NYC DATA SCIENCE ACADEMY

Data Science with R (Data Analytics)

Customizing Graphics in 'ggplot2'

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

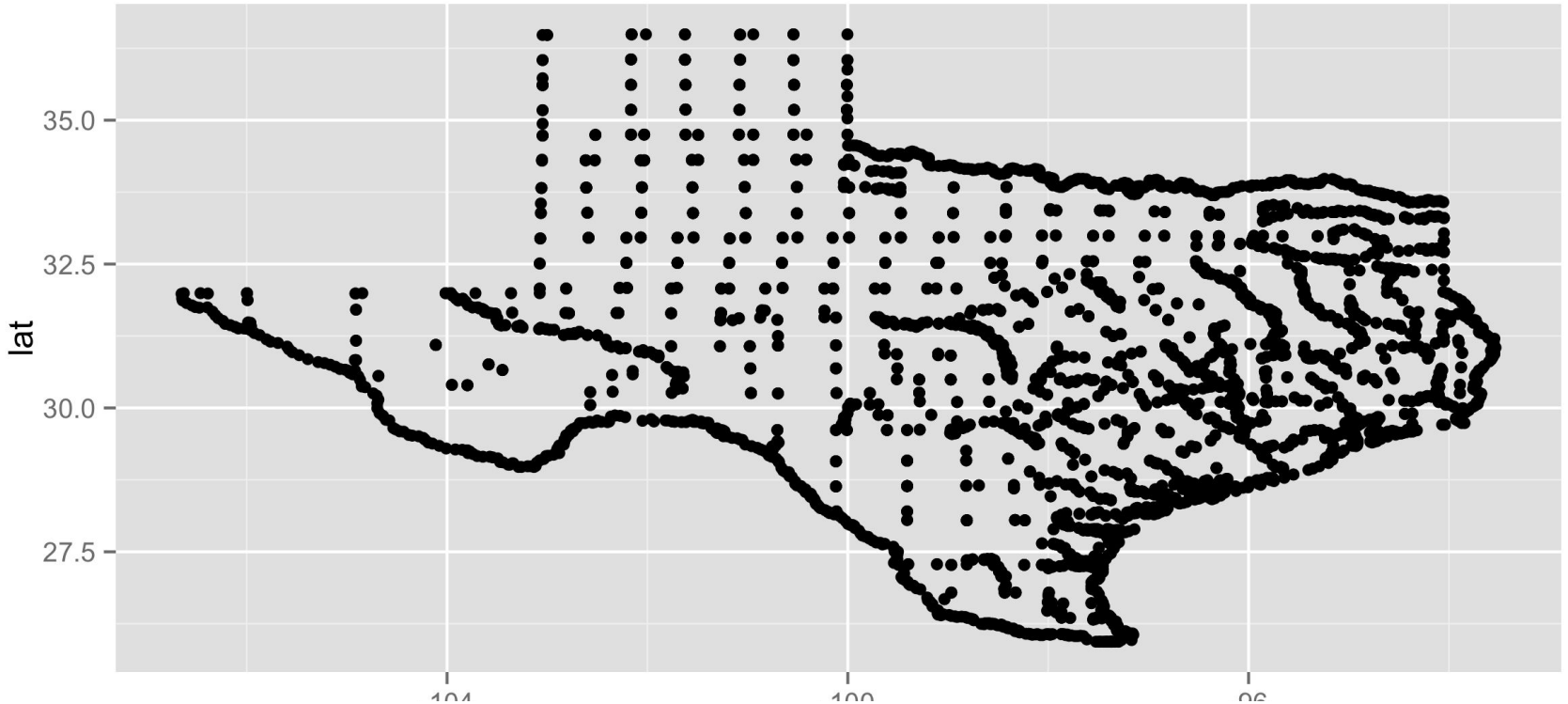
- For today, we will use a graph based on the Texas data set as an example; it displays the populations of counties in Texas.

```
library(ggplot2)
texas = read.csv("data/texas.csv")
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

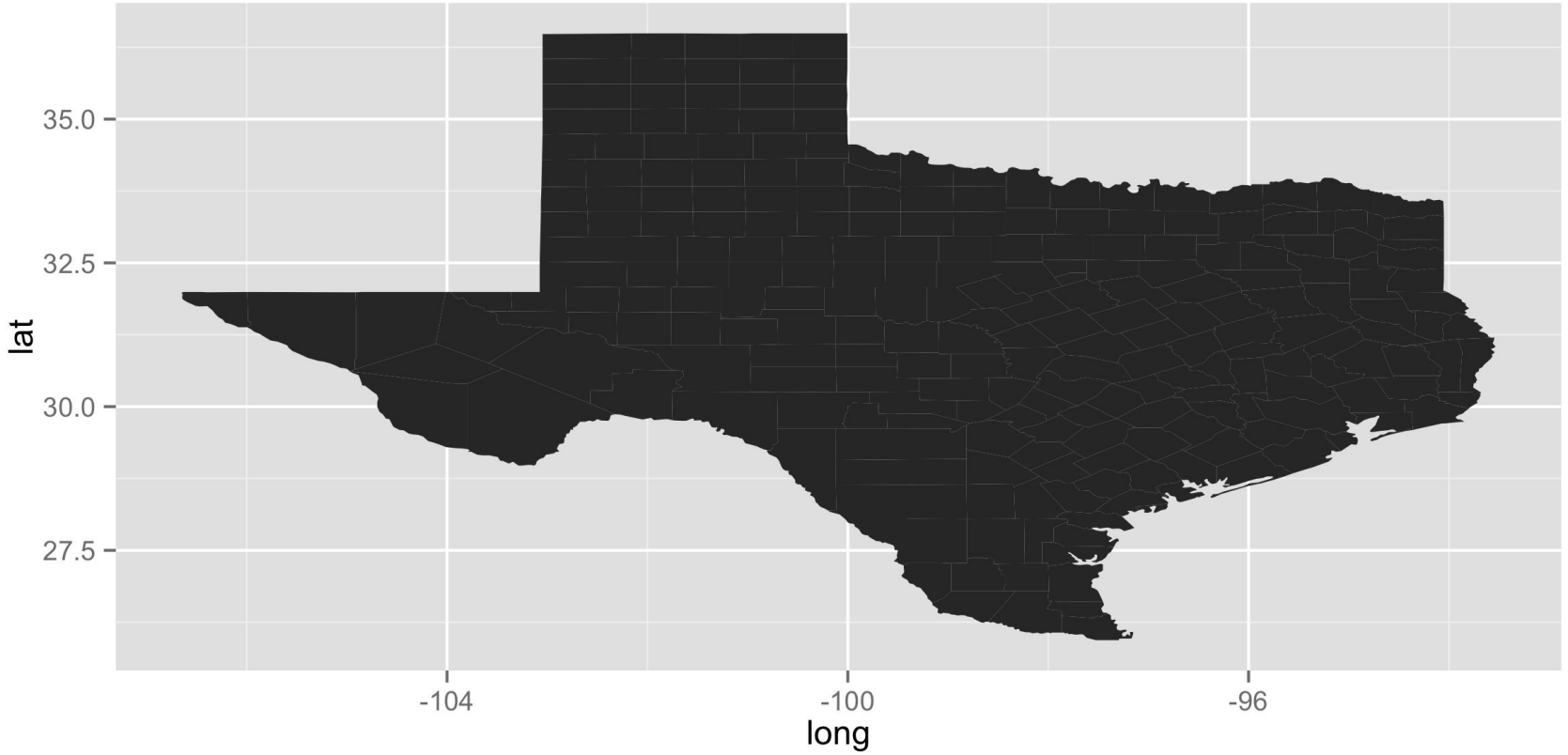
Customizing Graphics

```
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

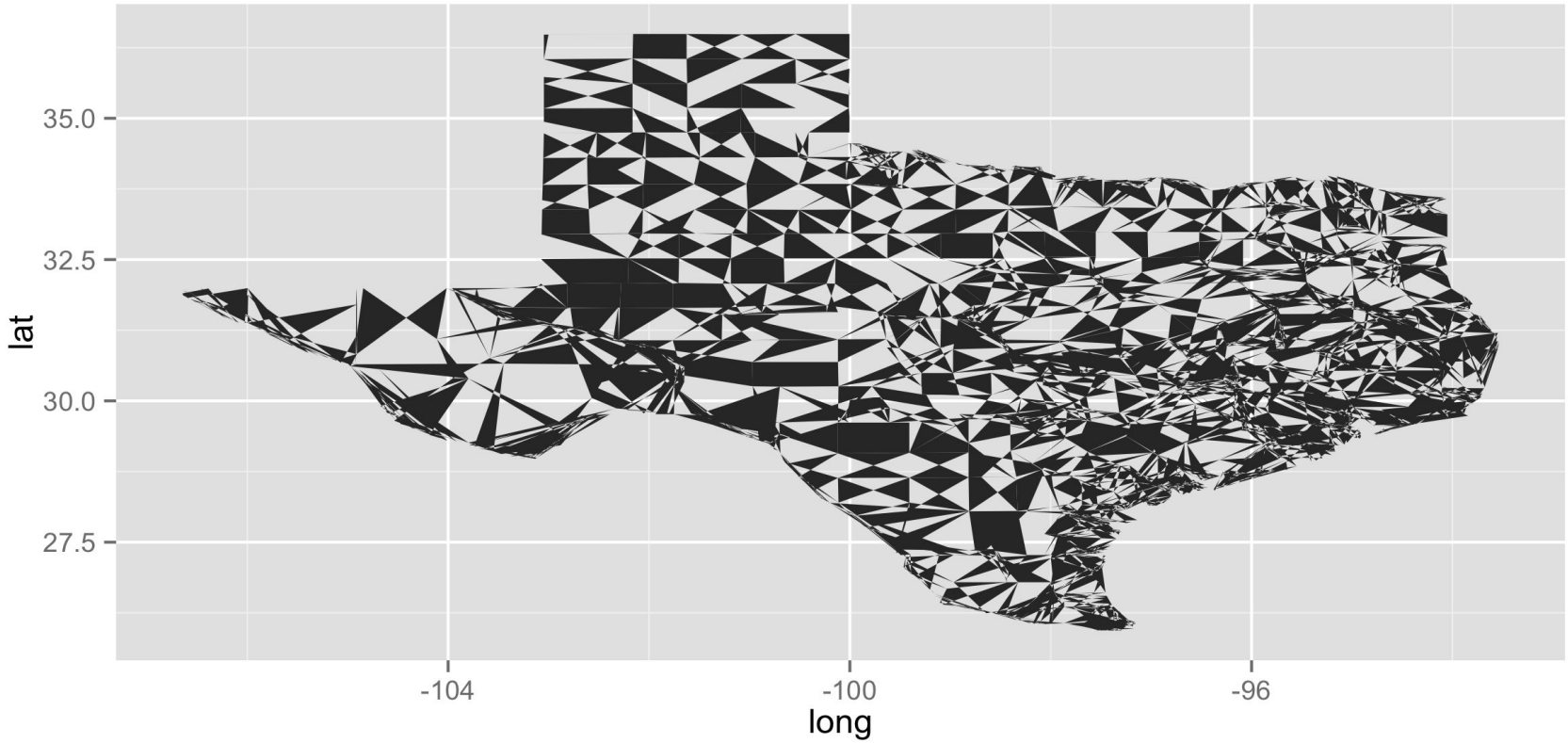
```
head(texas)
```

	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

Customizing Graphics

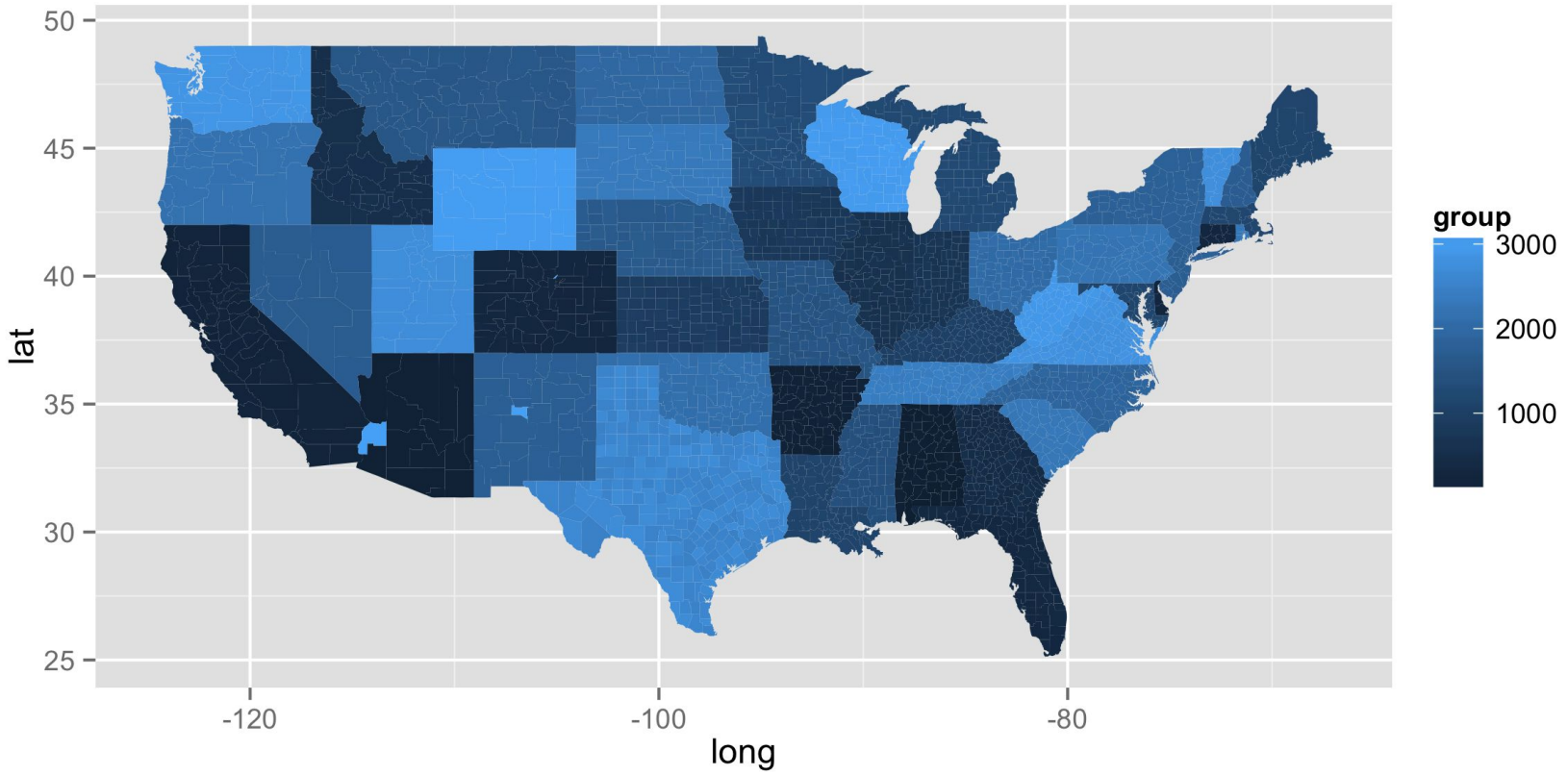
```
texas2 = texas[sample(nrow(texas)), ]  #Row order
matters!
ggplot(data = texas2, aes(x = long, y = lat)) +
```



Customizing Graphs

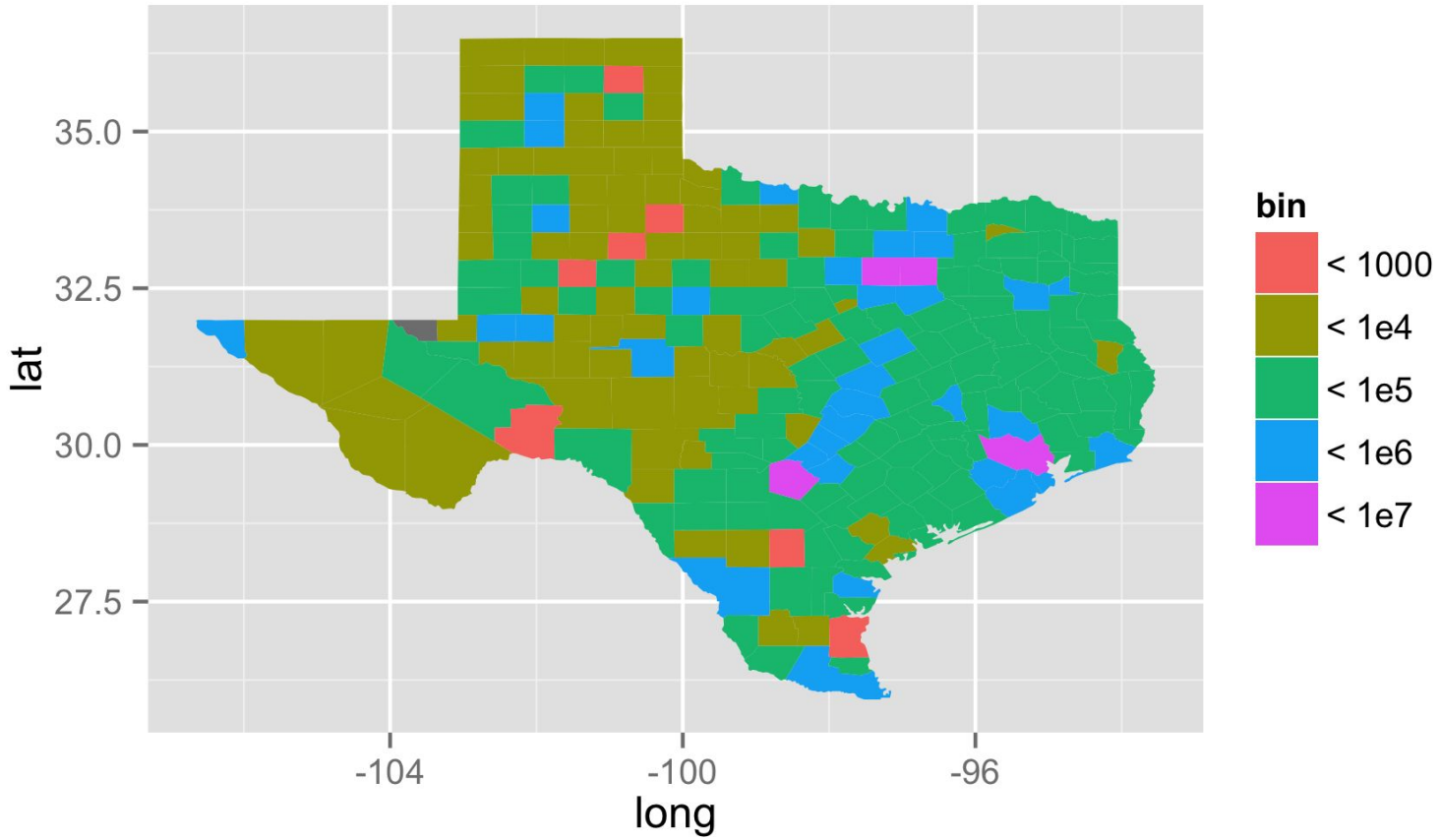
```
# install.packages("maps")
# help(package = "maps")
library(maps)
counties = map_data("county")    # Using the built-in USA county
                                 # map dataset.
ggplot(data = counties, aes(x = long, y = lat)) +
geom_polygon(aes(group = group, fill = group))
```

Customizing Graphs



Customizing Graphs

Use the texas dataset to recreate this map:



Customizing Graphs

Use texas to recreate this map.

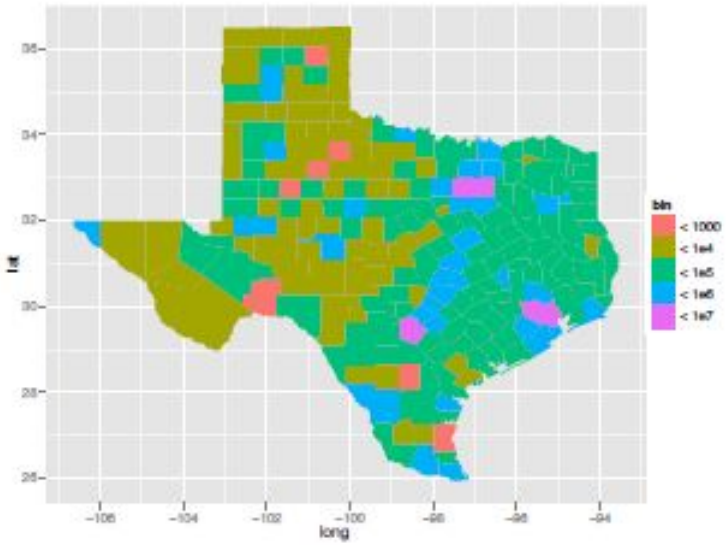
```
tx <- g + geom_polygon(aes(group = group, fill = bin))
```

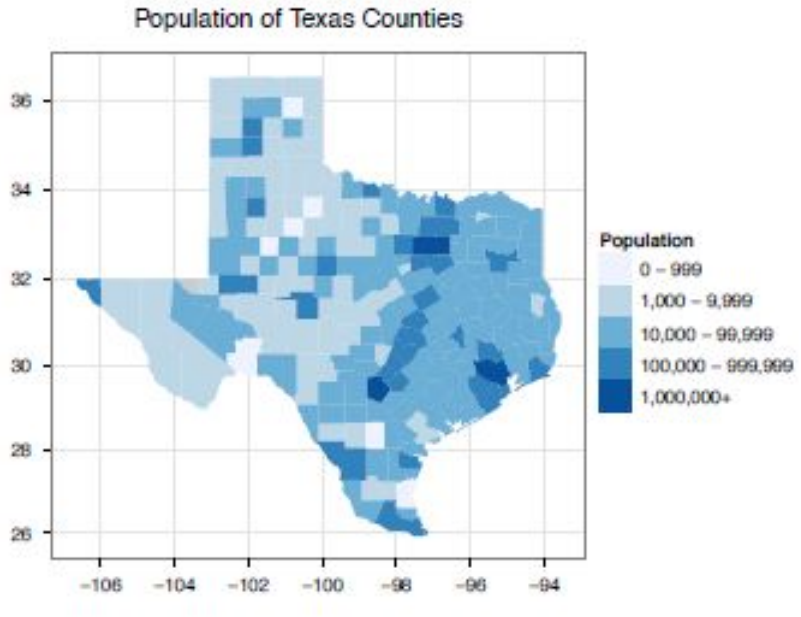
```
tx
```

- Notice that we're saving the plot to the object tx.
- You should, too.
- This will make it easy to refer to the plot later.

Customizing Graphs

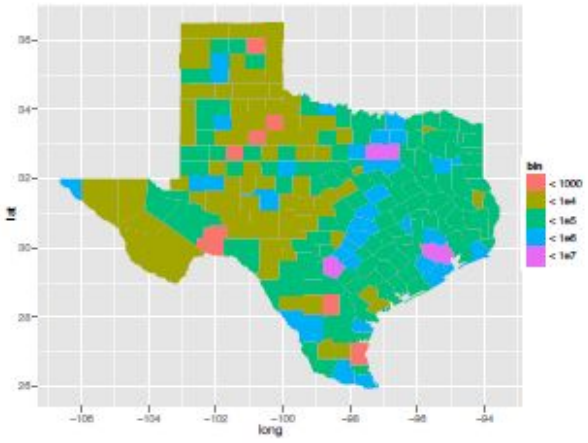
Which do you prefer?

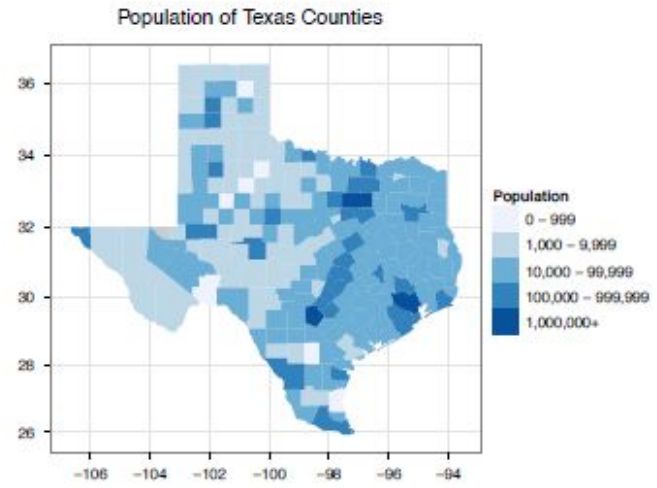




Our Goal

- title added and axis labels removed
- correct aspect ratio
- color scheme
- white background
- legend labels





Outline

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

Titles

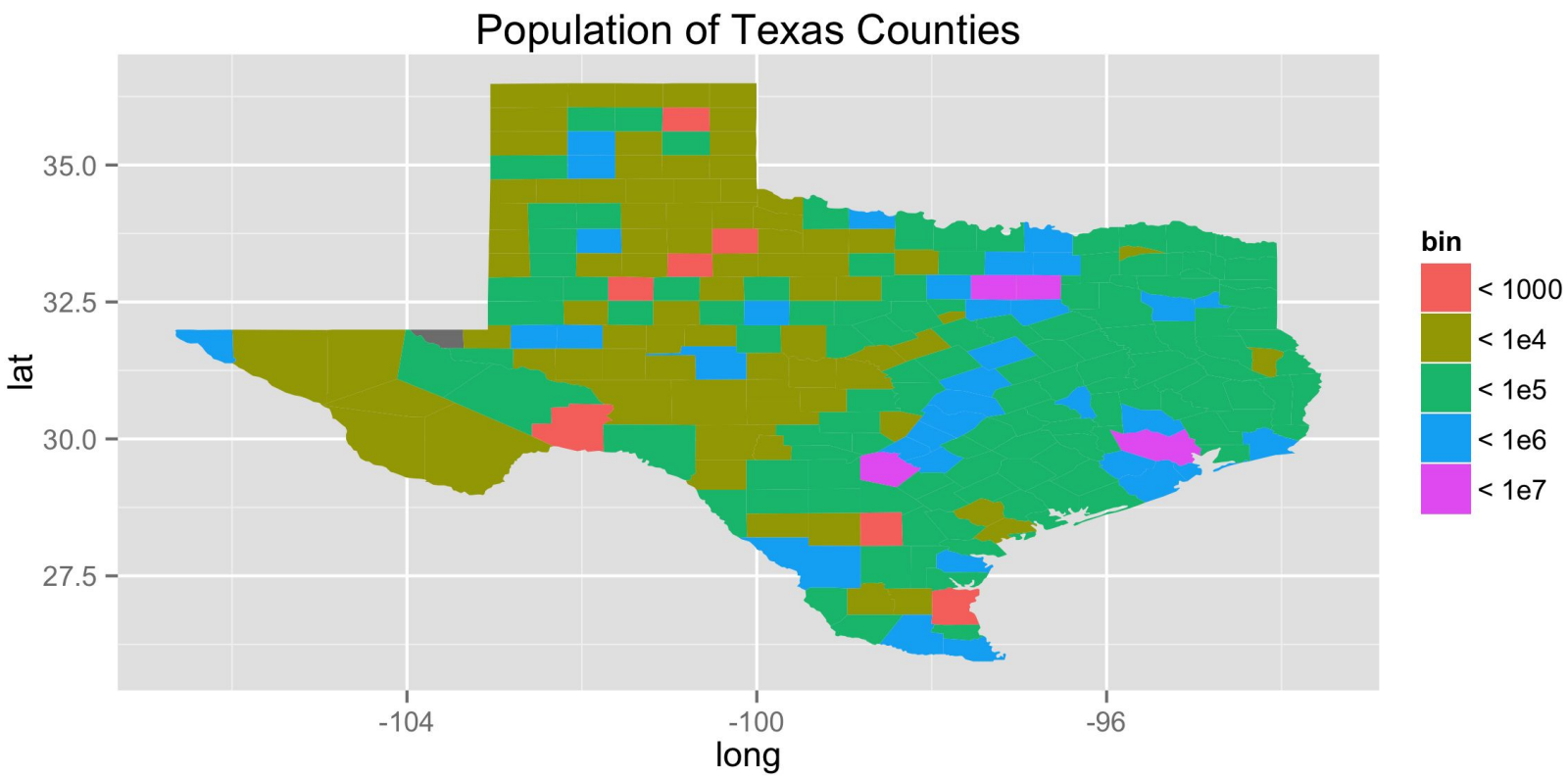
Title

- Recall that you can modify ggplot2 graphs by adding objects to them.
- Use 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

Use ggtitle to add a title:



Title

Additions are not permanent!

- To create a new graph that always has a title:

```
tx + ggtitle("Population of Texas Counties")
tx

#To create a new graph that always has a title, save a new
object.
tx2 <- tx + ggtitle("Population of Texas Counties")
tx2
```

Manipulating Plots

Manipulating Plots

What is in a ggplot object?

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

Manipulating Plots

```
str(cp)
# 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

Determine the coordinate plane to draw the graph on.

```
g$coordinates
```

Coordinate Systems

Determine the coordinate plane to draw the graph on.

```
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
```

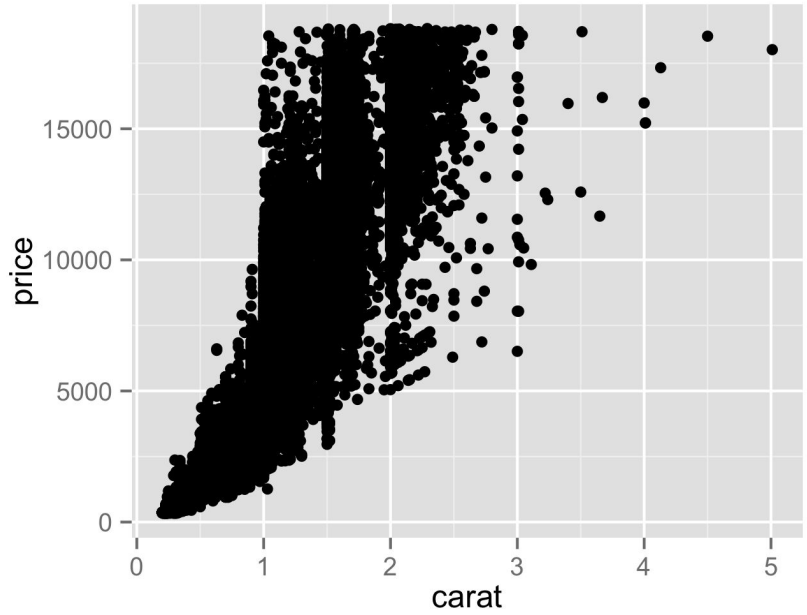
with

```
g2$coordinates
```

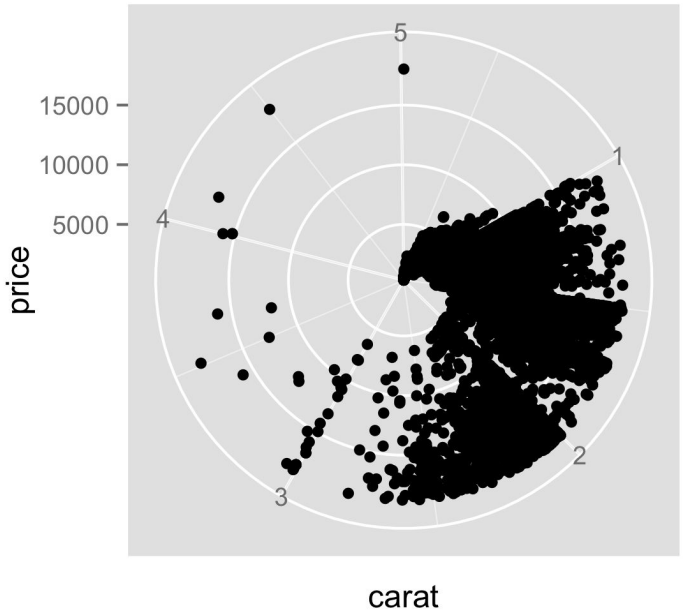
Coordinate Systems

polar

```
g
```



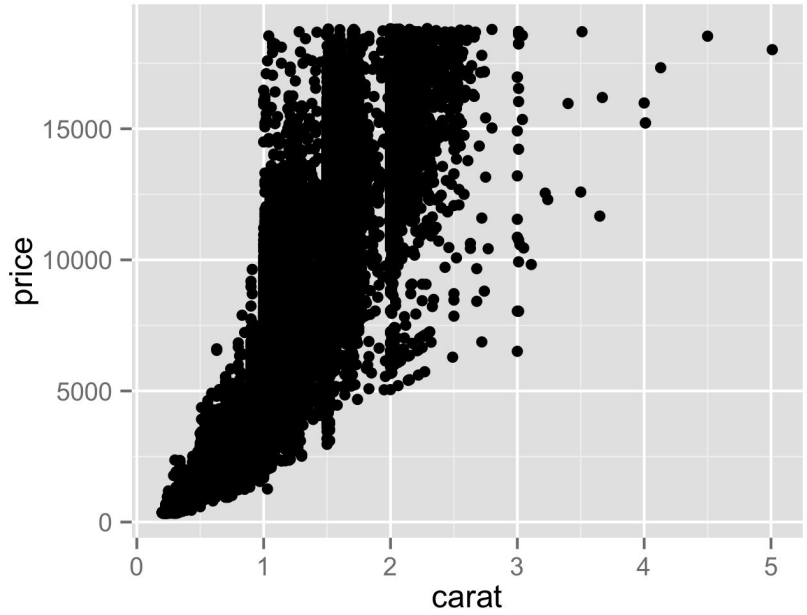

```
g + coord_polar()
```



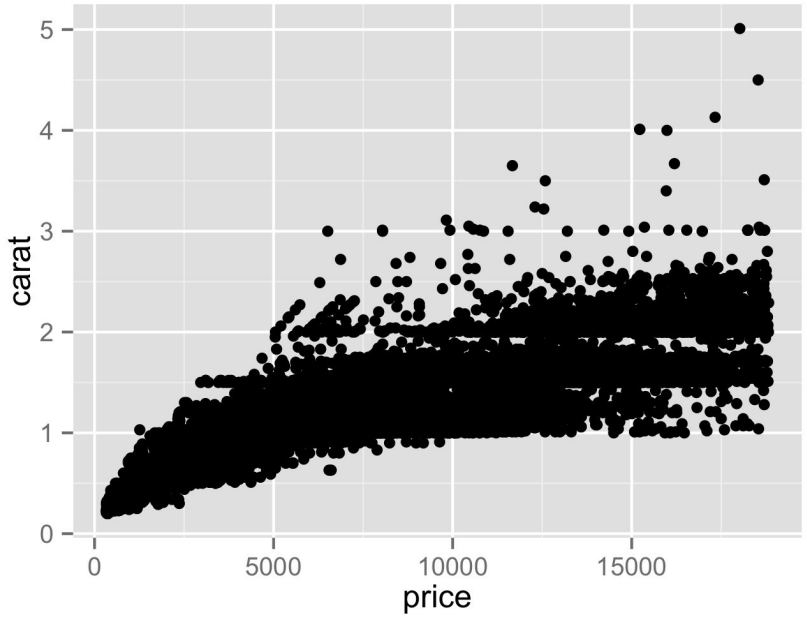
Coordinate Systems

flip

```
g
```



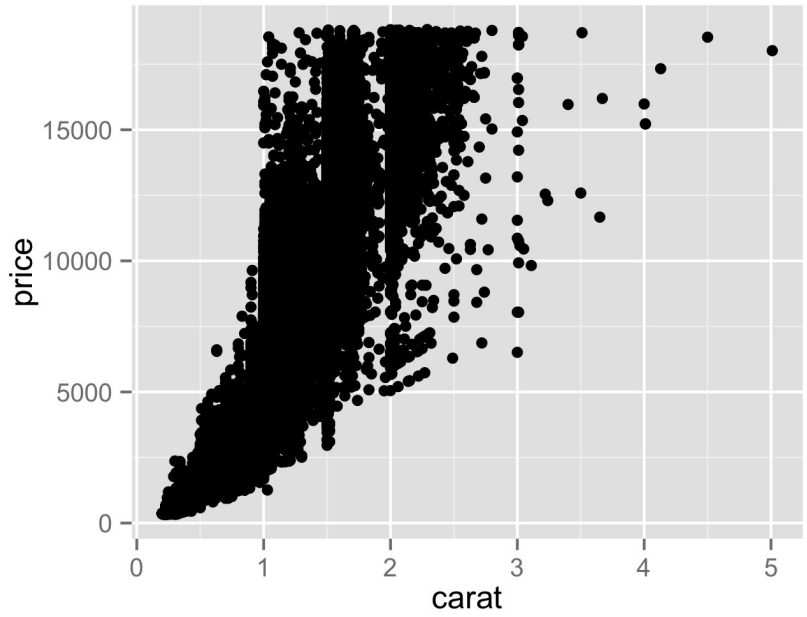

```
g + coord_flip()
```



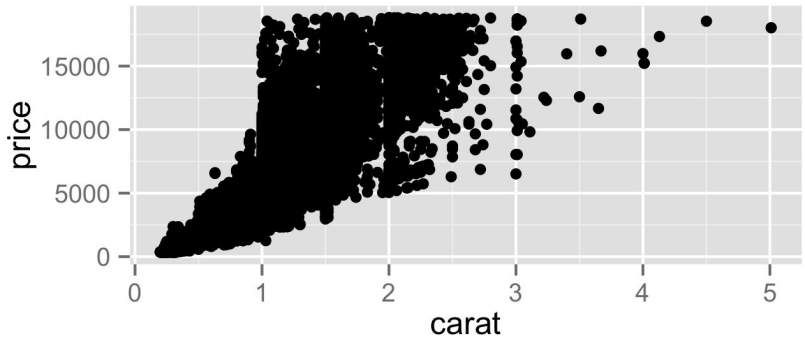
Coordinate Systems

fixed

```
g
```



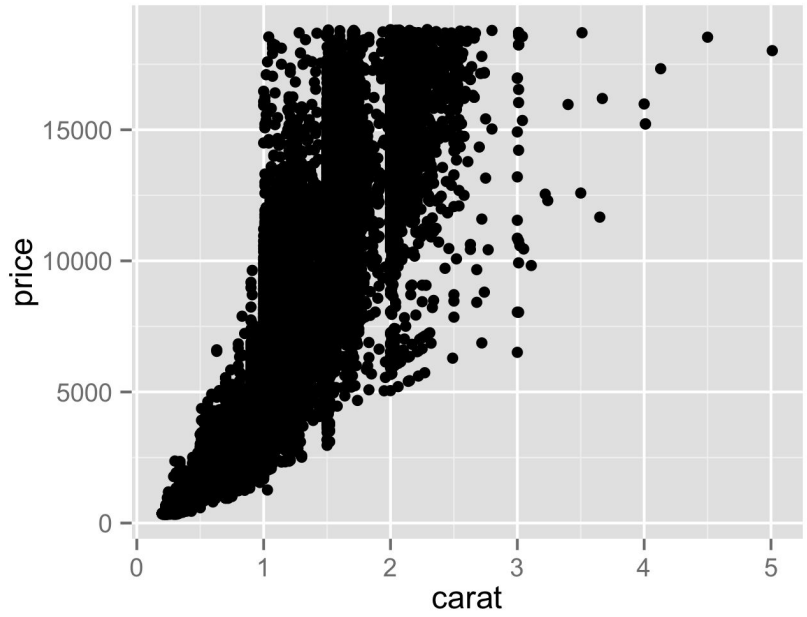

```
g + coord_fixed(ratio =
1/10000)
```



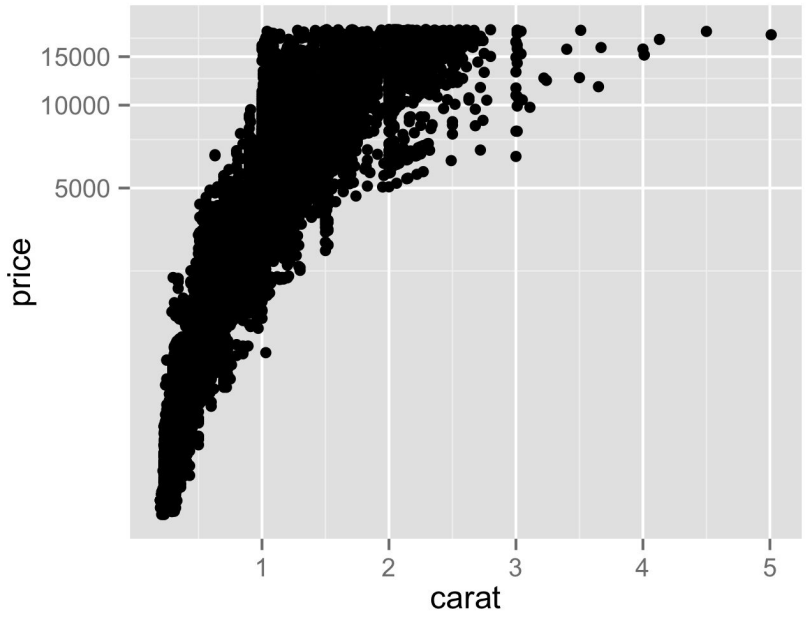
Coordinate Systems

trans

```
g
```




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

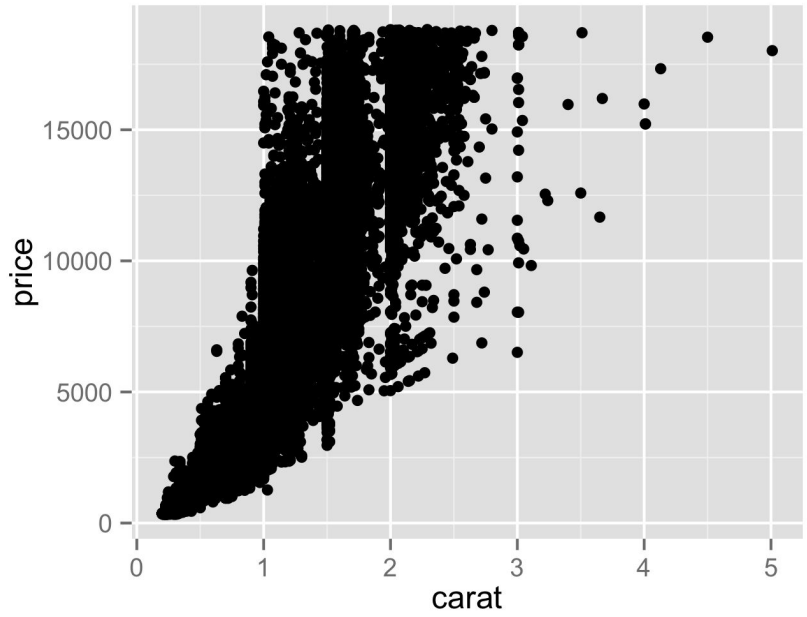


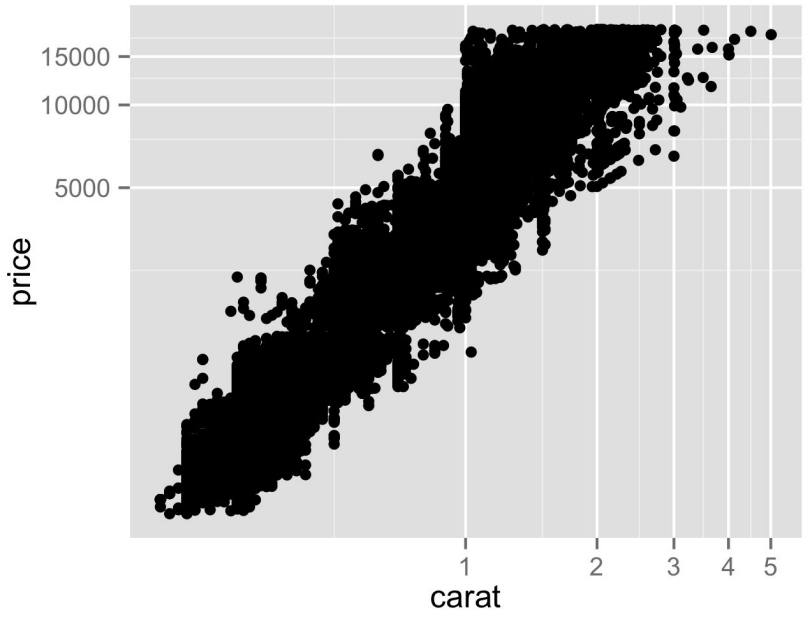
Coordinate Systems

trans

```
g
```

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

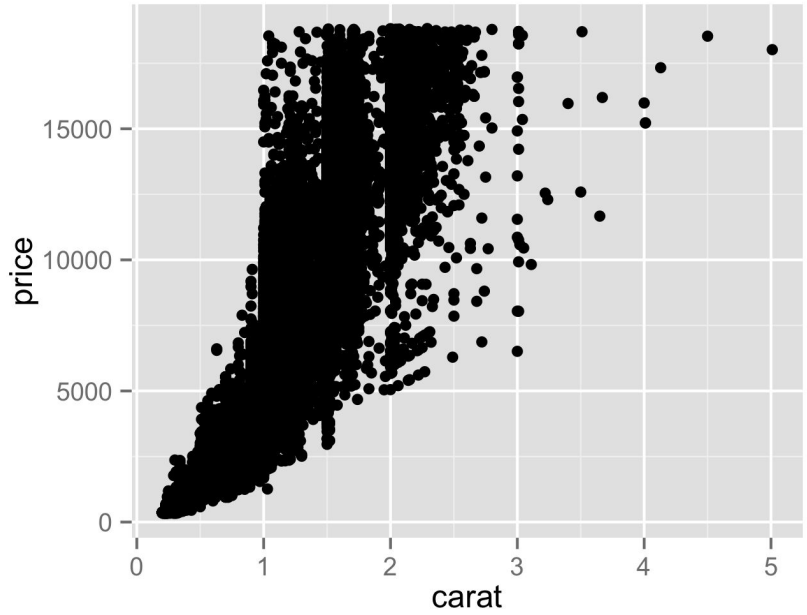




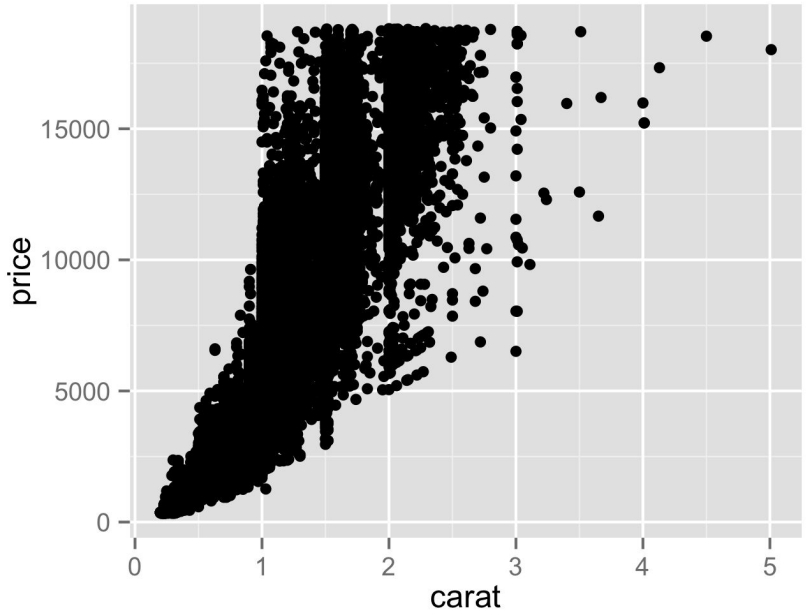
Coordinate Systems

cartesian (default)

```
g
```



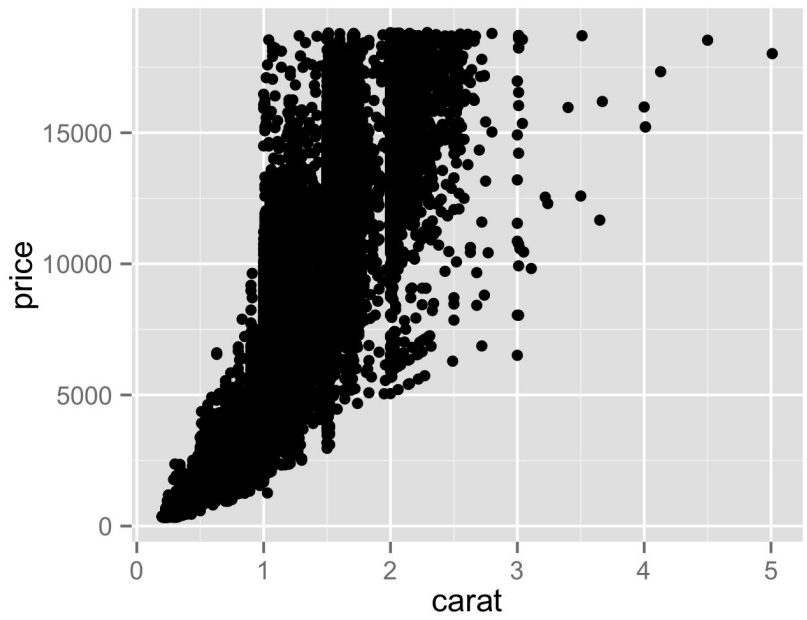

```
g + coord_cartesian()
```



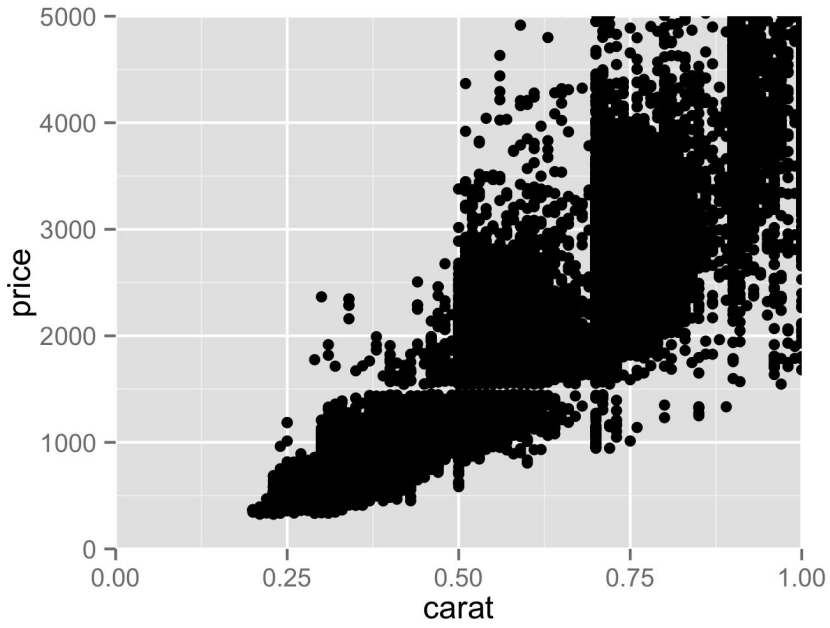
Coordinate Systems

cartesian (to zoom)

```
g
```

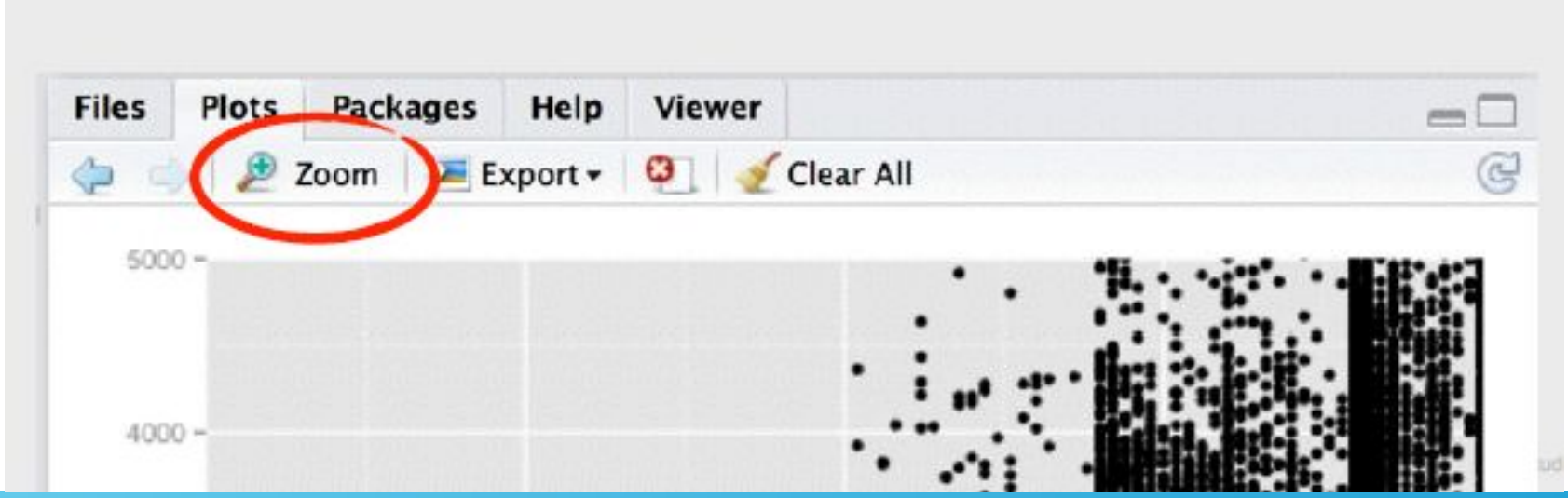



```
g + coord_cartesian(ylim =
c(0,5000), xlim = c(0, 1))
```



Coordinate Systems

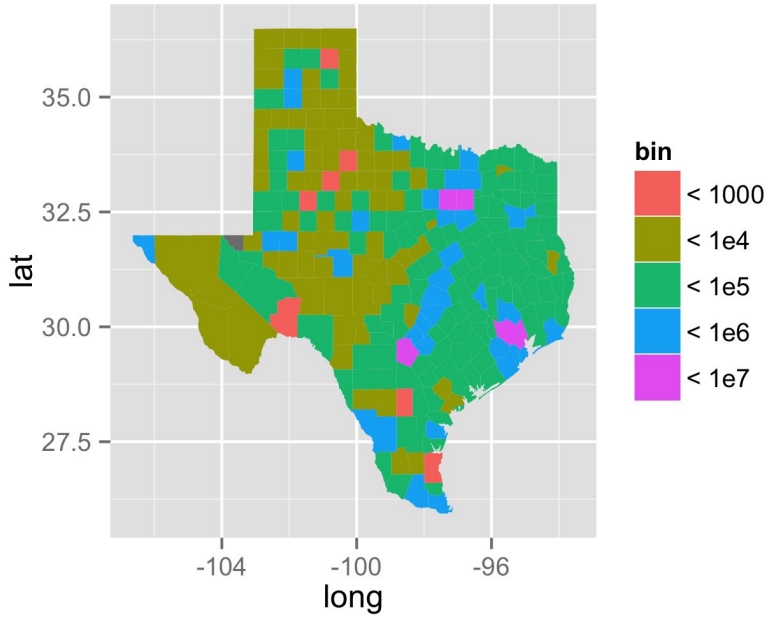
- There is also coord_map.
- Add it to tx and then open the plot in a new window by hitting zoom.
- Try rescaling the window. Can you tell what coord_map does?



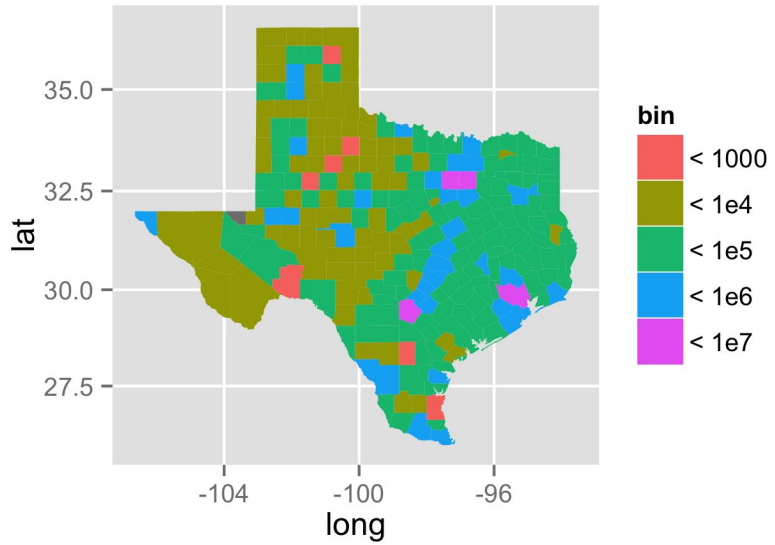
Coordinate Systems

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, a pie chart is a stacked bar graph in polar coordinates.

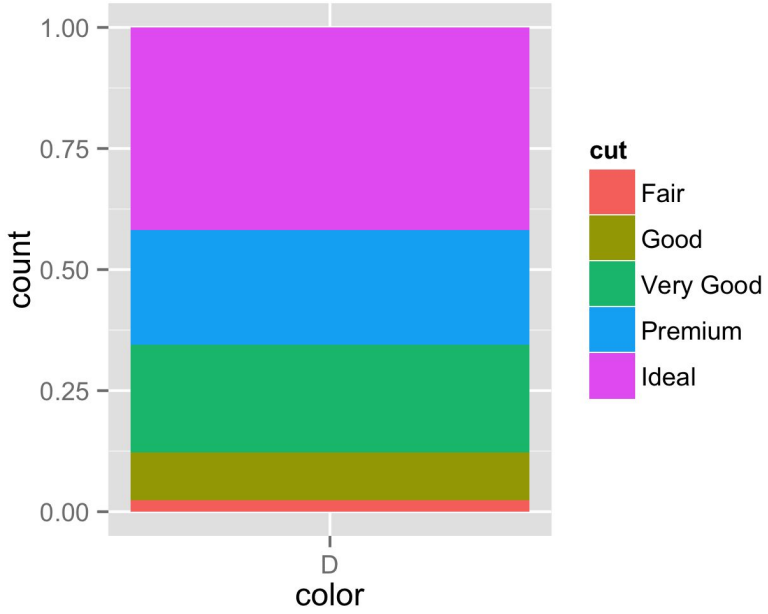
```
d2 <- subset(diamonds, color == "D")
cc <- ggplot(data = d2, aes(x = color)) + geom_bar(aes(fill = cut), position = "fill")
cc
```

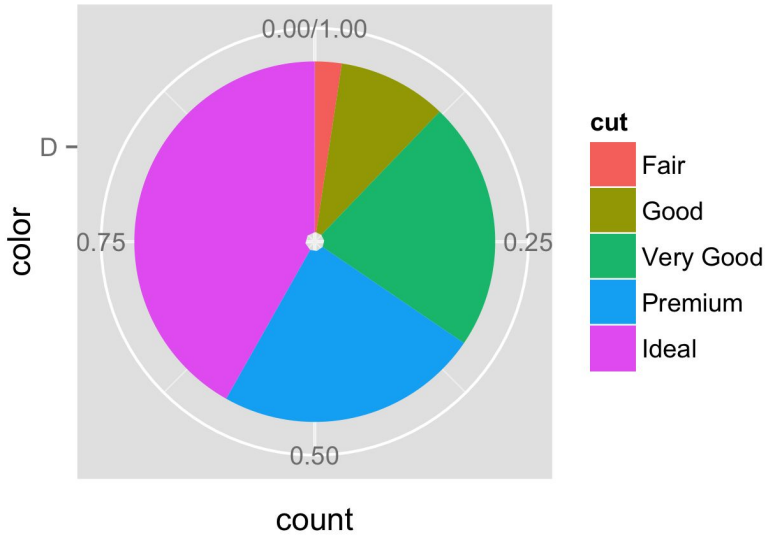
```
cc + coord_polar(theta = "y")
```

Coordinate Systems

Pie charts

- Aside: In the grammar of graphics, a pie chart is a stacked bar graph in polar coordinates.





Outline

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

Scales

Scales

Scale determines how data are mapped to an aesthetic.

```
scale_aesthetic_name()
```

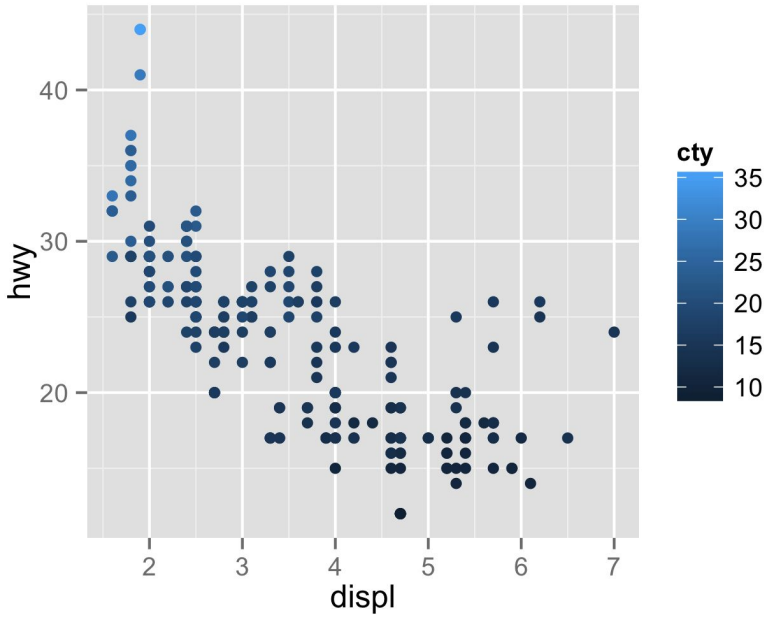
- scale_
  - Always begins with scale_...
- aesthetic_
  - ...the aesthetic to adjust...
- name
  - ...name of a scale object...
- ()
  - ...open and closed parentheses.

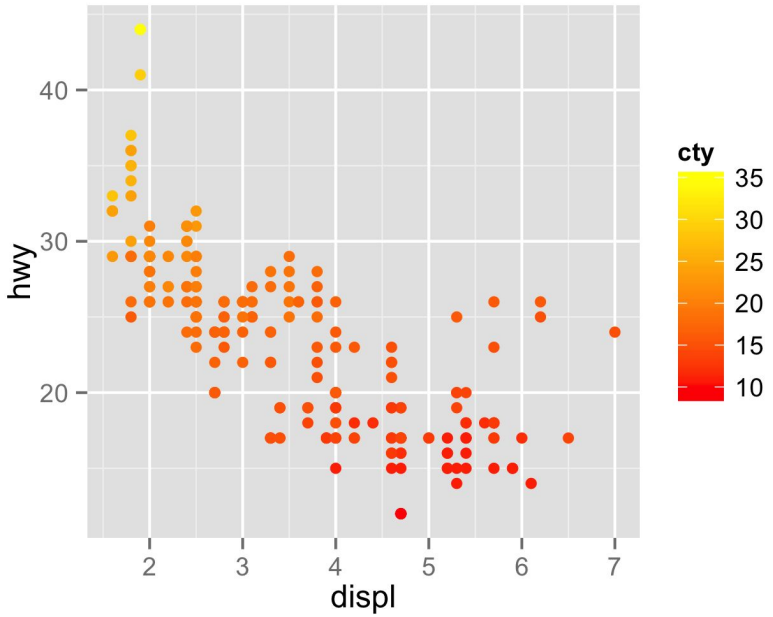
Scales

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")
```



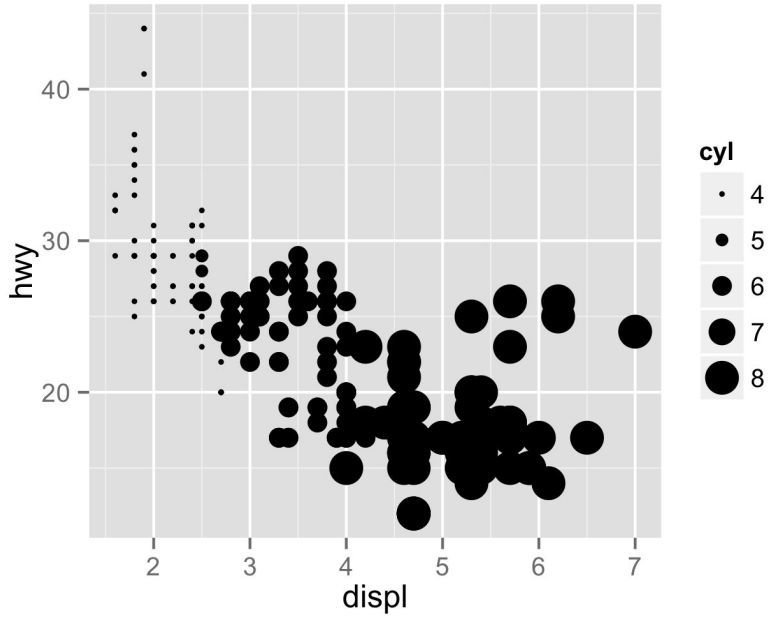


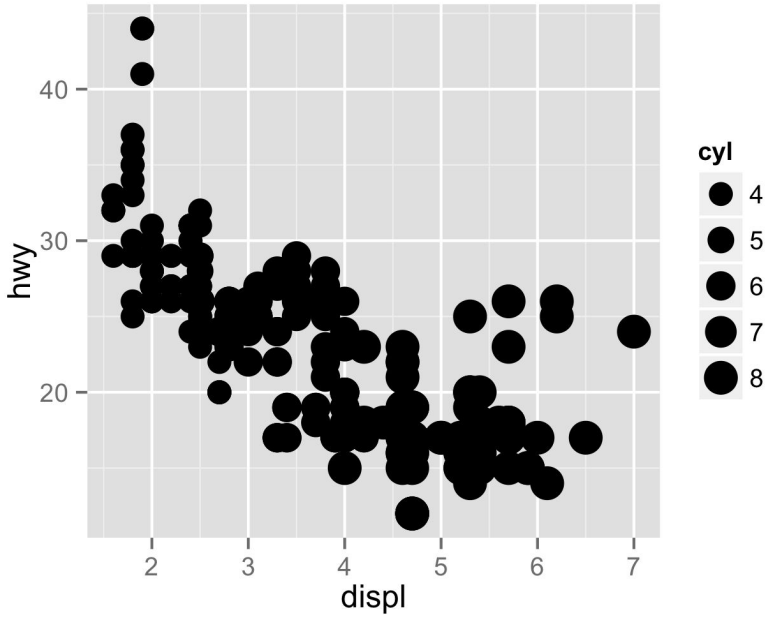
Scales

scale_size_area

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

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



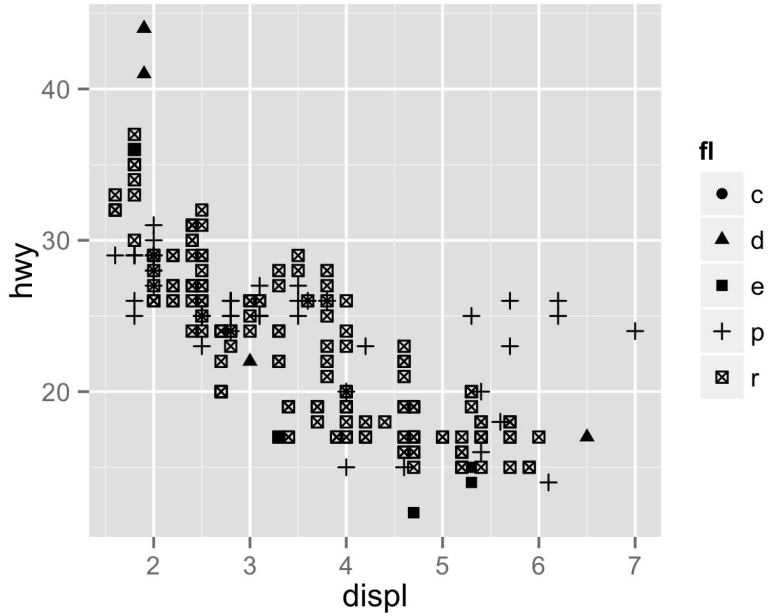


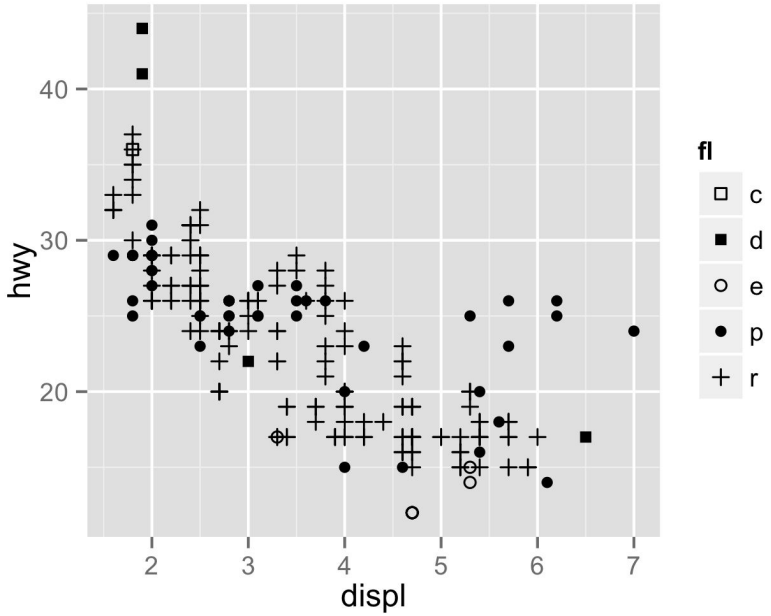
Scales

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))
```

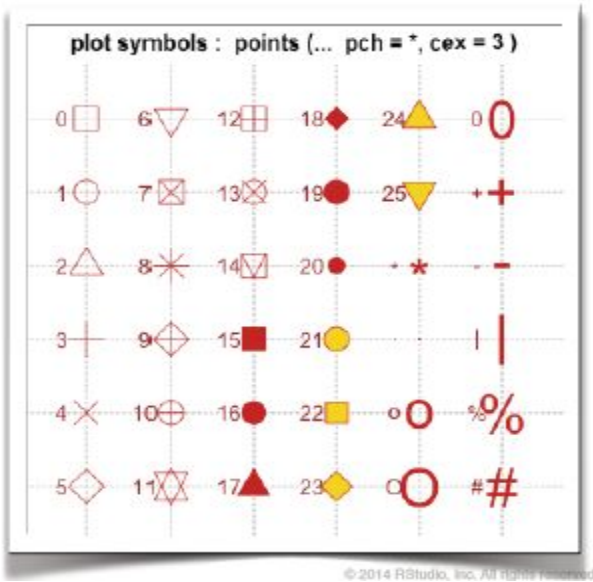




Scales

Aside: shapes.

```
r <- ggplot(data = mpg, aes(x = displ, y = cry)) +
geom_point(aes(color = drv, shape = fl))
#Specify the shapes manually
r + scale_shape_manual(values = c(0, 15, 1, 16, 3))
```

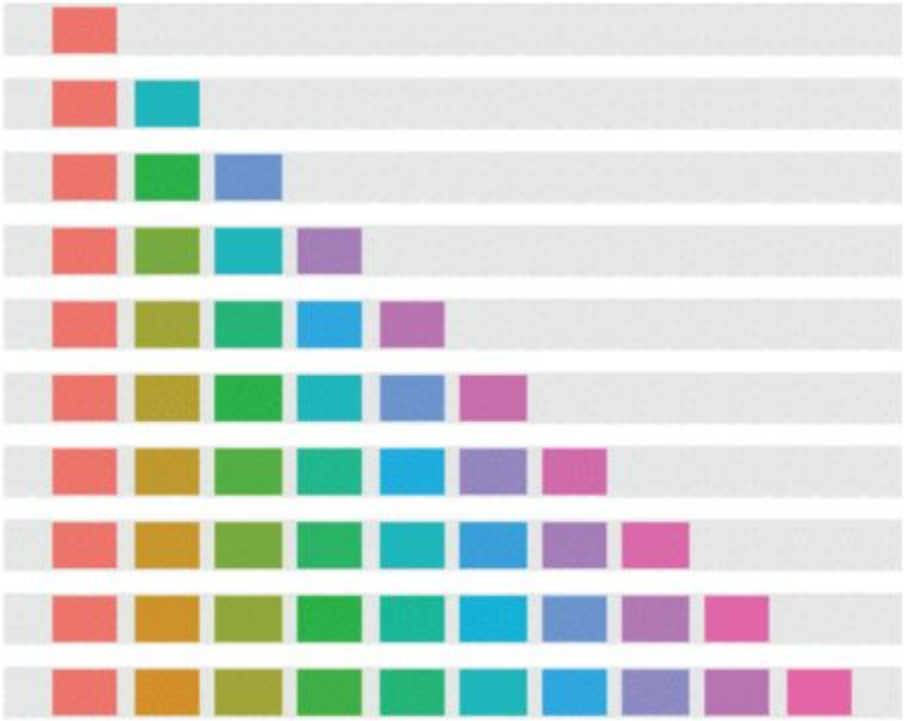


Color

- Color is the most popular aesthetic after position.
- It is also the easiest to misuse.

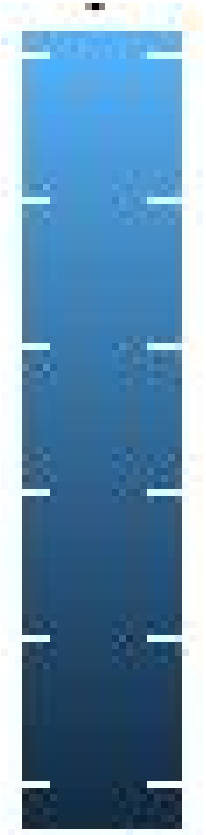
Color

- ggplot2's default discrete palettes
- (depends on how many colors are needed)



Default Continuous Palette

But what if you want something different?

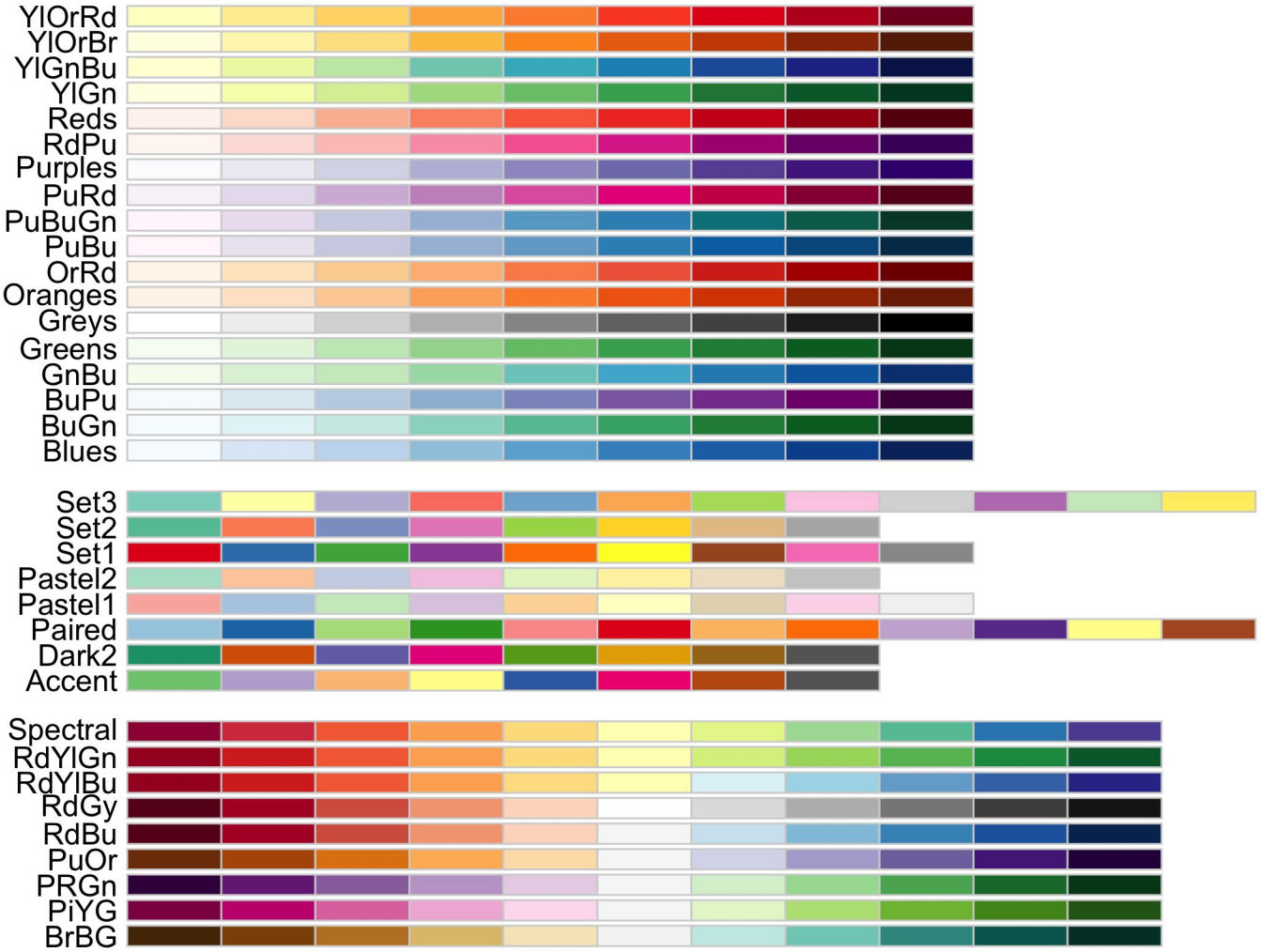


Color Brewer

- 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.

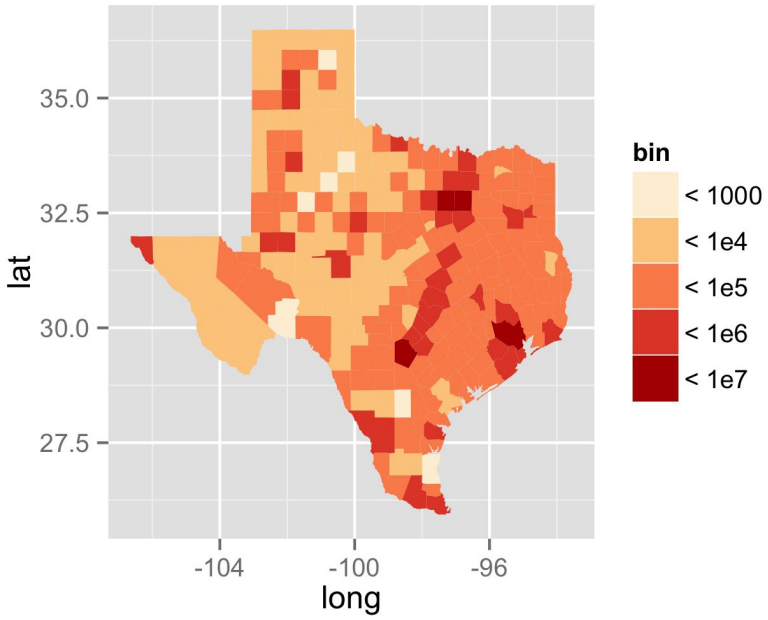
```
#install.packages("RColorBrewer")
library(RColorBrewer)
display.brewer.all()
```

Color Brewer

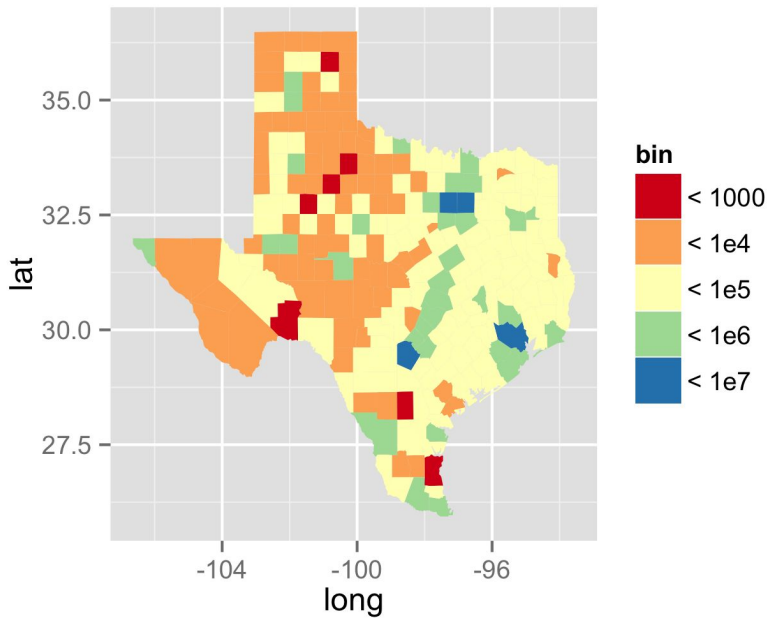


Brewer Scale

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

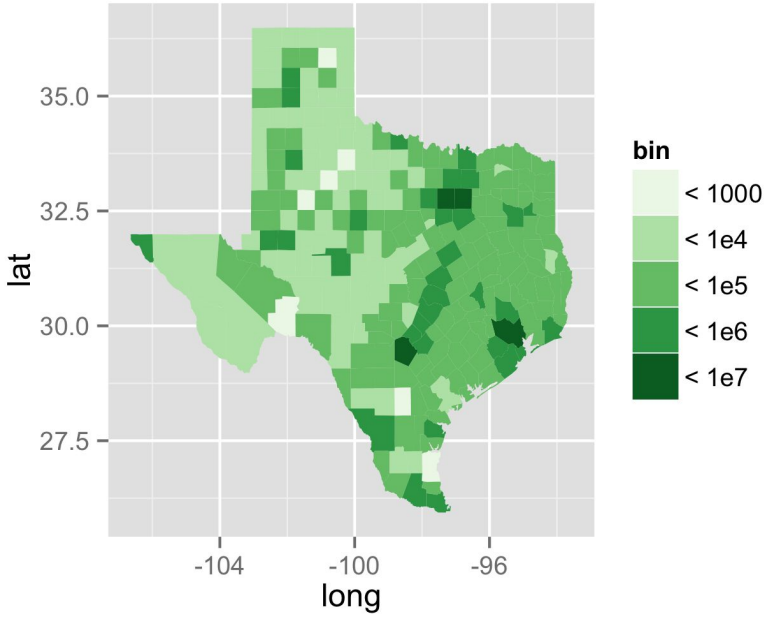



```
tx + scale_fill_brewer(
palette = "Spectral")
```

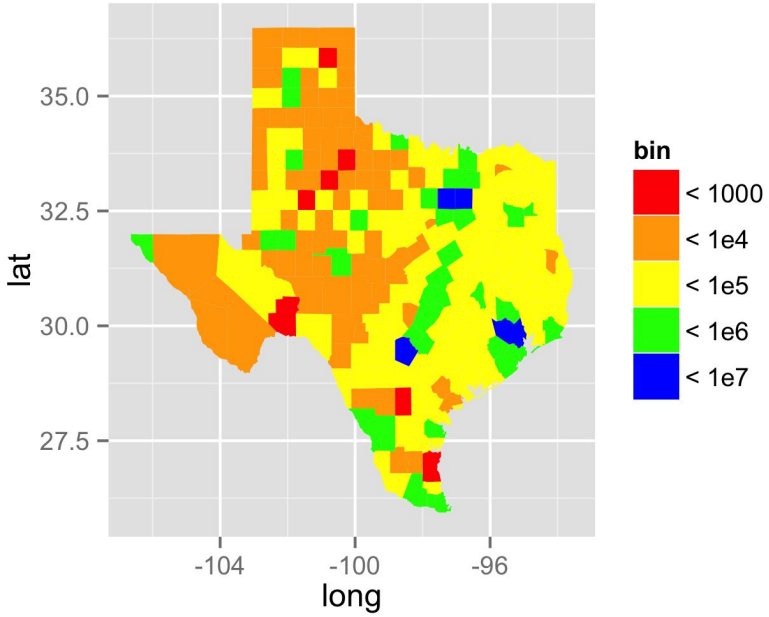


Custom != Pretty

```
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.






Other Aesthetics

- You can manage the relationship used for each aesthetic mapping in your plot the same way:
  - select a scale that applies to that mapping (using the three part scale name convention)
  - set additional arguments in the scale function if desired.

Outline

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

Themes

Themes

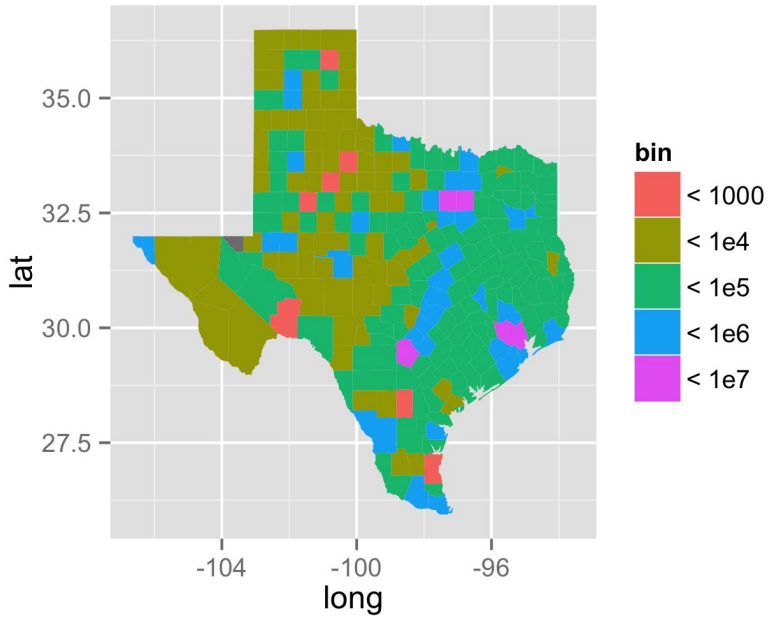
Visual appearance

- The theme controls the appearance of the non-geom parts of the plot.
- In other words, the theme determines how the plot looks.

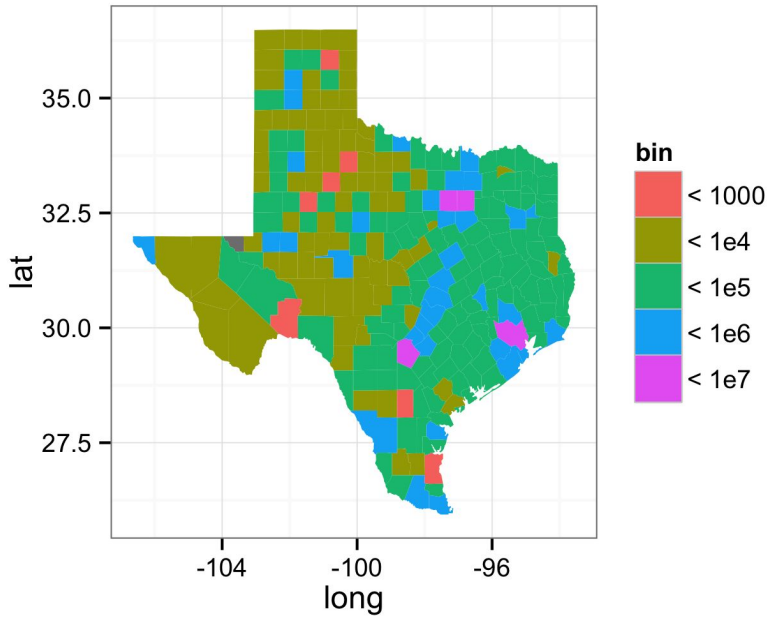
Themes

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

```
tx + theme_grey()
```




```
tx + theme_bw()
```

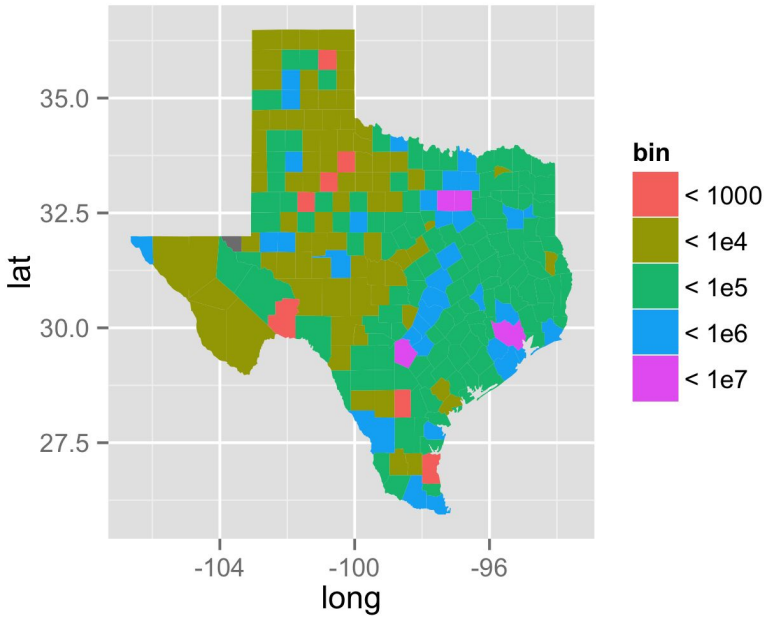


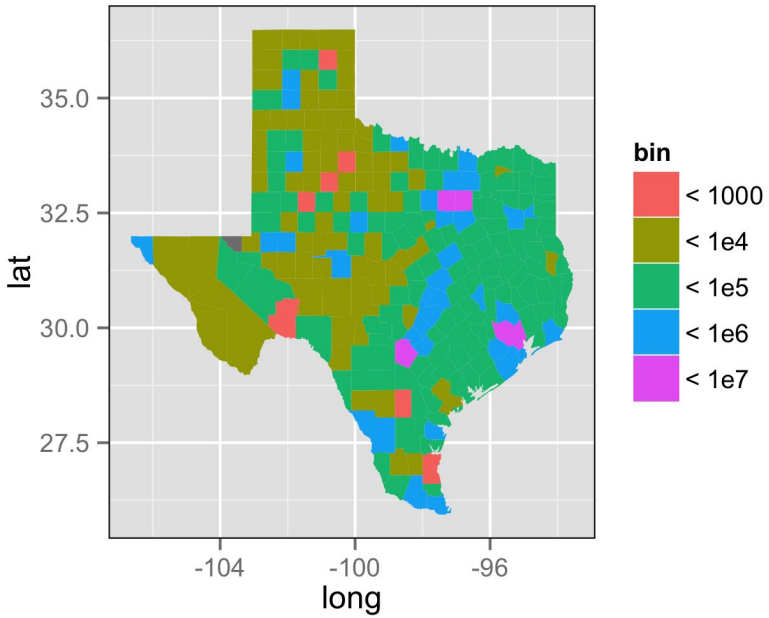
Themes

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

```
tx
```

```
tx + theme(panel.border =
element_rect(colour = "black",
fill = NA))
```





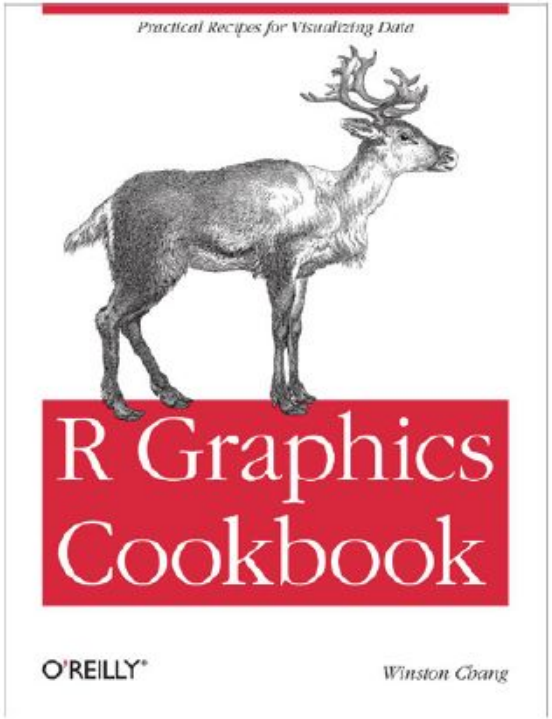
Themes

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#



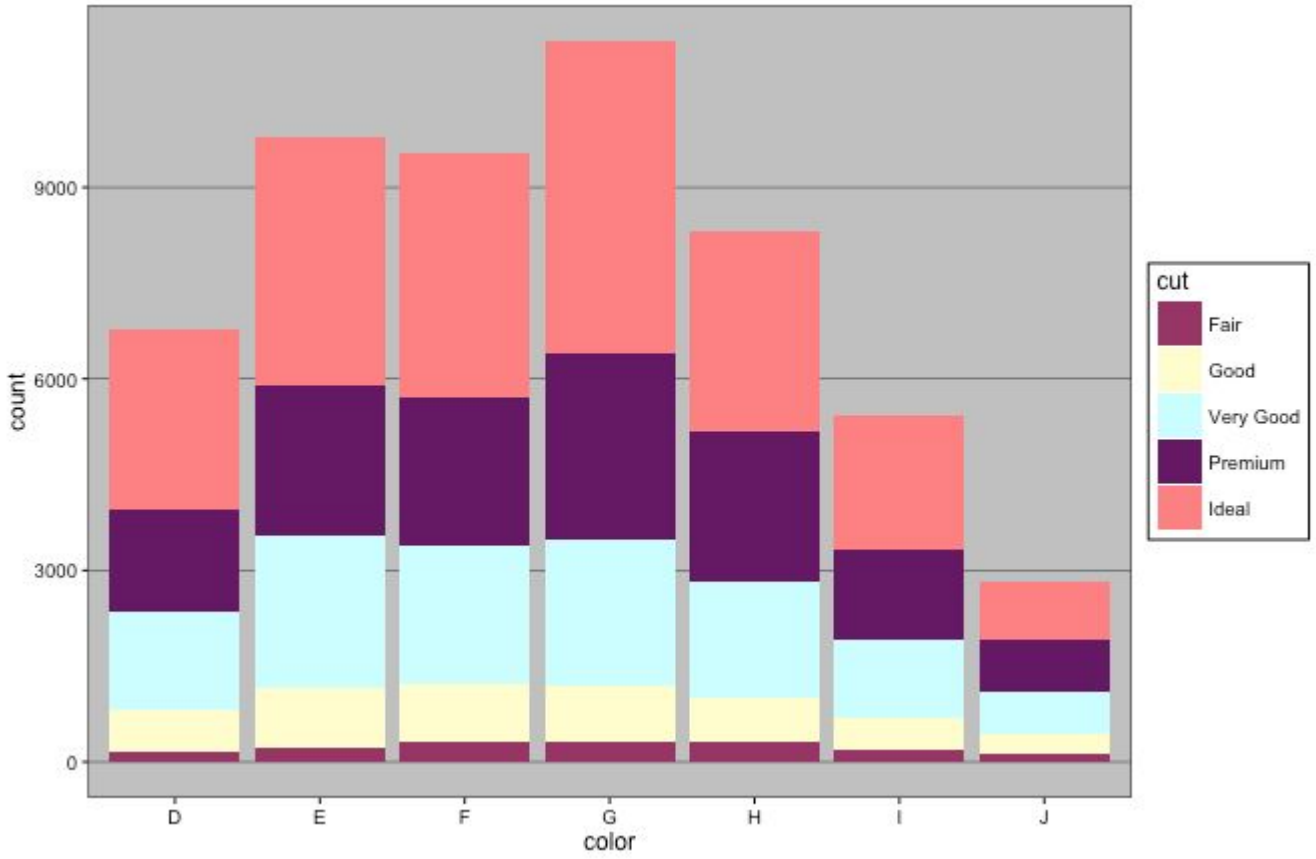
Themes

https://github.com/jrnold/ggthemes

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

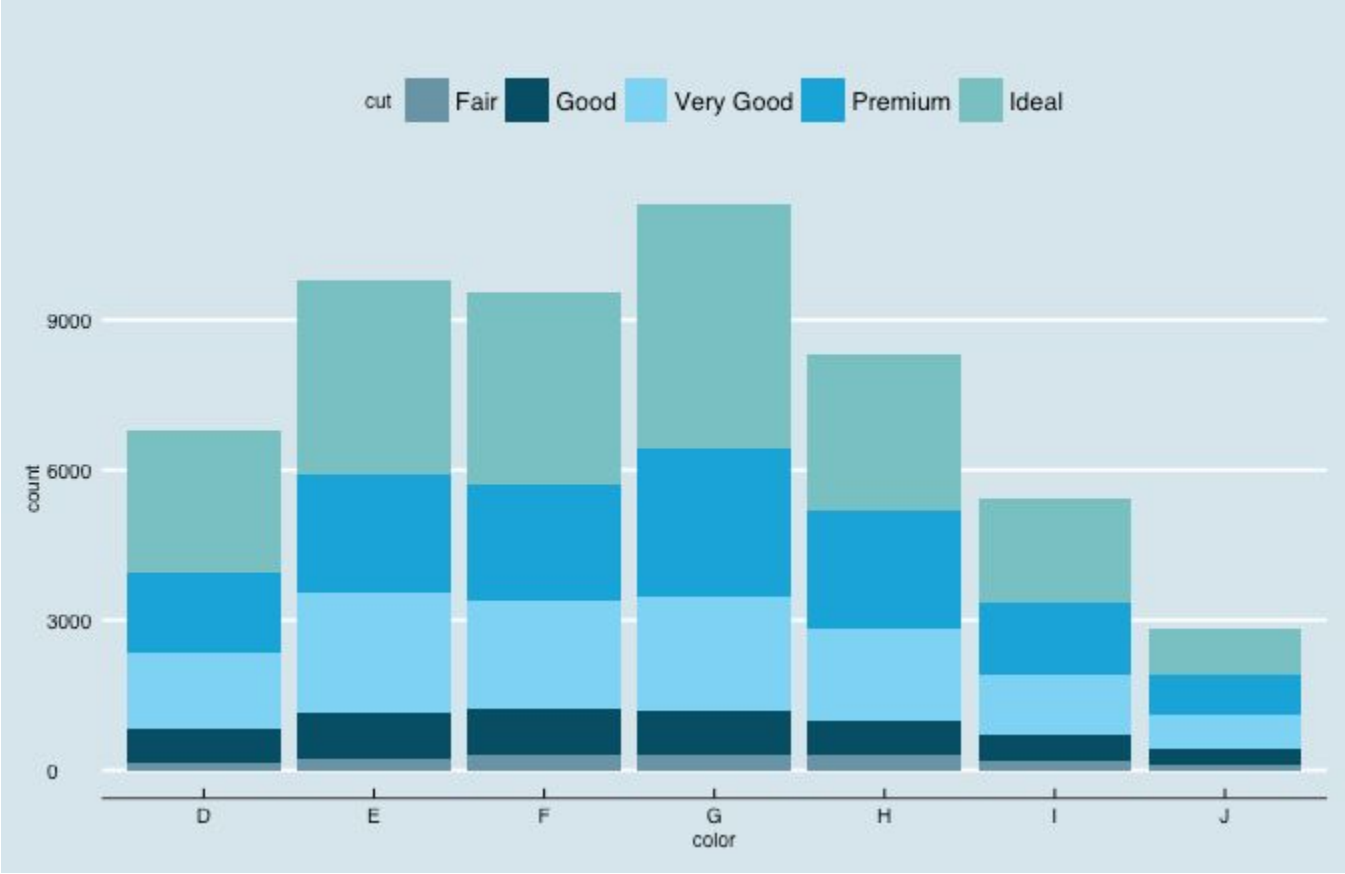
theme_excel()

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



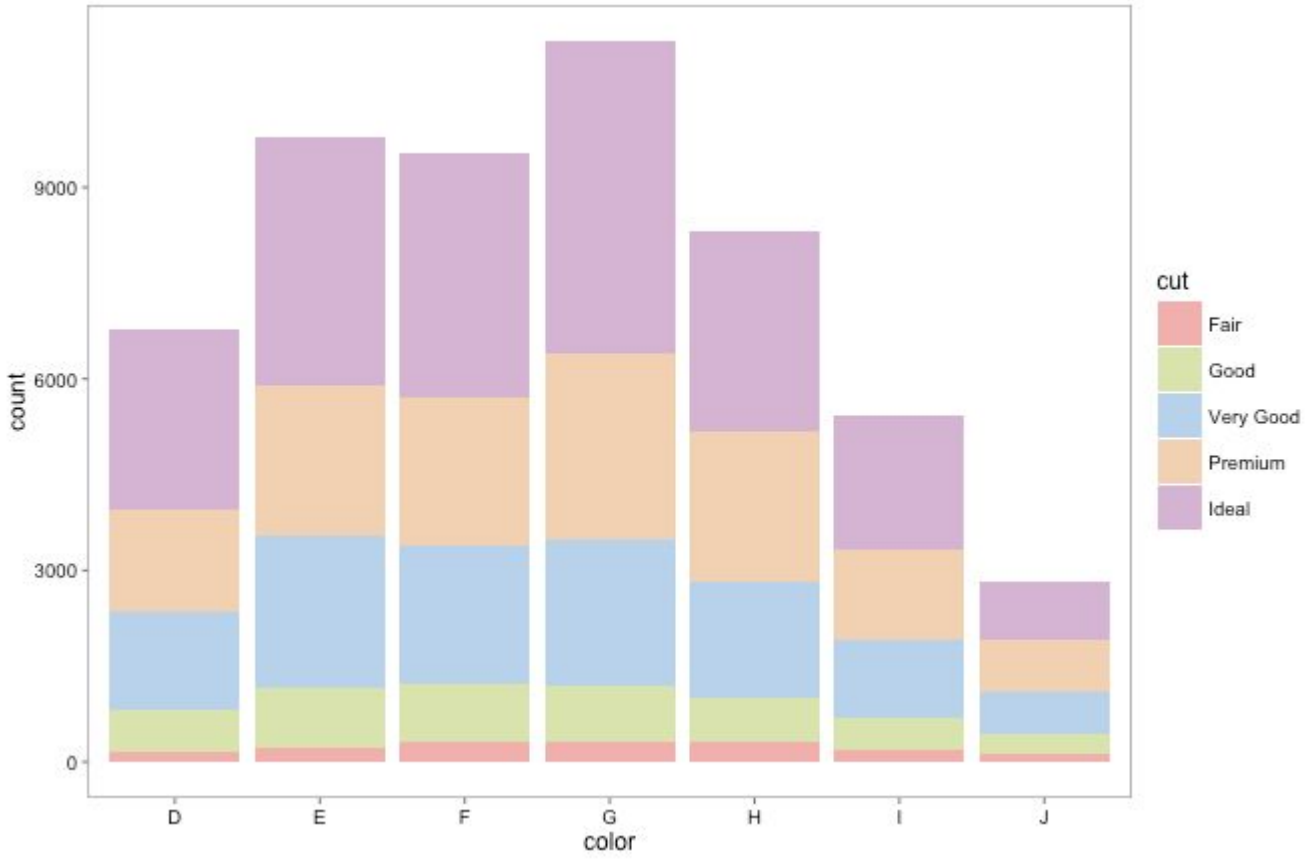
theme_economist()

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



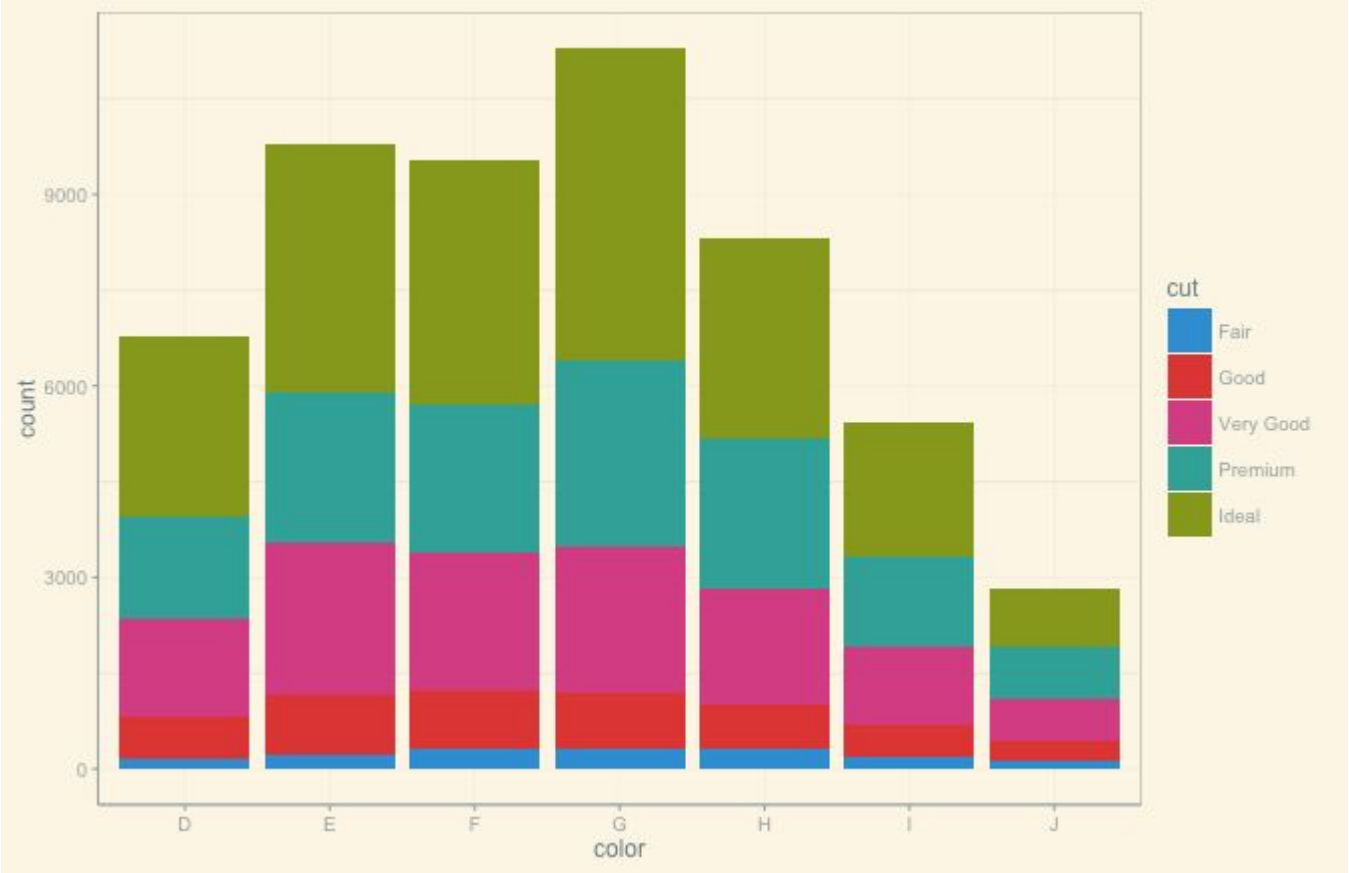
theme_few()

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



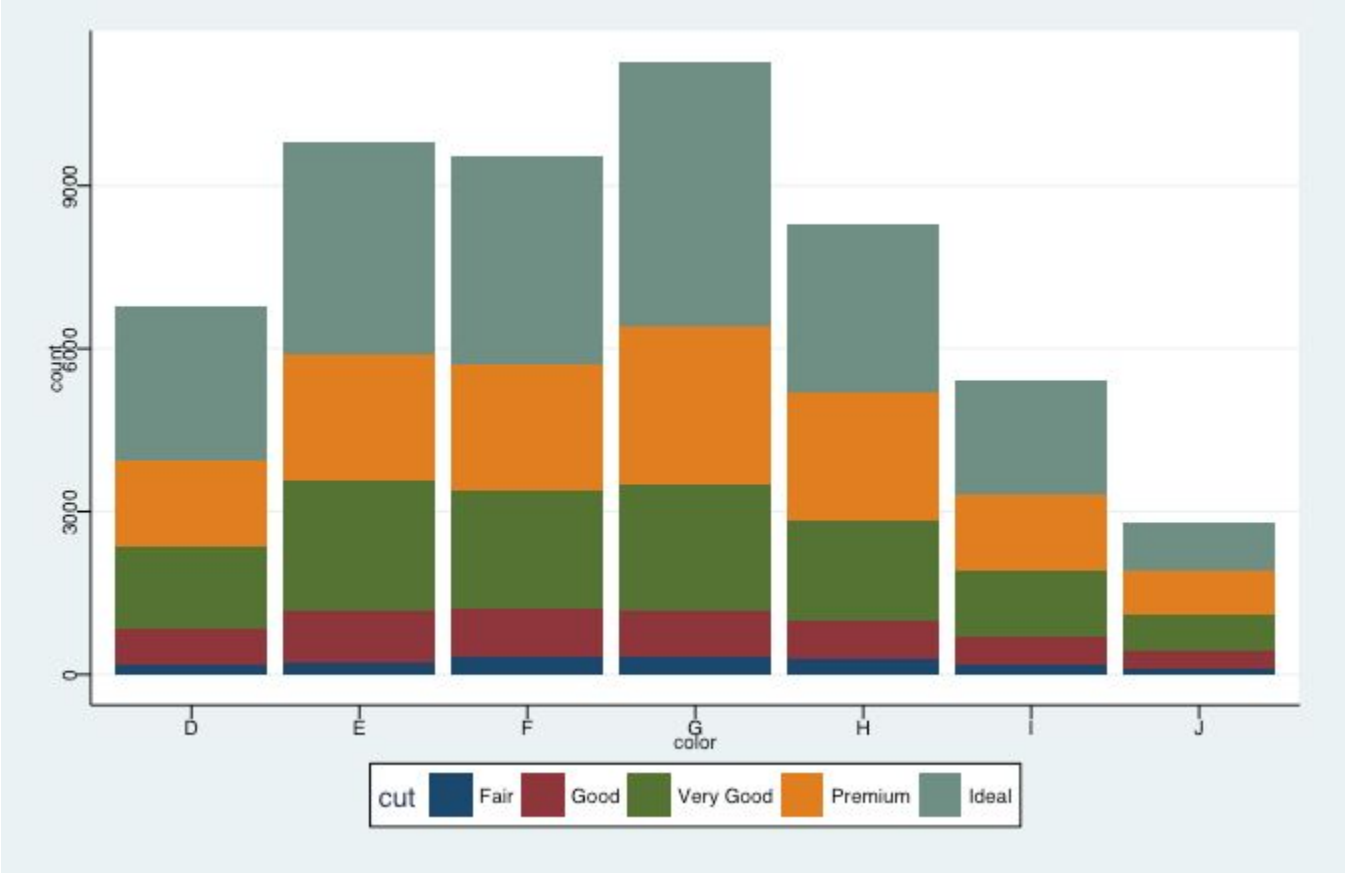
theme_solarized()

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



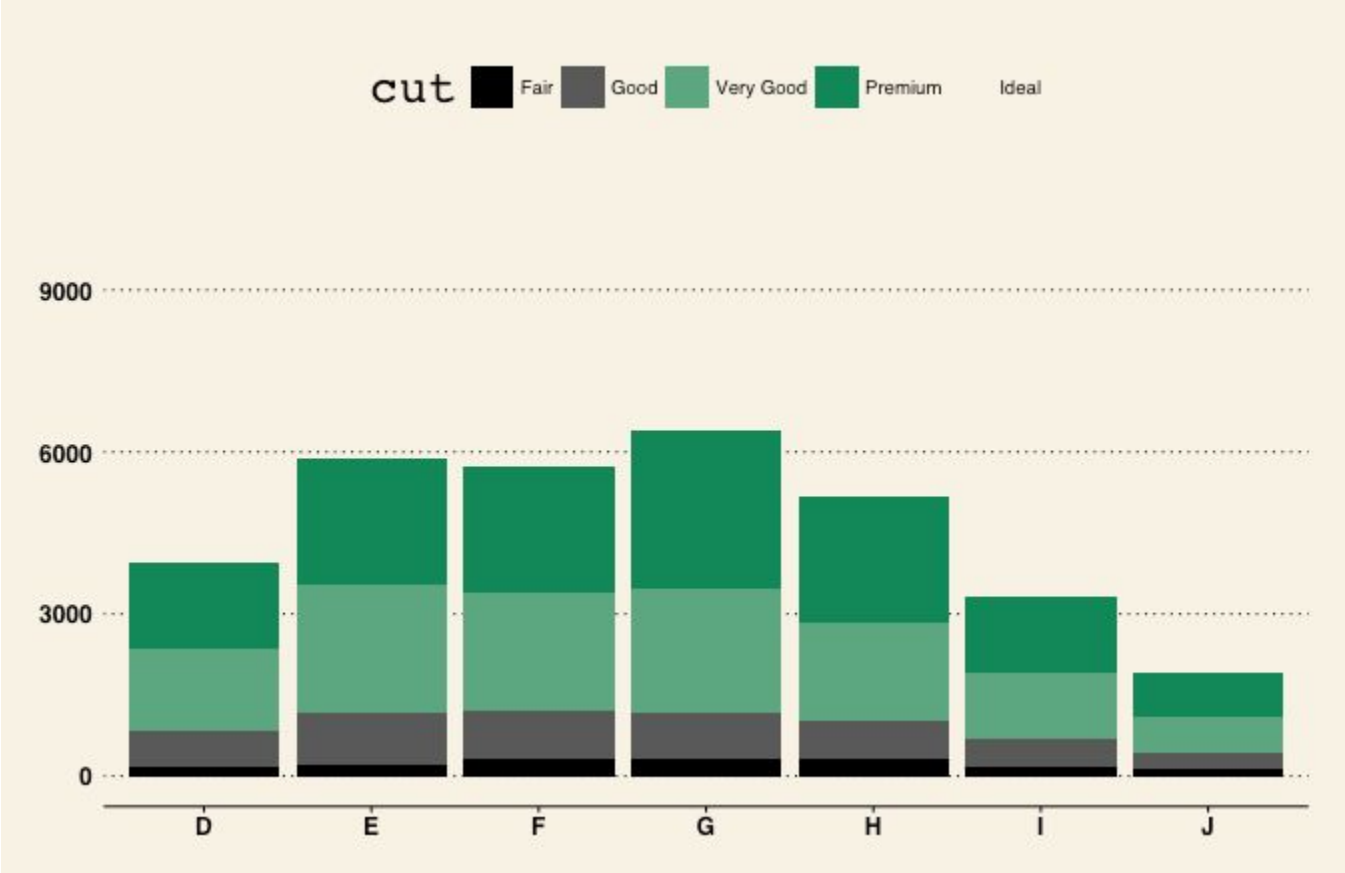
theme_stata()

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



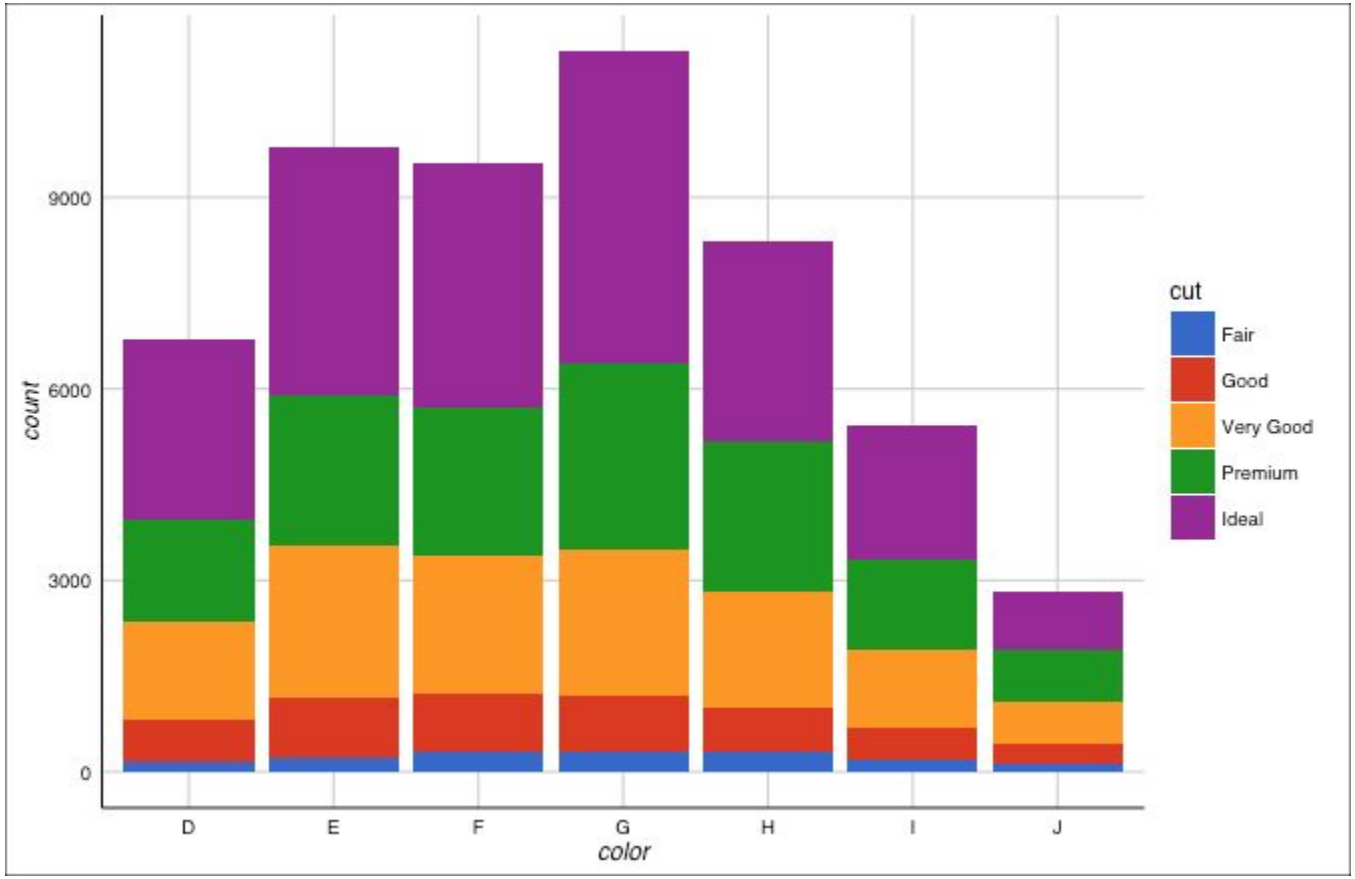
theme_wsj()

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



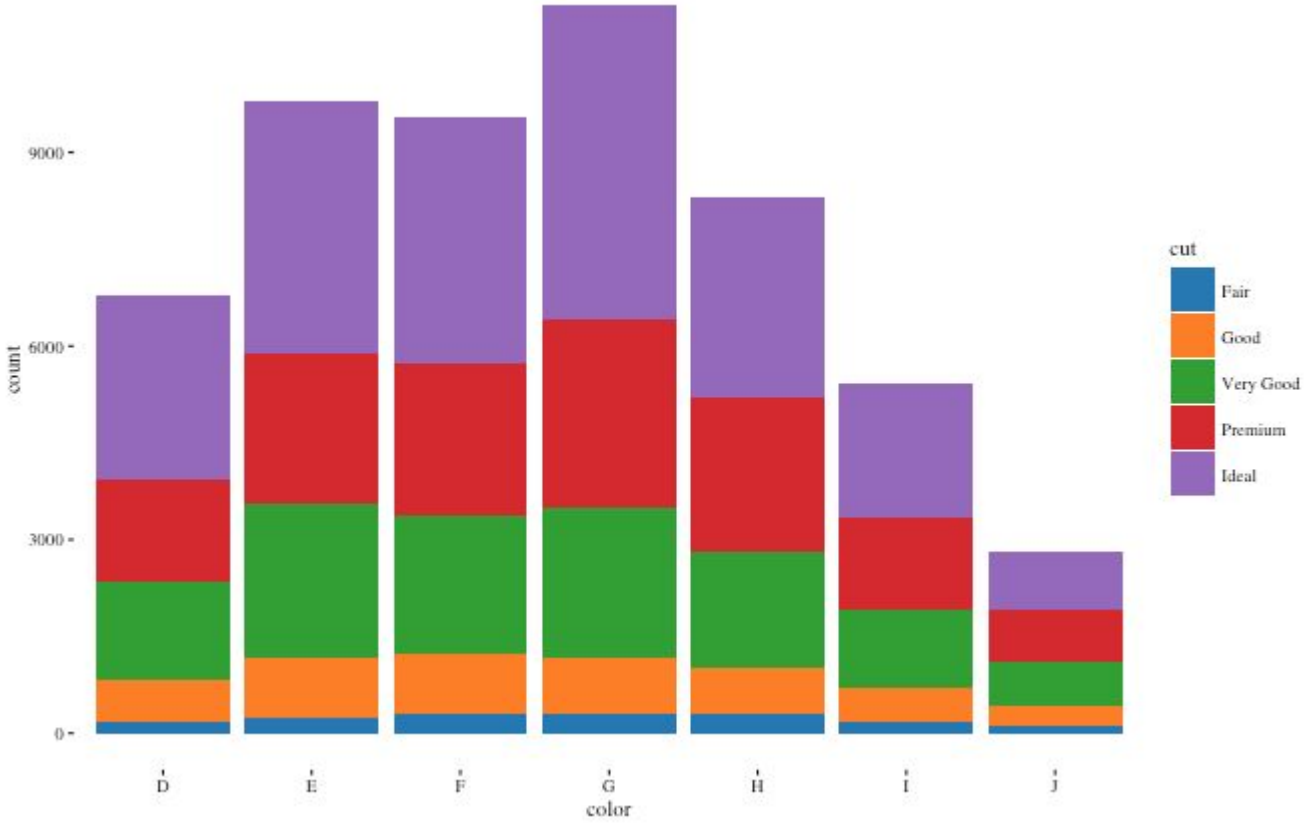
theme_gdocs()

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



theme_tufte()

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



Outline

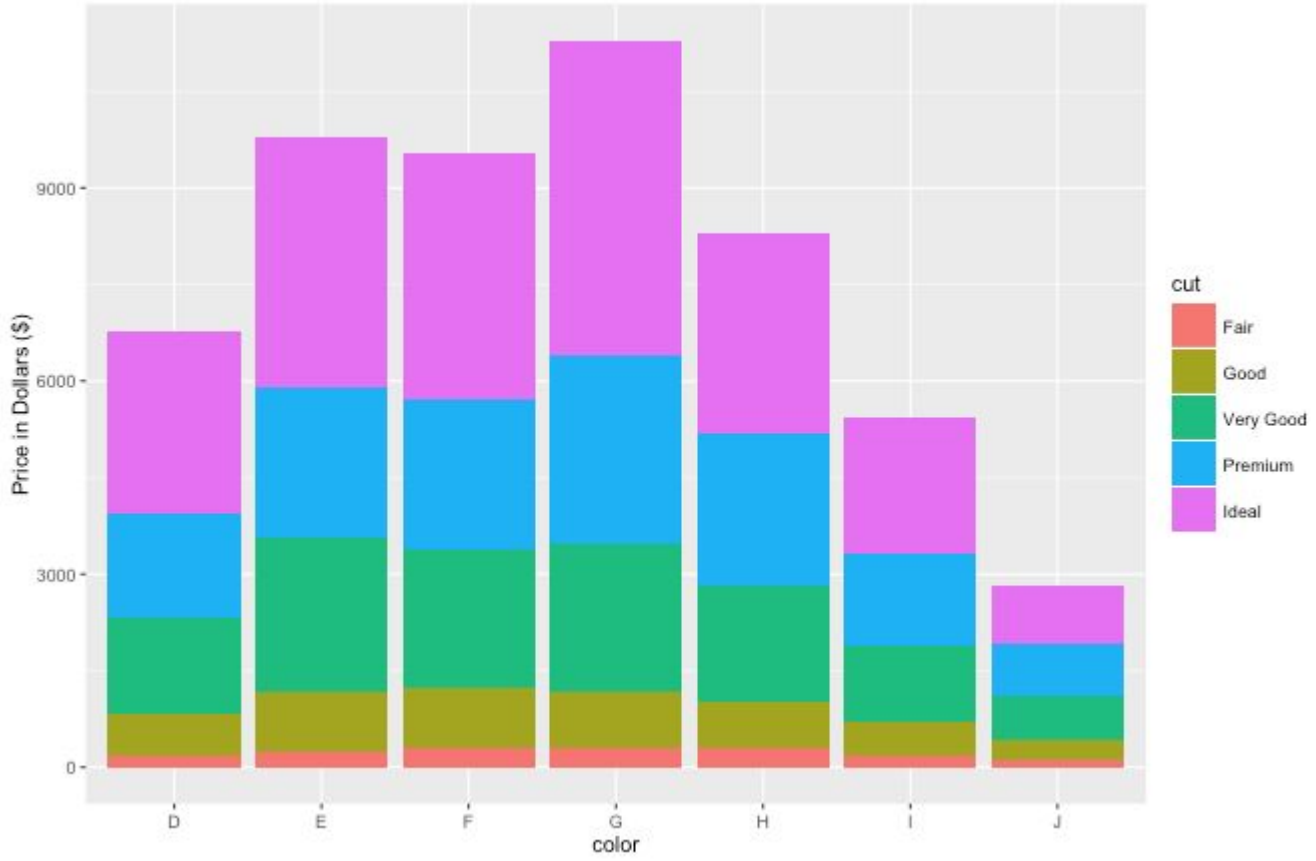
- Customizing Graphics
- Titles
- Coordinate systems
- Scales
- Themes
- **Axis labels**
- Legends
- Other visualizations

Axis Labels

Axis Labels

Modify axis labels with xlab and ylab

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



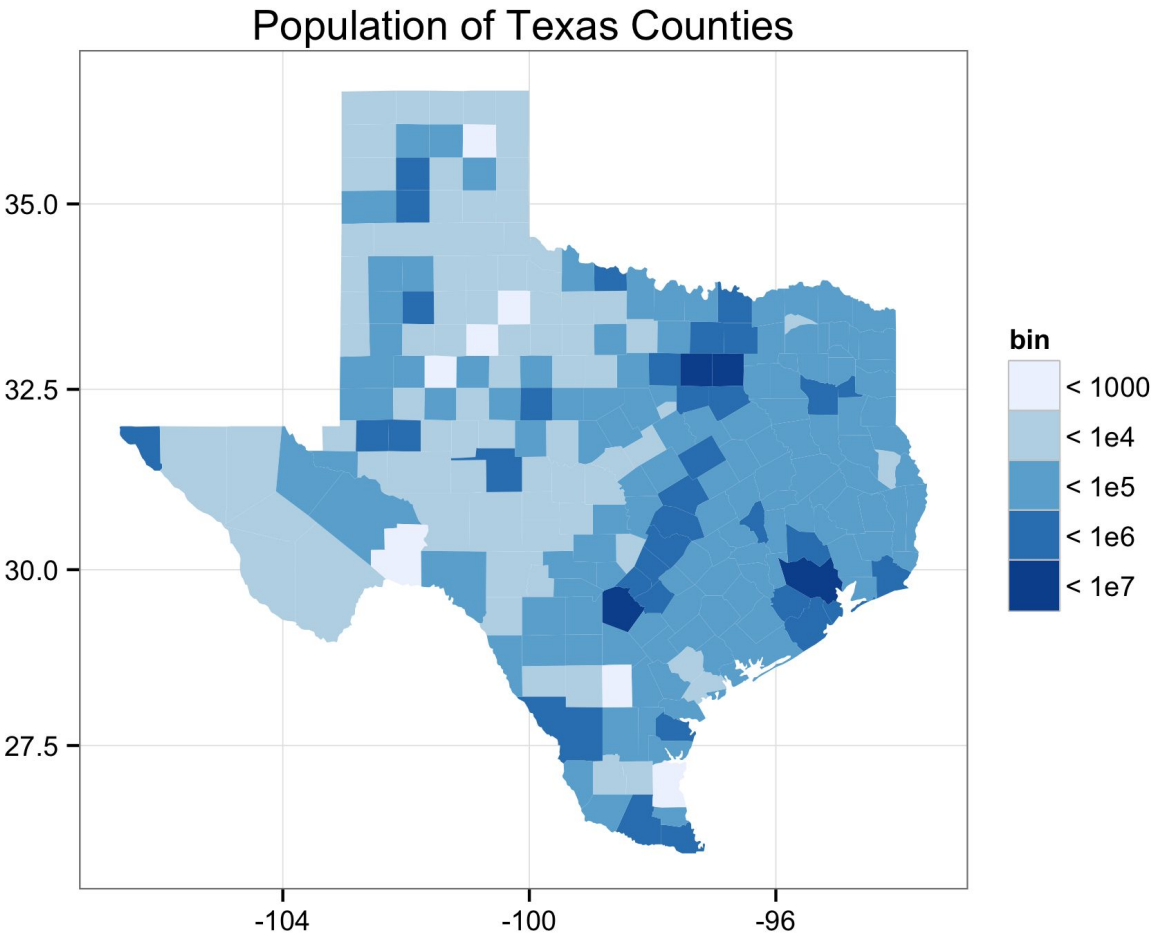
Your Turn

- Practice what you have learned so far on tx
- remove the long and lat axis labels
- add a white background
- add a title (ggtitle())
- add a brewer color scale
- an appropriate coordinate system

Answer

```
tx + scale_fill_brewer(palette = "Blues") +
    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
bin
< 1000
< 1e4
< 1e5
< 1e6
< 1e7

Outline

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

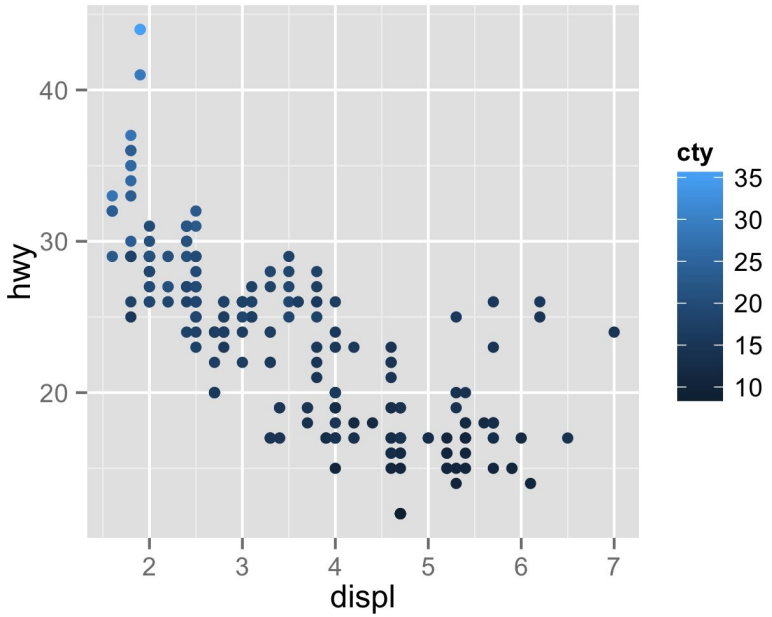
Legends

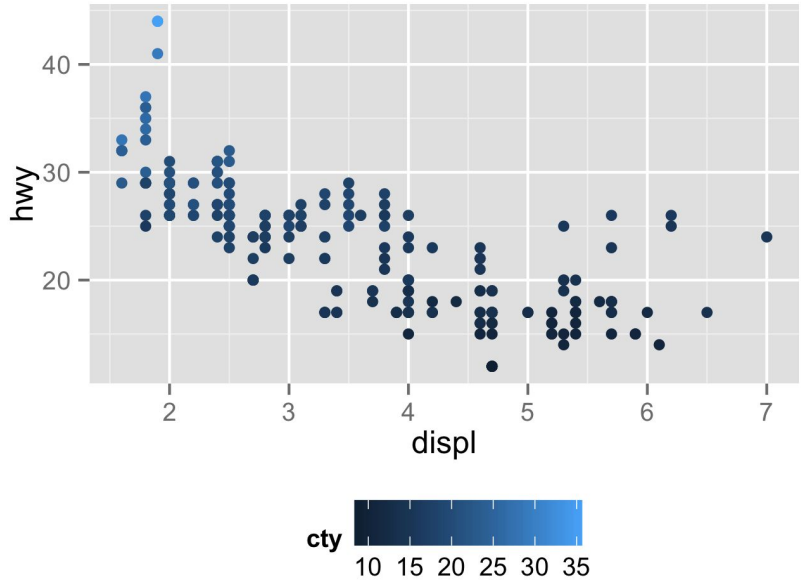
Legends

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
  - aesthetic
- "bottom"
  - one of “bottom”, “top”, or “left”, or “right”

Legends

Choose the type of legend with guide.

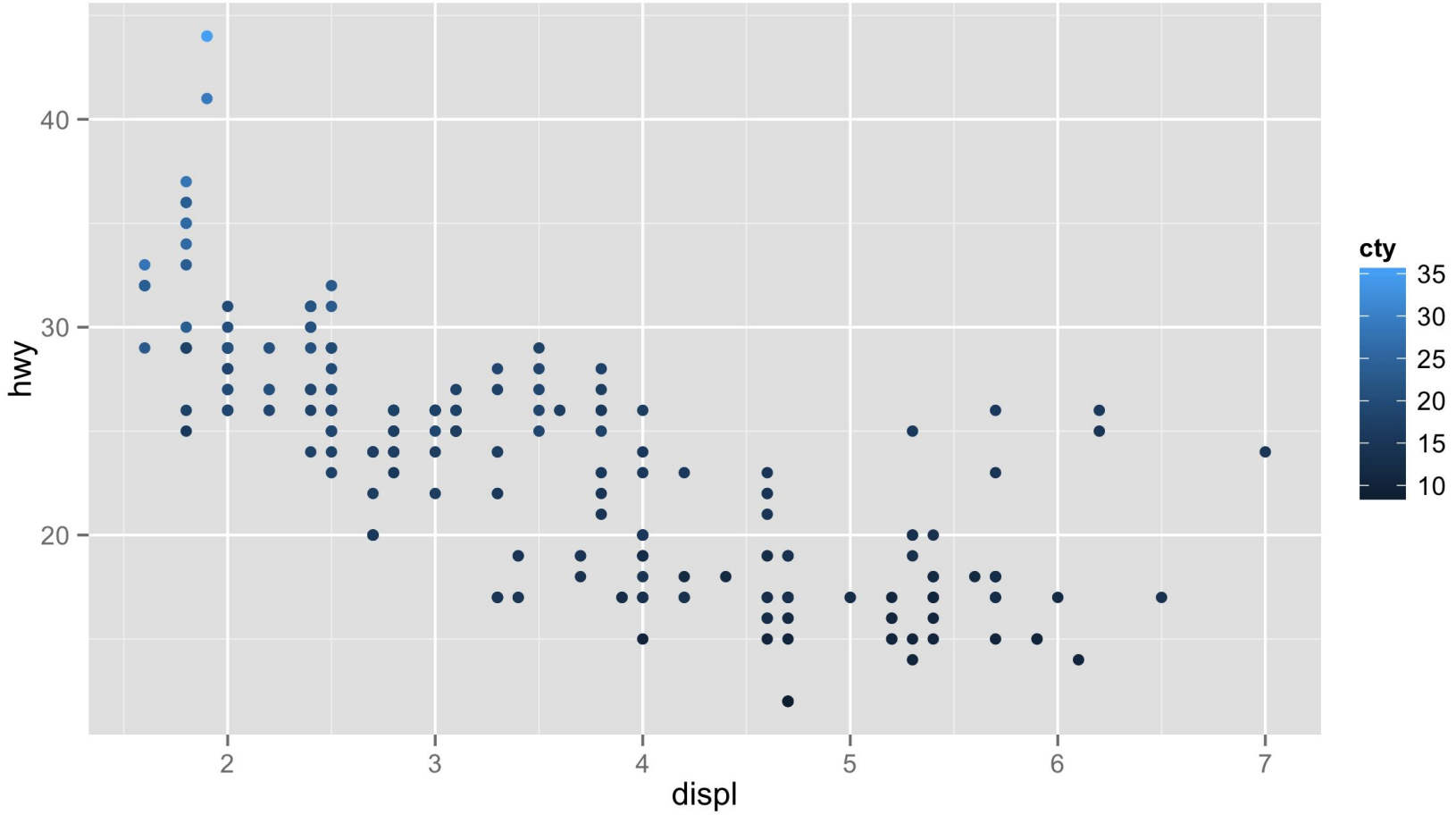
```
#Continuous.
q + guides(color = "colorbar")

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

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

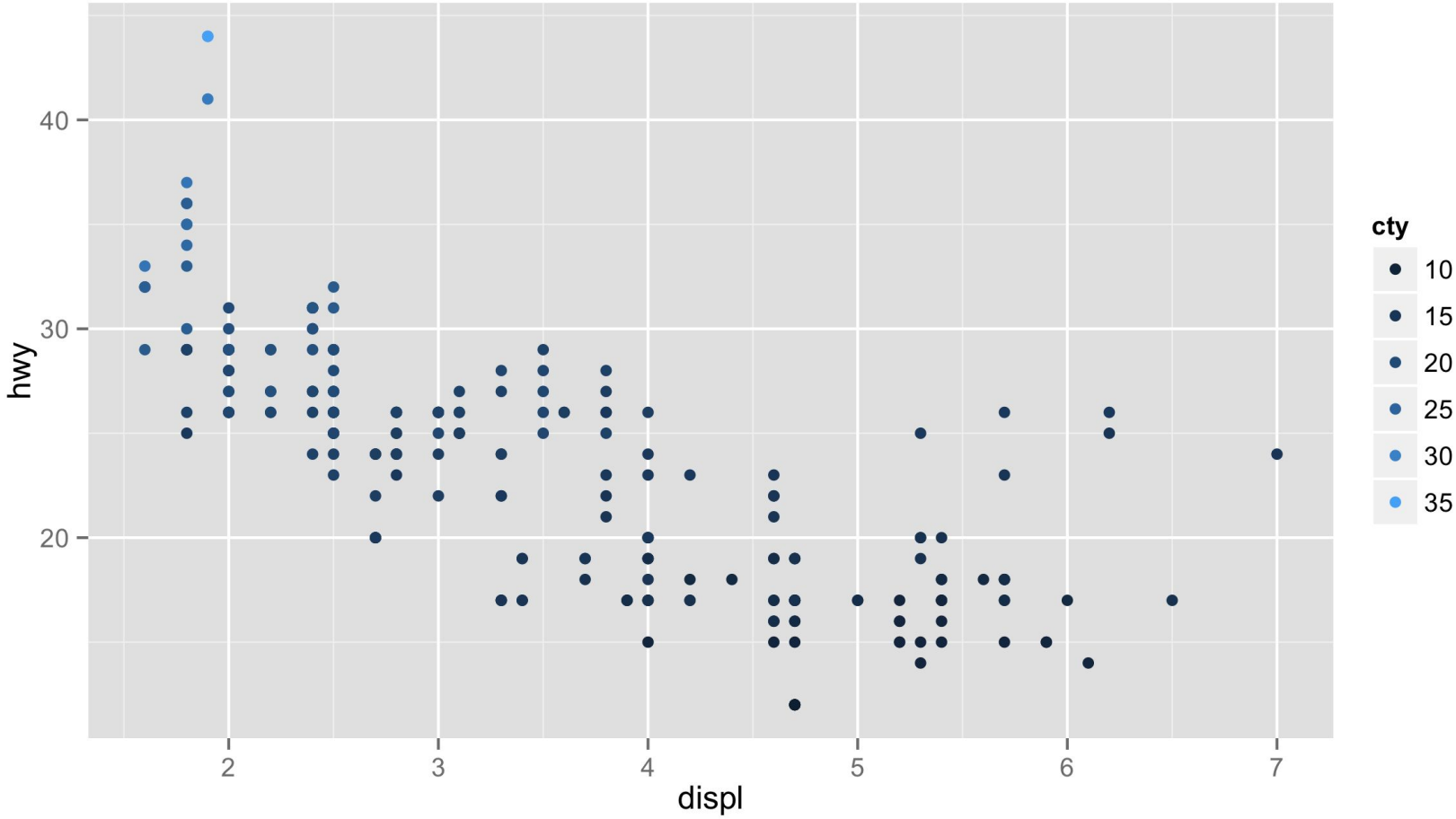
Legends

guide: "colorbar"



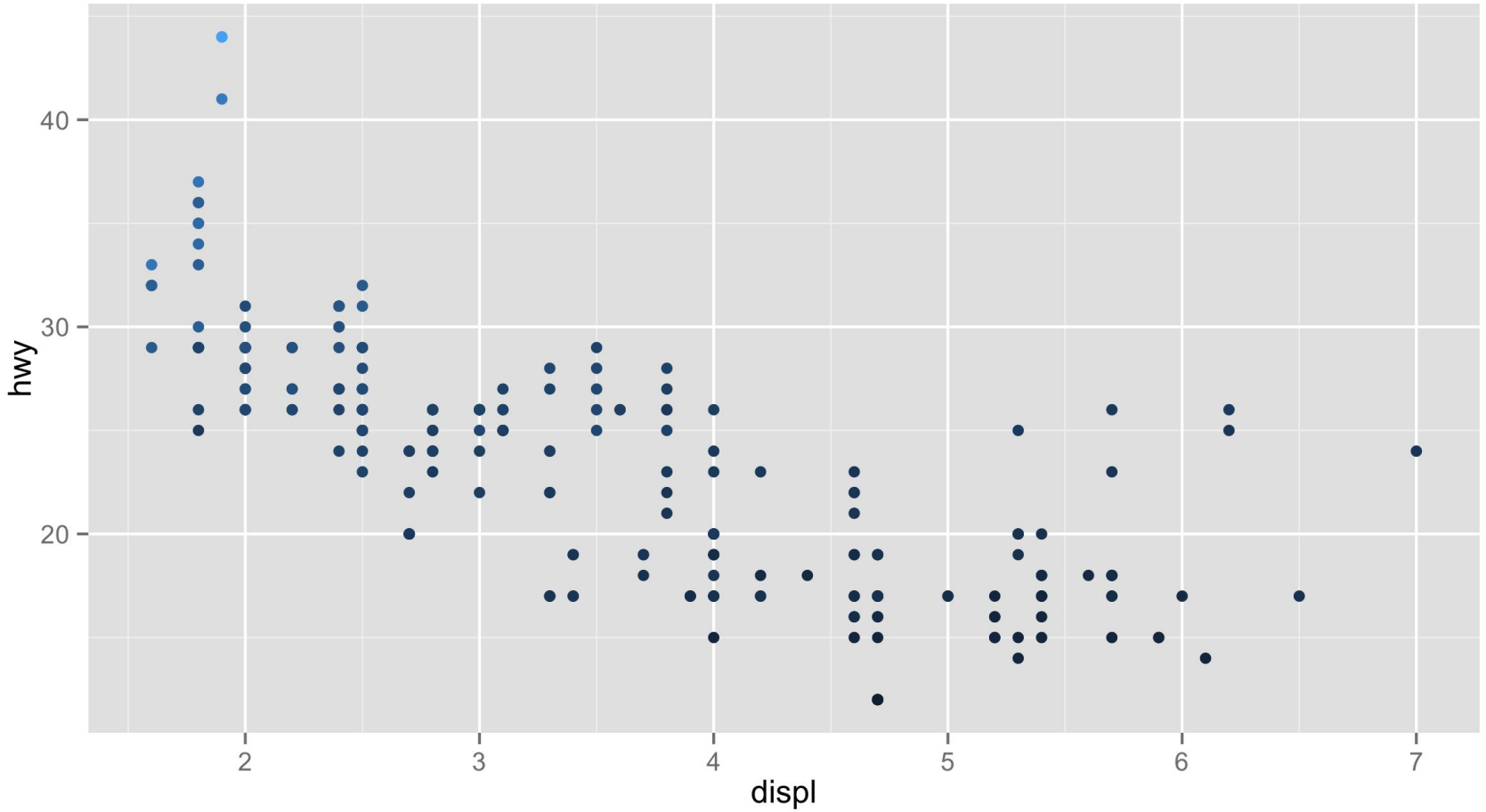
Legends

guide: "legend"



Legends

guide: "none"



Legends

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

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

Labels

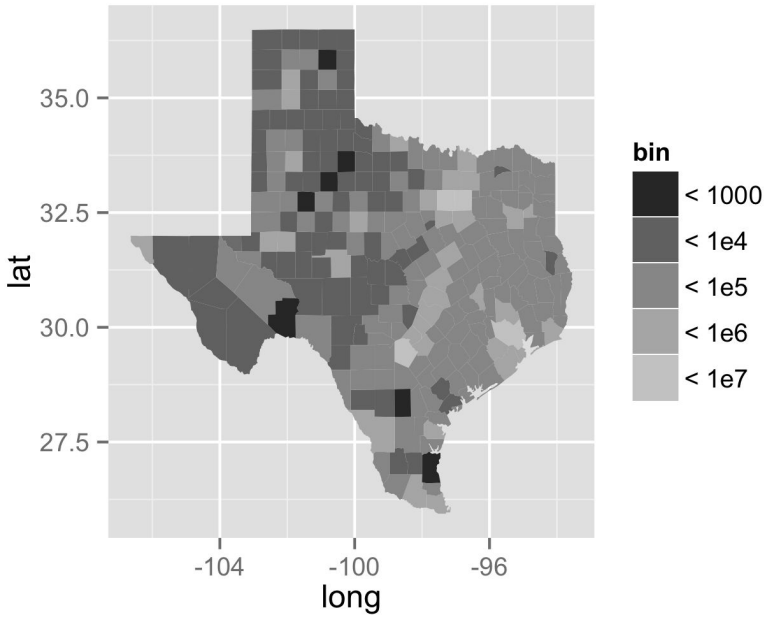
Every scale uses the following arguments

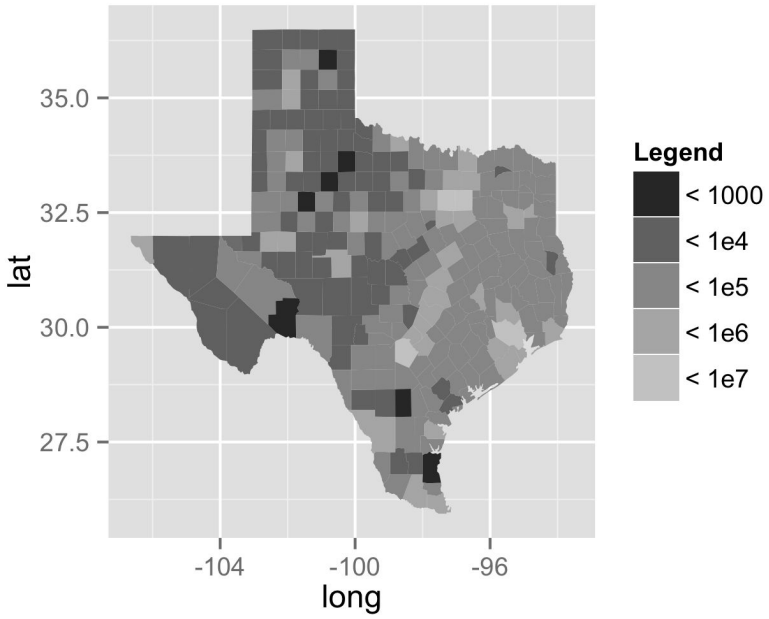
argument	controls
name	Title of legend (or axis label)
labels	Labels inside legend (or tick labels on axis) * must be a vector with one element for each label or tick mark

Labels

```
tx + scale_fill_grey()
```

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

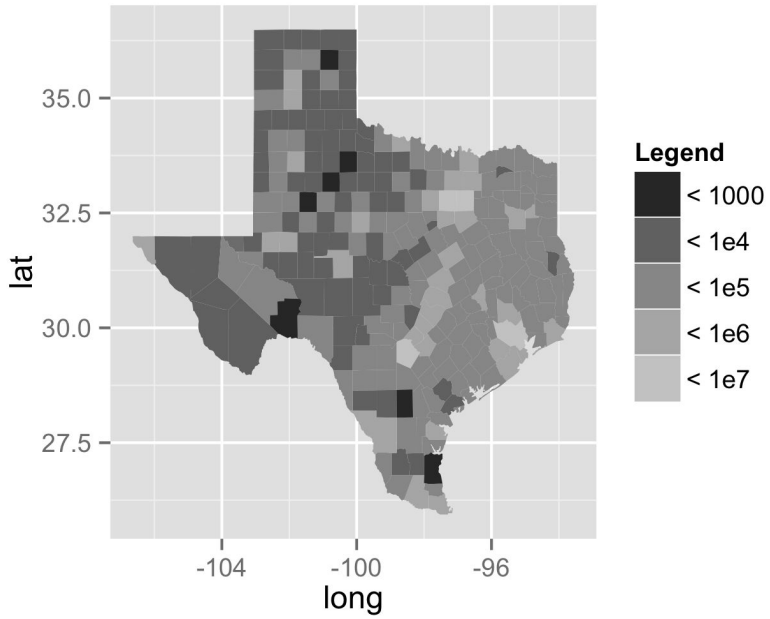


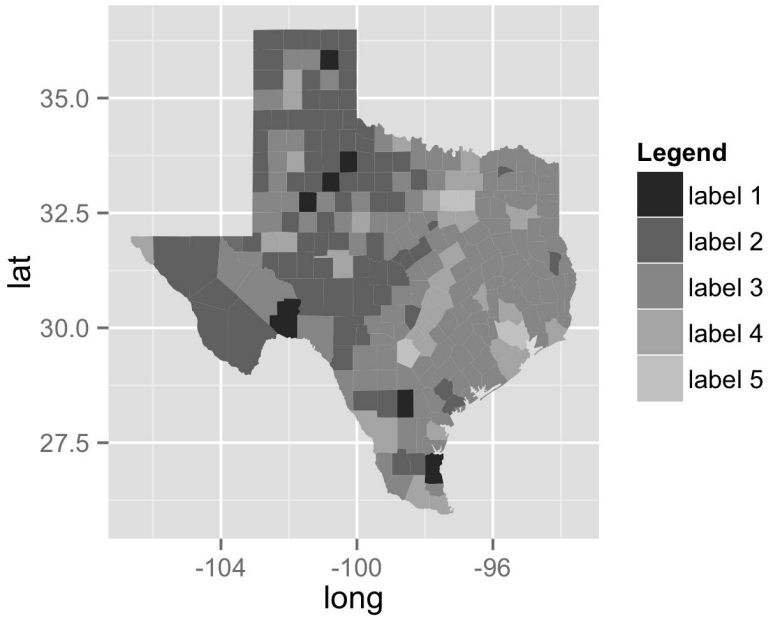


Labels

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

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

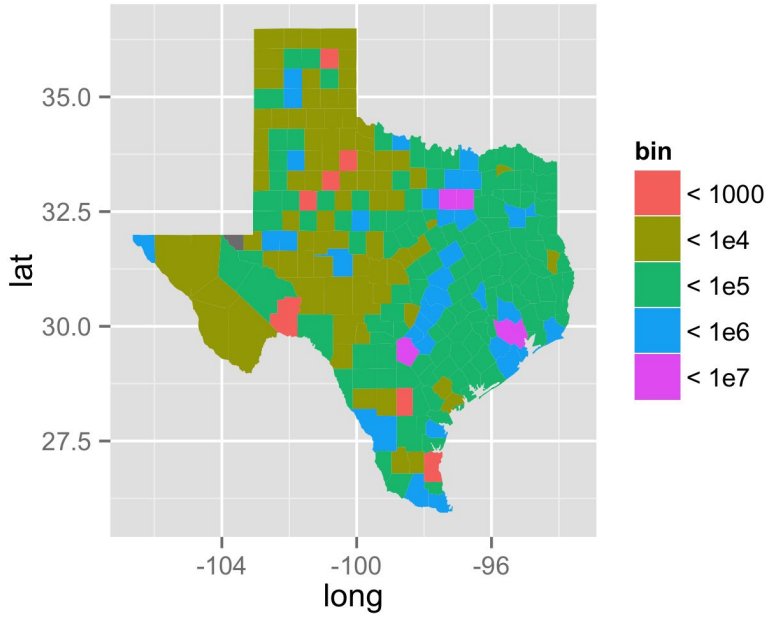


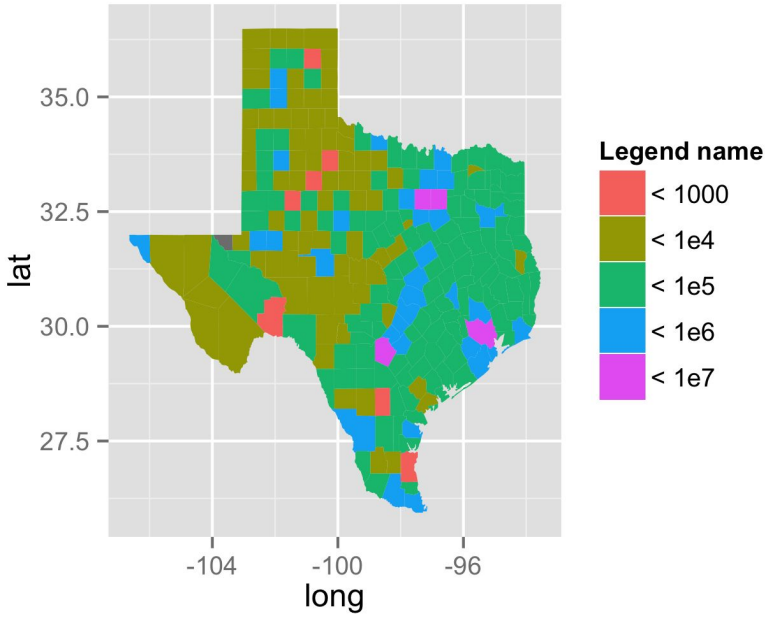


Labels

```
tx
```

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





Your Turn

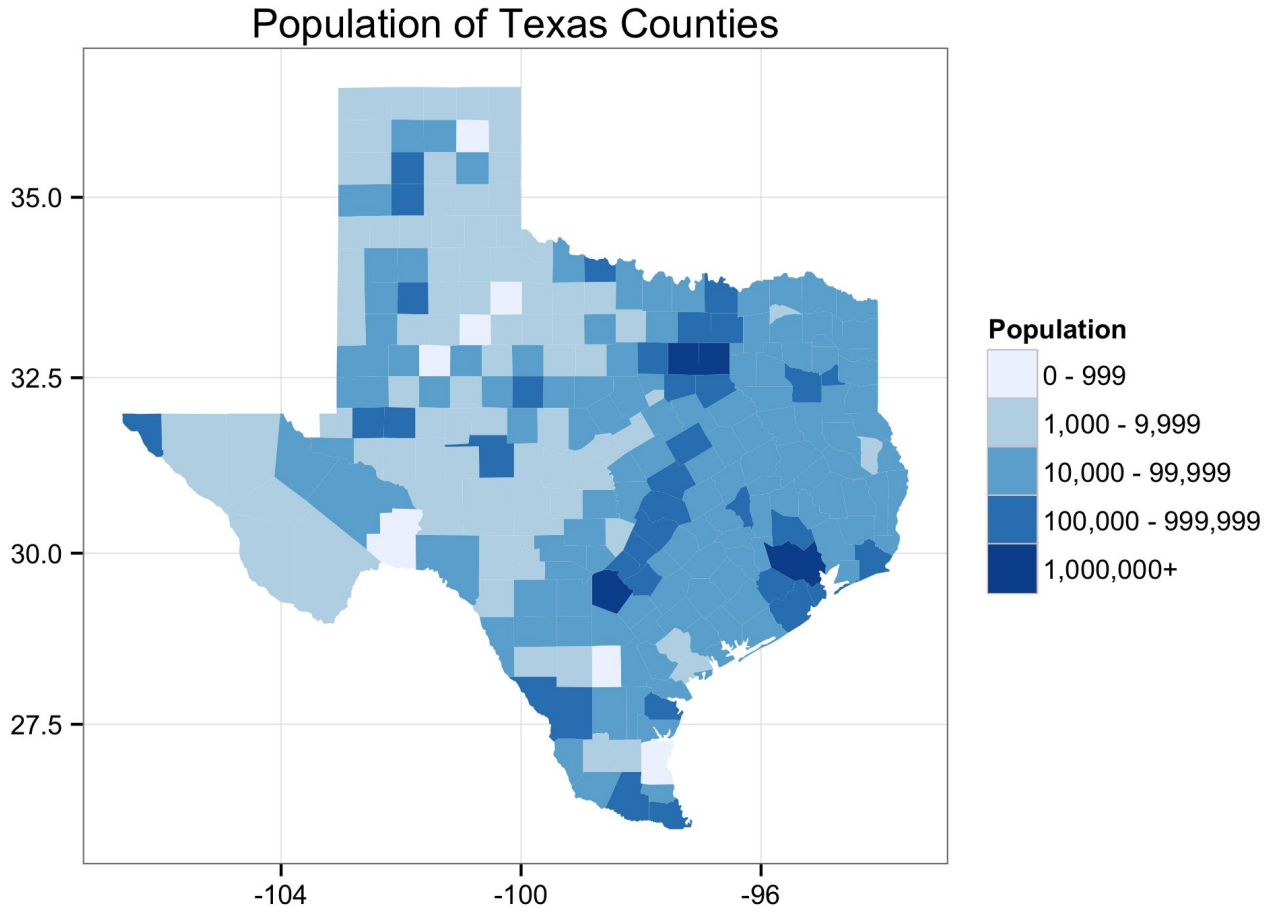
Continue editing the Texas map you began creating previously by adding an informative title and set of labels to the legend.

```
#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**

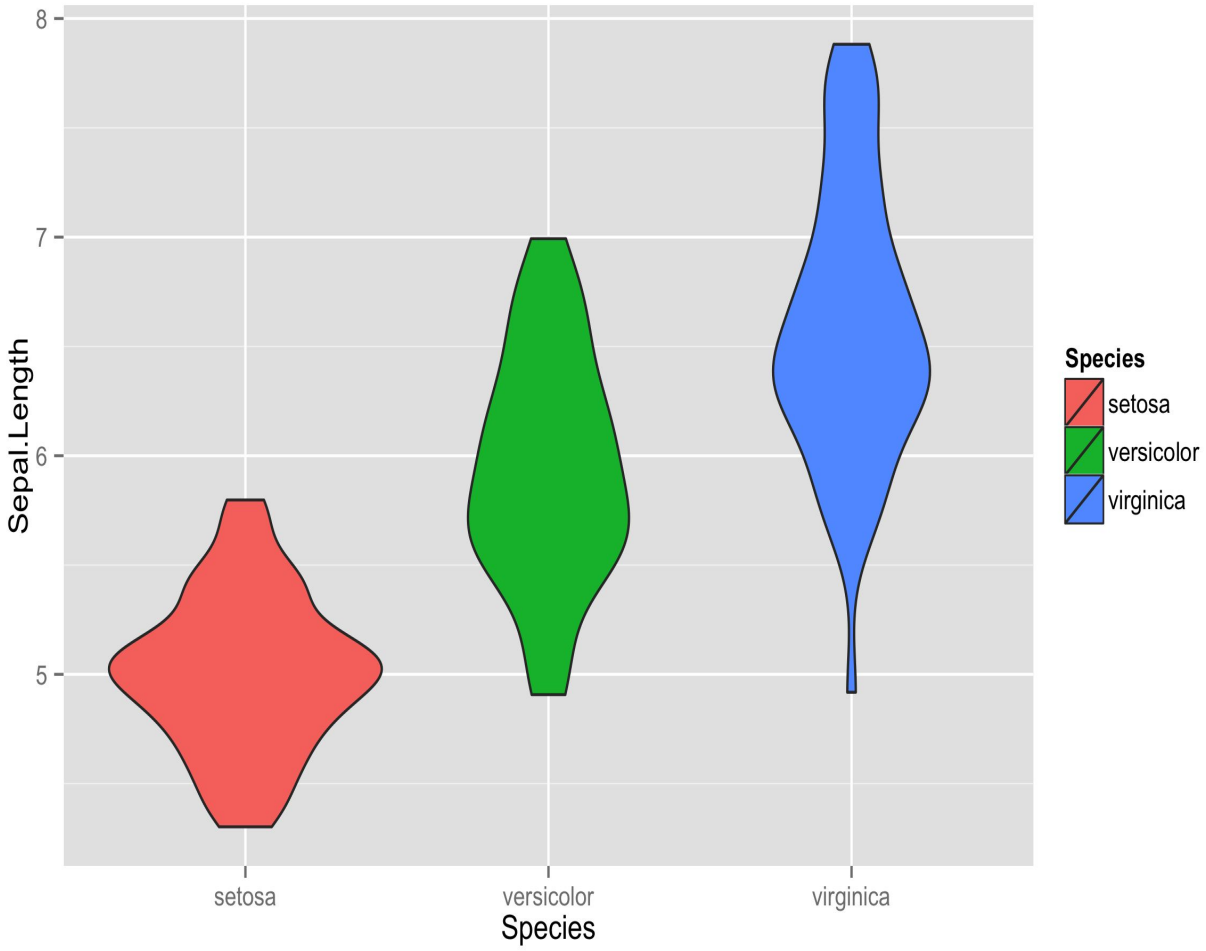
Other Visualizations

Violin Plots

Violin plots are similar to boxplots, but contain more information about the distribution of the data.

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

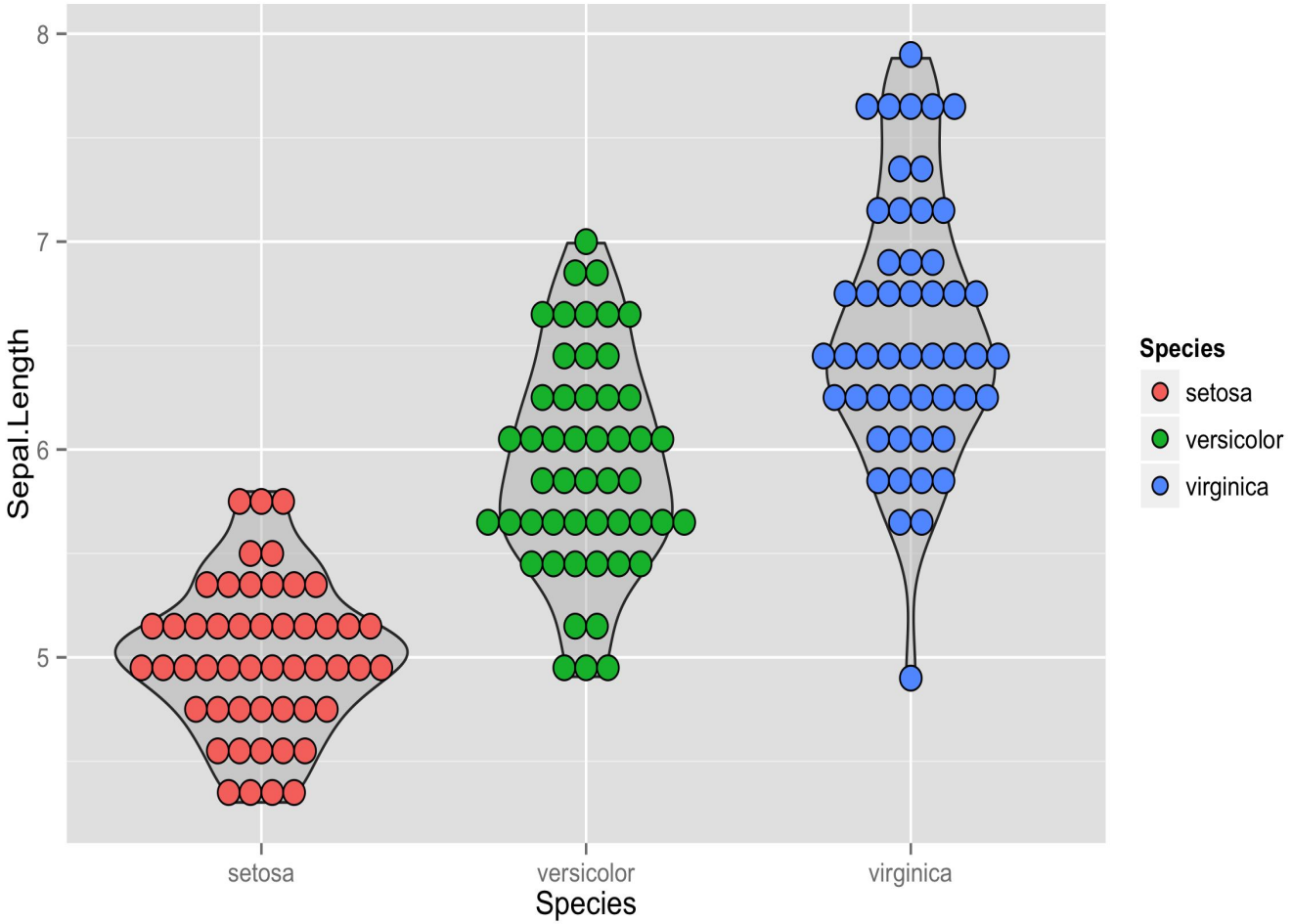
Violin Plots



Violin Plot 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")
```

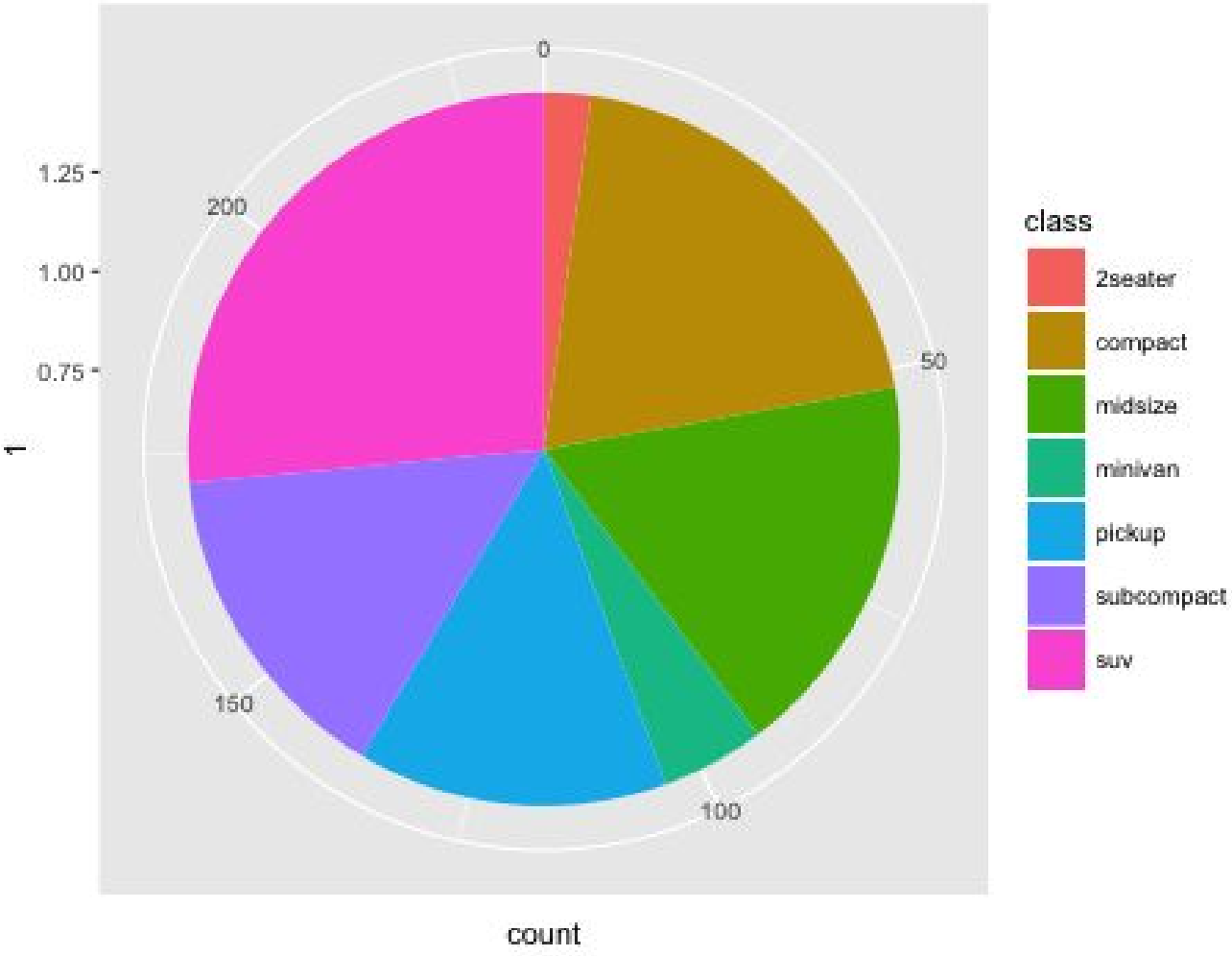
Violin Plots with Points



Pie Charts

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

Pie Charts

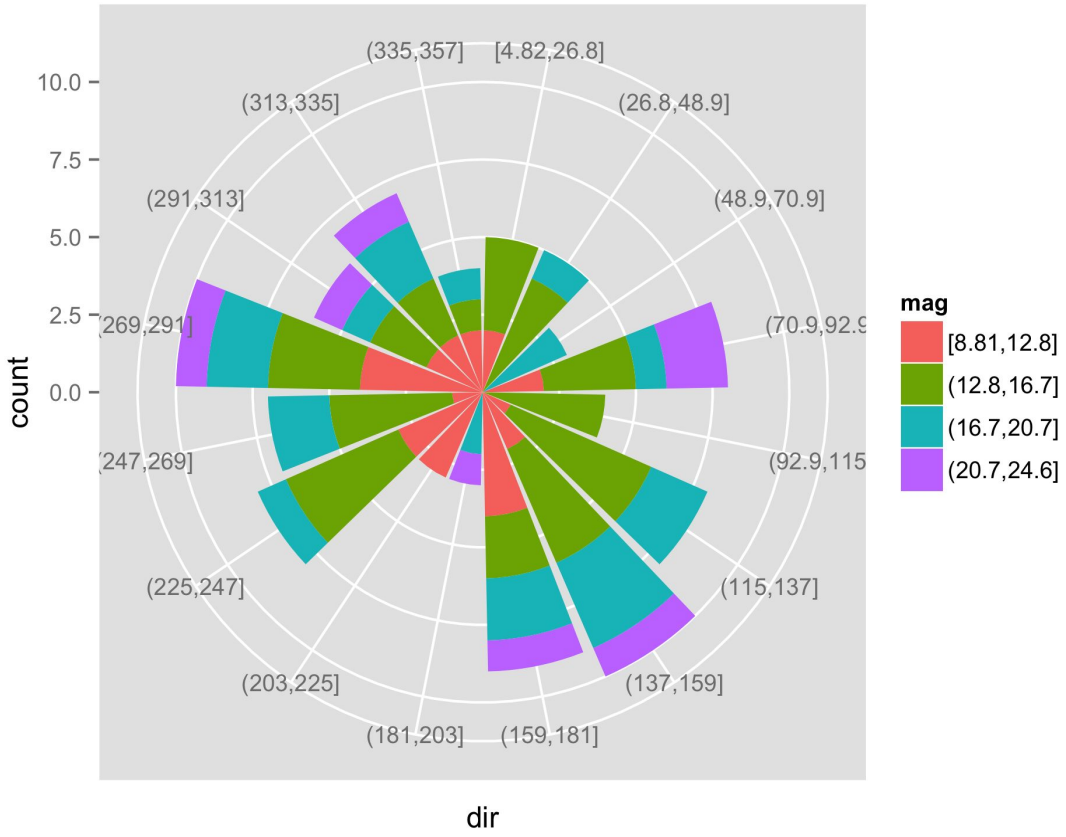

0
50
100
150
200
1.25
1.00
0.75
1
count
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

Rose Diagram

A wind rose, a commonly used graphics tool by meteorologists, 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()
```

Rose Diagram

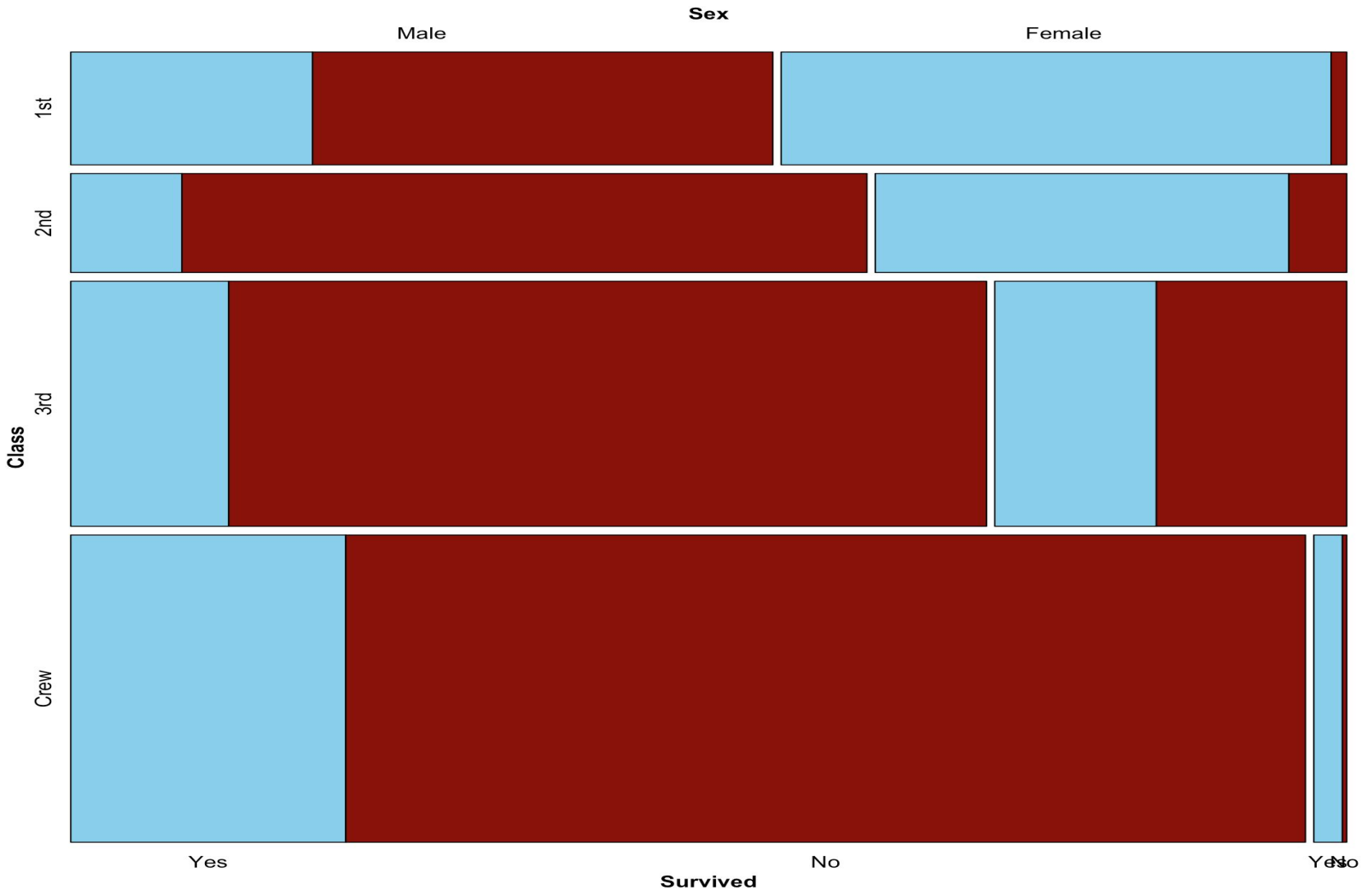


Mosaic Plot

Divide the data according to different variables, and then use rectangles of different sizes to represent different groups of data.

```
#install.packages("vcd")
library(vcd)
mosaic(Survived ~ Class + Sex, data=Titanic, shade=T,
       highlighting_fill=c('red4',"skyblue"),
       highlighting_direction="left")
```

Mosaic Plot



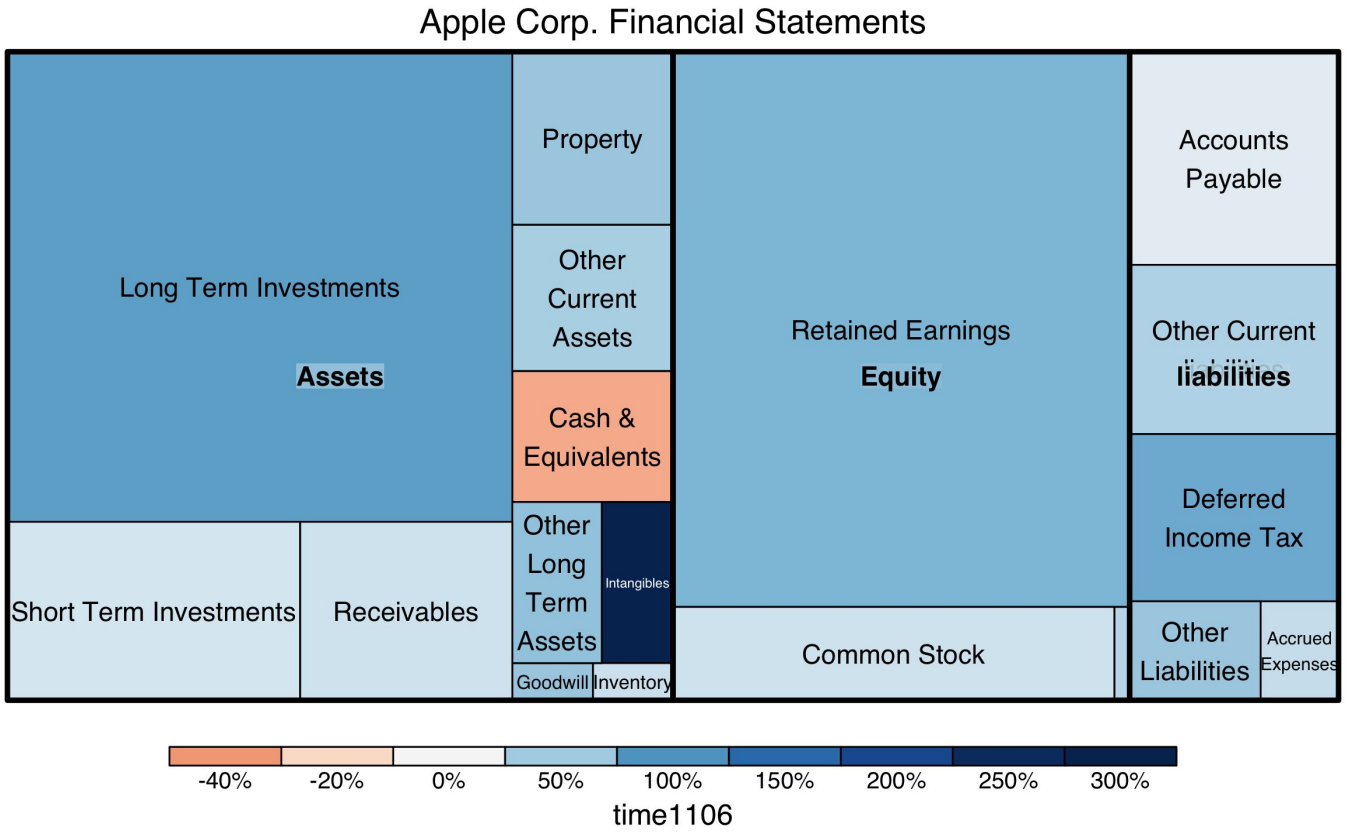
Hierarchical Tree Diagrams

Use nested rectangles of different sizes to represent hierarchical data. Hierarchical data includes file directories, organizational structure, etc.

Example: Visualizing Apple Corp.'s financial statements.

```
install.packages("treemap")
library(treemap)
data = read.csv('data/apple.csv', TRUE)
treemap(data,
        index=c("item", "subitem"),
        vSize="time1206",
        vColor="time1106",
        type="comp",
        title='Apple Corp. Financial Statements',
        palette='RdBu')
```

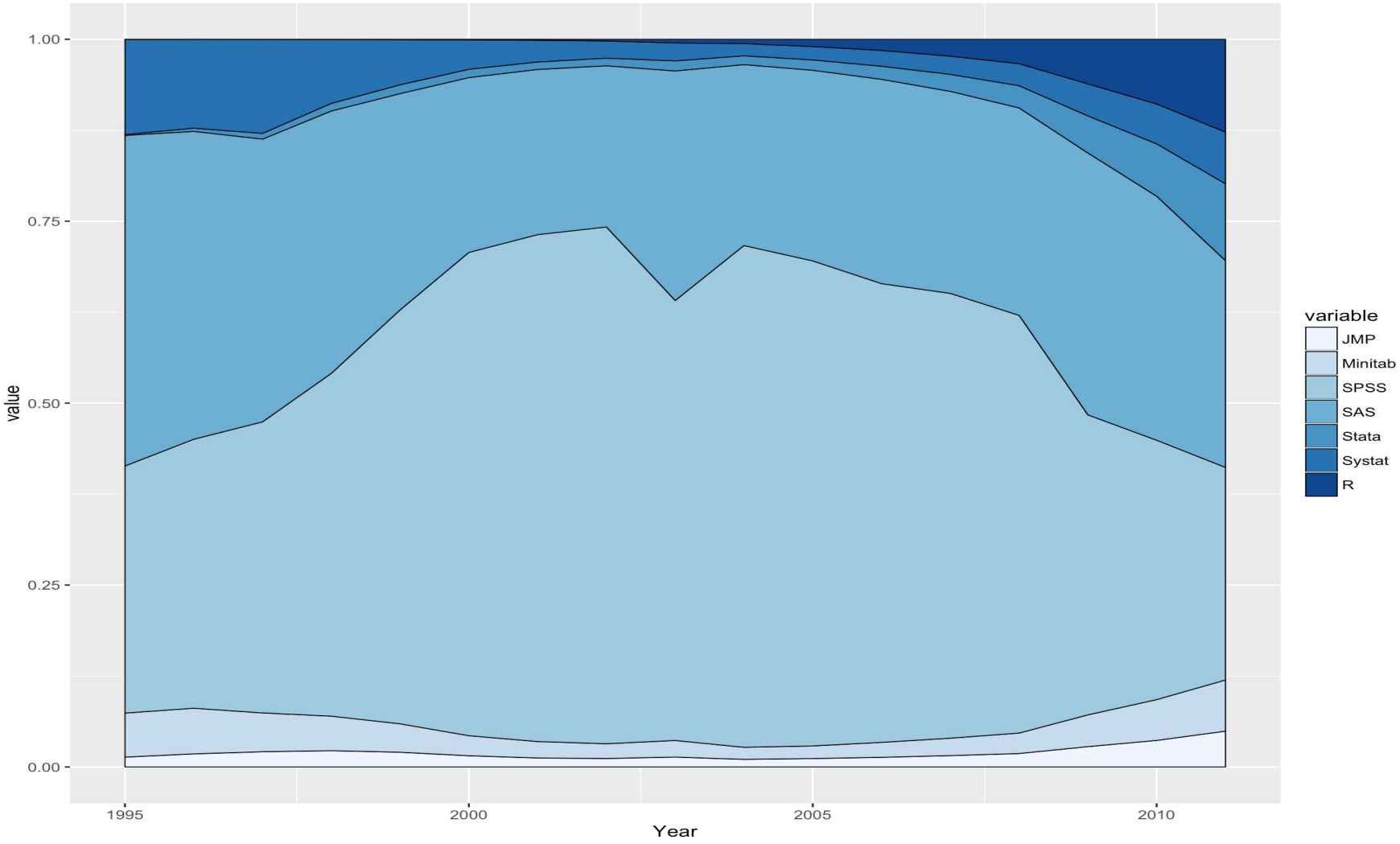
Hierarchical Tree Diagrams



Proportional structure in continuous data

```
data <- read.csv('data/soft_impact.csv', TRUE)
library(reshape2)
library(ggplot2)
data.melt <- melt(data, id='Year')
p <- ggplot(data.melt, aes(x=Year, y=value,
                           group=variable, fill=variable) ) +
    geom_area(color='black', size=0.3,
              position=position_fill() ) +
    scale_fill_brewer()
p
```

Proportional structure in continuous data



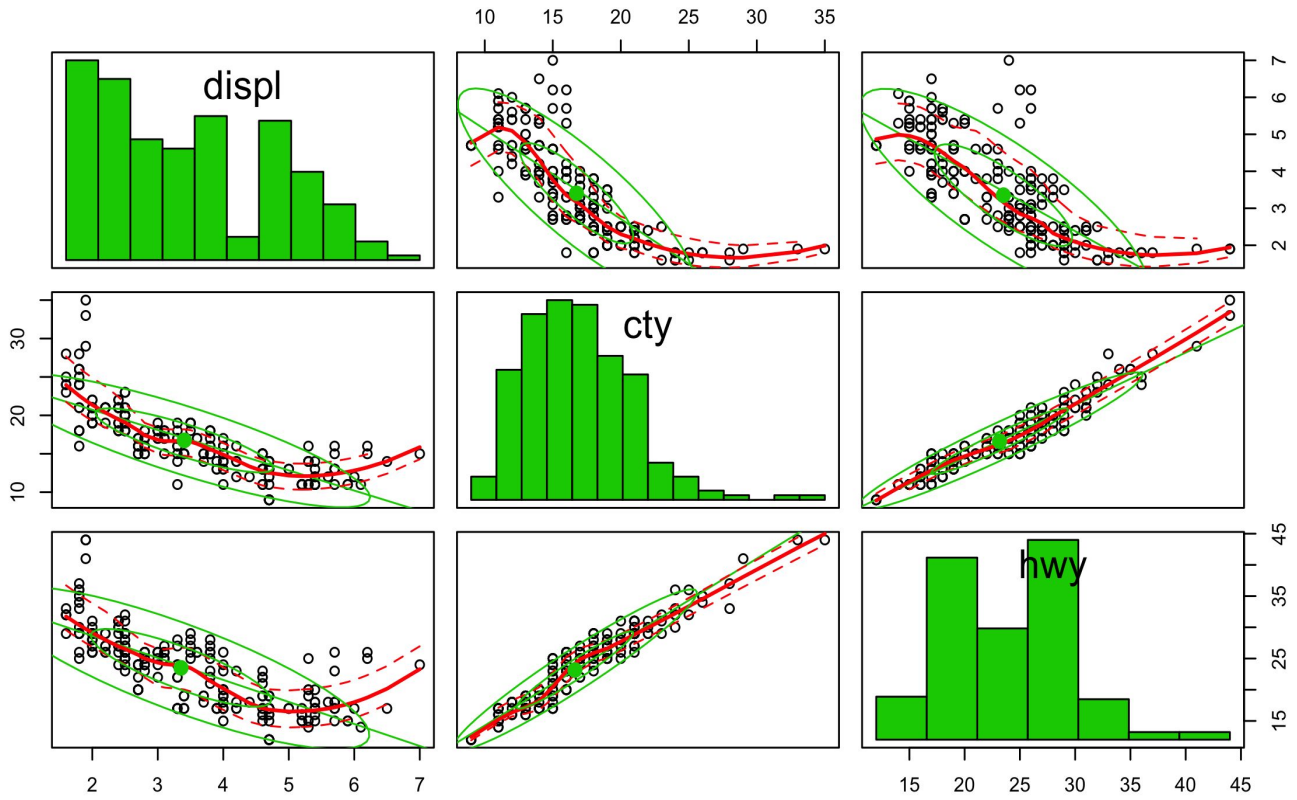
Relationships in Multivariate Data

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

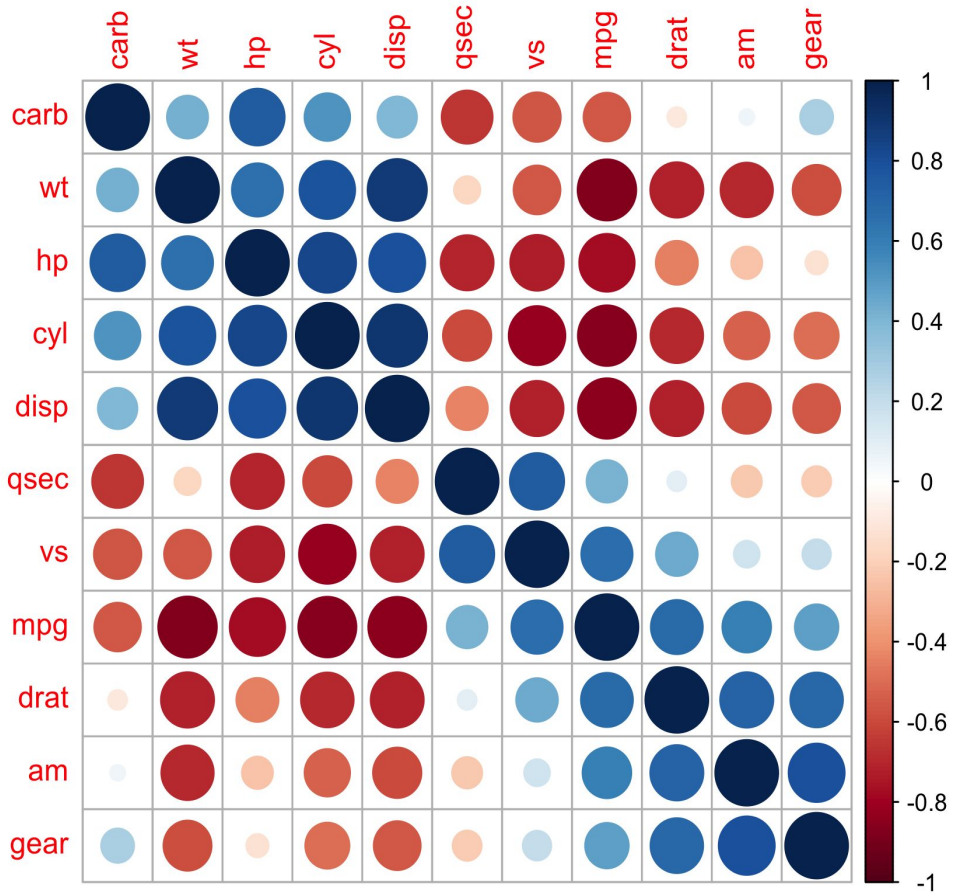


Scatter Plots with Multidimensional Data

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



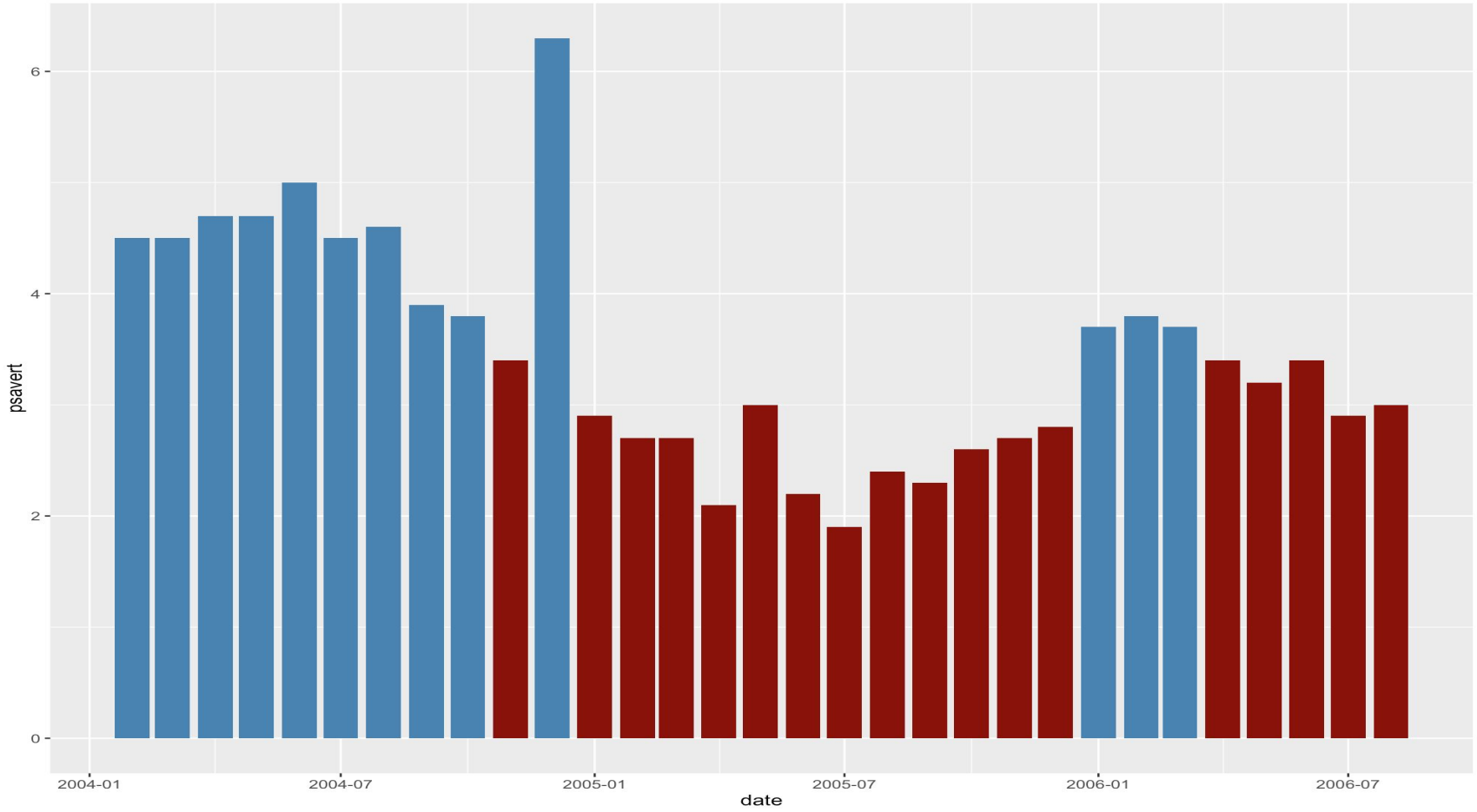
Exhibiting changes in time

Time-series visualizations

For visualizing time-series data, it is the most important to show both the general trend and fluctuations about that trend. For example, we can look at GDP and employment in the U.S. through the built-in economics dataset in the ggplot2 library.

```
fillcolor <- ifelse(economics[440:470,'psavert'] >
mean(economics[440:470,'psavert'])
, 'steelblue', 'red4')
p <- ggplot(economics[440:470,], aes(x=date, y=psavert) ) +
    geom_bar(stat='identity',
             fill=fillcolor)
p
```

Time-series visualizations

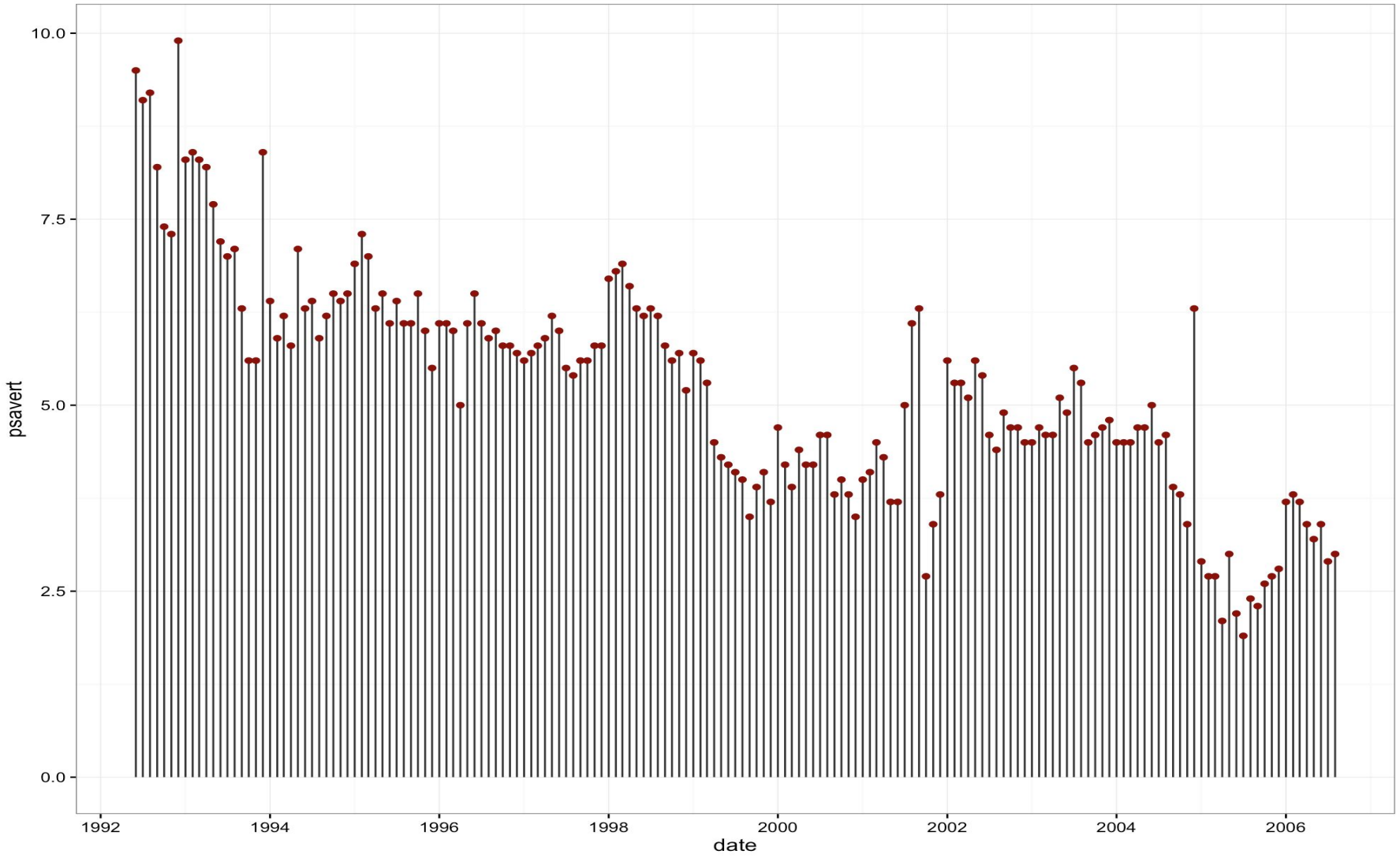


Time-series visualizations

For the time series of a small amount of data, we can use a bar graph to display the trend; here different colors are used for positive and negative values. But on a large scale this display would be too crowded. Density lines with points can be a clearer visual in this case.

```
p <- ggplot(economics[300:470,], aes(x=date, ymax=psavert, ymin=0) ) +
    geom_linerange(color='grey20', size=0.5) +
    geom_point(aes(y=psavert), color='red4') +
    theme_bw()
p
```

Time-series visualizations

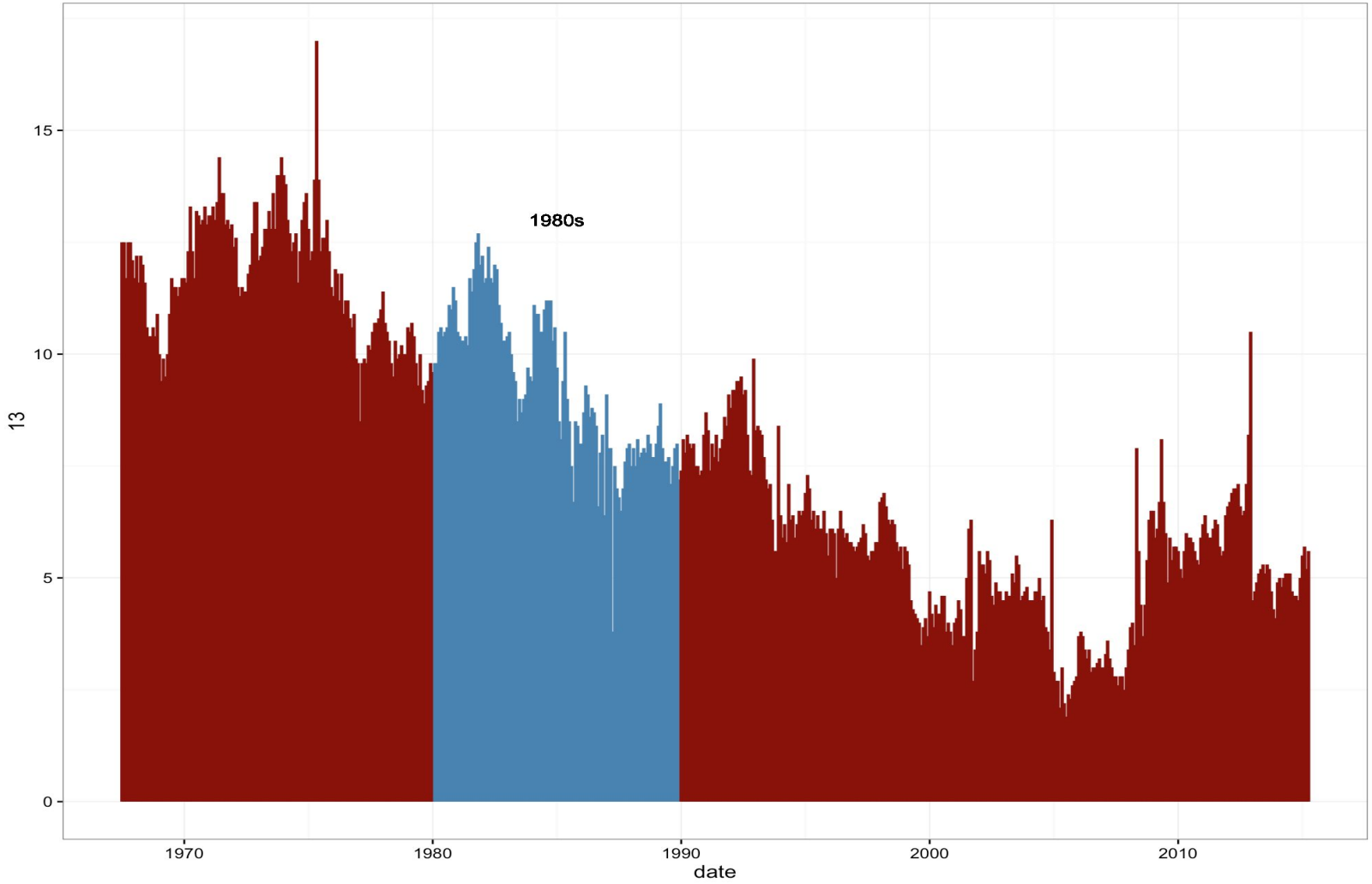


Time-series visualizations

When the data is more complicated, we can use line graphs or area charts to show changes in a trend.

```
fill.color <- ifelse(economics$date > '1980-01-01' &
                     economics$date < '1990-01-01',
                     'steelblue', 'red4')
p <- ggplot(economics, aes(x=date, ymax=psavert, ymin=0) ) +
    geom_linerange(color=fill.color, size=0.9) +
    geom_text(aes(x=as.Date("1985-01-01",'%Y-%m-%d'), y=13),
label="1980s") +
    theme_bw()
p
```

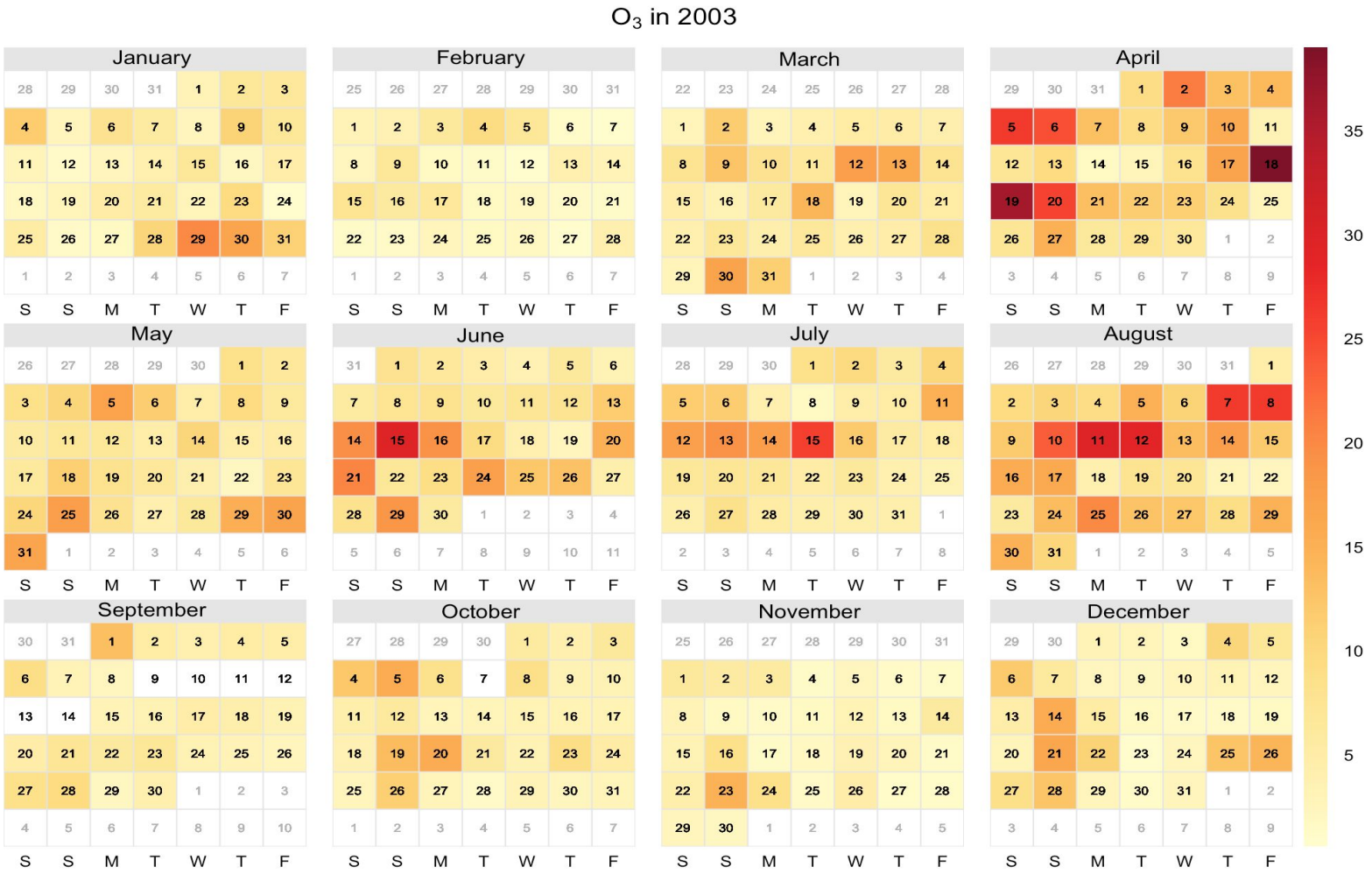
Time-series visualizations



Time-series visualizations

```
library(openair)
data(mydata)
calendarPlot(mydata, pollutant = "o3", year = 2003)
```

Time-series visualizations



Geographic information visualizations

Maps

- There are two main ways to draw maps:
  - Download the geographic information data, and then draw the geographical boundaries, identifying areas and locations according to the need
  - Download bitmap data from Google Maps, and then mark the location and path information on the Google Map itself

Maps

Generating a map of the world with maps and ggplot:

```
library(maps)
library(ggplot2)
world <- map_data("world")
naples <- ggplot(world, aes(x=long, y=lat, group=group) ) +
        geom_path(color='gray10', size=0.3) +
        geom_point(x=-81, y=26, size=10, shape='*') +
        scale_y_continuous(breaks=(-2:2) * 30) +
        scale_x_continuous(breaks=(-4:4) * 45) +
        coord_map("ortho", orientation=c(26, -80, 0))+
        theme(panel.grid.major = element_line(colour = "gray50"),
                        panel.background = element_rect(fill = "white"),
                        axis.text=element_blank(),
                        axis.ticks=element_blank(),
                        axis.title=element_blank() ) +
        ggtitle("Naples, Florida")
```

Maps

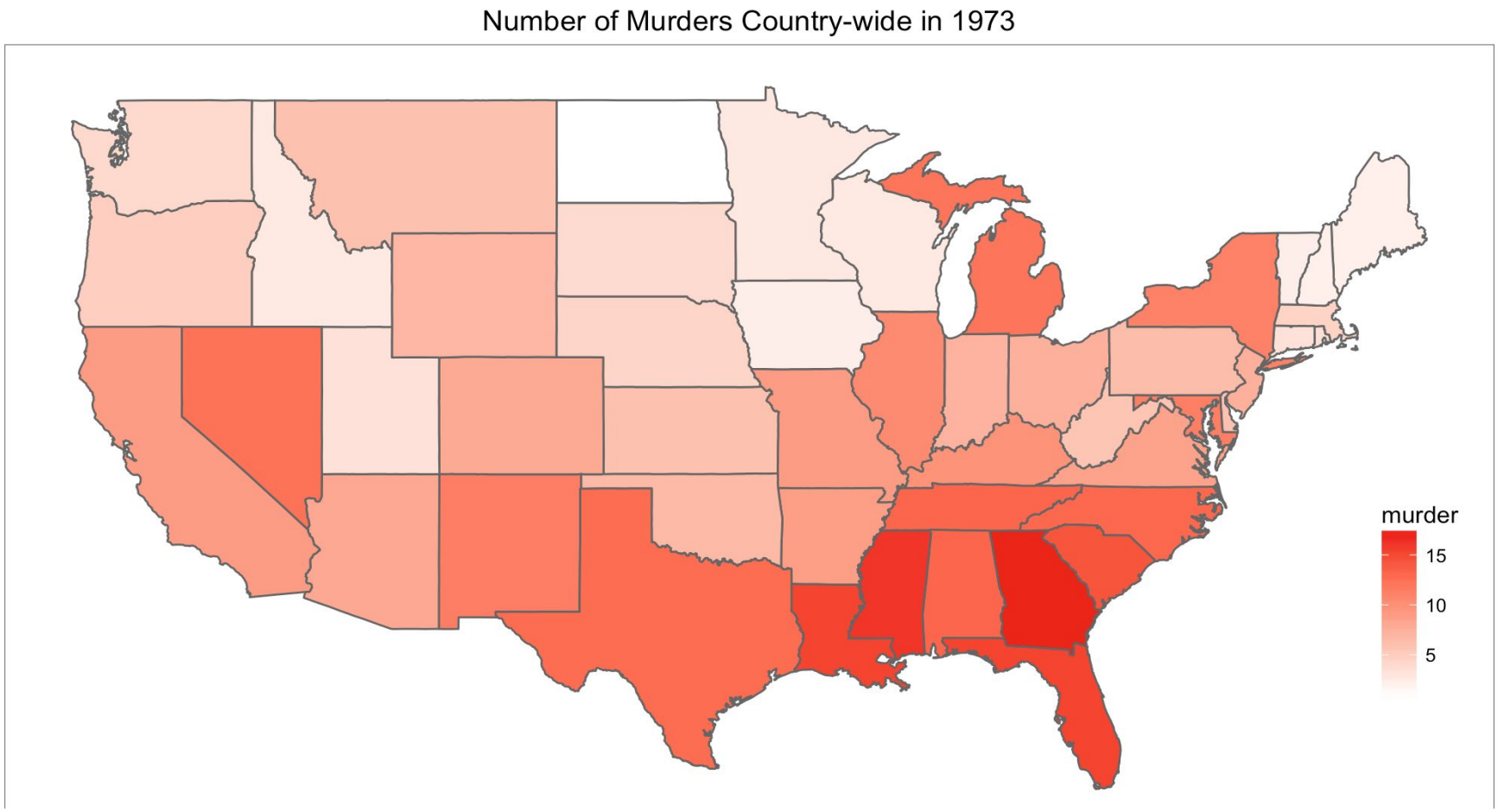
Maps

Generating a map of America with states color-coded by murder rate:

```
map <- map_data('state')
arrests <- USArrests; names(arrests) <- tolower(names(arrests))
arrests$region <- tolower(rownames(USArrests))

usmap <- ggplot(data=arrests) +
    geom_map(map =map, aes(map_id=region, fill=murder), color='gray40') +
    expand_limits(x=map$long, y=map$lat) +
    scale_fill_continuous(high='red2', low='white') +
    theme_bw() +
    theme(panel.grid.major = element_blank(),
          panel.background = element_blank(),
          axis.text=element_blank(), axis.ticks=element_blank(),
          axis.title=element_blank(), legend.position = c(0.95,0.28),
          legend.background=element_rect(fill="white", colour="white") ) +
          coord_map('mercator') +
          labs(title='Number of Murders Country-wide in 1973')
```

Maps



Maps

Another method of drawing a map, which we show through an example of China, is to download a document containing bitmap data from Google or openstreetmap, and then overlap points and lines on it with ggplot2.

```
library(ggmap)
```

```
Error in library(ggmap) : there is no package called 'ggmap
```

```
library(XML)
webpage <-
"http://data.earthquake.cn/datashare/globeEarthquake_csn.html"
tables <- readHTMLTable(webpage, stringsAsFactors=FALSE)
raw <- tables[[6]]
data <- raw[ ,c(1,3,4)]
names(data) <- c('date', 'lat', 'lon')
data$lat <- as.numeric(data$lat)
data$lon <- as.numeric(data$lon)
data$date <- as.Date(data$date, "%Y-%m-%d")
```

Maps

```
#read the map data from Google with the ggmap package
#mark the previous data on the
map
library(ggmap)
suppressPackageStartupMessages(library(ggmap))
quakemap <- ggmap(get_googlemap(center='china', zoom=4, maptype='terrain'),
          extent='device') +
            geom_point(data=data, aes(x=lon, y=lat),
           colour='red', alpha=0.7, na.rm=TRUE) +
            theme(legend.position="none")
```

Maps


Ulaanbaatar
Улаанбаатар
Mongolia
Almaty
Алматы
Beijing
北京
North Korea
Pyongyang
Seoul
South Korea
Gwangju
Busan
China
Nanning
Shanghai
East China Sea
Hangzhou
Nepal
Bhutan
Lucknow
Guangzhou
Taiwan
Bangladesh
India
Kolkata
Myanmar
(Burma)
Hong Kong
Laos
Yangon
Thailand
South
China Sea
Luzon
Manila
Vietnam
Bay of Bengal
Bangkok
Cambodia
Philippines
Andaman Sea
Google
Map data ©2015 Google, SK planet, ZENRIN

R and interactive visualizations

googleVis is an R package providing a interface between R and the Google visualization API. Compare the development trajectory of 20 country groups over the past several years:

```
library(googleVis); library(WDI);
library(RJSONIO)
suppressPackageStartupMessages(library(googleVis))
DF <- WDI(country=c("CN","RU","BR","ZA","IN",'DE','AU','CA','FR','IT',
                    'JP','MX','GB','US','ID','AR','KR','SA','TR'),
          indicator=c("NY.GDP.MKTP.CD", 'SP.DYN.LE00.IN', 'EN.ATM.CO2E.KT'),
                     start=2000, end=2009)
gdp_map <- gvisMotionChart(DF, idvar="country", timevar="year",
                    xvar='EN.ATM.CO2E.KT',
                    yvar='NY.GDP.MKTP.CD')
plot(gdp_map)
```

Case study exercises

Analyzing NBA data

- Calculate seasonal winning rates, and draw a bar chart.
- Calculate the seasonal winning rate at home and on the road, and draw a bar chart.
- According to the seasonal scores of home sides, draw a set of four histograms.
- According to the seasonal scores of home sides, draw boxplots of the five seasons.
- Draw the boxplots of scores of all competitions for home and away sides.
- Calculate the average winning percentage for each opponent, and make a scatterplot to find the strongest and the weakest teams.

Where to go from here

Where to go from here

- Learning ggplot2
  - ggplot2 mailing list
  - http://groups.google.com/group/ggplot2
- stackoverflow
  - http://stackoverflow.com/tags/ggplot2
- Cookbook for common graphics
  - http://wiki.stdout.org/rcookbook/Graphs/
- ggplot2 book
  - http://amzn.com/0387981403

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
- Cookbook for common graphics
  - http://wiki.stdout.org/rcookbook/Graphs/
- ggplot2 book
  - http://amzn.com/0387981403

More useful Packages

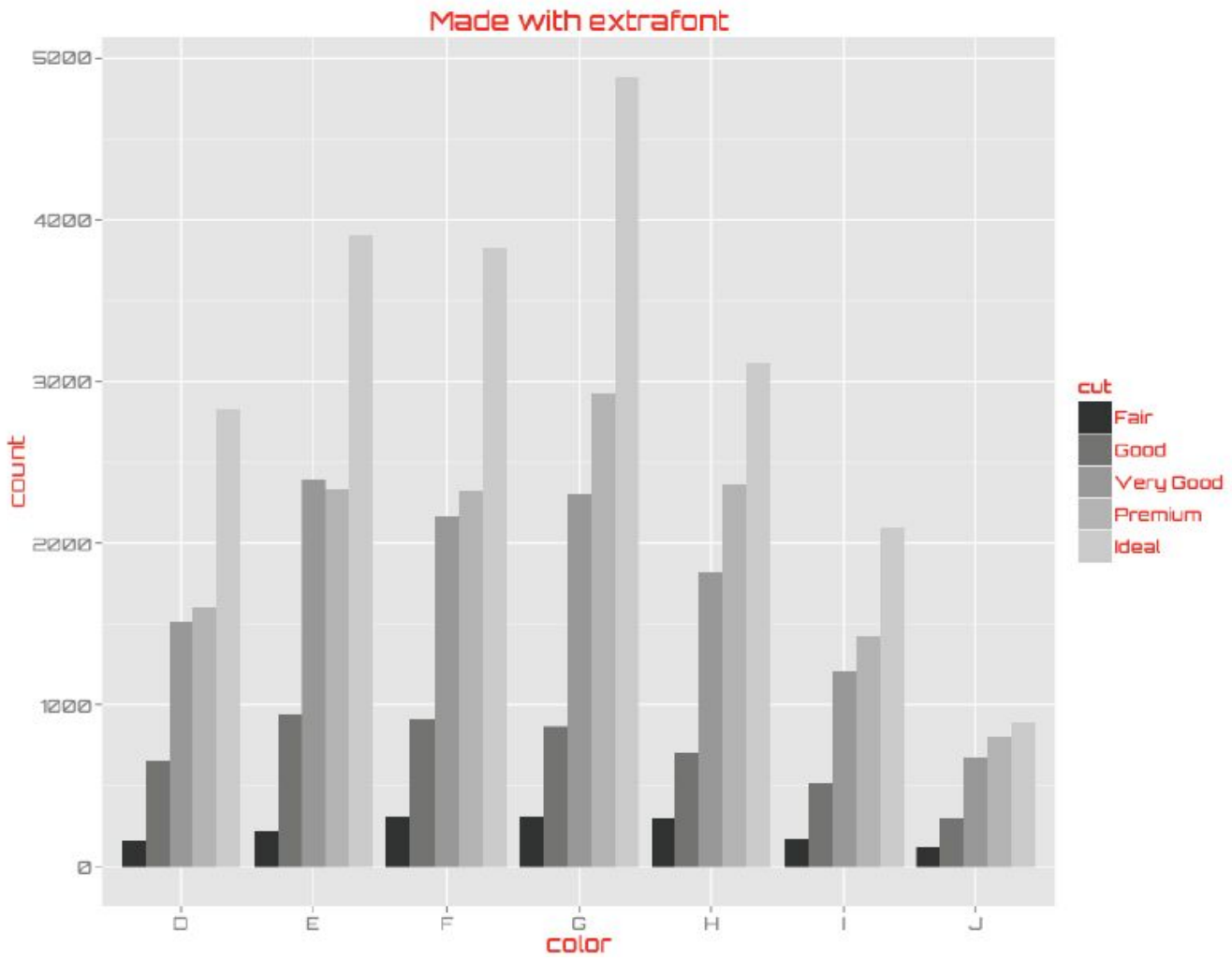
use system fonts in plots

extrafont

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

https://github.com/wch/extrafont

More Useful Packages



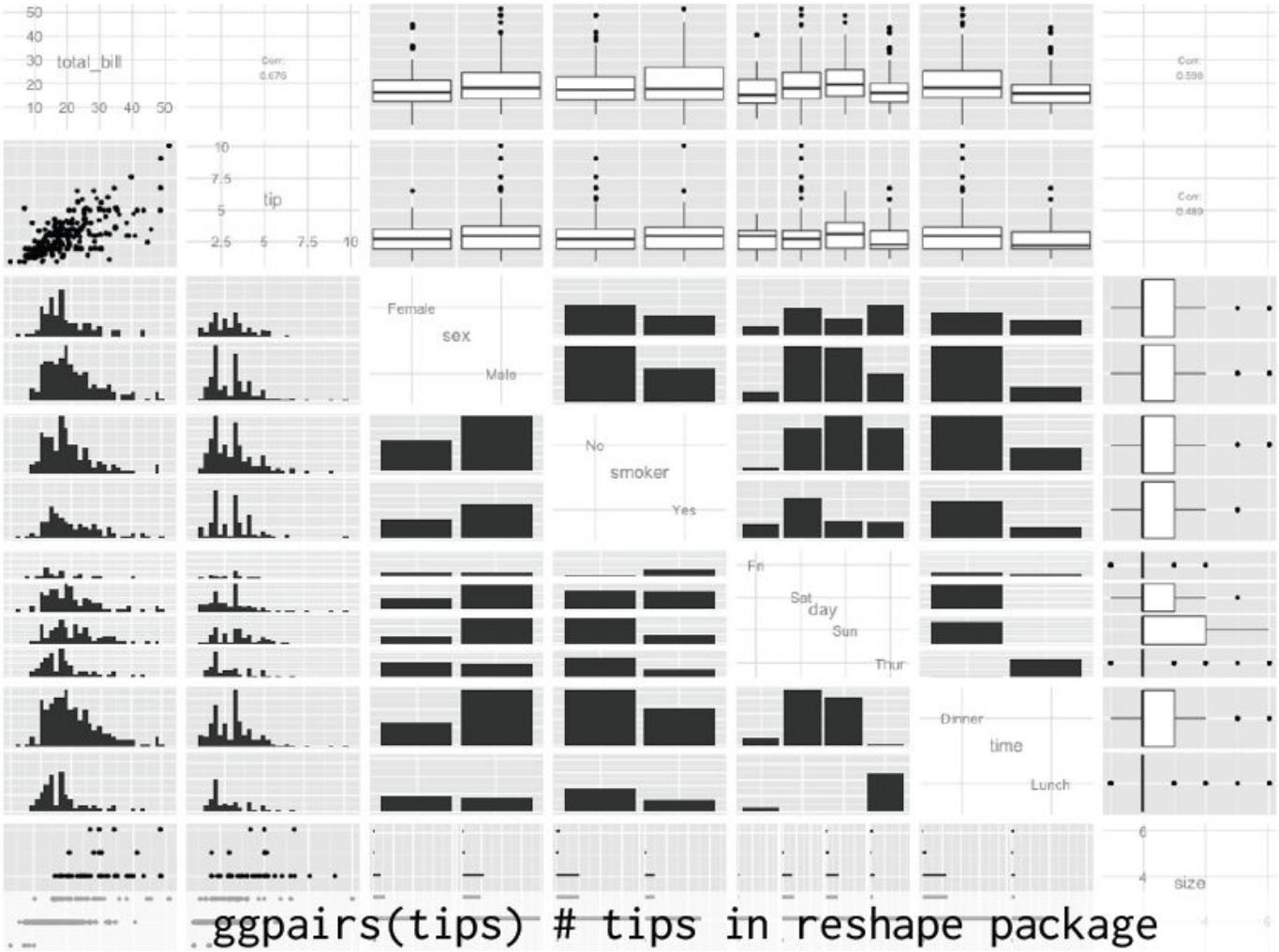
More Useful Packages

specialized plot types

GGally

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

More Useful Packages

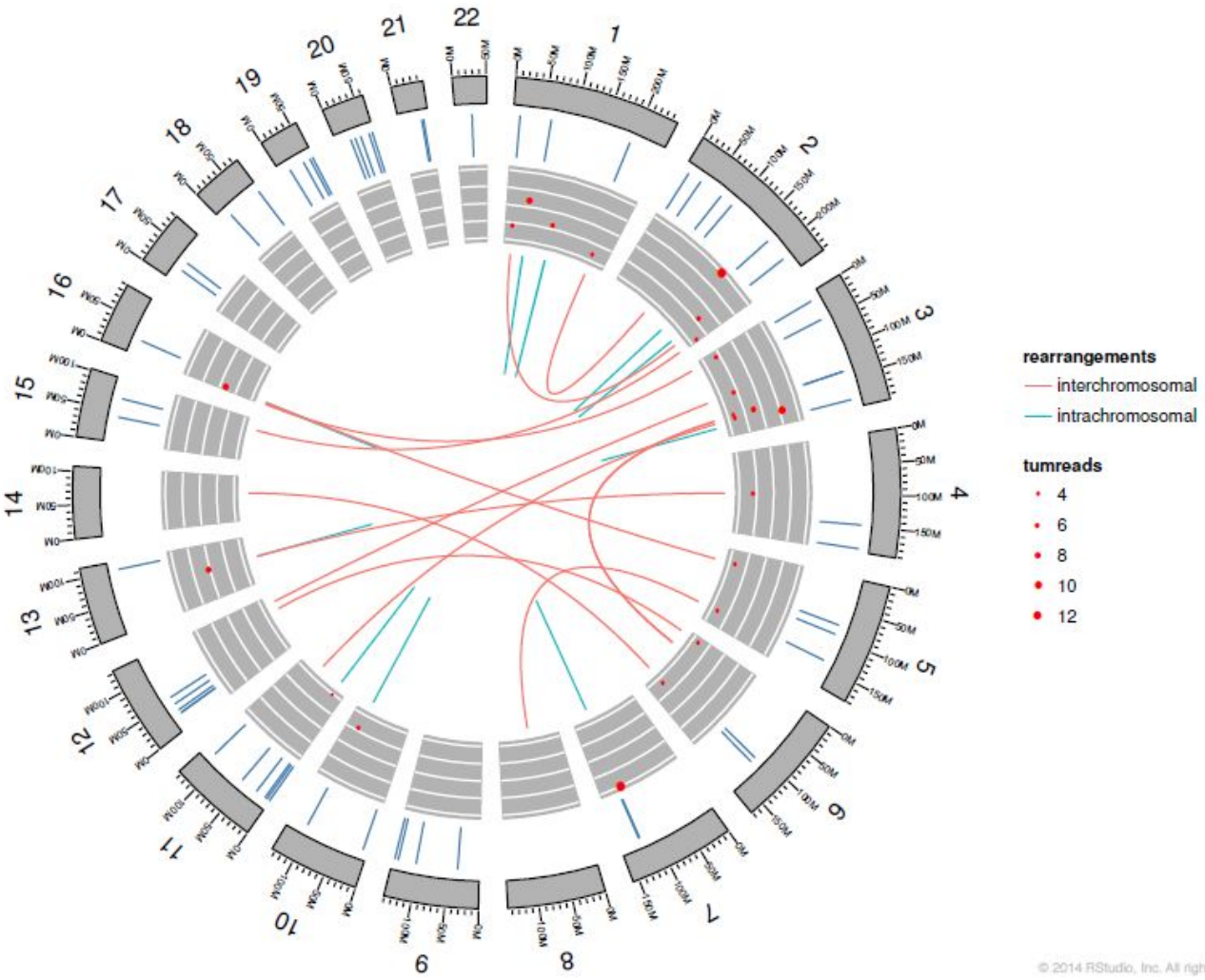



```
ggpairs(tips) # tips in reshape package
```

More Useful Packages


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

More Useful Packages



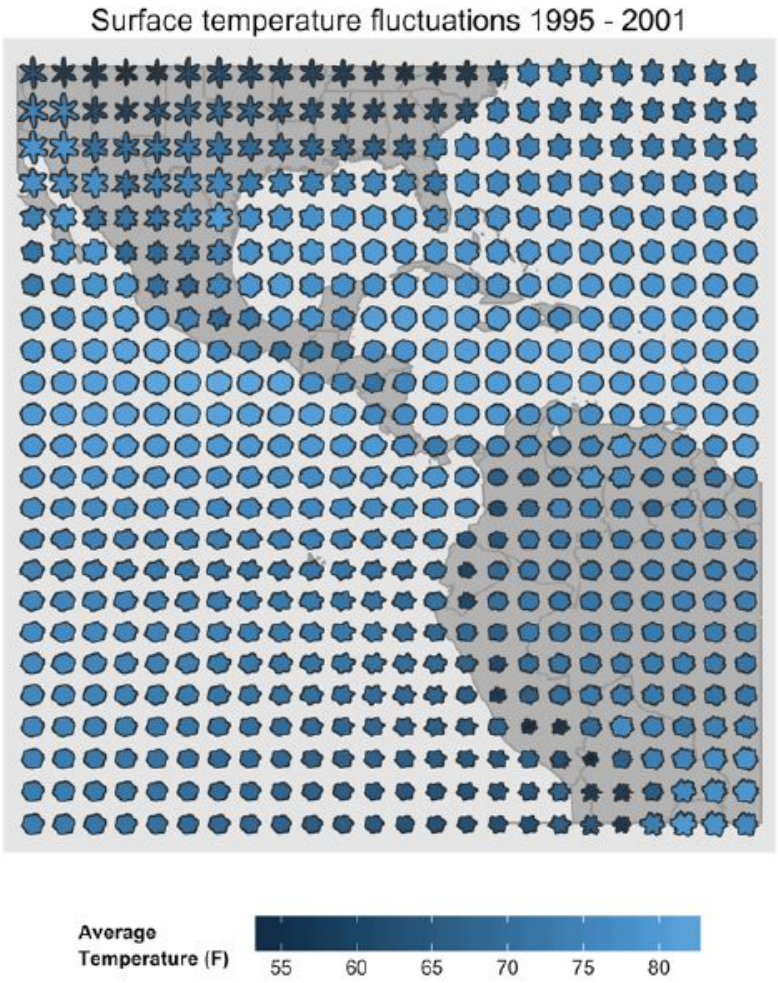
More Useful Packages

subplots and glyphs

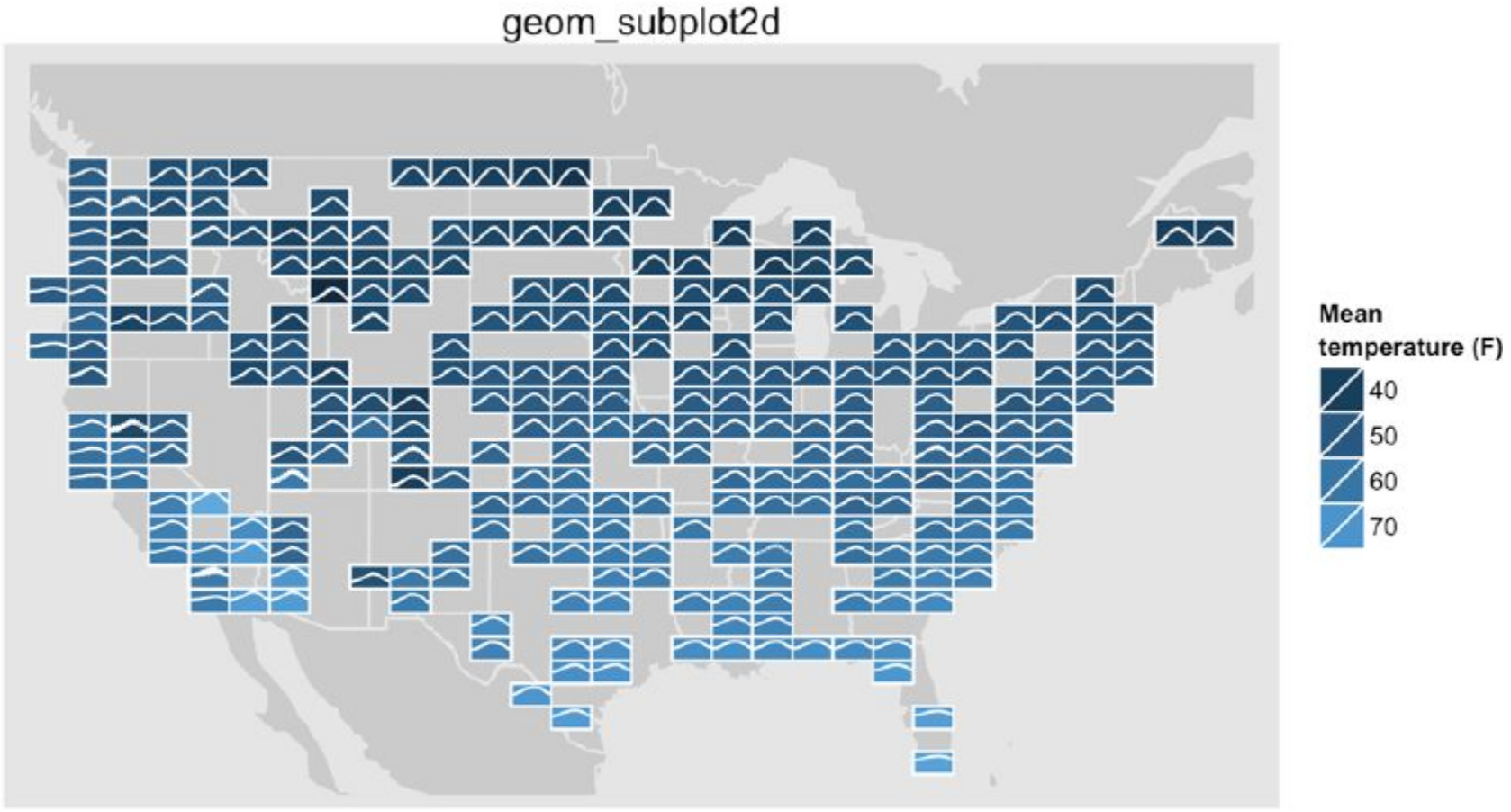
ggsubplot

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

More Useful Packages



More Useful Packages



More Useful Packages