

## Visualisation d'informations

### TP4 : Leaflet

Leaflet<sup>1</sup> est une librairie JavaScript permettant d'insérer dans vos pages Web des cartes interactives. Dans ce TP, vous allez voir comment insérer une carte dans une page Web et comment utiliser cette carte pour y visualiser des données géolocalisées. Les exercices sont librement adaptés d'un code proposé par Andy Woodruff, Ryan Mullins and Cristen Jones<sup>2</sup>.

#### Exercice 1 : afficher une carte

Commencez par créer une page HTML. Téléchargez la dernière version de Leaflet<sup>3</sup> et décompressez-la dans le répertoire contenant votre page. Chargez la librairie JavaScript et le style de Leaflet dans cette page :

```
<link rel="stylesheet" href="leaflet/leaflet.css" />
<script src="leaflet/leaflet.js"></script>
```

Dans le body, créez un div vide muni d'un identifiant (par exemple divmap). Donnez-lui une taille de 900 x 500 pixels et une bordure grise de 3 pixels. Dans votre navigateur, vérifier que vous obtenez bien une page contenant uniquement un rectangle vide.

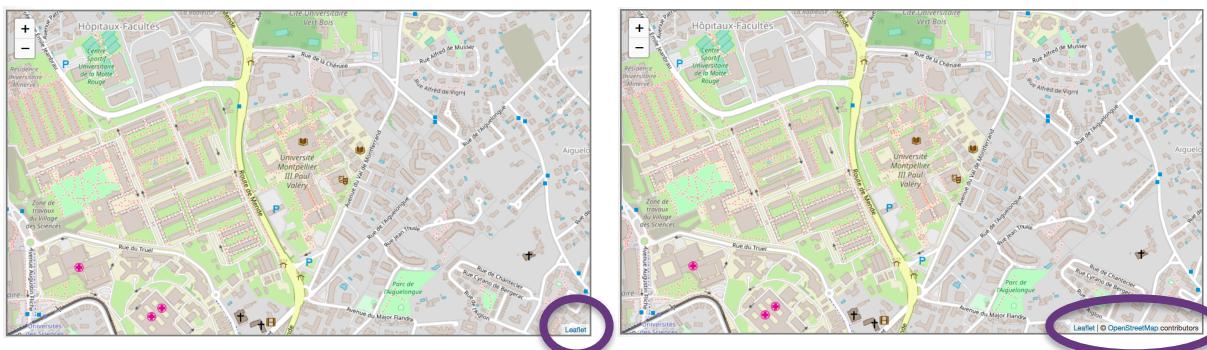
En-dessous de votre div, ouvrez un script JavaScript et ajoutez-y une carte leaflet :

```
var map = L.map('divmap'); Création de la carte
map.setView([43.6326, 3.87], 16); Spécification de la latitude et de la longitude du centre de la carte (entre crochets) et du niveau de zoom initial
```

Vous allez maintenant créer une couche (*layer*) contenant des tuiles du serveur d'OpenStreetMap<sup>4</sup> :

```
L.tileLayer('https://s.tile.openstreetmap.org/{z}/{x}/{y}.png')
            .addTo(map);
```

Testez votre code, vous devriez obtenir l'image de gauche ci-dessous :



<sup>1</sup> <https://leafletjs.com/>

<sup>2</sup> <https://maptimeboston.github.io/leaflet-intro/>

<sup>3</sup> <https://leafletjs.com/download.html>

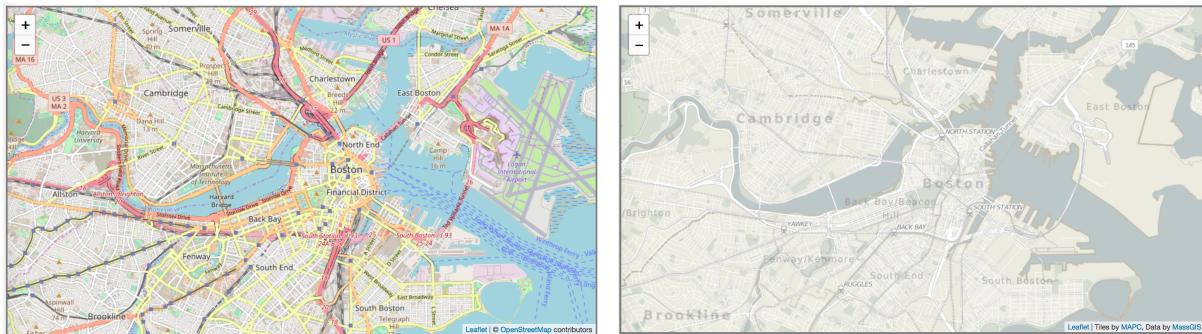
<sup>4</sup> <https://www.openstreetmap.org/>

Comme vous pouvez le remarquer, Leaflet est affiché en bas à droite de la carte mais il manque le nom du fournisseur des tuiles. Il vous faut donc l'ajouter, surtout si vous devez diffuser votre visualisation ! Pour cela, ajoutez le paramètre suivant à la méthode `tileLayer` (après la chaîne de caractères contenant l'adresse du fournisseur de tuiles) :

```
{ attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors' }
```

Vous devriez voir apparaître les informations en bas à droite de la carte, comme sur l'image de droite ci-dessus.

Sur Internet, trouvez la latitude et la longitude de la ville de Boston et modifiez les paramètres de la méthode `setView` pour la centrer sur cette ville. Changez aussi le niveau de zoom initial pour avoir par défaut une vue plus large de la zone (image à gauche ci-dessous).



Nous allons maintenant changer de fournisseur de tuiles afin d'obtenir une carte plus « sobre ». Il en existe un grand nombre sur le Web. Ici, nous allons en prendre une provenant de MassGIS<sup>5</sup> : <http://tiles.mapc.org/basemap/{z}/{x}/{y}.png>.

Vous devez aussi modifier le champ `attribution` :

```
Tiles by <a href="http://mapc.org">MAPC</a>,  
Data by <a href="http://mass.gov/mgis">MassGIS</a>
```

Vous devriez obtenir l'image de droite ci-dessus.

Dans votre navigateur, diminuez plusieurs fois le niveau de zoom (avec la molette ou à l'aide du bouton dédié). Comme vous pouvez l'observer, le serveur ne vous fournit pas de tuiles pour les niveaux les plus bas. Vous allez donc devoir limiter le nombre de niveaux disponibles. Pour cela, vous allez devoir ajouter 2 paramètres à l'intérieur des accolades du deuxième paramètre de la méthode `tileLayer` (n'oubliez pas d'ajouter une virgule après la ligne `attribution`, juste avant d'inclure ces deux paramètres) :

```
maxZoom: 17,  
minZoom: 9
```

Testez !

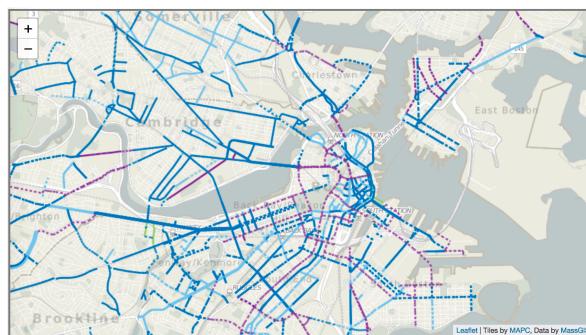
---

<sup>5</sup> <https://www.mass.gov/orgs/massgis-bureau-of-geographic-information>

Il est possible d'ajouter plusieurs couches sur une même carte. Par exemple, ajoutez une couche contenant les pistes cyclables de Boston :

```
L.tileLayer(  
    'http://tiles.mapc.org/trailmap-onroad/{z}/{x}/{y}.png',  
    {  
        maxZoom: 17,  
        minZoom: 9  
    }  
)addTo(map);
```

Vous devriez obtenir la carte suivante :



Vous savez maintenant insérer une carte dans une page Web. Vous allez donc pouvoir y ajouter des informations supplémentaires. Avant de continuer, supprimez la couche des pistes cyclables.

## Exercice 2 : ajouter des points

Lors du cours de sémiologie graphique, nous avons fait la distinction entre **images matricielles(bitmap)** (ensembles de pixels avec une couleur attribuée à chacun d'entre eux) et **images vectorielles** (ensemble de points, ligne, polygones). Ce vocabulaire est celui couramment utilisé par les informaticiens. Pour rappel, la librairie D3 permet de créer des images SVG appartenant à la seconde catégorie. Au cours de l'enseignement d'analyse spatiale, une autre terminologie vous a été présentée, celle employée par les géomaticiens. Dans ce domaine, le mot **raster** désigne ce que les informaticiens appellent des images matricielles. Les images vectorielles sont quant à elles appelées **vecteurs**. La carte chargée dans le premier exercice est une image matricielle (raster). Dans cet exercice, nous allons voir comment y incorporer une image vectorielle (vecteur).

Un format couramment utilisé sur le Web pour décrire un vecteur est le GeoJSON. Il suit la norme JSON tout en y ajoutant une normalisation portant sur la façon dont doivent être déclarées les données géographiques (points, lignes, polygones, coordonnées, autres propriétés). Lealfet et D3 sont conçues pour pouvoir afficher des données de ce format. Commencez par aller sur la page Wikipedia de GeoJSON<sup>6</sup> pour voir quels en sont les principales caractéristiques.

Récupérez le répertoire data.zip et ouvrez dans un éditeur de texte le fichier rodents.geojson qui se trouve à l'intérieur. Les données sont issues du site de la ville de Boston<sup>7</sup> et représentent les observations de rongeurs dans la ville. Comme vous pouvez le voir, il y en a un grand nombre ! Retrouvez pour chaque observation les éléments spécifiques au format GeoJSON (`type`, `geometry`, `coordinates`...).

<sup>6</sup> <https://fr.wikipedia.org/wiki/GeoJSON>

<sup>7</sup> <https://data.boston.gov/dataset>

Vous allez maintenant charger ce fichier dans votre page Web. Pour cela, différentes techniques existent. La plus basique consiste à utiliser un code JavaScript comme nous l'avons fait en cours de sémiologie graphique. L'inconvénient de cette méthode est la lourdeur du code. La seconde méthode consiste à utiliser la fonction D3 appropriée comme dans les TP précédents. Cependant, puisque nous n'avons pas besoin des autres fonctionnalités de D3, il paraît plus judicieux d'utiliser une librairie plus légère comme jQuery<sup>8</sup>, que beaucoup d'entre vous ont déjà manipulé. Pour cela, vous devez commencer par télécharger la dernière version de la librairie<sup>9</sup> et la charger dans votre page HTML.

Voici ensuite la syntaxe pour charger le fichier :

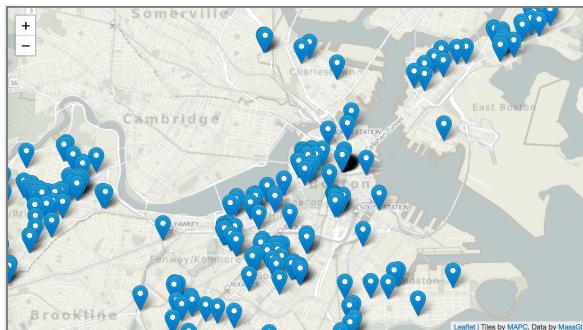
```
$.ajax({
    url: "rodents.geojson", Nom du fichier à charger.
    dataType: "json", Type du fichier à charger.
    success: function(rodents){
        Le code à réaliser lors de la réception du fichier se place ici.
        Le contenu du fichier se trouve dans la variable rodents.
    }
});
```

Chargez le fichier rodents.geojson en dessous du code précédent et affichez-le dans la console d'erreur JavaScript à l'aide de la commande `console.log(rodents);`. Vérifiez ainsi que le fichier est bien chargé et supprimez ensuite l'affichage dans la console.

Il ne vous reste maintenant plus qu'à ajouter une couche contenant les données GeoJSON à votre carte :

```
L.geoJson(rodents).addTo(map);
```

Voici ce que vous devriez obtenir :



La méthode `tileLayer()` permet donc de créer une couche raster, et la méthode `geoJson()` une couche vecteur.

Comme vous pouvez l'observer, par défaut, Leaflet affiche un marqueur bleu. Il est cependant possible de modifier ce marqueur par exemple en le remplaçant par une image. Vous allez le faire avec le gif animé se trouvant dans data.zip : rat.gif. Pour cela, vous devez d'abord créer une icône Leaflet que vous allez stocker dans une variable :

```
var ratIcon = L.icon({
    iconUrl: 'rat.gif', Image de l'icône
    iconSize: [25,25] Taille de l'icône en pixels
});
```

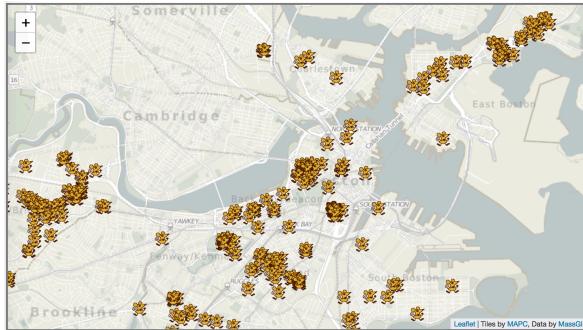
<sup>8</sup> <https://jquery.com/>

<sup>9</sup> <https://jquery.com/download/>

Il faut maintenant préciser à la couche vecteur que cette icône doit être utilisée comme marqueur. Pour cela, il faut ajouter le paramètre suivant à la méthode geoJson, juste après le nom de la variable contenant les données (n'oubliez pas de mettre une virgule entre les deux) :

```
{ pointToLayer: function(feature,latlng){  
    var marker = L.marker(latlng,{icon: ratIcon});  
    return marker;  
}}
```

Ce code permet de dire que les points seront représentés à l'aide d'un marqueur basé sur l'icône `ratIcon` que vous avez créé précédemment. Testez, vous devriez obtenir l'image suivant :



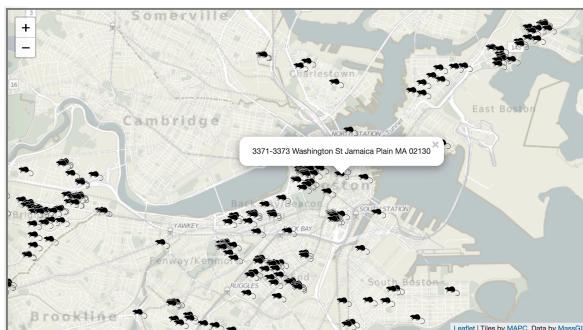
Les rats dansants sont rigolos mais un peu perturbants. Avant de passer à l'exercice suivant, remplacez `rat.gif` par `rat.png`, qui se trouve aussi dans `data.zip`.

### Exercice 3 : détail à la demande (points)

Leaflet permet d'ajouter très simplement une infobulle sur des points lorsque l'utilisateur clique dessus. Pour cela, il va falloir ajouter cette infobulle à votre marqueur de la façon suivante, juste après avoir créé la variable `marker` :

```
marker.bindPopup(feature.properties.Location);
```

Ce code permet d'afficher dans l'infobulle le champs `Location` du champs `properties` de l'élément (`feature`) courant. Regardez dans le fichier `rodents.geojson`, ce champs contient l'adresse où l'observation du rongeur a été faite. Maintenant, lorsque vous cliquez sur un rat, vous voyez apparaître cette adresse dans une infobulle :

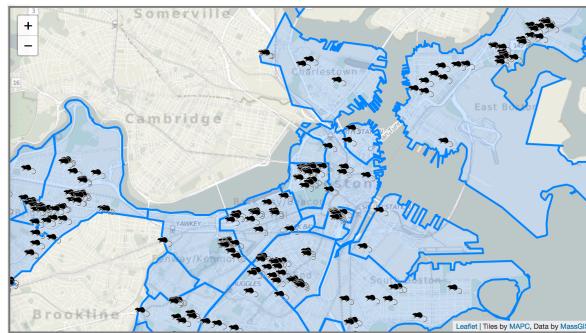


Modifiez le marqueur de façon à faire apparaître le contenu du champ `OPEN_DT` à la suite de l'adresse (il contient la date de l'observation).

#### Exercice 4 : ajouter des polygones

Ouvrez le fichier neighborhoods.geojson dans un éditeur de texte. Comme vous pouvez l'observer, contrairement au fichier précédent, il ne contient pas des points mais des polygones. Ces polygones correspondent aux différents quartiers de la ville de Boston.

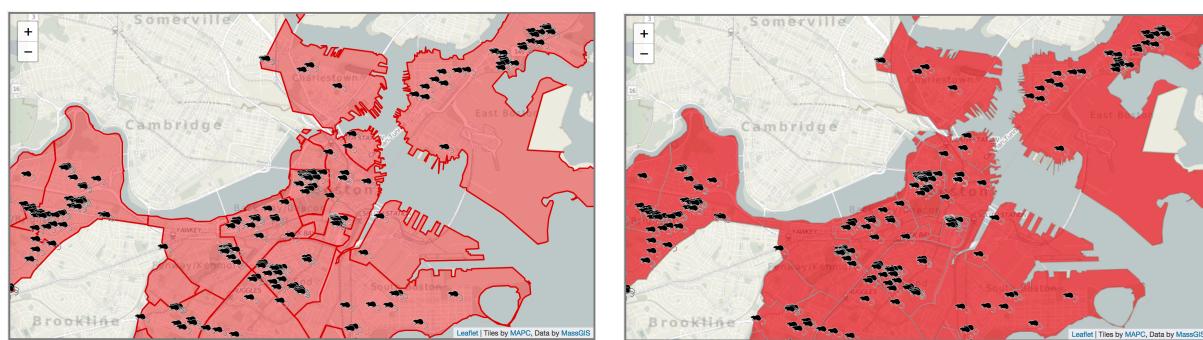
Ajoutez une couche à votre carte de façon à voir ces polygones, de la même façon que vous avez ajouté la couche contenant les points. Vous devriez obtenir le résultat suivant :



Vous allez maintenant modifier la couleur des quartiers en fonction de la propriété `density` présente dans le fichier neighborhoods.geojson. Pour cela, dans la couche des polygones, commencez par ajouter le paramètre suivant à la méthode `geoJSON`, juste après le nom de la variable contenant les données (n'oubliez pas de mettre une virgule entre les deux) :

```
{  
    style: function(feature){  
        return { color: "#CC0000", Couleur de la bordure du polygone (ici rouge)  
                weight: 2, Epaisseur de la bordure (ici rouge)  
                fillColor: "#CC0000", Couleur de l'intérieur du polygone (ici rouge)  
                fillOpacity: 0.4 Opacité de l'intérieur du polygone  
            };  
    }  
}
```

Le style d'un polygone dépend ainsi d'une fonction prenant en paramètre ce polygone. Voici ce que vous devez obtenir (image de gauche) :

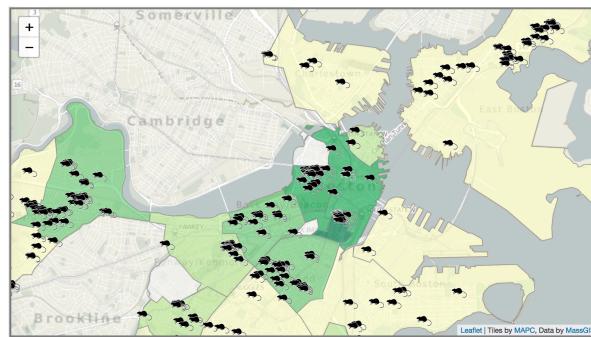


Avant d'aller plus loin, testez différentes valeurs pour les propriétés `color`, `weight`, `fillColor` et `fillOpacity`. Une fois vos tests réalisés, fixez `color` à "`#999999`", `weight` à 1 et `fillOpacity` à 0.6 (image de droite ci-dessus).

Vous allez maintenant attribuer une couleur aux polygones en fonction de la valeur `density`. Pour cela, vous allez créer une variable `fc` contenant la couleur dans la fonction :

```
var fc;
if ( feature.properties.density > 80 )
    fc = "#006837";
else if ( feature.properties.density > 40 )
    fc = "#31a354";
else if ( feature.properties.density > 20 )
    fc = "#78c679";
else if ( feature.properties.density > 10 )
    fc = "#c2e699";
else if ( feature.properties.density > 0 )
    fc = "#ffffcc";
else
    fc = "#f7f7f7";
```

Il ne vous reste plus qu'à passer cette variable comme valeur de la propriété `fillColor` dans le return de la fonction pour obtenir la carte suivante :

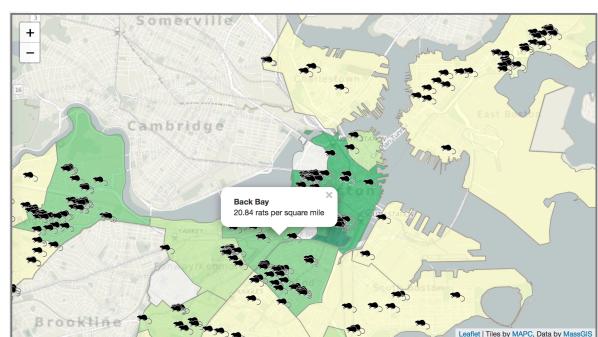
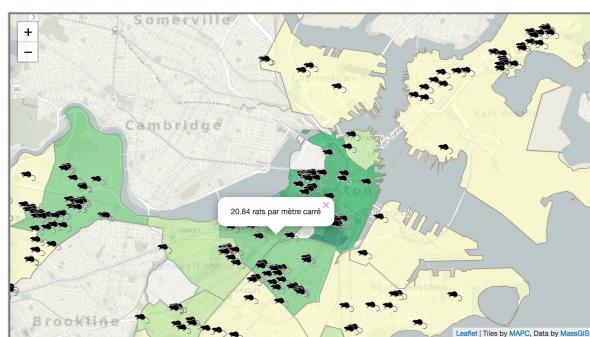


### Exercice 5 : détail à la demande (polygones)

Comme vous l'avez fait précédemment pour les points, vous allez maintenant ajouter une infobulle sur les polygones (juste après le style, n'oubliez pas la virgule après l'accolade du style) :

```
onEachFeature: function( feature, layer ){
    layer.bindPopup( feature.properties.density + " rats par mètre
                    carré" )
}
```

Vérifiez que vous obtenez bien la figure de gauche ci-dessus lorsque vous cliquez sur un quartier. Ajoutez ensuite le champ `Name` présent dans le fichier `neighborhoods.geojson` à vos infobulles en gras (balise `<strong>`) et suivi d'un retour à la ligne (balise `<br />`). Vous devriez obtenir la même image que la figure de droite ci-dessous.



## Exercice 6 : zoom sémantique

Un problème classique en visualisation apparaît lorsque l'on veut représenter un trop grand nombre de points sur une carte : il devient difficile de distinguer les différences de densité, e.g. vous ne verrez pas la différence entre 1000 et 10000 points si ceux-ci sont placés dans une zone très réduite. Pour résoudre ce problème, il est possible de mettre en place un zoom séquentiel agrégeant les points lorsque le niveau de zoom est bas en précisant, pour chaque cluster ainsi formé, le nombre de points qui y ont été agrégés. Une chance, il existe un plugin Leaflet qui le fait pour vous !

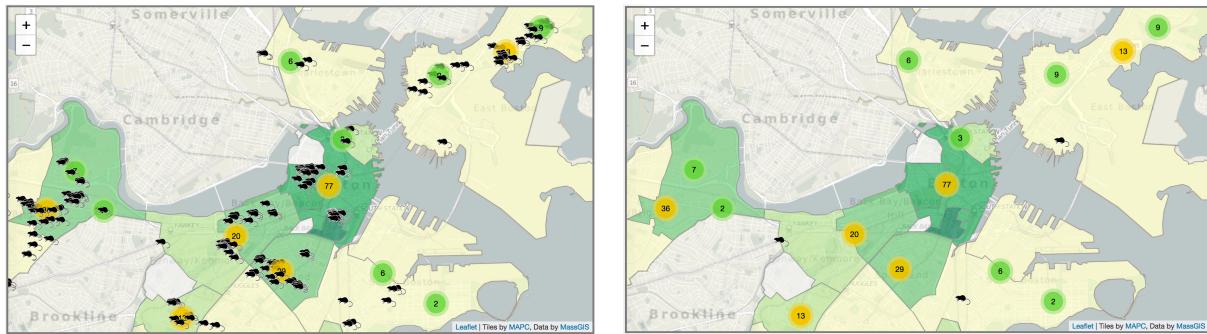
Téléchargez ce plugin<sup>10</sup> dans votre répertoire de travail et chargez dans votre page HTML les fichiers js et css du sous-répertoire dist :

```
<link rel="stylesheet" href="Leaflet.markercluster/dist/  
MarkerCluster.css" />  
<link rel="stylesheet" href="Leaflet.markercluster/dist/  
MarkerCluster.Default.css" />  
<script src="Leaflet.markercluster/dist/leaflet.markercluster.js">  
</script>
```

Ensuite, il suffit de stocker la couche des points dans une variable et ajouter les clusters :

```
var ratLayer = L.geoJson(...).addTo(map);  
  
var clusters = L.markerClusterGroup();  
clusters.addLayer(ratLayer);  
map.addLayer(clusters);
```

Vous devriez obtenir la carte de gauche ci-dessous. Changez le niveau de zoom et observez comment les points sont agrégés/désagrégés.



Pour finir, il suffit de ne plus ajouter la couche `ratLayer` à la carte en supprimant la méthode `addTo(map)` à la fin de la création de la couche. Votre code doit maintenant ressembler à ça et vous devez observer l'image de droite ci-dessus :

```
var ratLayer = L.geoJson(...);  
  
var clusters = L.markerClusterGroup();  
clusters.addLayer(ratLayer);  
map.addLayer(clusters);
```

Dans votre navigateur, modifiez le niveau de zoom et observez comment les clusters se désagrègent pour finalement afficher les points à des niveaux élevés.

---

<sup>10</sup> <https://github.com/Leaflet/Leaflet.markercluster/tree/v1.4.1>

## Exercice 7 : agrégation

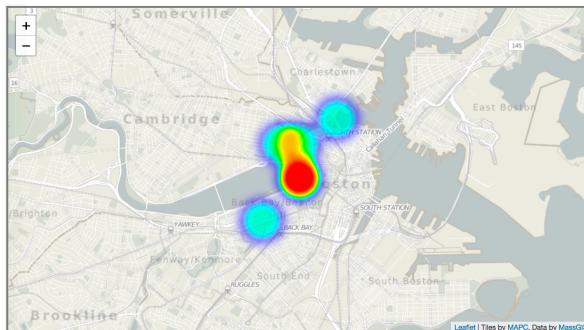
Une alternative permettant de mieux visualiser la densité des points, surtout quand le niveau de zoom est très bas, consiste à les remplacer par une carte de chaleur. Vous allez donc dans cet exercice commencer par supprimer les couches des points et des polygones ainsi que les chargements des deux fichiers GeoJSON.

Vous allez maintenant pouvoir ajouter une nouvelle couche contenant la carte de chaleur. Commencez par chercher et télécharger le plugin Leaflet.heat accessible à partir de la page des plugins leaflet<sup>11</sup> (comme vous pouvez le constater, il existe d'autres plugins pour réaliser des cartes de chaleur, n'hésitez pas à les tester quand vous aurez fini l'exercice). Chargez le fichier dist/leaflet-heat.js dans votre page HTML.

Voici un code permettant d'ajouter une carte de chaleur :

```
var heat = L.heatLayer(Création de la couche
  [
    [42.36, -71.07, 10],
    [42.357, -71.07, 10],
    [42.37, -71.06, 10],
    [42.365, -71.07, 10],
    [42.365, -71.075, 10],
    [42.35, -71.08, 10]
  ],
  {
    radius: 35Rayon des cercles colorés
  }
);
map.addLayer(heat);Ajout de la couche à la carte
```

Ajoutez cette carte de chaleur à votre fichier HTML, voici ce que vous devriez obtenir :



Faites varier le niveau de zoom et observez comment la carte de chaleur est modifiée.

Il est souvent utile de créer le tableau de points séparément afin de le remplir avec un fichier extérieur, donc modifiez le code précédent de la façon suivante :

```
var heatPoints = [
  [42.36, -71.07, 10],
  [42.357, -71.07, 10],
  [42.37, -71.06, 10],
  [42.365, -71.07, 10],
  [42.365, -71.075, 10],
  [42.35, -71.08, 10]
];
```

<sup>11</sup> <https://leafletjs.com/plugins.html>

```

var heat = L.heatLayer(
  heatPoints,
  {
    radius: 35
  }
);
map.addLayer(heat);

```

Vous allez maintenant voir comment charger les points contenus dans le fichier rodents.geojson. Commencez par charger le fichier à l'aide de la fonction jQuery appropriée (cf. exercice 2). Placez le code précédent dans la fonction appelée à la réception des données (`success`), et vérifiez que votre carte de chaleur s'affiche toujours.

Modifiez la variable `heatPoints` de façon à la remplir avec les données chargées :

```

var heatPoints = rodents.features.map(function(rat) {
  var location = rat.geometry.coordinates.reverse();
  location.push(1);
  return location;
});

```

La méthode `features.map()` permet de créer un tableau contenant, pour chaque observation contenue dans la variable `rodents`, la variable qui est retournée par la fonction qu'elle prend en paramètre. Cette fonction récupère les coordonnées des observations du fichier et les retourne. La méthode `reverse` permet d'inverser ces coordonnées, car si vous regardez dans le fichier, elles sont sous la forme `[longitude, latitude]` et non `[latitude, longitude]`. On ajoute ensuite 1 pour le poids.

Vérifiez que vous obtenez bien la carte suivante :

