Session 2: Data visualization with ggplot2

Natalie Nelson, PhD, Biological & Agricultural Engineering, NCSU

01/26/2018

1 Data

The dataset analyzed in this exercise was curated by the Food and Agricultural Organization (FAO) of the United Nations through the FAOSTAT program. FAOSTAT maintains food and agriculture data for over 245 countries and territories spanning from 1961 until the present. For this analysis, we will be evaluating sweet potato yield (hectogram/hectare), area harvested (hectares), and production quantity (tons) globally. The data are yearly and include observations from 1961 - 2016 for countries and territories where data were available on these measures. For free access to these and other food and agricultural data, visit the FAOSTAT webpage.

2 Questions

In today's problem session, we will visualize data to investigate the following questions:

- How has sweet potato yield, area harvested, and total production varied globally since 1961? Describe at the continental scale. Support your findings with a minimum of 4 plot types.
- For countries in one continent of your choosing, how have these measures (yield, area, production) varied over time? What is the relationship between yield, area harvested, and production in these countries? Support your findings with a minimum of 3 plot types.
- What relationships exist between sweet potato yield, area harvested, and production quantity in the United States? Have these relationships changed over time? Support your findings with what you consider to be **the most effective** plot types for presenting these particular trends and relationships. Explain **why** you think these plot types best communicate the results (think about the "key principles of data visualization").

For each plot you present, provide its practical interpretation. What is the data visualization saying? What are the real-world implications of the patterns you've uncovered in the data?

Figure 1: Photos from the USDA

3 Analysis

Setting up your R script

Let's setup our script. This includes clearing the workspace, loading packages, and setting the working directory. The packages we'll be using today include `ggplot2` and `plyr`. You should already have `plyr` installed, but you need to install `ggplot2`. This can be done with the `install.packages("name-of-package")` function.

```
# Clear workspace
rm(list=ls(all=TRUE))

# Load packages
library(ggplot2)
library(plyr)

# Set working directory
#setwd("your-path-name") # Note: this line should not be commented out in your code!
```

Read in your data

The data are saved across several .rds files:

- "World.rds" - includes data at the *continental* scale. Continental regions include the Americas, Europe, Africa, Asia, and Oceania (5 total).
- One file for each continental region (i.e., "Americas.rds", "Europe.rds", "Asia.rds", "Africa.rds", "Oceania.rds") - includes data at the *national* scale.

To load the .rds files to our workspace, use the `readRDS()` function. The data were saved to the .rds files as dataframes, so the data will be in dataframe format once we upload them. The first question presented in section 2 asks you to evaluate yield, area harvested, and production quantity across the continental scale, so we will explore the "World.rds" data and assign the dataframe to an object named "w".

```
w <- readRDS("World.rds")
```

Check the packaging and the top and bottom of your data

You know the drill!

```
str(w)
```

```
## 'data.frame':    280 obs. of  5 variables:
##  $ Region    : Factor w/ 5 levels "Africa","Americas",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Year      : int  1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 ...
##  $ Area      : int  621236 676583 663998 641371 643481 599399 587569 608318 782055 954153 ...
##  $ Production: int  3280871 3572524 3419916 3241336 3181104 3226807 3570573 3617209 3778912 4578892
##  $ Yield     : int  52812 52802 51505 50538 49436 53834 60769 59462 48320 47989 ...
```

```
dim(w)
```

```
## [1] 280   5
```

```
head(w)
```

```
##   Region Year   Area Production Yield
## 1 Africa 1961 621236    3280871 52812
## 2 Africa 1962 676583    3572524 52802
## 3 Africa 1963 663998    3419916 51505
## 4 Africa 1964 641371    3241336 50538
## 5 Africa 1965 643481    3181104 49436
## 6 Africa 1966 599399    3226807 53834
```

```r
tail(w)
```

```
##      Region Year   Area Production Yield
## 275 Oceania 2011 140498     813374 57892
## 276 Oceania 2012 144214     831640 57667
## 277 Oceania 2013 149174     857390 57476
## 278 Oceania 2014 150573     882839 58632
## 279 Oceania 2015 156235     897679 57457
## 280 Oceania 2016 157774     906101 57430
```

Everything looks correct, but let's also verify that our *Regions* variable includes all of the continental regions we expect to see.

```r
unique(w$Regions)
```

```
## NULL
```

We should also validate our data with one external source to make sure the numbers seem correct. For this particular case, let's just do a "sanity check" to make sure the numbers seem correct (e.g., the dates span the correct range, there aren't any negative values). We can do this by running a quick `summary()` of w.

```r
summary(w)
```

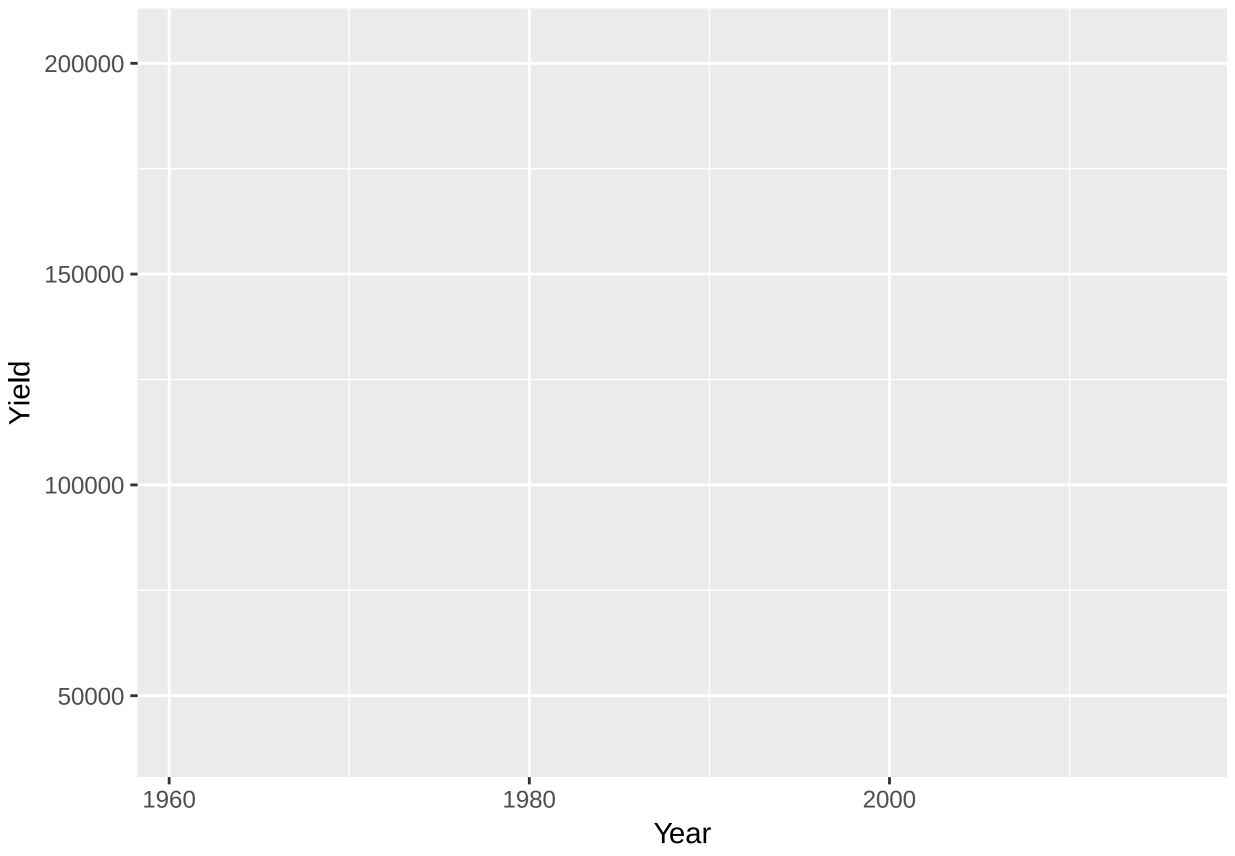
```
##       Region        Year           Area             Production       
##  Africa  :56   Min.   :1961   Min.   :    1816   Min.   :    35039  
##  Americas:56   1st Qu.:1975   1st Qu.:   99412   1st Qu.:   498202  
##  Asia    :56   Median :1988   Median :  323348   Median :  2838598  
##  Europe  :56   Mean   :1988   Mean   : 2016802   Mean   : 24668134  
##  Oceania :56   3rd Qu.:2002   3rd Qu.: 2877283   3rd Qu.: 12306944  
##                Max.   :2016   Max.   :12427372   Max.   :144012185  
##      Yield       
##  Min.   : 38982  
##  1st Qu.: 51481  
##  Median : 83549  
##  Mean   : 94005  
##  3rd Qu.:120943  
##  Max.   :204654
```

Here we can see that each region has 56 observations, which corresponds to yearly values for 1961-2016 (56 years total). The minimum *Year* is 1961 and the maximum is 2016, which is correct. Area, production, and yield all have positive values. Everything appears to checkout. Note that the units of each of the three measures are included in the initial data summary at the top of this document.
Which of these variables are continuous, and which are discrete?

Data viz with ggplot: line and point plots

Let's start by creating a simple line plot of *Yield* vs. *Year* for each of our continental regions. First, let's call our base plot with `ggplot()`. When first calling `ggplot()`, we include our dataframe as the first argument, and then include the details of our plot *aesthetics*, which is done through `aes()`. In `aes()`, we will specify which variable should be assigned to x, and which variable should be assigned to y.

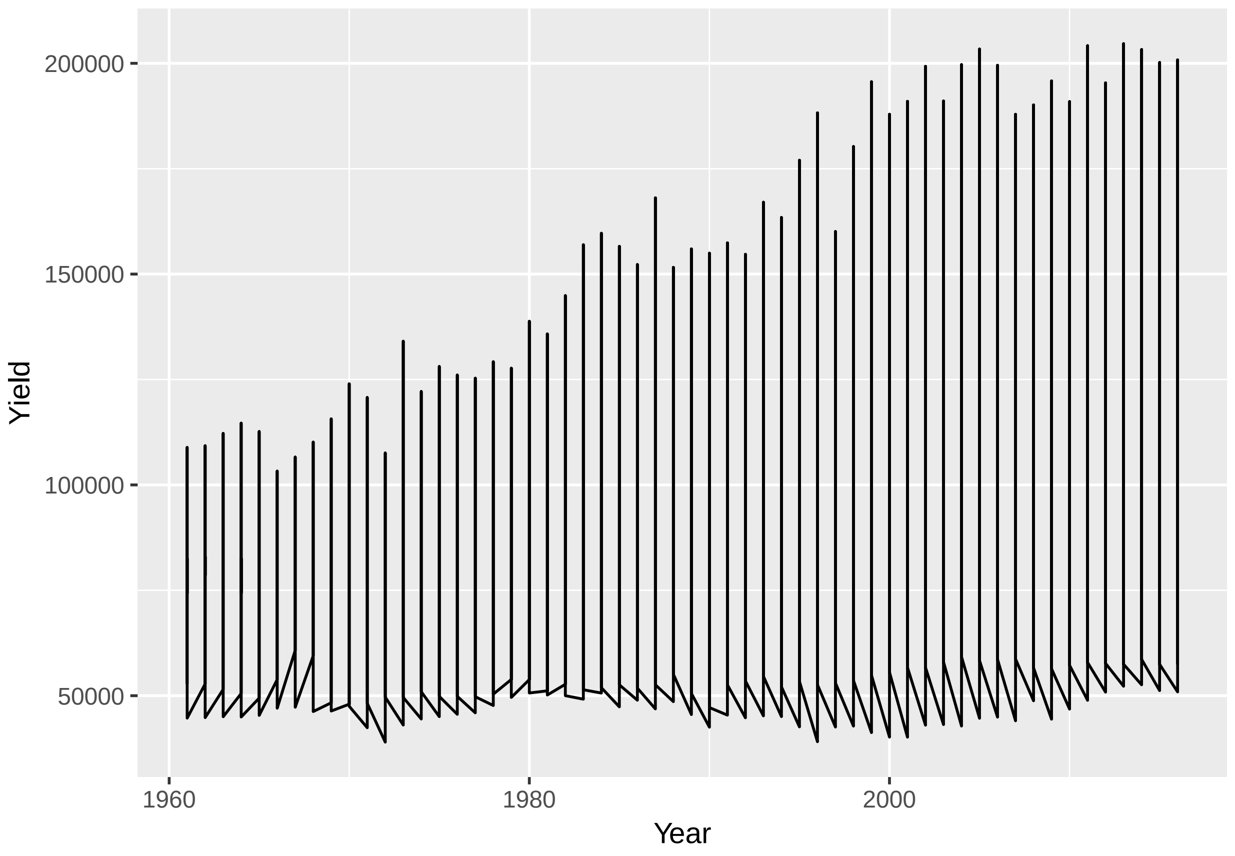
```r
ggplot(w, aes(x = Year, y = Yield))
```



The axes are automatically set to the limits of our *Year* and *Yield* data, and labelled correspondingly.

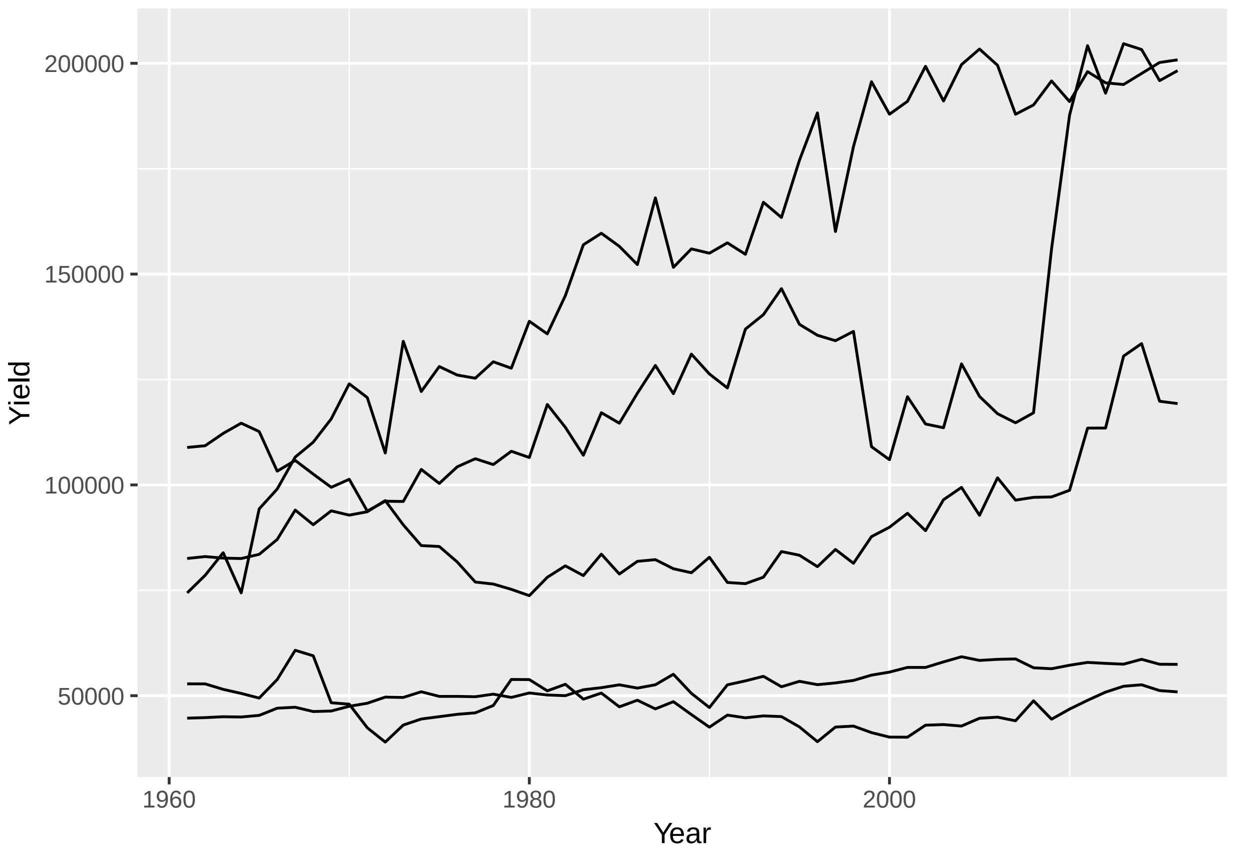
`ggplot` works in layers - we keep layering additional functions on our original base plot until our graphic includes all of the information and features we want to highlight. For example, let's add lines to our base plot. If we look at our Data Visualization Cheat Sheet, we can see under "Two Variables" -> "Continuous Function" that we use the `geom_line()` function to add lines to our plot. We don't need to add any arguments to the `geom_line()` function because it automatically assumes that the arguments are the same as those included in `ggplot()`.

```r
ggplot(w, aes(x = Year, y = Yield))+
  geom_line()
```



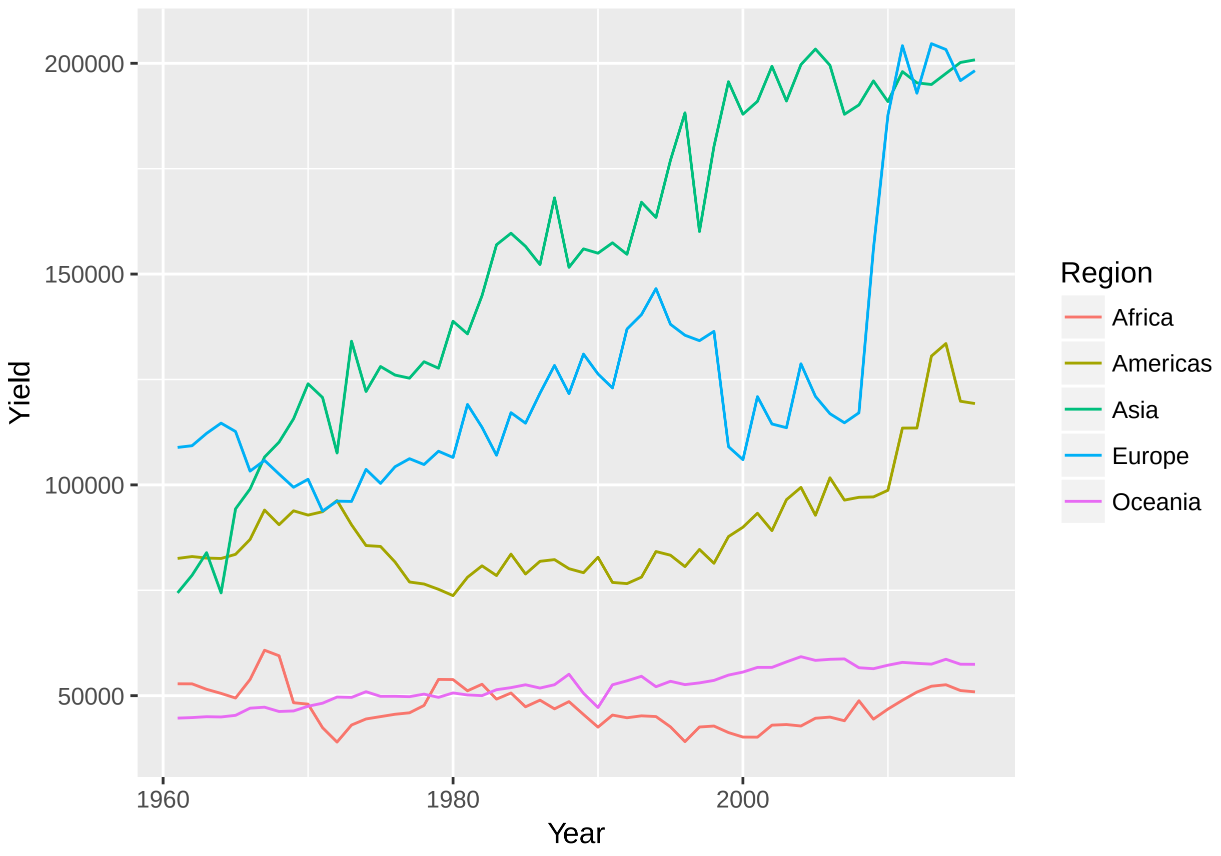
The output looks strange. This is because the function doesn't know that we have multiple groups (i.e., regions) that we want to plot separately. Since we didn't specify, ggplot assumed that all of the data in the *Yield* column of our dataframe should be plotted as one data series. To specify that we want the data to be plotted by each region in the *Region* column, we can include a `group =` argument in the `aes()` of our base `ggplot()` function.

```
ggplot(w, aes(x = Year, y = Yield, group = Region))+
  geom_line()
```



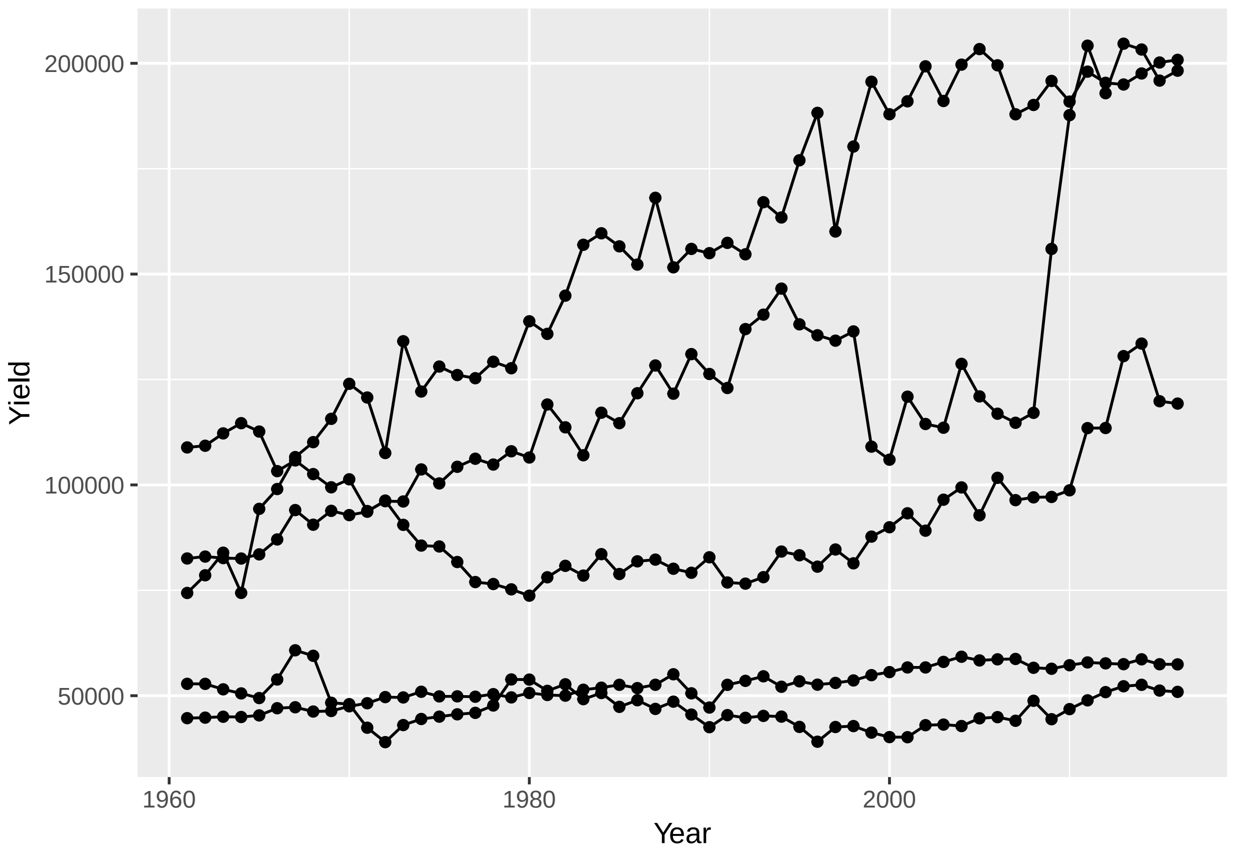
Now we have lines for each region, but we can't distinguish them because they all look identical. We can distinguish them in a few different ways. One option is to change the color. This would be done by adding `color = Region` to either the `aes()` in `ggplot()`, or we can add it within `aes()` of `geom_line()`.

```r
ggplot(w, aes(x = Year, y = Yield, group = Region))+
  geom_line(aes(color = Region))
```



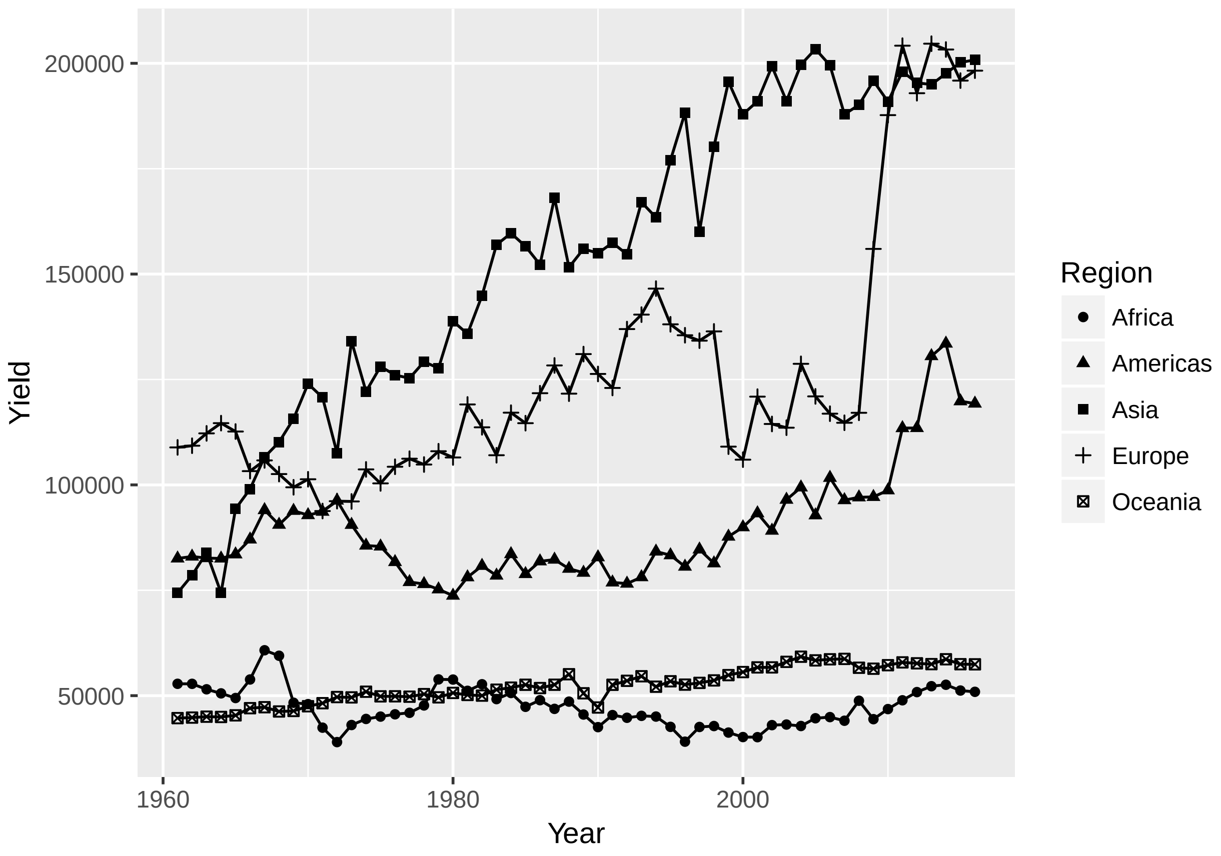
Note that a legend has now automatically been added to our plot. Instead of color, we also could have coded our regions by symbol. To do this, we would first need to add points to our plot. We can see from our Cheat Sheet under “Two Variables” -> “Continuous X, Continuous Y” that the function used for adding points is `geom_point()`.

```
ggplot(w, aes(x = Year, y = Yield, group = Region))+
  geom_line()+
  geom_point()
```



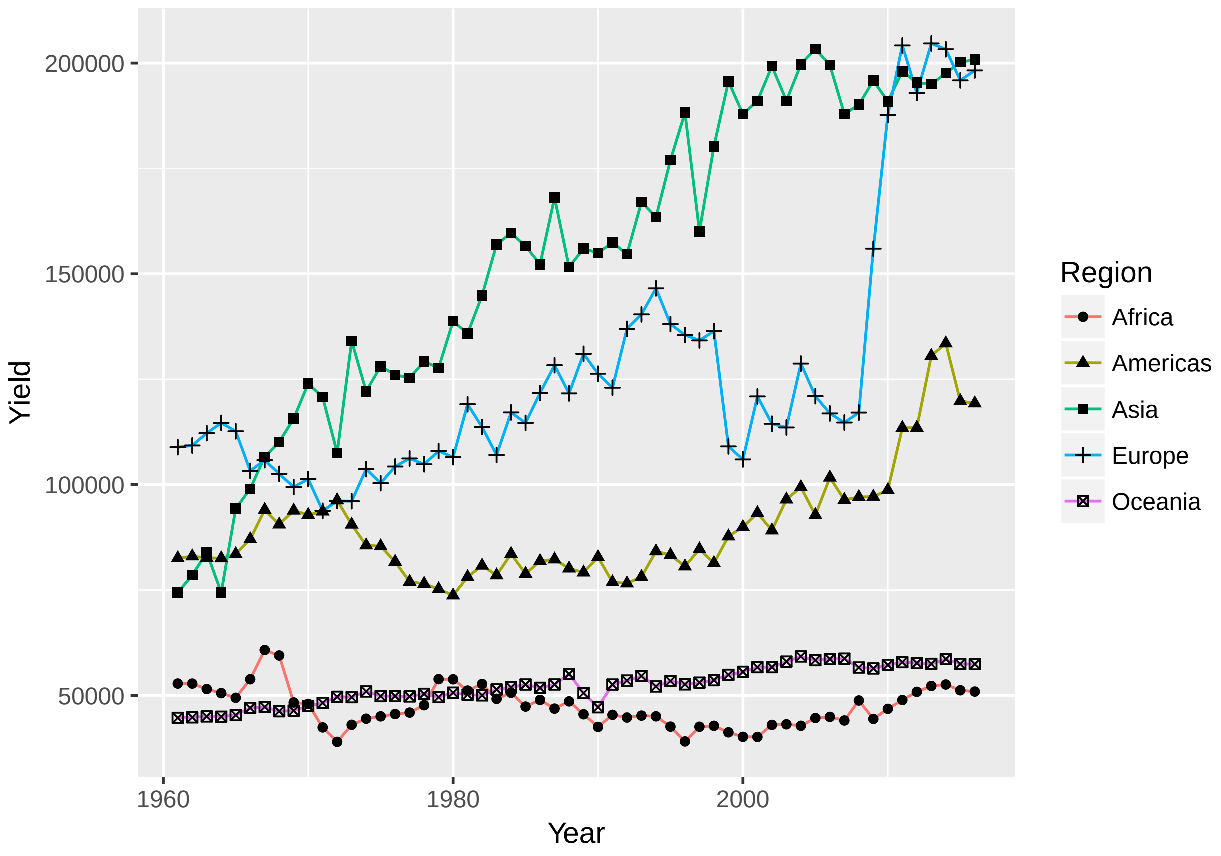
We now need to specify within `geom_point()` that the symbol of the points should be unique for each region. Just as we changed the color of our lines, we can change the symbol type by specifying `shape =` within `aes()` of `geom_point()`.

```
ggplot(w, aes(x = Year, y = Yield, group = Region))+
  geom_line()+
  geom_point(aes(shape = Region))
```



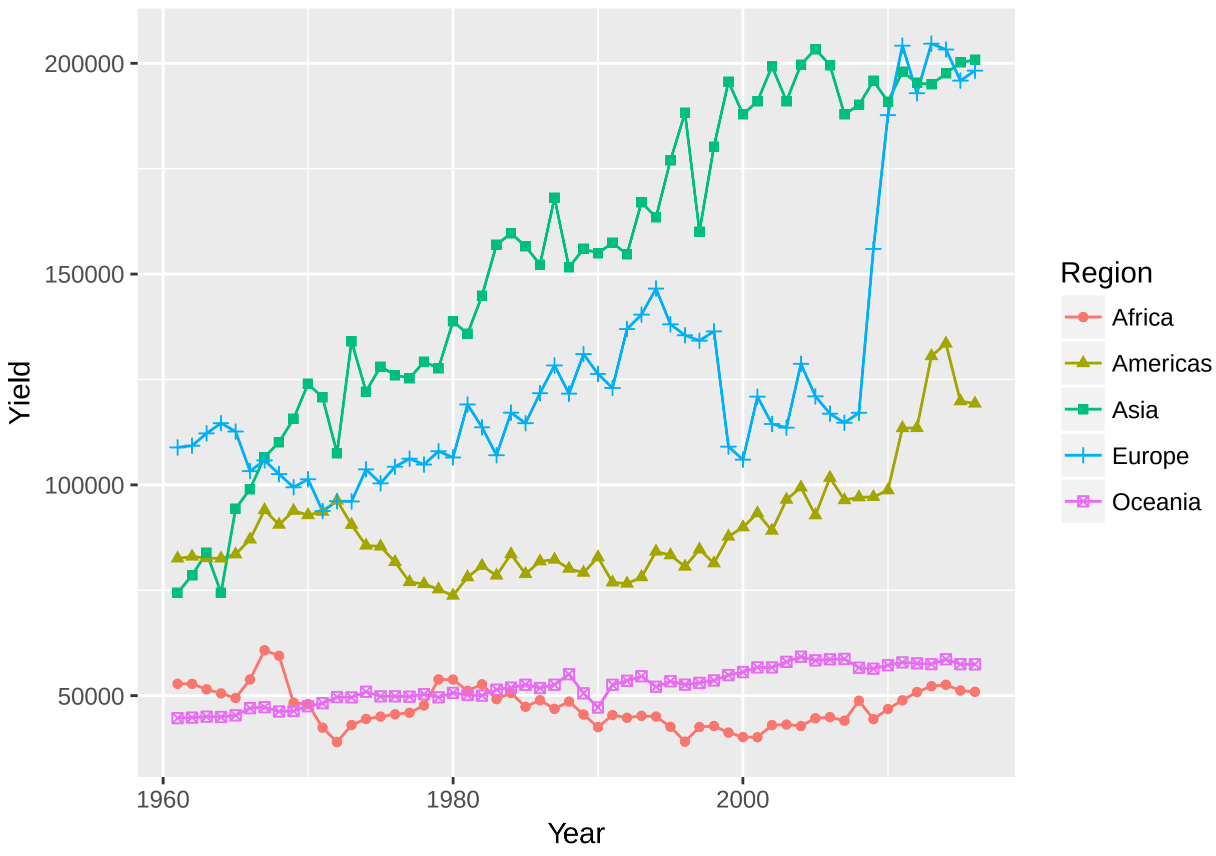
We can merge these codes to have both color and symbols vary by region. This could be a good strategy to employ if we want to ensure that the graphic can be interpreted in color or greyscale.

```
ggplot(w, aes(x = Year, y = Yield, group = Region))+
  geom_line(aes(color = Region))+
  geom_point(aes(shape = Region))
```



Note that, with this code, color was returned to the lines, but the symbols remained in black. We can color the symbols using one of two approaches: include color in the original `ggplot(aes())`, or add the color argument to `geom_point(aes())`. If we add the color argument to the original `ggplot(aes())`, we no longer need to include it in the `geom_line()` function.

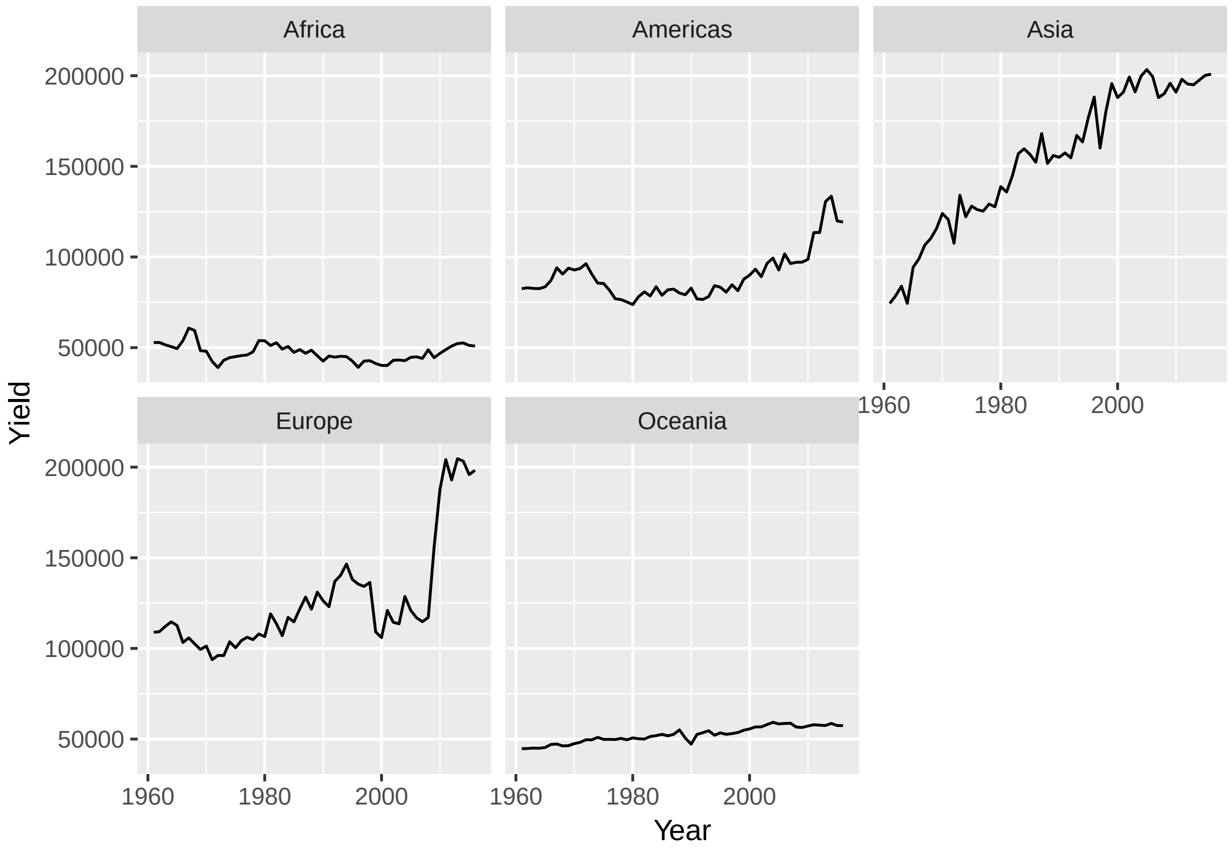
```
ggplot(w, aes(x = Year, y = Yield, group = Region, color = Region))+
  geom_line()+
  geom_point(aes(shape = Region))
```



Multi-panel plots

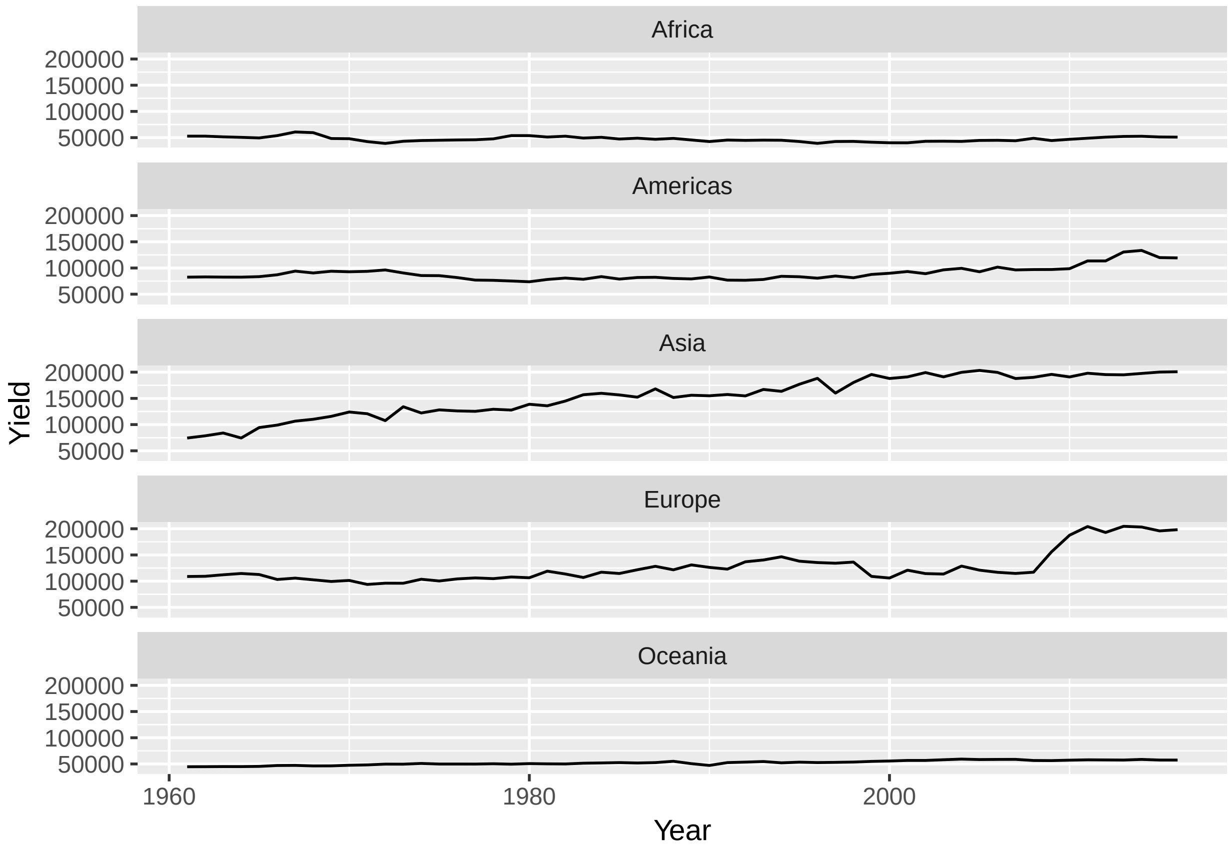
Rather than include all of the lines in one plot, we can divide the lines across region-specific panels. This is done using either the `facet_grid()` or `facet_wrap()`. To see how these two functions divide data across different panels, look at your Cheat Sheet under "Faceting" (on the back of the Cheat Sheet). In the code below, we've applied `facet_wrap()` to divide our plot.

```
ggplot(w, aes(x = Year, y = Yield))+
  geom_line()+
  facet_wrap(~Region)
```



Note that the color and symbols were removed from this plot. This is because we no longer need color or symbols to differentiate the lines since each line has its own panel. Remember one of the key principles of data visualization: simplify!
To allow for easy one-to-one comparisons, we can stack the panels one on top of the other. This is done by adding `ncol = 1` as an argument in the `facet_wrap()` function, where "ncol" stands for "number of columns" (there is also a nrow option).

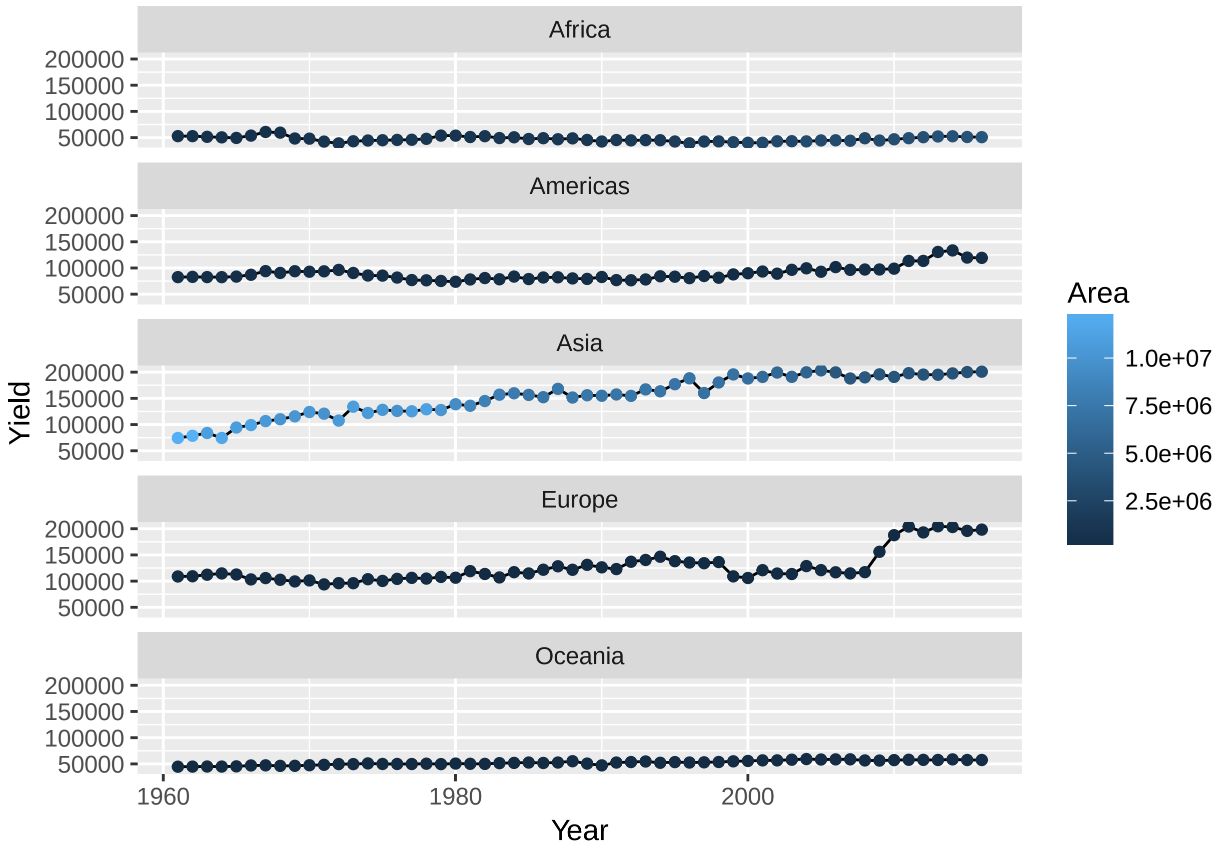
```r
ggplot(w, aes(x = Year, y = Yield))+
  geom_line()+
  facet_wrap(~Region, ncol = 1)
```



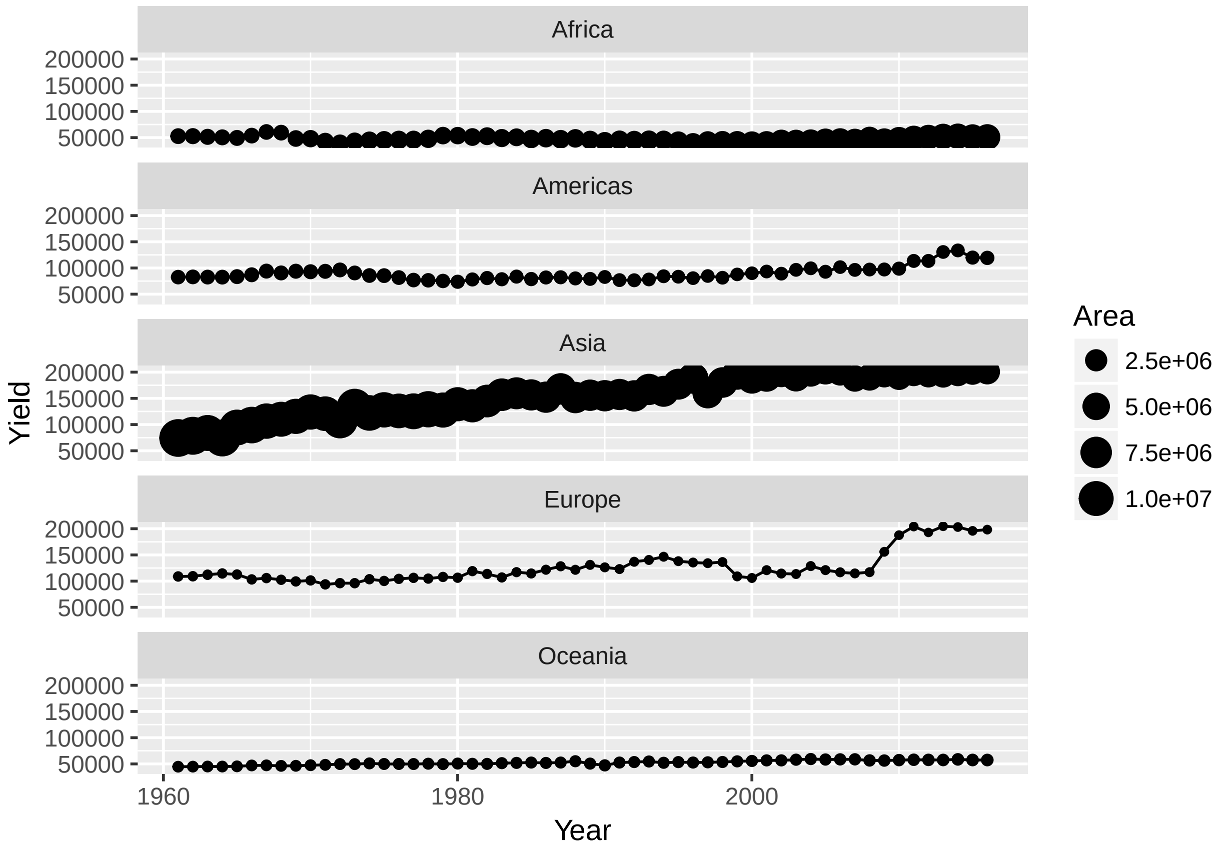
Layering additional variables

Let's say we want to look at how yields change over time, but we also want to see how the harvested area has changed along with yield. One option would be to include two separate axes, but another approach would be to scale the colors or sizes of the *Yield* lines or points based on the corresponding values of *Area*. Generally speaking, this approach works a bit better with points than lines (try for yourself!). To do this with color, we would add `aes(color = Area)`; to do this with size, we would add `aes(size = Area)`.

```r
ggplot(w, aes(x = Year, y = Yield))+
  geom_line()+
  geom_point(aes(color = Area))+
  facet_wrap(~Region, ncol = 1)
```

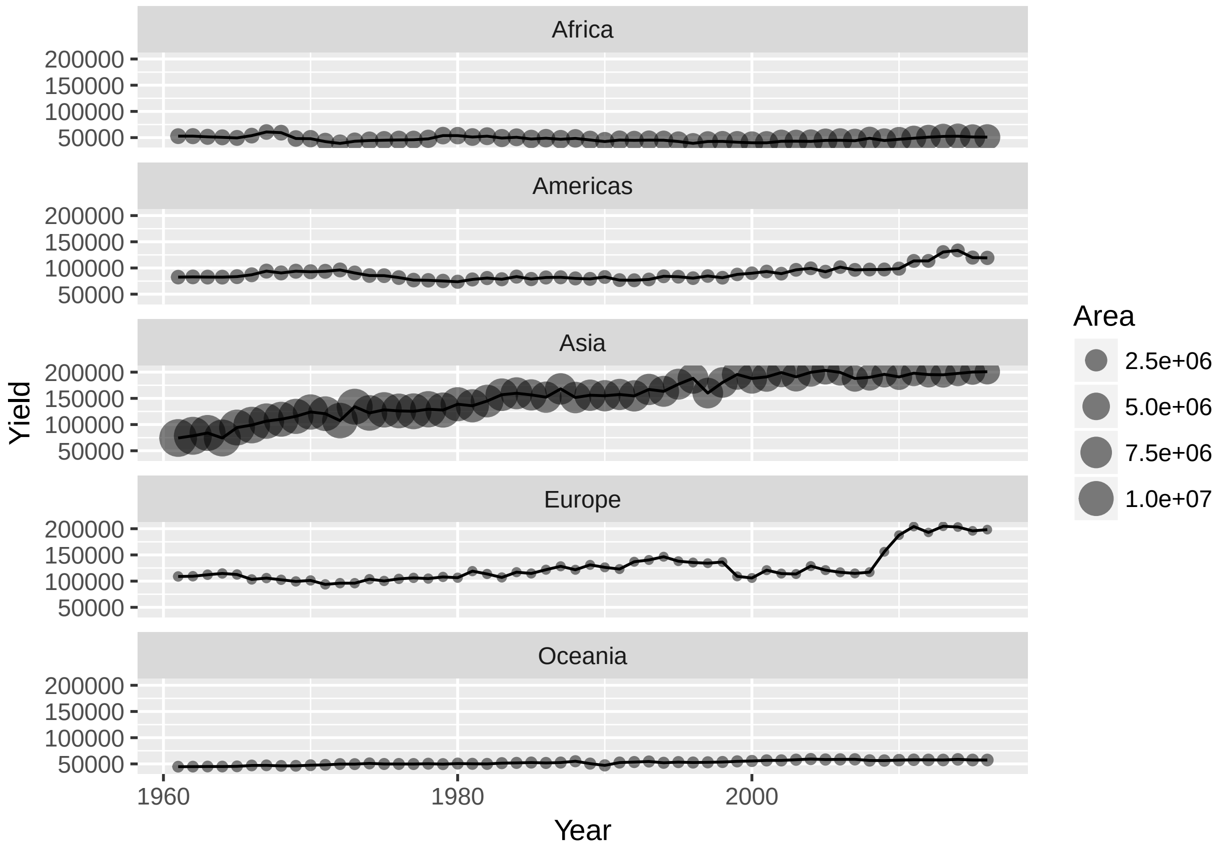


```r
ggplot(w, aes(x = Year, y = Yield))+
  geom_line()+
  geom_point(aes(size = Area))+
  facet_wrap(~Region, ncol = 1)
```



In the above plot, the points bleed together in a few of the panels. This can be modified if we change the transparency of the points, which is done through `alpha`. The value you assign to `alpha` corresponds to the % of transparency. For example, `alpha = 0.5` corresponds to 50% transparency.

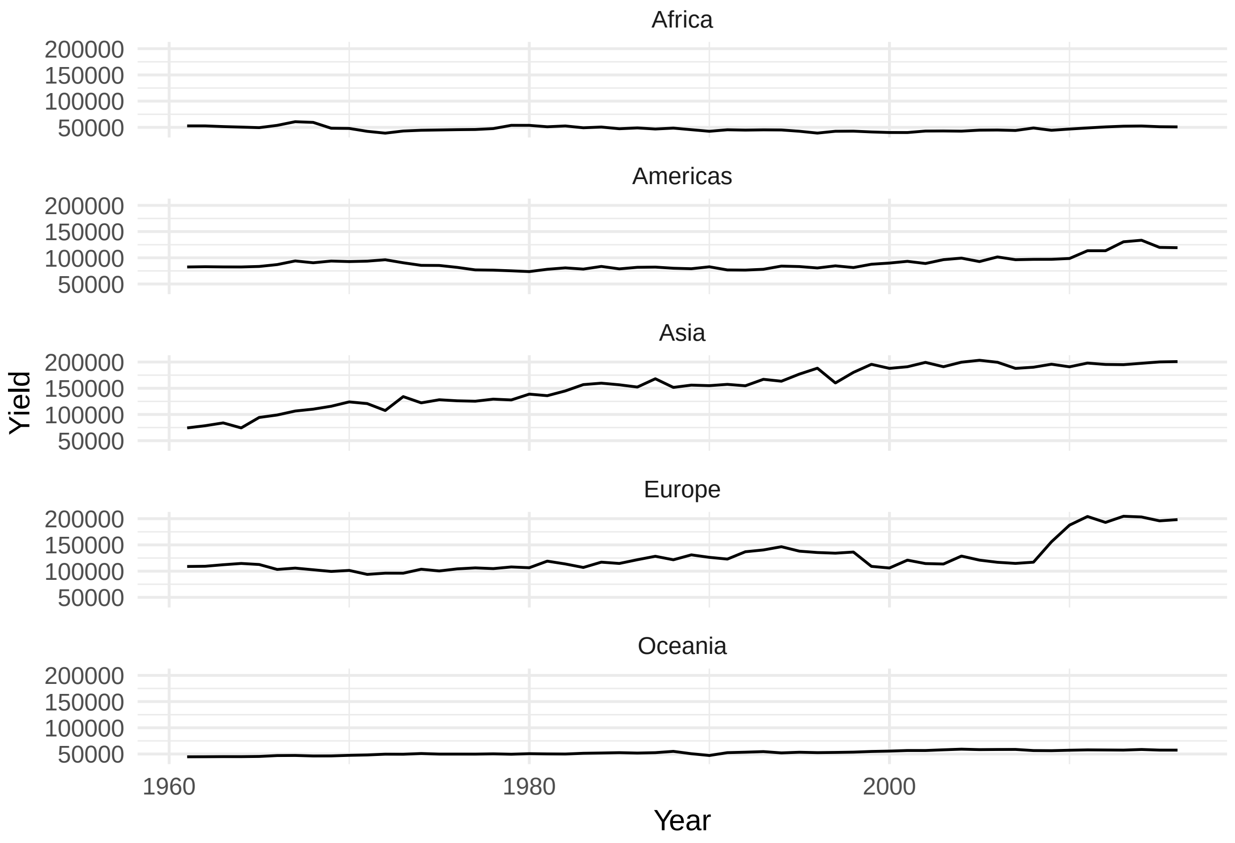
```r
ggplot(w, aes(x = Year, y = Yield, group = Region))+
  geom_line()+
  geom_point(aes(size = Area), alpha = 0.5)+
  facet_wrap(~Region, ncol = 1)
```



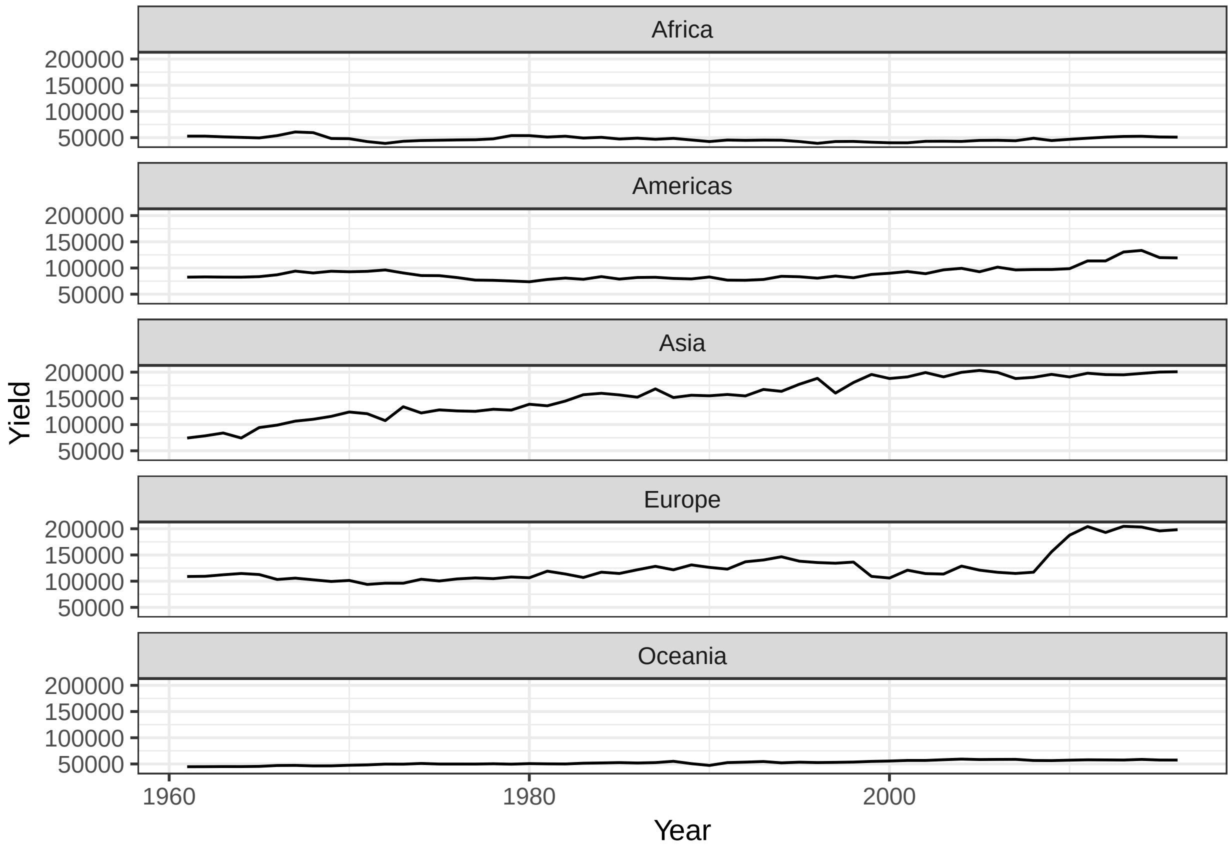
Applying themes

ggplot2 includes standardized themes that we can apply to our plots in order to change the overall appearance. These themes are listed on the back of your Cheat Sheet under "Themes". Let's look at two: (1) `theme_minimal()` and (2) `theme_classic()`.

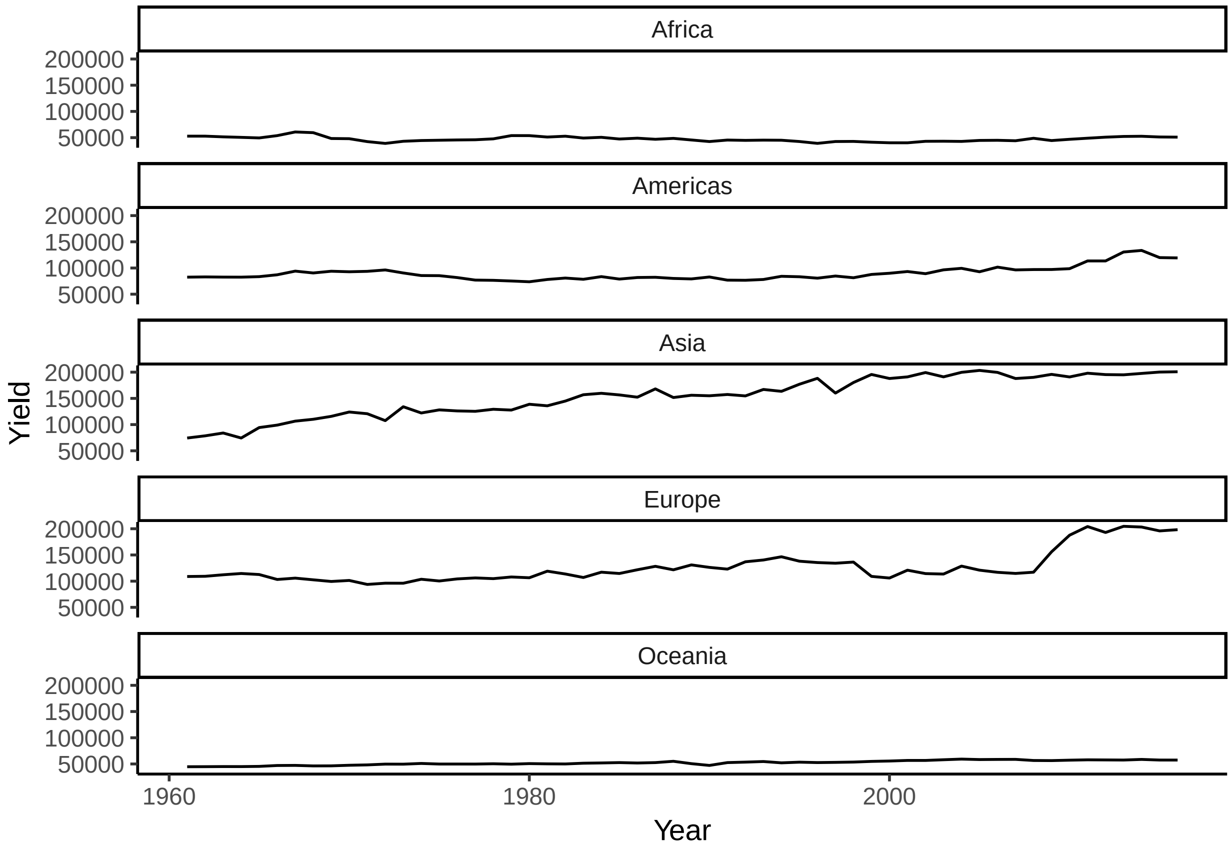
```r
ggplot(w, aes(x = Year, y = Yield))+
  geom_line()+
  facet_wrap(~Region, ncol = 1)+
  theme_minimal()
```



```
ggplot(w, aes(x = Year, y = Yield))+
  geom_line()+
  facet_wrap(~Region, ncol = 1)+
  theme_bw()
```



```
ggplot(w, aes(x = Year, y = Yield))+
  geom_line()+
  facet_wrap(~Region, ncol = 1)+
  theme_classic()
```

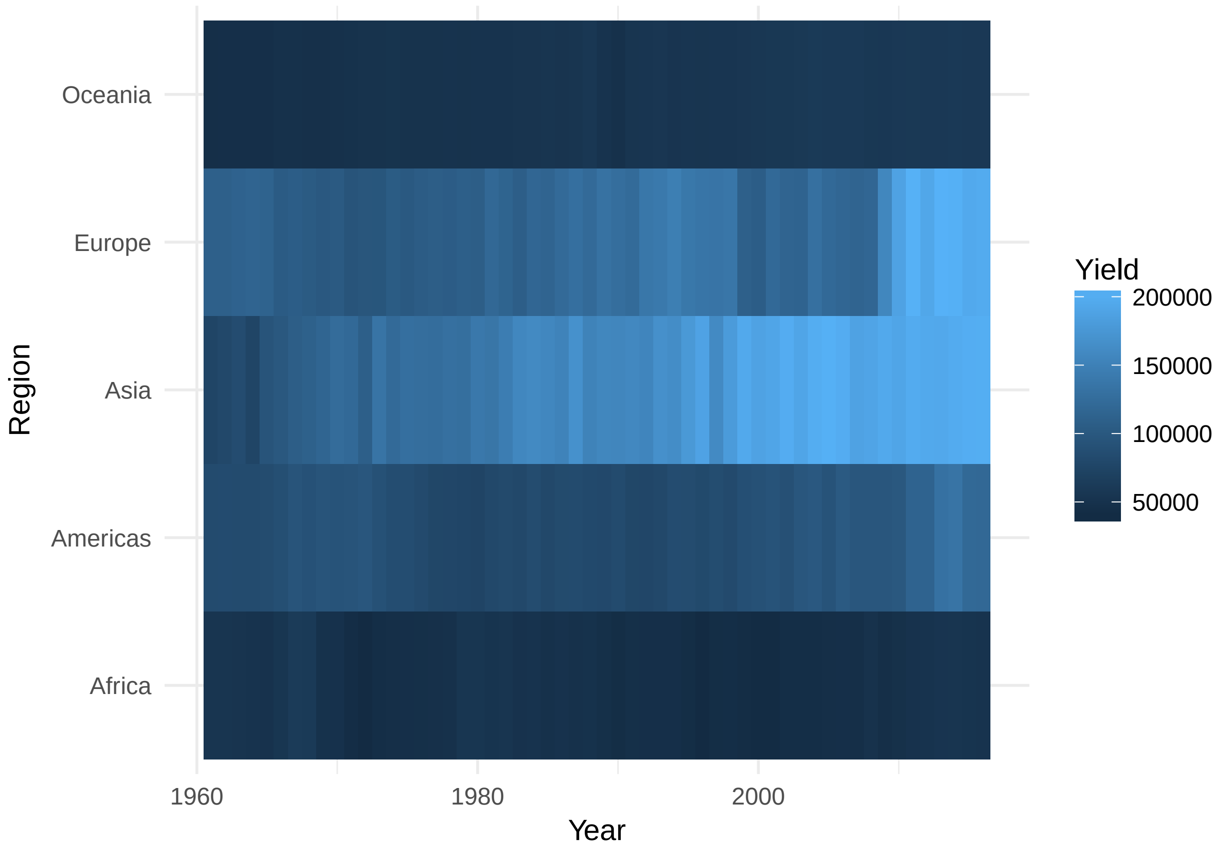


There is also a package called `ggthemes` that includes several other themes that we can apply to our ggplots. See the ggthemes webpage for examples of the other themes you can choose from. Remember, if you'd like to use a ggtheme, you'll have to install and load the `ggthemes` package.

Heatmap

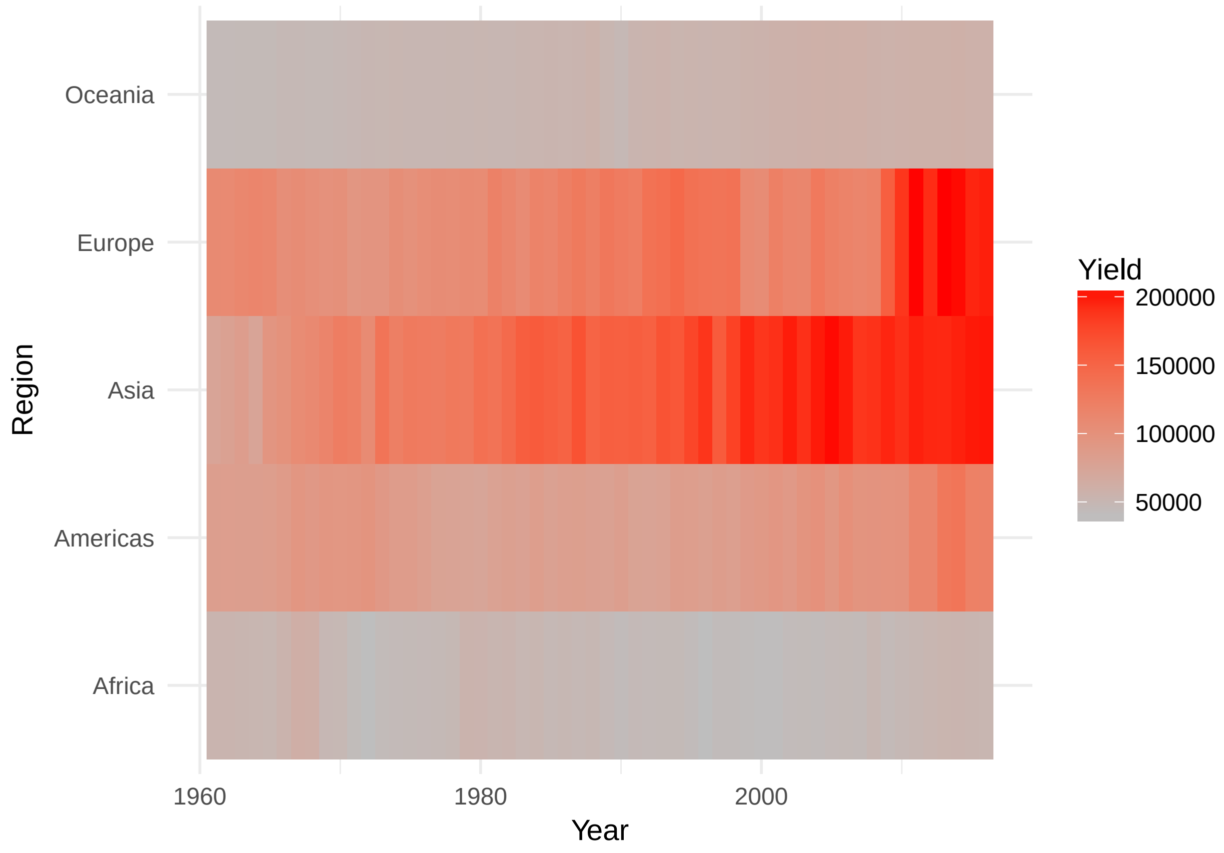
Instead of using `geom_line()` and `geom_point()`, we can easily swap out other `geom` functions to see what types of trends may pop out when we use different visualization styles. We can look to our Cheat Sheet for inspiration as to what types of functions we might apply. One example is a heatmap, also referred to as a "raster". Heatmaps are considered to be three-dimensional plots because the groupings count as a dimension. This makes more sense visually:

```r
ggplot(w, aes(x = Year, y = Region))+
  geom_raster(aes(fill = Yield))+
  theme_minimal()
```



Note that we specified `aes(fill = Yield)` in the `geom_raster()` argument. This is because the fill of each pixel corresponds to the *Yield*, which is the “third dimension” of the plot. The colors here are counterintuitive - the lighter color corresponds to larger yields, whereas the darker color corresponds to smaller yields. In this particular case in which we’re working with a *continuous color ramp*, which is how we describe the color gradient in the legend. To modify these colors, we can add the function `scale_fill_gradient(low = "color", high = "color")`. Details on changing the color and fill scales is on the back of your Cheat Sheet under “Scales” -> “Color and fill scales”.

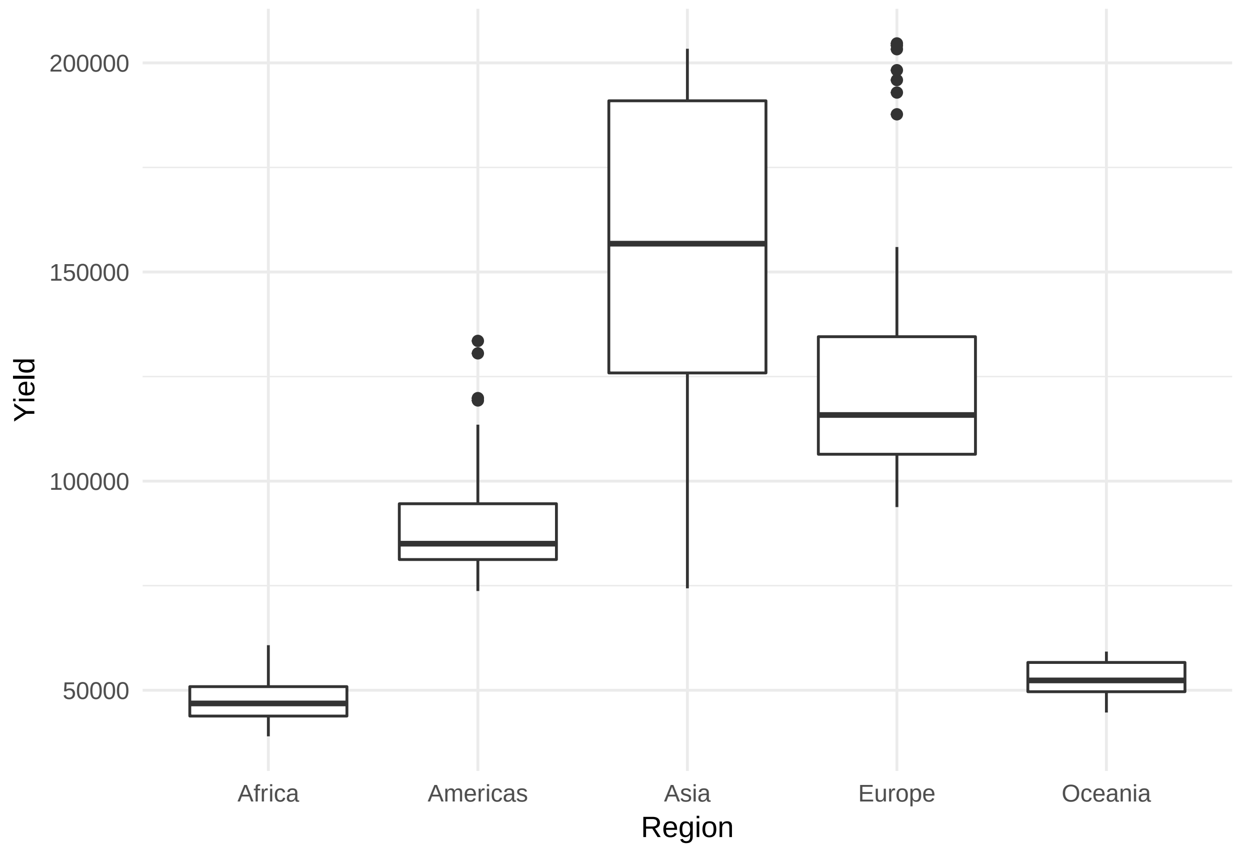
```r
ggplot(w, aes(x = Year, y = Region))+
  geom_raster(aes(fill = Yield))+
  scale_fill_gradient(low = "grey", high = "red")+
  theme_minimal()
```



Histograms and boxplots

We can also look at the yield data a few different ways, such as through histograms. Note that the variables we specify as x and y will change depending on how we plot the data.

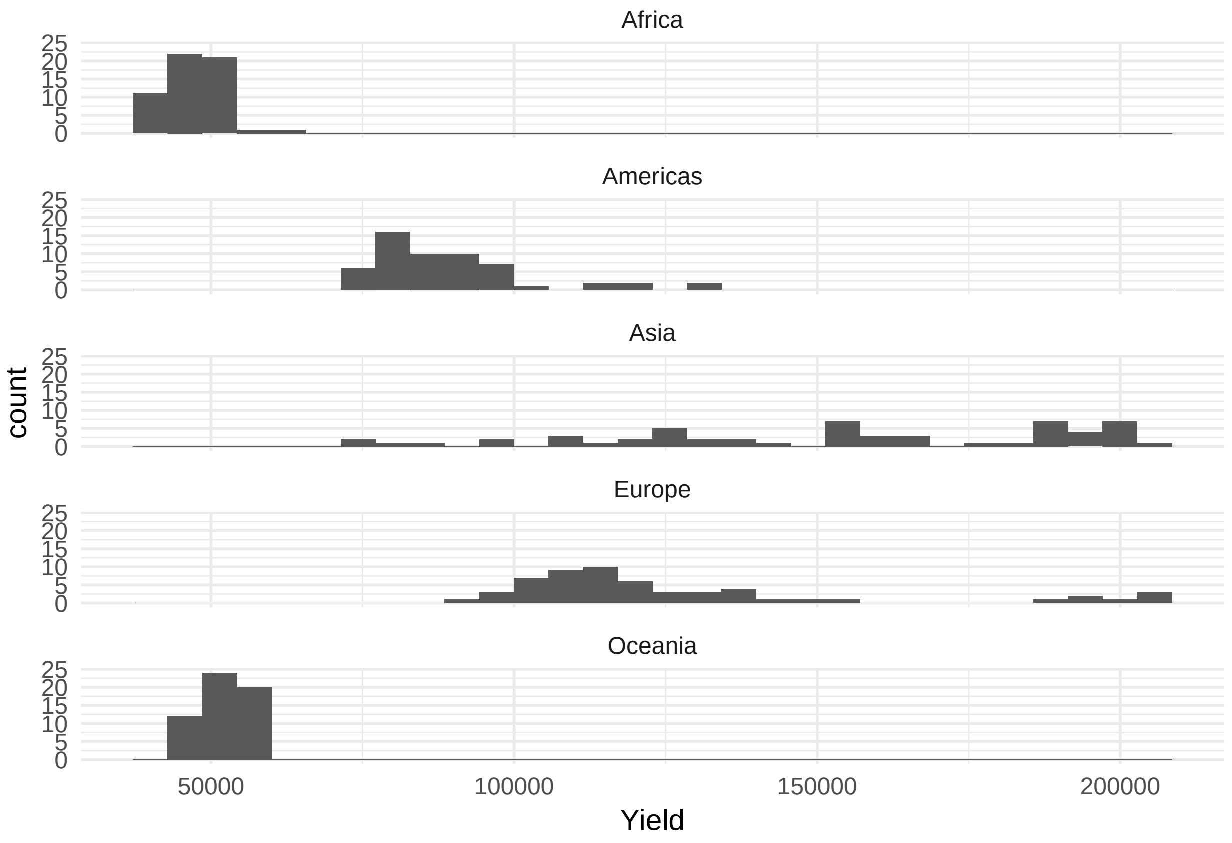
```r
ggplot(w, aes(x = Region, y = Yield))+
  geom_boxplot()+
  theme_minimal()
```



Histograms are plots that only include **1 variable**, so only x needs to be specified. Note that `geom_histogram()` is on your Cheat Sheet under the "One Variable" section.

```
ggplot(w, aes(x = Yield))+
  geom_histogram()+
  facet_wrap(~Region, ncol = 1)+
  theme_minimal()
```

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



Some additional ggplot functions that might be helpful for this assignment

- `ggtitle("Your Title")` can adds a title to a ggplot
- `ylim(c(lower, upper))` can be used to adjust the y-axis limits
- `xlim(c(lower, upper))` can be used to adjust the x-axis limits