

Le projet ASLTAM

BOUARROUDJ, CAPEL, CARVAILLO



UNIVERSITÉ
DE MONTPELLIER

12 décembre 2021

Introduction

Ce beamer est une brève présentation du package ASLTAM.



Introduction

Ce beamer est une brève présentation du package ASLTAM.



Ce projet est né d'une volonté d'étudier la politique des prix d'une compagnie d'autoroute (ASF) dans le sud de la France.



Introduction

Une documentation est disponible pour plus de détail concernant les modules requis, et les procédures d'installation.

ASLTAM

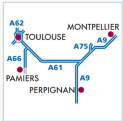
Search docs

- ASLTAM
- Récupération des données
- Structure des données
- Guide d'installation de chef API

Documentation du programme ASLTAM

Bienvenue dans la documentation du programme ASLTAM ! Un package très utile de modélisation statistique.

Présentation



Le nom **asltam** provient de l'acronyme ASL (une mauvaise lecture de ASF, pour Autoroute du Sud de la France) et des premières lettres des créateurs du package : Thomas CARVAILLO, Alexandre CAPEL et Abdelmalek BOUARROUDJ (appelé aussi Malek). Ce module python a été conçu pour répondre à une problématique sur l'étude de la politique des prix de la compagnie Vinci, en particulier dans la région Occitanie. Cependant, il est possible pour vous d'utiliser ce package à votre guise, pour répondre à un de vos problèmes.

Procédure d'installation

Introduction

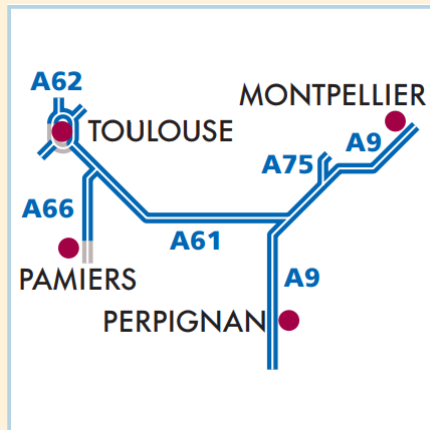


Fig 0.1. Réseau routier sur lequel se base nos exemples.

Plan de la présentation

- 1 Récupération et structure des données
 - Données géographiques
 - Matrices des distances/prix
- 2 Carte interactive
- 3 Distribution
- 4 Graphe
 - La fonction *get_way*
 - L'algorithme des meilleurs sorties

1 Récupération et structure des données

- Données géographiques
- Matrices des distances/prix

2 Carte interactive

3 Distribution

4 Graphe

- La fonction *get_way*
- L'algorithme des meilleurs sorties

1 Récupération et structure des données

- Données géographiques
- Matrices des distances/prix

2 Carte interactive

3 Distribution

4 Graphe

- La fonction *get_way*
- L'algorithme des meilleurs sorties

Récupération et structure des données

Format des données géographique

On a choisi un format WGS84 (epsg :4326).

Récupération et structure des données

Format des données géographique

On a choisi un format WGS84 (epsg :4326).

Voici un extrait du tableau :

Gares de péage	Latitude	Longitude
Montpellier	43.56321197997598	3.832152851175532
Sete	43.48122735717297	3.683459688948031
Agde	43.37686025015689	3.4158808248192942
Beziers Cabrials (Sortie)	43.34325721988552	3.2885692228136043
Beziers Ouest	43.303664399696856	3.2229492234270127

Fig 1.1. Extrait du tableau des coordonnées

1 Récupération et structure des données

- Données géographiques
- Matrices des distances/prix

2 Carte interactive

3 Distribution

4 Graphe

- La fonction *get_way*
- L'algorithme des meilleurs sorties

Récupération et structure des données

- class `load_dist` et class `load_price`
- Le tableau de données est symétrique et de diagonale nulle

	x1	x2	...	xn
x1	0	a1 2	...	a1 n
x2	a1 2	0	...	a2 n
...
xn	a1 n	a2 n	...	0

Fig 1.2. Tableau des distances/prix

- Compatibilité

1 Récupération et structure des données

- Données géographiques
- Matrices des distances/prix

2 Carte interactive

3 Distribution

4 Graphe

- La fonction *get_way*
- L'algorithme des meilleurs sorties

Module Map

Ce module contient uniquement la fonction *trajet* et permet l'affichage d'une carte *folium* interactive. Il nécessite les packages suivant pour fonctionner

- folium
- openrouteservice
- json
- requests

trajet()

Cette fonction prend en argument :

- une gare de départ et d'arrivée
- un dataframe des prix, un des distances et un des coordonnées géographiques
- une clef API

Exemple extrait de *script.py* avec *interact*

On utilise le package *ipywidgets*.

```
#%% Construction de la carte interactive
interact(atm.trajet,
         DEPART=list(geo.index),
         ARRIVEE=list(geo.index),
         data_geo=fixed(geo),
         data_price=fixed(prix),
         data_dist=fixed(dist),
         KEY='5b3ce3597851110001cf62486f5564a064e34f3895221e5a0d9a2405')
```

Code 2.1 Code pour ouvrir la carte interactive dans un notebook

Résultat de la carte interactive

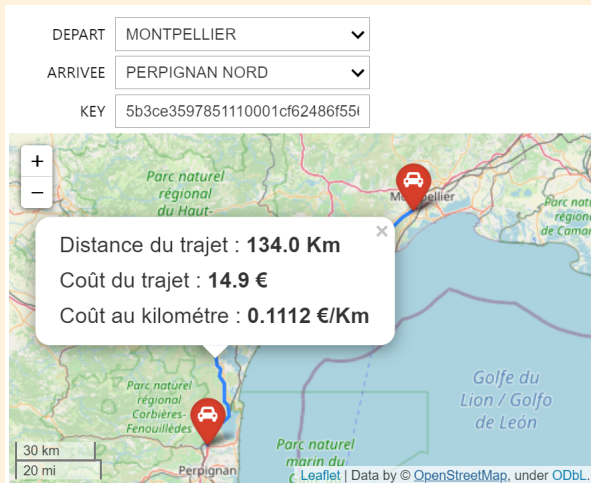


Fig 2.1 Image illustrative de la carte

1 Récupération et structure des données

- Données géographiques
- Matrices des distances/prix

2 Carte interactive

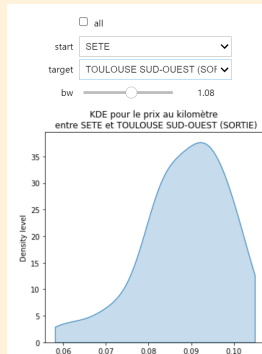
3 Distribution

4 Graphe

- La fonction *get_way*
- L'algorithme des meilleurs sorties

kde_gare

fonction `kde_gare(all, data_price, data_dist, start, target, bw)`
Elle affiche la distribution des prix au kilomètre entre deux gares de péages sur un trajet donné (ou sur tout le réseau).



3.1. KDE interactif

mean_gare

Ne prends en paramètre qu'un *DataFrame* :

MONTPELLIER	13.339130
SETE	11.247826
AGDE	9.239130

Fig 3.2 Chargement des moyennes pour les villes de Montpellier, Sète et Agde

swarm_gare_price

fonction `swarm_gare_price(data_price, name)`

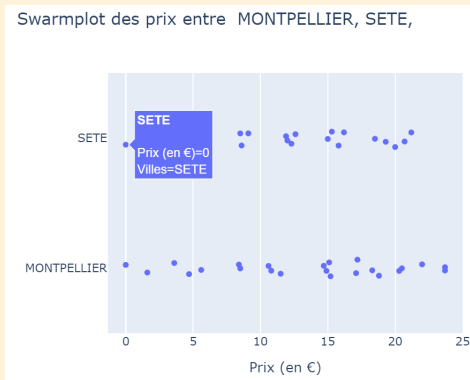


Fig 3.3 Exemple d'affichage de la fonction

1 Récupération et structure des données

- Données géographiques
- Matrices des distances/prix

2 Carte interactive

3 Distribution

4 Graphe

- La fonction `get_way`
- L'algorithme des meilleurs sorties

Cette partie est vraiment centrale dans notre projet car elle a permis de fournir un algorithme du plus court trajet, mais aussi à optimiser certaines fonctions que vous avez pu voir dans les parties précédentes.

- 1 Récupération et structure des données
 - Données géographiques
 - Matrices des distances/prix
- 2 Carte interactive
- 3 Distribution
- 4 Graphe**
 - **La fonction *get_way***
 - L'algorithme des meilleurs sorties

Retour sur la fonction *kde_gare*

Dans *kde_gare*, nous avons plusieurs possibilités pour afficher les distributions :

- soit tout le réseau routier,
- soit

Retour sur la fonction *kde_gare*

Dans *kde_gare*, nous avons plusieurs possibilités pour afficher les distributions :

- soit tout le réseau routier,
- soit tous les trajets **entre deux gares**.

Pour pouvoir généraliser nos fonctions, il a fallu s'adapter et trouver un moyen d'afficher le nom des gares intermédiaires pour n'importe quel trajet : pour cela nous avons créé `get_way`.

Une heuristique

Pour obtenir nos gares, nous nous sommes basés sur une heuristique :

Une heuristique

Pour obtenir nos gares, nous nous sommes basés sur une heuristique :

- on voit le réseau routier comme un graphe pondéré.

Une heuristique

Pour obtenir nos gares, nous nous sommes basés sur une heuristique :

- on voit le réseau routier comme un graphe pondéré.
- on lui applique l'algorithme de Kruskal, qui fournit un arbre de poids minimal qui va forcément tracer le graphe qui liera deux gares successives.

Une heuristique

Pour obtenir nos gares, nous nous sommes basés sur une heuristique :

- on voit le réseau routier comme un graphe pondéré.
- on lui applique l'algorithme de Kruskal, qui fourni un arbre de poids minimal qui va forcément tracer le graphe qui liera deux gares successives.
- on applique un algorithme du plus court trajet entre deux péages qui nous donnera à coup sûr les gares souhaitées.

Un package utile : *networkx*

Pour pouvoir faire tous ces calculs, nous avons utilisé les fonctions *minimum_spanning_tree* et *shortest_path* du package *networkx*.

1 Récupération et structure des données

- Données géographiques
- Matrices des distances/prix

2 Carte interactive

3 Distribution

4 Graphe

- La fonction *get_way*
- L'algorithme des meilleurs sorties

Un étonnant fait

Regardons un peu ce tableau :

Prix	Montpellier	Sete	Agde	Beziers Cabrials
Montpellier	0.0	1.6	3.6	4.7
Sete	1.6	0.0	1.9	3.3
Agde	3.6	1.9	0.0	1.0
Beziers Cabrials	4.7	3.3	1.0	0.0

Tab 4.1. Extrait de notre tableau de prix.

Si on sort à toutes les gares, le coût sera vraiment réduit mais...

Si on sort à toutes les gares, le coût sera vraiment réduit mais...
quelle perte de temps !

Si on sort à toutes les gares, le coût sera vraiment réduit mais...

quelle perte de temps !

L'objectif est donc de produire un algorithme qui sort le trajet le moins coûteux en limitant le nombre sortie.

La fonction `kmin_cost_out`

La programmation de cette fonction est très basique mais très gourmande en calcul...

La fonction `kmin_cost_out`

La programmation de cette fonction est très basique mais très gourmande en calcul...

Si il y a n gares sur le trajet et que nous nous accordons k sortie, le nombre de graphe total calculé sera de :

La fonction `kmin_cost_out`

La programmation de cette fonction est très basique mais très gourmande en calcul...

Si il y a n gares sur le trajet et que nous nous accordons k sortie, le nombre de graphe total calculé sera de :

$$\binom{n-2}{k}$$

Conclusion

ASLTAM est très fonctionnel !

Il permet à la fois de faire de la visualisations des données et donne accès à des outils sympatiques comme la carte ou l'algorithme des trajets.

Conclusion

Le package est puissant dans le sens où il ne se limite pas forcément aux fonctions qu'il le constitue :

Conclusion

Le package est puissant dans le sens où il ne se limite pas forcément aux fonctions qu'il le constitue :

et on peut vous le montrer avec un exemple d'utilisation intelligente de ce-dernier!!!

Conclusion

Merci pour votre écoute !!!

