

Week 5

Drawing a Scatterplot Part I

Data Import and Parsing

Quick Review

DOM Manipulation Using D3

I. Starting with a DOM element

```
<div class= “container”></div>
```

II. `d3.select()` selects the element, producing a “selection object”

—————→ a “selection” object

```
d3.select(“container”)
```

III. You can then use D3 methods to manipulate this selection

```
d3.select(“container”)
  .append(“div”)
```

“CHAINING” IN D3

```
d3.select(".canvas")  
  .append("svg")  
  .attr("width",width)  
  .attr("height",height)  
  .append("circle")  
  .attr("cx",100)  
  .attr("cy",100)  
  .attr("r",50);
```

d3 allows chaining, but be aware that **the “context”, or the implicit selection, changes.**

- Each **.attr()** call returns the old selection, for you to call a new method onto it;
- Each **.append()** call returns the newly appended element as the new selection, for you to call a new method onto it.

“CHAINING” IN D3

```
.canvas → d3.select(".canvas")  
<svg> → .append("svg")  
          .attr("width",width)  
          .attr("height",height)  
<circle> → .append("circle")  
            .attr("cx",100)  
            .attr("cy",100)  
            .attr("r",50);
```

d3 allows chaining, but be aware that the “context”, or the implicit selection, changes.

- Each `.attr()` call returns the old selection, for you to call a new method onto it;
- Each `.append()` call returns the newly appended element as the new selection, for you to call a new method onto it.

Objects and Arrays as Data Structures

Arrays represent a serial, parallel data structure. It has several important methods and properties:

`array.` Adds a new element to the end

`array.` Returns the number of elements in the array

Objects and Arrays as Data Structures

Arrays represent a serial, parallel data structure. It has several important methods and properties:

`array.push()` Adds a new element to the end

`array.length` Returns the number of elements in the array

How do I access the 3rd value in this array `[7,23,4,12,0]` ?

Objects and Arrays as Data Structures

How do I access the 3rd value in this array

```
var newArray = [7,23,4,12,0] ?
```

```
newArray[3] //4
```

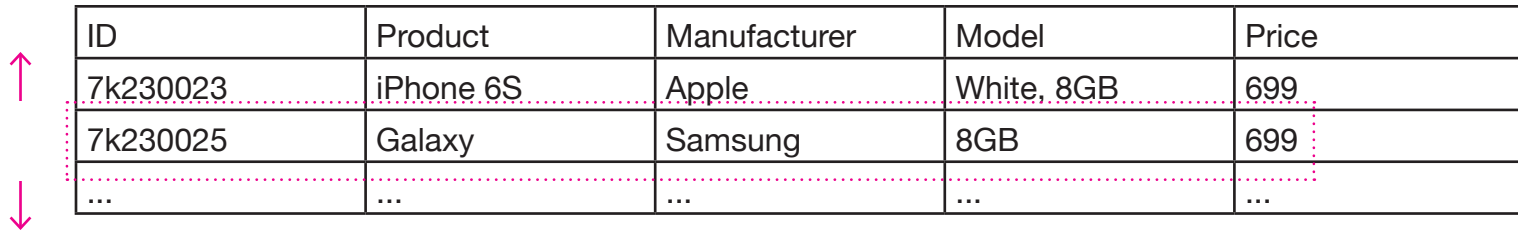

Objects and Arrays as Data Structures

Arrays and objects in combination can begin to help us represent more complex data structures, such as a database:

ID	Product	Manufactuer	Model	Price
7k230023	iPhone 6S	Apple	White, 8GB	699
7k230025	iPhone 6S	Apple	Gold, 8GB	699
...

Objects and Arrays as Data Structures

Arrays and objects in combination can begin to help us represent more complex data structures, such as a database:



ID	Product	Manufacturer	Model	Price
7k230023	iPhone 6S	Apple	White, 8GB	699
7k230025	Galaxy	Samsung	8GB	699
...

```
[
  {
    ID: "7k230023",
    Product: "iPhone 6S",
    Manufacturer: "Apple",
    ...
  },
  { ID: "7k230025", Product: "Galaxy", ...}
]
```

Array of objects

Exercise 1

Let's quickly refresh these concepts with a drawing exercise. In particular, we'll focus on using the `.select()`, `.selectAll()`, `.attr()`, and `.style()` methods.

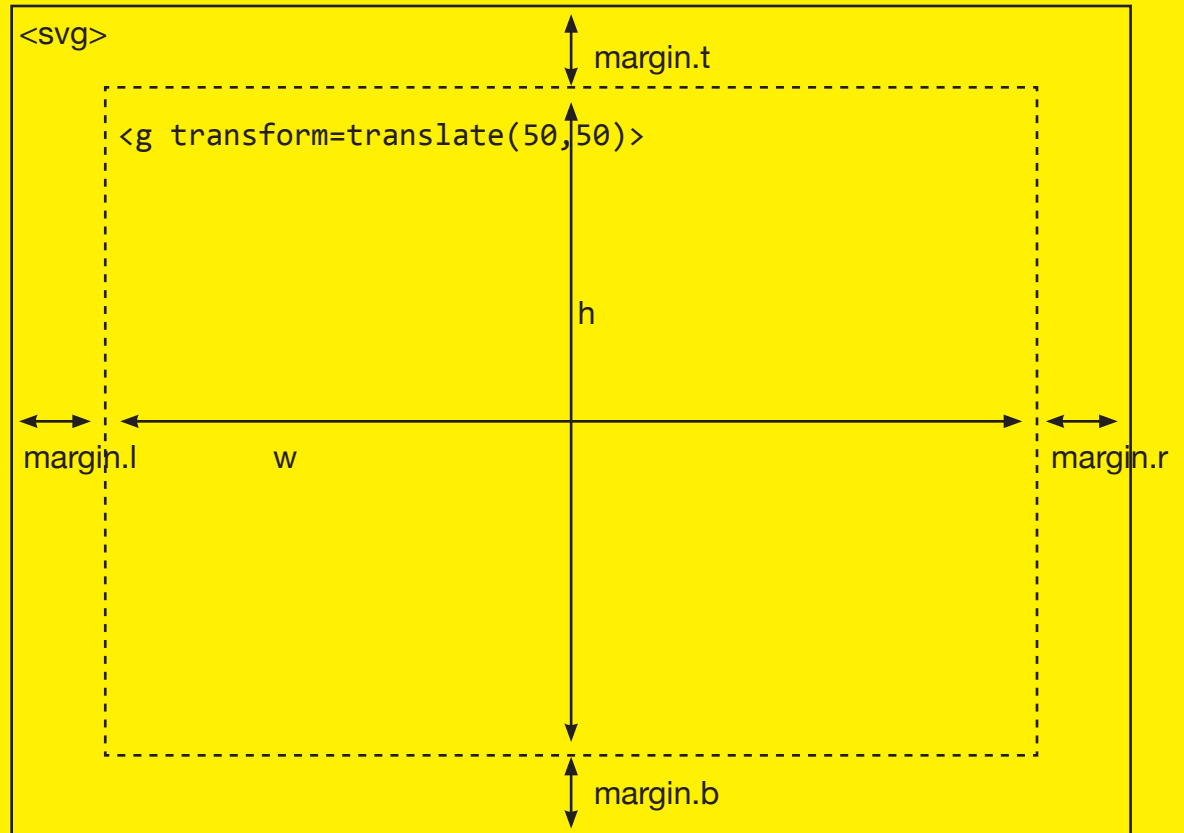
Aside: Margin Conventions

As a convention, we use a `<g>` element offset from the edges of the `<svg>` element to create a margin.



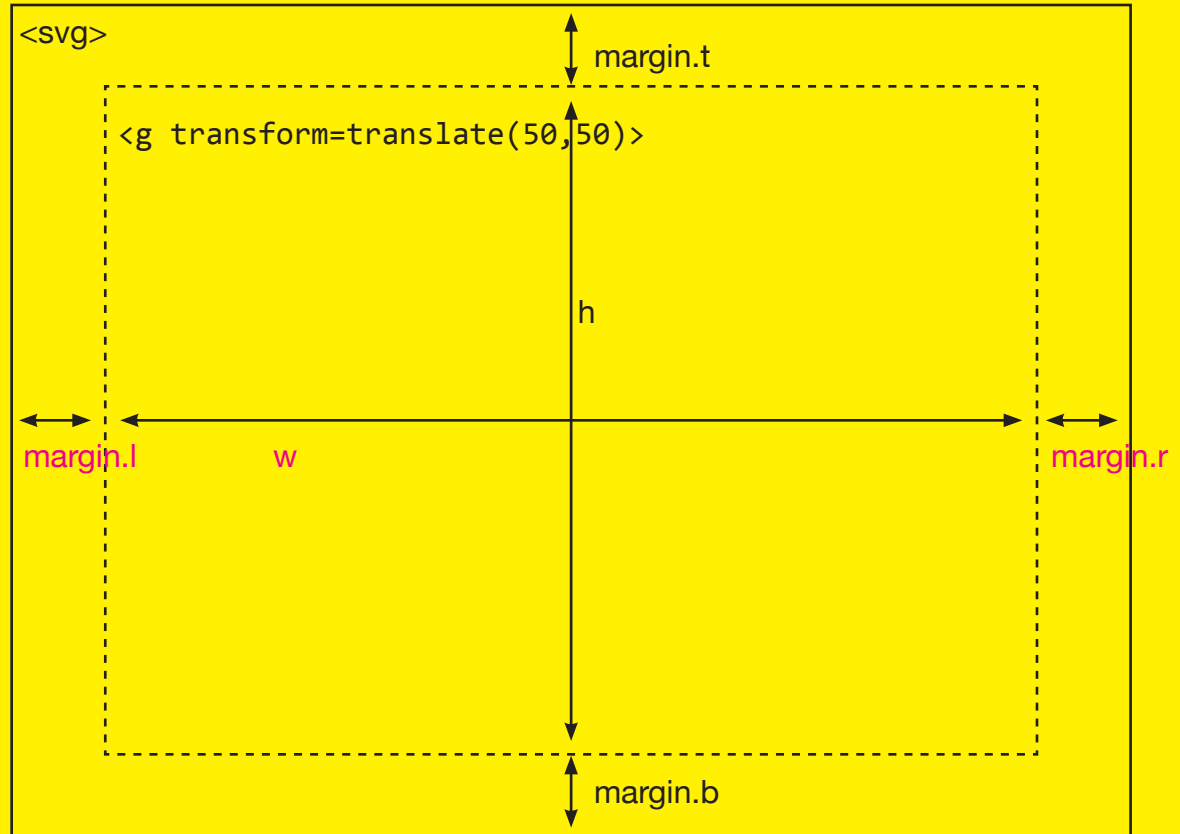
Aside: Implementing Margin Conventions

As a convention, we use a `<g>` element offset from the edges of the `<svg>` element to create a margin.



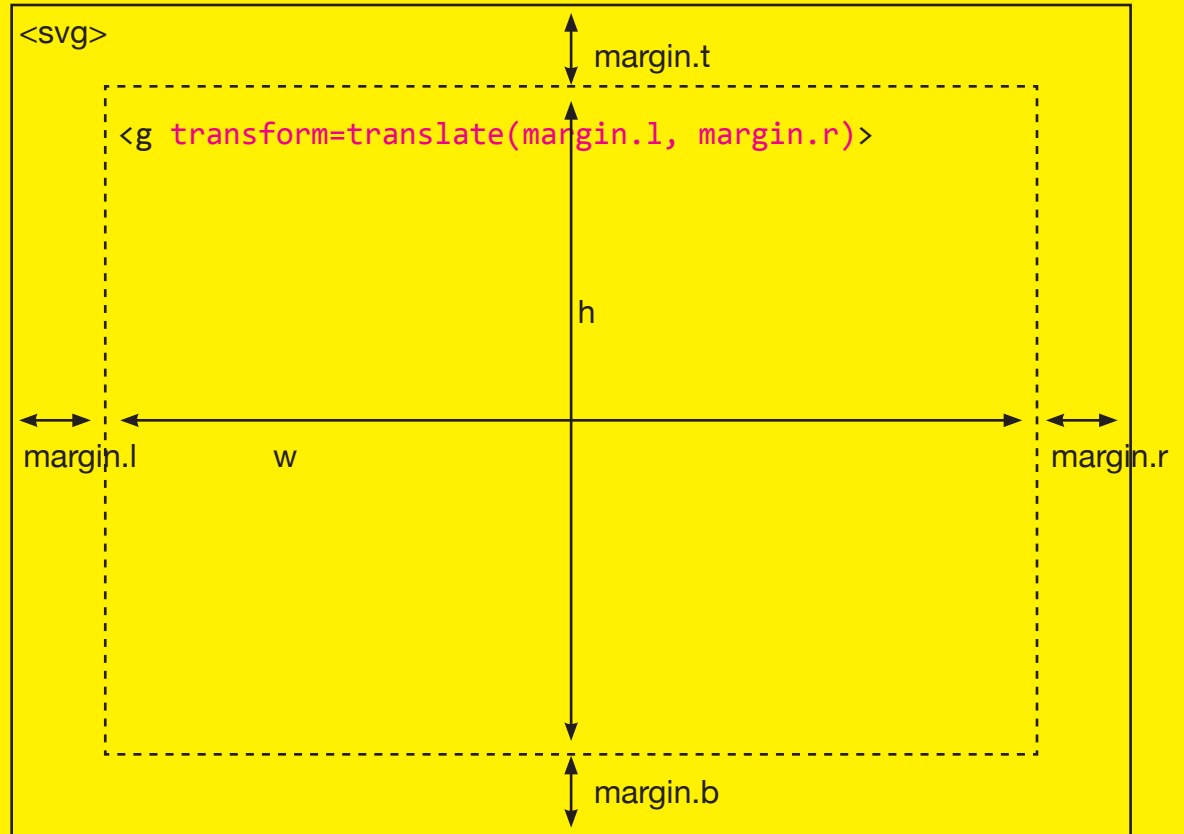
Aside: Implementing Margin Conventions

```
var margin = {  
  t: 50,  
  r: 50,  
  b: 50,  
  l: 50  
};  
  
var plot = ...  
  .append('svg')  
  .attr('width', w +  
margin.l + margin.r)  
  .attr('height', h +  
margin.t + margin.b)  
  .attr('g')  
  .attr('transform',  
'translate('+margin.l+  
'/'+margin.r+ '');
```



Aside: Implementing Margin Conventions

```
var margin = {  
  t: 50,  
  r: 50,  
  b: 50,  
  l: 50  
};  
  
var plot = ...  
  .append('svg')  
  .attr('width', w +  
margin.l + margin.r)  
  .attr('height', h +  
margin.t + margin.b)  
  .attr('g')  
  .attr('transform',  
'translate('+margin.l+  
'/'+margin.r+ '');
```



Working with Data: Drawing a Scatterplot

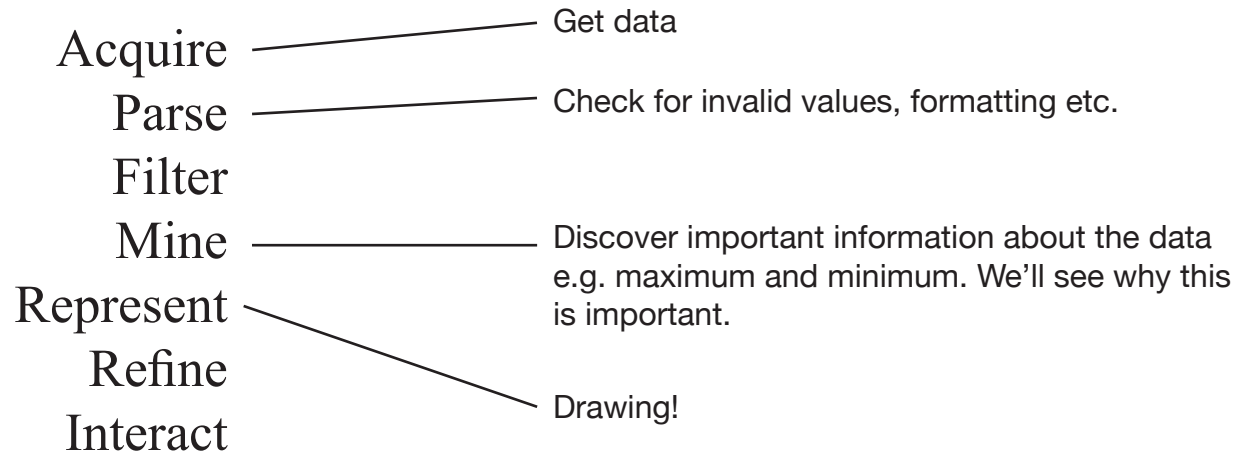
From Wikipedia, a scatterplot is a kind of data visualization where:

“The data is displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis.”

Scatterplots are useful for studying the relationship between two variables.

To Begin, Let's Generalize the Problem

Ben Fry's famous visualization pipeline is a useful guide for what we'll try to do.



To Begin, Let's Generalize the Problem

Let's begin with a simple example.

Acquire ————— How do we do this?
Parse
~~Filter~~
~~Mine~~
Represent
~~Refine~~
~~Interact~~

Exercise 2

Two things: first, create a simple python HTTP server using the following terminal command

```
python -m SimpleHTTPServer or  
python -m http.server
```

Next, open up the /data folder in Exercise 2, and take a look at `data.csv`. This file contains some mock data that we need to import.

What is .csv

A special text file representing tabular data, with “,” (commas) separating, or “delimiting”, data fields.

Tabular Data

ID	Product	Manufacturer	Model	Price
7k230023	iPhone 6S	Apple	White, 8GB	699
7k230025	Galaxy	Samsung	8GB	699
...

CSV

```
ID,Product,Manufacturer,Model,Price
7k230023,iPhone 6S,Apple,"White, 8GB",699
7k230025,Galaxy,Samsung,8GB,699
```

From .csv to Array of Objects

But .csv files are one step removed from the array-object representation we are familiar with:

CSV

```
ID,Product,Manufacturer,Model,Price
7k230023,iPhone 6S,Apple,"White, 8GB",699
7k230025,Galaxy,Samsung,8GB,699
```

JavaScript array of objects

```
[
  {
    ID: "7k230023",
    Product: "iPhone 6S",
    Manufacturer: "Apple",
    ...
  },
  { ID: "7k230025", Product: "Galaxy", ...}
]
```

Importing Data Using D3

A d3 function allows us to import .csv data as an array of objects, and optionally allows parsing.

[] denotes optional argument

```
d3.csv(url, [accessor, ] callback)
```

url is a **string**, and is the file path of the .csv file.

accessor is a **function**. It receives 1 argument, which is the pre-parse data of a single row.

callback is a **function**. It receives 2 argument. The first is an error object. The second is the array of objects.

Exercise 2

Now let's import the data contained in `/data/data.csv`, using the `d3.csv()` function.

Let's also think about the sequence in which things happen.

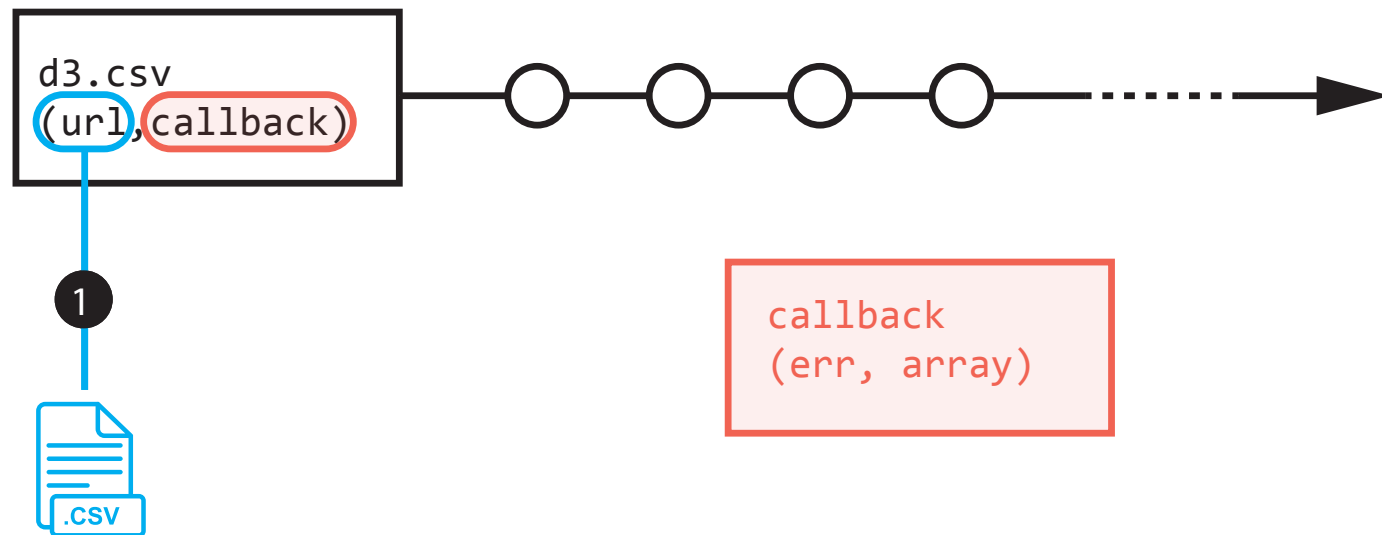
Importing Data Using D3

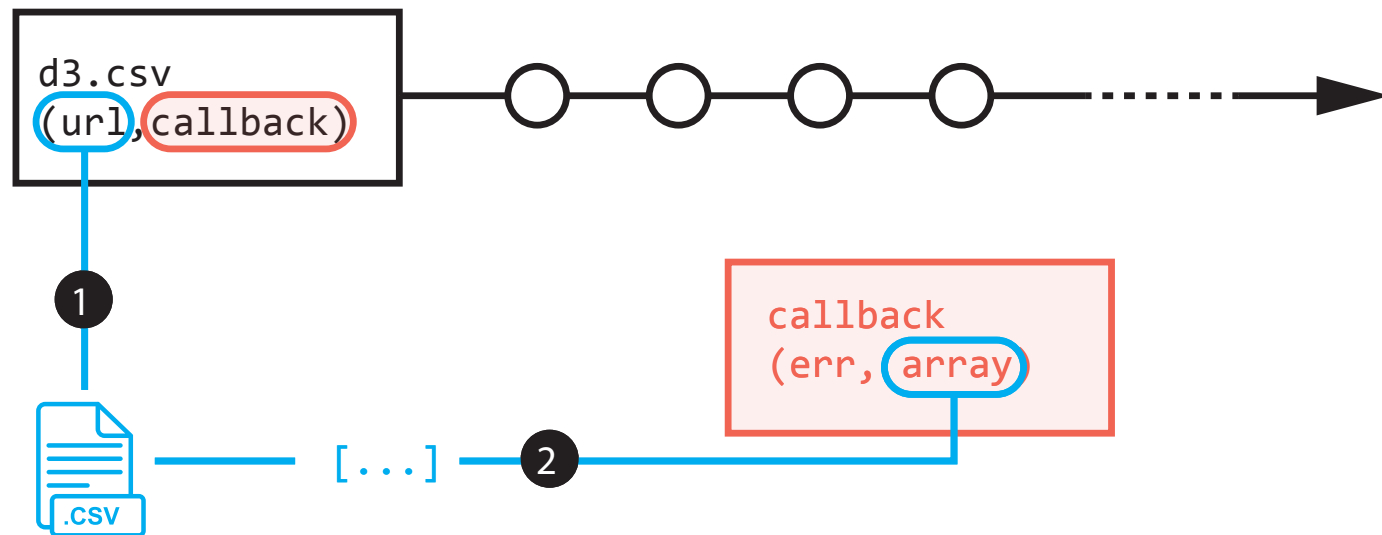
Let's pay particular attention to the third argument.

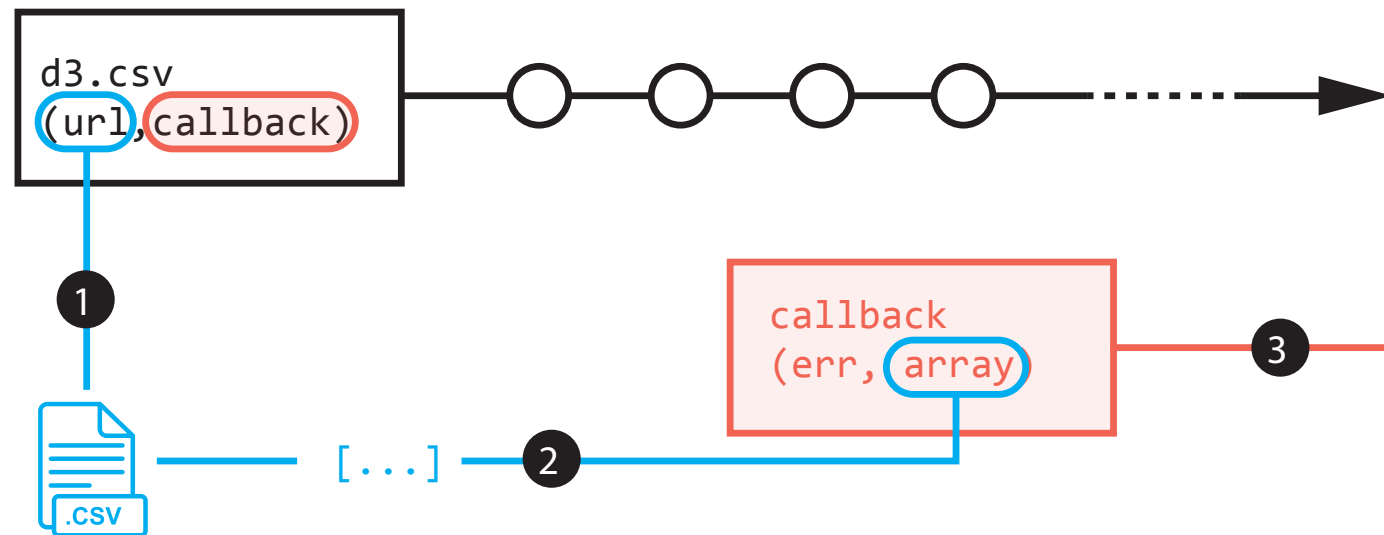
```
d3.csv(url, accessor, callback)
```

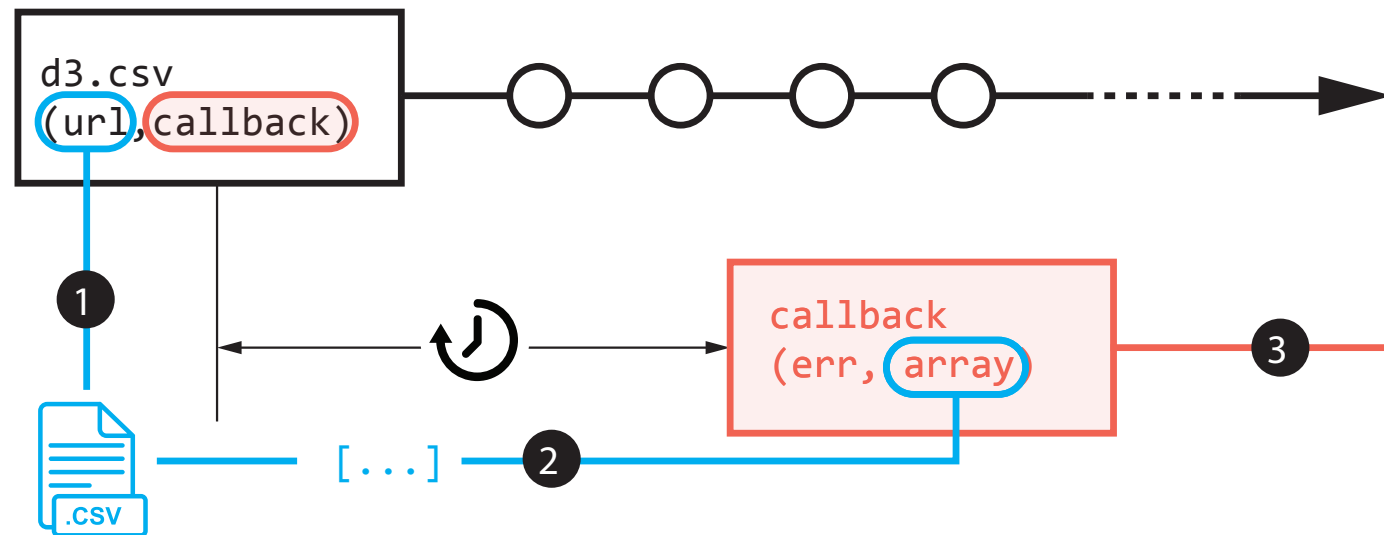
The reason it's called a callback function is that it is executed only after the data is fully loaded and parsed. We don't know how long that takes.

This example of the event-callback pattern allows us to deal with this complication.









Exercise 2

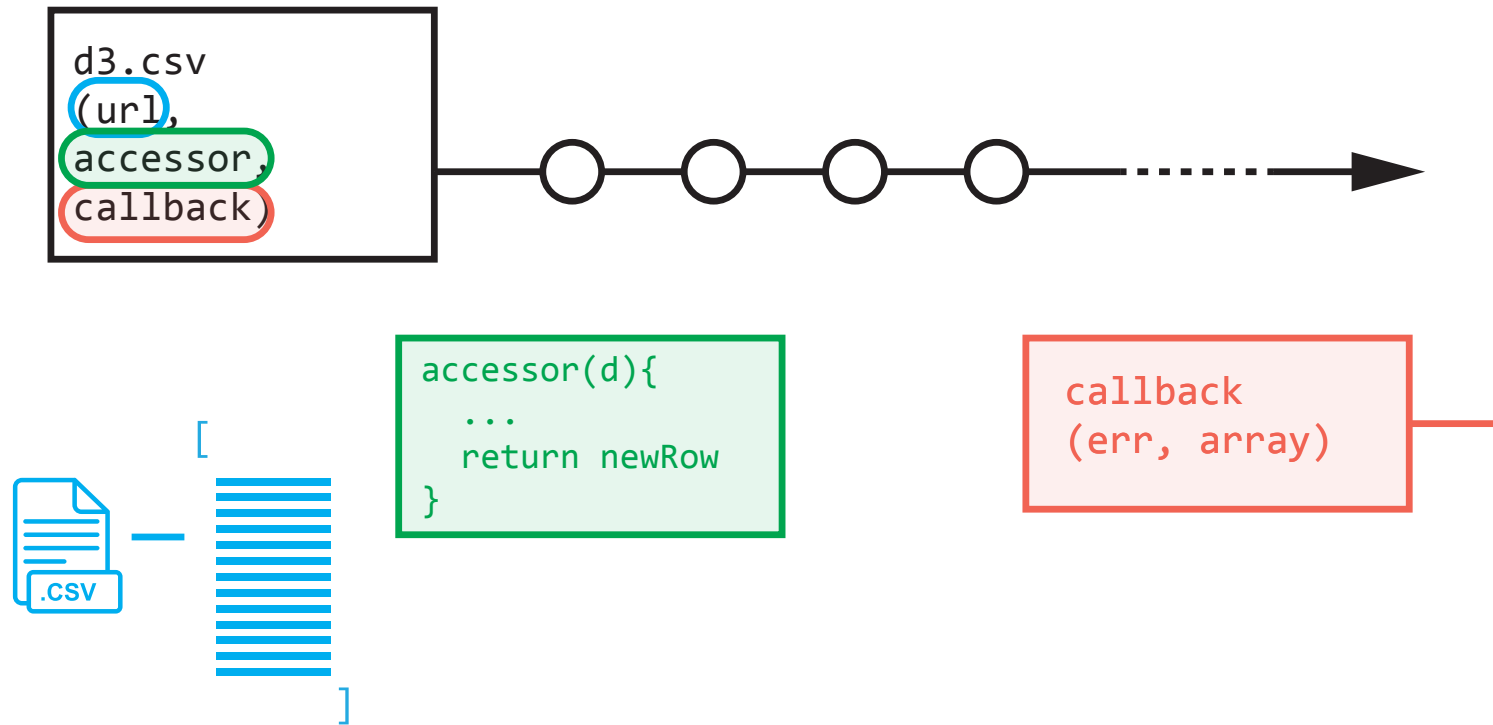
Now that we've loaded the data fully, we can follow through with the rest of the visualization pipeline.

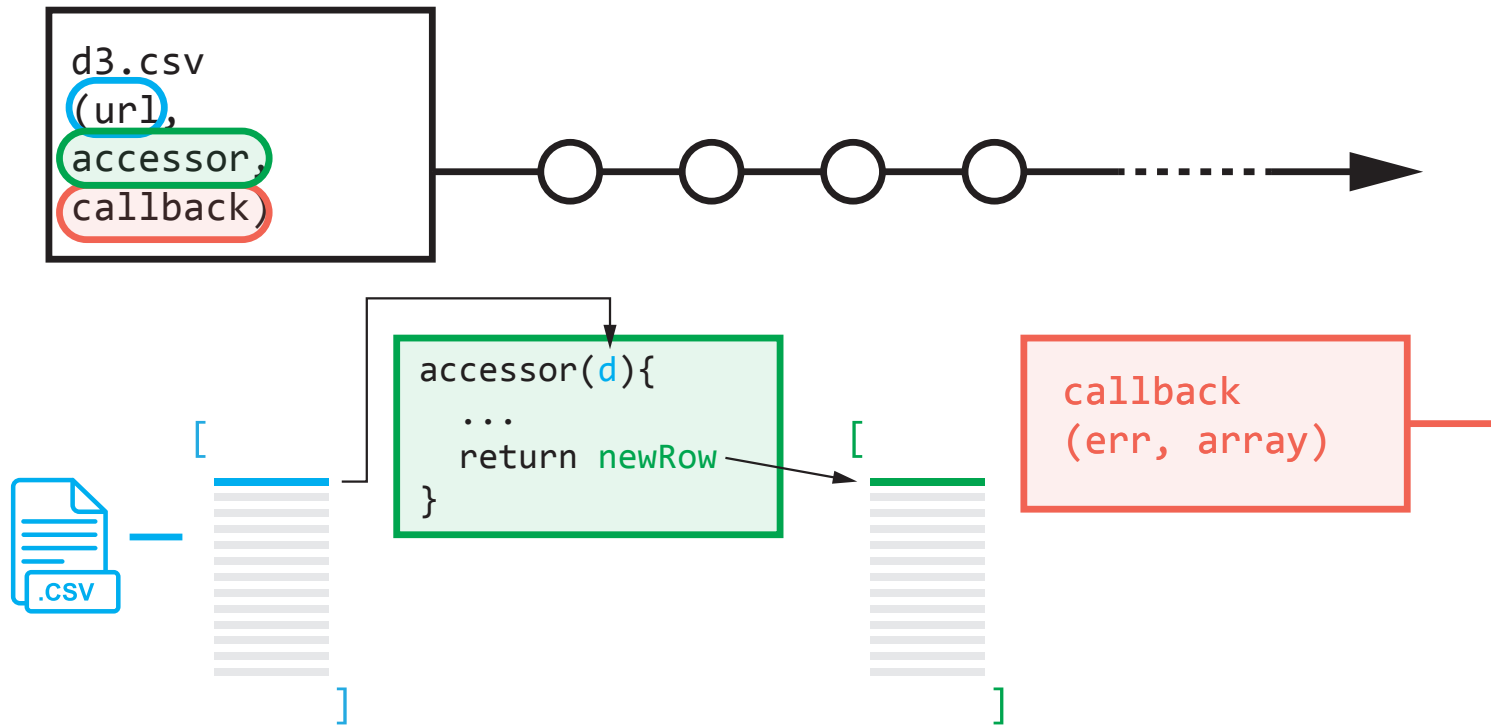


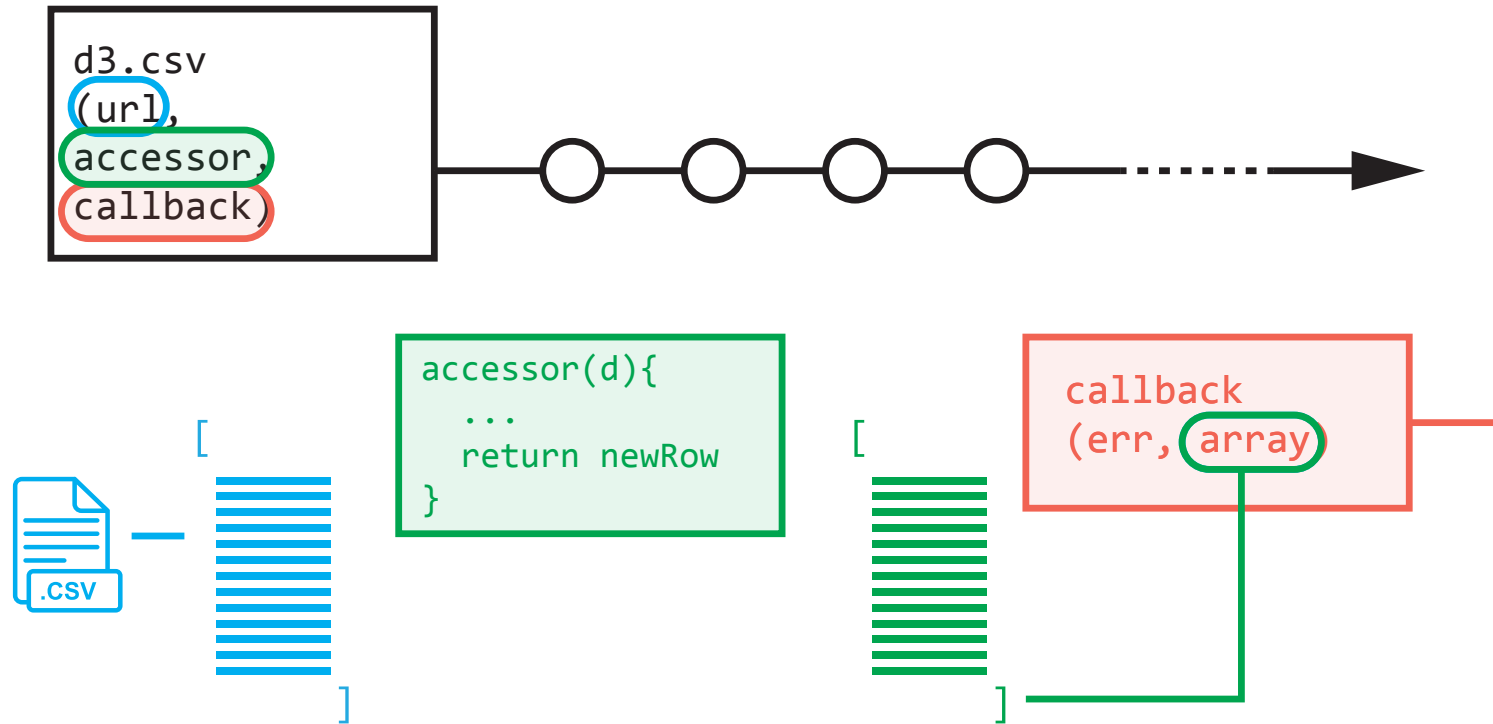
Using the Accessor to Parse

`d3.csv(url, accessor, callback)`

```
d3.csv("data/data.csv", function(d){  
    console.log(d);  
    var parsedRow = {  
        x: +d.x,  
        y: +d.y,  
        r: +d.r  
    }  
    console.log(parsedRow);  
    return parsedRow;  
}, function(error, rows){  
    //...  
});
```

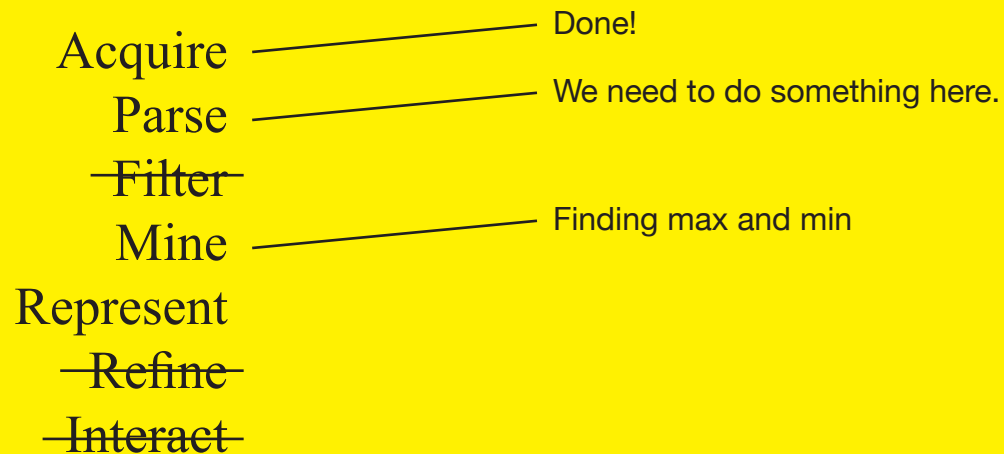






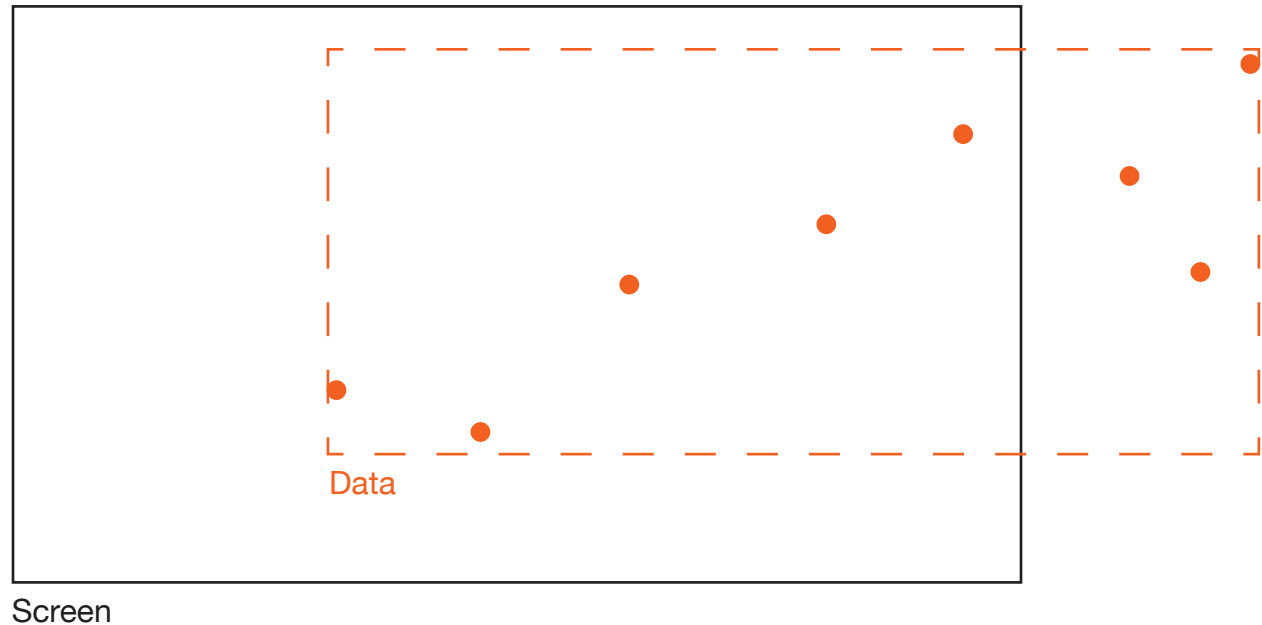
Exercise 3

Let's look at an even more realistic data set, and see the kind of work we have to do.



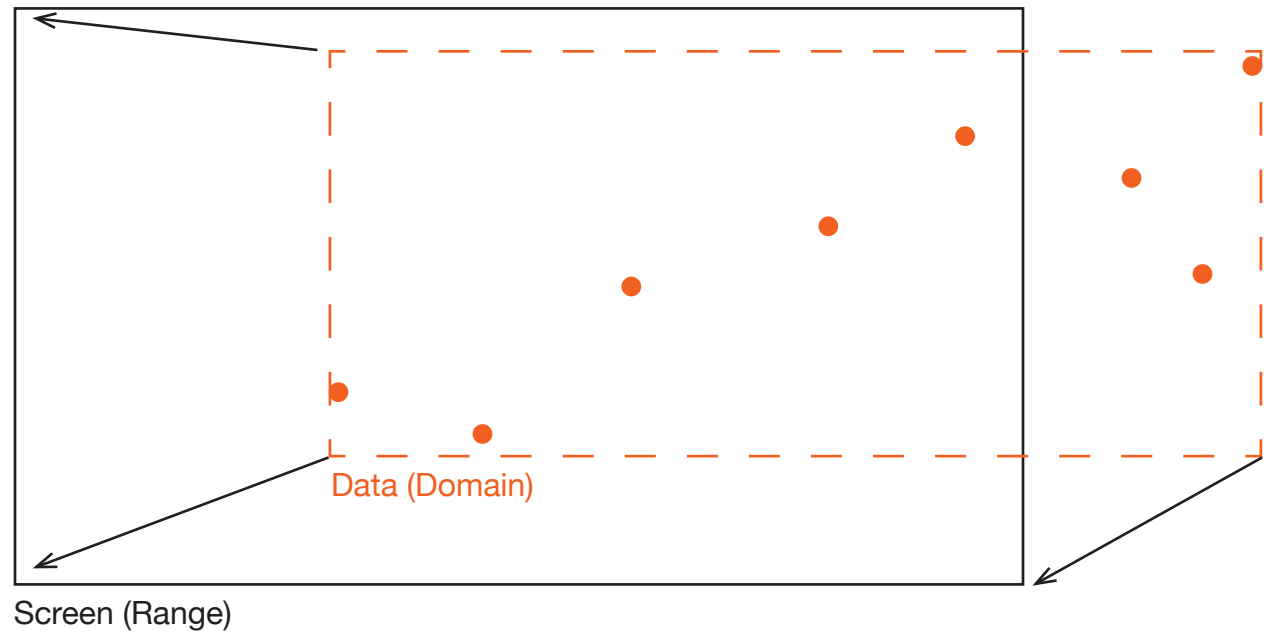
Scaling Data

The data in Exercise 3 is different from Exercise 2 in that it doesn't fit on the screen.



Scaling Data

Often, we need to map data from one space to another. This is referred to as mapping from “domain” to “range”.



Using `d3.scale`

`d3` provides a convenience function for us to map data from domain to range: `d3.scaleLinear`

First, we need to define a new scale, and define its domain and range:

```
var newScale = d3.scaleLinear()  
    .domain([100,600])  
    .range([0,1200]);
```

This returns a scaling function. Later, we can convert any number from domain space to range space by calling this function.

```
newScale(350); //returns 600
```

Using `d3.scale`

Many scales to choose from, but `d3.scale.linear()` is the most intuitive one.

```
var newScale2 = d3.scaleLinear()  
  .domain([100,150])  
  .range([0,1600]);
```

```
newScale2(125); //??  
newScale2(150); //??  
newScale2(50); //??  
newScale2.invert(1600); //??
```

Using `d3.min/max`

One last issue: I know what my range is (the dimension of my `<div>` element), but what is the domain?

We need to manually find the max and min of the data in both the x- and y-directions! For that, we use:

```
d3.min(array, accessor)
```

Where `accessor` is a function that iterates over the array.

Exercise 3

This is why we need to mine the data!

