Rapport de projet Webmapping

Matthieu AHR, avril 2023

Voici le compte-rendu de mon projet webmapping, qui correspond au sujet EF98 conçu pour les ING 20. Ce projet avait été entamé l'année passée, mais j'ai dû le compléter dans le cadre de mon rattrapage dans le module correspondant.

En premier lieu, je tiens à signaler que, comme l'année dernière, ce projet côté client devait prendre pour base un web service de métadonnées créé dans le projet EF96. Or, je n'ai jamais eu accès au dit webservice, j'ai donc dû récupérer les données par mes propres moyens. Malgré que l'objectif du projet soit donc en grande partie atteint, plusieurs étapes basées sur le lien avec le supposé web service n'ont pas pu être réalisées, et je m'en excuse

Je présenterai tout d'abord un guide utilisateur, puis dans une deuxième partie, j'expliquerai le déroulé du projet ainsi que les principaux éléments de programmation utilisés.

1. Guide d'utilisateur du site web

Afin de faire fonctionner ce site web, seule une connexion à internet est obligatoire, aucune connexion à un autre serveur n'est nécessaire, comme expliqué en introduction. Après avoir donc téléchargé le dossier, il suffit juste d'ouvrir le fichier index.html dans le navigateur de votre choix, et vous devriez voir le site suivant s'afficher :

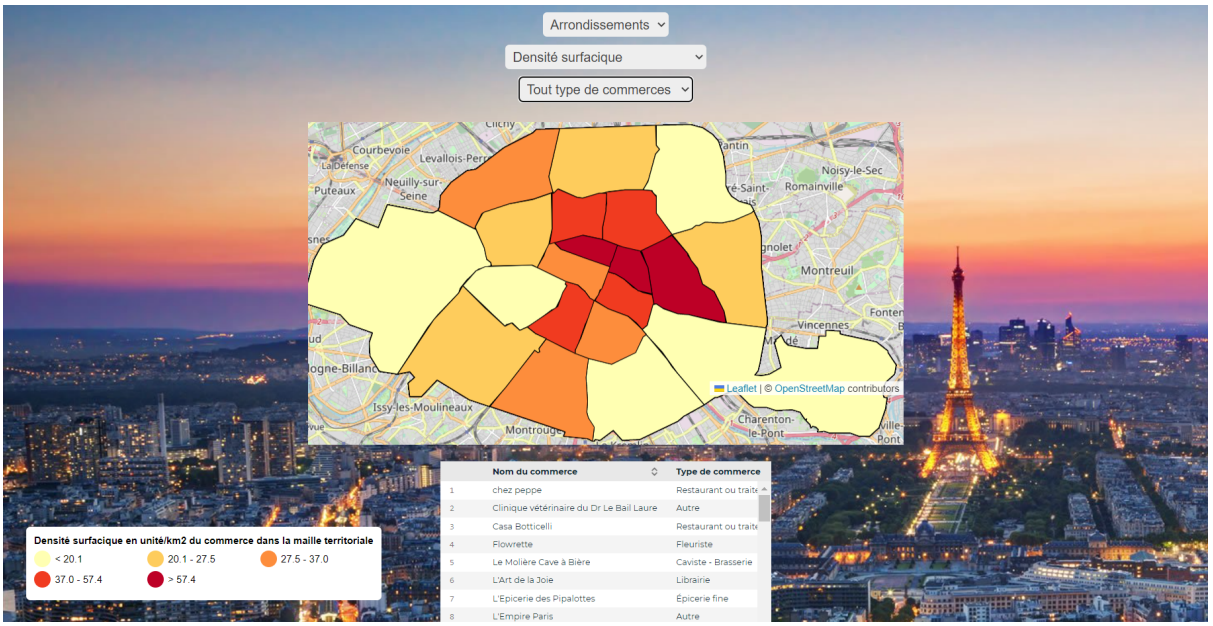

Figure 1 : Interface générale du site web

Ce site est constitué de 4 éléments principaux :

- 3 menus déroulants, servant à sélectionner une maille territoriale (arrondissements ou quartiers), un type de statistique à produire (densité surfacique ou part du commerce dans Paris), et un type de commerce (tous les commerces, les boulangeries-patisseries, les restaurants/traiteurs ou les librairies).
- Une carte leaflet avec fond OSM, fixe et sans outils de navigation, représentant le type de statistique produit à l'échelle de Paris.
- La légende correspondant à la carte, et s'actualisant selon le type de statistique affiché.
- Un tableau regroupant tous les commerces de la base de données, et toutes les informations les concernant.

Au départ, la page s'affiche par défaut sur la statistique de densité surfacique de tous types de commerce pour les mailles d'arrondissement. Il est dès lors possible de changer le type de commerce, ainsi que le type de statistique pour changer l'affichage de la carte et de la légende en conséquence. Pour ce qui est du changement de maille, il est possible de visualiser les quartiers et arrondissements, mais les statistiques sont seulement disponibles pour les arrondissements.

2. Etapes de réalisation et éléments de programmation

L'objectif de ce projet est donc de construire, à partir des données représentant la liste des commerçants proposant un service de retrait ou de livraison à domicile, des représentations choroplèthe représentant des statistiques, sur la base des arrondissements et des quartiers. On souhaite obtenir un outil flexible, qui permette de modifier le maillage territorial, les types de commerce pris en compte, et la nature de la statistique produite.

La première étape de mon projet a consisté à trouver un moyen de récupérer directement les données depuis le site opendata, où sont disponibles à la fois les données des commerces, mais aussi celles correspondant aux mailles territoriales (arrondissements et quartiers). J'ai pour cela utilisé 2 moyens de récupérer ces données. Pour les données de commerce, j'ai fetch l'url contenant les données (voir capture d'écran ci-dessous). Cela permet de simuler ce que j'aurai pu faire si je récupérais le travail du sujet EF96, à la simple différence que je ne fetch pas depuis une base de donnée d'un webservice mais depuis l'API du site opendata Paris.

```
const url_commerces = "https://opendata.paris.fr/api/records/1.0/search/?dat

fetch(url_commerces)
.then(result => result.json())
.then(result => {
```

Figure 2 : Récupération des données de commerce

Cependant pour ce qui est des arrondissements et quartiers, si on essaye de fetch depuis l'API, les géométries ne sont pas gardées. En effet, ces mailles sont représentées en termes de géométrie par des points correspondant certainement aux centroïdes des dites mailles, et pas en polygone.

J'ai donc préféré créer dans un dossier "données", deux fichiers JS comprenant chacun une variable contenant les données d'arrondissement et de quartier, téléchargées en GeoJSON sur le site opendata paris. En appelant ces fichiers JS avant mon fichier principal script.js, je peux facilement récupérer les informations contenues dans les mailles.

Par exemple, pour récupérer la géométrie et le numéro de mes arrondissements, j'ai seulement à effectuer les commandes suivantes dans mon fichier script.js :

```
var geom_arrondissement = arrondissements[0].features[i].geometry.coordinates[0];
var num_arrondissement = arrondissements[0].features[i].properties.c_ar;
```

Figure 3 : Récupération des infos contenues dans les fichiers de mailles territoriales

Au préalable, j'ai construit l'architecture de mon site web dans mon fichier HTML. Le site contient seulement 4 divisions, qui sont les menus, la carte, la légende et le tableau des données. J'appelle ensuite les fichiers JS, en commençant par le package leaflet, puis mes fichiers contenant les données de mailles, et enfin mon fichier principal script.js dans lequel se trouve le programme. Finalement, j'importe les bibliothèques javascript qui me seront utiles, notamment la librairie turf, qui permet de faire des traitements d'analyse spatiale.

Je commence mon fichier script.js en créant la carte comme un objet leaflet, en la centrant sur Paris, en supprimant les options de zoom, de mouvement et les boutons de zoom et en définissant le fond de carte Openstreetmap. Je peux maintenant passer à la partie calcul des valeurs des statistiques, puis ensuite l'affichage des données sur la carte.

a) Calcul des ratios

Je passe ensuite au Fetch des données de commerce, où la plupart de mon code se situera. Afin de calculer pour chaque arrondissement, une valeur de ratio correspondant à la statistique produite, j'ai choisi de créer des dictionnaires associant à chaque arrondissement les valeurs de ratio. Je fais ensuite une double boucle for, qui va tout d'abord parcourir les arrondissements, puis ensuite les commerces. Afin de calculer mes ratios, je crée 2 compteurs pour chaque commerce. Le premier va servir à avoir le nombre total de commerce du type correspondant dans paris, puis le second va vérifier si le commerce se situe bien dans l'arrondissement traité dans la boucle for (Exemple pour les boulangeries ci-dessous).

```
// Fonction renvoyant un booléen qui détermine si un point est dans un polygone ou non
var isInside = turf.booleanPointInPolygon(point_turf, polygon_arr);

// On regarde si le commerce est une boulangerie, si oui on incrémente les compteurs total et inside
if (categorie == "Boulangerie - pâtisserie"){
  compteur_total_boulangerie +=1;

  if (isInside){
    compteur_in_boulangerie += 1;
    }
}
```

Figure 4 : Détermination de l'intersection entre commerces et mailles

Je calcule ensuite mes deux différents ratio, ci dessous l'exemple des boulangeries :

```
var ratio_boulangerie = compteur_in_boulangerie/compteur_total_boulangerie;
var ratio_surf_boulangerie = compteur_in_boulangerie/surface;
```

Le premier correspondant au nombre de commerces dans l'arrondissement par rapport au nombre de commerces total dans paris, et le second à la densité surfacique des commerces dans l'arrondissement. La surface de l'arrondissement est récupérée par la fonction *turf.area(polygon)*. Je remplis enfin les dictionnaires suivant la valeur de ratio calculée.

b) Affichage des résultats sur la page web

Afin de pouvoir afficher des représentations choroplèthes suivant la valeur du ratio, il va falloir séparer chaque statistique calculée dans la partie précédente en quantiles, et affecter une palette de couleurs à ces quantiles. Pour cela, j'ai créé la fonction getQuantileColor qui prend en argument une valeur de ratio, ainsi qu'un dictionnaire. Je transforme le dictionnaire en Array pour plus de facilité de traitement, puis je calcule les quantiles pour chaque statistiques (voir Figure 5). J'ai choisi ici de ne représenter que des quintiles, mais on peut choisir le nombre de quantiles voulus en changeant la variable numQuantiles, ou en la mettant en paramètre de la fonction.

```
var quantile = ratioArray[Math.floor(i * ratioArray.length / numQuantiles)];
```

Figure 5 : Calcul des quintiles du dictionnaire de ratio par arrondissement (ratioArray)

Avec ces valeurs de quantile, je peux maintenant afficher la légende de ma carte. J'ai au préalable créer cette légende dans mon fichier html (<colorbrewer: Color Advice for Maps (colorbrewer2.org)div id="legend">), puis je l'appelle dans mon fichier script.js à l'aide d'un getElementById, et pour chaque type de statistiques, je crée une légende correspondant à l'aide d'un innerHTML, dans lequel j'appelle ma valeur de quantile.

Je multiplie par 1 million la valeur de densité surfacique (pour passer de unité/m2 à unité/km2), et je multiplie par 100 ma valeur de part dans paris, pour passer le ratio en pourcentage, puis je simplifie la valeur au dixième (.toFixed(1)).

Les couleurs utilisées pour la carte et la légende sont issues d'une palette de couleur disponible sur le site ColorBrewer: Color Advice for Maps. J'ai choisi une palette de couleur chaude en simple gradation qui correspond à la représentation. J'assigne ensuite une couleur à chaque quantile, couleur qui sera renvoyée par la fonction.

Maintenant que mes mailles ont une couleur affectée suivant leur valeur de ratio, il faut les afficher sur la carte. Pour cela, je pensais faire une fonction d'affichage relativement simple (voir la fonction affichage_arrdt() dans le code), mais je n'ai pas réussi à la faire fonctionner. J'ai donc dû passer par une fonction beaucoup moins optimisé mais quand même fonctionnelle, qui est basé sur l'identification de l'arrondissement avec une fonction switch case (voir Figure 6). Cette fonction permet de changer le style suivant la valeur retournée par la fonction getQuantileColor expliquée plus tôt. Cependant, vu son optimisation relative, il aurait été compliqué de la faire fonctionner pour les quartiers, qui représentent 80 éléments, donc 80 *cases* de la fonction *switch*, même si cela aurait fonctionné.

```
//Affichage choroplète par arrondissements
{style: function(feature) {
    switch (feature.properties.c_ar) {
      case 1:
        // On associe la couleur de choroplete en fonction de la valeur du ratio
        var couleur_1 = getQuantileColor(dico_ratio["1"], dico_ratio)
        return {
          color: "#000000",
          fillColor: couleur_1,
          weight: 1,
          fillOpacity: 1
        };
```

Figure 6 : Définition du style de la couche Leaflet correspondant à la maille

La dernière étape de la programmation consiste à ajouter les eventListener sur les menus, afin d'afficher les statistiques demandées. On vérifie donc tout d'abord le type de statistiques, puis ensuite le type de commerce pour afficher en conséquence le dictionnaire avec la fonction *affichage_choro()*.