

Package Coberny

A.Bernard, F.Chery, O.Côme



Faculté des sciences de Montpellier

13 Décembre 2021



Sommaire

1 Introduction

- Création de la base de données
- Présentation du package Coberny

2 Carte interactive

- Création de la carte
- Utilisation
- Exemple

3 Distribution des prix

- Création du code
- Utilisation
- Exemple d'utilisation

4 Minimisation coût du trajet



Création de la base de données

Dataframe intermédiaire:

- Pour créer le data nous avons utilisé **pandas** pour sélectionner uniquement les sorties d'autoroute concernées par le projet et enlever les portions gratuites.
- Nous avons utilisé **pyproj** pour transformer les coordonnées L93 en WGS84. Nous avons donc obtenu à la suite un dataframe avec les noms des autoroutes, les noms des péages et les coordonnées GPS.



Création de la base de données

Dataframe des prix

Nous avons simplement reporté le fichier que nous avions en format .csv pour l'utiliser avec **pandas** et choisir les péages voulus. Puis nous avons renommé les colonnes pour être cohérent avec les autres dataframe.

Dataframe des distances

Nous avons utilisé **requests** et **json** pour faire des requêtes de distance entre chaque coordonnées du dataframe créé précédemment. Ces packages utilisent les données de **openstreetmap**.



Présentation du package Coberny

Installer Coberny

```
pip install git+https://github.com/ABernard27/PROJET-groupe-3
```

Ce package permet de réaliser 3 actions primaires :

- Réaliser une carte interactive d'un trajet sur l'autoroute en affichant les noms des stations, le prix entre deux stations et le temps en kilomètres.
- Afficher la distribution des prix entre deux sorties
- Déterminer, en fonction du nombre de sorties acceptées, le trajet le moins coûteux



Création de la carte

La carte repose sur 3 packages essentiels : **openrouteservice**, **json** et **folium**. Il y a une petite manipulation en plus pour exécuter cette fonction : créer une clé API sur *openrouteservice.org*.



Création de la carte

La carte repose sur 3 packages essentiels : **openrouteservice**, **json** et **folium**. Il y a une petite manipulation en plus pour exécuter cette fonction : créer une clé API sur openrouteservice.org.

Le code créé exécute des requêtes avec les points de coordonnées entrés pour relier tous ses points. L'API de direction renvoie des polylignes codés (série de coordonnées sous la forme d'une seule chaîne), à l'aide de `convert.decode polyline` on peut les décoder.

Création de la carte

La carte repose sur 3 packages essentiels : **openrouteservice**, **json** et **folium**. Il y a une petite manipulation en plus pour exécuter cette fonction : créer une clé API sur openrouteservice.org.

Le code créé exécute des requêtes avec les points de coordonnées entrés pour relier tous ses points. L'API de direction renvoie des polylignes codés (série de coordonnées sous la forme d'une seule chaîne), à l'aide de `convert.decode_polyline` on peut les décoder.

Ensuite `folium.GeoJson(decoded).add_to(m)` ajoute à la carte les points reliés. `summary` permet de récupérer les distances dans le fichier json pour les afficher. Enfin, `folium.Marker` permet d'ajouter les marqueurs.



Utilisation

Pour utiliser cette fonction les dataframe doivent être de la forme suivante : (extrait des tableaux)

	MONTPELLIER	SETE	AGDE
0	0.0	1.6	3.6
1	1.6	0.0	1.9
2	3.6	1.9	0.0

Extrait tableau des prix



Utilisation

Pour utiliser cette fonction les dataframe doivent être de la forme suivante : (extrait des tableaux)

	MONTPELLIER	SETE	AGDE
MONTPELLIER	0.0	17.0	41.0
SETE	17.0	0.0	26.0
AGDE	41.0	26.0	0.0

Extrait tableau des distances



Exemple

Par exemple pour utiliser la fonction carte il suffit d'importer **Coberny** puis de taper la commande suivante :

Exemple d'utilisation

```
Coberny.carte(np.column_stack([data['x'],data['y']]), data[' Nom  
gare' ], 'Key', prix)
```



Exemple

Par exemple pour utiliser la fonction carte il suffit d'importer **Coberny** puis de taper la commande suivante :

Exemple d'utilisation

```
Coberny.carte(np.column_stack([data['x'],data['y']]), data[' Nom  
gare' ], 'Key', prix)
```

Ici data est un dataframe contenant les noms des péages ainsi que leur coordonnée (longitude, latitude), Key est la clé openrouteservice, et prix est le dataframe contenant les prix entre toutes les sections.



Exemple

Par exemple pour utiliser la fonction carte il suffit d'importer **Coberny** puis de taper la commande suivante :

Exemple d'utilisation

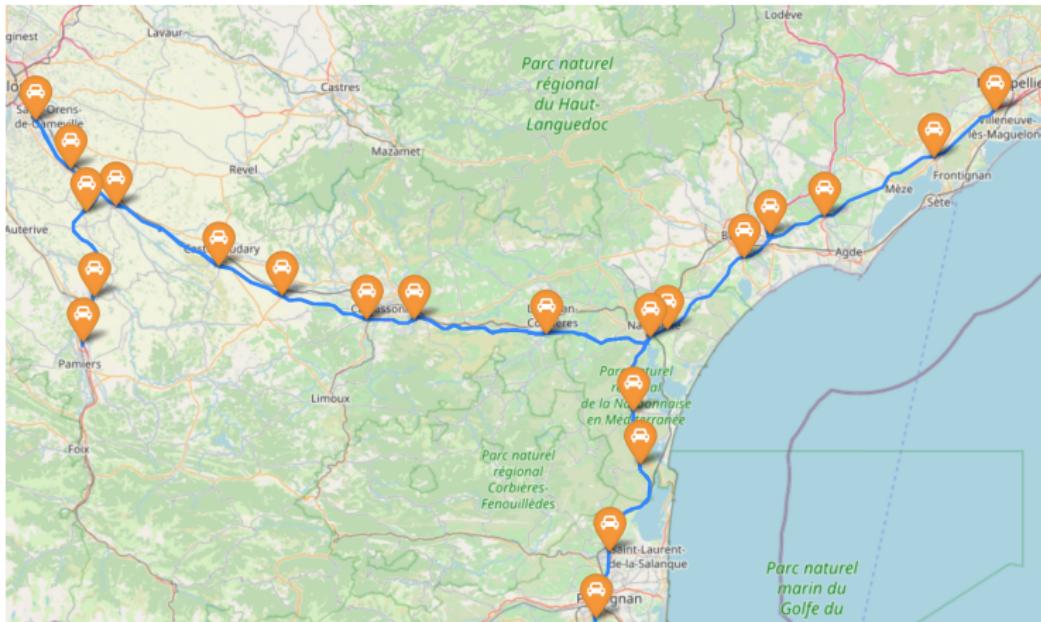
```
Coberny.carte(np.column_stack([data['x'],data['y']]), data[' Nom  
gare' ], 'Key', prix)
```

Ici data est un dataframe contenant les noms des péages ainsi que leur coordonnée (longitude, latitude), Key est la clé openrouteservice, et prix est le dataframe contenant les prix entre toutes les sections.

Après avoir exécuter ce code nous obtenons la carte suivante où l'on peut cliquer sur les marqueurs et les portions de route :



Exemple



COBERNY

Code

Structure de la classe **Distribution**

- Une fonction **Graph** qui contient deux définitions, et qui permet de visualiser le graphe du trajet.



Code

Structure de la classe **Distribution**

- Une fonction **Graph** qui contient deux définitions, et qui permet de visualiser le graphe du trajet.
- Définition du Kernel Density Estimation : l'estimation par noyau est une méthode d'estimation de la densité de probabilité d'une variable aléatoire.



Code

Structure de la classe **Distribution**

- Une fonction **Graph** qui contient deux définitions, et qui permet de visualiser le graphe du trajet.
- Définition du Kernel Density Estimation : l'estimation par noyau est une méthode d'estimation de la densité de probabilité d'une variable aléatoire.
- Définition du Diagramme en bâton : représentation graphique de données à caractères discrets.



Code

Structure de la classe **Distribution**

- Une fonction **Graph** qui contient deux définitions, et qui permet de visualiser le graphe du trajet.
- Définition du Kernel Density Estimation : l'estimation par noyau est une méthode d'estimation de la densité de probabilité d'une variable aléatoire.
- Définition du Diagramme en bâton : représentation graphique de données à caractères discrets.
- Le graphe du trajet avec le package **networkx**.



Code

Structure de la classe **Distribution**

- Une fonction **Graph** qui contient deux définitions, et qui permet de visualiser le graphe du trajet.
- Définition du Kernel Density Estimation : l'estimation par noyau est une méthode d'estimation de la densité de probabilité d'une variable aléatoire.
- Définition du Diagramme en bâton : représentation graphique de données à caractères discrets.
- Le graphe du trajet avec le package **networkx**.

Nous avons utilisé la fonction **interact** du package **ipywidget**, afin de tracer le KDE et le diagramme avec des widgets interactifs.



Code

Pour créer ces deux fonctions, nous avons eu besoin d'étapes intermédiaires :

- Récupération du trajet entre l'entrée et la sortie avec **networkx et les graphes**.
- Construction du vecteur des prix au kilomètre par portion d'autoroute.



Code

Pour créer ces deux fonctions, nous avons eu besoin d'étapes intermédiaires :

- Récupération du trajet entre l'entrée et la sortie avec **networkx et les graphes**.
- Construction du vecteur des prix au kilomètre par portion d'autoroute.

De plus, pour le bon fonctionnement du package nous avons créer une fonction **indice**.



Utilisation

Pour l'utilisation de cette classe, il nous faudra des dataframes de la même forme que pour la carte interactive.



Utilisation

Pour l'utilisation de cette classe, il nous faudra des dataframes de la même forme que pour la carte interactive.

Avec la structure de classe, la commande de code est à utiliser sous cette forme là :



Utilisation

Pour l'utilisation de cette classe, il nous faudra des dataframes de la même forme que pour la carte interactive.

Avec la structure de classe, la commande de code est à utiliser sous cette forme là :

Exemple d'utilisation:

```
Coberny.distribution(dataframe distances, dataframe prix).Graph()
```



Exemple d'utilisation

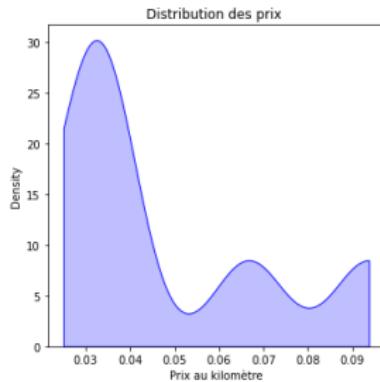
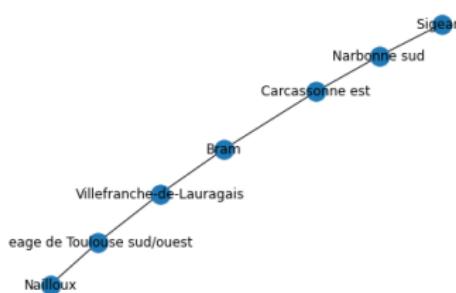
Exemple d'utilisation avec nos dataframes :

Coberny.distribution(Distance, Prix).Graph()

Lors de l'exécution de ce code, nous obtenons :

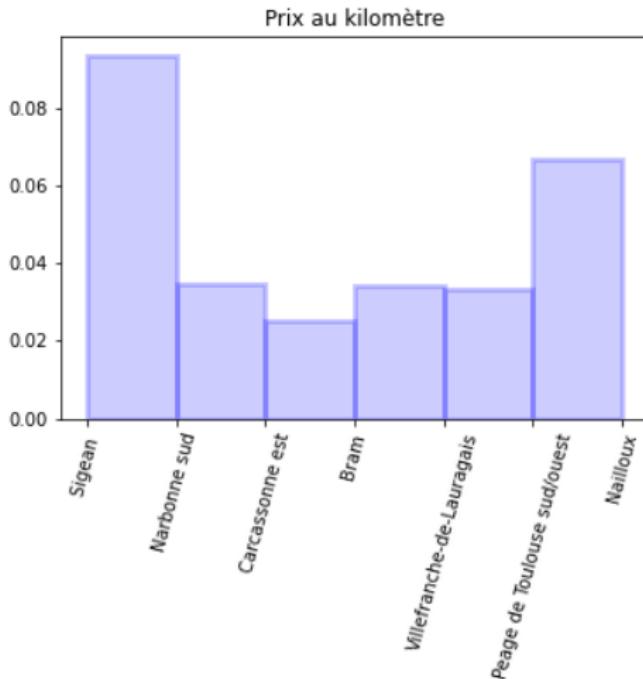
bw 0.42

Entrée	Sigean
Sortie	Nailloux



Entrée ▾

Sortie ▾



Minimisation coût du trajet

L'algorithme

L'algorithme permettant de déterminer le trajet le moins couteux en fonction du nombre de sorties acceptées repose sur 3 packages essentiels:

- **pandas:** Librairie utile afin de manipuler et d'analyser les données.
- **NetworkX:** Librairie pour le traitement de graphs.
- **itertools:** Module proposant un grand nombre d'itérateurs.

L'algorithme possède deux fonctions principales:

- **FindBestPathForPrice** qui renvoie un couple contenant:
 - La liste des péages par lesquelles il faudra passer pour payer le moins cher (il s'agit du trajet optimal).
 - Le prix du trajet optimal.
- **CreateGraphOfBestPathForPrice** qui trace le graph du trajet optimal.
 - La ville de départ et celle d'arrivée sont coloriées en bleu.
 - Les sorties intermédiaires sont coloriées en oranges.



Minimisation coût du trajet

Prérequis pour utilisation

Pour que l'algorithme puisse fonctionner, il faut que les données de l'utilisateur soient dans un dataframe ayant la forme suivante:

Extrait tableau des prix

	MONTPELLIER	SETE	AGDE	...
MONTPELLIER	0.0	1.6	3.6	...
SETE	1.6	0.0	1.9	...
AGDE	3.6	1.9	0.0	...
:



Minimisation coût du trajet

Par exemple, pour utiliser la fonction **FindBestPathForPrice**, il suffit d'importer **Coberny** et d'utiliser la commande suivante:

Exemple d'utilisation

```
Coberny.FindBestPathForPrice(  
    data, ville_depart, ville_arrivee, nbr_sorties_max  
)
```

Vous remarquerez que cette fonction possède 4 arguments:

- **Data:** Le dataframe (avec le format adéquat) dans lequel seront stockées les données de l'utilisateur.
- **ville_depart:** La ville de départ choisie par l'utilisateur.
- **ville_arrivee:** La ville d'arrivée choisie par l'utilisateur.
- **k:** Le nombre maximal de sorties intermédiaires souhaité par l'utilisateur.



Minimisation coût du trajet

Exemple de sortie obtenue

Voici un exemple de ce que retourne la fonction

FindBestPathForPrice en prenant comme argument:

- Le dataframe des prix.
- Le péage de Saint-Jean-de-Védas comme lieu de départ.
- Le péage de Nailloux comme lieu d'arrivée.
- Le nombre de sorties maximal autorisé $k = 5$.

```
Couple meilleur chemin et prix:  (['St-Jean-de-Vedas', 'Sete', 'Agde Pezenas', 'Narbonne sud', 'Nailloux'], 18.5)
Le temps d'exécution du programme vaut:  48.82379126548767  secondes.
cad  0:00:48.823791  dans le format heures minutes secondes
```

Figure: Le trajet optimal avec son coût



Minimisation coût du trajet

Par exemple, pour utiliser la fonction **CreateGraphOfBestPathForPrice**, il suffit d'importer **Coberny** et d'utiliser la commande suivante:

Exemple d'utilisation

```
Coberny.CreateGraphOfBestPathForPrice(  
    data, ville_depart, ville_arrivee, nbr_sorties_max  
)
```

Les arguments de cette fonction sont les même que ceux de la fonction **FindBestPathForPrice**.

Exemple de sortie obtenue

Voici un exemple de ce que retourne la fonction **CreateGraphOfBestPathForPrice** en prenant les mÃªme arguments que ceux utilisÃ©s dans la slide précédente.

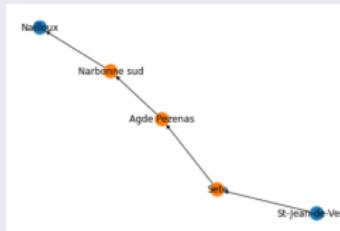


Figure: Graph du trajet optimal



Minimisation coût du trajet

Utilisation de l'interface utilisateur

Nous avons crée une interface utilisateur à partir du fichier **best_price_pathUI.py**. Ce dernier utilise les fonctions du fichier **bestPricePathForUI.py**. Cette interface va permettre à l'utilisateur de sélectionner ses données puis de charger les villes de départ. Ensuite, il pourra directement via des menus déroulants, sélectionner la ville de départ, la ville d'arrivée ainsi que le nombre de sorties maximal souhaité (Cela permet d'éviter des erreurs de saisies). Enfin, il pourra exécuter l'algorithme en cliquant sur le bouton "*Trouver le meilleur trajet au meilleur prix*".



Minimisation coût du trajet

Affichage de l'interface utilisateur et exécution de l'algorithme pour le trajet de Sete à Sigean avec $k = 4$:

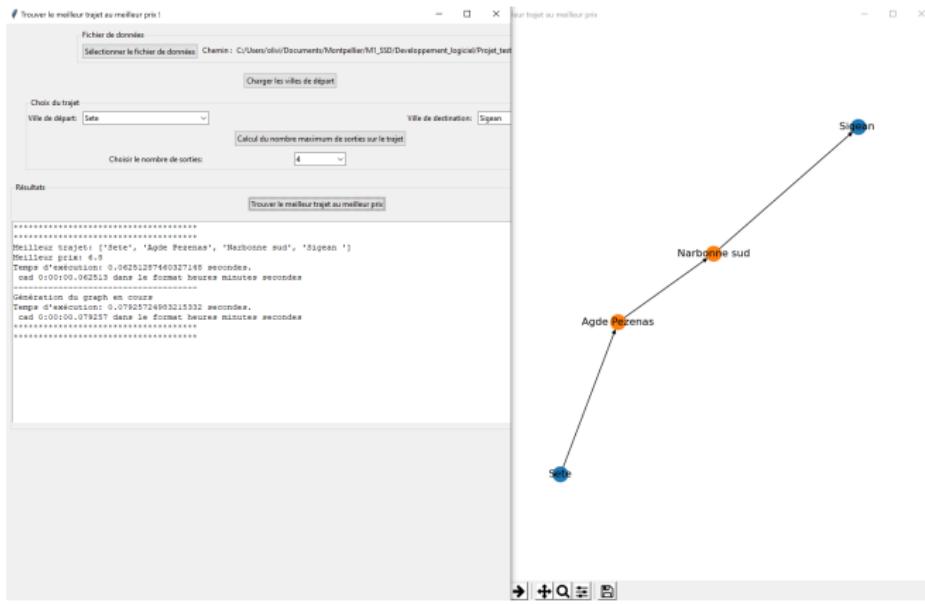


Figure: Interface utilisateur



