

Data Visualization with ggplot2

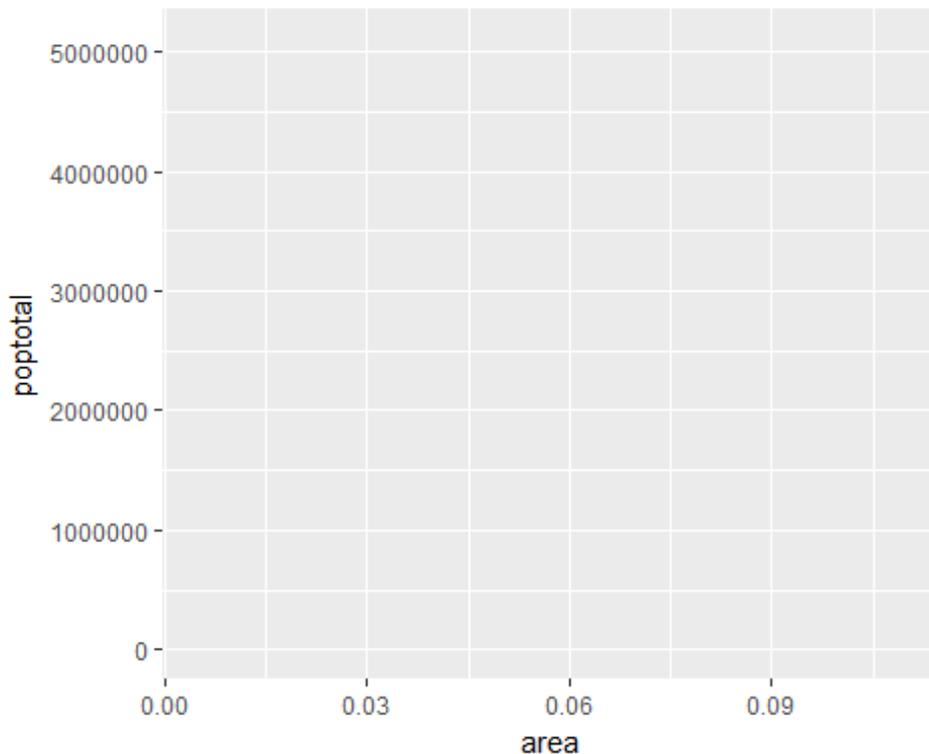
1. Understanding the ggplot syntax

The syntax for constructing ggplots could be puzzling if you are a beginner or work primarily with base graphics. The main difference is that, unlike base graphics, ggplot works with dataframes and not individual vectors. All the data needed to make the plot is typically be contained within the dataframe supplied to the ggplot() itself or can be supplied to respective geoms. More on that later.

The second noticeable feature is that you can keep enhancing the plot by adding more layers (and themes) to an existing plot created using the ggplot() function.

Let's initialize a basic ggplot based on the midwest dataset.

```
# Setup  
options(scipen=999) # turn off scientific notation like 1e+06  
library(ggplot2)  
data("midwest", package = "ggplot2") # Load the data  
# midwest <- read.csv("http://goo.gl/G1K41K") # alt source  
  
# Init Ggplot  
ggplot(midwest, aes(x=area, y=poptotal)) # area and poptotal are columns in 'midwest'
```



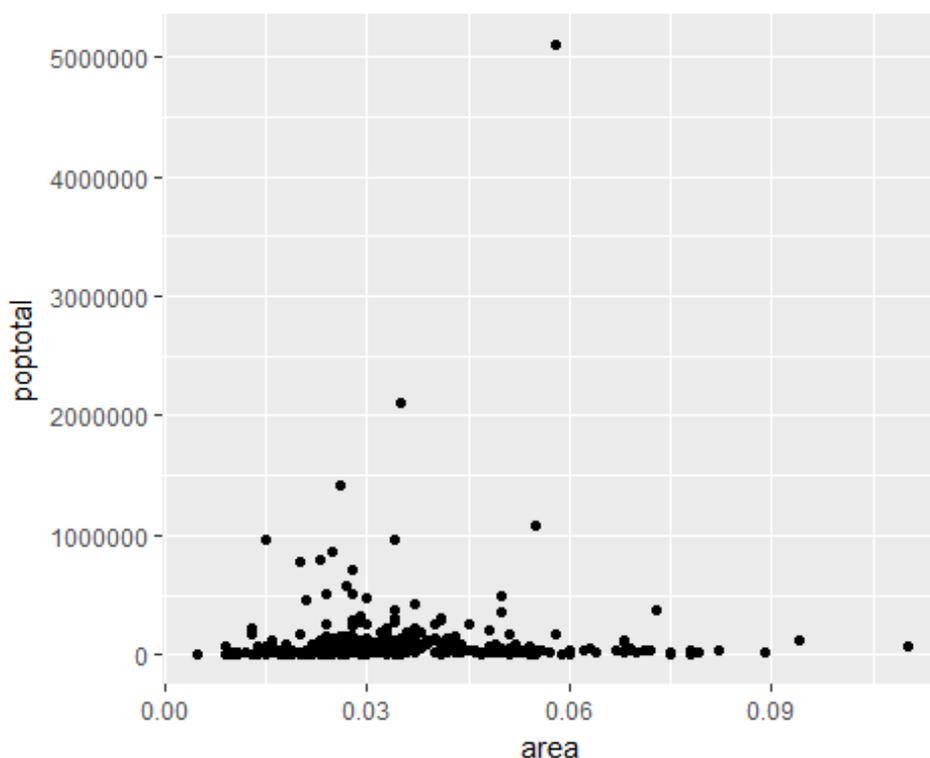
A blank ggplot is drawn. Even though the x and y are specified, there are no points or lines in it. This is because, ggplot doesn't assume that you meant a scatterplot or a line chart to be drawn. I have only told ggplot what dataset to use and what columns should be used for X and Y axis. I haven't explicitly asked it to draw any points.

Also note that aes() function is used to specify the X and Y axes. That's because, any information that is part of the source dataframe has to be specified inside the aes() function.

2. How to Make a Simple Scatterplot

Let's make a scatterplot on top of the blank ggplot by adding points using a geom layer called geom_point.

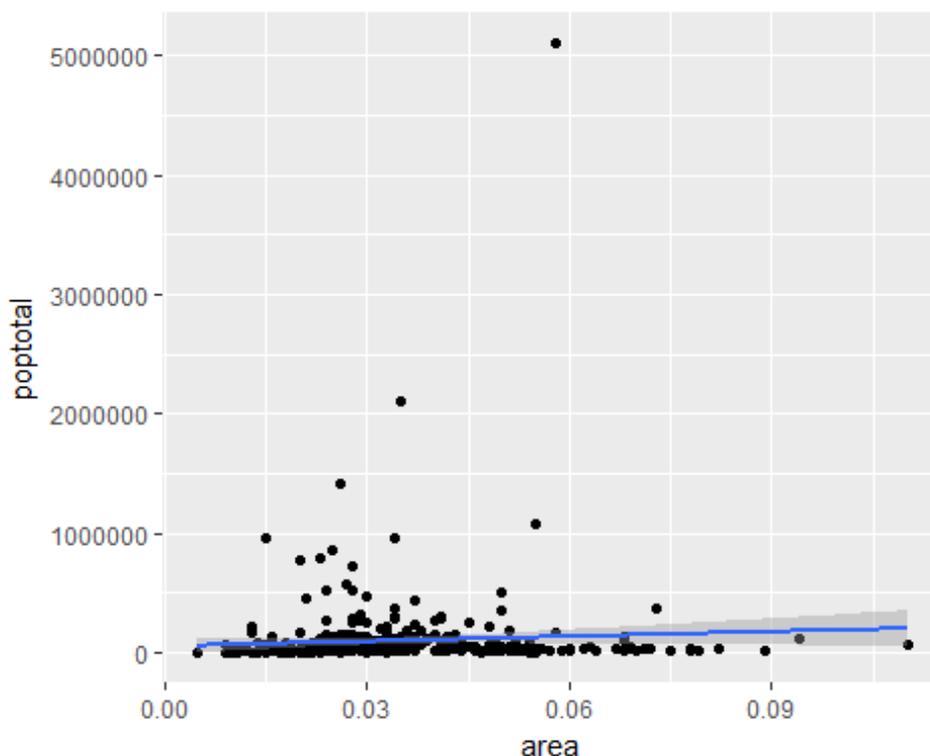
```
library(ggplot2)
ggplot(midwest, aes(x=area, y=poptotal)) + geom_point()
```



We got a basic scatterplot, where each point represents a county. However, it lacks some basic components such as the plot title, meaningful axis labels etc. Moreover most of the points are concentrated on the bottom portion of the plot, which is not so nice. You will see how to rectify these in upcoming steps.

Like geom_point(), there are many such geom layers which we will see in a subsequent part in this tutorial series. For now, let's just add a smoothing layer using geom_smooth(method='lm'). Since the method is set as lm (short for linear model), it draws the line of best fit.

```
library(ggplot2)
g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom_point() + geom_smooth(method="lm") # set se=FALSE to turnoff confidence bands
plot(g)
```



The line of best fit is in blue. Can you find out what other method options are available for `geom_smooth`? (note: see `?geom_smooth`). You might have noticed that majority of points lie in the bottom of the chart which doesn't really look nice. So, let's change the Y-axis limits to focus on the lower half.

3. Adjusting the X and Y axis limits

The X and Y axis limits can be controlled in 2 ways.

Method 1: By deleting the points outside the range

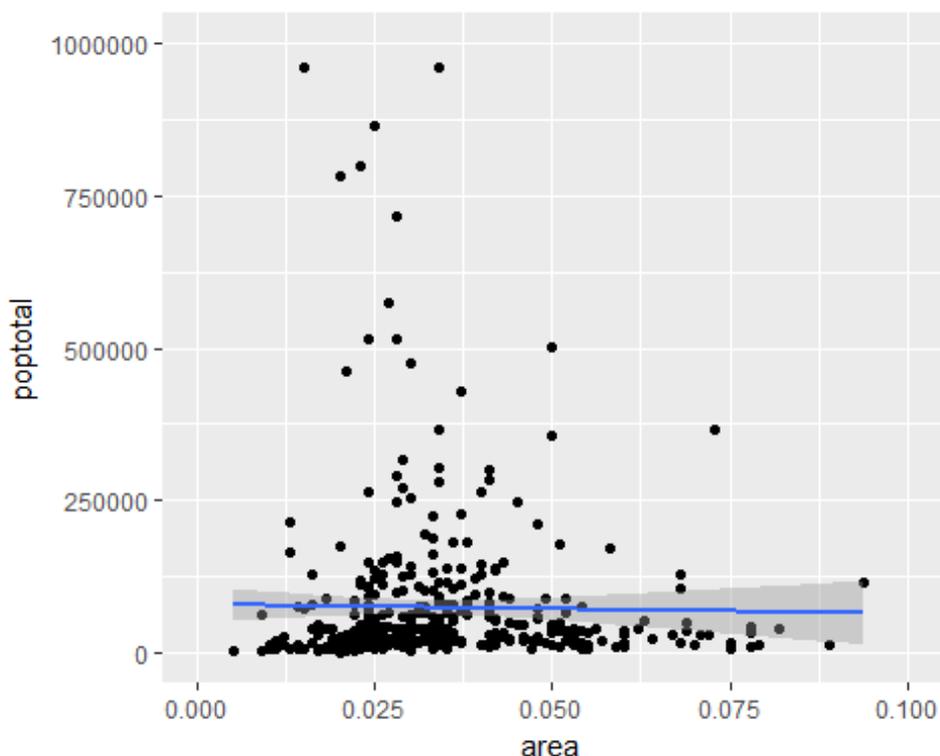
This will change the lines of best fit or smoothing lines as compared to the original data.

This can be done by `xlim()` and `yylim()`. You can pass a numeric vector of length 2 (with max and min values) or just the max and min values itself.

```
library(ggplot2)
g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom_point() + geom_smooth(method="lm") # set se=FALSE to turnoff confidence bands

# Delete the points outside the limits
g + xlim(c(0, 0.1)) + ylim(c(0, 1000000)) # deletes points
```

```
## Warning: Removed 5 rows containing non-finite values (stat_smooth).  
## Warning: Removed 5 rows containing missing values (geom_point).
```



```
# g + xlim(0, 0.1) + ylim(0, 1000000) # deletes points
```

In this case, the chart was not built from scratch but rather was built on top of `g`. This is because, the previous plot was stored as `g`, a `ggplot` object, which when called will reproduce the original plot. Using `ggplot`, you can add more layers, themes and other settings on top of this plot.

Did you notice that the line of best fit became more horizontal compared to the original plot? This is because, when using `xlim()` and `ylim()`, the points outside the specified range are deleted and will not be considered while drawing the line of best fit (using `geom_smooth(method='lm')`). This feature might come in handy when you wish to know how the line of best fit would change when some extreme values (or outliers) are removed.

Method 2: Zooming In

The other method is to change the X and Y axis limits by zooming in to the region of interest without deleting the points. This is done using `coord_cartesian()`.

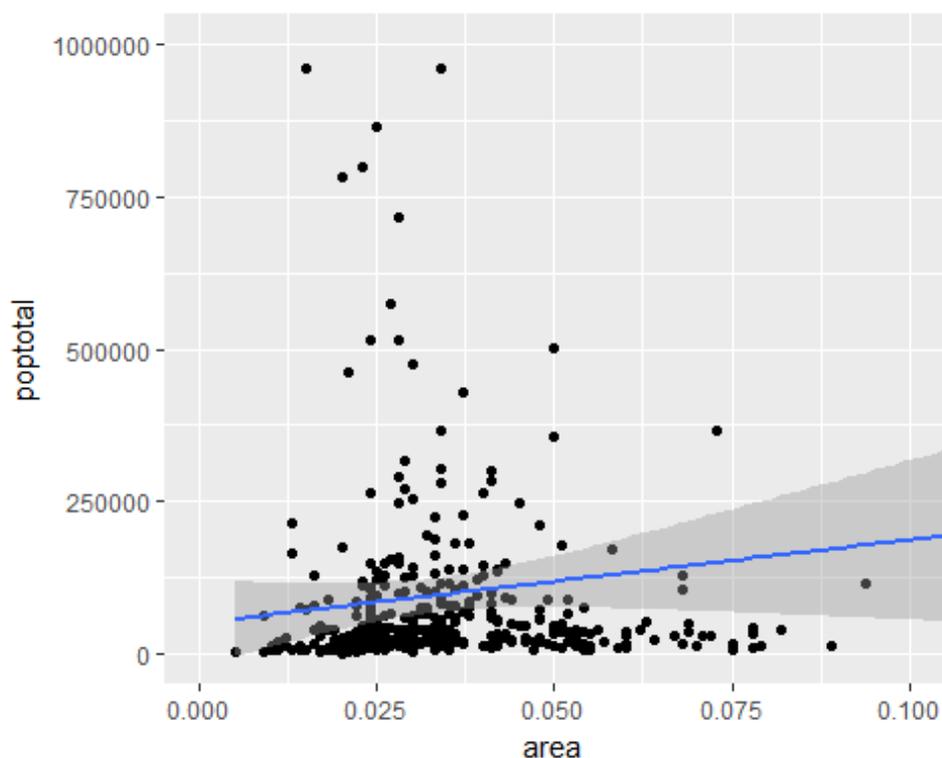
Let's store this plot as `g1`.

```

library(ggplot2)
g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom_point() + geom_smooth(method=
"lm") # set se=FALSE to turnoff confidence bands

# Zoom in without deleting the points outside the limits.
# As a result, the line of best fit is the same as the original plot.
g1 <- g + coord_cartesian(xlim=c(0,0.1), ylim=c(0, 1000000)) # zooms in
plot(g1)

```



Since all points were considered, the line of best fit did not change.

4. How to Change the Title and Axis Labels

I have stored this as g1. Let's add the plot title and labels for X and Y axis. This can be done in one go using the labs() function with title, x and y arguments. Another option is to use the ggtitle(), xlab() and ylab().

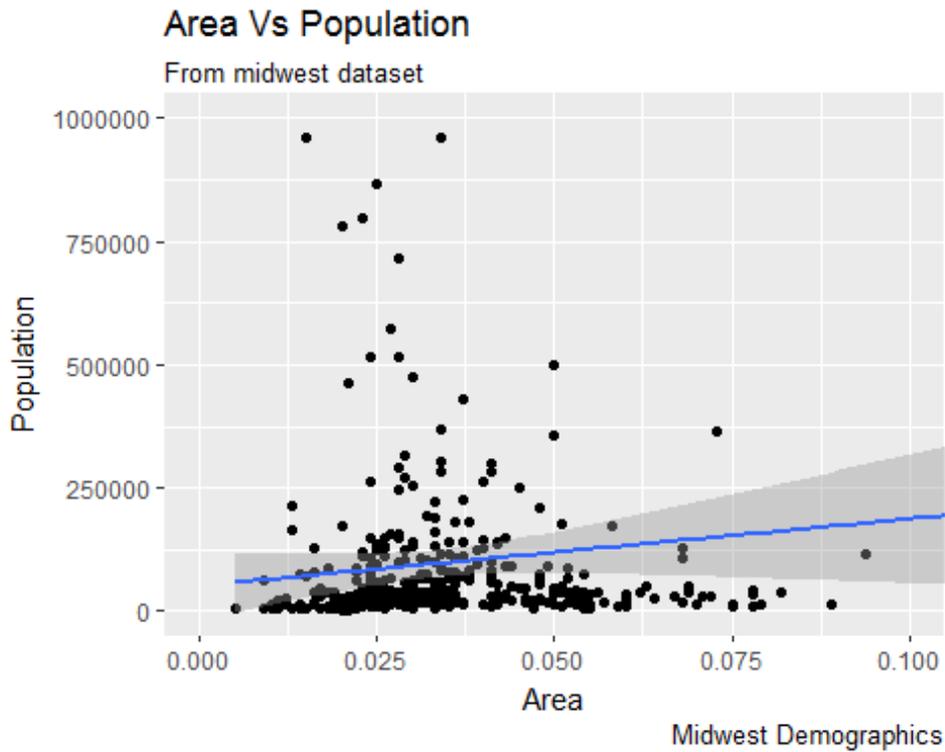
```

library(ggplot2)
g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom_point() + geom_smooth(method=
"lm") # set se=FALSE to turnoff confidence bands

g1 <- g + coord_cartesian(xlim=c(0,0.1), ylim=c(0, 1000000)) # zooms in

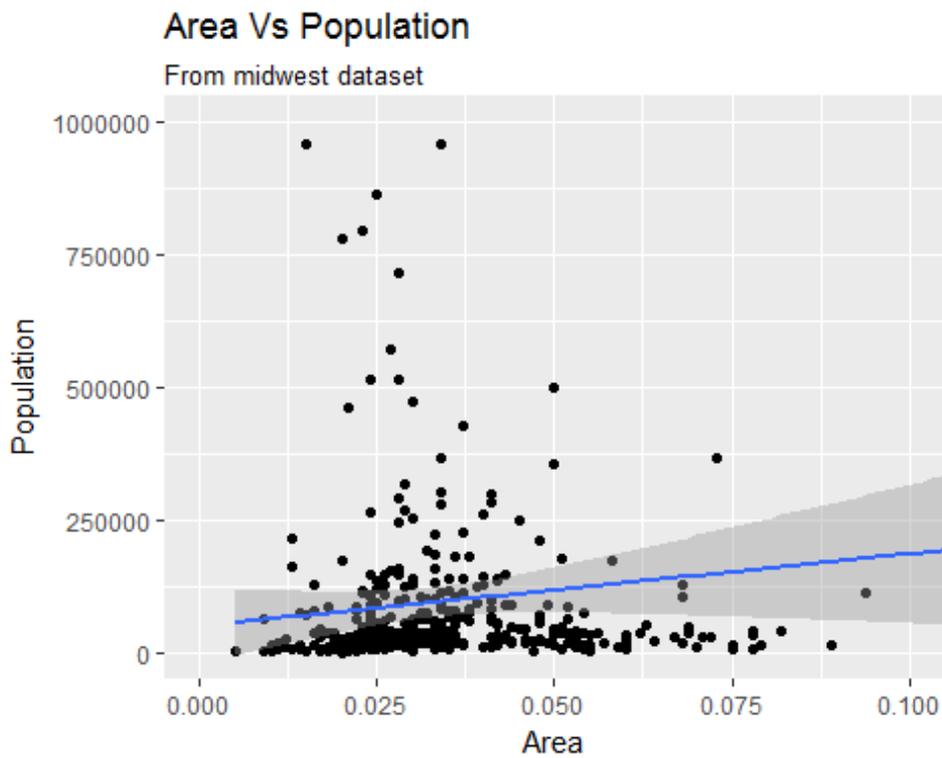
# Add Title and Labels
g1 + labs(title="Area Vs Population", subtitle="From midwest dataset", y="Populati
on", x="Area", caption="Midwest Demographics")

```



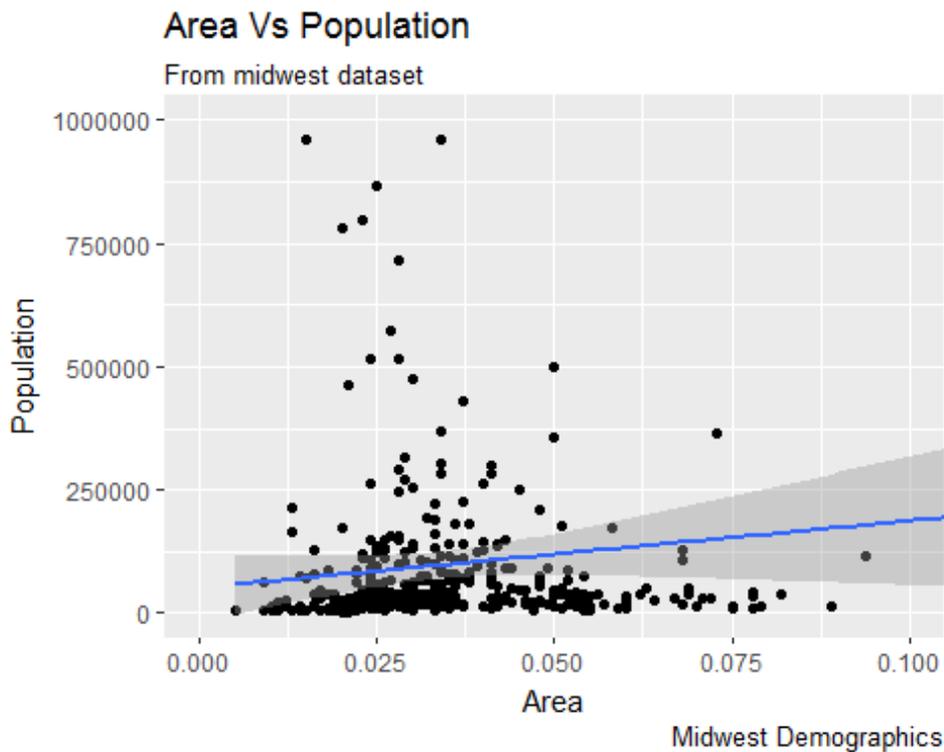
or

```
g1 + ggtitle("Area Vs Population", subtitle="From midwest dataset") + xlab("Area")  
+ ylab("Population")
```



Excellent! So here is the full function call.

```
# Full Plot call
library(ggplot2)
ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point() +
  geom_smooth(method="lm") +
  coord_cartesian(xlim=c(0,0.1), ylim=c(0, 1000000)) +
  labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population",
, x="Area", caption="Midwest Demographics")
```



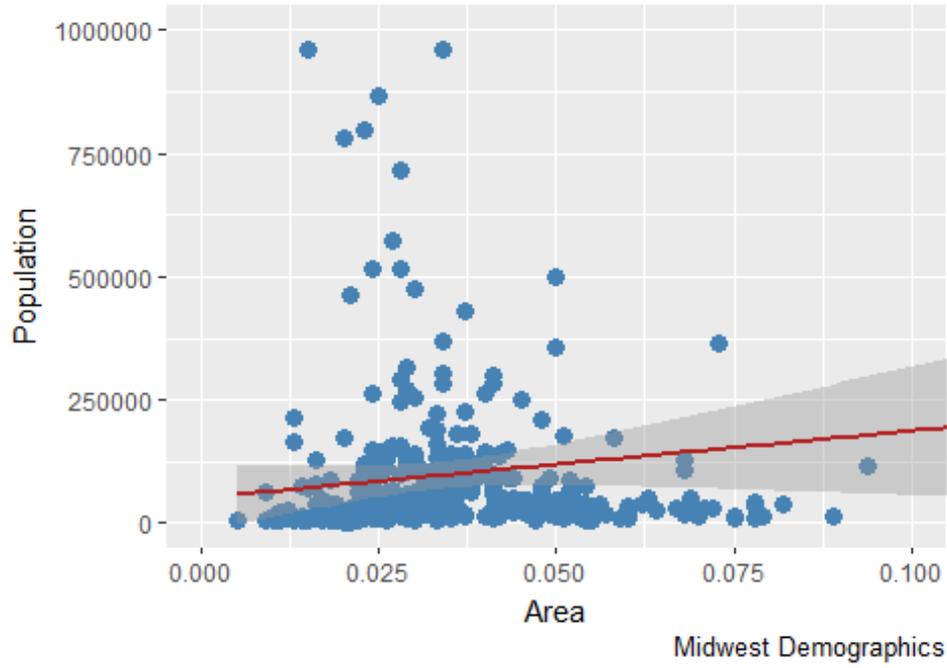
5. How to Change the Color and Size of Points

How to Change the Color and Size To Static? We can change the aesthetics of a geom layer by modifying the respective geoms. Let's change the color of the points and the line to a static value.

```
library(ggplot2)
ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point(col="steelblue", size=3) + # Set static color and size for points
  geom_smooth(method="lm", col="firebrick") + # change the color of line
  coord_cartesian(xlim=c(0, 0.1), ylim=c(0, 1000000)) +
  labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population",
, x="Area", caption="Midwest Demographics")
```

Area Vs Population

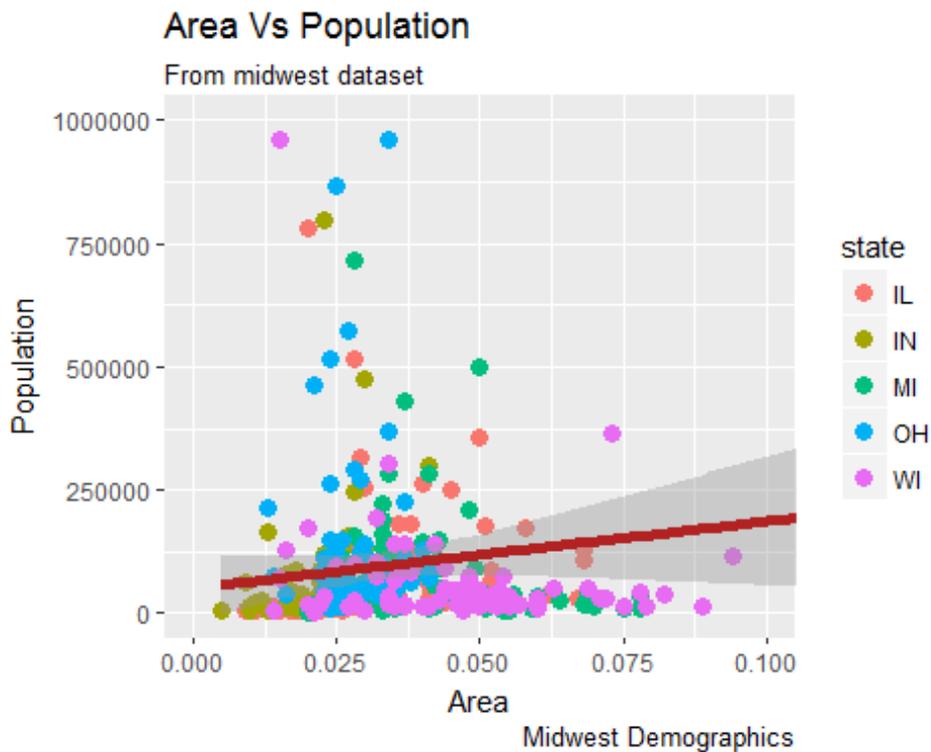
From midwest dataset



How to Change the Color To Reflect Categories in Another Column?

Suppose if we want the color to change based on another column in the source dataset (midwest), it must be specified inside the aes() function.

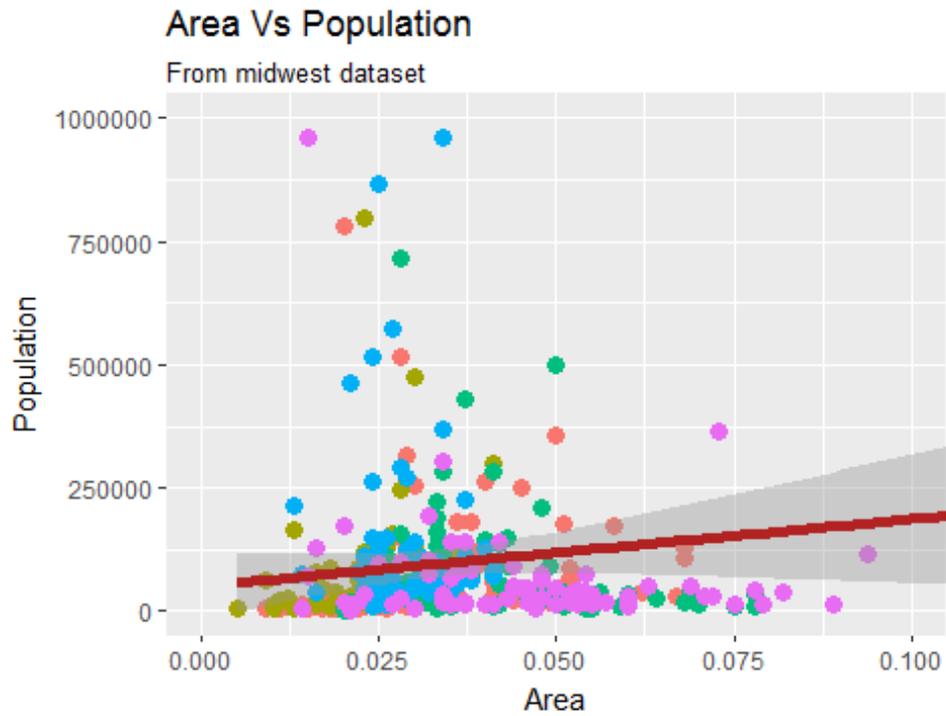
```
library(ggplot2)
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point(aes(col=state), size=3) + # Set color to vary based on state categories.
  geom_smooth(method="lm", col="firebrick", size=2) +
  coord_cartesian(xlim=c(0, 0.1), ylim=c(0, 1000000)) +
  labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population",
    x="Area", caption="Midwest Demographics")
plot(gg)
```



Now each point is colored based on the state it belongs to because of aes(col=state). Not just color, but size, shape, stroke (thickness of boundary) and fill (fill color) can be used to discriminate groupings.

As an added benefit, the legend is added automatically. If needed, it can be removed by setting the legend.position to None from within a theme() function.

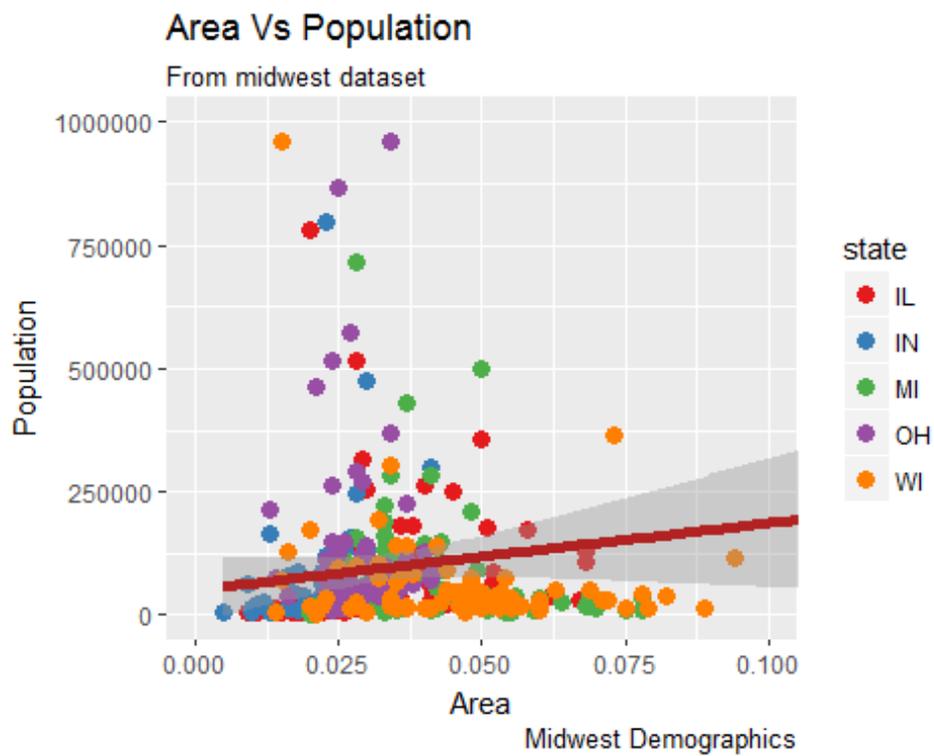
```
gg + theme(legend.position="None") # remove Legend
```



Also, You can change the

color palette entirely.

```
gg + scale_colour_brewer(palette = "Set1") # change color palette
```



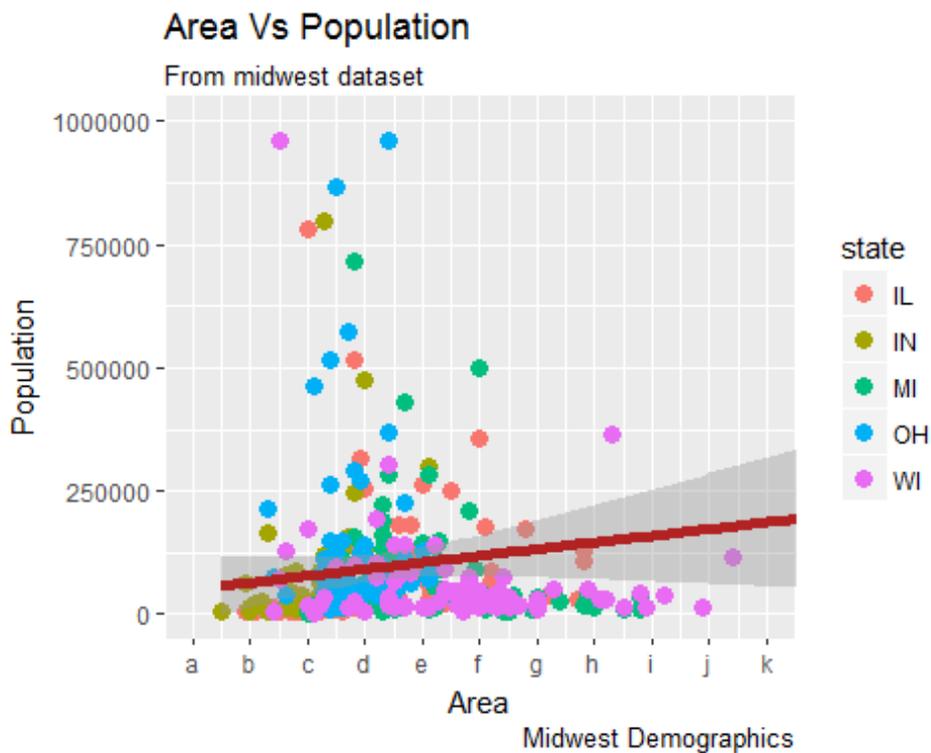
Step 2: Change the labels You can optionally change the labels at the axis ticks. labels take a vector of the same length as breaks.

Let me demonstrate by setting the labels to alphabets from a to k (though there is no meaning to it in this context).

```
library(ggplot2)

# Base Plot
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point(aes(col=state), size=3) + # Set color to vary based on state categories.
  geom_smooth(method="lm", col="firebrick", size=2) +
  coord_cartesian(xlim=c(0, 0.1), ylim=c(0, 1000000)) +
  labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population",
       x="Area", caption="Midwest Demographics")

# Change breaks + Label
gg + scale_x_continuous(breaks=seq(0, 0.1, 0.01), labels = letters[1:11])
```



scale, use `scale_x_reverse()`.

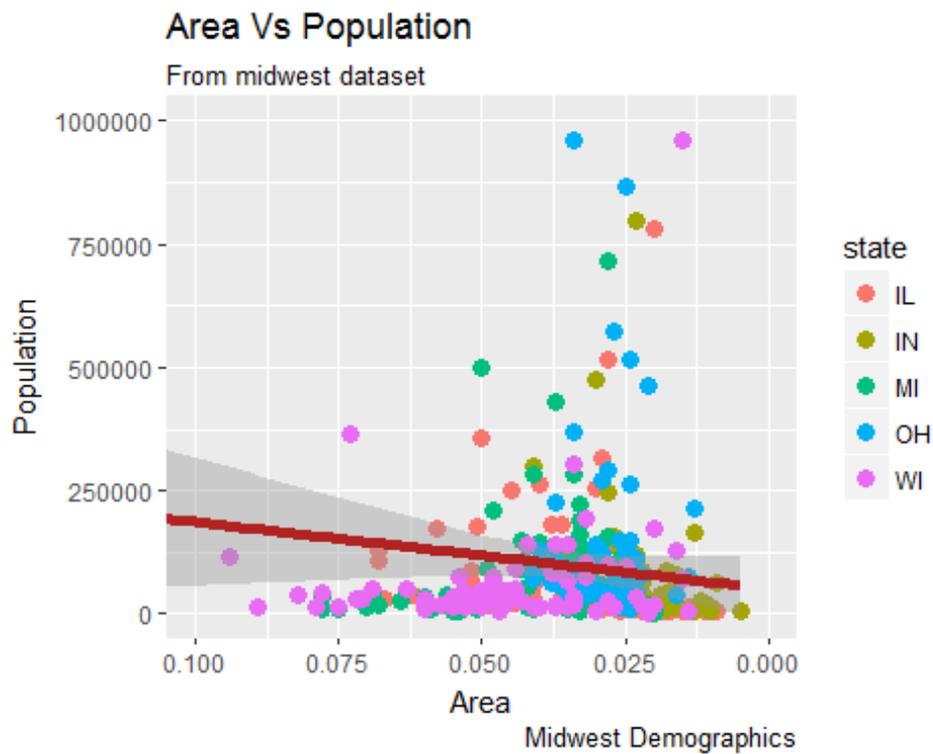
If you need to reverse the

```

library(ggplot2)
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point(aes(col=state), size=3) + # Set color to vary based on state categories.
  geom_smooth(method="lm", col="firebrick", size=2) +
  coord_cartesian(xlim=c(0, 0.1), ylim=c(0, 1000000)) +
  labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population",
  x="Area", caption="Midwest Demographics")

# Reverse X Axis Scale
gg + scale_x_reverse()

```



How to Customize the Entire Theme in One Shot using Pre-Built Themes?

Finally, instead of changing the theme components individually (which I discuss in detail in part 2), we can change the entire theme itself using pre-built themes. The help page `?theme_bw` shows all the available built-in themes.

This again is commonly done in couple of ways. Use the `theme_set()` to set the theme before drawing the ggplot. Note that this setting will affect all future plots. Draw the ggplot and then add the overall theme setting (eg. `theme_bw()`)

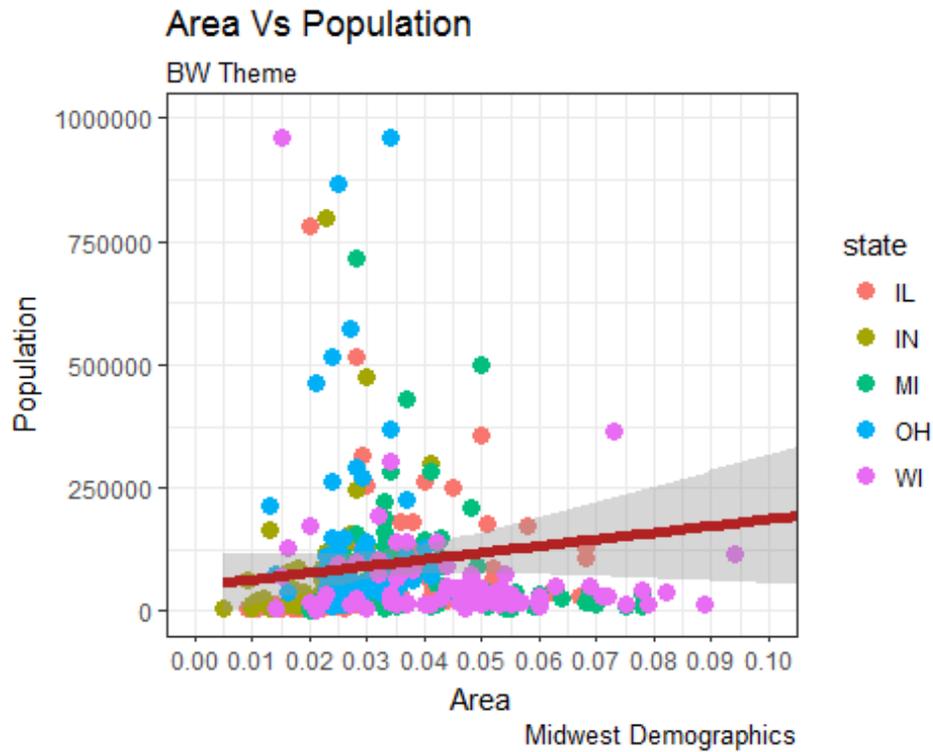
```
library(ggplot2)

# Base plot
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point(aes(col=state), size=3) + # Set color to vary based on state categories.
  geom_smooth(method="lm", col="firebrick", size=2) +
  coord_cartesian(xlim=c(0, 0.1), ylim=c(0, 1000000)) +
  labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population",
        x="Area", caption="Midwest Demographics")

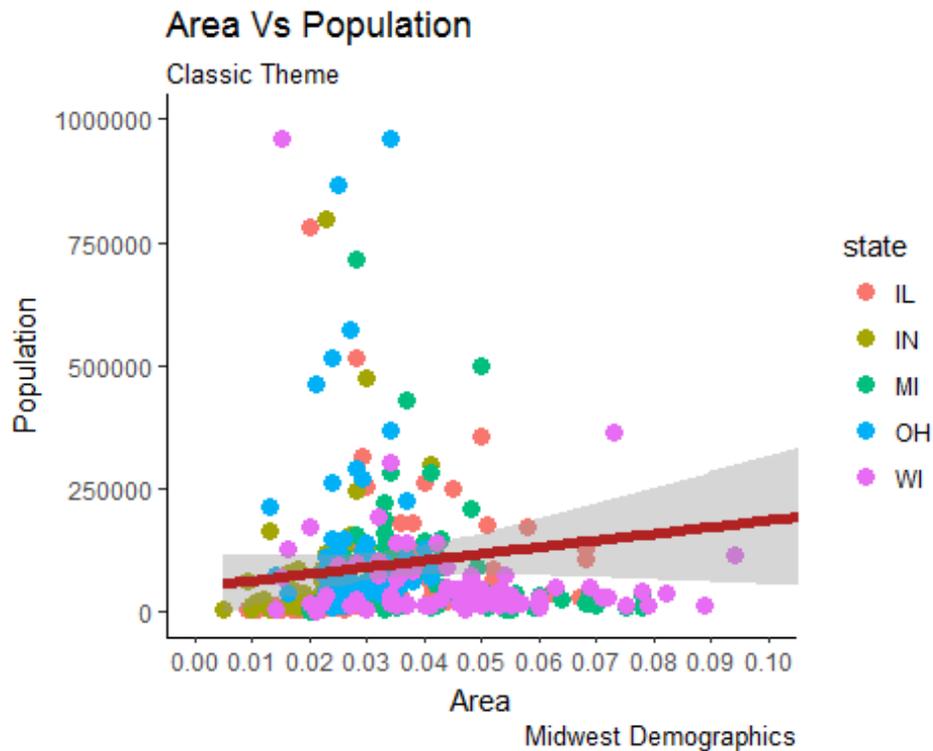
gg <- gg + scale_x_continuous(breaks=seq(0, 0.1, 0.01))

# method 1: Using theme_set()
theme_set(theme_classic()) # not run gg

# method 2: Adding theme Layer itself.
gg + theme_bw() + labs(subtitle="BW Theme")
```



```
gg + theme_classic() + labs(subtitle="Classic Theme")
```



For more customized and fancy themes have a look at the `ggthemes` package and the `ggthemr` package.

Bubble plot

While scatterplot lets you compare the relationship between 2 continuous variables, bubble chart serves well if you want to understand relationship within the underlying groups based on:

A Categorical variable (by changing the color) and Another continuous variable (by changing the size of points). In simpler words, bubble charts are more suitable if you have 4-Dimensional data where two of them are numeric (X and Y) and one other categorical (color) and another numeric variable (size).

The bubble chart clearly distinguishes the range of displ between the manufacturers and how the slope of lines-of-best-fit varies, providing a better visual comparison between the groups.

```
# Load package and data
library(ggplot2)
data(mpg, package="ggplot2")
# mpg <- read.csv("http://goo.gl/uEeRGu")

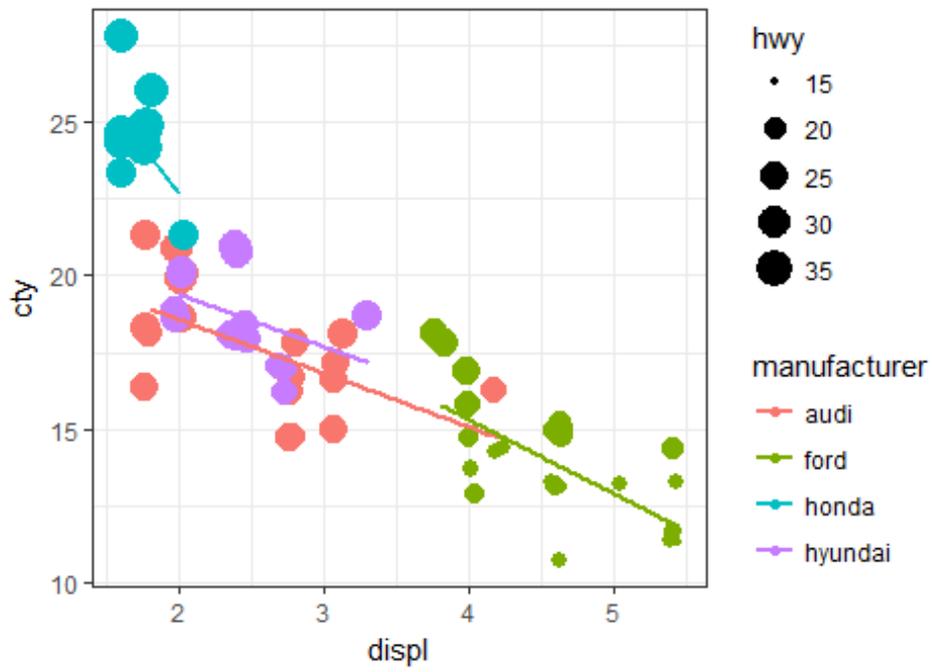
mpg_select <- mpg[mpg$manufacturer %in% c("audi", "ford", "honda", "hyundai"), ]

# Scatterplot
theme_set(theme_bw()) # pre-set the bw theme.
g <- ggplot(mpg_select, aes(displ, cty)) +
  labs(subtitle="mpg: Displacement vs City Mileage",
       title="Bubble chart")

g + geom_jitter(aes(col=manufacturer, size=hwy)) +
  geom_smooth(aes(col=manufacturer), method="lm", se=F)
```

Bubble chart

mpg: Displacement vs City Mileage



Ordered Bar Chart

Ordered Bar Chart is a Bar Chart that is ordered by the Y axis variable. Just sorting the dataframe by the variable of interest isn't enough to order the bar chart. In order for the bar chart to retain the order of the rows, the X axis variable (i.e. the categories) has to be converted into a factor.

Let's plot the mean city mileage for each manufacturer from mpg dataset. First, aggregate the data and sort it before you draw the plot. Finally, the X variable is converted to a factor.

Let's see how that is done.

```
# Prepare data: group mean city mileage by manufacturer.
cty_mpg <- aggregate(mpg$cty, by=list(mpg$manufacturer), FUN=mean) # aggregate
colnames(cty_mpg) <- c("make", "mileage") # change column names
cty_mpg <- cty_mpg[order(cty_mpg$mileage), ] # sort
cty_mpg$make <- factor(cty_mpg$make, levels = cty_mpg$make) # to retain the order
in plot.
head(cty_mpg, 4)

##      make  mileage
## 9  lincoln 11.33333
## 8  land rover 11.50000
## 3    dodge 13.13514
## 10  mercury 13.25000
```

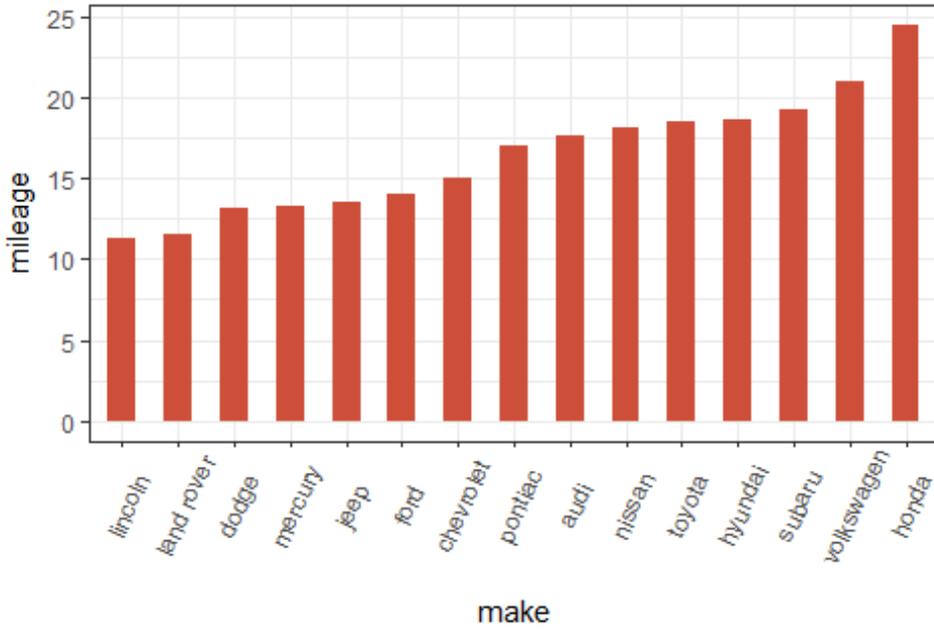
The X variable is now a factor, let's plot.

```
library(ggplot2)
theme_set(theme_bw())

# Draw plot
ggplot(cty_mpg, aes(x=make, y=mileage)) +
  geom_bar(stat="identity", width=.5, fill="tomato3") +
  labs(title="Ordered Bar Chart",
       subtitle="Make Vs Avg. Mileage",
       caption="source: mpg") +
  theme(axis.text.x = element_text(angle=65, vjust=0.6))
```

Ordered Bar Chart

Make Vs Avg. Mileage



source: mpg

Histogram

By default, if only one variable is supplied, the `geom_bar()` tries to calculate the count. In order for it to behave like a bar chart, the `stat=identity` option has to be set and x and y values must be provided.

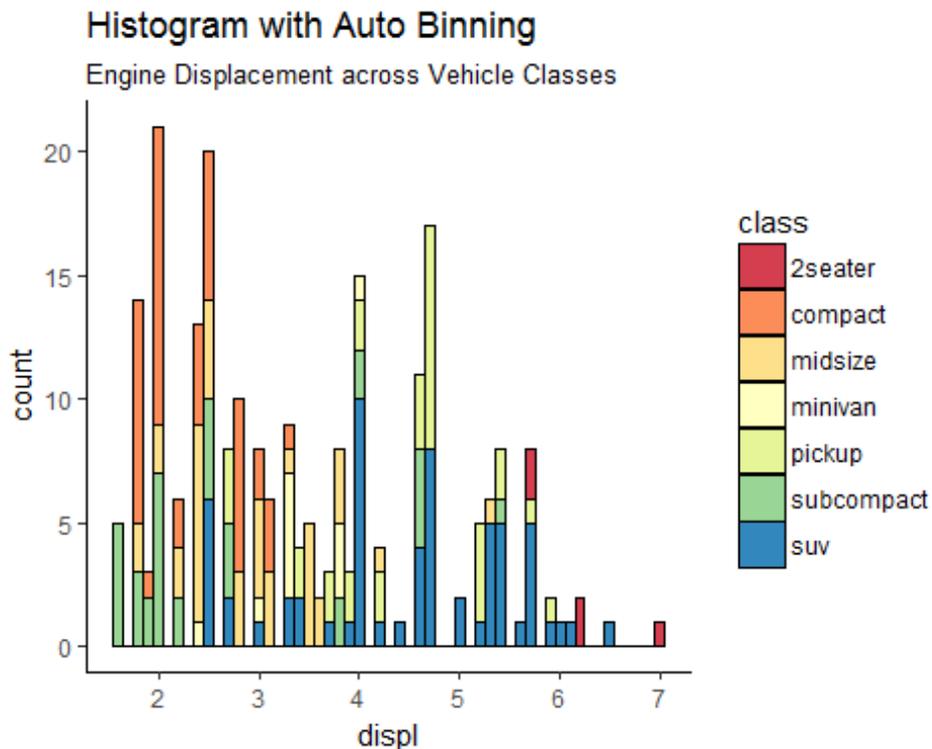
Histogram on a continuous variable

Histogram on a continuous variable can be accomplished using either `geom_bar()` or `geom_histogram()`. When using `geom_histogram()`, you can control the number of bars using the `bins` option. Else, you can set the range covered by each bin using `binwidth`. The value of `binwidth` is on the same scale as the continuous variable on which histogram is built. Since, `geom_histogram` gives facility to control both number of bins as well as `binwidth`, it is the preferred option to create histogram on continuous variables.

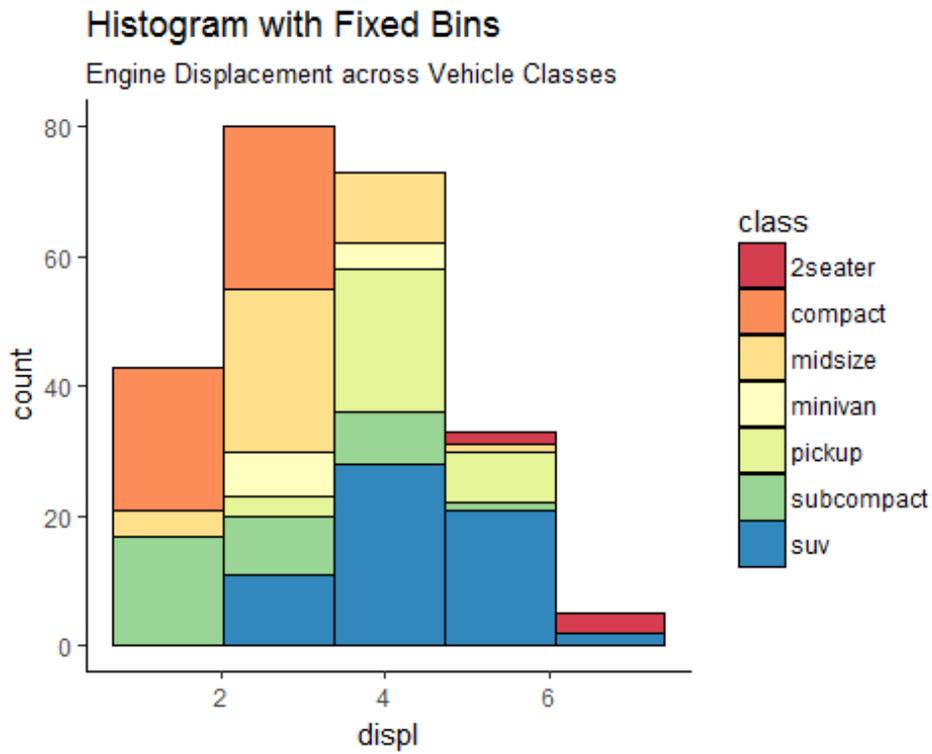
```
library(ggplot2)
theme_set(theme_classic())

# Histogram on a Continuous (Numeric) Variable
g <- ggplot(mpg, aes(displ)) + scale_fill_brewer(palette = "Spectral")

g + geom_histogram(aes(fill=class),
                  binwidth = .1,
                  col="black",
                  size=.1) + # change binwidth
labs(title="Histogram with Auto Binning",
      subtitle="Engine Displacement across Vehicle Classes")
```



```
g + geom_histogram(aes(fill=class),
  bins=5,
  col="black",
  size=.1) + # change number of bins
labs(title="Histogram with Fixed Bins",
  subtitle="Engine Displacement across Vehicle Classes")
```



Density plot

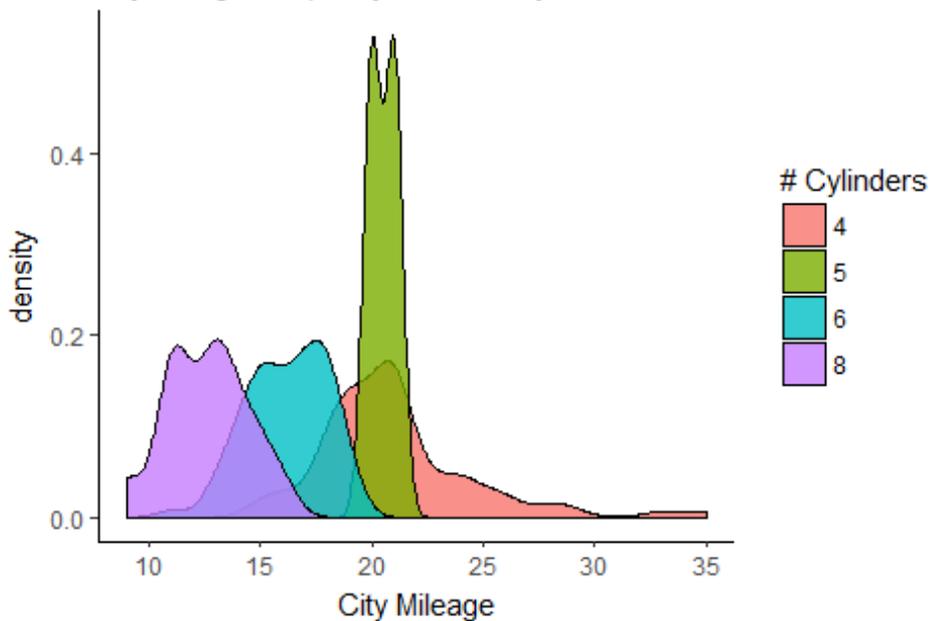
```
library(ggplot2)
theme_set(theme_classic())
```

```
# Plot
```

```
g <- ggplot(mpg, aes(cty))
g + geom_density(aes(fill=factor(cyl)), alpha=0.8) +
  labs(title="Density plot",
        subtitle="City Mileage Grouped by Number of cylinders",
        caption="Source: mpg",
        x="City Mileage",
        fill="# Cylinders")
```

Density plot

City Mileage Grouped by Number of cylinders



Source: mpg

Box Plot

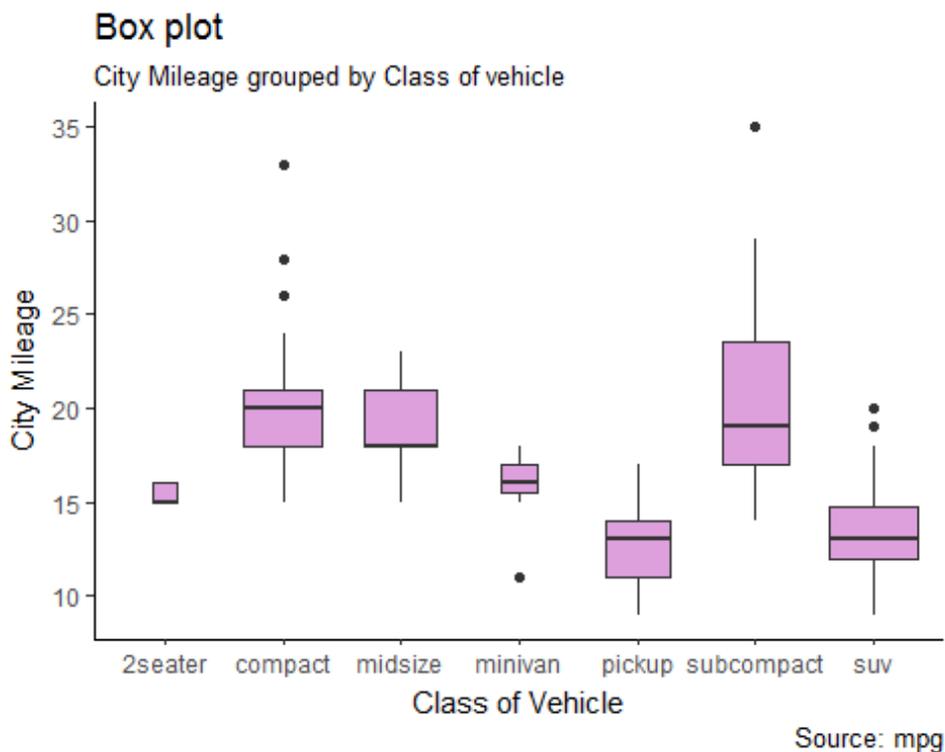
Box plot is an excellent tool to study the distribution. It can also show the distributions within multiple groups, along with the median, range and outliers if any.

The dark line inside the box represents the median. The top of box is 75%ile and bottom of box is 25%ile. The end points of the lines (aka whiskers) is at a distance of $1.5 \times \text{IQR}$, where IQR or Inter Quartile Range is the distance between 25th and 75th percentiles. The points outside the whiskers are marked as dots and are normally considered as extreme points.

Setting `varwidth=T` adjusts the width of the boxes to be proportional to the number of observation it contains.

```
library(ggplot2)
theme_set(theme_classic())

# Plot
g <- ggplot(mpg, aes(class, cty))
g + geom_boxplot(varwidth=T, fill="plum") +
  labs(title="Box plot",
        subtitle="City Mileage grouped by Class of vehicle",
        caption="Source: mpg",
        x="Class of Vehicle",
        y="City Mileage")
```



Time Series Plot From Wide Data Format: Data in Multiple Columns of Dataframe

As noted in the part 2 of this tutorial, whenever your plot's geom (like points, lines, bars, etc) changes the fill, size, col, shape or stroke based on another column, a legend is automatically drawn.

But if you are creating a time series (or even other types of plots) from a wide data format, you have to draw each line manually by calling `geom_line()` once for every line. So, a legend will not be drawn by default.

However, having a legend would still be nice. This can be done using the `scale_aesthetic_manual()` format of functions (like, `scale_color_manual()` if only the color of your lines change). Using this function, you can give a legend title with the name argument, tell what color the legend should take with the values argument and also set the legend labels.

Even though the below plot looks exactly like the previous one, the approach to construct this is different.

You might wonder why I used this function in previous example for long data format as well. Note that, in previous example, it was used to change the color of the line only. Without `scale_color_manual()`, you would still have got a legend, but the lines would be of a different (default) color. But in current example, without `scale_color_manual()`, you wouldn't even have a legend. Try it out!

```
library(ggplot2)
library(lubridate)

##
## Attaching package: 'lubridate'

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

theme_set(theme_bw())

df <- economics[, c("date", "psavert", "uempmed")]
df <- df[lubridate::year(df$date) %in% c(1967:1981), ]

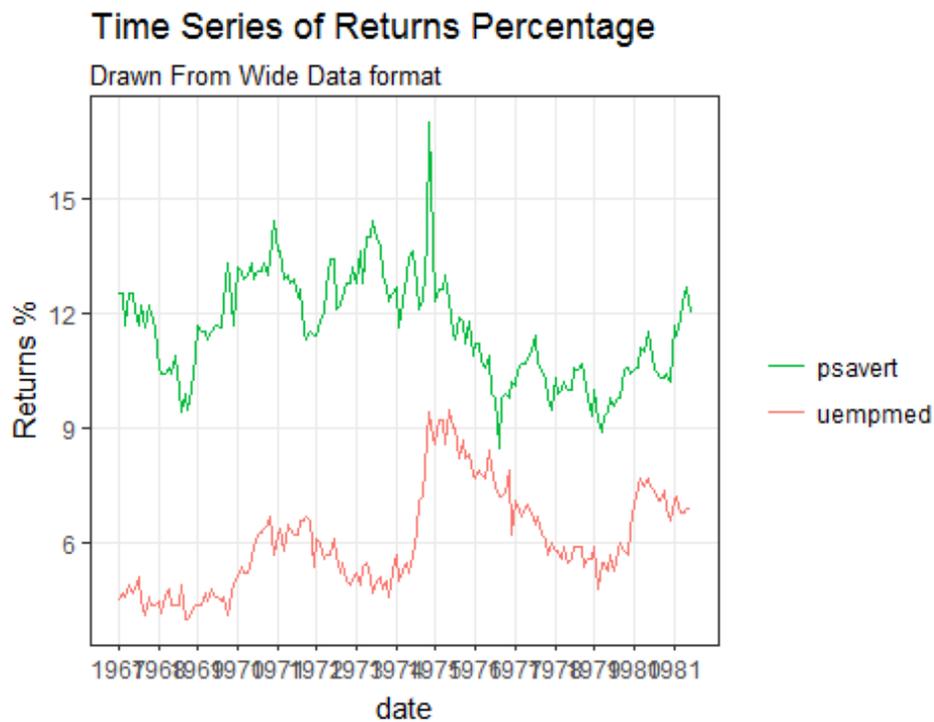
# Labels and breaks for X axis text
brks <- df$date[seq(1, length(df$date), 12)]
lbls <- lubridate::year(brks)

# plot
ggplot(df, aes(x=date)) +
  geom_line(aes(y=psavert, col="psavert")) +
  geom_line(aes(y=uempmed, col="uempmed")) +
  labs(title="Time Series of Returns Percentage",
        subtitle="Drawn From Wide Data format",
        caption="Source: Economics", y="Returns %") + # title and caption
```

```

scale_x_date(labels = lbls, breaks = brks) + # change to monthly ticks and labels
scale_color_manual(name="",
                   values = c("psavert"="#00ba38", "uempmed"="#f8766d")) + # line color
theme(panel.grid.minor = element_blank()) # turn off minor grid

```



source: <http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html>