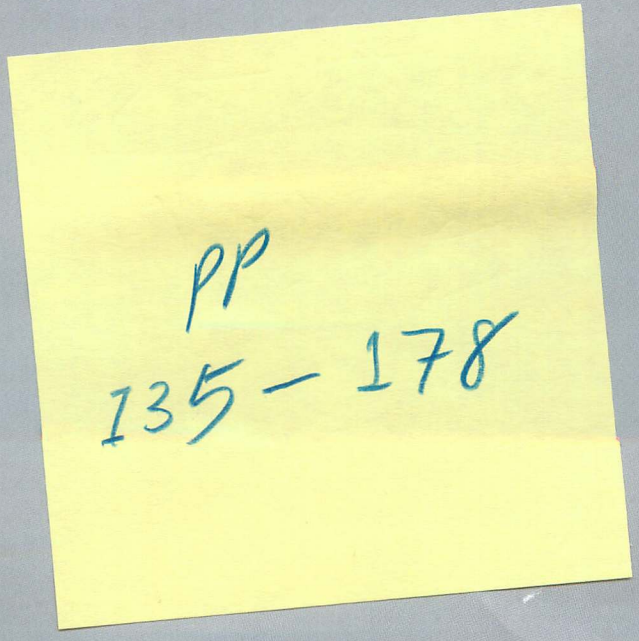
Visualizing Proportions

5

Time series data is naturally grouped by, well, time. A series of events happen during a specific time frame. Proportion data is also grouped, but by categories, subcategories, and population. By population, I don't mean just human population. Rather, population in this case represents all possible choices or outcomes. It's the sample space.

In a poll, people might be asked if they approve, disapprove, or have no opinion on some issue. Each category represents something, and the sum of the parts represent a whole.

This chapter discusses how to represent the individual categories, but still provides the bigger picture of how each choice relates to the other. You use some of what you learned in the previous chapter and get your first taste of interactive graphics using HTML, CSS, and JavaScript and then have a look at graphics with Flash.



What to Look for in Proportions

For proportions you usually look for three things: maximum, minimum, and the overall distribution. The first two are straightforward. Sort your data from least to greatest, and pick the ends for your maximum and minimum. If you were dealing with poll results, these could mean the most popular and least popular answers from participants; or if you were graphing calories from separate parts of a meal, you would see the biggest and smallest contributor to the overall calorie count.

You don't need a chart though to show you minimum and maximum. What you are most interested in is the distribution of proportions. How does the selection of one poll choice compare to the others? Are calories spread evenly across fat, protein, and carbohydrates, or does one group dominate? The following chart types can help you figure that out.

Parts of a Whole

This is proportions in their simplest form. You have a set of proportions that add up to 1 or a set of percentages that add up to 100 percent. You want to show the individual parts relative to the other parts but you also want to maintain the sense of a whole.

The Pie

Pie charts are the old standby. You see them everywhere these days, from business presentations to sites that use charts as a medium for jokes. The first known pie chart was published by William Playfair, who also invented the line graph and bar chart, in 1801. Smart guy.

You know how they work. As shown in Figure 5-1, you start with a circle, which represents a whole, and then cut wedges, like you would a pie. Each wedge represents a part of the whole. Remember that last part, because a lot of beginners make this mistake. The percentage of all the wedges should add up to 100 percent. If the sum is anything else, you have done something wrong.

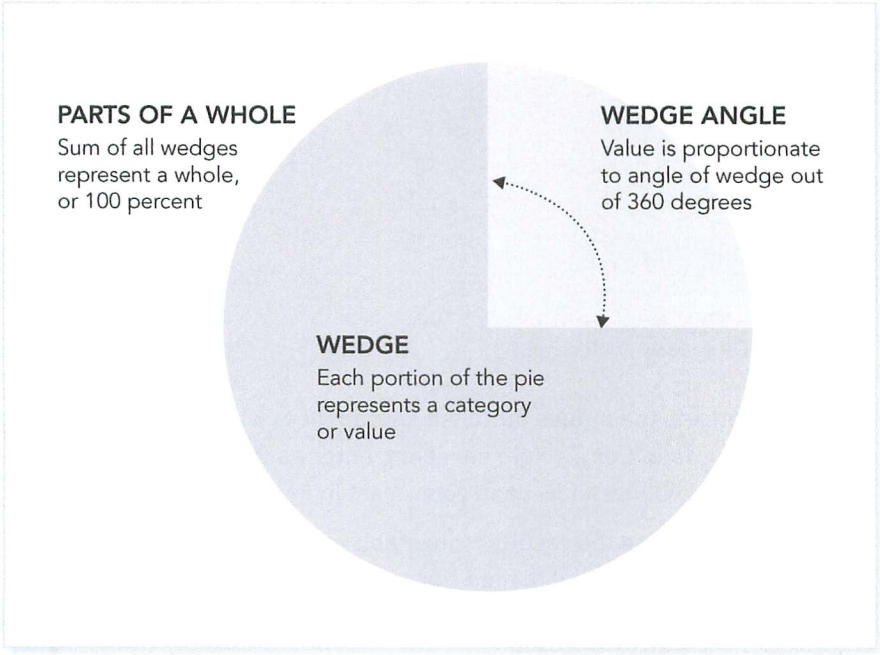


FIGURE 5-1 Pie chart generalized

Pie charts have developed a stigma for not being as accurate as bar charts or position-based visuals, so some think you should avoid them completely. It's easier to judge length than it is to judge areas and angles. That doesn't mean you have to completely avoid them though.

You can use the pie chart without any problems just as long you know its limitations. It's simple. Keep your data organized, and don't put too many wedges in one pie.

CREATE A PIE CHART

Although just about every charting program enables you to make pie charts, you can make one in Illustrator just like you did in the previous chapter. The process of adding data, making a default chart, and then refining should feel familiar.

To build the base of your chart—the actual pie—is fairly straightforward. After you create a new document, select the Pie Graph tool from the Tool window, as shown in Figure 5-2. Click and drag a rectangle so that it is roughly the size of what you want your graph to be. You can resize it later.

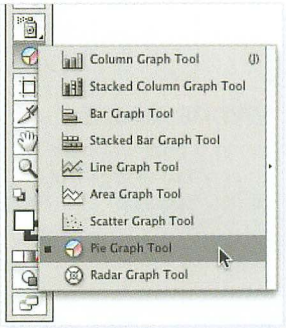


FIGURE 5-2 Tool window in Illustrator

When you release the mouse button, a spreadsheet window pops up where you enter your data. For a single pie chart, enter each data point from left to right, and the values show up in your chart in the same order.

For this example, use the results from a poll on FlowingData. Readers were asked what data-related field they were most interested in. There were 831 responses.

AREA OF INTEREST	NUMBER OF VOTES
Statistics	172
Design	136
Business	135
Cartography	101
Information Science	80
Web Analytics	68
Programming	50
Engineering	29
Mathematics	19
Other	41

Enter the numbers in the spreadsheet in Illustrator, as shown in Figure 5-3. The order you enter the numbers will match the order the wedges appear in your pie chart, starting at the top and then rotating clockwise.

Notice that the poll results are organized from greatest to least and then end with the Other category. This kind of sorting can make your pie charts

easier to read. Click the check mark in the top right of the pop-up when you finish.

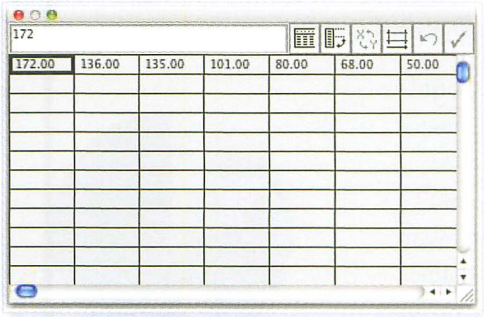


FIGURE 5-3 Spreadsheet in Illustrator

The default pie chart appears with eight shades of gray, in a seemingly random sequence with a black border, as shown in Figure 5-4. It kind of looks like a grayscale lollipop, but you can easily do something about that. The important thing here is that you have the base of your pie chart.

Now make the pie chart more readable by changing some colors and adding text to explain to readers what they are looking at. As it is now, the colors don't make much sense. They just separate the wedges, but you can use colors as a way to tell readers what to look at and in what order. You did after all go through the trouble of sorting your data from greatest to least.

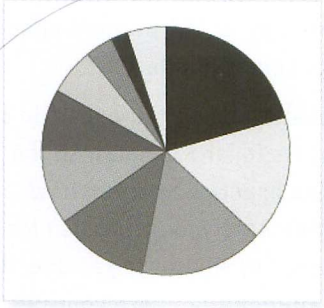


FIGURE 5-4 Default pie chart

If you start at 12 o'clock and rotate clockwise, you should see a descending order. However, because of the arbitrary color scheme, some of the smaller wedges are emphasized with darker shades. The dark shade acts as a highlighter, so instead, make larger wedges darker and smaller wedges lighter. If for some reason, you want to highlight answers that have fewer responses, you might want to color in reverse. In the case of this poll though, you want to know what data-related fields were the most popular.

TIP

The spreadsheet in Illustrator is fairly bare bones, so you can't easily manipulate or rearrange your data. A way to get around this is to do all your data handling in Microsoft Excel, and then copy and paste into Illustrator.

TIP

Color can play an important role in how people read your graph. It's not just an aesthetic component—although sometimes it can be. Color can be a visual cue just like length or area, so choose wisely.

Choose the Direct Selection tool in the Tool window, and then click a wedge. Change fill and stroke color via the controls in the Color window. Figure 5-5 shows the same pie chart with a white stroke and wedges colored from darkest to lightest. Now it's easier to see that numbers go from greatest to least, with the exception of the last wedge for Other.

NOTE

When you use opacity, the fill of the shape you are changing will blend with the color of the background. In this case, the background is white, which gives a faded look the higher the transparency. If, however, the background were blue, the shape would appear purple.

Of course, you don't have to be so frugal with your colors. You can use whatever colors you want, as shown Figure 5-6. Although it's usually a good idea to not use colors that are bright—you don't want to blind your readers. If a blinding color scheme fits with your topic though, go wild.

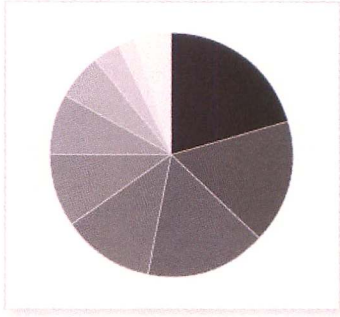


FIGURE 5-5 Pie chart with colors arranged darkest to lightest

Because this is a FlowingData poll, I used the shade of red from the FlowingData logo and then made lighter color wedges by decreasing opacity. You can find the option in the Transparency window. At 0 percent opacity, the fill is completely see-through; at 100 percent opacity, the fill is not see-through.

Finally, add a title, a lead-in sentence, and labels for the graph with the Type tool. With practice, you can have a better idea what fonts you like to use for headers and copy, but whatever you use, Illustrator's alignment tools are your best friend when it comes to placing your text. Properly aligned and evenly spaced labels make your charts more readable. You can also make use of the Pen tool to create pointers, as shown in Figure 5-7, for the last three poll categories. These sections are too small to put the labels inside and are too close together to place the labels adjacent.

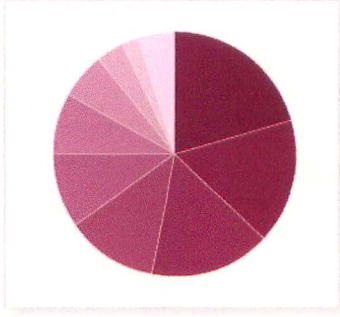


FIGURE 5-6 Colored pie chart

WHAT DATA-RELATED AREA ARE YOU MOST INTERESTED IN?

Below are results of a poll on FlowingData in May 2009. Readers come from a variety of fields, but Statistics, Design, and Business led the way.

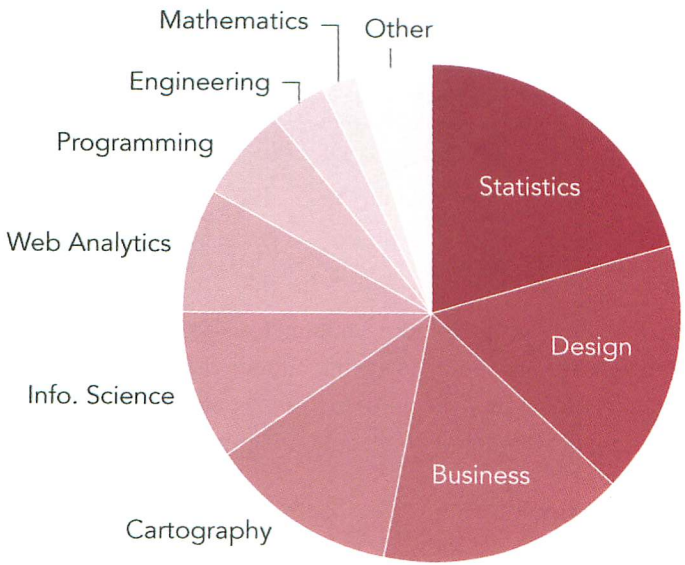


FIGURE 5-7 Final pie chart with labels and lead-in copy

The Donut

Your good friend the pie chart also has a lesser cousin: the donut chart. It's like a pie chart, but with a hole cut out in the middle so that it looks like a donut, as shown in Figure 5-8.

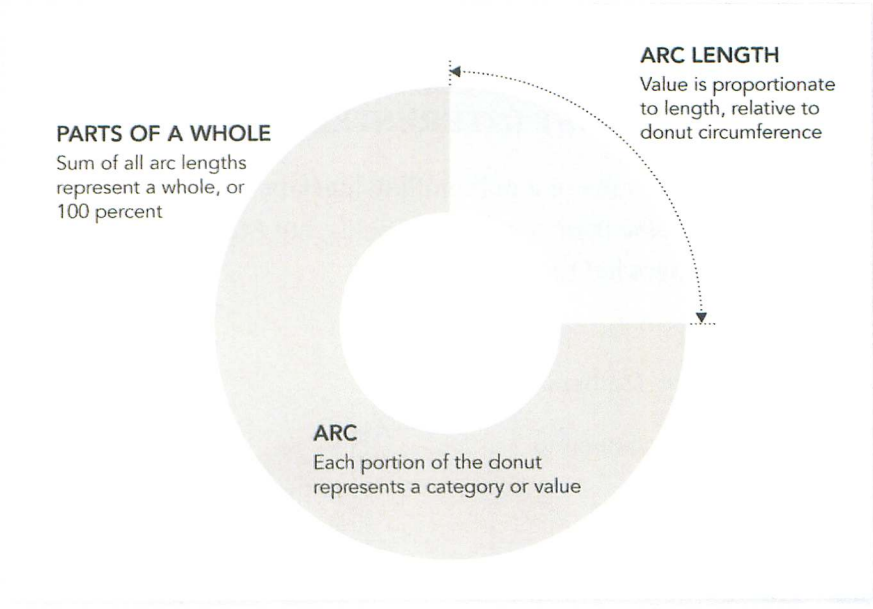


FIGURE 5-8 Donut chart framework

TIP

The main thing to remember, whether you use a pie chart or donut chart, is that they can quickly become cluttered. They're not meant to represent a lot of values.

Because there's a hole in the middle, you don't judge values by angle anymore. Instead you use arc length. This introduces many of the same problems when you use a single chart with too many categories, but in cases with fewer categories the donut chart can still come in handy.

CREATE A DONUT CHART

It's straightforward to make a donut chart in Illustrator. Create a pie chart like you just did; then put a filled circle in the middle, as shown in Figure 5-9. Again, use color to guide readers' eyes.

A lot of the time the middle of donut charts are used for a label or some other content like was done in the figure.

Now make the same chart using Protovis, the free and open-source visualization toolkit. It's a library implemented in JavaScript and makes use of modern browsers' Scalable Vector Graphics (SVG) capabilities. Graphics are generated dynamically and enable animation and interactivity, which makes Protovis great for online graphics.

▶ Download Protovis at `http://vis.stanford.edu/protovis/` and put it in the same directory that you use to save example files.

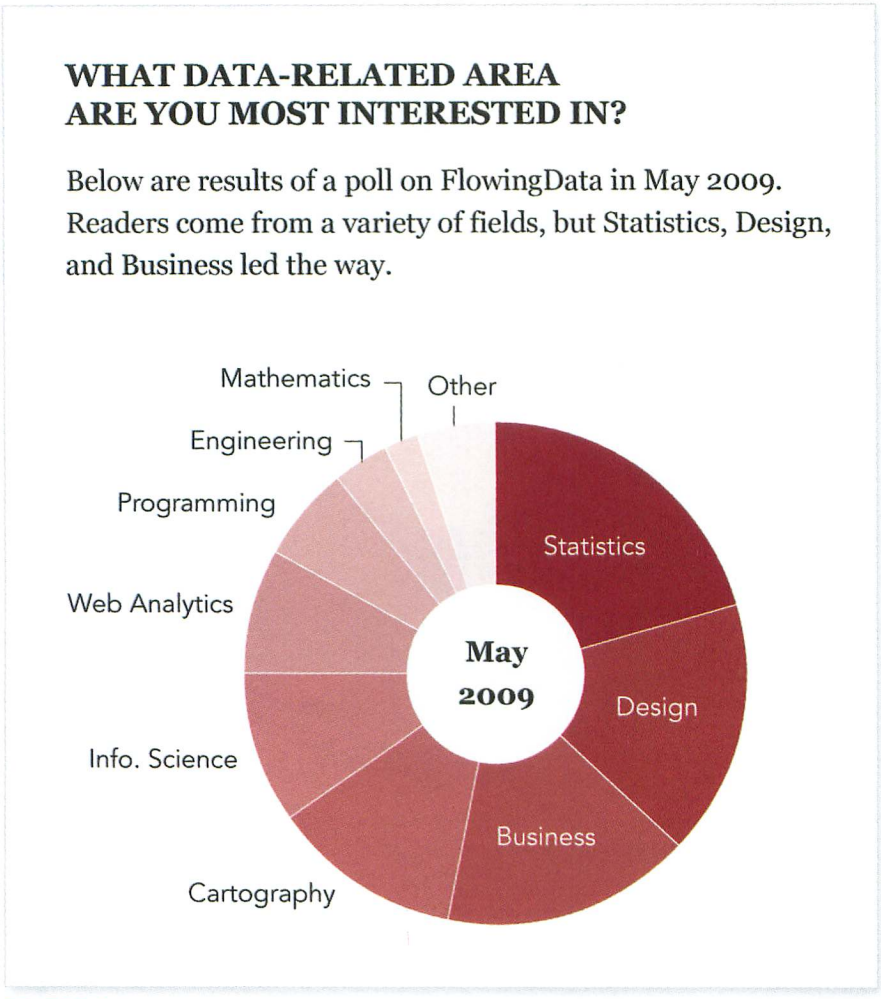


FIGURE 5-9 From pie to donut chart

Although you're about to get into a different programming language, you still follow the same process like you did in R and Illustrator. First, load the data, then build the base, and finally, customize the aesthetics.

Figure 5-10 shows what you want to make. It's similar to Figure 5-9, except the labels are set at an angle, and when you mouse over an arc, you can see how many votes there were for the corresponding category. Interaction can get much more advanced, but you have to learn the basics before you get fancy.

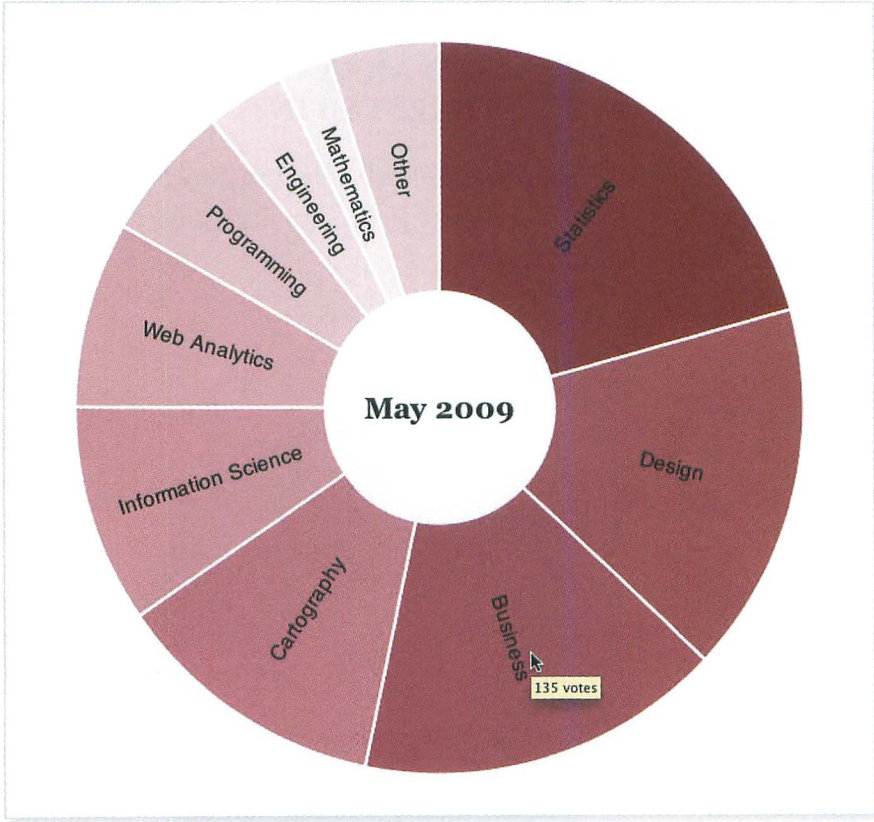


FIGURE 5-10 Donut chart using Protovis

The first thing you do is create an HTML page—call it **donut.html**.

```
<html>
<head>
    <title>Donut Chart</title>
    <script type="text/javascript" src="protovis-r3.2.js"></script>
    <style type="text/css">
        #figure {
            width: 400px;
            height: 400px;
        }
    </style>
</head>
<body>
```

```
    <div id="figure">

    </div><!-- @end figure -->
</body>
</html>
```

If you've ever created a web page, this should be straightforward, but in case you haven't, the preceding is basic HTML that you'll find almost everywhere online. Every page starts with an `<html>` tag and is followed by a `<head>` that contains information about the page but doesn't show in your browser window. Everything enclosed by the `<body>` tag is visible. Title the page **Donut Chart** and load the Protovis library, a JavaScript file, with the `<script>` tag. Then specify some CSS, which is used to style HTML pages. Keeping it simple, set the width and height of the `<div>` with the id "figure" at 400 pixels. This is where you draw our chart. The preceding HTML isn't actually part of the chart but necessary so that the JavaScript that follows loads properly in your browser. All you see is a blank page if you load the preceding `donut.html` file in your browser now.

Inside the figure `<div>`, specify that the code that you're going to write is JavaScript. Everything else goes in these `<script>` tags.

```
<script type="text/javascript+protovis">
</script>
```

Okay, first things first: the data. You're still looking at the results from the FlowingData poll, which you store in arrays. The vote counts are stored in one array, and the corresponding category names are stored in another.

```
var data = [172,136,135,101,80,68,50,29,19,41];
var cats = ["Statistics", "Design", "Business", "Cartography",
    "Information Science", "Web Analytics", "Programming",
    "Engineering", "Mathematics", "Other"];
```

Then specify the width and height of the donut chart and the radius length and scale for arc length.

```
var w = 350,
    h = 350,
    r = w / 2,
    a = pv.Scale.linear(0, pv.sum(data)).range(0, 2 * Math.PI);
```

The width and height of the donut chart are both 350 pixels, and the radius (that is, the center of the chart to the outer edge) is half the width, or 175 pixels. The fourth line specifies the arc scale. Here's how to read it. The actual data is on a linear scale from 0 to the sum of all votes, or total votes. This scale is then translated to the scale to that of the donut, which is from 0 to 2π radians, or 0 to 360 degrees if you want to think of it in that way.

Next create a color scale. The more votes a category receives, the darker the red it should be. In Illustrator, you did this by hand, but Protovis can pick the colors for you. You just pick the range of colors you want.

```
var depthColors = pv.Scale.linear(0, 172).range("white", "#821122");
```

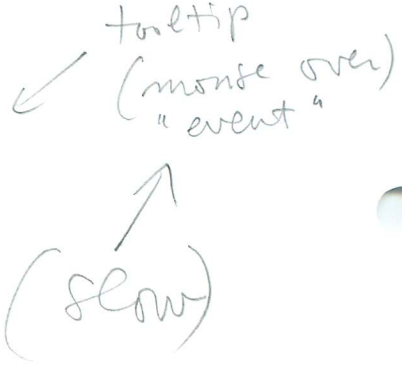
Now you have a color scale from white to a dark red (that is #821122) on a linear range from 0 to 172, the highest vote count. In other words, a category with 0 votes will be white, and one with 172 votes will be dark red. Categories with vote counts in between will be somewhere in between white and red.

So far all you have are variables. You specified size and scale. To create the actual chart, first make a blank panel 350 (w) by 350 (h) pixels.

```
var vis = new pv.Panel()
    .width(w)
    .height(h);
```

Then add stuff to the panel, in this case wedges. It might be a little confusing, but now look over it line by line.

```
vis.add(pv.Wedge)
    .data(data)
    .bottom(w / 2)
    .left(w / 2)
    .innerRadius(r - 120)
    .outerRadius(r)
    .fillStyle(function(d) depthColors(d))
    .strokeStyle("#fff")
    .angle(a)
    .title(function(d) String(d) + " votes")
    .anchor("center").add(pv.Label)
        .text(function(d) cats[this.index]);
```



The first line says that you're adding wedges to the panel, one for each point in the data array. The `bottom()` and `left()` properties orient the wedges so that the points are situated in the center of the circle. The `innerRadius()` specifies the radius of the hole in the middle whereas the `outerRadius` is the radius of the full circle. That covers the structure of the donut chart.

Rather than setting the fill style to a static shade, fill colors are determined by the value of the data point and the color scale stored as `depthColors`, or in other words, color is determined by a function of each point. A white (#fff) border is used, which is specified by `strokeStyle()`. The circular scale you made can determine the angle of each wedge.

To get a tooltip that says how many votes there were when you mouse over a section, `title()` is used. Another option would be to create a *mouseover event* where you specify what happens when a user places a pointer over an object, but because browsers automatically show the value of the title attribute, it's easier to use `title()`. Make the title the value of each data point followed by "votes." Finally, add labels for each section. The only thing left to do is add May 2009 in the hole of the chart.

```
vis.anchor("center").add(pv.Label)
    .font("bold 14px Georgia")
    .text("May 2009");
```

This reads as, "Put a label in the center of the chart in bold 14-pixel Georgia font that says May 2009."

The full chart is now built, so now you can render it.

```
vis.render();
```

When you open donut.html in your browser, you should see Figure 5-10.

If you're new to programming, this section might have felt kind of daunting, but the good news is that Protovis was designed to be learned by example. The library's site has many working examples to learn from and that you can use with your own data. It has traditional statistical graphics to the more advanced interactive and animated graphics. So don't get discouraged if you were a little confused. The effort you put in now will pay off

▶ Visit `http://book.flowingdata.com/ch05/donut.html` to see the live chart and view the source for the code in its entirety.

after you get the hang of things. Now have another look at Protovis in the next section.

Stack Them Up

In the previous chapter you used the stacked bar chart to show data over time, but it's not just temporal data. As shown in Figure 5-11, you can also use the stacked bar chart for categorical data.

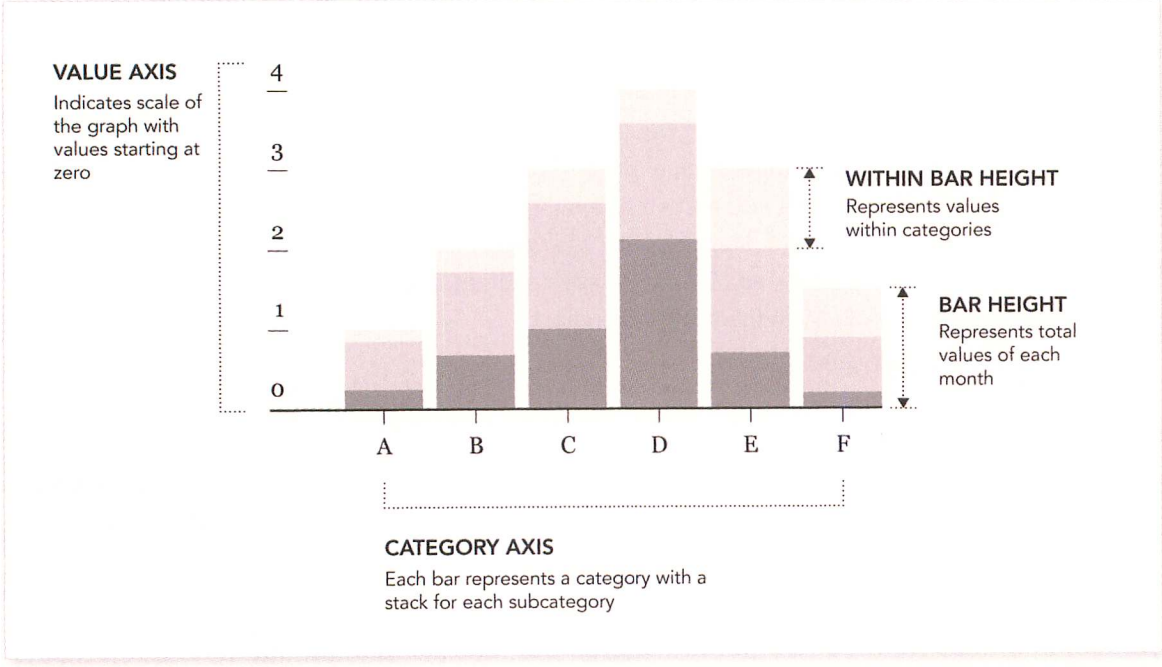


FIGURE 5-11 Stacked bar chart with categories

For example, look at approval ratings for Barack Obama as estimated from a Gallup and CBS poll taken in July and August 2010. Participants were asked whether they approved or disapproved of how Obama has dealt with 13 issues.

Here are the numbers in table form.

ISSUE	APPROVE	DISAPPROVE	NO OPINION
Race relations	52	38	10
Education	49	40	11
Terrorism	48	45	7
Energy policy	47	42	11
Foreign affairs	44	48	8
Environment	43	51	6
Situation in Iraq	41	53	6
Taxes	41	54	5
Healthcare policy	40	57	3
Economy	38	59	3
Situation in Afghanistan	36	57	7
Federal budget deficit	31	64	5
Immigration	29	62	9

One option would be to make a pie chart for every issue, as shown in Figure 5-12. To do this in Illustrator, all you have to do is enter multiple rows of data instead of just a single one. One pie chart is generated for each row.

However, a stacked bar chart enables you to compare approval ratings for the issues more easily because it's easier to judge bar length than wedge angles, so try that. In the previous chapter, you made a stacked bar chart in Illustrator using the Stacked Graph tool. This time you add some simple interactions.

CREATE AN INTERACTIVE STACKED BAR CHART

Like in the donut chart example, use Protovis to create an interactive stacked bar chart. Figure 5-13 shows the final graphic. There are two basic interactions to implement. The first shows the percentage value of any given stack when you place the mouse pointer over it. The second highlights bars in the approve, disapprove, and no opinion categories based on where you put your mouse.

APPROVAL RATINGS FOR BARACK OBAMA

Recent polls show a 52% approval rating for Barack Obama in race relations. It is the only issue out of the below thirteen where he has a majority approval. In eight of the thirteen, results show a majority disapproval.

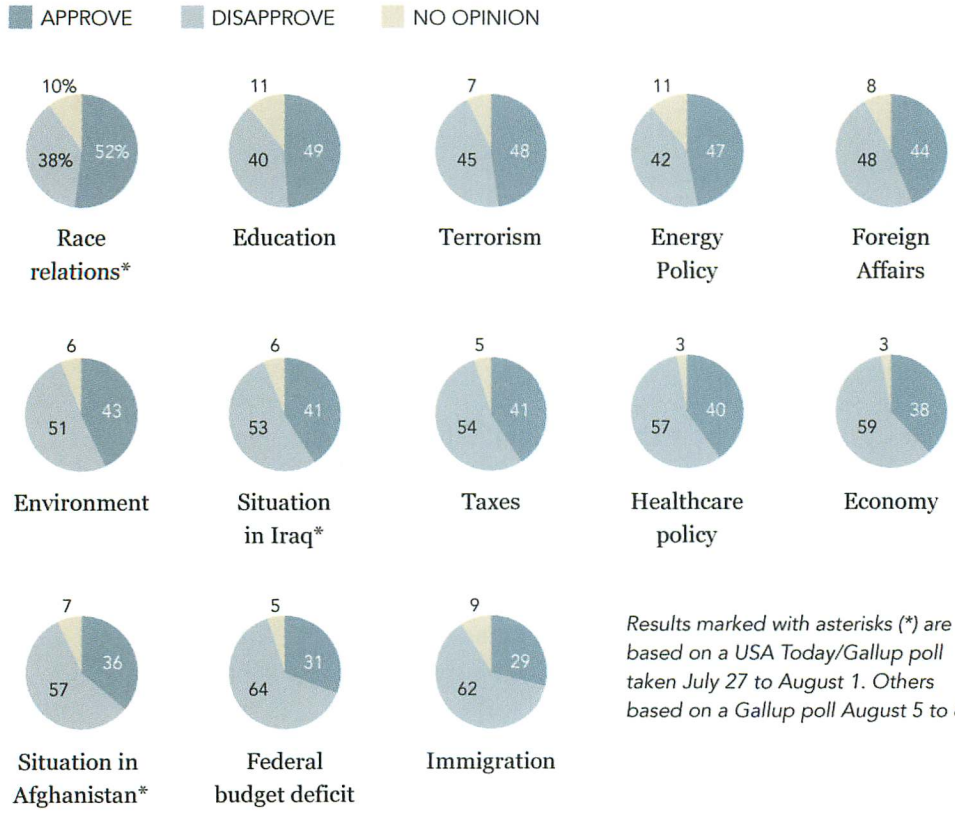


FIGURE 5-12 Series of pie charts

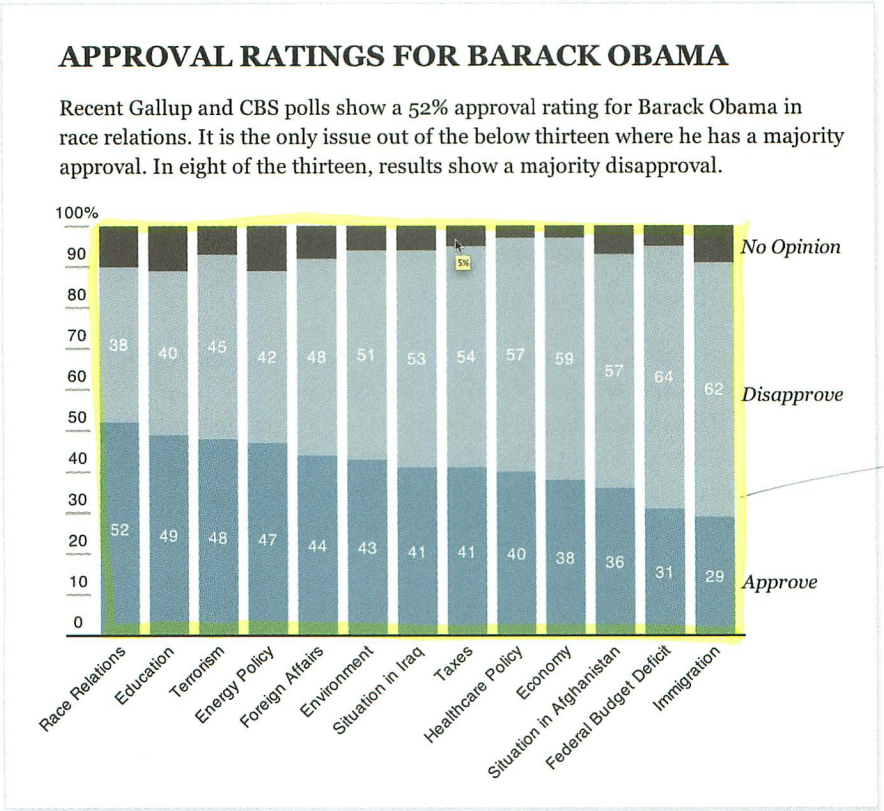


FIGURE 5-13 Interactive stacked bar chart in Protovis

To start, set up the HTML page and load the necessary Protovis JavaScript file.

```
<html>
<head>
    <title>Stacked Bar Chart</title>
    <script type="text/javascript" src="protovis-r3.2.js"></script>
</head>
<body>
    <div id="figure-wrapper">
        <div id="figure">

    </div><!-- @end figure -->
    </div><!-- @end figure-wrapper -->
</body>
</html>
```

This should look familiar. You did the same thing to make a donut chart with Protovis. The only difference is that the title of the page is "Stacked Bar Chart" and there's an additional with a "figure-wrapper" id. We also haven't added any CSS yet to style the page, because we're saving that for later.

Now on to JavaScript. Within the figure , load and prepare the data (Obama ratings, in this case) in arrays.

```
<script type="text/javascript+protovis">
    var data = {
        "Issue":["Race Relations","Education","Terrorism","Energy Policy",
            "Foreign Affairs","Environment","Situation in Iraq",
            "Taxes","Healthcare Policy","Economy","Situation in Afghanistan",
            "Federal Budget Deficit","Immigration"],
        "Approve":[52,49,48,47,44,43,41,41,40,38,36,31,29],
        "Disapprove":[38,40,45,42,48,51,53,54,57,59,57,64,62],
        "None":[10,11,7,11,8,6,6,5,3,3,7,5,9]
    };
</script>
```

You can read this as 52 percent and 38 percent approval and disapproval ratings, respectively, for race relations. Similarly, there were 49 percent and 40 percent approval and disapproval ratings for education.

To make it easier to code the actual graph, you can split the data and store it in two variables.

```
var cat = data.Issue;
var data = [data.Approve, data.Disapprove, data.None];
```

The issues array is stored in cat and the data is now an array of arrays.

Set up the necessary variables for width, height, scale, and colors with the following:

```
var w = 400,
h = 250,
x = pv.Scale.ordinal(cat).splitBanded(0, w, 4/5),
y = pv.Scale.linear(0, 100).range(0, h),
fill = ["#809EAD", "#B1C0C9", "#D7D6CB"];
```

The graph will be 400 pixels wide and 250 pixels tall. The horizontal scale is ordinal, meaning you have set categories, as opposed to a continuous scale. The categories are the issues that the polls covered. Four-fifths of

the graph width will be used for the bars, whereas the rest is for padding in between the bars.

The vertical axis, which represents percentages, is a linear scale from 0 to 100 percent. The height of the bars can be anywhere in between 0 pixels to the height of the graph, or 250 pixels.

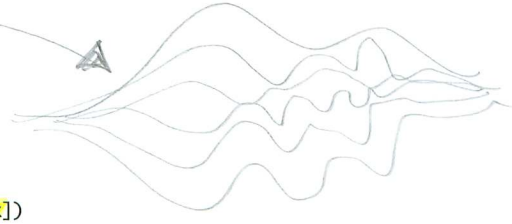
Finally, fill is specified in an array with hexadecimal numbers. That's dark blue for approval, light blue for disapproval, and light gray for no opinion. You can change the colors to whatever you like.

Next step: Initialize the visualization with specified width and height. The rest provides padding around the actual graph, so you can fit axis labels. For example, `bottom(90)` moves the zero-axis up 90 pixels. Think of this part as setting up a blank canvas.

```
var vis = new pv.Panel()
    .width(w)
    .height(h)
    .bottom(90)
    .left(32)
    .right(10)
    .top(15);
```

▶ If you're not sure what colors to use, ColorBrewer at `http://colorbrewer2.org` is a good place to start. The tool enables you to specify the number of colors you want to use and the type of colors, and it provides a color scale that you can copy in various formats. 0to255 at `http://0to255.com` is a more general color tool, but I use it often.

To add stacked bars to your canvas, Protovis provides a special layout for stacked charts appropriately named Stack. Although you use this for a stacked bar chart in this example, the layout can also be used with stacked area charts and streamgraphs. Store the new layout in the "bar" variable.



```
var bar = vis.add(pv.Layout.Stack)
    .layers(data)
    .x(function() x(this.index))
    .y(function(d) y(d))
    .layer.add(pv.Bar)
        .fillStyle(function() fill[this.parent.index])
    .width(x.range().band)
    .title(function(d) d + "%")
    .event("mouseover", function() this.fillStyle("#555"))
    .event("mouseout", function()
        this.fillStyle(fill[this.parent.index]));
```

to highlight

function

Another way to think about this chart is as a set of three layers, one each for approval, disapproval, and no opinion. Remember how you structured

those three as an array of three arrays? That goes in `layers()`, where x and y follow the scales that you already made.

For each layer, add bars using `pv.Bar`. Specify the fill style with `fillStyle()`. Notice that we used a function that goes by `this.parent.index`. This is so that the bar is colored by what layer it belongs to, of which there are three. If you were to use `this.index`, you would need color specifications for every bar, of which there are 39 (3 times 13). The width of each bar is the same across, and you can get that from the ordinal scale you already specified.

TIP

Interaction in Protovis isn't just limited to mouse over and out. You can also set events for things such as click and double-click. See Protovis documentation for more details.

The final three lines of the preceding code are what make the graph interactive. Using `title()` in Protovis is the equivalent of setting the title attribute of an HTML element such as an image. When you roll over an image on a web page, a tooltip shows up if you set the title. Similarly, a tooltip appears as you place the mouse pointer over a bar for a second. Here simply make the tooltip show the percentage value that the bar represents followed with a percent sign (%).

To make the layers highlight whenever you mouse over a bar, use `event()`. On "mouseover" the fill color is set to a dark gray (#555), and when the mouse pointer is moved off, the bar is set to its original color using the "mouseout" event.

To make the graph appear, you need to render it. Enter this at the end of our JavaScript.

```
vis.render();
```

This basically says, "Okay, we've put together all the pieces. Now draw the visualization." Open the page in your web browser (a modern one, such as Firefox or Safari), and you should see something like Figure 5-14.

Mouse over a bar, and the layer appears highlighted. A tooltip shows up, too. A few things are still missing, namely the axes and labels. Add those now.

In Figure 5-13, a number of labels are on the bars. It's only on the larger bars though, that is, not the gray ones. Here's how to do that. Keep in mind that this goes before `vis.render()`. Always save rendering for last.

```
bar.anchor("center").add(pv.Label)
    .visible(function(d) d > 11)
    .textStyle("white")
    .text(function(d) d.toFixed(0));
```

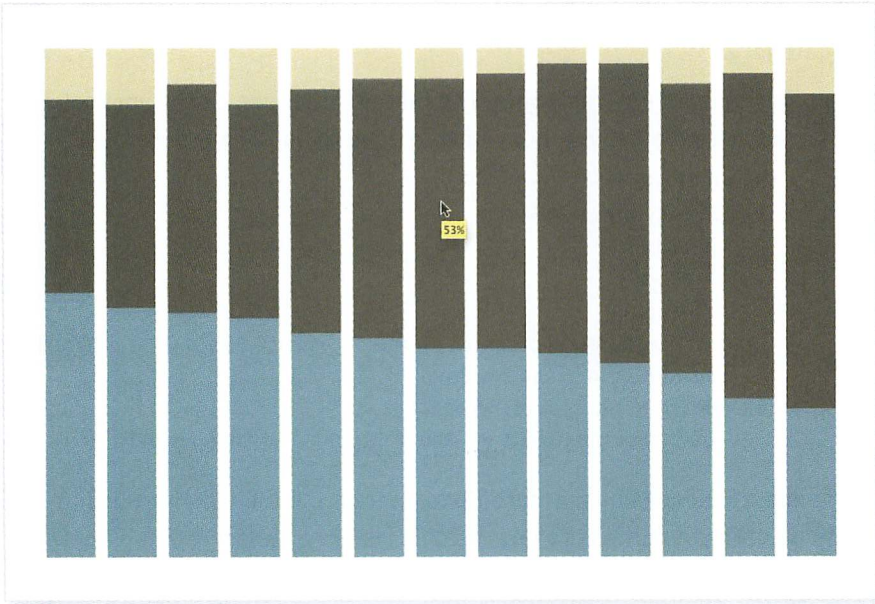


FIGURE 5-14 Stacked bar graph without any labels

For each bar, look to see if it is greater than 11 percent. If it is, a white label that reads the percentage rounded to the nearest integer is drawn in the middle of the bar.

Now add the labels for each issue on the x-axis. Ideally, you want to make all labels read horizontally, but there is obviously not enough space to do that. If the graph were a horizontal bar chart, you could fit horizontal labels, but for this you want to see them at 45-degree angles. You can make the labels completely vertical, but that'd make them harder to read.

```
bar.anchor("bottom").add(pv.Label)
    .visible(function() !this.parent.index)
    .textAlign("right")
    .top(260)
    .left(function() x(this.index)+20)
    .textAngle(-Math.PI / 4)
    .text(function() cat[this.index]);
```

This works in the same way you added number labels to the middle of each bar. However, this time around add labels only to the bars at the bottom, that is, the ones for approval. Then right-align the text and set their absolute vertical position with `textAlign()` and `top()`. Their x-position is

based on what bar they label, each is rotated 45 degrees, and the text is the category.

That gives the categorical labels. The labels for values on the vertical axis are added in the same way, but you also need to add tick marks.

```
vis.add(pv.Rule)
    .data(y.ticks())
    .bottom(y)
    .left(-15)
    .width(15)
    .strokeStyle(function(d) d > 0 ? "rgba(0,0,0,0.3)" : "#000")
    .anchor("top").add(pv.Label)
    .bottom(function(d) y(d)+2)
    .text(function(d) d == 100 ? "100%" : d.toFixed(0));
```

This adds a Rule, or lines, according to `y.ticks()`. If the tick mark is for anything other than the zero line, its color is gray. Otherwise, the tick is black. The second section then adds labels on top of the tick marks.

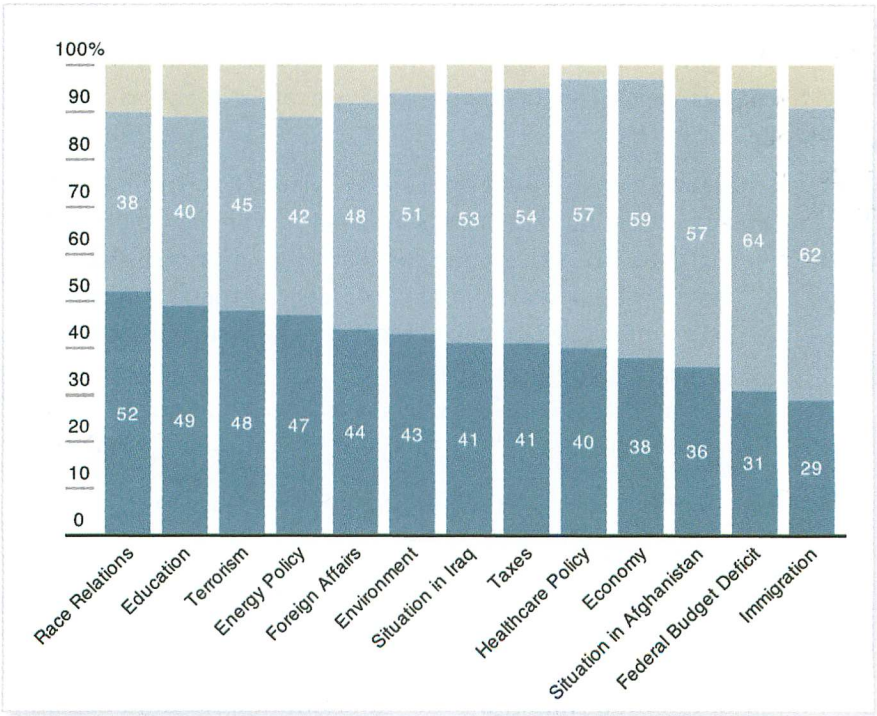


FIGURE 5-15 Adding the horizontal axis

You're still missing the horizontal axis, so add another Rule, separately to get what you see in Figure 5-15.

```
vis.add(pv.Rule)
    .bottom(y)
    .left(-15)
    .right(0)
    .strokeStyle("#000")
```

Lead-in copy and remaining labels are added with HTML and CSS. There are entire books for web design though, so I'll leave it at that. The cool thing here is that you can easily combine the HTML and CSS with Protovis, which is just JavaScript and still make it look seamless.

▶ To see and interact with the stacked bar graph, visit `http://book.flowingdata.com/ch05/stacked-bar.html`. Check out the source code to see how HTML, CSS, and JavaScript fit together.

Hierarchy and Rectangles

In 1990, Ben Shneiderman, of the University of Maryland, wanted to visualize what was going on in his always-full hard drive. He wanted to know what was taking up so much space. Given the hierarchical structure of directories and files, he first tried a tree diagram. It got too big too fast to be useful though. Too many nodes. Too many branches.

The treemap was his solution. As shown in Figure 5-16, it's an area-based visualization where the size of each rectangle represents a metric. Outer rectangles represent parent categories, and rectangles within the parent are like subcategories. You can use a treemap to visualize straight-up proportions, but to fully put the technique to use, it's best served with hierarchical, or rather, tree-structured data.

▶ See `http://datafl.ws/11m` for a full history of treemaps and additional examples described by the creator, Ben Shneiderman.

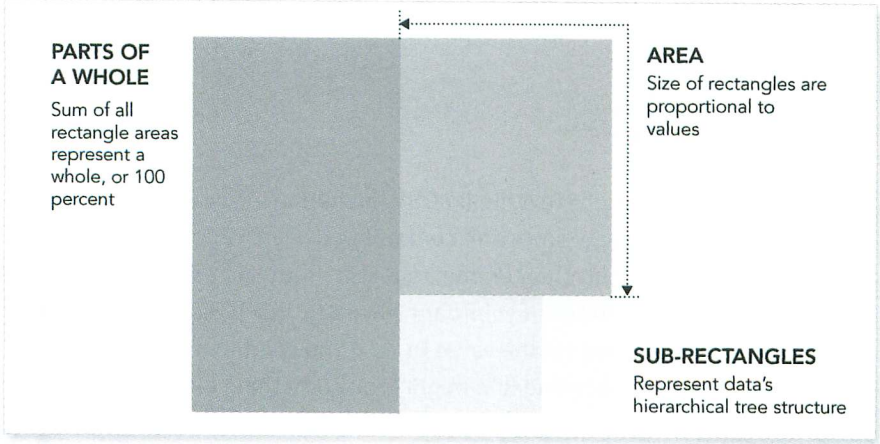


FIGURE 5-16 Treemap generalized

TIP

R is an open-source software environment for statistical computing. You can download it for free from www.r-project.org/. The great thing about R is that there is an active community around the software that is always developing packages to add functionality. If you're looking to make a static chart, and don't know where to start, the R archives are a great place to look.

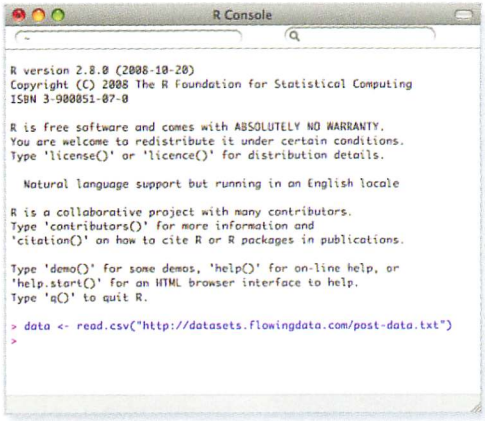
CREATE A TREEMAP

Illustrator doesn't have a Treemap tool, but there is an R package by Jeff Enos and David Kane called Portfolio. It was originally intended to visualize stock market portfolios (hence the name), but you can easily apply it to your own data. Look at page views and comments of 100 popular posts on FlowingData and separate them by their post categories, such as visualization or data design tips.

As always, the first step is to load the data into R. You can load data directly from your computer or point to a URL. Do the latter in this example because the data is already available online. If, however, you want to do the former when you apply the following steps to your own data, just make sure you put your data file in your working directory in R. You can change your working directory through the Miscellaneous menu.

Loading a CSV file from a URL is easy. It's only one line of code with the read.csv()function in R (Figure 5-17).

```
posts <- read.csv("http://datasets.flowingdata.com/post-data.txt")
```



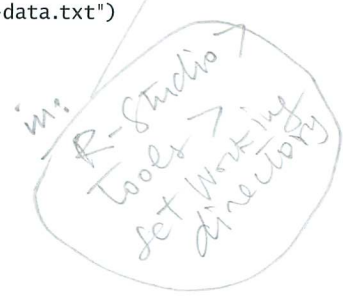


FIGURE 5-17 Loading CSV in R

Easy, right? We've loaded a text file (in CSV format) using read.csv() and stored the values for page views and comments in a variable called *posts*. As mentioned in the previous chapter, the read.csv() function assumes that your data file is comma-delimited. If your data were say, tab-delimited, you would use the sep argument and set the value to \t. If you want to load the data from a local directory, the preceding line might look something like this.

```
posts <- read.csv("post-data.txt")
```

This is assuming you've changed your working directory accordingly. For more options and instructions on how to load data using the `read.csv()` function, type the following in the R console:

```
?read.csv
```

Moving on, now that the data is stored in the *posts* variable, enter the following line to see the first five rows of the data.

```
posts[1:5,]
```

You should see four columns that correspond to the original CSV file, with *id*, *views*, *comments*, and *category*. Now that the data is loaded in R, make use of the Portfolio package. Try loading it with the following:

```
library(portfolio)
```

Get an error? You probably need to install the package before you begin:

```
install.packages("portfolio")
```

You should load the package now. Go ahead and do that. Loaded with no errors? Okay, good, now go to the next step.

The Portfolio package does the hard work with a function called `map.market()`. The function takes several arguments, but you use only five of them.

```
map.market(id=data$id, area=posts$views, group=posts$category,
    color=posts$comments, main="FlowingData Map")
```

The `id` is the column that indicates a unique point, and you tell R to use *views* to decide the areas of the rectangles in the treemap, the categories to form groups, and the number of comments in a post to decide color. Finally, enter **FlowingData Map** as the main title. Press Enter on your keyboard to get a treemap, as shown in Figure 5-18.

It's still kind of rough around the edges, but the base and hierarchy is set up, which is the hard part. Just like you specified, rectangles, each of which represent a post, are sized by the number of page views and sorted by category. Brighter shades of green indicate posts that received more comments; posts with a lot of views don't necessarily get the most comments.

You can save the image as a PDF in R and then open the file in Illustrator. All regular edit options apply. You can change stroke and fill colors, fonts, remove anything extraneous, and add comments if you like.

TIP

You can also install packages in R through the user interface. Go to Packages & Data ➪ Package Installer. Click Get List, and then find the package of interest. Double-click to install.

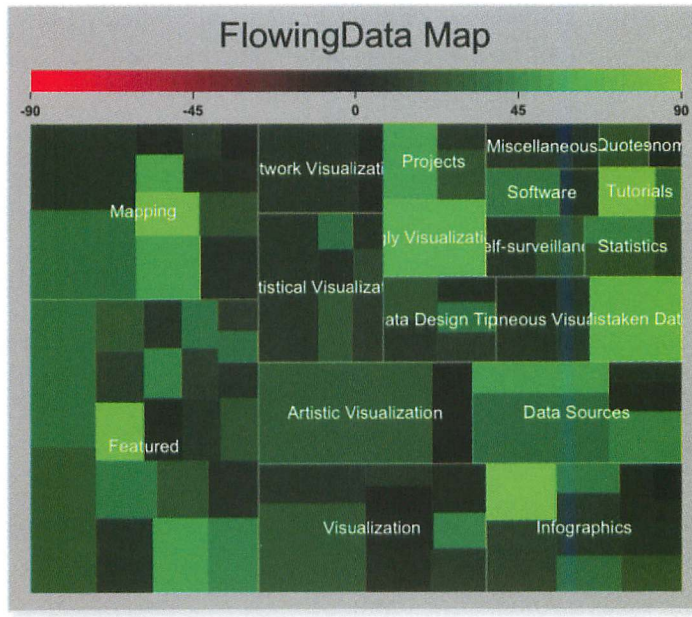


FIGURE 5-18 Default treemap in R

▶ *The New York Times* used an animated treemap to show changes in the stock market during the financial crisis in its piece titled "How the Giants of Finance Shrank, Then Grew, Under the Financial Crisis." See it in action at `http://nyti.ms/9JUkWL`.

For this particular graphic you need to change the scale of the legend that goes from –90 to 90. It doesn't make sense to have a negative scale because there's no such thing as a negative number of comments. You can also fix the labels. Some of them are obscured in the small rectangles. Size the labels by popularity instead of the uniform scale it now has using the Selection tool. Also thicken the category borders so that they're more prominent. That should give you something like Figure 5-19.

There you go. The graphic is much more readable now with unobscured labeling and a color scale that makes more sense. You also got rid of the dark gray background, which makes it cleaner. Oh, and of course, you included a title and lead-in to briefly explain what the graphic shows.

Because the Portfolio package does most of the heavy lifting, the only tough part in applying this to your own data is getting it into the right format. Remember, you need three things. You need a unique id for each row, a metric to size rectangles, and parent categories. Optionally, you can use a fourth metric to color your rectangles. Check out Chapter 2, "Handling Data," for instructions on how to get your data into the format you need.

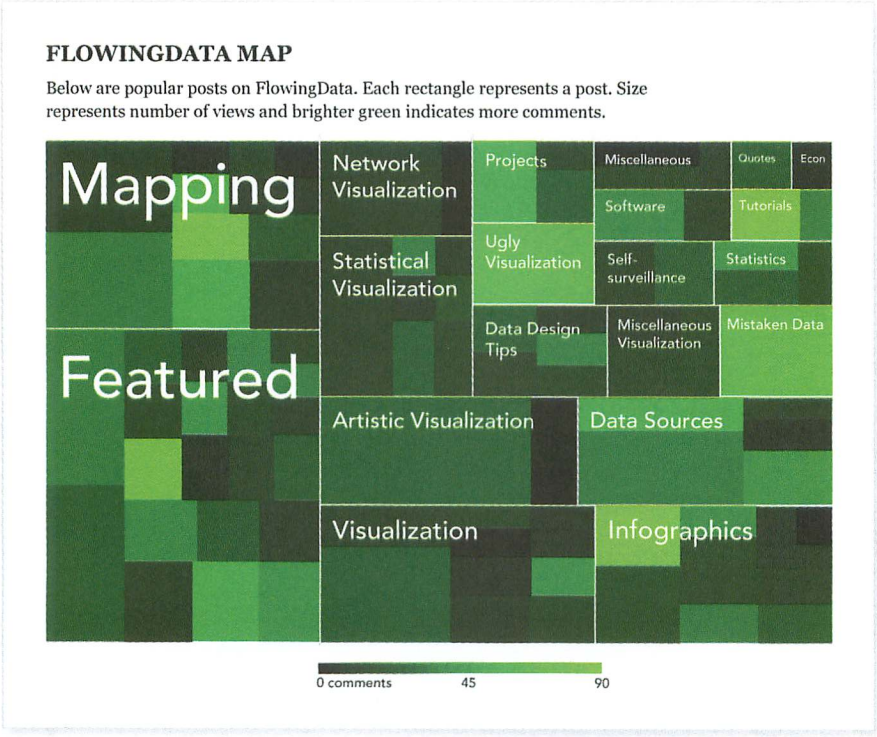


FIGURE 5-19 Revised treemap from R to Illustrator

Proportions over Time

Often you'll have a set of proportions over time. Instead of results for a series of questions from a single polling session, you might have results from the same poll run every month for a year. You're not just interested in individual poll results; you also want to see how views have changed over time. How has opinion changed from one year ago until now?

This doesn't just apply to polls, of course. There are plenty of distributions that change over time. In the following examples, you take a look at the distribution of age groups in the United States from 1860 to 2005. With improving healthcare and average family size shrinking, the population as a whole is living longer than the generation before.

Stacked Continuous

Imagine you have several time series charts. Now stack each line on top of the other. Fill the empty space. What you have is a stacked area chart, where the horizontal axis is time, and the vertical axis is a range from 0 to 100 percent, as shown in Figure 5-20.

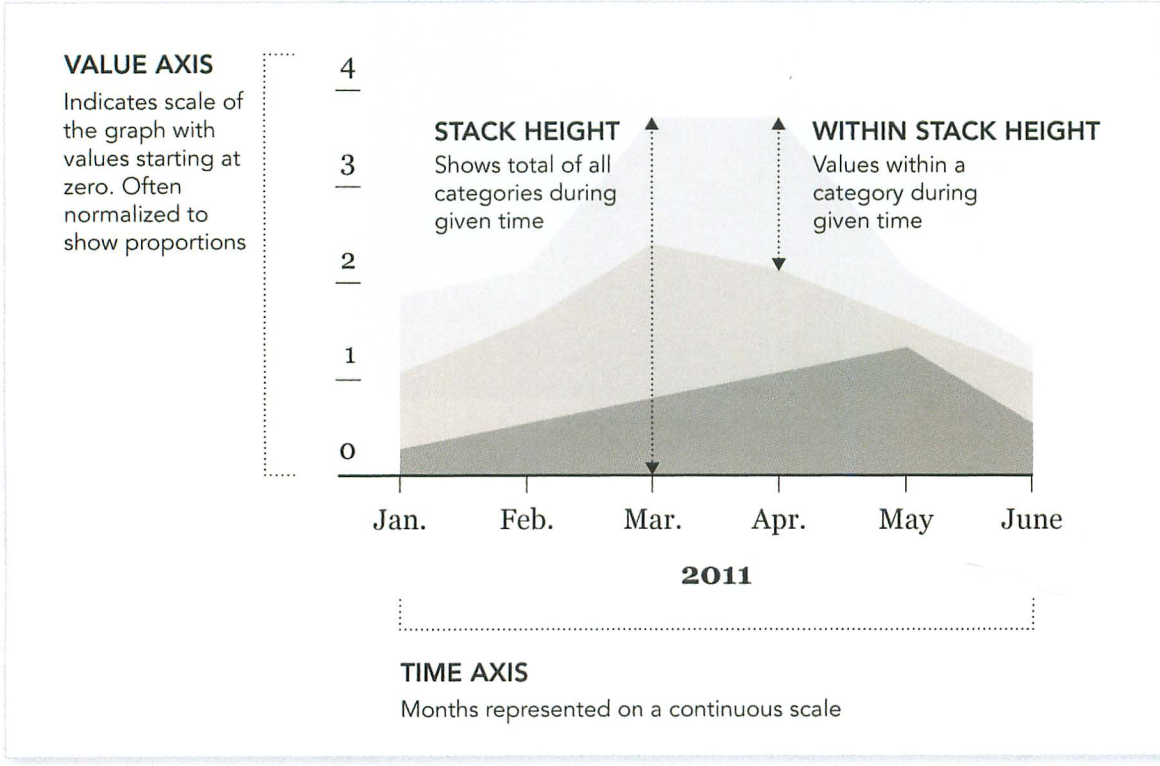


FIGURE 5-20 Stacked area chart generalized

So if you were to take a vertical slice of the area chart, you would get the distribution of that time slice. Another way to look at it is as a series of stacked bar charts connected by time.

CREATE A STACKED AREA CHART

In this example, look at the aging population. Download the data at `http://book.flowingdata.com/ch05/data/us-population-by-age.xls`. Medicine and healthcare have improved over the decades, and the average lifespan

continues to rise. As a result, the percentage of the population in older age brackets has increased. By how much has this age distribution changed over the years? Data from the U.S. Census Bureau can help you see via a stacked area chart. You want to see how the proportion of older age groups has increased and how the proportion of the younger age groups has decreased.

You can do this in a variety of ways, but first use Illustrator. For the stacked area graph, it comes in the form of the Area Graph tool (Figure 5-21).

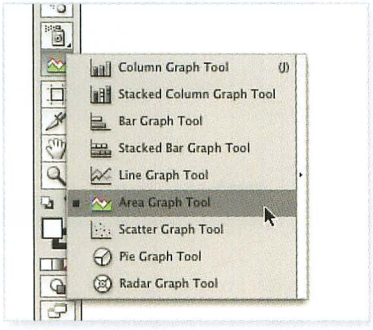


FIGURE 5-21 Area Graph Tool

Click and drag somewhere on a new document, and enter the data in the spreadsheet that pops up. You're familiar with the load data, generate graphic, and refine process now, right?

You can see a stacked area chart, as shown in Figure 5-22, after you enter the data.

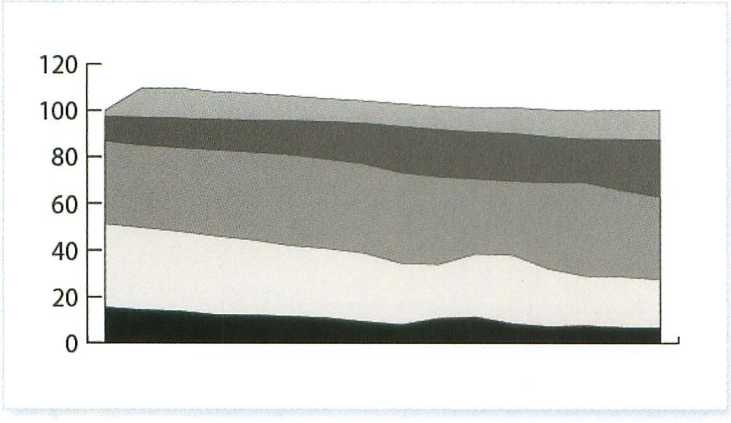


FIGURE 5-22 Default stacked area chart in Illustrator

The top area goes above the 100 percent line. This happened because the stacked area graph is not just for normalized proportions or a set of values that add up to 100 percent. It can also be used for raw values, so if you want each time slice to add up to 100 percent, you need to normalize the data. The above image was actually from a mistake on my part; I entered the data incorrectly. Oops. A quick fix, and you can see the graph

in Figure 5-23. Although, you probably entered the data correctly the first time, so you're already here.

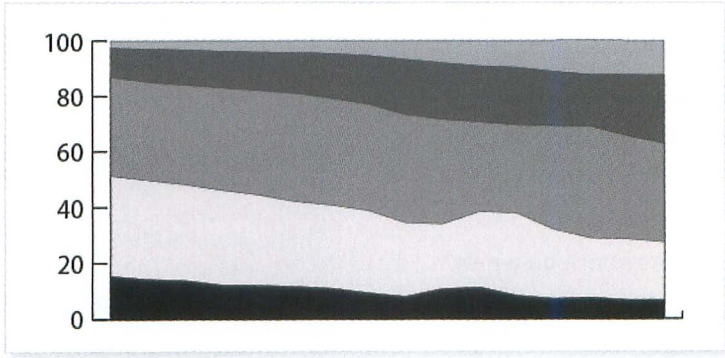


FIGURE 5-23 Fixed area chart

TIP

Be careful when you enter data manually. A lot of silly mistakes come from transferring data from one source to another.

Keep an eye out for stuff like this in your graph design though. It's better to spot typos and small data entry errors in the beginning than it is to finish a design and have to backtrack to figure out where things went wrong.

Now that you have a proper base, clean up the axis and lines. Make use of the Direct Selection tool to select specific elements. I like to remove the vertical axis line and leave thinner tick marks for a cleaner, less clunky look, and add the percentage sign to the numbers because that's what we're dealing with. I also typically change the stroke color of the actual graph fills from the default black to a simpler white. Also bring in some shades of blue. That takes you to Figure 5-24.

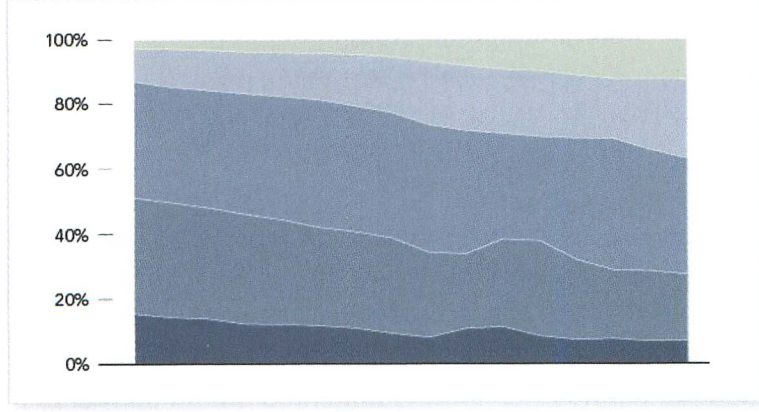


FIGURE 5-24 Modified colors from default

Again, this is just my design taste, and you can do what you want. Color selection can also vary by case. The more graphs that you design, the better feel you'll develop for what you like and what works best.

Are you missing anything else? Well, there are no labels for the horizontal axis. Now put them in. And while you're at it, label the areas to indicate the age groups (Figure 5-25).

TIP

Use colors that fit your theme and guide your readers' eyes with varying shades.

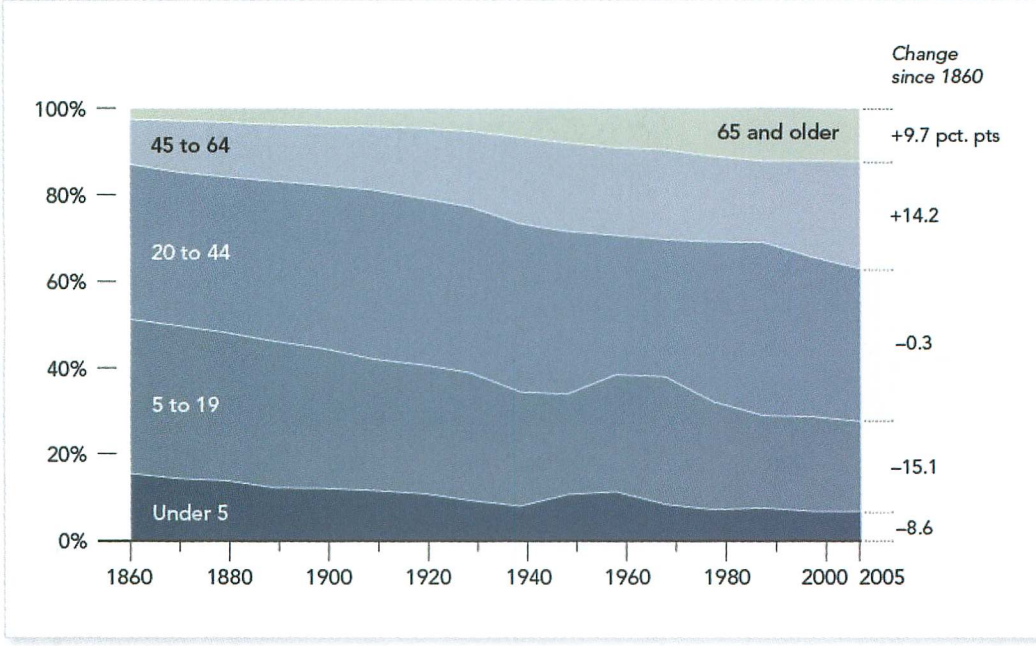


FIGURE 5-25 Labeled stacked area chart

I also added annotation on the right of the graph. What we're most interested in here is the change in age distribution. We can see that from the graph, but the actual numbers can help drive the point home.

Lastly, put in the title and lead-in copy, along with the data source on the bottom. Tweak the colors of the right annotations a little bit to add some more meaning to the display, and you have the final graphic, as shown in Figure 5-26.

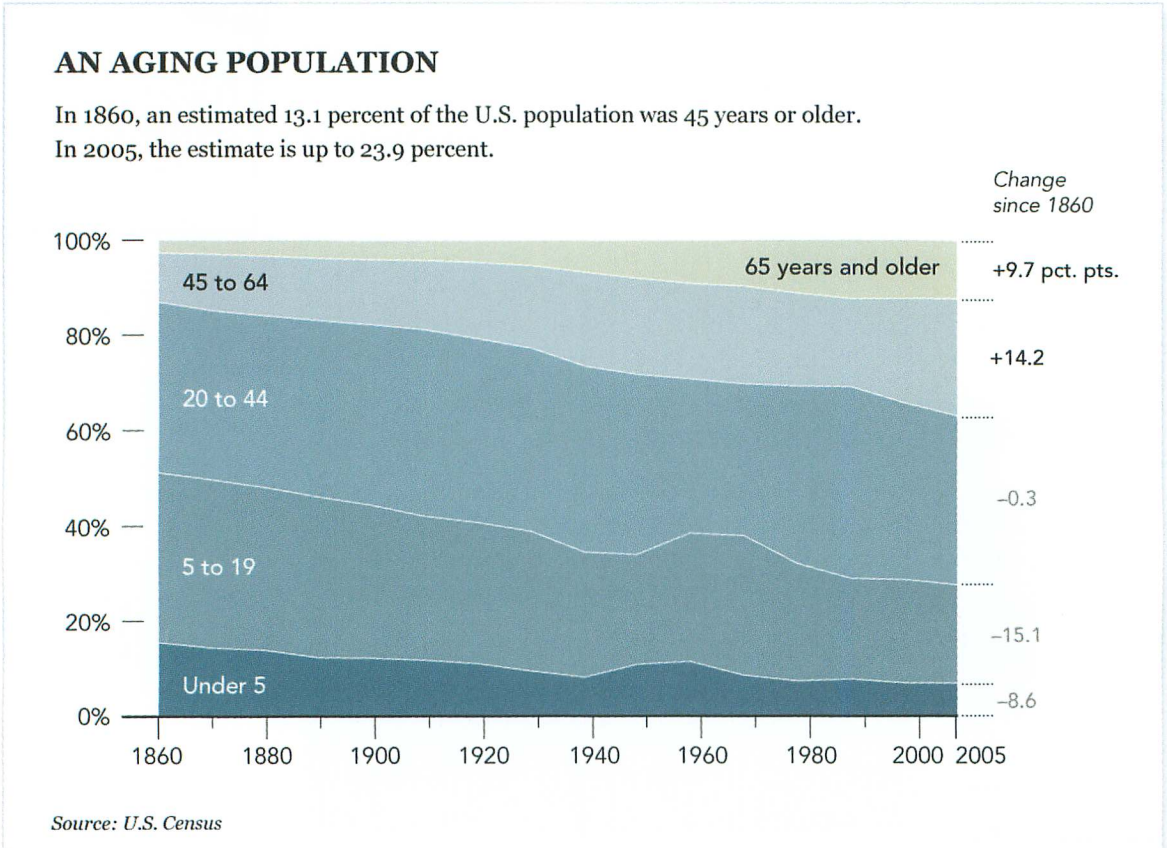


FIGURE 5-26 Final stacked area chart

CREATE AN INTERACTIVE STACKED AREA CHART

One of the drawbacks to using stacked area charts is that they become hard to read and practically useless when you have a lot of categories and data points. The chart type worked for age breakdowns because there were only five categories. Start adding more, and the layers start to look like thin strips. Likewise, if you have one category that has relatively small counts, it can easily get dwarfed by the more prominent categories. Making the stacked area graph interactive, however, can help solve that problem.

You can provide a way for readers to search for categories and then adjust the axis to zoom in on points of interest. Tooltips can help readers see values in places that are too small to place labels. Basically, you can take

data that wouldn't work as a static stacked area chart, but use it with an interactive chart, and make it easy to browse and explore. You could do this in JavaScript with Protovis, but for the sake of learning more tools (because it's super fun), use Flash and ActionScript.

NOTE

Online visualization has slowly been shifting away from Flash toward JavaScript and HTML5, but not all browsers support the latter, namely Internet Explorer. Also, because Flash has been around for years, there are libraries and packages that make certain tasks easier than if you were to try to do it with native browser functionality.

▶ The *NameVoyager* by Martin Wattenberg made the interactive stacked area chart popular. It is used to show baby names over time, and the graph automatically updates as you type names in the search box. Try it out at `www.babynamewizard.com/voyager`.

Luckily you don't have to start from scratch. Most of the work has already been done for you via the Flare visualization toolkit, designed and maintained by the UC Berkeley Visualization Lab. It's an ActionScript library, which was actually a port of a Java visualization toolkit called Prefuse. We'll work off one of the sample applications on the Flare site, JobVoyager, which is like NameVoyager, but an explorer for jobs. After you get your development environment set up, it's just a matter of switching in your data and then customizing the look and feel.

You can write the code completely in ActionScript and then compile it into a Flash file. Basically this means you write the code, which is a language that you understand, and then use a compiler to translate the code into bits so that your computer, or the Flash player, can understand what you told it to do. So you need two things: a place to write and a way to compile.

The hard way to do this is to write code in a standard text editor and then use one of Adobe's free compilers. I say hard because the steps are definitely more roundabout, and you have to install separate things on your computer.

The easy way to do this, and the way I highly recommend if you're planning on doing a lot of work in Flash and ActionScript, is to use Adobe Flex Builder. It makes the tedious part of programming with ActionScript quicker, because you code, compile, and debug all in the same place. The downside is that it does cost money, although it's free for students.

NOTE

Download Flare for free at `http://flare.prefuse.org/`.

If you're not sure if it's worth the money, you can always download a free trial and make your decision later. For the stacked area chart example, I'll explain the steps you have to take in Flex Builder.

NOTE

At the time of this writing, Adobe changed the name of Flex Builder to Flash Builder. They are similar but there are some variations between the two. While the following steps use the former, you can still do the same in the latter. Download Flash Builder at `www.adobe.com/products/flashbuilder/`. Be sure to take advantage of the student discount. Simply provide a copy of your student ID, and you get a free license. Alternatively, find an old, lower-priced copy of Flex Builder.

When you've downloaded and installed Flex Builder, go ahead and open it; you should see a window, as shown in Figure 5-27.

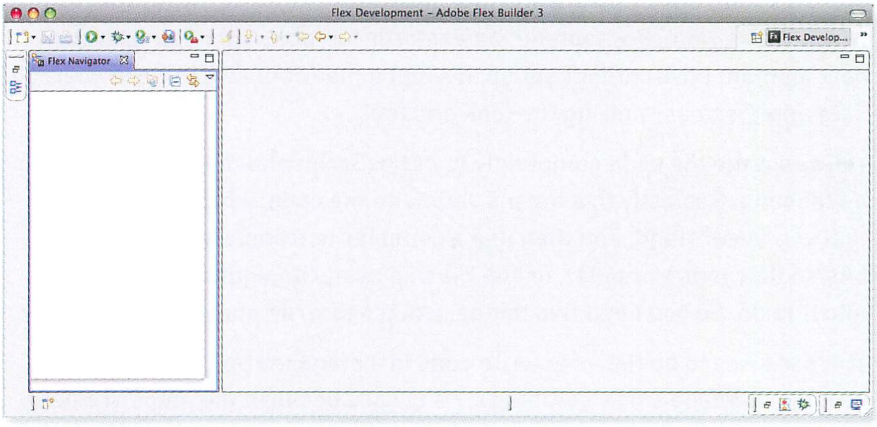


FIGURE 5-27 Initial window on opening Flex Builder

Right-click the Flex Navigator (left sidebar) and click Import. You'll see a pop-up that looks like Figure 5-28.

Select Existing Projects into Workspace and click Next. Browse to where you put the Flare files. Select the flare directory, and then make sure Flare is checked in the project window, as shown in Figure 5-29.

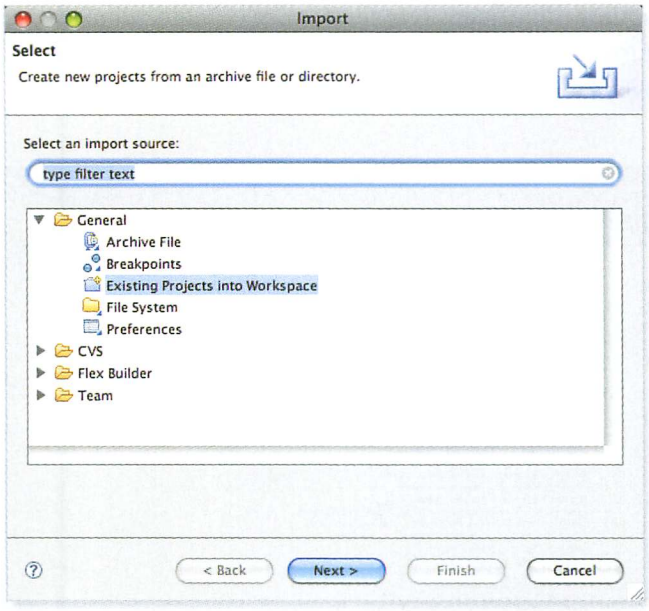


FIGURE 5-28 Import window in Flex Builder

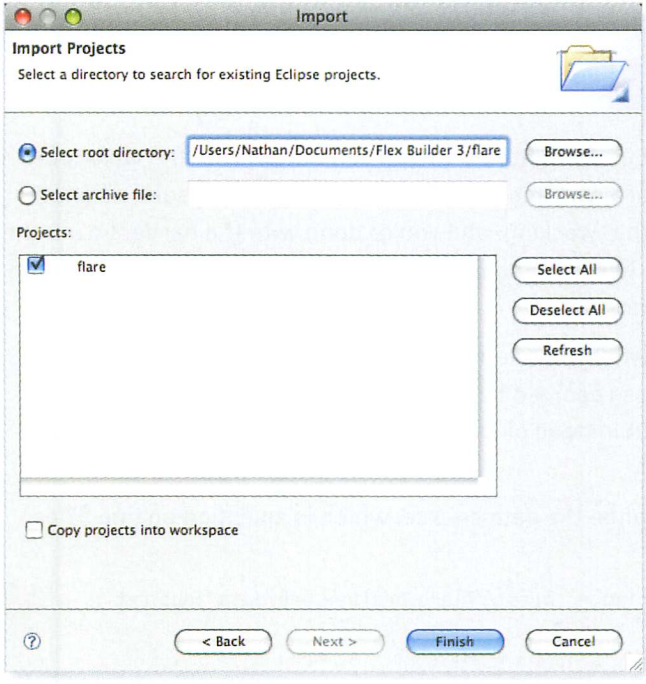


FIGURE 5-29 Existing projects window

Do the same thing with the flare.apps folder. Your Flex Builder window should look like Figure 5-30 after you expand the `flare.apps/flare/apps/` folder and click JobVoyager.as.

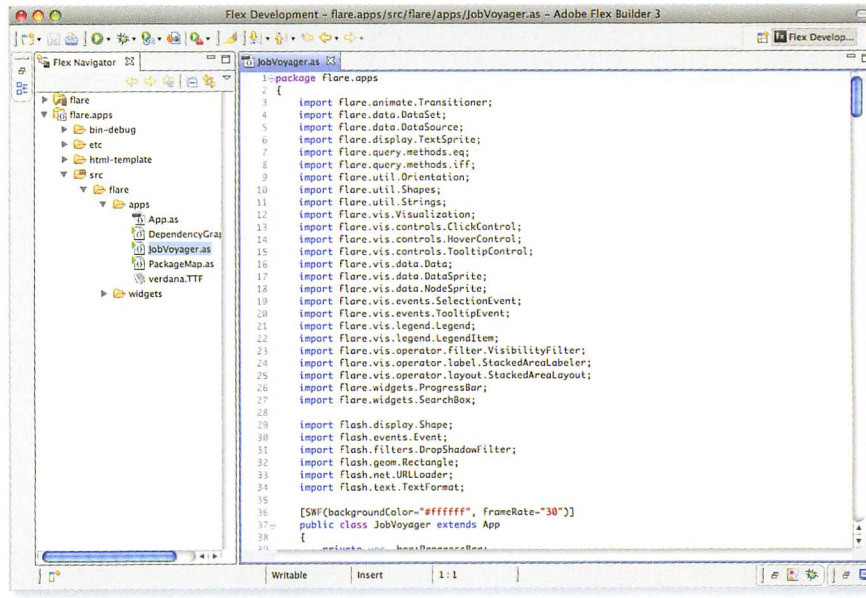


FIGURE 5-30 JobVoyager code opened

If you click the run button right now (the green button with the white play triangle at the top left), you should see the working JobVoyager, as shown in Figure 5-31. Get that working, and you're done with the hardest part: the setup. Now you just need to plug in your own data and customize it to your liking. Sound familiar?

▶ Visit `http://datafl.ws/16r` to try the final visualization and to see how the explorer works with consumer spending.

Figure 5-32 shows what you're after. It's a voyager for consumer spending from 1984 to 2008, as reported by the U.S. Census Bureau. The horizontal axis is still years, but instead of jobs, there are spending categories such as housing and food.

Now you need to change the data source, which is specified on line 57 of JobVoyager.as.

```
private var _url:String = "http://flare.prefuse.org/data/jobs.txt";
```

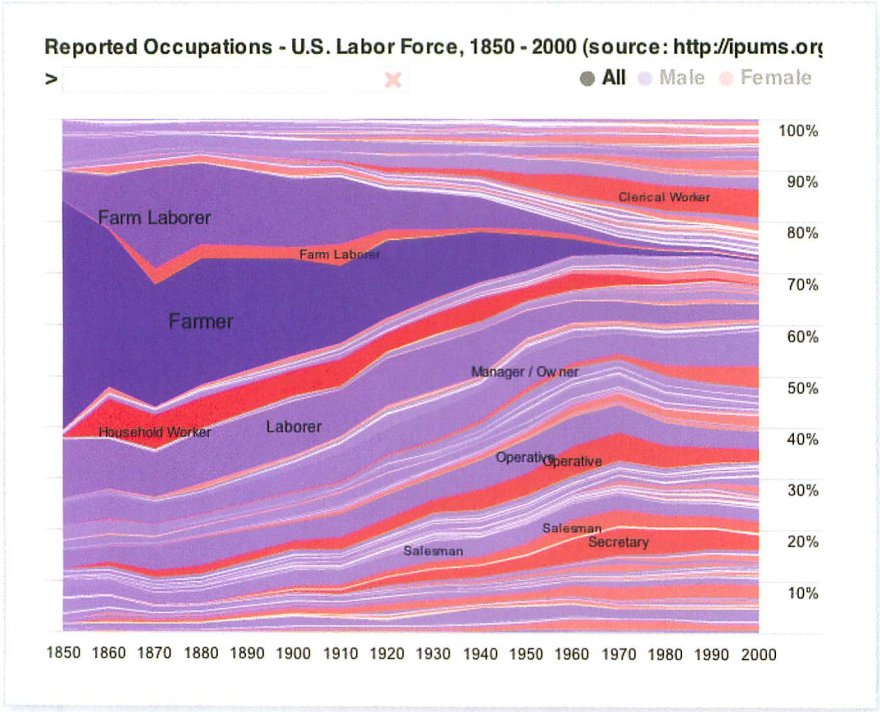


FIGURE 5-31 JobVoyager application

Change the `_url` to point at the spending data available at `http://datasets.flowingdata.com/expenditures.txt`. Like `jobs.txt`, the data is also a tab-delimited file. The first column is year, the second category, and the last column is expenditure.

```
private var _url:String =
    "http://datasets.flowingdata.com/expenditures.txt";
```

Now the file will read in your spending data instead of the data for jobs. Easy stuff so far.

The next two lines, line 58 and 59, are the column names, or in this case, the distinct years that job data was available. It's by decade from 1850 to 2000. You could make things more robust by finding the years in the loaded data, but because the data isn't changing, you can save some time and explicitly specify the years.

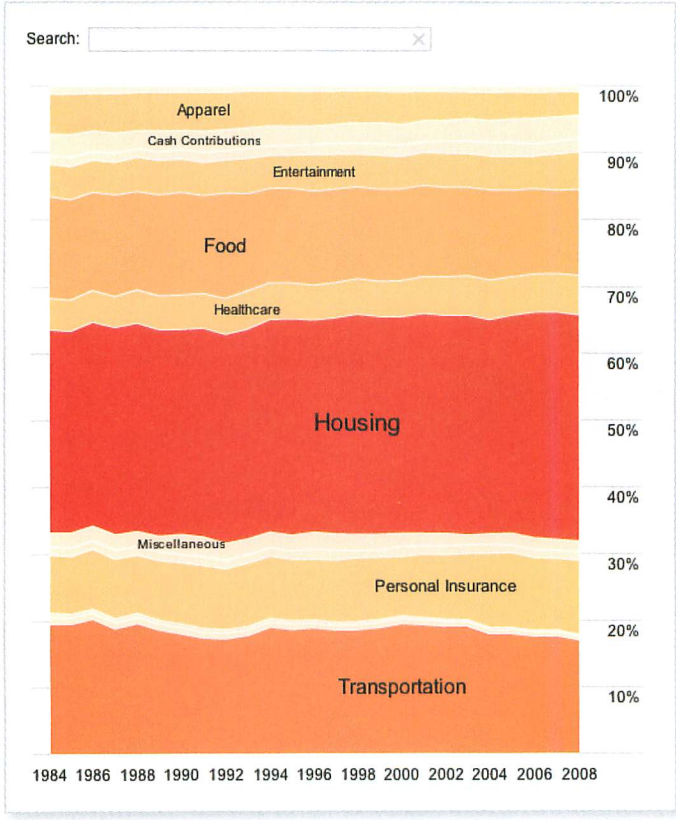


FIGURE 5-32 Interactive voyager for consumer spending

The expenditures data is annual from 1984 to 2008, so Change lines 58–59 accordingly.

```
private var _cols:Array =
    [1984,1985,1986,1987,1988,1989,1990,1991,1992,
    1993,1994,1995,1996,1997,1998,1999,2000,2001,2002,
    2003,2004,2005,2006,2007,2008];
```

Next change references to the data headers. The original data file (`jobs.txt`) has four columns: year, occupation, people, and sex. The spending data has only three columns: year, category, and expenditure. You need to adapt the code to this new data structure.

Luckily, it's easy. The year column is the same, so you just need to change any people references to expenditure (vertical axis) and any occupation references to category (the layers). Finally, remove all uses of gender.

At line 74 the data is reshaped and prepared for the stacked area chart. It specifies by occupation and sex as the categories (that is, layers) and uses year on the x-axis and people on the y-axis.

```
var dr:Array = reshape(ds.nodes.data, ["occupation","sex"],
    "year", "people", _cols);
```

Change it to this:

```
var dr:Array = reshape(ds.nodes.data, ["category"],
    "year", "expenditure", _cols);
```

You only have one category (sans sex), and that's uh, category. The x-axis is still year, and the y-axis is expenditure.

Line 84 sorts the data by occupation (alphabetically) and then sex (numerically). Now just sort by category:

```
data.nodes.sortBy("data.category");
```

Are you starting to get the idea here? Mostly everything is laid out for you. You just need to adjust the variables to accommodate the data.

Line 92 colors layers by sex, but you don't have that split in the data, so you don't need to do that. Remove the entire row:

```
data.nodes.setProperty("fillHue", iff(eq("data.sex",1), 0.7, 0));
```

We'll come back to customizing the colors of the stacks a little later.

Line 103 adds labels based occupation:

```
_vis.operators.add(new StackedAreaLabeler("data.occupation"));
```

You want to label based on spending category, so change the line accordingly:

```
_vis.operators.add(new StackedAreaLabeler("data.category"));
```

Lines 213–231 handle filtering in JobVoyager. First, there's the male/female filter; then there's the filter by occupation. You don't need the former, so you can get rid of lines 215–218 and then make line 219 a plain `if` statement.

TIP

There's some great open-source work going on in visualization, and although coding can seem daunting in the beginning, many times you can use existing code with your own data just by changing variables. The challenge is reading the code and figuring out how everything works.

Similarly, lines 264–293 create buttons to trigger the male/female filter. We can get rid of that, too.

You're close to fully customizing the voyager to the spending data. Go back to the `filter()` function at line 213. Again, update the function so that you can filter by the spending category instead of occupation.

Here's line 222 as-is:

```
var s:String = String(d.data["occupation"]).toLowerCase();
```

Change occupation to category:

```
var s:String = String(d.data["category"]).toLowerCase();
```

Next up on the customization checklist is color. If you compiled the code now and ran it, you would get a reddish stacked area graph, as shown in Figure 5-33. You want more contrast though.

Color is specified in two places. First lines 86–89 specify stroke color and color everything red:

```
shape: Shapes.POLYGON,
lineColor: 0,
fillValue: 1,
fillSaturation: 0.5
```

Then line 105 updates saturation (the level of red), by count. The code for the `SaturationEncoder()` is in lines 360–383. We're not going to use saturation; instead, explicitly specify the color scheme.

First, update lines 86–89 to this:

```
shape: Shapes.POLYGON,
lineColor: 0xFFFFFFFF
```

Now make stroke color white with `lineColor`. If there were more spending categories, you probably wouldn't do this because it'd be cluttered. You don't have that many though, so it'll make reading a little easier.

Next, make an array of the colors you want to use ordered by levels. Put it toward the top around line 50:

```
private var _reds:Array = [0xFFFEF0D9, 0xFFFDD49E, 0xFFFDBB84, 0xFFFC8D59,
0xFFE34A33, 0xFFB30000];
```

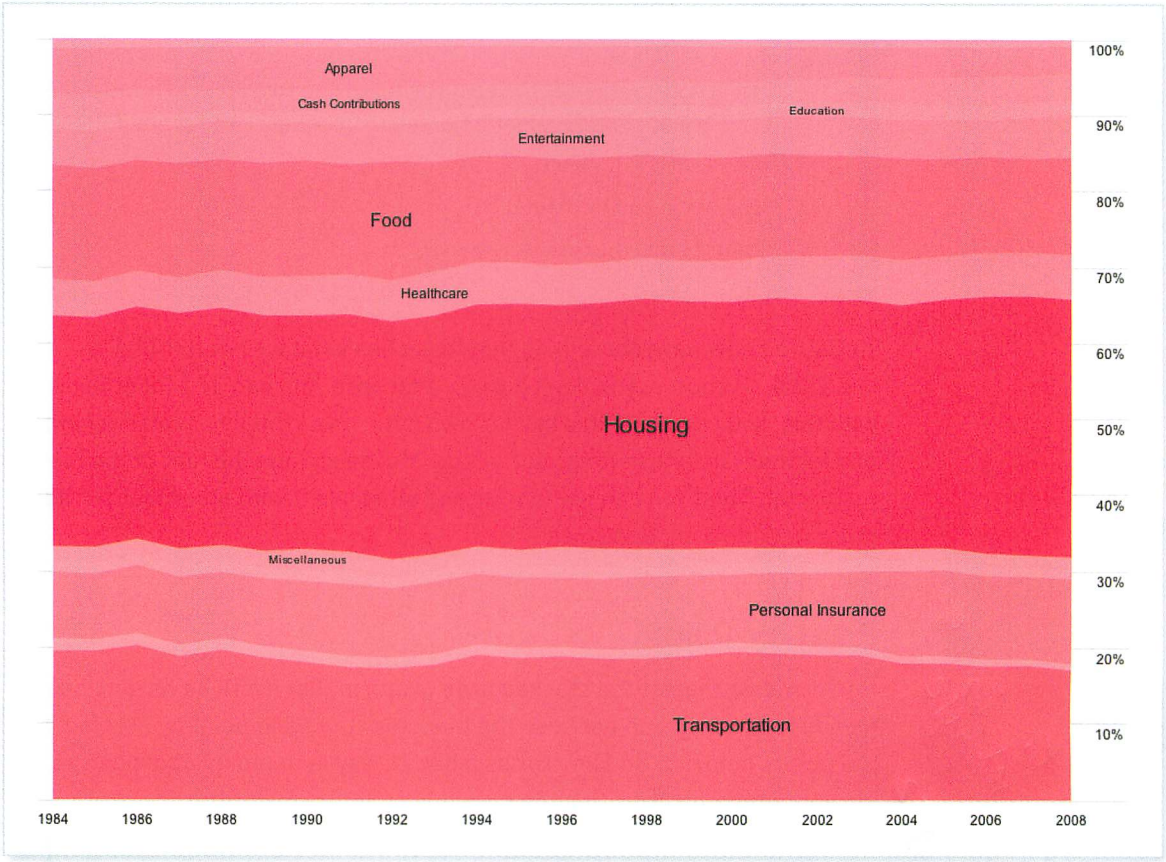


FIGURE 5-33 Stacked area graph with basic coloring

I used the ColorBrewer (referenced earlier) for these colors, which suggests color schemes based on criteria that you set. It's intended to choose colors for maps but works great for general visualization, too.

Now add a new ColorEncoder around line 110:

```
var colorPalette:ColorPalette = new ColorPalette(_reds);
vis.operators.add(new ColorEncoder("data.max", "nodes",
    "fillColor", null, colorPalette));
```

NOTE

If you get an error when you try to compile your code, check the top of JobVoyager.as to see if the following two lines to import the ColorPallete and Encoder objects are specified. Add them if they are not there already.

```
import "are.util.palette.*;
import "are.vis.operator.encoder.*;
```

Ta Da! You now have something that looks like what we're after (Figure 5-32). Of course, you don't have to stop here. You can do a lot of things with this. You can apply this to your own data, use a different color scheme, and further customize to fit your needs. Maybe change the font or the tooltip format. Then you can get fancier and integrate it with other tools or add more ActionScript, and so on.

Point-by-Point

One disadvantage of the stacked area graph is that it can be hard to see trends for each group because the placement of each point is affected by the points below it. So sometimes a better way is to plot proportions as a straight up time series like the previous chapter covered.

Luckily, it's easy to switch between the two in Illustrator. The data entry is the same, so you just need to change the graph type. Select the line plot instead of the stacked area in the beginning, and you get this, the default graph in Figure 5-34.

Clean up and format to your liking in the same way you did with the time series examples, and you have the same data from a different point of view (Figure 5-35).

It's easier to see the individual trends in each age group with this time series plot. On the other hand, you do lose the sense of a whole and distributions. The graph you choose should reflect the point you' want to get across or what you want to find in your data. You can even show both views if you have the space.

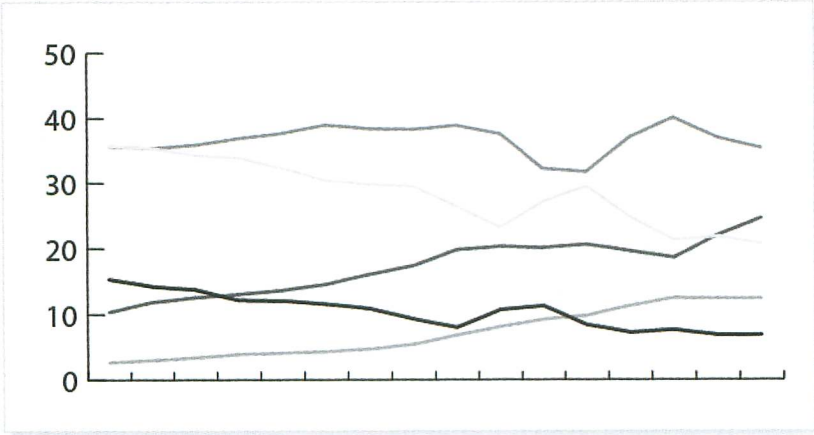


FIGURE 5-34 Default line plot

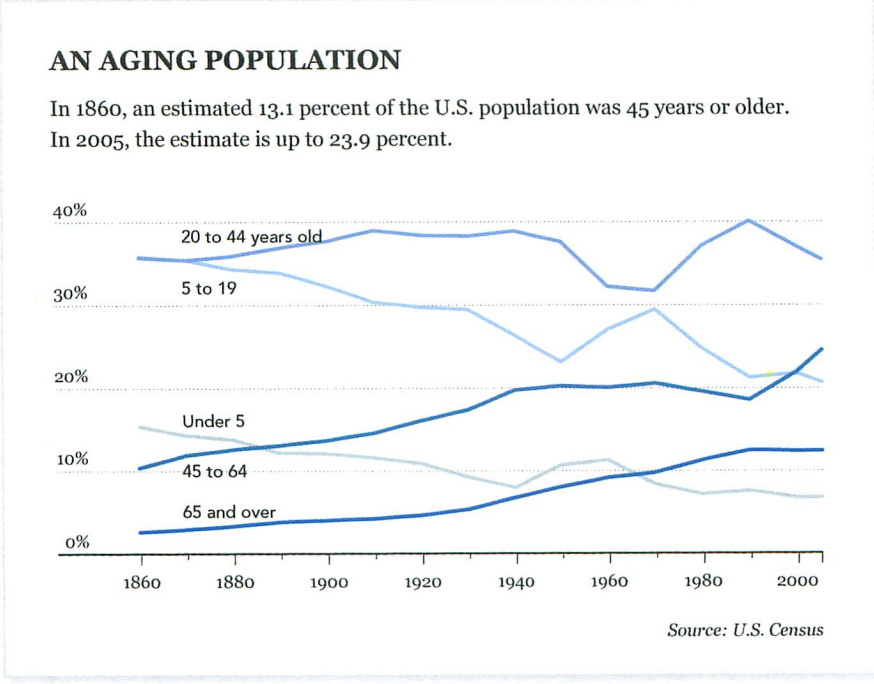


FIGURE 5-35 Labeled line plot cleaned up

Wrapping Up

The main thing that sets proportions apart from other data types is that they represent parts of a whole. Each individual value means something, but so do the sum of all the parts or just a subset of the parts. The visualization you design should represent these ideas.

Only have a few values? The pie chart might be your best bet. Use donut charts with care. If you have several values and several categories, consider the stacked bar chart instead of multiple pie charts. If you're looking for patterns over time, look to your friend the stacked area chart or go for the classic time series. With these steady foundations, your proportions will be good to go.

When it comes time to design and implement, ask yourself what you want to know about your data, and then go from there. Does a static graphic tell your story completely? A lot of the time the answer will be yes, and that's fine. If, however, you decide you need to go with an interactive graphic, map out on paper what should happen when you click objects and what shouldn't. It gets complicated quickly if you add too much functionality, so do your best to keep it simple. Have other people try interacting with your designs to see if they understand what's going on.

Finally, while you're programming—especially if you're new to code—you're undoubtedly going to reach a point where you're not sure what to do next. This happens to me all the time. When you get stuck, there's no better place than the web to find your solution. Look at documentation if it's available or study examples that are similar to what you're trying to do. Don't just look at the syntax. Learn the logic because that's what's going to help you the most. Luckily there are libraries such as Protovis and Flare that have many examples and great documentation.

In the next chapter, we move towards deeper analysis and data interpretation and come back to your good statistical friend. You put R to good use as you study relationships between data sets and variables. Ready? Let's go.