Modifying ggplot objects - intermediate tutorial

Jasleen Grewal

2018-05-23

Contents

Getting started

If this is your first time using the ggplot2 library within R, please refer to the **Beginners Tutorial**.

```
# If you don't have ggplot2 installed already
install.packages("ggplot2")
```

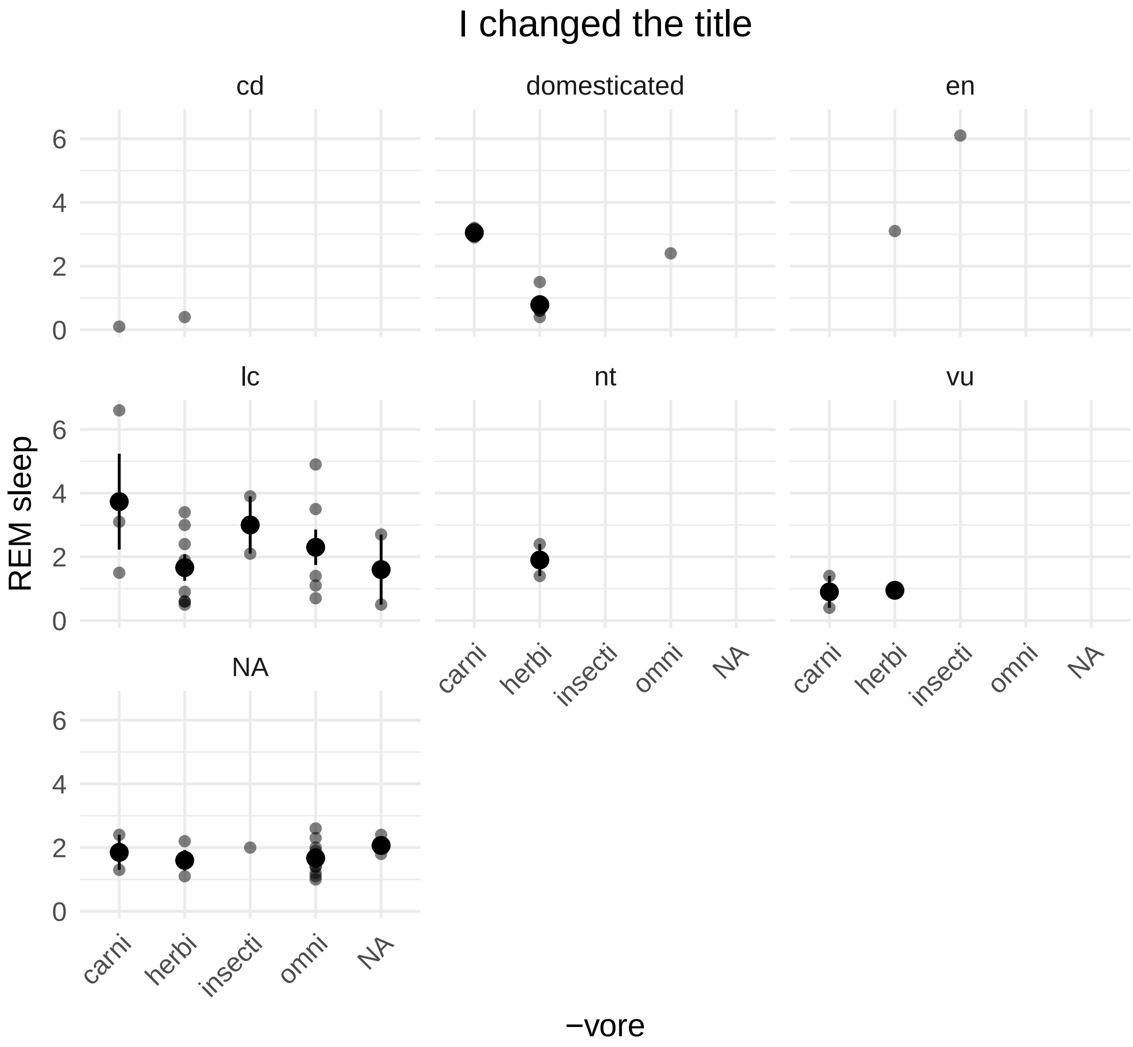
Let us load up ggplot2 into our current environment.

```
require(ggplot2)
```

ggplot - Review

Let us take a quick look at the plot script for the last ggplot figure we generated in the preceding tutorial.

```
# Get data we were working with earlier
data("msleep", package = "ggplot2")
# Plot the ggplot
ggplot(msleep, aes(x = vore, y = sleep_rem)) + geom_point(alpha = 0.5) +
    stat_summary() + theme_minimal() + theme(text = element_text(size = 12)) +
    ggtitle("I changed the title") + theme(plot.title = element_text(hjust = 0.5),
    axis.text.x = element_text(angle = 45, hjust = 1)) + labs(x = "-vore",
    y = "REM sleep") + facet_wrap(~conservation, ncol = 3)
```



There are 5 key points in this long line of code:

- The **data** object is *msleep*.
- We used `theme_minimal()` to apply a clean background (not grey!) to our plot.
- We modified the text size and plot title placement using `theme()`.
- We use `theme_minimal` **before** we add any modifications to `theme()`, because otherwise

`theme_minimal()` will reset any changes we added to our figure using `theme()`.

- We separated different categories (in 'conservation') using `facet_wrap()`, and fixed the maximum number of 'columns' we wanted for the facet panels.

Here's a quick way to shorten this code a bit more. The *theme_minimal()* element has 2 options we can define in it - font size, and font family. Instead of defining these in *theme*, we can define them in `theme_minimal()`. Check to see if this gives the same figure as above.

```
ggplot(msleep, aes(x = vore, y = sleep_rem)) + geom_point(alpha = 0.5) +
    stat_summary() + theme_minimal(base_size = 12) + ggtitle("I changed the title") +
    theme(plot.title = element_text(hjust = 0.5), axis.text.x = element_text(angle = 45,
        hjust = 1)) + labs(x = "-vore", y = "REM sleep") + facet_wrap(~conservation,
    ncol = 2)
```

- `hjust=0.5` means align text at the center, whereas `hjust=1` means right, and `hjust=0` means left.

Before we dive into today's tutorial, let me highlight an error message you may receive as you run the code above,
Warning message: Removed 22 rows containing missing values (geom_point).
This is because we have 'NA' values in our dataframe. We can overcome this by updating our msleep to get rid of rows that contain 'NA'. This is done using the function `na_omit()`.

```
msleep = na.omit(msleep)
```

> Fair warning, `na.omit` will remove rows that have an 'NA' value in *any* column, so if you have certain columns that are 'NA' more often than not, you might want to remove those first, or replace the value with 0.

Preparing our data

Get Data

We will be working with a sample dataset available within the ggplot2 library, called *midwest*. This dataset contains statistics on populations in US Midwest.

```
data("midwest", package = "ggplot2")
```

We can take a peek at the data, using the `head` command.

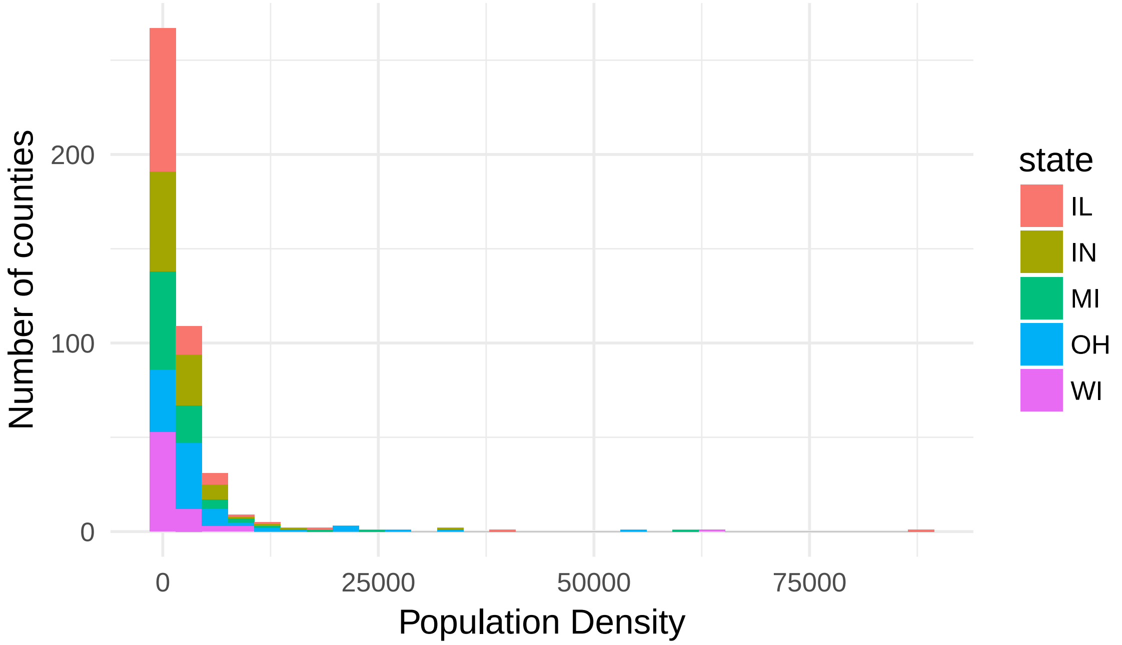
```
## Kable makes output table look pretty (requires package - knitr)
kable(head(midwest))
```

PID	county	state	area	poptotal	popdensity	popwhite	popblack	popamerindian	popasian	popoth
561	ADAMS	IL	0.052	66090	1270.9615	63917	1702	98	249	1
562	ALEXANDER	IL	0.014	10626	759.0000	7054	3496	19	48	
563	BOND	IL	0.022	14991	681.4091	14477	429	35	16	
564	BOONE	IL	0.017	30806	1812.1176	29344	127	46	150	11
565	BROWN	IL	0.018	5836	324.2222	5264	547	14	5	
566	BUREAU	IL	0.050	35688	713.7600	35157	50	65	195	2

Exploratory plots

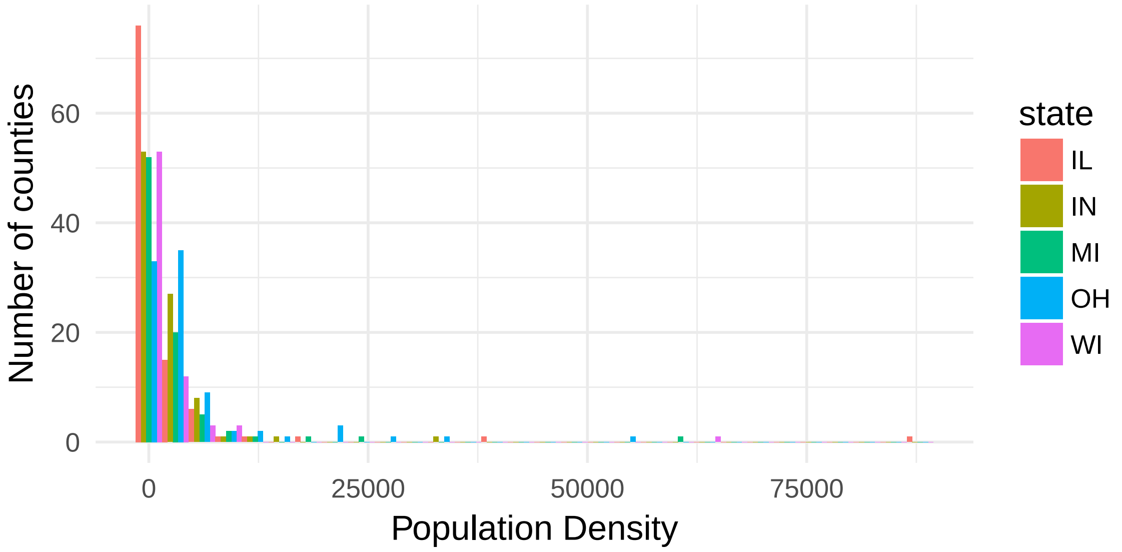
Let us generate some exploratory figures from our dataset. Firstly we will try to plot the population density across different counties in different states, as histograms:

```
# What does the state-wise population density look like?
ggplot(midwest, aes(fill = state, x = popdensity)) + geom_histogram() +
    theme_minimal(base_size = 13) + labs(x = "Population Density", y = "Number of counties")
```



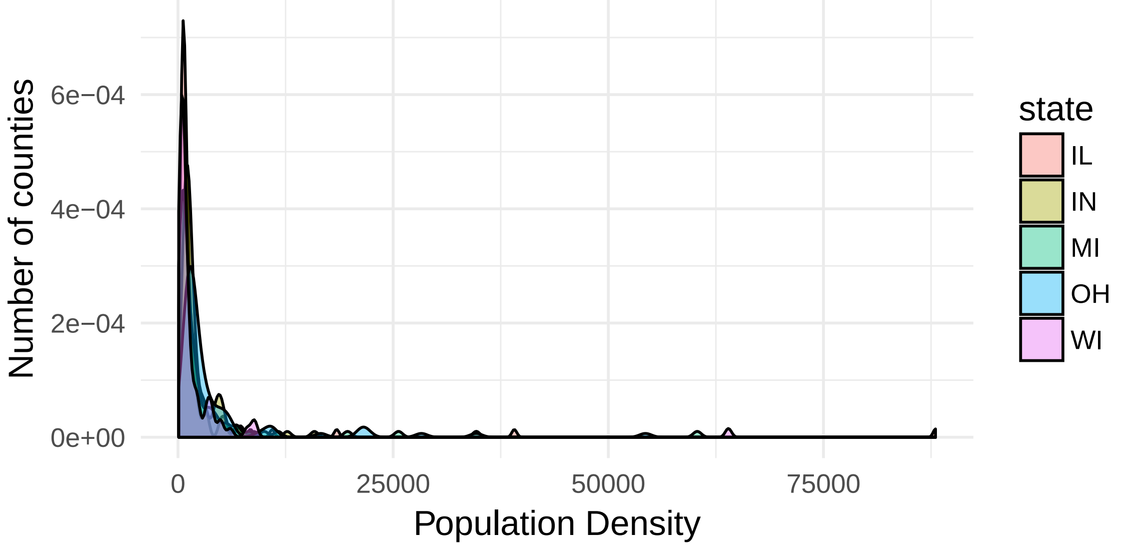
Notably, the histogram is currently stacked for different states. We can 'separate' the states as separate bars, by modifying `position` in 'geom_histogram():

```r
ggplot(midwest, aes(fill = state, x = popdensity)) + geom_histogram(position = "dodge") +
    theme_minimal(base_size = 13) + labs(x = "Population Density", y = "Number of counties")
```



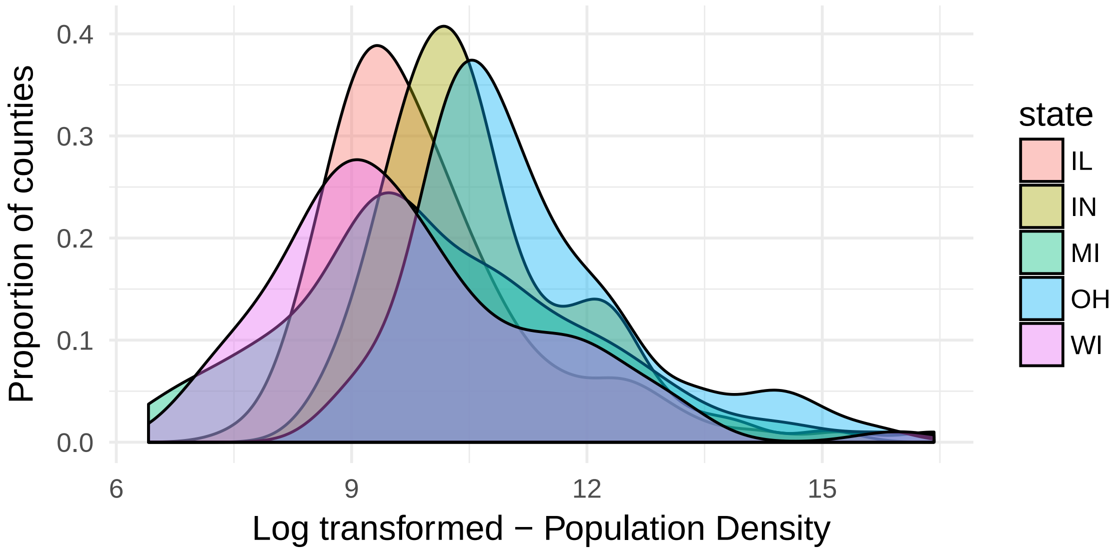
We can also visualize these as state-specific data distributions, like so:

```r
ggplot(midwest, aes(fill = state, x = popdensity)) + geom_density(alpha = 0.4) +
    theme_minimal(base_size = 13) + labs(x = "Population Density", y = "Number of counties")
```



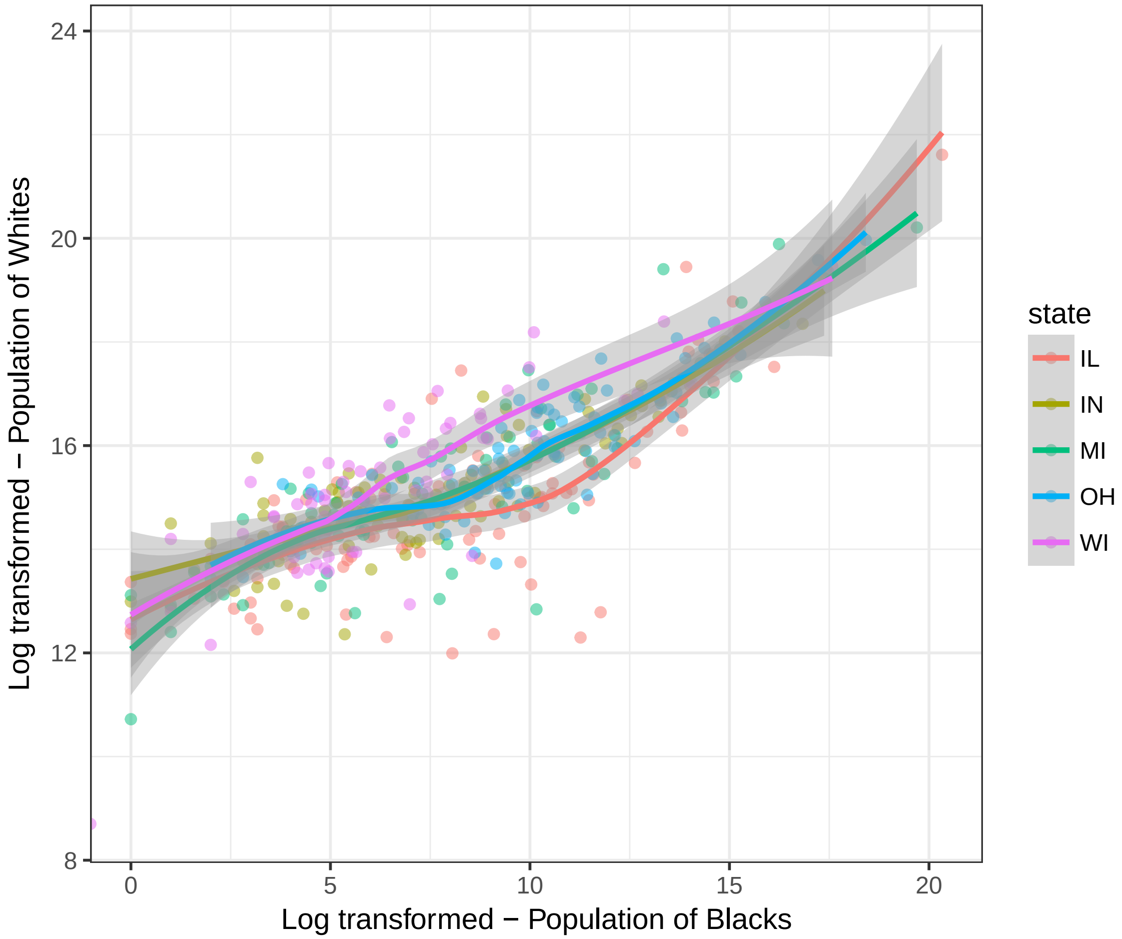
Notice how the y-axis has such a wide range but most of our data is in a tiny range - we can log transform the data to 'spread it out'.
- Instead of `log` transforming my column on the actual dataset, I can simply apply the function to the `y` variable within `aes()`.
- In the next parts of the tutorial we will be modifying things like panels, font size, colour, panel colour, grid-lines - since most of our ggplot will retain the same structure (we will only be **stylizing** aspects of it), I will 'initialize' each new exploratory plot as a separate object, using `obj_name <- ggplot(something something)` . This comes in handy if we want to keep adding new 'geometric objects' on top of our original ggplot figure, without copying the code every single time.

```
g1_poplog <- ggplot(midwest, aes(fill = state, x = log2(popdensity))) +
    geom_density(alpha = 0.4) + theme_minimal(base_size = 13) + labs(x = "Log transformed - Population
    y = "Proportion of counties")
plot(g1_poplog)
```



Maybe I want to compare the population of whites against black people in the different counties of all states?
- We can fit a 'loess' curve to see specific trends for different states, with the `stat_smooth()` element.
- This can be modified with the `method` option in `stat_smooth()`

```
# Define plot
scatter_popbw <- ggplot(midwest, aes(x = log2(popblack), y = log2(popwhite),
    colour = state)) + geom_point(alpha = 0.5) + theme_bw(base_size = 11) +
    stat_smooth() + labs(x = "Log transformed - Population of Blacks",
    y = "Log transformed - Population of Whites")
# Print plot
scatter_popbw
```



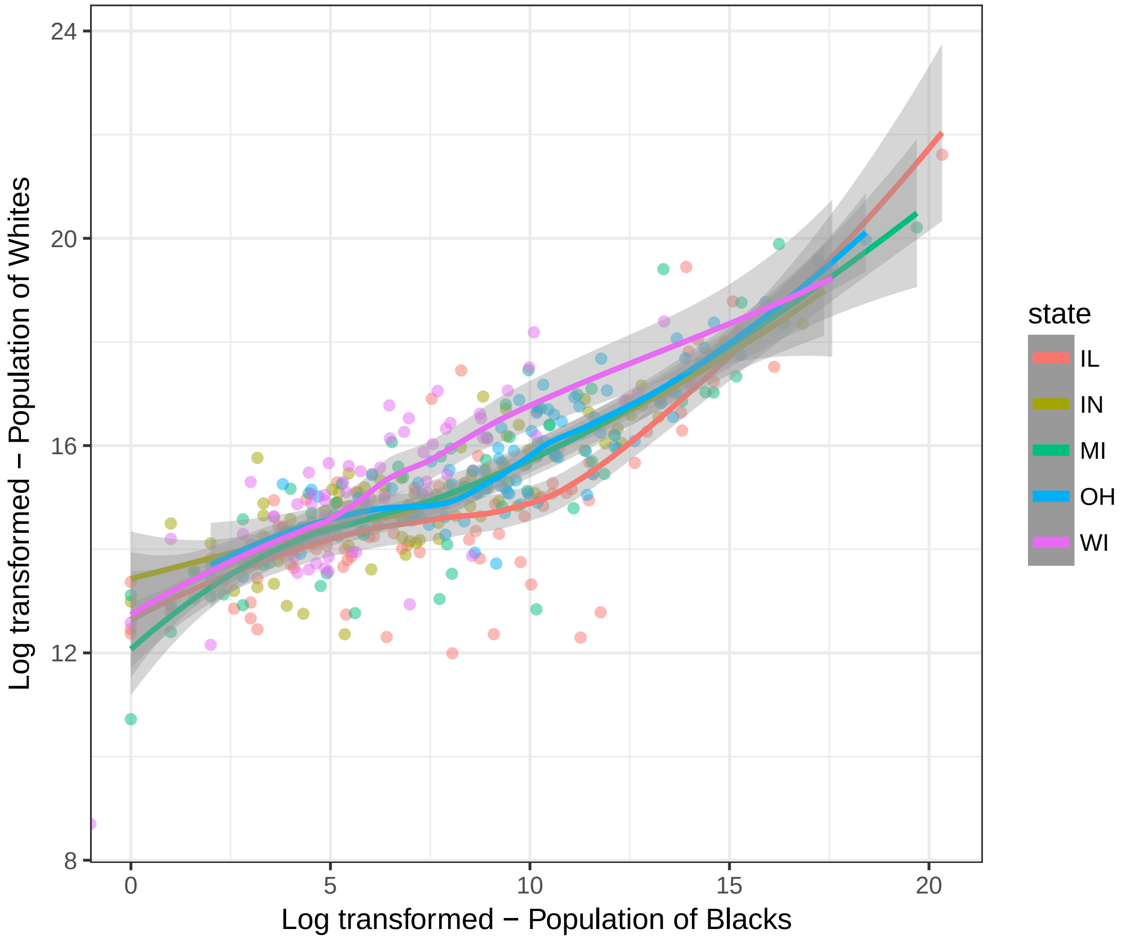
Try the following:
- Remove the SD spread (set `se = FALSE` in `stat_smooth()`)
- Plot the SD spread in the 90% confidence interval (set `level = 0.90` in `stat_smooth()`)

Modifying guides

Notice how, in the legend for colours in our last plot, the points appear faint - this is because by default, we plot them with alpha=0.5. However, we ideally want our points to appear with full opaqueness in our legend. For this, we modify the `guides()` element.

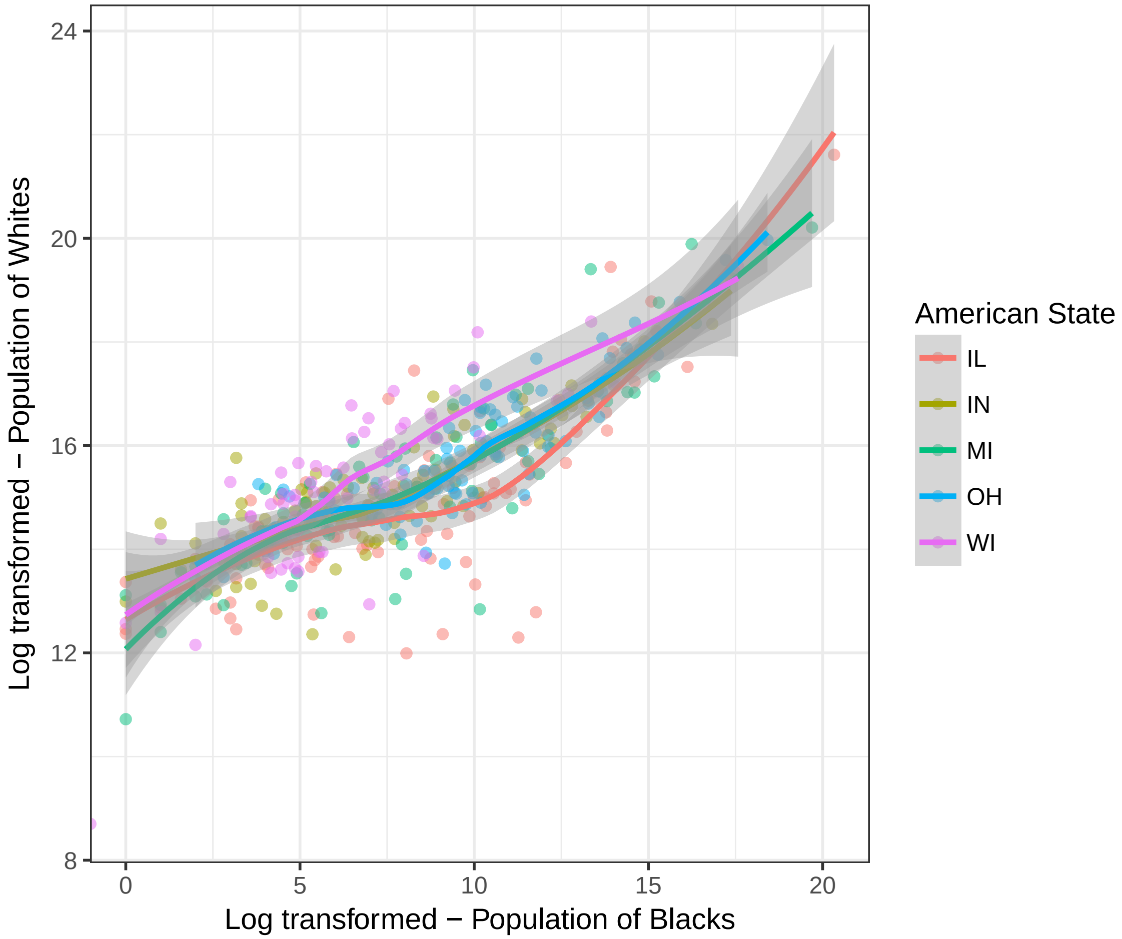
> Guides for different attributes can be set using this argument. Depending on the attribute we want to modify, we edit it using a nested attribute called `guide_legend()` (for fill, colour, line etc) or `guide_colourbar` (for bars of colour gradients).

```
scatter_popbw + guides(colour = guide_legend(override.aes = list(alpha = 1,
    size = 2)))
```



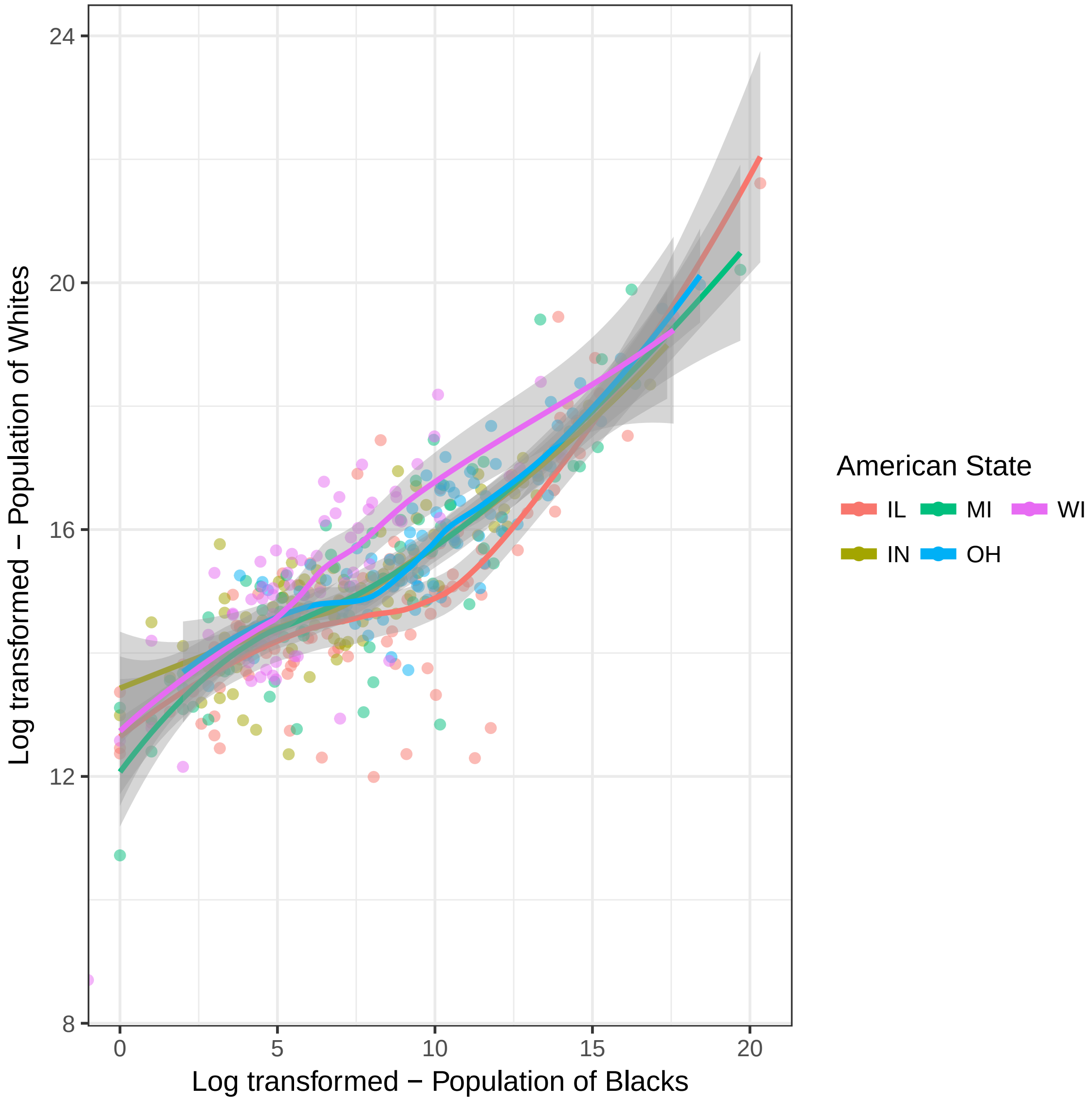
What did we do? While there are several attributes we can modify using `guide_legend`, here we simply overwrote some of the default aesthetics for ‘colour’ in the colour legend for our plot.
We can, ofcourse, go ahead and change other things for our colour legend.

```
# First I'll save the previous plot
scatter_goodalpha <- scatter_popbw + guides(colour = guide_legend(override.aes = list(alpha = 1,
    size = 2)))
# Then I'll add another 'guides()' element with a different set of
# attributes to modify
scatter_goodalpha + guides(colour = guide_legend(title = "American State"))
```



Oops- notice how adding `guides` twice to a plot only keeps the last definition? Best to just define it once. We will also remove the annoying 'grey' colour around the line by setting `fill=NA` in the `override.aes` section.

```
# Define all legend modifications in 1 guides() statement
multiline_labels <- scatter_popbw + guides(colour = guide_legend(override.aes = list(alpha = 1,
    size = 2, fill = NA), title = "American State", nrow = 2))


horiz_labels <- scatter_popbw + guides(colour = guide_legend(override.aes = list(alpha = 1,
    size = 2, fill = NA), title = "American State", direction = "horizontal",
    nrow = 2, label.position = "bottom"))

multiline_labels
```

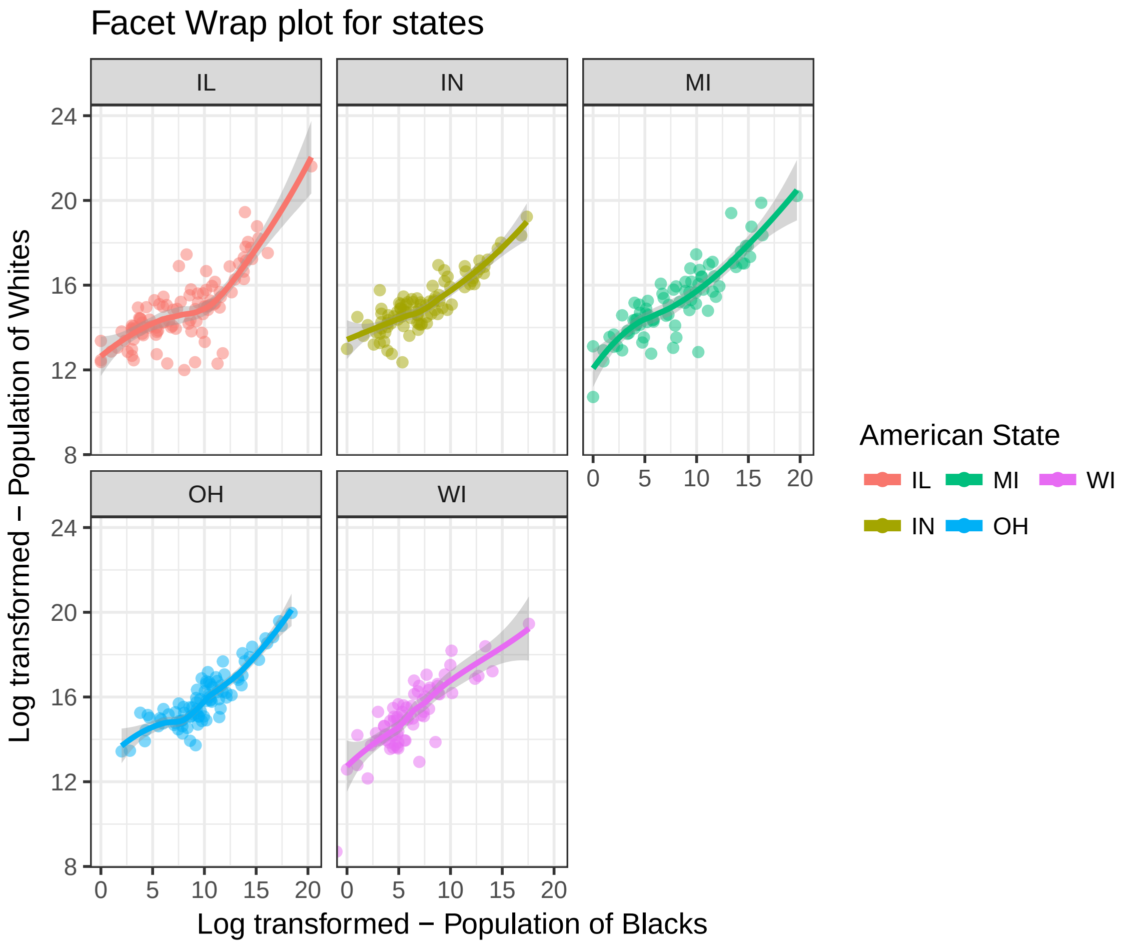


In the last plot, you may be having a hard time distinguishing the different states because of the high amount of overlap in them. We can separate these elements out into multiple ggplots using `facet_wrap()`.

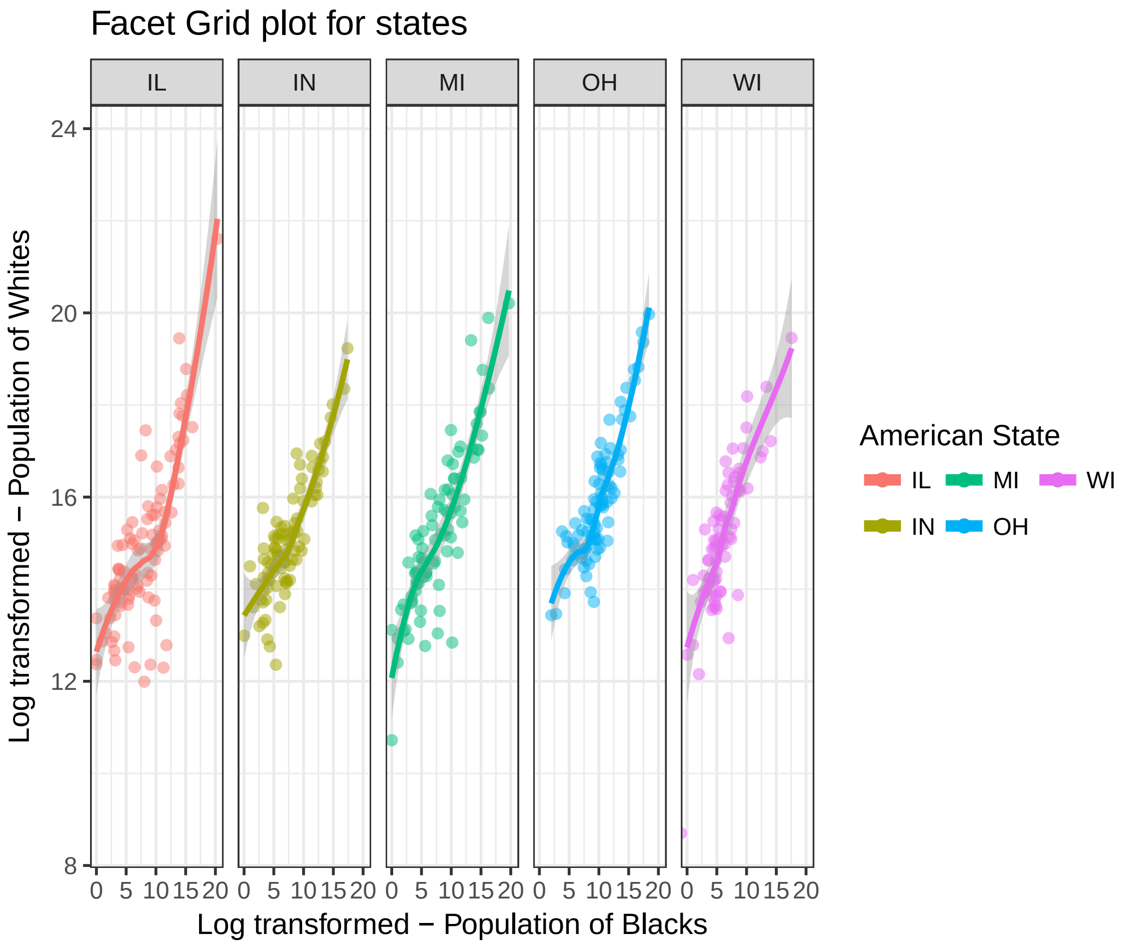
Facet Wrap and Facet Grid - plotting variables

The `facet_wrap()` function lets us split our data-points into separate groups based on a single column. We can also do this to compare two columns. If we are interested in visualizing all pair-wise comparisons between two columns (even when there may be no data points available for certain pairs of values), we use `facet_grid()`.
The following two plots show the difference between the outputs of `facet_wrap()` and `facet_grid()`.

```
# Create Facet Wrap plot
g1_fw <- multiline_labels + ggtitle("Facet Wrap plot for states") + facet_wrap(~state)
# Print this
g1_fw
```



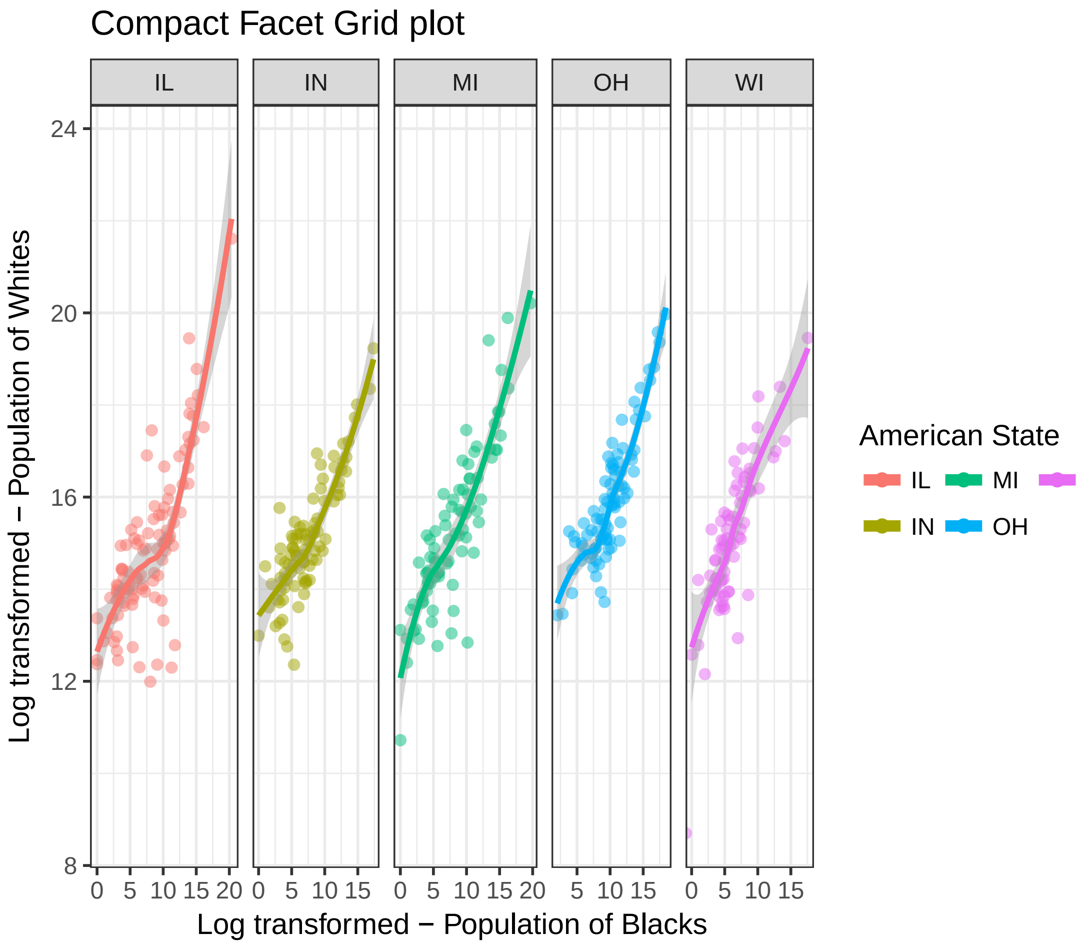
```
# Create Facet Grid plot
g1_fg <- multiline_labels + ggtitle("Facet Grid plot for states") + facet_grid(~state)
# Print this
g1_fg
```



As you can see, `facet_grid()` is much more readable. We can improve readibility by removing entries along the x-axis that do not have any values in a given 'panel'. This is done using the `scales` option in `facet_grid()` - the possible values are "free_x", "free_y", "free". Depending on what you want your reader to focus on, one of these options may be chosen.

We can also make sure the panel width is complementary to the amount of space the data points *actually* take up. We can set this using the `space` option, which has the same set of possible values as `scales`.

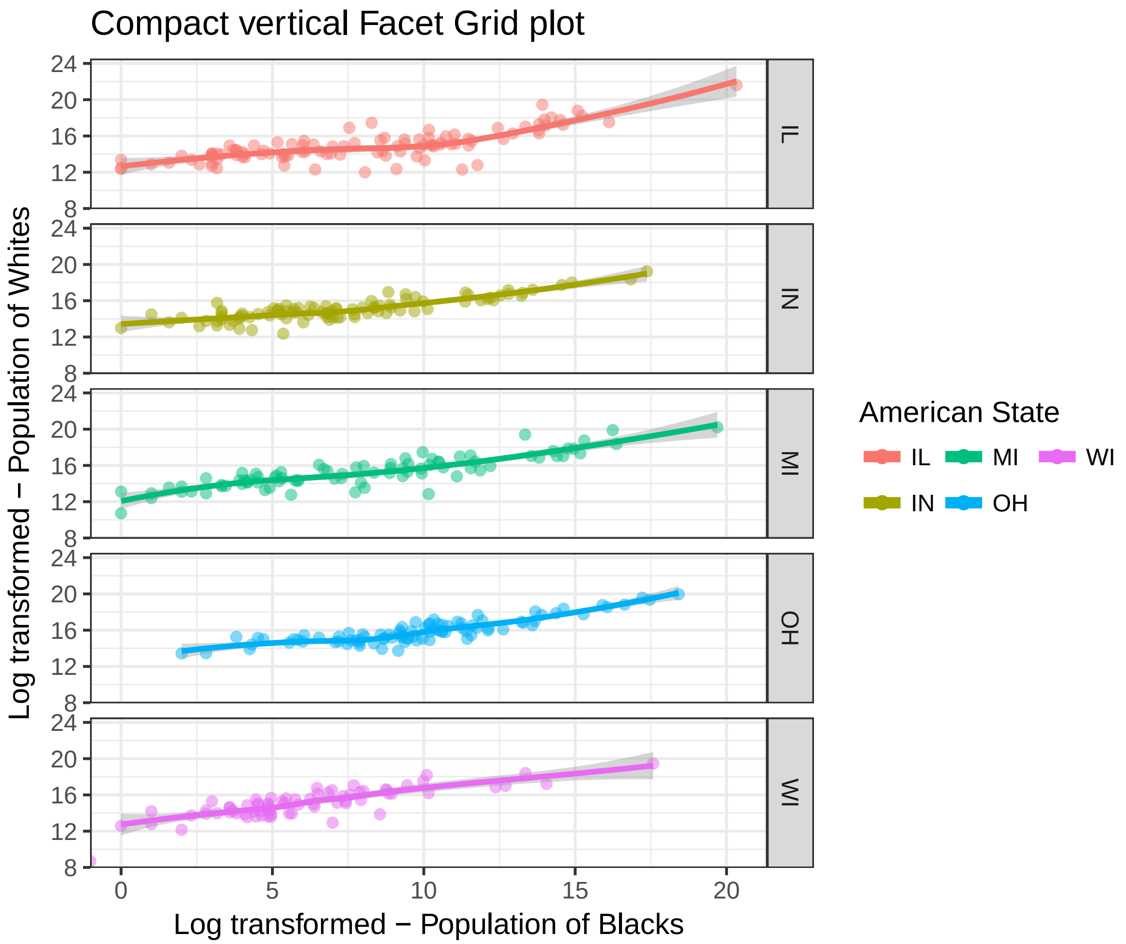
```
# Create plot object
g1_fg_clean <- multiline_labels + ggtitle("Compact Facet Grid plot") +
    facet_grid(~state, scales = "free_x", space = "free_x")
# Print plot object
g1_fg_clean
```



- Try removing the `space` option and see how the plot changes.
- What happens if you set `scales = "free"`? Do you see why including a free scale for 'y' might be misleading for readers?
- Do you also see how it is handy to create a new plot object?

Instead of having horizontally aligned columns in `facet_grid`, we can generate vertically aligned columns by defining the facetting relationship of the interesting variable against the rest of the data, `state~.`, instead of everything versus variable, `~state`.

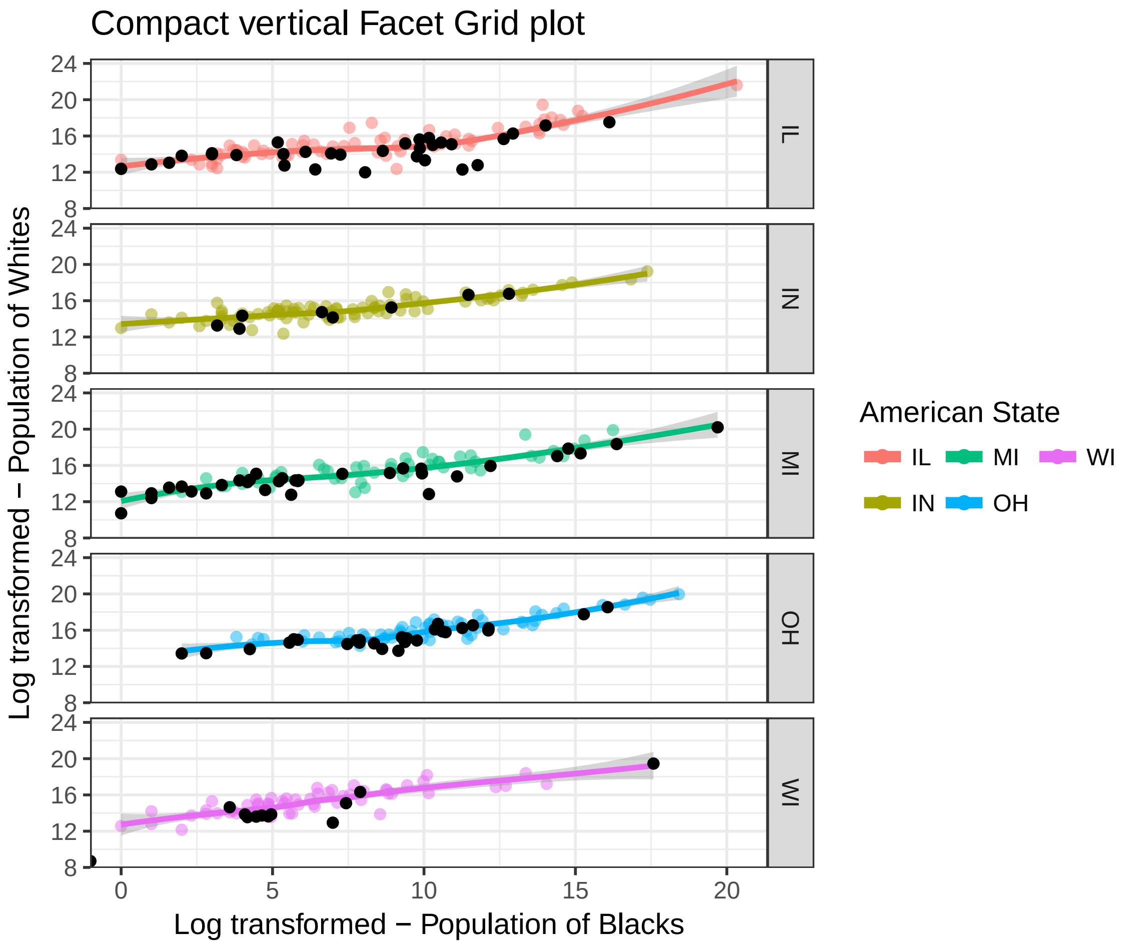
```
# Create plot object
g1_fg_clean_vertical <- multiline_labels + ggtitle("Compact vertical Facet Grid plot") +
    facet_grid(state ~ ., scales = "free_x", space = "free_x")

# Print plot object
g1_fg_clean_vertical
```



Highlighting a part of the plot

Sometimes we may want to draw attention to only specific entries in the plot. For example, in the plot above, we may want to focus specifically on data points (representative of State countys) where poverty is high (`percbelowpoverty > 15`) . We can explicitly plot these points on top of our existing figure, as shown below:

```
g1_fg_clean_vertical_pov <- g1_fg_clean_vertical + geom_point(data = midwest[midwest$percbelowpoverty >
    15, ], colour = "black")
g1_fg_clean_vertical_pov
```

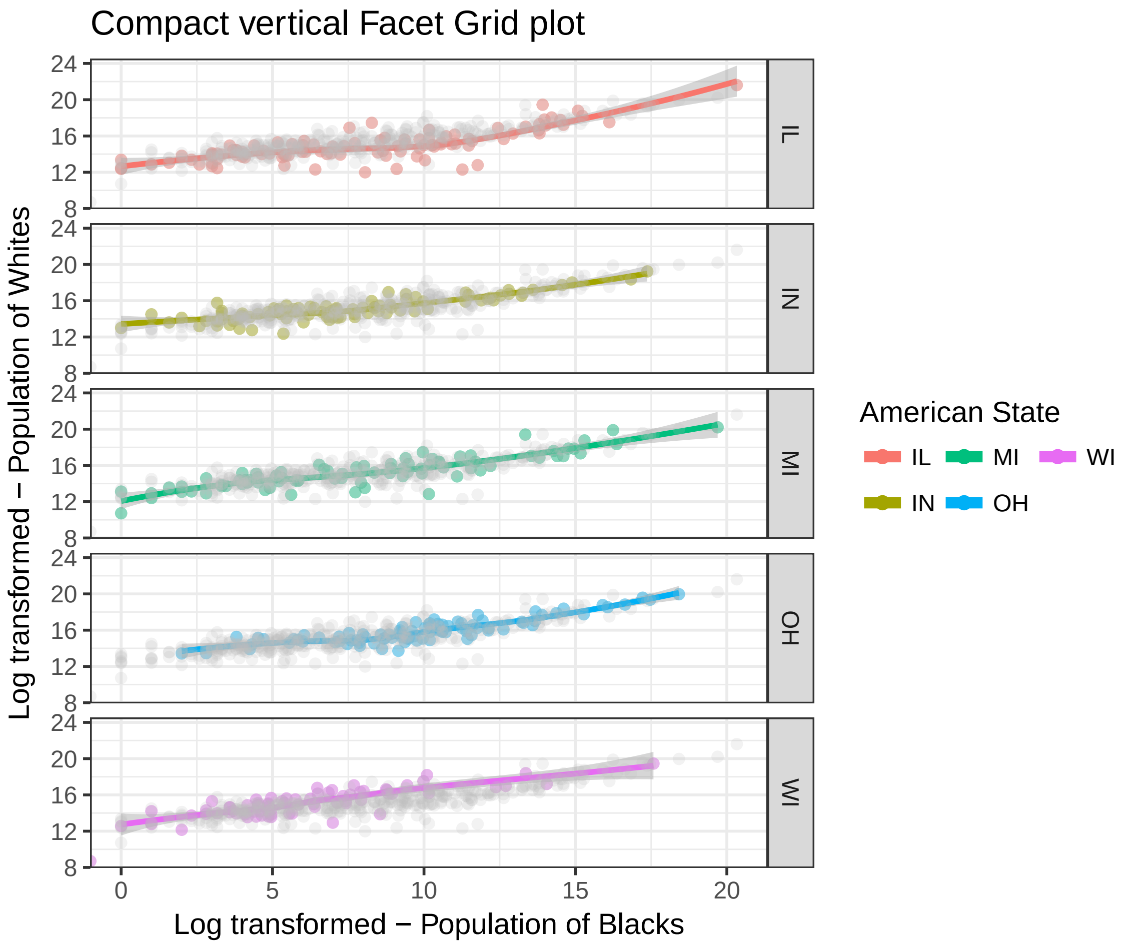


Does this seem to suggest that high poverty is found in areas where the relative white population is lower than the expected white population based on state trends? - find out by doing grad school!

What did we do here? We took our pre-existing plot where everything had been plotted. We then specified only a *part* of our original dataset to be passed to a new geometric function, a `geom_point()`, and we set the `colour="black"` for all the points that this `geom_point()` is plotting.

Optional: We can also go the other way, where we add **all** the points to **all** the panels as **background**. For this, however, we simply have to make sure that the new data we add into the new `geom_point()` does not contain the variable we facet on. Thus, we only keep the columns we need for our 'x' and 'y' axes.

```
g1_fg_clean_vertical_bg <- g1_fg_clean_vertical + geom_point(data = midwest[,
    c("popwhite", "popblack")], colour = "grey", alpha = 0.2)
g1_fg_clean_vertical_bg
```



- What happens if you add `"state"` to the list of columns selected in the data attribute of the new `geom_point()` component?

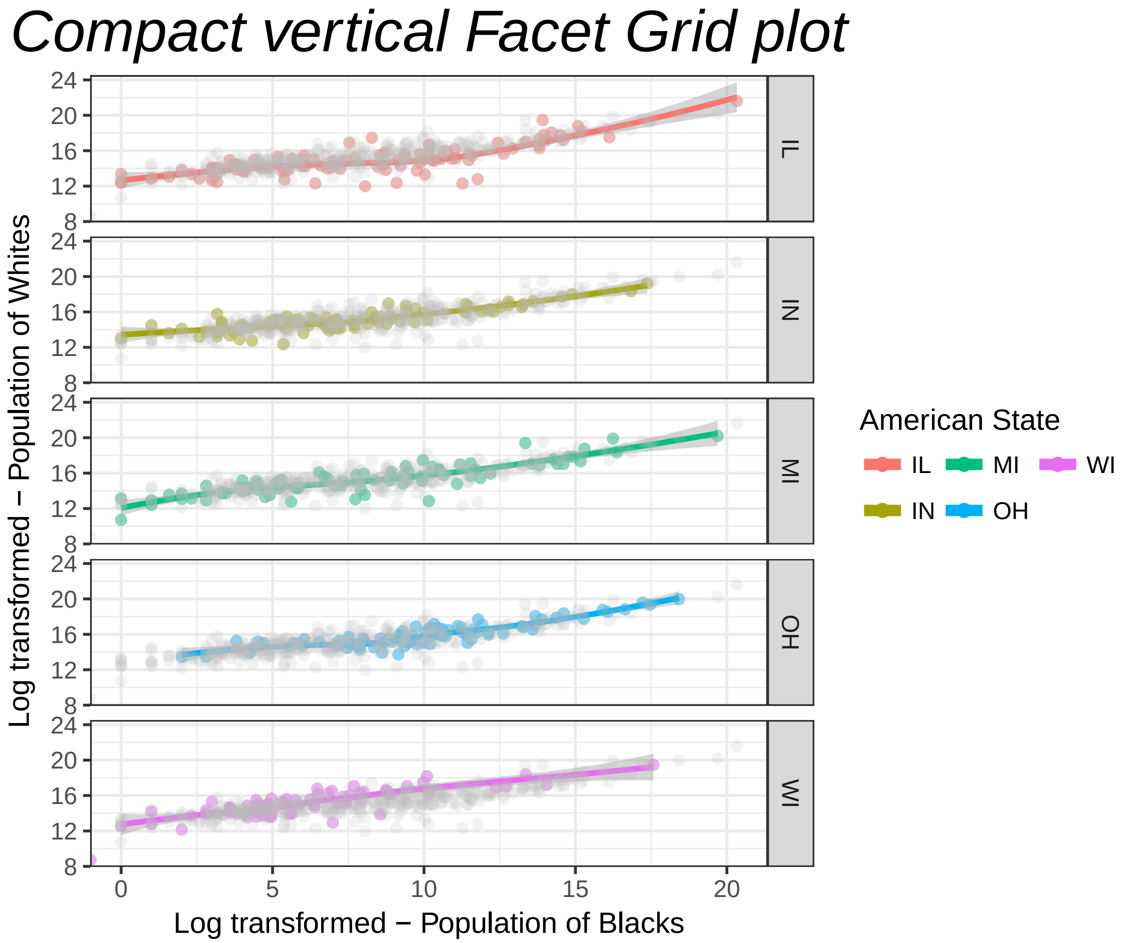
We have managed to make a plot that is separated by different points of interest, and looks generally readable. However, our legend is taking up a lot of space, and our title doesn't really stand out.
The next part, hence, is to stylize our plot.

The theme() element

The `theme()` element in ggplot lets us stylize specific parts of the plot's layout - specifically the following 4:
- `element_text()` : axis and graph titles and text
- `element_lines` : components like axis and grid lines
- `element_rect()` : rectangular components like plots and panel background
- `element_blank` : set other theme components to 'blank'

Firstly, I will edit the text styling for our plot title. This can be done by defining specific font-related attributes for plot.title in the `theme` component, as shown here:

```
g1_fg_clean_vertical_bg + theme(plot.title = element_text(size = 22, face = "italic",
    hjust = 0.5))
```

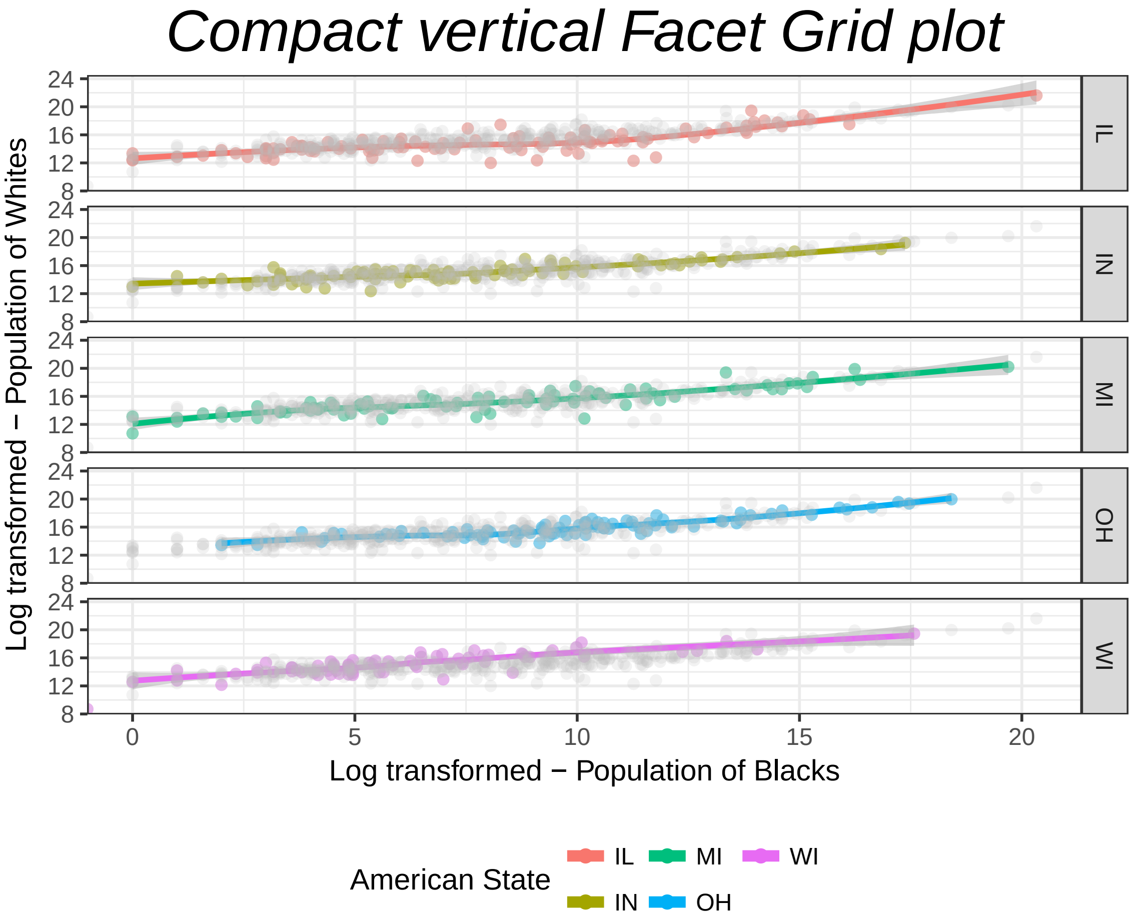


> If you are typing the command in the RStudio console, pressing 'tab' while you are within the element_text() part reveals other attributes you can customize.

Follow this link to learn more about customizing the font on your ggplots. It requires an additional package called *extrafont*.

Next, we can change the position of the legend. Notice how this one doesn't require any `element_??()`. This attribute is defined as one of the following options, `"none"`, `"left"`, `"right"`, `"bottom"`, `"top"`. Feel free to play around with this.

```
base_plt <- g1_fg_clean_vertical_bg + theme(plot.title = element_text(size = 22,
    face = "italic", hjust = 0.5), legend.position = "bottom")
base_plt
```



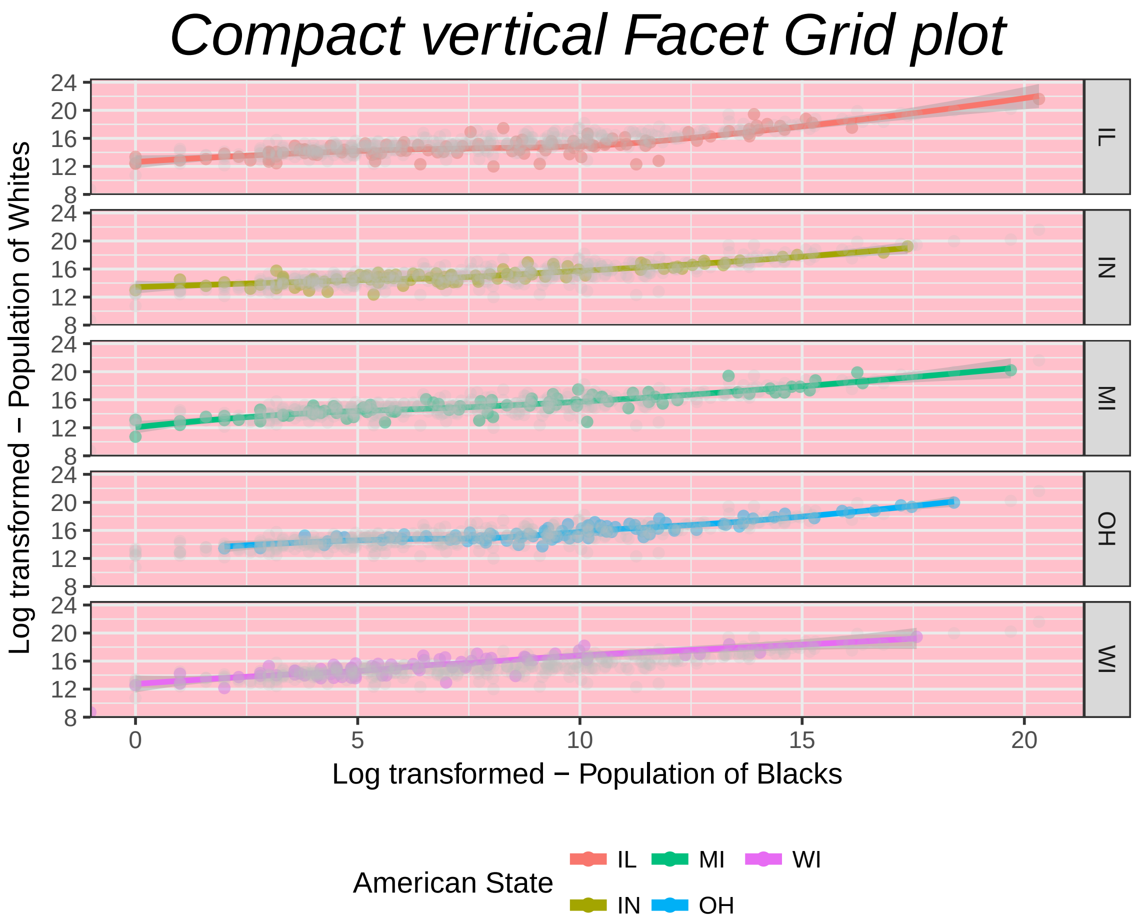
Notice how the keys are in two rows. This goes back to our `guides()` section, where we defined our colour legend to be printed in 2 rows (nrow=2).

Lastly, say I want to make the background of my plot **pink**. I will modify the `panel.background` option within `theme()`.

```
base_plt + theme(panel.background = "pink")
```

Does that work? Since `panel.background` is a 'rectangular attribute' that we are stylizing, we need to modify its characteristics using an `element_rect` option, like below:

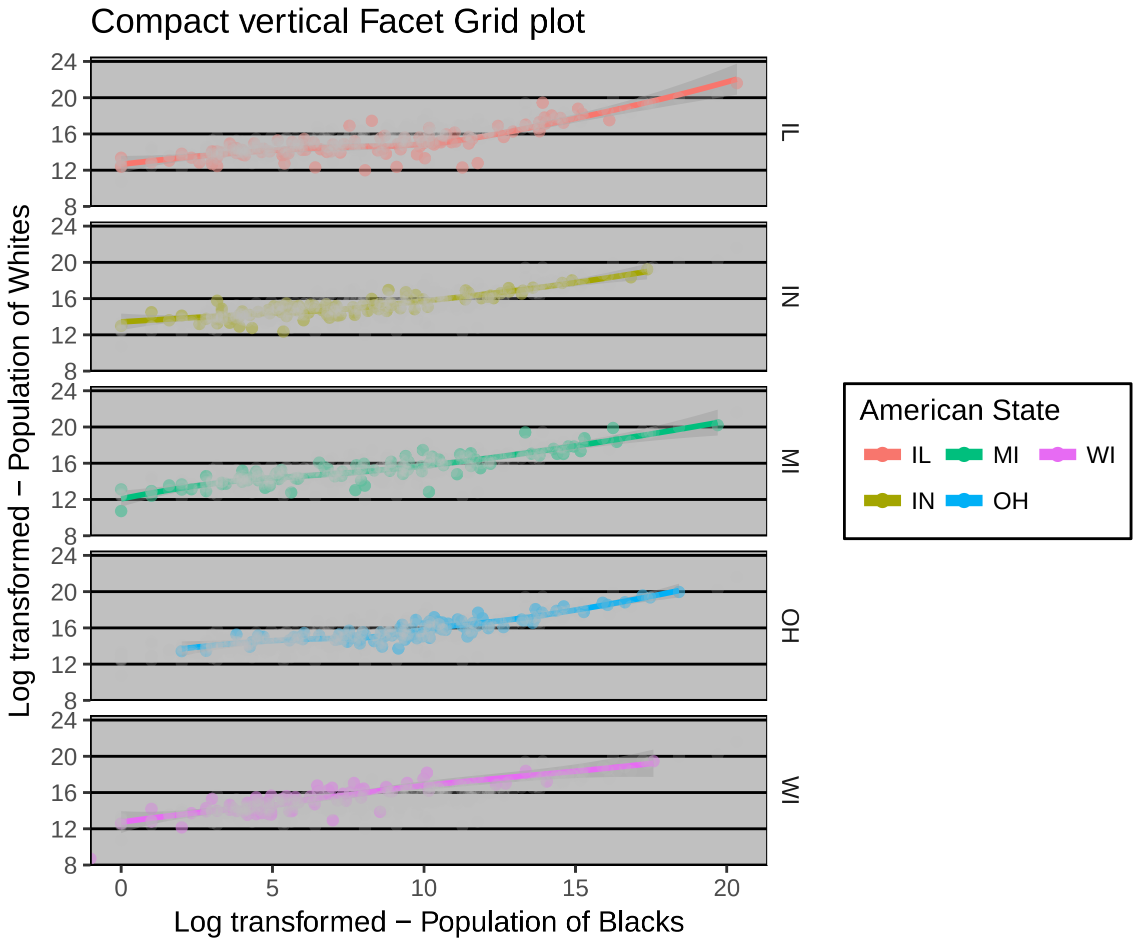
```
base_plt + theme(panel.background = element_rect("pink"))
```


Compact vertical Facet Grid plot
24
20
16
12
8
IL
IN
MI
OH
WI
Log transformed − Population of Whites
0
5
10
15
20
Log transformed − Population of Blacks
American State
IL
MI
WI
IN
OH

Make my plot look like it was made in Excel

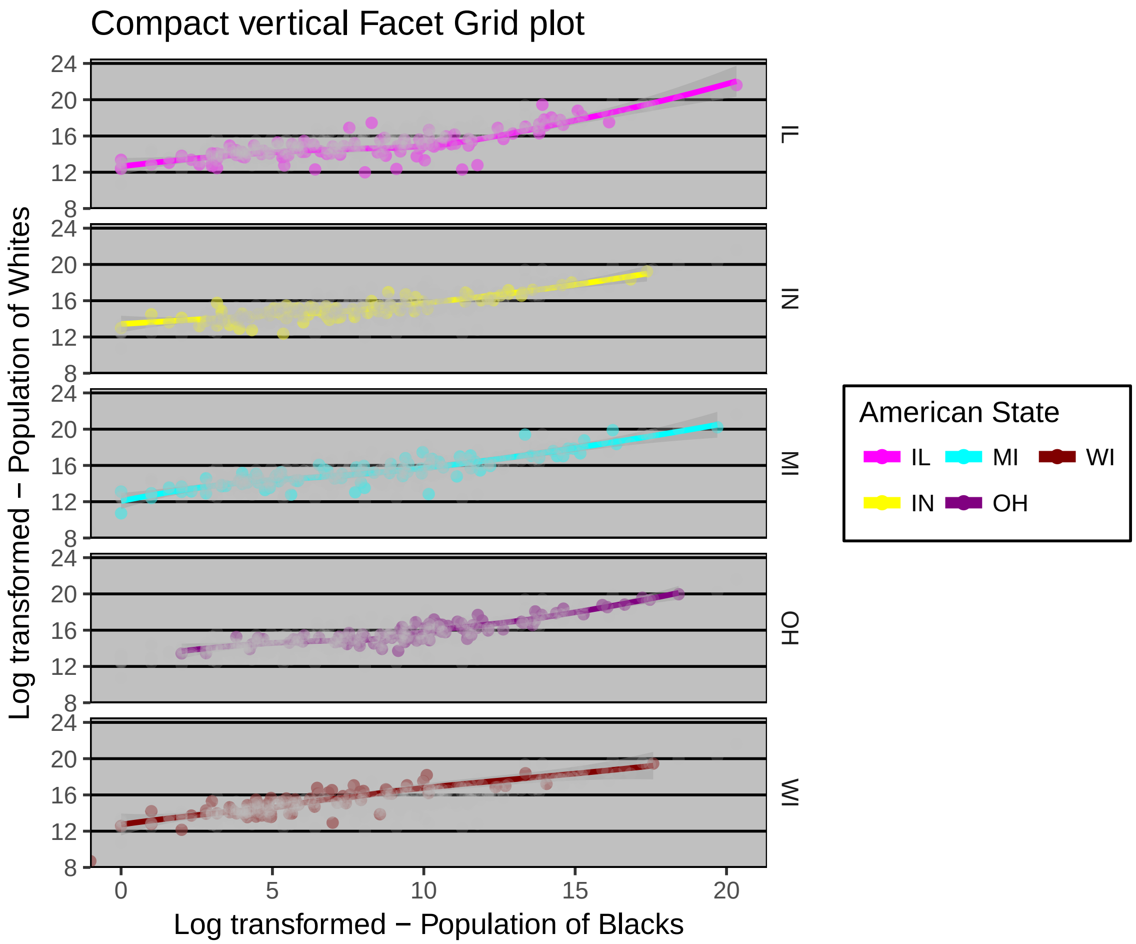
Sometimes, all these fancy shmancy customizations just aren't enough. **ggthemes** is a handy package that lets you quickly customize colour palettes and overall plot appearance using simple commands. The following is a brief sampling of what it has to offer.

```
library(ggthemes)
# Going for an Excel look
base_plt + theme_excel()
```

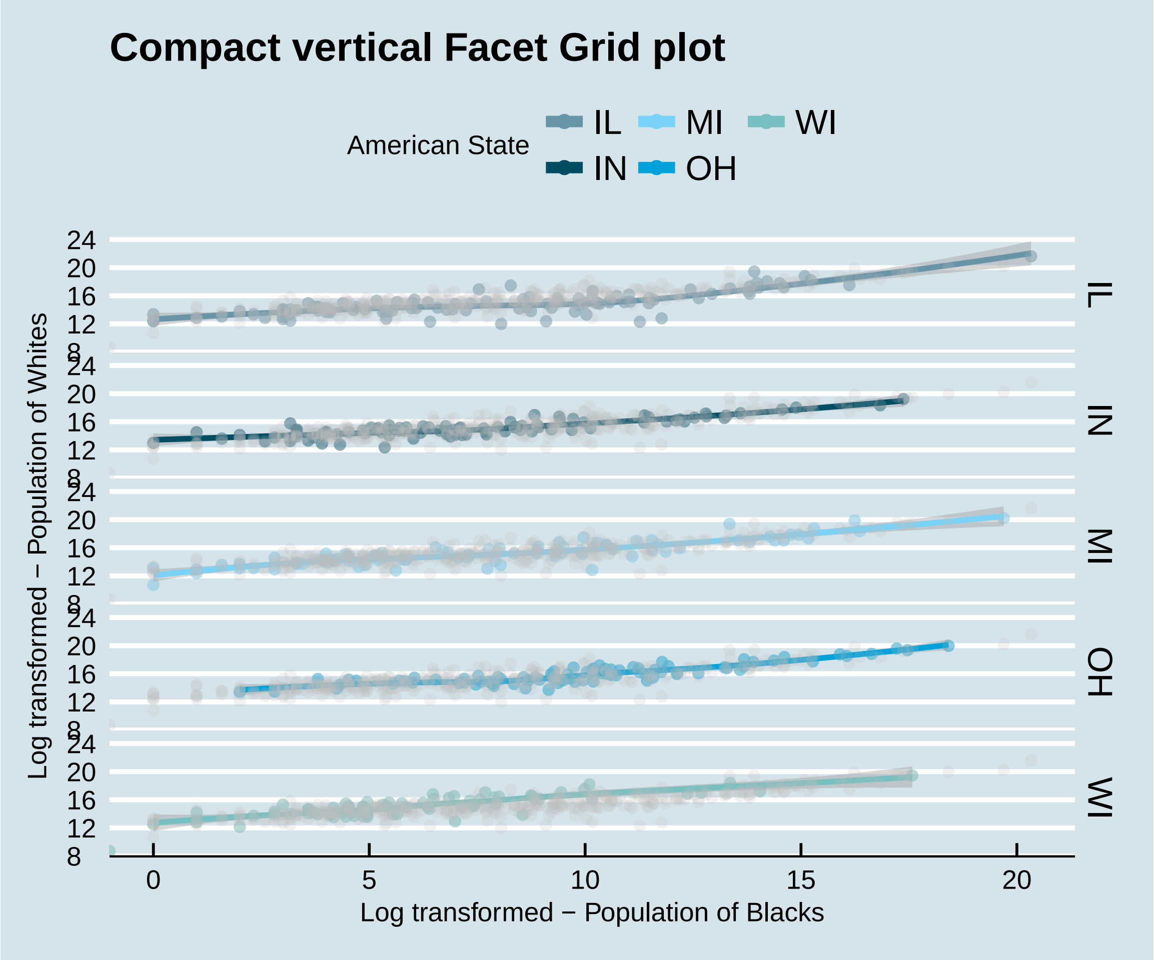


. . . as ugly as you would expect it to be.

```
# Going for an Excel look with Excel colours
base_plt + theme_excel() + scale_colour_excel()
```



```
# Going for an Economist look with different colours
base_plt + theme_economist() + scale_colour_economist()
```



Have fun!

Exercises

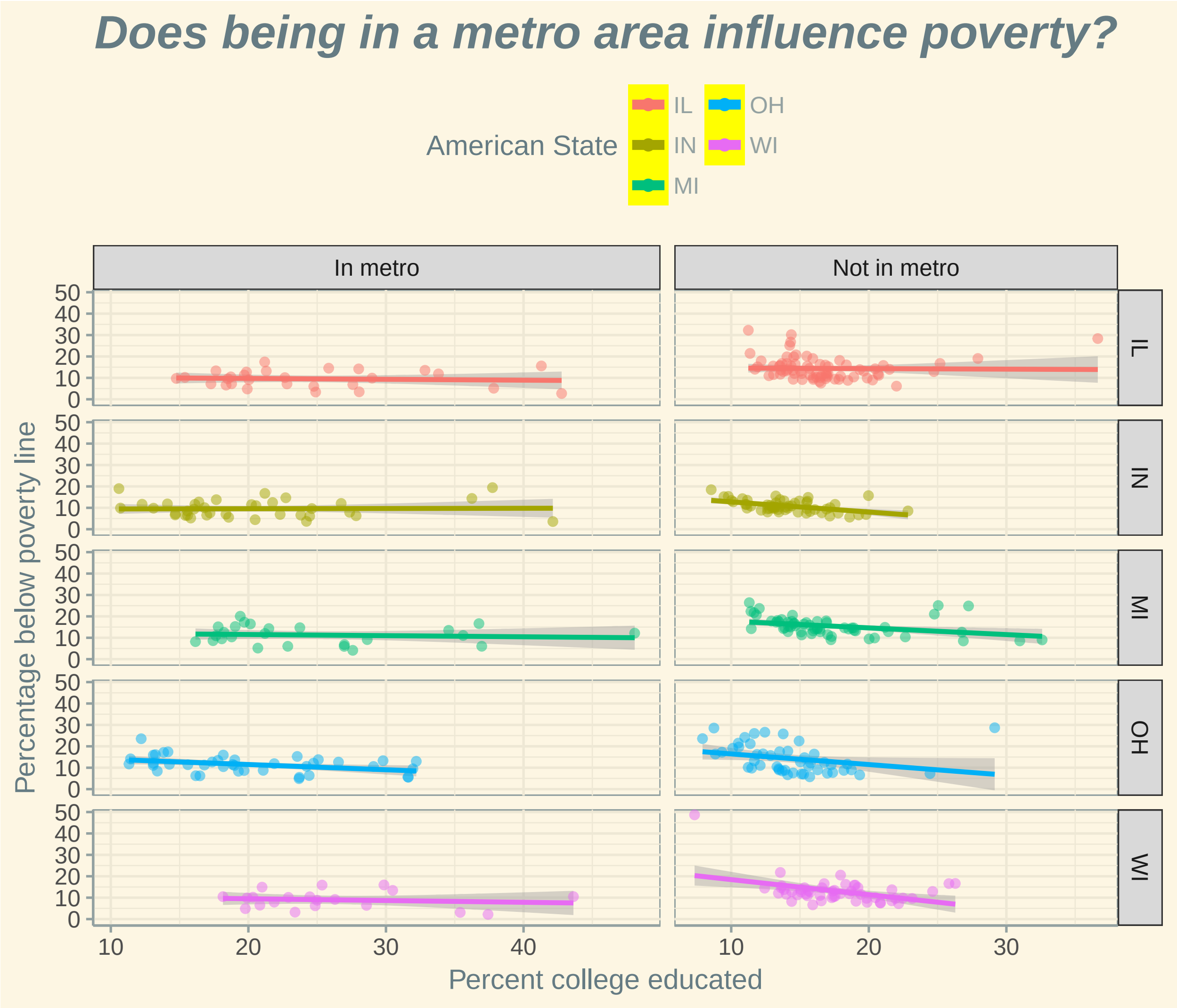
1. Can you trace back what the expanded plot definition in `base_plt` object is? That is, rewrite base_plt as the ggplot() definition it stands for.

Solution:

```
# Trace back to all objects defined
ggplot(midwest, aes(x = log2(popblack), y = log2(popwhite), colour = state)) +
    geom_point(alpha = 0.5, ) + theme_bw(base_size = 11) + stat_smooth() +
    labs(x = "Log transformed - Population of Blacks", y = "Log transformed - Population of Whites") +
    guides(colour = guide_legend(override.aes = list(alpha = 1, size = 2,
        fill = NA), title = "American State", nrow = 2)) + ggtitle("Compact vertical Facet Grid plot")
    facet_grid(state ~ ., scales = "free_x", space = "free_x") + geom_point(data = midwest[,
    c("popwhite", "popblack")], colour = "grey", alpha = 0.2) + theme(plot.title = element_text(size =
    face = "italic", hjust = 0.5), legend.position = "bottom")
```

2. Can you create the following plot?.

```
ggplot(midwest, aes(x = percollege, y = percbelowpoverty, colour = state)) +
    theme_solarized(base_size = 12) + stat_smooth(method = "lm") + geom_point(alpha = 0.5) +
    labs(x = "Percent college educated", y = "Percentage below poverty line") +
    guides(colour = guide_legend(override.aes = list(alpha = 1, size = 2,
        fill = "yellow"), title = "American State", ncol = 2)) + ggtitle("Does being in a metro area in
    facet_grid(state ~ ifelse(inmetro == 0, "Not in metro", "In metro"),
        scales = "free_x", space = "free") + theme(plot.title = element_text(size = 20,
    face = "bold.italic", hjust = 1), legend.position = "top")
```



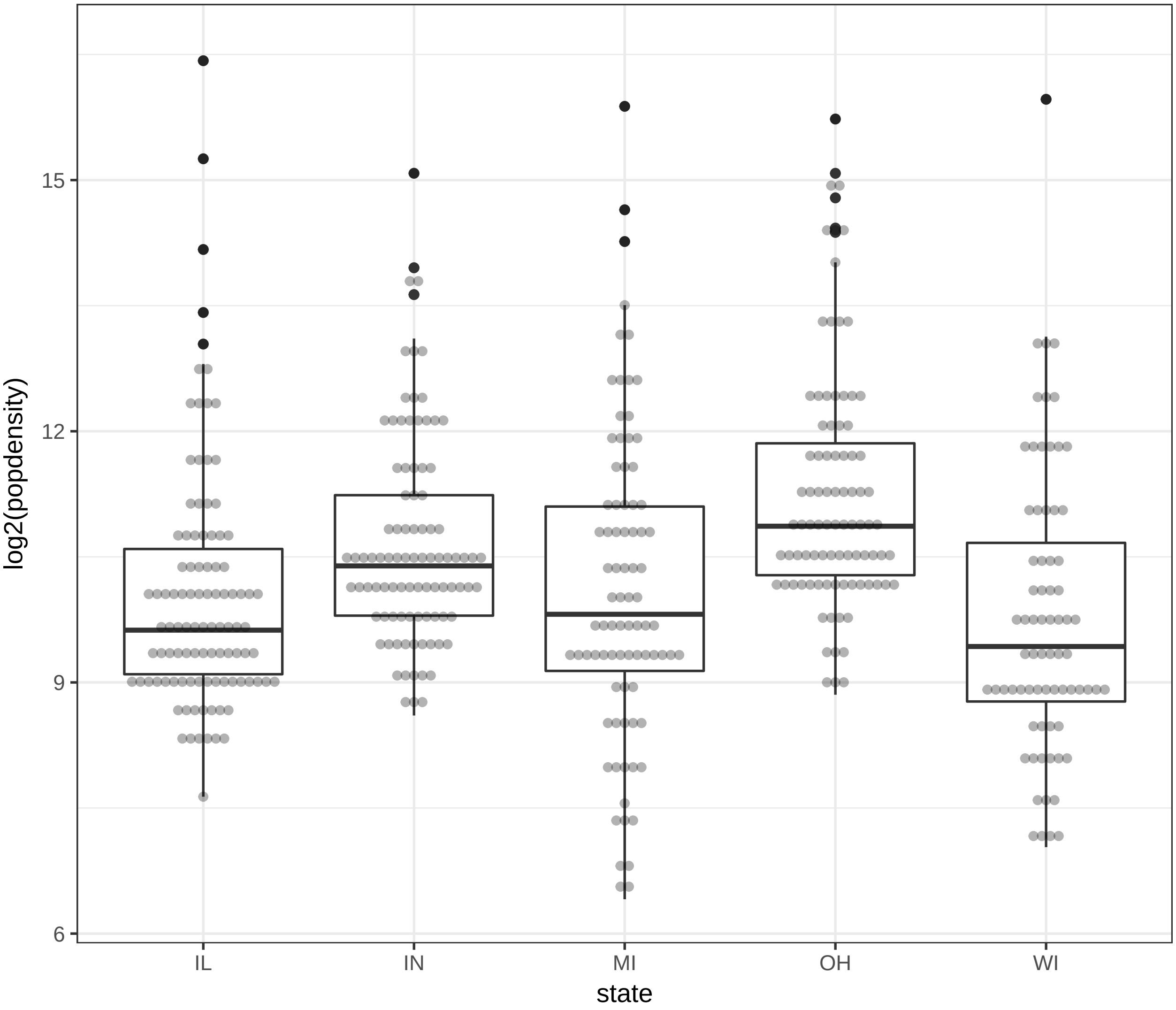
Hints:
- We are comparing 'percbelowpoverty' with 'percollege'
- The Metro variable is 'inmetro'; look up the usage of `ifelse()`
- I may not have used a **loess** fit

Solution will be covered in tutorial (time allowing), and posted here after tutorial.

Additional tid-bits

Towards the end of the tutorial I was asked how to add 'bee-swarm plots' with boxplots. If that's your jam, here is one way to do it:

```
ggplot(midwest, aes(x = state, y = log2(popdensity))) + geom_boxplot() +
    geom_dotplot(binaxis = "y", stackdir = "center", alpha = 0.3, dotsize = 0.3) +
    theme_bw()
```



Another way to do it is using the ggbeeswarm package, if you want other specialized types of 'swarms'.

You can also learn about other cool types of ggplot functions and figures using this cheat sheet

Take-aways

1. Saving ggplots as objects
2. Customize panels in ggplot
3. Highlight specific data points
4. Stylize parts of a plot
5. Automating ggplot makeovers with ggthemes package