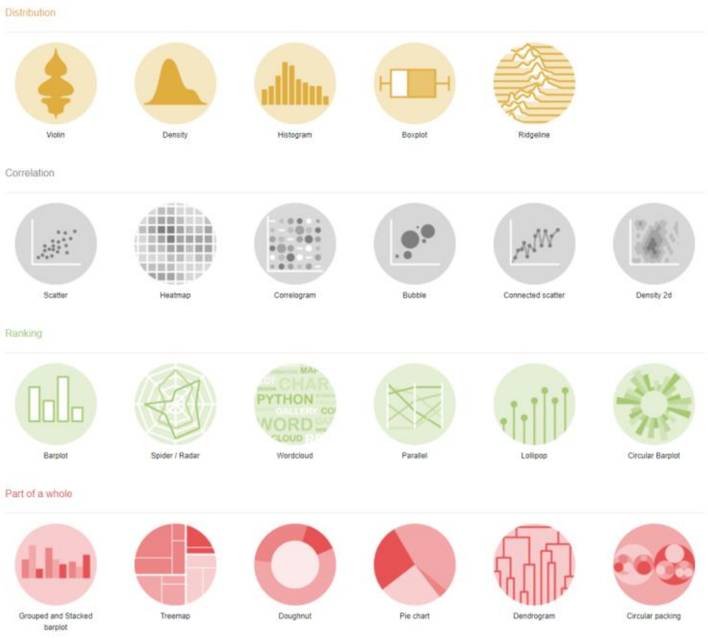


R Graph: ggplot2 Part I

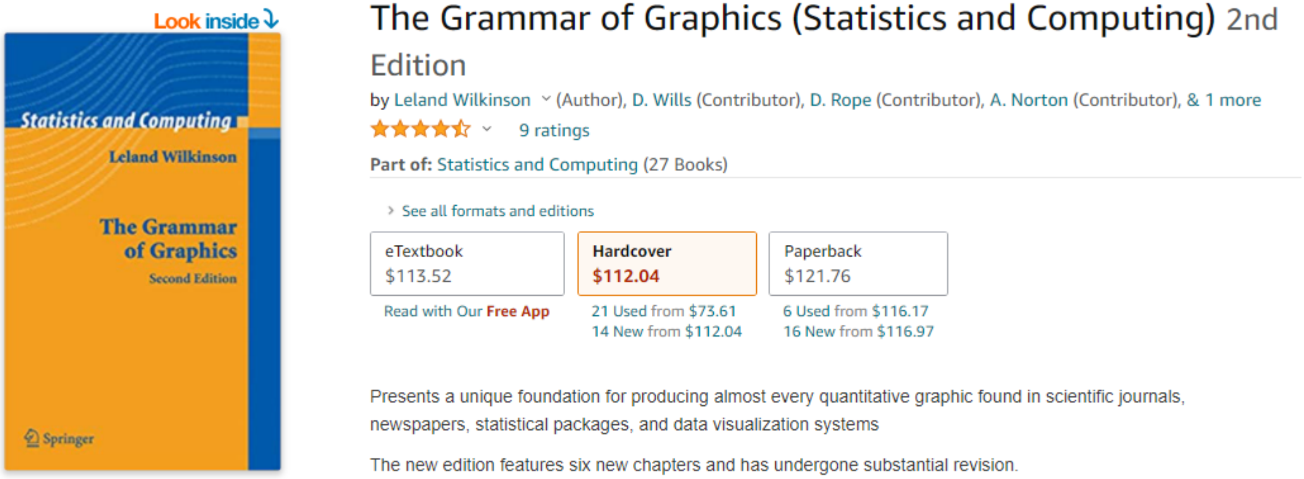
Pilsung Kang

School of Industrial Management Engineering

Korea University

ggplot2( )

- ggplot2
  - ✓ A system for declaratively creating graphics, based on The Grammar of Graphics
  - ✓ You provide the data , tell ggplot2
    - ▪ how to map variables to aesthetics
    - ▪ what graphical primitives to use
  - ✓ and the ggplot2 takes care of details



The Grammar of Graphics (Statistics and Computing) 2nd Edition

by Leland Wilkinson (Author), D. Wills (Contributor), D. Rope (Contributor), A. Norton (Contributor), & 1 more

★★★★☆ 9 ratings

Part of: Statistics and Computing (27 Books)

› See all formats and editions

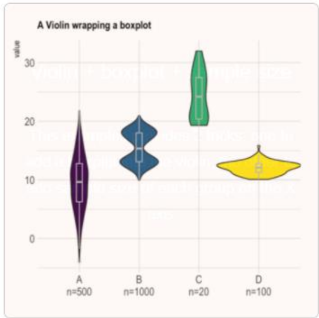
eTextbook $113.52	**Hardcover** **$112.04**	Paperback $121.76
Read with Our **Free App**	21 Used from $73.61 14 New from $112.04	6 Used from $116.17 16 New from $116.97

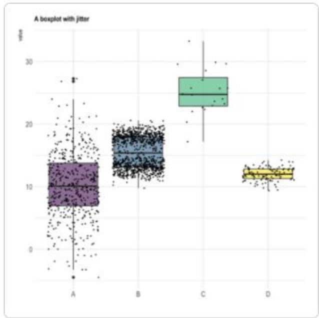
Presents a unique foundation for producing almost every quantitative graphic found in scientific journals, newspapers, statistical packages, and data visualization systems

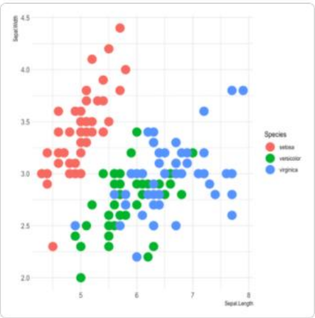
The new edition features six new chapters and has undergone substantial revision.

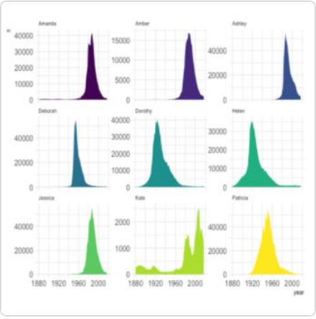
ggplot2( )

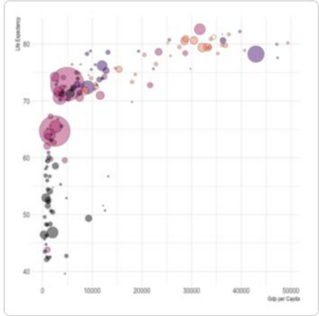
- Graphs you can create with ggplot2

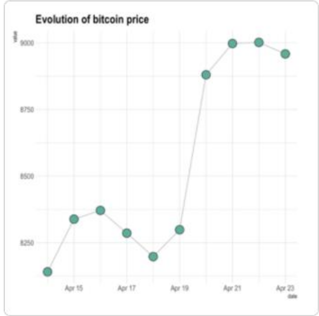


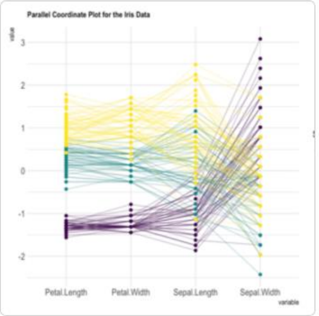


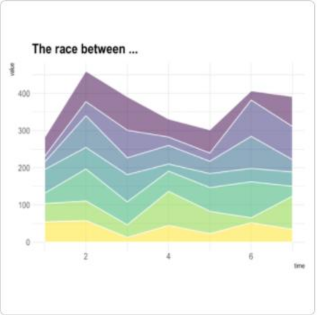


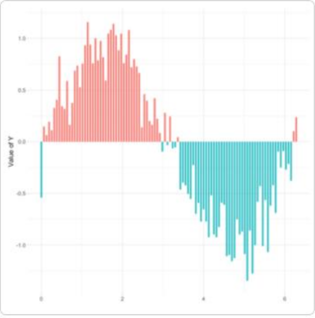


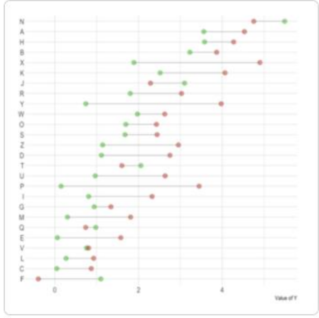


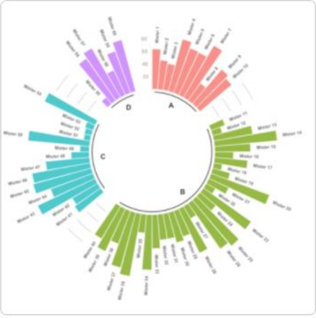










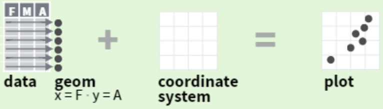


https://www.r-graph-gallery.com/ggplot2-package.html

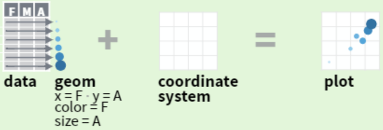
Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

ggplot (data = <DATA>) +
<GEOM_FUNCTION> (mapping = aes(**<MAPPINGS>**),
stat = **<STAT>**, position = **<POSITION>**) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>

required

Not required, sensible defaults supplied

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings data geom

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1),curvature=1) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend=lat+1, xend=long+1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)) x, y, alpha, color, fill, linetype, size, weight

discrete

d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"), x, y, alpha, color, linetype, size

e + geom_smooth(method = lm), x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

f <- ggplot(mpg, aes(class, hwy))

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group

f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

g <- ggplot(diamonds, aes(cut, color))

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map)
+ expand_limits(x = map$long, y = map$lat), map_id, alpha, color, fill, linetype, size

THREE VARIABLES

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

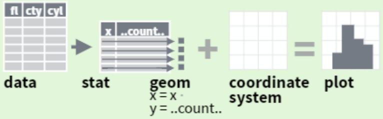
l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width

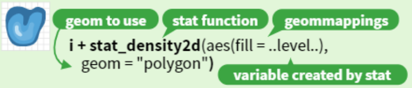
RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **http://ggplot2.tidyverse.org** • ggplot2 3.1.0 • Updated: 2018-12

Stats An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.



i + stat_density2d(aes(fill = ..level..), geom = "polygon")

c + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y,** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = “gaussian")
x, y, | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins=30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5) **x, y** | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = “area") **x, y** | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se=T, level=0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) **sample, x, y** | ..sample.., ..theoretical..

e + stat_sum() x, y, size | ..n.., ..prop..

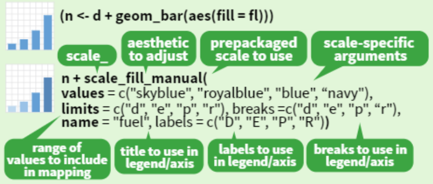
e + stat_summary(fun.data = "mean_cl_boot")

h + stat_summary_bin(fun.y = "mean", geom = "bar")

e + stat_unique()

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.


scale_
aesthetic to adjust
prepackaged scale to use
scale-specific arguments
range of values to include in mapping
title to use in legend/axis
labels to use in legend/axis
breaks to use in legend/axis

(n <- d + geom_bar(aes(fill = fl)))

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", “navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", “r"),
name = "fuel", labels = c("D", "E", "P", "R"))

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - map cont’ values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

COLOR AND FILL SCALES (DISCRETE)

n <- d + geom_bar(aes(fill = fl))

n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

o + scale_fill_distiller(palette = "Blues")

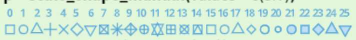
o + scale_fill_gradient(low="red", high="yellow")

o + scale_fill_gradient2(low="red", high=“blue", mid = "white", midpoint = 25)

o + scale_fill_gradientn(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7))



p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(ytrans = “sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation=c(41, -74, 0))projection, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height

e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

e + geom_label(position = "nudge")
Nudge labels away from points

s + geom_bar(position = "stack")
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments
s + geom_bar(position = position_dodge(width = 1))

Themes

r + theme_bw()
White background with grid lines

r + theme_gray()
Grey background (default theme)

r + theme_dark()
dark for contrast

r + theme_classic()

r + theme_light()

r + theme_linedraw()

r + theme_minimal()
Minimal themes

r + theme_void()
Empty theme

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(cols = vars(fl))
facet into columns based on fl

t + facet_grid(rows = vars(year))
facet into rows based on year

t + facet_grid(rows = vars(year), cols = vars(fl))
facet into both rows and columns

t + facet_wrap(vars(fl))
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(rows = vars(drv), cols = vars(fl),
scales = "free")
x and y axis limits adjust to individual facets
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(cols = vars(fl), labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

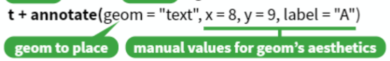
t + facet_grid(rows = vars(fl),
labeller = label_bquote(alpha ^ .(fl)))

α^c	α^d	α^e	α^p	α^r

Labels

t + labs(**x** = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
<AES> = "New <AES> legend title")

Use scale functions to update legend labels



t + annotate(geom = "text", x = 8, y = 9, label = "A")

Legends

n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"

n + guides(fill = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred)

t + coord_cartesian(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)

t + xlim(0, 100) **+ ylim**(10, 20)

t + scale_x_continuous(limits = c(0, 100)) **+ scale_y_continuous**(limits = c(0, 100))

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **http://ggplot2.tidyverse.org** • ggplot2 3.1.0 • Updated: 2018-12

ggplot2( )

- Installation

```
# The easiest way to get ggplot2 is to install the whole tidyverse:
install.packages("tidyverse")

# Alternatively, install just ggplot2:
install.packages("ggplot2")

# Or the development version from GitHub:
# install.packages("devtools")
devtools::install_github("tidyverse/ggplot2")
```

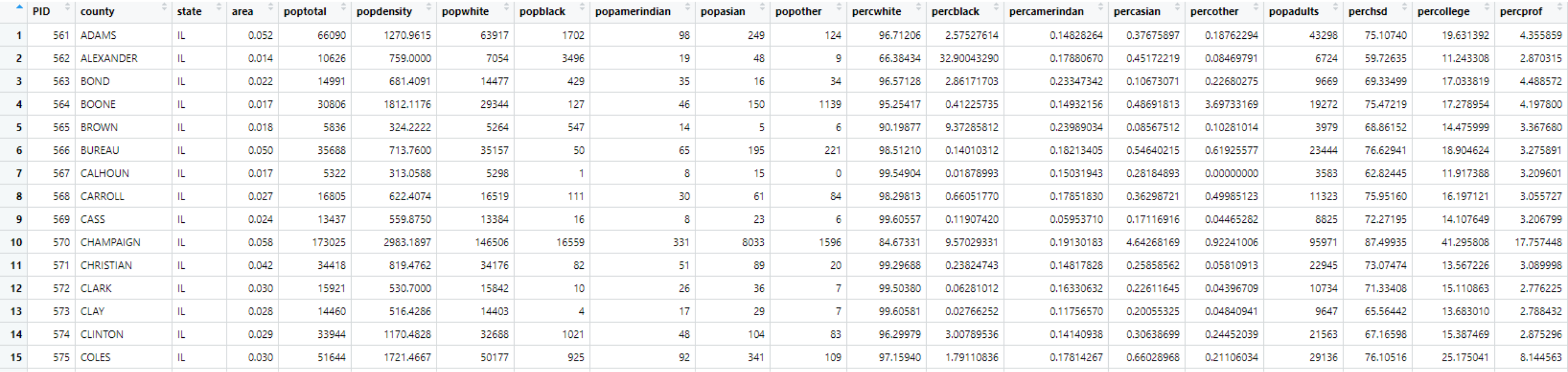
ggplot2( )

- Understanding the ggplot2( ) syntax
  - ✓ The main difference between the ggplot2 and the base graphics is that ggplot2 works with dataframes and not individual vectors
  - ✓ With ggplot2, we can keep enhancing the plot by adding more layers and themes to an existing plot

ggplot2( )

- Initialize a basic ggplot

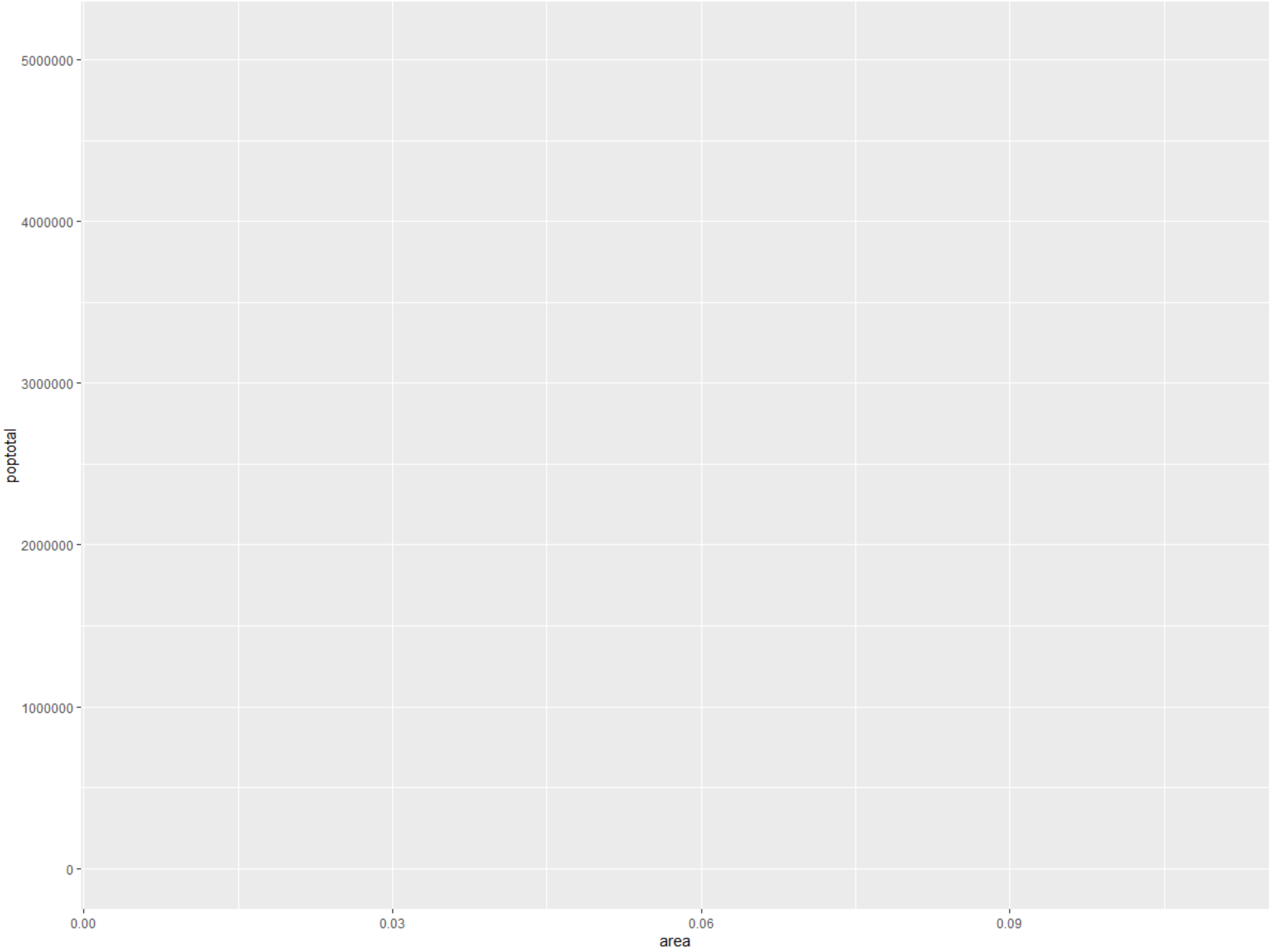
```r
# Setup options(scipen=999) # turn off scientific notation like 1e+06
library(ggplot2)
data("midwest", package = "ggplot2") # load the data
View(midwest)

# Init ggplot
ggplot(Midwest, aes(x=area, y=poptotal))
# area and poptotal are columns in 'midwest'
```



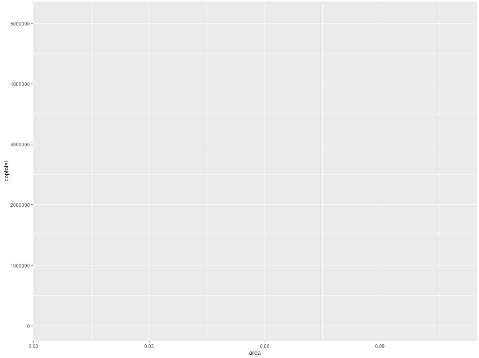
	PID	county	state	area	poptotal	popdensity	popwhite	popblack	popamerindian	popasian	popother	percwhite	percblack	percamerindan	percasian	percother	popadults	perchsd	percollege	percprof
1	561	ADAMS	IL	0.052	66090	1270.9615	63917	1702	98	249	124	96.71206	2.57527614	0.14828264	0.37675897	0.18762294	43298	75.10740	19.631392	4.355859
2	562	ALEXANDER	IL	0.014	10626	759.0000	7054	3496	19	48	9	66.38434	32.90043290	0.17880670	0.45172219	0.08469791	6724	59.72635	11.243308	2.870315
3	563	BOND	IL	0.022	14991	681.4091	14477	429	35	16	34	96.57128	2.86171703	0.23347342	0.10673071	0.22680275	9669	69.33499	17.033819	4.488572
4	564	BOONE	IL	0.017	30806	1812.1176	29344	127	46	150	1139	95.25417	0.41225735	0.14932156	0.48691813	3.69733169	19272	75.47219	17.278954	4.197800
5	565	BROWN	IL	0.018	5836	324.2222	5264	547	14	5	6	90.19877	9.37285812	0.23989034	0.08567512	0.10281014	3979	68.86152	14.475999	3.367680
6	566	BUREAU	IL	0.050	35688	713.7600	35157	50	65	195	221	98.51210	0.14010312	0.18213405	0.54640215	0.61925577	23444	76.62941	18.904624	3.275891
7	567	CALHOUN	IL	0.017	5322	313.0588	5298	1	8	15	0	99.54904	0.01878993	0.15031943	0.28184893	0.00000000	3583	62.82445	11.917388	3.209601
8	568	CARROLL	IL	0.027	16805	622.4074	16519	111	30	61	84	98.29813	0.66051770	0.17851830	0.36298721	0.49985123	11323	75.95160	16.197121	3.055727
9	569	CASS	IL	0.024	13437	559.8750	13384	16	8	23	6	99.60557	0.11907420	0.05953710	0.17116916	0.04465282	8825	72.27195	14.107649	3.206799
10	570	CHAMPAIGN	IL	0.058	173025	2983.1897	146506	16559	331	8033	1596	84.67331	9.57029331	0.19130183	4.64268169	0.92241006	95971	87.49935	41.295808	17.757448
11	571	CHRISTIAN	IL	0.042	34418	819.4762	34176	82	51	89	20	99.29688	0.23824743	0.14817828	0.25858562	0.05810913	22945	73.07474	13.567226	3.089998
12	572	CLARK	IL	0.030	15921	530.7000	15842	10	26	36	7	99.50380	0.06281012	0.16330632	0.22611645	0.04396709	10734	71.33408	15.110863	2.776225
13	573	CLAY	IL	0.028	14460	516.4286	14403	4	17	29	7	99.60581	0.02766252	0.11756570	0.20055325	0.04840941	9647	65.56442	13.683010	2.788432
14	574	CLINTON	IL	0.029	33944	1170.4828	32688	1021	48	104	83	96.29979	3.00789536	0.14140938	0.30638699	0.24452039	21563	67.16598	15.387469	2.875296
15	575	COLES	IL	0.030	51644	1721.4667	50177	925	92	341	109	97.15940	1.79110836	0.17814267	0.66028968	0.21106034	29136	76.10516	25.175041	8.144563

ggplot2( )

- Initialize a basic ggplot



ggplot2( )

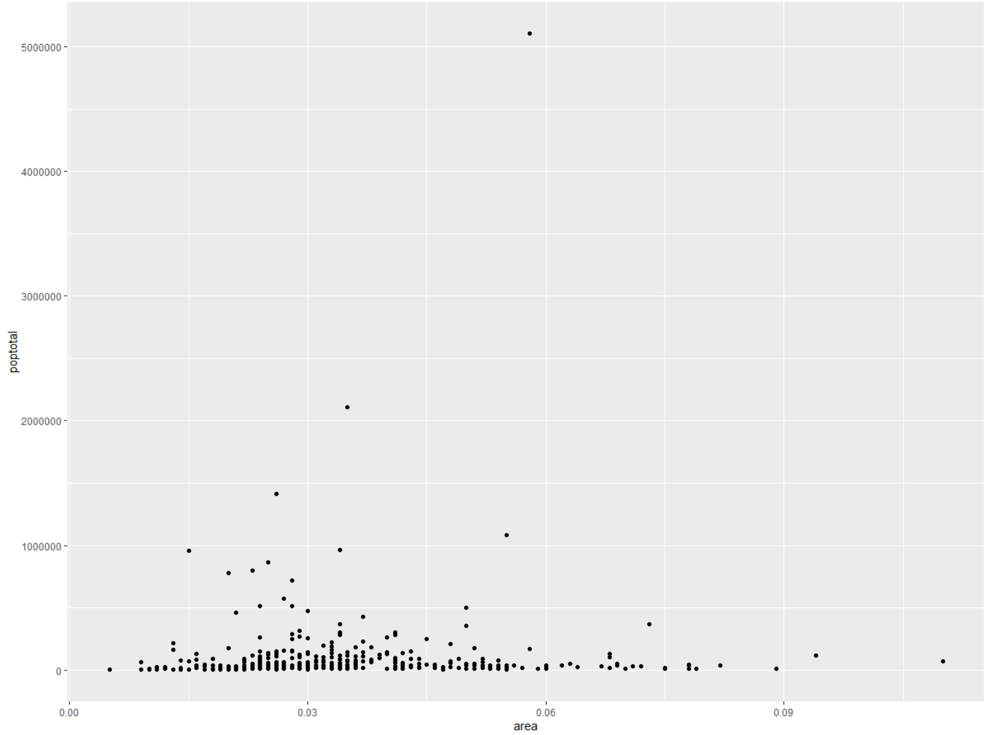


- Initialize a basic ggplot
  - ✓ A blank ggplot is drawn
  - ✓ Even though the x and y are specified, there are no points or lines in it
  - ✓ This is because, ggplot doesn't assume that you meant a scatterplot or a line chart to be drawn
  - ✓ We have only told ggplot what dataset to use and what columns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

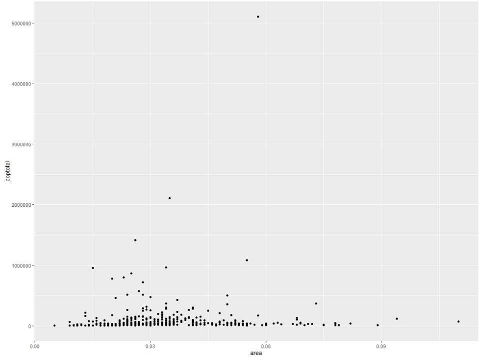
ggplot2( )

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

```
# 2. How to make a simple scatterplot
ggplot(midwest, aes(x=area, y=poptotal)) + geom_point()
```



ggplot2( )



- A simple scatterplot
  - ✓ 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

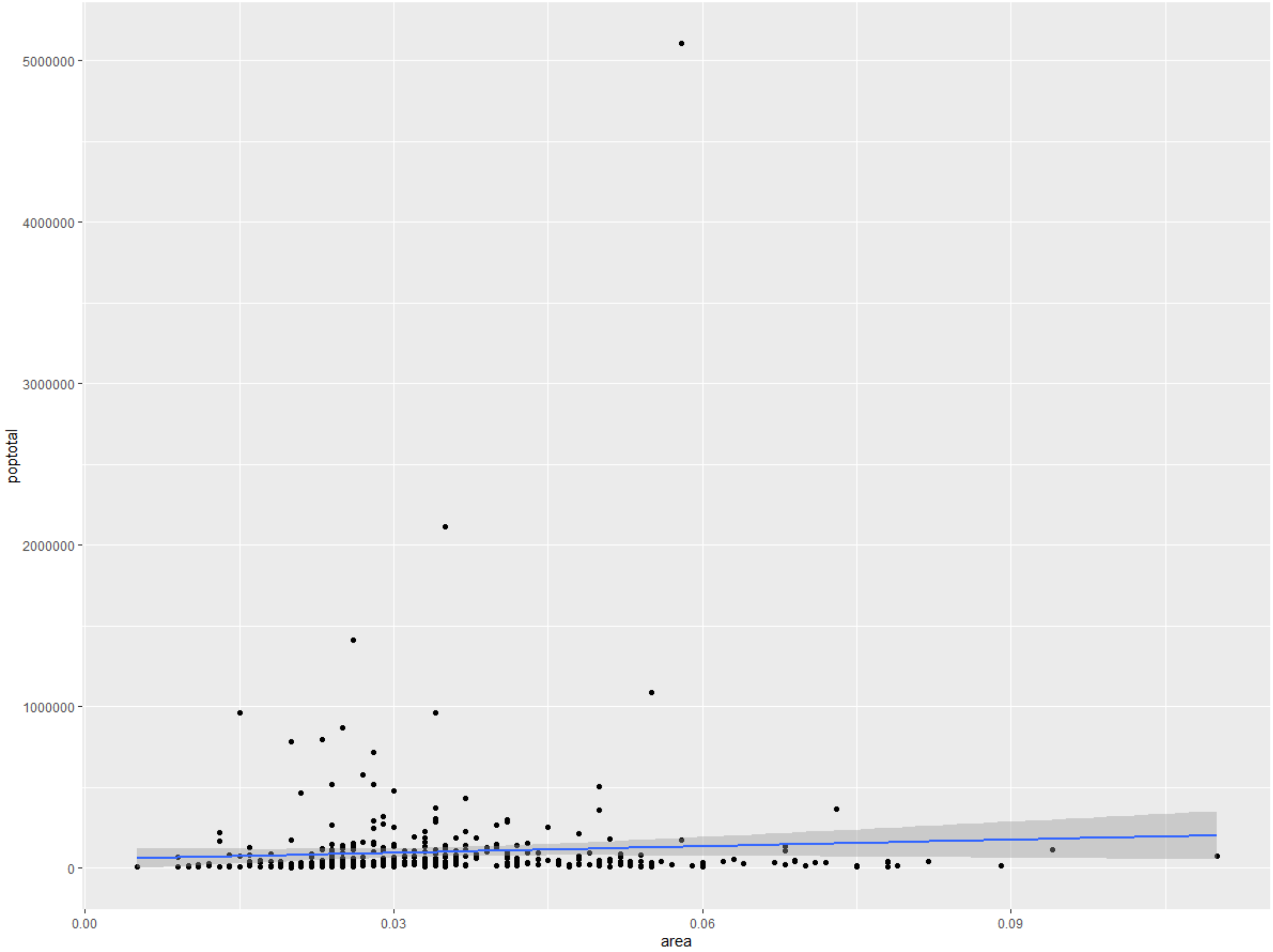
ggplot2( )

- A simple scatterplot (with a fitted line)
  - ✓ Like geom_point(), there are many such geom layers
  - ✓ 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

```
g <- ggplot(midwest, aes(x=area, y=poptotal)) +
        geom_point() +
        geom_smooth(method="lm")

# set se=FALSE to turnoff confidence bands
plot(g)
```

ggplot2( )

- A simple scatterplot (with a fitted line)



ggplot2( )

- A simple scatterplot: axis adjustment
  - ✓ The X and Y axis limits can be controlled in two ways
    - ▪ Method 1: By deleting the points outside the range

```
# Method 1: by deleting the points outside the range
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
```

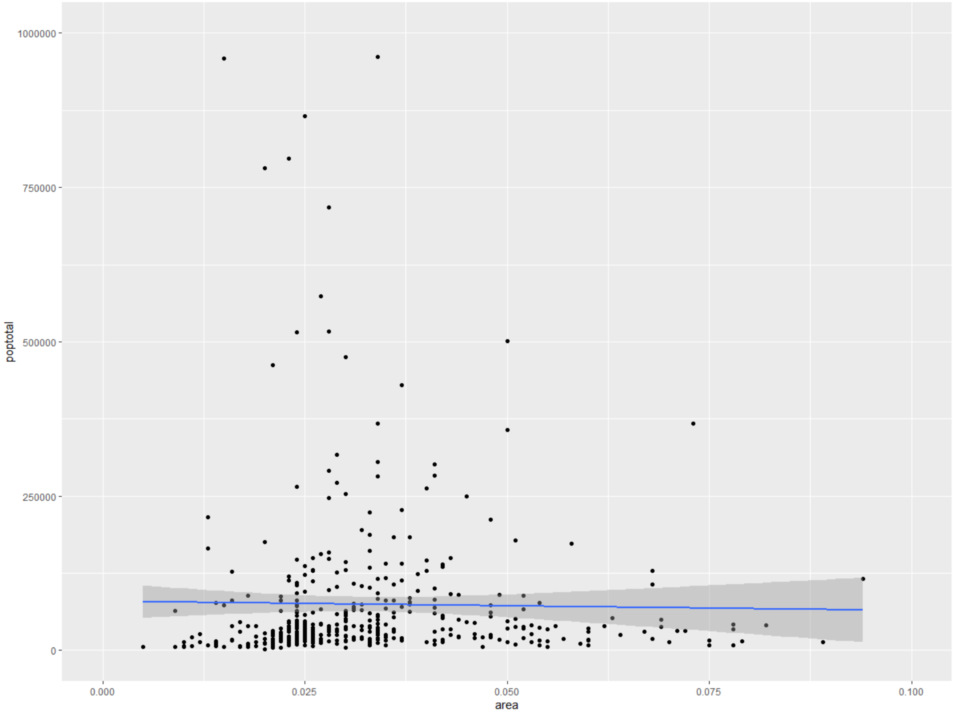
- ▪ Methods 2: Zooming In

```
# Method 2: Zooming in
g <- ggplot(midwest, aes(x=area, y=poptotal)) +
        geom_point() +
        geom_smooth(method="lm")
# 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)
```

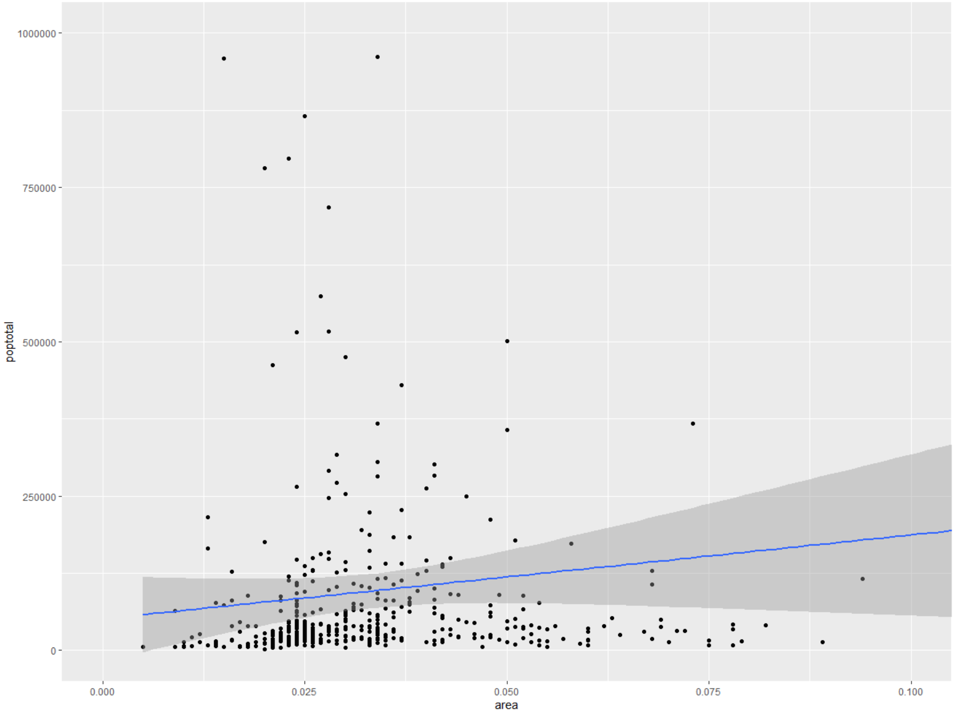
ggplot2( )

- A simple scatterplot: axis adjustment

Method 1: Deleting points



Method 2: Zooming in



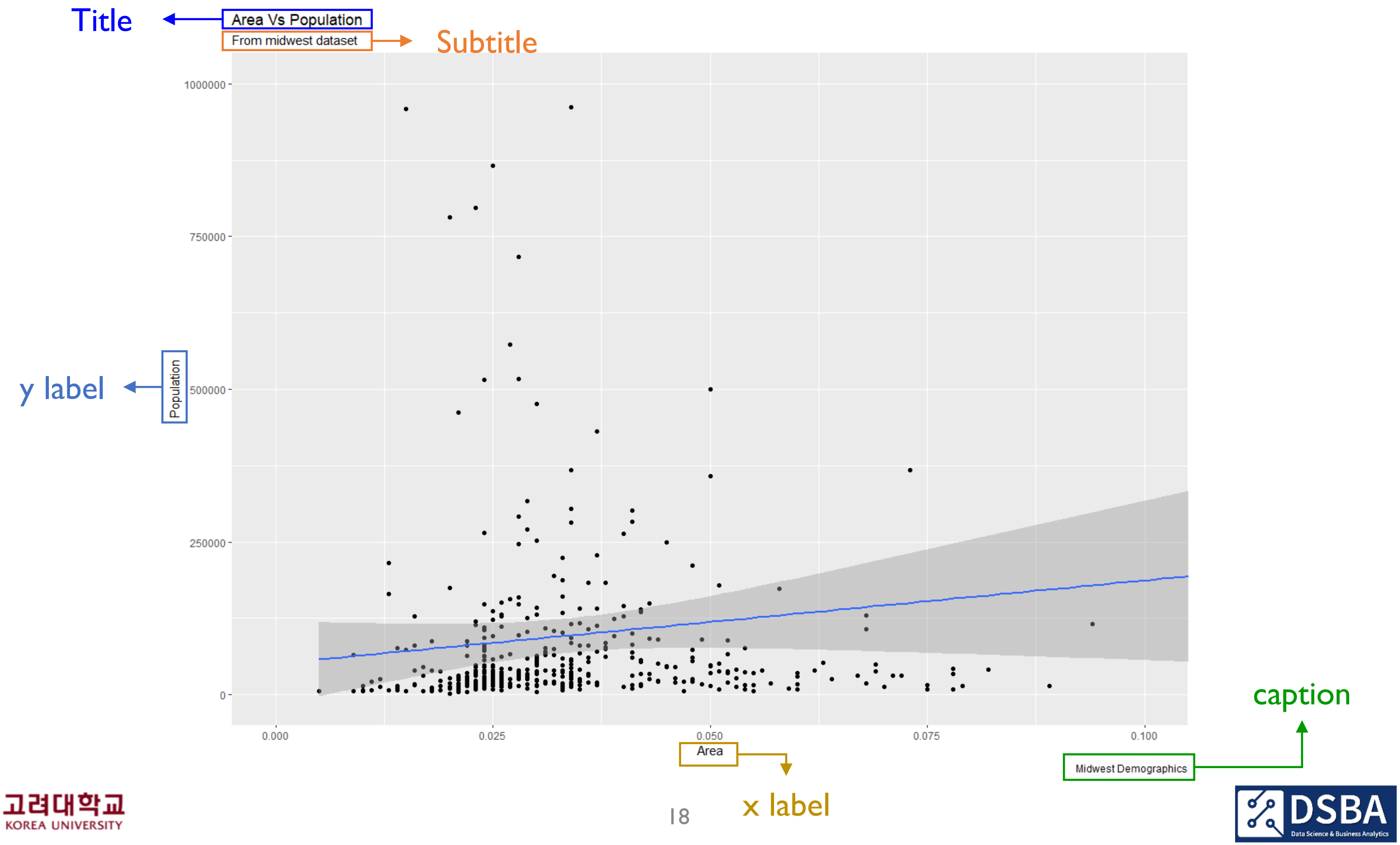
ggplot2( )

- A simple scatterplot: add title and labels

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

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

ggplot2( )

- A simple scatterplot: add title and labels



ggplot2( )

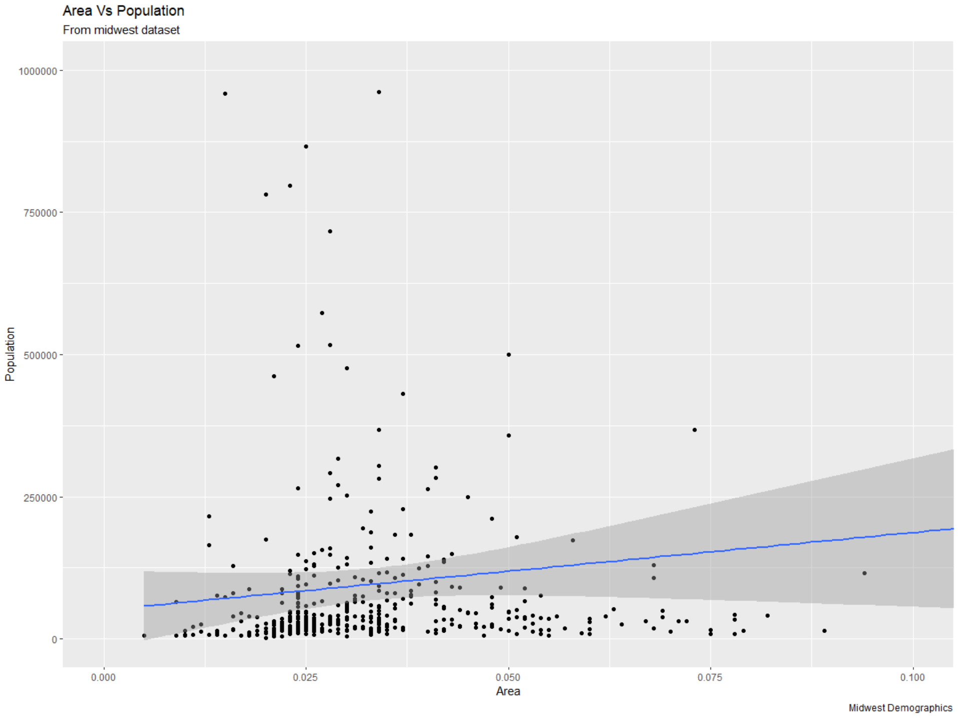
- A simple scatterplot: change color and size of points

```r
# How to change the color and size of points
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")
```

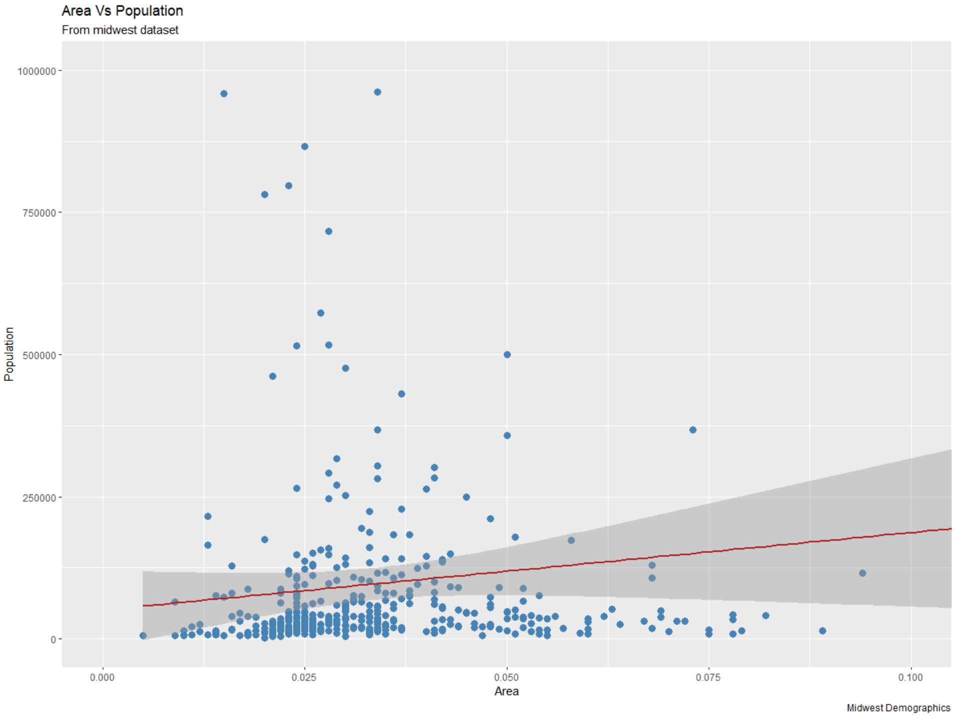
ggplot2( )

- A simple scatterplot: change color and size of points

Before



After



ggplot2( )

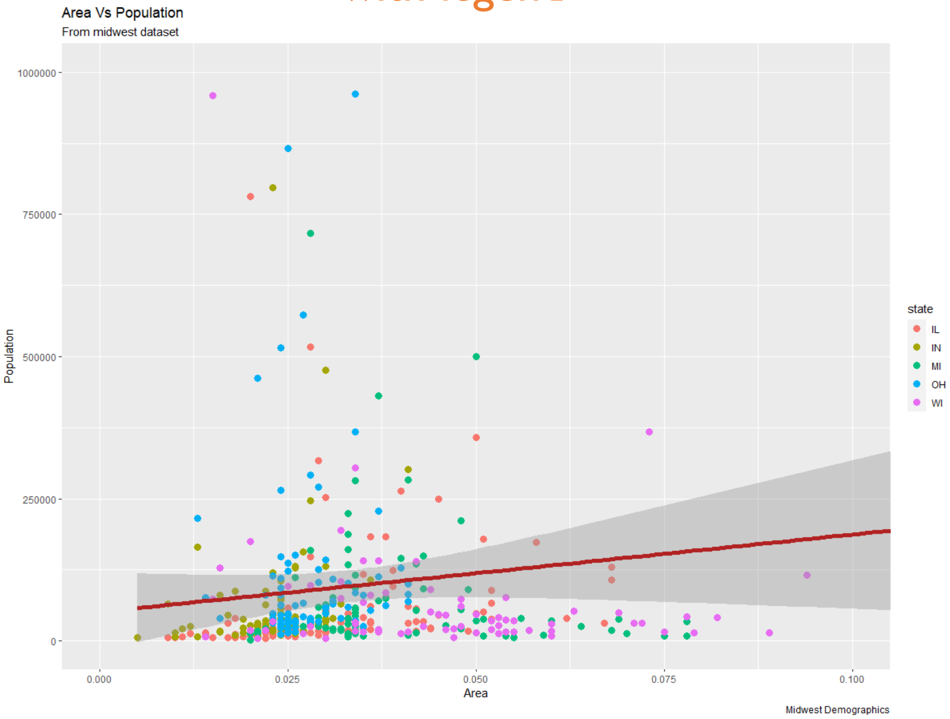
- A simple scatterplot: change the color to reflect categories in another column

```
# How to change the color to reflect categories in another column?
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)
gg + theme(legend.position="None") # remove legend

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

library(RColorBrewer)
head(brewer.pal.info, 10) # show 10 palettes
```

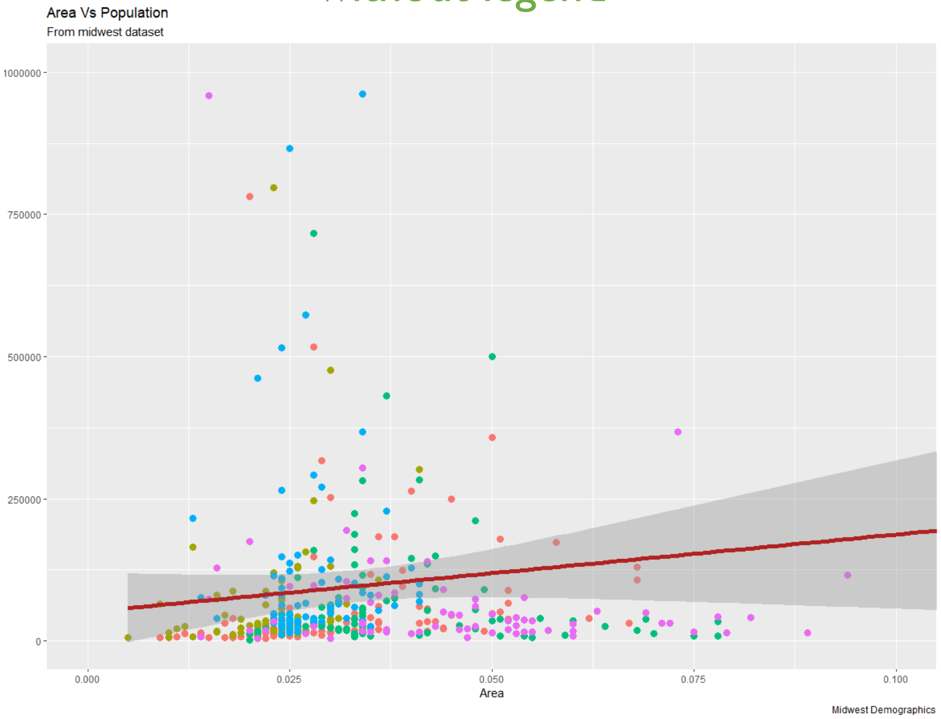
ggplot2( )

- A simple scatterplot: change the color to reflect categories in another column

with legend



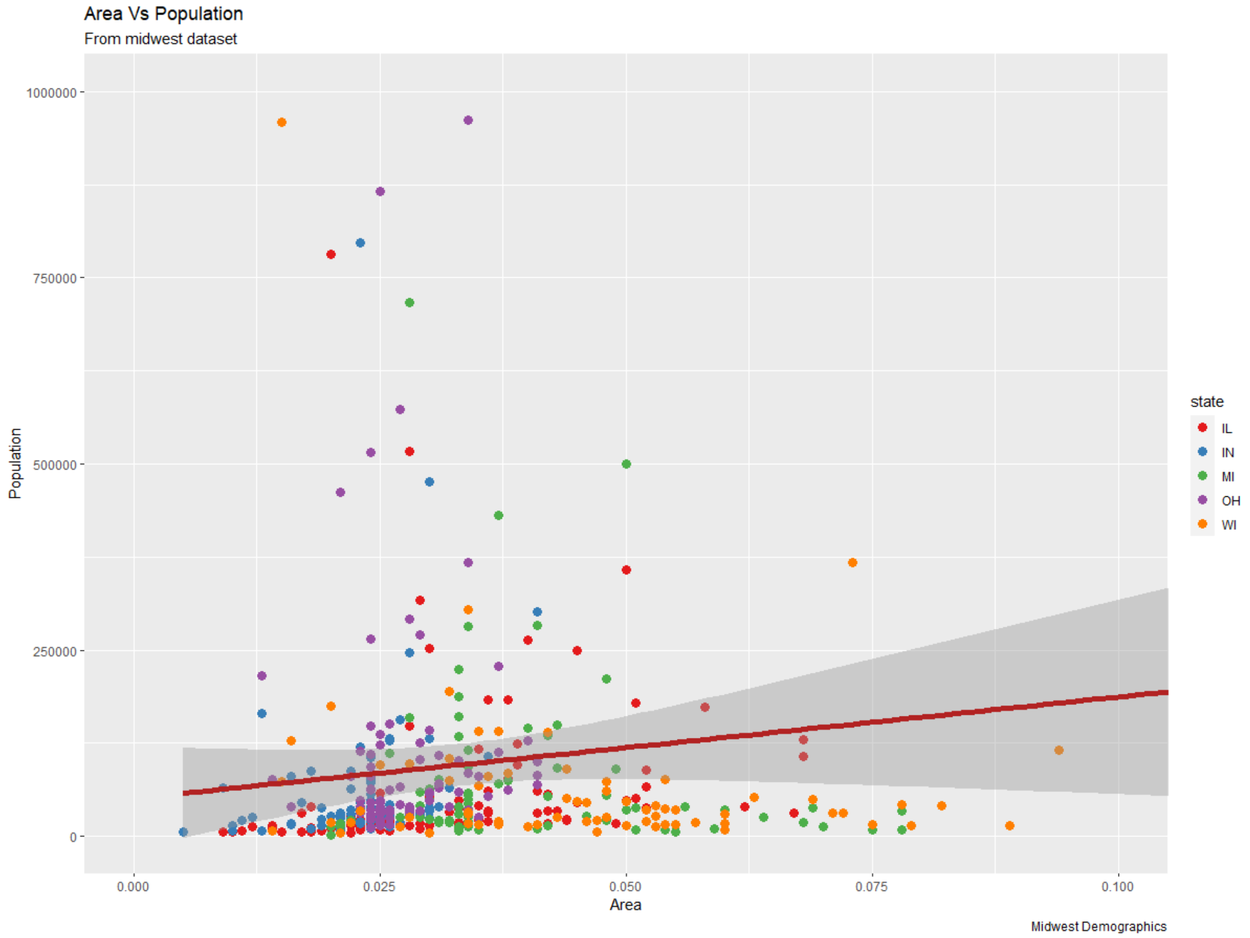
without legend



ggplot2( )

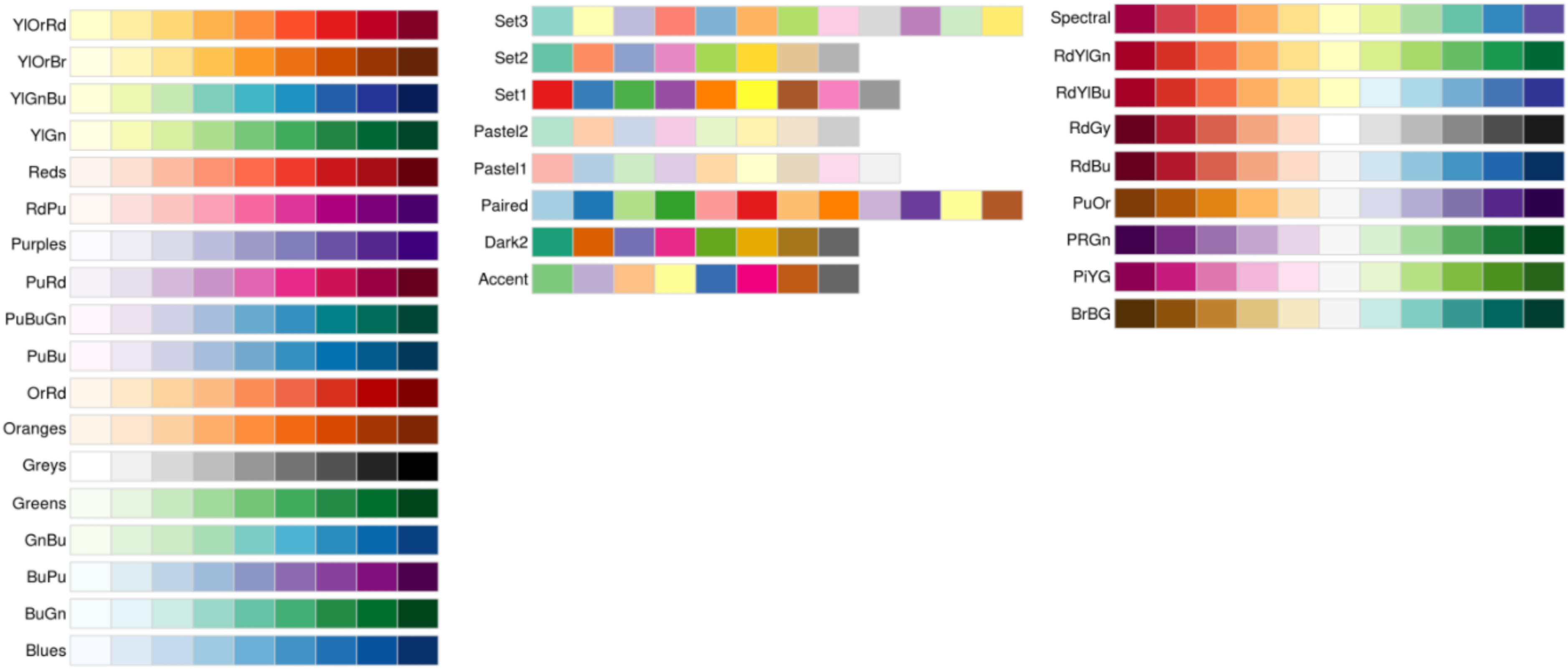
- A simple scatterplot: change the color to reflect categories in another column

another color palette



ggplot2( )

- A simple scatterplot: change the color to reflect categories in another column



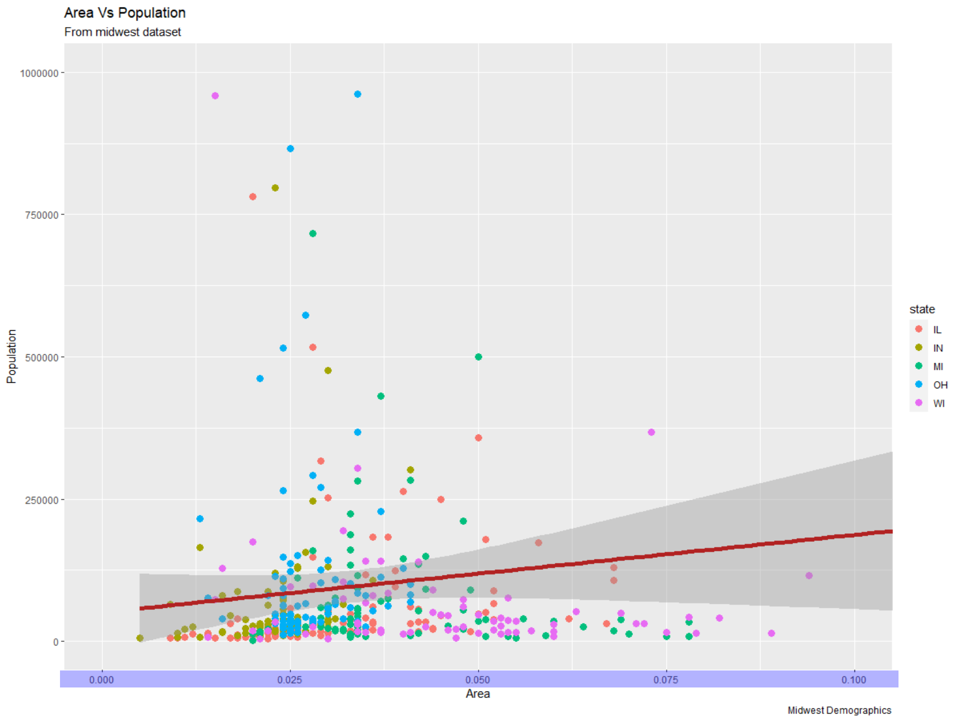
ggplot2( )

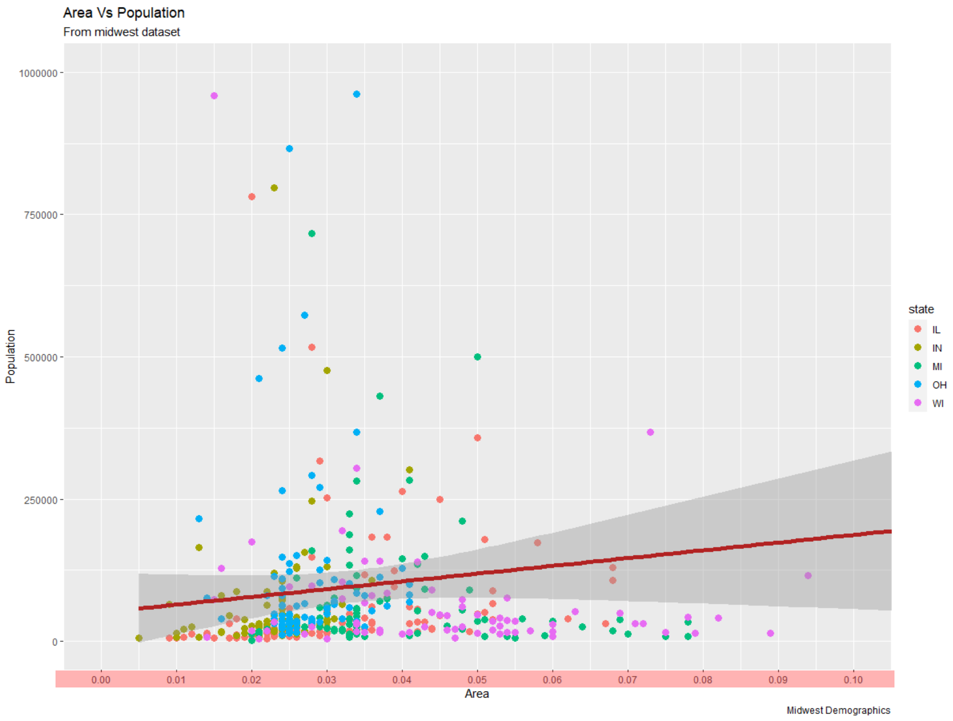
- A simple scatterplot: change x axis texts and ticks location

```
# How to change the X axis texts and ticks location
# Step 1: Set the breaks
# 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

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

# Step 2: Change the labels # Change breaks + label
gg + scale_x_continuous(breaks=seq(0, 0.1, 0.01), labels = letters[1:11])
```

ggplot2( )

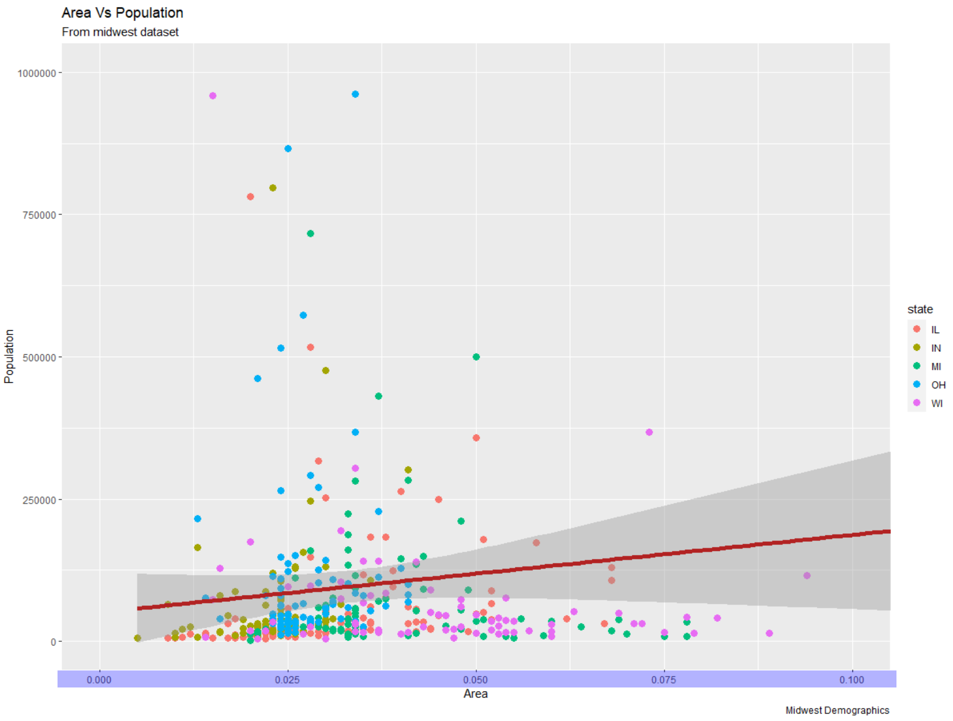
- A simple scatterplot: change x axis texts and ticks location

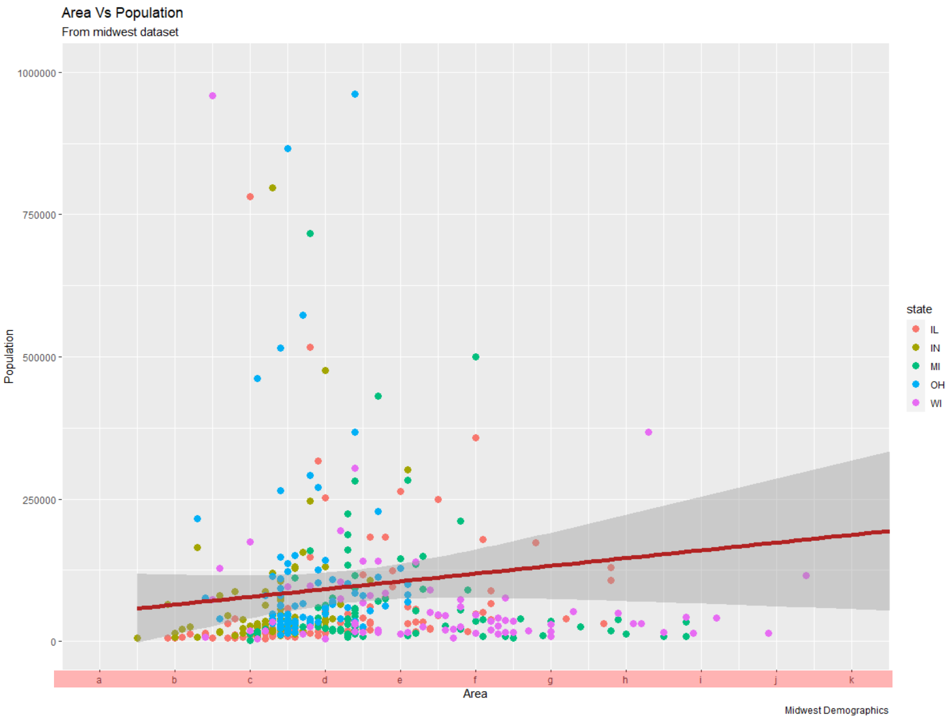




ggplot2( )

- A simple scatterplot: change x axis texts and ticks location





ggplot2( )

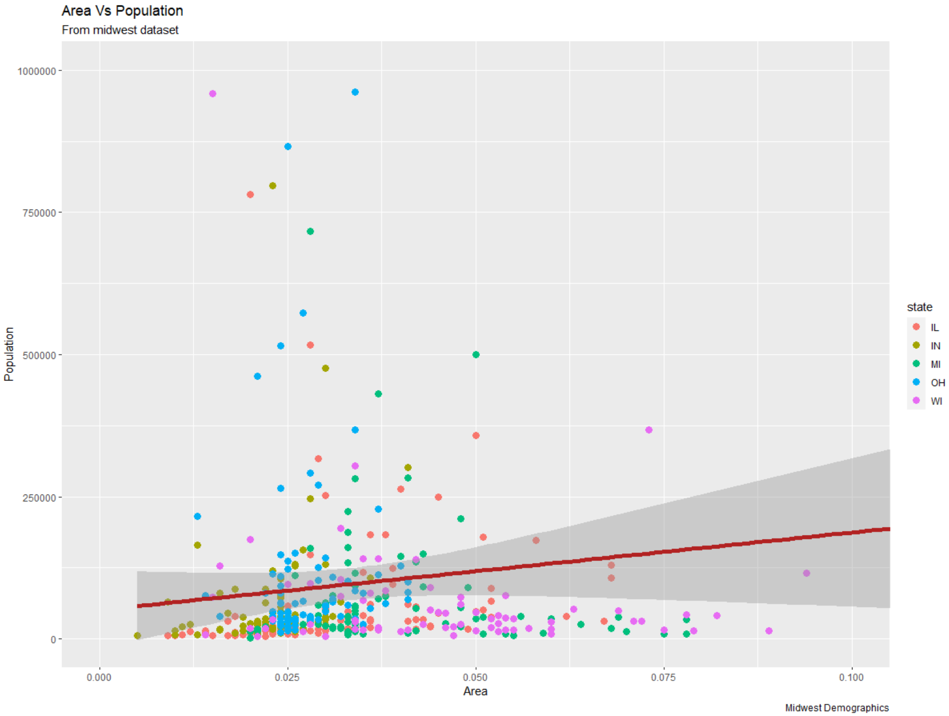
- A simple scatterplot: reverse X axis scale

```
# Reverse X Axis Scale
gg <- ggplot(midwest, aes(area, poptotal)) +
         geom_point(aes(col=state), size=3) +
         geom_smooth(method="lm", col="firebrick", size=2) +
         labs(title="Area Vs Population", subtitle="From midwest dataset",
             y="Population", x="Area", caption="Midwest Demographics") +
         scale_x_reverse()

gg + coord_cartesian(xlim=c(0.1, 0), ylim=c(0, 1000000))
```

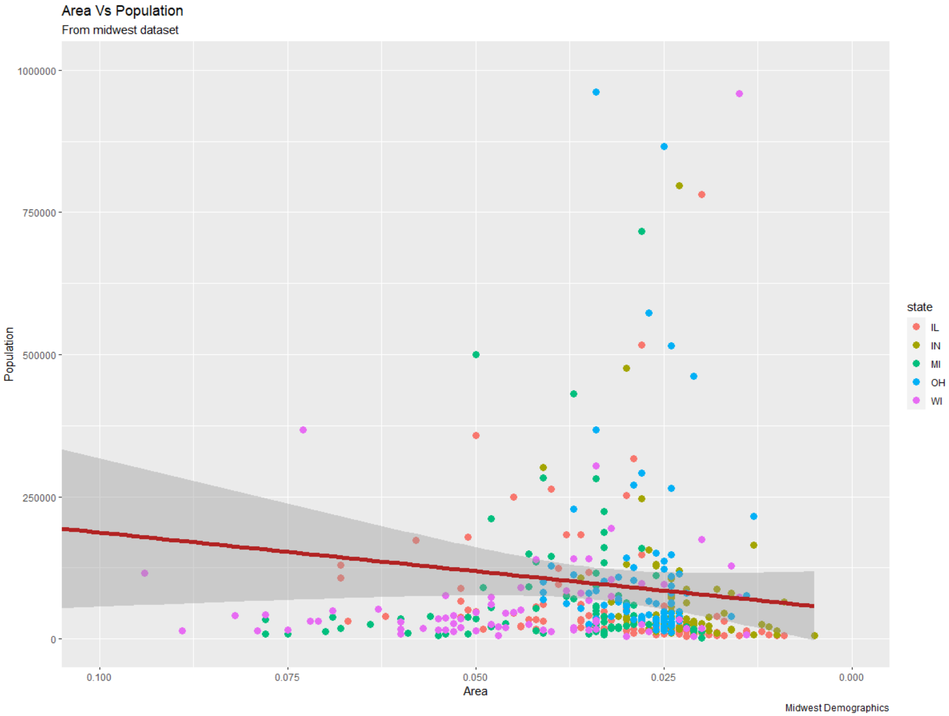
ggplot2( )

- A simple scatterplot: reverse X axis scale

Original



Reversed



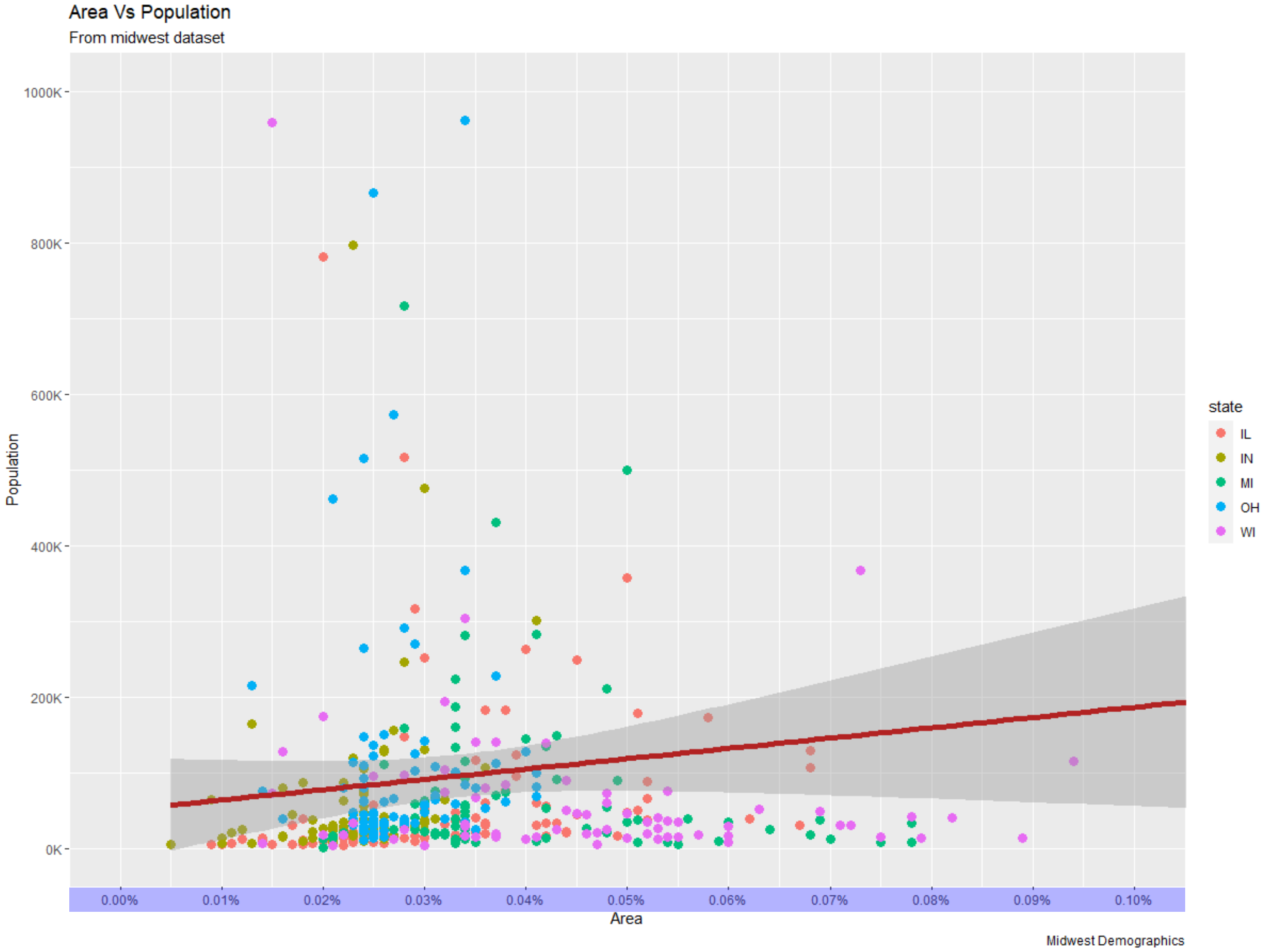
ggplot2( )

- A simple scatterplot: customized texts by formatting the original values

```
# How to write customized texts for axis labels, by formatting the original value?
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 Axis Texts
gg + scale_x_continuous(breaks=seq(0, 0.1, 0.01),
             labels = sprintf("%1.2f%%", seq(0, 0.1, 0.01))) +
         scale_y_continuous(breaks=seq(0, 1000000, 200000),
             labels = function(x){paste0(x/1000, 'K')})
```

ggplot2( )

- A simple scatterplot: customized texts by formatting the original values



ggplot2( )

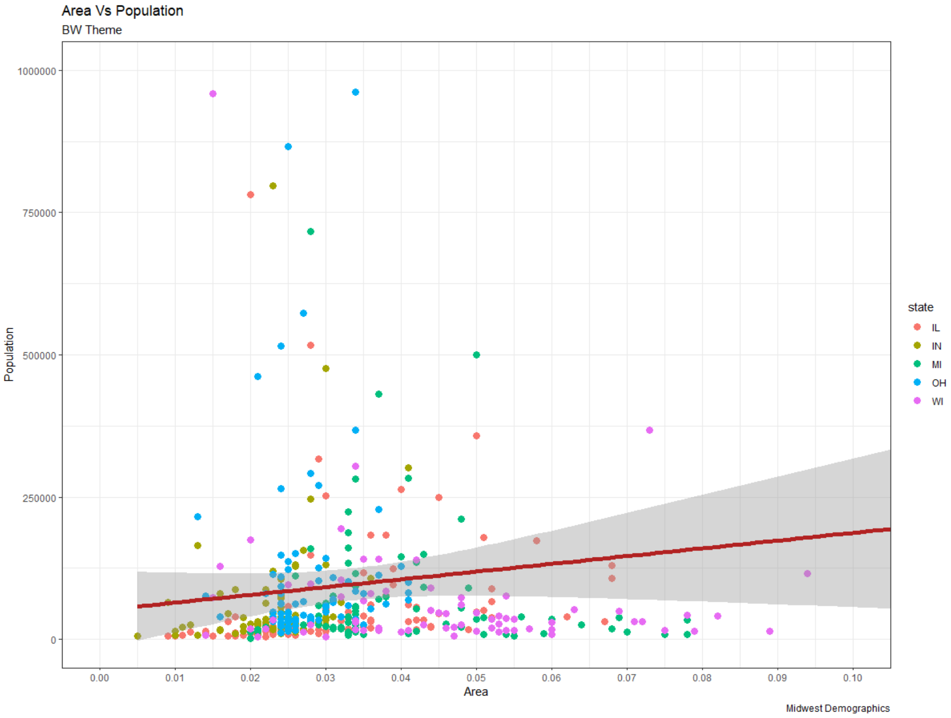
- A simple scatterplot: customized the entire theme in one shot using pre-built themes

```
# How to customize the entire theme in one shot using pre-built themes?
# 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())
gg

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

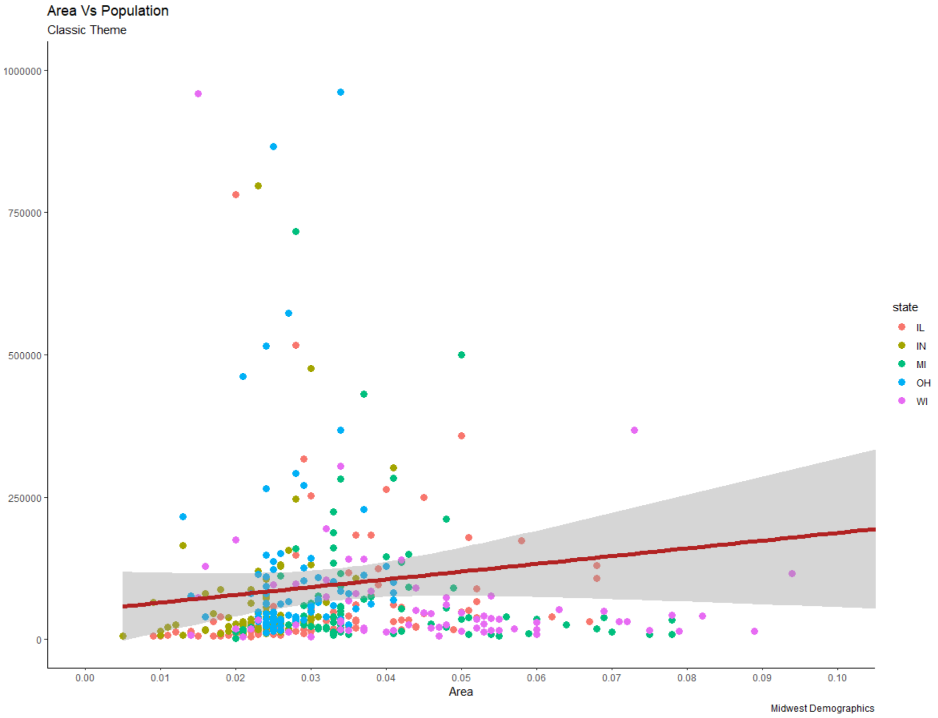
ggplot2( )

- A simple scatterplot: customized the entire theme in one shot using pre-built themes

BW Theme



Classic Theme



ANY
questions?