Introduction to R for Disease Surveillance and Outbreak Forecasting: Day 2

Exploring data with effective visualizations

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

October 23 2018, Bahir Dar, Ethiopia

Contents

1 Introduction

This tutorial will teach you how to work with and visualize your data in R. By the end of today's lesson, you will be able to visualize data from data tables.

Over the past 20 years R has become increasingly popular for statistical programming. One of R's strengths has always been that it makes it easy for anybody to join the R community and contribute to R's capabilities by creating new packages, and recently the number of R packages on CRAN and elsewhere has grown at an astonishing rate. As a result, there are many new packages that improve on the data manipulation and plotting functions included with the basic R installation.

Today we begin to explore one of those packages: ggplot2. The package ggplot2 offers a versatile system for plotting data frames and can even be used to generate maps.

2 Basic graphics

In R, a function takes one or more *arguments* as inputs and then produces an object as an output. So far we have used a variety of relatively simple functions to do basic data manipulation and print out results to the console. Moving forward, the key to understanding how to do more advanced data processing, data visualization, and statistical analysis and R is learning which functions to use and how to specify the appropriate arguments to get these functions to do what you want. The following section will illustrate the use of some functions to generate basic graphics.

First, we will read a dataset from an external file named `"demo_data.csv"`. Make sure this file is in your working directory before you try to read it. We first have to load the `readr` package to get access to the `read_csv()` function. Then we use this function to read the file and put it into a data frame.

```r
library(readr)
demo_data <- read_csv(file = "data_day1-2.csv")
```

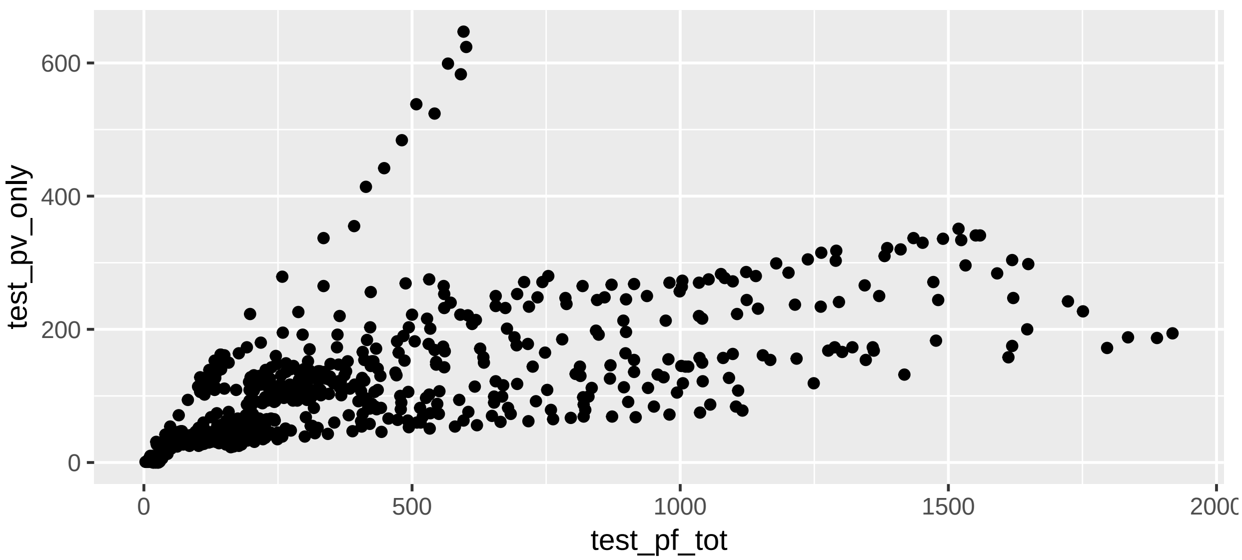
```
## Parsed with column specification:
## cols(
##   WID = col_double(),
##   woreda_name = col_character(),
##   obs_date = col_date(format = ""),
##   test_pf_tot = col_double(),
##   test_pv_only = col_double(),
##   pop_at_risk = col_double(),
##   mal_case = col_double(),
##   iso_year = col_double(),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

```r
summary(demo_data)
```

```
##       WID       woreda_name           obs_date           test_pf_tot
##  Min.   :12.0   Length:676         Min.   :2012-07-15   Min.   :   3.0
##  1st Qu.:12.0   Class :character   1st Qu.:2014-02-23   1st Qu.: 125.0
##  Median :17.5   Mode  :character   Median :2015-10-07   Median : 264.0
##  Mean   :17.5                      Mean   :2015-10-07   Mean   : 414.4
##  3rd Qu.:23.0                      3rd Qu.:2017-05-21   3rd Qu.: 568.2
##  Max.   :23.0                      Max.   :2018-12-30   Max.   :1918.0
##   test_pv_only    pop_at_risk        mal_case          iso_year
##  Min.   :  0.0   Min.   :101822   Min.   :   4.0   Min.   :2012
##  1st Qu.: 39.0   1st Qu.:144711   1st Qu.: 182.2   1st Qu.:2014
##  Median : 97.5   Median :181101   Median : 374.0   Median :2015
##  Mean   :110.5   Mean   :186840   Mean   : 524.9   Mean   :2015
##  3rd Qu.:149.2   3rd Qu.:252718   3rd Qu.: 757.0   3rd Qu.:2017
##  Max.   :647.0   Max.   :266612   Max.   :2112.0   Max.   :2018
##     iso_week     data_source
##  Min.   : 1.00   Length:676
##  1st Qu.:15.00   Class :character
##  Median :28.50   Mode  :character
##  Mean   :27.58
##  3rd Qu.:41.00
##  Max.   :53.00
```

Now we can use the qplot function to generate a scatterplot of *Plasmodium faciparum*/mixed cases versus *Plasmodium vivax* cases. This function is part of the `ggplot2` package, and it allows us to quickly create basic graphs with relatively simple function calls. We will learn how to make more customizable graphs using the powerful `ggplot()` function later this week. For now, we will specify a few arguments to indicate the variable to be plotted on the x-axis, the variable to be plotted on the y-axis, and the data frame containing these variables.

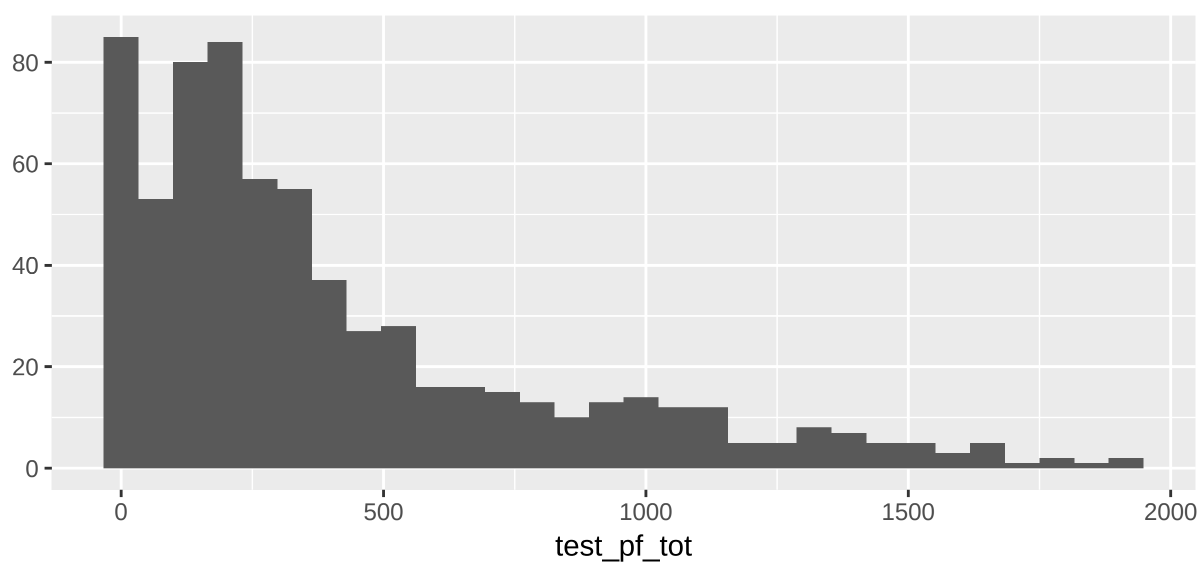
```r
library(ggplot2)
qplot(x=test_pf_tot, y=test_pv_only, data=demo_data)
```



Here we specify only a single data argument along with "histogram" for the geom argument to create a histogram

```
qplot(x=test_pf_tot, data=demo_data, geom="histogram")
```

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



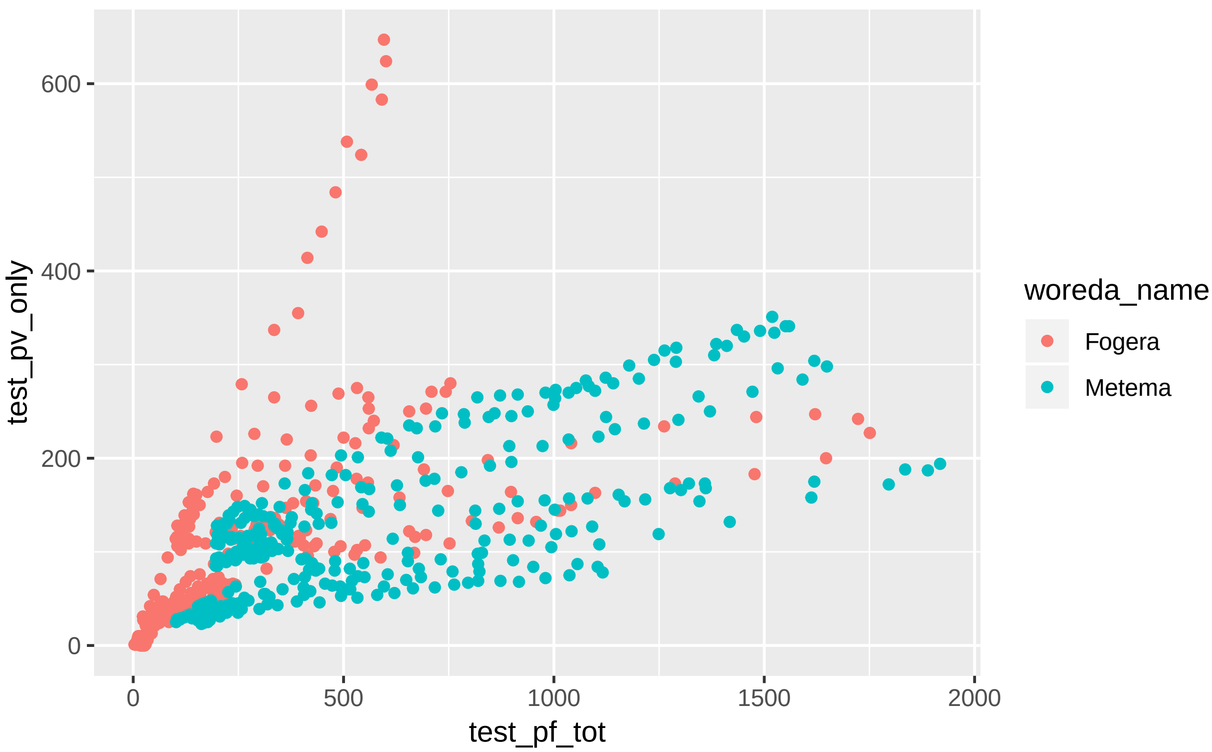
Switching back to the scatterplot, we can specify the colour argument to plot woredas or years as different colors.

```
library(tidyverse)
```

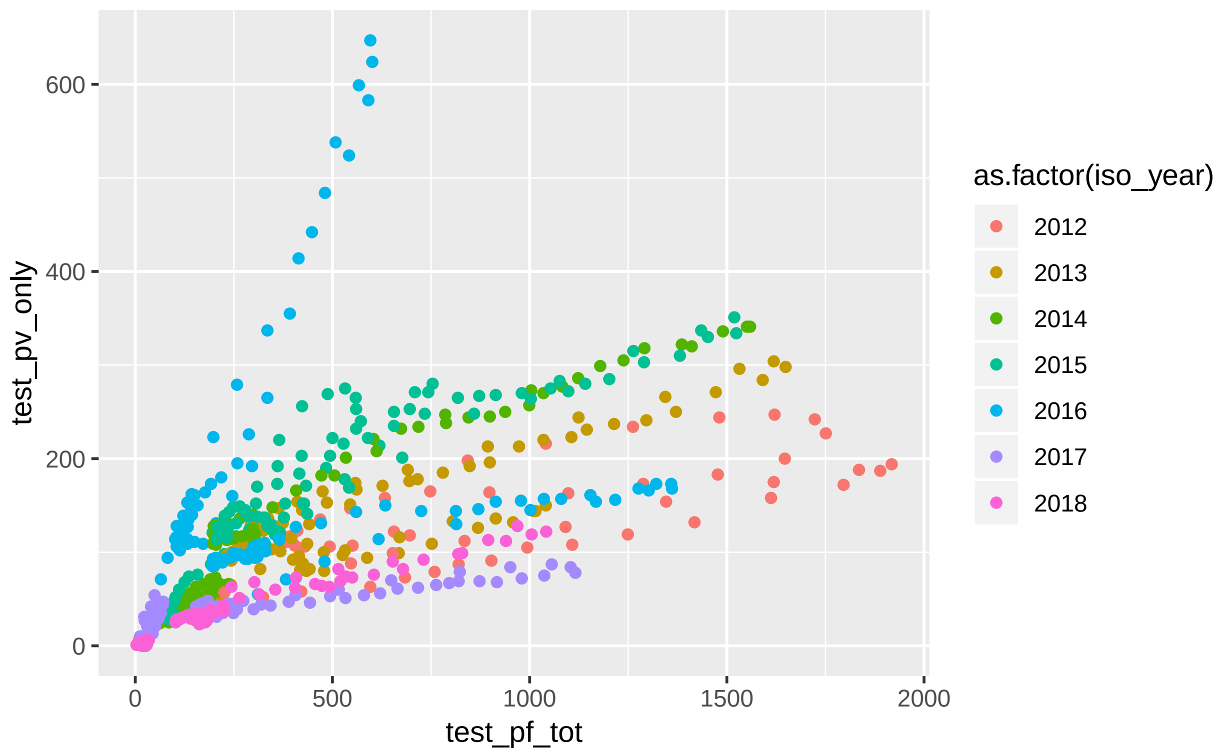
```
## -- Attaching packages --------------------------------------------------- tidyverse 1.2.1 --

## v tibble  2.0.1     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v purrr   0.3.0     v forcats 0.3.0

## -- Conflicts ------------------------------------------------------- tidyverse_conflicts() --
```

```
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
qplot(x=test_pf_tot, y=test_pv_only, colour=woreda_name, data=demo_data)
```




```r
qplot(x=test_pf_tot, y=test_pv_only, colour=as.factor(iso_year), data=demo_data)
```

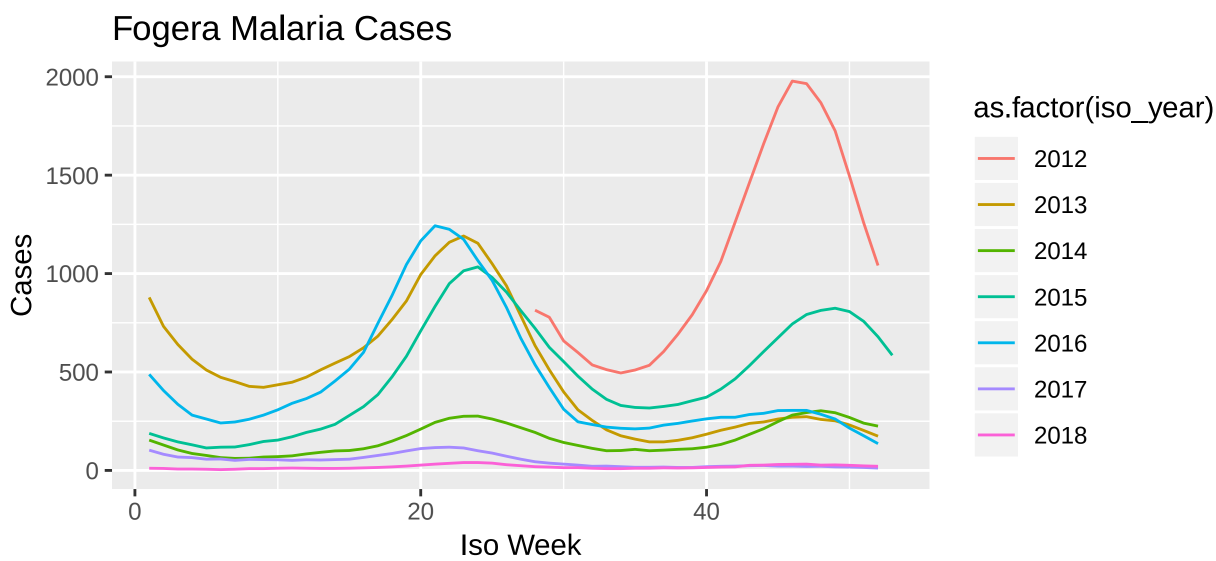


Here, we select a subset of the data for one of the woredas and plot malaria cases using lines instead of points. Additional arguments add text labels to the axes.

```
library(lubridate)
```

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

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

```
qplot(x=isoweek(obs_date), y=mal_case, colour=as.factor(iso_year),
      data=filter(demo_data, woreda_name=="Fogera"), geom="line",
      main="Fogera Malaria Cases", xlab="Iso Week", ylab = "Cases")
```



Clearly, knowing the arguments that can be specified for a specific function is very important. The quickest way to learn more about an R function is to display its documentation using the `help()` function.

```r
help(qplot)
```

3 Plotting with ggplot2

While R has several different systems for making plots (also called graphs or visualizations), the ggplot2 package is one of the most versatile because it allows you to use one system to visualize many different kinds of data.

To start, we will read in a simple dataset with which to practice plotting. The file `fogera.csv` contains one year of malaria case data from Fogera woreda. Read the dataset using `read_csv()` as you did yesterday.

```r
library(readr)
fogera <- read_csv("data_fogera.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
fogera
```

```
## # 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    23 Fogera      2018-01-07           9            2     261898.
```

```
## 2    23 Fogera     2018-01-14          8             2     261898.
## 3    23 Fogera     2018-01-21          6             1     261898.
## 4    23 Fogera     2018-01-28          6             1     261898.
## 5    23 Fogera     2018-02-04          5             1     261898.
## 6    23 Fogera     2018-02-11          3             1     261898.
## 7    23 Fogera     2018-02-18          5             1     261898.
## 8    23 Fogera     2018-02-25          7             2     261898.
## 9    23 Fogera     2018-03-04          7             2     261898.
## 10   23 Fogera     2018-03-11          8             3     261898.
## # ... with 42 more rows, and 4 more variables: mal_case <dbl>,
## #   iso_year <dbl>, iso_week <dbl>, data_source <chr>
```

To access the ggplot2 functions and help pages, load the ggplot package:

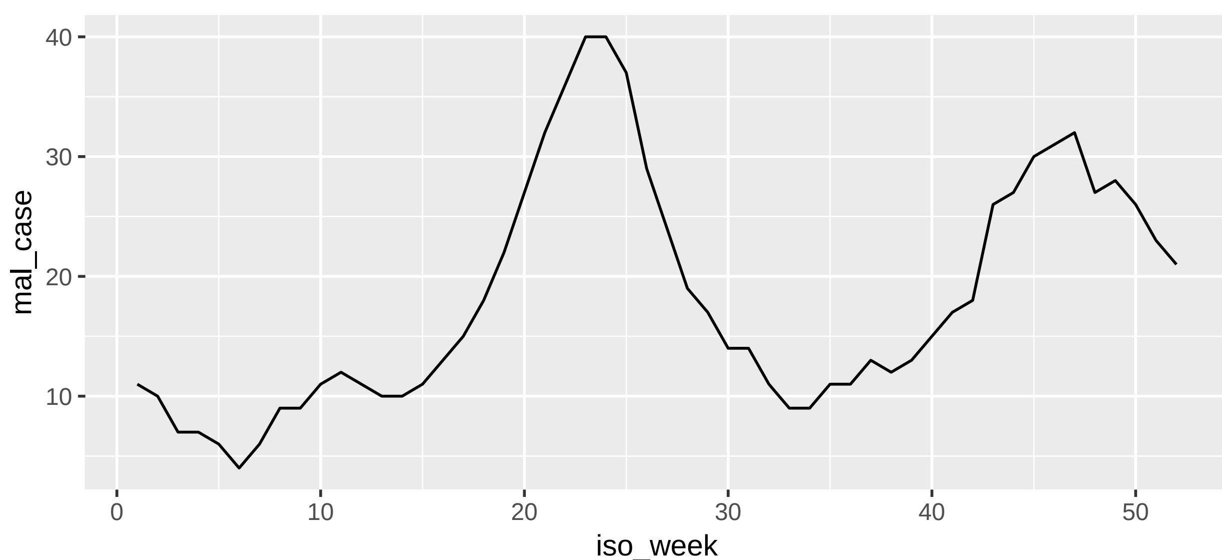
```
library(ggplot2)
```

3.1 Creating a ggplot

One of the most common types of plot in epidemiology is a time series plot, in which the date or time element is on the x-axis and the measured variable of interest is on the y-axis. The data values are usually connected by a line to indicate progression through time.

Use this code to plot the number of malaria cases in Fogera during 2017.

```
ggplot(data = fogera) +
  geom_line(mapping = aes(x = iso_week, y = mal_case))
```



You use `ggplot()` to create a coordinate system onto which you may add layers. The first argument to `ggplot()` is always `data`, the dataset to plot. You could run `ggplot(fogera)` to create just the blank coordinate system, but it's not very interesting so its not shown here.

Once you have a blank plot, you can add layers to it. For example, `geom_line()` in the example above adds lines to the plot. The `mapping` argument specifies the aesthetic mapping to use. Aesthetic mappings tell ggplot which columns in the dataset get used for (or "mapped to") which features of the plot. The mapping is always specified by the `aes()` function.

In our example, the mapping tells ggplot that the `iso_week` column contains x-axis values and the `mal_case` column contains y-axis values.

3.1.1 A reuseable template

The above code can be generalized to a reuseable template for plotting with `ggplot()`:

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

3.2 Aesthetic mappings

You've already learned about mapping data columns to x and y axes. If you want to visualize a third column, you will have to map it to some other part of the plot.

To illustrate, we will read in the dataset in `mecha.csv`, which contains the same variables as the Fogera dataset, but includes three years of data.

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

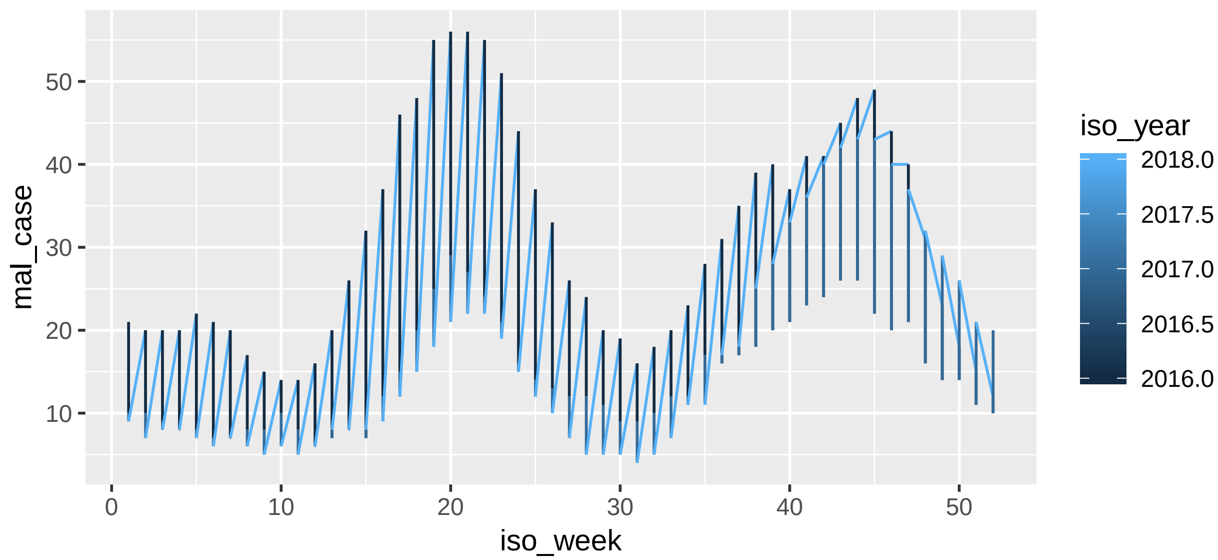
```
## Parsed with column specification:
## cols(
##   WID = col_double(),
##   woreda_name = col_character(),
##   obs_date = col_date(format = ""),
##   test_pf_tot = col_double(),
##   test_pv_only = col_double(),
##   pop_at_risk = col_double(),
##   mal_case = col_double(),
##   iso_year = col_double(),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

```
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>
```

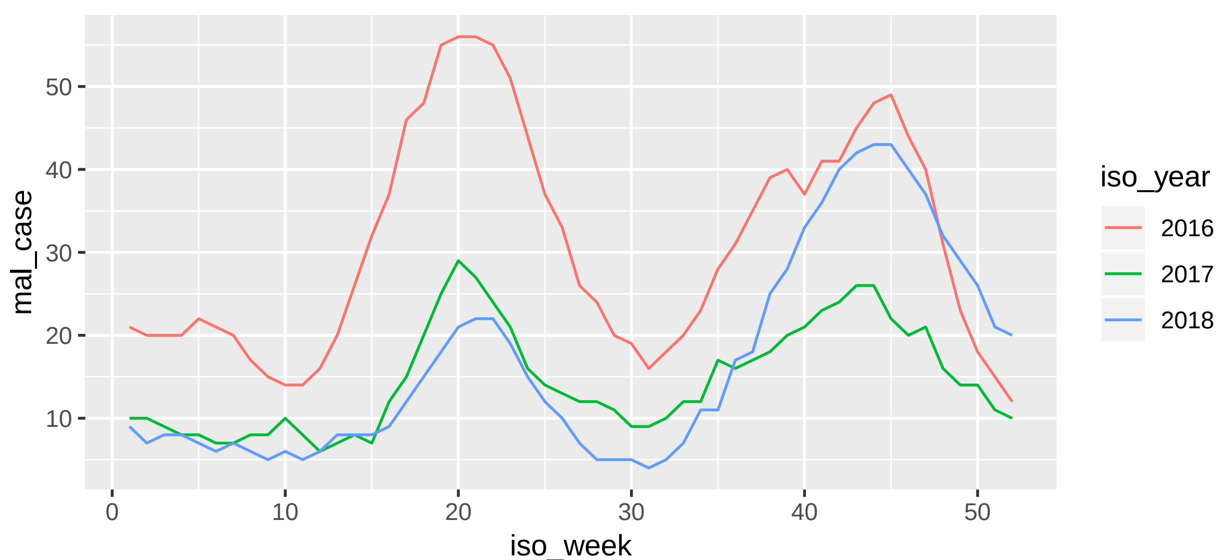
Because there are three years of data in this dataset, we need some way to tell the lines for the three years apart. One common choice is to show different lines with different colors. Here we will "map" the `iso_year` column to the `color` aesthetic like this:

```
ggplot(data = mecha) +
  geom_line(mapping = aes(x = iso_week, y = mal_case, color = iso_year))
```



Note that ggplot is treating iso_year as a continuous variable and trying to draw a line connecting points across years. This is the default behavior for data columns of class "numeric". To treat iso_year as a categorical variable, we can use `factor()` to convert the numerical vector of years to a categorical vector of years.

```
mecha$iso_year <- factor(mecha$iso_year)
ggplot(data = mecha) +
  geom_line(mapping = aes(x = iso_week, y = mal_case, color = iso_year))
```

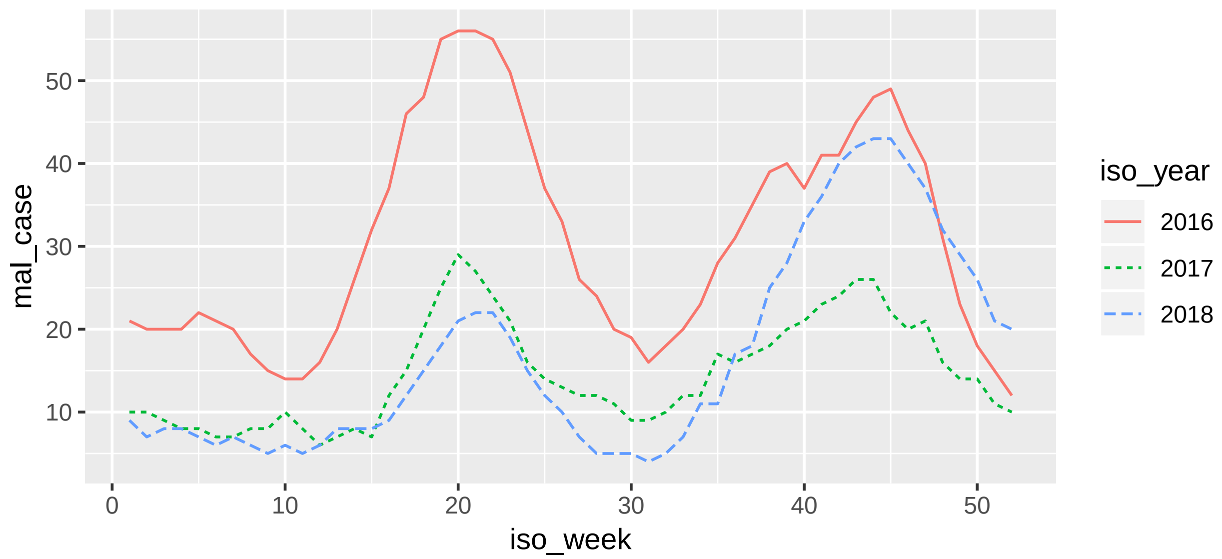


The blue color gradient used in the legend has been replaced by a discrete color legend with unique colors for each year. You can always tell whether ggplot is treating one of your variables as continuous or categorical by whether it uses a gradient or discrete color legend.

Other aesthetics for lines include linetype, size, and alpha, which controls transparency. For points, there is also a shape aesthetic.

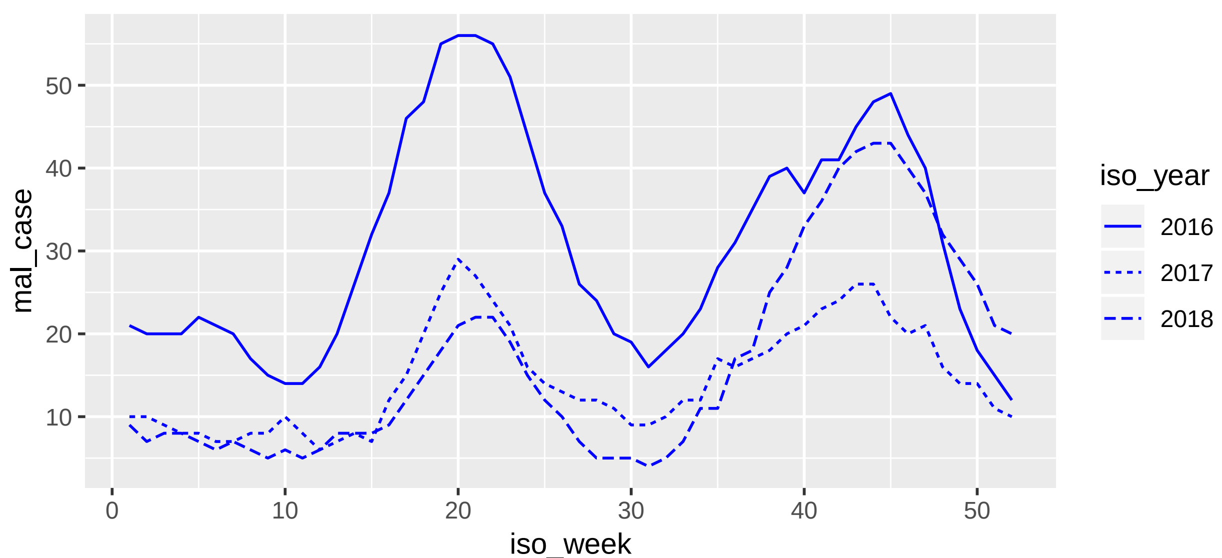
You can even assign multiple aesthetics to the same column:

```
ggplot(data = mecha) +
  geom_line(mapping = aes(x = iso_week, y = mal_case,
                          color = iso_year, linetype = iso_year))
```



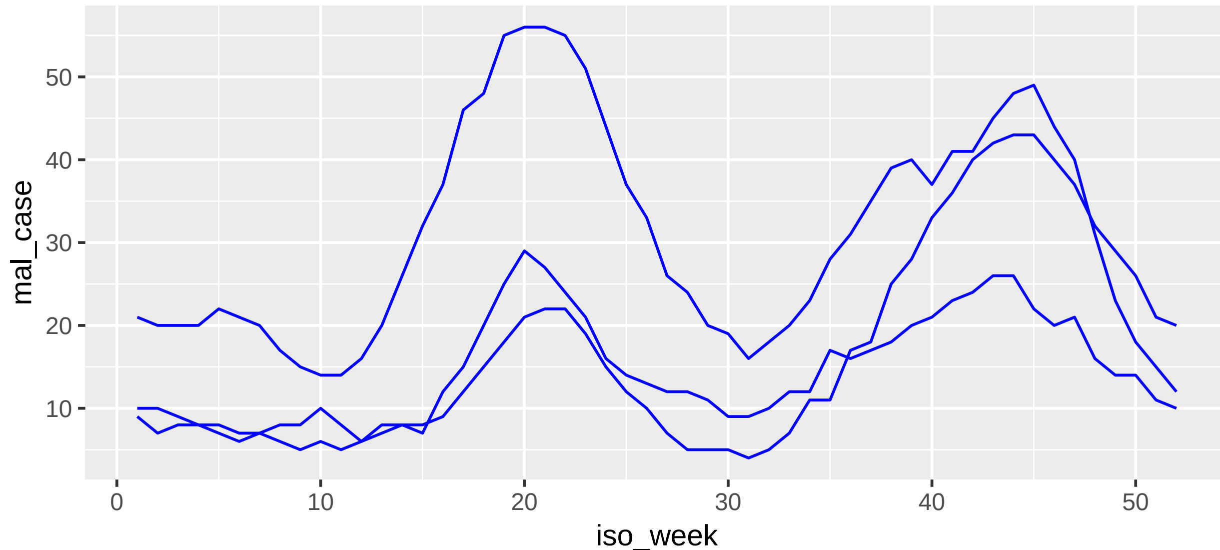
You can also set an aesthetic to a fixed value by defining that aesthetic outside of the `aes()` function. For example, to turn all the lines blue, you would use this:

```
ggplot(data = mecha) +
  geom_line(aes(x = iso_week, y = mal_case, linetype = iso_year),
            color = "blue")
```



Sometimes we want to plot groups of data but we don't want to map the groups to any particular aesthetic such as color or linetype. In these cases, you can use the `group` argument inside the `aes()` function:

```
# no year aesthetic
ggplot(data = mecha) +
  geom_line(aes(x = iso_week, y = mal_case, group = iso_year),
            color = "blue")
```



The years are still plotted correctly, but it is no longer possible to tell which year is which. This may be desirable if the goal is just to show the variability from year to year without identifying individual years.

For a complete overview of all the possible aesthetic specifications, run the command `vignette('ggplot2-specs')` to view the **Aesthetic specifications** vignette.

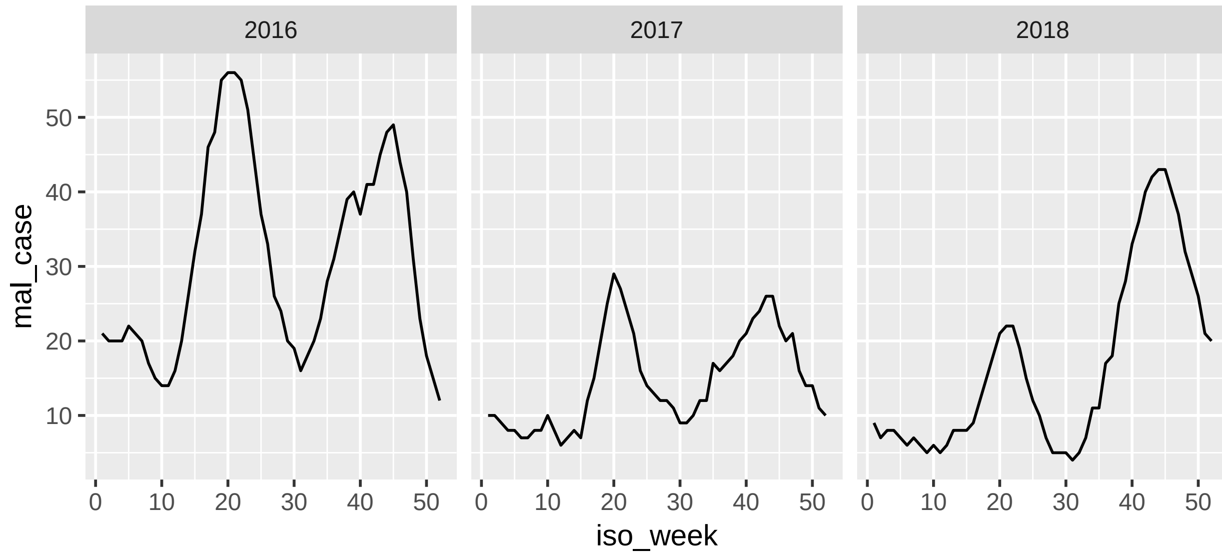
3.3 Facets

Another way to view additional variables on your plot is to use facets. Facetting allows you to split your dataset into subsets.

For example, the plot above is very crowded with three years of data on it. It might be easier to read if each year of data was on its own subplot, or facet.

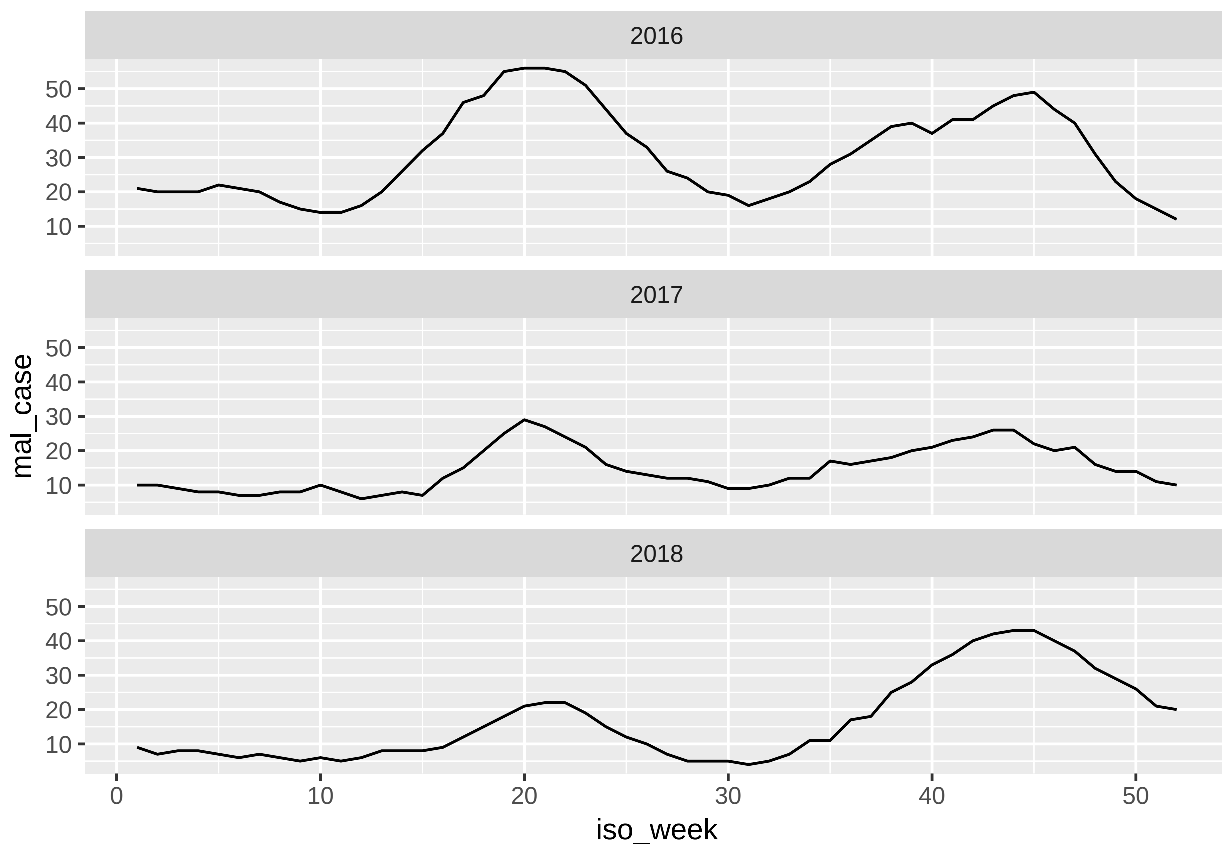
To facet by a single variable, use `facet_wrap()` like this.

```
ggplot(data = mecha) +
  geom_line(mapping = aes(x = iso_week, y = mal_case)) +
  facet_wrap(~ iso_year)
```



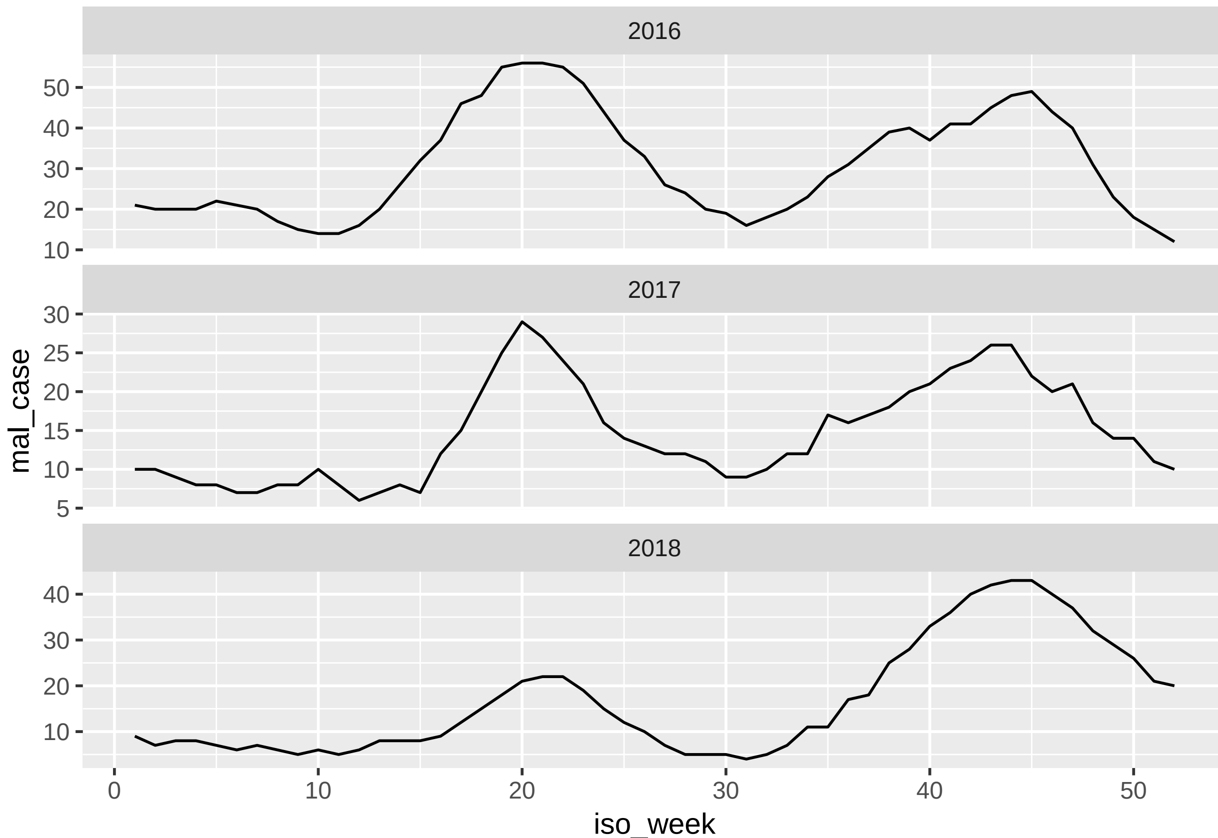
Because these are time series data, it might be nice to have the stack the subplots on top of each other instead of having them side-by-side. You can change the layout using `ncol` or `nrow` to specify the number of colums or rows.

```
ggplot(data = mecha) +
  geom_line(mapping = aes(x = iso_week, y = mal_case)) +
  facet_wrap(~ iso_year, ncol = 1)
```



You may notice that the maximum number of cases in 2014 was significantly higher than in other years, making it hard to see the seasonality 2015 and 2016. To allow the data in those years to stretch to take up the entire vertical space, you can use the `scales` argument. In this case, set it to `"free_y"`.

```
ggplot(data = mecha) +
  geom_line(mapping = aes(x = iso_week, y = mal_case)) +
  facet_wrap(~ iso_year, ncol = 1, scales = "free_y")
```



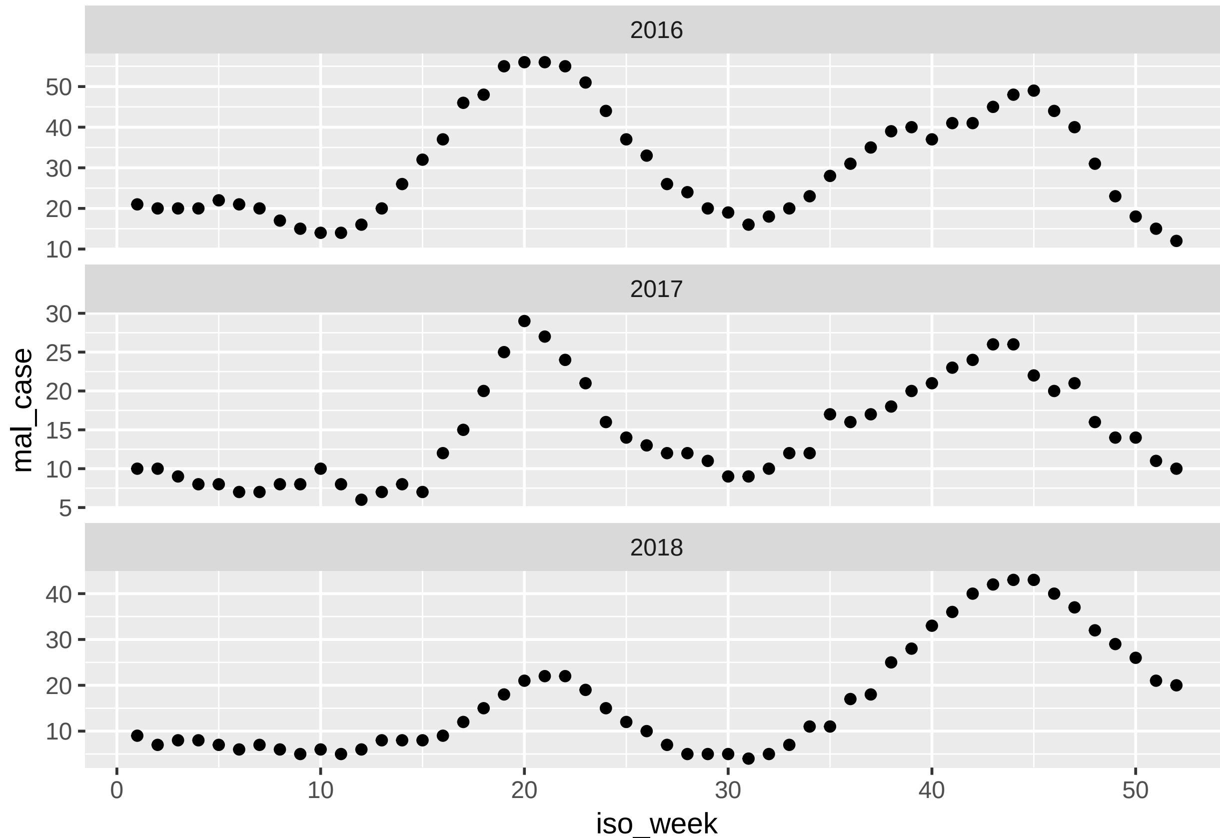
Notice how the range of the y axis varies between subplots now.

3.4 Geometric objects

A **geom** is the geometrical object that a plot uses to represent data. There are often multiple ways to represent the same data visually. For example, we have been using line geometries to visualize our time series of malaria case data. An alternative might be to use points for each value.

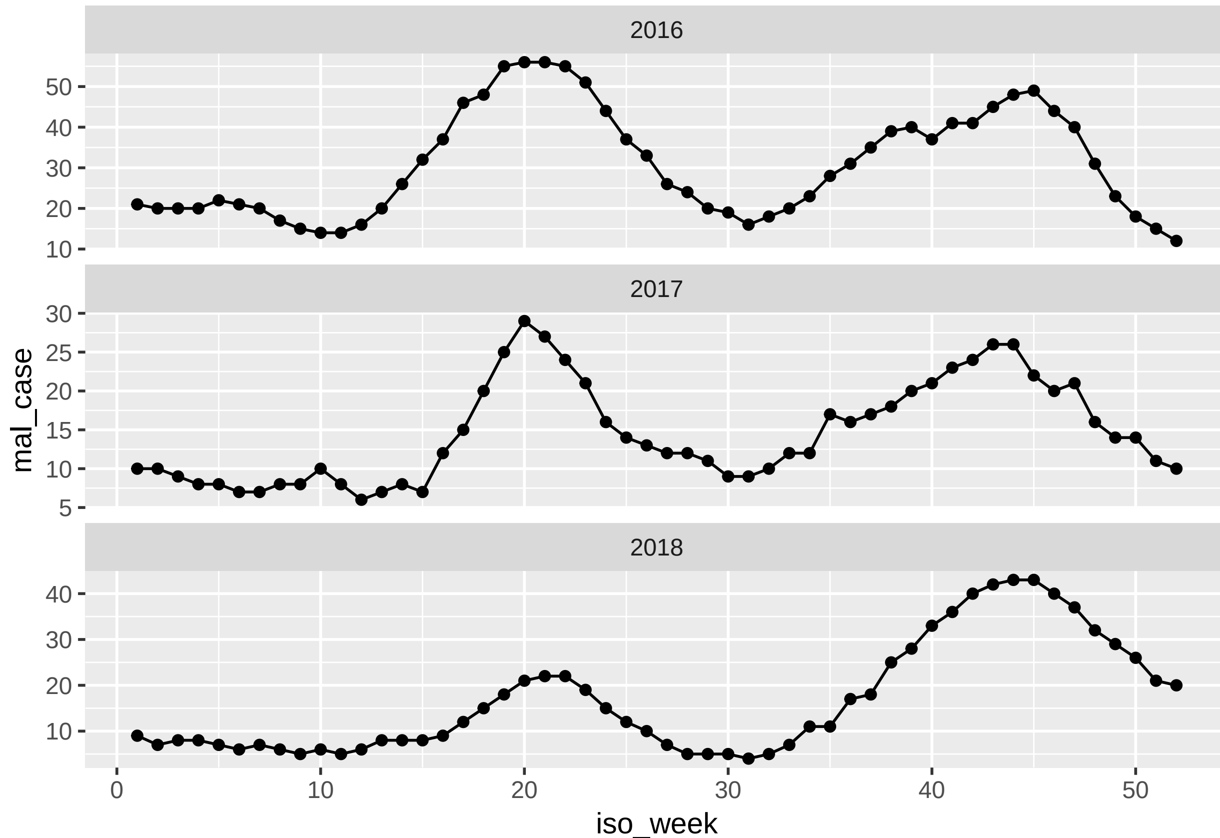
The function for a point geom is `geom_point()`.

```
ggplot(data = mecha) +
  geom_point(mapping = aes(x = iso_week, y = mal_case)) +
  facet_wrap(~ iso_year, ncol = 1, scales = "free_y")
```



You can even represent the same data using multiple geoms!

```
ggplot(data = mecha) +
  geom_line(mapping = aes(x = iso_week, y = mal_case)) +
  geom_point(mapping = aes(x = iso_week, y = mal_case)) +
  facet_wrap(~ iso_year, ncol = 1, scales = "free_y")
```



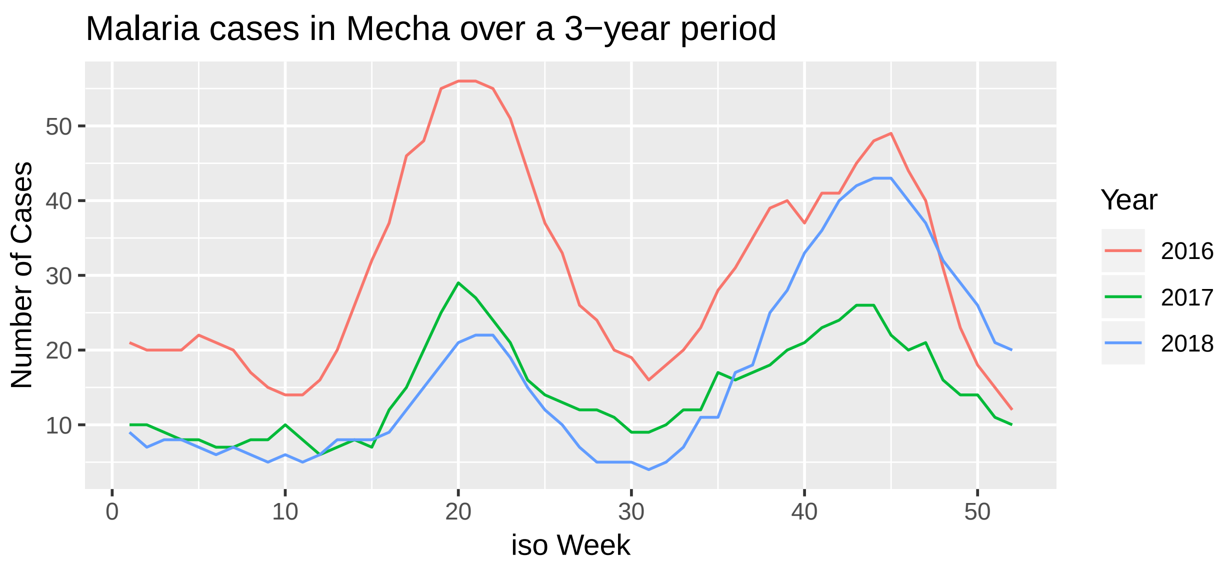
Some aesthetics can only be used with certain geoms. For example, points can have a shape aesthetic, but lines cannot. Conversely, lines can have a linetype aesthetic, but points can not.

3.5 Scales

Scales control the details of how data values are translated to visual properties. We can override the default scales to tweak details like the axis labels or legend keys, or to use a completely different translation from data to aesthetic.

For example, to change the axis, legend, and plot labels we can use the `labs()` function:

```
ggplot(data = mecha) +
  geom_line(mapping = aes(x = iso_week, y = mal_case, color = iso_year)) +
  labs(x = "iso Week", y = "Number of Cases", color = "Year",
       title = "Malaria cases in Mecha over a 3-year period")
```

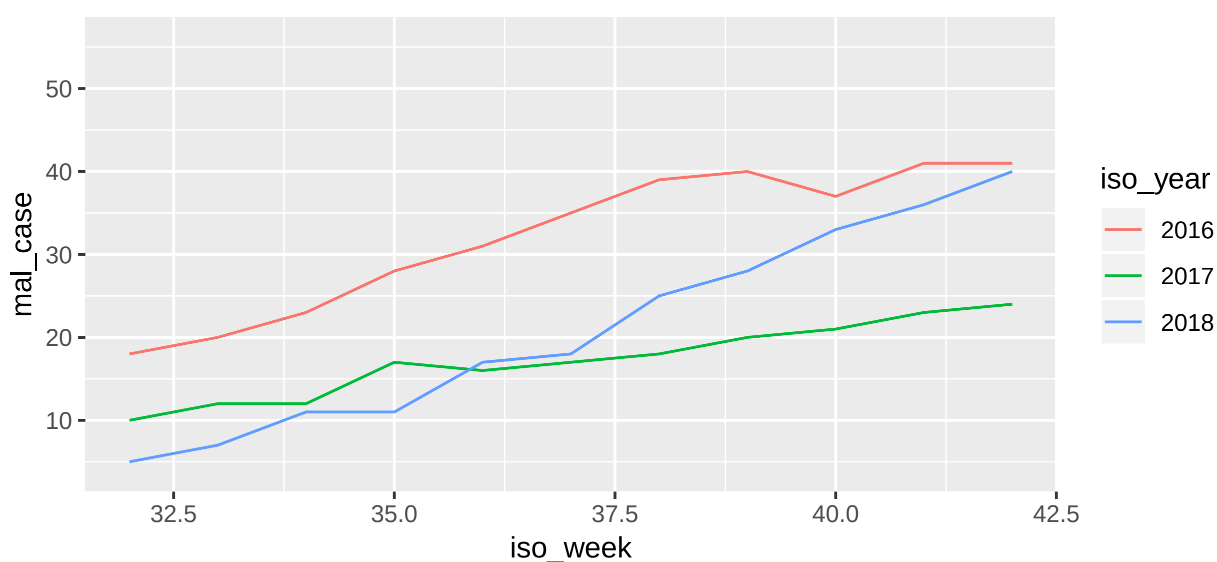


Other possible arguments to `labs()` include `subtitle`, `caption`, and any other aesthetics you have mapped, e.g. `linetype` or `shape`.

To adjust the limits of the x and y axes, you can use `lims()`:

```
ggplot(data = mecha) +
  geom_line(mapping = aes(x = iso_week, y = mal_case, color = iso_year)) +
  lims(x = c(32, 42))
```

```
## Warning: Removed 123 rows containing missing values (geom_path).
```

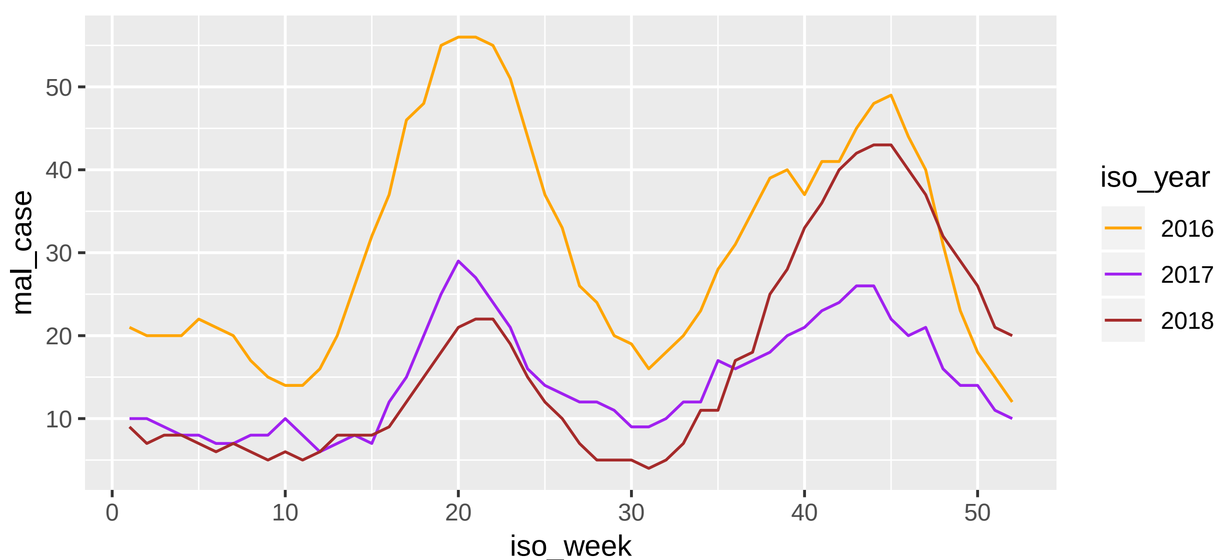


Other scales let you adjust the x and y axes even more. For example, to plot the y axis on the log scale, which is often desirable when dealing with count data where most values are low but a few very high values occur, you can use `scale_y_log10()`:

```
ggplot(data = mecha) +
  geom_line(mapping = aes(x = iso_week, y = mal_case, color = iso_year)) +
  scale_y_log10()
```

To set the color manually, we can use `scale_color_manual()`:

```
ggplot(data = mecha) +
  geom_line(mapping = aes(x = iso_week, y = mal_case, color = iso_year)) +
  scale_color_manual(values = c("orange", "purple", "brown"))
```



Colors can be specified in several ways in R. The simplest way is with a character string giving the color name (e.g., "red"). A list of the possible colors can be obtained with the function `colors()`. Alternatively, colors can be specified directly in terms of their RGB components with a string of the form "#RRGGBB". For more information see the "Color Specification" section under `help("par")`.

Each aesthetic has its own scale functions, e.g. `scale_linetype()` and `scale_size()`. For more on scales, the best references is the ggplot documentation website: http://ggplot2.tidyverse.org/reference/#section-scales.

3.6 The generalized ggplot template

Up to this point, you have learned how to:

1. create a ggplot using `ggplot()`
2. add geometric representations of data to a plot using geoms
3. map data columns to plot aesthetics
4. split your dataset into subplots using facets.
5. control the visual properties of your plot using scales

These steps can be combined to create generalized code for plotting in ggplot.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>)) +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <SCALE_FUNCTION>
```

This is the template that we use for most of the plots in this workshop. Once you master it, the plots you can create will allow you to learn a lot about your data. This process is part of *data exploration*.

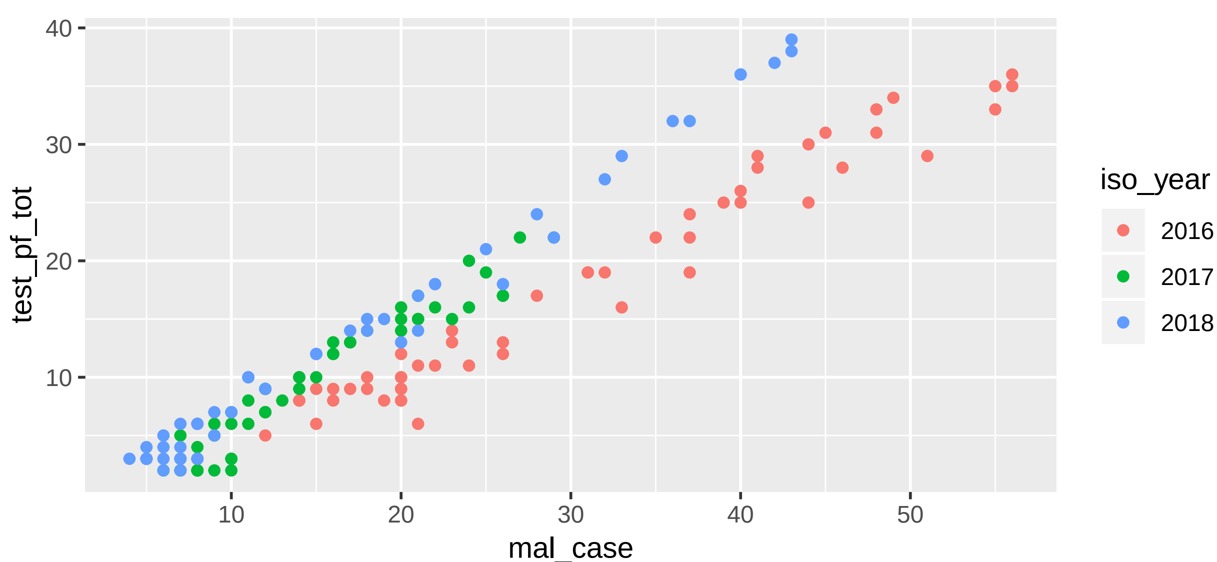
On the other hand, communicating your understanding to others requires considerable effort in making your plots as self-explanatory as possible. To learn more about how to do this, we recommend the "Graphics for communication" chapter of "R for Data Science" by Grolemund and Wickham: http://r4ds.had.co.nz/graphics-for-communication.html.

3.7 Other types of plots

3.7.1 Scatterplot

Scatterplots are used to show the relationship between two variables. They are like time series plots, but they always use points for the geometry and both axes are measured variables rather than just the y axis.

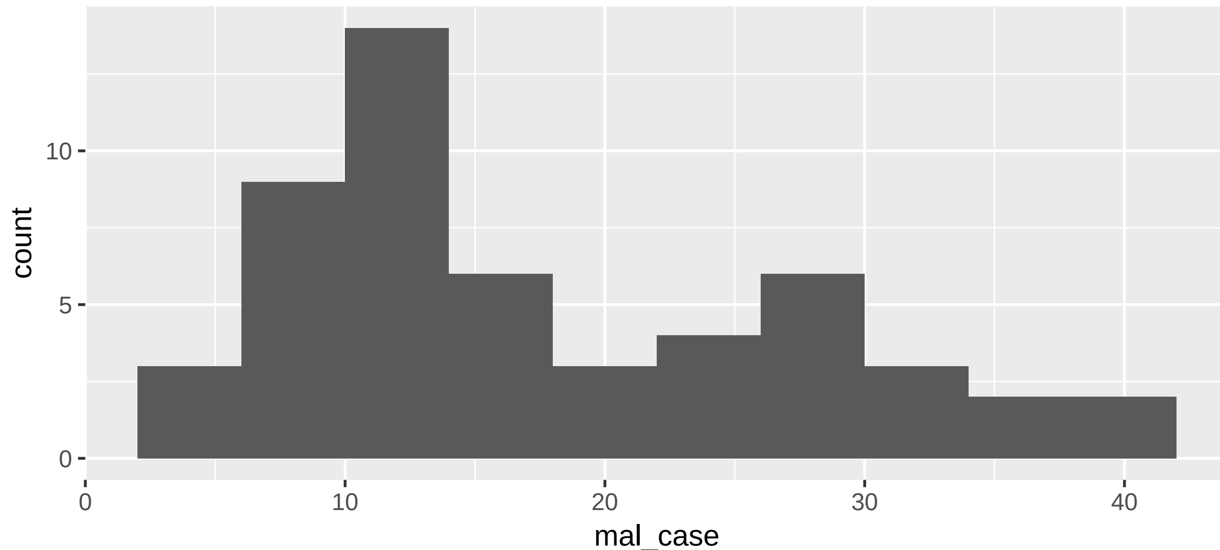
```
ggplot(data = mecha) +
  geom_point(mapping = aes(x = mal_case, y = test_pf_tot, color = iso_year))
```



3.7.2 Histograms

A histogram is a graphical representation of the distribution of numerican data. It usually has boxes whose area is proportional to the frequency of a class of values. In this example, we will plot the frequency of malaria count data in the fogera dataset.

```r
ggplot(fogera) +
  geom_histogram(aes(x = mal_case), bins = 10)
```

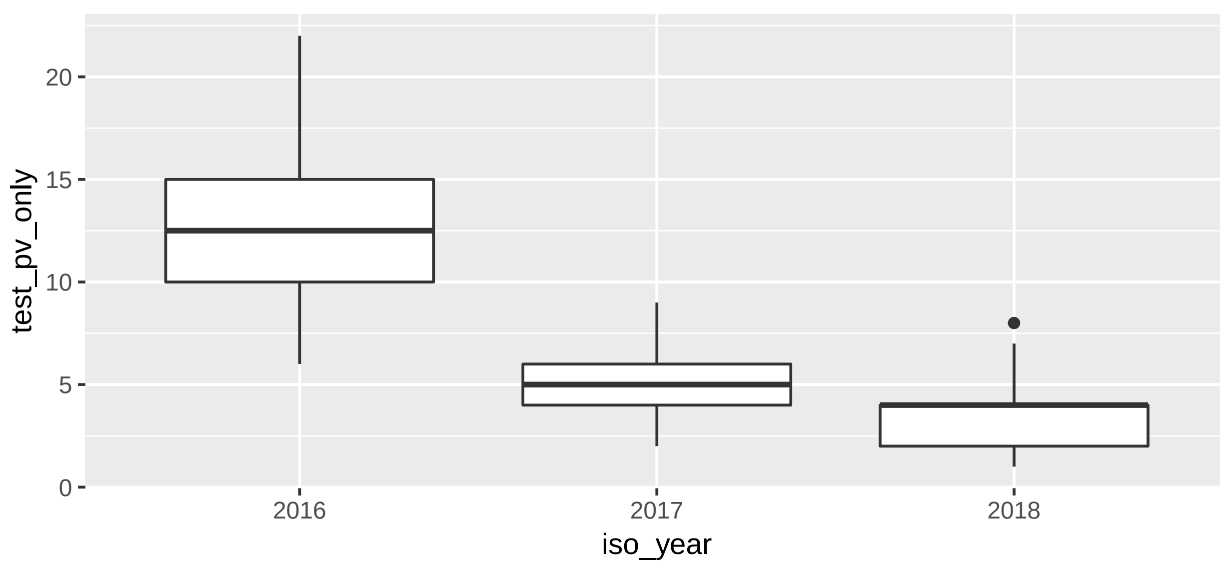


Each column represents a discrete "bin", or range of values. The height of the rectangles represents the number of values in the bin, i.e. the number of weeks where the number of malaria cases fell within the values covered by the bin.

3.7.3 Boxplots

Like histograms, boxplots also visualize the distribution of data.

```r
ggplot(mecha) +
  geom_boxplot(aes(iso_year, test_pv_only))
```



In ggplot, the horizontal line represents the median value, the box represents the inter quartile range (IQR), i.e. the 25th through 75th percentiles. The upper whisker extends from the hinge to the largest value no further than `1.5 * IQR` from the hinge (where IQR is the inter-quartile range, or distance between the first and third quartiles). The lower whisker extends from the hinge to the smallest value at most `1.5 * IQR` of the hinge. Data beyond the end of the whiskers are called "outlying" points and are plotted individually.

4 Mapping in ggplot

We can also use ggplot to make maps. For this demonstration we focus on creating coropleth maps from a shapefile. We first use the `readOGR()` function to read the layer we want to display from a given directory (data source name) where the shapefile is stored.

```
library(ggplot2)
library(maptools)
```

```
## Warning: package 'maptools' was built under R version 3.5.3
```

```
## Loading required package: sp
```

```
## Checking rgeos availability: TRUE
```

```
library(dplyr)
library(rgdal)
```

```
## rgdal: version: 1.3-6, (SVN revision 773)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.2.3, released 2017/11/20
##  Path to GDAL shared files: C:/Users/neko0001/Documents/R/win-library/3.5/rgdal/gdal
##  GDAL binary built with GEOS: TRUE
##  Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
##  Path to PROJ.4 shared files: C:/Users/neko0001/Documents/R/win-library/3.5/rgdal/proj
##  Linking to sp version: 1.3-1
```

```
shp <- readOGR(dsn = "./shapefile", layer = "woreda_epidemia_20170207", stringsAsFactors = F)
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\neko0001\Work\R_dev\r_workshop_epidemia\shapefile", layer: "woreda_epidemia_201702
## with 141 features
## It has 6 fields
```

```
summary(shp)
```

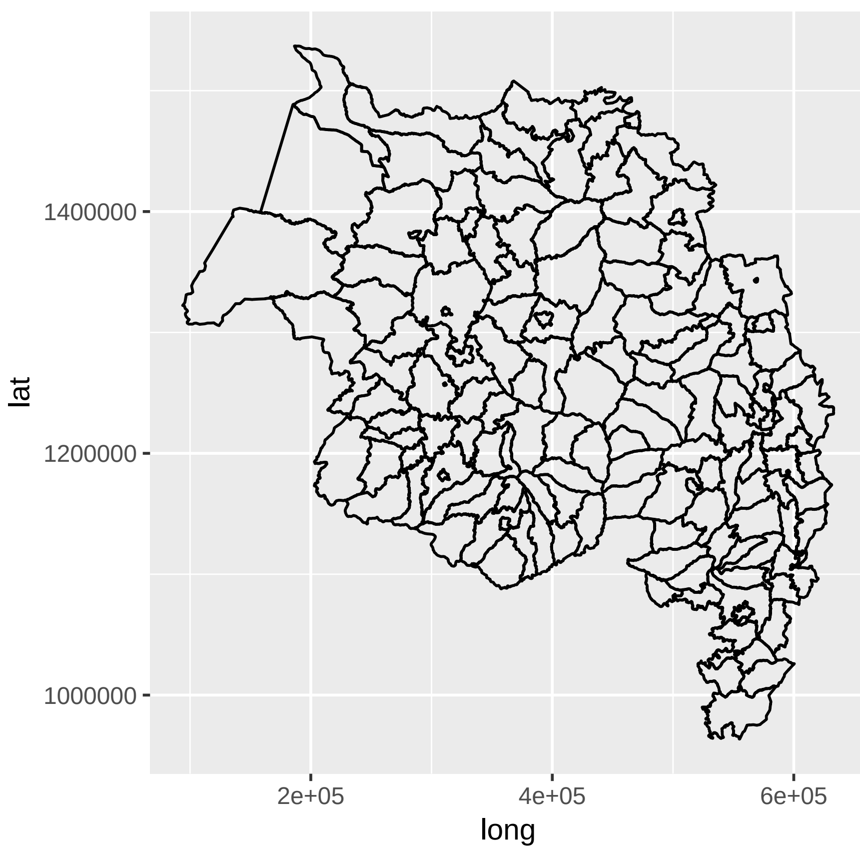
```
## Object of class SpatialPolygonsDataFrame
## Coordinates:
##         min       max
## x  93853.35  633095.9
## y 963619.09 1536962.8
## Is projected: TRUE
## proj4string :
## [+proj=utm +zone=37 +datum=WGS84 +units=m +no_defs +ellps=WGS84
## +towgs84=0,0,0]
## Data attributes:
##      zone              woreda               wid          pilot
##  Length:141         Length:141         Min.   :  0   Min.   :0.0000
##  Class :character   Class :character   1st Qu.: 35   1st Qu.:0.0000
##  Mode  :character   Mode  :character   Median : 70   Median :0.0000
##                                        Mean   : 70   Mean   :0.4539
##                                        3rd Qu.:105   3rd Qu.:1.0000
##                                        Max.   :140   Max.   :2.0000
##    Shape_Leng       Shape_Area
```

```
##  Min.   :  8257   Min.   :3.752e+06
##  1st Qu.:152113   1st Qu.:6.210e+08
##  Median :189111   Median :9.329e+08
##  Mean   :200691   Mean   :1.098e+09
##  3rd Qu.:245075   3rd Qu.:1.303e+09
##  Max.   :489267   Max.   :7.707e+09
```

To generate a map of Woredas using `ggplot()`, we use the `geom_polygon()` function. The `coord_equal()` function forces the map to have the same scale for x and y coordinates.

```r
ggplot()+
  geom_polygon(data=shp,
          aes(x=long, y=lat, group=group), colour="black", fill=NA)+
  coord_equal()
```

```
## Regions defined for each Polygons
```



To map data with ggplot2, we need to take an additional step and convert the imported sp object to a data frame that contains all of the geospatial data for the polygon locations along with their associated attributes in a single table. This step is necessary because ggplot2, along with other `tidyverse` packages such as dplyr and tidyr, all use data frames as their core data objects. A data frame is actually a rather inefficient format for strong polygon data, but this approach allows us to access to a wide range of powerful mapping functions via the `ggplot()` function.

```r
shp_data <- shp@data
shp_f <- fortify(shp, region = "woreda")
head(shp_f)
```

```
##       long      lat order  hole piece        id       group
```

```
## 1 463548.5 1469498     1 FALSE     1 Abargelie Abargelie.1
## 2 463992.0 1469433     2 FALSE     1 Abargelie Abargelie.1
## 3 464441.8 1469475     3 FALSE     1 Abargelie Abargelie.1
## 4 464579.3 1469533     4 FALSE     1 Abargelie Abargelie.1
## 5 464600.5 1469522     5 FALSE     1 Abargelie Abargelie.1
## 6 464574.0 1469586     6 FALSE     1 Abargelie Abargelie.1
```

```
shp_f <- left_join(shp_f, shp_data, by=c("id" = "woreda"))
```

Now as we have converted our shapefile into a data frame containing the spatial information of our Woredas, we can add our epidemiological data to it. We will use `read_csv()` to load the epidemiological data. Then we use `group_by()` and `summarize()` to summarize cases by Woreda. Finally we use `left_join()` to add our epidemiological data (for three years 2016 - 2018) to the data frame of our shapefile. Tomorrow we will learn even more ways of summarizing data and of joining multiple dataset.

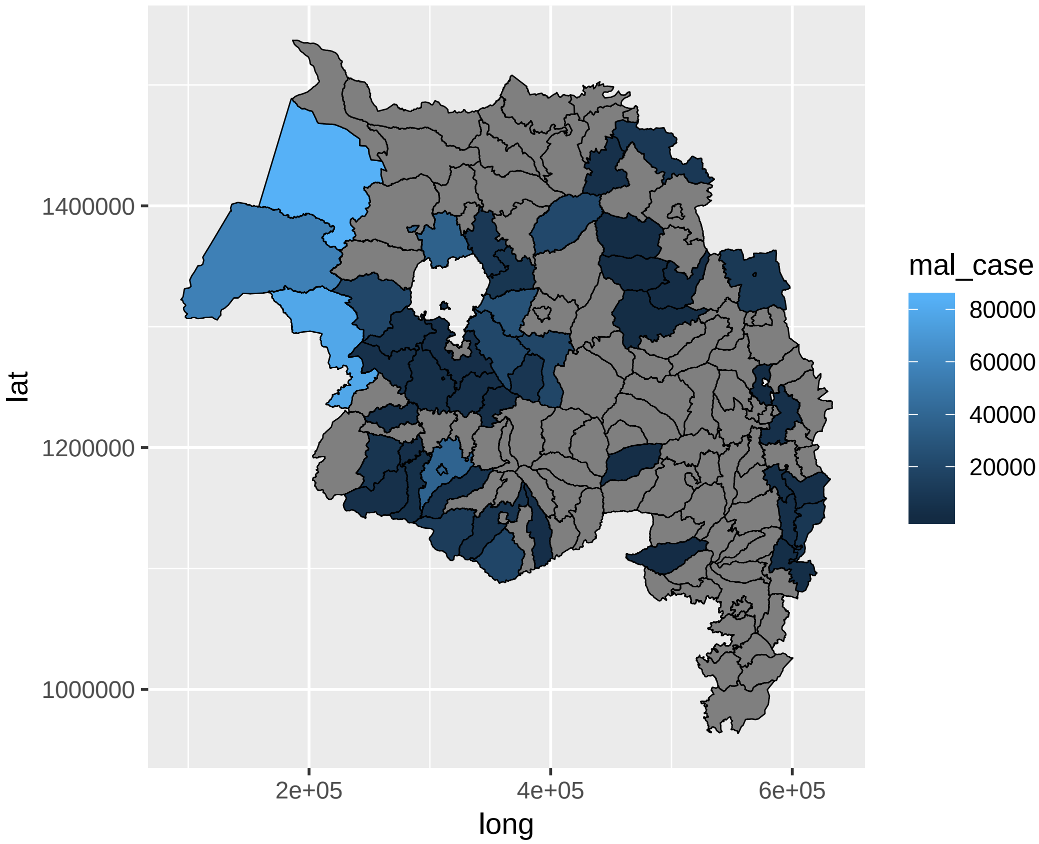
```
library(readr)
epi_dat <- read_csv("data_2016_2018.csv")
```

```
## 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()
## )
```

```
by_woreda <- group_by(epi_dat, woreda_name)
summarized_cases <- summarize(by_woreda,
                              mal_case = sum(mal_case, na.rm = TRUE))
shp_f <- left_join(shp_f, summarized_cases, by=c("id" = "woreda_name"))
```

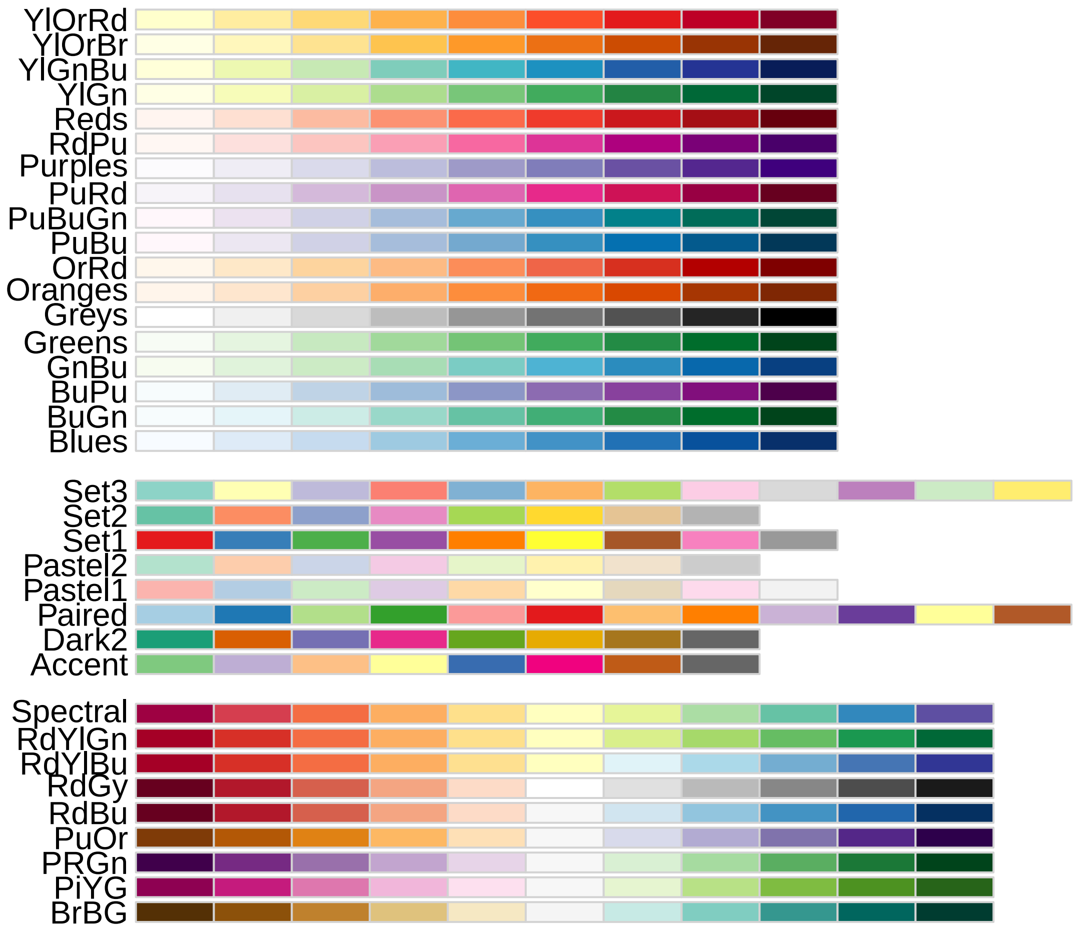
Key arguments of `geom_polygon()` include spatial coordinates, the variable to be used to fill in the polygons, and the color and size of the polygon borders. By default, `ggplot()` will use a default color ramp (light to dark blue) to represent the number of cases in each polygon.

```
ggplot(data = shp_f) +
  geom_polygon(aes(x = long, y = lat, group = group, fill = mal_case),
               color = "black", size = 0.25) +
  coord_equal()
```



As with any other sort of ggplot, we can easily change the color displayed. Here we can work with different color palettes The RColorBrewer package provides a variety of color palettes that have been determined to be effective for choropleth mapping. The following functions can be used to explore the different palettes that are available. More information is also available at http://colorbrewer2.org.

```
library(RColorBrewer)
display.brewer.all()
```



We can do a few additional things to improve the appearance of the map. The `scale_fill_distiller()` function can be used to specify a different color pallette. In this case, we use the "YlGn" palette from the RColorBrewer package and call the `pretty_breaks()` function from the scales package to automatically select a set of breaks for the legend. The `theme_void()` function gets rid of the gray background grid and the axis scales and labels to produce a nicer looking map layout. Finally, a title is added using the labs() function.

```
library(ggmap)  # Additional functions for mapping with ggplot()
```

```
## Warning: package 'ggmap' was built under R version 3.5.3
```

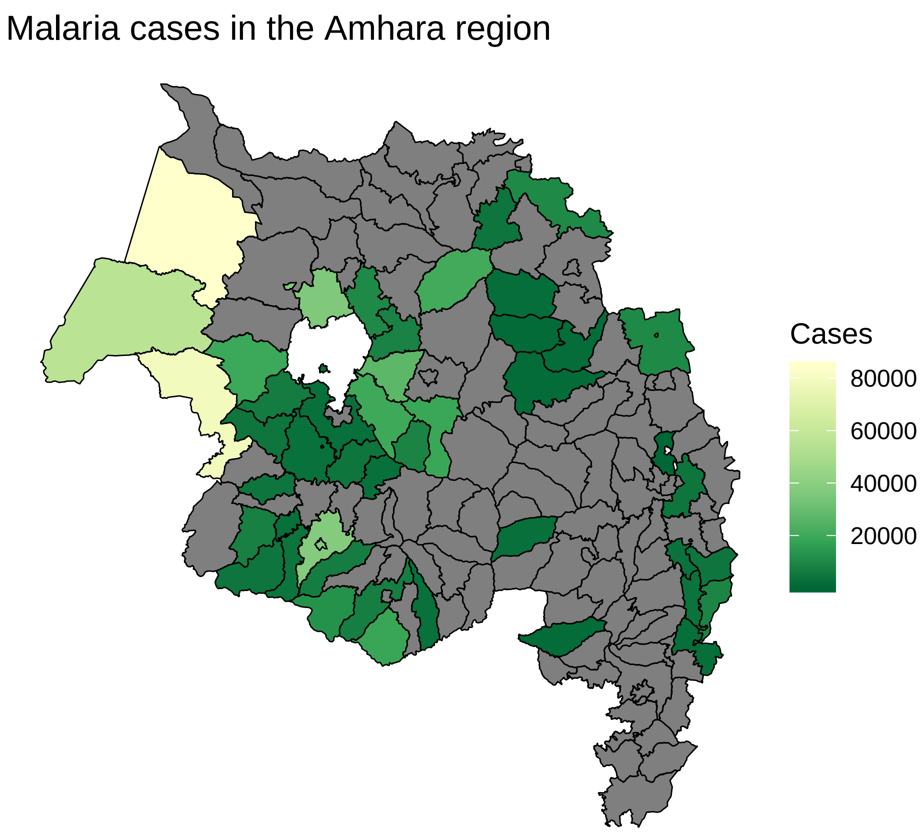
```
## Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.
```

```
## Please cite ggmap if you use it! See citation("ggmap") for details.
```

```r
library(scales) # Additional graphics functions
```

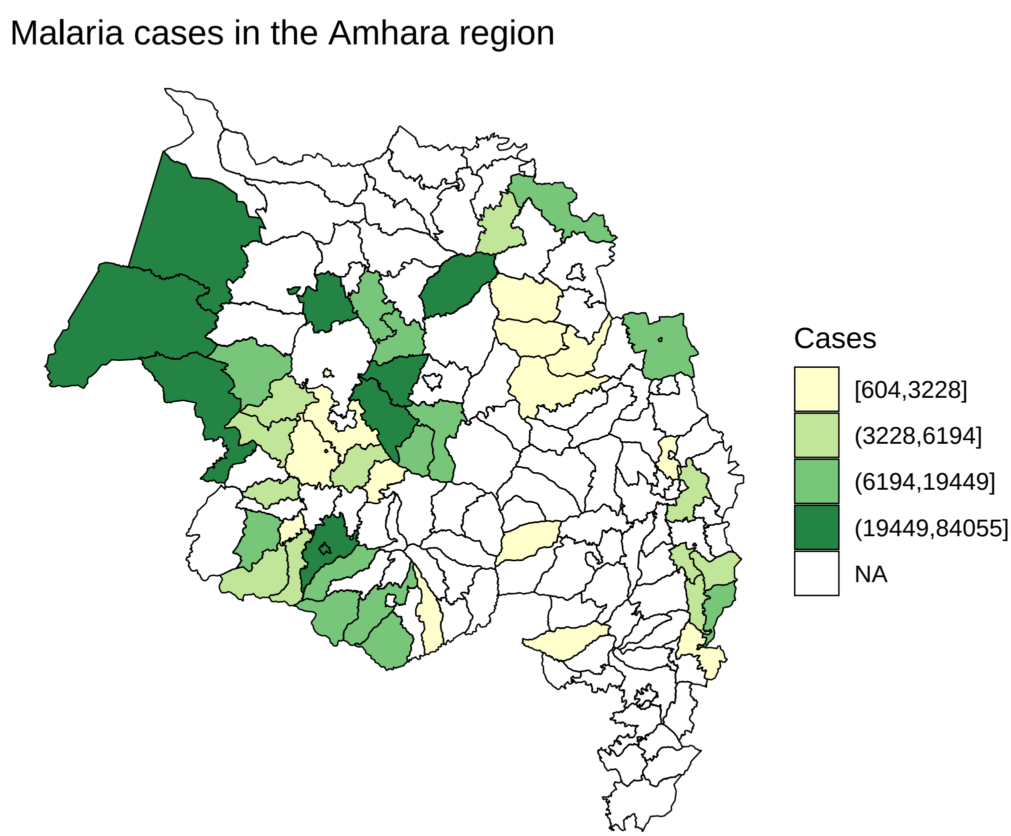
```
##
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':
##
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor
```

```r
ggplot(data = shp_f) +
  geom_polygon(aes(x = long, y = lat, group = group, fill = mal_case),
               color = "black", size = 0.25) +
  scale_fill_distiller(name="Cases", palette = "YlGn", breaks = pretty_breaks()) +
  coord_equal() +
  theme_void()+
  labs(title="Malaria cases in the Amhara region")
```



Rather than using continuous scales for color and size, it is usually recommended to aggregate the data into a relatively small number of classes (typically 3-6). To accomplish this, we need to add a couple of new classified variables using `mutate()`. The `cut()` function is used to split the incidence rate into four quantiles, and to split population into three classes using manually-selected breaks. Note that different `scale()` functions are now needed to handle the discrete variables instead of continuous variables.

```
shp_f <- mutate(shp_f,
                mal_case_discrete = cut(mal_case,
                                        breaks = quantile(mal_case, na.rm=T),
                                        include.lowest = T, dig.lab=10)
                )


ggplot(data = shp_f) +
  geom_polygon(aes(x = long, y = lat, group = group, fill = mal_case_discrete),
               color = "black", size = 0.25) +
  scale_fill_brewer(name="Cases", palette = "YlGn")+
  coord_equal() +
  theme_void()+
  labs(title="Malaria cases in the Amhara region")
```



5 Day 2 exercises

1. Read the data file `data_2016_2018.csv`.
2. Create a time series plot of confirmed P. vivax malaria incidence `pv_inc`. Show each year as a different color.
3. Use the `subset()` function to create a new data frame that contains data for two woredas of your choice.
4. Create a time series plot of confirmed P. vivax malaria incidence `pv_inc`. Show each year as a different color. Facet by woreda.
5. Facet by woreda *and* year using `facet_grid()`. You will need to use `help("facet_grid")` to learn

how to specify the faceting in this new function.

6. Create a scatterplot showing the relationship between confirmed P. falciparum malaria incidence and confirmed P. vivax malaria incidence.
7. Create a map showing the total cases of Malaria caused by P. falciparum for each Woreda.