part of the demonstration, you will learn the basic commands for manipulating data frames (and tibbles). Such manipulations can be accomplished in a variety of ways, but the most intuitive and efficient way is by using the functions provided in the **dplyr** package.

dplyr is a member of the tidyverse which focuses exclusively on manipulating data frames.

Start by loading dplyr.

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(readr)
```

The dplyr package has some excellent vignettes, including an Introduction to dplyr which can be accessed by running `vignette("dplyr")`

2.1 Single table verbs

Each data transformation has a corresponding dplyr function that accomplishes it. These functions take the form of verbs that describe the action. Some dplyr verbs operate on a single data frame or tibble, while others operate on two or more tables. Here is a list of the main dplyr verbs for single tables:

- `filter()` selects observations (rows) based on their values.
- `arrange()` reorders observations.
- `select()` and `rename()` select variables (columns) based on their names.
- `mutate()` and `transmute()` add new variables that are functions of existing variables.

2.1.1 Filter

```r
mecha <- read_csv("data_mecha.csv")
```

```
## Parsed with column specification:
## cols(
##   WID = col_double(),
##   woreda_name = col_character(),
##   obs_date = col_date(format = ""),
##   test_pf_tot = col_double(),
##   test_pv_only = col_double(),
##   pop_at_risk = col_double(),
##   mal_case = col_double(),
##   iso_year = col_double(),
##   iso_week = col_double(),
##   data_source = col_character()
## )
```

```r
mecha
```

```
## # A tibble: 156 x 10
##      WID woreda_name obs_date   test_pf_tot test_pv_only pop_at_risk
##    <dbl> <chr>       <date>           <dbl>        <dbl>       <dbl>
##  1   106 Mecha       2016-01-10           6           15     377232.
##  2   106 Mecha       2016-01-17           8           12     377232.
##  3   106 Mecha       2016-01-24           9           11     377232.
##  4   106 Mecha       2016-01-31           9           11     377232.
##  5   106 Mecha       2016-02-07          11           11     377232.
##  6   106 Mecha       2016-02-14          11           10     377232.
##  7   106 Mecha       2016-02-21          10           10     377232.
##  8   106 Mecha       2016-02-28           9            8     377232.
##  9   106 Mecha       2016-03-06           9            6     377232.
## 10   106 Mecha       2016-03-13           8            6     377232.
## # ... with 146 more rows, and 4 more variables: mal_case <dbl>,
## #   iso_year <dbl>, iso_week <dbl>, data_source <chr>
```

```r
filter(mecha, iso_week == 1)
```

```
## # A tibble: 3 x 10
##     WID woreda_name obs_date   test_pf_tot test_pv_only pop_at_risk
##   <dbl> <chr>       <date>           <dbl>        <dbl>       <dbl>
## 1   106 Mecha       2016-01-10           6           15     377232.
## 2   106 Mecha       2017-01-08           2            8     384022.
## 3   106 Mecha       2018-01-07           5            4     390935.
## # ... with 4 more variables: mal_case <dbl>, iso_year <dbl>,
## #   iso_week <dbl>, data_source <chr>
```

```r
filter(mecha, iso_year %in% 2015:2016)
```

```
## # A tibble: 52 x 10
##      WID woreda_name obs_date   test_pf_tot test_pv_only pop_at_risk
##    <dbl> <chr>       <date>           <dbl>        <dbl>       <dbl>
##  1   106 Mecha       2016-01-10           6           15     377232.
##  2   106 Mecha       2016-01-17           8           12     377232.
##  3   106 Mecha       2016-01-24           9           11     377232.
##  4   106 Mecha       2016-01-31           9           11     377232.
##  5   106 Mecha       2016-02-07          11           11     377232.
##  6   106 Mecha       2016-02-14          11           10     377232.
##  7   106 Mecha       2016-02-21          10           10     377232.
##  8   106 Mecha       2016-02-28           9            8     377232.
##  9   106 Mecha       2016-03-06           9            6     377232.
## 10   106 Mecha       2016-03-13           8            6     377232.
## # ... with 42 more rows, and 4 more variables: mal_case <dbl>,
## #   iso_year <dbl>, iso_week <dbl>, data_source <chr>
```

```r
filter(mecha, mal_case > 200, iso_year != 2016)
```

```
## # A tibble: 0 x 10
## # ... with 10 variables: WID <dbl>, woreda_name <chr>, obs_date <date>,
## #   test_pf_tot <dbl>, test_pv_only <dbl>, pop_at_risk <dbl>,
## #   mal_case <dbl>, iso_year <dbl>, iso_week <dbl>, data_source <chr>
```

2.1.2 Arrange

```
arrange(mecha, iso_week)
```

```
## # A tibble: 156 x 10
##      WID woreda_name obs_date   test_pf_tot test_pv_only pop_at_risk
##    <dbl> <chr>       <date>           <dbl>        <dbl>       <dbl>
##  1   106 Mecha       2016-01-10           6           15     377232.
##  2   106 Mecha       2017-01-08           2            8     384022.
##  3   106 Mecha       2018-01-07           5            4     390935.
##  4   106 Mecha       2016-01-17           8           12     377232.
##  5   106 Mecha       2017-01-15           3            7     384022.
##  6   106 Mecha       2018-01-14           3            4     390935.
##  7   106 Mecha       2016-01-24           9           11     377232.
##  8   106 Mecha       2017-01-22           2            7     384022.
##  9   106 Mecha       2018-01-21           3            5     390935.
## 10   106 Mecha       2016-01-31           9           11     377232.
## # ... with 146 more rows, and 4 more variables: mal_case <dbl>,
## #   iso_year <dbl>, iso_week <dbl>, data_source <chr>
```

Use desc() to order a column in descending rather than ascending order.

```
arrange(mecha, desc(mal_case))
```

```
## # A tibble: 156 x 10
##      WID woreda_name obs_date   test_pf_tot test_pv_only pop_at_risk
##    <dbl> <chr>       <date>           <dbl>        <dbl>       <dbl>
##  1   106 Mecha       2016-05-22          36           20     377232.
##  2   106 Mecha       2016-05-29          35           21     377232.
##  3   106 Mecha       2016-05-15          35           20     377232.
##  4   106 Mecha       2016-06-05          33           22     377232.
##  5   106 Mecha       2016-06-12          29           22     377232.
##  6   106 Mecha       2016-11-13          34           15     384022.
##  7   106 Mecha       2016-05-08          31           17     377232.
##  8   106 Mecha       2016-11-06          33           15     384022.
##  9   106 Mecha       2016-05-01          28           18     377232.
## 10   106 Mecha       2016-10-30          31           14     384022.
## # ... with 146 more rows, and 4 more variables: mal_case <dbl>,
## #   iso_year <dbl>, iso_week <dbl>, data_source <chr>
```

2.1.3 Select and Rename

Select columns to keep by name. Other columns are removed from the data frame.

```
select(mecha, woreda_name, iso_year, iso_week, mal_case)
```

```
## # A tibble: 156 x 4
##    woreda_name iso_year iso_week mal_case
##    <chr>          <dbl>    <dbl>    <dbl>
##  1 Mecha           2016        1       21
##  2 Mecha           2016        2       20
##  3 Mecha           2016        3       20
##  4 Mecha           2016        4       20
##  5 Mecha           2016        5       22
##  6 Mecha           2016        6       21
##  7 Mecha           2016        7       20
##  8 Mecha           2016        8       17
##  9 Mecha           2016        9       15
## 10 Mecha           2016       10       14
## # ... with 146 more rows
```

Use the `:` operator to select a continuous series of columns. The helper functions `starts_with()`, `ends_with()`, and `contains()` can be used to find multiple columns by matching part of the column name.

```
select(mecha,
       woreda_name:mal_case)
```

```
## # A tibble: 156 x 6
##    woreda_name obs_date   test_pf_tot test_pv_only pop_at_risk mal_case
##    <chr>       <date>           <dbl>        <dbl>       <dbl>    <dbl>
##  1 Mecha       2016-01-10           6           15     377232.       21
##  2 Mecha       2016-01-17           8           12     377232.       20
##  3 Mecha       2016-01-24           9           11     377232.       20
##  4 Mecha       2016-01-31           9           11     377232.       20
##  5 Mecha       2016-02-07          11           11     377232.       22
##  6 Mecha       2016-02-14          11           10     377232.       21
##  7 Mecha       2016-02-21          10           10     377232.       20
##  8 Mecha       2016-02-28           9            8     377232.       17
##  9 Mecha       2016-03-06           9            6     377232.       15
## 10 Mecha       2016-03-13           8            6     377232.       14
## # ... with 146 more rows
```

```
select(mecha,
       starts_with("test"))
```

```
## # A tibble: 156 x 2
##    test_pf_tot test_pv_only
##          <dbl>        <dbl>
##  1           6           15
##  2           8           12
##  3           9           11
##  4           9           11
##  5          11           11
##  6          11           10
##  7          10           10
##  8           9            8
##  9           9            6
## 10           8            6
## # ... with 146 more rows
```

Remove columns by prefixing their names with a `-`. Other columns will be kept.

```
select(mecha,
       -test_pf_tot,
       -test_pv_only)
```

```
## # A tibble: 156 x 8
##      WID woreda_name obs_date   pop_at_risk mal_case iso_year iso_week
##    <dbl> <chr>       <date>           <dbl>    <dbl>    <dbl>    <dbl>
##  1   106 Mecha       2016-01-10     377232.       21     2016        1
##  2   106 Mecha       2016-01-17     377232.       20     2016        2
##  3   106 Mecha       2016-01-24     377232.       20     2016        3
##  4   106 Mecha       2016-01-31     377232.       20     2016         4
##  5   106 Mecha       2016-02-07     377232.       22     2016         5
##  6   106 Mecha       2016-02-14     377232.       21     2016         6
##  7   106 Mecha       2016-02-21     377232.       20     2016         7
##  8   106 Mecha       2016-02-28     377232.       17     2016         8
##  9   106 Mecha       2016-03-06     377232.       15     2016         9
## 10   106 Mecha       2016-03-13     377232.       14     2016        10
## # ... with 146 more rows, and 1 more variable: data_source <chr>
```

Rename columns using the = operator, placing the new name first and the old name second.

```
rename(mecha, district = woreda_name,
       malaria_case = mal_case)
```

```
## # A tibble: 156 x 10
##      WID district obs_date   test_pf_tot test_pv_only pop_at_risk
##    <dbl> <chr>    <date>           <dbl>        <dbl>       <dbl>
##  1   106 Mecha    2016-01-10           6           15     377232.
##  2   106 Mecha    2016-01-17           8           12     377232.
##  3   106 Mecha    2016-01-24           9           11     377232.
##  4   106 Mecha    2016-01-31           9           11     377232.
##  5   106 Mecha    2016-02-07          11           11     377232.
##  6   106 Mecha    2016-02-14          11           10     377232.
##  7   106 Mecha    2016-02-21          10           10     377232.
##  8   106 Mecha    2016-02-28           9            8     377232.
##  9   106 Mecha    2016-03-06           9            6     377232.
## 10   106 Mecha    2016-03-13           8            6     377232.
## # ... with 146 more rows, and 4 more variables: malaria_case <dbl>,
## #   iso_year <dbl>, iso_week <dbl>, data_source <chr>
```

2.1.4 Mutate and Transmute

Add new variables using `mutate()`. You can create multiple new variables at a time, and you can refer to ones you've just created.

```
mutate(mecha,
       inc = mal_case / pop_at_risk,
       inc_per_1000 = inc * 1000)
```

```
## # A tibble: 156 x 12
##      WID woreda_name obs_date   test_pf_tot test_pv_only pop_at_risk
##    <dbl> <chr>       <date>           <dbl>        <dbl>       <dbl>
##  1   106 Mecha       2016-01-10           6           15     377232.
##  2   106 Mecha       2016-01-17           8           12     377232.
##  3   106 Mecha       2016-01-24           9           11     377232.
##  4   106 Mecha       2016-01-31           9           11     377232.
##  5   106 Mecha       2016-02-07          11           11     377232.
##  6   106 Mecha       2016-02-14          11           10     377232.
##  7   106 Mecha       2016-02-21          10           10     377232.
##  8   106 Mecha       2016-02-28           9            8     377232.
##  9   106 Mecha       2016-03-06           9            6     377232.
## 10   106 Mecha       2016-03-13           8            6     377232.
## # ... with 146 more rows, and 6 more variables: mal_case <dbl>,
## #   iso_year <dbl>, iso_week <dbl>, data_source <chr>, inc <dbl>,
## #   inc_per_1000 <dbl>
```

If you only want to keep the new variables, use `transmute()`.

```
transmute(mecha,
       inc = mal_case / pop_at_risk,
       inc_per_1000 = inc * 1000)
```

```
## # A tibble: 156 x 2
##          inc inc_per_1000
##        <dbl>        <dbl>
##  1 0.0000557       0.0557
##  2 0.0000530       0.0530
##  3 0.0000530       0.0530
##  4 0.0000530       0.0530
##  5 0.0000583       0.0583
##  6 0.0000557       0.0557
##  7 0.0000530       0.0530
##  8 0.0000451       0.0451
##  9 0.0000398       0.0398
## 10 0.0000371       0.0371
## # ... with 146 more rows
```

For the next few exercises, you will use the `data_day3` dataset. Let's begin by loading the dataset again using `read_csv()`.

```
library(readr)
data <- read_csv("data_day3_subset.csv")
```

3 Grouped summaries with `summarize()`

The only main dplyr verb we did not cover yesterday was `summarize()`, which collapses a data frame into a single row:

```
library(dplyr)
```

```
summarize(data, n_cases = sum(mal_case, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   n_cases
##     <dbl>
## 1  268421
```

Don't worry about the `na.rm` argument for now. We will cover that below.

`summarize()` becomes particularly useful when we pair it with another dplyr function, `group_by()`, which groups rows of data together to produce a "grouped data frame". When a grouped data frame is summarized, the summaries are for individual groups rather than the entire dataset, and the result is a data frame with one row per group.

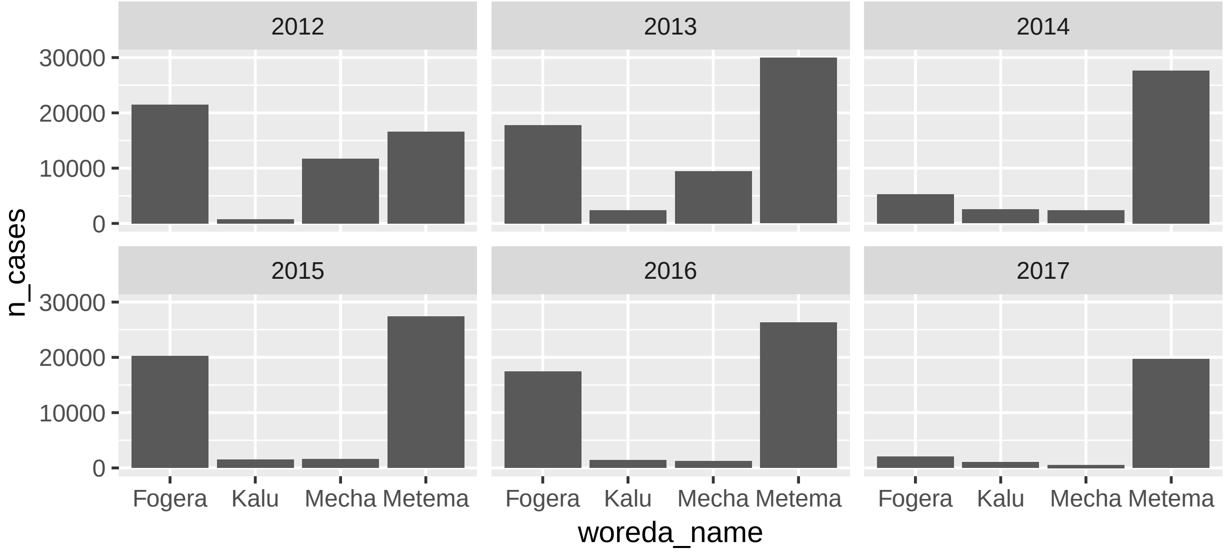
```
by_woreda_year <- group_by(data, woreda_name, iso_year)
summarized_cases <- summarize(by_woreda_year, n_cases = sum(mal_case, na.rm = TRUE))
summarized_cases
```

```
## # A tibble: 24 x 3
## # Groups:   woreda_name [?]
##    woreda_name iso_year n_cases
##    <chr>          <dbl>   <dbl>
##  1 Fogera          2012   21521
##  2 Fogera          2013   17781
##  3 Fogera          2014    5298
##   4 Fogera          2015   20231
##   5 Fogera          2016   17415
##   6 Fogera          2017    2011
##   7 Kalu            2012     739
##   8 Kalu            2013    2410
##   9 Kalu            2014    2527
##  10 Kalu            2015    1473
## # ... with 14 more rows
```

In this example, we calculated the total number of cases observed in each woreda in each year. In this case, we grouped by two variables, but you can group any number of variables you want.

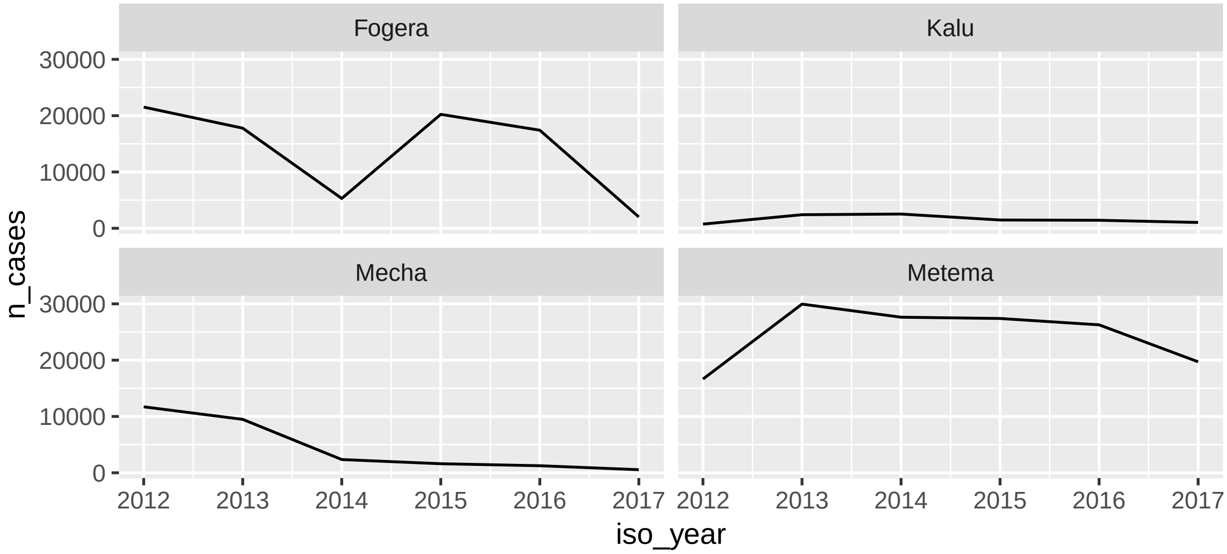
We can also visualize the summaries with a column plot:

```
library(ggplot2)
ggplot(summarized_cases) +
  geom_col(aes(x = woreda_name, y = n_cases)) +
  facet_wrap(~ iso_year)
```



Or with as a time series plot:

```
ggplot(summarized_cases) +
  geom_line(aes(x = iso_year, y = n_cases)) +
  facet_wrap(~ woreda_name)
```



Grouped summaries are a powerful tool for data exploration or for transforming data into group summaries for futher analysis or visualization.

3.1 Missing data

In the previous section, you saw the argument `na.rm = TRUE`. That argument told R to remove the missing values, identified by `NA`s, from the data before summarizing.

If you set `na.rm = FALSE`, which is the default value, then `NA`s are not removed. Any vector that includes an `NA` value will yield an `NA` value when summarized. The logic is that if you are missing some elements in a set of values, you can't accurately summarize the set. Thus we see the following behaviors:

```
sum(c(1, 2, 3))
```

```
## [1] 6
```

```
sum(c(1, 2, 3, NA))
```

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

```
sum(c(1, 2, 3, NA), na.rm = TRUE)
```

```
## [1] 6
```

3.2 Counts

When summarizing, it is often useful to count the number of values used. This can be done with the dplyr function `n()`, which will include `NA` values, or `sum(!is.na(x))` which will exclude them. It's often a good idea to count when summarizing to make sure your summaries are not based on a very small sample size.

```
summarize(by_woreda_year,
          n_cases = sum(test_pv_only, na.rm = TRUE),
          n_weeks = n())
```

```
## # A tibble: 24 x 4
## # Groups:   woreda_name [?]
##    woreda_name iso_year n_cases n_weeks
##    <chr>          <dbl>   <dbl>   <int>
##  1 Fogera          2012    3300      20
##  2 Fogera          2013    3214      36
##  3 Fogera          2014    1409      33
##  4 Fogera          2015    6156      41
##  5 Fogera          2016    8634      35
##  6 Fogera          2017     803      39
##  7 Kalu            2012     393      13
##  8 Kalu            2013    1282      36
##  9 Kalu            2014    1733      36
## 10 Kalu            2015    1084      37
## # ... with 14 more rows
```

If the only summary you want to perform is counting, you can also use the dplyr function `count()`. This is basically a shortcut to perform a grouped summarize.

```
count(by_woreda_year, woreda_name, iso_year)
```

```
## # A tibble: 24 x 3
## # Groups:   woreda_name, iso_year [24]
##    woreda_name iso_year     n
##    <chr>          <dbl> <int>
##  1 Fogera          2012    20
##  2 Fogera          2013    36
##  3 Fogera          2014    33
##  4 Fogera          2015    41
##  5 Fogera          2016    35
##  6 Fogera          2017    39
##  7 Kalu            2012    13
##  8 Kalu            2013    36
##  9 Kalu            2014    36
## 10 Kalu            2015    37
## # ... with 14 more rows
```

3.3 Useful summary functions

Up to this point, you've only seen two summary functions: `sum()` and `n()`. Other useful summary functions include:

- Measures of location: `mean(x)` and `median(x)`. The mean is the sum divided by the length; the median is a value where 50% of `x` is above it, and 50% is below it.
- Measures of spread: `sd(x)`, `IQR(x)`, `mad(x)`. The mean squared deviation, or standard deviation or sd for short, is the standard measure of spread. The interquartile range `IQR()` and median absolute deviation `mad(x)` are robust equivalents that may be more useful if you have outliers.
- Measures of rank: `min(x)`, `quantile(x, 0.25)`, `max(x)`. Quantiles are a generalisation of the median. For example, `quantile(x, 0.25)` will find a value of `x` that is greater than 25% of the values, and less than the remaining 75%.
- Measures of position: `first(x)`, `nth(x, 2)`, `last(x)`. These work similarly to x[1], x[2], and x[length(x)] but let you set a default value if that position does not exist (i.e. you're trying to get the 3rd element from a group that only has two elements).
- Counts: You've seen `n()`, which takes no arguments, and returns the size of the current group. To count the number of non-missing values, use `sum(!is.na(x))`. To count the number of distinct (unique) values, use `n_distinct(x)`.

4 Working with epidemiological weeks (WHO ISO 8601)

Up until this point we have been working with epidemiological datasets organized by epidemiological week number (or iso week) and epidemiological year (or iso year). We will use a trimmed down version of our full (without missing rows) dataset earlier in the day as an example.

```
day3 <- read_csv("data_day3.csv")
```

```
## Parsed with column specification:
## cols(
##   WID = col_double(),
##   woreda_name = col_character(),
##   obs_date = col_date(format = ""),
##   test_pf_tot = col_double(),
##   test_pv_only = col_double(),
##   pop_at_risk = col_double(),
##   mal_case = col_double(),
##   iso_year = col_double(),
##   iso_week = col_double(),
##   pfm_inc = col_double(),
##   pv_inc = col_double(),
##   data_source = col_character()
## )
```

```
incid <- select(day3, woreda_name, iso_year, iso_week, pfm_inc, pv_inc)
incid
```

```
## # A tibble: 1,144 x 5
##    woreda_name iso_year iso_week pfm_inc pv_inc
##    <chr>          <dbl>    <dbl>   <dbl>  <dbl>
##  1 Fogera          2012       28    3.32  0.563
##  2 Fogera          2012       29    3.13  0.582
##  3 Fogera          2012       30    2.63  0.511
##  4 Fogera          2012       31    2.35  0.506
##  5 Fogera          2012       32    2.05  0.506
##  6 Fogera          2012       33    1.93  0.511
##  7 Fogera          2012       34    1.83  0.530
##  8 Fogera          2012       35    1.88  0.558
##  9 Fogera          2012       36    1.96  0.587
## 10 Fogera          2012       37    2.24  0.644
## # ... with 1,134 more rows
```

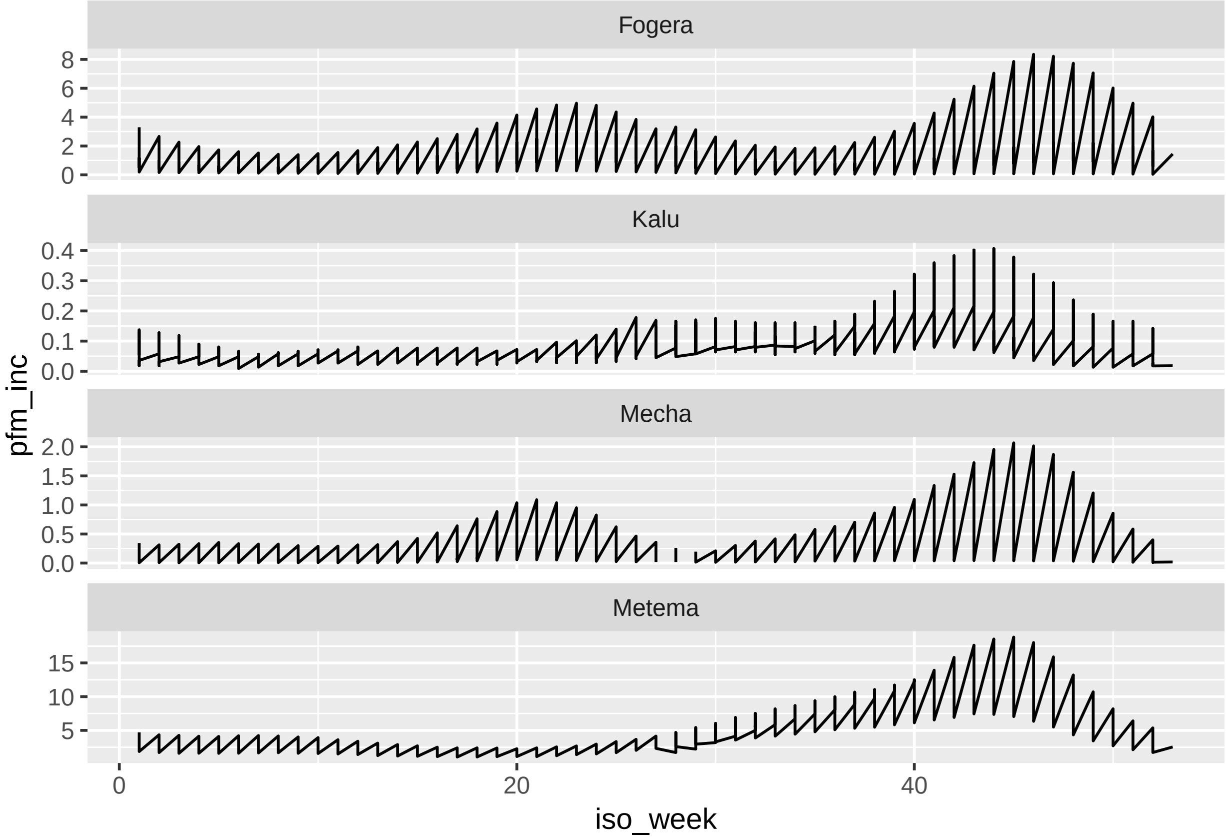
```
count(incid, woreda_name)
```

```
## # A tibble: 4 x 2
##   woreda_name     n
##   <chr>       <int>
## 1 Fogera        286
## 2 Kalu          286
## 3 Mecha         286
## 4 Metema        286
```

We can see that we have n weeks of data for each woreda. If we wanted to plot this data as a single time series of n weeks, we might do something like this:

```
ggplot(incid) +
  geom_line(aes(x = iso_week, y = pfm_inc)) +
```

```r
  facet_wrap(~ woreda_name, ncol = 1, scales = "free_y")
```



The problem is that ggplot does not know that we have data from different years, so for each week it plots all years on top of each other and tries to connect them with a line. Unfortunately, the aesthetic mappings in ggplot plot must be 1 to 1, meaning you can't map two columns in the data frame (iso_year and iso_week) as the x axis values.

In this situation we need to add a new column to our dataset that contains properly ordered values to use as the x-axis variable. But what values do we use? We could simple number the rows as 1 to n for each woreda, but then we lose the date-related information on the x axis when we plot it. The solution is to convert from epi weeks to dates, i.e. the Date class you learned about on day 1. Dates contain all the necessary information to be ordered correctly, and they fit in a single column.

Although the lubridate package in R contains many useful functions for working with dates in general, it does not yet include a function to convert from iso weeks and iso years to calendar dates. For that we will need to use a custom function, which we will load (or "source") from another R file.

```r
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##     date
```

```r
source("date_functions.R")
```

If you look at the Environment tab in RStudio you will see that it now includes a function named make_date_yw(). This function takes vectors of iso years, week numbers (1–53), and weekdays (1–7) and returns a vector of Dates. For example:

```
make_date_yw(year = 2017, week = 1, weekday = 1)    # first day of 2017 week 1
```

```
## [1] "2017-01-02"
```

```
make_date_yw(2017, 1, 2)                             # second day of that week
```

```
## [1] "2017-01-03"
```

```
make_date_yw(2017, 1)                                # weekday defaults to 1
```

```
## [1] "2017-01-02"
```

```
make_date_yw(2015:2017, 1:3)                         # arguments can be vectors
```

```
## [1] "2014-12-29" "2016-01-11" "2017-01-16"
```

```
make_date_yw(2017, 1:3)                              # arguments are "recycled"
```

```
## [1] "2017-01-02" "2017-01-09" "2017-01-16"
```

Other very useful functions exist in the lubridate package.

```
isoweek(Sys.Date())   # today's ISO week number
```

```
## [1] 16
```

```
isoyear(Sys.Date())   # today's ISO year
```

```
## [1] 2019
```

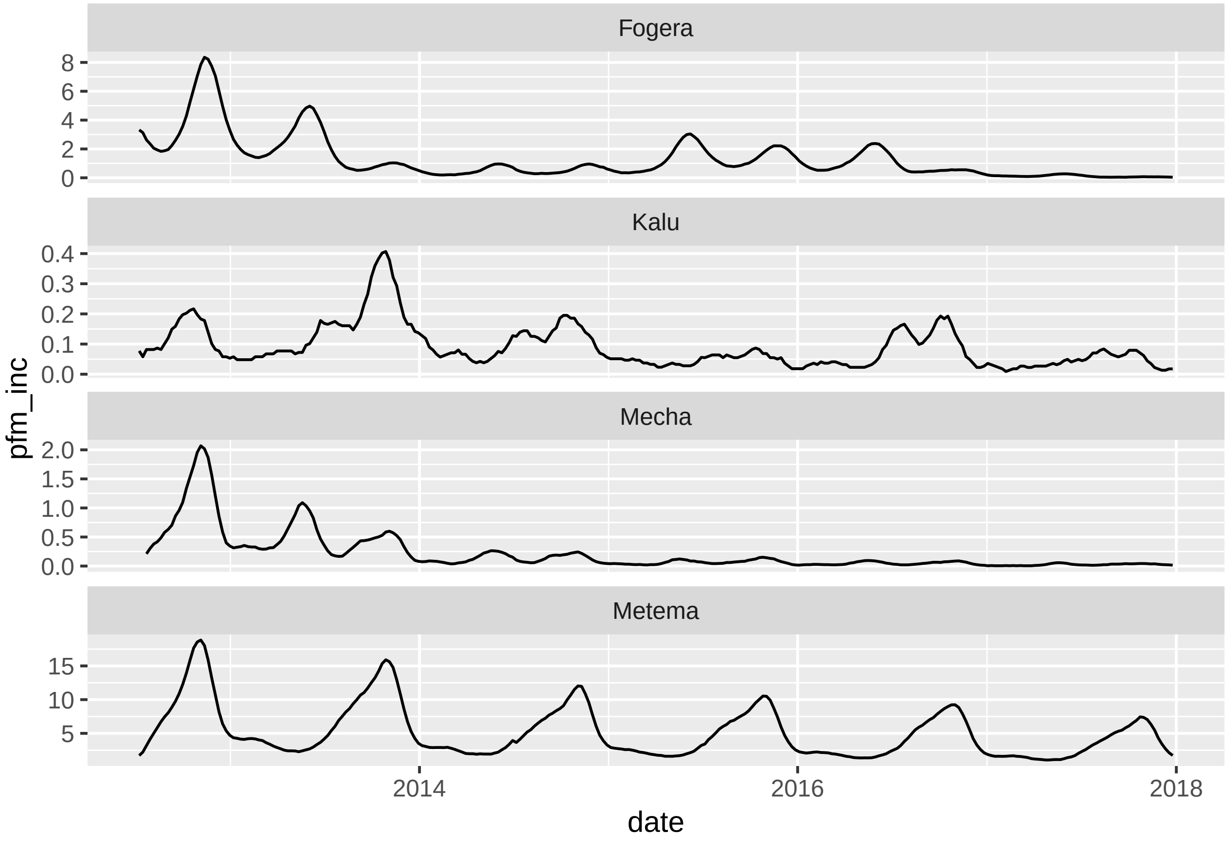
Because make_date_yw() is accepts vectors as arguments, we can use it with mutate().

```
incid_date <- mutate(incid, date = make_date_yw(iso_year, iso_week))
incid_date <- select(incid_date, woreda_name, iso_year, iso_week,
                     date, pfm_inc, pv_inc) # put the new column after date
incid_date
```

```
## # A tibble: 1,144 x 6
##    woreda_name iso_year iso_week date       pfm_inc pv_inc
##    <chr>          <dbl>    <dbl> <date>       <dbl>  <dbl>
##  1 Fogera          2012       28 2012-07-09    3.32  0.563
##  2 Fogera          2012       29 2012-07-16    3.13  0.582
##  3 Fogera          2012       30 2012-07-23    2.63  0.511
##  4 Fogera          2012       31 2012-07-30    2.35  0.506
##  5 Fogera          2012       32 2012-08-06    2.05  0.506
##  6 Fogera          2012       33 2012-08-13    1.93  0.511
##  7 Fogera          2012       34 2012-08-20    1.83  0.530
##  8 Fogera          2012       35 2012-08-27    1.88  0.558
##  9 Fogera          2012       36 2012-09-03    1.96  0.587
## 10 Fogera          2012       37 2012-09-10    2.24  0.644
## # ... with 1,134 more rows
```

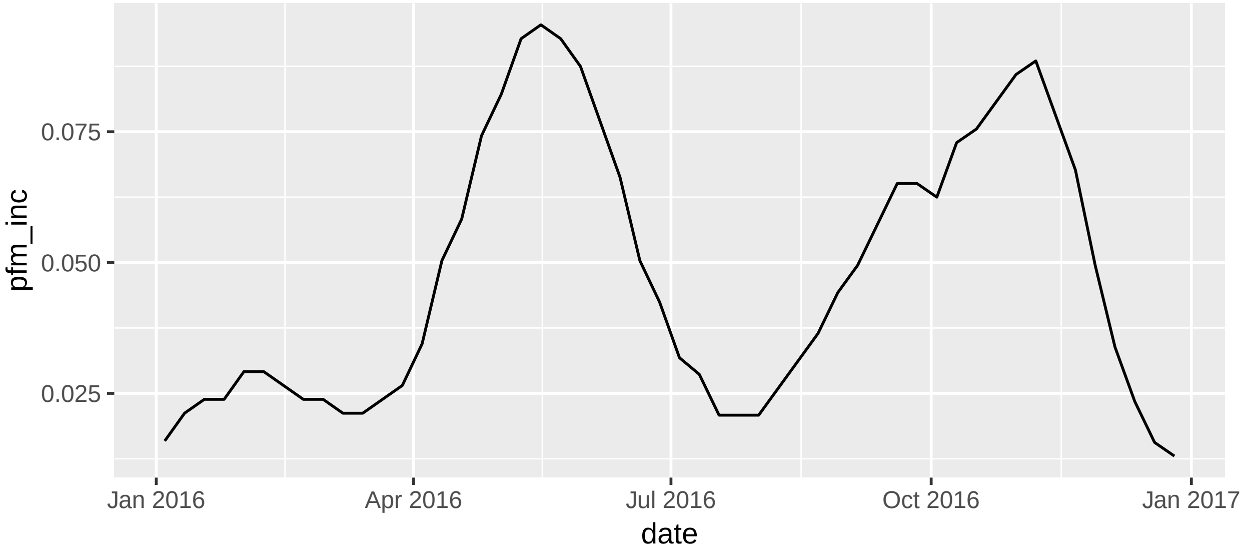
Now when we plot, we can map the x axis value to the date column date:

```
ggplot(incid_date) +
  geom_line(aes(x = date, y = pfm_inc)) +
  facet_wrap(~ woreda_name, ncol = 1, scales = "free_y")
```

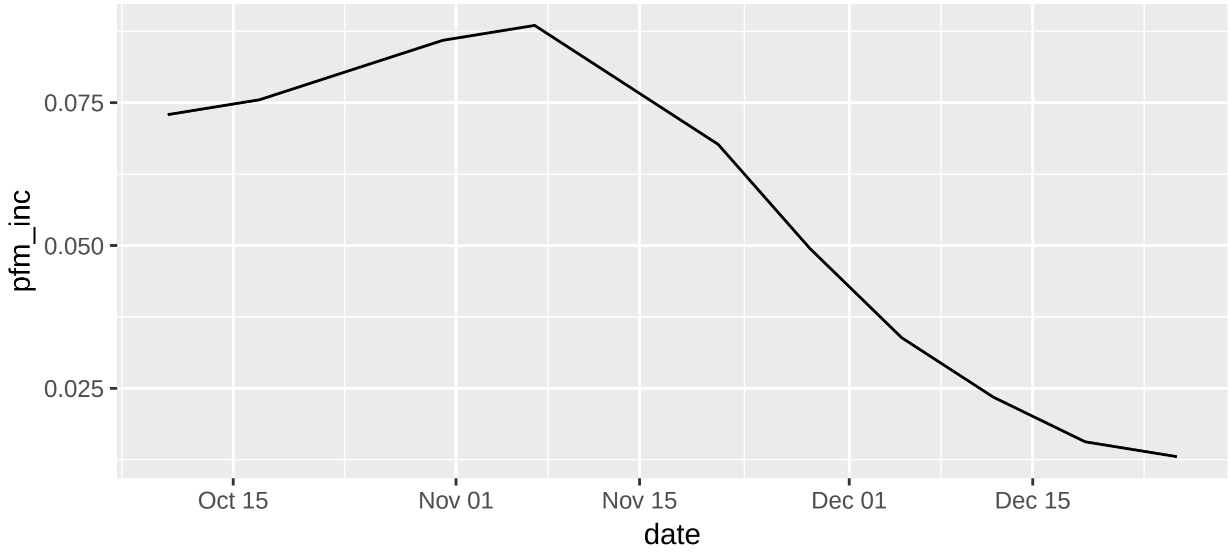


R automatically adjusts the x-axis labels based on the range of dates. In this case, the major axis lines and labels are on January 1 of each year and the minor axis lines are on June 1. Notice how the labels change when a smaller set of data is used:

```
# plot a single year of Mecha incidence
ggplot(filter(incid_date, woreda_name == "Mecha", iso_year == 2016)) +
  geom_line(aes(x = date, y = pfm_inc))
```

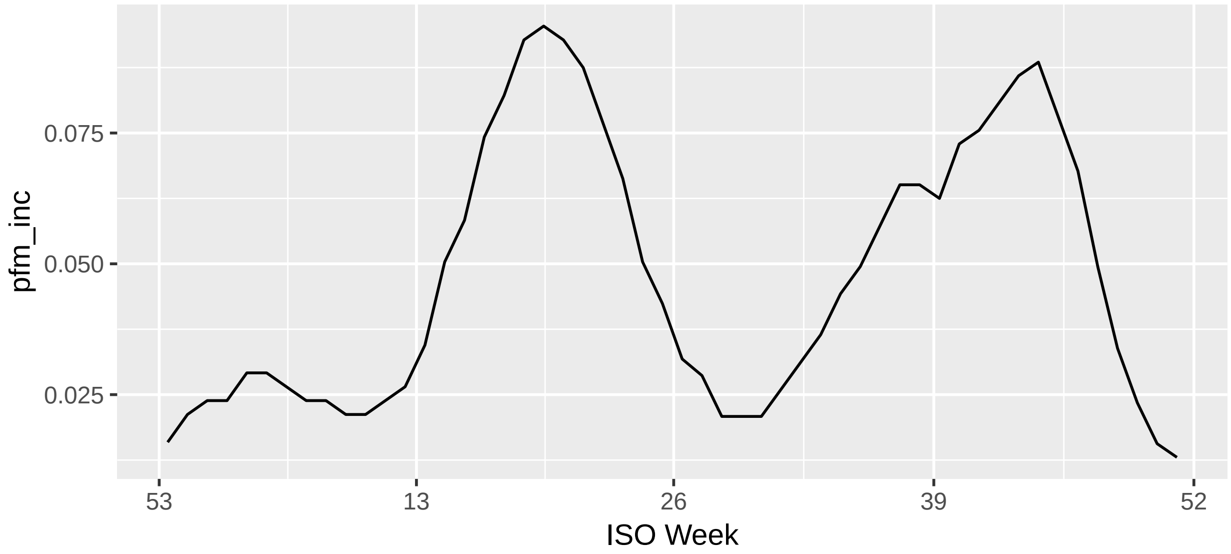


```
# plot a few months of Mecha incidence
ggplot(filter(incid_date, woreda_name == "Mecha", iso_year == 2016, iso_week > 40)) +
  geom_line(aes(x = date, y = pfm_inc))
```



For epidemiological data, of course, it might be more useful to label the weeks by iso week number rather than date. In this case, we can use a new function `scale_x_date()` to modify how the x axis labels appear. `scale_x_date()` takes an argument named labels, which can be defined in several ways (see the help page), but importantly for our purposes it can be given the name of a function which converts Dates to character or numeric values to use as labels. We have just such a function in `isoweek()`!

```
# plot a single year with iso week labels
ggplot(filter(incid_date, woreda_name == "Mecha", iso_year == 2016)) +
  geom_line(aes(x = date, y = pfm_inc)) +
  scale_x_date(labels = isoweek) +
  labs(x = "ISO Week") # dont forget to change the axis label to match
```



5 Tidying data with the tidyr package

The concept of **tidy** data has become increasing popular among R users over the past few years thanks to the tidyr package, and more recently the tidyverse family of packages. Tidy data refers to data tables in which each row is an **observation**, each column is a **variable**, and each cell is a **value**.

Tidy data has three main advantages: it is consistent and therefor easily replicated and easily taught, R's emphasis on vectors works well when variables are organized into columns, and an increasing number of R packages are designed to work on tidy data, e.g. those in the tidyverse, but also dozens of others and more every month.

Unfortunately data is not always stored in tidy format. For example, data is often stored in a format that facilitates data entry. The functions in the tidyr package help you tidy your data so that it may be used for further visualization and analysis.

5.1 Spreading and gathering

Two common problems with un-tidy datasets are when one variable is spread across multiple columns or one observation is spread across multiple rows. In these cases, you can use the two tidyr functions `gather()` and `spread()`.

As an example, let us look at our `incid_data` data again.

```
incid_date
```

```
## # A tibble: 1,144 x 6
##    woreda_name iso_year iso_week date       pfm_inc pv_inc
##    <chr>          <dbl>    <dbl> <date>       <dbl>  <dbl>
##  1 Fogera          2012       28 2012-07-09    3.32  0.563
##  2 Fogera          2012       29 2012-07-16    3.13  0.582
##  3 Fogera          2012       30 2012-07-23    2.63  0.511
##  4 Fogera          2012       31 2012-07-30    2.35  0.506
##  5 Fogera          2012       32 2012-08-06    2.05  0.506
##  6 Fogera          2012       33 2012-08-13    1.93  0.511
##  7 Fogera          2012       34 2012-08-20    1.83  0.530
##  8 Fogera          2012       35 2012-08-27    1.88  0.558
##  9 Fogera          2012       36 2012-09-03    1.96  0.587
## 10 Fogera          2012       37 2012-09-10    2.24  0.644
## # ... with 1,134 more rows
```

Both pfm_inc and pv_inc are values of incidence, the first for *P. falciparum* and mixed infection malaria, the second for *P. vivax* malaria. In that sense, *falciparum* and *vivax* are not variables, but rather values identifying the malaria-causing agent. Both columns contain incidence values, and each row represents two observations, not one.

To tidy a dataset like this we need to **gather** those columns into two new variables, one that holds the name of the malaria agent and the other that holds the incidence.

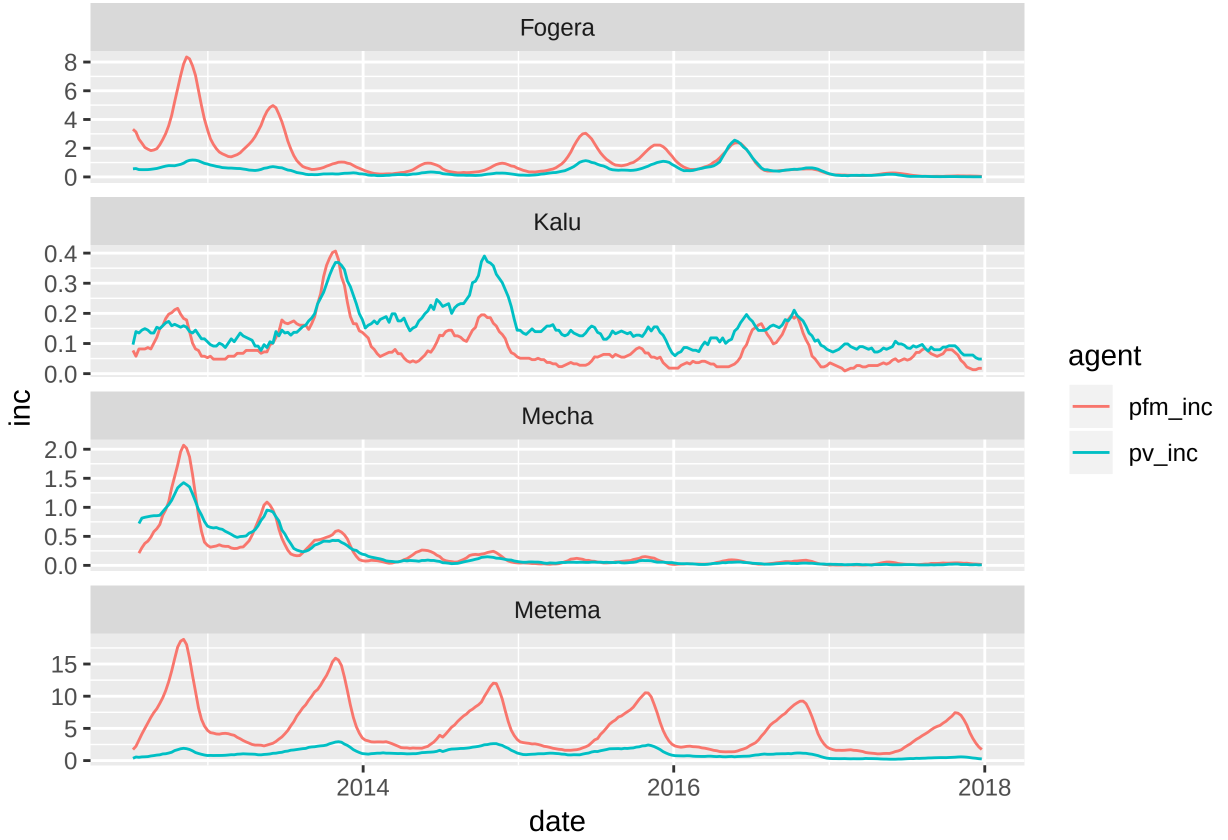
The names of the columns will become values in a new variable called the **key**, which we will call "agent". The values of the two columns will become a single new *value* variable, which we will call "inc".

```
library(tidyr)
incid_tidy <- gather(incid_date, key = agent, value = inc,
                     pfm_inc, pv_inc) # these are the columns to gather
incid_tidy
```

```
## # A tibble: 2,288 x 6
##    woreda_name iso_year iso_week date       agent     inc
##    <chr>          <dbl>    <dbl> <date>     <chr>   <dbl>
##  1 Fogera          2012       28 2012-07-09 pfm_inc  3.32
##  2 Fogera          2012       29 2012-07-16 pfm_inc  3.13
##  3 Fogera          2012       30 2012-07-23 pfm_inc  2.63
##  4 Fogera          2012       31 2012-07-30 pfm_inc  2.35
##  5 Fogera          2012       32 2012-08-06 pfm_inc  2.05
##  6 Fogera          2012       33 2012-08-13 pfm_inc  1.93
##  7 Fogera          2012       34 2012-08-20 pfm_inc  1.83
##  8 Fogera          2012       35 2012-08-27 pfm_inc  1.88
##  9 Fogera          2012       36 2012-09-03 pfm_inc  1.96
## 10 Fogera          2012       37 2012-09-10 pfm_inc  2.24
## # ... with 2,278 more rows
```

Now that we have the dataset in tidy format, we can plot it and "map" the new agent variable to the color aesthetic in the plot.

```
ggplot(incid_tidy) +
  geom_line(aes(date, inc, color = agent)) +
  facet_wrap(~ woreda_name, ncol = 1, scales = "free_y")
```



When observations are scattered across multiple rows, we need to **spread** the data into multiple columns. The `incid_tidy` dataset is already in tidy format, but we can use it as an example for how to use `spread()`:

```
library(tidyr)
spread(incid_tidy, key = agent, value = inc)
```

```
## # A tibble: 1,144 x 6
##    woreda_name iso_year iso_week date       pfm_inc pv_inc
##    <chr>          <dbl>    <dbl> <date>       <dbl>  <dbl>
##  1 Fogera          2012       28 2012-07-09    3.32  0.563
##  2 Fogera          2012       29 2012-07-16    3.13  0.582
##  3 Fogera          2012       30 2012-07-23    2.63  0.511
##  4 Fogera          2012       31 2012-07-30    2.35  0.506
##  5 Fogera          2012       32 2012-08-06    2.05  0.506
##  6 Fogera          2012       33 2012-08-13    1.93  0.511
##  7 Fogera          2012       34 2012-08-20    1.83  0.530
##  8 Fogera          2012       35 2012-08-27    1.88  0.558
##  9 Fogera          2012       36 2012-09-03    1.96  0.587
## 10 Fogera          2012       37 2012-09-10    2.24  0.644
## # ... with 1,134 more rows
```

5.2 Separating and uniting

Sometimes you may find that a single column contains multiple values. In that case, you need to **separate** the two values into multiple columns.

Image we have a dataset where the iso week is provided in year-week format, e.g. "2017-W01".

```
yw_example <- tibble(yw = c("2016-W01", "2016-W02", "2016-W03"),
                     mal_case = c(2565, 2042, 1803))
yw_example
```

```
## # A tibble: 3 x 2
##   yw       mal_case
##   <chr>       <dbl>
## 1 2016-W01     2565
## 2 2016-W02     2042
## 3 2016-W03     1803
```

In this case, 2017 is the year and 1 is the week number. You can use separate()' to put them into their own columns. Here,colis the column to separate,intois a character vector of the names of the new columns,sepis the character string separating the two values, andconvert = TRUE‘ causes the new columns to be converted from character to numeric format if possible.

```
yw_sep <- separate(yw_example, col = yw,
                   into = c("iso_year", "iso_week"),
                   sep = "-W", convert = TRUE)
yw_sep
```

```
## # A tibble: 3 x 3
##   iso_year iso_week mal_case
##      <int>    <int>    <dbl>
## 1     2016        1     2565
## 2     2016        2     2042
## 3     2016        3     1803
```

The complement of `separate()` is `unite()`, which can be used to **unite** multiple columns into one.

```
unite(yw_sep, col = "yw", iso_year, iso_week, sep = "-W")
```

```
## # A tibble: 3 x 2
##   yw      mal_case
##   <chr>      <dbl>
## 1 2016-W1     2565
## 2 2016-W2     2042
## 3 2016-W3     1803
```

5.3 Dealing with missing rows

An important distinction can be made with regards to missing values. Missing values can be missing in one of two ways: **explicitly**, as with `NA` values, or **implicitly**, as when they are simply not present in the data.

To illustrate this point, we will modify our `incid` data created above by removing some of the rows to create implicitly missing values.

To begin with, `incid` has n weeks of data for each woreda.

```
count(incid, woreda_name)
```

```
## # A tibble: 4 x 2
##   woreda_name     n
##   <chr>       <int>
## 1 Fogera        286
## 2 Kalu          286
## 3 Mecha         286
## 4 Metema        286
```

Now let us exclude some of the rows using the dplyr function `sample_frac()`, which selects a random fraction of rows to keep and discards the rest. First we will `set.seed()` so that the R's random number generator will yield the same results for everybody.

```
set.seed(1)
incid_incompl <- sample_frac(incid, 0.9)
count(incid_incompl, woreda_name)
```

```
## # A tibble: 4 x 2
##   woreda_name     n
##   <chr>       <int>
## 1 Fogera        255
## 2 Kalu          261
## 3 Mecha         259
## 4 Metema        255
```

As you can see, each woreda is missing some weeks of data. To convert these missing weeks from implicitly missing to explicitly missing, we can use the dplyr function `complete()`.

```
complete(incid_incompl, woreda_name, iso_year, iso_week)
```

```
## # A tibble: 1,272 x 5
##    woreda_name iso_year iso_week pfm_inc pv_inc
##    <chr>          <dbl>    <dbl>   <dbl>  <dbl>
##  1 Fogera          2012        1      NA     NA
##  2 Fogera          2012        2      NA     NA
##  3 Fogera          2012        3      NA     NA
##  4 Fogera          2012        4      NA     NA
##  5 Fogera          2012        5      NA     NA
##  6 Fogera          2012        6      NA     NA
##  7 Fogera          2012        7      NA     NA
##  8 Fogera          2012        8      NA     NA
##  9 Fogera          2012        9      NA     NA
## 10 Fogera          2012       10      NA     NA
## # ... with 1,262 more rows
```

New rows have been added and `NA`s have been inserted for each variable not named in `complete`, in this case pfm_inc and pv_inc.

But wait! We can see that at least some of the weeks that were added were never in `incid` to begin with. The original dataset before being "completed" began on 2012 week 28. Now it begins on week 1. That's because `complete()` created new rows for every distinct combination of iso_year and iso_week.

What if we only wanted combinations of year and week that were already in the data, i.e. we want the completed dataset to start on 2012 week 28? The answer is to use the dplyr helper function `nesting()`, which causes `complete()` to only add existing combinations of two or more variables, in this case iso_year and iso_week.

```
complete(incid_incompl, woreda_name, nesting(iso_year, iso_week))
```

```
## # A tibble: 1,144 x 5
##    woreda_name iso_year iso_week pfm_inc pv_inc
##    <chr>          <dbl>    <dbl>   <dbl>  <dbl>
##  1 Fogera          2012       28    3.32  0.563
##  2 Fogera          2012       29    3.13  0.582
##  3 Fogera          2012       30    2.63  0.511
##  4 Fogera          2012       31    2.35  0.506
##  5 Fogera          2012       32    2.05  0.506
##  6 Fogera          2012       33    1.93  0.511
##  7 Fogera          2012       34    1.83  0.530
##  8 Fogera          2012       35   NA    NA    
##  9 Fogera          2012       36    1.96  0.587
## 10 Fogera          2012       37    2.24  0.644
## # ... with 1,134 more rows
```

Now we're back to starting on week 28 and having n rows, the same number we had before we randomly removed rows. And we now have a dataset "complete" with explicitly missing values.

5.4 Combining datasets with dplyr

A common need when working with multiple data files is to combine the data into a single dataset. There are two conceptual ways to do this: **binding** and **joining**.

Binding two tables together matches rows or columns by position. The datasets to be bound need to have the same variables or the rows need to be aligned for binding to work.

Joining two tables together matches rows by value rather than position. The datasets need to have some variable in common between them for joining to work.

5.4.1 Binding tables

If data have the same columns, you can use `bind_rows()` to "stack" them one on top of the other in a new data frame.

```
mecha <- read_csv("data_mecha.csv")
fogera <- read_csv("data_fogera.csv")
mf_bind <- bind_rows(mecha, fogera)
mecha
```

```
## # A tibble: 156 x 10
##      WID woreda_name obs_date   test_pf_tot test_pv_only pop_at_risk
##    <dbl> <chr>       <date>           <dbl>        <dbl>       <dbl>
##  1   106 Mecha       2016-01-10           6           15     377232.
##  2   106 Mecha       2016-01-17           8           12     377232.
##  3   106 Mecha       2016-01-24           9           11     377232.
##  4   106 Mecha       2016-01-31           9           11     377232.
##  5   106 Mecha       2016-02-07          11           11     377232.
## 6    106 Mecha       2016-02-14          11            10     377232.
## 7    106 Mecha       2016-02-21          10            10     377232.
## 8    106 Mecha       2016-02-28           9             8     377232.
## 9    106 Mecha       2016-03-06           9             6     377232.
## 10   106 Mecha       2016-03-13           8             6     377232.
## # ... with 146 more rows, and 4 more variables: mal_case <dbl>,
## #   iso_year <dbl>, iso_week <dbl>, data_source <chr>
```

```
count(mf_bind, woreda_name, iso_year)
```

```
## # A tibble: 4 x 3
##   woreda_name iso_year     n
##   <chr>          <dbl> <int>
## 1 Fogera          2018    52
## 2 Mecha           2016    52
## 3 Mecha           2017    52
## 4 Mecha           2018    52
```

If data contain different variables that belong to the same observation, you can combine them by row position using `bind_cols()`. For this to work, the two data frames must have the same number of rows, and the same rows in each dataset must belong the the same observation.

```
mecha_mal_case <- select(mecha, woreda_name, iso_year, iso_week, mal_case)
mecha_pf_pv <- select(mecha, c("test_pf_tot", "test_pv_only"))
bind_cols(mecha_mal_case, mecha_pf_pv)
```

5.5 Joining tables

Table joins are a concept shared across many data science disciplines and are implemented in relational database management systems such as MySQL.

With a join, two tables are connected to each other conceptually through variables called **keys**, which are variables found in both tables. For the datasets you've seen so far, possible key columns include woreda, iso_year, or iso_week.

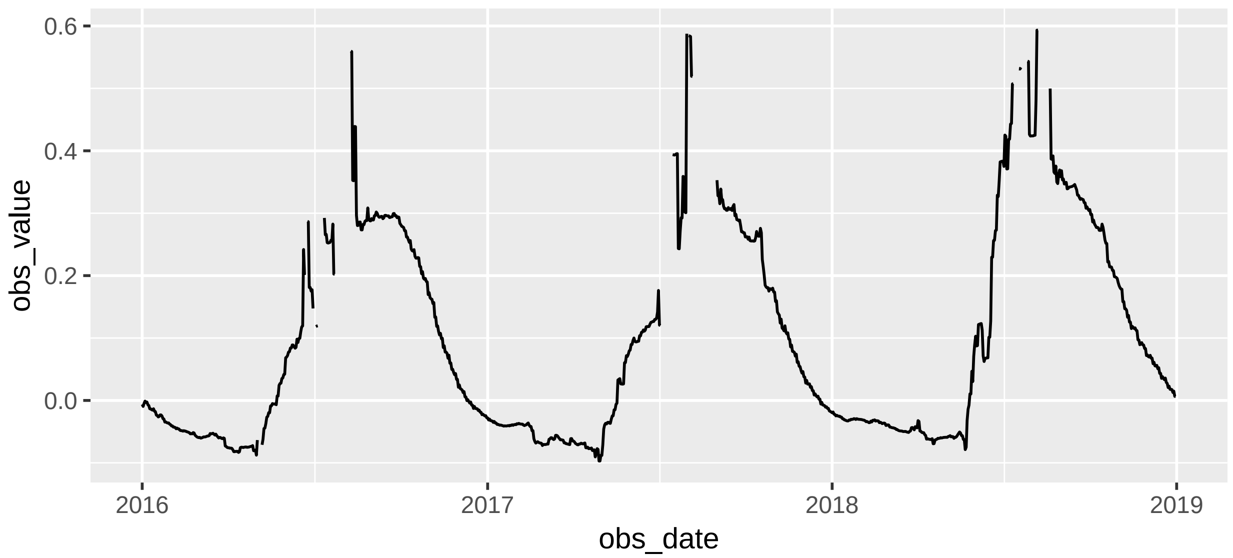
For the following exercises, we will use a new dataset found in `data_ndwi.csv`. This dataset contains satellite-derived values of NDWI, normalized difference water index, an index of vegetation water content (VWC). VWC is related to the abundance of suitable mosquito breeding habitat and is therefore useful when predicting future malaria outbreaks. This dataset is a subset containing years 2016, 2017, and 2018.

The problem is that the NDWI data is daily, while our epidemiological data is weekly. If we want to connect one to the other, we will need to perform two main tasks: summarize the daily data by epi week, and join the new weekly environmental data to the weekly epidemiological data.

```
ndwi <- read_csv("data_ndwi.csv")
glimpse(ndwi)  # like str() but but simpler output
```

```
## Observations: 51,512
## Variables: 5
## $ WID              <dbl> 121, 121, 121, 121, 121, 121, 121, 121, 121, ...
## $ woreda_name      <chr> "Abargelie", "Abargelie", "Abargelie", "Abarg...
## $ environ_var_code <chr> "ndwi6", "ndwi6", "ndwi6", "ndwi6", "ndwi6", ...
## $ obs_date         <date> 2016-01-01, 2016-01-02, 2016-01-03, 2016-01-...
## $ obs_value        <dbl> -0.08590598, -0.08791568, -0.08760408, -0.087...
```

```
ggplot(filter(ndwi, woreda_name=="Mecha")) +
  geom_line(aes(x = obs_date, y = obs_value))
```



The next step is to the daily data by iso week. Unfortunately, we do not have columns for iso_year and iso_week.

```
# convert date to epi year and epi week
ndwi <- mutate(ndwi,
               iso_year = isoyear(obs_date),
               iso_week = isoweek(obs_date))
ndwi
```

```
## # A tibble: 51,512 x 7
##      WID woreda_name environ_var_code obs_date   obs_value iso_year
##    <dbl> <chr>       <chr>            <date>         <dbl>    <dbl>
##  1   121 Abargelie   ndwi6            2016-01-01   -0.0859     2015
##  2   121 Abargelie   ndwi6            2016-01-02   -0.0879     2015
##  3   121 Abargelie   ndwi6            2016-01-03   -0.0876     2015
##  4   121 Abargelie   ndwi6            2016-01-04   -0.0878     2016
##  5   121 Abargelie   ndwi6            2016-01-05   -0.0888     2016
##  6   121 Abargelie   ndwi6            2016-01-06   -0.0879     2016
##  7   121 Abargelie   ndwi6            2016-01-07   -0.0885     2016
##  8   121 Abargelie   ndwi6            2016-01-08   -0.0887     2016
##  9   121 Abargelie   ndwi6            2016-01-09   -0.0906     2016
## 10   121 Abargelie   ndwi6            2016-01-10   -0.0912     2016
## # ... with 51,502 more rows, and 1 more variable: iso_week <dbl>
```

Note that the calendar year in our obs_date column is not always the same as iso_year:

```
filter(ndwi, year(ndwi$obs_date) != iso_year, woreda_name == "Mecha")
```

```
## # A tibble: 5 x 7
##     WID woreda_name environ_var_code obs_date   obs_value iso_year iso_week
##   <dbl> <chr>       <chr>            <date>         <dbl>    <dbl>    <dbl>
## 1   106 Mecha       ndwi6            2016-01-01  -0.00634     2015       53
## 2   106 Mecha       ndwi6            2016-01-02  -0.00964     2015       53
## 3   106 Mecha       ndwi6            2016-01-03  -0.00472     2015       53
## 4   106 Mecha       ndwi6            2017-01-01  -0.0279      2016       52
## 5   106 Mecha       ndwi6            2018-12-31   0.00632     2019        1
```

This is exactly why we must calculate iso_year rather than using calendar year. If we used calendar year, some values would not be grouped correctly when we summarize.

Now we can summarize by epi week.

```
ndwi_wk <- group_by(ndwi, woreda_name, iso_year, iso_week)
ndwi_wk <- summarize(ndwi_wk,
                     ndwi = mean(obs_value, na.rm = TRUE),
                     n = sum(!is.na(obs_value)))
ndwi_wk
```

```
## # A tibble: 7,426 x 5
## # Groups:   woreda_name, iso_year [?]
##    woreda_name iso_year iso_week    ndwi     n
##    <chr>          <dbl>    <dbl>   <dbl> <int>
##  1 Abargelie       2015       53 -0.0871     3
##  2 Abargelie       2016        1 -0.0891     7
##  3 Abargelie       2016        2 -0.0923     7
##  4 Abargelie       2016        3 -0.0933     7
##  5 Abargelie       2016        4 -0.0915     7
##  6 Abargelie       2016        5 -0.0917     7
##  7 Abargelie       2016        6 -0.0923     7
##  8 Abargelie       2016        7 -0.0915     7
##  9 Abargelie       2016        8 -0.0949     7
## 10 Abargelie       2016        9 -0.0955     7
## # ... with 7,416 more rows
```

However, many of those mean values were estimated using fewer than half the days in the week, and may therefor be poor approximations of the true mean. To be safe, we will exclude summaries calcualted from fewer than 6 days of the week.

```
ndwi_wk <- filter(ndwi_wk, n >= 6)
ndwi_wk
```

```
## # A tibble: 6,300 x 5
## # Groups:   woreda_name, iso_year [141]
##    woreda_name iso_year iso_week    ndwi     n
##    <chr>          <dbl>    <dbl>   <dbl> <int>
##  1 Abargelie       2016        1 -0.0891     7
##  2 Abargelie       2016        2 -0.0923     7
##  3 Abargelie       2016        3 -0.0933     7
##  4 Abargelie       2016        4 -0.0915     7
##  5 Abargelie       2016        5 -0.0917     7
##  6 Abargelie       2016        6 -0.0923     7
##  7 Abargelie       2016        7 -0.0915     7
##  8 Abargelie       2016        8 -0.0949     7
##  9 Abargelie       2016        9 -0.0955     7
## 10 Abargelie       2016       10 -0.0948     7
## # ... with 6,290 more rows
```

At this point, we are ready to join the NDWI data with the epidemiological data.

5.5.1 Inner join

The first type of join we will perform is called an **inner join**. With this type of join, rows are included in the output only if there are matching key values in both tables.

We will start with our `mecha` dataset and join the `ndwi_wk` dataset using `inner_join()`. Before we do that, let's remove some columns from `mecha` so we can more easily see what is happening when we join.

```
mecha_mal_case <- select(mecha, woreda_name, iso_year, iso_week, mal_case)
```

Because our datasets have three columns in common (woreda, iso_year, iso_week) and it takes all three columns to uniquely identify an observation in one datset or the other, all three columns will serve as our keys.

```
inner_join(mecha_mal_case, ndwi_wk, by = c("woreda_name", "iso_year", "iso_week"))
```

```
## # A tibble: 139 x 6
##    woreda_name iso_year iso_week mal_case     ndwi     n
##    <chr>          <dbl>    <dbl>    <dbl>    <dbl> <int>
##  1 Mecha           2016        1       21 -0.00697     7
##  2 Mecha           2016        2       20 -0.0181      7
##  3 Mecha           2016        3       20 -0.0261      7
##  4 Mecha           2016        4       20 -0.0362      7
##  5 Mecha           2016        5       22 -0.0429      7
##  6 Mecha           2016        6       21 -0.0476      7
##  7 Mecha           2016        7       20 -0.0507      7
##  8 Mecha           2016        8       17 -0.0545      7
##  9 Mecha           2016        9       15 -0.0595      7
## 10 Mecha           2016       10       14 -0.0569      7
## # ... with 129 more rows
```

Looking at the resulting table, we can see that the columns in `mecha` are all present in the output, in the same order, followed by the non-key columns from `ndwi_wk`.

We can also see that the resulting dataset has only n rows. `mecha` had n rows, so some rows must not have had a key match between the two tables. Examining more closely and we can see that YYYY week W was not present in `ndwi_wk` and is also not present in the joined tables.

5.5.2 Outer joins

It may be perfectly fine to have rows with non-matching keys removed by an inner join. It depends on your goal. However for our demonstration lets assume we want to keep all rows from `mecha`. This is possible using a **left join**. With this type of join, all rows from the table on the "left" side of the join are kept, and missing values are inserted where there is no matching row in the table on the "right" side.

We can perform a left join using `left_join()`, where the first argument is the left table and the second argument is the right table.

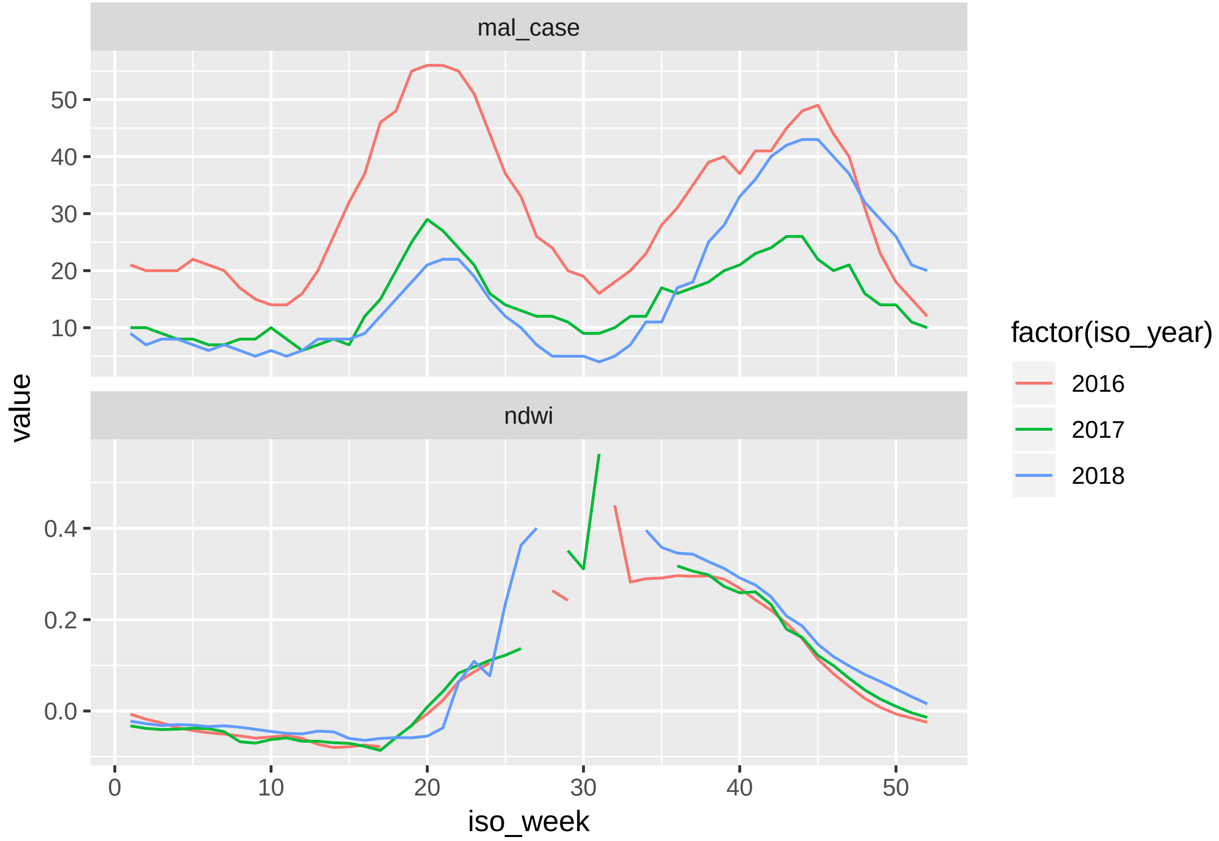
```
lj <- left_join(mecha_mal_case, ndwi_wk, by = c("woreda_name", "iso_year", "iso_week"))
lj
```

```
## # A tibble: 156 x 6
##    woreda_name iso_year iso_week mal_case     ndwi     n
##    <chr>          <dbl>    <dbl>    <dbl>    <dbl> <int>
##  1 Mecha           2016        1       21 -0.00697     7
##  2 Mecha           2016        2       20 -0.0181      7
##  3 Mecha           2016        3       20 -0.0261      7
##  4 Mecha           2016        4       20 -0.0362      7
##  5 Mecha           2016        5       22 -0.0429      7
##  6 Mecha           2016        6       21 -0.0476      7
##  7 Mecha           2016        7       20 -0.0507      7
##  8 Mecha           2016        8       17 -0.0545      7
##  9 Mecha           2016        9       15 -0.0595      7
## 10 Mecha           2016       10       14 -0.0569      7
## # ... with 146 more rows
```

You can see that the resulting table still has n rows. The first row, for YYYY week W, was missing after the left join. Here it is present, with missing values for the variables ndwi and n, the two that came from the table on the right.

Using `gather()` we can easily plot the joined dataset.

```
ggplot(gather(lj, key, value, mal_case, ndwi)) +
  geom_line(aes(iso_week, value, color = factor(iso_year))) +
  facet_wrap(~ key, ncol = 1, scales = "free_y")
```



6 Day 3 exercises

1. Read the `data_2016_2018.csv` dataset.
2. Summarize the data. Calculate the min, max, and mean values of mal_case for each woreda.
3. Now summarize the data by woreda *and* year.
4. Starting with the data_mecha.csv data, create a tidy data frame with columns for iso_year, iso_week, and malaria agent (P. falciparum vs. P. vivax). Unnecessary columns can be removed.
5. Plot this tidy data frame using colors for malaria agent and facets for years.
6. Create a data frame that contains both the daily *and* weekly NDWI data for a woreda and year (2016 - 2018) of your choice. Plot the values on the same plot, with Date on the x axis and NDWI on the y axis. Use either point or line geoms and choose aesthetics to help differentiate the daily and weekly data.
7. Create a data frame that contains both the daily and weekly NDWI data for a woreda and year of your choice. Plot the values on the same plot, with Date on the x axis and NDWI on the y axis. Use either point or line geoms and choose aesthetics to help differentiate the daily and weekly data.