# Interaction

**Start here**

***Note:*** *consider treating gender and sex data with more nuance than in these examples [see here]*

# Interaction as an aesthetic mapping

- In some cases interaction allows us to simply add another “dimension”: **time**
  - Very limited in scope
  - Not always the right choice!
  - Example

# Best practices for interaction

*Based on slides from Cody Dunne & Miriah Meyer*

- **Benefits** of interaction
    - Enables visualization of large amounts of data
    - Amplifies user cognition (supports sensemaking)
    - Increases engagement (vis becomes personal to user)
    - Increases deep learning and learning transfer

# Best practices for interaction

*Based on slides from Cody Dunne & Miriah Meyer*

- **Drawbacks** of interaction
  - Requires human time and attention
  - Increase perceptual and exploration costs (van Wijk 2005)
  - Interaction costs (Lam 2008)
  - Multiple user studies find no increase in performance in specific situations (Ragan et al. 2012,

“Overview first, zoom and filter, and details on demand.”

- Ben Shneiderman

*“The Shneiderman Mantra”*

“Overview first, zoom and filter, and details on demand.”

- Ben Shneiderman

*“The Shneiderman Mantra”*

**Overview**—provide high-level view/summary

**Zoom and Filter**—enable data discovery and exploration, support search/tasks

**Details on Demand**—do not overwhelm the viewer. Provide extra information as needed

“Search, show context, expand on demand”

- van Ham & Perer

“Search, show context, expand on demand”
- van Ham & Perer

**Search**—pick subset of data to focus on.

**Show context**—show connected or relevant data for the user’s current interests.

**Expand on demand**—user chooses to expand the context in a direction of interest.

# Taxonomy of interactions

*Based on slides from Jeffery Heer*

- **Data and view specification**
  - Filter, query, derive
- **View manipulation**
  - Select, Navigate, coordinate, organize
- **Process and provenance**
  - Record annotate, share, guide

# Querying and filtering

- Determine a subset of data to highlight by applying *filters*
  - Example: Simple filtering
  - Example: ZIP codes
  - Example: baby names
  - Example: Gapminder DimpVis
  - Example: Segregation in U.S. Cities

# Pointing and selection

Select an observation for more details

- Example: tooltips!
- Example: Airports
- Example: College mobility

# Brushing and linking

- Select a subset of data using a brush
  - See the selected data in other views
  - Views must be *linked*
  - Example: Stocks and IMDB
  - Example: Brushable scatterplot
  - Example: Crossfiltering
  - Example: Parallel coordinates

# Zooming and panning

- Allow user to manipulate the scales
  - Example: maps!
  - Example: Vega-Altair

# Sorting

- Allow user to manipulate the scales
  - Example: Sortable bar plot

# Scrollytelling

- Update visualization as a reader progresses through a story
  - Example: California fires

# Prompting reflection

- Allow user to set their own expectations
  - Example: college mobility

# Implementing interaction

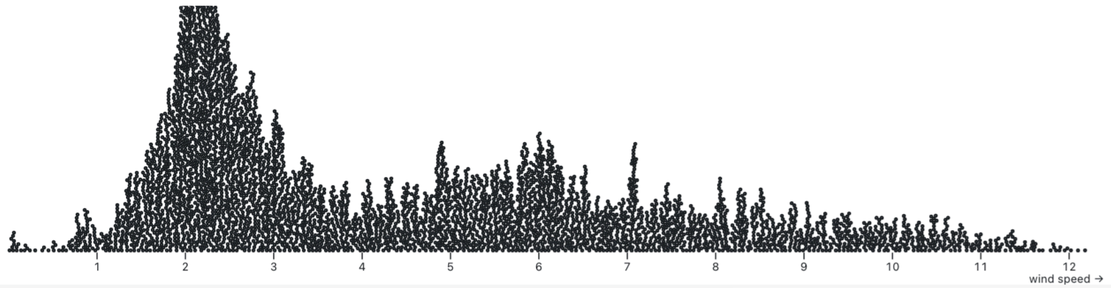
# How do we implement interaction?

- Bootstrapping applet

wind speed

```
<select name="dataset-select" id="dataset-select">
    <option value="wind speed">wind speed</option>
    <option value="IMDB rating">IMDB rating</option>
  </select>
```

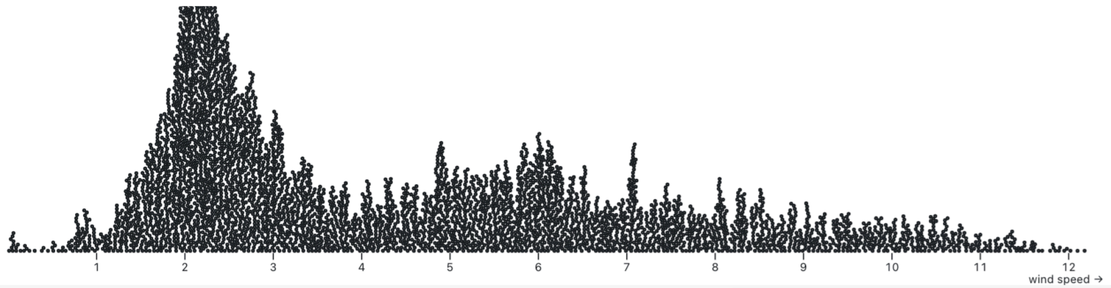
```
let select = document.getElementById('dataset-select');
select.addEventListener("change", (d) => updatePlot(d.target.value));
```




```
<figure id='population-figure'></figure>
```

```
let figure = document.getElementById('population-figure');
```

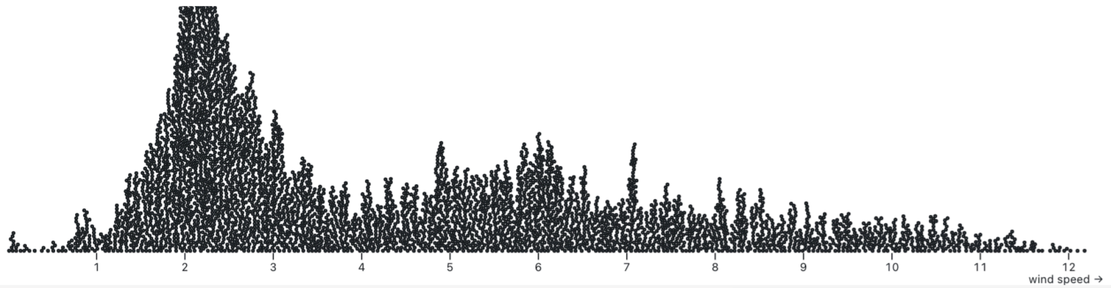
```
function updatePlot(dataset) {
  let populations = ({
    "wind speed": winddata.map(d => [d['speed']]),
    "IMDB rating": imdb.map(d => [d['IMDB Rating']]),
  })

  let newPlot = Plot.plot({
    height: 250,
    width: 1000,
    x: {label: dataset, domain: [Math.min(...populations[dataset]),
                                 Math.max(...populations[dataset])]},
    marks: [
      Plot.dot(populations[dataset], Plot.dodgeY({x: "0", r:1}))
    ]
  })
  figure.innerHTML = ""; // Clear the current plot
  figure.appendChild(newPlot); // Add in the new plot
}
updatePlot(select.value);
```




```
<figure id='population-figure'></figure>
```

```
let figure = document.getElementById('population-figure');
```

```
function updatePlot(dataset) {
  let populations = ({
    "wind speed": winddata.map(d => [d['speed']]),
    "IMDB rating": imdb.map(d => [d['IMDB Rating']]),
  })

  let newPlot = Plot.plot({
    height: 250,
    width: 1000,
    x: {label: dataset, domain: [Math.min(...populations[dataset]),
                                 Math.max(...populations[dataset])]},
    marks: [
      Plot.dot(populations[dataset], Plot.dodgeY({x: "0", r:1}))
    ]
  })
  figure.innerHTML = ""; // Clear the current plot
  figure.appendChild(newPlot); // Add in the new plot
}
updatePlot(select.value);
```




```
<figure id='population-figure'></figure>
```

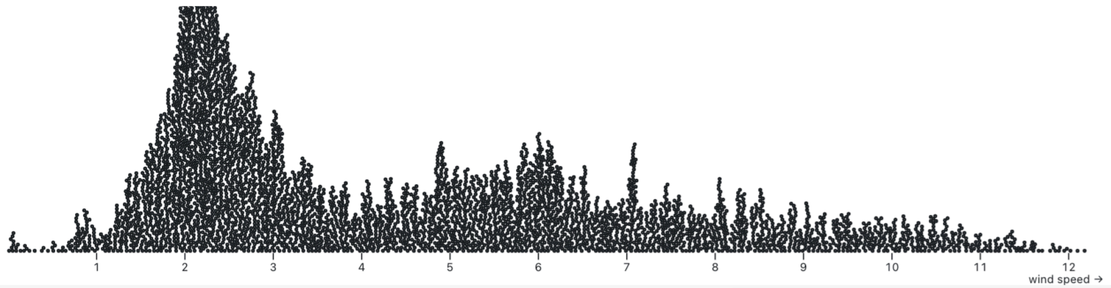
```
let figure = document.getElementById('population-figure');
```

wind speed

```
<select name="dataset-select" id="dataset-select">
    <option value="wind speed">wind speed</option>
    <option value="IMDB rating">IMDB rating</option>
  </select>
```

```
let select = document.getElementById('dataset-select');
```

```
function updatePlot(dataset) {
  let populations = ({
    "wind speed": winddata.map(d => [d['speed']]),
    "IMDB rating": imdb.map(d => [d['IMDB Rating']]),
  })

  let newPlot = Plot.plot({
    height: 250,
    width: 1000,
    x: {label: dataset, domain: [Math.min(...populations[dataset]),
                                 Math.max(...populations[dataset])]},
    marks: [
      Plot.dot(populations[dataset], Plot.dodgeY({x: "0", r:1}))
    ]
  })
  figure.innerHTML = ""; // Clear the current plot
  figure.appendChild(newPlot); // Add in the new plot
}
updatePlot(select.value);
```




```
<figure id='population-figure'></figure>
```

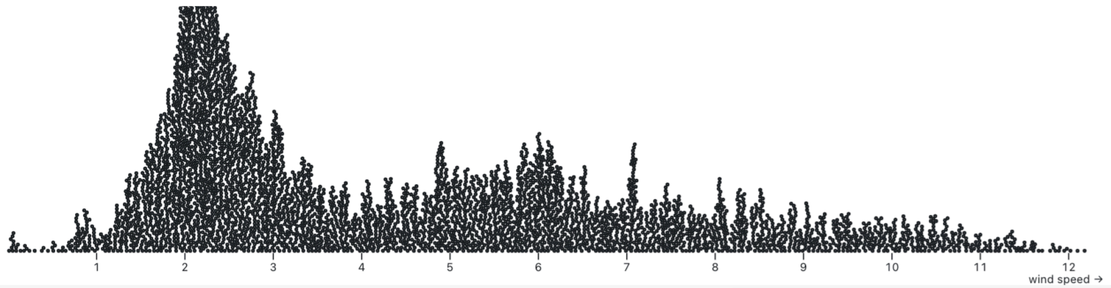
```
let figure = document.getElementById('population-figure');
```

wind speed

```
<select name="dataset-select" id="dataset-select">
    <option value="wind speed">wind speed</option>
    <option value="IMDB rating">IMDB rating</option>
  </select>
```

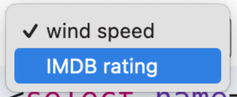
```
let select = document.getElementById('dataset-select');
select.addEventListener("change", (d) => updatePlot(d.target.value));
```

```
function updatePlot(dataset) {
  let populations = ({
    "wind speed": winddata.map(d => [d['speed']]),
    "IMDB rating": imdb.map(d => [d['IMDB Rating']]),
  })

  let newPlot = Plot.plot({
    height: 250,
    width: 1000,
    x: {label: dataset, domain: [Math.min(...populations[dataset]),
                                 Math.max(...populations[dataset])]},
    marks: [
      Plot.dot(populations[dataset], Plot.dodgeY({x: "0", r:1}))
    ]
  })
  figure.innerHTML = ""; // Clear the current plot
  figure.appendChild(newPlot); // Add in the new plot
}
updatePlot(select.value);
```




```
<figure id='population-figure'></figure>
```

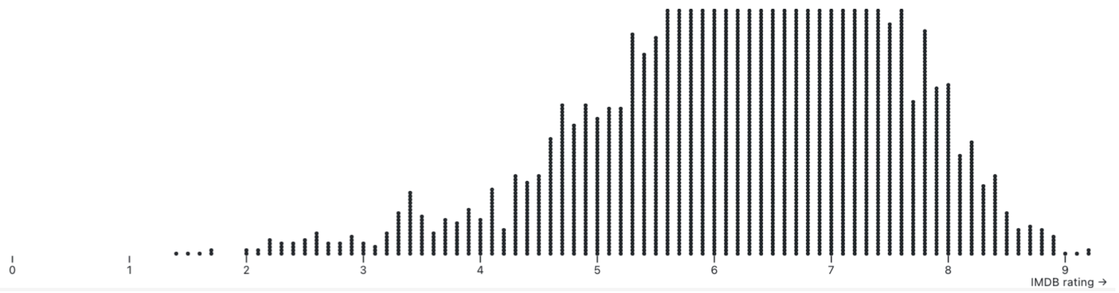
```
let figure = document.getElementById('population-figure');
```




```
<select name="dataset-select" id="dataset-select">
    <option value="wind speed">wind speed</option>
    <option value="IMDB rating">IMDB rating</option>
  </select>
```

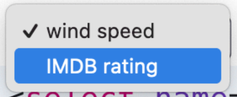
```
let select = document.getElementById('dataset-select');
select.addEventListener("change", (d) => updatePlot(d.target.value));
```

```
function updatePlot(dataset) {
  let populations = ({
    "wind speed": winddata.map(d => [d['speed']]),
    "IMDB rating": imdb.map(d => [d['IMDB Rating']]),
  })

  let newPlot = Plot.plot({
    height: 250,
    width: 1000,
    x: {label: dataset, domain: [Math.min(...populations[dataset]),
                                 Math.max(...populations[dataset])]},
    marks: [
      Plot.dot(populations[dataset], Plot.dodgeY({x: "0", r:1}))
    ]
  })
  figure.innerHTML = ""; // Clear the current plot
  figure.appendChild(newPlot); // Add in the new plot
}
updatePlot(select.value);
```




```
<figure id='population-figure'></figure>
```

```
let figure = document.getElementById('population-figure');
```

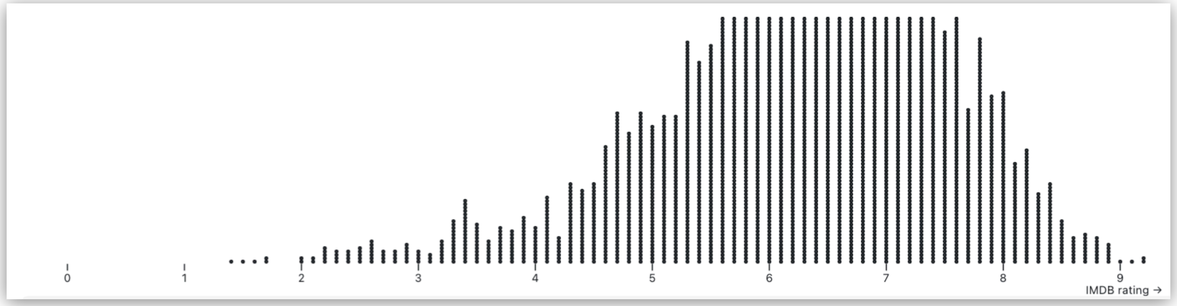



```
<select name="dataset-select" id="dataset-select">
    <option value="wind speed">wind speed</option>
    <option value="IMDB rating">IMDB rating</option>
  </select>
```

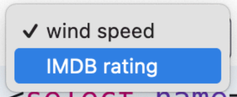
```
let select = document.getElementById('dataset-select');
```

```
select.addEventListener("change", (d) => updatePlot(d.target.value));
```

```
function updatePlot(dataset) {
  let populations = ({
    "wind speed": winddata.map(d => [d['speed']]),
    "IMDB rating": imdb.map(d => [d['IMDB Rating']]),
  })

  let newPlot = Plot.plot({
    height: 250,
    width: 1000,
    x: {label: dataset, domain: [Math.min(...populations[dataset]),
                                 Math.max(...populations[dataset])]},
    marks: [
      Plot.dot(populations[dataset], Plot.dodgeY({x: "0", r:1}))
    ]
  })
  figure.innerHTML = ""; // Clear the current plot
  figure.appendChild(newPlot); // Add in the new plot
}
updatePlot(select.value);
```

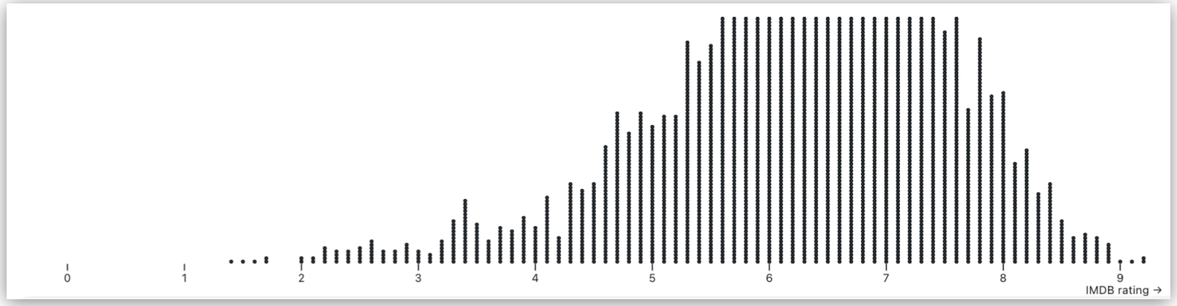



```
function updatePlot(dataset) {
  let populations = ({
    "wind speed": winddata.map(d => [d['speed']]),
    "IMDB rating": imdb.map(d => [d['IMDB Rating']]),
  })

  let newPlot = Plot.plot({
    height: 250,
    width: 1000,
    x: {label: dataset, domain: [Math.min(...populations[dataset]),
                                 Math.max(...populations[dataset])]},
    marks: [
      Plot.dot(populations[dataset], Plot.dodgeY({x: "0", r:1}))
    ]
  })
  figure.innerHTML = ""; // Clear the current plot
  figure.appendChild(newPlot); // Add in the new plot
}
updatePlot(select.value);
```



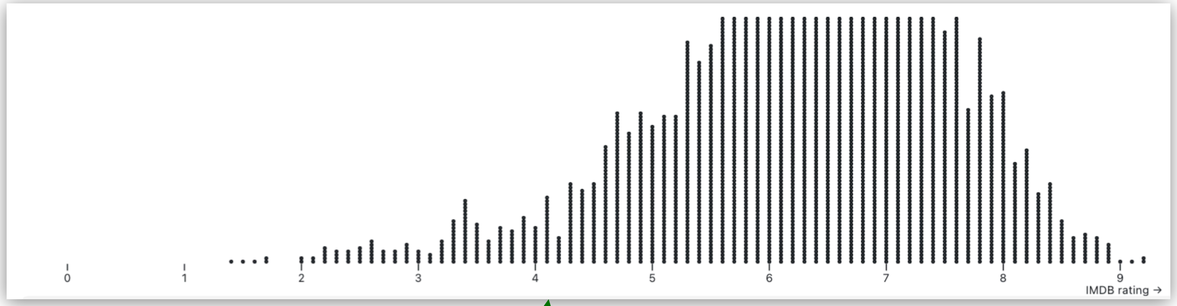

```
<select name="dataset-select" id="dataset-select">
    <option value="wind speed">wind speed</option>
    <option value="IMDB rating">IMDB rating</option>
  </select>
```

```
let select = document.getElementById('dataset-select');
select.addEventListener("change", (d) => updatePlot(d.target.value));
```



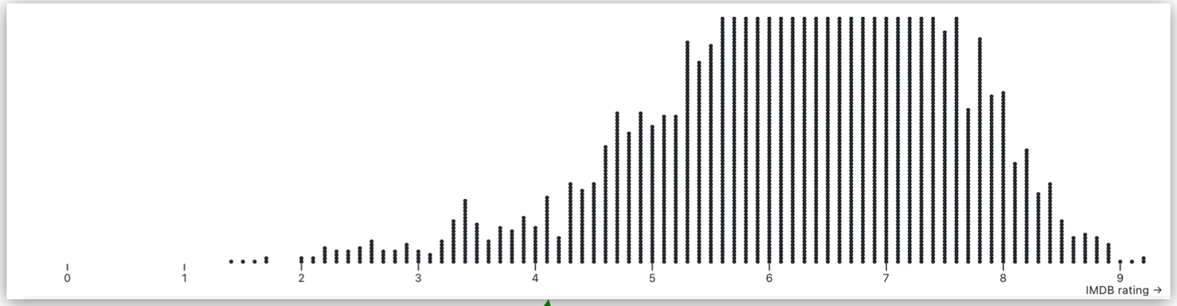

```
function updatePlot(dataset) {
  let populations = ({
    "wind speed": winddata.map(d => [d['speed']]),
    "IMDB rating": imdb.map(d => [d['IMDB Rating']]),
  })

  let newPlot = Plot.plot({
    height: 250,
    width: 1000,
    x: {label: dataset, domain: [Math.min(...populations[dataset]),
                                 Math.max(...populations[dataset])]},
    marks: [
      Plot.dot(populations[dataset], Plot.dodgeY({x: "0", r:1}))
    ]
  })
  figure.innerHTML = ""; // Clear the current plot
  figure.appendChild(newPlot); // Add in the new plot
}
updatePlot(select.value);
```

IMDB rating



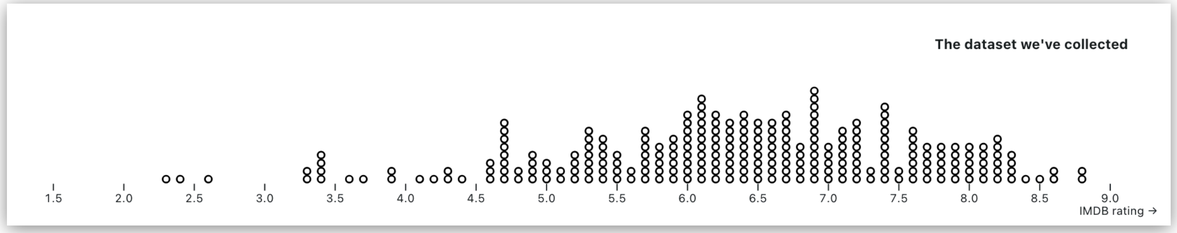

Updates

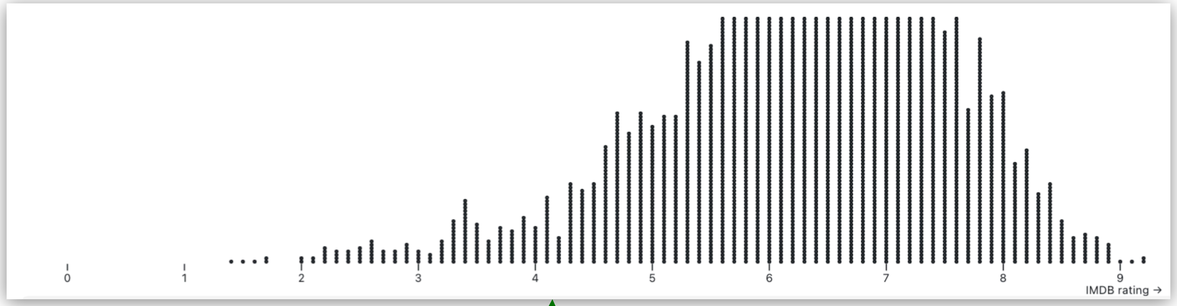
IMDB rating

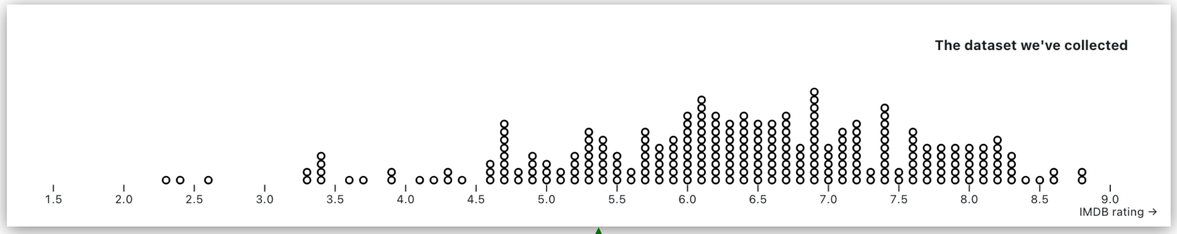



Updates

IMDB rating



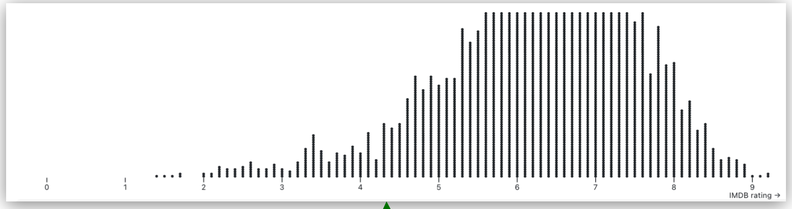


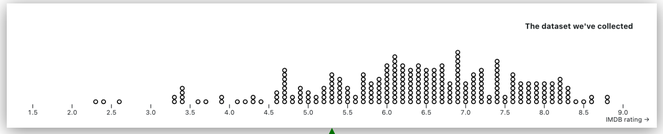



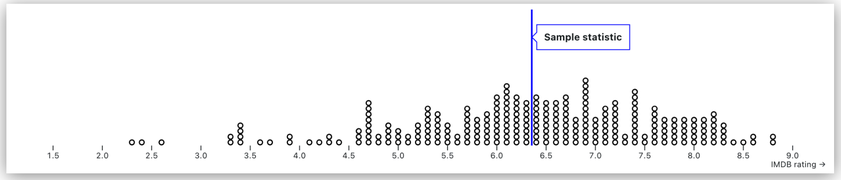

Population

Updates

IMDB rating



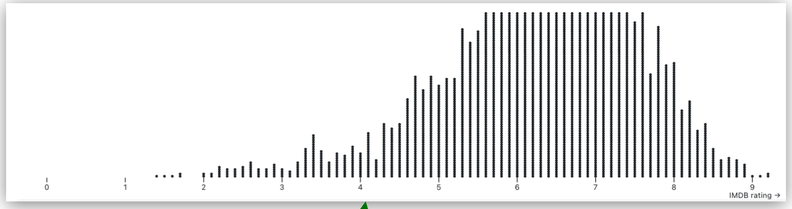



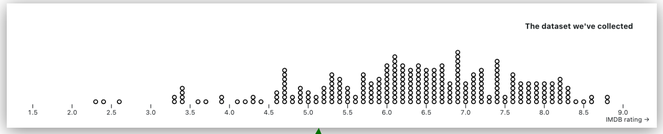



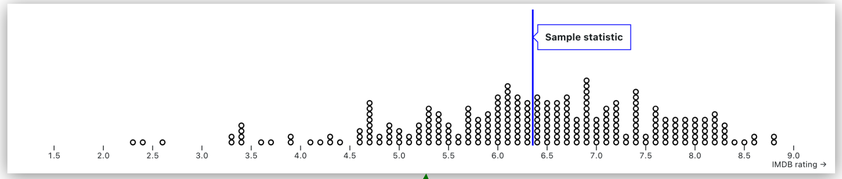

Population

Updates

IMDB rating



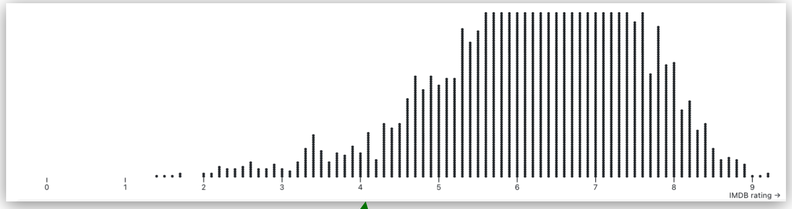



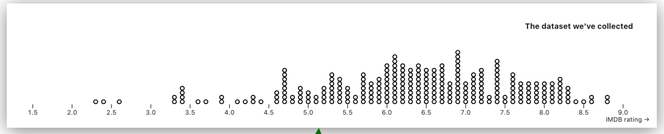

Sample

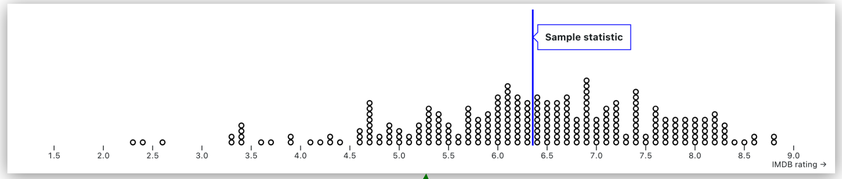
Population

Updates

IMDB rating



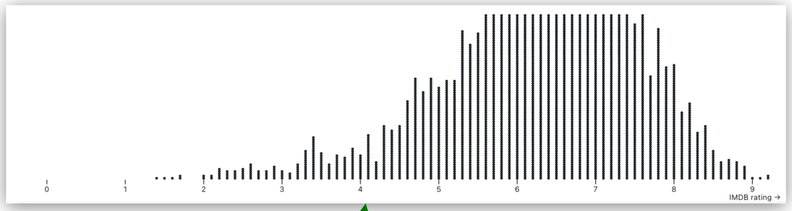

Sample

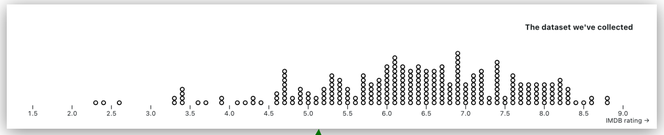
Population

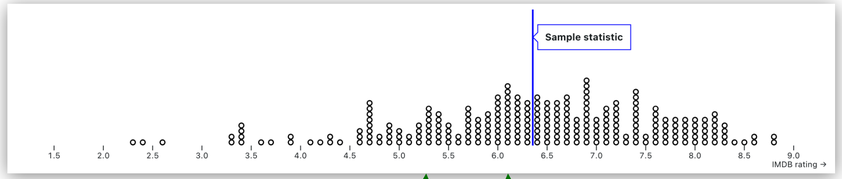
Updates

IMDB rating

Mean



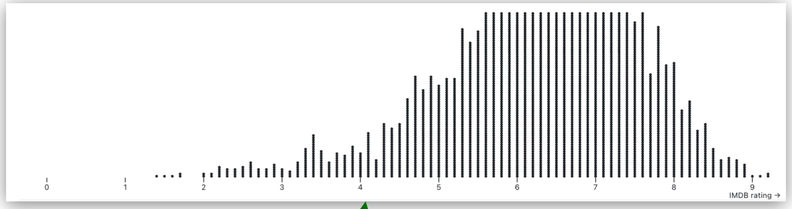

Sample

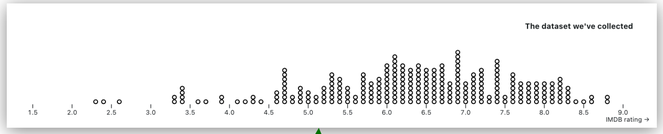
Population

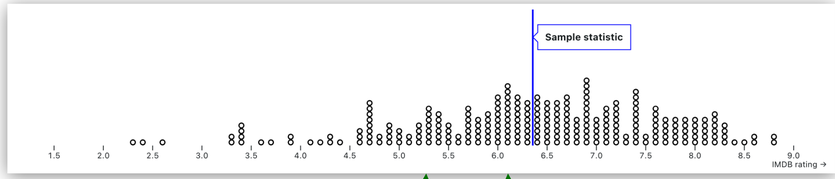
Updates

IMDB rating

Mean




Sample

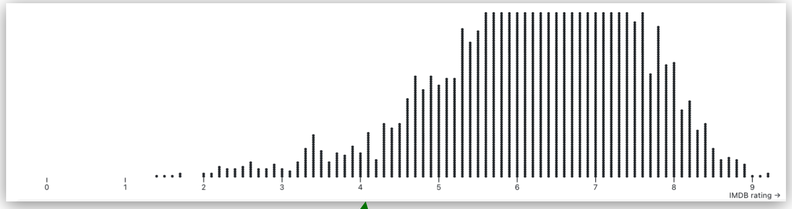
Population

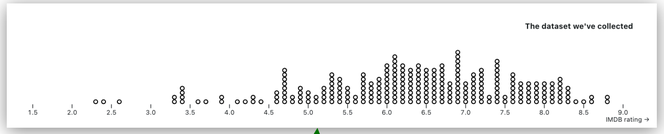
Updates

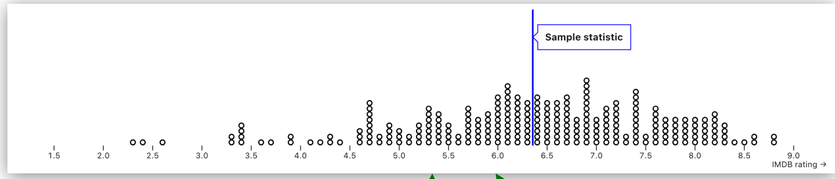
*Here our **Mean** of `IMDB rating` is 6.35*

IMDB rating

Mean




Sample

Statistic

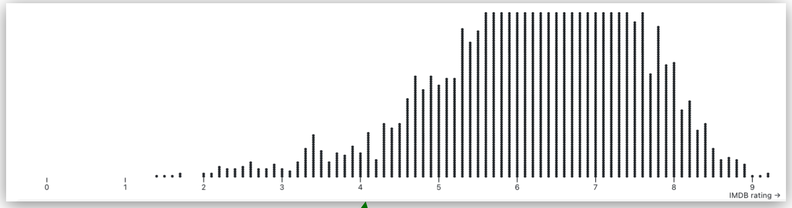
Population

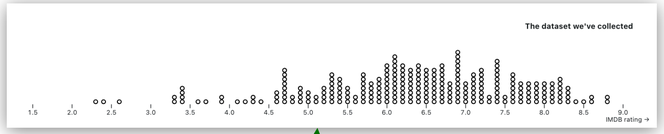
Updates

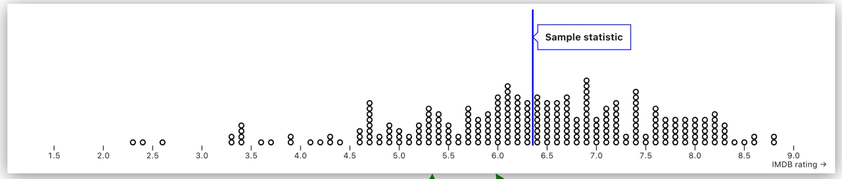
*Here our **Mean** of `IMDB rating` is 6.35*

IMDB rating

Mean




Here we needed to:

- Add *statistic* to our state
- Change *sample* code to update the *statistic*
- Change *statistic type selector* code to update text
- Change *statistic* code to update text
- ...

Sample

Statistic

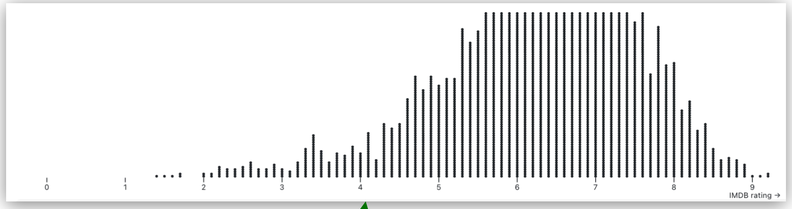
Population

Updates

*Here our **Mean** of `IMDB rating` is 6.35*

IMDB rating

Mean






Adding new components gets very complicated very quickly

Here we needed to:

- Add *statistic* to our state
- Change *sample* code to update the *statistic*
- Change *statistic type selector* code to update text
- Change *statistic* code to update text
- ...

Sample

Statistic

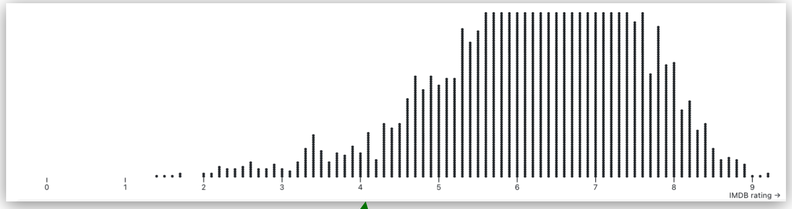
Population

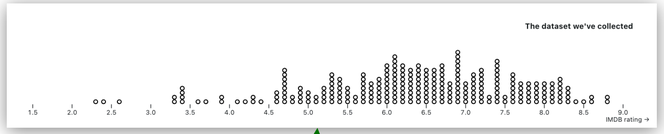
Updates

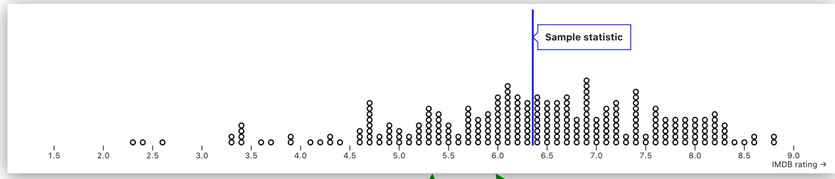
*Here our **Mean** of `IMDB rating` is 6.35*

IMDB rating

Mean




Sample

Statistic

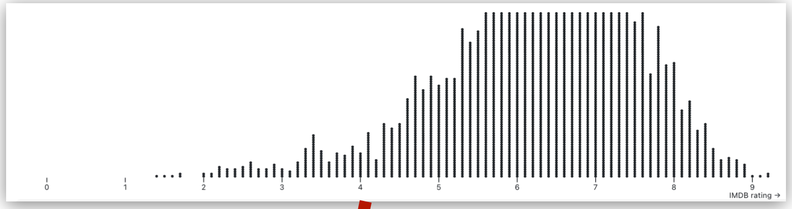
Population

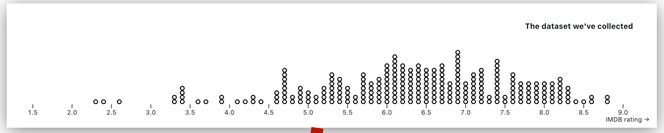
Updates

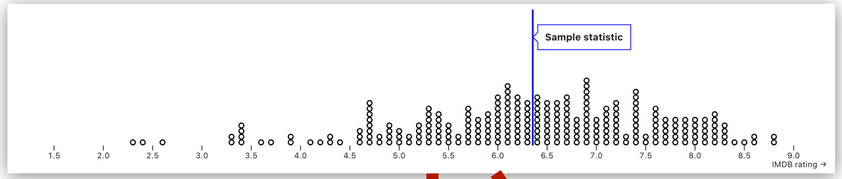
*Here our **Mean** of `IMDB rating` is 6.35*

IMDB rating

Mean




Sample

Statistic

Population

Observes

*Here our **Mean** of `IMDB rating` is 6.35*

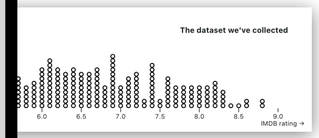
IMDB rating

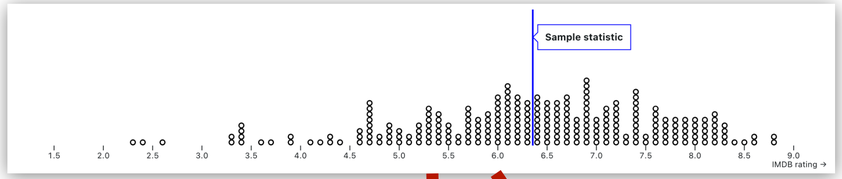
Mean

# Observer pattern: *flip responsibility*

Each *component* is responsible for knowing what other pieces of state it depends on

- When parent state changes - *recompute*!




Sample

Statistic

Population

Observes

*Here our **Mean** of `IMDB rating` is 6.35*

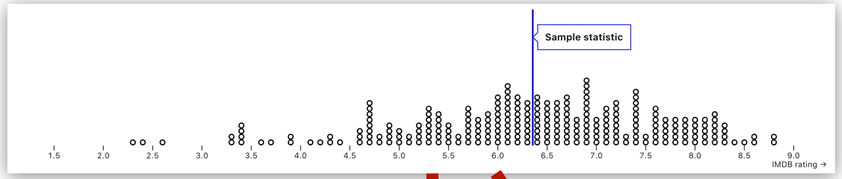
IMDB rating

Mean

# Observer pattern: *implementation*

Each component *registers* itself as an observer of its parent

- When parent state changes - *signal all observers*!


Python

```python
class Observable:
    def __init__(self):
        self._observers = []

    def register_observer(self, observer) -> None:
        self._observers.append(observer)

    def notify_observers(self, *args, **kwargs) -> None:
        for observer in self._observers:
            observer.notify(self, *args, **kwargs)


class Observer:
    def __init__(self, observable):
        observable.register_observer(self)

    def notify(self, observable, *args, **kwargs) -> None:
        print("Got", args, kwargs, "From", observable)


subject = Observable()
observer = Observer(subject)
subject.notify_observers("test", kw="python")

# prints: Got ('test',) {'kw': 'python'} From <__main__.Observable object at
0x0000019757826FD0>
```

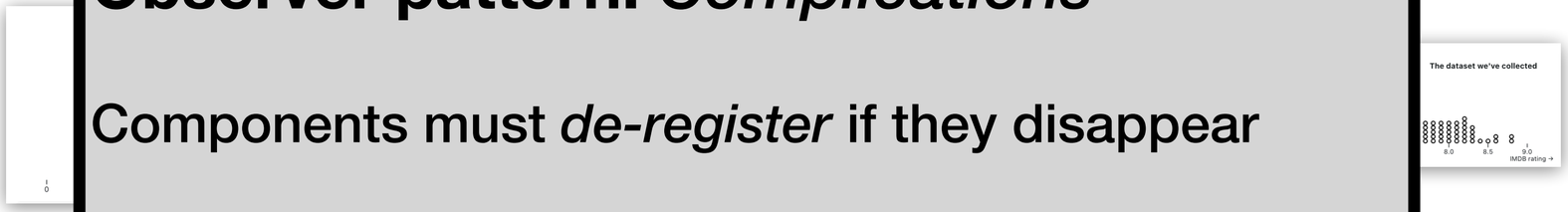
Javascript

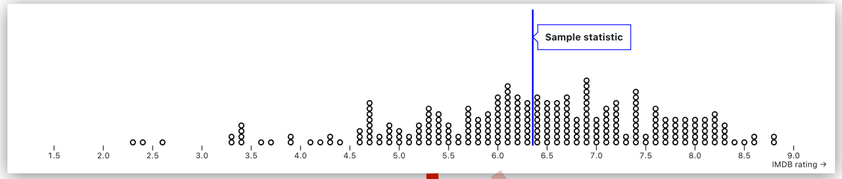
```javascript
let Subject = {
    _state: 0,
    _observers: [],
    add: function(observer) {
        this._observers.push(observer);
    },
    getState: function() {
        return this._state;
    },
    setState: function(value) {
        this._state = value;
        for (let i = 0; i < this._observers.length; i++)
        {
            this._observers[i].signal(this);
        }
    }
};

let Observer = {
    signal: function(subject) {
        let currentValue = subject.getState();
        console.log(currentValue);
    }
}

Subject.add(Observer);
Subject.setState(10);
//Output in console.log - 10
```

*From Wikipedia*

# Observer pattern: *Complications*

Components must *de-register* if they disappear




Sample

Population

Observes

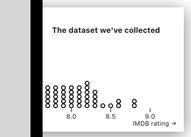
*Here our **Mean** of `IMDB rating` is 6.35*

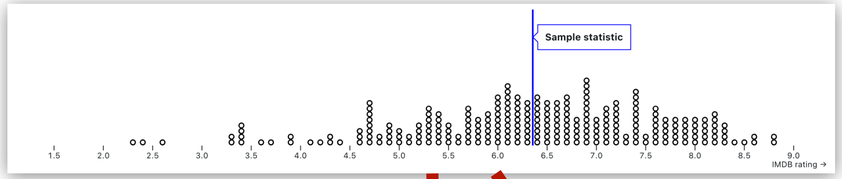
IMDB rating

Mean

# Observer pattern: *Complications*

Cycles cannot be allowed!




Sample

Statistic

Population

Observes

*Here our **Mean** of `IMDB rating` is 6.35*

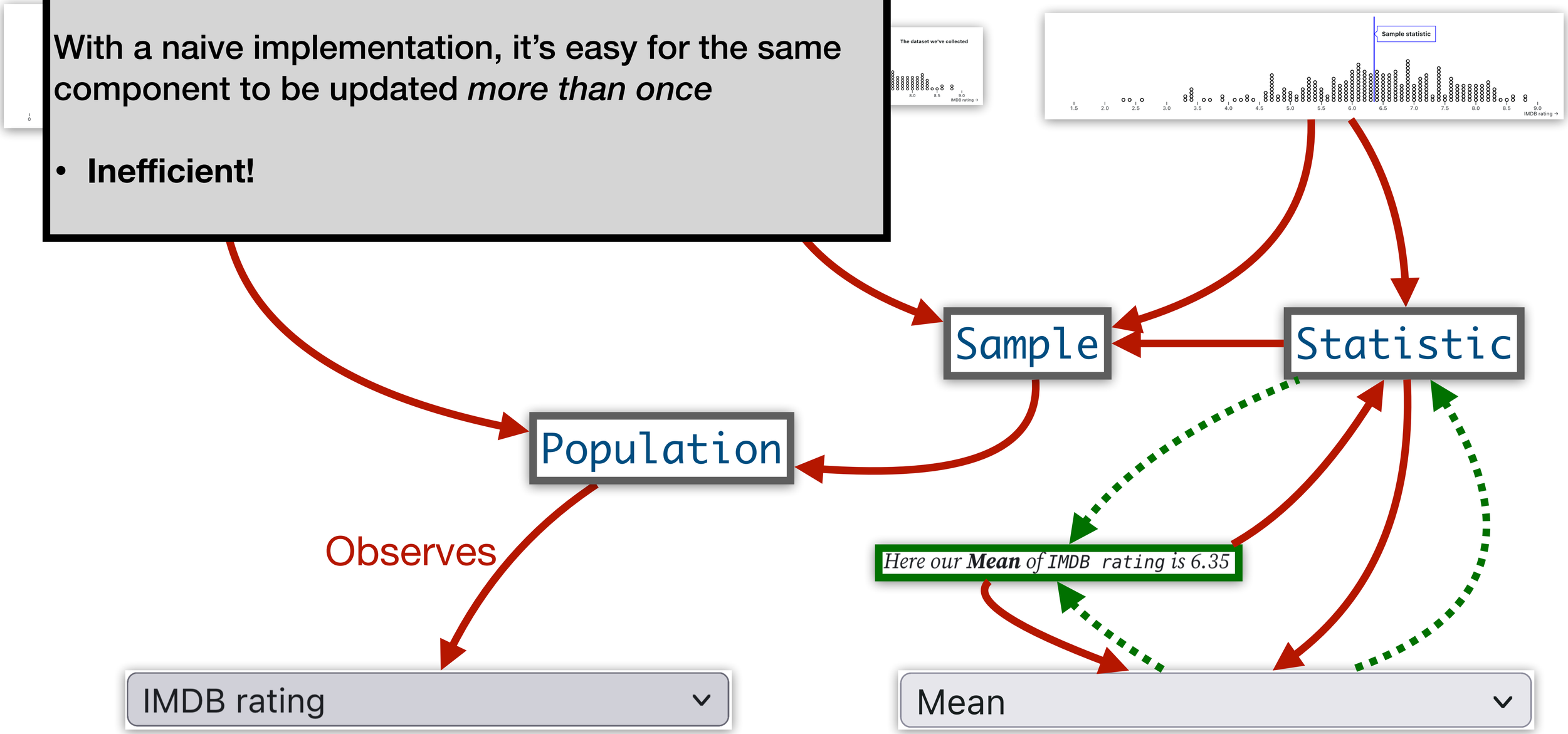
IMDB rating

Mean

# Observer pattern: *Complications*

With a naive implementation, it's easy for the same component to be updated *more than once*

- **Inefficient!**



# Observer pattern: *Complications*

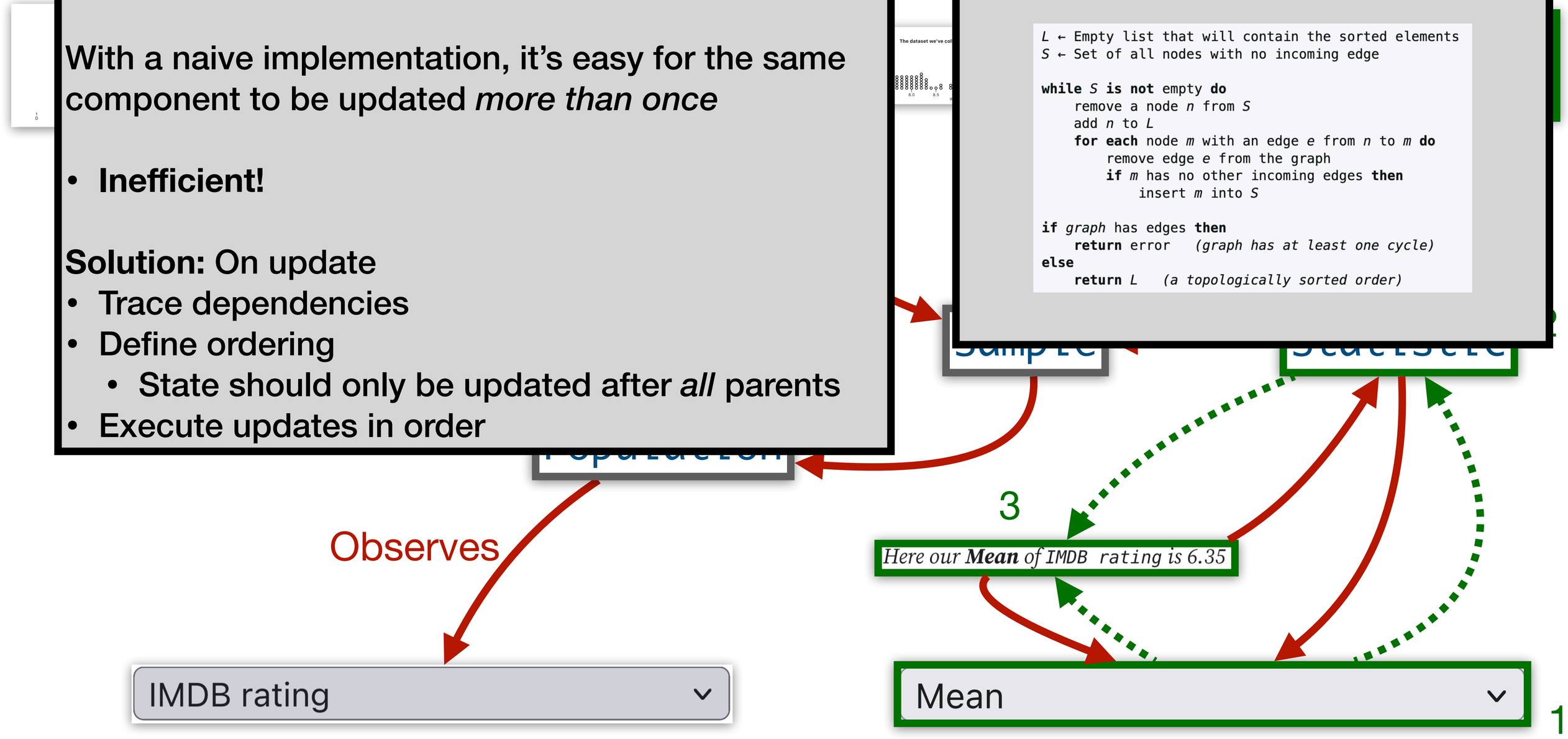
With a naive implementation, it's easy for the same component to be updated *more than once*

- **Inefficient!**

**Solution:** On update
- Trace dependencies
- Define ordering
  - State should only be updated after *all* parents
- Execute updates in order

## Define ordering: *topological sort*

```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edge

while S is not empty do
    remove a node n from S
    add n to L
    for each node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S

if graph has edges then
    return error   (graph has at least one cycle)
else
    return L   (a topologically sorted order)
```



# Reactive programming

Build observer pattern into core language or framework



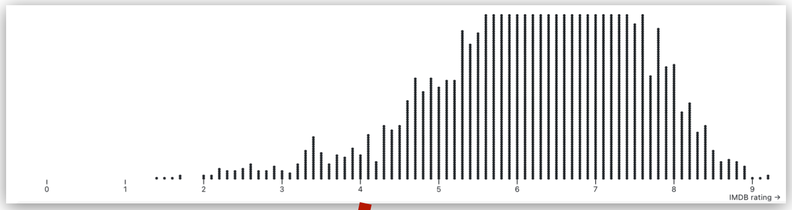


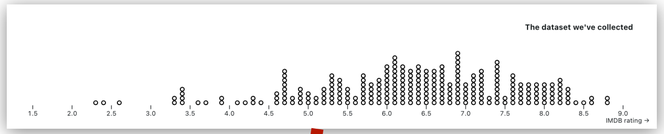

# Reactive programming

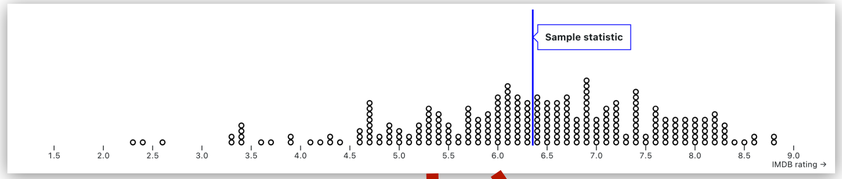
Build observer pattern into core language or framework






```
sample = resample(population)
```

```
stat = computeStat(sample)
```

Sample

Statistic

```
population = (dataset == "wind speed") ? windpopulation : imdbpopulation;
```

Population

```
*Here our **${statistic}** of `${dataset}` is ${estimate}*
```

*Here our **Mean** of IMDB rating is 6.35*

Observes

IMDB rating

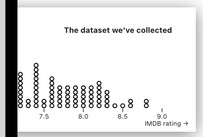
Mean

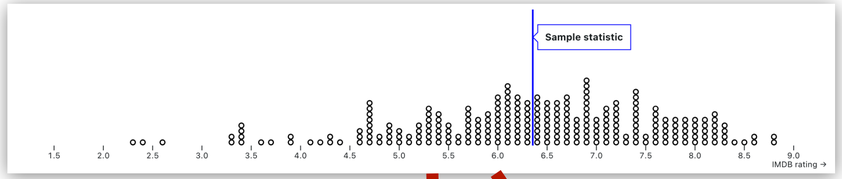
```
viewof dataset = Inputs.select(["wind speed", "IMDB rating"], {label: "Dataset"});
```

```
viewof statistic = Inputs.select(["Mean", "Median"])
```

# Reactive framework

Each line re-runs when variables it uses change!




```
sample = resample(population)
```

```
stat = computeStat(sample)
```

Sample

Statistic

```
population = (dataset == "wind speed") ? windpopulation : imdbpopulation;
```

Population

Observes

```
*Here our **${statistic}** of `${dataset}` is ${estimate}*
```

*Here our **Mean** of IMDB rating is 6.35*

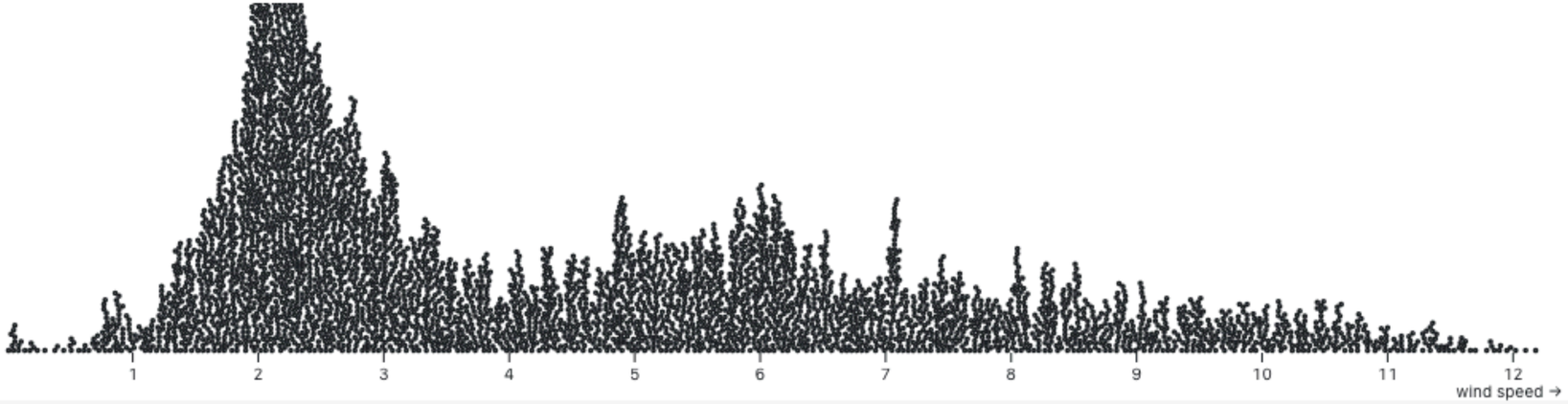
IMDB rating

Mean

```
viewof dataset = Inputs.select(["wind speed", "IMDB rating"], {label: "Dataset"});
```

```
viewof statistic = Inputs.select(["Mean", "Median"])
```

# Dataset choice




```
Plot.plot({
  height: 250,
  width: 1000,
  x: {label: dataset, domain: [Math.min(...populations[dataset]),
                               Math.max(...populations[dataset])]},
  marks: [
    Plot.dot(populations[dataset], Plot.dodgeY({x: "0", r:1}))
  ]
})
```

Updated on change!

wind speed

```
viewof dataset = Inputs.select(["wind speed", "IMDB rating"])
```

```
populations = ▸ Object {wind speed: Array(4800), IMDB rating: Array(3201)}
```

```
 populations = ({
      "wind speed": winddata.map(d => [d['speed']]),
      "IMDB rating": imdb.map(d => [d['IMDB Rating']]),
    })
```

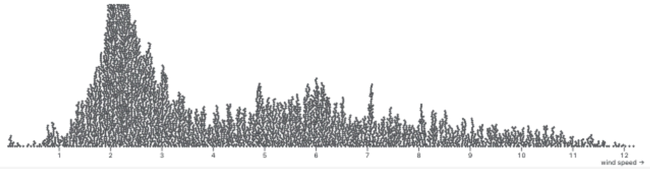
```
▸ Array(3201) [6.1, 6.9, 6.8, null, 3.4, null, 7.7, 3.8, 5.8, 7, 7, 7.5, 8.4, null, 6.8, null, 7, 6.1, 2.5, 8.9, …]
```

```
imdb.map(d => d['IMDB Rating'])
```

```
import {windpopulation, imdbpopulation, imdb, wind, winddata, tf, windspeedDistributionPlot, imdbDistributionPlot} from
"3523e4b3244dbb93"
```

# Traditional

## Dataset choice




```
<figure id='population-figure'></figure>
```

wind speed

```
<select name="dataset-select" id="dataset-select">
    <option value="wind speed">wind speed</option>
    <option value="IMDB rating">IMDB rating</option>
  </select>
```

```
undefined
```

```
{
  let figure = document.getElementById('population-figure');

  let select = document.getElementById('dataset-select');

  function updatePlot(dataset) {
    let populations = ({
      "wind speed": winddata.map(d => [d['speed']]),
      "IMDB rating": imdb.map(d => [d['IMDB Rating']]),
    })

    let newPlot = Plot.plot({
      height: 250,
      width: 1000,
      x: {label: dataset, domain: [Math.min(...populations[dataset]),
                                   Math.max(...populations[dataset])]},
      marks: [
        Plot.dot(populations[dataset], Plot.dodgeY({x: "0", r:1}))
      ]
    })
    figure.innerHTML = ""; // Clear the current plot
    figure.appendChild(newPlot); // Add in the new plot
  }
  updatePlot(select.value);

  select.addEventListener("change", (d) => updatePlot(d.target.value));
}
```

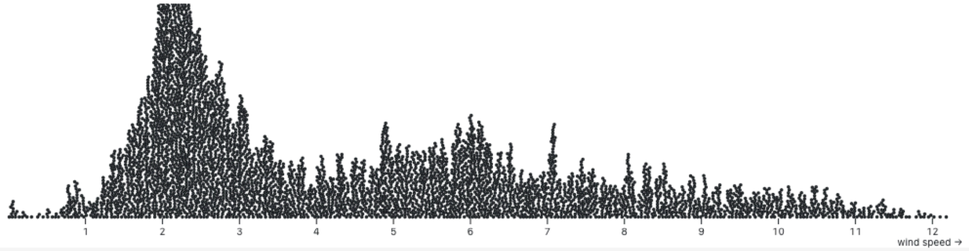
```
▸ Array(3201) [6.1, 6.9, 6.8, null, 3.4, null, 7.7, 3.8, 5.8, 7, 7, 7.5, 8.4, null, 6.8, null, 7, 6.1, 2.5, 8.9, …]
```

```
imdb.map(d => d['IMDB Rating'])
```

```
import {windpopulation, imdbpopulation, imdb, wind, winddata, tf, windspeedDistributionPlot, imdbDistributionPlot} from
"3523e4b3244dbb93"
```

# Reactive

## Dataset choice




```
Plot.plot({
  height: 250,
  width: 1000,
  x: {label: dataset, domain: [Math.min(...populations[dataset]),
                               Math.max(...populations[dataset])]},
  marks: [
    Plot.dot(populations[dataset], Plot.dodgeY({x: "0", r:1}))
  ]
})
```

wind speed

```
viewof dataset = Inputs.select(["wind speed", "IMDB rating"])
```

```
populations = ▸ Object {wind speed: Array(4800), IMDB rating: Array(3201)}
```

```
 populations = ({
      "wind speed": winddata.map(d => [d['speed']]),
      "IMDB rating": imdb.map(d => [d['IMDB Rating']]),
    })
```

```
▸ Array(3201) [6.1, 6.9, 6.8, null, 3.4, null, 7.7, 3.8, 5.8, 7, 7, 7.5, 8.4, null, 6.8, null, 7, 6.1, 2.5, 8.9, …]
```

```
imdb.map(d => d['IMDB Rating'])
```

```
import {windpopulation, imdbpopulation, imdb, wind, winddata, tf, windspeedDistributionPlot, imdbDistributionPlot} from
"3523e4b3244dbb93"
```

# Each *top-level* variable is a reactive value

- Can be computed with a *chunk* of code

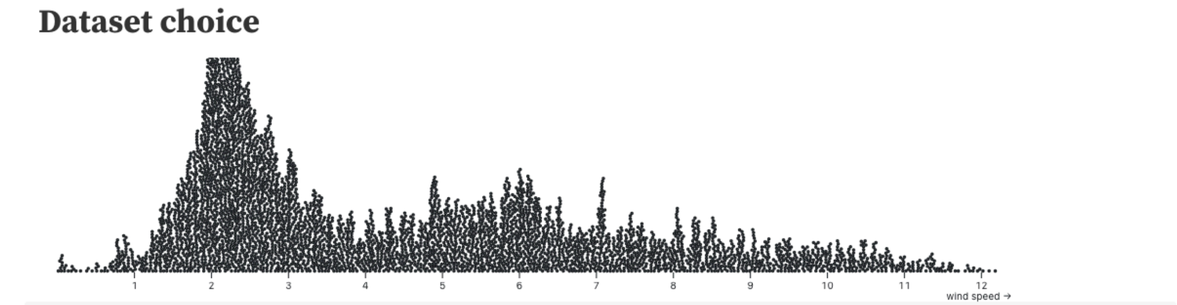
Reactive value

Dependencies

Code chunk

```
resampled = {
  resampleToggle2
  step2
  let x = resample(weights);
  let est = computeStat(x);
  estimates.push([est]);
  let output = x.arraySync();
  x.dispose()
  return output;
}
```

## Reactive




```
Plot.plot({
  height: 250,
  width: 1000,
  x: {label: dataset, domain: [Math.min(...populations[dataset]),
                               Math.max(...populations[dataset])]},
  marks: [
    Plot.dot(populations[dataset], Plot.dodgeY({x: "0", r:1}))
  ]
})
```

wind speed

```
viewof dataset = Inputs.select(["wind speed", "IMDB rating"])
```

```
populations = ▸ Object {wind speed: Array(4800), IMDB rating: Array(3201)}
```

```
 populations = ({
      "wind speed": winddata.map(d => [d['speed']]),
      "IMDB rating": imdb.map(d => [d['IMDB Rating']]),
    })
```

```
▸ Array(3201) [6.1, 6.9, 6.8, null, 3.4, null, 7.7, 3.8, 5.8, 7, 7, 7.5, 8.4, null, 6.8, null, 7, 6.1, 2.5, 8.9, …]
```

```
imdb.map(d => d['IMDB Rating'])
```

```
import {windpopulation, imdbpopulation, imdb, wind, winddata, tf, windspeedDistributionPlot, imdbDistributionPlot} from
"3523e4b3244dbb93"
```

# Your turn!