NYC DATA SCIENCE ACADEMY

Customizing Graphics in ggplot2

Data Science Bootcamp

Outline

- Customizing Graphics
- Titles
- Coordinate systems
- Scales
- Themes
- Axis labels
- Legends
- Other visualizations

Outline

- **Customizing Graphics**
  - Titles
  - Coordinate systems
  - Scales
  - Themes
  - Axis labels
  - Legends
  - Other visualizations

Customizing Graphics

Customizing Graphics

Texas population data

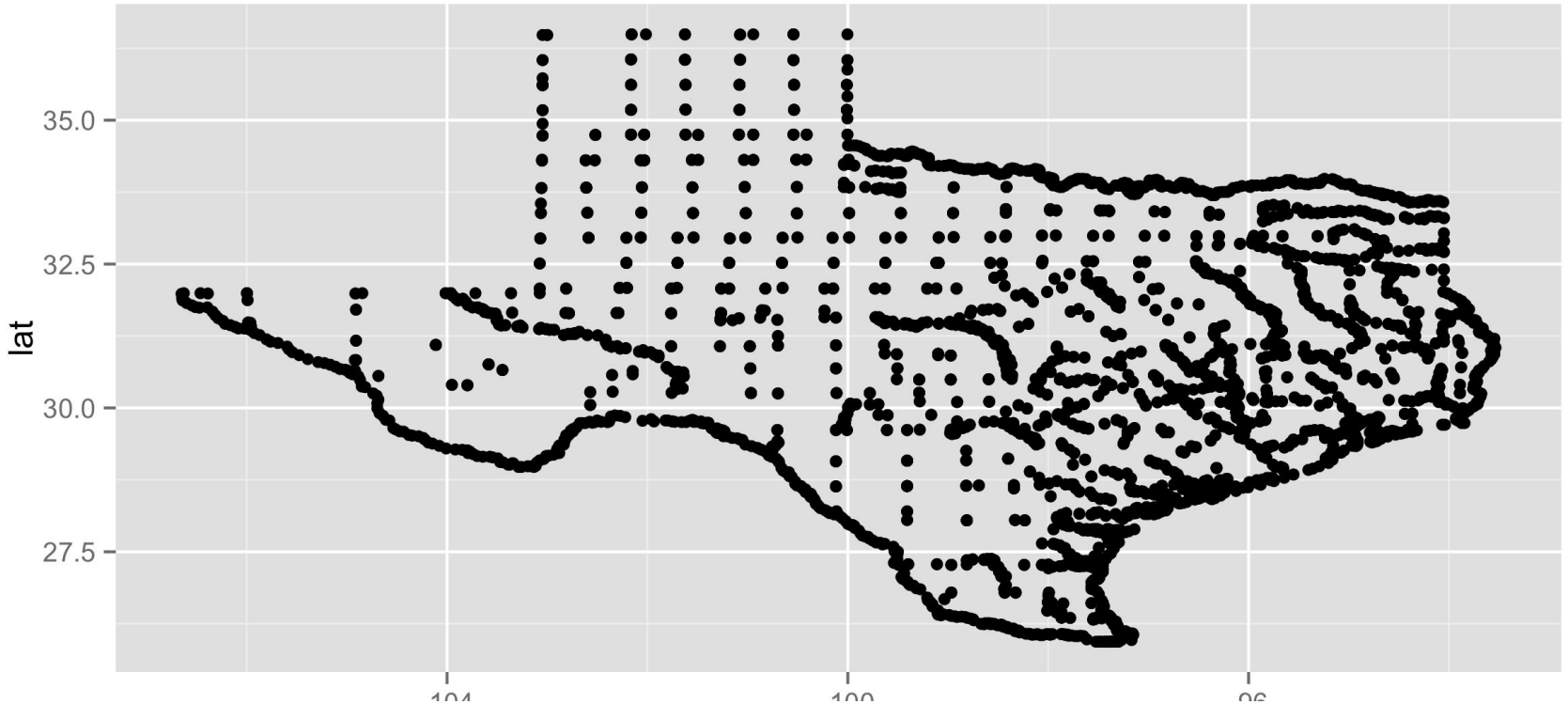
- For today, we will use a graph based on the Texas data set as an example; it displays the populations of counties in Texas.

```
library(ggplot2)
texas = read.csv("texas.csv") #Change to your working directory.
head(texas)
```

	long	lat	group	order	state	county	pop	bin
1	-95.8	31.5	1	1	texas	anderson	56474	< 1e5
2	-95.8	31.6	1	2	texas	anderson	56474	< 1e5
3	-95.8	31.6	1	3	texas	anderson	56474	< 1e5
4	-95.7	31.6	1	4	texas	anderson	56474	< 1e5
5	-95.7	31.6	1	5	texas	anderson	56474	< 1e5
6	-95.7	31.6	1	6	texas	anderson	56474	< 1e5

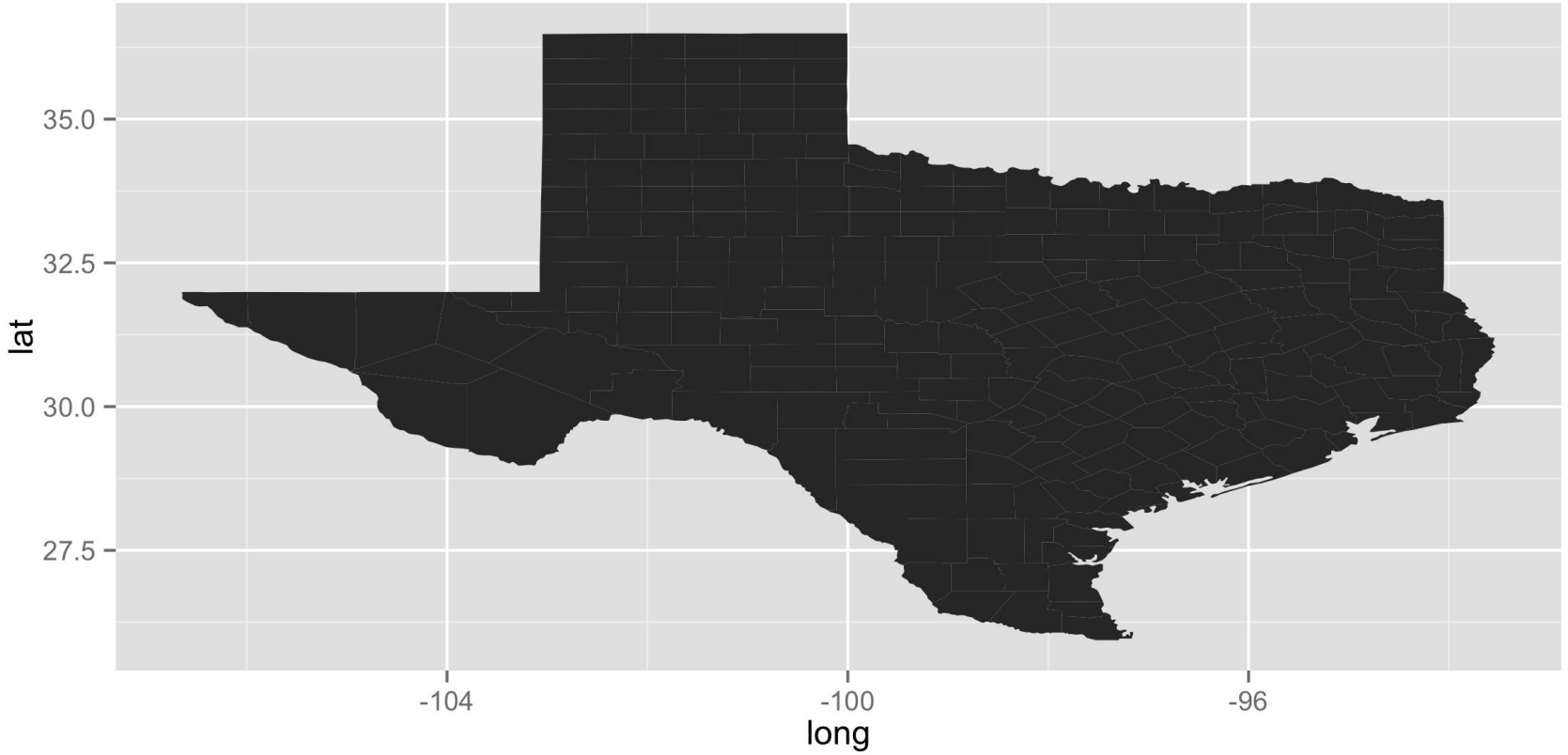
Customizing Graphics

```
g <- ggplot(data = texas, aes(x = long, y = lat))
g + geom_point()  #What's going on here?
```



Customizing Graphics

```
g + geom_polygon(aes(group = group))
```



Customizing Graphics

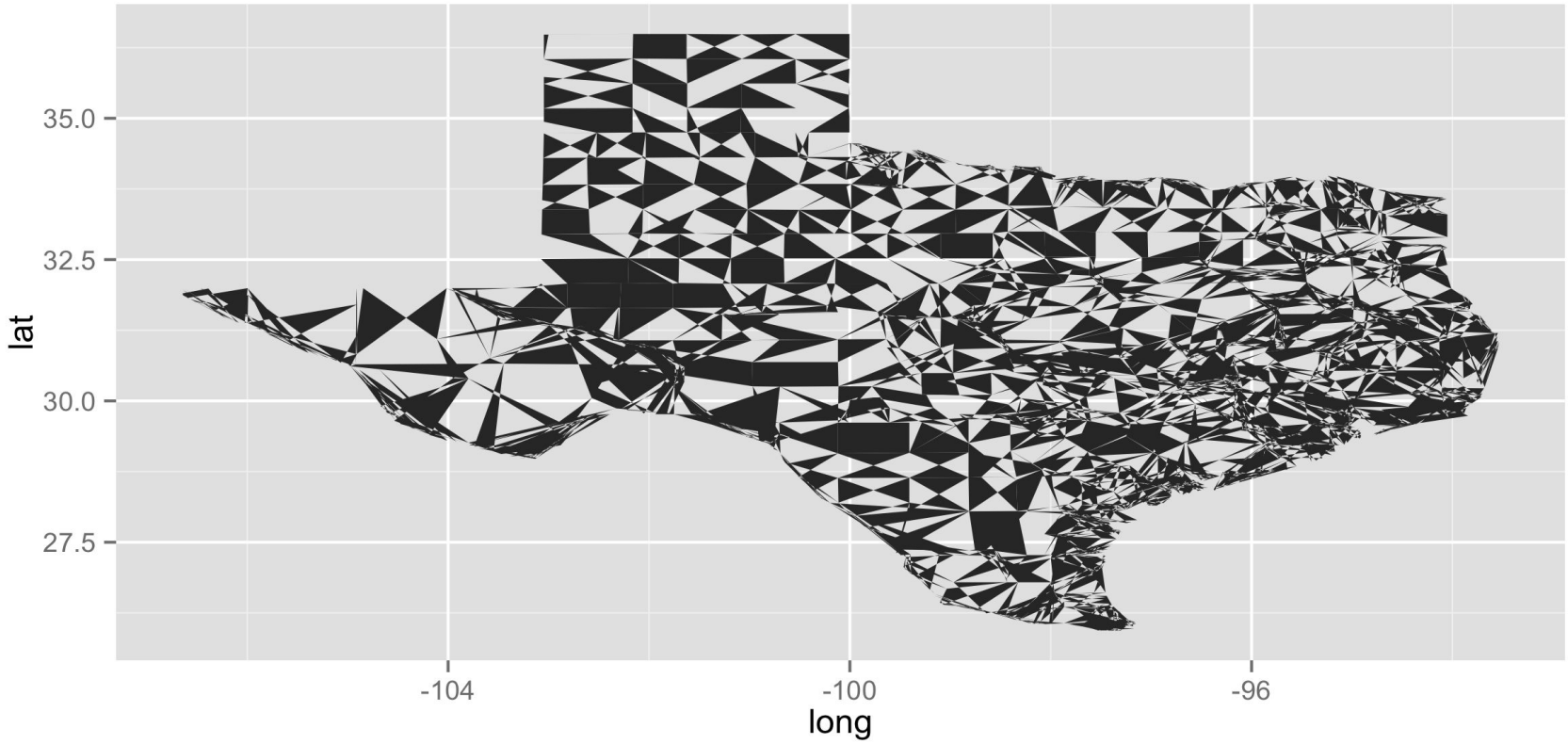
```
head(texas)
```

	long	lat	group	order	state	county	pop	bin
1	-95.8	31.5	1	1	texas	anderson	56474	< 1e5
2	-95.8	31.6	1	2	texas	anderson	56474	< 1e5
3	-95.8	31.6	1	3	texas	anderson	56474	< 1e5
4	-95.7	31.6	1	4	texas	anderson	56474	< 1e5
5	-95.7	31.6	1	5	texas	anderson	56474	< 1e5
6	-95.7	31.6	1	6	texas	anderson	56474	< 1e5

- group:
  - Groups points into different polygons.
- order:
  - The row number, marking the order of points to plot.

Customizing Graphics

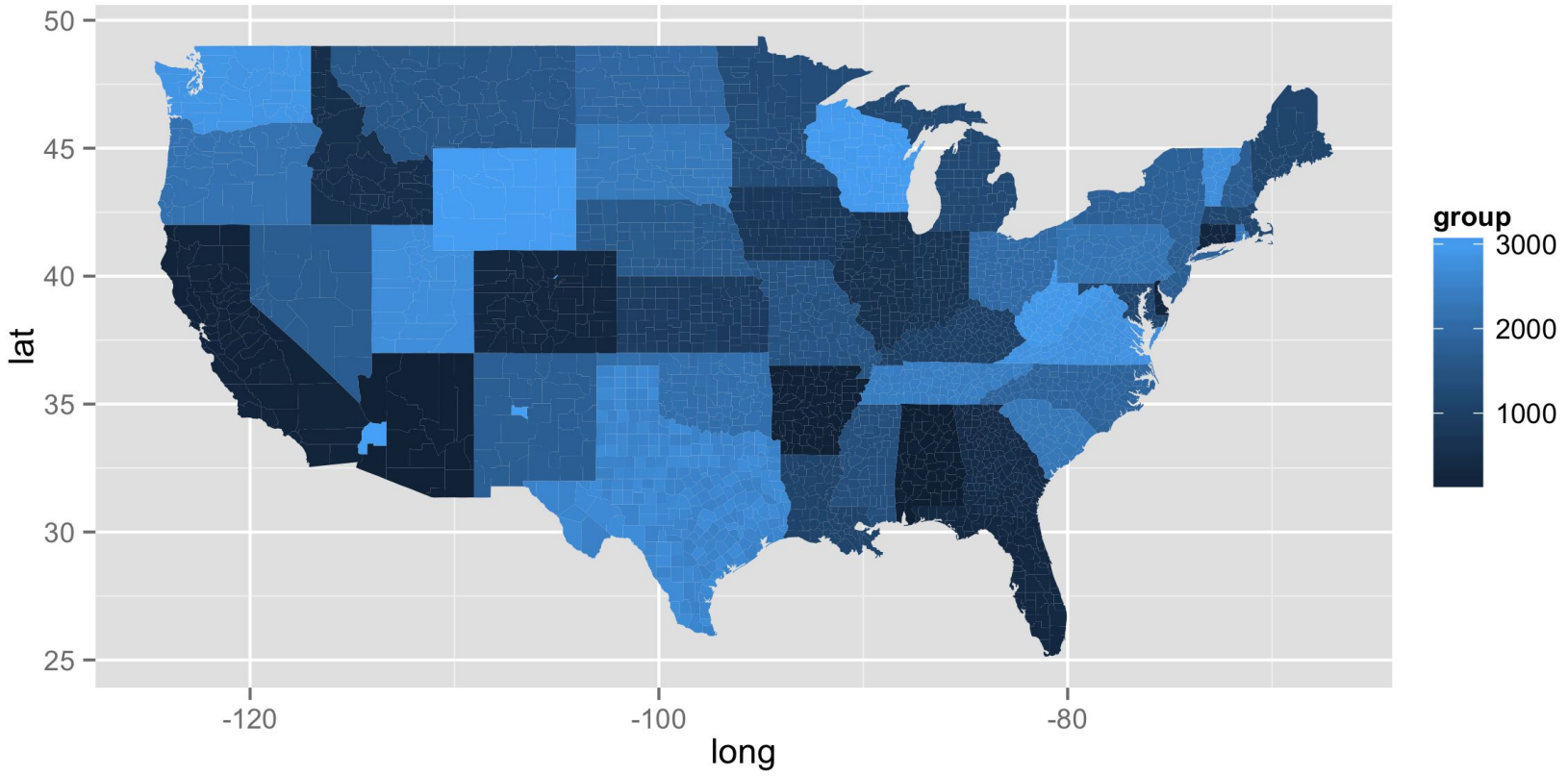
```
texas2 = texas[sample(nrow(texas)), ]  #Row order matters!
ggplot(data = texas2, aes(x = long, y = lat)) + geom_polygon(aes
(group = group))
```



Customizing Graphs

```
# install.packages("maps")
# help(package = "maps")
library(maps)
counties = map_data("county")    # Using the built-in USA county
                                 # map dataset.
ggplot(data = counties, aes(x = long, y = lat)) + geom_polygon
(aes(group = group, fill = group))
```

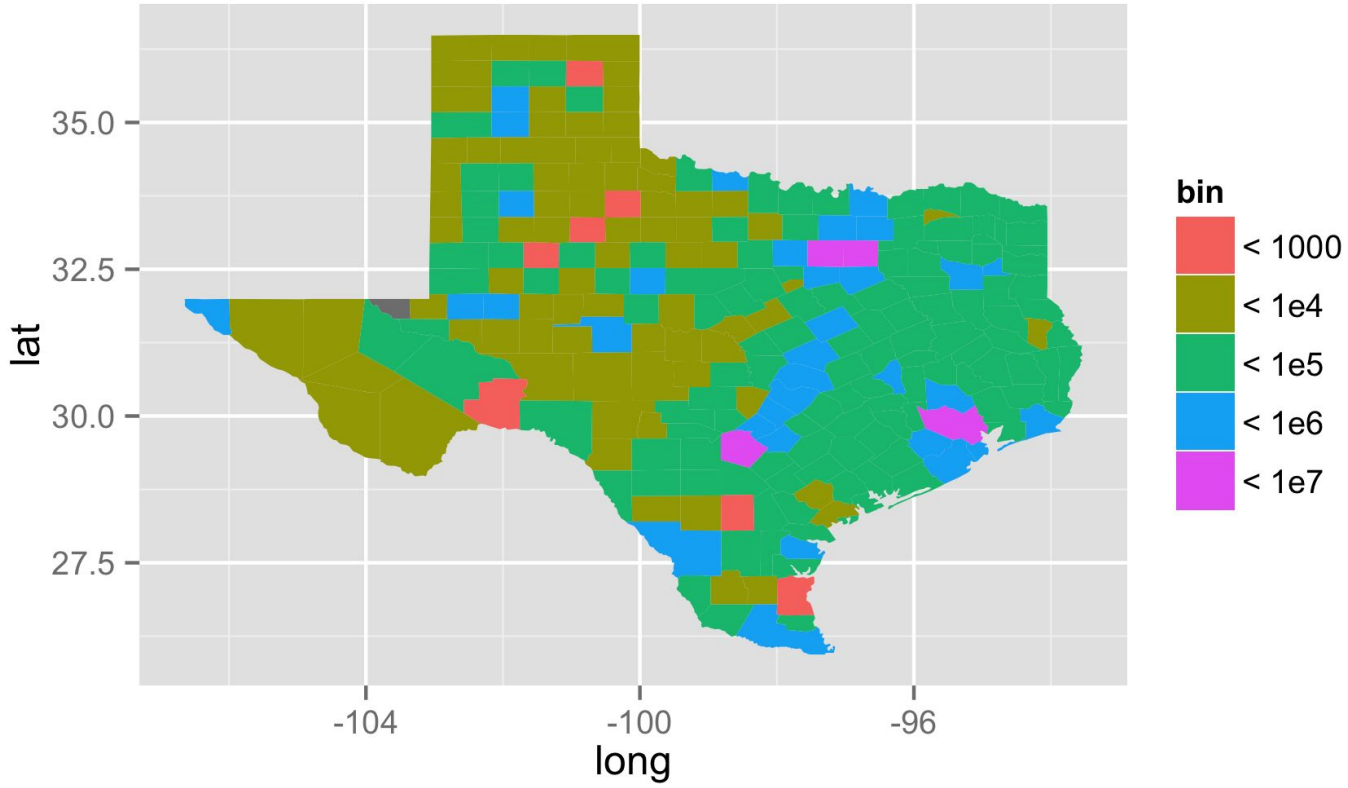
Customizing Graphs



Customizing Graphs

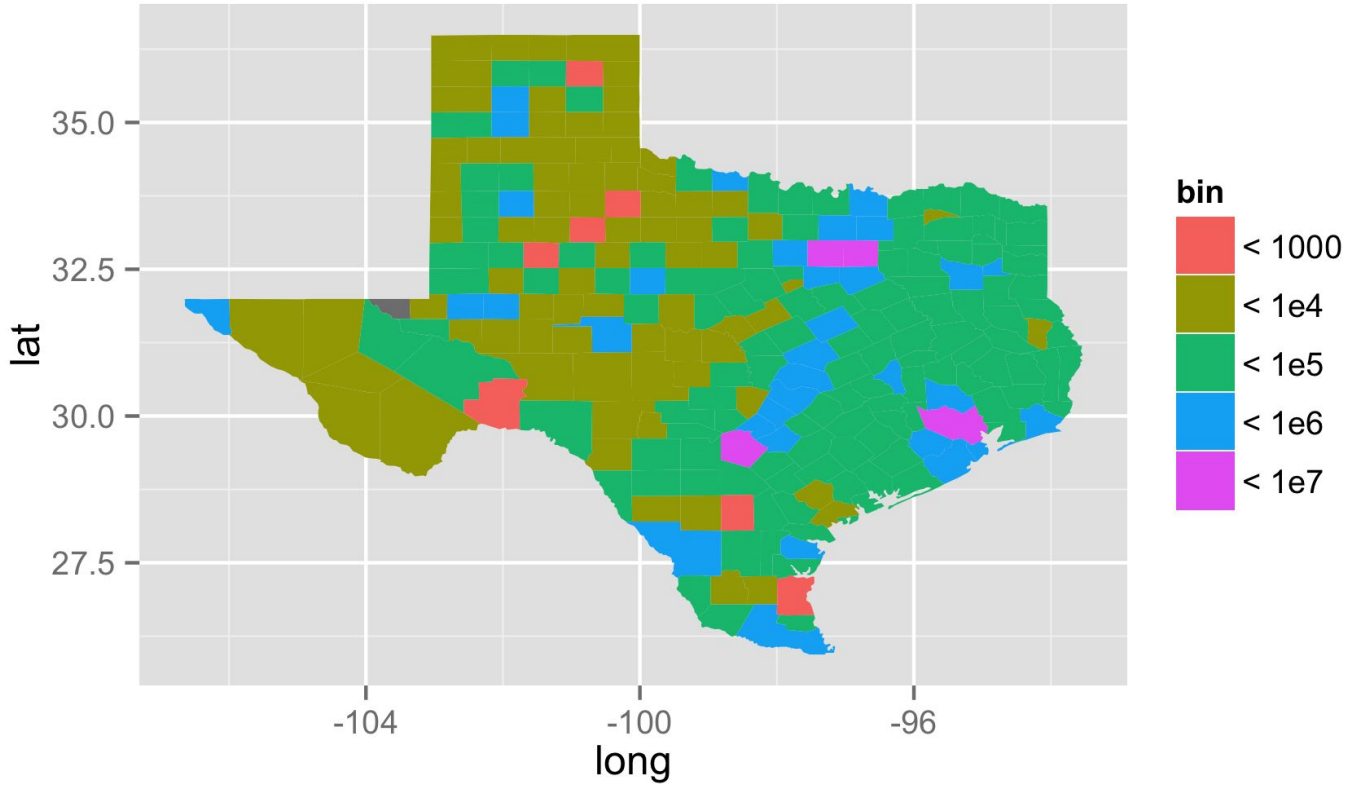
Your turn:

Use the `texas` dataset to recreate this map:



Customizing Graphs

```
ggplot(texas, aes(x = long, y = lat)) +
  geom_polygon(aes(group = group, fill = bin))
```



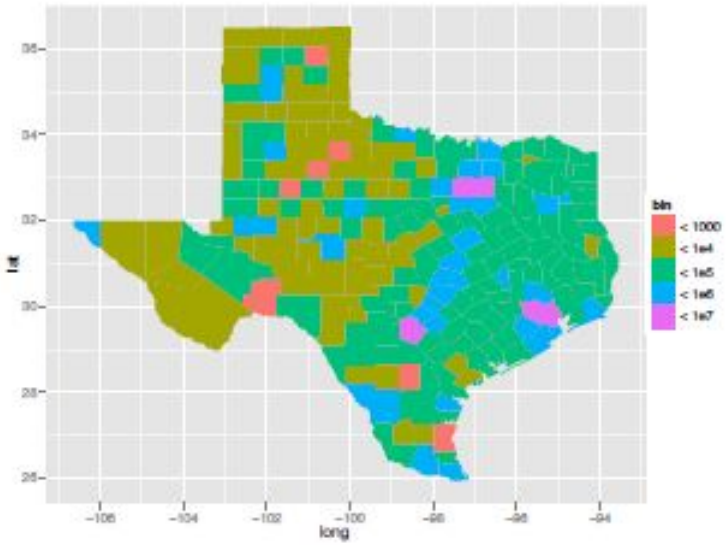
Customizing Graphs

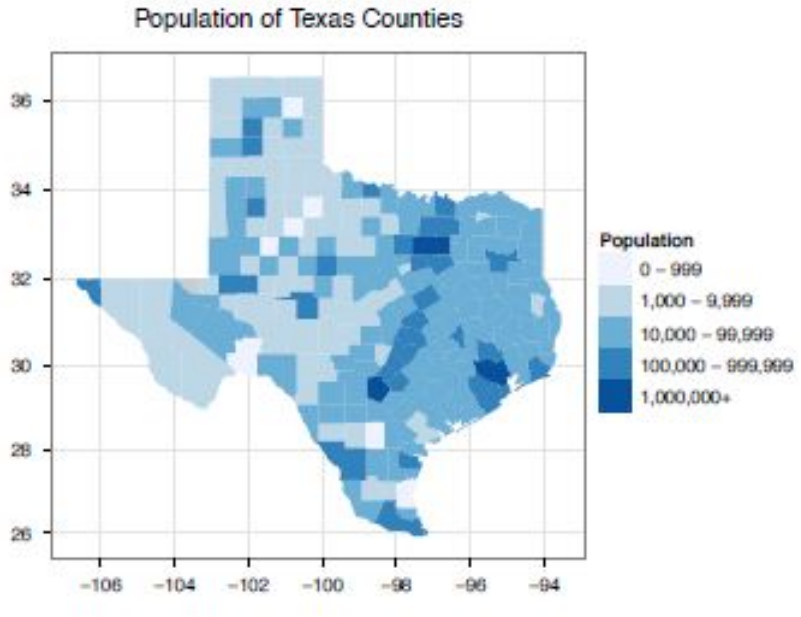
- Let's save this plot into the object `tx`; this will make it easy to refer to the plot later:

```
tx = ggplot(texas, aes(x = long, y = lat)) +
  geom_polygon(aes(group = group, fill = bin))
```

Customizing Graphs

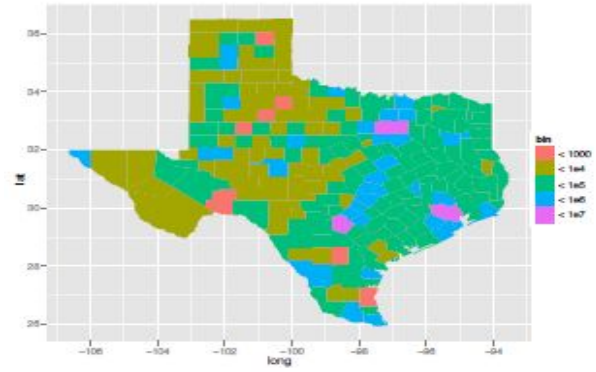
Which do you prefer? What differs between the plots?

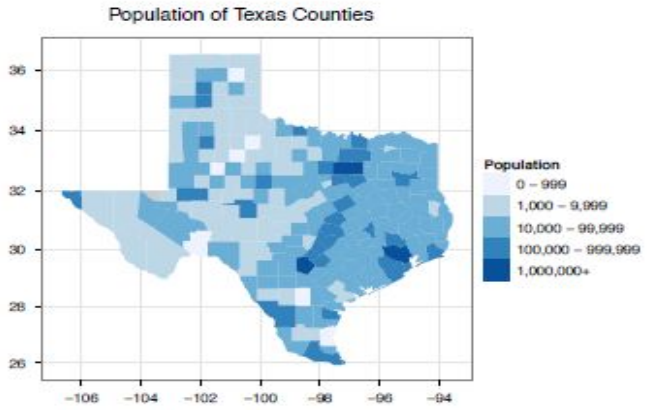




What are the Differences?

- A title has been added.
- Axis labels have been removed.
- The color scheme has been changed.
- The background scheme has been changed.
- The legend labels have been modified.
- The aspect ratio has been changed.





Outline

- Customizing Graphics
- **Titles**
- Coordinate systems
- Scales
- Themes
- Axis labels
- Legends
- Other visualizations

Titles

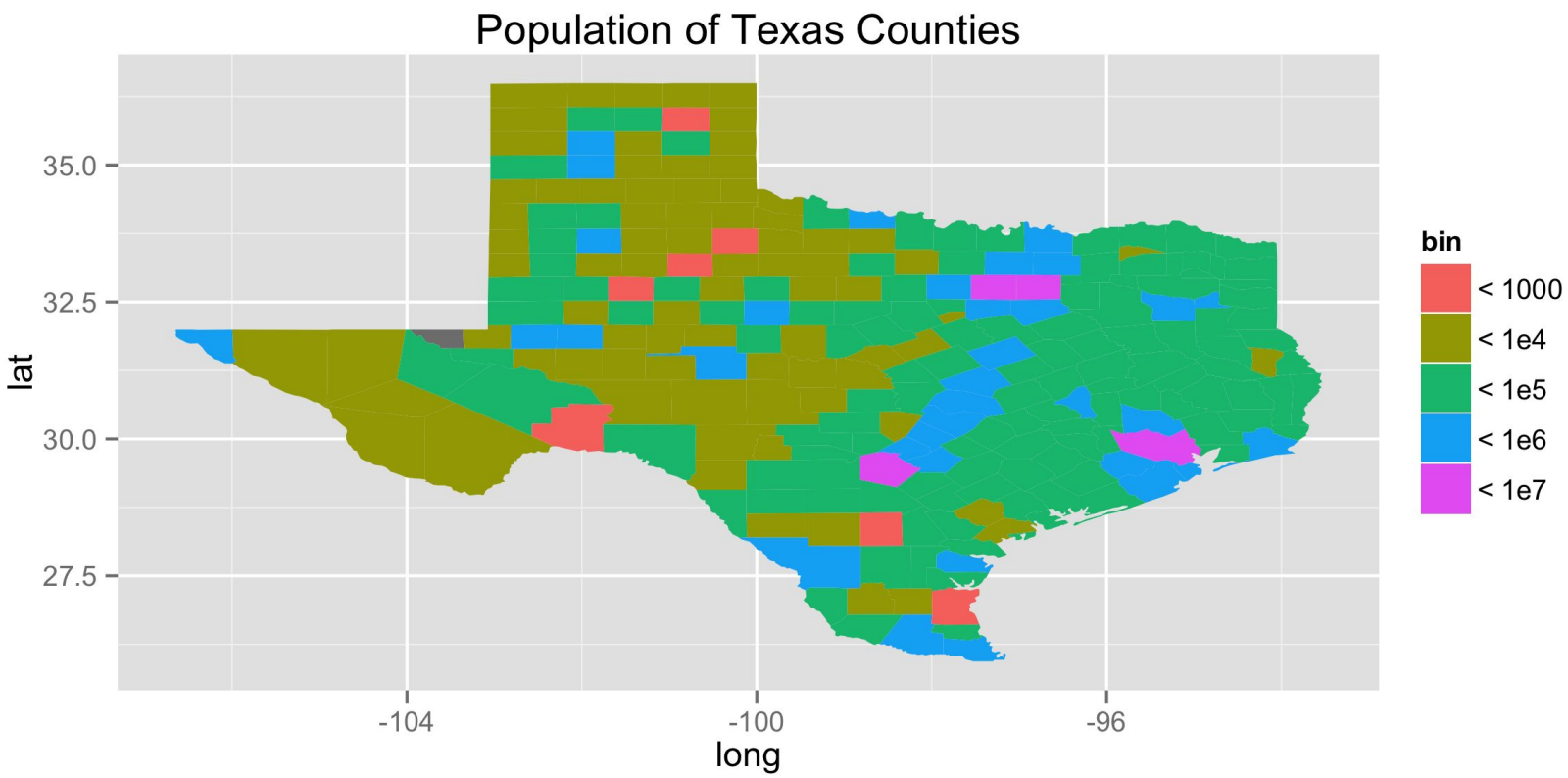
Title

- Recall that you can modify ggplot2 graphs by adding objects to them.
- Use the function ggtitle() to add a title.
- Use + to add it to graph.

```
tx + ggtitle("Population of Texas Counties")
#Creates a ggplot2 title, and adds it to the tx graph.
```

Title

Now the graph has a title:



Title

Remember, additions are not permanent!

- To create a new graph that always has a title, we have to save the object:

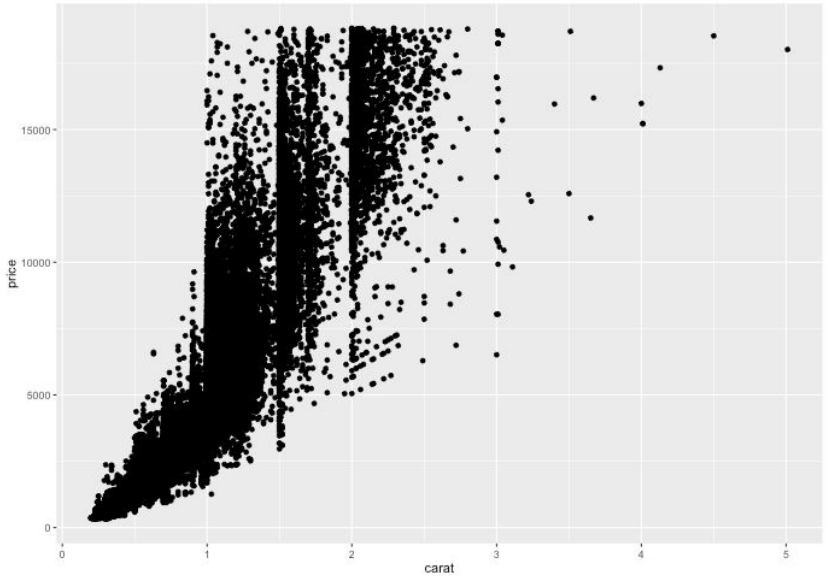
```
tx + ggtitle("Population of Texas Counties")
tx
#Resulting graphic from tx has no title.

tx2 <- tx + ggtitle("Population of Texas Counties")
tx2
#Resulting graphic from tx2 has a title.
```

Manipulating Plots

Manipulating Plots

Let's explore what exactly is in a ggplot object.

```
g <- ggplot(data = diamonds, aes(x = carat, y = price)) +
geom_point()
g
str(g)
```



Manipulating Plots

```
# List of 9
#  $ data       :'data.frame':    53940 obs. of  10 variables:
#   ..$ carat  : num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
#   ..$ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
#   ..$ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
#   ..$ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
#   ..$ depth  : num [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
#   ..$ table  : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...
#   ..$ price  : int [1:53940] 326 326 327 334 335 336 336 337 337 338 ...
#   ..$ x      : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
#   ..$ y      : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
#   ..$ z      : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
#  $ layers     :List of 1
#   ..$ :Classes 'proto', 'environment' <environment: 0x7fcf5d08da38>
#  $ scales     :Reference class 'Scales' [package "ggplot2"] with 1 fields
#   ..$ scales: list()
#   ..and 21 methods, of which 9 are possibly relevant:
#   ..  add, clone, find, get_scales, has_scale, initialize, input, n, non_position_scales
#  $ mapping    :List of 2
#   ..$ x: symbol carat
#   ..$ y: symbol price
#  $ theme      : list()
#  $ coordinates:List of 1
#   ..$ limits:List of 2
#   .. ..$ x: NULL
#   .. ..$ y: NULL
#   ..- attr(*, "class")= chr [1:2] "cartesian" "coord"
#  $ facet      :List of 1
#   ..$ shrink: logi TRUE
#   ..- attr(*, "class")= chr [1:2] "null" "facet"
#  $ plot_env   :<environment: R_GlobalEnv>
#  $ labels     :List of 2
#   ..$ x: chr "carat"
#   ..$ y: chr "price"
#  - attr(*, "class")= chr [1:2] "gg" "ggplot"#
```

We are going to override things that are already here as defaults

Outline

- Customizing Graphics
- Titles
- **Coordinate systems**
- Scales
- Themes
- Axis labels
- Legends
- Other visualizations

Coordinate Systems

Coordinate Systems

In the structure of the graph object we saw many different components. Let's take a quick look at just one of them:

```
g$coordinates

<ggproto object: Class CoordCartesian, Coord>
    aspect: function
    distance: function
    expand: TRUE
    is_linear: function
    labels: function
    limits: list
    range: function
    render_axis_h: function
    render_axis_v: function
    render_bg: function
    render_fg: function
    train: function
    transform: function
    super:  <ggproto object: Class CoordCartesian, Coord>
```

Coordinate Systems

Determine the coordinate plane on which to draw the graph, then follow the following pattern:

```
coord_cartesian()
```

- coord_
  - Always begins with coord_...
- cartesian
  - ...followed by the coordinate system's name...
- ()
  - ...and open and closed parentheses.

Coordinate Systems

```
g2 <- g + coord_polar()
```

Now compare

```
g$coordinates
```

with

```
g2$coordinates
```

Coordinate Systems

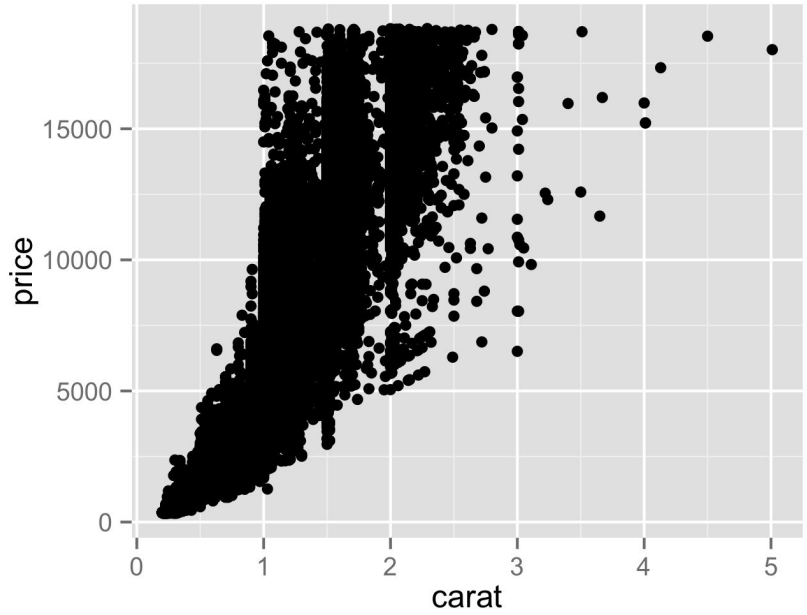
In the structure of the new graph object, the coordinate component changed quite a bit:

```
g2$coordinates
<ggproto object: Class CoordPolar, Coord>
    aspect: function
    direction: 1
    distance: function
    is_linear: function
    labels: function
    r: y
    range: function
    render_axis_h: function
    render_axis_v: function
    render_bg: function
    render_fg: function
    start: 0
    theta: x
    train: function
    transform: function
    super:  <ggproto object: Class CoordPolar, Coord>
```

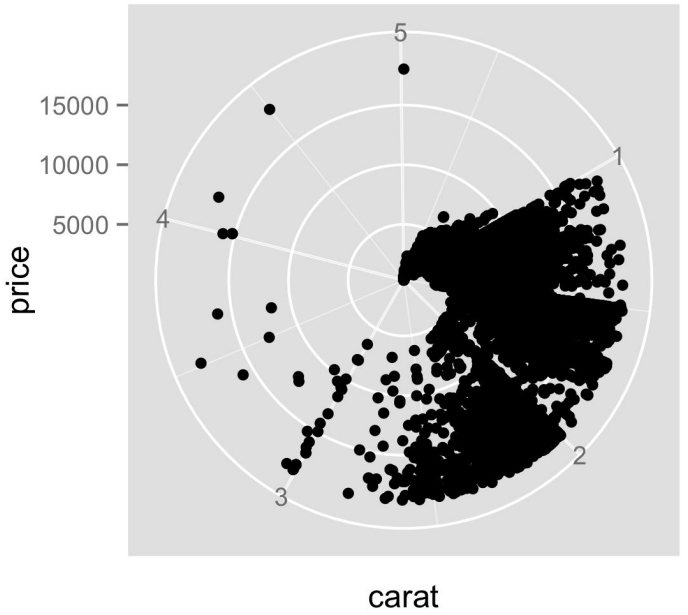
Coordinate Systems

polar

```
g
```



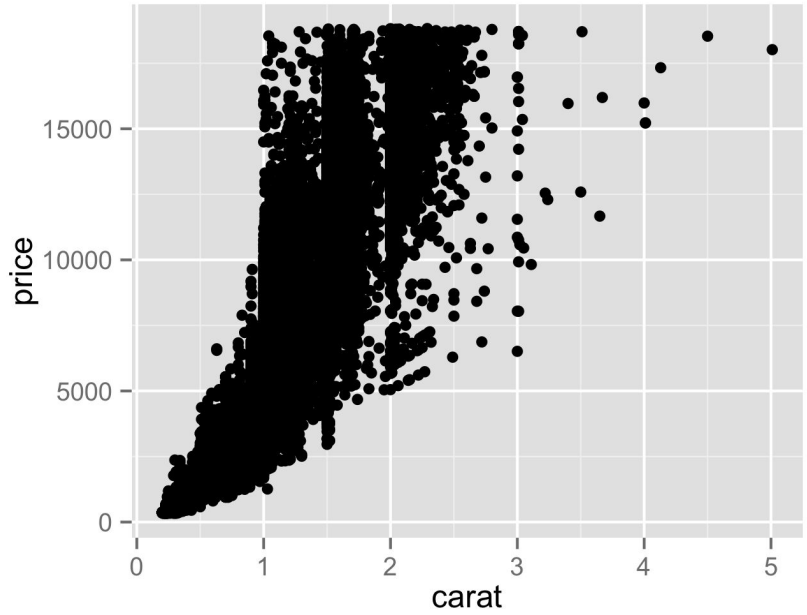

```
g + coord_polar()
```



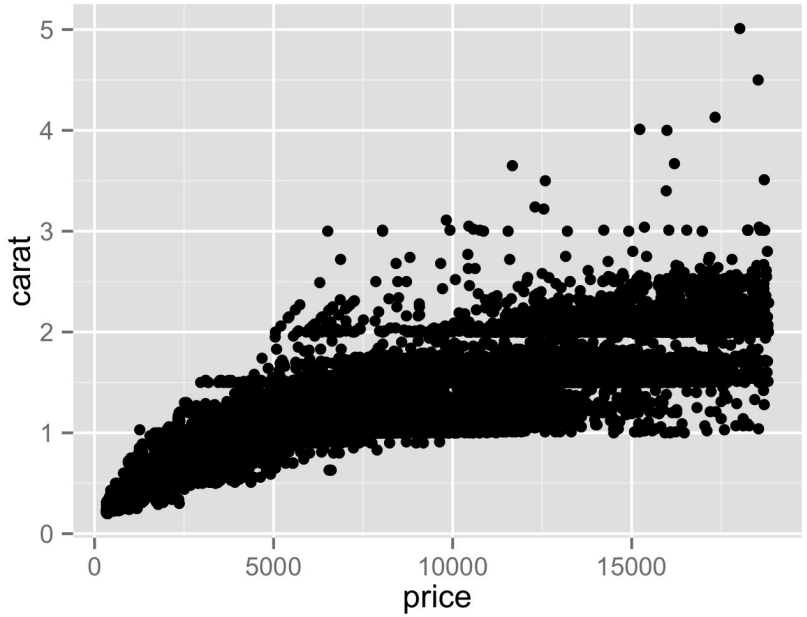
Coordinate Systems

flip

```
g
```



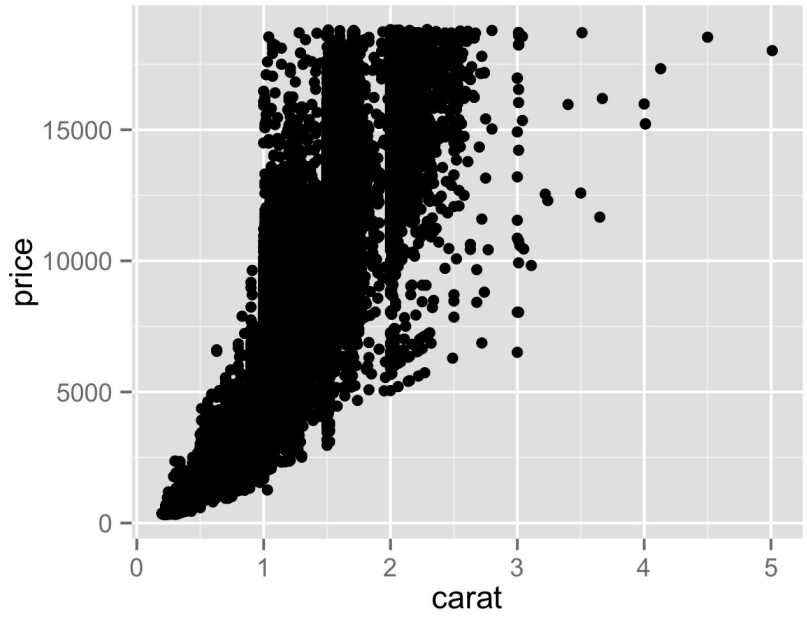

```
g + coord_flip()
```



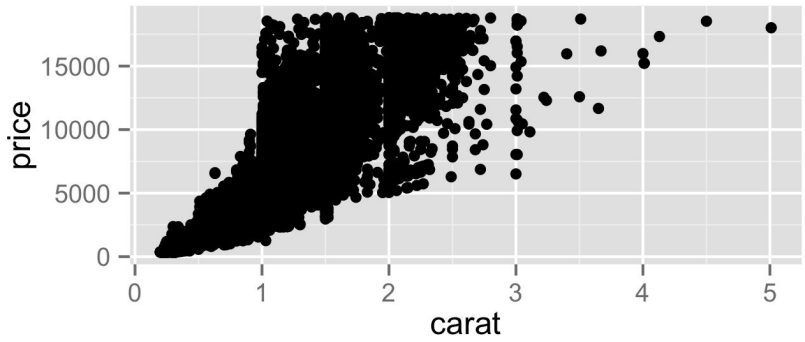
Coordinate Systems

fixed: Cartesian coordinates with a fixed aspect ratio between x and y units.

```
g
```



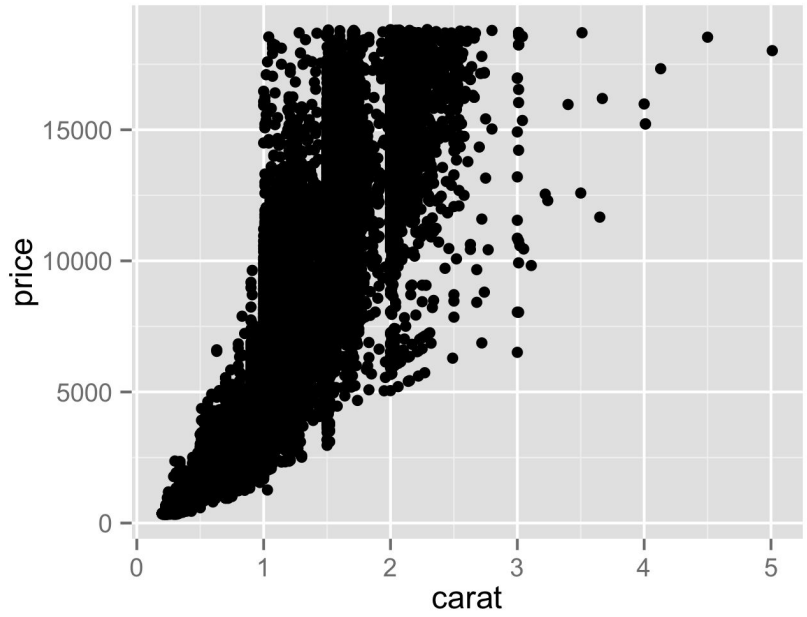

```
g + coord_fixed(ratio =
1/10000)
```



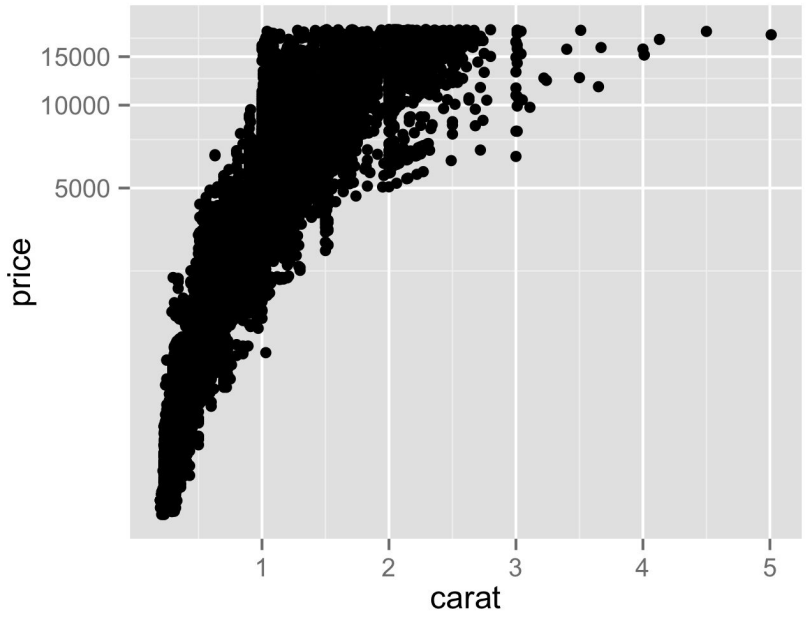
Coordinate Systems

trans

```
g
```




```
g + coord_trans(y = "log10")
```

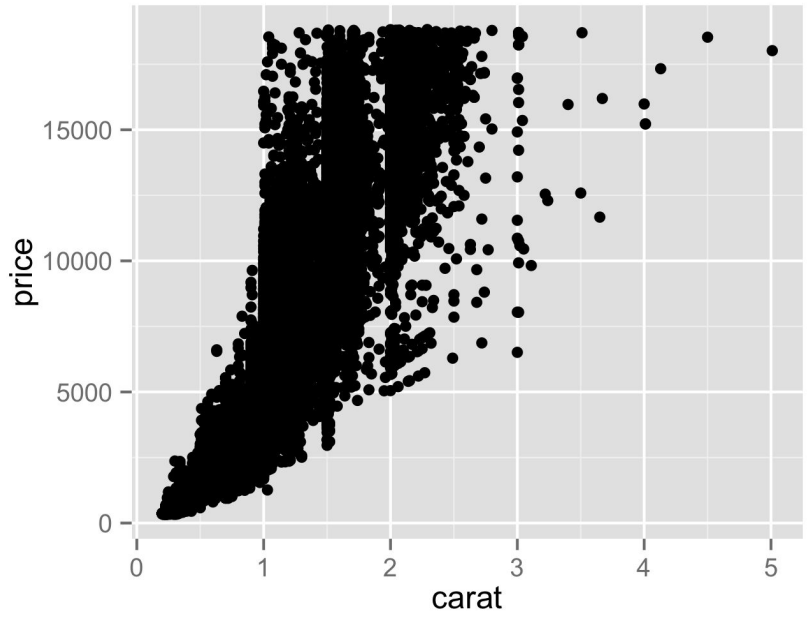


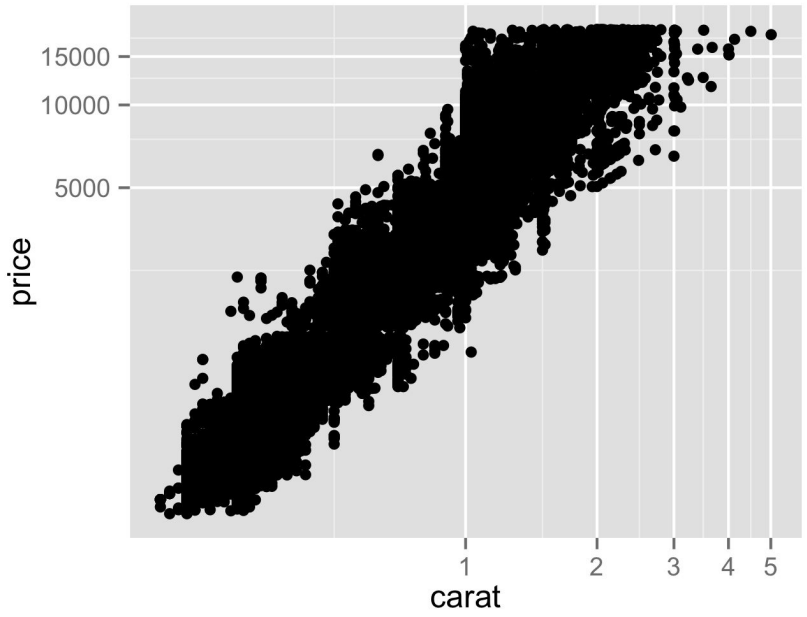
Coordinate Systems

trans

```
g
```

```
g + coord_trans(x = "log10", y = "log10")
```

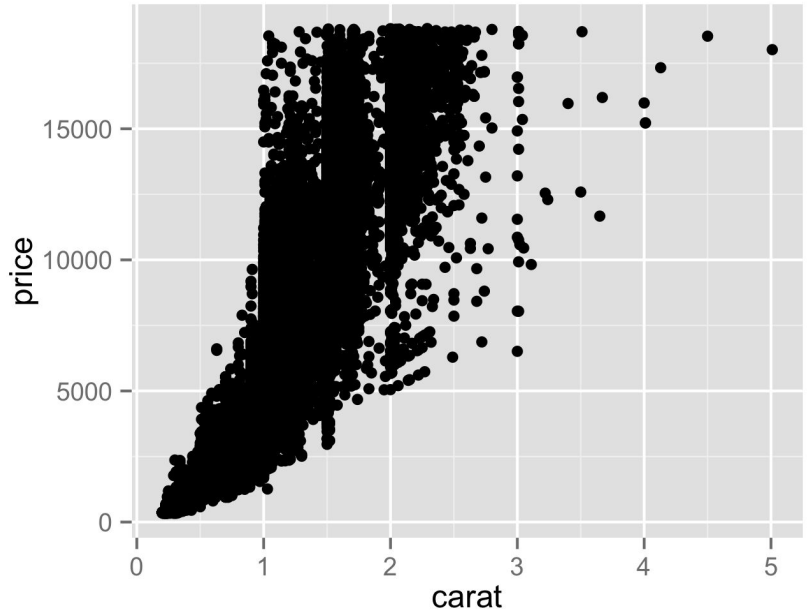




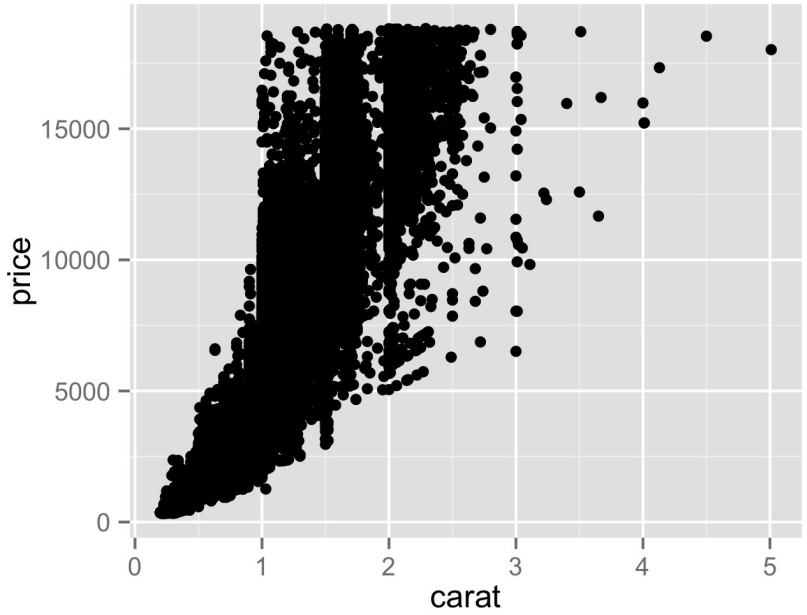
Coordinate Systems

cartesian (default)

```
g
```



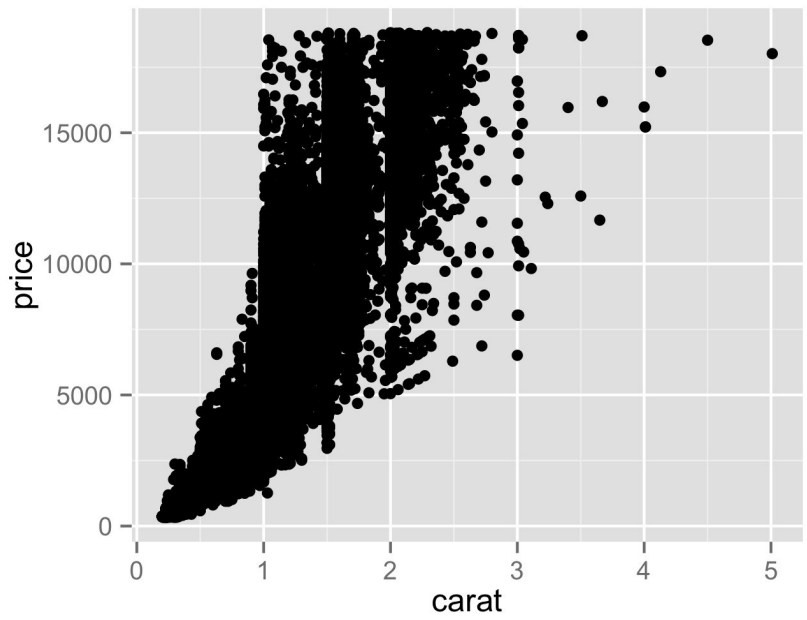

```
g + coord_cartesian()
```



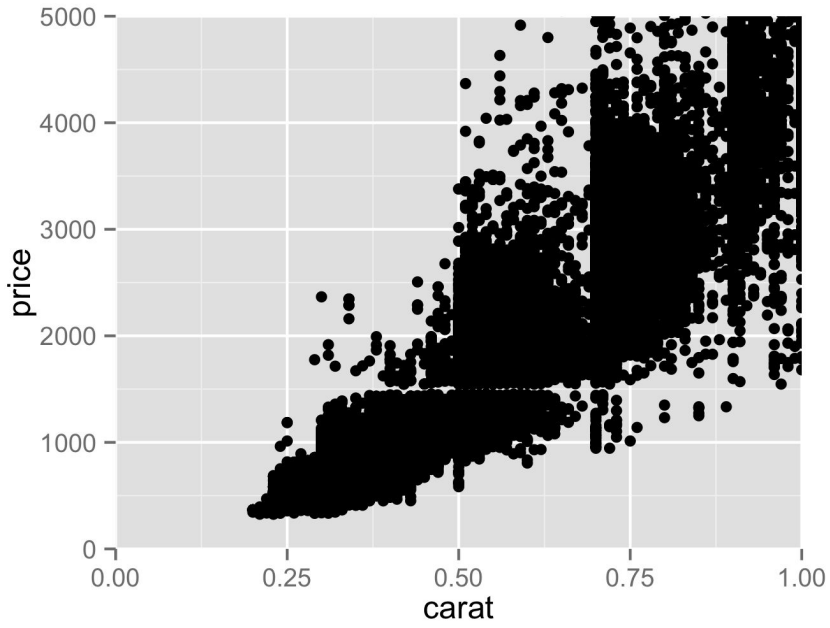
Coordinate Systems

cartesian (to zoom)

```
g
```

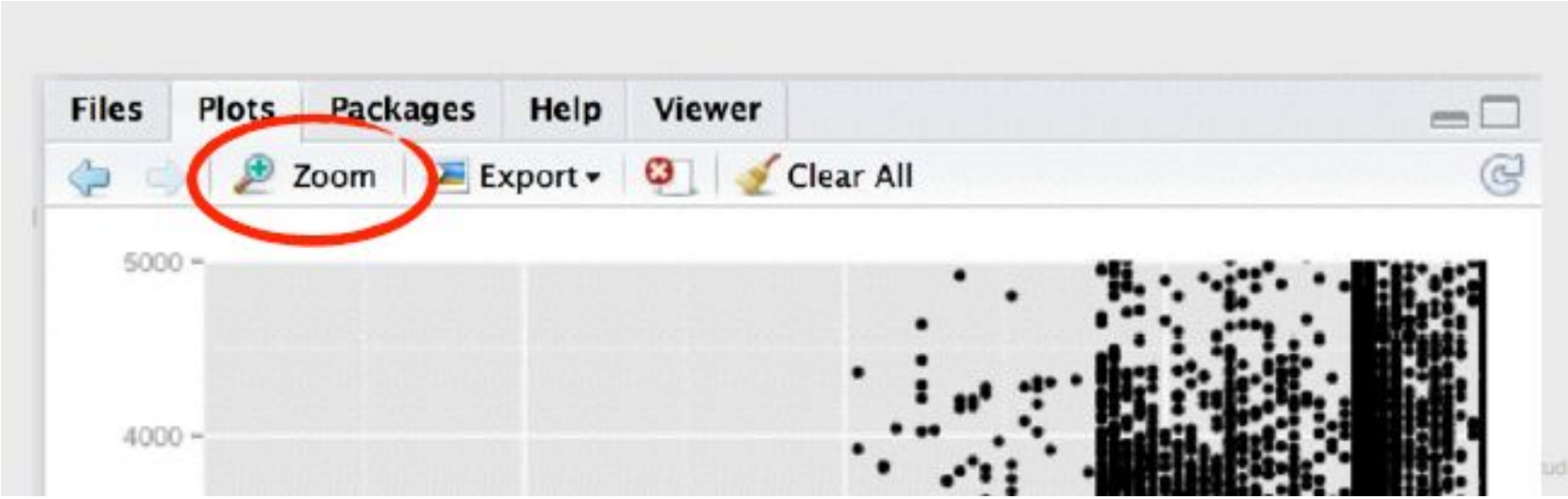



```
g + coord_cartesian(xlim = c(0, 1), ylim = c(0,5000))
```



Coordinate Systems

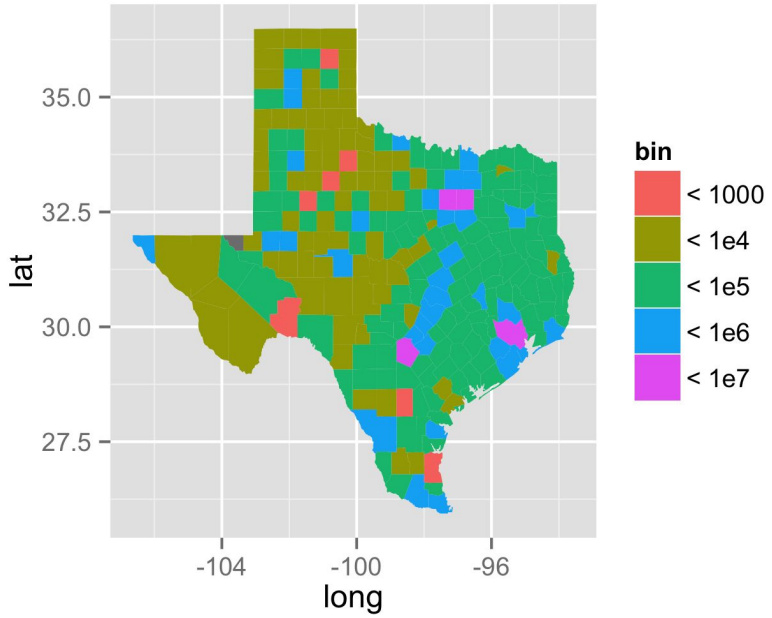
- There is also another coordinate system called coord_map().
- Add it to tx and then open the plot in a new window by hitting zoom in RStudio (see the diagram below).
- Try rescaling the window. Can you tell what coord_map() does?



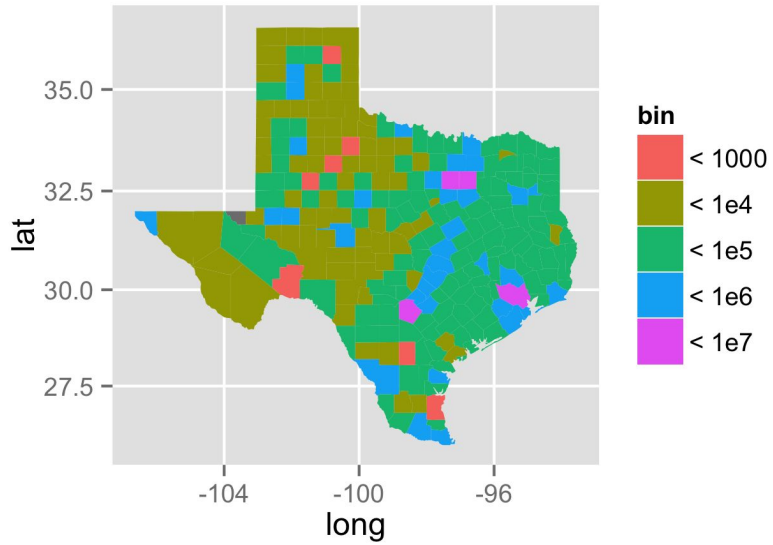
Coordinate Systems

A plot followed by coord_map() will display in a mercator projection, often used for typical maps.

```
tx
```




```
tx + coord_map()
```



Coordinate Systems

Pie charts

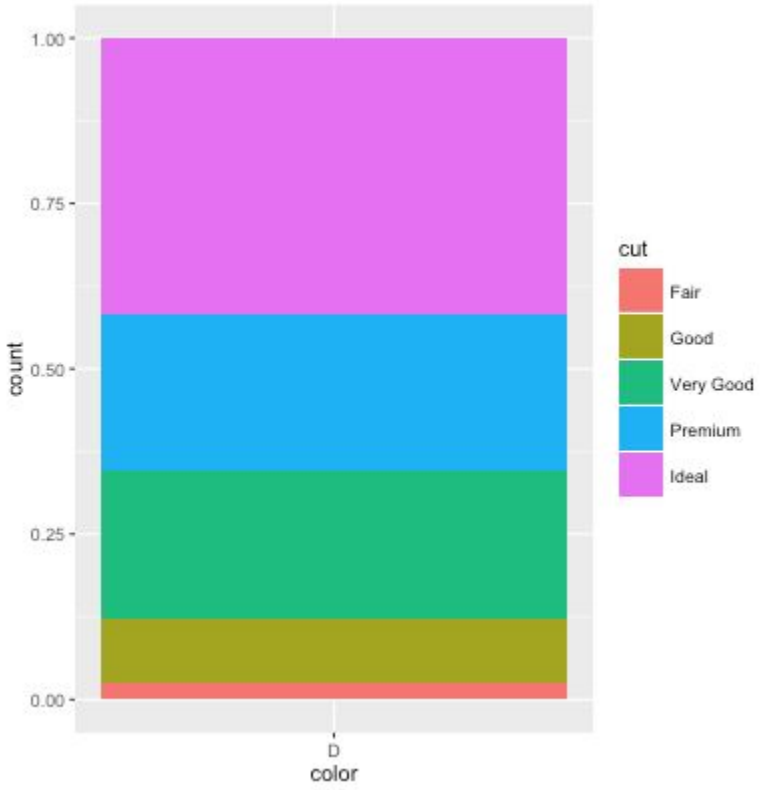
- Aside: In the grammar of graphics, there is no straightaway method used for creating a pie chart; however, we can manually create one by graphing a stacked bar graph in polar coordinates.

```
d2 <- subset(diamonds, color == "D")
cc <- ggplot(data = d2, aes(x = color)) +
    geom_bar(aes(fill = cut), position = "fill")
cc

cc + coord_polar(theta = "y")
```

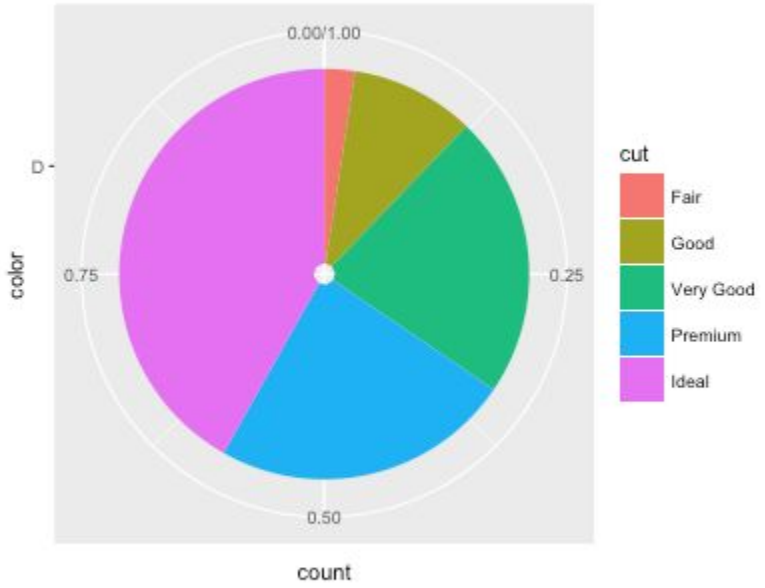
Coordinate Systems

Pie charts

```
cc
```




```
cc + coord_polar(theta = "y")
```



Outline

- Customizing Graphics
- Titles
- Coordinate systems
- **Scales**
- Themes
- Axis labels
- Legends
- Other visualizations

Scales

Scales

Scale determines how data are mapped to an aesthetic. The general pattern is as follows:

```
scale_aesthetic_name()
```

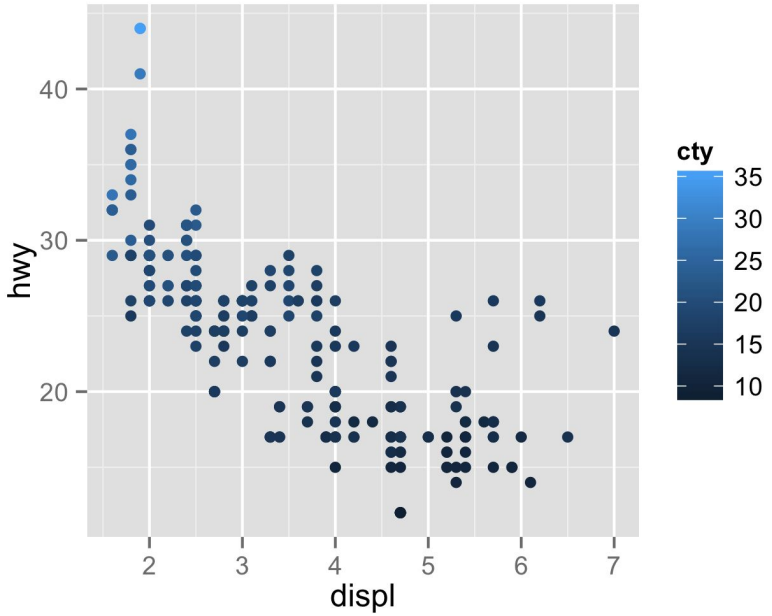
- scale_
  - Always begins with scale_...
- aesthetic_
  - ...the aesthetic to adjust...
- name
  - ...name of a scale object...
- ()
  - ...open and closed parentheses.

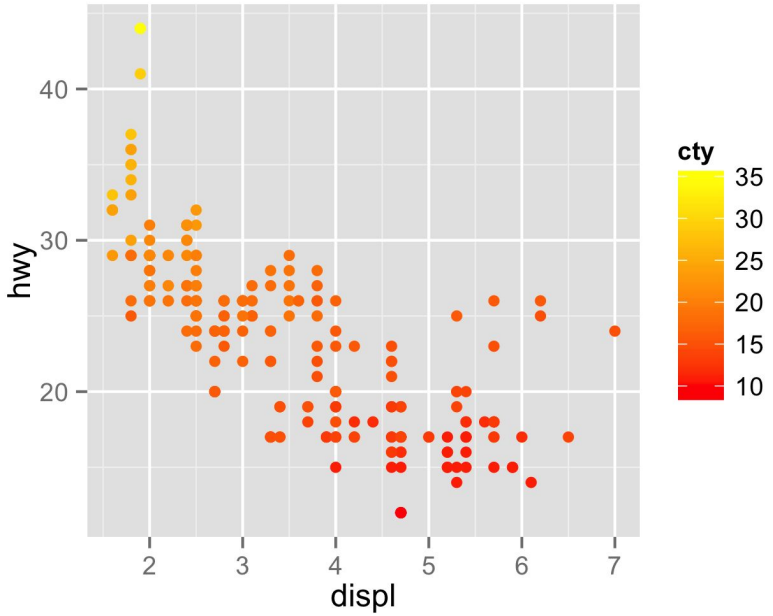
Scales

scale_color_gradient

```
ggplot(data = mpg, aes(x = displ, y = hwy)) + geom_point(aes(color = cty))
```

```
last_plot() + scale_color_gradient(low = "red", high = "yellow")
```



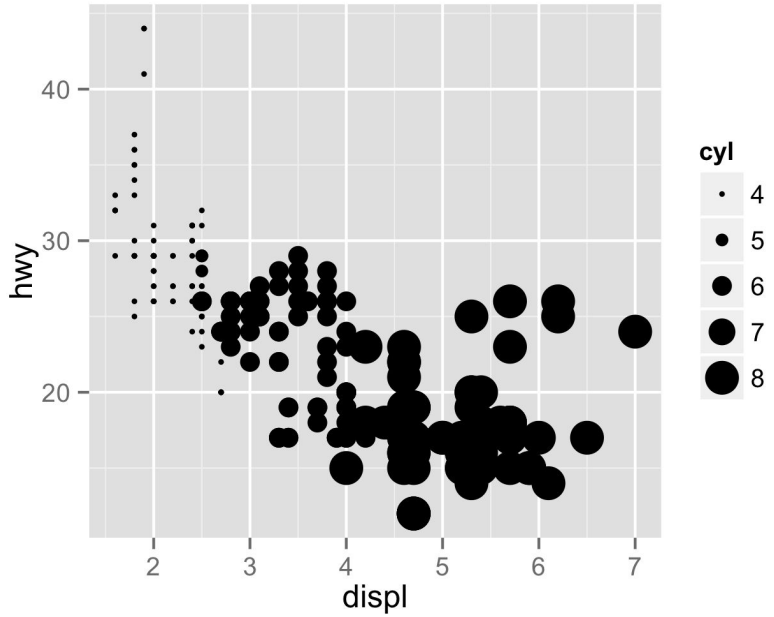


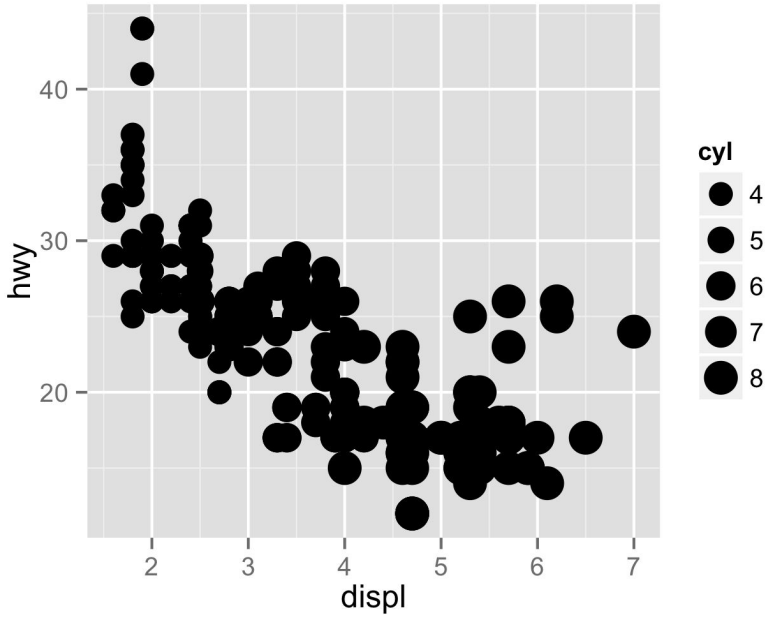
Scales

scale_size_area

```
ggplot(data = mpg, aes(x = displ, y = hwy)) + geom_point(aes(size = cyl))
```

```
last_plot() + scale_size_area()
#Area of points proportional to value.
```

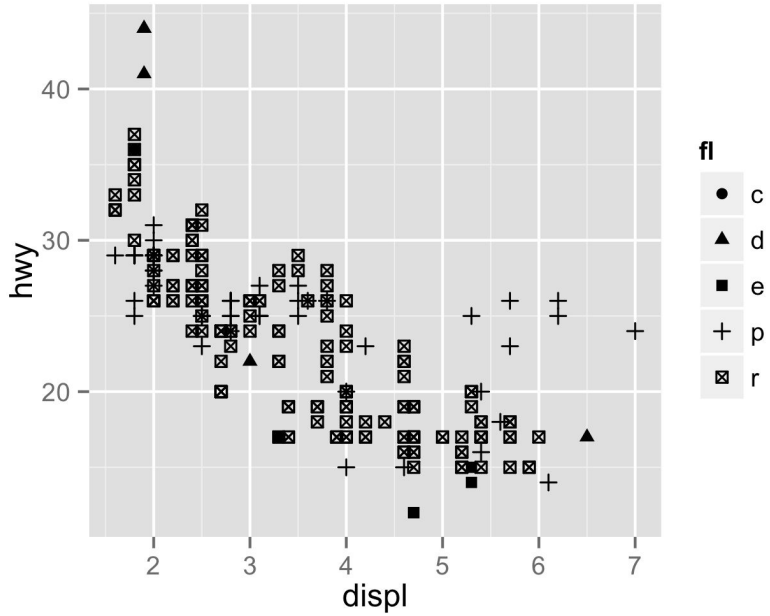




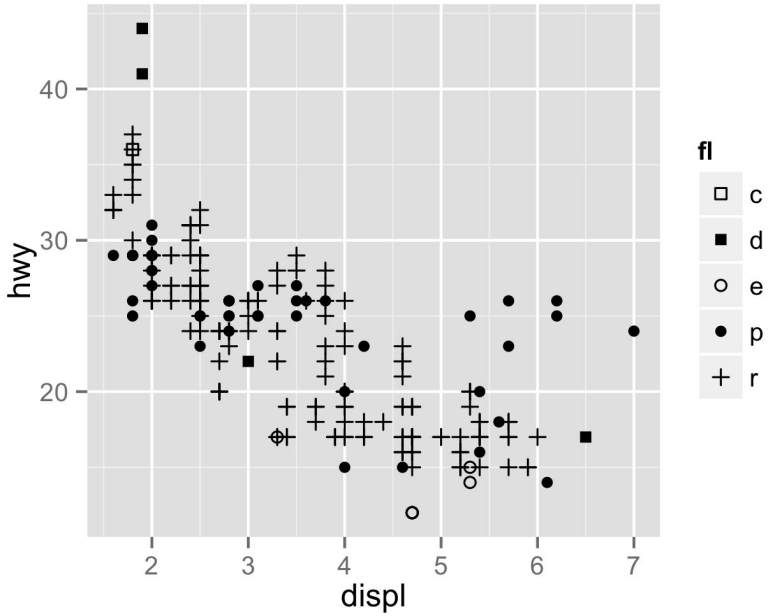
Scales

scale_shape_manual

```
ggplot(data = mpg, aes(x =
displ, y = hwy)) + geom_point
(aes(shape = fl))
```



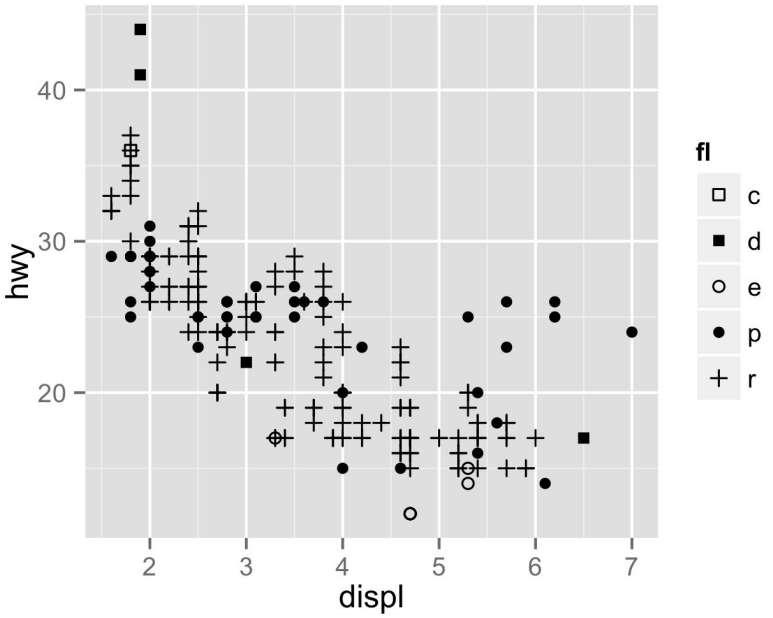

```
last_plot() +
scale_shape_manual(values = c
(0, 15, 1, 16, 3))
```

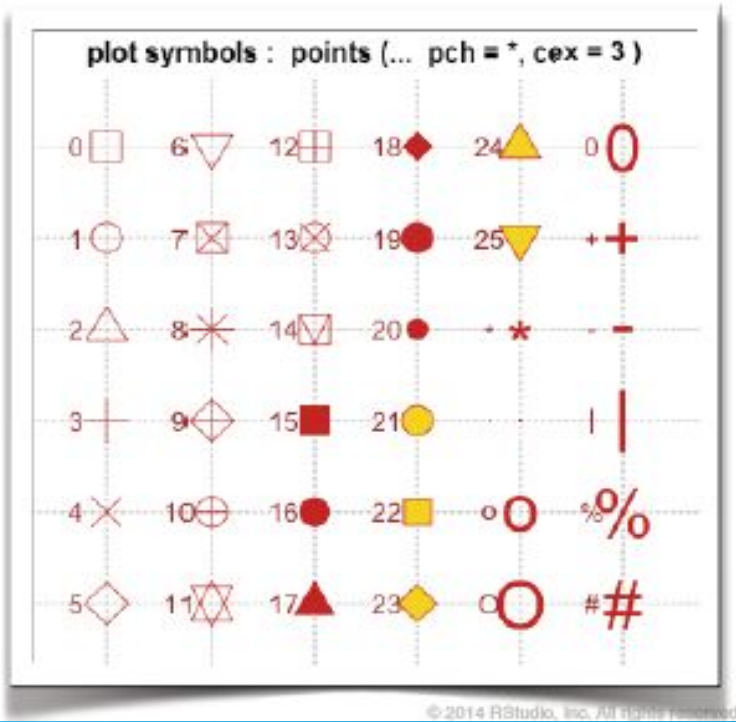


Scales

Aside: shapes.

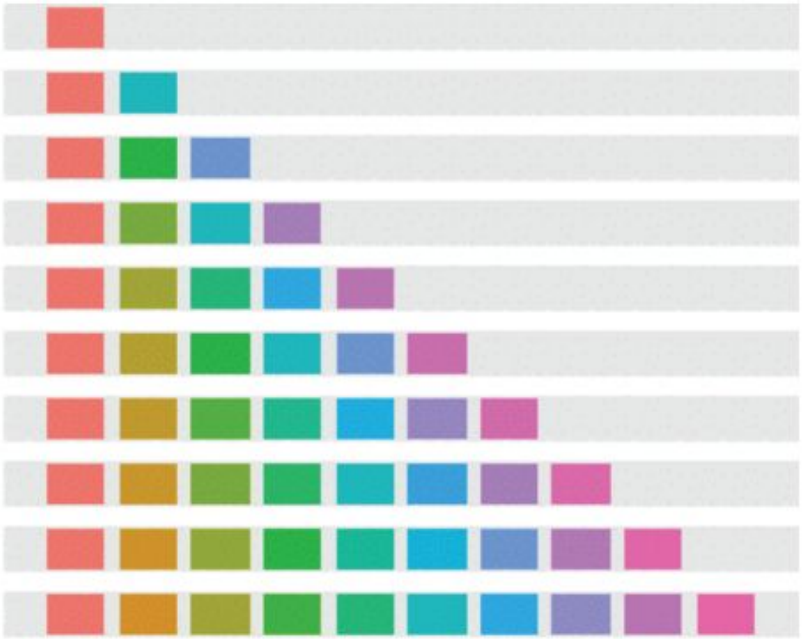
```
#Specify the shapes manually
last_plot() + scale_shape_manual(values = c(0, 15, 1, 16, 3))
```





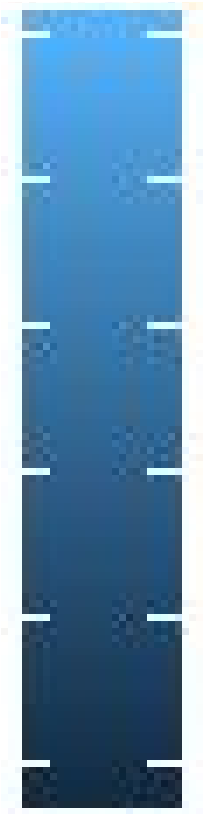
Color

- Color is the most popular aesthetic after position. While popular, it is also easiest to misuse.
- ggplot2 has some nice default discrete palettes which depend on how many colors are needed:



Default Continuous Palette

But what if you want something different?

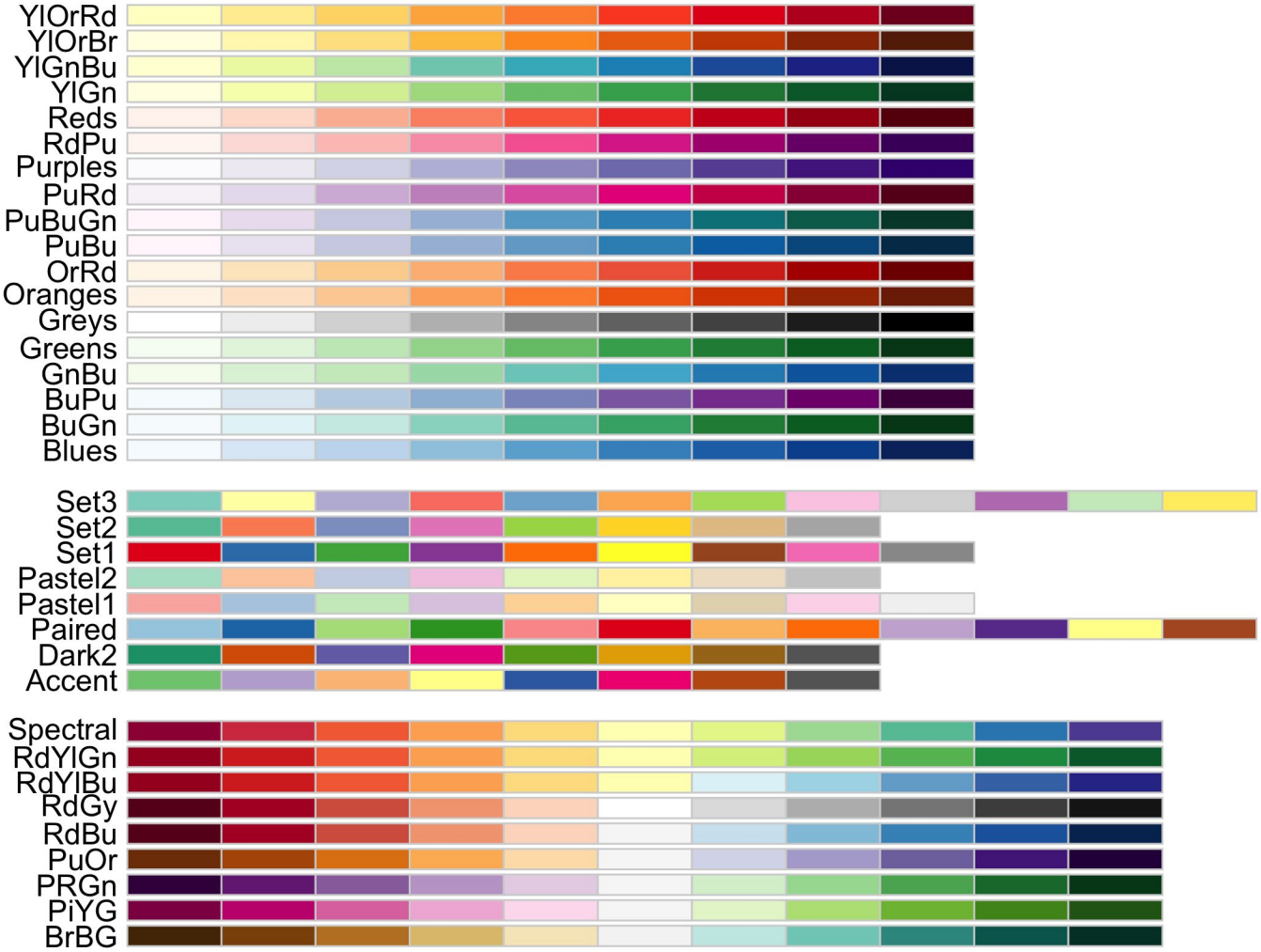


Color Brewer

- Cynthia Brewer developed useful, pleasing palettes, particularly tailored for maps:
- http://colorbrewer2.org
- To see a list of each brewer palette, run the following command in the RColorBrewer library.

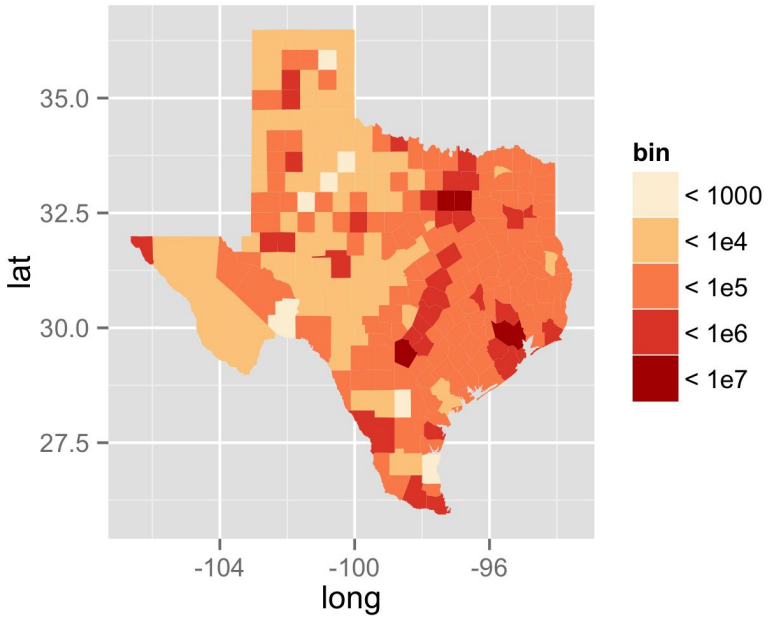
```
#install.packages("RColorBrewer")
library(RColorBrewer)
display.brewer.all()
```

Color Brewer

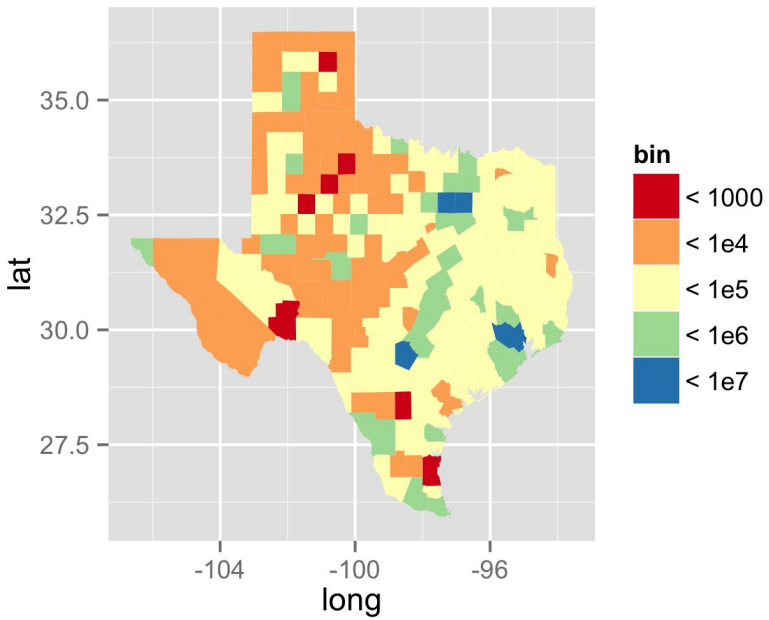


Brewer Scale

```
tx + scale_fill_brewer(
palette = "OrRd")
```

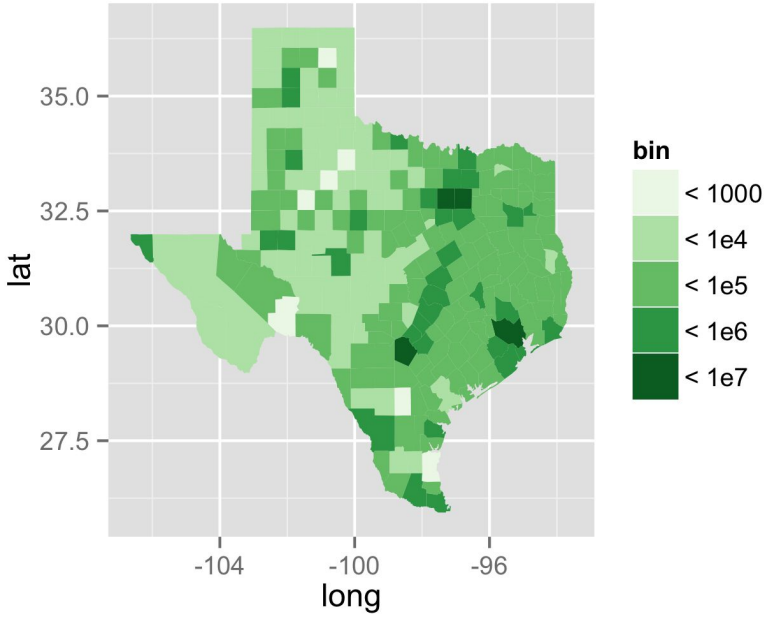



```
tx + scale_fill_brewer(
palette = "Spectral")
```

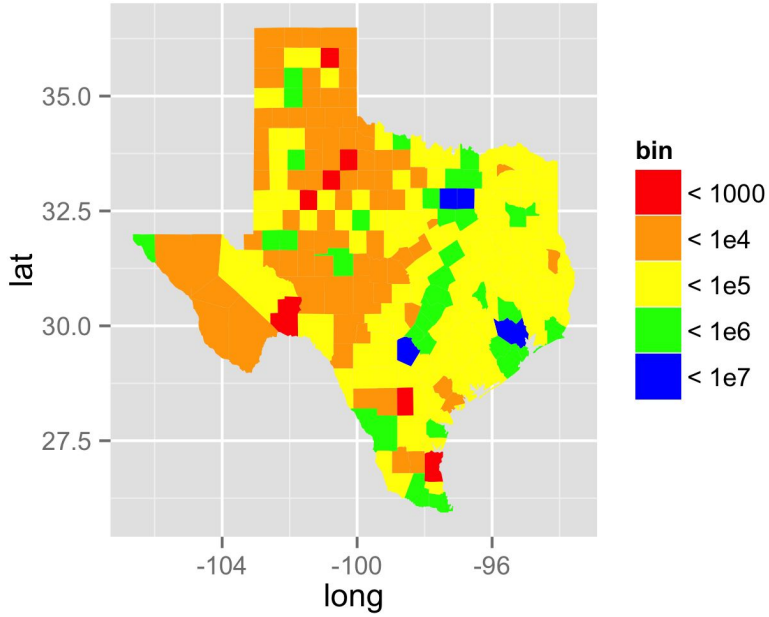


Custom != Pretty

```
tx +
scale_fill_brewer(palette =
"Greens")
```




```
tx +
scale_fill_manual(values = c
("red", "orange", "yellow",
"green", "blue"))
```



Scales

There are many other scales available in ggplot2. To see complete list with examples, visit: http://docs.ggplot2.org/current

Scales

Scales control the mapping between data and aesthetics.

- `expand_limits`
  Expand the plot limits with data.
- `guides`
  Set guides for each scale.
- `guide_legend`
  Legend guide.
- `guide_colourbar` (guide_colorbar)
  Contiuous colour bar guide.
- `scale_alpha` (scale_alpha_continuous, scale_alpha_discrete)
  Alpha scales.
- `scale_area`
  Scale area instead of radius (for size).
- `scale_colour_brewer` (scale_color_brewer, scale_fill_brewer)
  Sequential, diverging and qualitative colour scales from colorbrewer.org
- `scale_colour_gradient` (scale_color_continuous, scale_color_gradient, scale_colour_continuous, scale_fill_continuous, scale_fill_gradient)
  Smooth gradient between two colours
- `scale_colour_gradient2` (scale_color_gradient2, scale_fill_gradient2)
  Diverging colour gradient
- `scale_colour_gradientn` (scale_color_gradientn, scale_fill_gradientn)
  Smooth colour gradient between n colours
- `scale_colour_grey` (scale_color_grey, scale_fill_grey)
  Sequential grey colour scale.






Outline

- Customizing Graphics
- Titles
- Coordinate systems
- Scales
- **Themes**
- Axis labels
- Legends
- Other visualizations

Themes

Themes

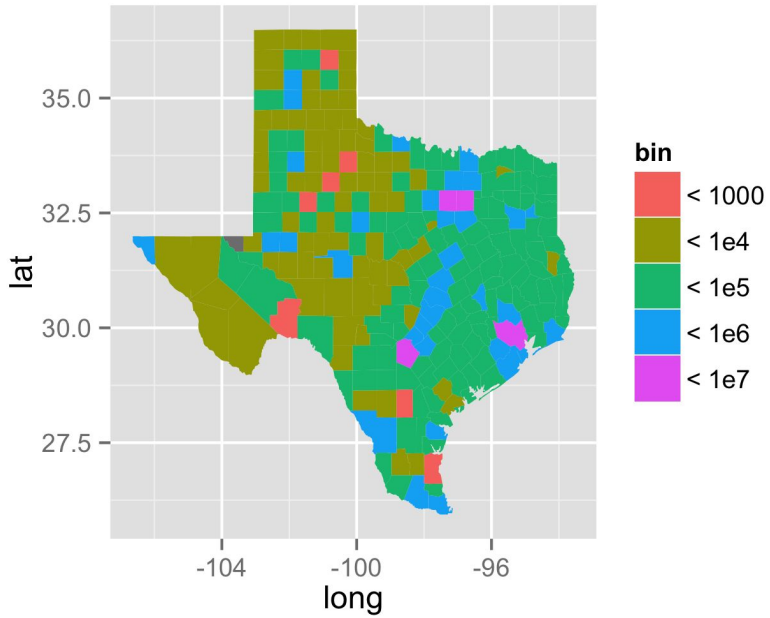
Visual appearance

- The theme controls the appearance of the non-geom parts of the plot.
- In other words, the theme also helps determine how the general plot looks.

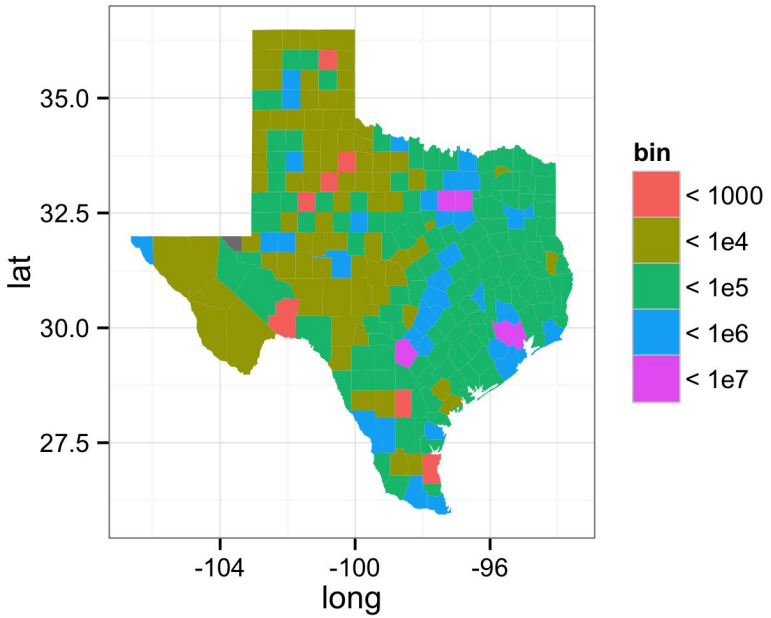
Themes

ggplot2 comes with two pre-loaded themes that control the appearance of non-data elements:

```
tx + theme_grey() #Default
```



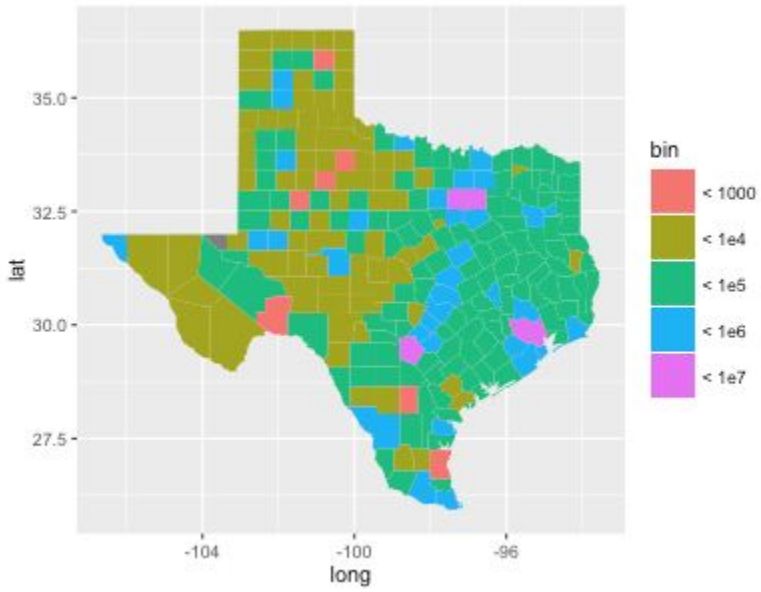

```
tx + theme_bw()
```



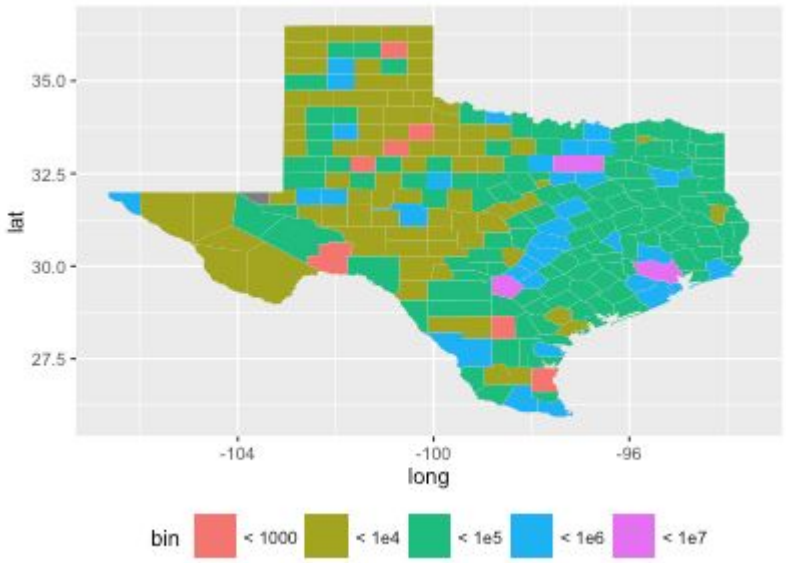
Themes

You can modify individual details of the current theme with theme() or even create your own theme:

```
tx
```




```
tx + theme(legend.position =
"bottom")
```



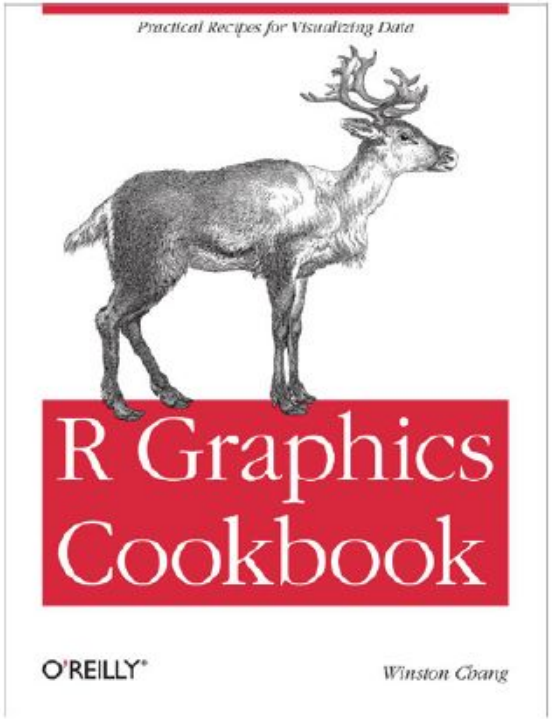
Themes

Elements

- Axis:
  - axis.line, axis.text.x, axis.text.y, axis.ticks,axis.title.x, axis.title.y
- Legend:
  - legend.background, legend.key, legend.text, legend.title
- Panel:
  - panel.background, panel.border, panel.grid.major, panel.grid.minor
- Strip:
  - strip.background, strip.text.x, strip.text.y
- see ?theme
- https://github.com/wch/ggplot2/wiki/New-theme-system

Themes

- To learn how to manually change individual elements of a theme, one of the best resources is the R Graphics Cookbook by Winston Chang
- http://shop.oreilly.com/product/0636920023135.do#



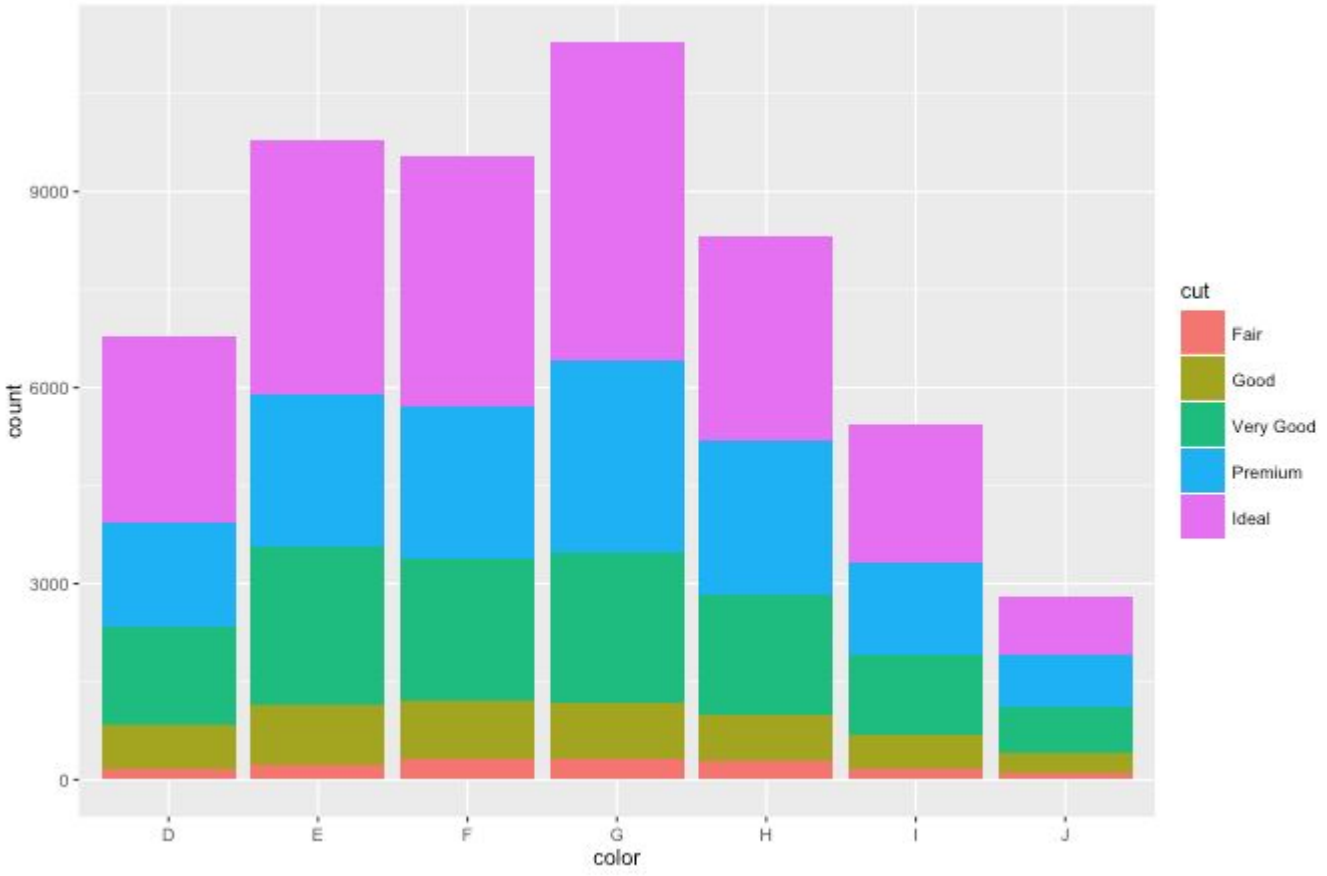
Themes

https://github.com/jrnold/ggthemes

```
# install.packages("ggthemes")
library(ggthemes)
p <- ggplot(data = diamonds, aes(x = color)) + geom_bar(aes(fill
= cut))
```

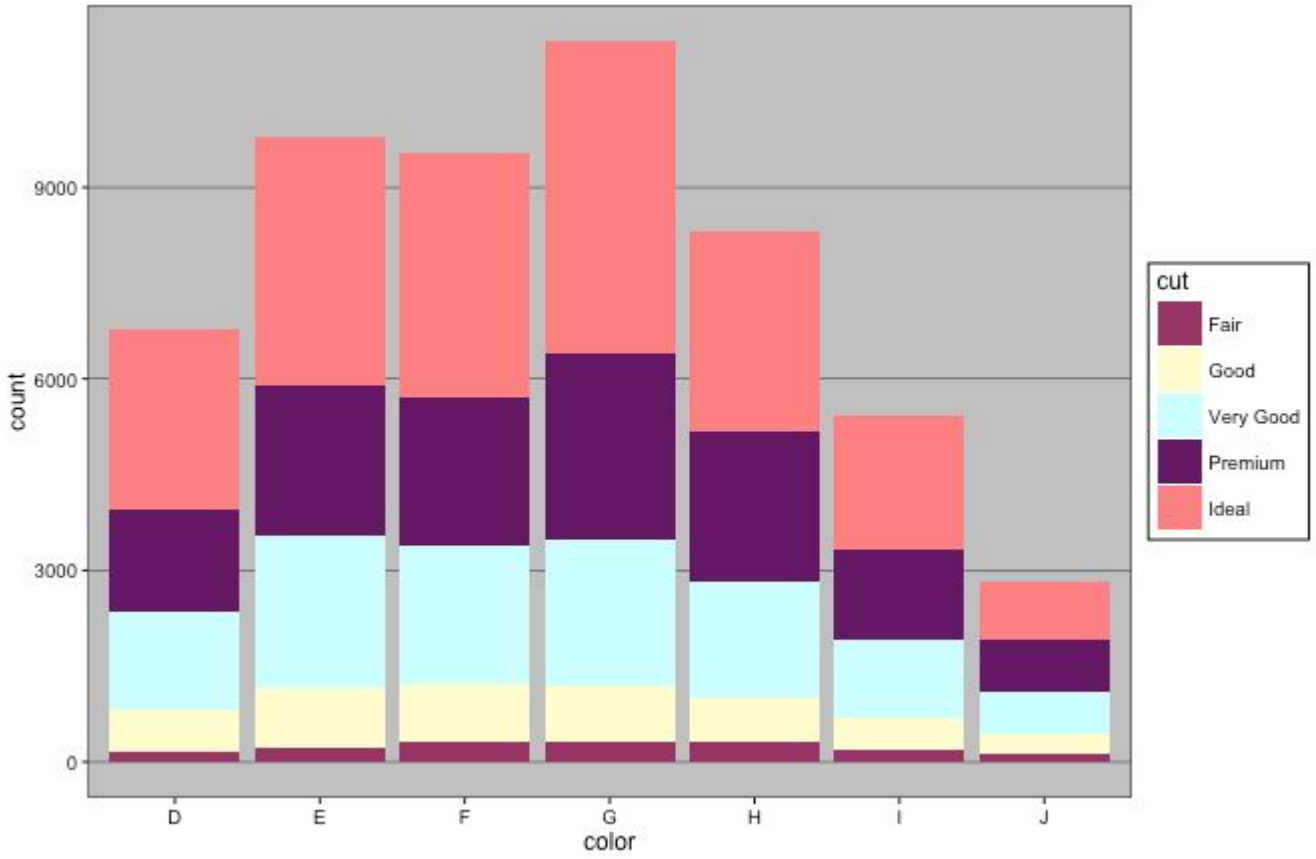
Default Theme

```
p
```



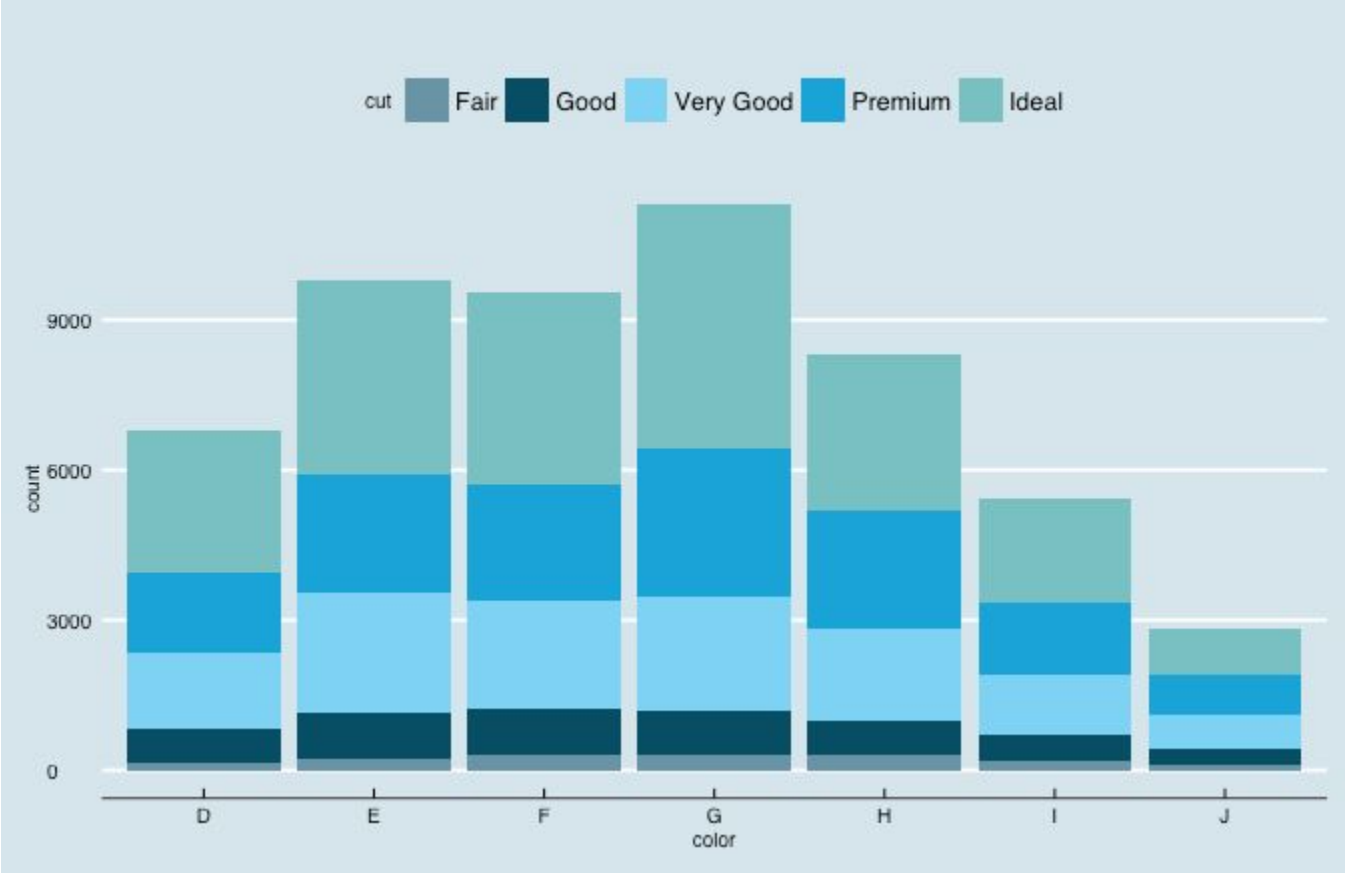
theme_excel()

```
p + theme_excel() + scale_fill_excel()
```



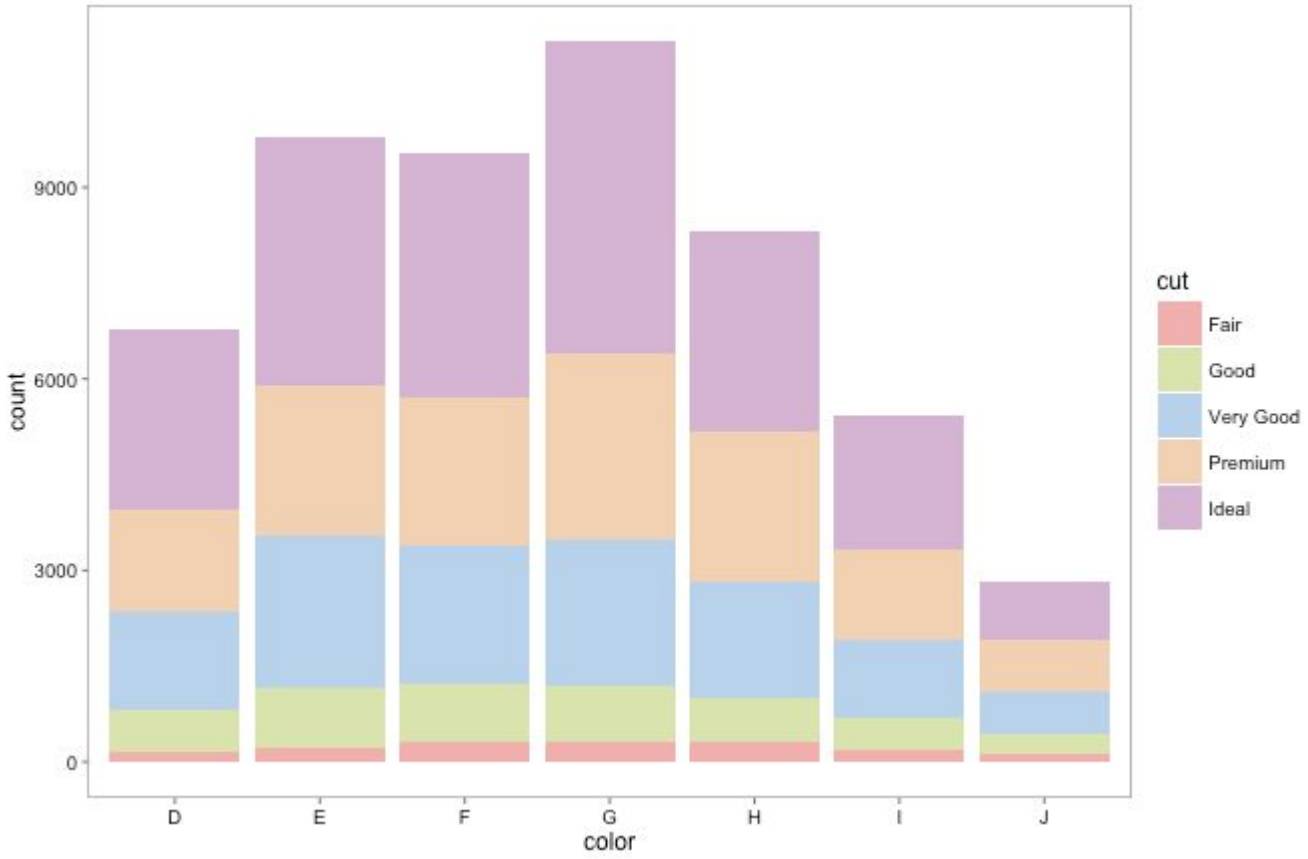
theme_economist()

```
p + theme_economist() + scale_fill_economist()
```



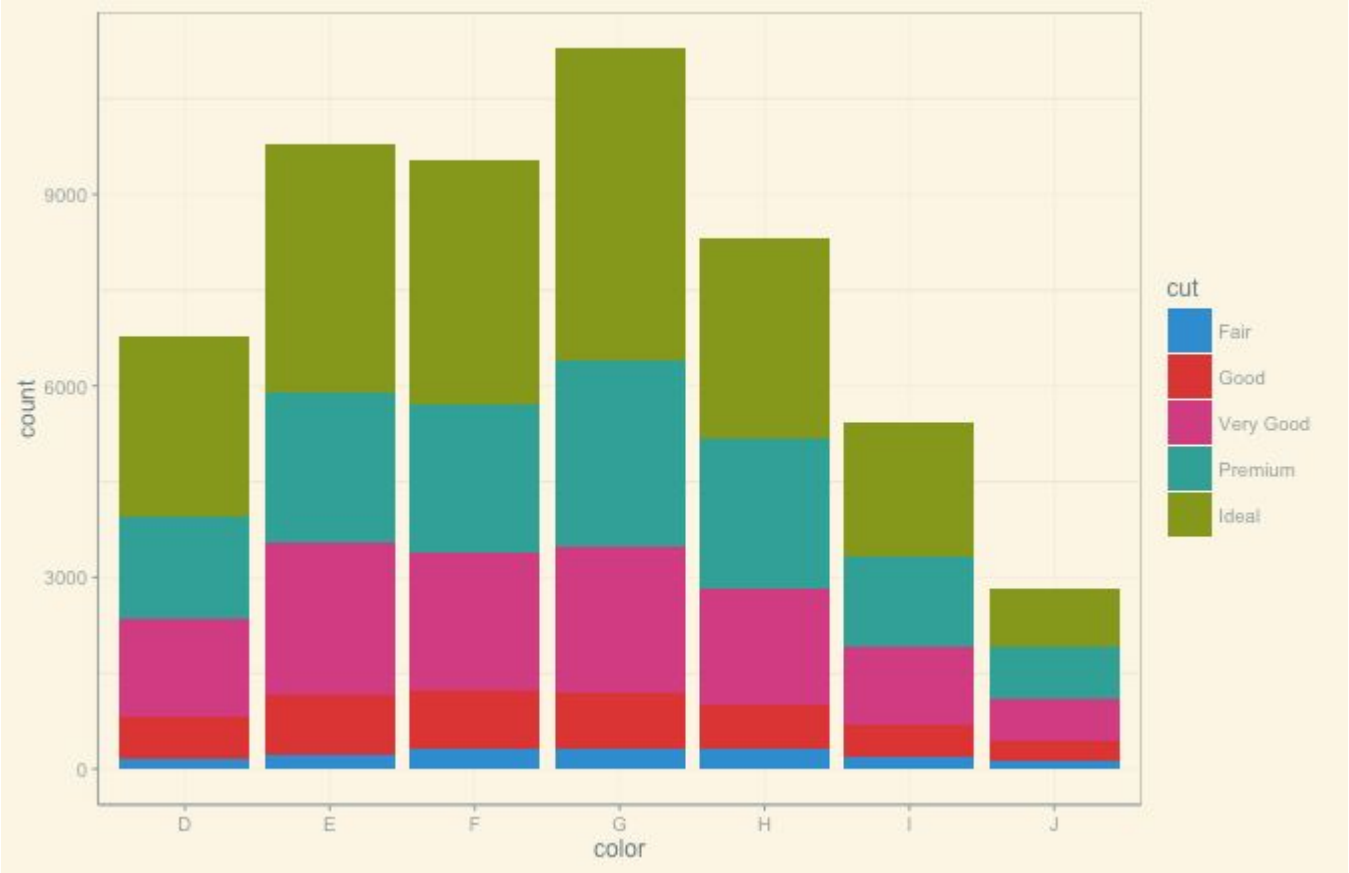
theme_few()

```
p + theme_few() + scale_fill_few()
```



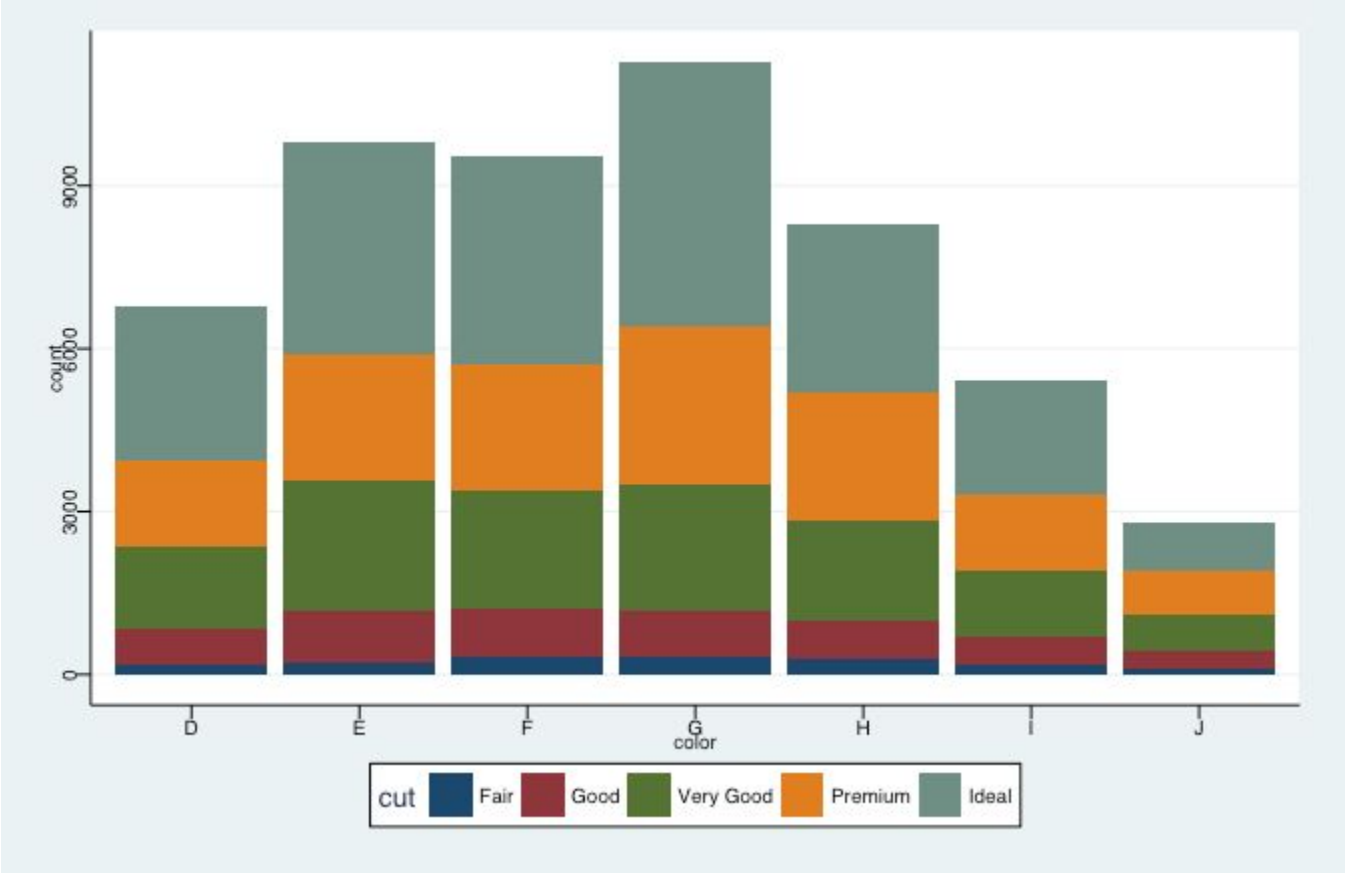
theme_solarized()

```
p + theme_solarized() + scale_fill_solarized()
```



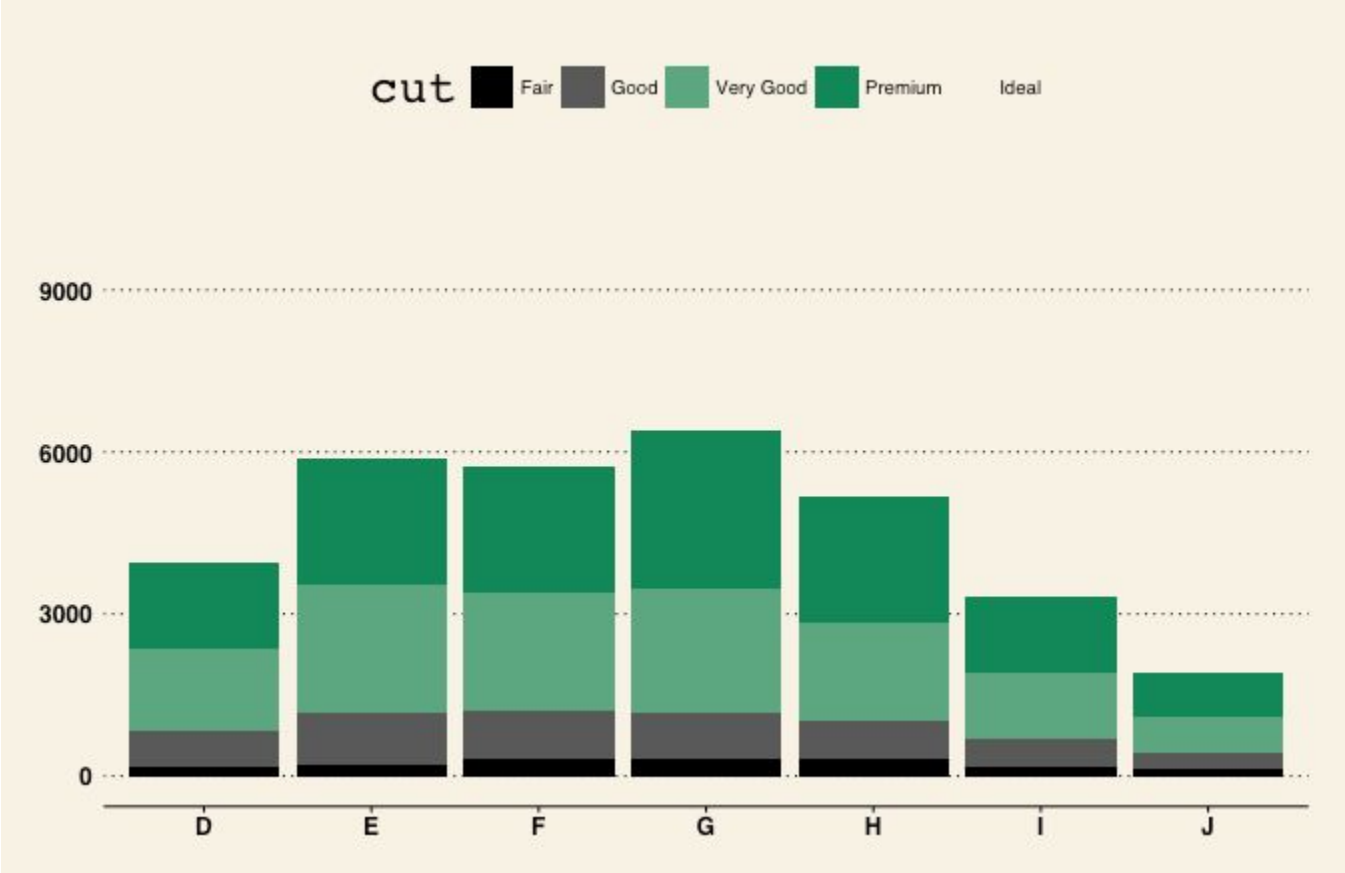
theme_stata()

```
p + theme_stata() + scale_fill_stata()
```



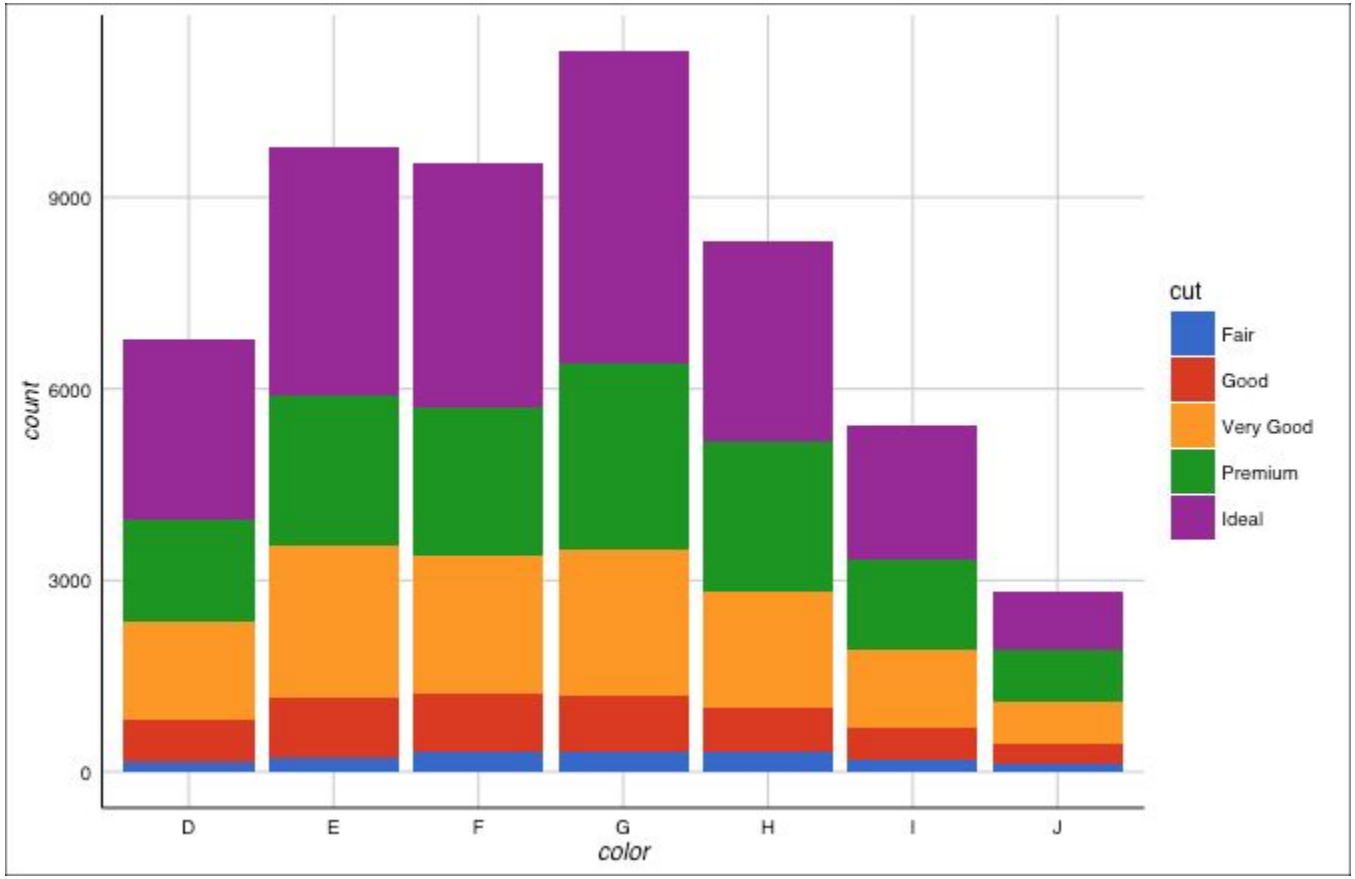
theme_wsj()

```
p + theme_wsj() + scale_fill_wsj(palette = "black_green")
```



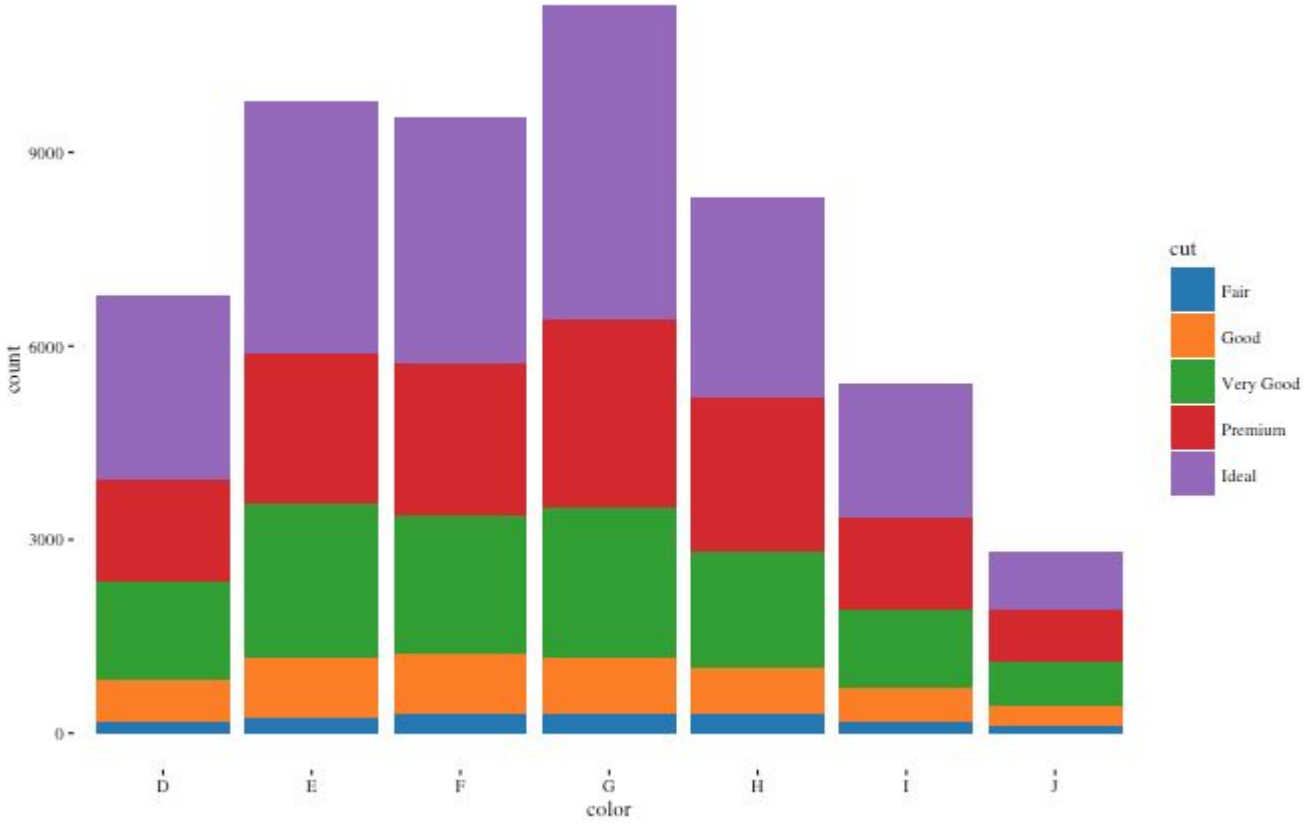
theme_gdocs()

```
p + theme_gdocs() + scale_fill_gdocs()
```



theme_tufte()

```
p + theme_tufte() + scale_fill_tableau()
```



Outline

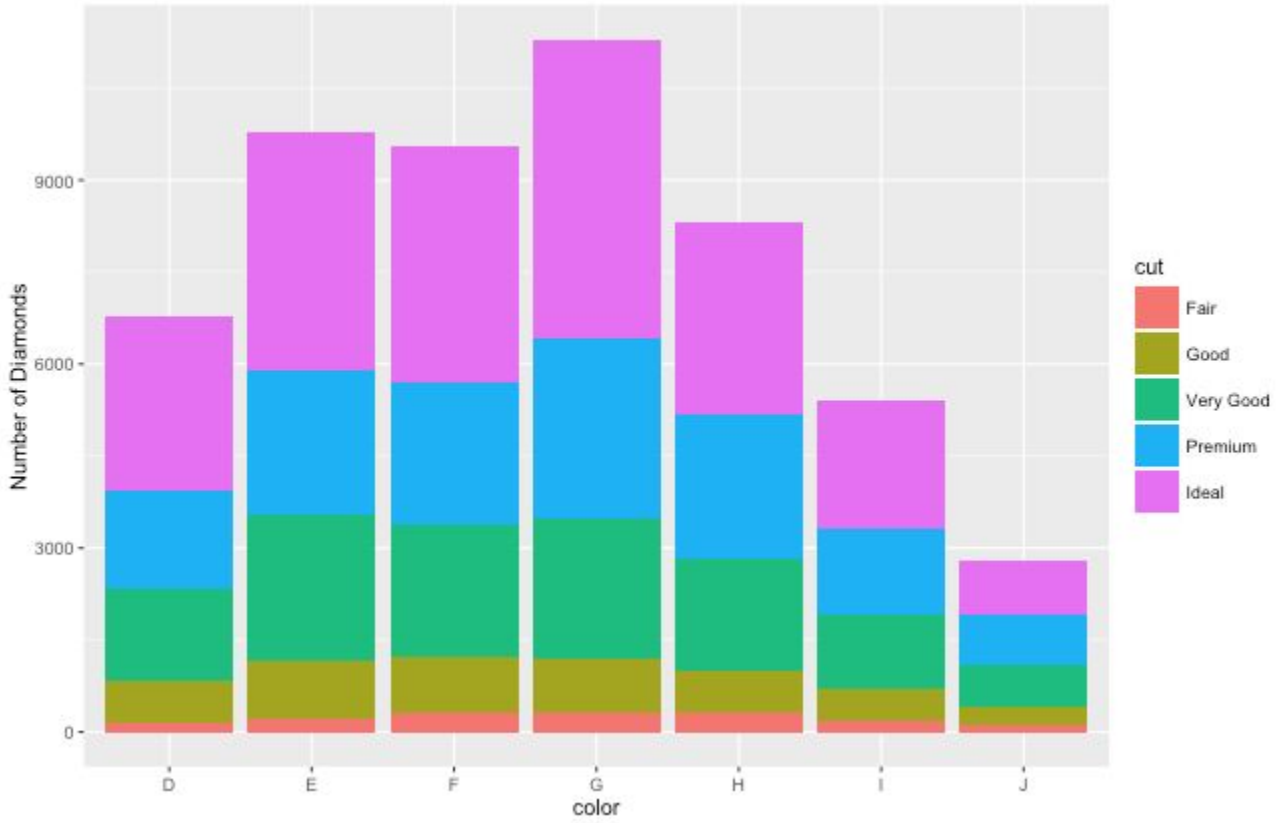
- Customizing Graphics
- Titles
- Coordinate systems
- Scales
- Themes
- **Axis labels**
- Legends
- Other visualizations

Axis Labels

Axis Labels

We can modify axis labels by using the functions xlab() and ylab():

```
p + ylab("Number of Diamonds")
```



Your Turn

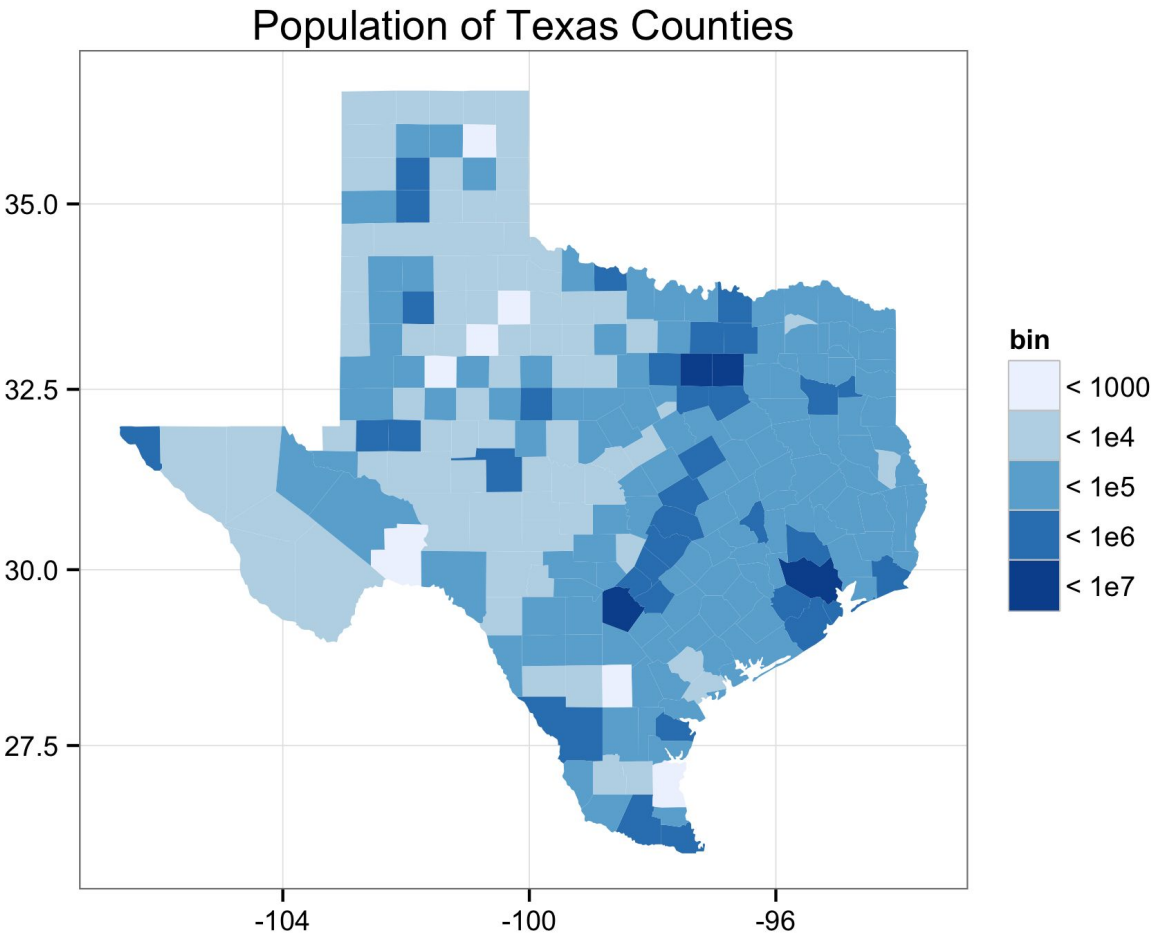
- Practice what you have learned so far on the base graph tx; try doing the following:
  - Remove the long and lat axis labels.
  - Add a white background.
  - Add a title.
  - Add a color brewer scale.
  - Change to an appropriate coordinate system.

Answer

```
tx +
    scale_fill_brewer(palette = "Blues") + #Adding colors.
    xlab("") +           #Removing the x-axis label.
    ylab("") +           #Removing the y-axis label.
    theme_bw() +         #Adding the white background.
    coord_map() +        #Changing the coordinate system.
    ggtitle("Population of Texas Counties") #Adding a title.
```

- This ultimately ends up producing the following graph...

Answer


Population of Texas Counties
35.0
32.5
30.0
27.5
-104
-100
-96
bin
< 1000
< 1e4
< 1e5
< 1e6
< 1e7

Outline

- Customizing Graphics
- Titles
- Coordinate systems
- Scales
- Themes
- Axis labels
- **Legends**
- Other visualizations

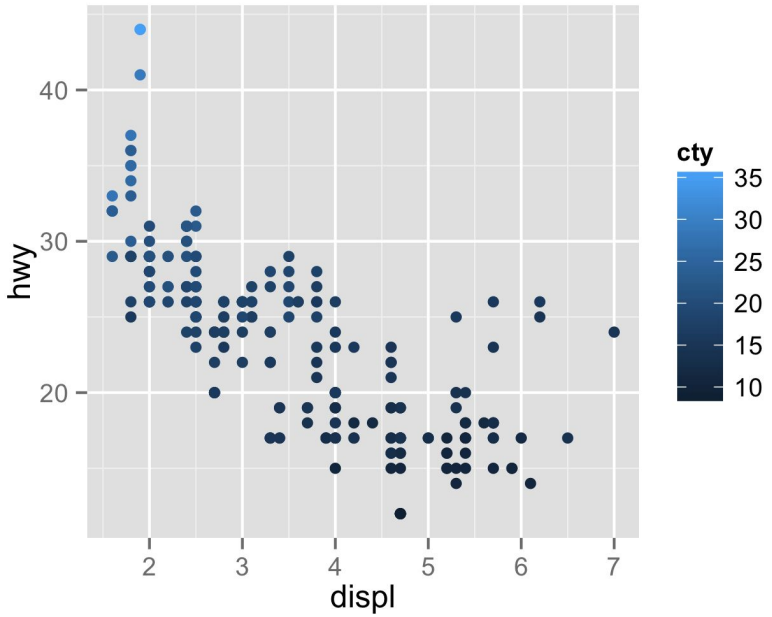
Legends

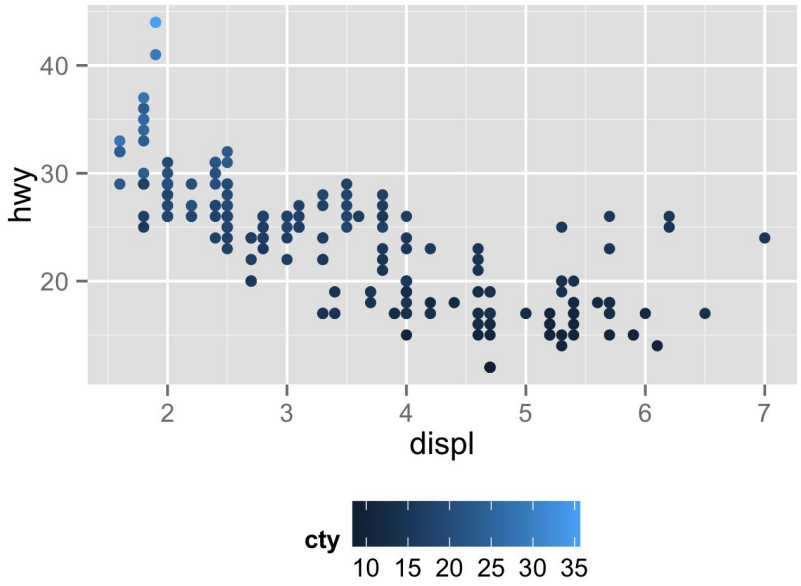
Legends

Change position with theme.

```
q <- ggplot(data = mpg, aes(x = displ, y = hwy)) + geom_point(aes(color = cty))
q
```

```
q + theme(legend.position = "bottom")
```





Legends

```
q + theme(legend.position = "bottom")
```

- legend.position
  - The aesthetic.
- "bottom"
  - One of “bottom”, “top”, or “left”, or “right”.

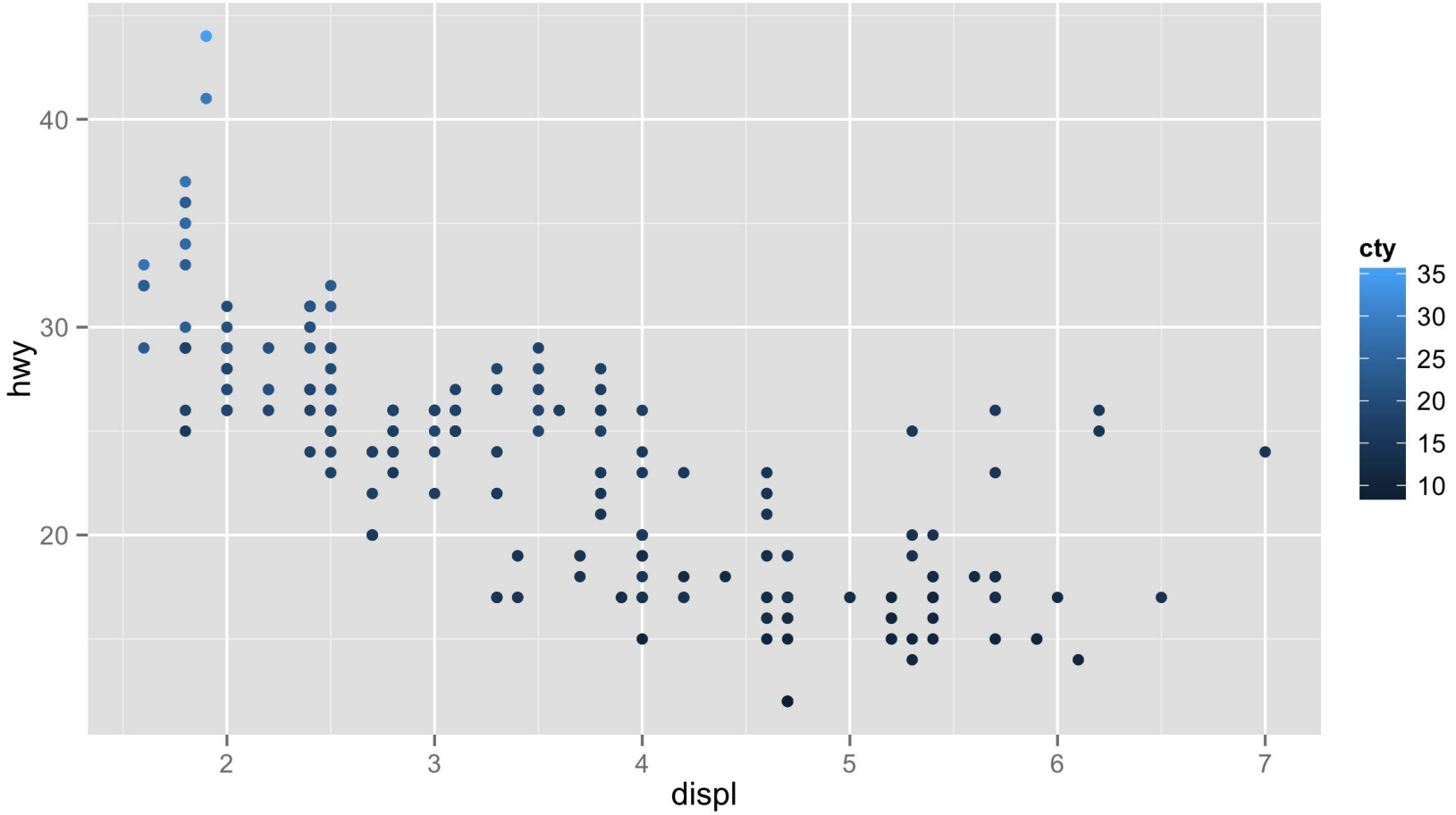
Legends

Choose the type of legend with guide.

```
#Continuous.
q + guides(color = "colorbar")

#Discrete.
q + guides(color = "legend")

#Neither.
q + guides(color = "none")
```

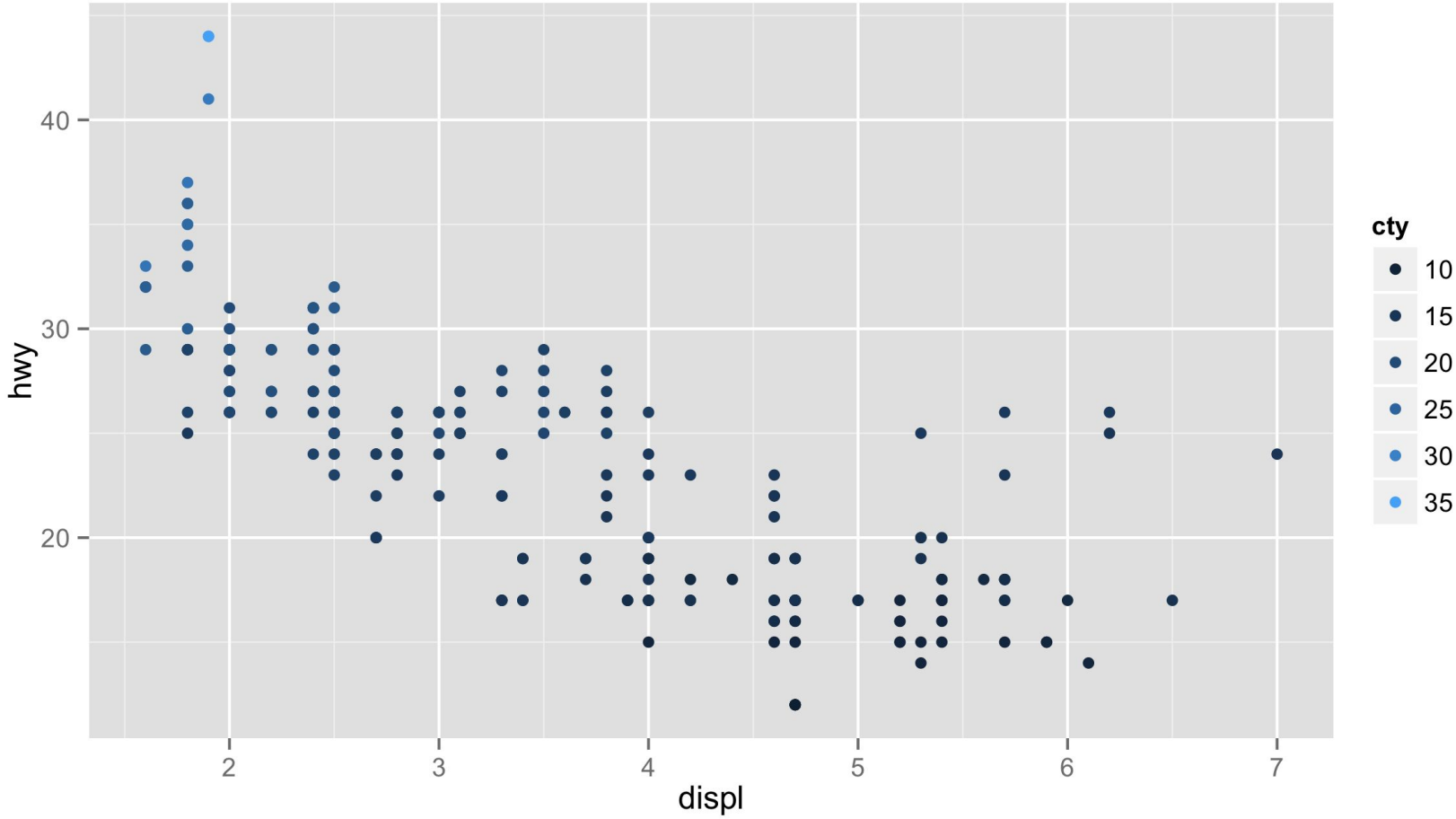
Legends

guide: "colorbar"



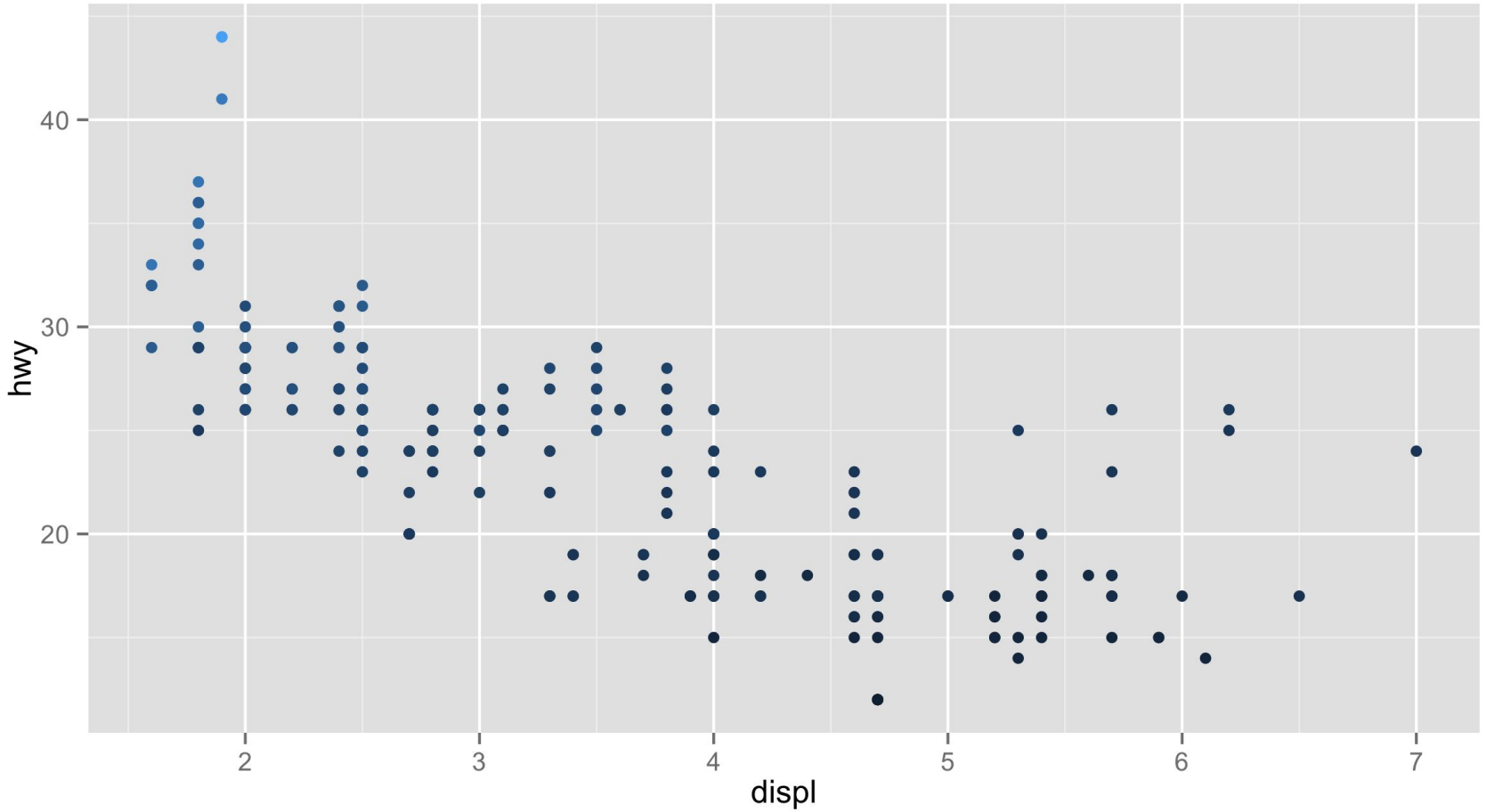
Legends

guide: "legend"



Legends

guide: "none"



Legends

```
q + guides(color = "legend")
```

- color
  - The aesthetic.
- "legend"
  - One of "legend", "none", or "colorbar" (for continuous values only).

Labels

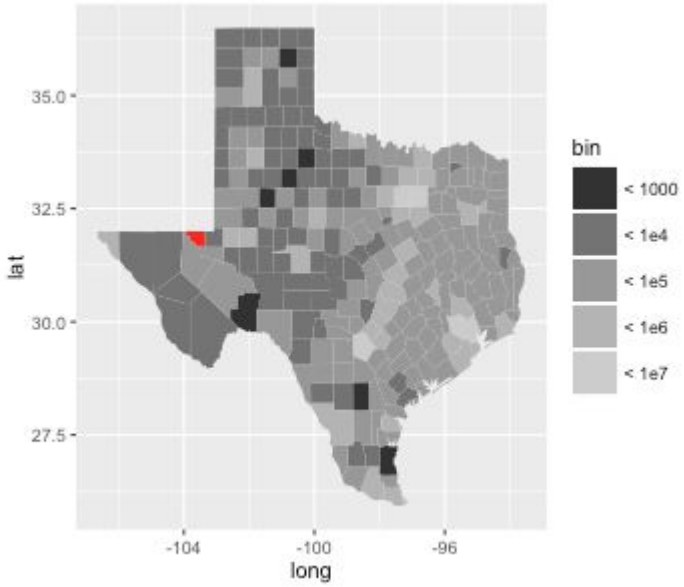
Every scale uses the following arguments

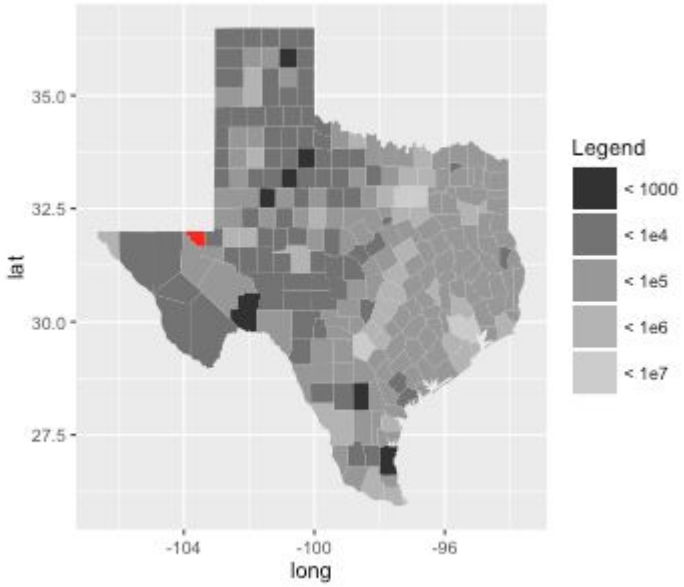
argument	controls
name	Title of legend (or axis label)
labels	Labels inside legend (or tick labels on axis) * must be a vector with one element for each label or tick mark

Labels

```
tx + scale_fill_grey()
```

```
tx + scale_fill_grey(name =
"Legend")
```

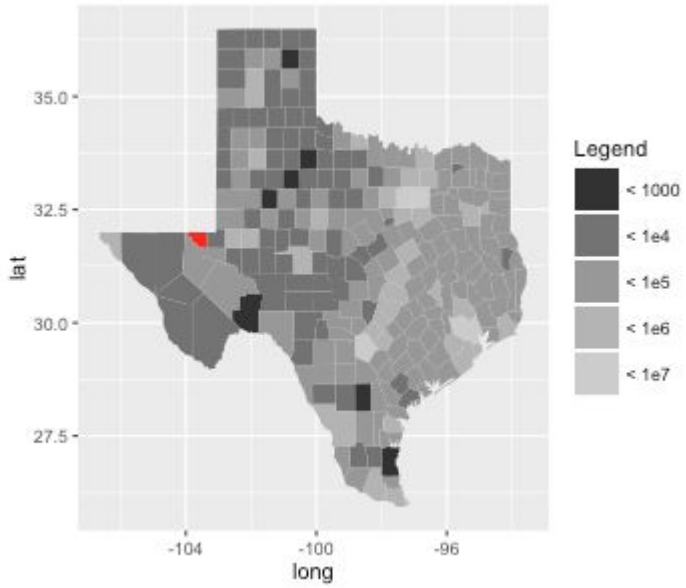


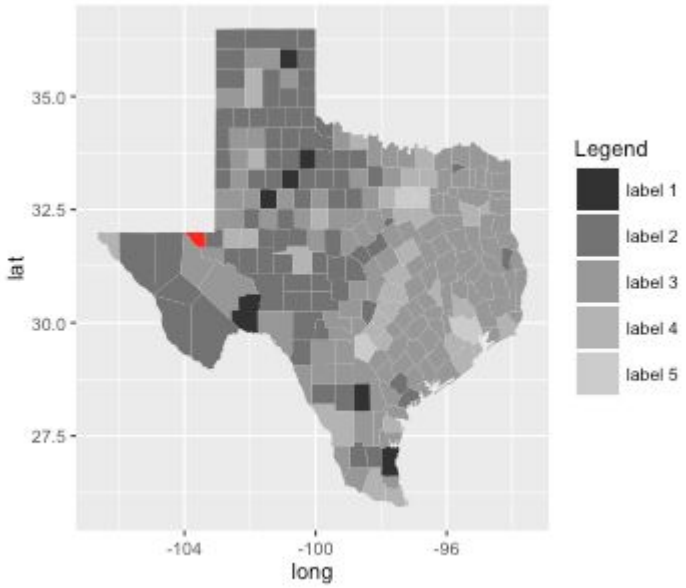


Labels

```
tx + scale_fill_grey(name =
"Legend")
```

```
tx + scale_fill_grey(name =
"Legend", labels = c("label 1",
"label 2", "label 3","label 4",
"label 5"))
```





Labels

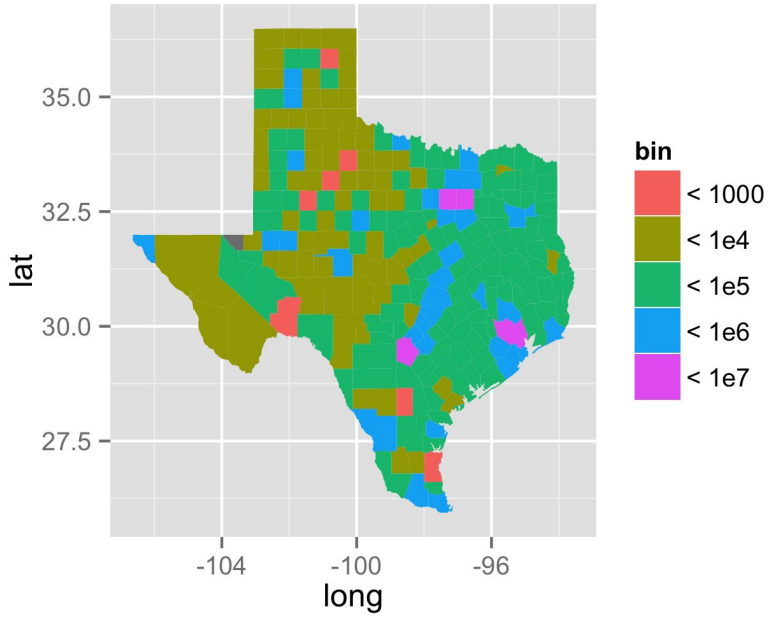
Defaults

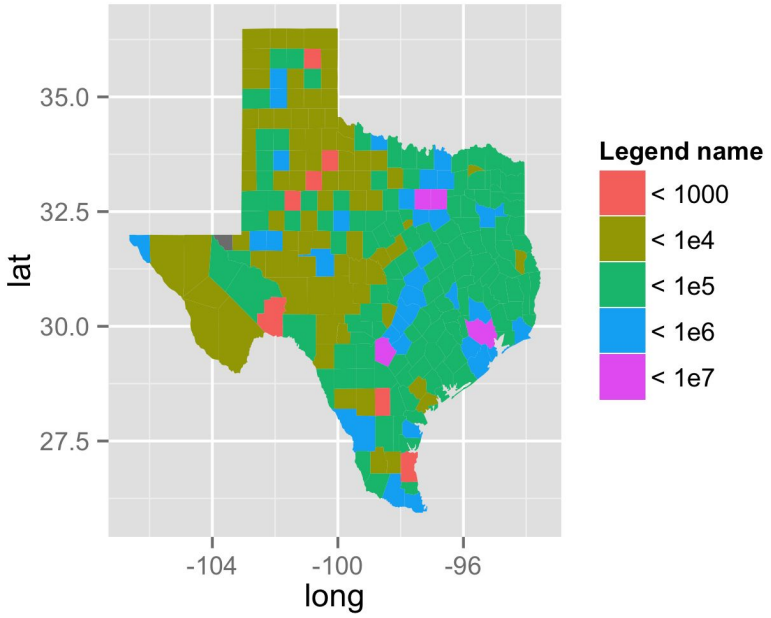
- What if you want to change the legend, but don't want to switch to a new scale?
- We can use the default continuous or discrete scale to the appearance of the plot itself won’t change:
  - scale_aesthetic_continuous()
  - scale_aesthetic_discrete()

Labels

```
tx
```

```
tx + scale_fill_discrete(name =
"Legend name")
```





Your Turn

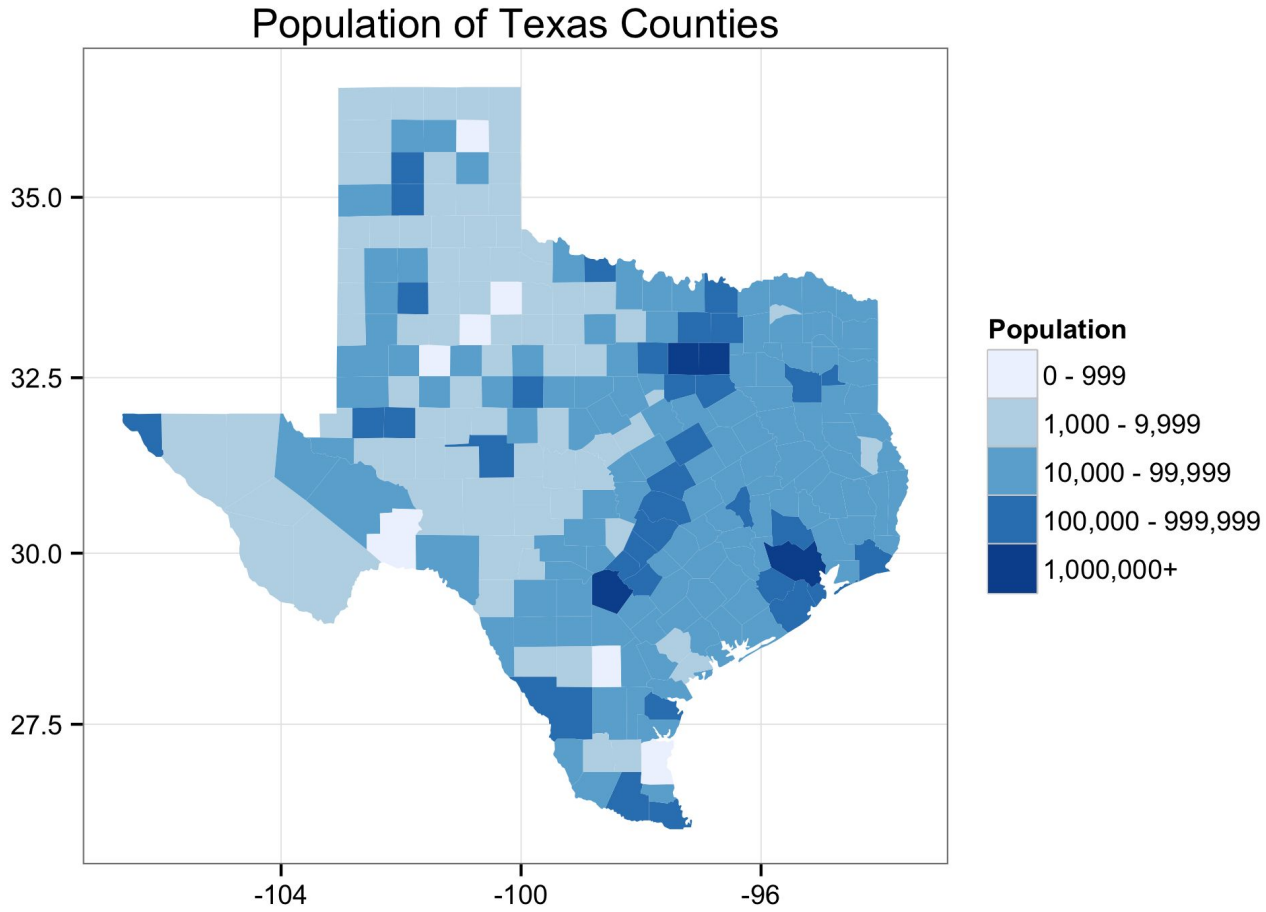
Continue editing map of Texas you created earlier. This time, add to the legend an informative title and a set of more descriptive labels. Use the base code below:

```
#Code from the earlier section; modify to add legend aspects.
tx + scale_fill_brewer(palette = "Blues") +
    xlab("") +
    ylab("") +
    theme_bw() +
    coord_map() +
    ggtitle("Population of Texas Counties")
```

Answer

```
tx + scale_fill_brewer(
  palette = "Blues",
  name = "Population",
  labels = c("0 - 999", "1,000 - 9,999",
  "10,000 - 99,999", "100,000 - 999,999",
  "1,000,000+")) +
  xlab("") +
  ylab("") +
  theme_bw() +
  coord_map() +
  ggtitle("Population of Texas Counties")
```

Answer


Population of Texas Counties
35.0
32.5
30.0
27.5
-104
-100
-96
Population
0 - 999
1,000 - 9,999
10,000 - 99,999
100,000 - 999,999
1,000,000+

Outline

- Customizing Graphics
- Titles
- Coordinate systems
- Scales
- Themes
- Axis labels
- Legends
- **Other visualizations**

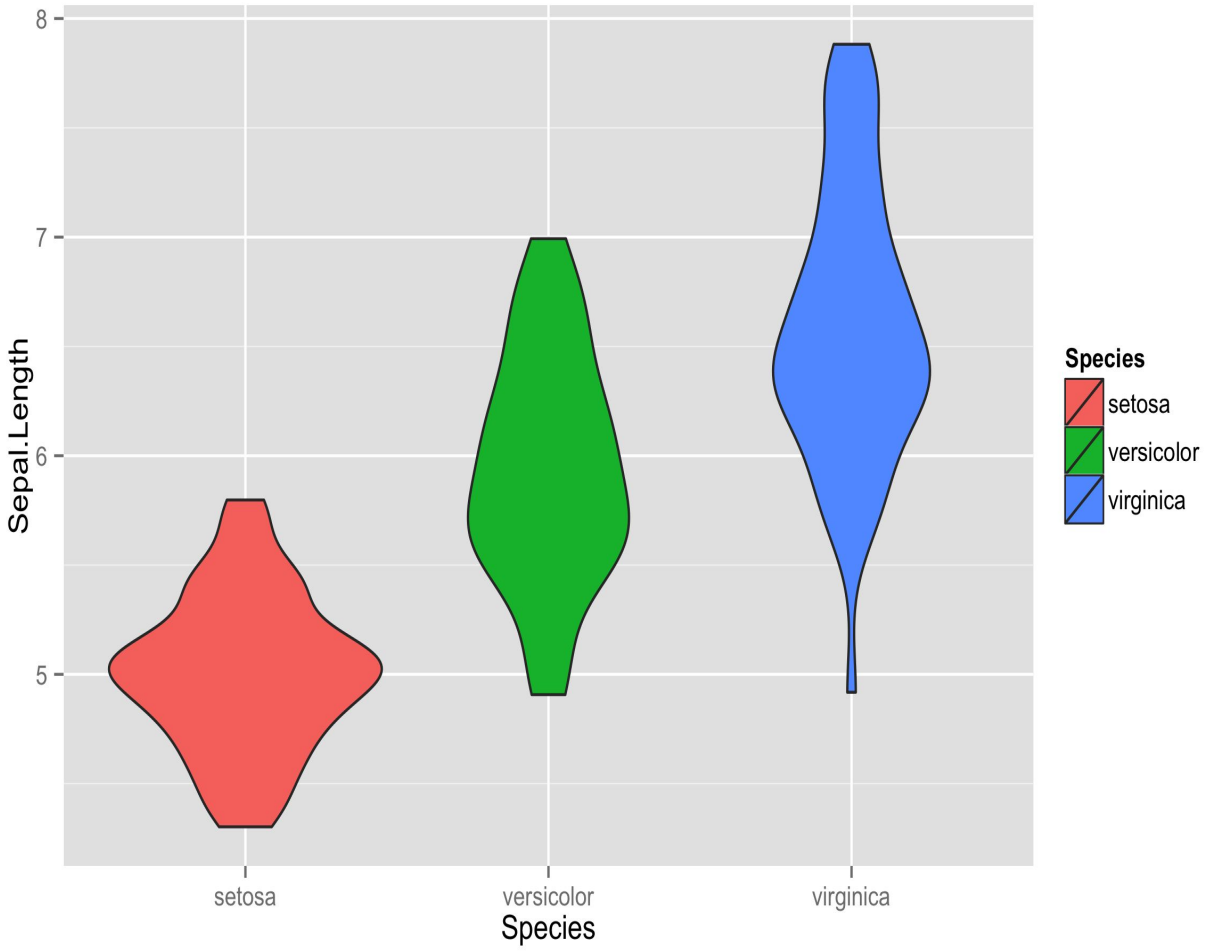
Other Visualizations

Violin Plots

Violin plots are similar to boxplots, but contain more information about the distribution of the data. To produce a violin plot, just use the geom_violin():

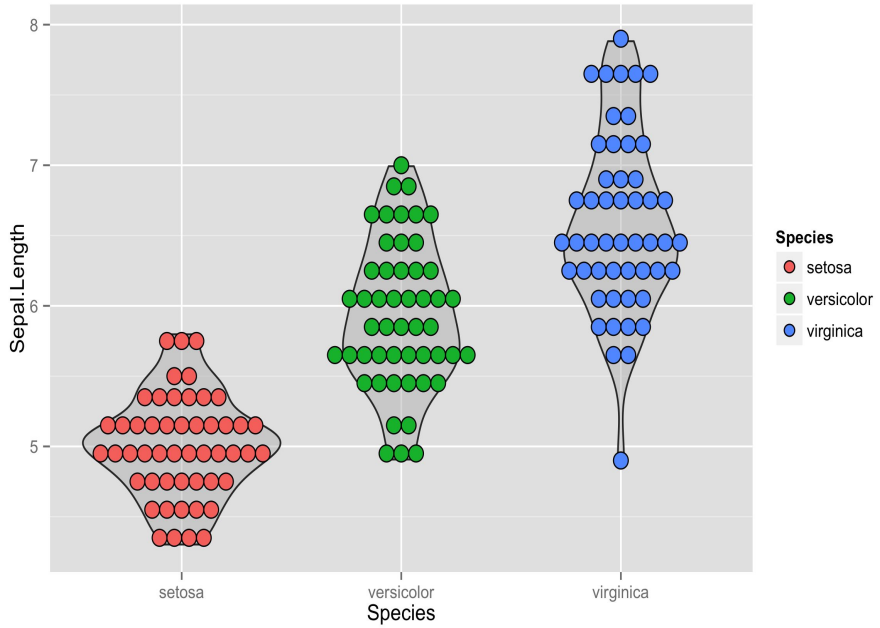
```
p <- ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +
  geom_violin(aes(fill = Species))
p
```

Violin Plots



Violin Plots with Points

```
p <- ggplot(iris, aes(x = Species, y = Sepal.Length)) +
     geom_violin(fill = 'gray', alpha = 0.5) +
     geom_dotplot(aes(fill = Species), binaxis = "y", stackdir = "center")
p
```



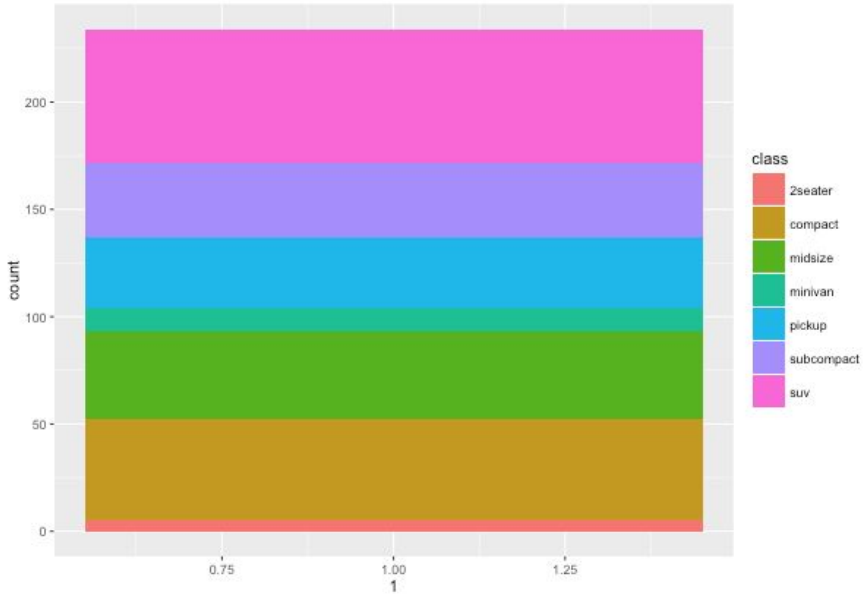
Pie Charts

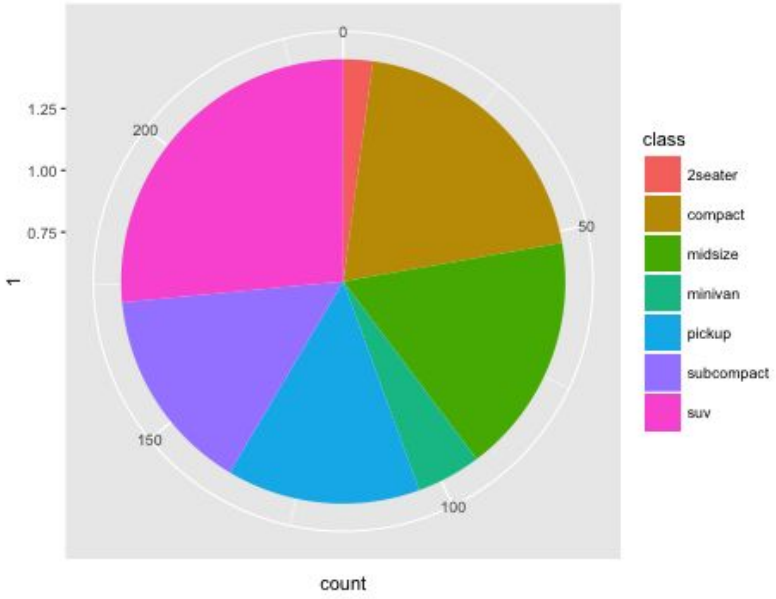
```
p <- ggplot(mpg, aes(x = 1)) +
     geom_bar(aes(fill = class)) +
     coord_polar(theta = "y")
p
```

How is this plot working? Aren't we making a bar plot with the geom_bar()?

Yes, but we're visualizing it in coord_polar()...

Pie Charts


count
200
150
100
50
0
0.75
1.00
1.25
1
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

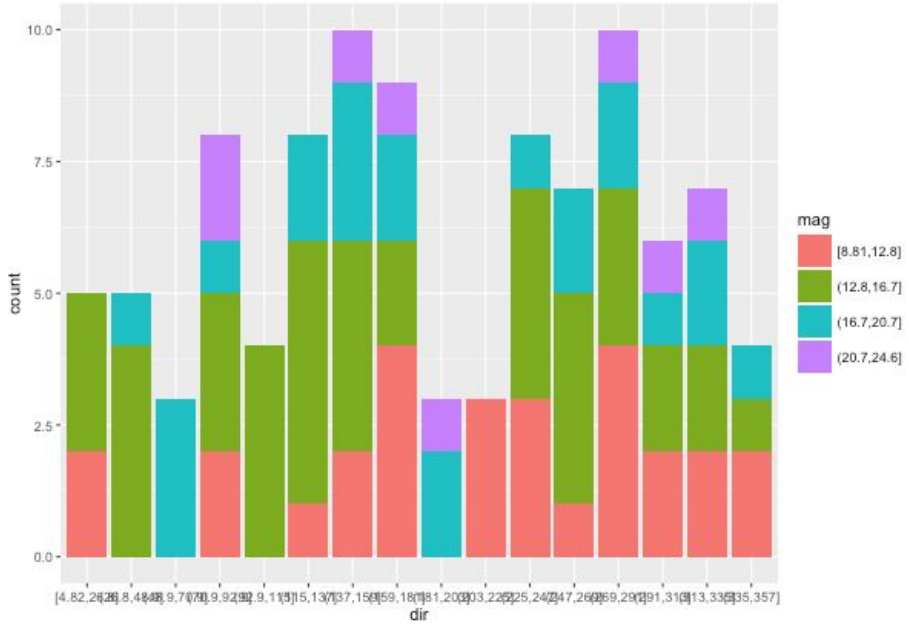

1.25
1.00
0.75
1
0
50
100
150
200
count
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

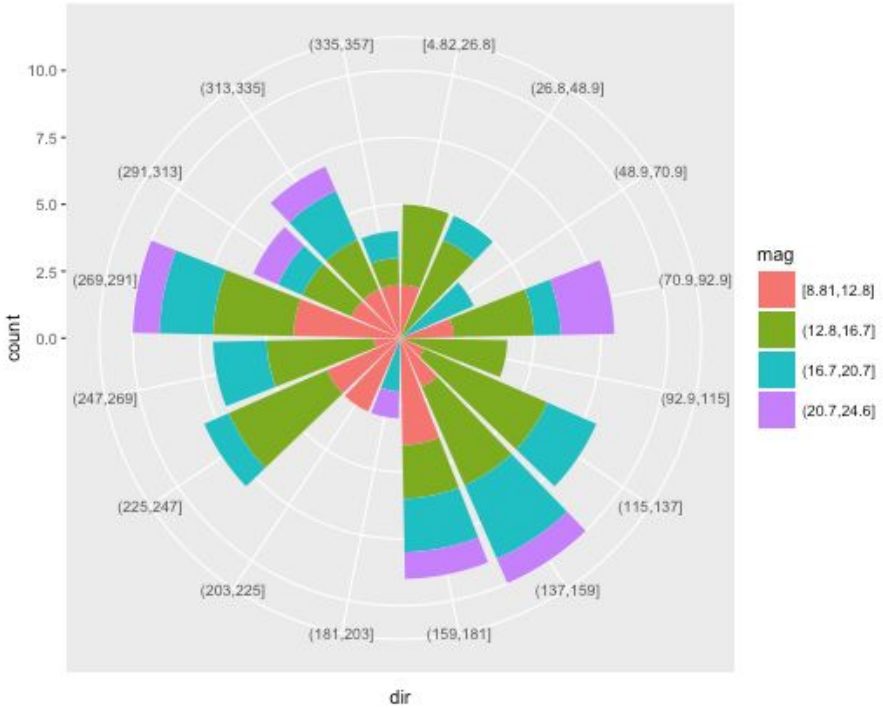
Rose Diagram

A rose diagram, or wind rose, is a commonly used graphics tool by meteorologists. It describes the wind speed and direction distributions in a specific place:

```
set.seed(1)
dir <- cut_interval(runif(100,0,360), n=16)
mag <- cut_interval(rgamma(100,15), n=4)
sample = data.frame(dir=dir, mag=mag)
p <- ggplot(sample, aes(x=dir, fill=mag) ) +
     geom_bar() +
     coord_polar()
p
```

Rose Diagram





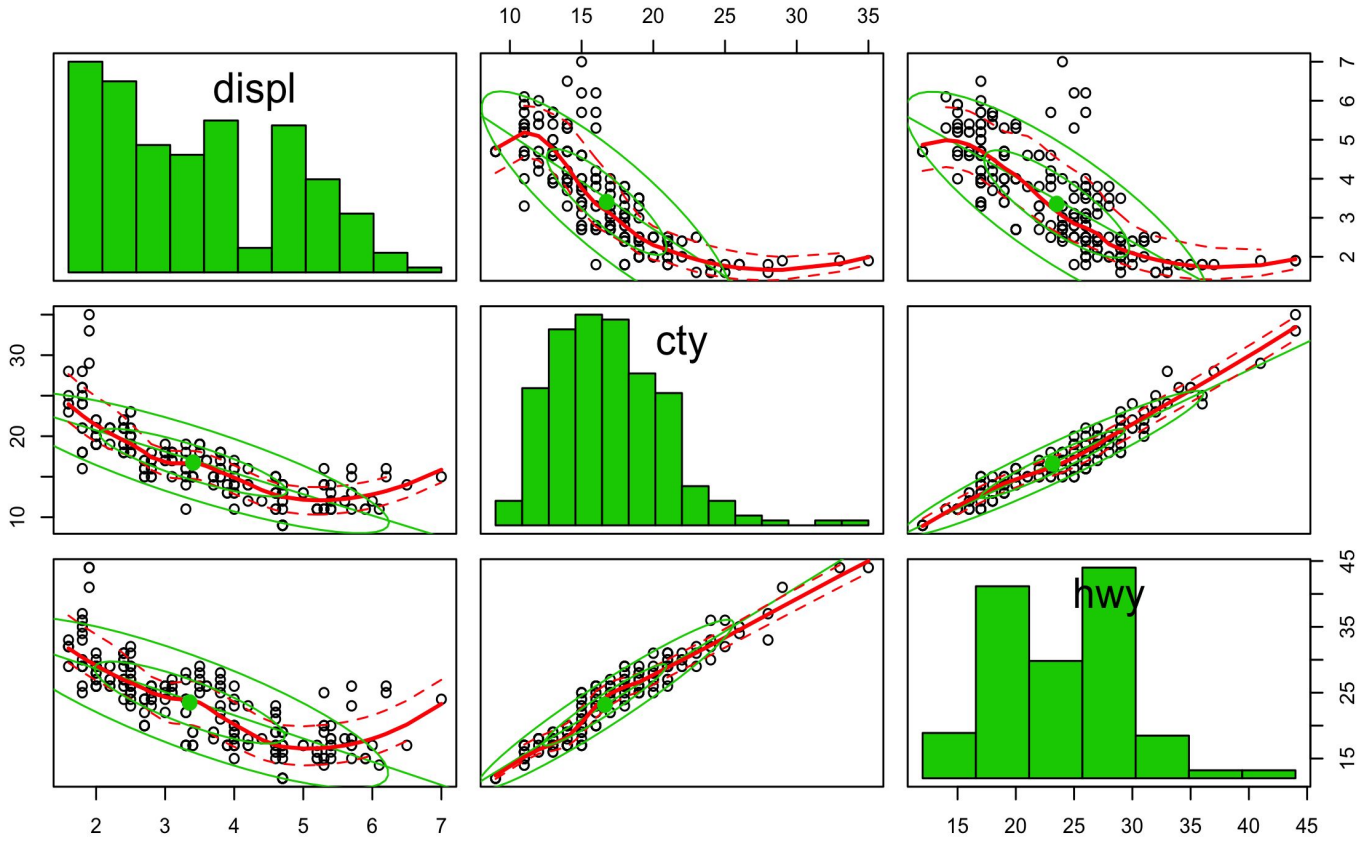
Relationships in Multivariate Data

Scatter Plots with Multidimensional Data

For larger multivariate sets where pairwise comparisons are of interest, you can create a scatterplot matrix:

```
install.packages("car")
library(car)
scatterplotMatrix(mpg[ ,c(3,8,9)],
                  diagonal='histogram',
                  ellipse=TRUE)
```

Scatter Plots with Multidimensional Data

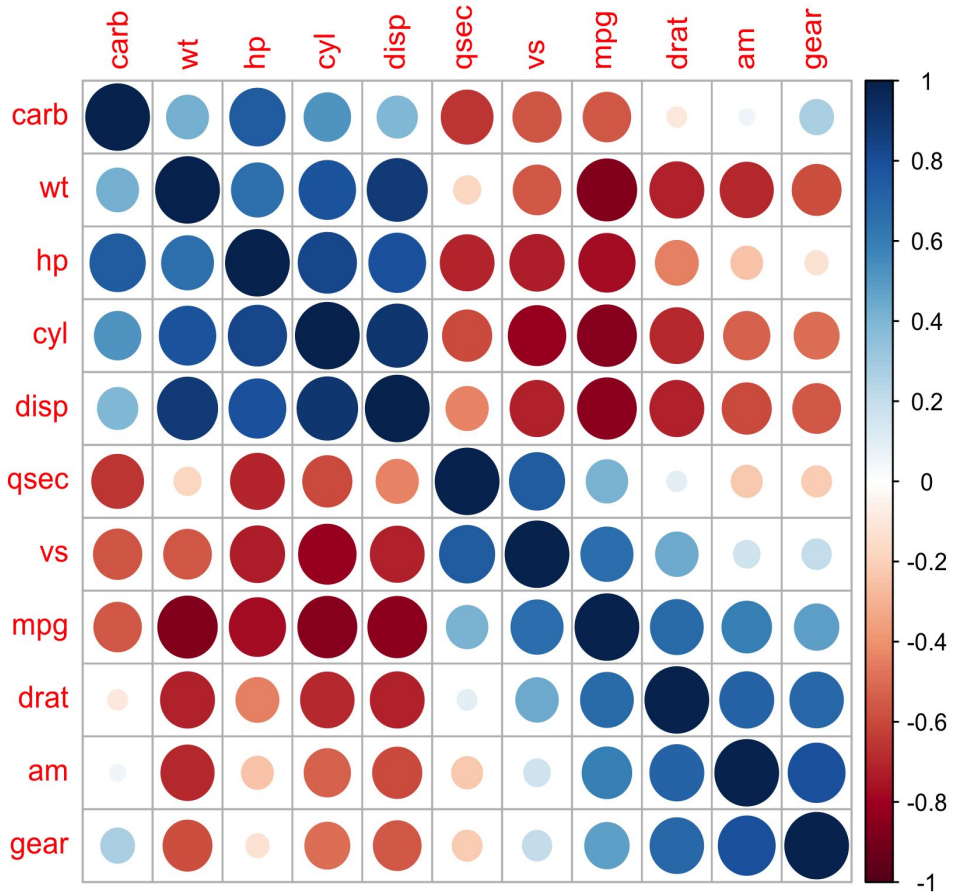


Scatter Plots with Multidimensional Data

If given many numerical variables, concentrated displays can be useful; for example, we can use a correlation plot.

```
install.packages("corrplot")
library(corrplot)
corrplot(cor(mtcars), order = "hclust")
```

Scatter Plots with Multidimensional Data



Appendix

- More Useful Packages & Resources

More Useful Packages & Resources

Where to go from here

- Learning ggplot2
  - ggplot2 mailing list
  - http://groups.google.com/group/ggplot2
- stackoverflow
  - http://stackoverflow.com/tags/ggplot2
- ggplot2 book
  - http://amzn.com/0387981403

More Useful Packages

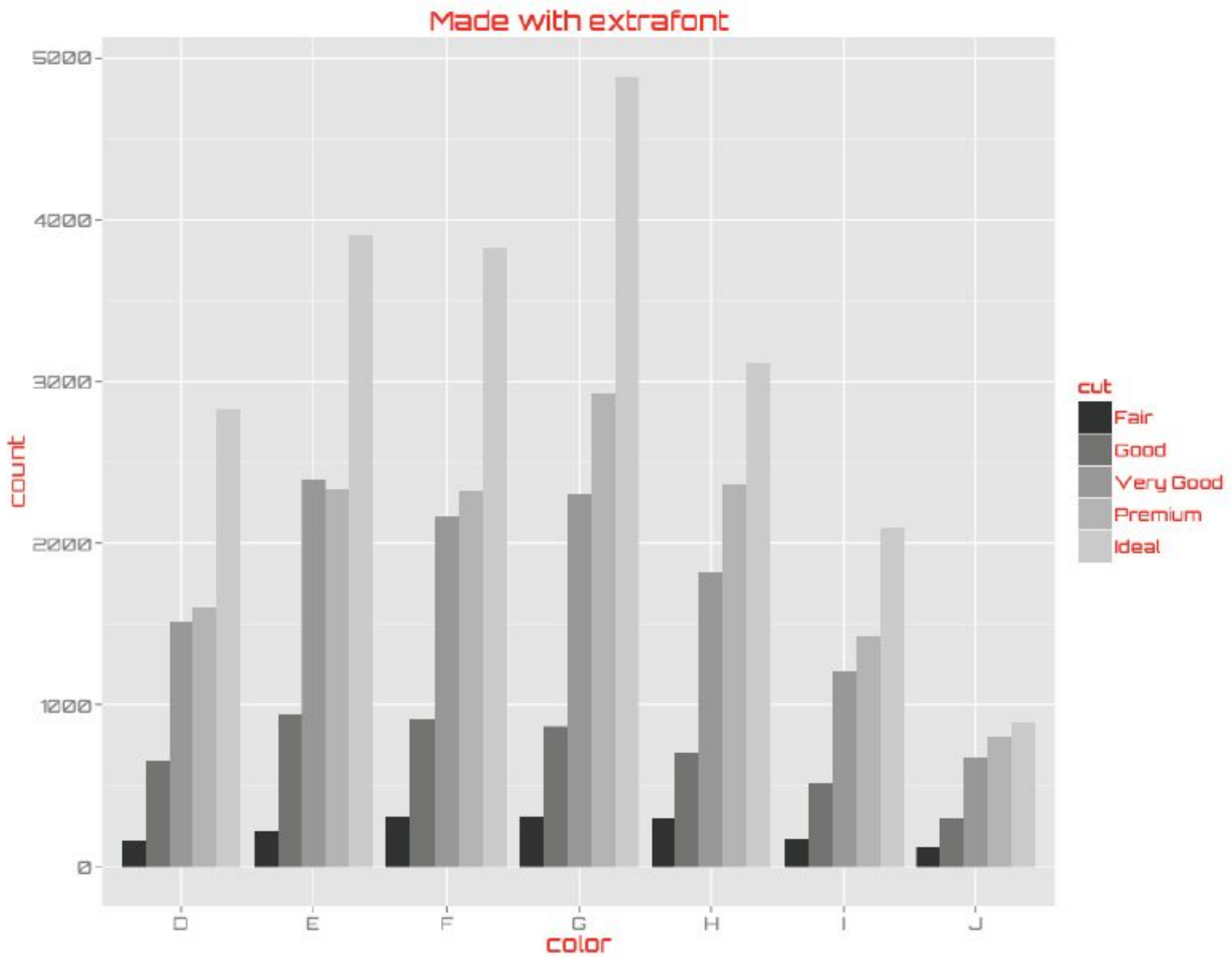
use system fonts in plots

extrafont

```
install.packages("extrafont")
```

https://github.com/wch/extrafont

More Useful Packages



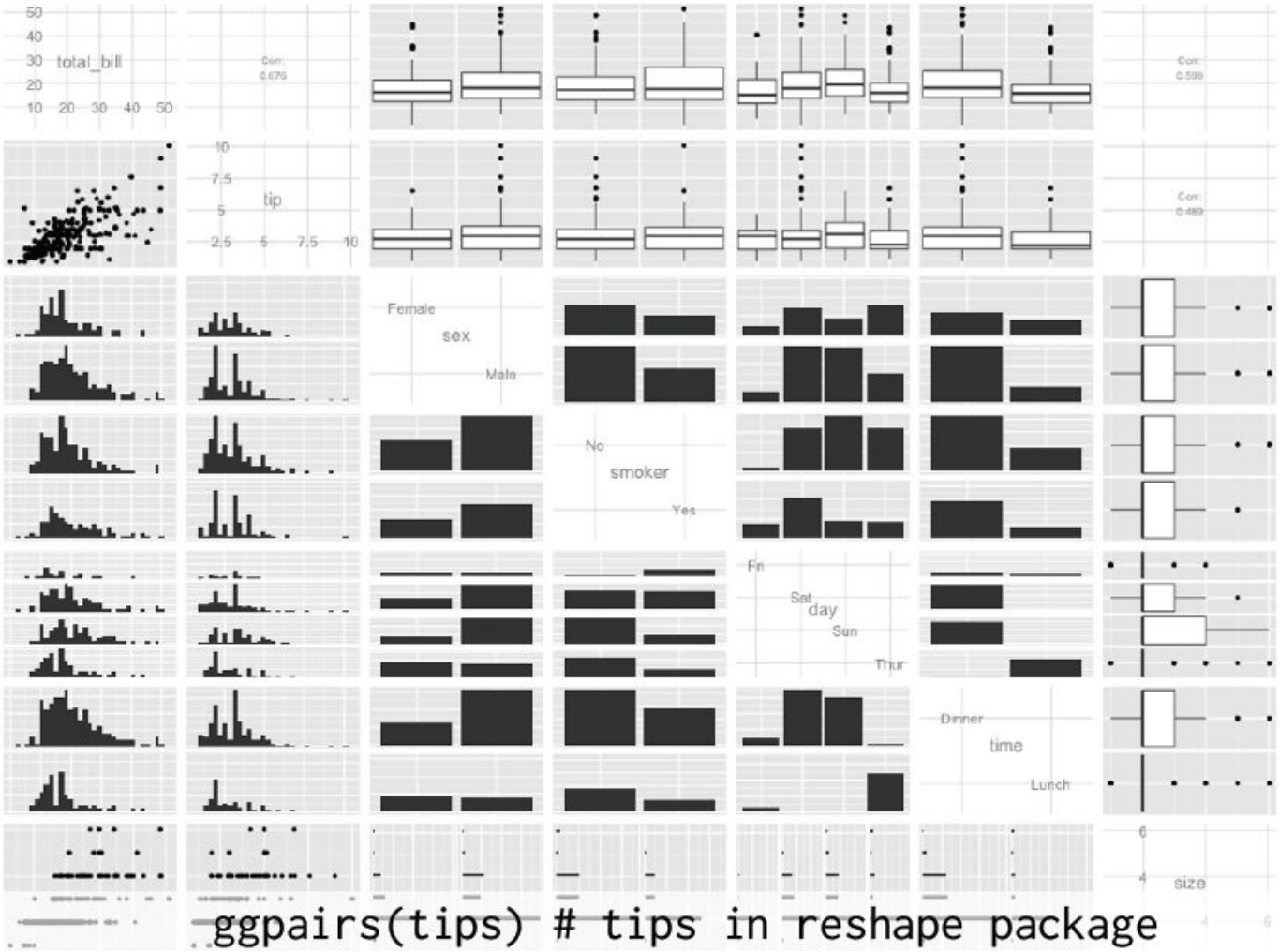
More Useful Packages

specialized plot types

GGally

```
install.packages("GGally")
```

More Useful Packages

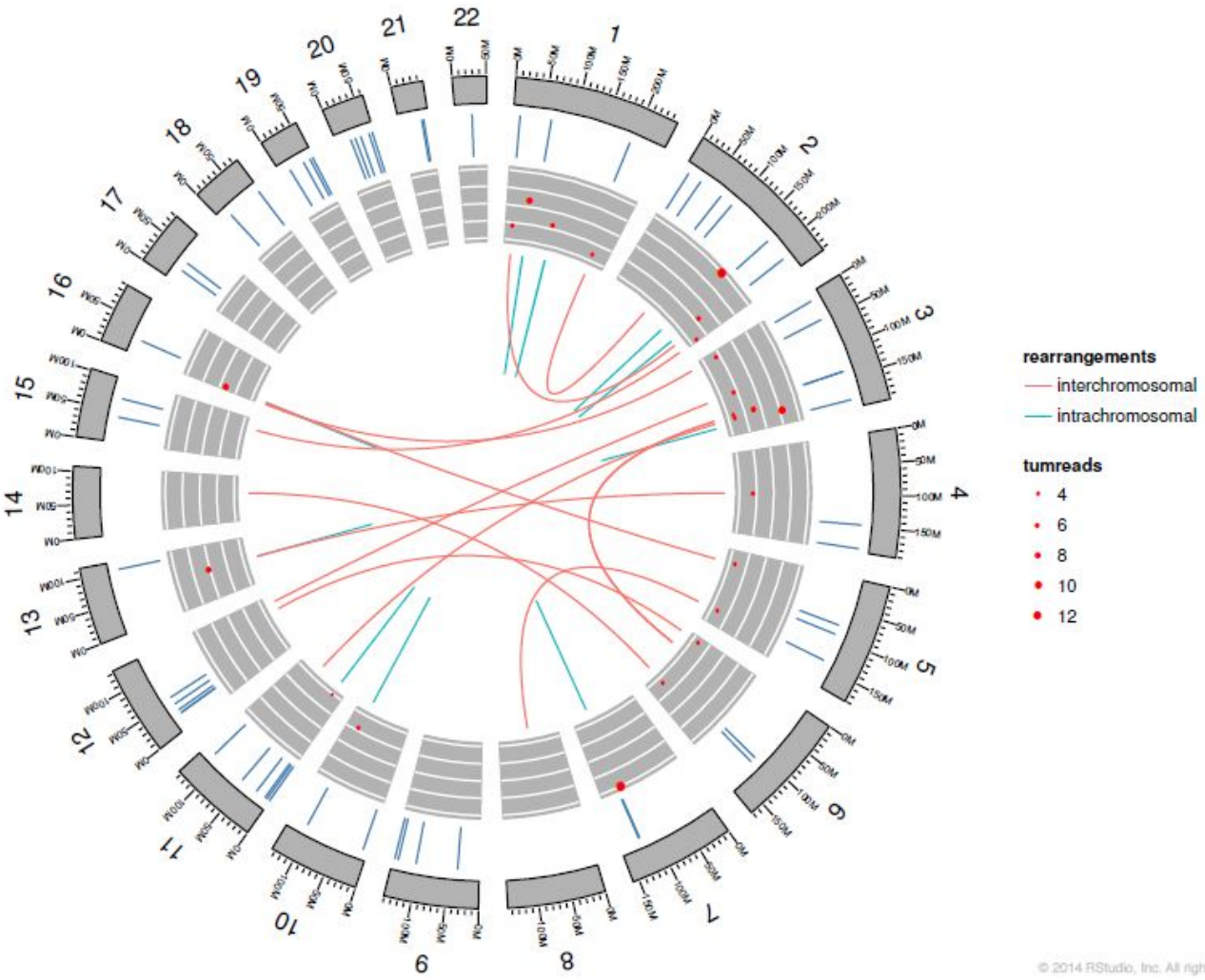



```
ggpairs(tips) # tips in reshape package
```

More Useful Packages


```
source("http://bioconductor.org/biocLite.R")
biocLite("ggbio")
```

More Useful Packages



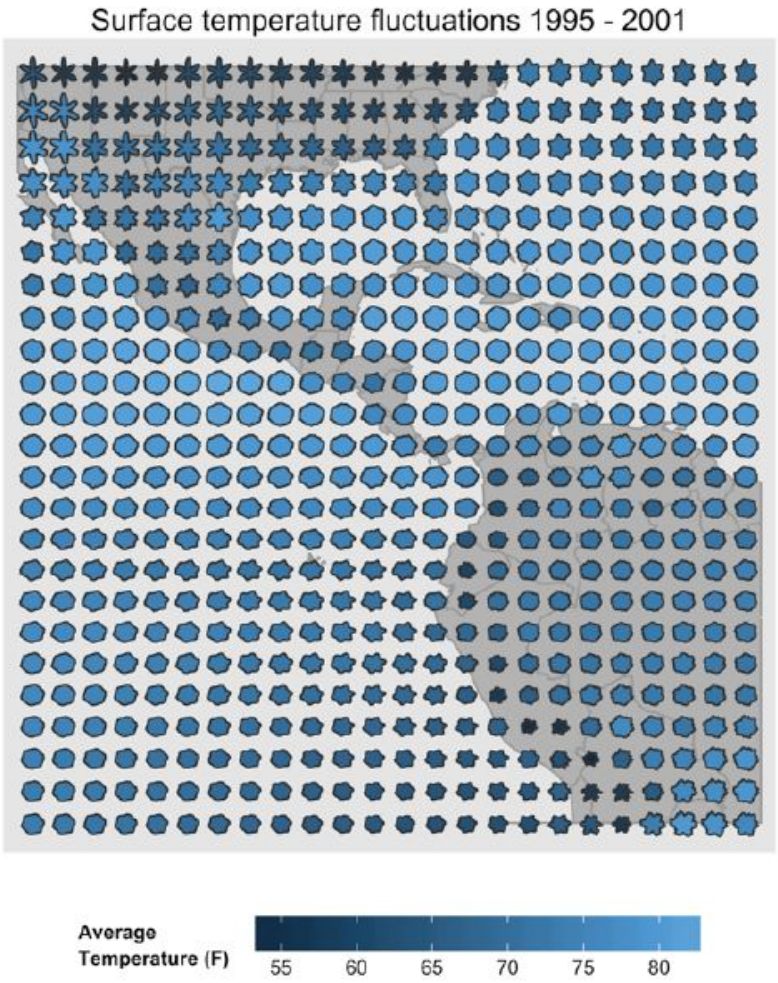
More Useful Packages

subplots and glyphs

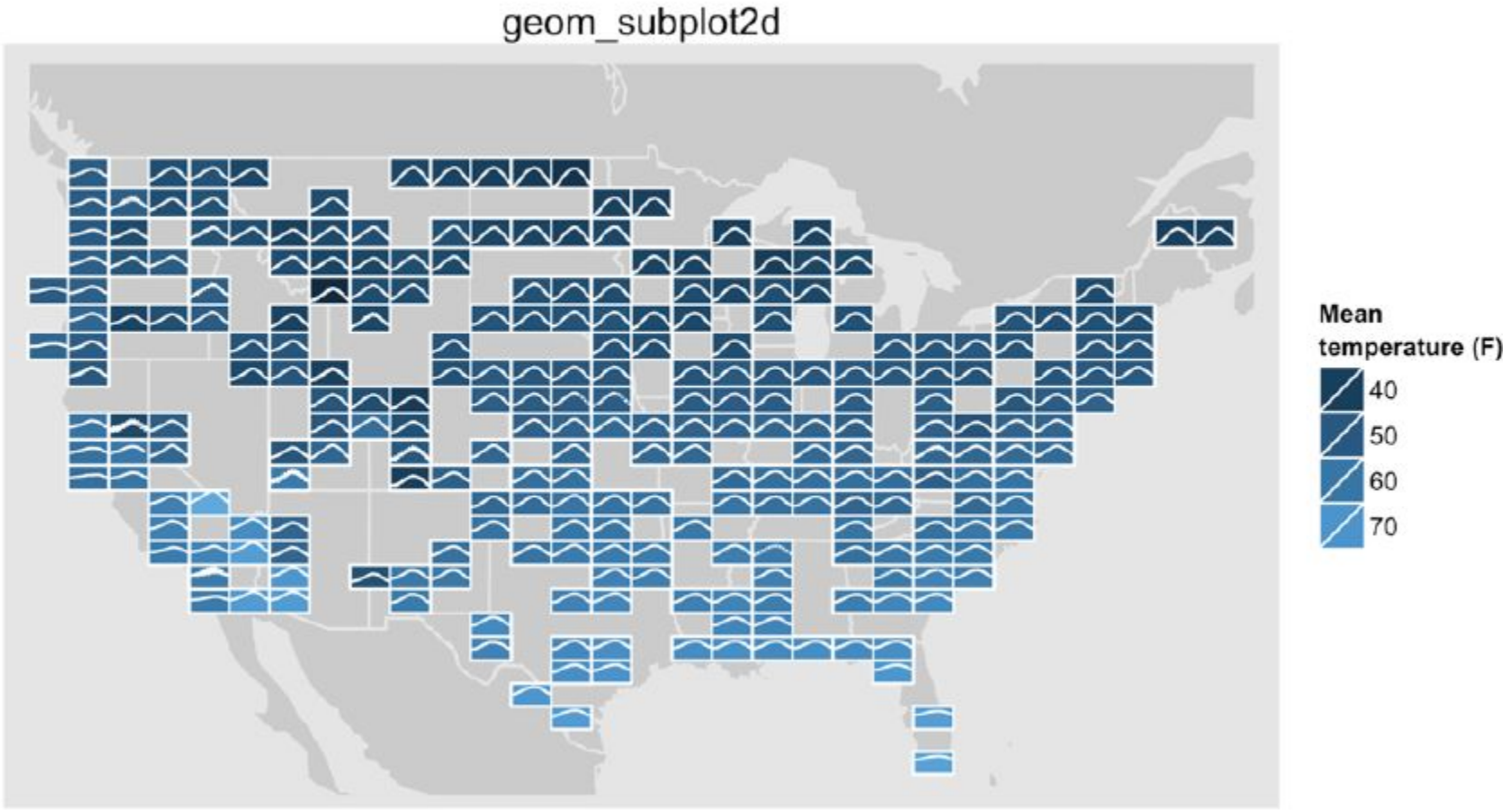
ggsubplot

```
install.packages("ggsubplot")
```

More Useful Packages



More Useful Packages



More Useful Packages