Ecosystème logistique

Judith Bellon, Gabrielle Vernet, César Almecija, Louis-Justin Tallo

juin 29, 2021

Contents:

CHAPITRE 1

Judith Bellon, Gabrielle Vernet, César Almecija, Louis-Justin Tallot

1.1 Dépendances

Ce projet dépend des technologies et ressources suivantes :

- Langages :
  - `HTML 5`, `CSS 3`, et `Javascript`
  - `Python`
  - `C++` compilé avec `g++`
- Formats de fichiers :
  - JSON majoritairement
  - GEOJSON également, parfois plus adapté
  - CSV pour explorer les données de manière plus convéniente avec Excel
- Logiciels :
  - Google Earth pour explorer le terrain de l'Île-de-France et faire des choix géographiques
  - QGIS pour construire les shapefiles à partir des bases de données géographiques
- Ressources, plugins, packages :
  - OpenStreetMap pour les fonds de carte
  - Base OpenData Île de France et notamment les bases de données suivantes :
  - Base Sirene des entreprises et de leurs établissements
  - BAN - Base Adresse Nationale - Paris
  - BAN - Base Adresse Nationale - Hauts-de-Seine
  - BAN - Base Adresse Nationale - Val-de-Marne
  - BAN - Base Adresse Nationale - Val-de-Marne
  - BAN - Base Adresse Nationale - Seine-Saint-Denis
  - Base IRIS pour les contours de l'Île-de-France
  - Base APUR, hydrographie surfacique de l'Île-de-France
- Les plugins Javascript :
  - Leaflet pour insérer des cartes OSM dans les pages web
  - le projet Leaflet/Leaflet.markercluster pour regrouper les points et accélèrer l'affichage
  - le projet pointhi/leaflet-color-markers pour des marqueurs de couleurs variées
- Les packages Python :
  - GeoPandas pour analyser et traiter les données géographiques

  - Folium pour générer des cartes et fichiers `HTML`
  - json pour traiter des fichiers `JSON`
  - ijson pour traiter de manière itérative de lourds fichiers `JSON`
  - time pour mesurer le temps de traitement
  - matplotlib pour analyser les données issues des bases ainsi que visualiser le résultat du clustering
  - Jupyter pour développer de manière plus rapide (supporte même `Folium`)
  - PyQt5 pour réaliser l'interface homme-machine
  - QtWebEngine pour afficher les fichiers `HTML` générés par `Folium` dans l'interface `PyQt`
  - Cython pour compiler certains de nos modules et accélérer notre code
- Les librairies `C++` :
  - `iostream` pour les entrées/sorties
  - `fstream` pour lire/écrire les fichiers
  - `string` pour manipuler les chaînes de caractères
  - `chrono` pour mesurer le temps d'exécution des différentes parties du programme

CHAPITRE 2

Clusterizer

2.1 Fichier principal

src.clusterizer.clusterizer.**calcule_nb_clusters_par_zone**(*liste_df*, *nb_clusters*)

Calcule le nombre de clusters à mettre dans chaque zone pour équilibrer les poids des clusters entre les zones

Paramètres

- **liste_df** – la liste des Dataframe correspondant aux différentes zones
- **nb_clusters** – le nombre de clusters total voulu

Renvoie la liste des nombres de clusters par zones

src.clusterizer.clusterizer.**clusterize**(*df : pandas.core.frame.DataFrame*, *k : int*, *column_geometry : str = 'geometry'*, *is_dict : bool = False*, *weight : bool = True*) → Tuple[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame]

Clusterise à l'aide de l'algorithme des k-moyennes. Attention, fait du en-place.

Paramètres

- **df** – La (Geo)DataFrame contenant les points à clusteriser.
- **k** – Le nombre de clusters à calculer.
- **column_geometry** – A spécifier si la colonne contenant les points n'est pas la colonne par défaut (« geometry »)
- **is_dict** – Indiquer True si jamais la colonne contenant les points ne contient pas d'objets shapely.geometry.Points, mais un dictionnaire (en général, lorsque le fichier provient d'un GeoJSON)

Renvoie Deux GeoDataFrame. Une première GeoDataFrame entrée contenant une colonne en plus (« cluster ») : celle-ci permet de savoir pour chaque point le numéro du cluster qui lui a été affecté. Une deuxième GeoDataFrame contenant les informations détaillées de chaque cluster : centre de masse (« centroids »), enveloppe convexe (« hulls ») et nombre d'établissements dans le cluster (« taille »)

src.clusterizer.clusterizer.**main_json**(*rayon : int = 8*, *secteur_NAF : List[str] = ['']*, *nb_clusters : int = 50*, *adresse_map : str = 'output/clusterized_map_seine.html'*, *seine_divide : bool = True*, *reduce : bool = False*, *threshold : int = 1000*) → None

Fonction principale à exécuter pour successivement ouvrir la DataFrame contenant les données, nettoyer la DataFrame, filtrer par secteurs NAF, ne garder que les magasins proche du centre de Paris, séparer par la Seine, clusteriser et sauvegarder dans une carte. La répartition entre les secteurs de la Seine est calculée automatiquement.

Paramètres

- **rayon** – le rayon (à partir du centre de Paris).
- **secteur_NAF** – les secteurs NAF à sélectionner.
- **nb_clusters** – le nombre de clusters à calculer.
- **adresse_map** – l'adresse de la carte en sortie.
- **seine_divide** – mettre *True* pour séparer les clusters par la Seine
- **reduce** – mettre `True` pour n'utiliser qu'une version allégée des données (plus rapide).
- **threshold** – nombre de données utilisées si reduce= `True`

Renvoie `None`

src.clusterizer.clusterizer.**nettoyer**(*df : pandas.core.frame.DataFrame*, *reduce : bool = False*, *threshold : int = 1000*, *column_geometry : str = 'geometry'*) → pandas.core.frame.DataFrame

Nettoie la DataFrame. Enlève les na. Si spécifié, ne retient que les premières données de la DataFrame.

Paramètres

- **df** – La DataFrame.
- **reduce** – Si True, ne prend que les premières données.
- **threshold** – Dans le cas où reduce=True, nombre de données à sélectionner.
- **column_geometry** – A spécifier si la colonne contenant les points n'est pas la colonne par défaut (« geometry »)

Renvoie Une DataFrame nettoyée.

src.clusterizer.clusterizer.**save_to_map**(*df_clusters : pandas.core.frame.DataFrame*, *map : Optional[folium.folium.Map] = None*) → folium.folium.Map

Sauvegarde les informations des clusters dans une carte Leaflet. Retourne la carte

Paramètres

- **df_clusters** – La DataFrame contenant les informations de chaque cluster (cf. deuxième sortie de la fonction clusterize)
- **map** – la carte à utiliser si un paramètre est spécifié : réecrit par dessus. si rien n'est spécifié, génère une nouvelle carte

Renvoie une carte complétée.

src.clusterizer.clusterizer.**test_geojson**()

Fonction interne (utilisée pour vérifier le bon fonctionnement de la clusterisation).

src.clusterizer.clusterizer.**test_naf**()

Fonction interne (utilisée pour vérifier le bon fonctionnement du filtrage par NAF).

2.2 Fichiers utilitaires

2.2.1 Utilitaires pour la clusterisation

Ce module permet d'extraire simplement nos données des GeoDataFrames, de trouver leurs coordonnées, de restreindre le calcul aux points situés dans un certain rayon autour de Paris ; il permet également de manipuler les clusters, de calculer leur poids et leur taille.

src.clusterizer.utils.clusterizer_utils.**calculer_poids_cluster**(*df : pandas.core.frame.DataFrame*, *naf_column_name : str*) → int

Calcule le poids d'un ensemble d'établissements.

Paramètres

- **df** – La DataFrame contenant tous les établissements. Rien n'est requis, à part avoir une colonne où sont situés les codes NAF.
- **naf_column_name** – Le nom de la colonne contenant les codes NAF.

Renvoie Le poids du cluster.

src.clusterizer.utils.clusterizer_utils.**calculer_poids_cluster_wrapper**(*naf_column_name : str*) → Callable[[pandas.core.frame.DataFrame, str], int]

Wrappe calculer_poids_cluster pour pouvoir l'utiliser dans un groupby.

Paramètres **naf_column_name** – La colonne où se situent les codes NAF.

Renvoie cf. la fonction calculer_poids_cluster.

src.clusterizer.utils.clusterizer_utils.**calculer_poids_code_NAF**(*code_naf : str*) → int

Calcule le poids d'un code NAF.

Paramètres **code_naf** – Le code NAF à calculer (dans une des deux conventions : avec ou sans points).

Renvoie Le poids du code NAF.

src.clusterizer.utils.clusterizer_utils.**filter_nearby_paris**(*df : pandas.core.frame.DataFrame*, *radius : int*, *column_geometry : str = 'geometry'*, *is_dict : bool = False*) → pandas.core.frame.DataFrame

Filtre les données proches du centre de Paris.

Paramètres

- **df** – la DataFrame à filtrer
- **radius** – le rayon (en kilomètres)
- **column_geometry** – la colonne où se trouvent les données géométriques (par défaut : "geometry")

Renvoie la DataFrame filtrée

src.clusterizer.utils.clusterizer_utils.**get_coords_from_object**(*df : pandas.core.frame.DataFrame*, *column_geometry : str = 'geometry'*, *is_dict : bool = False*) → numpy.ndarray

Récupère les coordonnées des points de la DataFrame.

Paramètres

- **df** – la DataFrame.
- **column_geometry** – la colonne contenant les données géométriques.

— **is_dict** – les données sont-elles en dictionnaire ?

Renvoie les coordonnées sous la forme d'une matrice de deux colonnes (et d'autant de lignes qu'il y a de points)

src.clusterizer.utils.clusterizer_utils.**get_infos_clusters_enveloppes_convexes**(*k : int*, *df : pandas.core.frame.DataFrame*, *column_geometry : str = 'geometry'*, *is_dict : bool = False*) → pandas.core.frame.DataFrame

Fonction permettant de récupérer des infos sur les clusters (enveloppes convexes).

Paramètres

— **k** – Nombre de clusters

— **df** – La DataFrame où l'on a déjà ajouté le numéro des clusters (laissée intacte).

— **column_geometry** – Le nom de la colonne où se situent les données géometriques (par défaut, « geometry »).

— **is_dict** – True si les paramètres sont sous forme de dictionnaire

Renvoie Une GeoDataFrame associant à chaque numéro de cluster son enveloppe convexe.

src.clusterizer.utils.clusterizer_utils.**get_infos_clusters_poids**(*df : pandas.core.frame.DataFrame*, *column_naf_code : str*) → pandas.core.frame.DataFrame

Fonction permettant de récupérer des infos sur les clusters (poids).

Paramètres

— **df** – La DataFrame où l'on a déjà ajouté le numéro des clusters (laissée intacte).

— **column_naf_code** – Le nom de la colonne où se situent les codes NAF.

Renvoie Une nouvelle GeoDataFrame associant à chaque numéro de cluster le poids de celui-ci

src.clusterizer.utils.clusterizer_utils.**get_infos_clusters_taille**(*df : pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame

Fonction permettant de récupérer des infos sur les clusters (tailles).

Paramètres **df** – La DataFrame où l'on a déjà ajouté le numéro des clusters (laissée intacte).

Renvoie Une nouvelle GeoDataFrame associant à chaque numéro de cluster la taille de celui-ci (nombre d'établissements)

src.clusterizer.utils.clusterizer_utils.**swap_xy**(*geom*)

Inverse les coordonnées de l'objet shapely.geometry. Utile pour passer objets shapely dans folium (la convention est inversée). Auteur : https://gis.stackexchange.com/a/291293

Paramètres **geom** – L'objet dont on veut inverser les coordonnées (Point, Polygon, MultiPolygon, etc.)

Renvoie l'objet inversé

2.2.2 Utilitaires pour la gestion des codes NAF

Fonctions pour switcher les conventions de NAF (avec ou sans point intermédiaire)

src.clusterizer.utils.NAF_utils.**ajouter_point**(*code_naf : str*) → Optional[str]
Fait passer le code NAF à la convention avec point (s'il n'y est pas)

Paramètres **code_naf** – Le code à changer

Renvoie Le code avec un point.

src.clusterizer.utils.NAF_utils.**filter_by_naf**(*df : pandas.core.frame.DataFrame*, *codes_naf : List[str]*, *column_codes : str*) → pandas.core.frame.DataFrame
Retourne les établissements dont le code NAF est contenu dans la liste.

Paramètres
- **df** – La liste des établissements (convention NAF : sans le point)
- **codes_naf** – Les codes NAF (avec ou sans le point) (sous forme de liste)
- **column_codes** – La colonne où est située le code NAF dans la DataFrame des établissements

Renvoie La DataFrame filtrée.

src.clusterizer.utils.NAF_utils.**get_NAFs_by_section**(*section : str*) → pandas.core.series.Series
Fournit la liste des codes NAF de la section correspondante.

Paramètres **section** – La lettre de la section

Renvoie La liste des codes NAF contenus dans la section (convention : avec points)

src.clusterizer.utils.NAF_utils.**get_description**(*code_naf : str*) → str
Fournit la description correspondant au code NAF.

Paramètres **code_naf** – le code, avec ou sans point.

Renvoie la description complète.

src.clusterizer.utils.NAF_utils.**retirer_point**(*code_naf : str*) → Optional[str]
Fait passer le code NAF à la convention sans point (s'il y est)

Paramètres **code_naf** – Le code à changer

Renvoie Le code sans point.

2.2.3 Utilitaires pour la séparation par la Seine

src.clusterizer.utils.seine_data_utils.**rapport_a_la_seine_spatial_index_point**(*array_coords : numpy.ndarray*) → numpy.ndarray
Trouve les zones où se situent les points de l'"array fournie. Utilise un *R-Tree* pour ce faire pour accélérer le calcul.

Paramètres **array_coords** (*np.ndarray*) – Array (nb_points, 2) contenant les coordonnées des points

Renvoie Une array « masque » qui a chaque point associe son numéro de zone

Type renvoyé np.ndarray

CHAPITRE 3

Traitement de la base SIRENE

Functions

int **main**()

Traite la base SIRENE (fichier JSON de 1,7 Go) et en extrait les informations utiles.

Renvoie int

CHAPITRE 4

Interface Homme-Machine

4.1 Interface complète

class src.ihm.ihm_complet.**Wind**

Classe contenant l'interface Homme-Machine pour le projet.

appui_bouton_OK() → None

Listener pour le bouton ok. Prépare les données pour lancer la clusterisation et l'affichage de la carte. cf. lancement_clustering

lancement_clustering() → None

Lance la clusterisation à l'aide des paramètres entrés par l'utilisateur. Ensuite, affiche la carte.

4.2 Interaction avec l'utilisateur *via* fichier CSV

Première interface Homme-machine : utilisation d'un tableau *CSV* pour récupérer les informations données par l'utilisateur

Paramètres modifiables dans la fonction `clusterize` : le nombre de clusters

TODO : Paramètres modifiables souhaités en plus : encadrement du nombre de clusters, taille des clusters

4.3 Utilitaire : fenêtre d'accueil pour `ihm_complet`

class src.ihm.ihm_pyqt.**InputFenetre**

Le widget qui permet à l'utilisateur de rentrer les paramètres de clustering

4.4 (Obsolète) Ouverture d'un fichier HTML dans un navigateur Web :

src.ihm.web.**open_html**(*adresse*)

Affichage du html depuis python. Il faut être dans le répertoire ihm pour le lancer.

Paramètres **adresse** – l'adresse du fichier à ouvrir

CHAPITRE 5

Indices and tables

— genindex
— modindex
— search

Index des modules Python

S

Index

T

W