5
Displaying Results Using D3

Data visualization involves displaying information graphically (visually) to present a point or perspective on specific data. Beyond the simple graphs and charts in Excel, sourcing from aggregated transaction rows within a spreadsheet, today's businesses expect far more.

In this chapter, we will explore the process of visualizing data using a web browser and **Data Driven Documents** (**D3**) to present results from your big data analysis projects.

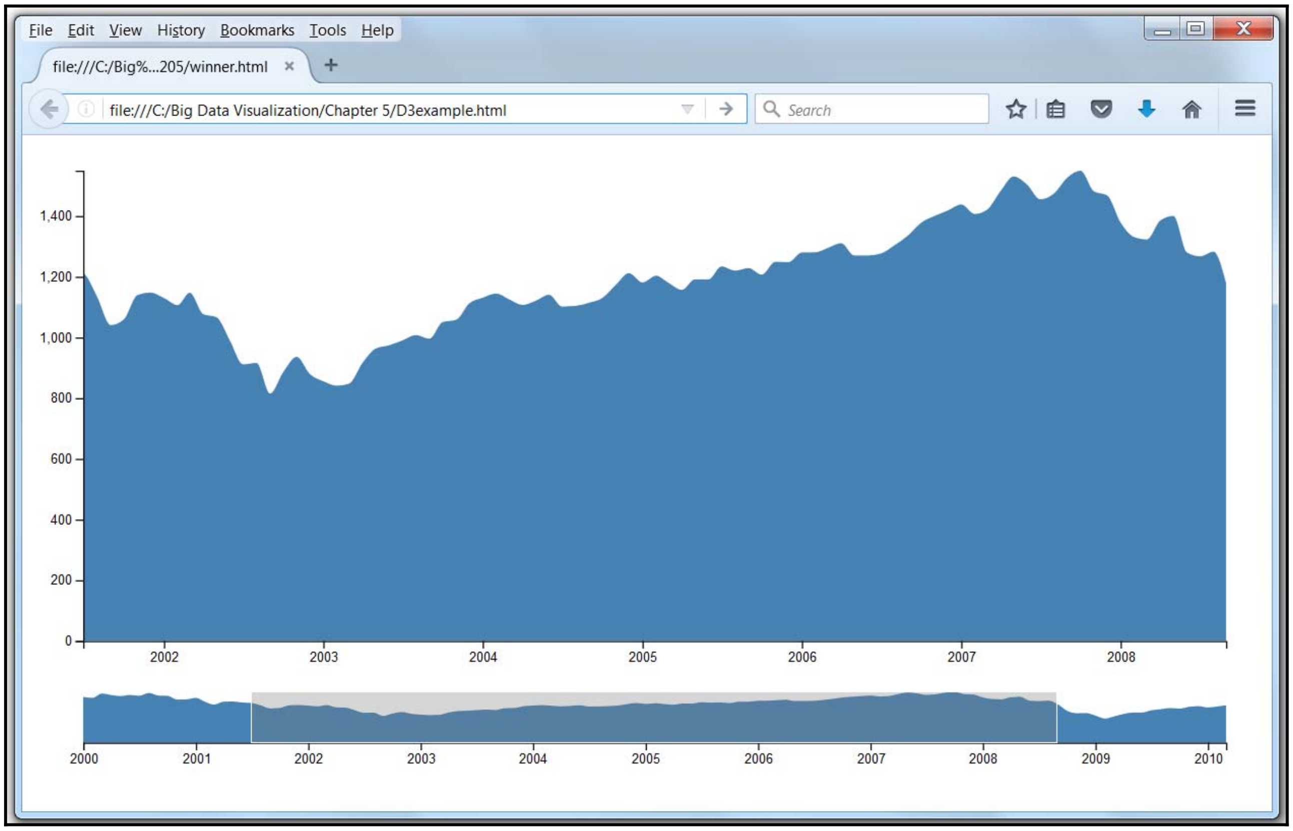
This chapter is organized into the following main sections:

- About D3
- D3 and big data
- Some basic examples
- More examples

About D3

D3 (or `D3.js`) is actually an open source JavaScript library (based upon its predecessor, the Protovis framework), designed with the intention of visualizing data using todays web standards.

D3 helps put life into your data utilizing **Scalable Vector Graphics** (**SVG**), Canvas, and standard HTML.


File Edit View History Bookmarks Tools Help
file:///C:/Big%...205/winner.html
file:///C:/Big Data Visualization/Chapter 5/D3example.html
Search
1,400
1,200
1,000
800
600
400
200
0
2002
2003
2004
2005
2006
2007
2008
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010

D3 combines powerful visualization and interaction techniques with a data-driven approach to DOM manipulation, giving you the full capabilities of modern browsers and the freedom to design the right visual interface for your data.

In contrast to many other libraries, `D3.js` allows inordinate control over the visualization of your data. Its development was noted in 2011, as version 2.0.0 was released in August 2011.

D3 is embedded within an HTML webpage, and uses prebuilt JavaScript functions to select elements, create SVG objects, style them, or add transitions, dynamic effects, and so on.

Detailed information, including the `D3.js` libraries can be accessed at `https://D3js.org`.

D3 and big data

First, let me say that you can easily bind or use your large datasets to common SVG objects using the functions available in the `D3.js` libraries.

The data can even be in a variety of formats, most commonly JSON, **comma-separated values** (**CSV**), or **geoJSON**, but, if required, JavaScript functions can be written to read other data format.

However, large isn't big in the sense of big data. Realistically, binding a CSV file of 500 records cannot be likened to binding it to a file of 500,000 records.

So, can D3 really help in the context of big data?

Since it is low-level, D3 may seem like a bad fit for big data visualization projects. The `D3.js` libraries just won't work with gigabytes of data, but once you perform some preprocessing on the data, D3 can help make sense of the results.

In fact, in each of the previous chapters of this book, `Chapter 2`, *Access, Speed, and Storage with Hadoop*, (where we loaded data into a Hadoop environment and then used Hive to manipulate that data into workable summaries), `Chapter 3`, *Understanding Your Data Using R*, (where we used the power of R scripting to profile the data into meaningful summaries for visualizations), and, `Chapter 4`, *Addressing Big Data Quality* (where we utilized DataManager to address the quality of the data so it could be visualized) we used the following strategy:

- Assembling the data
- Profiling the data
- Addressing quality concerns
- Processing for visualization, that is, summarize or aggregate, and so on
- Visualizing!

So, let us now move on–and look at some pretty interesting examples of visualizing big data using the power offered by opened sourced, D3.

Some basic examples

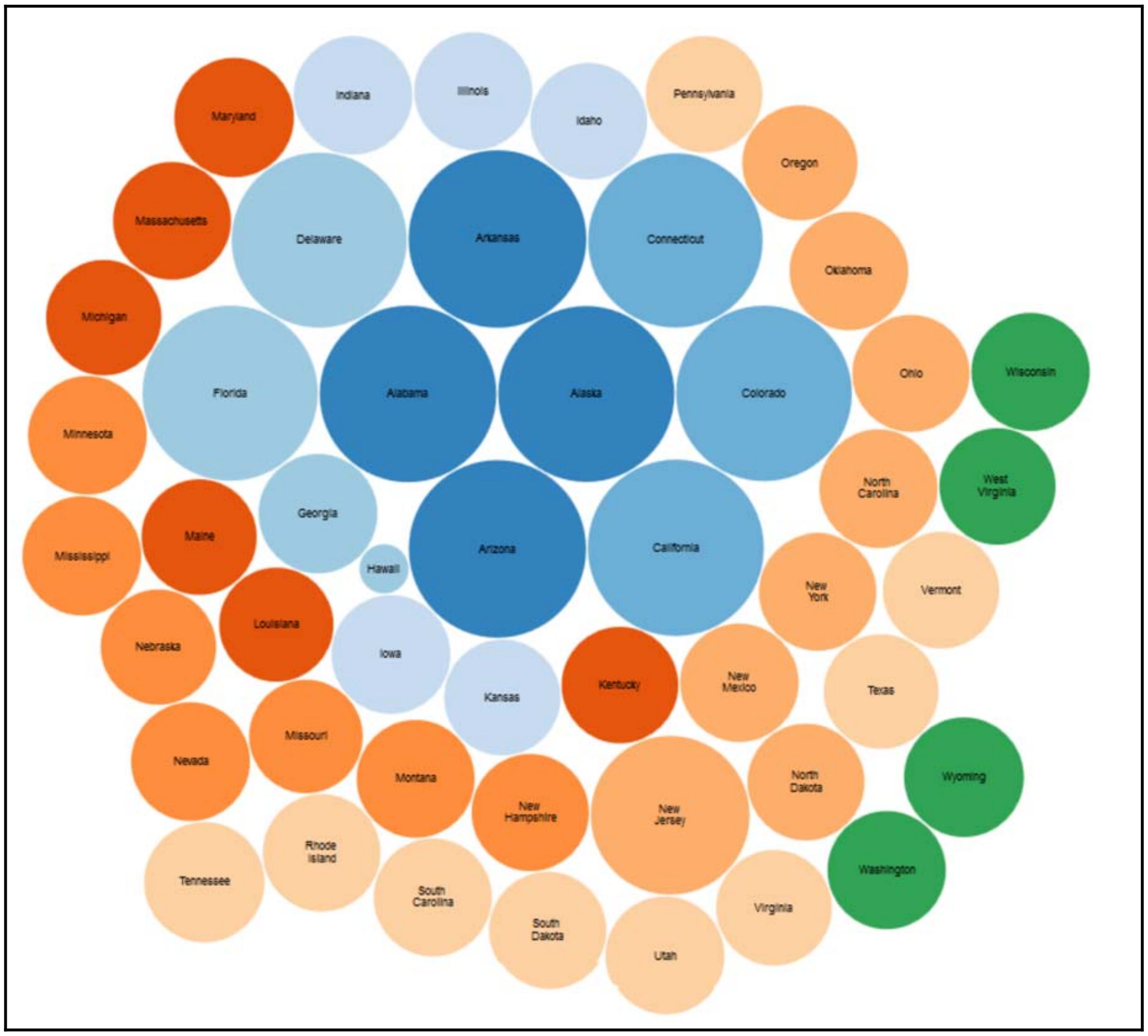
Just to get us underway, let's take another look at an example we used in this book's `Chapter 3`, *Context – Understanding Your Data Using R*.

In that scenario, we had used some simple R scripting to summarize patient survey information into a result or summarized a file showing a total for the number of visits for each state in the United States. Now, using that same data file, perhaps we can get a general idea of what D3 may be able to do for us in the form of valuable data visualization.

We'll get into the details in the next section, but for now, all we need to do to get started is to create an HTML page based upon a bubble chart template downloaded from the D3 website (this particular template utilizes the D3 flare class libraries to create a bubble chart from our data file).

A **bubble chart** is an interesting way to display data in an efficient, reasonable way since, in this example, we want to show all 50 states from our data, it can be represented in a clear fashion.

The following figure shows the resulting bubble visualization generated by the D3 library and bubble template (viewed with a web browser):



Getting started with D3

As this book does not intend to be a lecture on the interworking's of D3 (rather focusing on the use of D3 as an option to visualize big data), we will simply point out here a bit of basic information around the reader's efforts into getting started with using D3:

- The website may be found at `https://D3js.org`
- The latest version (at the time of writing) is V4.2.8, and that version can be downloaded from the following link:

```
https://github.com/D3/D3/releases/download/v4.2.8/D3.zip
```

- You can simply link directly to the latest release libraries by inserting the following line in your projects:

  ```
  <script src="https://D3js.org/D3.v4.min.js"></script>
  ```

Based on your environment, or at least while you are experimenting with the D3 examples, I recommend that you download the actual source files so that you can use a local reference (shown as follows), as it eliminates the possibility of encountering any problems accessing the D3 libraries (and, if you are inclined, you can see what the actual code is doing):

```
<script src="D3.v4.min.js"></script>
```

You can download the files from: `https://github.com/D3/D3`.

Finally, there are great instructional tutorials offered at: `https://github.com/D3/D3/wiki/Tutorials`.

Now, on to the examples!

When we talk of visualizing big data, what we actually mean (and expect) is visualizing the aggregated results of an analysis (no, contrary to popular opinion, no one visualizes raw big data directly). To this point, the following is a key concept to understand and embrace that the capture and storing, manipulation (profiling, addressing quality, and aggregating), and visualizing are all separate (and encapsulated) components and can (and most likely should) each be addressed with different tools or technologies.

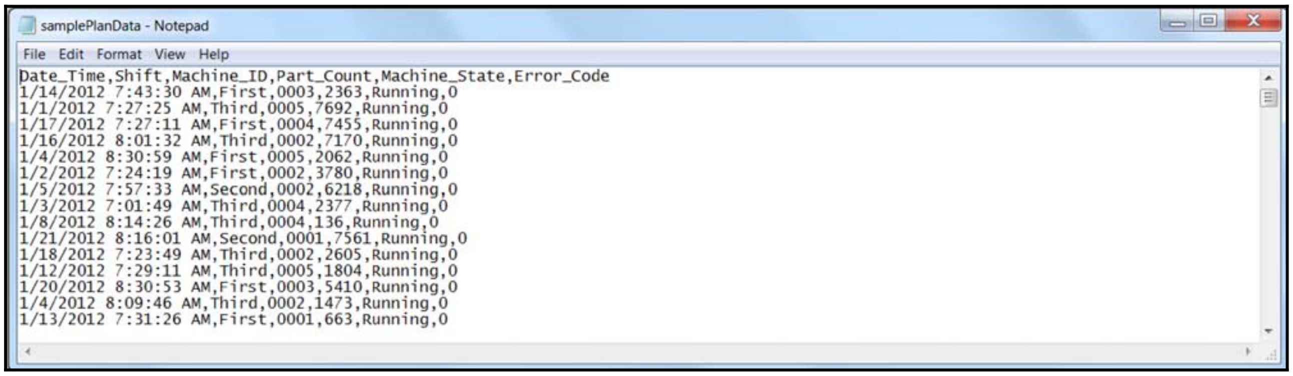
This way, each component leverages the tool that is just right for the particular purpose.

In the following example, we are dealing with manufacturing data. Specifically, we have data captured from a number of machines at a particular plant utilizing a data logging system. The **data logger** is a program that gathers production-line data and writes it to a log file. In this scenario, the data logger is installed on each manufacturing machine. The data logger collects production data directly from the machine, stores it in memory, and periodically sends it off to the data repository (a log file). In the event of a network outage, the onboard data logger can continue to collect production information from the machine while the network is down and then back-fill the data to the log file when the network comes back up. This results in millions of machine transaction status records accumulating daily, a big data scenario for sure.

The transaction records contain the following information:

Date/Time	**This is the exact date and time of day that the transaction log record was captured.**
Shift ID	The plant runs three shifts, so this would be 1, 2, or 3.
Machine ID	The plant has five machines in operation; 001, 002, 003, 004, and 005.
Part count	Numeric total of products produced by the machine since the previous polling cycle.
Machine state	The machine state is the current condition of the machine. Typical states are running, idle, unplanned down, and planned down, changeover/setup, and offline.
Error code	These error codes can be automatically generated by the controller on the machine or can correspond to a list of downtime reasons that are manually selected by the machine operator.

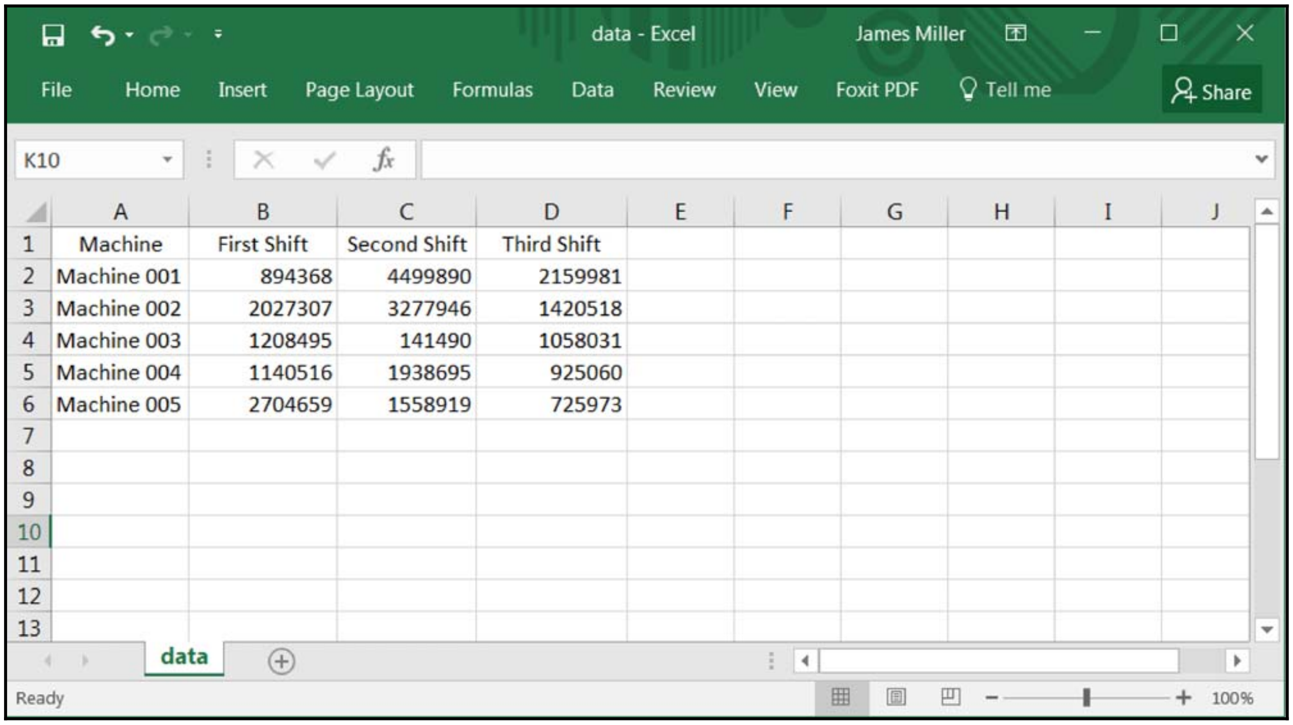
The following screenshot shows a sample of the raw data records:



```
Date_Time,Shift,Machine_ID,Part_Count,Machine_State,Error_Code
1/14/2012 7:43:30 AM,First,0003,2363,Running,0
1/1/2012 7:27:25 AM,Third,0005,7692,Running,0
1/17/2012 7:27:11 AM,First,0004,7455,Running,0
1/16/2012 8:01:32 AM,Third,0002,7170,Running,0
1/4/2012 8:30:59 AM,First,0005,2062,Running,0
1/2/2012 7:24:19 AM,First,0002,3780,Running,0
1/5/2012 7:57:33 AM,Second,0002,6218,Running,0
1/3/2012 7:01:49 AM,Third,0004,2377,Running,0
1/8/2012 8:14:26 AM,Third,0004,136,Running,0
1/21/2012 8:16:01 AM,Second,0001,7561,Running,0
1/18/2012 7:23:49 AM,Third,0002,2605,Running,0
1/12/2012 7:29:11 AM,Third,0005,1804,Running,0
1/20/2012 8:30:53 AM,First,0003,5410,Running,0
1/4/2012 8:09:46 AM,Third,0002,1473,Running,0
1/13/2012 7:31:26 AM,First,0001,663,Running,0
```

In this example, we'll skip over the process of manipulating the raw data, assuming we have performed some profiling of the data, addressed any quality issues, and formed an aggregation of the data that we wish to base our visualization upon (or drive the data from).

What we've come up with is a simple comma-delimited text file (we named it `data.csv`) with the total number of products produced for each machine ID, broken out by shift:



Machine	First Shift	Second Shift	Third Shift
Machine 001	894368	4499890	2159981
Machine 002	2027307	3277946	1420518
Machine 003	1208495	141490	1058031
Machine 004	1140516	1938695	925060
Machine 005	2704659	1558919	725973

Again, one of the already available D3 visualization sample templates can be used to create a grouped bar chart. Assuming you have already downloaded the D3 libraries, the steps to adopt the template and create our visualization are as follows:

1. Download the grouped bar HTML template document.
2. Open the document in any HTML editor (or any programmer's editor).
3. Under the document's `<body>` tag, enter or modify the following code:

```
<body>
<! --- added a simple heading -->
<h1><center>Total Parts by Shift</cellspacing></h>
<! --- local include for D3 libraries -->
<script src="D3.v3.min.js"></script>
```

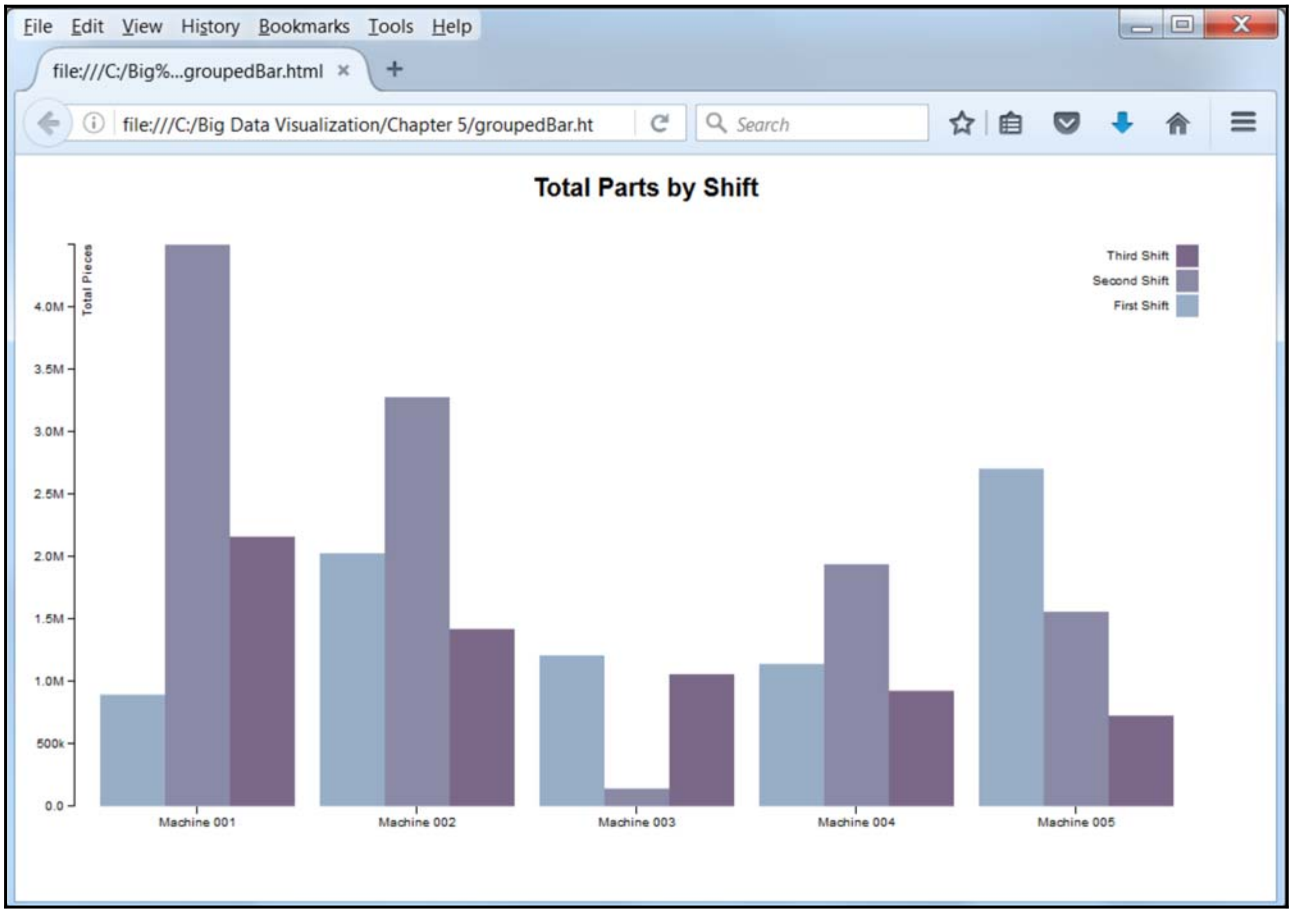
4. What this modification does is to add text (`Total Parts by Shift`) to be used as our visualization's heading and changes the reference to the D3 libraries (the `src= D3.v3.min.js`) to be a local reference.

5. Next, again assuming our data file is in the same local location as our HTML file, we can find the document's file reference (`D3.csv`) and verify the filename:

```
<!--- here is the data -->
D3.csv("data.csv", function(error, data) {
```

Once we've saved the updated HTML document, we can view it using any web browser.

Voila! We've created our first big data visualization using D3:

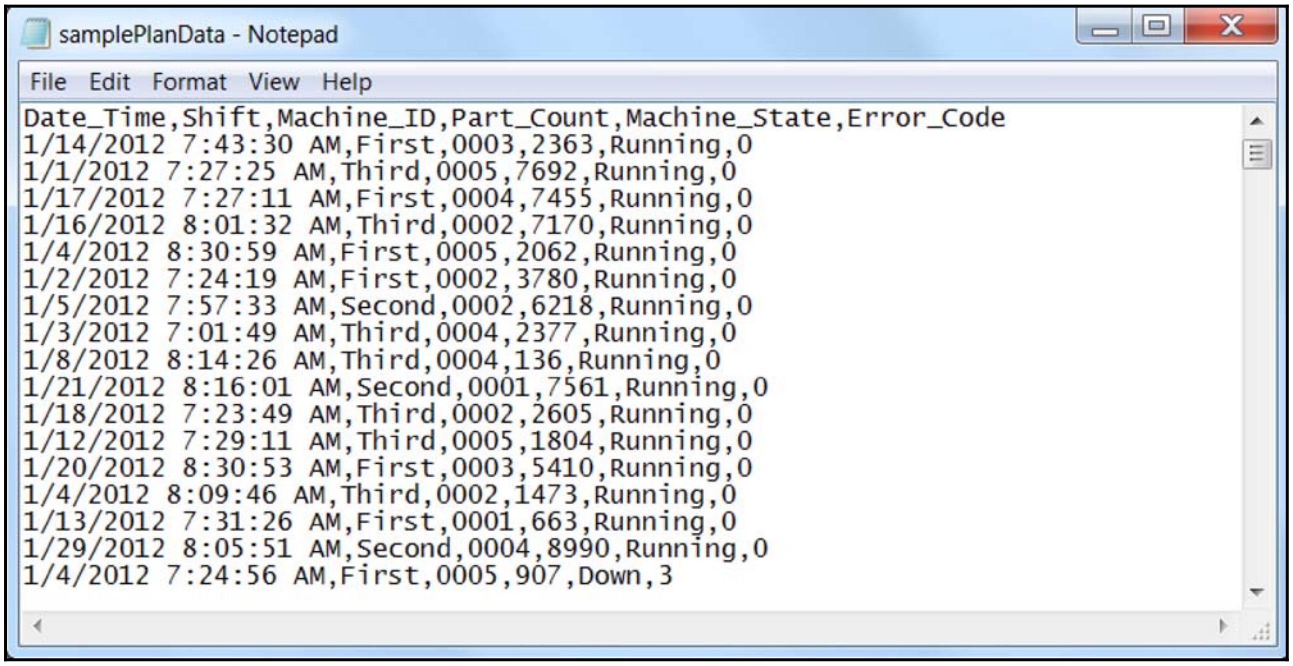


If you take a few minutes and review the extensive visual gallery available (at least at the time of writing) at `https://github.com/D3/D3/wiki/Gallery`, you will be able to see literally hundreds of D3 template samples showing virtually any kind of visualization you may want, all available to adopt (download, modify, and use).

Let's look at some more examples, digging deeper into the specifics of D3 use.

Another area of interest at our manufacturing plant is shift performance. Management wants to determine how the different shifts contribute to overall profitability or perhaps how each individual shift compares to the others. To do this, there are several **key performance indicators** (**KPIs**) to be scrutinized. One such indicator is total parts delivered by shift. Again, the process will be to take our big data source of raw plan records and aggregate them into a usable form.

The following screenshot shows a portion of our raw data:



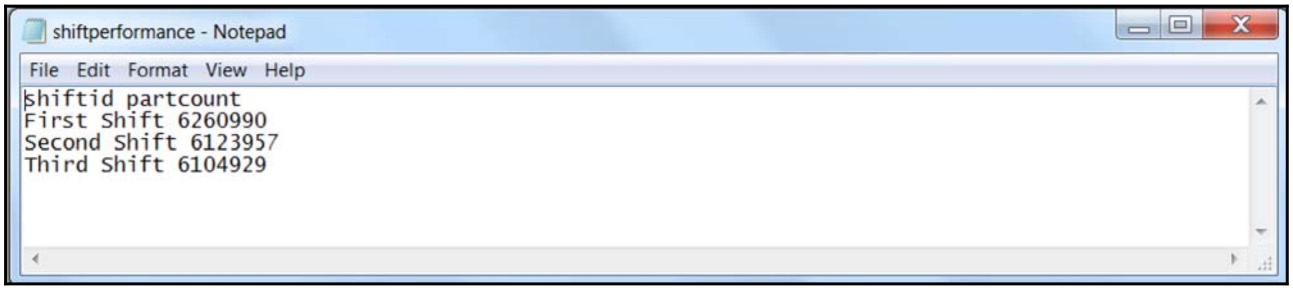
```
Date_Time,Shift,Machine_ID,Part_Count,Machine_State,Error_Code
1/14/2012 7:43:30 AM,First,0003,2363,Running,0
1/1/2012 7:27:25 AM,Third,0005,7692,Running,0
1/17/2012 7:27:11 AM,First,0004,7455,Running,0
1/16/2012 8:01:32 AM,Third,0002,7170,Running,0
1/4/2012 8:30:59 AM,First,0005,2062,Running,0
1/2/2012 7:24:19 AM,First,0002,3780,Running,0
1/5/2012 7:57:33 AM,Second,0002,6218,Running,0
1/3/2012 7:01:49 AM,Third,0004,2377,Running,0
1/8/2012 8:14:26 AM,Third,0004,136,Running,0
1/21/2012 8:16:01 AM,Second,0001,7561,Running,0
1/18/2012 7:23:49 AM,Third,0002,2605,Running,0
1/12/2012 7:29:11 AM,Third,0005,1804,Running,0
1/20/2012 8:30:53 AM,First,0003,5410,Running,0
1/4/2012 8:09:46 AM,Third,0002,1473,Running,0
1/13/2012 7:31:26 AM,First,0001,663,Running,0
1/29/2012 8:05:51 AM,Second,0004,8990,Running,0
1/4/2012 7:24:56 AM,First,0005,907,Down,3
```

Since we've already worked with R in this book, we will use it again to manipulate our raw plant data. The following is the simple R scripting used to aggregate the part counts by shift ID to a summary file:

```
# --- reads the raw file into a R table named "parts"
parts<-read.table(file="C:/Big Data Visualization/Chapter
5/samplePlanData.txt",sep=",")
data.df <- data.frame(parts)

# --- create a subset of the raw data for each shift id
FirstShift<-subset(data.df,data.df[,2]=="First")
SecondShift<-subset(data.df,data.df[,2]=="Second")
ThirdShift<-subset(data.df,data.df[,2]=="Third")

# --- aggregate part totals by shift ID
sum(as.numeric(FirstShift[,4]))
sum(as.numeric(SecondShift[,4]))
sum(as.numeric(ThirdShift[,4]))

# --- create a summary file for visualization
sink("C:/Big Data Visualization/Chapter 5/data.tsv")
cat("shiftid")
cat("\t")
cat("partcount")
cat("\n")
cat(paste("First Shift", "\t", sum(as.numeric(FirstShift[,4]))),sep = "\t",
collapse = NULL)
cat("\n")
cat(paste("Second Shift", "\t", sum(as.numeric(SecondShift[,4]))),sep =
"\t", collapse = NULL)
cat("\n")
cat(paste("Third Shift", "\t", sum(as.numeric(ThirdShift[,4]))),sep = "\t",
collapse = NULL)
cat("\n")
sink()
```

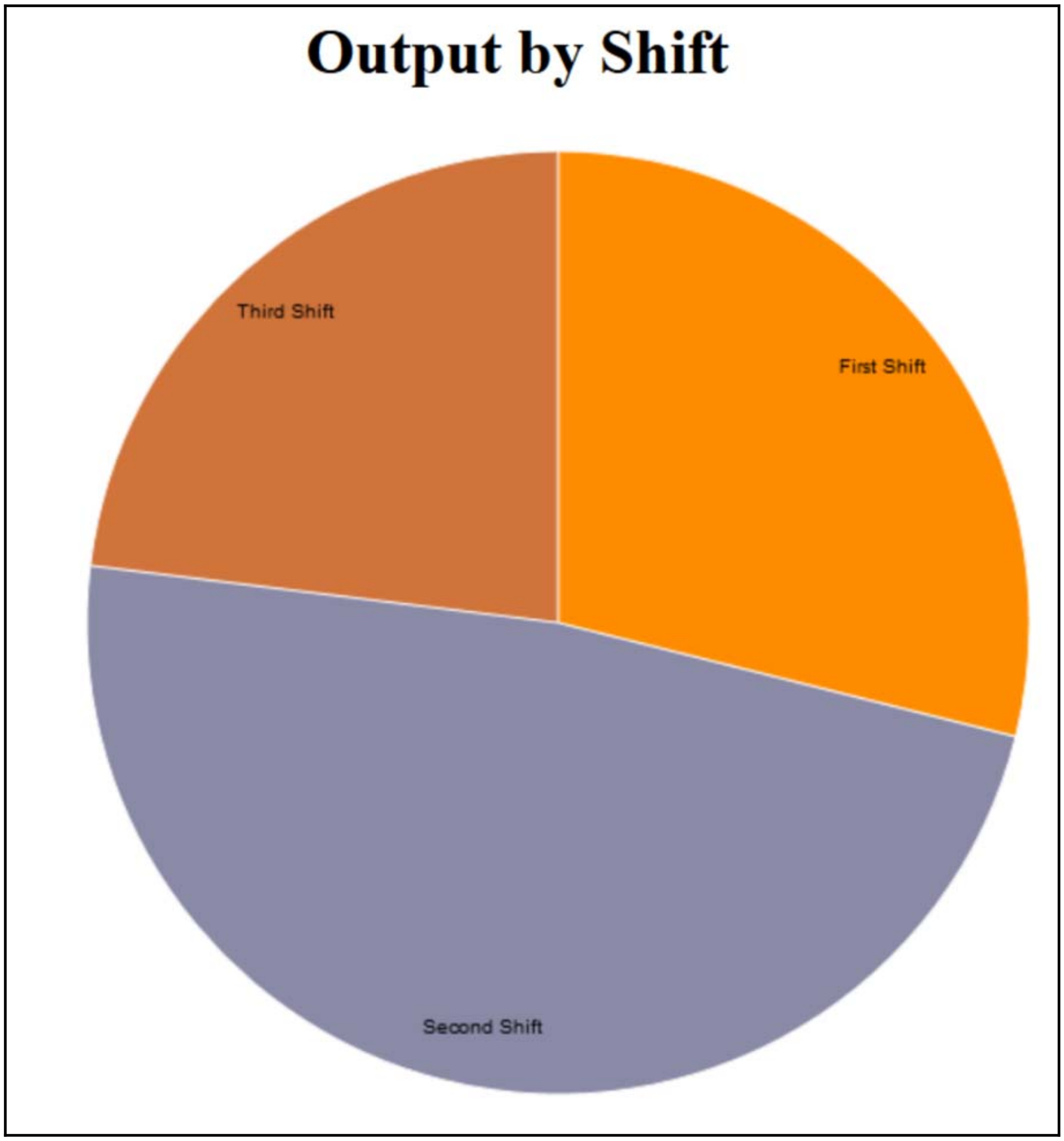
Reminder: as we mentioned in `Chapter 3`, *Understanding Your Data Using R,* there are many approaches to using R and the previous script is just one approach. The reader is invited to use it or improve it.

The following screenshot shows the aggregated or summary file (named `data.tsv`) that the R script generates:



```
shiftid partcount
First Shift 6260990
Second Shift 6123957
Third Shift 6104929
```

This summary file is then the source of our D3 data visualization (shown in the following figure):



This visualization is generated using the minimalist pie chart D3-shape sample template.

To display the (preceding) visualization, the following HTML document was used (our changes are highlighted):

```
<!DOCTYPE html>
<meta charset="utf-8">
<!-add the heading -->
<h1><center>Output by Shift</center></h1>
<canvas width="960" height="500"></canvas>
<!-local reference to D3 libraries --->
<script src="D3.v4.0.0-alpha.4.min.js"></script>
<script>
var canvas = document.querySelector("canvas"),
    context = canvas.getContext("2d");

var width = canvas.width,
    height = canvas.height,
    radius = Math.min(width, height) / 2;

var colors = ["#ff8c00", "#8a89a6", "#d0743c", "#6b486b", "#a05d56",
"#d0743c", "#ff8c00"];

var arc = D3.arc()
    .outerRadius(radius - 10)
    .innerRadius(0)
    .context(context);

var labelArc = D3.arc()
    .outerRadius(radius - 40)
    .innerRadius(radius - 40)
    .context(context);

var pie = D3.pie()
    .sort(null)
    .value(function(d) { return d.partcount; });

context.translate(width / 2, height / 2);

<-- read our summary file-->
D3.requestTsv("data.tsv", function(d)
{
  d.partcount = +d.partcount;
  return d;
}, function(error, data)
{
  if (error) throw error;
  var arcs = pie(data);
  arcs.forEach(function(d, i)
 {
        context.beginPath();
        arc(d);
        context.fillStyle = colors[i];
        context.fill();
      });
      context.beginPath();
      arcs.forEach(arc);
      context.strokeStyle = "#fff";
      context.stroke();
      context.textAlign = "center";
      context.textBaseline = "middle";
      context.fillStyle = "#000";
      arcs.forEach(function(d)
      {
      var c = labelArc.centroid(d);
      context.fillText(d.data.shiftid, c[0], c[1]);
      });
    });
    </script>
```

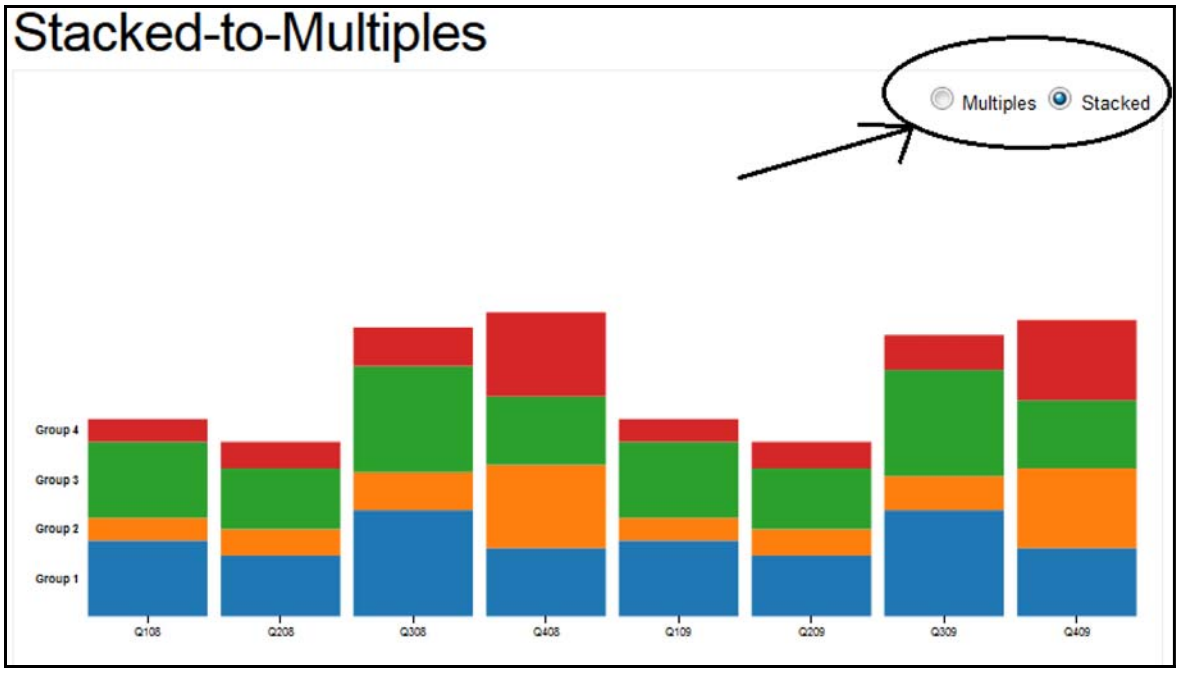
A little down time

Another prospective opportunity for visualization of our raw plant data is around down time. In our raw data file, there is a field named `Machine_State`. The machine state is the current condition of the machine, typically running, idle, unplanned down, planned down, changeover/setup, and offline.

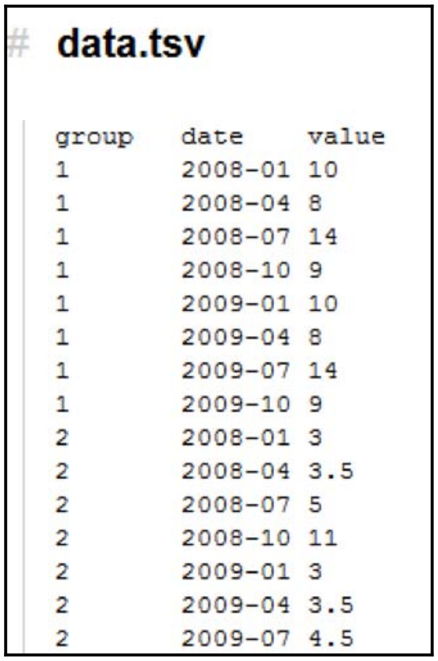
In this scenario, we want to have a total count of the number of times each machine (001 through 005) recorded or wrote a transaction record where the machine was not in a running state and we'd like to see these numbers broken down by a quarter.

Searching the D3 site, we find that there is a fine stacked bar sample template that lends itself nicely to our requirements. This visual also demonstrates the use of runtime reconfiguration in that it uses a standard HTML radio button that the user can use to switch or transition the visualization view from a stacked bar to a multiples bar display format.

The template is named **Stacked-to-Multiples** and it can be found at `http://blocks.org/mbostock/4679202` (the following figure points out the HTML radio buttons in the upper right):



Once again, this D3 sample template uses a summarized data file to drive the visualization. The data file (named `data.tsv`) contains records with three fields: **group**, **date**, and **value**, and it is (partially) shown in the following screenshot:

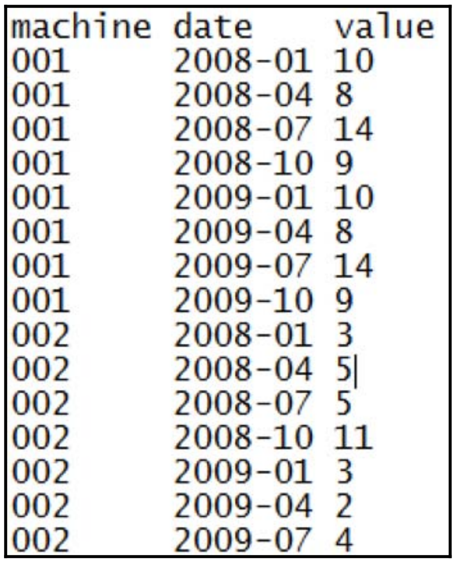


data.tsv

```
group   date    value
1       2008-01 10
1       2008-04 8
1       2008-07 14
1       2008-10 9
1       2009-01 10
1       2009-04 8
1       2009-07 14
1       2009-10 9
2       2008-01 3
2       2008-04 3.5
2       2008-07 5
2       2008-10 11
2       2009-01 3
2       2009-04 3.5
2       2009-07 4.5
```

This is a pretty straightforward example that we can adapt for our purposes without much effort. First, we can pre-process our raw plant data into the preceding file format–but with one slight difference: rather than using a group field, we'll use that first field (or column) of data as our machine ID (the other two fields can remain the same).

I have also named our data file as `datastacked.tsv`, and it is partially shown here:



```
machine date    value
001     2008-01 10
001     2008-04 8
001     2008-07 14
001     2008-10 9
001     2009-01 10
001     2009-04 8
001     2009-07 14
001     2009-10 9
002     2008-01 3
002     2008-04 5|
002     2008-07 5
002     2008-10 11
002     2009-01 3
002     2009-04 2
002     2009-07 4
```

Once again, the pre-processing of the raw plant data could be accomplished using *R* scripts or other tools. Big data sources would typically be (perhaps) chunked or processed in segments to effectively arrive at the desired aggregated or summarized file, ready for visualization. As we saw in `Chapter 3`, *Understanding Your Data Using R*, offers the ability to easily aggregate data and then merge the multiple aggregated files into a single file for visualization.

The next step (in adopting this D3 sample template) is to download the sample template and save it as an HTML document. From there, we can make a few minor modifications:

1. As mentioned earlier in this chapter, I have downloaded and saved the D3 library files, so I need to change the `src=` reference within the HTML document (to make it a local reference). It should look as follows:

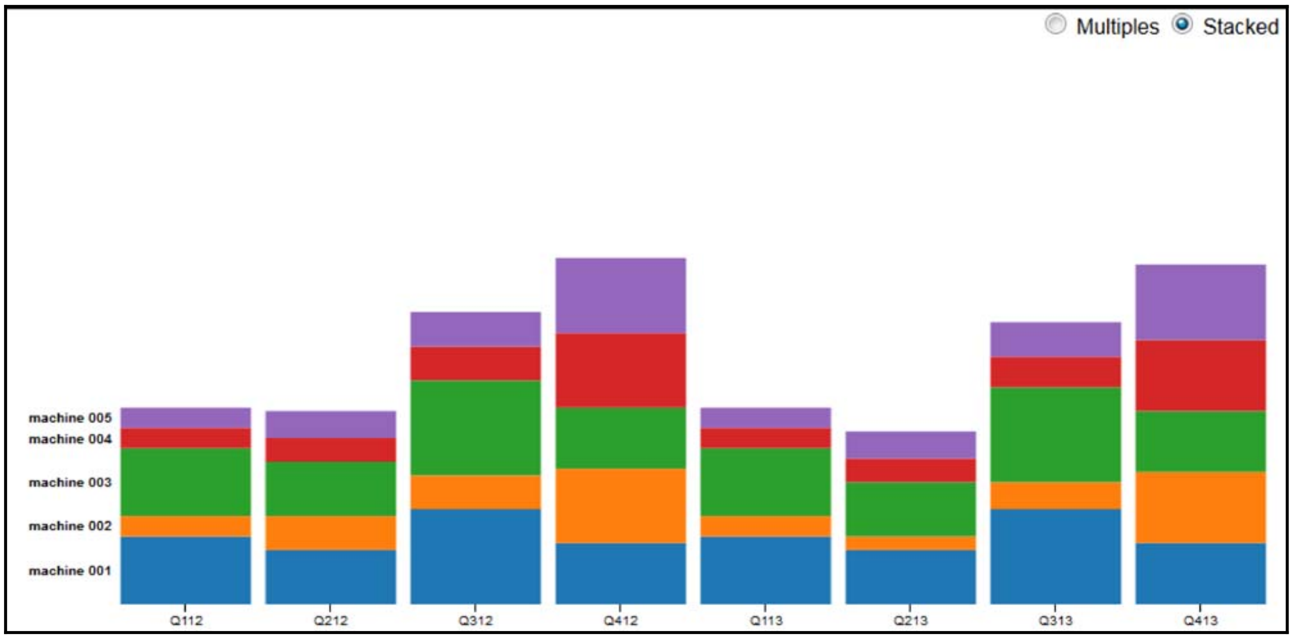
```
<script src="D3.v3.min.js"></script>
```

2. Since I renamed the data file, locate the line in the HTML document referencing the file name and change it to reflect our data file name:

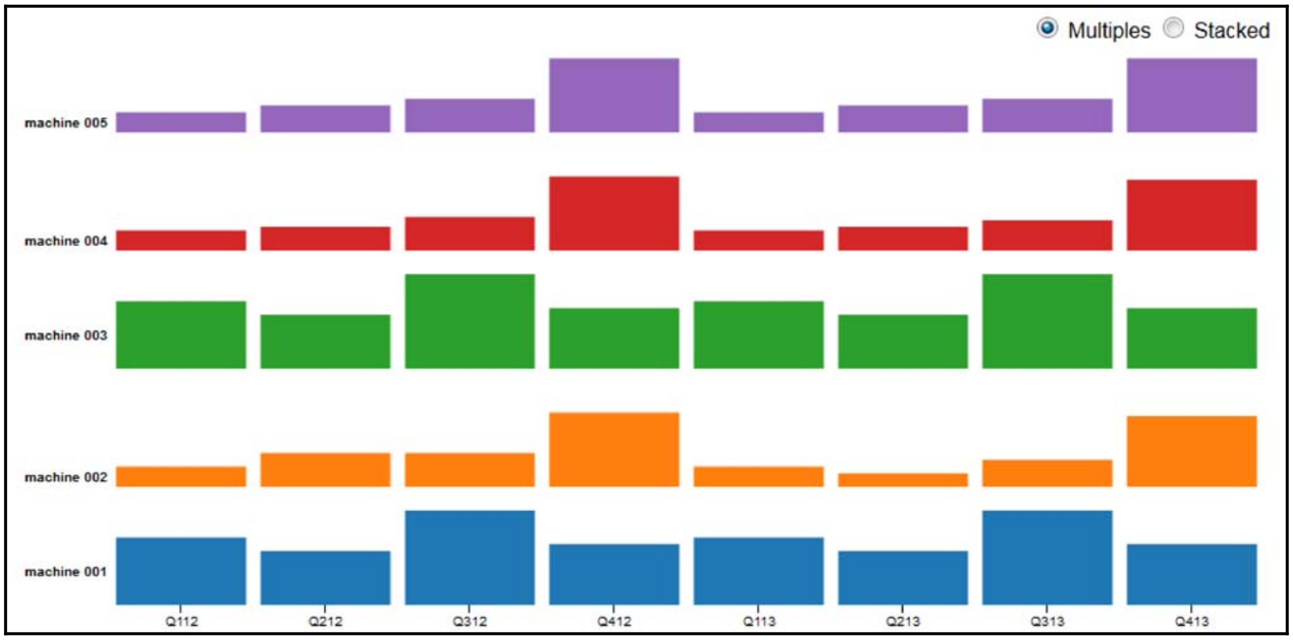
```
D3.tsv("datastacked.tsv", function(error, data) {
```

3. Finally, since we've changed the first field name in our data file (from group to machine), we need to change all references to that field within the HTML document. Hint: a simple global find and replace in a text editor does the trick!
4. Save the updated HTML file and view it in your web browser!

The following figure shows our data visualization showing the stacked view:



If you click on the radio button labeled **Multiples**, the visualization changes format:

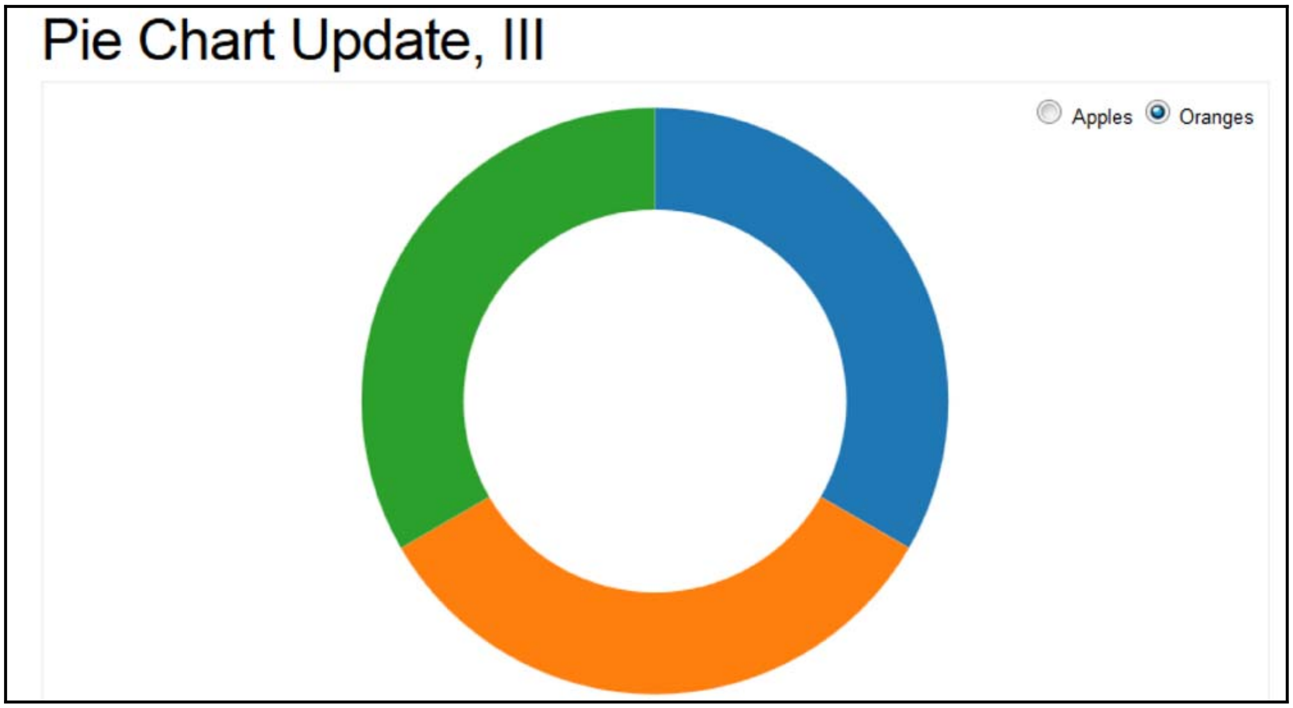


Visual transitions

The procedure of clicking on the HTML radio buttons to change the format of the visualization is known as **transitioning**. In the preceding example, we transitioned from one format to another. We can also use transitioning to change what data the visualization is driven from. Let us take a look at another example to illustrate this concept.

Going back to our manufacturing plant's raw data, let's suppose that we want to look at the output (total part count) by machine ID and by shift. We'd like to build a data visualization that displays each machines part count (its output) broken out by shift. We also want the ability to change the shift and see the visualization update (transition) appropriately.

For this example, I elected to use a D3 sample template that builds a donut pie chart (which you can find at `http://blocks.org/mbostock/5681842`). This template transitions the visualization between **Apples** data and **Oranges** data:



Another thought-provoking feature of this template is that it handles missing data by filling in the null or missing values with zeros (you could use any default value instead of zeros). We will examine this feature shortly.

To adopt this sample template for our purposes, we will go about following the usual steps: download the HTML template, locate and change the D3 library reference, and update the data.

Let's look at the three specific customizations:

1. I added a simple heading:

```
<center><H1>Parts by Shift</H1></center>
```

2. I modified the HTML form, changing it from apples and oranges to indicate our three plant shifts. Note that I had to add a third HTML radio button:

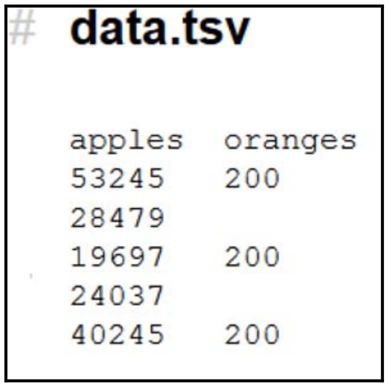
```
<form>
  <label><input type="radio" name="dataset" value="first"
    checked>First Shift
  </label>
  <label><input type="radio" name="dataset" value="second">Second
    Shift
  </label>
  <label><input type="radio" name="dataset" value="third">Third
    Shift
  </label>
</form>
```

3. I modified the function that handles missing data to validate all three shifts:

```
function type(d)
{
  d.first = +d.first || 0;
  d.second = +d.second || 0;
  d.third = +d.third || 0;
  return d;
}
```

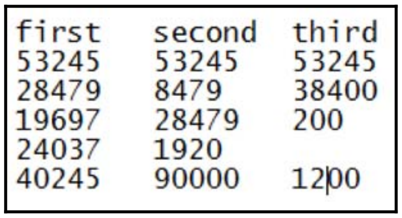
Again, notice that I simply changed the references from `apples` and `oranges` to first and second, and then added a third reference, for our third shift.

The last step is to pre-process our raw plant data into a summary file that this D3 template can use. It is a pretty simple file, with just two fields, **apples** and **oranges** (shown in the following screenshot). You will notice that the second field (**oranges**) is missing values:



```
# data.tsv

apples  oranges
53245   200
28479
19697   200
24037
40245   200
```

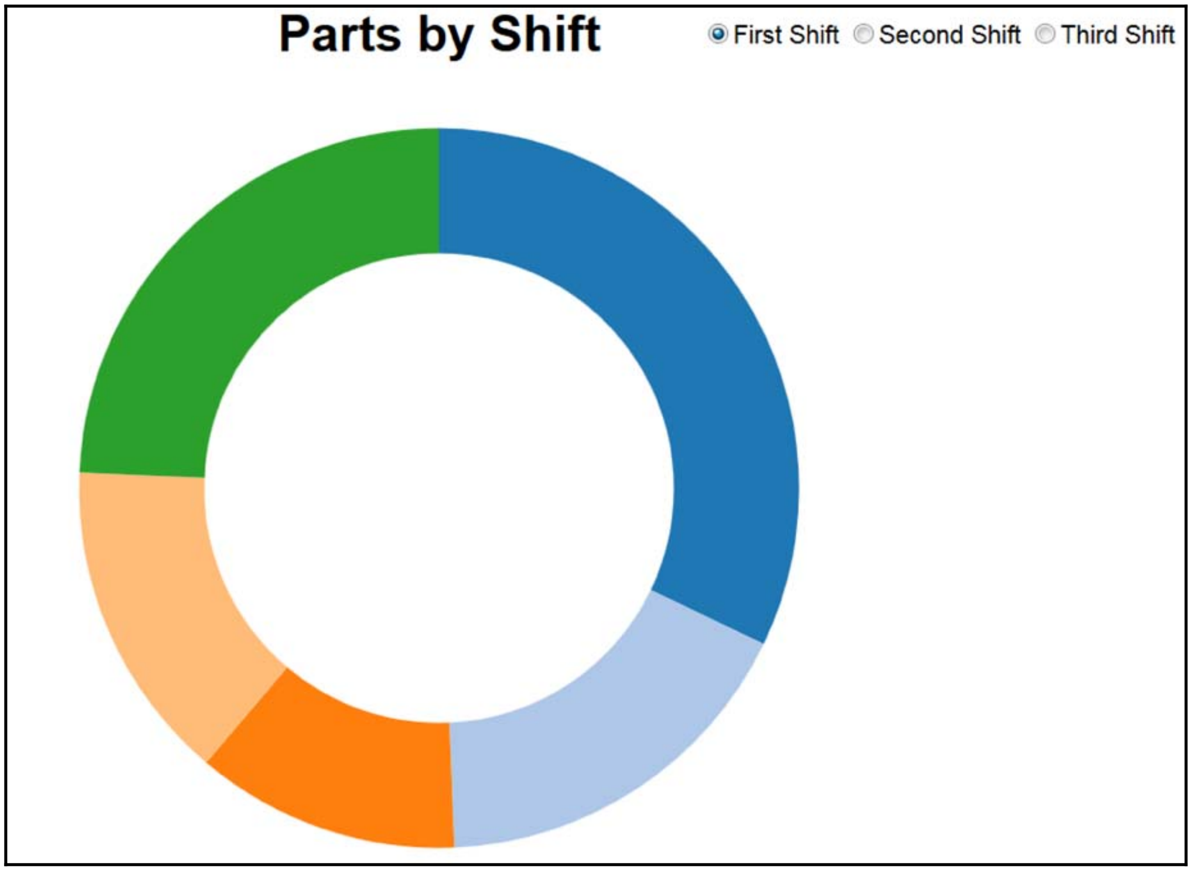
For our data, we will have three fields (one for each shift): **first**, **second**, and **third**. After having summarized our data, we see the following:



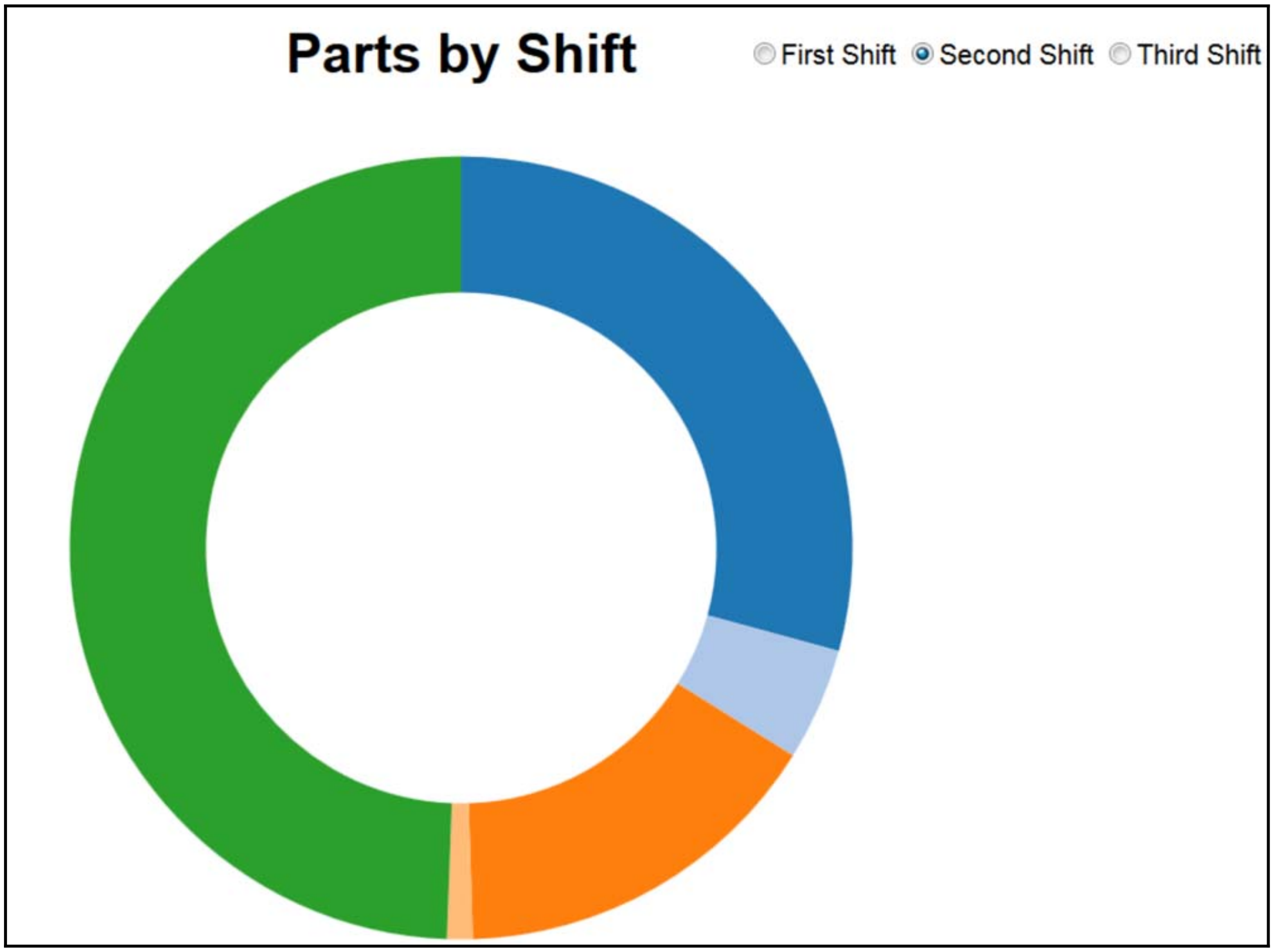
```
first   second  third
53245   53245   53245
28479   8479    38400
19697   28479   200
24037   1920
40245   90000   1200
```

Each record in our summary file indicates a summary record for a particular machine ID with a part count for each of the three shifts. Notice that within our data, machine 004 was offline or down during the third shift so we are missing a value for that time. This will prove to be a good test for the function we modified (to handle missing values).

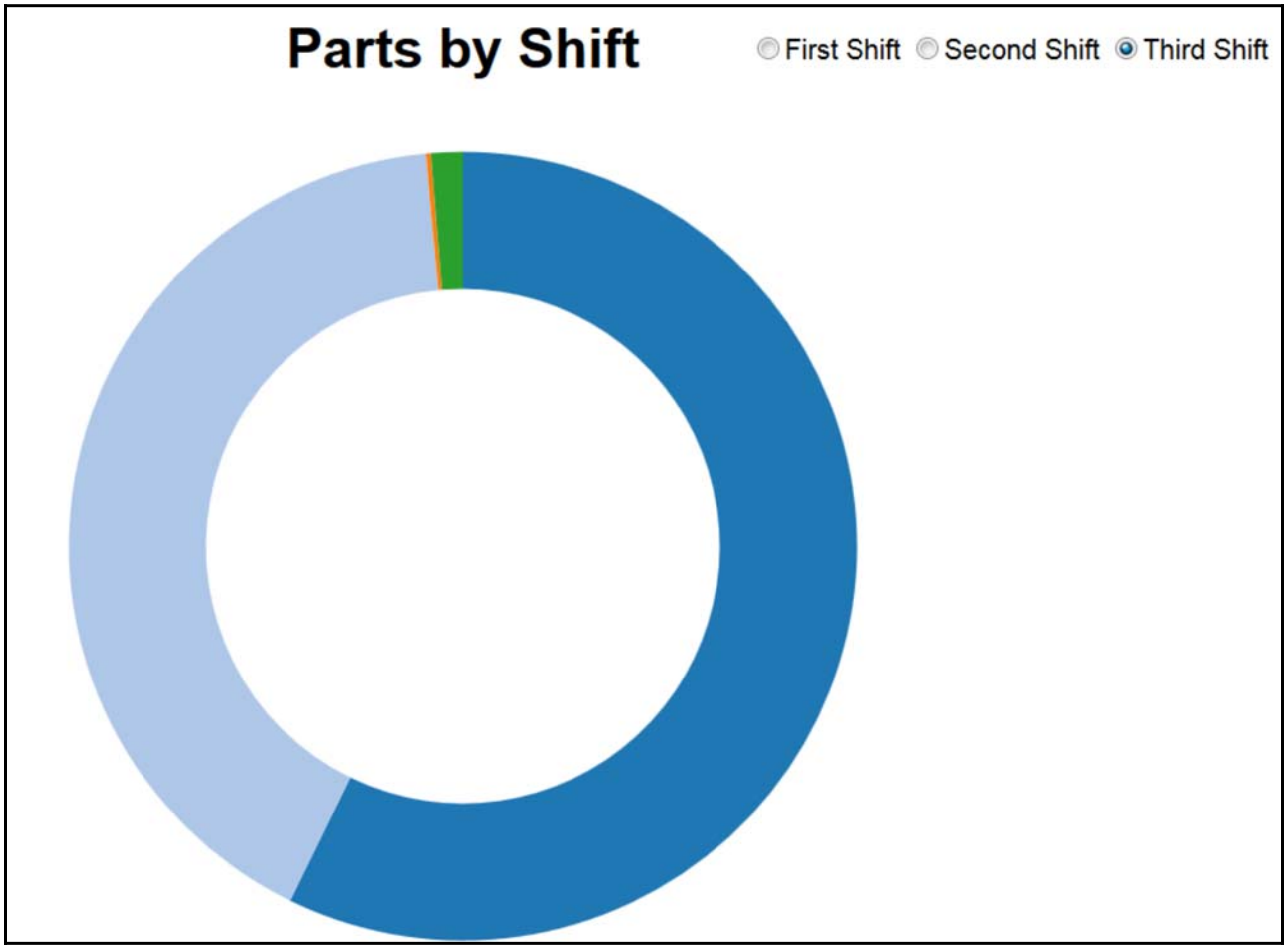
Once we update our HTML document one last time to reflect our summarized file name, we can open it in our web browser.



Now we see a donut pie split by machines for the first shift, but if we click on the radio button to **Change the shift** to **Second Shift**, we see the following:



The third shift looks as follows:



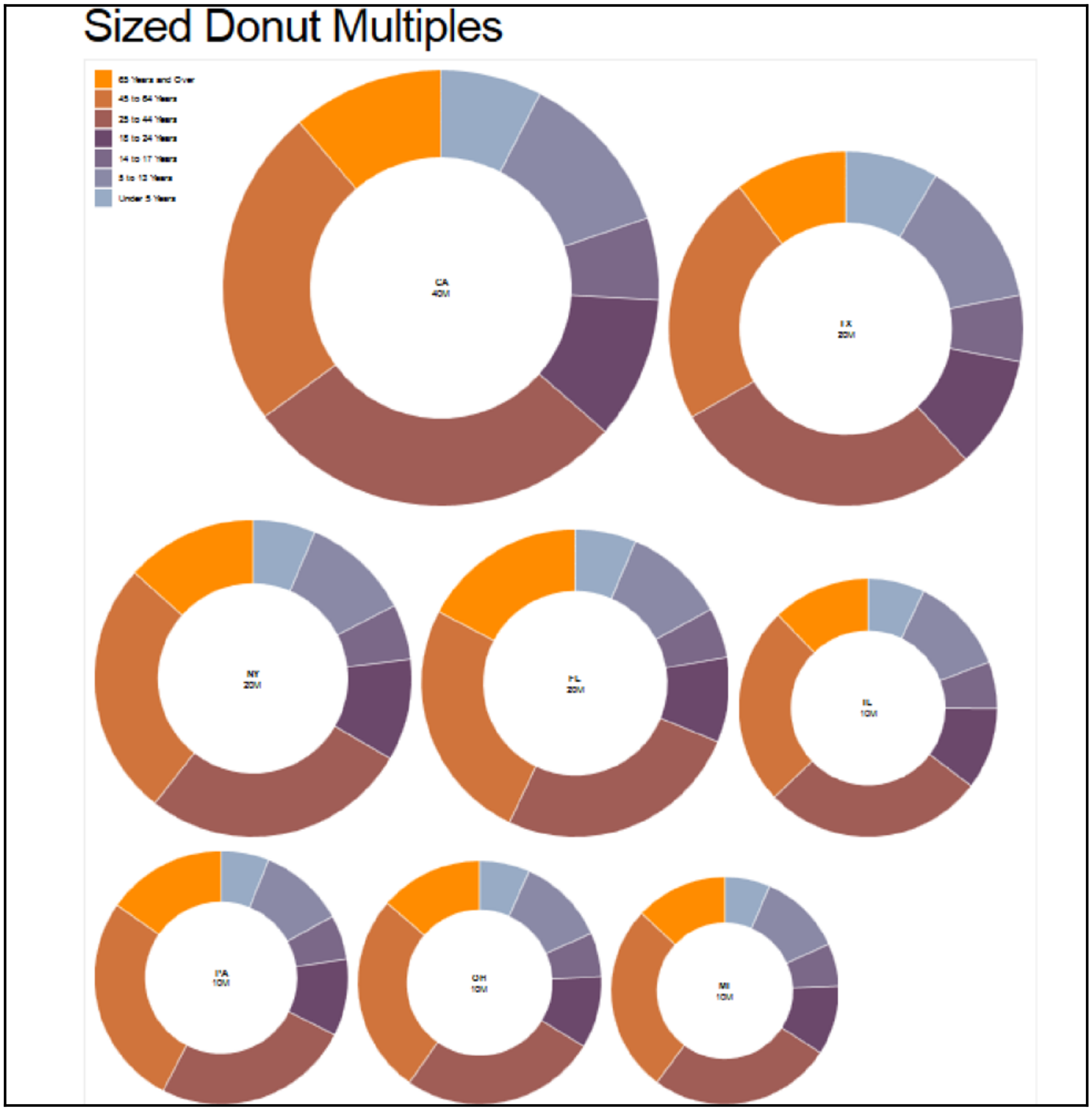
You will notice that for the third shift, machine 004 is not represented (there are only four colors shown in the chart), since it was offline and had no part count in the file.

Multiple donuts

Another interesting and perhaps useful available D3 visualization sample template is the **Sized Donut Multiples** template. This template can be viewed and downloaded from: `http://blocks.org/mbostock/4c5fad723c87d2fd8273` and it shows multiple donuts that are sized so that the area of each donut is proportionate to a total number such that the area of the donut arcs is comparable across all donuts.

In the template, the example uses a summarized data file of state populations, broken out by age group. The file is a comma-delimited file. The example uses the first field (state) as the key to determine the number of donuts to show and the total for each key indicates the size of the donut.

The example D3 visualization is shown in the following figure:

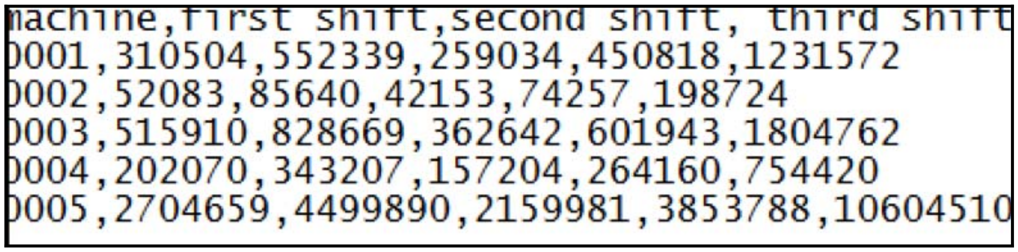


We can easily modify this template to work for us. Instead of states and population totals by age group, we will visualize machines and total parts by shift.

I've pre-processed our raw plant data again, this time creating a summary file of four columns:

- Machine
- First shift
- Second shift
- Third shift

The file has five rows of data, one for each machine. Our summary file looks as follows:



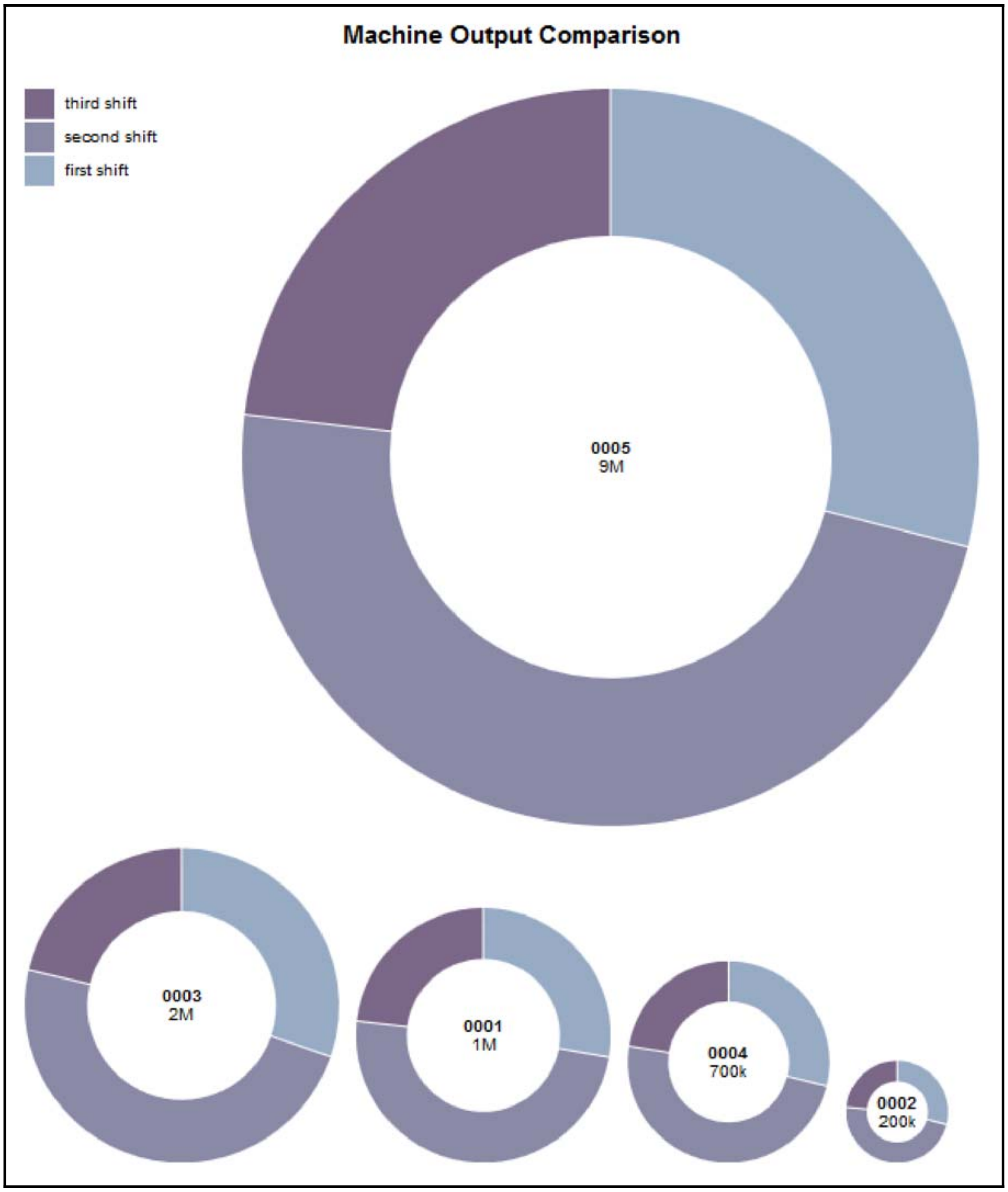
```
achine,first shift,second shift, third shift
001,310504,552339,259034,450818,1231572
002,52083,85640,42153,74257,198724
003,515910,828669,362642,601943,1804762
004,202070,343207,157204,264160,754420
005,2704659,4499890,2159981,3853788,10604510
```

Using the same process to adopt the sample template that we have used for all of the examples throughout this chapter, we:

1. Download the sample template and save it as an HTML document.
2. Change the `src=` reference (to a local reference for the D3 libraries).
3. Change the name of the data file.
4. Add a heading.

In this example, since the D3 sample template uses the first field in the file by name (and that name is used as the visualization key), we need to again make a global change of all references to the word state to the machine.

Once you have completed the preceding changes, we can view our visualization (in a web browser):



You should notice that this visualization adds a legend to each donut, which shows the machine ID or name (for example, **0005**) and the rounded total number of parts with a suffix of M for millions or K for thousands). There is also a specific donut for each machine ID and from the visualization shown, we can see that machine **0005** is the largest contributor of total parts overall (followed by machines **0003**, **0001**, **0004**, and then **0002**).

More examples

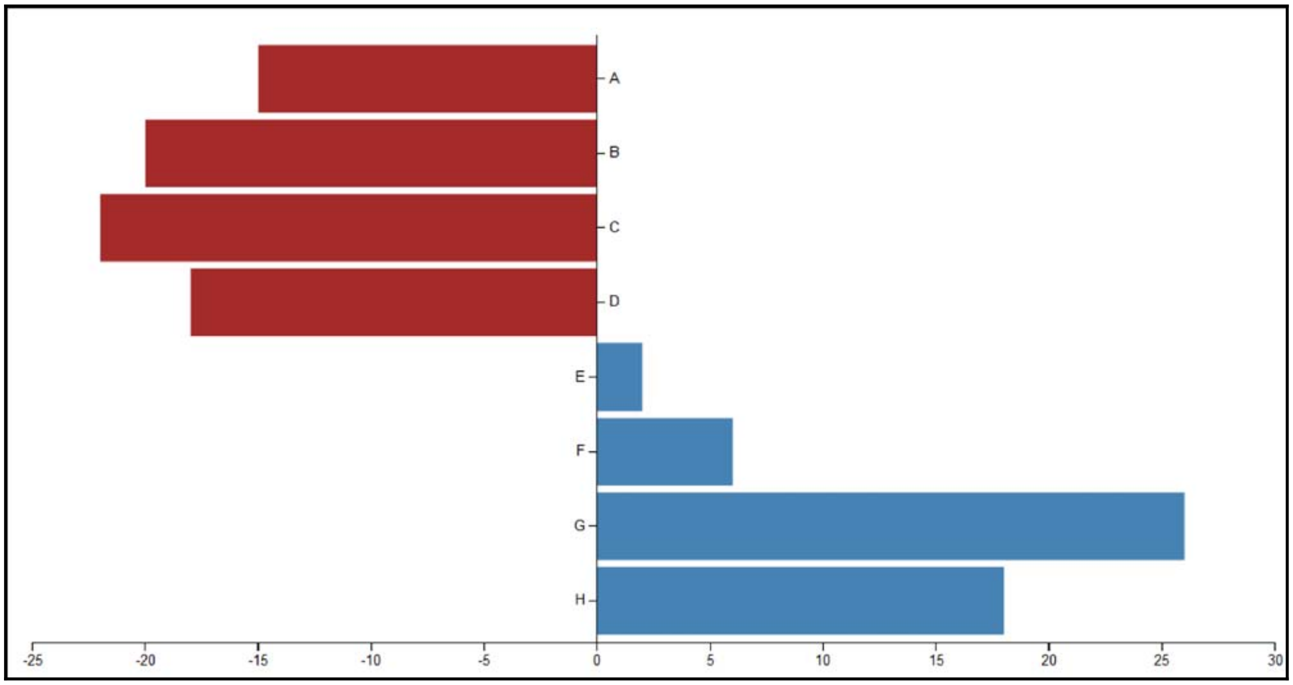
Let us now go a bit further into D3 by looking at some additional examples.

Another twist on bar chart visualizations

A bar chart is a pretty common type of data visualization and we've already seen some examples using the D3 libraries. One more bar-chart D3 template is worth a quick look. This one is able to handle negative values. Think about it–most bar charts show positive values so flipping the tic of an axis (as its referred to) requires some special logic.

The D3 example named *Bar Chart with Negative Values II* handles this kind of scenario nicely and it can be viewed and downloaded from the following location:
`http://blocks.org/mbostock/79a82f4b9bffb69d89ae`.

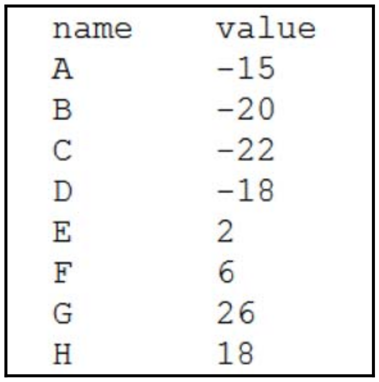
This sample template shows both negative and positive values for the letters **A** through **H**, and it is shown in the following figure:



Getting back to our manufacturing plant, a new challenge has been implemented by management. That is a product output target has been set for each of the plant's machines. Management wants to monitor each machines ability to hit the target and see any deltas to the target. In other words, at any given time how does the machines part count compare to the target number?

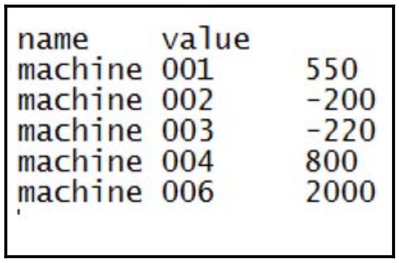
Instead of the eight letters, we want to display our five machines and a horizontal bar showing each machines output compared to the target. So, if we use the previous visualization, machines may have a value of zero, indicating that the machine has hit the target (and there is no delta or difference between the machines part count and the target number), a positive value if that machine has surpassed the targeted total, or even a negative value if the output is less than the targeted value. Since the initial targeted value was set low, management is hoping to see all positive values, of course.

Again, adopting the sample D3 template is an easy process. Really, the key to leveraging the sample D3 templates is to first understand the format of the data that is driving the particular visualization, and then determining what pre-processing and/or manipulating of the raw data is required. In this example, the data is again a very simple summary:



name	value
A	-15
B	-20
C	-22
D	-18
E	2
F	6
G	26
H	18

To make our plant data work, we can aggregate the data to look something like the following:

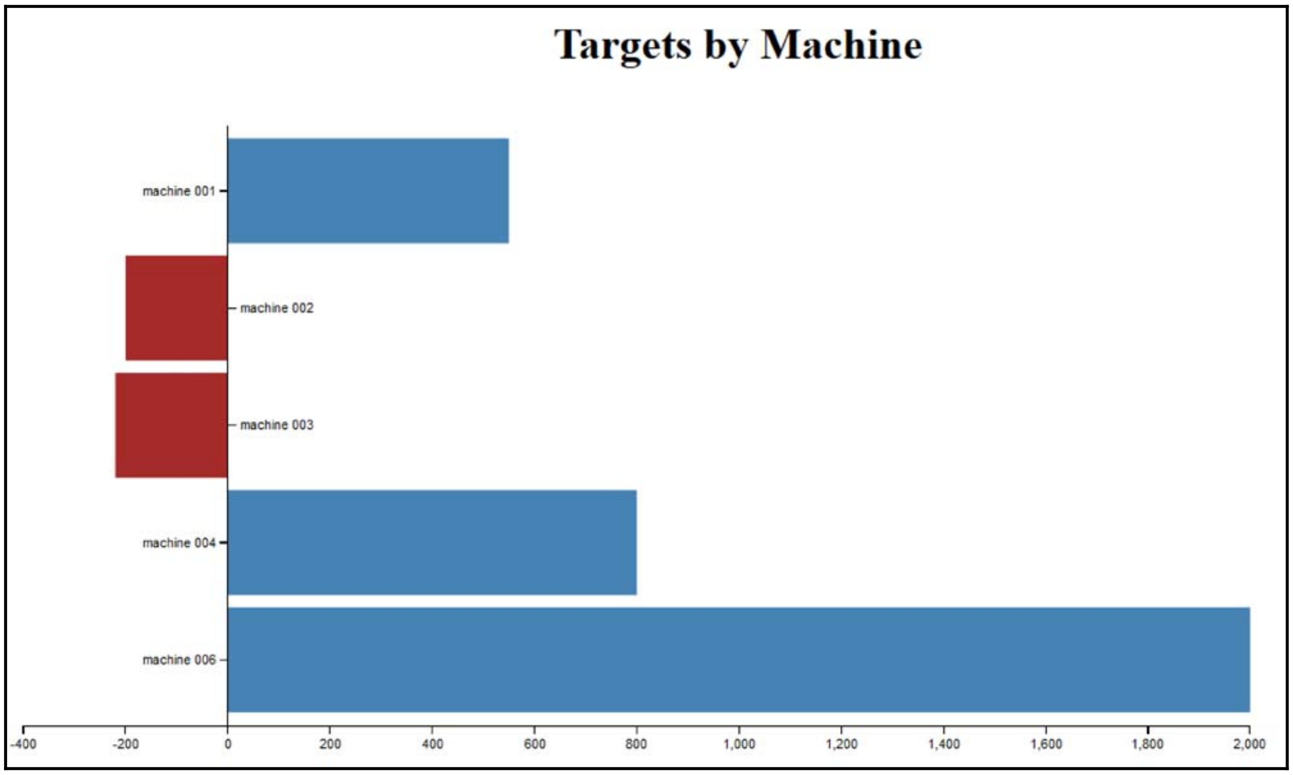


name	value	
machine	001	550
machine	002	-200
machine	003	-220
machine	004	800
machine	006	2000

In this particular example, the process to adopt the sample D3 template is even more straightforward:

1. Download the template and save it as an HTML document.
2. Modify the `src=` reference to be a local reference.
3. Add a heading.
4. Change the file name reference.

Once we have completed the preceding steps, we can view our version of the visualization (in a web browser):



You can see from the visualization generated (shown prior) that **machine 002** and **machine 003** are not hitting the target, while **machine 001**, **machine 004**, and **machine 005** have actually surpassed the target value. The **machine 005** is actually hitting it out of the park by surpassing the target value by 2,000 parts!

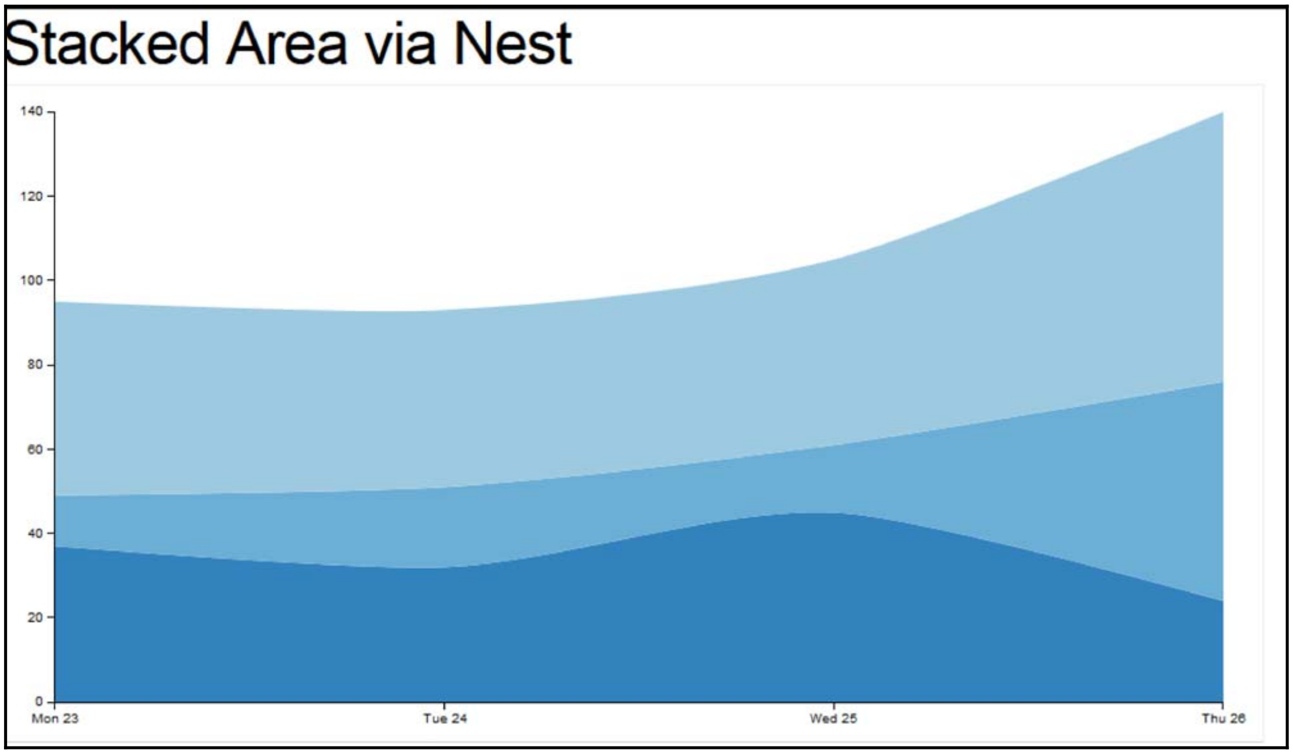
One more example

In our final example, let us now look at the D3 **Stacked Area via Nest** template.

This sample template creates a data visualization driven by a summary data file with three fields:

- A key
- A numeric value
- A date (MM/DD/YY)

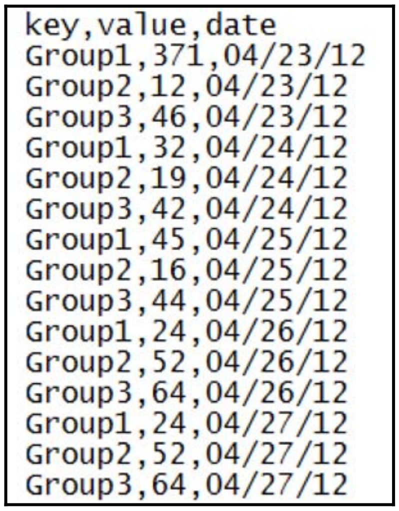
The following figure shows the data visualization generated by the sample D3 template:



Adopting the sample

As usual, the initial step in adopting any D3 sample template is examining the data source and determining what the similarities might be to a visualization we'd like to create.

In this case, we see a file source with only three fields: a key, a numeric value, and a date:



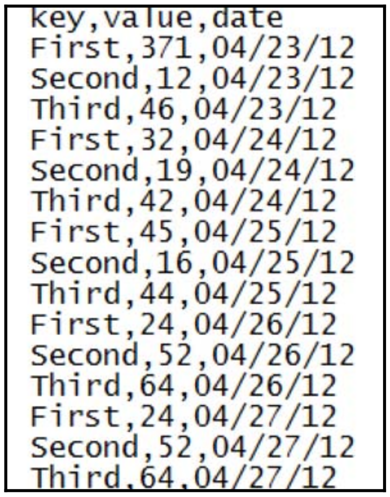
```
key,value,date
Group1,371,04/23/12
Group2,12,04/23/12
Group3,46,04/23/12
Group1,32,04/24/12
Group2,19,04/24/12
Group3,42,04/24/12
Group1,45,04/25/12
Group2,16,04/25/12
Group3,44,04/25/12
Group1,24,04/26/12
Group2,52,04/26/12
Group3,64,04/26/12
Group1,24,04/27/12
Group2,52,04/27/12
Group3,64,04/27/12
```

Comparing the data to the generated output, we see three values represented in the visualization (Group1, Group2, and Group3) across four dates (**4/23/12**, **4/24/12**, **4/25/12**, and **4/25/12**).

Given that observation, we certainly can imagine a manufacturing plant's need to perhaps visualize the three shifts (shift 1, shift 2, and shift 3) across perhaps a week's worth of days (dates). So, continuing, we can then pre-process our raw plant data into a summarized file of the following fields:

- Date/Time
- Shift ID
- Part count

In this way we can create a new summarized file, such as the following:

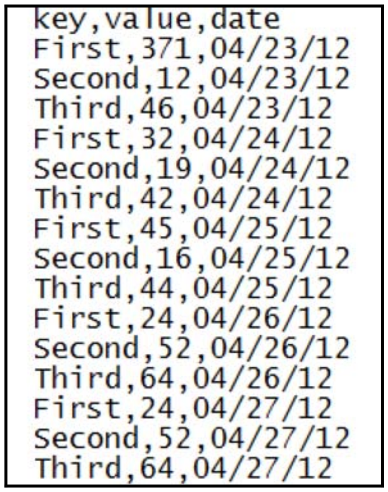


```
key,value,date
First,371,04/23/12
Second,12,04/23/12
Third,46,04/23/12
First,32,04/24/12
Second,19,04/24/12
Third,42,04/24/12
First,45,04/25/12
Second,16,04/25/12
Third,44,04/25/12
First,24,04/26/12
Second,52,04/26/12
Third,64,04/26/12
First,24,04/27/12
Second,52,04/27/12
Third,64,04/27/12
```

We should take a few moments at this time to point out that we followed the following thought process when reviewing the templates sample data:

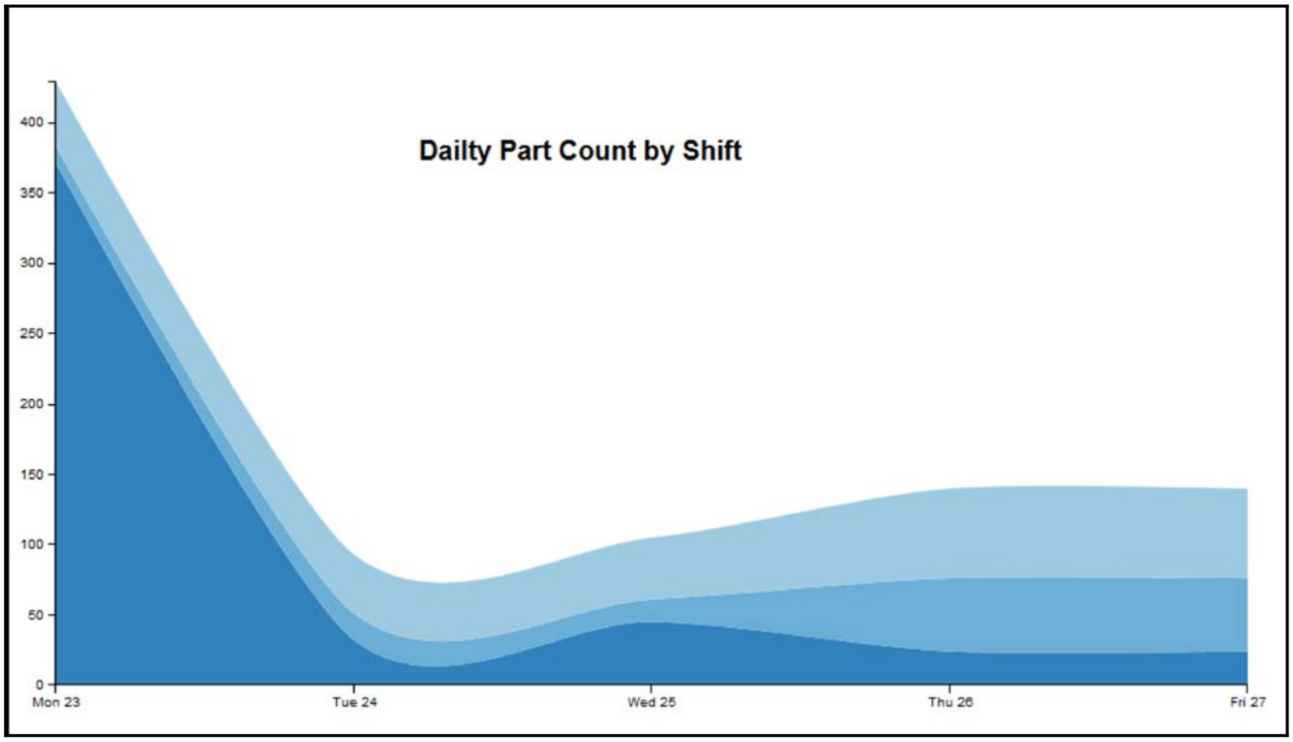
1. The key is grouped and there are three groups. This relates to our three plant shifts.
2. The (numeric) value relates to our (numeric) machine parts count.
3. The date relates to our date–time stamp.

Given these assumptions, we can pre-process or summarize our raw plant data into the appropriate format:



```
key,value,date
First,371,04/23/12
Second,12,04/23/12
Third,46,04/23/12
First,32,04/24/12
Second,19,04/24/12
Third,42,04/24/12
First,45,04/25/12
Second,16,04/25/12
Third,44,04/25/12
First,24,04/26/12
Second,52,04/26/12
Third,64,04/26/12
First,24,04/27/12
Second,52,04/27/12
Third,64,04/27/12
```

We can then generate our visualization:



We assume here that by now (after having read through the examples in this chapter) you know that the process to adopt a selected D3 sample template is:

1. Identify the template/example that seems to fit the objectives.
2. Download the template and save it as an HTML document.
3. Make any required HTML document changes (such as `src=` changes, adding a heading, or changing a data file name reference).
4. Pre-process the raw big data into a summarization file formatted to fit the sample D3 requirements.
5. View the document in a web browser.

For those of us so inclined, if you are unable to find a D3 sample that specifically fits your needs, you may endeavor to modify or enhance the existing D3 libraries as needed. That process is beyond the scope of this particular chapter.

In summary, the D3 libraries provide numerous sample templates–freely available for your adoption (to fit your particular data visualization objectives). What we have covered here in this chapter is just a very simple introduction to a small number of those samples.

By following a few simple, easy steps, one can leverage the D3 libraries to generate dynamic visualizations–driven by your big data summations.

The approach has been to adopt and use what is there, but it should be pointed out that since the D3 libraries are opened sourced, one can, if one is so inclined, customize the code to fit any conceivable specific need.

Summary

In this chapter, we covered the idea of visualizing the results of your big data analysis using D3. Simply put, we walked through step by step, the how to of locating and adopting a D3 template to fit the specific needs of your particular big data analysis.

In the next chapter, we will introduce the concept of visual dashboards and how Tableau can be used to create creative, value-add data dashboards.