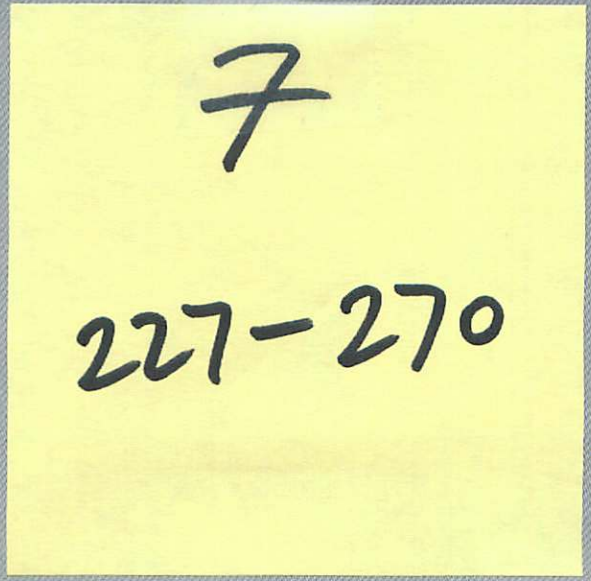
Spotting Differences 7

Sports commentators like to classify a select few athletes as superstars or as part of an elite group, while the rest are designated average or role players. These classifications usually aren't so much from sports statistics as they are from watching a lot of games. It's the know-it-when-I-see-it mentality. There's nothing wrong with this. The commentators (usually) know what they're talking about and are always considering the context of the numbers. It always makes me happy when a group of sports analysts look at performance metrics, and almost without fail someone will say, "You can't just look at the numbers. It's the intangibles that make so and so great." That's statistics right there.

Obviously this doesn't apply to just sports. Maybe you want to find the best restaurants in an area or identify loyal customers. Rather than categorizing units, you could look for someone or something that stands out from the rest. This chapter looks at how to spot groups within a population and across multiple criteria, and spot the outliers using common sense.



What to Look For

It's easy to compare across a single variable. One house has more square feet than another house, or one cat weighs more than another cat. Across two variables, it is a little more difficult, but it's still doable. The first house has more square feet, but the second house has more bathrooms. The first cat weighs more and has short hair, whereas the second cat weighs less and has long hair.

What if you have one hundred houses or one hundred cats to classify? What if you have more variables for each house, such as number of bedrooms, backyard size, and housing association fees? You end up with the number of units times the number of variables. Okay, now it is more tricky, and this is what we focus on.

Perhaps your data has a number of variables, but you want to classify or group units (for example, people or places) into categories and find the outliers or standouts. You want to look at each variable for differences, but you also want to see differences across all variables. Two basketball players could have completely different scoring averages, but they could be almost identical in rebounds, steals, and minutes played per game. You need to find differences but not forget the similarities and relationships, just like, oh yes, the sports commentators.

Comparing across Multiple Variables

One of the main challenges when dealing with multiple variables is to determine where to begin. You can look at so many variations and subsets that it can be overwhelming if you don't stop to think about what data you have. Sometimes, it's best to look at all the data at once, and interesting points could point you in the next interesting direction.

Getting Warmer

One of the most straightforward ways to visualize a table of data is to show it all at once. Instead of the numbers though, you can use colors to indicate values, as shown in Figure 7-1.

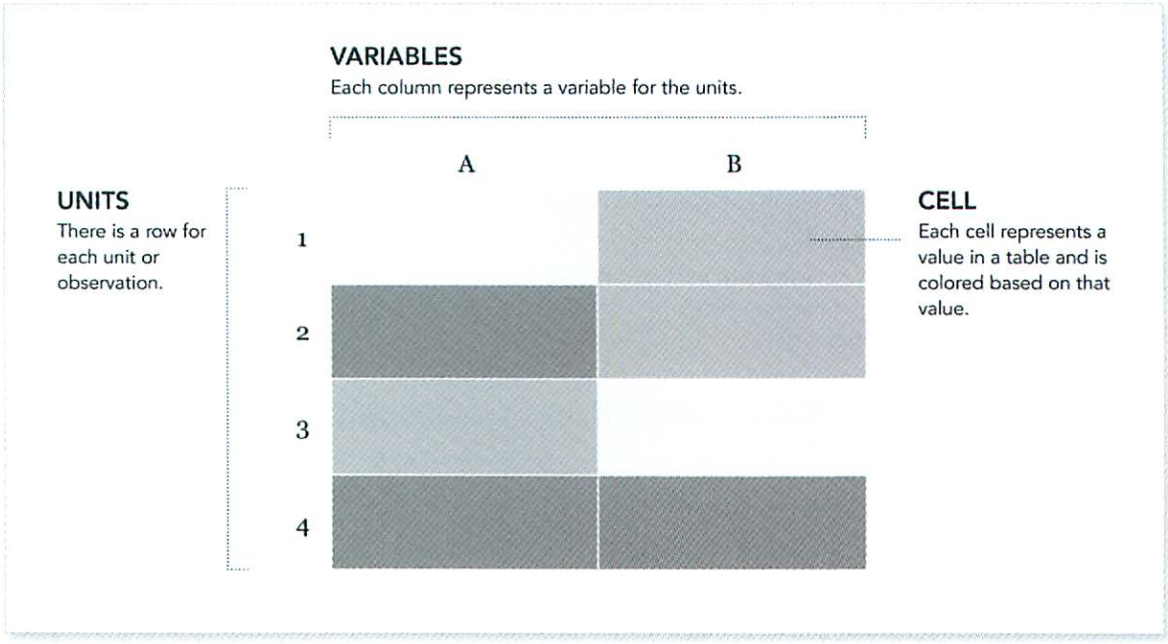


FIGURE 7-1 Heatmap framework

You end up with a grid the same size of the original data table, but you can easily find relatively high and low values based on color. Typically, dark colors mean greater values, and lighter colors represent lower values but that can easily change based on your application.

You also read the heatmap (or heat matrix) the same way you would a table. You can read a row left to right to see the values of all variables for a single unit, or you can see how all the units compare across a single variable.

This layout can still confuse you, especially if you have a large table of data, but with the right color scheme and some sorting, you can make a useful graphic.

CREATE A HEATMAP

It's easy to make heatmaps in R. There's a `heatmap()` function that does all the math work, which leaves you with picking colors best suited for your data and organizing labels so that they're still readable, even if you have a lot of rows and columns. In other words, R sets up the framework, and you handle the design. That should sound familiar by now.

In this example, take a look at NBA basketball statistics for 2008. You can download the data as a CSV file at `http://datasets.flowingdata.com/ppg2008.csv`. There are 22 columns, the first for player names, and the rest for stats such as points per game and field goal percentage. You can use `read.csv()` to load the data into R. Now look at the first five rows to get a sense of the data's structure (Figure 7-2).

```
bball <-
    read.csv("http://datasets.flowingdata.com/ppg2008.csv",
    header=TRUE)
bball[1:5,]
```

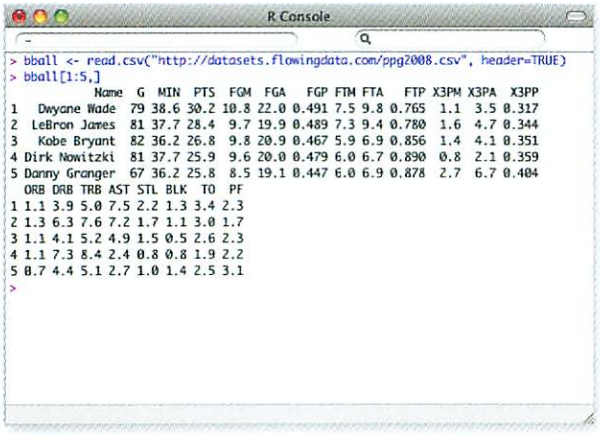


FIGURE 7-2 Structure of the first five rows of data

Players are currently sorted by points per game, greatest to least, but you could order players by any column, such as rebounds per game or field goal percentage, with `order()`.

```
bball_byfgp <- bball[order(bball$FGP, decreasing=TRUE),]
```

Now if you look at the first five rows of *bball_byfgp*, you see the list is led by Shaquille O'Neal, Dwight Howard, and Pau Gasol instead of Dwyane Wade, Lebron James, and Kobe Bryant. For this example, reverse the order on points per game.

```
bball <- bball[order(bball$PTS, decreasing=FALSE),]
```

TIP

The *decreasing* argument in `order()` specifies whether you want the data to be sorted in ascending or descending order.

As is, the column names match the CSV file's header. That's what you want. But you also want to name the rows by player name instead of row number, so shift the first column over to row names.

```
row.names(bball) <- bball$Name
bball <- bball[,2:20]
```

The first line changes row names to the first column in the data frame. The second line selects columns 2 through 20 and sets the subset of data back to *bball*.

The data also has to be in matrix format rather than a data frame. You'd get an error if you tried to use a data frame with the `heatmap()` function. Generally speaking, a data frame is like a collection of vectors where each column represents a different metric. Each column can have different formats such as numeric or a string. A matrix on the other hand is typically used to represent a two-dimensional space and the data type has to be uniform across all cells.

```
bball_matrix <- data.matrix(bball)
```

The data is ordered how you want it and formatted how you need it to be, so you can plug it into `heatmap()` to reap the rewards. By setting the *scale* argument to "column," you tell R to use the minimum and maximum of each column to determine color gradients instead of the minimum and maximum of the entire matrix.

```
bball_heatmap <- heatmap(bball_matrix, Rowv=NA,
    Colv=NA, col = cm.colors(256), scale="column", margins=c(5,10))
```

Your result should resemble Figure 7-3. Using `cm.colors()`, you specified a color range from cyan to magenta. The function creates a vector of hexadecimal colors with a cyan-to-magenta range by default with *n* shades in between (in this case, 256). So notice the third column, which is for points per game, starts at magenta, indicating the highest values for Dwyane Wade and Lebron James, and then shifts toward a darker cyan hue to the bottom for Allen Iverson and Nate Robinson. You can also quickly find other magenta spots representing leading rebounder Dwight Howard or assist leader Chris Paul.

TIP

A lot of visualization involves gathering and preparing data. Rarely, do you get data exactly how you need it, so you should expect to do some data munging before the visuals.

FIGURE 7-3 Default heatmap ordered by points per game

Maybe you want a different color scheme. Just change the *col* argument, which is `cm.colors(256)` in the line of code you just executed. Type **`?cm.colors`** for help on what colors R offers. For example, you could use more heat-looking colors, as shown in Figure 7-4.

```
bball_heatmap <- heatmap(bball_matrix,
    Rowv=NA, Colv=NA, col = heat.colors(256), scale="column",
    margins=c(5,10))
```

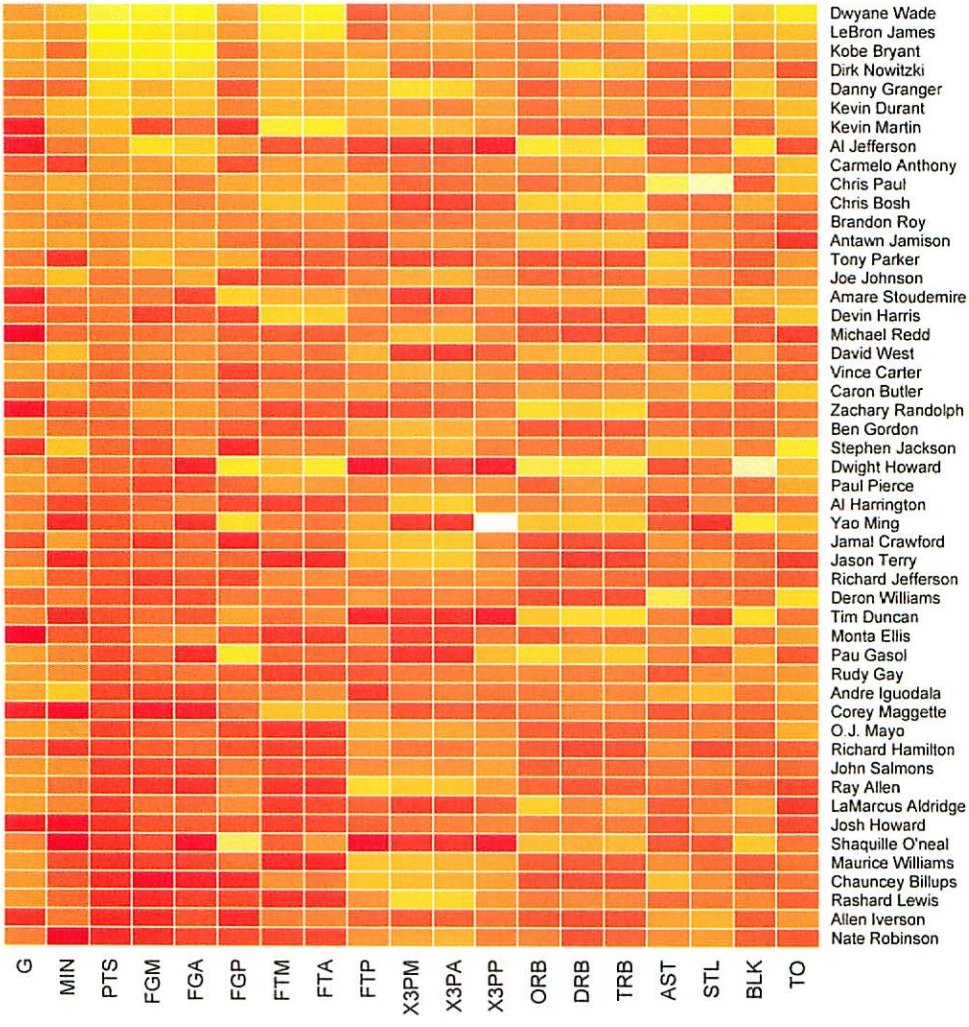


FIGURE 7-4 Heatmap with a red-yellow color scale

If you typed `cm.colors(10)` in the R console, you'd get an array of ten colors that range from cyan to magenta. Then `heatmap()` automatically chooses the color that corresponds to each value based on a linear scale.

```
[1] "#80FFFFFF" "#99FFFFFF" "#B3FFFFFF" "#CCFFFFFF" "#E6FFFFFF"
[6] "#FFE6FFFF" "#FFCCFFFF" "#FFB3FFFF" "#FF99FFFF" "#FF80FFFF"
```

website for colors

This is great, because you can easily create your own color scale. For example, you could go to 0to255.com and pick out the base color and go from there. Figure 7-5 shows a gradient with a red base. You can pick a handful of colors, from light to dark, and then easily plug them into `heatmap()`, as shown in Figure 7-6. Instead of using R to create a vector of colors, you define your own in the *red_colors* variable.

```
red_colors <- c("#ffd3cd", "#ffc4bc", "#ffb5ab",
    "#ffa69a", "#ff9789", "#ff8978", "#ff7a67", "#ff6b56",
    "#ff5c45", "#ff4d34")
bball_heatmap <- heatmap(bball_matrix, Rowv=NA,
    Colv=NA, col = red_colors, scale="column", margins=c(5,10))
```

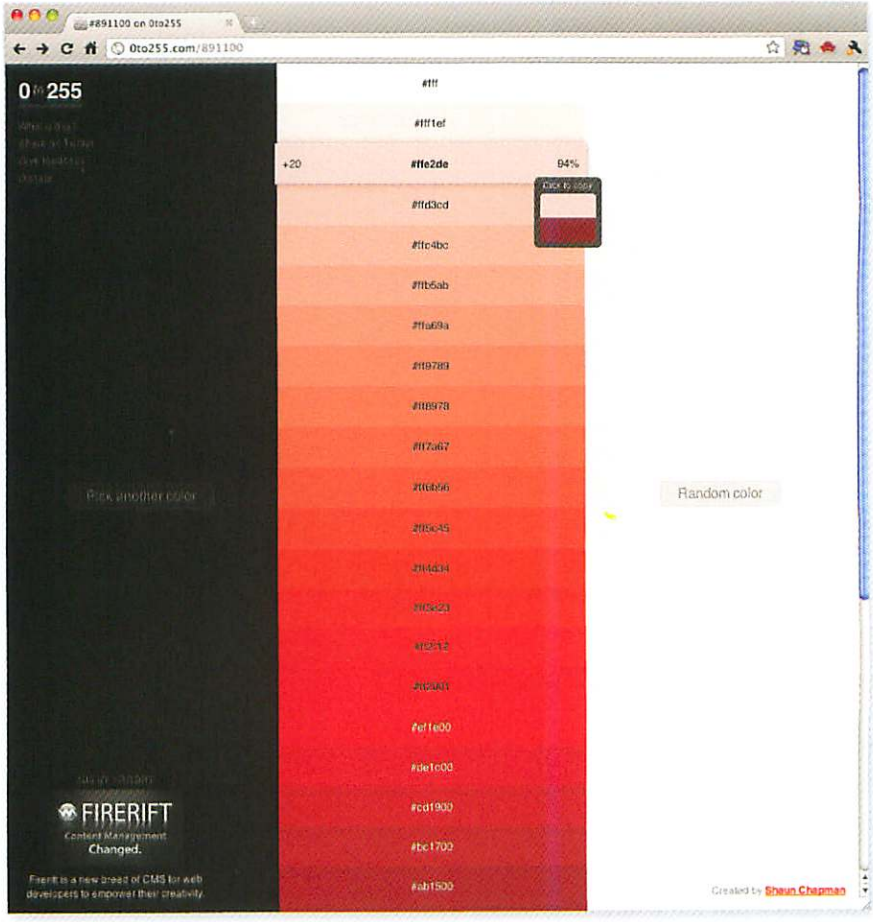


FIGURE 7-5 Red gradient from 0to255.com

TIP

Choose your colors wisely because they also set the tone for the context of your story. For example, if you deal with a somber topic, it's probably better to stay with more neutral, muted tones, whereas you can use vibrant colors for a more uplifting or casual topic.

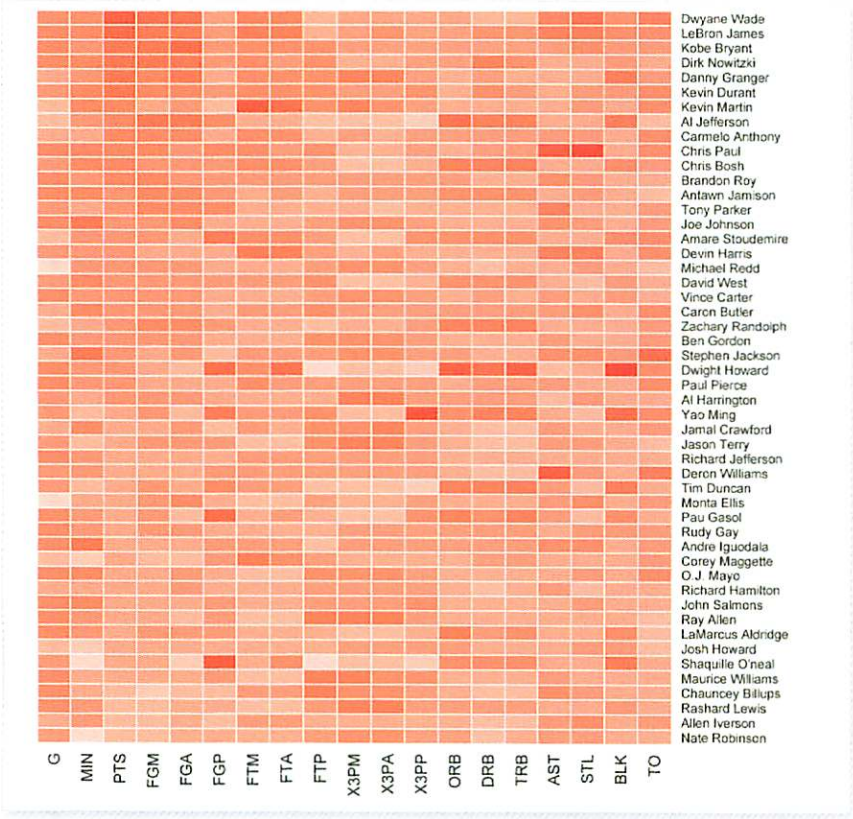


FIGURE 7-6 Heatmap using custom red color scale

If you don't want to pick your own colors, you can also use the RColorBrewer package. The package is not installed by default, so you need to download and install it via the Package Installer if you haven't already. ColorBrewer was designed by cartographer Cynthia Brewer and was originally intended for maps, but it can help you create data graphics in general. You can choose from a variety of options, such as a sequential or divergent color palette and number of shades. For the purposes of this example, go with a simple blue palette. Enter **?brewer.pal** in the R console for more options—it's fun to play with. Assuming you installed RColorBrewer, enter the following for a heatmap using a blue palette with nine shades. The result is shown in Figure 7-7.

```
library(RColorBrewer)
bball_heatmap <- heatmap(bball_matrix, Rowv=NA,
    Colv=NA, col = brewer.pal(9, "Blues"),
    scale="column", margins=c(5,10))
```

▶ Visit the interactive version of Colorbrewer at `http://colorbrewer2.com`. You can select options from drop-down menus to see how color schemes look in a sample map.

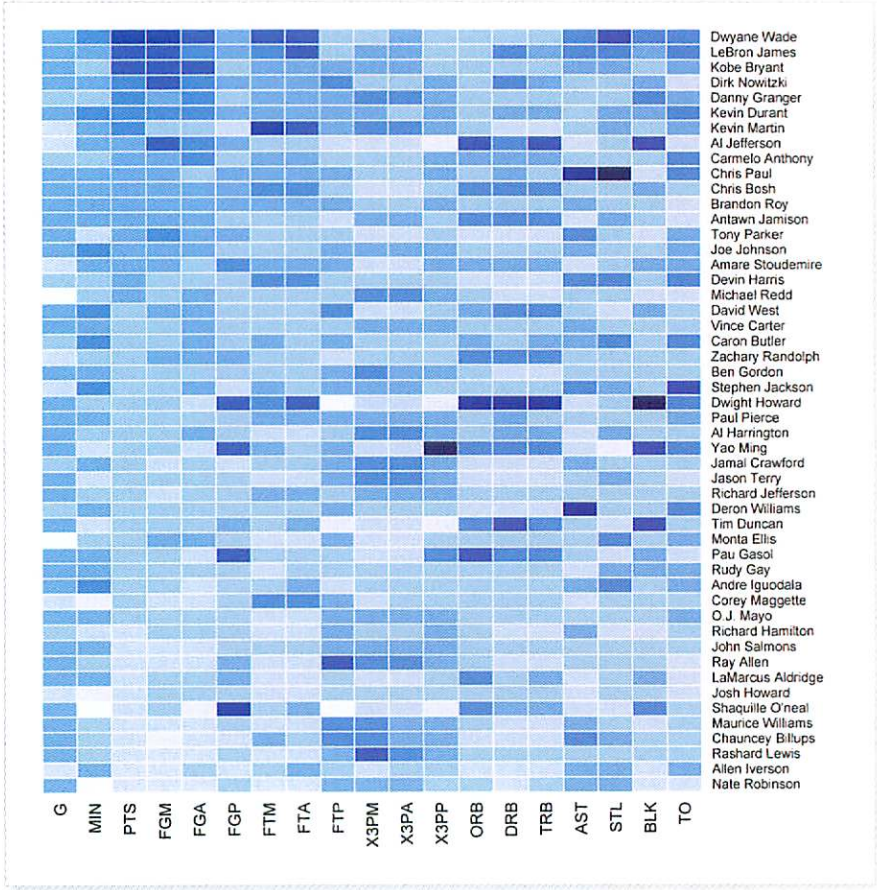


FIGURE 7-7 Heatmap using RColorBrewer for color palette

Check out Figure 7-7, which you can bring into Illustrator to spruce up. The graphic doesn't need a ton of edits, but you can make the labels more readable and soften the colors so that the graphic is easier to scan.

Addressing the former, it'd be better if the labels were the full descriptions. As a basketball fan, I know what each abbreviation stands for, but someone not so familiar with the sport would be confused. As for the latter, you can tone down the contrast by using transparency, available in the Color Window in Illustrator. Cell borders can also provide more definition to each cell so that the graphic is easier to scan left to right and top to bottom. Figure 7-8 shows the finished graphic.

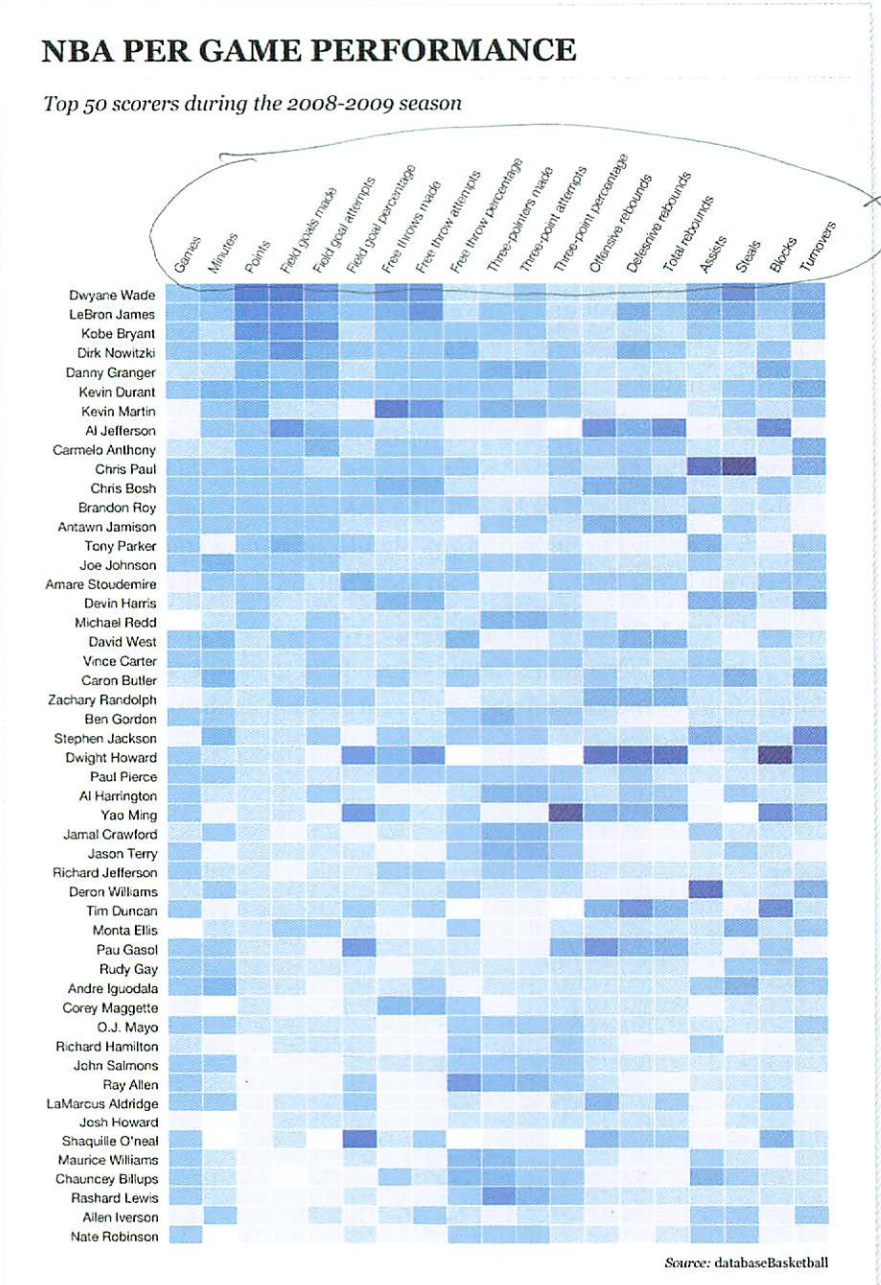


FIGURE 7-8 Heatmap showing NBA per game performance for the top 50 scorers during the 2008–2009 season

▶ Mike Bostock ported this example to Protovis, which you can find in the Protovis examples section. The aesthetic is the same, but it has the added bonus of tooltips as you mouse over each cell.

See It in His Face

The good thing about a heatmap is that it enables you to see all your data at once; however, the focus is on individual points. You can easily spot highs and lows for points or rebounds per game, but it's more challenging to compare one player to another.

Often you want to see each unit as a whole instead of split up by several metrics. Chernoff Faces is one way to do this; however the method isn't an exact one, and it's possible a general audience might become confused. That said, Chernoff Faces can be useful from time to time, it's good data nerd fun, which makes it totally worth it.

The point of Chernoff Faces is to display multiple variables at once by positioning parts of the human face, such as ears, hair, eyes, and nose, based on numbers in a dataset (Figure 7-9). The assumption is that you can read people's faces easily in real life, so you should recognize small differences when they represent data. That's a big assumption, but roll with it.

As you see in the following example, larger values stand out in the form of big hair or big eyes, whereas smaller values tend to shrink facial features. In addition to size, you can also adjust features such as the curve of the lips or shape of the face.

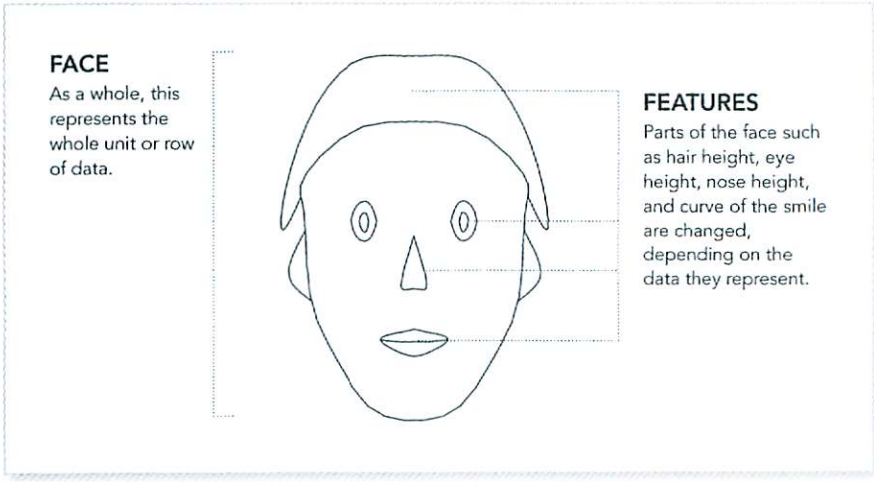


FIGURE 7-9 Chernoff Faces framework

CREATE CHERNOFF FACES

Go back to the basketball data, which represents the top 50 scorers in the NBA, during the 2008–2009 season. There will be one face per player. Don't worry—you don't have to create each face manually. The `aplpack` in R provides a `faces()` function to help get you to where you want.

If you haven't already, go ahead and install `aplpack` with `install.packages()` or via the Package Installer. The package name stands for "another plotting package" in case you were wondering, and it was designed by Hans Peter Wolf. When installed, the package is usually automatically loaded, but if not, you should do that, too.

```
library(aplpack)
```

You should have also already loaded the basketball data while creating a heatmap. If not, again use `read.csv()` to load the data directly from a URL.

```
bball <- read.csv("http://datasets.flowingdata.com/ppg2008.csv",
header=TRUE)
```

After you load the package and data, it's straightforward to make Chernoff Faces with the `faces()` function, as shown in Figure 7-10.

```
faces(bball[,2:16], ncolors=0)
```

Your dataset has 20 variables, plus player names; however, the `faces()` function offered by the `aplpack` enables only a maximum of 15 variables, because there are only so many facial features you can change. This is why you subset the data on columns 2 to 16.

What does each face represent? The `faces()` function changes features in the following order, matching the order of the data columns.

1. Height of face
2. Width of face
3. Shape of face
4. Height of mouth
5. Width of mouth
6. Curve of smile
7. Height of eyes
8. Width of eyes
9. Height of hair
10. Width of hair
11. Styling of hair
12. Height of nose
13. Width of nose
14. Width of ears
15. Height of ears

NOTE

The newest version of the `aplpack` lets you add color with the `faces()` function. In this example, you set *`ncolors`* to 0 to use only black and white. See **`?faces`** to see how you can use color vectors in the same way you used them with the previous heatmap example.

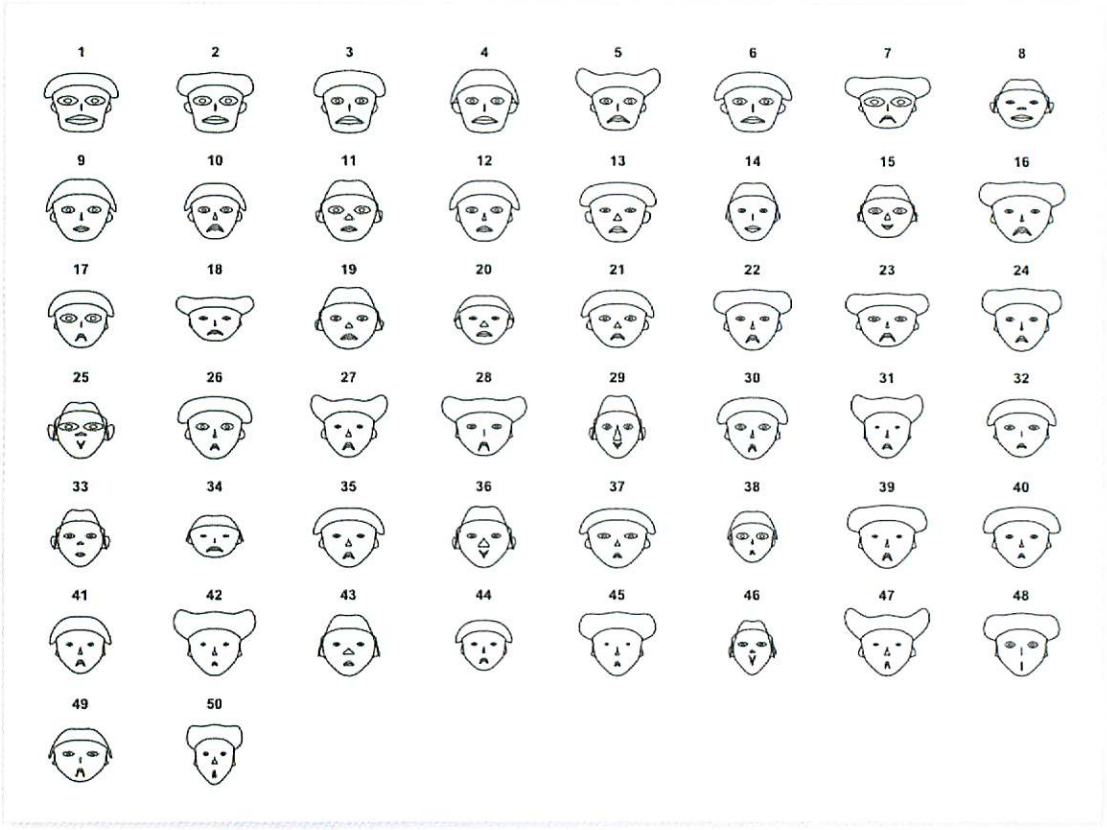


FIGURE 7-10 Default Chernoff Faces

TIP

When you have a lot of individuals, it can be useful to cluster by category so that the faces are easier to scan. For this example, you could separate faces by position: guards, forwards, and centers.

So for example, the height of a face represents the number of games played, and the height of a mouth represents field goals made per game. This is kind of useless as it is because you don't have any names to the faces, but you can see that the first few have more well-rounded gameplay than the others, whereas player 7, for instance, has relatively wide hair, which corresponds to three-pointers made.

Use the *labels* argument in `faces()` to add names to the faces, as shown in Figure 7-11.

```
faces(bball[,2:16], labels=bball$Name)
```

That's better. Now you can see which face corresponds to which player. To identify the point guards, you can start with say, Chris Paul, and look for similar faces such as the one for Devin Harris or Deron Williams. Chauncey Billups, in the bottom-right corner is also a point guard, but the

face looks different than the others. The hair is higher and mouth width is narrow, which corresponds to high free throw percentage and field goal attempts, respectively.

To make the graphic more readable, you can add some more spacing between rows, and at the least, provide a description of what feature provides what, as shown in Figure 7-12. Normally, I'd use a graphical legend, but we used every facial feature, so it's a challenge to provide that many pointers on a single face.

Again, the usefulness of Chernoff Faces can vary by dataset and audience, so you can decide whether you want to use the method. One thing though, is that people who aren't familiar with Chernoff Faces, tend to take the faces somewhat literally, as if the face that represents Shaquille O'Neal is actually supposed to look like the player. Most of you know that O'Neal is one of the physically biggest players of all time.

TIP

White space in your graphics can make them more readable, especially when there is a lot to look at and evaluate.

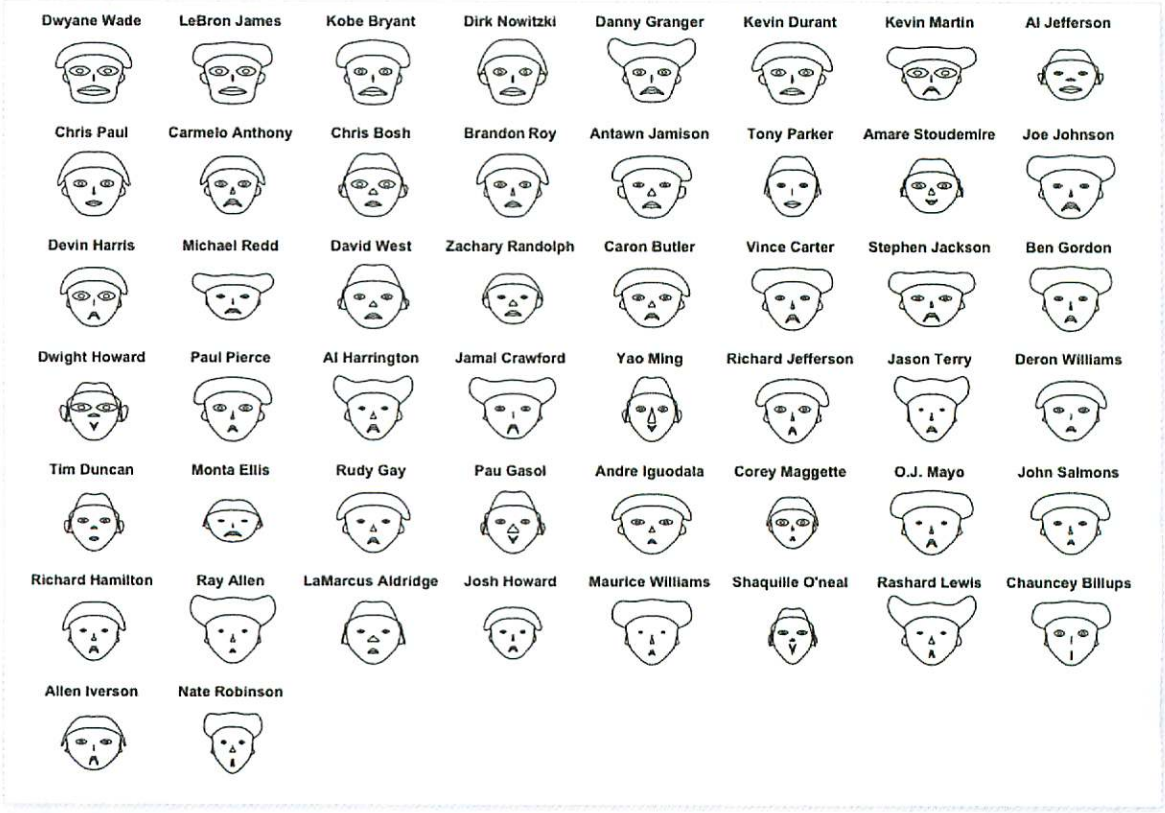


FIGURE 7-11 Chernoff Faces with names for players

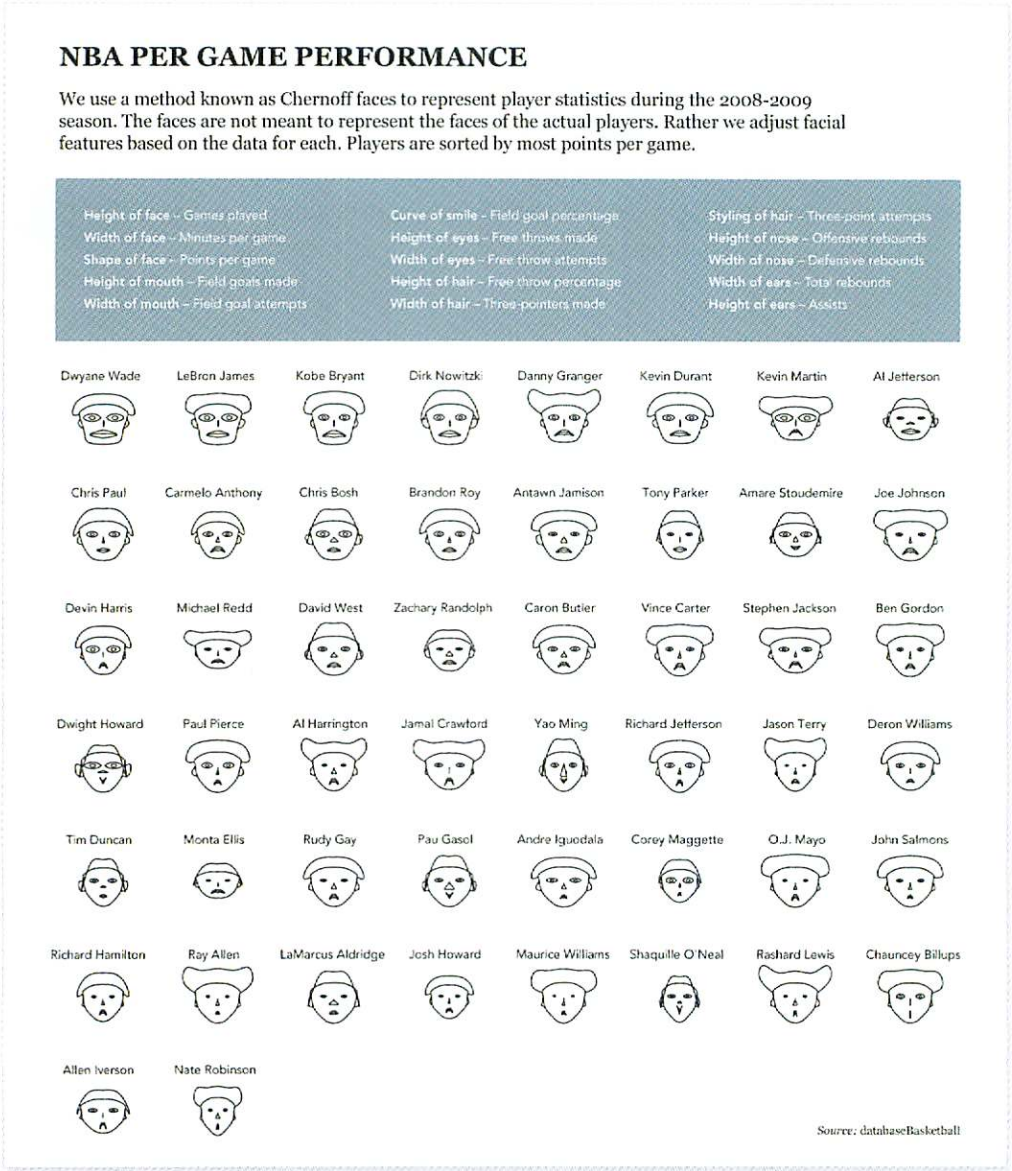


FIGURE 7-12 Chernoff Faces for top NBA scorers during the 2008–2009 season

Along the same lines, I designed a graphic for crime in the United States (Figure 7-13) using Chernoff Faces, and someone actually commented that it was racist because of how the face looked for states with high crime

rates. That never crossed my mind because changing a facial feature was like changing the length of a bar on a graph for me, but hey, it's something to think about.

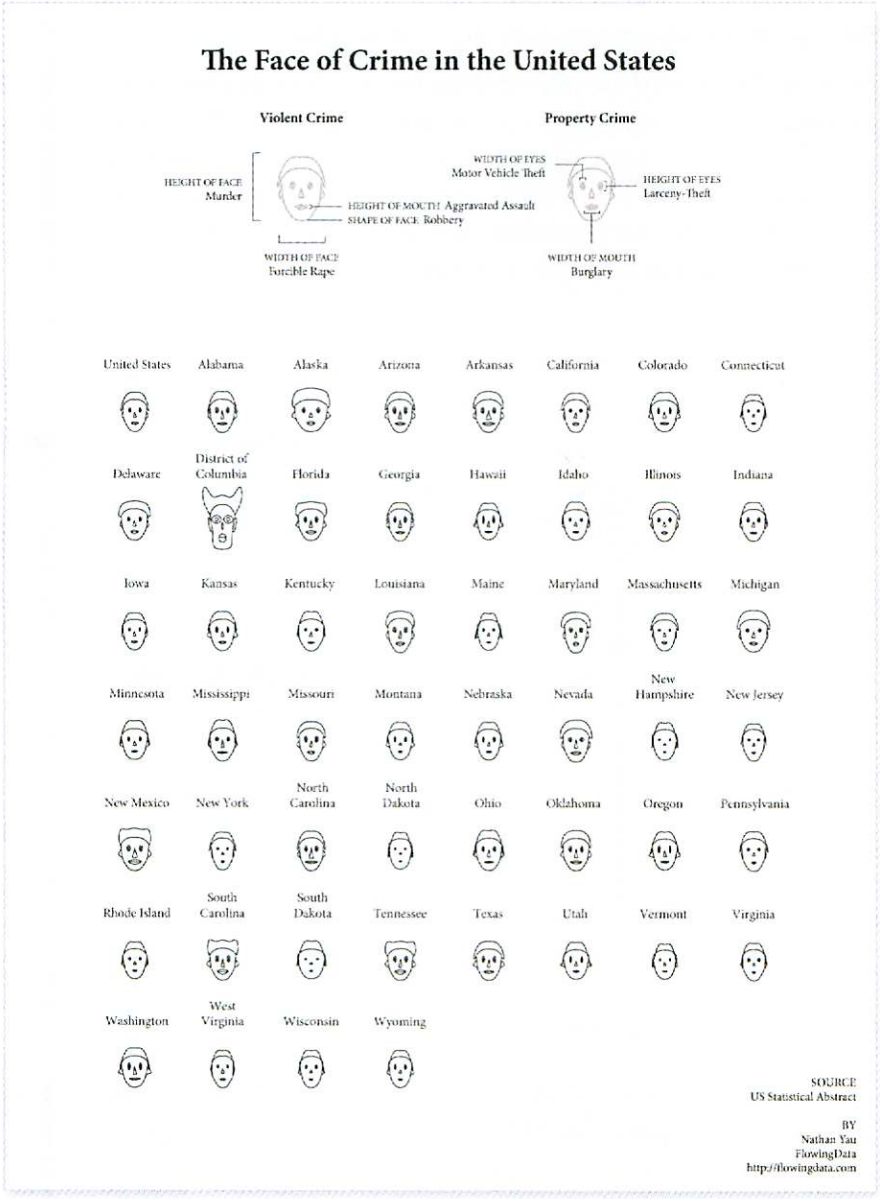


FIGURE 7-13 Crime in the United States represented by a face for each state

TIP

Put yourself in the mind of a reader as you design your graphics. They won't always be as familiar with visualization methods as you are or know what you know about the data, so it's up to you, the storyteller, to explain.

Starry Night

Instead of using faces to show multivariate data, you can use the same idea but abstract on it by using a different shape. Instead of changing facial features, you can modify the shape to match data values. This is the idea for star charts, also known as radar or spider charts.

As shown in Figure 7-14, you can draw several axes, one for each variable, starting from the middle and equally spaced in a circle. The center is the minimum value for each variable, and the ends represent the maximums. So if you draw a chart for a single unit, start at a variable and draw a connecting line to the corresponding spot on the next axis. Then you end up with something that looks like a star (or a radar or spider web).

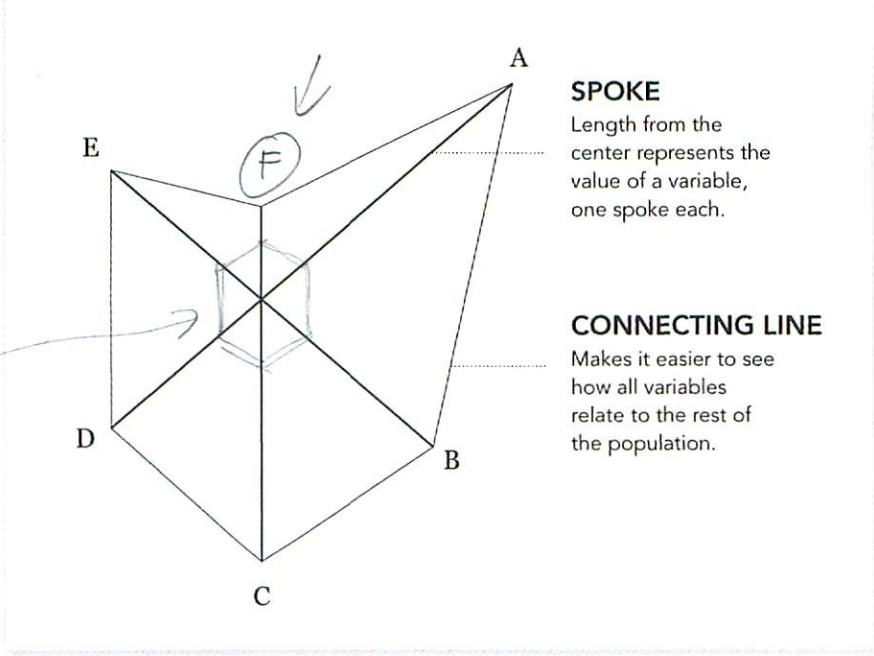


FIGURE 7-14 Star chart framework

You can represent several units on a single chart, but it'll become useless in a hurry, which makes for a poorly told story. So stick to separate star charts and compare.

CREATE STAR CHARTS

Now use the same crime data used for Figure 7-13 to see if it makes any difference to use these star charts. First things first. Load the data into R.

```
crime <- read.csv("http://datasets.flowingdata.com/
crimeRatesByState-formatted.csv")
```

Now it's as straightforward to make star charts as it was to make Chernoff Faces. Use the `stars()` function, which comes packaged with base R.

```
stars(crime)
```

The default charts are shown in Figure 7-15. Hopefully, these won't offend anyone. Of course, you still need state labels, but you also need a key to tell you which dimension is which. A certain order is followed, like with `faces()`, but you don't know where the first variable starts. So take care of both in one swoop. Notice that you can change to first column to row names just like you did with the heatmap. You can also set `flip.labels` to *FALSE*, because you don't want the labels to alternate heights. Figure 7-16 shows the results.

```
row.names(crime) <- crime$state
crime <- crime[,2:7]
stars(crime, flip.labels=FALSE, key.loc = c(15, 1.5))
```

It's relatively easy to spot the differences and similarities now. In the Chernoff version, the District of Columbia looked like a wild-eyed clown compared to all the other states, but with the star version, you see that yes, it does have high rates of crime in some categories, but relatively lower rates of forcible rape and burglary. It's also easy to find the states with relatively low crime rates such as New Hampshire and Rhode Island. Then there are states such as North Carolina that are high in just a single category.

For this dataset, I'm satisfied with this format, but there are two variations that you might want to try with your own data. The first restricts all data to the top half of the circle, as shown in Figure 7-17.

```
stars(crime, flip.labels=FALSE, key.loc = c(15, 1.5), full=FALSE)
```

The second variation uses the length of the segments instead of placement of points, as shown in Figure 7-18. These are actually Nightingale charts (also known as polar area diagrams) more than they are star charts, but there you go. If you do go with this option, you might want to try a different color scheme other than this crazy default one.

```
stars(crime, flip.labels=FALSE, key.loc = c(15, 1.5), draw.segments=TRUE)
```

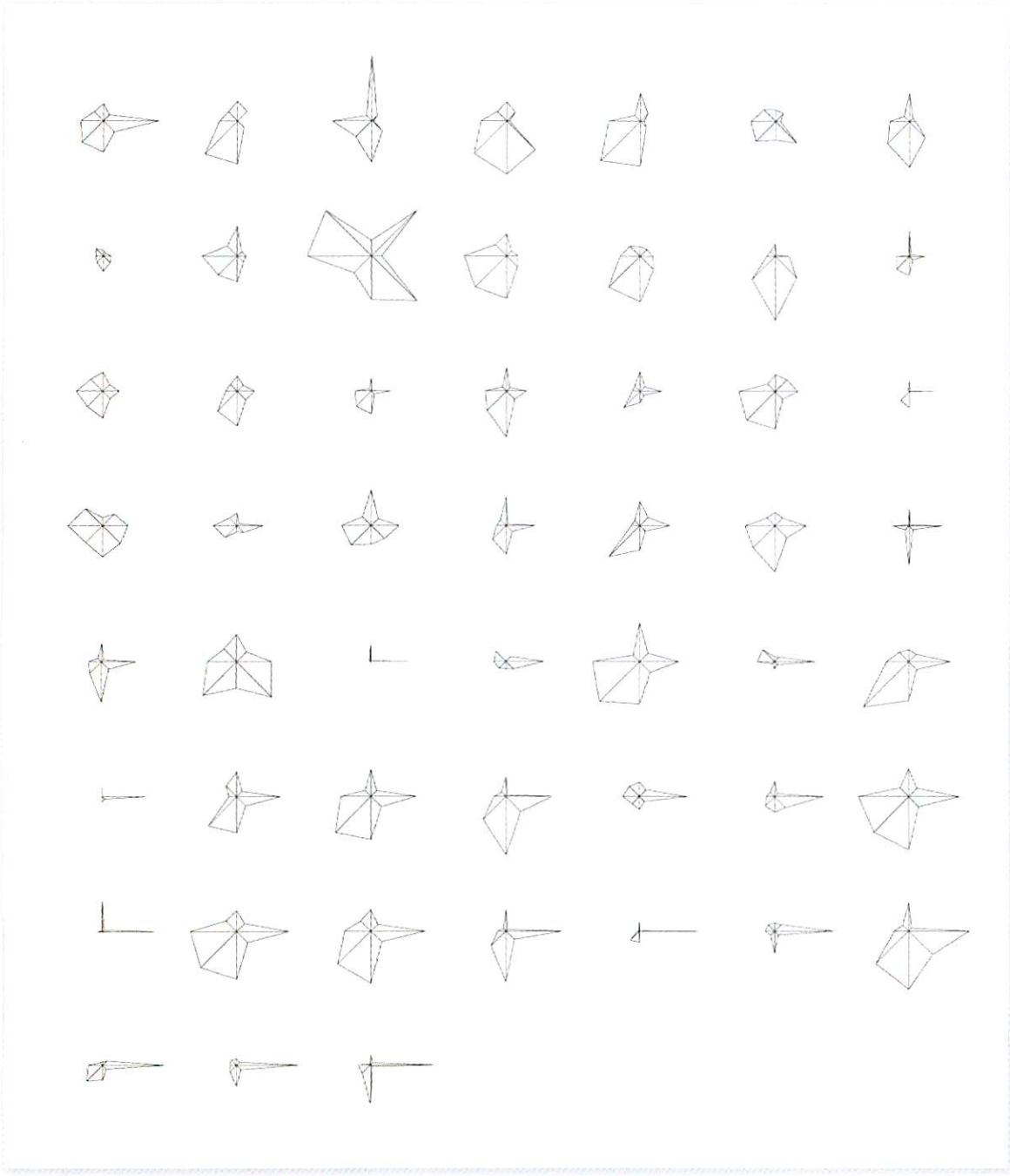


FIGURE 7-15 Default star charts showing crime by state

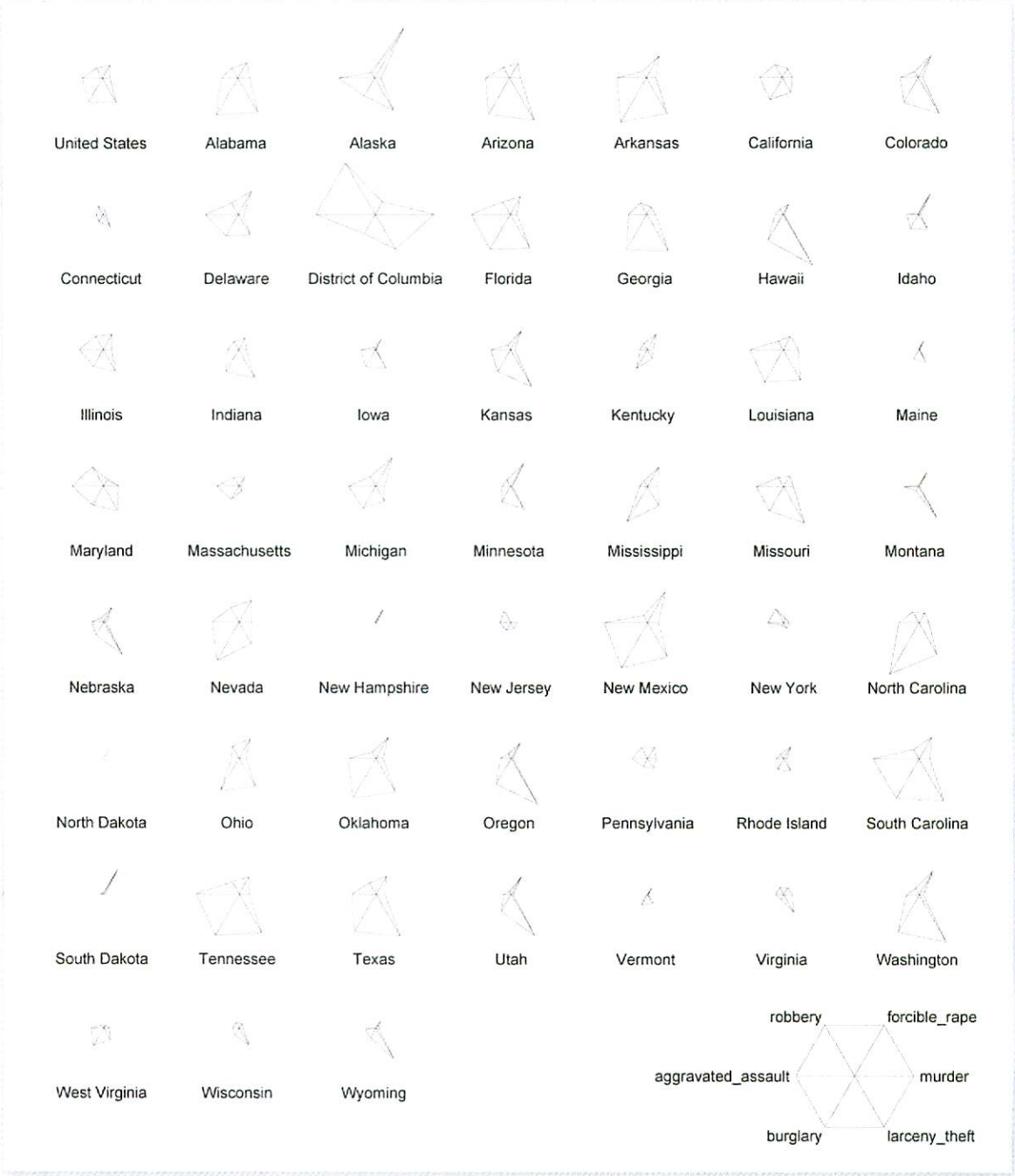


FIGURE 7-16 Star charts with labels and key

FIGURE 7-17 Star charts restricted to top half of circle

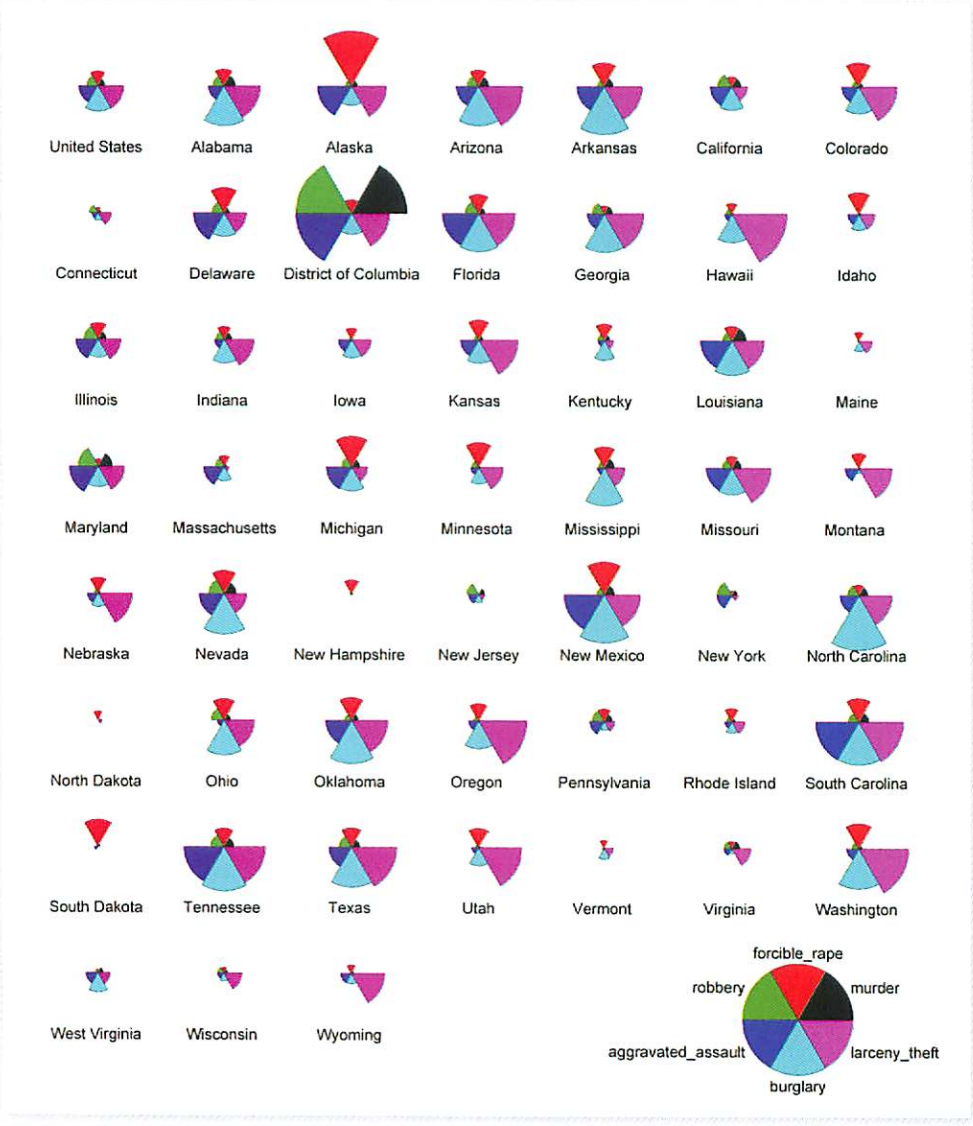


FIGURE 7-18 Crime displayed as Nightingale charts

Like I said though, I'm good with the original format in Figure 7-16, so you can take that into Illustrator to do some cleanup. It doesn't need a whole lot of modification. More white space between the rows could make the labels less ambiguous, and you can place the key on top so that readers know what they're getting into (Figure 7-19). Other than that, it's good to go.

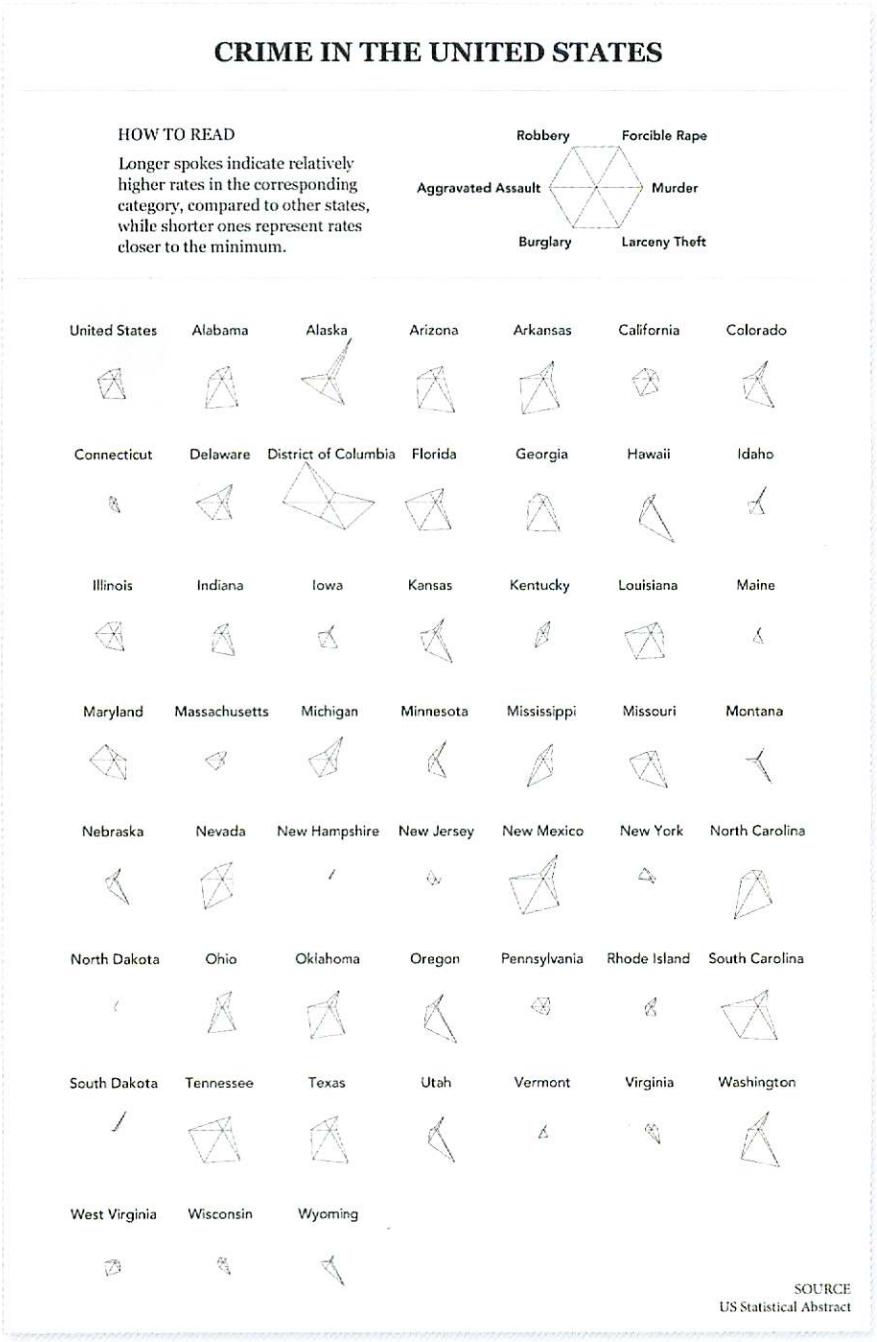


FIGURE 7-19 Series of star charts showing crime by state

Running in Parallel

Although star charts and Chernoff Faces can make it easier to spot units that are different from the rest of the pack, it's a challenge to identify groups or how variables could be related. Parallel coordinates, which were invented in 1885 by Maurice d'Ocagne, can help with this.

As shown in Figure 7-20, you place multiple axes parallel to each other. The top of each axis represents a variable's maximum, and the bottom represents the minimum. For each unit, a line is drawn from left to right, moving up and down, depending on the unit's values.

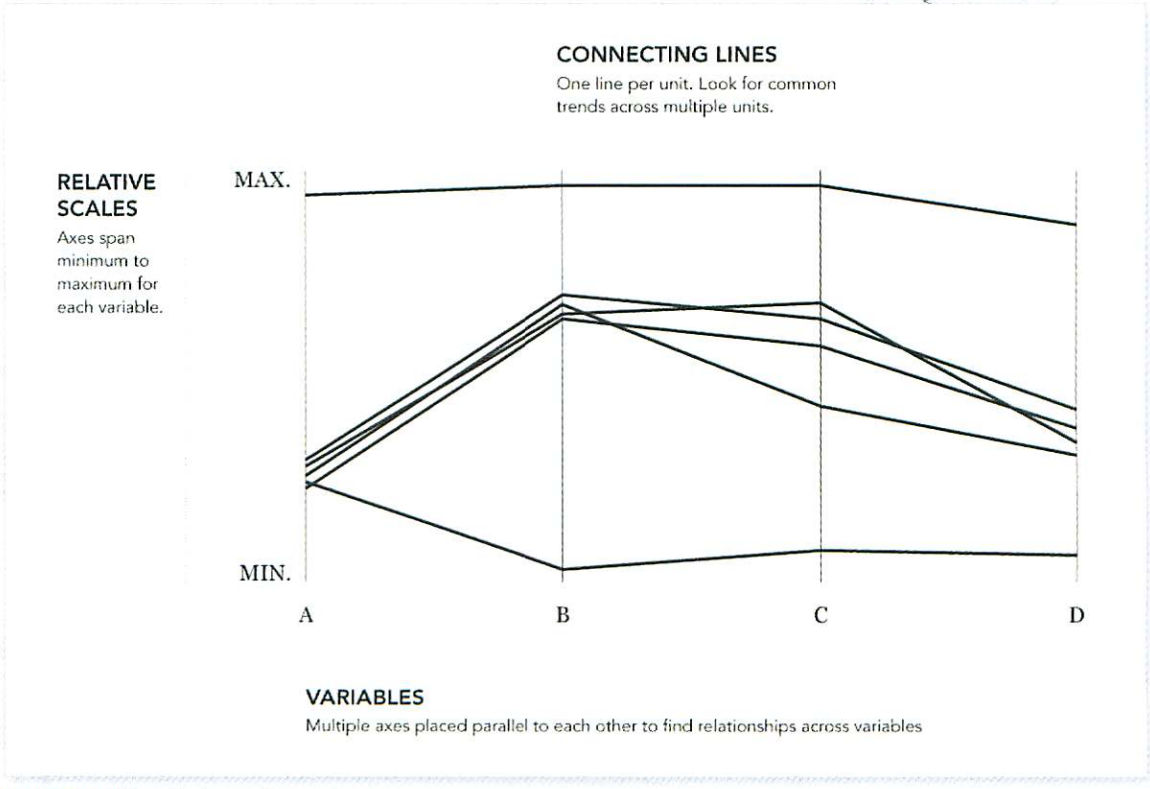


FIGURE 7-20 Parallel coordinates framework

For example, imagine you made a plot using the basketball data from earlier in the chapter. For the sake of simplicity, you only plot points, rebounding, and fouls, in that order. Now imagine a player who was a top scorer, a

weak rebounder, and fouled a lot. A line in the parallel coordinates plot for that player would start high, go low, and then come back up again.

When you plot more units, this method helps to spot groups and tendencies. In the following example, you can use parallel coordinates on data from the National Center for Education Statistics.

CREATE A PARALLEL COORDINATES PLOT

You have several interactive options for parallel coordinates. You can build it in Protovis if you want to make a custom graphic, or you can plug your data into an exploratory tool such as GGobi. These implementations enable you to filter and highlight the data points you're interested in; however, I still like to go with static parallel coordinates plots, namely because you can compare different filters all at once. With interactive versions, you have only one plot, and it's tough to make sense of what you're looking at when you have a bunch of highlighting all in one place.

▶ Download GGobi for free at `http://ggobi.org`.

You know the first step. Before doing any visualizing, you need data. Load our education data into R with `read.csv()`.

```
education <- read.csv("http://datasets.flowingdata.com/education.csv",
header=TRUE)
education[1:10,]
```

There are seven columns. The first is for state name, including "United States" for the national average. The next three are average reading, mathematics, and writing SAT scores. The fifth column is percentage of graduates who actually take the SAT, and the last two columns are pupil-to-staff ratio and high school dropout rate. What you're interested in is whether any of these variables are related and if there are any clear groupings. For example, do states with high dropout rates tend to have low SAT scores on average?

The base distribution of R doesn't supply a straightforward way to use parallel coordinates, but the lattice package does, so use that. Go ahead and load the package (or install if you haven't already).

```
library(lattice)
```

Great, now this will be super easy. The lattice package provides a `parallel()` function that you can quickly use.

```
parallel(education)
```

Use parallelplot

This produces the plot shown in Figure 7-21. Okay, that's quite useless, actually. There are a bunch of lines all over the place, and the variables go top to bottom instead of left to right. It looks like rainbow spaghetti at this point.

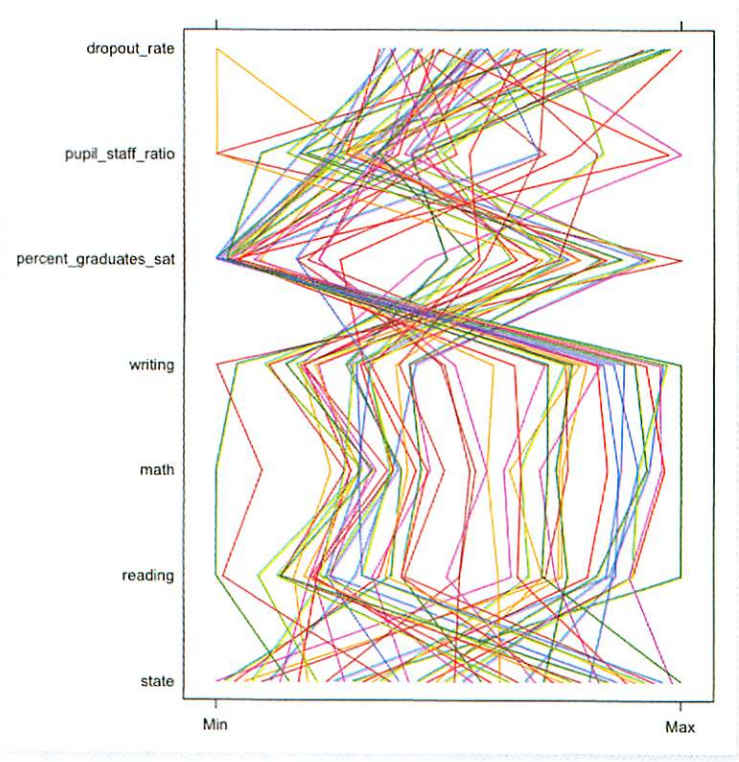


FIGURE 7-21 Default parallel coordinates plot with the lattice package

How can you modify this parallel coordinates plot so that you can actually get some information out of it? For starters, flip it on its side. This is more of a personal preference than it is a rule, but parallel coordinates left to right makes more sense, as shown in Figure 7-22.

```
parallel(education, horizontal.axis=FALSE)
```

You also don't need to include the *state* column, because for one, it's categorical, and second, every state has a different name. Now change the

color of the lines to black—I'm all for color, but it's too much as-is. Execute the line of code below and you get Figure 7-23.

```
parallel(education[,2:7], horizontal.axis=FALSE, col="#000000")
```

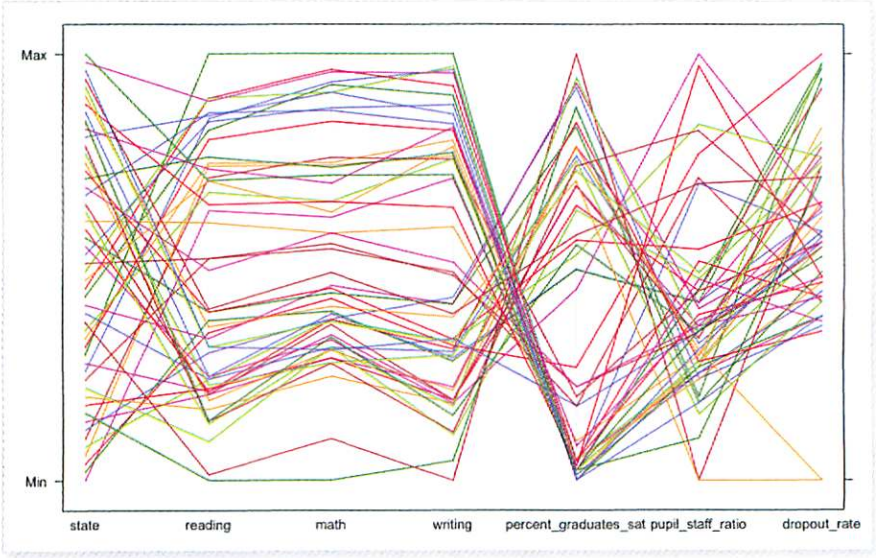


FIGURE 7-22 Horizontal parallel coordinates

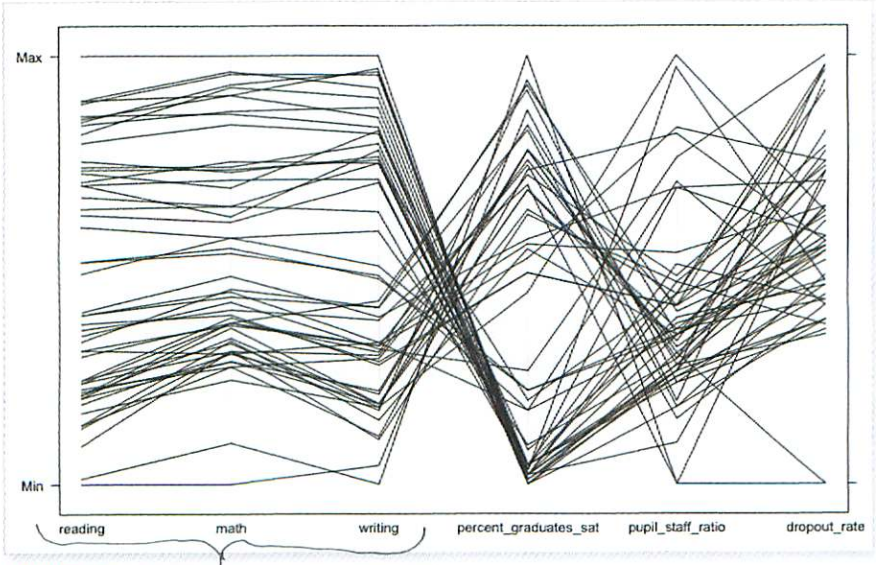


FIGURE 7-23 Parallel coordinates, simplified

SAT scores in these disciplines

That's a little better. The lines from reading, to math, to writing seldom criss-cross and almost run parallel. This makes sense that states with high reading scores also have math and writing scores. Similarly, states with low reading scores tend to have low math and writing scores.

Then something interesting happens as you go from SAT scores to percentage of graduates who take the SAT. It looks like the states with higher SAT score averages tend to have a lower percentage of graduates who take the test. It's the opposite for states with lower SAT averages. My specialty isn't in education, but my best bet is that in some states everyone takes the SAT, whereas in others people don't take the test if they're not planning to go to college. Thus, the average comes down when you require people who don't care about the results to take the test.

You can make this point more obvious by offering some contrasting colors. The `parallel()` function gives you full control over the colors with the `col` argument. Previously, you used only a single color (`#000000`), but you can also pass it an array of colors, a color value for each row of data. Now make the states in the 50th percentile of reading scores black and the bottom half gray. Use `summary()` to find the medians in the education data. Simply enter `summary(education)` in the console. This actually gives you summary stats for all columns, but it's a quick way to find that the median for reading is 523.

```
       state        reading          math          writing
Alabama    : 1   Min.   :466.0   Min.   :451.0   Min.   :455.0
Alaska     : 1   1st Qu.:497.8   1st Qu.:505.8   1st Qu.:490.0
Arizona    : 1   Median :523.0   Median :525.5   Median :510.0
Arkansas   : 1   Mean   :533.8   Mean   :538.4   Mean   :520.8
California: 1   3rd Qu.:571.2   3rd Qu.:571.2   3rd Qu.:557.5
Colorado   : 1   Max.   :610.0   Max.   :615.0   Max.   :588.0
(Other)    :46
percent_graduates_sat pupil_staff_ratio  dropout_rate
Min.   : 3.00         Min.   : 4.900     Min.   :-1.000
1st Qu.: 6.75         1st Qu.: 6.800     1st Qu.: 2.950
Median :34.00         Median : 7.400     Median : 3.950
Mean   :37.35         Mean   : 7.729     Mean   : 4.079
3rd Qu.:66.25         3rd Qu.: 8.150     3rd Qu.: 5.300
Max.   :90.00         Max.   :12.100     Max.   : 7.600
```

explain quartiles with a graph example?

Now iterate through each row of the data; check if it's above or below and specify the colors accordingly. The `c()` directive creates an empty vector, which you add to in each iteration.

```
reading_colors <- c()
for (i in 1:length(education$state)) {

    if (education$reading[i] > 523) {
        col <- "#000000"
    } else {
        col <- "#cccccc"
    }
    reading_colors <- c(reading_colors, col)
}
```

Then pass the `reading_colors` array into parallel instead of the lone `"#000000"`. This gives you Figure 7-24, and it's much easier to see the big move from high to low.

```
parallel(education[,2:7], horizontal.axis=FALSE, col=reading_colors)
```

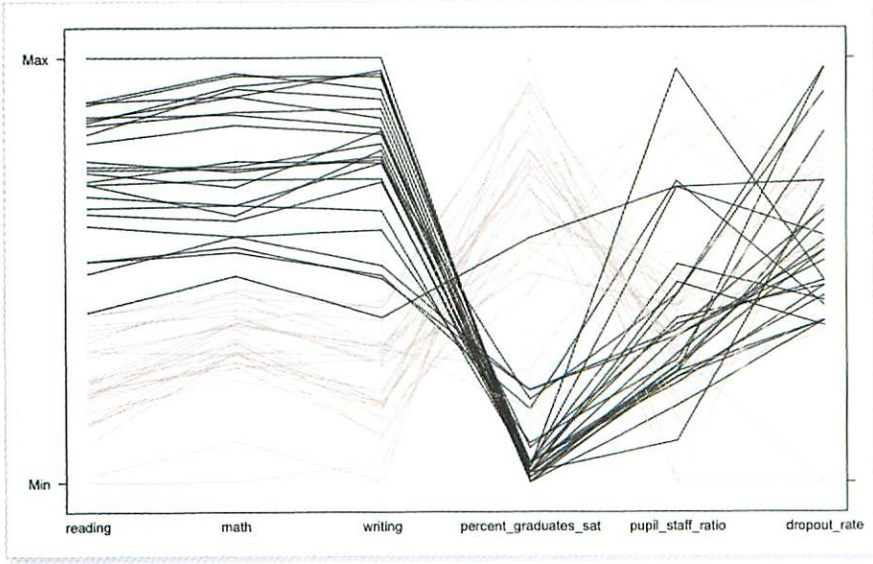


FIGURE 7-24 States with top reading scores highlighted

What about dropout rates? What if you do the same thing with dropout rates that you just did with reading scores, except you use the third quartile instead of the median? The quartile is 5.3 percent. Again, you iterate over each row of data, but this time check the dropout rate instead of reading score.

```
dropout_colors <- c()
for (i in 1:length(education$state)) {

    if (education$dropout_rate[i] > 5.3) {
        col <- "#000000"
    } else {
        col <- "#cccccc"
    }
    dropout_colors <- c(dropout_colors, col)
}
parallel(education[,2:7], horizontal.axis=FALSE, col=dropout_colors)
```

Figure 7-25 shows what you get, and it's not nearly as compelling as the previous graphic. Visually speaking, there aren't any obvious groupings across all of the variables.

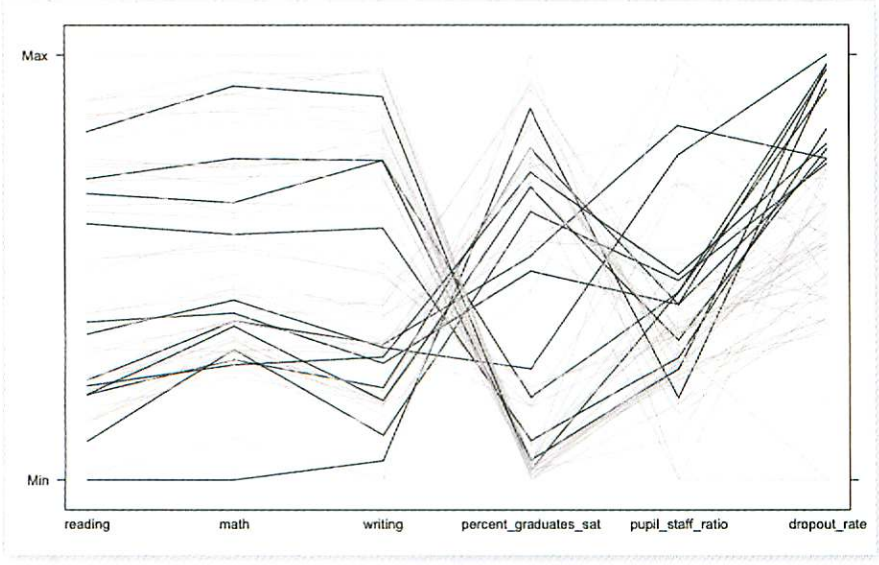


FIGURE 7-25 States with highest dropout rates highlighted

You can do more exploring on your own. Now go back to Figure 7-24 and tighten it up. Better looking labels that are more obvious would be good. Maybe add some color instead of all grayscale? How about a short blurb about why the top 50 percent of states are highlighted? What do you get? Figure 7-26.

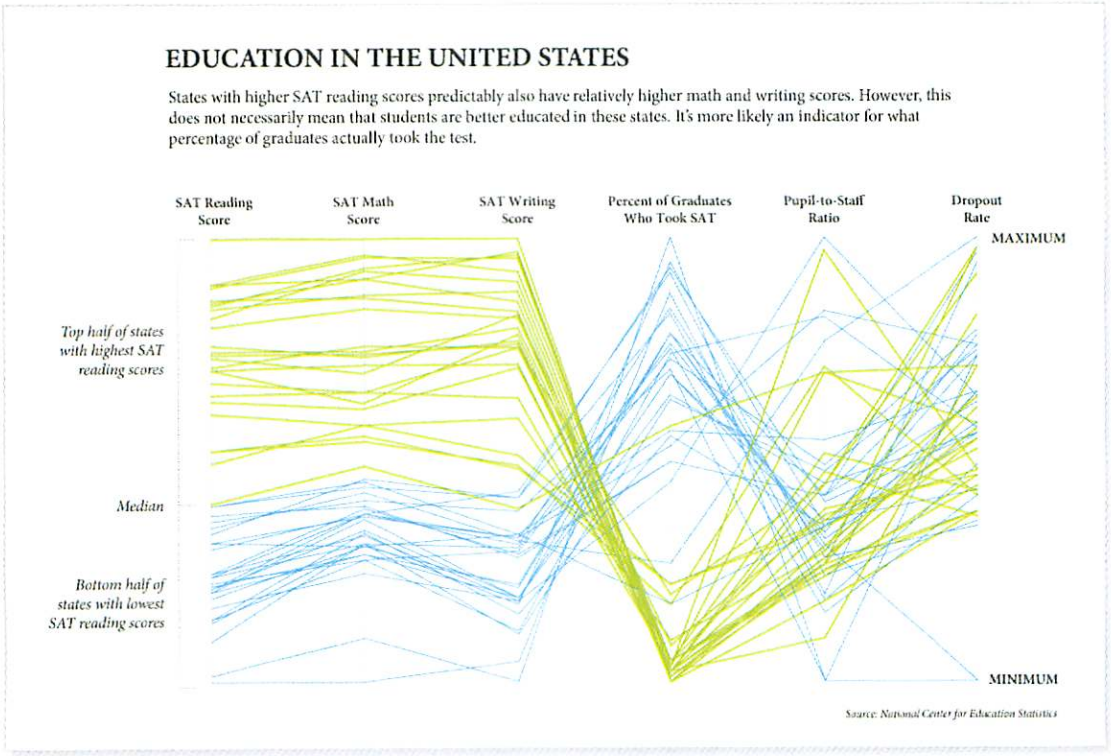


FIGURE 7-26 Standalone parallel coordinates plot on SAT scores

Reducing Dimensions

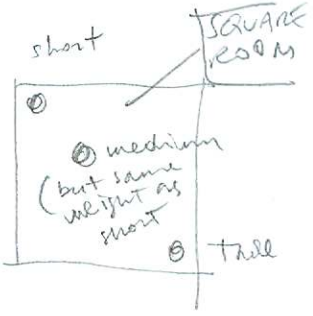
When you use Chernoff Faces or parallel coordinates, your main goal is to reduce. You want to find groups within the dataset or population. The challenge is that you don't always know where to start looking in the faces or

the connecting lines, so it'd be nice if you could cluster objects, based on several criteria. This is one of the goals of multidimensional scaling (MDS). Take everything into account, and then place units that are more similar closer together on a plot.

Entire books are written on this topic, so explanations can get technical, but for the sake of simplicity, I'll keep it at a high level and leave the math for another day. That said, MDS is one of the first concepts I learned in graduate school, and it is worth learning the mechanics behind it, if you're into that sort of thing.

▶ For more details on the method, look up multidimensional scaling or principal components analysis.

Imagine that you're in an empty, square-shaped room and there are two other people there. It's your job to tell those people where to stand in the room, based on their height. The more similar their height, the closer they should stand, and the more different their height, the farther away they should stand. One is really short. The other is really tall. Where should they go? The two people should stand at opposite corners, because they are complete opposites.



Now a third person comes in, and he's medium height. Sticking with the arrangement scheme, the new person should stand in the center of the room, right in between the first two. He's equally different from the tall and the short, so he's equal distance from each. At the same time, the tall and short people are still maximum distance from each other.

Okay, now introduce another variable: weight. You know the height and weight of all three people. The short and medium height people are actually the exact same weight whereas the tall person is, say, a third heavier. How can you, based on height and weight, place the three people in the room? Well, if you keep the first two people (short and tall) in their opposite positions, the third person (medium height) would need to move closer to the shorter person, because their weights are the same.

great example

replicate in class based on 3-4 variables
- height
- weight
- hair color
Black → Beard

Do you get what is occurring? The more similar two people are, the closer they should stand to each other. In this simple case, you have only three people and two variables, so it's easy to work this out manually, but imagine you have 50 people, and you have to place them in the room based on say, five criteria. It's trickier. And that's what multidimensional scaling is for.

Make Use of Multidimensional Scaling

Multidimensional scaling is much easier to understand with a concrete example, so jump right in. Come back to the education data, so if you haven't loaded it in R already, go ahead and do that first.

```
education <-
    read.csv("http://datasets.flowingdata.com/education.csv",
    header=TRUE)
```

Remember, there is a row for each state, which includes the District of Columbia and more rows for the United States averages. There are six variables for each state: reading, math, and writing SAT scores; percentage of graduates who took the SAT; pupil-to-staff ratio; and dropout rate.

It's just like the room metaphor, but instead of a square room, it's a square plot; instead of people, there are states; and instead of height and weight, you have education-related metrics. The goal is the same. You want to place the states on an x-y plot, so that similar states are closer together.

First step: Figure out how far each state should be from every other state. Use the `dist()` function, which does just that. You use only columns 2 through 7 because the first column is state names, and you know all those are different.

```
ed.dis <- dist(education[,2:7])
```

If you type **ed.dis** in the console, you see a series of matrices. Each cell represents how far one state should be from another (by Euclidean pixel distance). For example, the value in the second row, second column over is the distance Alabama should be from Alaska. The units aren't so important at this point. Rather it's the relative differences that matter.

How do you plot this 51 by 51 matrix on an x-y plot? You can't yet, until you have an x-y coordinate for each state. That's what `cmdscale()` is for. It takes a distance matrix as input and returns a set of points so that the differences between those points are about the same as specified in the matrix.

```
ed.mds <- cmdscale(ed.dis)
```

Type `ed.mds` in the console, and you see you now have x-y coordinates for each row of data. Store these in the variables *x* and *y*, and toss them into plot() to see what it looks like (Figure 7-27).

```
x <- ed.mds[,1]
y <- ed.mds[,2]
plot(x,y)
```

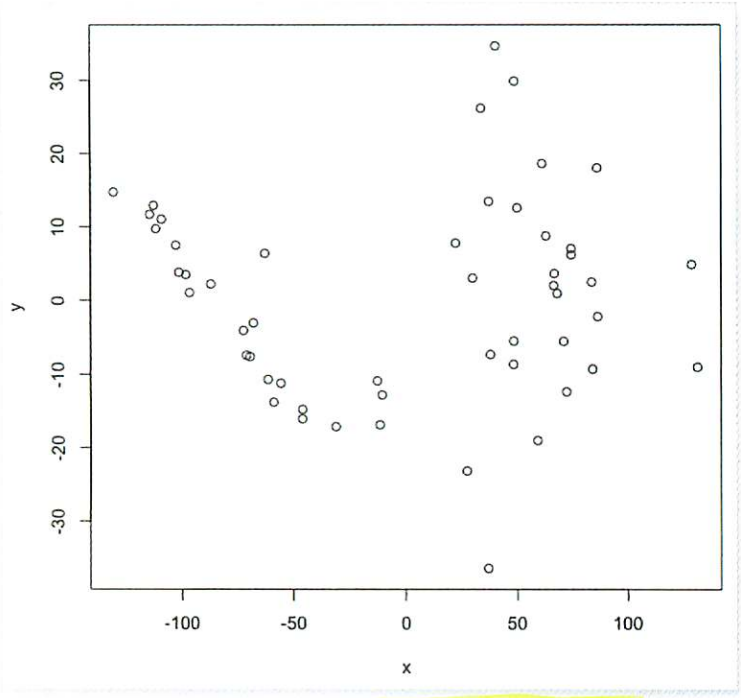


FIGURE 7-27 Dot plot showing results of multidimensional scaling

Not bad. Each dot represents a state. One problem, though: You don't know what state is which. You need labels, so like before use `text()` to put state names in place of the dots, as shown in Figure 7-28.

```
plot(x, y, type="n")
text(x, y, labels=education$state)
```

That's kind of cool. You see a couple of clusters emerge, with one on the left and the other on the right. United States is on the bottom of the right cluster, toward the middle, which seems about right. At this point, it's up to you to figure what the clusters mean, but it's a good jumping off point in your data exploration escapades.

You could, for example, color the states by `dropout_colors` like you did with the parallel coordinates, as shown in Figure 7-29. It doesn't tell you much, but it does confirm what you saw in Figure 7-25.

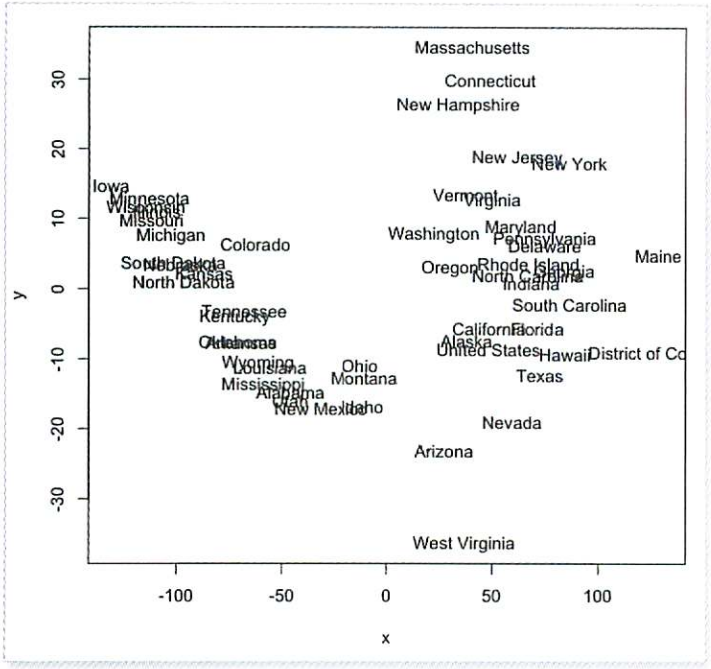


FIGURE 7-28 Using state names instead of dots to see where each state was placed

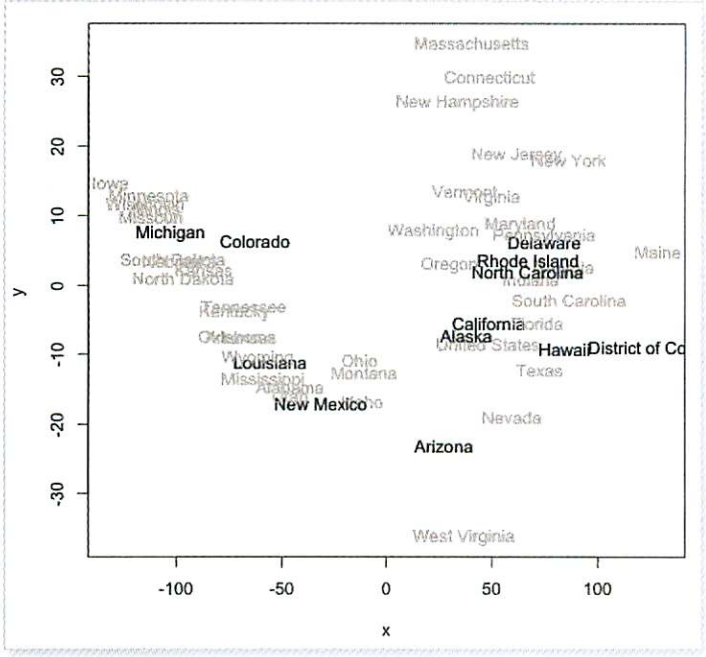


FIGURE 7-29 States colored by dropout rates

What about states colored by reading scores? Yeah, you can do that, too, as shown in Figure 7-30. Ah, it looks like there's a clear pattern there. High scores are on the left and lower scores are on the right? What makes Washington different? Look into that—you can tell me later.

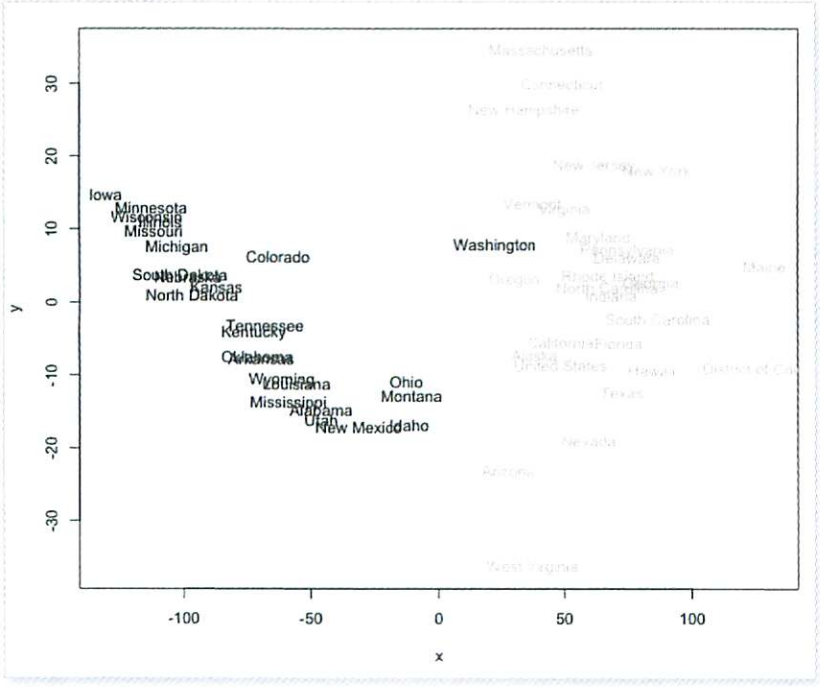


FIGURE 7-30 States colored by reading scores

If you want to be fancy, you can try something called model-based clustering. I'm not going to get into the details of it. I'll just show you how to do it, and you can take my word for it that we're not doing any magic here. There's actual math involved. Basically, use the `mclust` package to identify clusters in your MDS plot. Install `mclust` if you haven't already. Now run the following code for the plots in Figure 7-31.

```
library(mclust)
ed.mclust <- Mclust(ed.mds)
plot(ed.mclust, data=ed.mds)
```

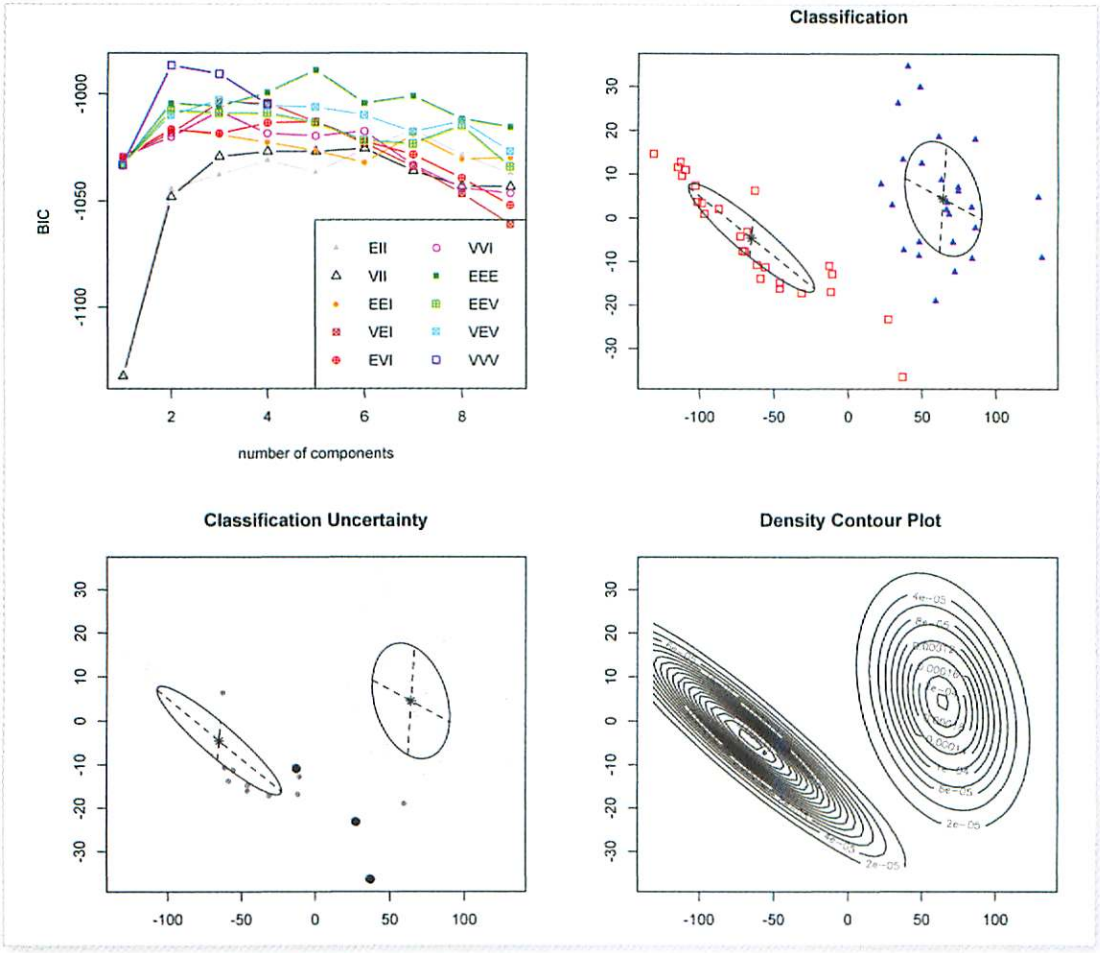


FIGURE 7-31 Results of model-based clustering

That first plot on the top left shows the results of running an algorithm to find the ideal number of clusters in the data. The remaining three plots show the clusters. Pretty awesome. You get the two clusters, now more well defined, showing high and low states.

This is usually when I tell you to bring your PDF files into Illustrator and apply some touchups, but I'm not so sure I'd ever publish these for a general audience. It's too abstract for the nontechnically minded to figure out what's going on. They're good for data exploration; however, if you were so inclined, all standard design principles apply. Figure out what you need to tell a clear story, and strip out the rest.

Searching for Outliers

Rather than looking for how units of data belong in certain groups, you should also be interested in how they don't belong in groups. That is, there will often be data points that stand out from the rest, which are called, you guessed it, *outliers*. These are data points that are different from the rest of the population. Sometimes they could be the most interesting part of your story, or they could just be boring typos with a missing zero. Either way, you need to check them out to see what's going on. You don't want to make a giant graphic on the premise of an outlier, only to find out later from a diligent reader that your hard work makes no sense.

Graphic types have been designed specifically to highlight outliers, but in my experience, nothing beats basic plots and common sense. Learn about the context of your data, do your homework, and ask experts about the data when you're not sure about something. Once you find the outliers, you can use the same graphical techniques that we've used so far to highlight them for readers: Use varied colors, provide pointers, or use thicker borders.

Now look at a simple example. Figure 7-32 shows a time series plot that shows weather data scraped from Weather Underground (like you did in Chapter 2, "Handling Data"), from 1980 to 2005. There are seasonal cycles like you'd expect, but what's going on in the middle? It seems to be unusually smooth, whereas the rest of the data has some noise. This is nothing to go crazy over, but if you happen to run weather models on this data, you might want to know what has been estimated and what's real data.

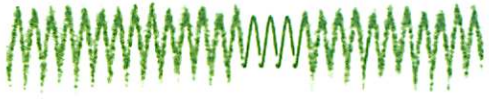


FIGURE 7-32 Estimated weather data from Weather Underground

Similarly, looking at the star charts you made that show crime, you can see the District of Columbia stands out. You could have seen this just as easily with a basic bar chart, as shown in Figure 7-33. Is it fair to compare Washington, DC to the states, considering it has more of a city makeup? You be the judge.

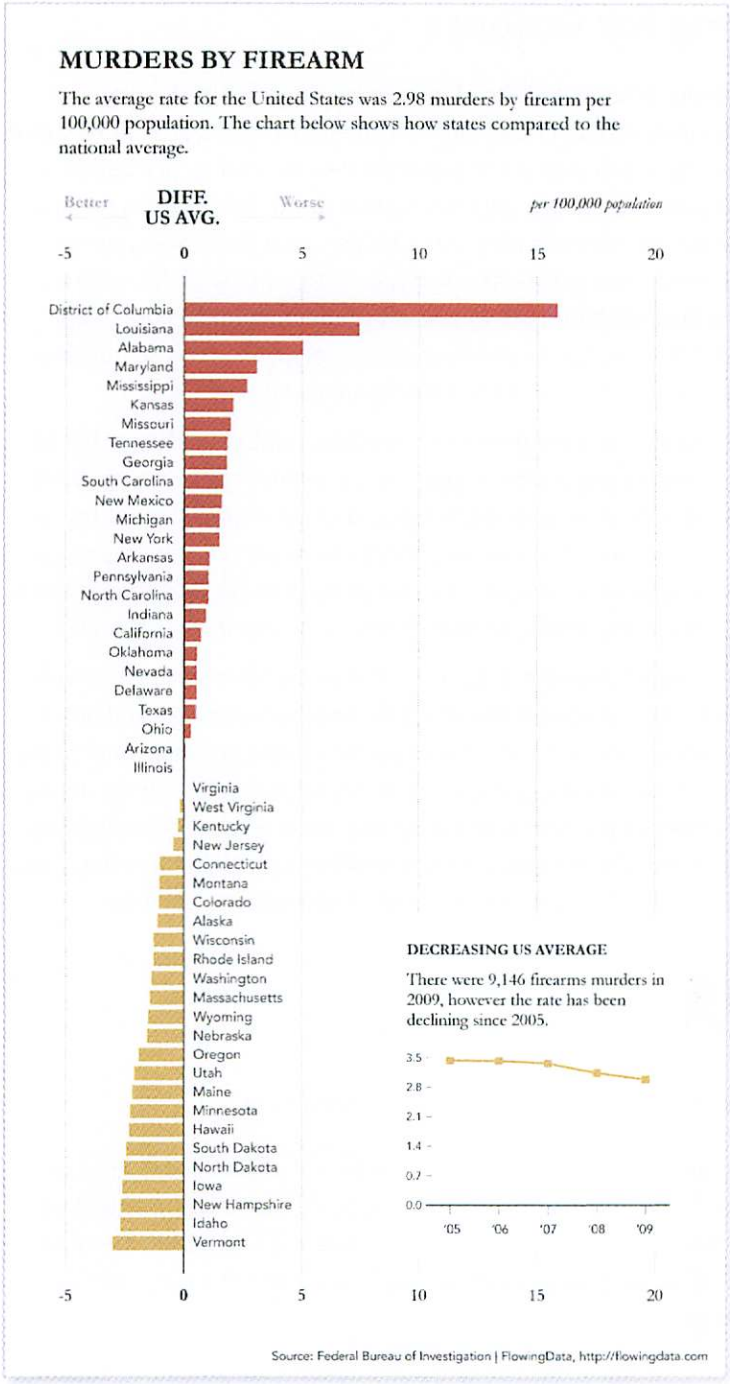


FIGURE 7-33 Murders by Firearm in the United States

How about the subscriber counts from Chapter 3, "Choosing Tools to Visualize Data," shown in Figure 7-34? There's that big dip in the middle where it looks like more than half of FlowingData's readership is lost.

You can also look at the distribution as a whole via a histogram, as shown in Figure 7-35. All counts are sitting on the right, except for a couple all the way to the left with nothing in the middle.

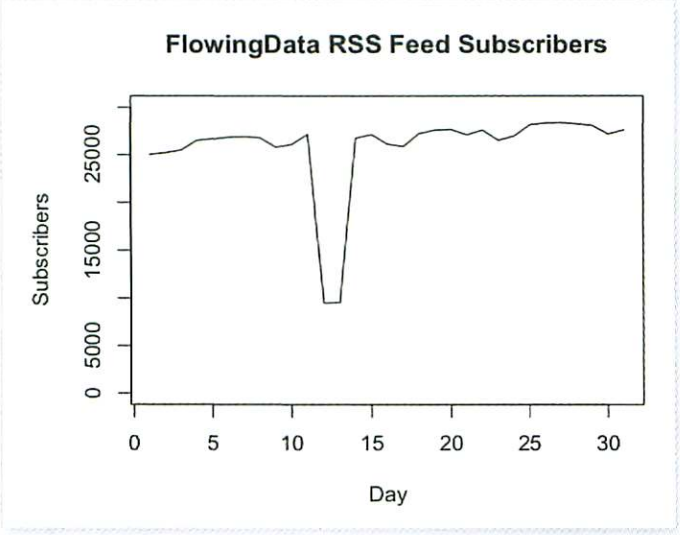


FIGURE 7-34 FlowingData subscriber counts over time

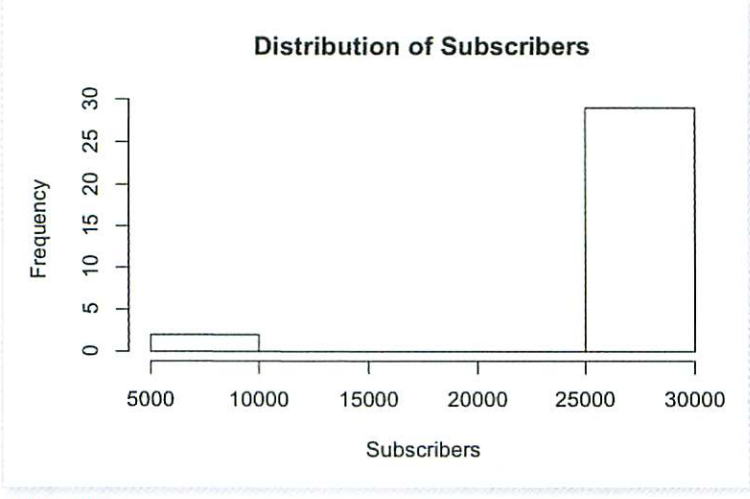


FIGURE 7-35 Histogram showing distribution of subscriber counts

More concretely, you can use a boxplot, which shows quartiles in a distribution. Boxplots generated in R with the `boxplot()` function can automatically highlight points that are more than 1.5 times more or less than the upper and lower quartiles, respectively (Figure 7-36).

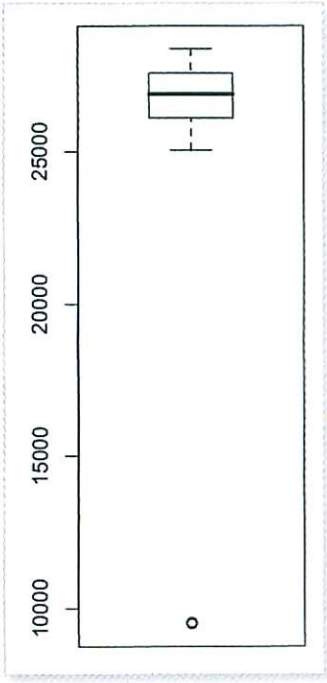


FIGURE 7-36 Boxplot showing distribution of subscriber counts

▶ A *quartile* is one of three points in a dataset, which marks quarter spots. The middle quartile is the median or the halfway point; the upper quartile marks the spot where 25 percent of the data is greater than that value; and the lower quartile marks the bottom 25 percent.

If I had a small number of subscribers in the single digits, then sure, such a big percentage-wise decrease could be possible, but it's unlikely that I said something so offensive to compel tens of thousands of readers to defect (and then come back a couple of days later). It's much more likely that Feedburner, the feed delivery service I use, made a reporting error.

These outliers in these datasets are obvious because you know a little bit about the data. It could be less obvious if you use datasets you're not familiar with. When that happens, it can be helpful to go directly to the source and just ask whoever is in charge. The person or group curating the data is usually happy that you're making use of it and will offer some

quick advice. If you can't find out any more details, you can at least say you tried, and make a note of the ambiguity in your explanation of your graphic.

Wrapping Up

For beginners, one of the hardest parts to design data graphics is to figure out where to start. You have all this data in front of you without a clue about what it is or what to expect. Usually, you should start with a question about the data and work off of that question, but what if you don't know what to ask? The methods described in this chapter can help a lot with this. They help you see all the data at once, which makes it easier to figure out what part of the data to explore next.

However, don't stop here. Use these as jumping off points to narrow down to spots that look interesting. This, in addition to what previous chapters cover should be enough to help you dig deep into your data, no matter what type of data you deal with. Well, except for one. The next chapter covers one more data type: spatial data. Get ready to make some maps.