Data Handling and Aggregation

VISUALIZATION ON THE WEB

Adapted from: http://learnjsdata.com/read_data.html

DATA LOADING

CSV File Format

- Text-based representation of tabular data
- A file contains a set of rows/lines
  - Each line has a sequence of fields, separated by commas
  - Optionally, the first row of the file may contain field names
- By convention, the separator is a comma ‘,’
  - We may also have tabs ‘\t’ (called TSV) or other symbolds (callsed DSV)

CSV Example

```
source,target,value,groupsource,grouptarget
Napoleon,Myriel,1,1,1
Mlle.Baptistine,Myriel,8,1,1
Mme.Magloire,Myriel,10,1,1
Mme.Magloire,Mlle.Baptistine,6,1,1
CountessdeLo,Myriel,1,1,1
Geborand,Myriel,1,1,1
Champtercier,Myriel,1,1,1
Cravatte,Myriel,1,1,1
Count,Myriel,2,1,1
OldMan,Myriel,1,1,1
Valjean,Labarre,1,2,2
Valjean,Mme.Magloire,3,2,1
[...]
```

JSON Format

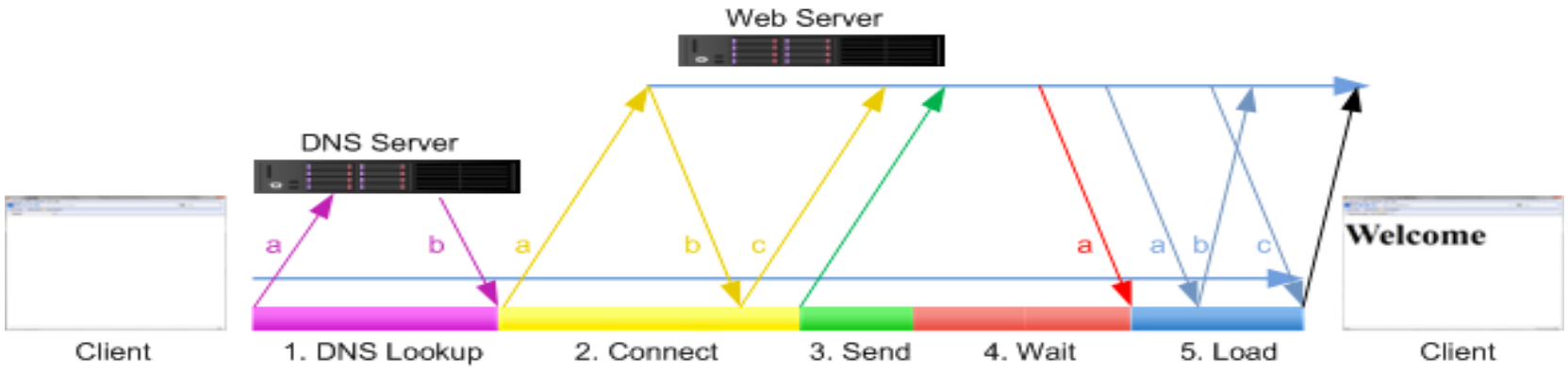
- Text-based representation of hierarchical data
- Used to encode structured objects
- Based on the definition of a key-value pairs

JSON Example

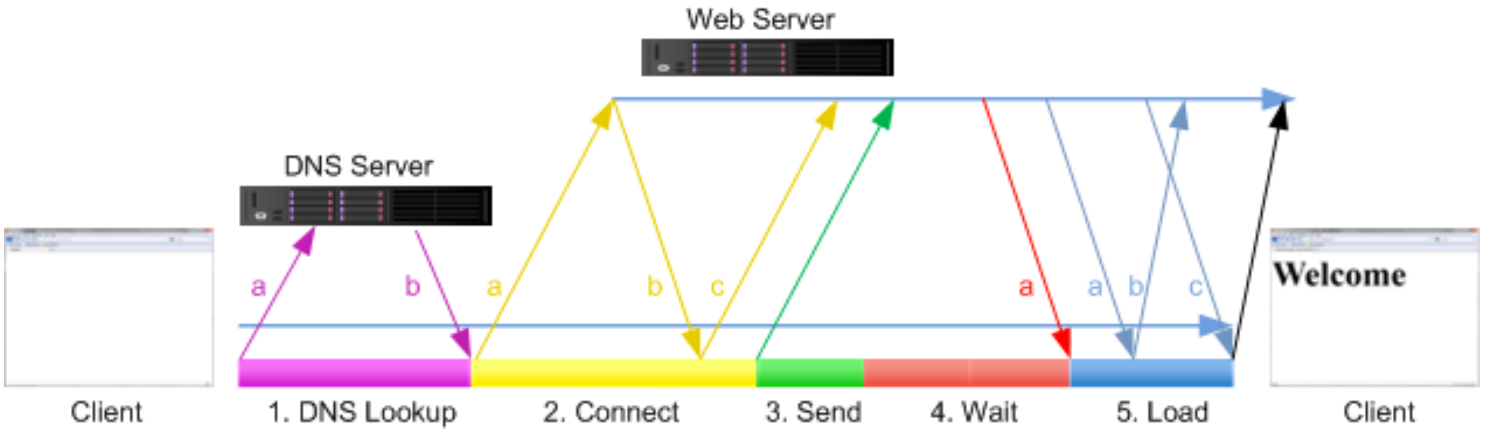

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

HTTP

- 3. Send
- HTTP Request
  - Methods to tell server what the client need
- HTTP Methods:
  - GET; POST; PUT; DELETE; OPTIONS;…



HTTP



- 4. Wait and 5. Load
- HTTP Response
  - Read Response Codes
  - Read data

- HTTP Response Codes
  - 1xx – Informational
  - 2xx – Success
  - 3xx – Redirection
  - 4xx – Client Error
  - 5xx – Server Error

Data loading - Promises

- The most common file types are handled by specific functions in D3
  - csv, json, tsv, dsv
- CSV

```
d3.csv("/data/cities.csv»)
   .then(function(data) {
      // data contains the whole dataset
      console.log(data[0]);
    })
    .catch(function(error){
       // handle error
    });
=> {city: "seattle", state: "WA", population: "652405",
land area: "83.9"}
```

Data processing and cleaning

```
=> {city: "seattle", state: "WA", population: "652405", land area: "83.9"}
```

- All values are parsed as string. We need to do type conversion manually.
- We iterate over an array of values by using an iterator anonymous function with map()

```
d3.csv("/data/cities.csv»)
 .then(function(data) {
   return data.map(function(d){
     return {
       city : d.city,
       state : d.state,
       population : +d.population,
       land_area : +d["land area"]
     })
 })
 .catch(function(error){
   // ...
 };
```

Loading Multiple files

- To load multiple files that are needed for the application, we can use `Promise.all()` method

```
const files = ["/data/cities.csv", "/data/animals.csv"];
let promises = [];

files.forEach(function(url){
  promises.push(d3.csv(url));
});

Promises.all(promises)
  .then(function(values){
    analyze(values[0], values[1])
  });

function analyze(cities, animals) {
  console.log(cities[0]);
  console.log(animals[0]);
}
=> {city: "seattle", state: "WA", population: "652405", land area: "83.9"}
{name: "tiger", type: "mammal", avg_weight: "260"}
```

Statistics and Summary

```
var data = [
  {"city":"seattle", "state":"WA",
"population":652405, "land_area":83.9},
  {"city":"new york", "state":"NY",
"population":8405837, "land_area":302.6},
  {"city":"boston", "state":"MA",
"population":645966, "land_area":48.3},
  {"city":"kansas city", "state":"MO",
"population":467007, "land_area":315}
];
```

Statistics and summary of data

- Min, Max and Extent

```
var minLand = d3.min(data, function(d)
{ return d.land_area; });
console.log(minLand); // => 48.3
var maxLand = d3.max(data, function(d)
{ return d.land_area; });
console.log(maxLand); // => 315
var landExtent = d3.extent(data, function(d)
{ return d.land_area; });
console.log(landExtent);  // => [48.3, 315]
```

- Average, Median, Deviation

Statistics and summary of data

- Min, Max and Extent
- Average, Median, Deviation

```
var landAvg = d3.mean(data, function(d) { return
d.land_area; });
console.log(landAvg); // => 187.45
var landMed = d3.median(data, function(d)
{ return d.land_area; });
console.log(landMed); // => 193.25
var landSD = d3.deviation(data, function(d)
{ return d.land_area; });
console.log(landSD); // => 140.96553952414519
```

Iteration, Map and Reduce

- Javascript `map()` function allows to transform our data into a new dataset
- The function takes an array in input and produce a new array with the result of calling a function on each element of the input

map() example

```
var smallData = data.map(function(d,i) {
  return {
    name: d.city.toUpperCase(),
    index: i + 1,
    rounded_area: Math.round(d.land_area)
  };
});
console.log(data[0]);
console.log(smallData[0]);
```

```
=> {city: "seattle", state: "WA", population:
652405, land_area: 83.9}
  {name: "SEATTLE", index: 1, rounded_area: 84}
```

Filtering

- To select a subset of avaiable rows we use the function filter()

```
var large_land = data.filter(function(d)
{ return d.land_area > 200; });
console.log(JSON.stringify(large_land));
=> [{"city":"new
york","state":"NY","population":
8405837,"land_area":302.6},
  {"city":"kansas
city","state":"MO","population":
467007,"land_area":315}]
```

Sorting

- To sort rows of a dataset, we use the function `sort()`
- The sorting is done inplace (it modifies the original data)
- Sorting is done according to a comparator function. The comparator is given two entries `a` and `b` of the data and should return -1 (if `a` is smaller than `b`), 0 (if `a` and `b` are equal), +1 (if `a` il larger than `b`)

Sorting Example

```
data.sort(function(a,b) {
  return b.population - a.population;
});
console.log(JSON.stringify(data));
=> [{"city":"new york","state":"NY","population":
8405837,"land_area":302.6},
   {"city":"seattle","state":"WA","population":
652405,"land_area":83.9},
   {"city":"boston","state":"MA","population":
645966,"land_area":48.3},
   {"city":"kansas city","state":"MO","population":
467007,"land_area":315}]
```

Reducing

- A family of functions that takes a whole array and reduce it to a single value
- Eg: sum, average, median
- Sum

```
var landSum = data.reduce(function(sum, d) {
  return sum + d.land_area;
}, 0);
console.log(landSum);
=> 749.8
```

Reducing

- Function reduce() takes two parameters
  - A function to compute the aggregated value, that takes in input the value computed at the previous step of the iteration and the current value
  - An initial value of the aggregate. If this value is not specified, the initial value is set to the value of the first element and the iteration starts from the second entry.

Chaining

- The functional declaration of these transformation enable function chaining

```
var bigCities = data.filter(function(d) { return
d.population > 500000; })
  .sort(function(a,b) { return a.population -
b.population; })
  .map(function(d) { return d.city; });
console.log(bigCities);
=> ["boston", "seattle", "new york"]
```

Grouping

Example data

```
var expenses = [
  {"name":"jim","amount":34,"date":"11/12/2015"},
  {"name":"carl","amount":120.11,"date":"11/12/2015"},
  {"name":"jim","amount":45,"date":"12/01/2015"},
  {"name":"stacy","amount":12.00,"date":"01/04/2016"},
  {"name":"stacy","amount":34.10,"date":"01/04/2016"},
  {"name":"stacy","amount":44.80,"date":"01/05/2016"}
];
```

Slice data by values

- Group rows by value

```
var expensesByName = d3.nest()
  .key(function(d) { return d.name; })
  .entries(expenses);
=> expensesByName = [
  {"key":"jim","values":[
    {"name":"jim","amount":34,"date":"11/12/2015"},
    {"name":"jim","amount":45,"date":"12/01/2015"}
  ]},
  {"key":"carl","values":[
    {"name":"carl","amount":120.11,"date":"11/12/2015"}
  ]},
  {"key":"stacy","values":[
    {"name":"stacy","amount":12.00,"date":"01/04/2016"},
    {"name":"stacy","amount":34.10,"date":"01/04/2016"},
    {"name":"stacy","amount":44.80,"date":"01/05/2016"}
  ]}
];
```

Summarize data by values in each group

```
var expensesAvgAmount = d3.nest()
  .key(function(d) { return d.name; })
  .rollup(function(v) { return d3.mean(v,
function(d) { return d.amount; }); })
  .entries(expenses);
console.log(expensesAvgAmount);
=> [
  {"key":"jim","values":39.5},
  {"key":"carl","values":120.11},
  {"key":"stacy","values":30.3}
]
```

Crossfilter.js

http://square.github.io/crossfilter/

Crossfilter

- Crossfilter is a library for multidimensional filtering
- Two basic concepts:
  - Dimension: a property of the data to exploit to split items (i.e. a column in a relational table)
  - Groups: to aggregate rows by values in a dimension (i.e. like a groupby in SQL)

Example from VC 2008


```
[
{"EncounterDate":"2005-04-26","NumDeaths":0,"Passengers":
6,"RecordNotes":null,"RecordType":"Interdiction","USCG_Vessel":"Cunningham",
"VesselType":"Raft","year":2005,"Month":"2005-04","EncounterCoords":
[-80.14622349209523,24.53605142362535],"LaunchCoords":[null,null]},
{"EncounterDate":"2005-05-15","NumDeaths":0,"Passengers":
11,"RecordNotes":null,"RecordType":"Interdiction","USCG_Vessel":"Forthright"
,"VesselType":"Rustic","year":2005,"Month":"2005-05","EncounterCoords":
[-80.75496221688965,24.72483828554483],"LaunchCoords":
[-79.65932674368925,23.70743135623052]},
{"EncounterDate":"2005-02-25","NumDeaths":0,"Passengers":
6,"RecordNotes":null,"RecordType":"Interdiction","USCG_Vessel":"Pompano","Ve
sselType":"Raft","year":2005,"Month":"2005-02","EncounterCoords":
[-80.32020594311533,25.02156920297054],"LaunchCoords":[null,null]},
{"EncounterDate":"2005-04-13","NumDeaths":0,"Passengers":
6,"RecordNotes":null,"RecordType":"Interdiction","USCG_Vessel":"Tripoteur","
VesselType":"Raft","year":2005,"Month":"2005-04","EncounterCoords":
[-80.15149489716094,24.57412215015249],"LaunchCoords":
[-79.65999190070923,23.73619147168514]}
]
```

Basic statistics

```
var cf = crossfilter(migrants);
// how many report?
// select count(*) from migrants
console.log("num reports",
cf.groupAll().reduceCount().value());

// select sum(Passengers) from migrants
console.log("num passengers",
cf.groupAll().reduceSum(function(d){return
d.Passengers}).value())

// select sum(NumDeaths) from migrants
console.log("num deaths",
cf.groupAll().reduceSum(function(d){return
d.NumDeaths}).value())
```

Dimensions and Filtering

- Define a dimension by providing a function to select a value for each row

```
var dVesselType = cf.dimension(function(d){return d.VesselType});
// select count(*) from migrants where VesselType=="Rustic”
dVesselType.filter("Rustic");
console.log("num reports
(Rustic)",cf.groupAll().reduceCount().value());
// select sum(Passengers) from migrants where VesselType=="Rustic”
console.log("num passengers (Rustic)",
cf.groupAll().reduceSum(function(d){return d.Passengers}).value())
// select sum(NumDeaths) from migrants where VesselType=="Rustic”
console.log("num deaths (Rustic)",
cf.groupAll().reduceSum(function(d){return d.NumDeaths}).value())
// select VesselType, count(*) from migrants group by VesselType
var countVesselType = dVesselType.group().reduceCount();
console.log(countVesselType.all());
```