TRAFIC ET COMPTAGES SUR LES AUTOROUTES - Le departement des Bouches-du-Rhone

March 2, 2019

0.1 Introduction

Afin de connaître le trafic qui s'écoule sur le réseau routier de la DIR Med, tout un système de recueil de données est mis en uvre. Ce système de recueil de données de trafic se compose de stations permanentes (boucles de comptages) et de capteurs installés de façon permanente ou temporaire. Ils permettent notamment de connaître le débit (le nombre de véhicules par km) qui passe quotidiennement sur le réseau et la vitesse. Dans l'ensemble de données de trafic sur les autoroutes de l'aire métropolitaine Aix-Marseille-Provence des mesures réalisées par tranche d'une heure sur les stations de comptages ont été utilisées sur une période de 9 mois : de Janvier a Septembre 2016. Les données sources utilisées dans le cadre de cette étude sont les données des stations RDT (Recueil de Données Trafic). Les données de comptages se composent des débits horaire (Q), taux d'occupation (To) et vitesses, date et jours sur l'ensemble des jours de l'année 2016 . Les données sont représentées sous forme plusieurs tables, chaque table contient les données par jour.

0.2 Organisation de données:

0.2.1 Formats de données :

La donnée de trafic routier : Une valeur de comptage est une valeur numérique, et un fichier de données de comptage routier doit toujours comporter un certain nombre d'informations. -données de comptages : débits, vitesses, taux d'occupation. -Une valeur de trafic : c'est un entier. -Le type de données compté : débit tous véhicules, poids lourds...(classe de véhicule). -Une fréquence de recueil : la donnée est horaire, journalière.... -Un identifiant permettant la localisation de la donnée (station) -La date du comptage.

Organisation de la table principale de l'année 2016 : Nous souhaitons concaténer les tables jour à fin d'avoir une table 'dataframe' contenant les données de toutes les autoroutes de l'aire métropolitaine Aix-Marseille-Provence. Elles sont représentées sous forme plusieurs tables, chaque table contient les données d'un jour. Nous souhaitons concaténer les tables à fin d'avoir une table 'dataframe' contenant les données de toutes les autoroutes de l'aire métropolitaine Aix-Marseille-Provence. Création de la table principale de l'année 2016 : Voici le lien vers les données utilisées de Trafic/Comptage : https://drive.google.com/drive/folders/1YiAVt6uI4ikUgwEuqTPmnjnSxWHvuOq9?usp=sharing

Dans le but de créer la table 'table_2016' nous allons tout d'abords regrouper les tables de jours nommées '0i.dat' dans une table de mois, et après on regroupe les tables mois dans table_2016. En

exécutant le code suivant, on crée la table des données du mois de janvier, sans oublier d'indiquer le chemin des données sur la machine après les avoir télécharger.

```
In [ ]: #Verifier le chemin et importer les packages.
        pwd
        #Chemin
        cd &path

In [ ]: import numpy as np
        import pandas as pd
        from pandas import read_csv
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np
        import string
        from subprocess import check_output

In [ ]: #Creation de la table du mois janvier 2016.
        table_01_16 = pd.concat([pd.read_csv('01.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('02.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('03.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('04.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('05.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('06.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('07.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('08.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('09.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('10.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('11.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('12.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('13.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('14.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('15.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('16.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('17.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('18.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('19.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('20.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('21.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('22.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('23.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('24.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('25.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('26.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('27.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('28.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('29.dat', header=None, sep=';', engine='python'),
                                 pd.read_csv('30.dat', header=None, sep=';', engine='python'),
```

```
                    pd.read_csv('31.dat', header=None, sep=';', engine='python')])
#transformer la table en dataframe
table_01_2016 = pd.DataFrame(table_01_16)
#Exporter la table
table_01_2016.to_csv('table_01_2016.csv', index = False, encoding = 'utf-8')
```

A fin de créer la table 'table_2016' nous allons tout d'abords regrouper les tables de jours nommées '0i.dat' dans une table de mois, et après on regroupe les tables mois dans table_2016, en effectuant le code suivant : On refait le code ci-dessus pour toutes les autres tables, jusqu'au mois d'octobre, vu que les tables du mois de novembre et décembre. Voici le lien vers les tables de données de janvier jusqu'à septembre 2016 : https://drive.google.com/drive/folders/1Z3MFMvPBtjWzgTjDra2VNLwL7u262uYx En appliquant le code suivant, on obtient la table 'table_2016' :

```
In [ ]: table_2016 = pd.concat([table_01_2016, table_02_2016, table_03_2016, table_04_2016, tal
                              table_07_2016, table_08_2016, table_09_2016])
```

Remarque : Vous trouverez la table des données dans le même lien au-dessus.

Après avoir importer la table, nous pouvons renommer les colonnes facilement sur Excel et supprimer les colonnes vides comme on supprime aussi les vitesses et on garde que la vitesse de la 2eme classe de véhicules car elle contient le plus de valeurs.

```
In [ ]: #Importation de la table
        #On précise le séparateur si les données sont sparées par ;
        table_2016 = pd.read_csv("table_2016.csv", sep = ';')
        #Lire la table en dataframe
        table_2016 = pd.DataFrame(table_2016)
        table_2016.head()
```

Les données de trafic sur l'autoroute A7 : L'autoroute A7 de la limite avec le département du Vaucluse jusqu'à son terminus à Marseille et prolonge l'autoroute A6 au niveau du centre de Lyon jusqu'à Marseille sur 312 km. Dans le but d'étudier le trafic sur l'autoroute A7 nous allons créer une nouvelle table qui contient que les stations de l'autoroute A7, à partir de la table initiale.

Image

```
In [83]: # J'ai copié la table pour la modifier et garder la table initiale
         dff = table_2016.copy()
         #La colonne débit Q contient le débit toutes les classes de véhicules
         dff['Q'] = table_2016['Q1'] + table_2016['Q2'] +table_2016['Q3'] +table_2016['Q4'] +ta
         #Supprimer les colonnes Qi
         lst = ['Q1','Q2','Q3','Q4','Q5','Q6']
         dff = dff.drop(lst, 1,errors='ignore')
         rgs = dff.rgs
         #On separe la colonne rgs pour pouvoir lire le code de la station
         #Cela aide a utiliser la fonction startswith
         dff['A'], dff['Code'] = dff['rgs'].str.split('#', 1).str
         dff.head()
```

Out[83]:

	rgs	pr	m	js	jjmmaa	hhmm	Vitesse	To	Qualite	Q	A	Code
0	101#M1a	0	430	Ve	01/01/2016	00:00	92.0	1.0	?	421.0	101	M1a
1	102#M1b	1	9	Ve	01/01/2016	00:00	92.0	3.0		835.0	102	M1b
2	103#M1c	1	508	Ve	01/01/2016	00:00	0.0	0.0		0.0	103	M1c
3	104#M1d	2	30	Ve	01/01/2016	00:00	92.0	3.0		792.0	104	M1d
4	105#M1e	2	462	Ve	01/01/2016	00:00	91.0	3.0		842.0	105	M1e

Le code de toutes les stations dans l'autoroute A7 commence par 'M7' et 'M8', donc nous gardons dans la table 'df' ques les stations de A7.

In [84]:
```
#Selectionner les codes qui commence par M7 et M8
df1 = dff[dff['Code'].str.startswith('M8')]
df2 = dff[dff['Code'].str.startswith('M7')]
df3 = pd.concat([df2,df1],axis = 0,ignore_index = True)
df3.head()
```

Out[84]:

	rgs	pr	m	js	jjmmaa	hhmm	Vitesse	To	Qualite	Q	A	Code
0	709#M7i	264	687	Ve	01/01/2016	00:00	98.0	2.0		487.0	709	M7i
1	710#M7j	265	320	Ve	01/01/2016	00:00	96.0	1.0		277.0	710	M7j
2	713#M7n	267	230	Ve	01/01/2016	00:00	0.0	0.0		0.0	713	M7n
3	715#M7o	267	861	Ve	01/01/2016	00:00	110.0	1.0		327.0	715	M7o
4	716#M7p	268	288	Ve	01/01/2016	00:00	0.0	0.0		0.0	716	M7p

In [85]:
```
#Spprimer les colonnes Code et A
df = df3.drop(df3.columns[[10, 11]], axis=1)
```

In [86]:
```
df.head()
```

Out[86]:

	rgs	pr	m	js	jjmmaa	hhmm	Vitesse	To	Qualite	Q
0	709#M7i	264	687	Ve	01/01/2016	00:00	98.0	2.0		487.0
1	710#M7j	265	320	Ve	01/01/2016	00:00	96.0	1.0		277.0
2	713#M7n	267	230	Ve	01/01/2016	00:00	0.0	0.0		0.0
3	715#M7o	267	861	Ve	01/01/2016	00:00	110.0	1.0		327.0
4	716#M7p	268	288	Ve	01/01/2016	00:00	0.0	0.0		0.0

```
In [49]: #Exporter la table
         df.to_csv('table_A7_2016.csv', index = False, encoding = 'utf-8')
```

Voici le lien vers la table A7 sous le nom de table_A7_2016 : https://drive.google.com/drive/folders/1Z3MFMvPBtjWzgTjDra2VNLwL7u262uYx

0.3 Nettoyage de données

Nous allons appliquer la méthode systématique, pour le nettoyage de données (Dans le lien ci-dessus). ### Les valeurs manquantes : Les données manquantes correspondent aux données non présentes ou ń vides ż ou bien les données marquées d'un indice de qualité mauvais, c'est à dire les données dont la mesure qualité indique ' ? '. #### Les données manquantes NaN: ###### Remarque : Dès qu'une des 3 valeurs (Q, V ou TO) est manquante, la mesure complète est considérée manquante, donc on élimine toute la mesure. A l'aide de la commande suivante on cherche les valeurs manquantes dans chaque variable. Le montre que cette table ne contient aucune donnée manquante:

```
In [50]: df.isnull().any()

Out[50]: rgs        False
         pr         False
         m          False
         js         False
         jjmmaa     False
         hhmm       False
         Vitesse    False
         To         False
         Qualite    False
         Q          False
         dtype: bool
```

Les données marquées d'un indice de qualité mauvais : Ces données ont l'indice ' ? ' dans la variable 'Qualité' dans la table, si la qualité de mesure est mauvaise.

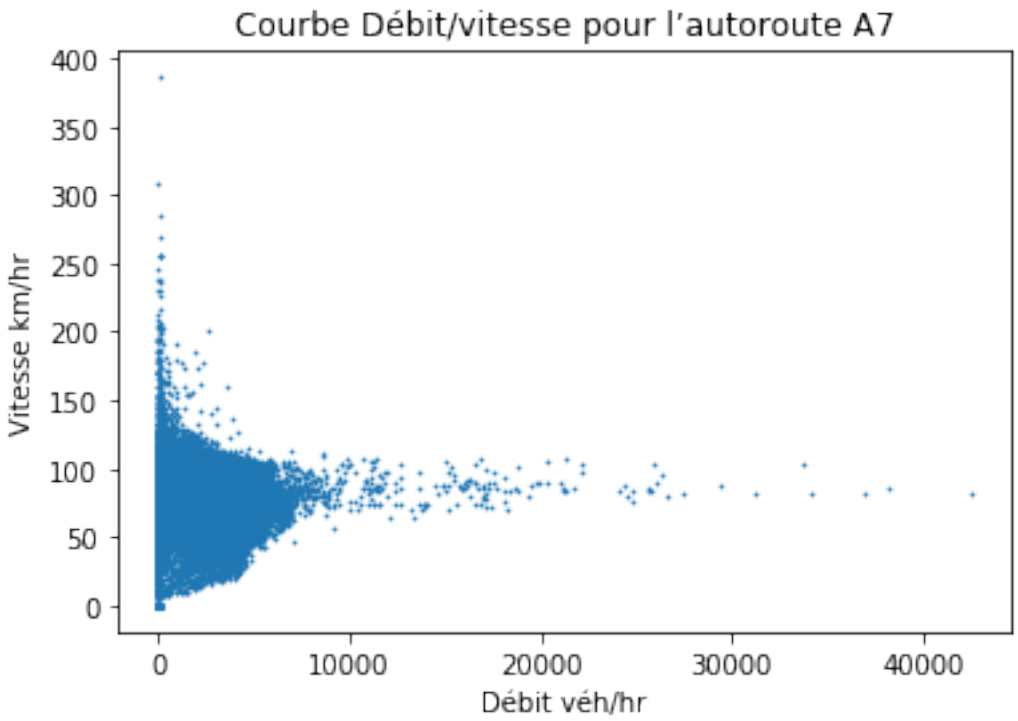
```
In [64]: selection = (df['Qualite'] = ?)
         # On applique la selection
         df = df[selection]


          File "<ipython-input-64-7b9dce827f5d>", line 1
        selection = (df['Qualite'] = ?)
                                     ^
    SyntaxError: invalid syntax
```

0.3.1 Sur vitesse/ sur-comptage :

Valeur aberrante : Une valeur aberrante est une valeur qui s'écarte fortement des valeurs des autres observations, anormalement faible ou élevée. Les valeurs aberrantes peuvent avoir un effet disproportionné sur les résultats statistiques, tels que la moyenne, ce qui peut conduire à des interprétations trompeuses. Et c'est souvent à cause des erreurs de saisie de données ou bien un problème lié au procédé. la couverture en terme de capteurs est rarement totale sur le réseau et les mesures peuvent comporter des erreurs ou être indisponibles dans le cas de panne de capteur. Le recueil des données brutes fournit par conséquent empreinte d'erreurs. C'est la raison pour laquelle des méthodes de qualication (=identication) des données aberranteset de reconstitution des données manquantes sont développées, ces méthodes donnent comme résultat les tests suivants: -Le test de sur-comptage est effectué en comparant chaque débit mesuré à un seuil de débit : a priori 6000 véh./heure (à vérifié). -le test de sur-vitesse, est effectué en comparant chaque vitesse mesurée à une vitesse limite : à priori 160 km/h. #### Utilisation de graphiques pour identifier des valeurs aberrantes : Nous pouvons aussi trouver le dernier résultat en identifiant les valeurs aberrantes par des graphes : Sur les nuages de points, les points très éloignés des autres sont de possibles valeurs aberrantes. En effectuant la commande suivant, on obtient un graphe de nuage :

```
In [58]: import matplotlib.pyplot as plt
         #La colonne Debit
         x = df.Q
         #La colonne Vitesse
         y = df.Vitesse
         plt.scatter(x,y,s =.5)
         plt.title('Courbe Débit/vitesse pour lautoroute A7')
         plt.xlabel('Débit véh/hr')
         plt.ylabel('Vitesse km/hr')
         #Sauvegarder le graphe
         plt.savefig('ScatterPlotA7.png')
         plt.show()
```



Ce nuage de points met en évidence des valeurs aberrantes, les points très éloignés des autres sont supérieur que 6000 pour le débit et 150 pour la vitesse. Ce qui confirme le résultat dernier.

Compter et supprimer les données aberrantes : Par la commande ci-dessous, nous allons récupérer les données qui vérifient les deux conditions : Débit horaire <= 6000 véh/hr. Vitesse horaire <= 160 km/hr.

On sélectionne les données qui ne vérifient un débit et une vitesse élevés, et on obtient la dataframe 'table_QV' qui contient que les données qui ont à la fois débit inferieur à 6000 et vitesse inferieur à 160, puis on détermine la part de ces données:

```
In [88]: maselection = (df['Q'] <= 6000) &  (df['Vitesse'] <= 160)
         # On applique la selection
         table_QV = df[maselection]
         #determinons la longueur de ces données
         l = len(df) - len(table_QV)
         l
         # calculer le pourcentage de données
         prc = (l*100)/ len(df)
         prc

Out[88]: 0.3525701212151406
```

La valeur 'prc' retourne le pourcentage de ces données par rapport à la dataframe 'df' de A7. Le pourcentage est négligeable **prc = 0.35%**. on se permet alors de supprimer ces données de la table initiale df. Autrement dit, on garde les données de la dataframe 'table_QV'.

```
In [92]: dfA7 = table_QV
         dfA7.head(7)
```

Out[92]:

	rgs	pr	m	js	jjmmaa	hhmm	Vitesse	To	Qualite	Q
0	709#M7i	264	687	Ve	01/01/2016	00:00	98.0	2.0		487.0
1	710#M7j	265	320	Ve	01/01/2016	00:00	96.0	1.0		277.0
2	713#M7n	267	230	Ve	01/01/2016	00:00	0.0	0.0		0.0
3	715#M7o	267	861	Ve	01/01/2016	00:00	110.0	1.0		327.0
4	716#M7p	268	288	Ve	01/01/2016	00:00	0.0	0.0		0.0
5	717#M7q	268	859	Ve	01/01/2016	00:00	0.0	0.0		0.0
6	718#M7r	269	326	Ve	01/01/2016	00:00	104.0	1.0		459.0

0.3.2 Débit nul :

Lorsque le débit est nul la voie est vide. Donc une heure sans donnée, revient à dire que la voie est vide cette heure ci.

```
In [99]: Q = dfA7.Q
         Q[Q==0] = np.nan

         #df1 = dff[dff['Code'].str.startswith('M8')]
```

```
/Users/mac/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
In [100]: #verifier est ce que les zeros sont bien des NaN dans Q
          dfA7.isnull().any()
```

```
Out[100]: rgs        False
          pr         False
          m          False
          js         False
          jjmmaa     False
          hhmm       False
          Vitesse    False
          To         False
          Qualite    False
          Q           True
          dtype: bool
```

```
In [ ]: #Gérer les Q= NaN
        from sklearn.impute import SimpleImputer, MissingIndicator
        #imputer = SimpleImputer(missing_values= np.nan, strategy='mean')
        # Lier imputer à la colonne Q
        #imputer.fit(dfA7[:,9])
```

```
        #imputer.fit_transform(dfA7)
        dfA7.fillna(dfA7.mean(), inplace=True)
        dfA7.head(10)
        df.head(10)
```

In [ ]:

In [ ]:

In [ ]: