VISUALISATIONS SIMPLES AVEC D3.JS

ATELIER THÉMATIQUE EN VISUALISATION DE DONNÉES

Antoine Béland

10 octobre 2018

PLAN DE L'APRÈS-MIDI

1. Chargement de données externes
2. Traitement des données
3. Échelles avec D3.js
4. Axes avec D3.js
5. Groupes et chemins en SVG
6. Mise en pratique

CHARGEMENT DE DONNÉES EXTERNES

CHARGEMENT DE DONNÉES EXTERNES

- Il est généralement nécessaire de charger des données à partir d'un fichier externe
- Les données à utiliser sont souvent volumineuses et peuvent provenir de sources diverses
- On utilise habituellement le format CSV ou JSON

DONNÉES EXTERNES — CSV

```
Sexe, Prénom, Année de naissance
M, Alphonse, 1932
F, Béatrice, 1964
F, Charlotte, 1988
```

Sexe	Prénom	Année de naissance
M	Alphonse	1932
F	Béatrice	1964
F	Charlotte	1988

Exemple tiré de *Comma-separated values* · Wikipedia, 2018

DONNÉES EXTERNES — JSON

```
[
  { "Sexe": "M", "Prénom": "Alphonse", "Année de naissance": 1932 },
  { "Sexe": "F", "Prénom": "Béatrice", "Année de naissance": 1964 },
  { "Sexe": "F", "Prénom": "Charlotte", "Année de naissance": 1988 }
]
```

Sexe	Prénom	Année de naissance
M	Alphonse	1932
F	Béatrice	1964
F	Charlotte	1988

DONNÉES EXTERNES — D3.JS

- Avec D3.js, il faut utiliser les fonctions **`d3.csv`** ou **`d3.json`** pour charger des données externes
- Ces fonctions chargent les fichiers de façon **asynchrone** et transforment les données récupérées en objet JavaScript

DONNÉES EXTERNES — EXEMPLE

```
d3.csv('https://example.org/fichier.csv').then(data => {
  d3.select('body')
    .append('ul')
    .selectAll('li')
    .data(data)
    .enter()
    .append('li')
    .html(d => `<strong>${d.parti}</strong> ` +
      `(${d.femmes} femmes, ${d.hommes} hommes)`);
});
```

Données Démo

TRAITEMENT DES DONNÉES

TRAITEMENT DES DONNÉES

- Une fois les données chargées, il est souvent nécessaire d'effectuer certaines manipulations
- Il est essentiel de convertir les nombres en type `number` lorsqu'un fichier CSV est chargé
- Il peut être important de **filtrer** ou de **réorganiser** certaines données

TRAITEMENT DES DONNÉES — CSV

- Lorsqu'un fichier CSV est chargé, tous les éléments sont considérés comme des **`string`**
- Il est **obligatoire** de convertir les nombres en type **`number`** avec les fonctions **`parseInt`** / **`parseFloat`**

```
// Exemple de conversion avec les données de l'exemple précédent
data.forEach(d => {
  d.hommes = parseInt(d.hommes);
  d.femmes = parseInt(d.femmes);
});
```

FILTRAGE/RÉORGANISATION DES DONNÉES

- Il peut être nécessaire de **filtrer** les données récupérées pour conserver les pertinentes
- Il peut être également utile de **renommer** ou de **supprimer** certains champs des données
- On peut utiliser les fonctions `filter` ou `map` pour effectuer ces opérations

TRAITEMENT DES DONNÉES — EXEMPLE

```
d3.csv('https://example.org/fichier.csv').then(data => {

  // Conversion des nombres en type "number"
  data.forEach(d => {
    d.hommes = parseInt(d.hommes);
    d.femmes = parseInt(d.femmes);
  })

  // Filtrage des données
  data = data.filter(d => d.hommes > 0 || d.femmes > 0);

  // Affichage des données...
});
```

Données ▶ Démo

TRAITEMENT — POUR EN SAVOIR PLUS

- Fonctions disponibles sur les tableaux en JavaScript
- Guide sur la manipulation de tableaux en JavaScript

D3.JS — ÉCHELLES (*SCALES*)

ÉCHELLES

- Concept central dans la bibliothèque
- Permet de transformer les données récupérées pour qu'elles soient affichables dans le repère SVG

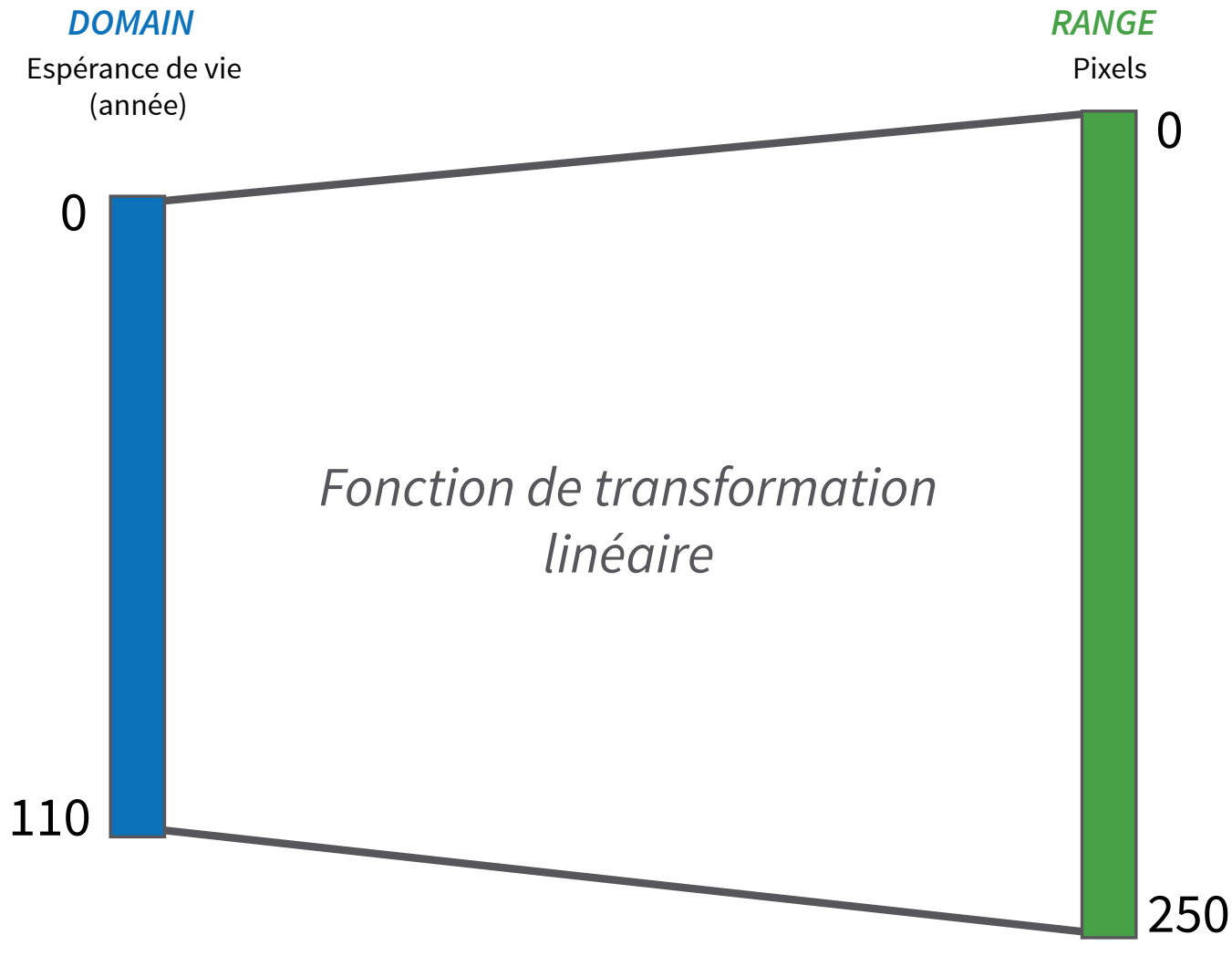
ÉCHELLES — MISE EN CONTEXTE

- Nous souhaitons réaliser un *bar chart* avec des données sur l'espérance de vie des pays
- Les données sur l'espérance de vie peuvent s'étendre entre **0** et **110** ans
- La taille maximale d'une barre doit être de **250 px**

Que pouvons-nous faire pour résoudre ce problème?

Utiliser une échelle linéaire!

ÉCHELLES — MISE EN CONTEXTE



ÉCHELLES — EXEMPLE

```
const scale = d3.scaleLinear()
  .domain([0, 110])
  .range([0, 250]);

console.log(scale(0));   // Sortie: 0
console.log(scale(50));  // Sortie: 113.6
console.log(scale(110)); // Sortie: 250
```

▶ Démo

ÉCHELLES CONTINUES

- Permet d'associer un intervalle continu (*domain*) à une autre plage continue (*range*)

ÉCHELLES CONTINUES — TYPES

- `d3.scaleLinear` (échelle linéaire)
- `d3.scalePow` (échelle exponentielle)
- `d3.scaleSqrt` (échelle racine carrée)
- `d3.scaleTime` (échelle linéaire pour des dates)
- etc.

ÉCHELLES CONTINUES — EXEMPLE (1)

```
// Domaine entre le 1er janvier et le 31 décembre 2018
const scale = d3.scaleTime()
  .domain([new Date(2018, 0), new Date(2018, 11, 31)])
  .range([0, 365]);

console.log(scale(new Date(2018, 0)));       // Sortie: 0
console.log(scale(new Date(2018, 5)));       // Sortie: 151.37
console.log(scale(new Date(2018, 11, 31))); // Sortie: 365
```

▶ Démo

ÉCHELLES CONTINUES — FONCTIONS

- D3.js fournit plusieurs fonctions utiles pour faciliter la définition de vos domaines continus
- Il est souvent nécessaire de déterminer les valeurs minimales et maximales de vos données
- On peut utiliser les fonctions `d3.min`, `d3.max` et `d3.extent` pour réaliser ces opérations

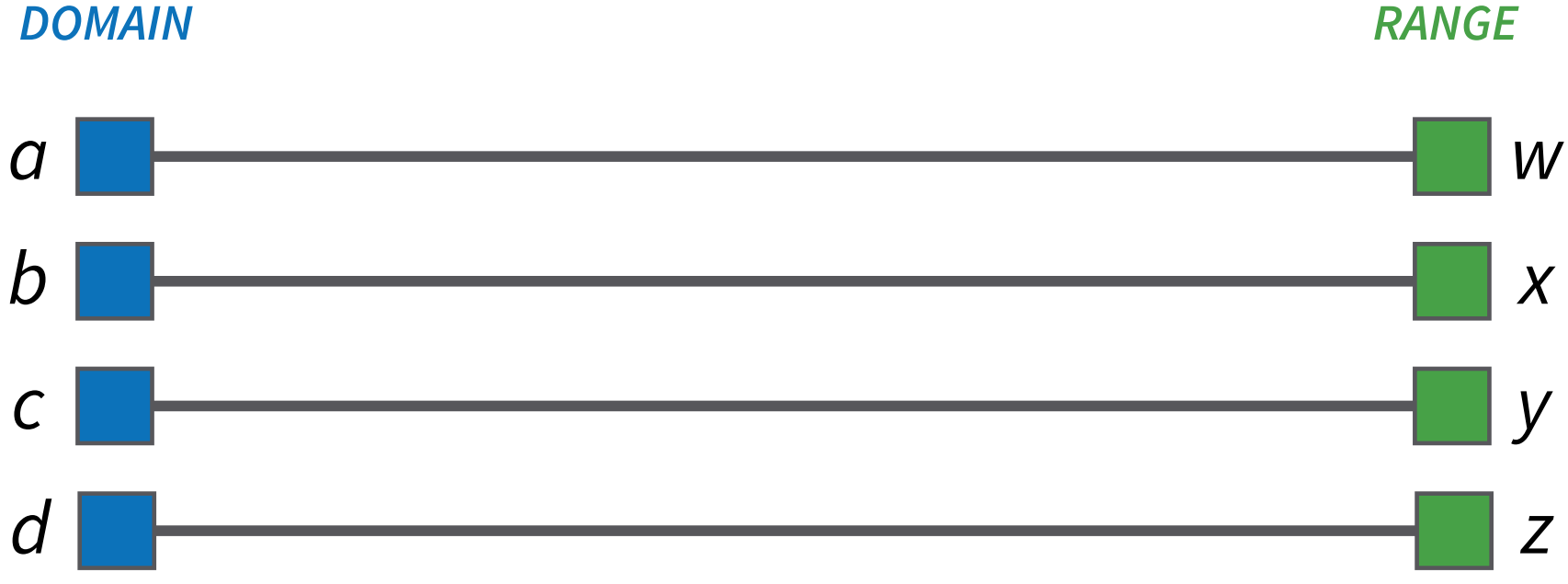
ÉCHELLES CONTINUES — EXEMPLE (2)

```
const data = [0, 1, 2, 3, 4, 5];

const scale1 = d3.scaleLinear()
  .domain([d3.min(data), d3.max(data)])
  .range([0, 500]);

// Équivalent à scale1
const scale2 = d3.scaleLinear()
  .domain(d3.extent(data))
  .range([0, 500]);

console.log(d3.min(data));    // Sortie: 0
console.log(d3.max(data));    // Sortie: 5
console.log(d3.extent(data)); // Sortie: [0, 5]
```

▶ Démo

ÉCHELLES ORDINALES

- Permet d'associer des valeurs discrètes (*domain*) à d'autres valeurs discrètes (*range*)



ÉCHELLES ORDINALES — TYPES

- `d3.scaleOrdinal` (échelle ordinale)
- `d3.scaleBand` (échelle ordinale pour faciliter la création de bandes dans un *bar chart*)
- etc.

ÉCHELLES ORDINALES — EXEMPLE

```
const scale = d3.scaleOrdinal()
  .domain(['CAQ', 'PLQ', 'PQ', 'QS'])
  .range(['#60bdf0', '#d75550', '#f0a330', '#002b6a']);

console.log(scale('CAQ')); // Sortie: #60bdf0
console.log(scale('PLQ')); // Sortie: #d75550
console.log(scale('PQ'));  // Sortie: #f0a330
console.log(scale('QS'));  // Sortie: #002b6a
```

▶ Démo

ÉCHELLES — EXEMPLE COMPLET

```
const BAR_HEIGHT = 50; // Hauteur d'une barre
const BAR_SPACING = 5; // Espacement entre les barres
const MAX_WIDTH = 300; // Largeur maximale d'une barre
const dataset = [ /* ... */ ];

const xScale = d3.scaleLinear()
  .domain([0, 100])
  .range([0, MAX_WIDTH]);

const yScale = d3.scaleBand()
  .domain(dataset.map(d => d.name))
  .range([0, BAR_HEIGHT * dataset.length])
  .paddingInner(BAR_SPACING / BAR_HEIGHT);
```

▶ Démo

ÉCHELLES — POUR EN SAVOIR PLUS

- Documentation sur les échelles avec D3.js

D3.JS — AXES (*AXIS*)

AXES

- Permet l'affichage d'une manière **compréhensible** des échelles de D3.js
- Relativement simple à utiliser
- Prend en paramètre l'échelle qu'il faut afficher lors de son initialisation

AXES — TYPES

- `d3.axisTop` (axe horizontal et étiquettes en haut)
- `d3.axisLeft` (axe vertical et étiquettes à gauche)
- `d3.axisRight` (axe vertical et étiquettes à droite)
- `d3.axisBottom` (axe horizontal et étiquettes en bas)

AXES — PARAMÈTRES

- `ticks` (nombre de marques à afficher l'axe)
- `tickFormat` (format de l'étiquette à utiliser)
- etc.

AXES — RENDU

- Une fois l'axe configuré, il faut effectuer son rendu
- Pour ce faire, il faut appeler la fonction `call` sur l'élément dans lequel l'axe doit être dessiné
- Habituellement, on crée un **groupe SVG** (voir sujet suivant) pour effectuer le rendu de l'axe

AXES — EXEMPLE

```
const scale = d3.scaleLinear()
  .domain([0, 110])
  .range([0, 250]);

// Création de l'axe
const axis = d3.axisBottom(scale)
  .ticks(5);

// Rendu de l'axe
const svg = d3.select('svg');
svg.call(axis);
```

▶ Démo

AXES — POUR EN SAVOIR PLUS

- Documentation sur les axes avec D3.js

SVG — GROUPES ET CHEMINS (*PATHS*)

GROUPES

- Définit par l'élément « `g` » en SVG
- Conteneur qui permet de regrouper plusieurs objets
- Il est possible d'appliquer une transformation sur le groupe avec la propriété `transform` (p. ex. `translate`)

GROUPES — EXEMPLE (1)

```
<svg width="500" height="200">

 <!-- Translation des éléments de 200 en X et de 50 en Y -->
 <g transform="translate(200, 50)">
  <rect width="100" height="80" x="0" y="70" fill="green"></re
  <line x1="5" y1="5" x2="250" y2="95" stroke="red"></line>
  <circle cx="90" cy="80" r="50" fill="blue"></circle>
 </g>

</svg>
```

▶ Démo

GROUPES — EXEMPLE (2)

```
const svg = d3.select('svg');
const scale = d3.scaleLinear().domain([0, 110]).range([0, 250]);

// Création des axes
const verticalAxis = d3.axisLeft(scale).ticks(5);
const horizontalAxis = d3.axisBottom(scale).ticks(5);

// Rendu des axes
svg.append('g')
  .attr('transform', 'translate(40, 5)')
  .call(verticalAxis);

svg.append('g')
  .attr('transform', 'translate(40, 255)')
  .call(horizontalAxis);
```

▶ Démo

CHEMINS (*PATHS*)

- Définit par l'élément « **`path`** » en SVG
- Il s'agit de l'objet le plus **versatile** en SVG puisqu'il permet de dessiner une forme quelconque
- La propriété « **`d`** » d'un `path` permet de spécifier comment dessiner la forme voulue

CHEMINS — EXEMPLE (1)

```
<svg viewBox="0 0 100 100">
  <path fill="none" stroke="red" d="M 10,30
        A 20,20 0,0,1 50,30
        A 20,20 0,0,1 90,30
        Q 90,60 50,90
        Q 10,60 10,30 z"></path>
</svg>
```

▶ Démo

Est-ce que je suis censé comprendre le charabia écrit dans la propriété « d » ?

Non! D3.js va s'occuper d'écrire cela!

CHEMINS — D3.JS

- La bibliothèque D3.js fournit la fonction `d3.line` permettant de tracer facilement des chemins
- Cette fonction définit les propriétés `x` et `y`
- Lorsque la fonction est configurée, celle-ci peut être appelée avec une liste de données à dessiner

CHEMINS — EXEMPLE (2)

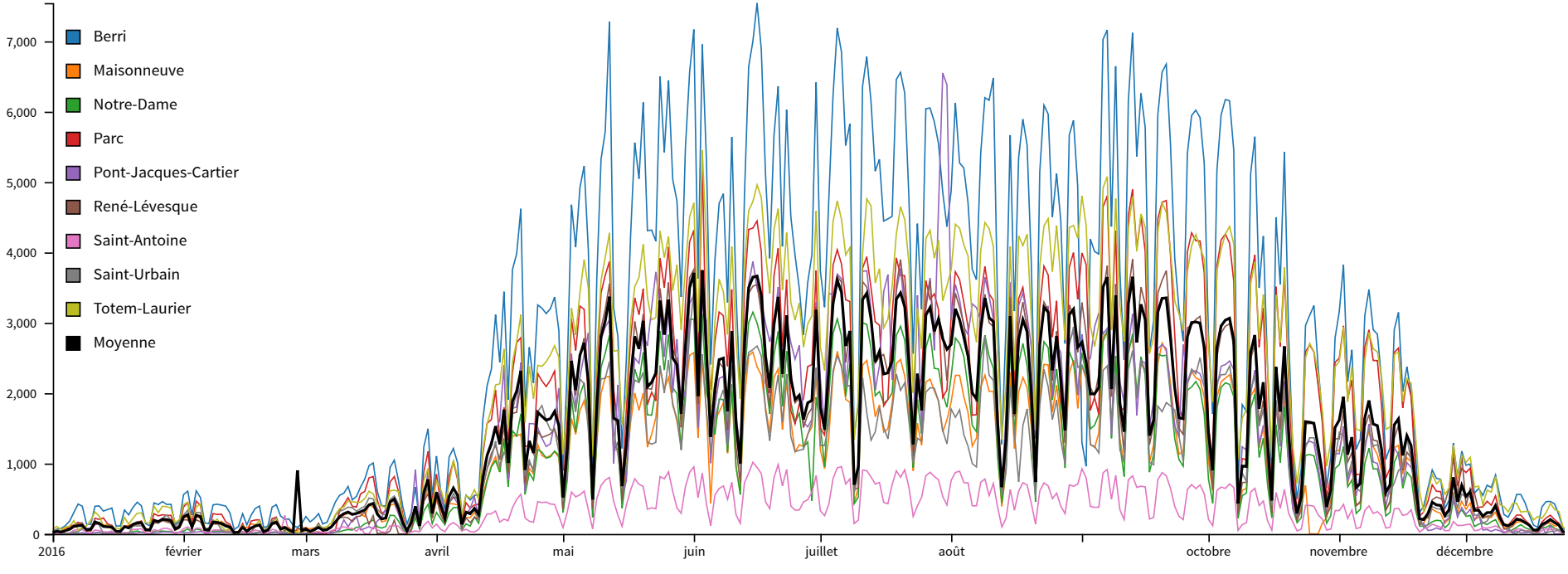
```
const data = [ /* ... */ ];

const x = d3.scaleTime()
  .domain(d3.extent(data, d => d.date))
  .range([0, 500]);
const y = d3.scaleLinear()
  .domain([0, d3.max(data, d => d.value)])
  .range([200, 0]);

const svg = d3.select('svg');
const line = d3.line()
  .x(d => x(d.date))
  .y(d => y(d.value));

svg.append('path')
  .attr('d', line(data));
```

▶ Démo

MISE EN PRATIQUE

MISE EN PRATIQUE

- Réaliser un *line chart* à l'aide des données sur le nombre de vélos passant quotidiennement dans certaines rues de Montréal en 2016.



MISE EN PRATIQUE — DONNÉES

- Les données sont issues des **vélos-compteurs** qui se trouvent sur les pistes cyclables de Montréal
- Les données au format CSV ont été récupérées sur le portail des données ouvertes de la ville de Montréal

Image d'un vélo-compteur se trouvant sur la rue Laurier · © François Démontagne, 2015

MISE EN PRATIQUE — DONNÉES

- Chaque ligne du fichier CSV indique le nombre de vélos ayant circulé sur les rues à une date précise

Données CSV

MISE EN PRATIQUE — DONNÉES

- Pour être en mesure d'afficher correctement les données sur le *line chart*, il faut les **restructurer**

```
[
  {
    name: "Nom de la rue",
    values: [
      {
        date: new Date(),
        count: 220
      },
      // Suite du tableau listant les données pour une rue...
    ]
  },
  // Suite du tableau listant les rues...
]
```

MISE EN PRATIQUE — DIRECTIVES

- Définir les échelles et les axes pour *x* et *y*
- Dessiner les **lignes** pour les rues et la moyenne
- Dessiner les **axes**
- Effectuer le rendu de la **légende**
- Ajouter la possibilité d'afficher ou de masquer une ligne (bonus)

MISE EN PRATIQUE

- Pour débuter, téléchargez le dossier ZIP contenant le code de départ pour l'exercice
- Complétez, par la suite, le fichier **script.js**
- Utilisez **Firefox** ou un serveur web local pour tester votre script*

Télécharger ▶ Correction

* Google Chrome bloque les requêtes asynchrones réalisées dans un fichier local si celui-ci n'est pas hébergé sur un serveur local