Data Science with R and Python

Visualization: ggplot2

Peng Zhang

School of Mathematical Sciences, Zhejiang Universtiy

2023/07/04

Objectives

Why do we start with data visualization? Not only is data viz a big part of analysis, it's a way to SEE your progress as you learn to code.

> "ggplot2 implements the grammar of graphics, a coherent system for describing and building graphs. With ggplot2, you can do more faster by learning one system and applying it in many places." - **R4DS**

- install our first package, `ggplot2`, by installing `tidyverse`
- learn ggplot2 with a national parks visitation dataset (important to play with other data than your own, you'll learn something.)
- practice writing a script in RMarkdown
- practice the rstudio-github workflow

Install package: `tidyverse`

You don't need to go to CRAN's website to install packages, we can do it from within R with the command `install.packages("package-name-in-quotes")`.

We are going to be using the package `ggplot2`, which is actually bundled into a huge package called `tidyverse`. We will install `tidyverse` now, and use a few functions from the packages within. Also, check out tidyverse.org/.

```
## from CRAN:
# install.packages("tidyverse") ## do this once only to install the p
```

```
library(tidyverse) ## do this every time you restart R and need it
```

When you do this, it will tell you which packages are inside of `tidyverse` that have also been installed. Note that there are a few name conflicts; it is alerting you that we'll be using two functions from `dplyr` instead of the built-in stats package.

What's the difference between `install.packages()` and `library()`? Why do you need both? Here's an analogy:

- `install.packages()` is setting up electricity for your house. Just need to do this once (let's ignore monthly bills).
- `library()` is turning on the lights. You only turn them on when you need them, otherwise it wouldn't be efficient. And when you quit R, it turns the lights off, but the electricity lines are still there. So when you come back, you'll have to turn them on again with `library()`, but you already have your electricity set up.

Load data

Copy and paste the code chunk below and read it in to your RStudio to load the five datasets we will use in this section.

```
#National Parks in California
ca <- read_csv("data/ca.csv")

#Acadia National Park
acadia <- read_csv("data/acadia.csv")

#Southeast US National Parks
se <- read_csv("data/se.csv")

#2016 Visitation for all Pacific West National Parks
visit_16 <- read_csv("data/visit_16.csv")

#All Nationally designated sites in Massachusetts
mass <- read_csv("data/mass.csv")
```

Plotting with `ggplot2`

`ggplot2` is a plotting package that makes it simple to create complex plots from data in a data frame. It provides a more programmatic interface for specifying what variables to plot, how they are displayed, and general visual properties. Therefore, we only need minimal changes if the underlying data change or if we decide to change from a bar plot to a scatterplot. This helps in creating publication quality plots with minimal amounts of adjustments and tweaking.

ggplot likes data in the 'long' format: i.e., a column for every dimension, and a row for every observation. Well structured data will save you lots of time when making figures with ggplot.

ggplot graphics are built step by step by adding new elements. Adding layers in this fashion allows for extensive flexibility and customization of plots.

What does ggplot2 stand for?

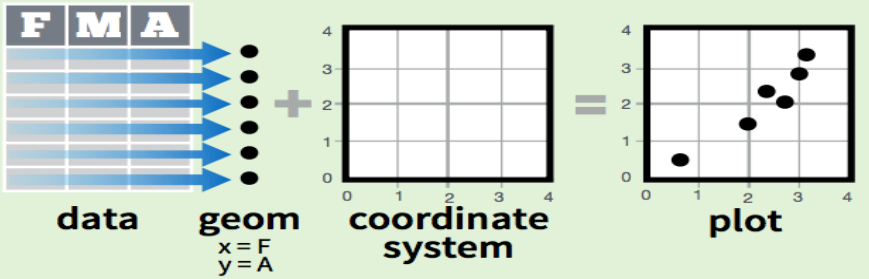
A Grammar of Graphics!

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

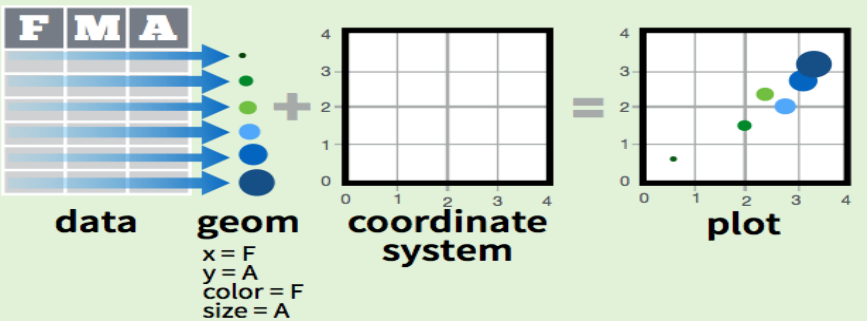
You can uniquely describe any plot as a combination of these 7 parameters.

Basics

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



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Data

We are going to use a National Park visitation dataset (from the National Park Service at https://irma.nps.gov/Stats/SSRSReports). Read in the data using `read_csv` and take a look at the first few rows using `head()` or `View()`.

```
head(ca)
```

```
## # A tibble: 6 × 7
##   region state code  park_name                          type          visitors
##   <chr>  <chr> <chr> <chr>                              <chr>            <dbl>
## 1 PW     CA    CHIS  Channel Islands National Park National Park     1200
## 2 PW     CA    CHIS  Channel Islands National Park National Park     1500
## 3 PW     CA    CHIS  Channel Islands National Park National Park     1600
## 4 PW     CA    CHIS  Channel Islands National Park National Park      300
## 5 PW     CA    CHIS  Channel Islands National Park National Park    15700
## 6 PW     CA    CHIS  Channel Islands National Park National Park    31000
```

This dataframe is already in a *long* format where all rows are an observation and all columns are variables. Among the variables in `ca` are:

1. `region`, US region where park is located.
2. `visitors`, the annual visitation for each `year`

To build a ggplot, we need to:

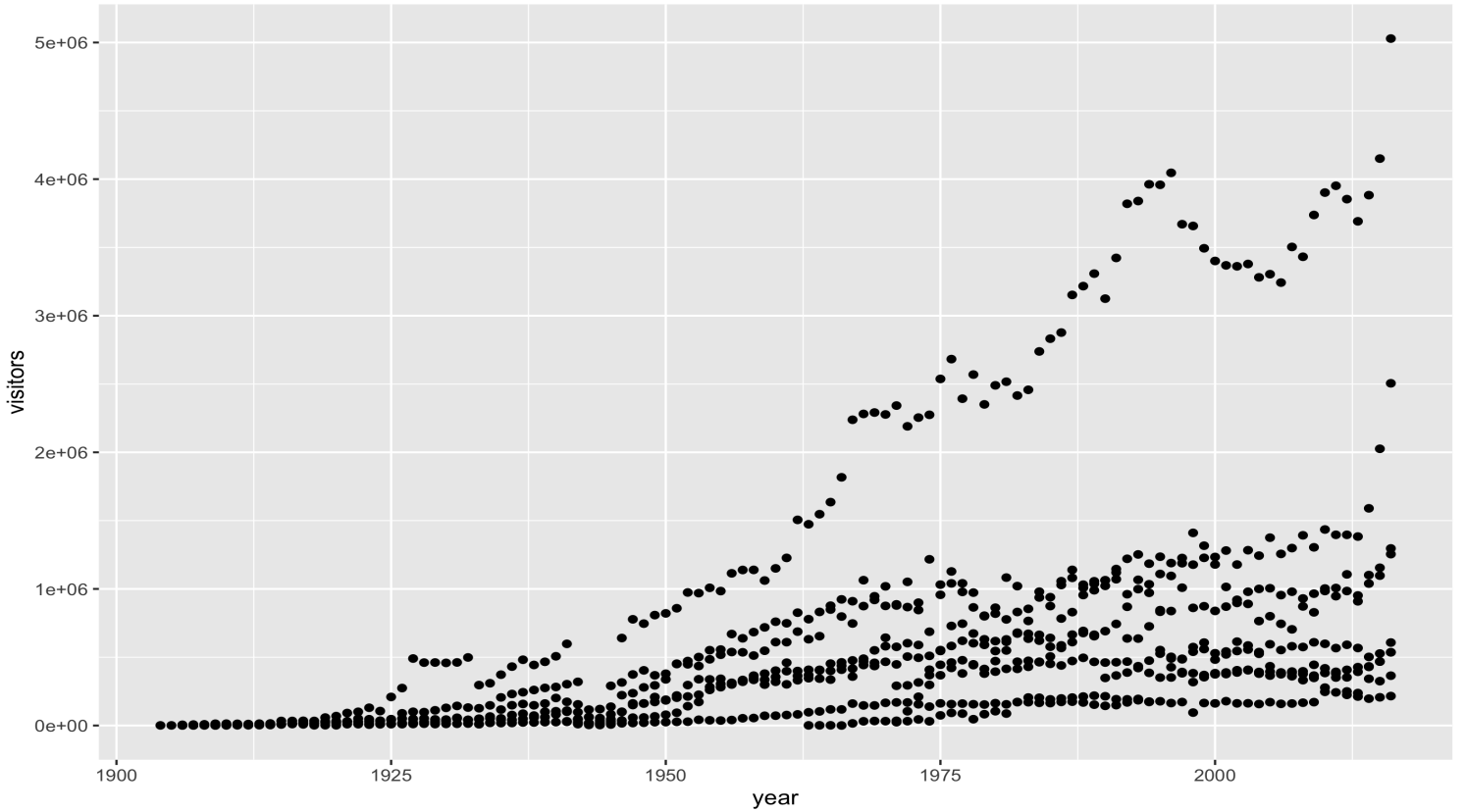
- use the `ggplot()` function and bind the plot to a specific data frame using the `data` argument

```
ggplot(data = ca)
```

- add `geoms` – graphical representation of the data in the plot (points, lines, bars). **ggplot2** offers many different geoms; we will use some common ones today, including: `geom_point()` for scatter plots, dot plots, etc. `geom_bar()` for bar charts `geom_line()` for trend lines, time-series, etc.

To add a geom to the plot use + operator. Because we have two continuous variables, let's use `geom_point()` first and then assign `x` and `y` aesthetics (`aes`):

```
ggplot(data = ca) +
  geom_point(aes(x = year, y = visitors))
```



Notes:

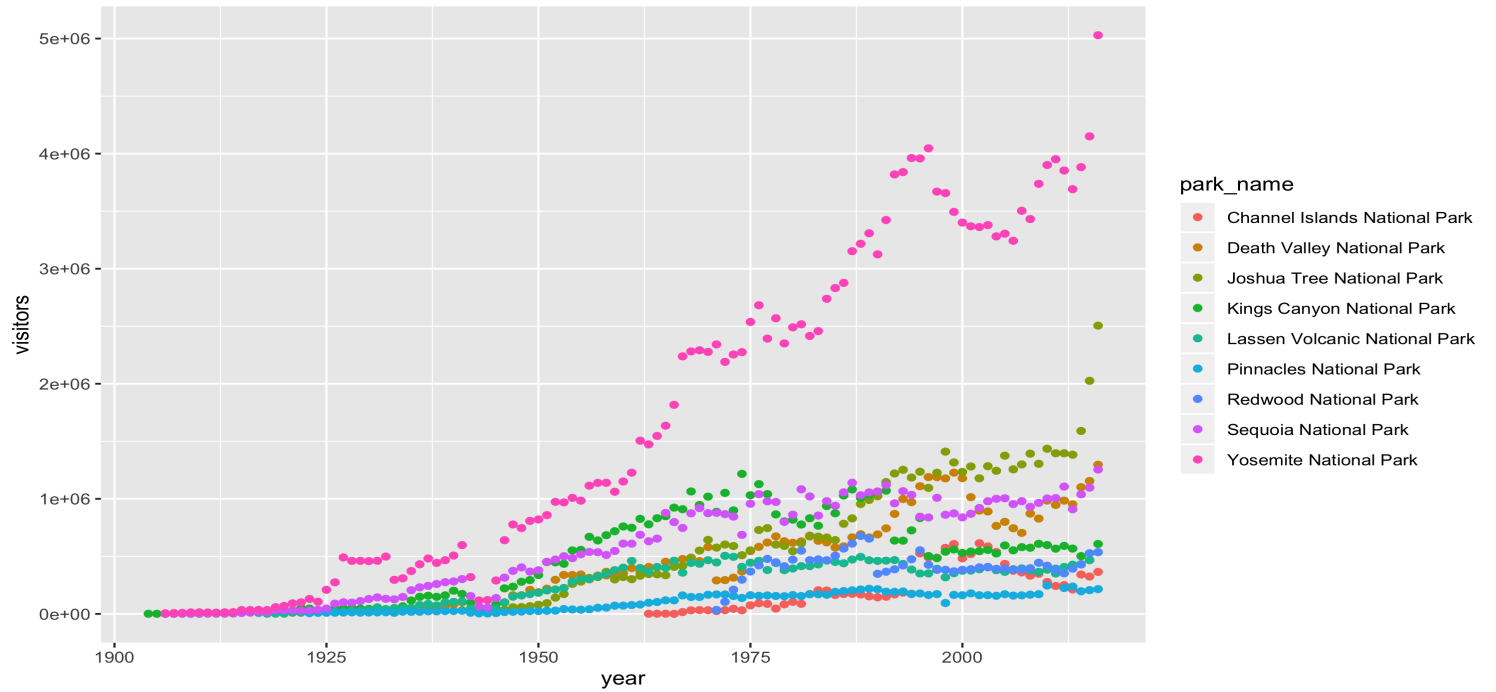
- Anything you put in the `ggplot()` function can be seen by any geom layers that you add (i.e., these are universal plot settings). This includes the x and y axis you set up in `aes()`.
- You can also specify aesthetics for a given geom independently of the aesthetics defined globally in the `ggplot()` function.
- The + sign used to add layers must be placed at the end of each line containing a layer. If, instead, the + sign is added in the line before the other layer, **ggplot2** will not add the new layer and will return an error message.

> **STOP: let's Commit, Pull and Push to GitHub**

Building your plots iteratively

Building plots with ggplot is typically an iterative process. We start by defining the dataset we'll use, lay the axes. We can distinguish each park by added the color argument to the aes:

```
ggplot(data = ca) +
    geom_point(aes(x = year, y = visitors, color = park_name))
```

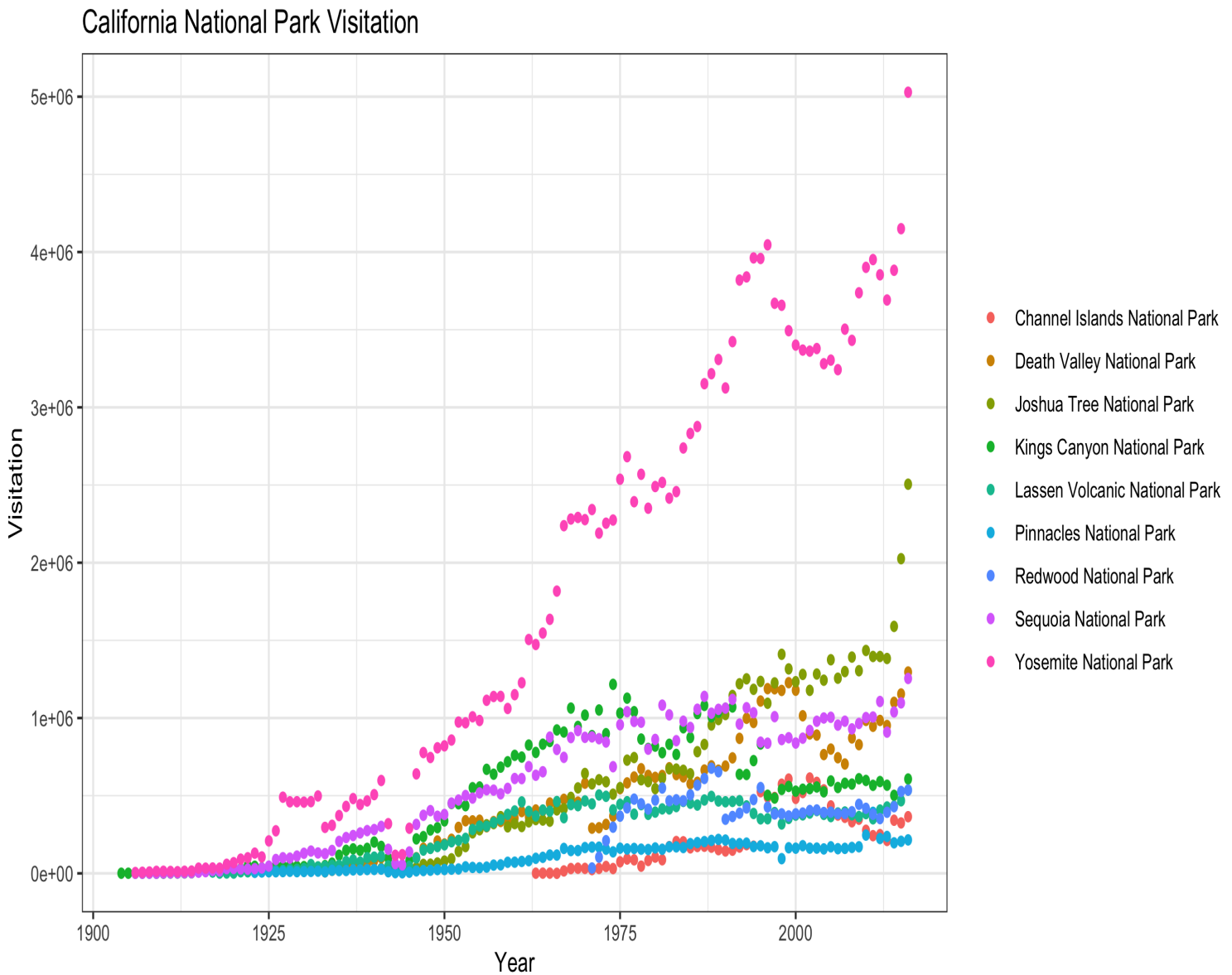


Customizing plots

Take a look at the **ggplot2** cheat sheet, and think of ways you could improve the plot.

Now, let's capitalize the x and y axis labels and add a main title to the figure. I also like to remove that standard gray background using a different `theme`. Many themes come built into the `ggplot2` package. My preference is `theme_bw()` but once you start typing `theme_` a list of options will pop up. The last thing I'm going to do is remove the legend title.

```
ggplot(data = ca) +
    geom_point(aes(x = year, y = visitors, color = park_name)) +
    labs(x = "Year",
       y = "Visitation",
       title = "California National Park Visitation") +
    theme_bw() +
    theme(legend.title=element_blank())
```


California National Park Visitation
Visitation
5e+06
4e+06
3e+06
2e+06
1e+06
0e+00
1900
1925
1950
1975
2000
Year
Channel Islands National Park
Death Valley National Park
Joshua Tree National Park
Kings Canyon National Park
Lassen Volcanic National Park
Pinnacles National Park
Redwood National Park
Sequoia National Park
Yosemite National Park

ggplot2 themes

In addition to `theme_bw()`, which changes the plot background to white, **ggplot2** comes with several other themes which can be useful to quickly change the look of your visualization.

The `ggthemes` package provides a wide variety of options (including an Excel 2003 theme). The **ggplot2** extensions website provides a list of packages that extend the capabilities of **ggplot2**, including additional themes.

Exercise (10 min)

1. Using the `se` dataset, make a scatterplot showing visitation to all national parks in the Southeast region with color identifying individual parks.

2. Change the plot so that color indicates `state`.

3. Customize by adding your own title and theme. You can also change the text sizes and angles. Try applying a 45 degree angle to the x-axis. Use your cheatsheet!

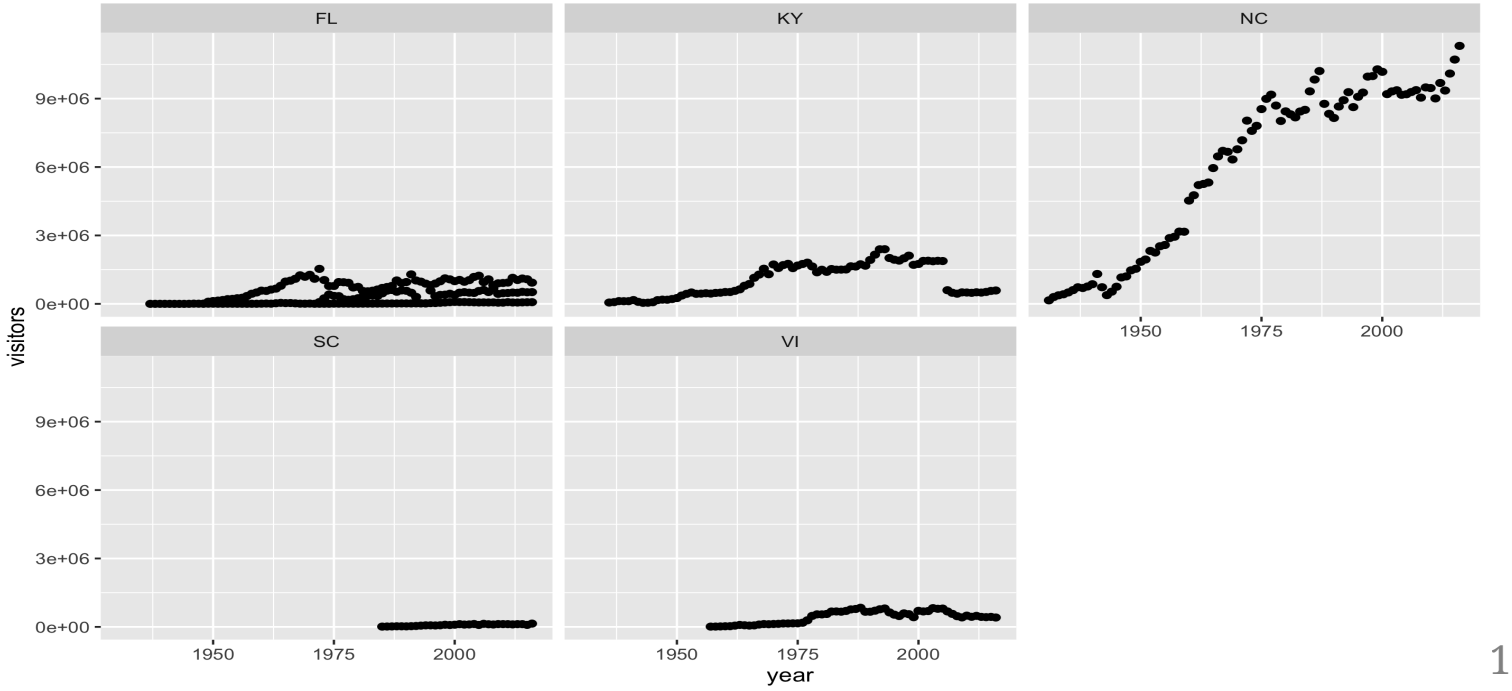
4. In the code below, why isn't the data showing up?

```
ggplot(data = se, aes(x = year, y = visitors))
```

Faceting

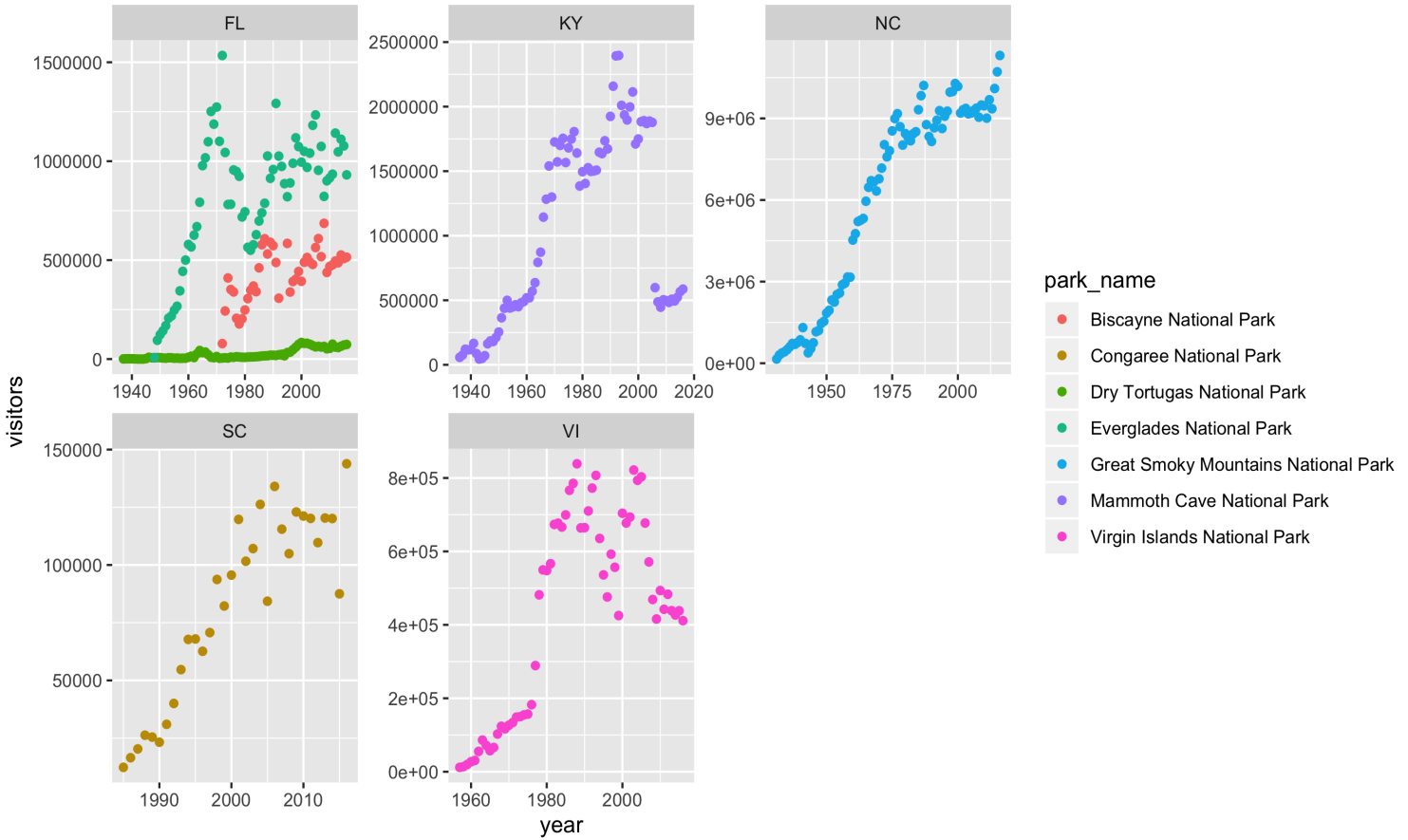
ggplot has a special technique called *faceting* that allows the user to split one plot into multiple plots based on data in the dataset. We will use it to make a plot of park visitation by state:

```
ggplot(data = se) +
    geom_point(aes(x = year, y = visitors)) +
    facet_wrap(~ state)
```



We can now make the faceted plot by splitting further by park using park_name (within a single plot):

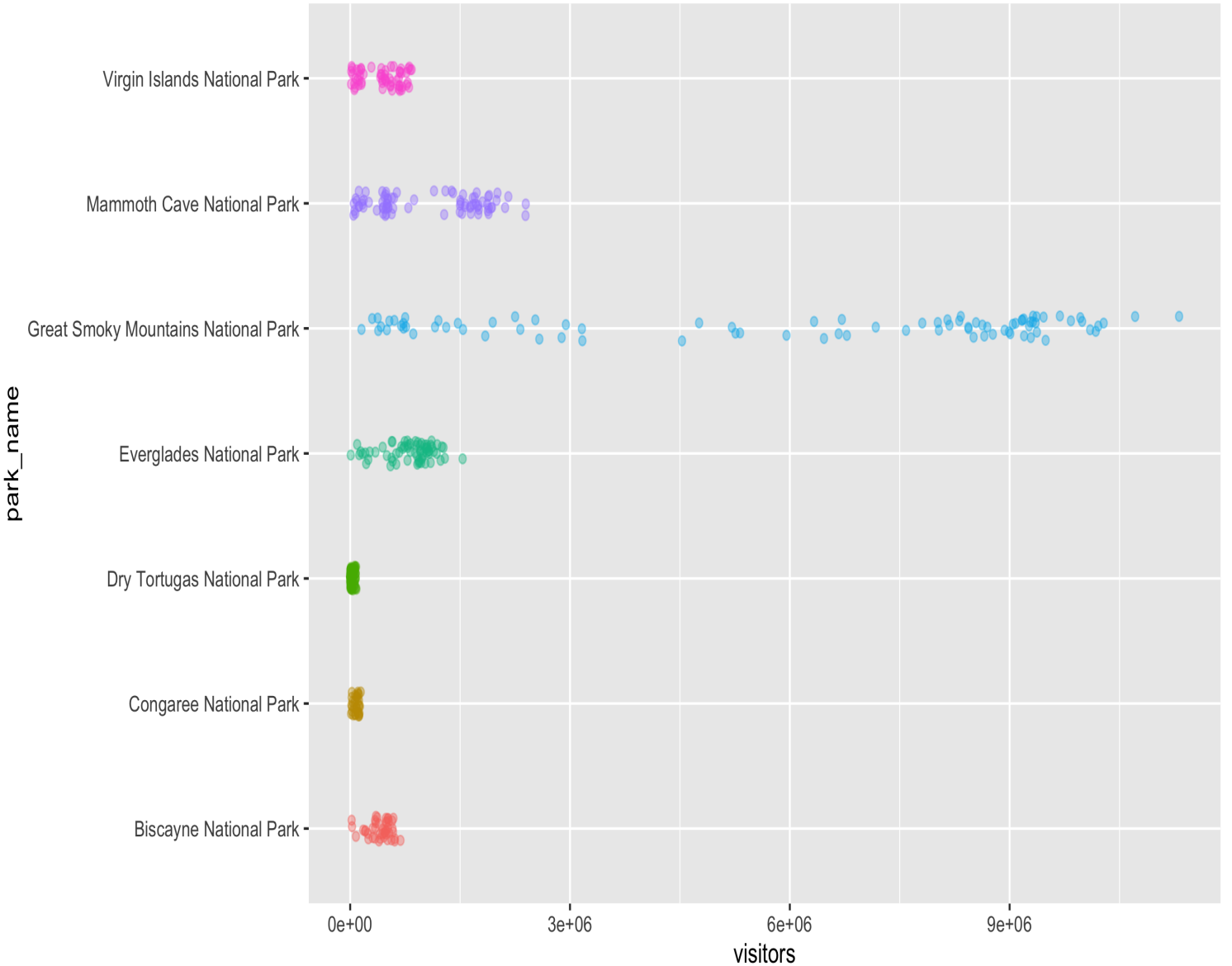
```
ggplot(data = se) +
     geom_point(aes(x = year, y = visitors, color = park_name)) +
     facet_wrap(~ state, scales = "free")
```



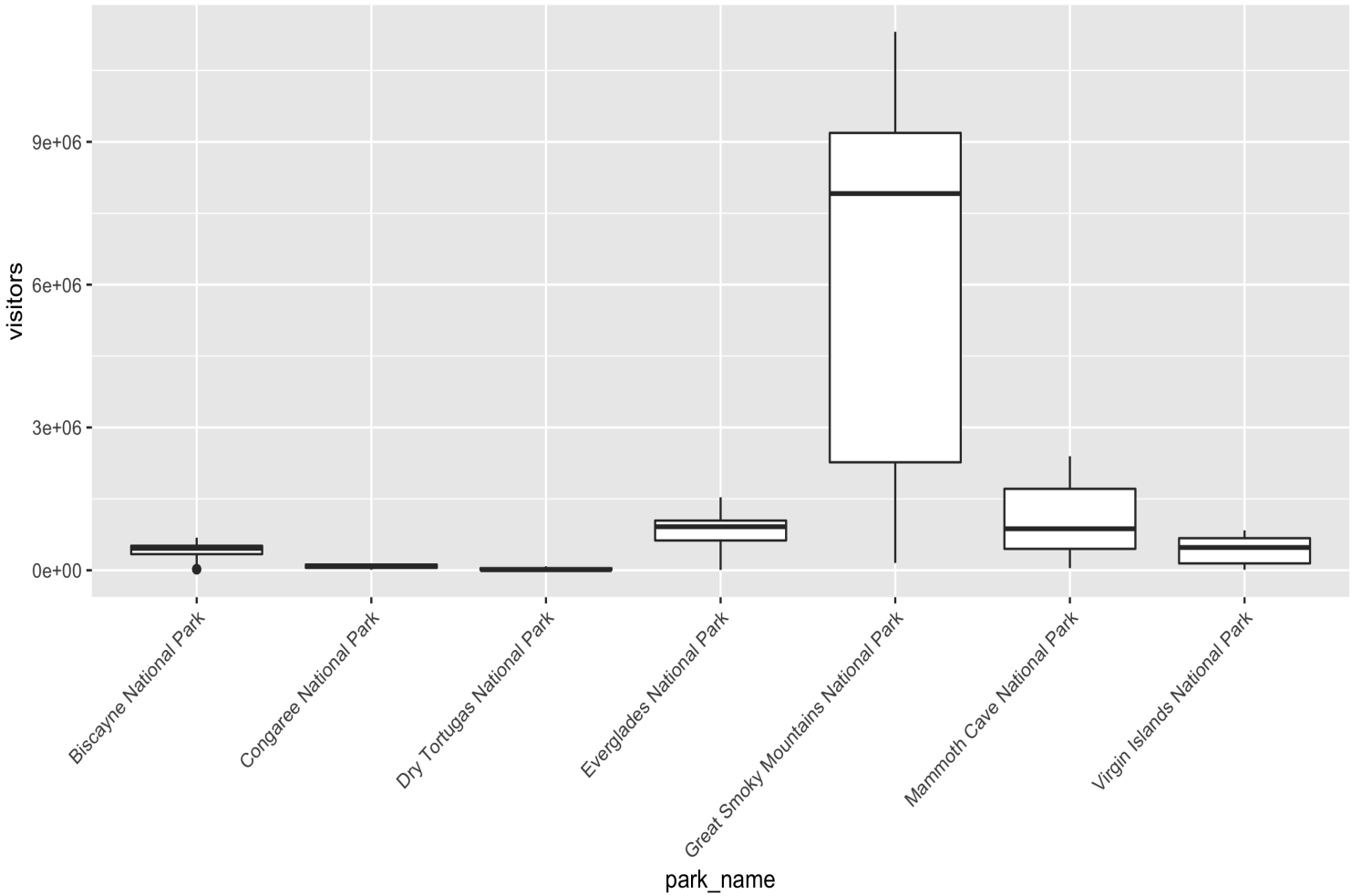
Geometric objects (geoms)

A **geom** is the geometrical object that a plot uses to represent data. People often describe plots by the type of geom that the plot uses. For example, bar charts use bar geoms, line charts use line geoms, boxplots use boxplot geoms, and so on. Scatterplots break the trend; they use the point geom. You can use different geoms to plot the same data. To change the geom in your plot, change the geom function that you add to `ggplot()`. Let's look at a few ways of viewing the distribution of annual visitation (`visitors`) for each park (`park_name`).

```
ggplot(data = se) +
  geom_jitter(aes(x = park_name, y = visitors, color = park_name),
              width = 0.1,
              alpha = 0.4) +
  coord_flip() +
  theme(legend.position = "none")
```

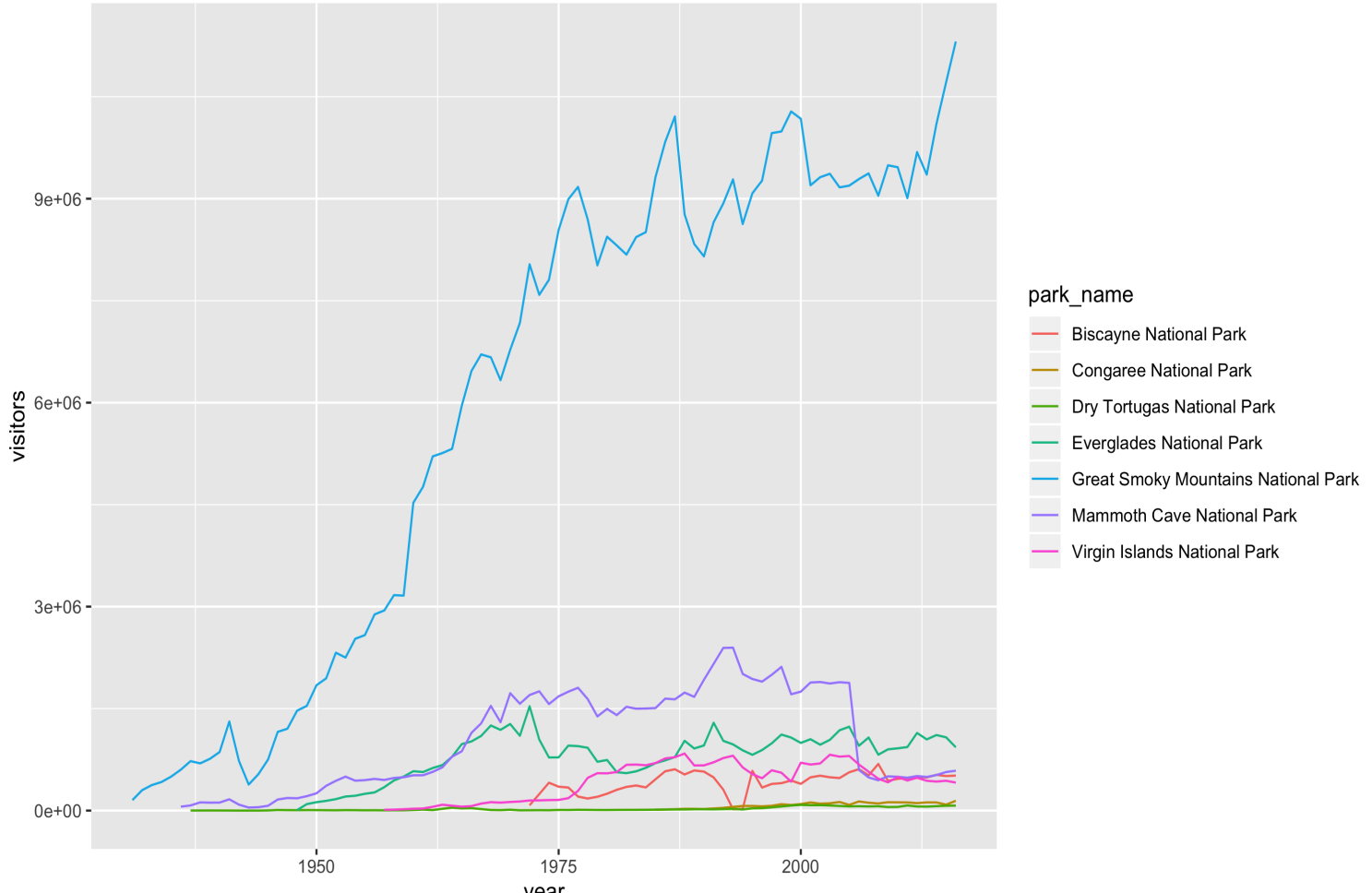

Virgin Islands National Park
Mammoth Cave National Park
Great Smoky Mountains National Park
Everglades National Park
Dry Tortugas National Park
Congaree National Park
Biscayne National Park
park_name
0e+00
3e+06
6e+06
9e+06
visitors

```
ggplot(se, aes(x = park_name, y = visitors)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



None of these are great for visualizing data over time. We can use `geom_line()` in the same way we used `geom_point`.

```
ggplot(se, aes(x = year, y = visitors, color = park_name)) +
   geom_line()
```

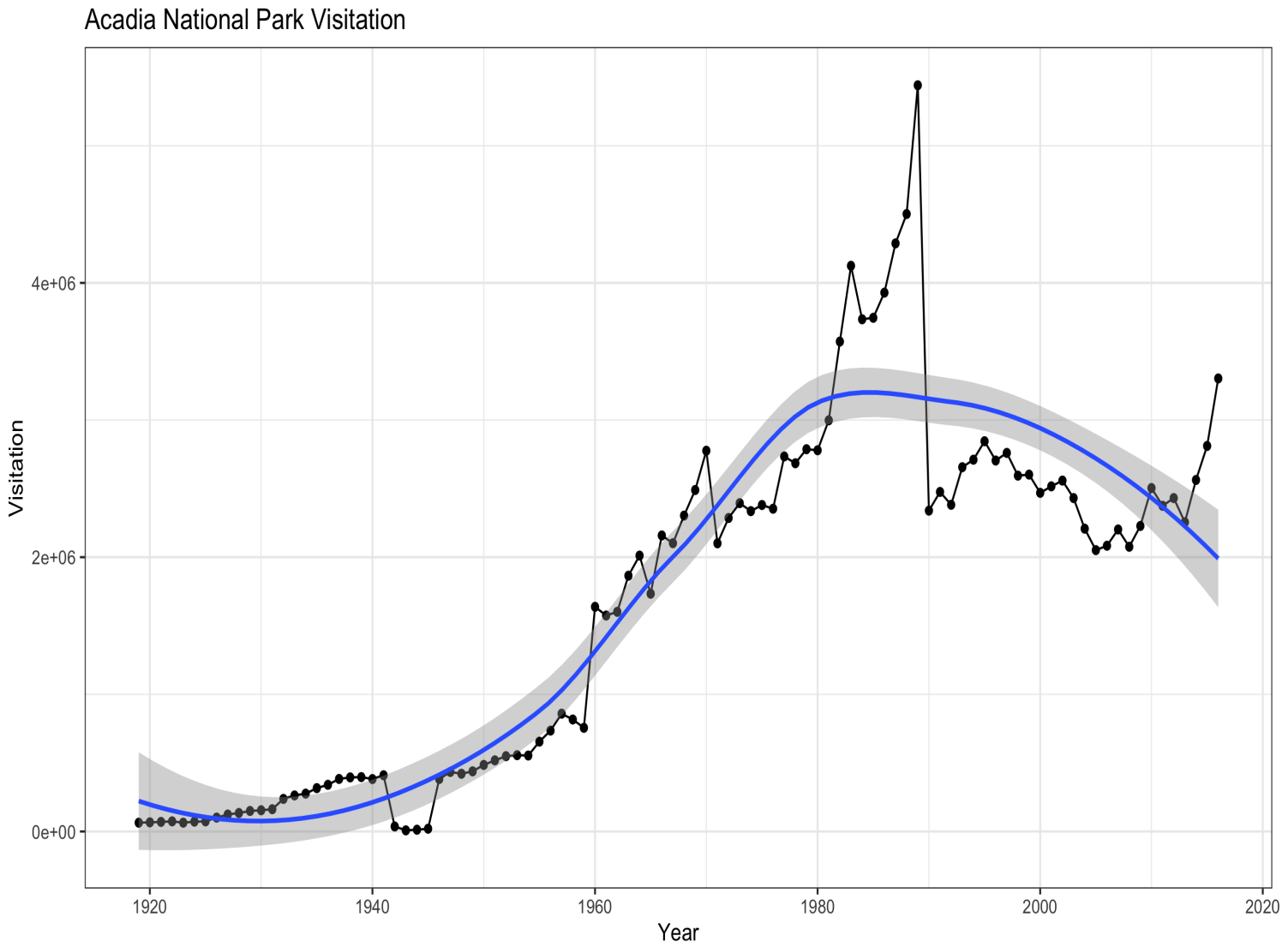


ggplot2 provides over 30 geoms, and extension packages provide even more (see https://www.ggplot2-exts.org for a sampling). The best way to get a comprehensive overview is the ggplot2 cheatsheet. To learn more about any single geom, use help: `?geom_smooth`.

To display multiple geoms in the same plot, add multiple geom functions to `ggplot()`:

`geom_smooth` allows you to view a smoothed mean of data. Here we look at the smooth mean of visitation over time to Acadia National Park:

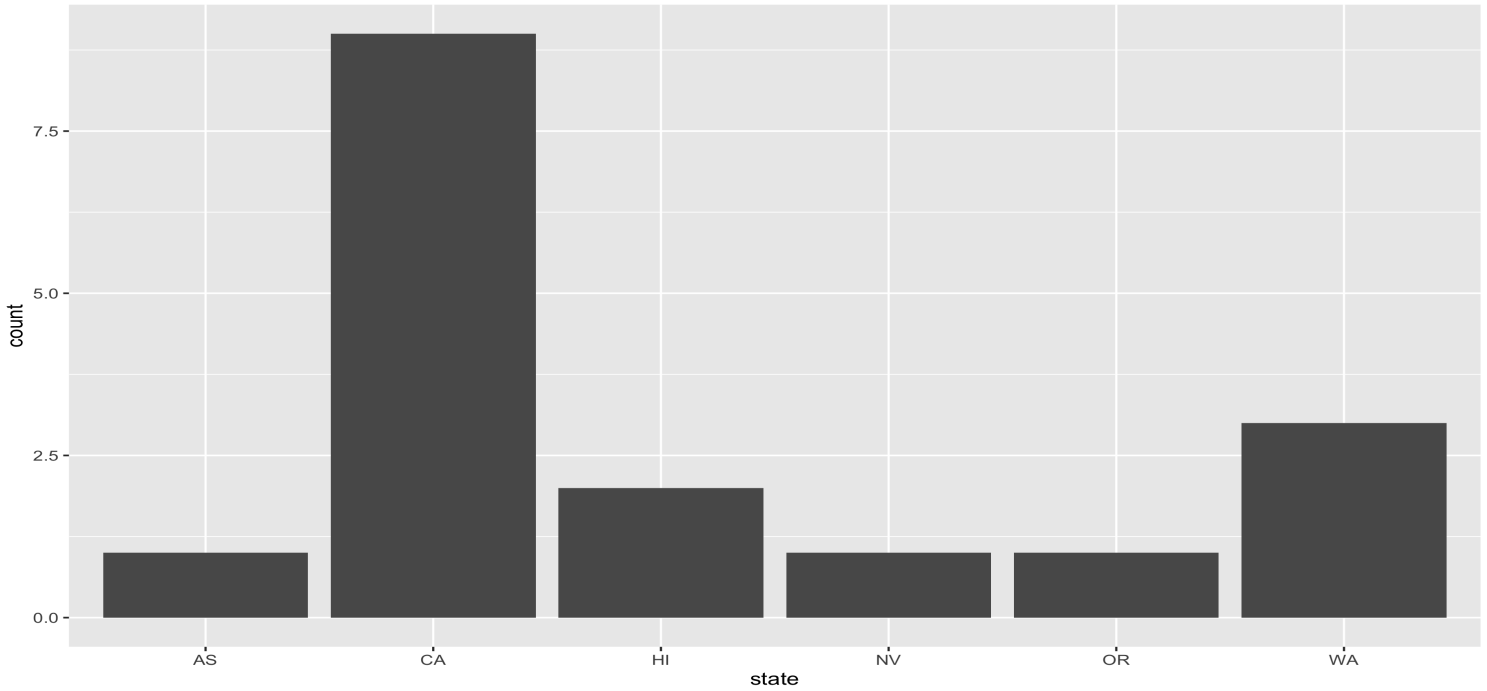
```
ggplot(data = acadia) +
  geom_point(aes(x = year, y = visitors)) +
  geom_line(aes(x = year, y = visitors)) +
  geom_smooth(aes(x = year, y = visitors)) +
  labs(title = "Acadia National Park Visitation",
       y = "Visitation",
       x = "Year") +
  theme_bw()
```


Acadia National Park Visitation
Visitation
4e+06
2e+06
0e+00
1920
1940
1960
1980
2000
2020
Year

Bar charts

Next, let's take a look at a bar chart. Bar charts seem simple, but they are interesting because they reveal something subtle about plots. Consider a basic bar chart, as drawn with geom_bar(). The following chart displays the total number of parks in each state within the Pacific West region.

```
ggplot(data = visit_16, aes(x = state)) +
  geom_bar()
```



On the x-axis, the chart displays state, a variable from `visit_16`. On the y-axis, it displays count, but count is not a variable in `visit_16`! Where does count come from? Many graphs, like scatterplots, plot the raw values of your dataset. Other graphs, like bar charts, calculate new values to plot:

- bar charts, histograms, and frequency polygons bin your data and then plot bin counts, the number of points that fall in each bin.
- smoothers fit a model to your data and then plot predictions from the model.
- boxplots compute a robust summary of the distribution and then display a specially formatted box.

The algorithm used to calculate new values for a graph is called a **stat**, short for statistical transformation.

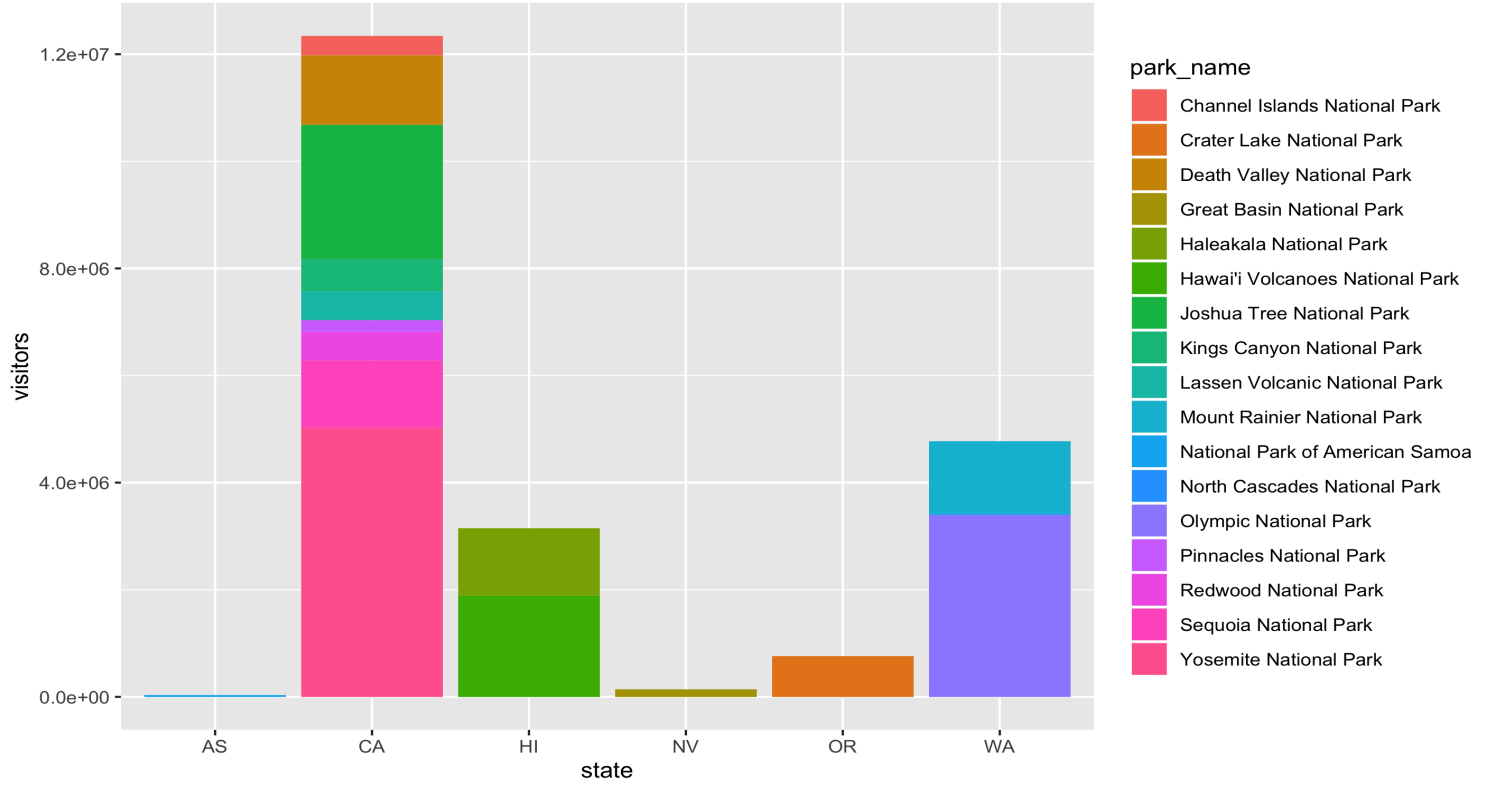
You can learn which stat a geom uses by inspecting the default value for the stat argument. For example, `?geom_bar` shows that the default value for stat is “count”, which means that `geom_bar()` uses `stat_count()`. `stat_count()` is documented on the same page as `geom_bar()`, and if you scroll down you can find a section called “Computed variables”. That describes how it computes two new variables: count and prop.

ggplot2 provides over 20 stats for you to use. Each stat is a function, so you can get help in the usual way, e.g. `?stat_bin`. To see a complete list of stats, try the ggplot2 cheatsheet.

Position adjustments

There's one more piece of magic associated with bar charts. You can colour a bar chart using either the `color` aesthetic, or, more usefully, `fill`:

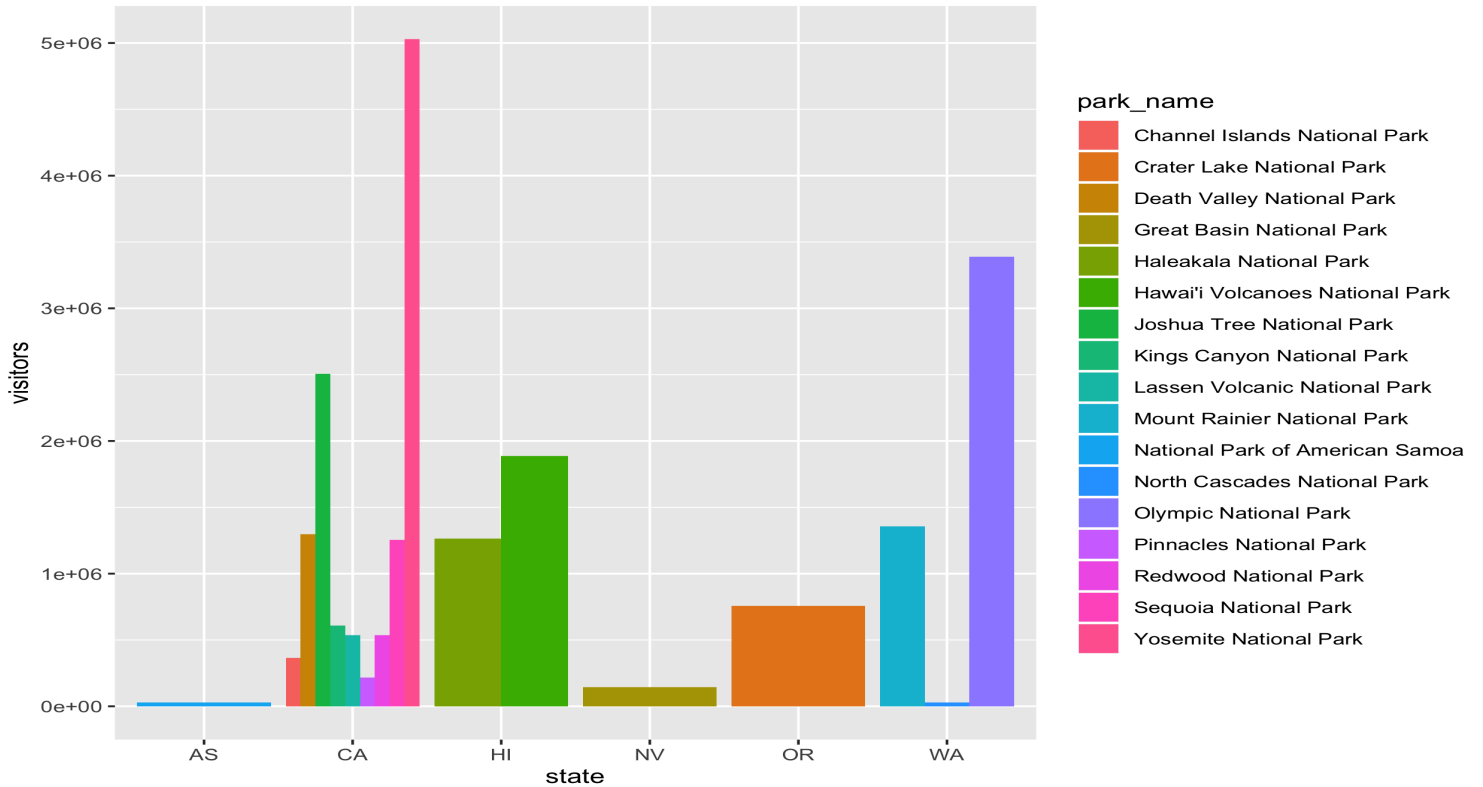
```
ggplot(data = visit_16, aes(x = state, y = visitors, fill = park_name
  geom_bar(stat = "identity")
```



The stacking is performed automatically by the **position adjustment** specified by the position argument. If you don't want a stacked bar chart, you can use dodge.

- `position = "dodge"` places overlapping objects directly beside one another. This makes it easier to compare individual values.

```
ggplot(data = visit_16, aes(x = state, y = visitors, fill = park_name
  geom_bar(stat = "identity", position = "dodge")
```



Arranging and exporting plots

After creating your plot, you can save it to a file in your favorite format. The Export tab in the **Plot** pane in RStudio will save your plots at low resolution, which will not be accepted by many journals and will not scale well for posters.

Instead, use the `ggsave()` function, which allows you easily change the dimension and resolution of your plot by adjusting the appropriate arguments (`width`, `height` and `dpi`):

```
my_plot <- ggplot(data = mass) +
      geom_bar(aes(x = type, fill = park_name)) +
  labs(x = "", y = "")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 7))
ggsave("name_of_file.png", my_plot, width = 15, height = 10)
```

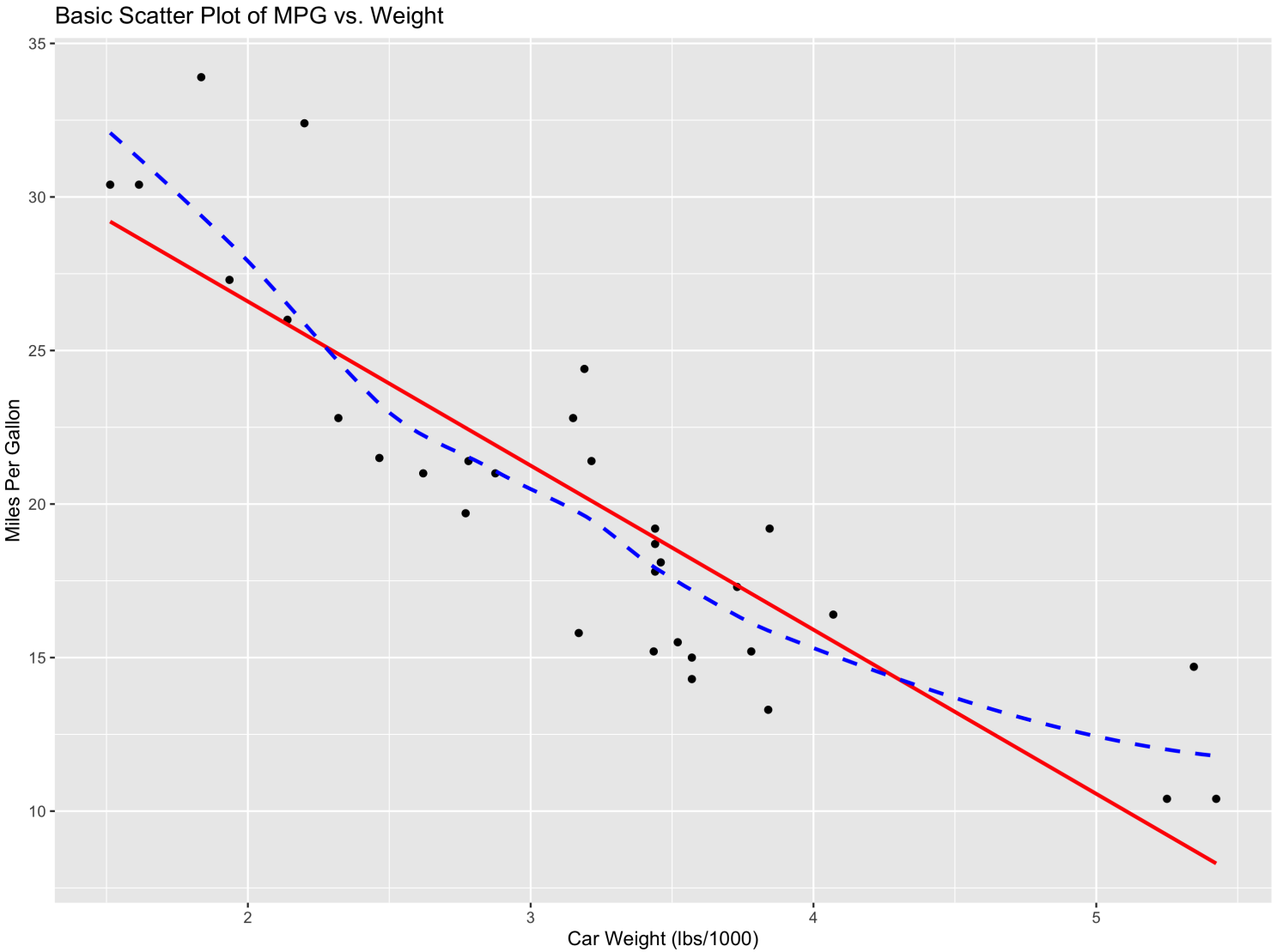
Note: The parameters `width` and `height` also determine the font size in the saved plot.

More advanced visualization methods

Scatter plots

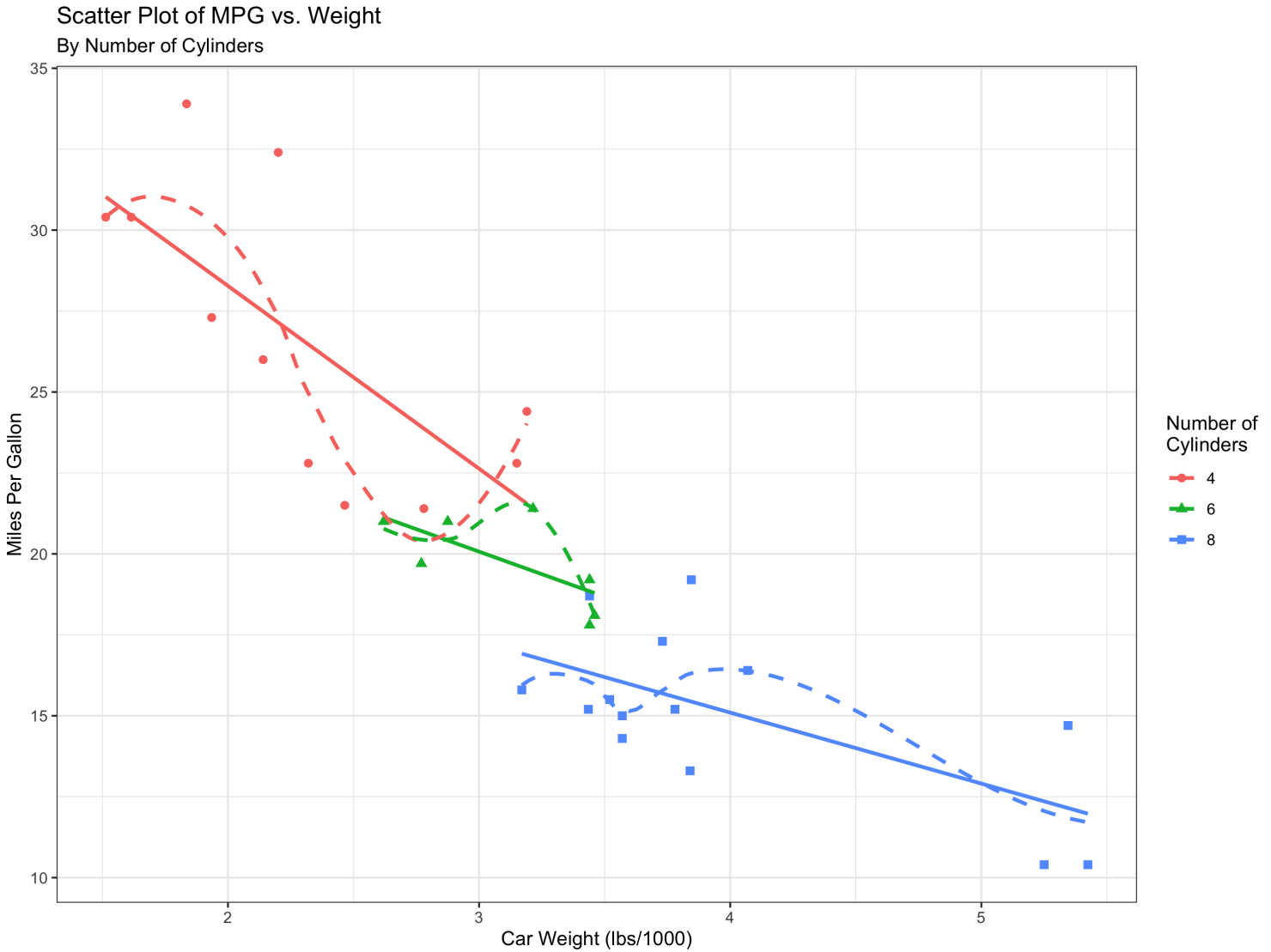
We'll start by visualizing the relationship between automobile weight and fuel efficiency.

```
data(mtcars) #1
mtcars %>% ggplot(aes(x=wt, y=mpg)) +
  geom_point() +
  geom_smooth(method="lm", se=FALSE, color="red") +
  geom_smooth(method="loess", se=FALSE,
    color="blue", linetype="dashed") +
    labs(title = "Basic Scatter Plot of MPG vs. Weight",
    x = "Car Weight (lbs/1000)", y = "Miles Per Gallon")
```


Basic Scatter Plot of MPG vs. Weight
Miles Per Gallon
35
30
25
20
15
10
2
3
4
5
Car Weight (lbs/1000)

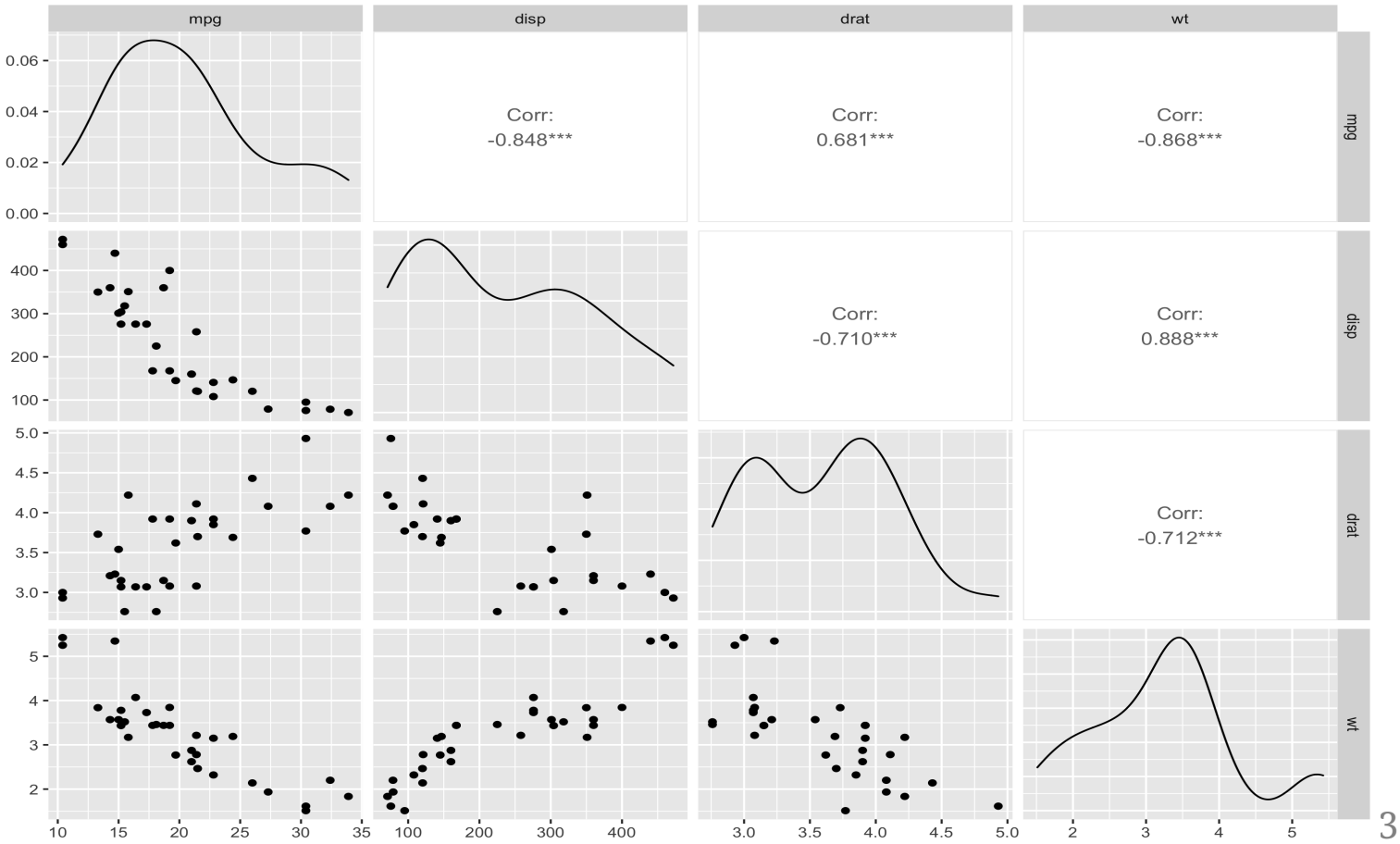
A scatter plot with separate best-fit lines

```
mtcars %>% ggplot(aes(x=wt, y=mpg, color=factor(cyl),
       shape=factor(cyl))) +
  geom_point(size=2) +
  geom_smooth(method="lm", se=FALSE) +
  geom_smooth(method="loess", se=FALSE, linetype="dashed") +
  labs(title = "Scatter Plot of MPG vs. Weight",
       subtitle = "By Number of Cylinders",
       x = "Car Weight (lbs/1000)", y = "Miles Per Gallon",
       color = "Number of \nCylinders",
       shape = "Number of \nCylinders") +
  theme_bw()
```


Scatter Plot of MPG vs. Weight
By Number of Cylinders
Miles Per Gallon
35
30
25
20
15
10
2
3
4
5
Car Weight (lbs/1000)
Number of Cylinders
4
6
8

Scatter-plot matrices

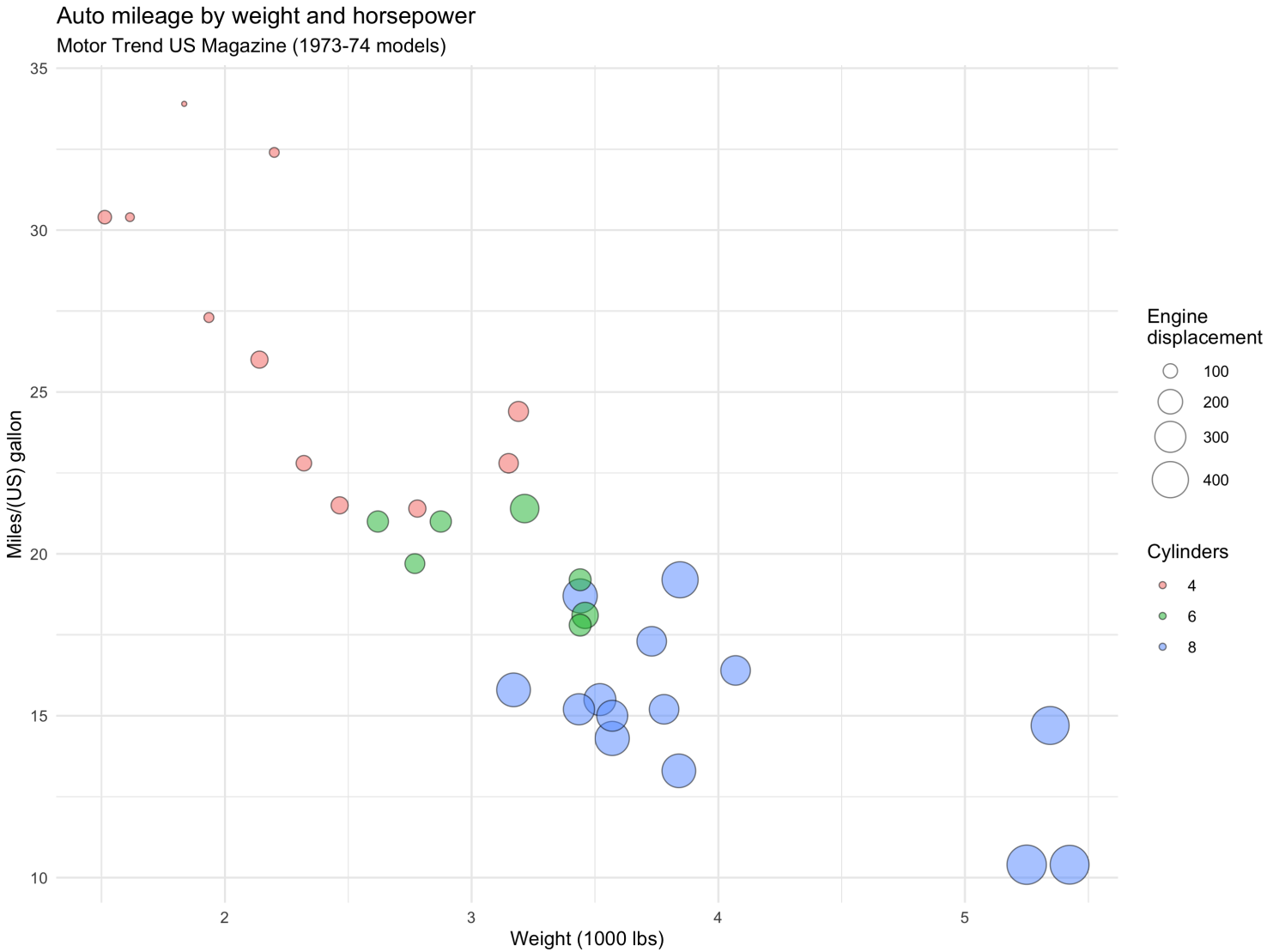
```
library(GGally)
ggpairs(mtcars[c("mpg","disp","drat", "wt")])
```



Bubble plots

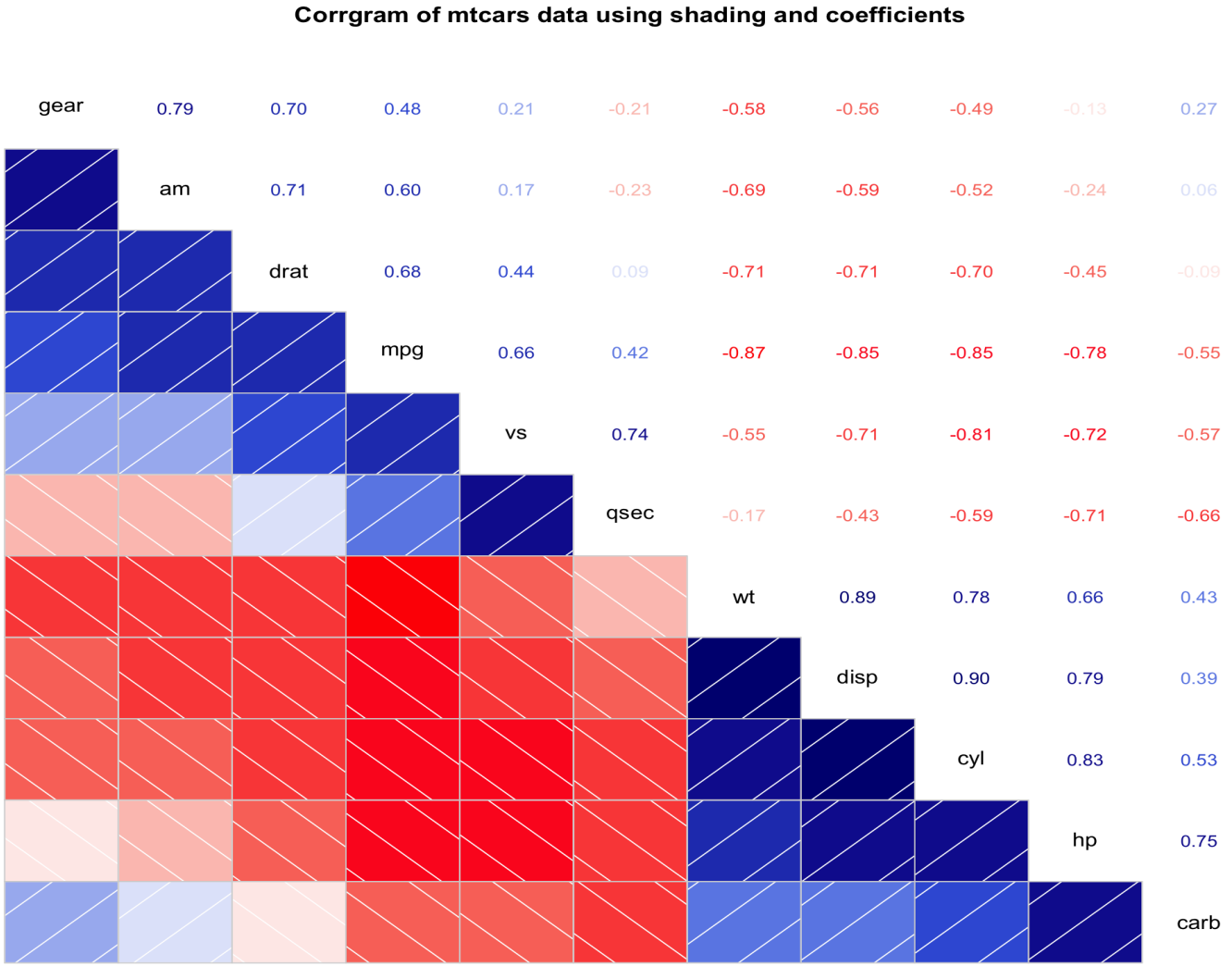
```
mtcars %>% ggplot(aes(x = wt, y = mpg, size = disp)) +
  geom_point() +
  labs(title="Bubble Plot with point size proportional to displacemer
    x="Weight of Car (lbs/1000)", y="Miles Per Gallon")
```

```
mtcars %>% ggplot(aes(x = wt, y = mpg, size = disp,
    fill=factor(cyl))) +
  geom_point(alpha = .5, color = "black", shape = 21) +
  scale_size_continuous(range = c(1, 10)) +
  labs(title = "Auto mileage by weight and horsepower",
    subtitle = "Motor Trend US Magazine (1973-74 models)",
    x = "Weight (1000 lbs)",
    y = "Miles/(US) gallon", size = "Engine\ndisplacement",
    fill = "Cylinders") +
  theme_minimal()
```


Auto mileage by weight and horsepower
Motor Trend US Magazine (1973-74 models)
Miles/(US) gallon
Weight (1000 lbs)
35
30
25
20
15
10
2
3
4
5
Engine displacement
100
200
300
400
Cylinders
4
6
8

Corrgram

```
library(corrgram)
corrgram(mtcars, order=TRUE, lower.panel=panel.shade,
         upper.panel=panel.cor,
  main="Corrgram of mtcars data using shading and coefficients")
```


Corrgram of mtcars data using shading and coefficients
gear 0.79 0.70 0.48 0.21 -0.21 -0.58 -0.56 -0.49 -0.13 0.27
am 0.71 0.60 0.17 -0.23 -0.69 -0.59 -0.52 -0.24 0.06
drat 0.68 0.44 0.09 -0.71 -0.71 -0.70 -0.45 -0.09
mpg 0.66 0.42 -0.87 -0.85 -0.85 -0.78 -0.55
vs 0.74 -0.55 -0.71 -0.81 -0.72 -0.57
qsec -0.17 -0.43 -0.59 -0.71 -0.66
wt 0.89 0.78 0.66 0.43
disp 0.90 0.79 0.39
cyl 0.83 0.53
hp 0.75
carb