Visualisation de données – BDTN

Département TIC

ESIGELEC

Octobre 2018

derrien@esigelec.fr

Table des matières

Présentation du cours

Contenu du cours :

- Un peu de théorie sur ce qui est nécessaire pour la visualisation de données,
- Lister les principaux types de diagrammes, avec cas d'utilisation,
- Création de diagrammes, à partir de jeux de données en utilisant des "outils" .

Mais aussi...

- Du *JavaScript* !

JavaScript ? ? ?

Pourquoi du *JavaScript* ?

- De nombreux outils de visualisation sont prévus pour le web,
- Et ils sont réalisés et utilisables avec du *JavaScript*.
- Le *JavaScript*, c'est beaucoup plus que la vérification de formulaire coté client ou *jQuery* :
  - Des *frameworks* coté client (*Angular*, *React*,...)
  - Mais également coté serveur : *node.js*,
  - Développement "*fullstack*" en *JavaScript*,
  - Des sur-ensembles du langage, comme par exemple *TypeScript*.

Organisation

- Les trois premières séances :
  - Plusieurs séquences de cours,
  - Entrecoupées d'exercices de mise en pratique.
- Pendant la quatrième séance, début du projet.
- Les séances suivantes (2 heures) sont pour le travail et le suivi du projet,
- À la dernière séance : soutenances.

Évaluation

Les évaluations :

- Les soutenances,
- Un examen d'une heure,
- Le suivi en TP.

Attention :

- Même s'il s'agit de blocs de 2×2 heures sur l'emploi du temps, en pratique il s'agit de séances de 4 heures,
- Être absent aux 2 premières heures, implique une absence de 4 heures.

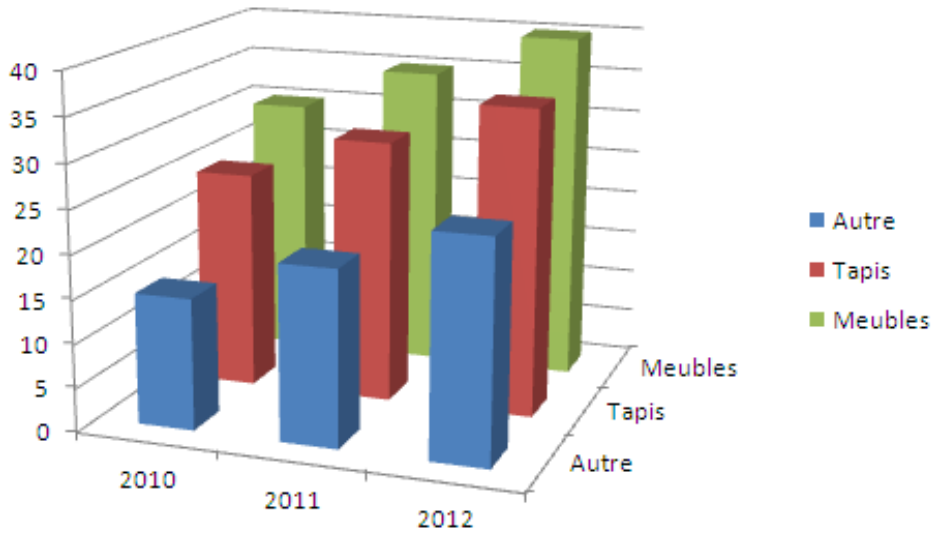
Les différentes catégories

Les diagrammes peuvent être classés en plusieurs catégories :

- Les diagrammes courants (histogrammes, courbes, pie-chart, ...)
- Les visualisations multi-dimensionnelles (nuages de points, inselberg, bulles, ...)
- Les visualisations multi-niveaux
- Les visualisations de réseaux

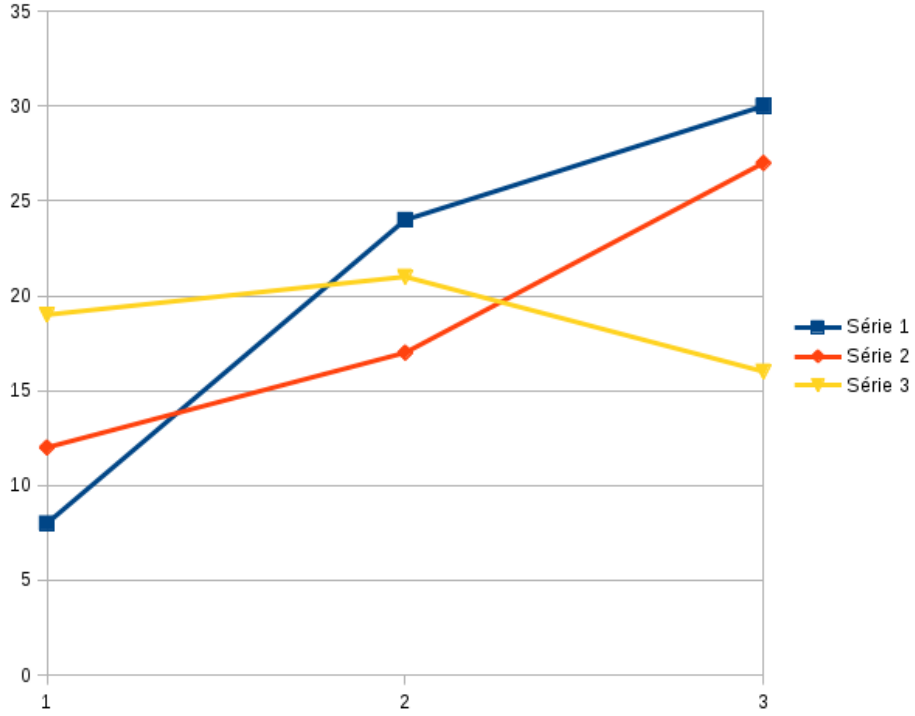
Histogrammes

- Diagrammes courants qui visent à comparer des séries entre elles sur plusieurs catégories,
- Les données sont représentées sous forme de bâtons (2D ou 3D)



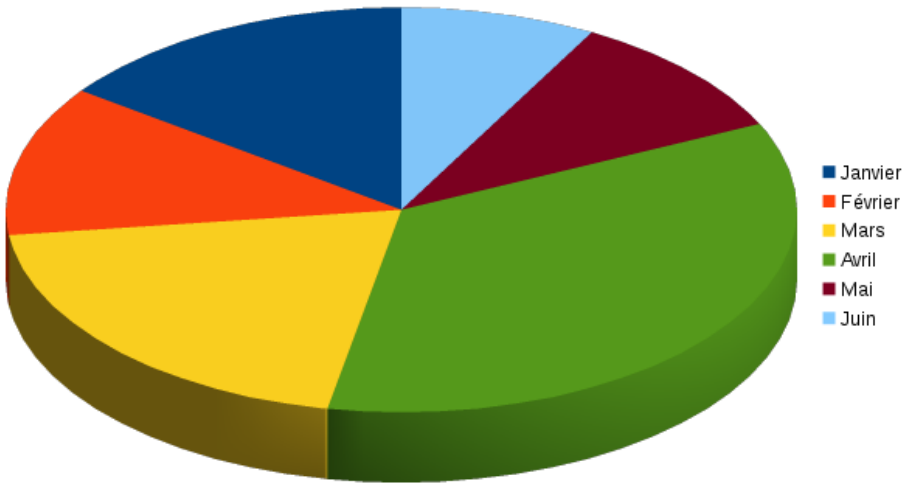
Courbes

- Autre diagramme courant permettant la comparaison de séries,
- Il permet de visualiser à quel moment deux courbes se croisent.



Pie Chart

- Présente les informations sous la forme d'un "camembert",
- Intéressant pour mettre en évidence les proportions des différentes valeurs,
- Peut être intéressant également quand on supprime certaines données (par un clic par exemple, voir exercice).



Nuages de points

- Permet de comparer des valeurs de différentes dimensions,
- Dans l'exemple, il y a 4 dimensions qui sont comparées,
- Sur la diagonale, chaque dimension est comparée avec elle-même.

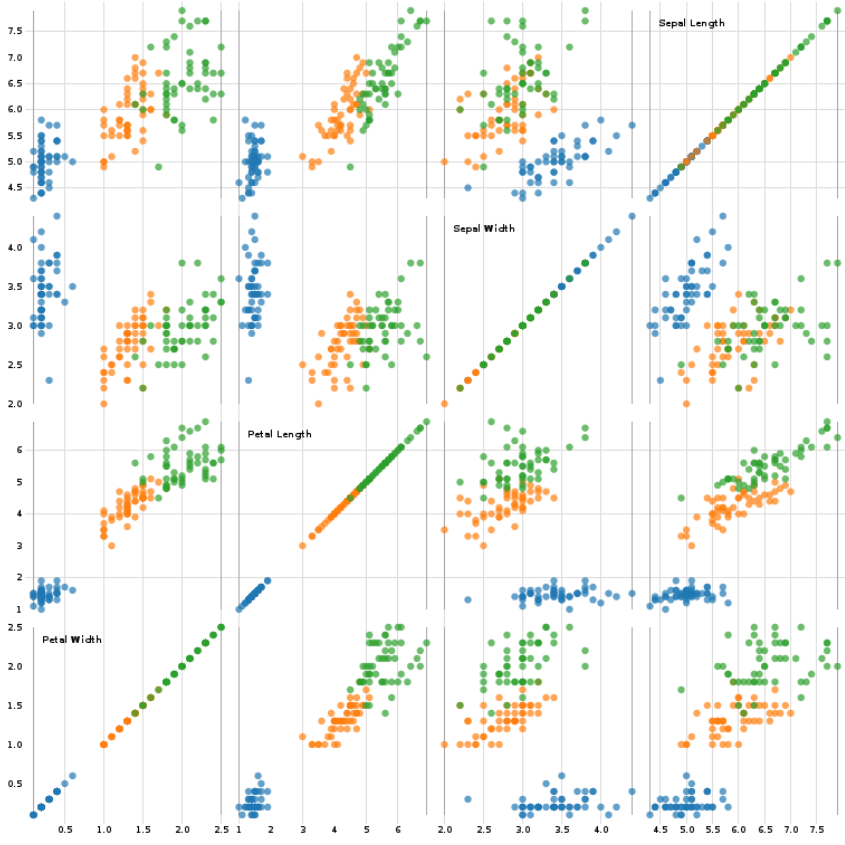


Diagramme d'inselberg

- Chaque dimension est représentée par un axe vertical,
- Ce diagramme permet de mettre en évidence les similarités et différences en fonction des dimensions,
- Exemple traité en exercice dans la suite du cours.

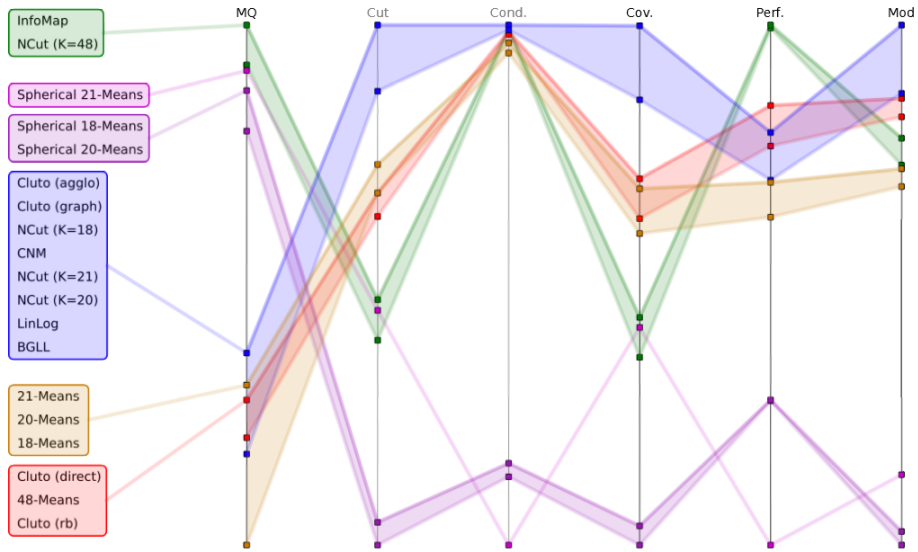


Diagramme bulles

- Diagramme pour une représentation de 3 dimensions,
- Les deux premières correspondent aux coordonnées,
- La troisième est exprimée par le diamètre de la bulle.

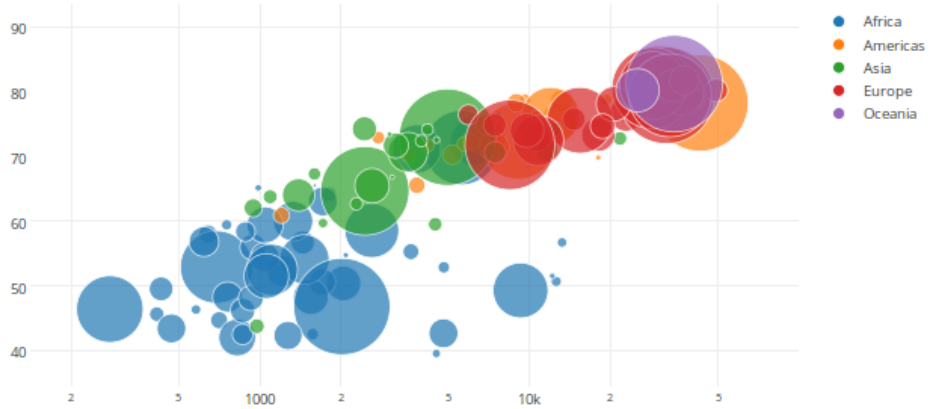
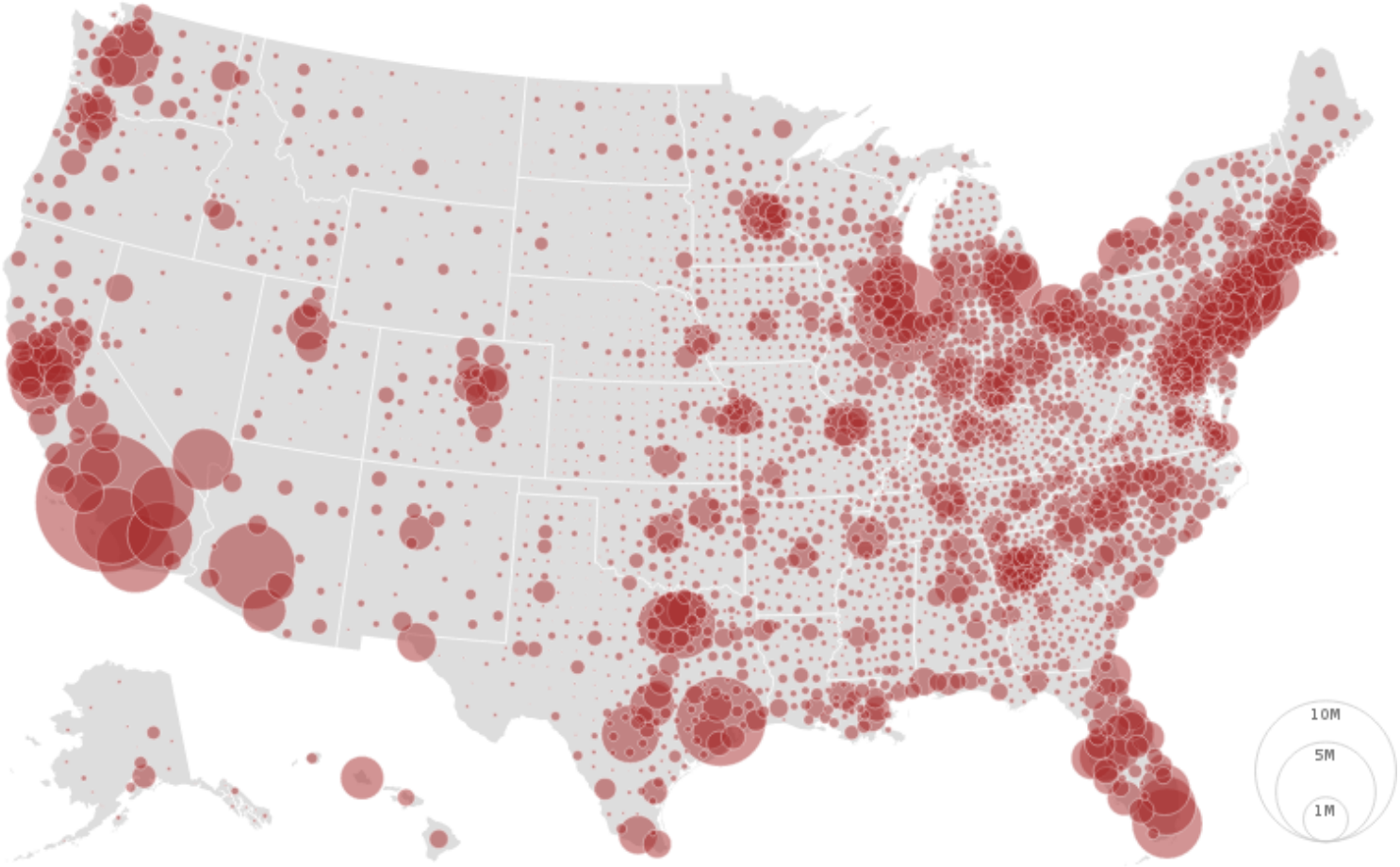


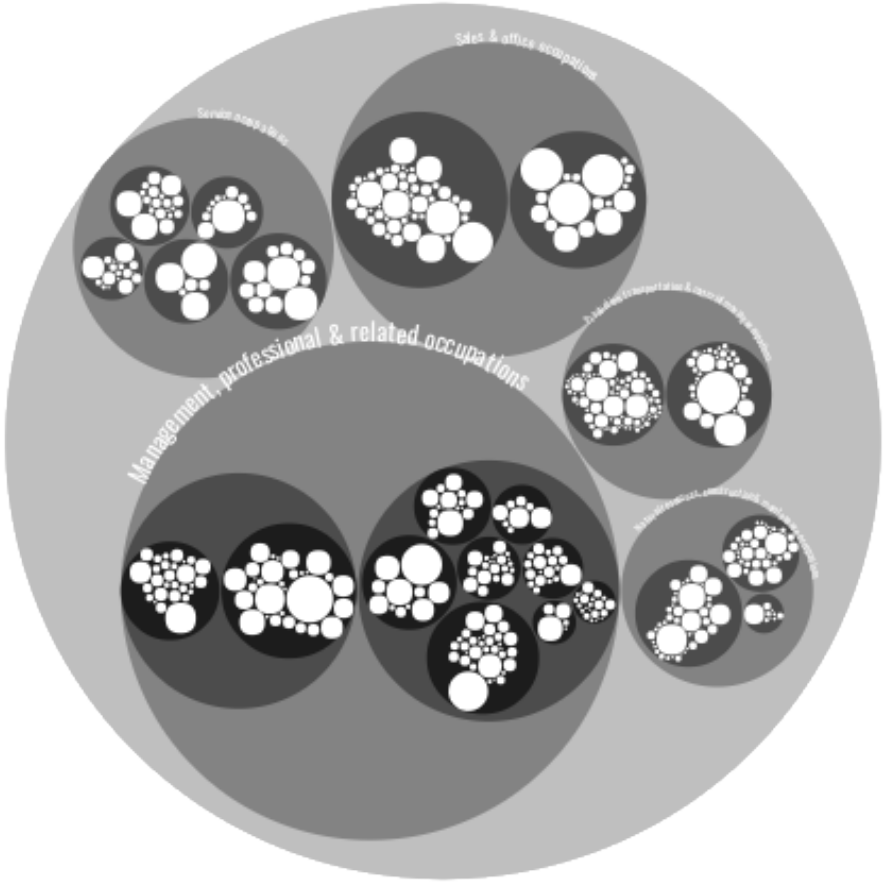
Diagramme bulles

Variante intéressante pour exprimer une valeur en fonction des coordonnées sur une carte :



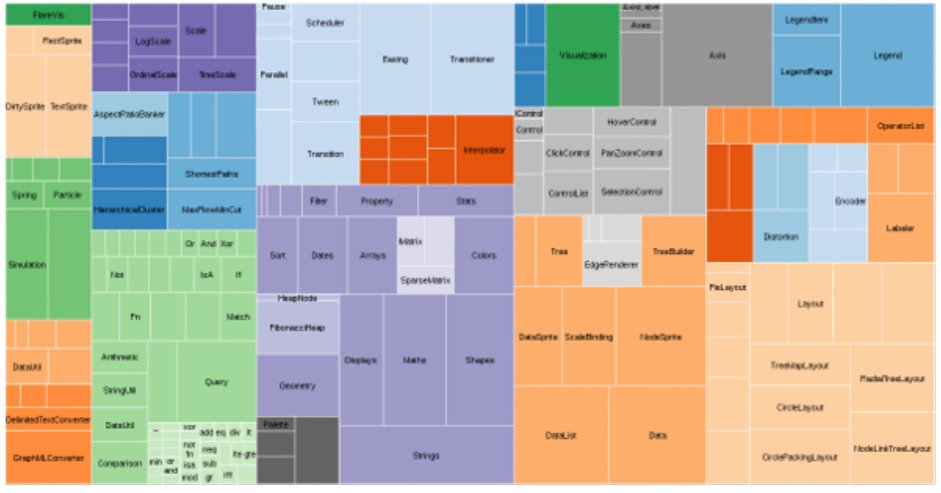
Les visualisations multi-niveaux

- Particulièrement utilisées lorsque la quantité d'information est importante,
- Le diagramme montre d'abord les informations les plus générales,
- Une interaction permet d'obtenir des informations plus détaillées.



Treemaps

- Ces diagrammes présentent également l'information sous forme hiérarchique,
- Mais l'espace est optimisé pour ne pas laisser de zones vides.

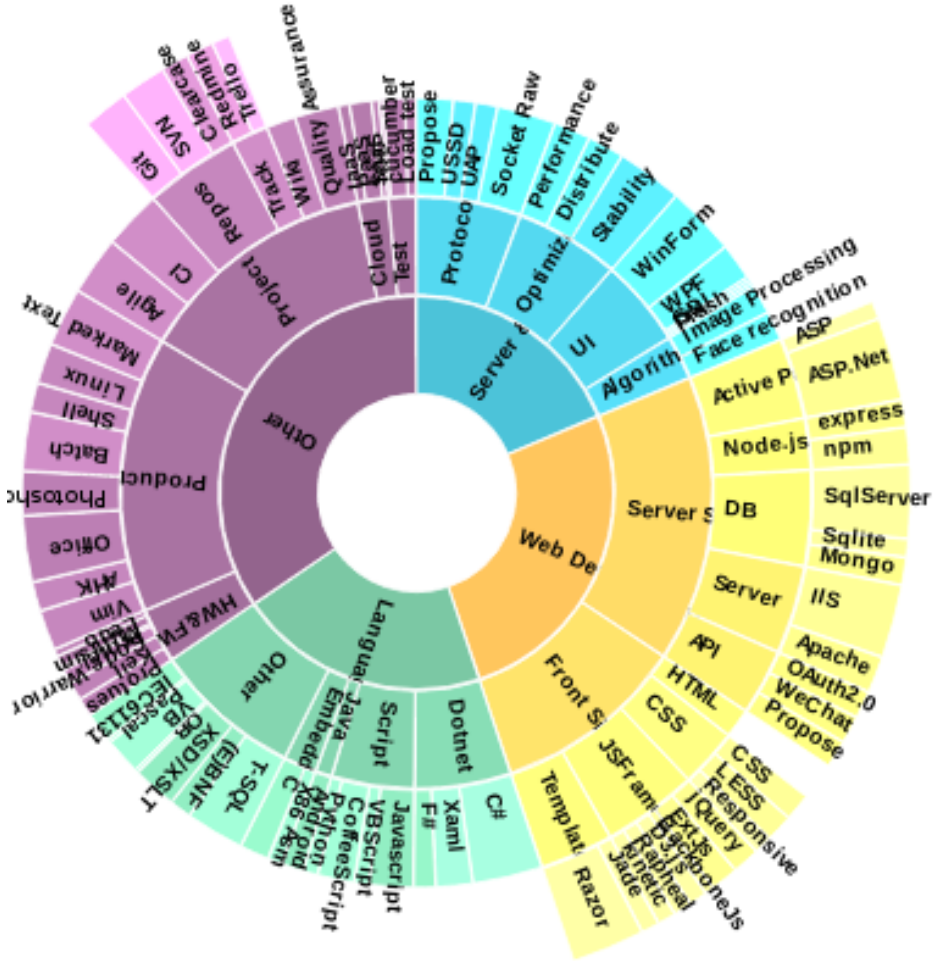


Cartographie

- La cartographie de type “Google Map” est un des meilleurs exemples de visualisation multi-niveaux,
- Par l’intermédiaire de la molette il est possible de réaliser d’importants changements d’échelle,
- Passage de la représentation d’un pays à la représentation de régions, villes, rues.

Secteurs hiérarchiques

- Représentation hiérarchique des données,
- Intéressant lorsqu'il est possible de définir un secteur comme nouveau centre du diagramme,
- D'autres interactions sont possibles.



La visualisation de réseaux

- Permet de lier entre eux des “acteurs”,
- Le lien permet de faire ressortir le critère que l'on souhaite mettre en évidence,
- De nombreuses variantes existent, comme par exemple les modèles de force.

Quelques frameworks

Voici quelques frameworks *JavaScript* :

- AmCharts
- DyGraph.js
- D3.js
- InfoVis
- vis.js
- Springy
- Polymaps.js
- Dimple.js
- Sigma.js
- Raphael
- gRaphael
- Leaflet
- EmberChart

Le framework Raphael

- Raphael est une “sur-couche” du canvas quel l'on trouve en html 5,
- Ce n'est pas un framework destiné uniquement à la réalisation de diagrammes,
- Il facilite la réalisation de dessin sur une page web,
- Donc nécessité de programmer en *JavaScript* ! ☺

Liens utiles :

- Le site de Raphael
- Tutoriel très complet.

Utilisation

- Raphael se présente sous la forme d'un fichier *JavaScript*,
- Ce fichier doit donc être inclus dans le code de la page html,
- Une version minifiée est disponible,
- Lors de la phase de développement, il est préférable de télécharger le fichier,
- En production vous pouvez faire appel directement au fichier sur le site du développeur.

Exemple de code

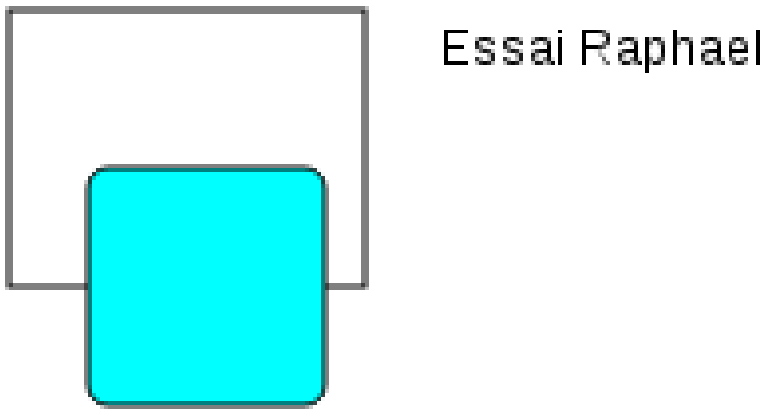
Le fichier html :

```
1  <!doctype html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>Premier exemple</title>
6      <script src="./raphael.min.js"></script>
7    </head>
8    <body>
9      <div id="exemple"></div>
10     <script src="./exemple.js"></script>
11   </body>
12 </html>
```

Exemple de code

Contenu du fichier `exemple.js` :

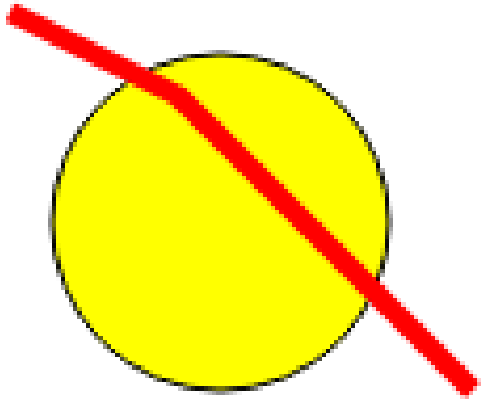
```
1 var paper = Raphael("exemple", 800, 400);
2 paper.rect(20, 20, 90, 70);
3 paper.rect(40, 60, 60, 60, 5).attr({fill: "
    #00FFFF"});
4 paper.text(170, 30, "Essai Raphael").attr({"
    font-size": "12px"});
```



Exemple de code – suite

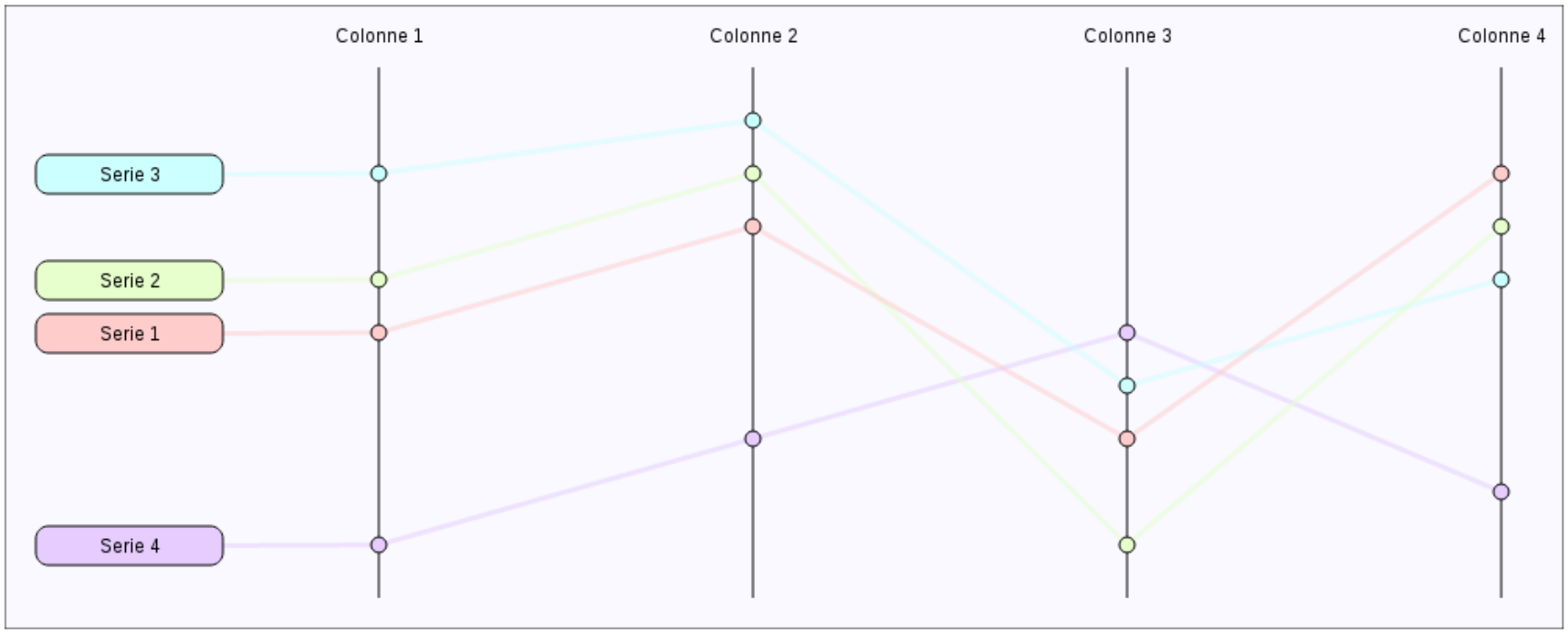
Contenu du fichier `exemple.js` :

```
1 var cercle = paper.circle(60,60,40);
2 cercle.attr({fill: "yellow"})
3 var path = paper.path("M10,10L40,20L70,70")
4 path.attr({"stroke-width":"5px", "stroke": "
    red"});
```



Application : Diagramme d'inselberg

On souhaite obtenir ce diagramme :



Application : Diagramme d'inselberg

Le jeu de données utilisé est le suivant :

```
var data = {
        series : [
            {name: "Serie 1", values: [0.5, 0.3, 0.7, 0.2]},
            {name: "Serie 2", values: [0.4, 0.2, 0.9, 0.3]},
            {name: "Serie 3", values: [0.2, 0.1, 0.6, 0.4]},
            {name: "Serie 4", values: [0.9, 0.7, 0.5, 0.8]}
        ],
        colonnes : ["Colonne 1", "Colonne 2", "Colonne 3", "Colonne 4"]
};
```

On constate que la variable data contient :

- Le nom des colonnes et des séries,
- Des valeurs comprises entre 0 et 1 (pourcentage).

Application : Diagramme d'inselberg

Mise en place :

1. Télécharger le fichier `raphael.min.js`,
2. Créer la page web (cf. diapo 23),
3. Créer le fichier *JavaScript* qui contiendra votre code,
4. Déclarer et initialiser la variable `paper`,
5. Déclarer autant de variables que nécessaire pour les dimensions de la zone, les marges, le nombre de colonnes, le nombre de séries, etc...
6. Ajouter un rectangle pour le fond, avec une couleur claire.

Application : Diagramme d'inselberg

Ajout des colonnes et des étiquettes :

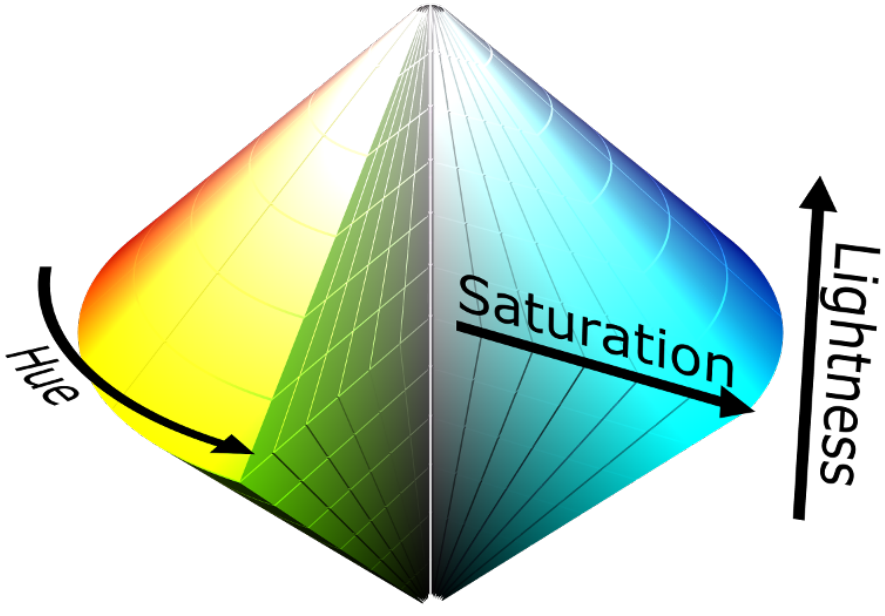
1. Calculer la distance entre les colonnes,
2. Dessiner les quatre axes,
3. Ajouter le nom des colonnes,
4. Ajouter les étiquettes,
5. Ajouter le nom de la série dans chaque étiquette.

Problème avec les couleurs !

La définition des couleurs en RGB n'est pas très pratique : si le nombre de séries augmente, comment choisir les couleurs ?

Le modèle HSL

- H : La teinte (*Hue*) comprise entre 0° et 360°,
- S : Le contraste (*Saturation*) compris entre 0 et 100%,
- L : La luminosité (*Lightness*) comprise entre 0 et 100%.

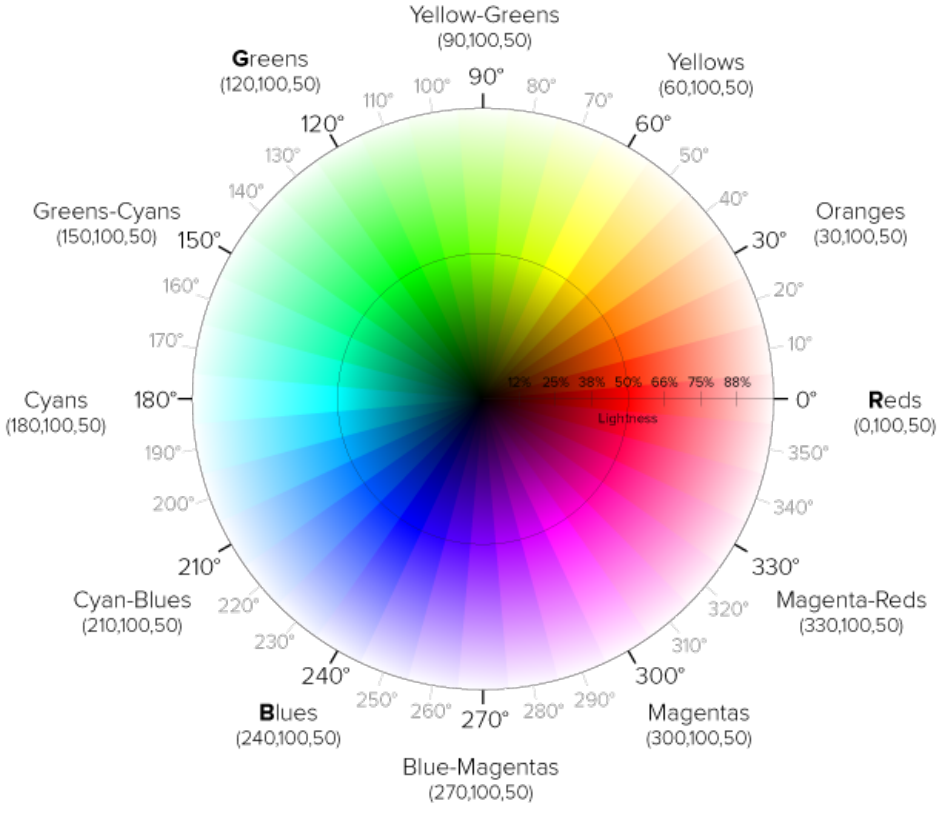


Exemple d'utilisation : `hsl(300, 100, 50)`.

Le modèle HSL

Intérêt : Pour obtenir n couleurs différentes, il suffit d'avoir un écart de $360/n$ entre chaque teinte.

Remarque : La fonction `hsla` prend la transparence en quatrième argument.



Application : Diagramme d'inselberg

Les séries :

1. Utiliser le modèle *HSL* pour la couleur de vos étiquettes,
2. Il est préférable de conserver ces couleurs dans un tableau, elles seront réutilisées par la suite,
3. Créer les chemins pour les différentes séries,
4. Ainsi que les points aux intersections avec les colonnes.

On souhaite maintenant pouvoir mettre une série en évidence en faisant un clic dessus. Pour cela il faut gérer les événements.

Gestion des événements

Exemple : Le clic sur un élément.

```
var rect1 = paper.rect(10, 10, 40, 50).attr({fill: "red"});
// Définition de la fonction appelée lors d'un clic
rect1.click(function(e, x, y) {
    alert("Le rectangle rouge a été cliqué en ("+x+","+y+")");
});
// Autre méthode :
var rect2 = paper.rect(30, 50, 40, 50).attr({fill: "blue"});
// On appelle une fonction définie en dehors
rect2.click(maFonction);
function maFonction(e, x, y) {
    alert("Le rectangle bleu a été cliqué en ("+x+","+y+")");
}
```

Application : Diagramme d'inselberg

Les événements :

1. Définir un deuxième tableau de couleurs, en gardant les mêmes teintes que dans le premier, et en modifiant le contraste,
2. Lorsque l'utilisateur clique sur une série (étiquette, path, nœud) elle s'affiche avec la couleur de ce tableau,
3. Toute autre série déjà sélectionnée, reprend sa couleur initiale,
4. Un clic sur le fond du diagramme désélectionne toutes les séries.

Pour aller plus loin :

- Utiliser les événements `mousedown`, `mousemove` et `mouseup` pour déplacer les étiquettes (drag and drop).

Application : Diagramme d'inselberg

- Pour empêcher la sélection du texte pendant le drag and drop d'un élément, le plus simple consiste à désactiver la sélection,
- Pour cela, créer un style qui s'appliquera aux `div` contenant vos diagrammes,
- La règle n'est pas la même selon les navigateurs...

```
.noselect {
    -webkit-user-select: none; /* Chrome, Opera, Safari */
    -moz-user-select: none; /* Firefox 2+ */
    -ms-user-select: none; /* IE 10+ */
    user-select: none; /* Standard syntax */
}
```

AmCharts

- Sur le site d'AmCharts il est facile de récupérer un modèle de graphique,
- Puis de l'éditer pour mettre nos propres données,
- Et l'ajouter dans une page html.

Récupération d'AmCharts :

1. Récupérer le code correspondant au graphique "Simple Column Chart",
2. Ajouter les balises html nécessaires pour en faire une page web,
3. Tester la page.

AmCharts

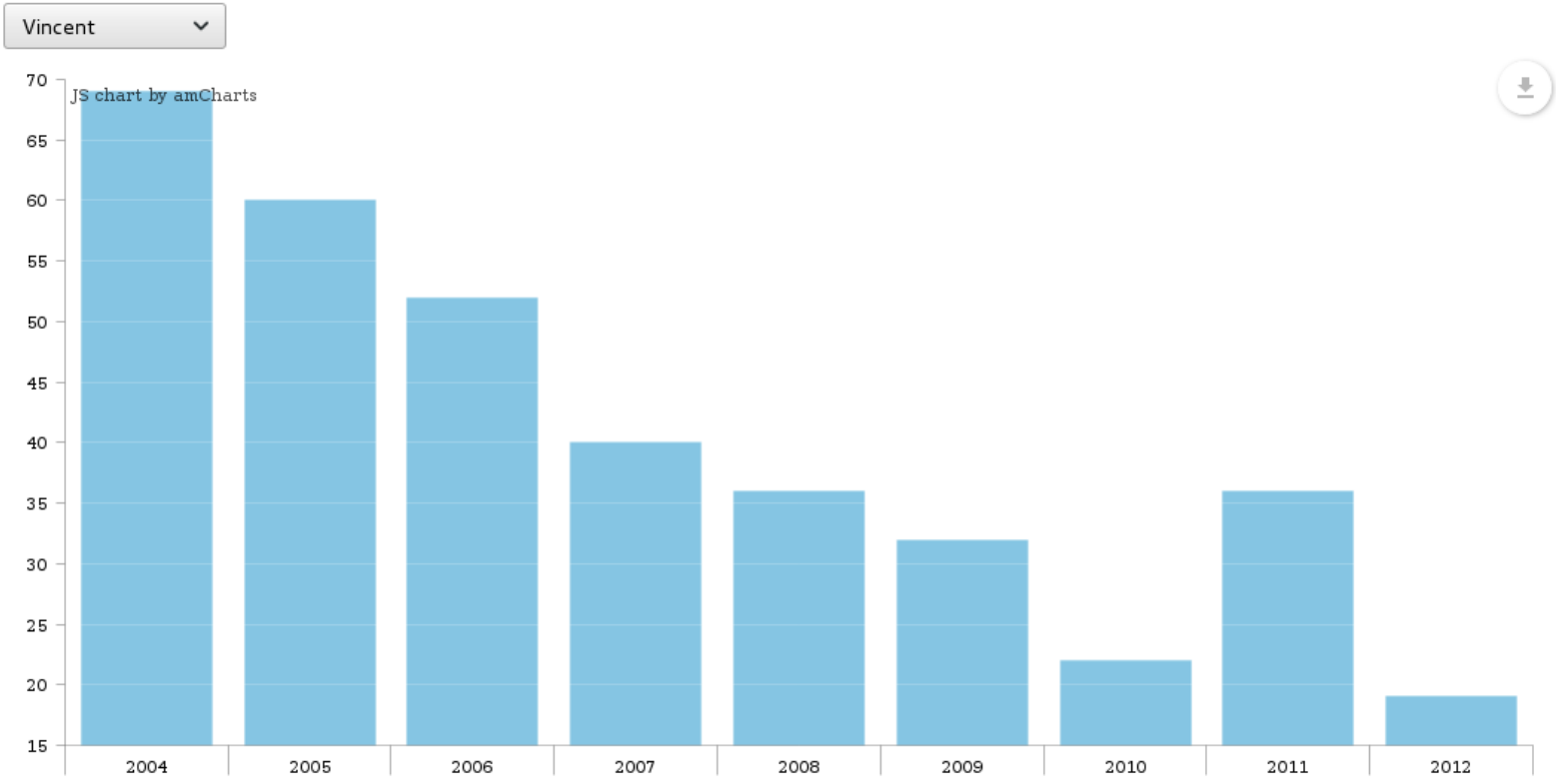
Nous allons utiliser les données de la ville de Paris : opendata.paris.fr

Les données :

1. Trouver le jeu de données des prénoms,
2. Construire une requête pour 2004 dans l'onglet API,
3. Nous allons utiliser *jQuery* pour récupérer ces données sur notre page,
4. Ajouter *jQuery* dans la page (lien ou CDN).

AmCharts

Résultat attendu :



AmCharts

Principe :

- Déclarer une liste qui va contenir toutes les informations utiles,
- À la fin du traitement, cette liste est de la forme :

```
{
  "toto": [{annee: 2004, nombre: 10},
           {annee: 2005, nombre: 8},
           ...],
  "titi": [...],
  ...
}
```

AmCharts

Attention !

- Pour une année donnée, la liste obtenue peut contenir plusieurs fois un même prénom,
- Dans ce cas, il faut faire la somme de toutes les valeurs
- Pensez au cas où le prénom n'est pas encore présent dans la liste !

Le framework leaflet

- Leaflet est un framework permettant de créer facilement des cartes,
- Lors de la création vous pouvez choisir les coordonnées du centre ainsi que le niveau de zoom,
- Ensuite il est très facile d'ajouter des formes géométriques, popup, événements, etc...

Exemple :

- Vous trouverez sur ENT un exemple commenté,
- Décompressez le fichier dans un dossier,
- Ajoutez le code de Leaflet,
- Testez !

Le framework leaflet

Exercice 1 :

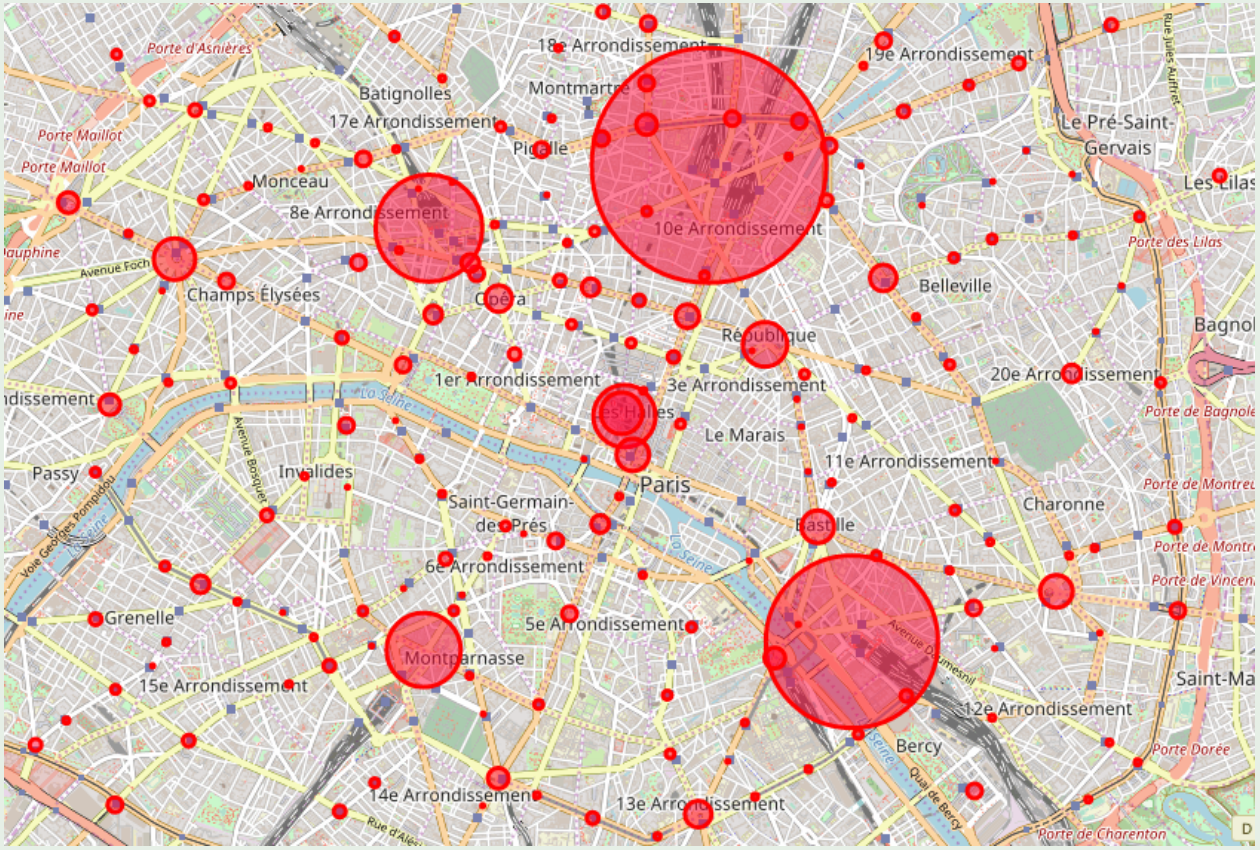
À partir du code précédent :

- Créer un tableau vide,
- Lors d'un clic de l'utilisateur sur la carte, ajouter les coordonnées du point coorespondant dans le tableau,
- Ajouter un bouton sur la page web,
- Lorsque l'utilisateur clique sur ce bouton, tracer le polygone formé par les points contenus dans le tableau,
- Puis réinitialiser le contenu du tableau.

Le framework leaflet

Exercice 2 :

L'objectif est de créer une carte des stations RATP mettant en évidence la fréquentation :



Le framework leaflet

Exercice 2 (suite) :

- Repartir du code sur ENT,
- Supprimer ce qui n'est pas utile,
- Centrer la carte sur Paris,
- Les données sur le trafic sont ici,
- Les données sur les positions des stations sont ici.

Présentation de *D3.js*

- D3 : "*Data-Driven Documents*",
- Il s'agit d'une bibliothèque *JavaScript* qui permet d'afficher facilement des données sous forme graphique,
- Elle utilise notamment le format SVG (*Scalable Vector Graphics*),
- Se format est directement accessible en html avec la balise `<svg></svg>`,
- Avant de commencer à voir le fonctionnement de *D3.js*, nous allons regarder rapidement ce qui peut être fait en SVG.

Le format *SVG* – le code de base

La balise <svg> se place directement dans le corps du document.

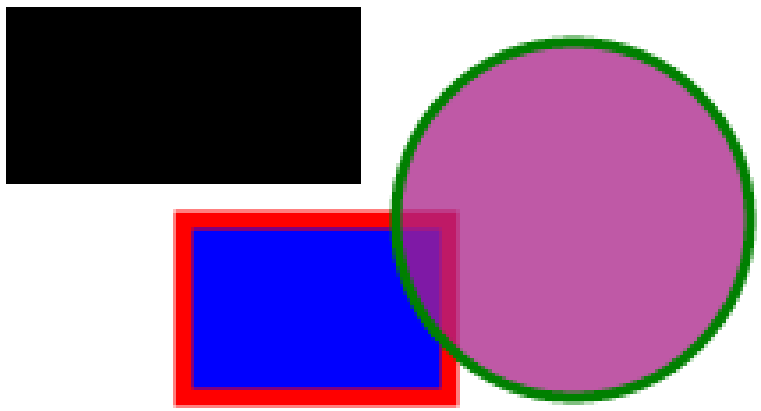
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Essai SVG</title>
    </head>
    <body>
        <svg width="300px" height="300px">
            <!-- Contenu SVG -->
        </svg>
    </body>
</html>
```

Le format *SVG* – exemple 1

Le code suivant :

```
<svg width="300px" height="300px">
  <rect width="100" height="50"/>
  <rect x="50px" y="60px" width="75" height="50"
      fill="blue" stroke="red" stroke-width="5px"/>
  <circle cx="160" cy="60" r="50" fill="#AA2288"
      fill-opacity="0.75" stroke="green" stroke-width="3px"/>
</svg>
```

Donne :

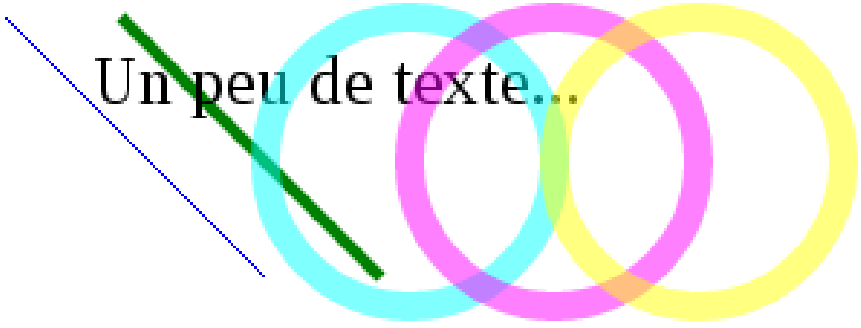


Le format *SVG* – exemple 2

Le code suivant :

```
<line x1="10" y1="10" x2="100" y2="100" stroke="blue"  />
<line x1="50" y1="10" x2="140" y2="100" stroke="green" stroke-width="4px"/>
<text x="40px" y="40px" style="font-size:24px;">Un peu de texte...</text>
<circle cx="150" cy="60" r="50" stroke="#00FFFF" stroke-width="10" fill="none" stroke-opacity="0.5" />
<circle cx="200" cy="60" r="50" stroke="#FF00FF" stroke-width="10" fill="none" stroke-opacity="0.5" />
<circle cx="250" cy="60" r="50" stroke="#FFFF00" stroke-width="10" fill="none" stroke-opacity="0.5" />
```

Donne :

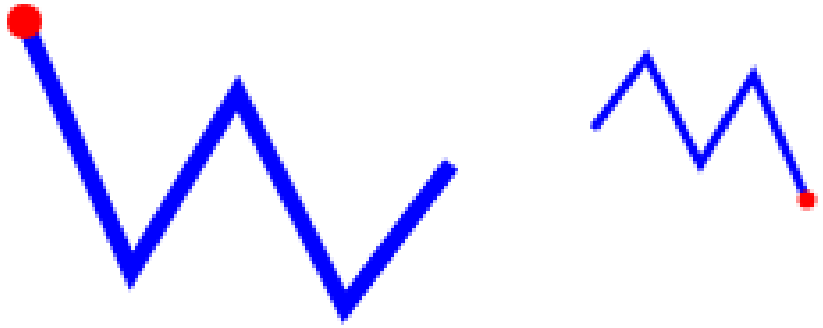


Le format *SVG* – exemple 3

Le code suivant :

```
<path d="M20,20L50,90L80,40,L110,100,L140,60" stroke="blue"
    stroke-width="5" fill="none"/>
<circle cx="20" cy="20" r="5" fill="red" />
<g transform="translate(250,80),rotate(180),scale(0.5)">
    <path d="M20,20L50,90L80,40,L110,100,L140,60" stroke="blue"
        stroke-width="5" fill="none"/>
    <circle cx="20" cy="20" r="5" fill="red" />
</g>
```

Donne :



Principe de *D3.js*

Nous allons voir le principe de ce *pattern* au travers de cinq exemples :

- La sélection d'éléments dans le *DOM*,
- Comment dessiner au format SVG,
- Un premier graphique (histogramme),
- Le pattern "Enter-Update-Exit",
- Un second graphique (pie chart).