Programming in Data science
Introduction to ggplot2

Asad Waqar Malik

Dept. of Information Systems
Faculty of Computer Science and Information Technology
University of Malaya
Malaysia

Nov 21, 2019

Lecture Outline

1. Ggplot - Basic Syntax
2. Bar Chart
3. Line chart
4. Facets
5. Explore online websites

Outline

Understanding the Ggplot Syntax

- 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.
- Another 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.

Understanding the Ggplot Syntax

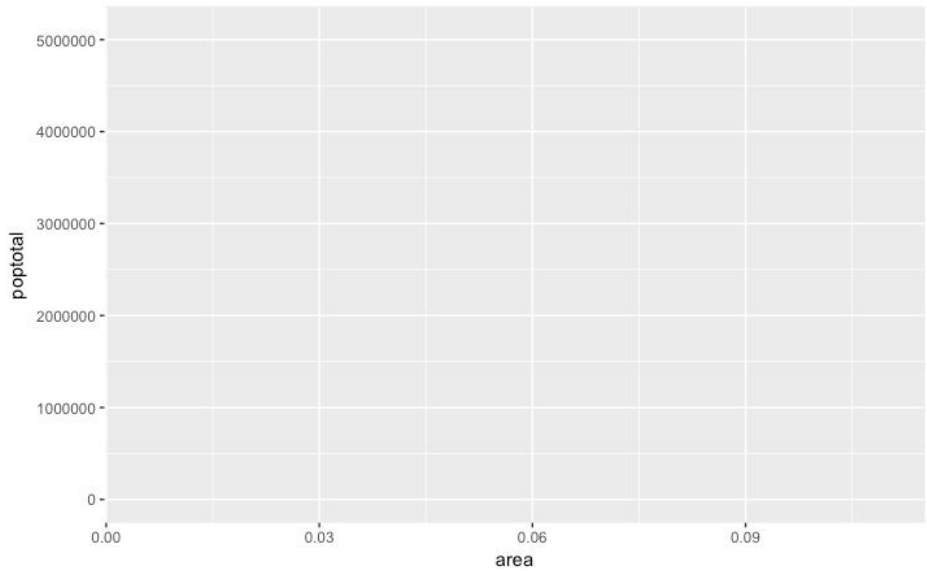
- Let's initialize a basic ggplot based on the midwest dataset.

Code

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

Note: Manually download the dataset from the given link.

Understanding the Ggplot Syntax



- 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.
- Also note that aes() function is used to specify the X and Y axes. Any information part of the source dataframe has to be specified inside the aes() function.

Scatterplot

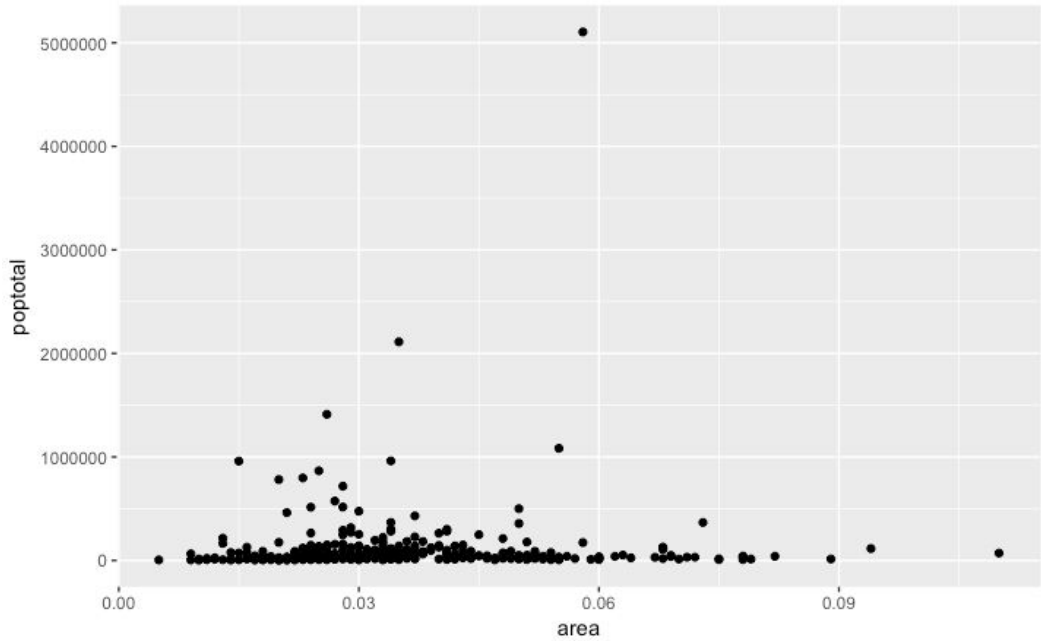
- Let's make a scatterplot on top of the blank ggplot by adding points using a geom layer called geom_point.

Sample

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

Scatterplot

- The output is:



- Thescatterplot, 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.

Scatterplot

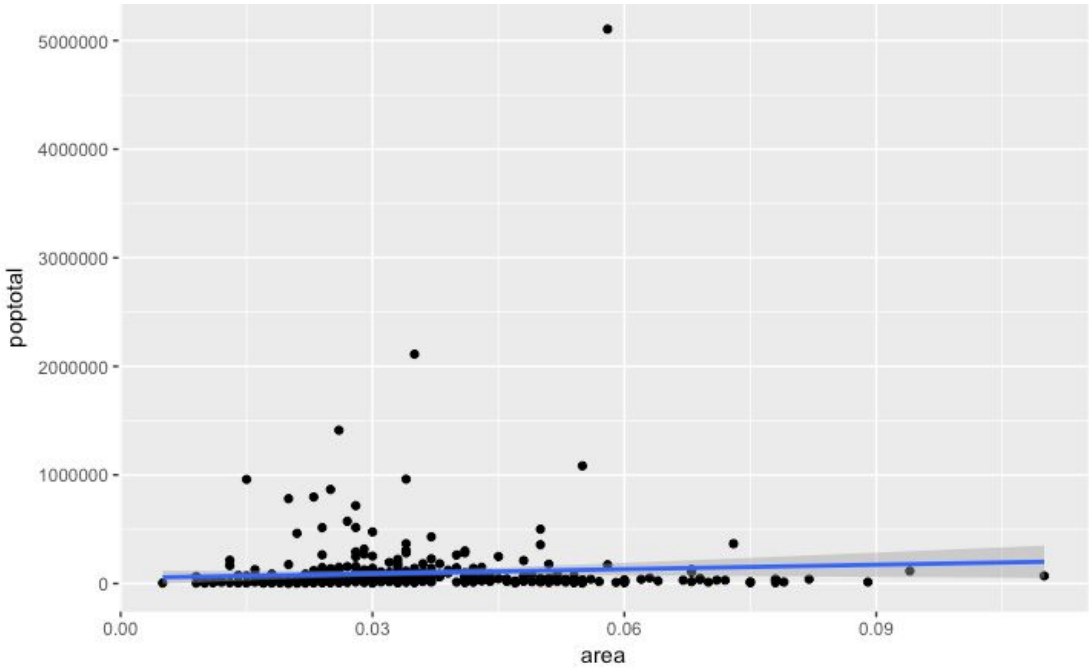
- Like geom_point(), there are many such geom layers which you can explore by yourself. For now, let's just add a smoothing layer using geom_smooth(method='lm'). Since the method is set as lm (short for linear model), it draws the line of best fit.

Sample

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

Scatterplot

- 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).



Adjusting the X and Y axis limits by deleting the points outside the range

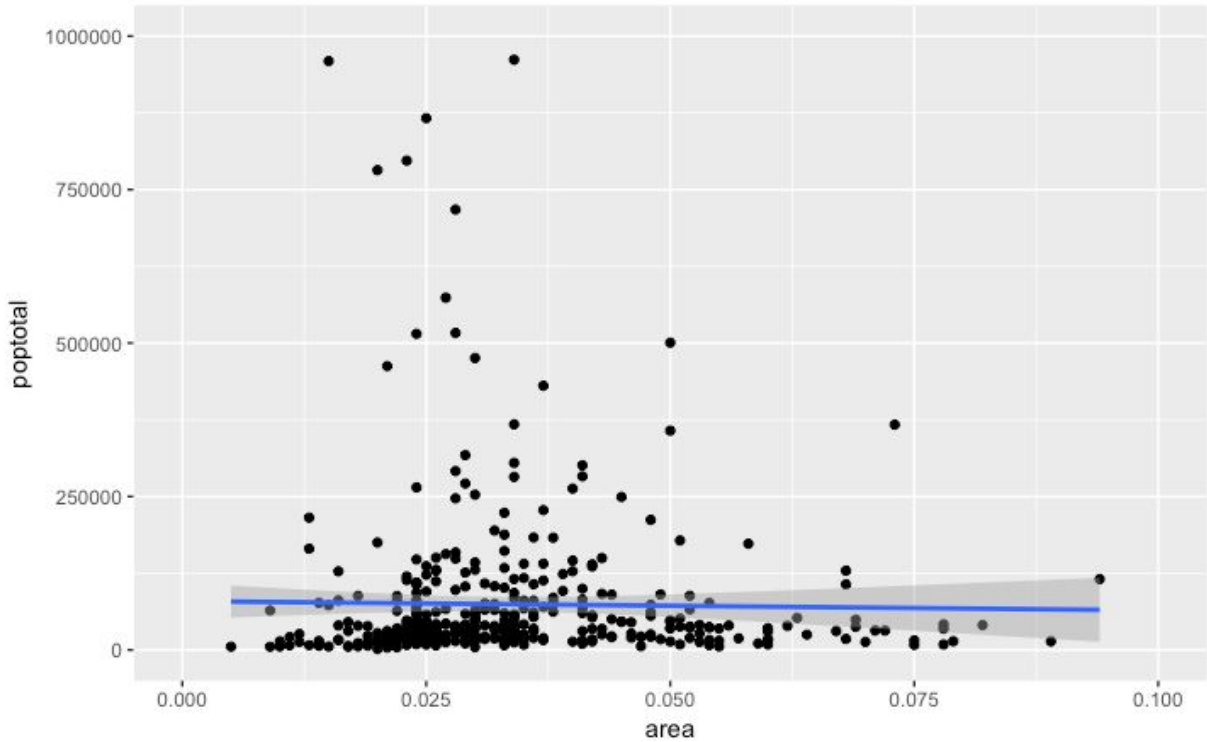
- This will change the lines of best fit or smoothing lines as compared to the original data.
- This can be done by xlim() and ylim(). You can pass a numeric vector of length 2 (with max and min values) or just the max and min values itself.

Code

```
1  library(ggplot2)
2  g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom_point↩
       () + geom_smooth(method="lm")  # set se=FALSE to ↩
       turnoff confidence bands
3  # Delete the points outside the limits
4  g + xlim(c(0, 0.1)) + ylim(c(0, 1000000))   # deletes ↩
       points
5  # g + xlim(0, 0.1) + ylim(0, 1000000)   # deletes points
```

Adjusting the X and Y axis limits by deleting the points outside the range

- Output



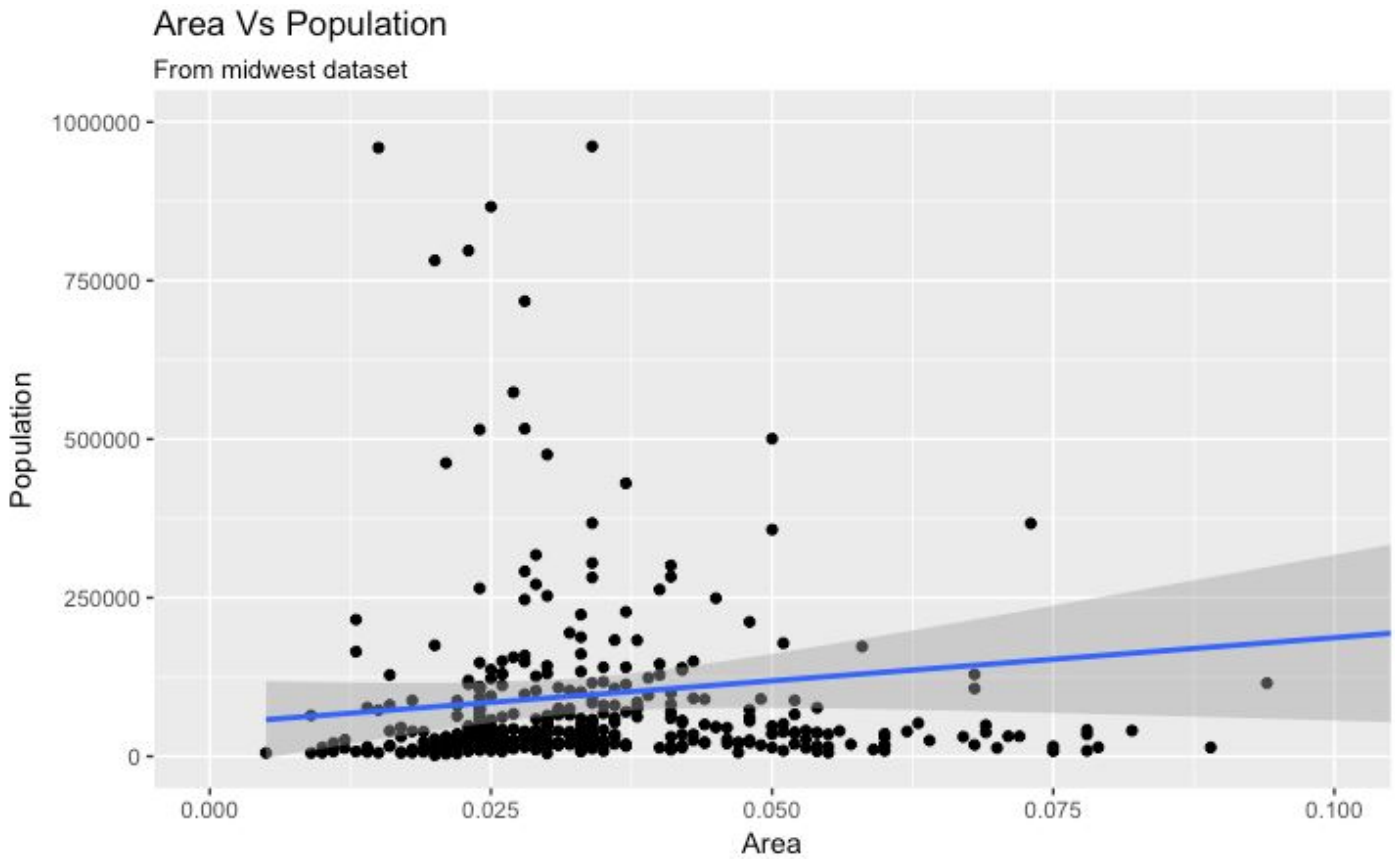
Add Title and Axis Labels

Code

```
1  library(ggplot2)
2  g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom_point() + geom_smooth(method="lm")  # set se=FALSE to turnoff confidence bands
3
4  g1 <- g + coord_cartesian(xlim=c(0,0.1), ylim=c(0, 1000000))  # zooms in
5
6  # Add Title and Labels
7  g1 + labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population", x="Area", caption="Midwest Demographics")
8
9  # or
10
11 g1 + ggtitle("Area Vs Population", subtitle="From midwest dataset") + xlab("Area") + ylab("Population")
```

Add Title and Axis Labels

- Output



Add Title and Axis Labels

Here is the full function call.

Code

```
# 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")
```

Change the Color and Size of Points

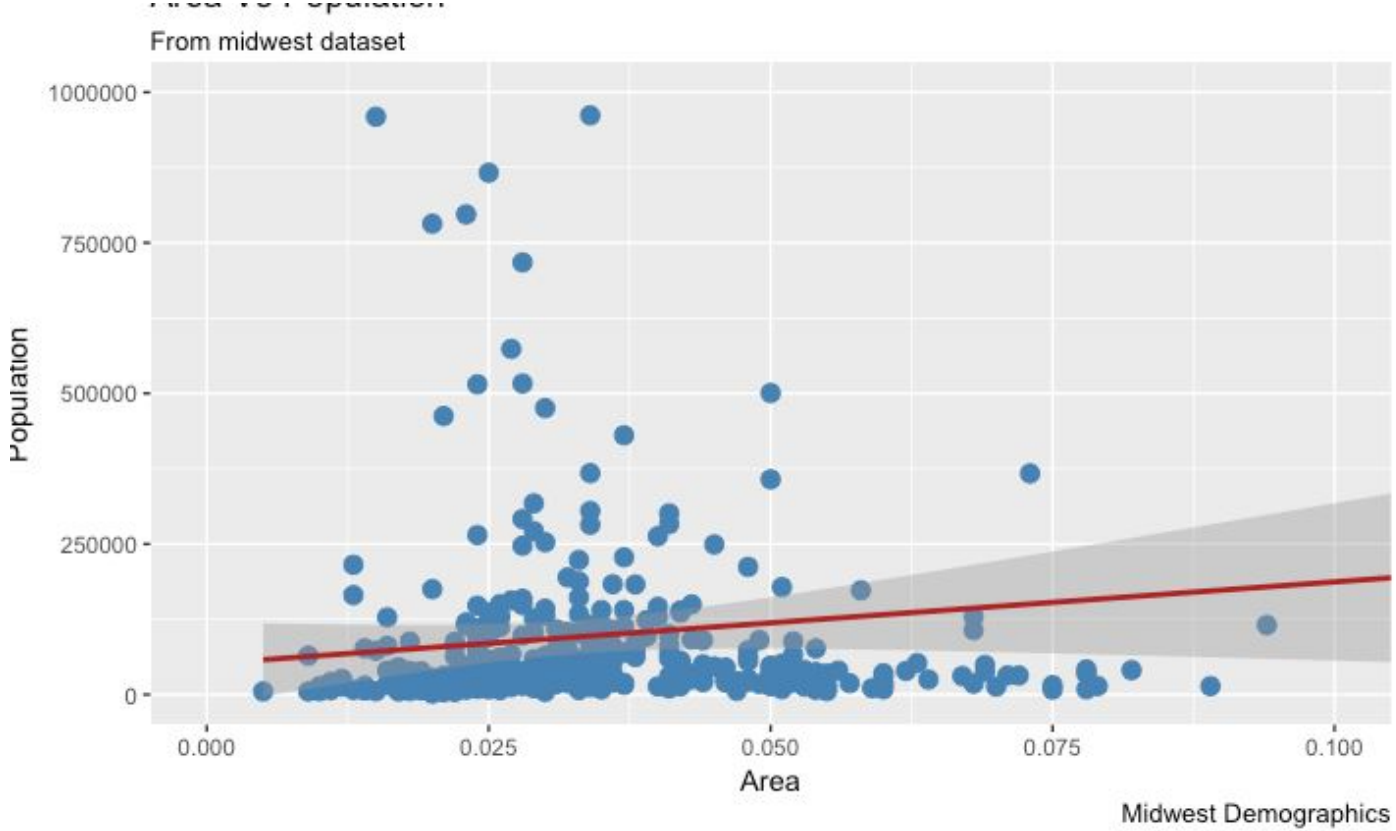
- We can change the aesthetics of a geom layer by modifying the respective geoms. Let's change the color of the points and the line to a static value.

Code

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

Change the Color and Size of Points

- Output



Change the Color To Reflect Categories

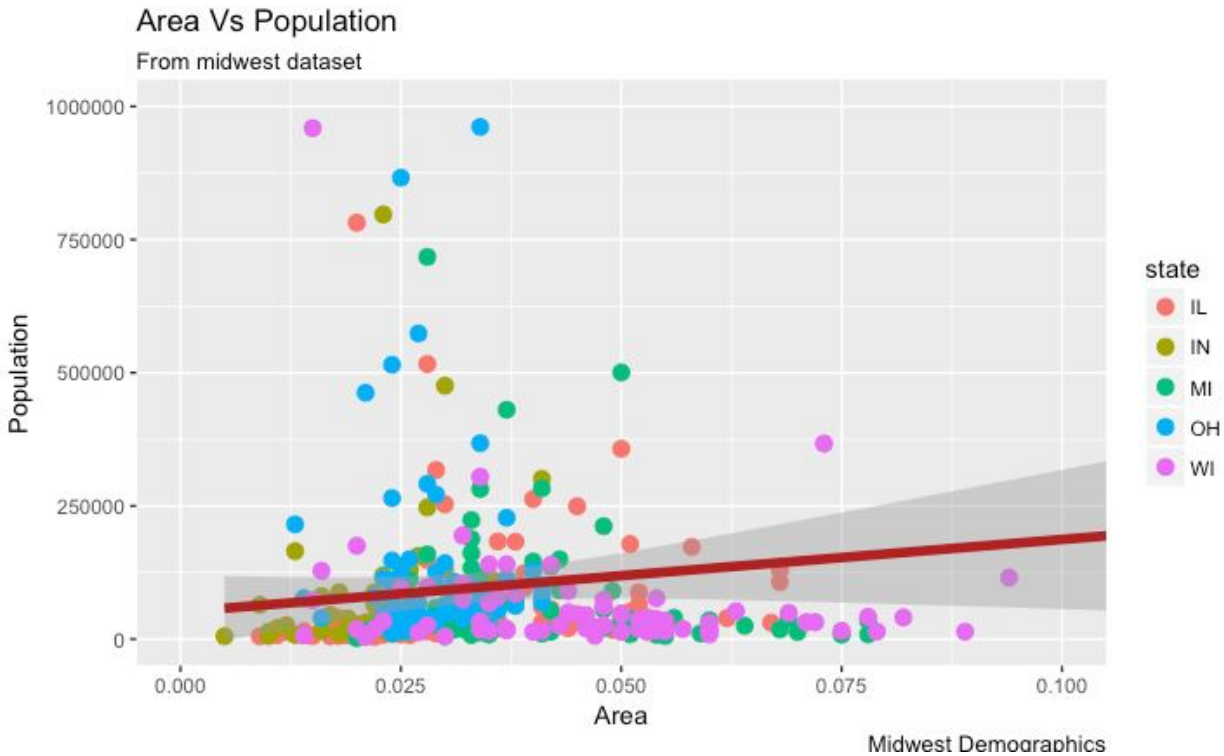
- Suppose if we want the color to change based on another column in the source dataset (midwest), it must be specified inside the aes() function.

Code

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

Change the Color To Reflect Categories

- Output



- Now each point is colored based on the state it belongs 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.

Change the Color To Reflect Categories

- 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.

Code

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

Change the X-axis texts and ticks location

Change the X and Y axis text and its location. This involves two aspects: breaks and labels.

- Step 1: Set the breaks – The breaks should be of the same scale as the X axis variable. Use scale_x_continuous because, the X-axis variable is a continuous variable. Had it been a date variable, scale_x_date could be used. Like scale_x_continuous() an equivalent scale_y_continuous() is available for Y axis.

Change the X-axis texts and ticks location

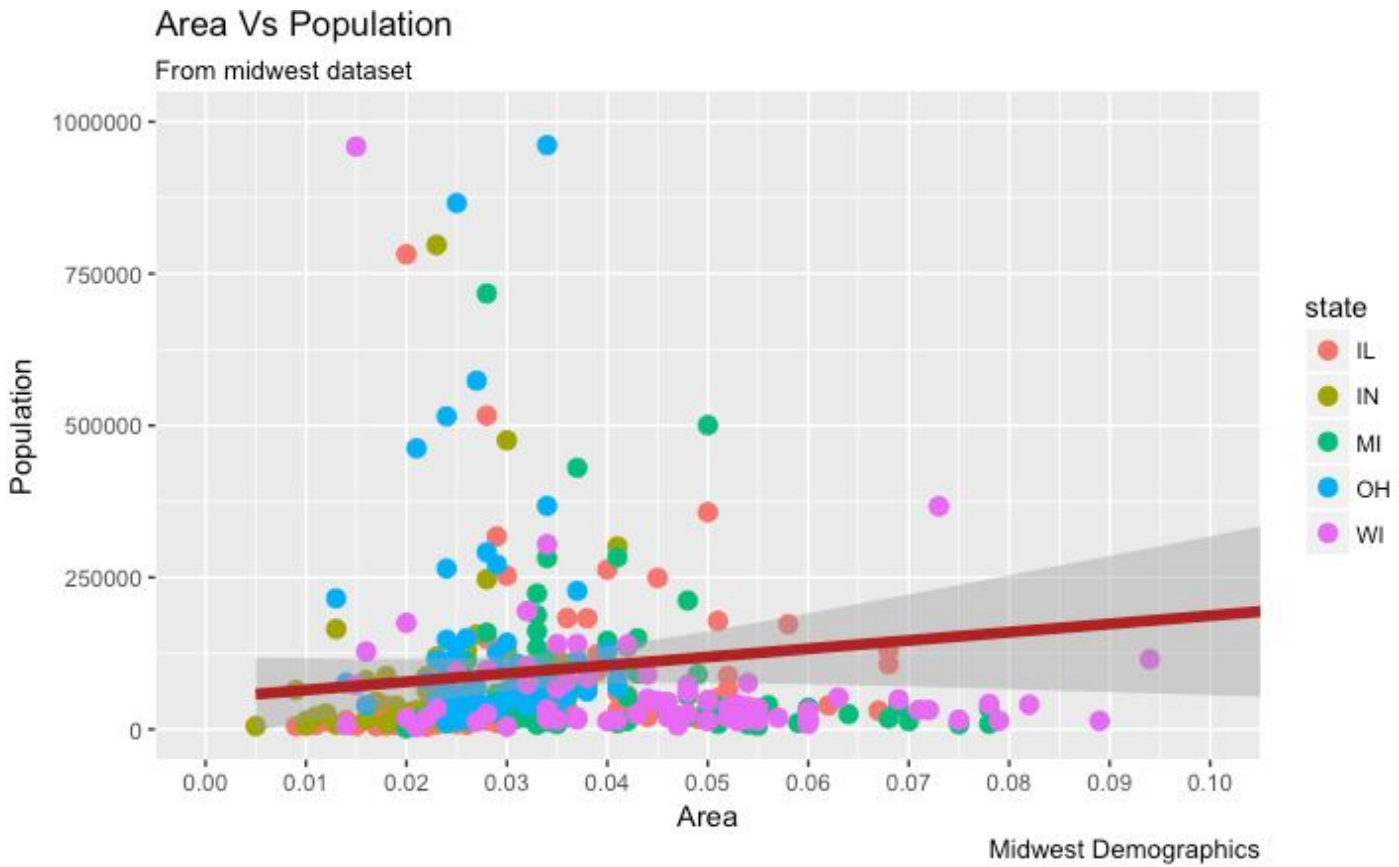
Code

```
# 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
gg + scale_x_continuous(breaks=seq(0, 0.1, 0.01))
```

Change the X-axis texts and ticks location

- Output



Change the X-axis texts and ticks location

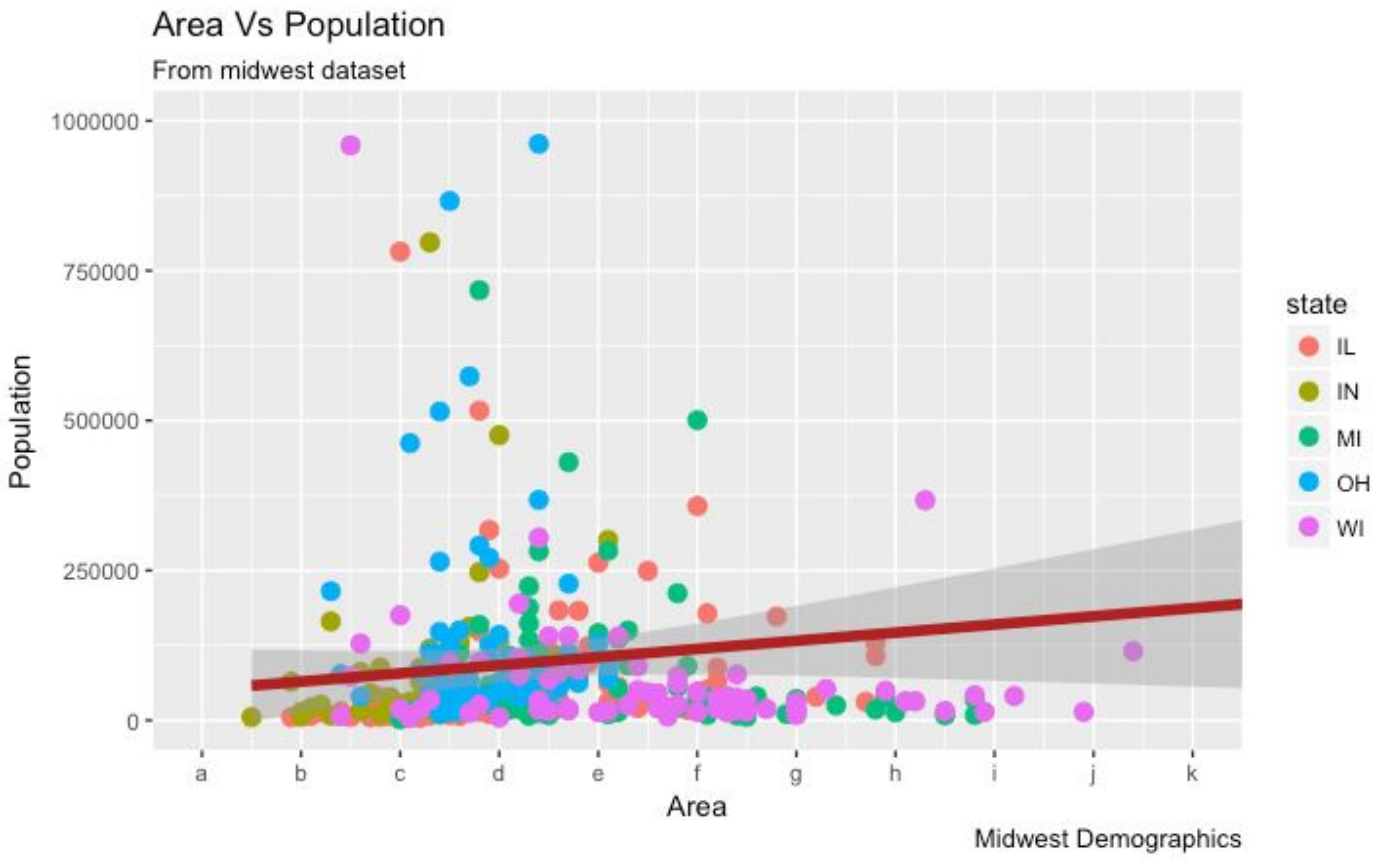
- Step 2: Change the labels You can optionally change the labels at the axis ticks. labels take a vector of the same length as breaks.

Code

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

Change the X-axis texts and ticks location

- Output



Look and feel

- Most of the requirements related to look and feel can be achieved using the theme() function. It accepts a large number of arguments. Type ?theme in the R console and see for yourself.

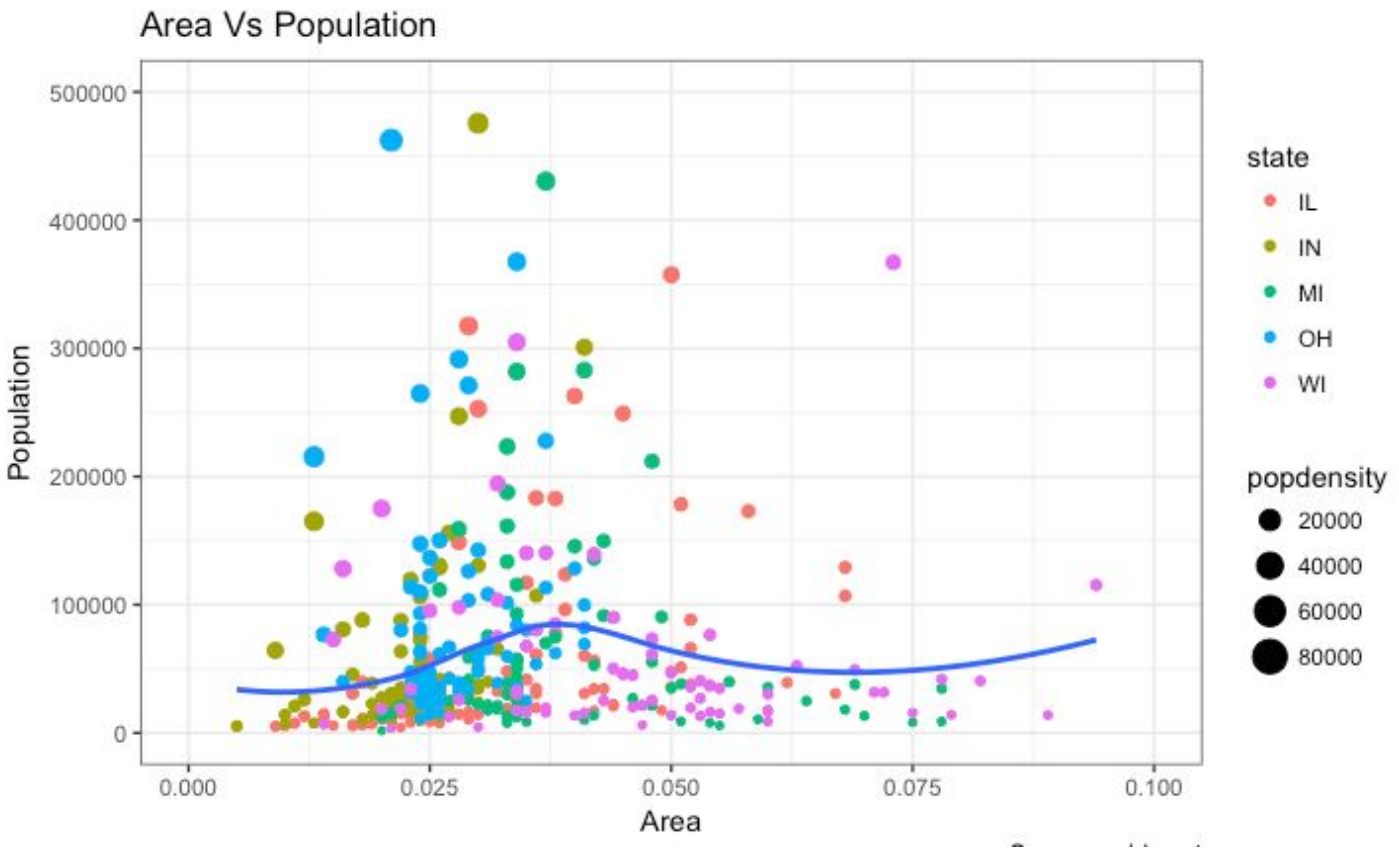
Code

```
# Add plot components ---------------------------------
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point(aes(col=state, size=popdensity)) +
  geom_smooth(method="loess", se=F) + xlim(c(0, 0.1)) + ylim(c(0, 500000)) +
  labs(title="Area Vs Population", y="Population", x="Area", caption="Source: midwest")

# Call plot ---------------------------------------------
plot(gg)
```

Look and feel

- Output



Outline

Make Bar Chart with ggplot2

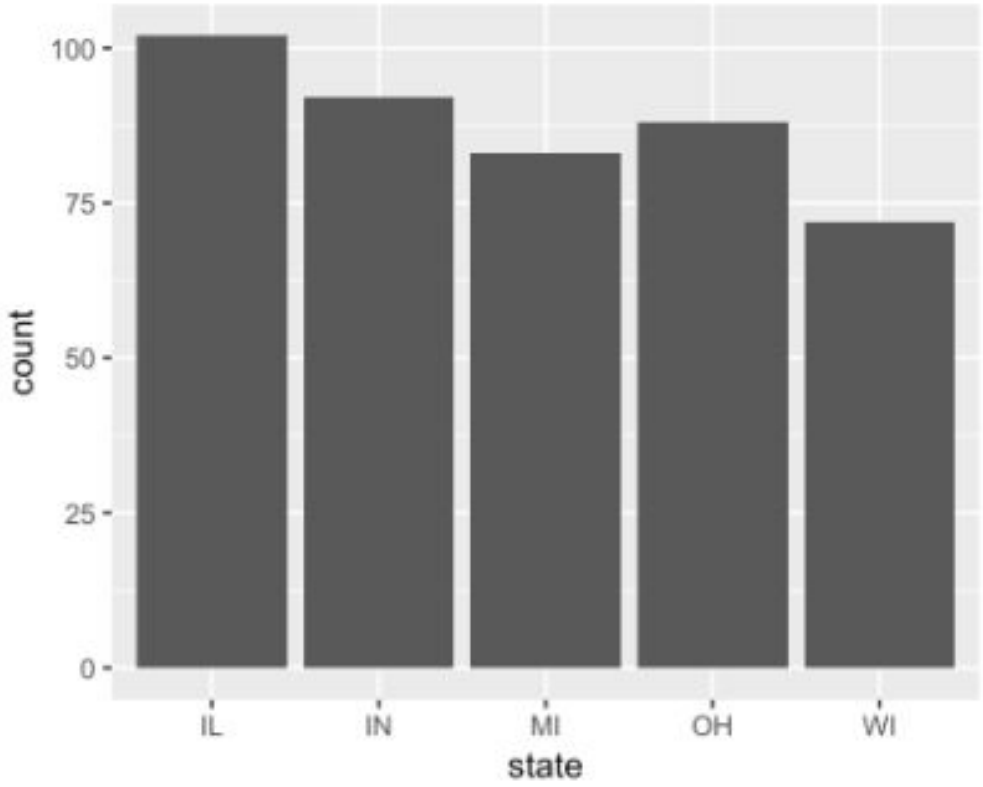
- First, here's the code. You can paste this into RStudio and run it.
- The ggplot() function initiates plotting. Then immediately inside the ggplot() function, the code data = midwest indicates that we'll be plotting data from the midwest dataframe.
- On the second line of code, the geom_bar() function indicates that we'll be drawing bars.
- Then, take a look at the aes() function. As always, the aes() function tells ggplot which variables to plot on the chart.

Code

```
1  ggplot(data = midwest, aes(x = state)) +
2    geom_bar()
```

Make Bar Chart with ggplot2

- Output



Make Bar Chart with STATE == IDENTITY

- There's also another way to make a bar chart. It's possible to map a variable to the y axis too, so the length of the bar correspond to the value of the y axis variable (instead of the count).
- To show this, create a new dataset that calculates the total population by state. In order to create this summarised dataset, use the group_by() and the summarise() functions from dplyr.
- Ultimately, this code produces a summarised dataset that contains two variables: state and total_population.

Code

```
1    midwest_populations <- midwest %>%
2    group_by(state) %>%
3    summarise(total_population = sum(poptotal))
```

Make Bar Chart with STATE == IDENTITY

- The important detail here is that there is one observation for every state. This is different from the original midwest dataset, where there was one record for every county, and therefore multiple records for every state.
- This is relevant, because now we can map the state variable to the x axis and the total_population variable to the y axis.

Code

```
1    ggplot(midwest_populations, aes(x = state, y = total_↩
         population)) +
2    geom_bar(stat = "identity")
```

Histogram

For histrogram explore:
`www.sharpsightlabs.com/blog/ggplot-histogram/`.

Outline

Line Chart with ggplot2

- To make this line chart with ggplot2, use the following command:

Code

```
1    geom_line()
```

Outline

Facets

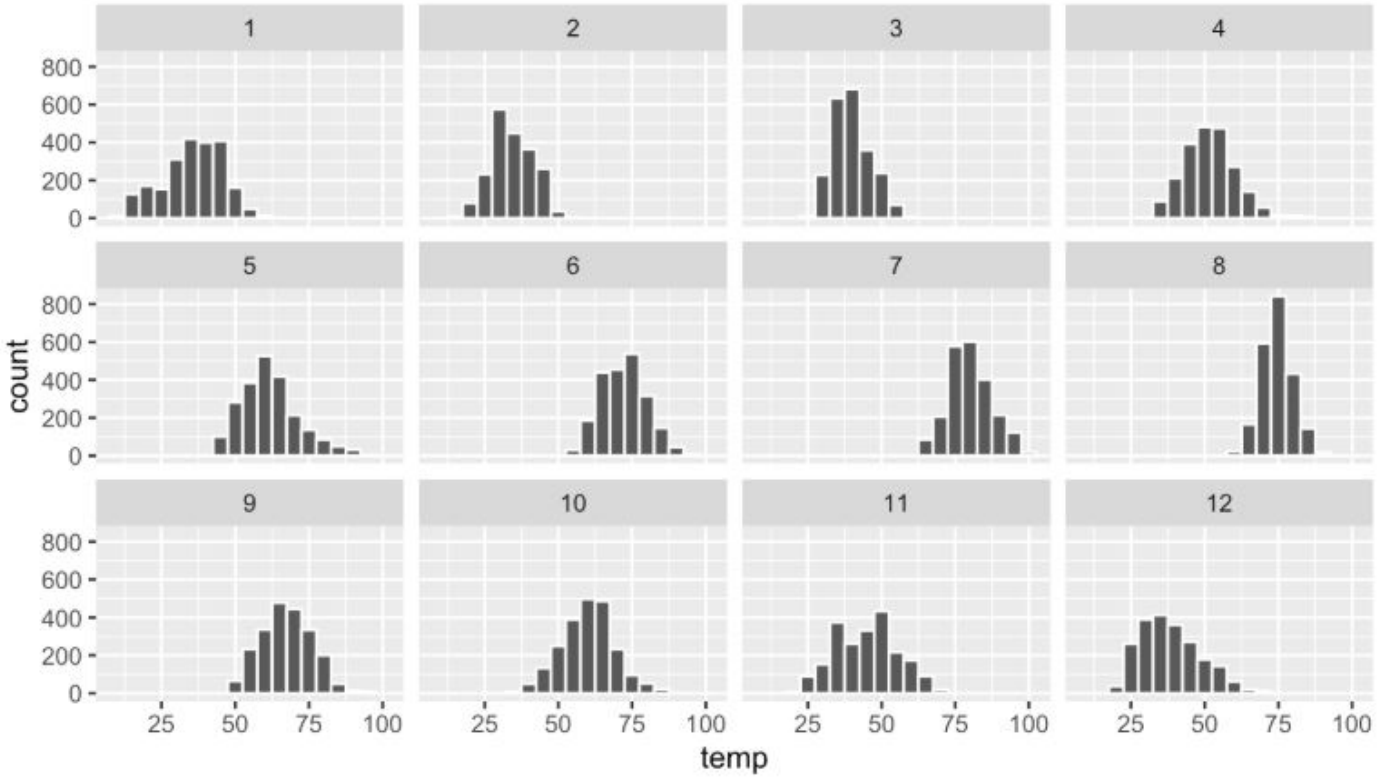
- Lets introduce a new concept called faceting. Faceting is used when we'd like to split a particular visualization by the values of another variable. This will create multiple copies of the same type of plot with matching x and y axes, but whose content will differ.
- For example, suppose we were interested in looking at how the histogram of hourly temperature recordings at the three NYC airports differed in each month. We could "split" this histogram by the 12 possible months in a given year. In other words, we would plot histograms of temp for each month separately. We do this by adding facet_wrap(month) layer.

Code

```
1  ggplot(data = weather, mapping = aes(x = temp)) +
2    geom_histogram(binwidth = 5, color = "white") +
3    facet_wrap(~ month)
```

Facets

- Output



Facets

- We can also specify the number of rows and columns in the grid by using the nrow and ncol arguments inside of facet_wrap(). For example, say we would like our faceted histogram to have 4 rows instead of 3. We simply add a nrow = 4 argument to facet_wrap(month)

Code

```
1    ggplot(data = weather, mapping = aes(x = temp)) +
2    geom_histogram(binwidth = 5, color = "white") +
3    facet_wrap(~ month, nrow = 4)
```

Outline

Further explore, online websites

- www.sharpsightlabs.com/blog/ggplot-histogram/
- Line graph: www.tutorialspoint.com/r/r_line_graphs.htm