

# **Rapport de Stage de Licence informatique**

## **Analyse de donnée sur de l'Open Data**

Rédigé et réalisé par Paul Monimeau & Sous la tutelle de Mr F.Flouvat (UNC) et de Mr A.SALES (OPT)

# Rapport

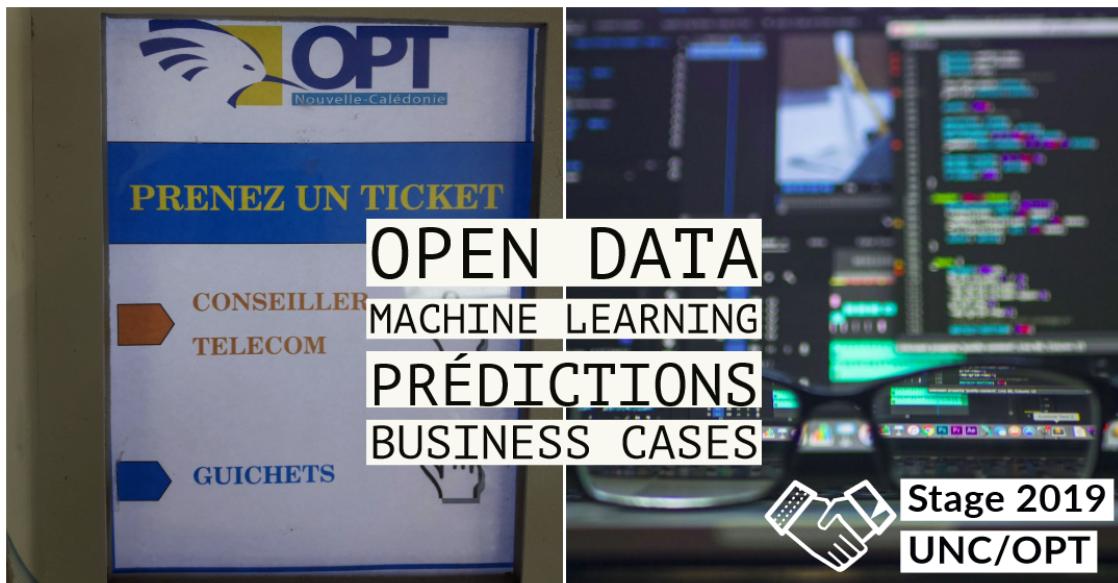
January 30, 2020

Rapport de Stage

Analyse de donnée sur de l'Open Data

Rédigé et réalisé par Paul Monimeau

Sous la tutelle de Mr F.Flouvat (UNC) et de Mr A.SALES (OPT)



## 1 INDEX

1. Section 2.1
2. Section 2.2
3. Section ??
4. Section ??
5. Section ??
6. Section ??
7. Section ??
8. Section ??
9. Section ??

10. Section ??
  11. Section ??
  12. Section 2.5
  13. Section 3
- 

## 2 PRESENTATION

### 2.1 Sujet du stage

L'objectif initial du stage est d'analyser des données de Scal'Air NC, obtenues grâce à une API développée qui vient scrapper le site web de Scal'Air. Le but du stage étant de produire un “data story telling” ainsi qu'un modèle de prédiction pouvant possiblement être exporté sur une API. Ce schéma de travail pourrait quant à lui servir de base pour toutes autres approches d'analyses de données sur de l'open Data lors d'innovations à l'OPT.

- Lien du Sujet du stage
- Lien de la Présentation reveal.js
- Lien du Rapport jupyterNoteBook sur Git
- Lien du Rapport sur GitPages

### 2.2 Apprentissage avec OpenClassRooms et autres

Section 3.2

### 2.3 Première analyses des données de Scal'Air

Ces données contiennent l'indice de qualité de l'air pour certains quartiers de Nouméa et à des dates données. Cet indice est calculé au préalable par Scal'Air en fonction de la quantité des différentes molécules qui composent l'air. Cet indice est appelé **indice ATMO**

**Un exemple du jeu de données :**

On voit ici un DataFrame créé par python.

	typologie	lieu	nom	indice	message	color	id	date
0	INDUSTRIELLE	LOGICOOP	Logicoop	3	Bon	GREEN	98001	07/09/2019
1	URBAINE	FAUBOURG_BLANCHOT	Faubourg Blanchot	3	Bon	GREEN	98003	07/09/2019
2	URBAINE	ANSE_VATA	Anse Vata	3	Bon	GREEN	98004	07/09/2019
3	URBAINE_INDUSTRIELLE	MONTRAVEL	Montravel	3	Bon	GREEN	98002	07/09/2019
4	URBAINE	GENERAL	Général	3	Bon	GREEN	0	07/09/2019

Les données de Scal'Air nous ont permises d'apprendre la méthodologie pour arriver à la publication d'une API prédictive.



# Méthodologie

1

## RÉCUPÉRER DE LA DONNÉE

La traiter et l'analyser, puis utiliser un algorithme de prédition et l'exporter.

2

## CRÉATION D'UNE API

Créer une api contenant le modèle de prédition créer à l'étape précédente.

3

## PUBLICATION DE L'API

Publication de l'api sur heroku grâce à une image docker. Puis publication sur un marketplace: RapidAPI.

4

## CRÉATION D'UN FRONT WEB

Création d'une interface utilisateur, permettant d'accèder facilement a l'api, ici une interface web.

OPEN DATA

## 2.4 Un tout nouvel horizon et de nouvelles données

Après cet entraînement sur les données de Scal'Air et un processus bien rodé, nous pouvons maintenant s'attaquer à un exemple possiblement utile à l'OPT, le temps d'attente dans les Agences OPT. Cette donnée est récupérée grâce à un script python qui via un cron tourne toutes les 5 min et va récupérer les données grâce à une API disponible sur le site de l'OPT. Après un petit tri et un gros nettoyage, les données sont stockées sur un csv qui ne cesse de grossir.

### 2.4.1 L'utilité possible

Ce que l'on peut espérer de ces analyses, c'est qu'on puisse observer une régularité du temps d'attente en fonction des horaires, et du nombre de guichet ouvert, et de l'agence. Ces informations pourraient être utiles pour une réorganisation des Agences, ou bien même pour le client qui voudrait prédire, ou avoir une idée du temps à attendre pour aller déposer ou récupérer son colis/courrier/etc...

### 2.4.2 Récupération de la donnée

Pour récupérer cette donnée sur les temps d'attentes, il nous a fallut récupérer l'endpoint d'une API sur le site de l'opt, qui permet d'afficher en direct le temps d'attente dans les agences. Une fois cet endpoint connu, il nous a juste fallut l'exploiter en python, en faissant une requête http sur cet endpoint. Puis nous avons du nettoyer le résultat pour qu'il ne nous reste plus que ce que l'on voulait. Section ??

Autrement dit :

	AVGWaitingTime	MAXWaitingTime	adresse	id	nom	type	date
0	00:01:44	00:02:37	7 RUE EUGÈNE PORCHERON	4314	Agence de NOUMEA AGENCE PRINCIPALE	AGENCE	2019-10-22 09:44:10
1	00:00:00	00:00:00	28 RUE HENRI GASPARD	4169	Agence de NOUMEA SUD N'GEA	ANNEXE	2019-10-22 09:44:10
2	00:00:00	00:00:00	224 RUE JACQUES IEKAWE	4166	Agence de NOUMEA BELLE VIE	AGENCE	2019-10-22 09:44:10
3	00:01:42	00:03:54	53 AVENUE BONAPARTE	4167	Agence de NOUMEA RIVIERE SALEE	AGENCE	2019-10-22 09:44:10
4	00:02:53	00:05:41	35 RUE DU 18 JUIN	4313	Agence de NOUMEA MAGENTA	AGENCE	2019-10-22 09:44:10

Ce script python est alors lancé, grâce à un crontab, toutes les 5 minutes. Ce qui nous procure pas mal de données.

### 2.4.3 Analyse de la donnée

**Nettoyage** Une fois notre csv prêt à l'emploi, nous avons commencé à retravailler cette donnée en python et à afficher les premiers graphiques, le principe de nettoyage et d'amélioration de la donnée est identique aux méthodes utilisées pour le jeu de donnée Scal'Air (changement des types/suppression de la donnée inutilisable/etc). A une chose près, dans ces données là, nous avons remarqué une anomalie qui revenait assez souvent. Le problème venait directement de la borne Esirus ou de l'agence elle-même. En effet, de temps en temps, le temps d'attente moyen et maximum ne cessait d'augmenter et atteignait des fois des sommes folles (+7h). Nous avons donc pris la décision de supprimer toutes les lignes avec un temps d'attente moyen de plus de 2h ainsi que toutes les lignes précédentes pour la même agence et pour le même jour jusqu'à ce que l'on tombe sur un temps d'attente plus élevé.



**Représentations graphiques** Notre premier graphique a était un heatmap représentant le nombre de seconde d'attente moyenne pour chaque agences et chaque horaires.

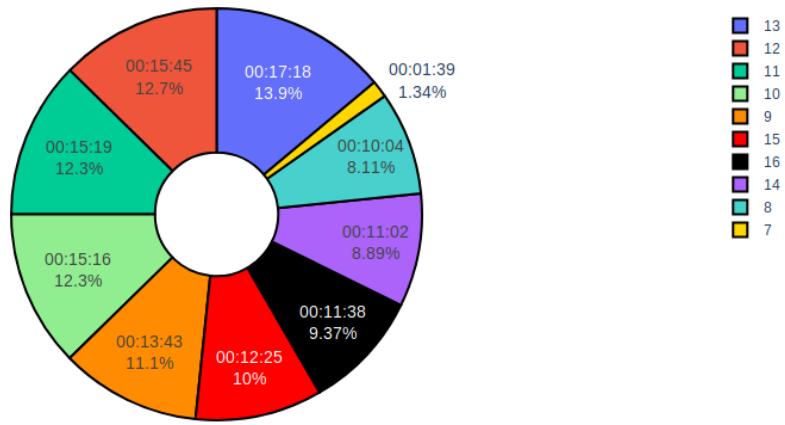
#### Section 3.4

Bien entendu toutes ces moyennes sont plutôt bien représentative, mais tout de même à prendre à la légère et observer la médiane ainsi que l'écart type. Grâce à ce schéma, on observe très clairement quelles sont les agences les plus visitées et sur quels créneaux horaires. Cela nous permet de voir aussi quelles agences sont ouvertes au public et celles qui ne le sont pas. A contrario, la médiane nous montre aussi qu'à certaines horaires, il peut y avoir beaucoup d'attente, comme aucune attente : par exemple MAGENTA A 15H. Mais la médiane nous montre aussi que nos données ne sont pas faussé par de faux résultats.

Ce schéma suivant nous permet d'observer le temps d'attente moyen en seconde suivant les agences et le jour de la semaine. On y voit un temps d'attente plus long le lundi et mardi pour l'agence de magenta par exemple.

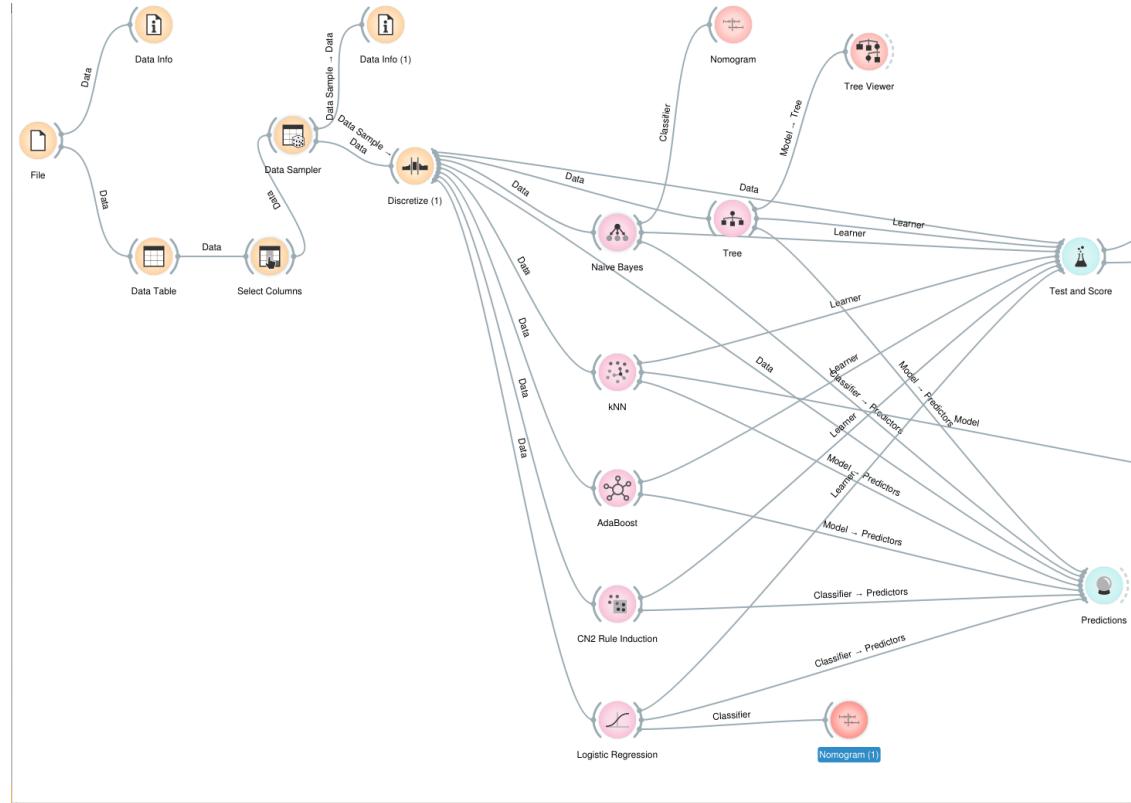
Mis à part ces heatmaps, des graphiques ‘camembert’ ont étaient utilisés pour visualiser la donnée. Pour chaque agence un graphique, représentant en moyenne le nombre de secondes (en survol, sinon heure:minutes:secondes) d'attente pour chaque horaires.

Moyenne du temps d'attente en seconde pour l'agence Agence de NOUMEA MAGENTA



#### 2.4.4 Algorithme de prédiction

**Premier model de prédiction avec Orange** Orange est un logiciel fournit avec la suite Anaconda3 qui permet de créer facilement des modèles de prédiction et de visualiser des données. C'est un logiciel utilisable par quiconque, même ceux qui ne savent pas coder.



**Model Tree** Premièrement avant d'attaquer quoi que ce soit, il faut être bien sûr de savoir faire la différence entre modèle de régression et modèle de classification. Tout d'abord un modèle statistique est un algorithme qui va pouvoir prendre en entrée un jeu de donnée avec une cible ( qui est une variable du jeu de donnée ) et qui va pouvoir s'entraîner sur d'autre donnée afin de pouvoir à l'avenir prédire ou classifier le plus précisément possible cette cible avec des paramètres donnés.

Les problèmes de classement consistent à prédire les classes ou étiquettes d'un ensemble de données à partir d'une base d'apprentissage pré-étiquetée.

Pour la régression, il n'y a pas d'ambiguïté : il s'agit de prédire des valeurs numériques continues pour un ensemble de données à partir d'une base d'apprentissage.

Le modèle Tree est un modèle de classification, l'objectif de notre arbre de décision est donc de classer les temps d'attente en 3 classes en fonction des minutes. A chaque étape, l'algorithme va chercher le critère permettant de séparer au mieux ces 2 populations. Il existe plusieurs critères de séparation, ici l'heure ainsi que le jour. Dans notre exemple nous allons utiliser l'entropie.

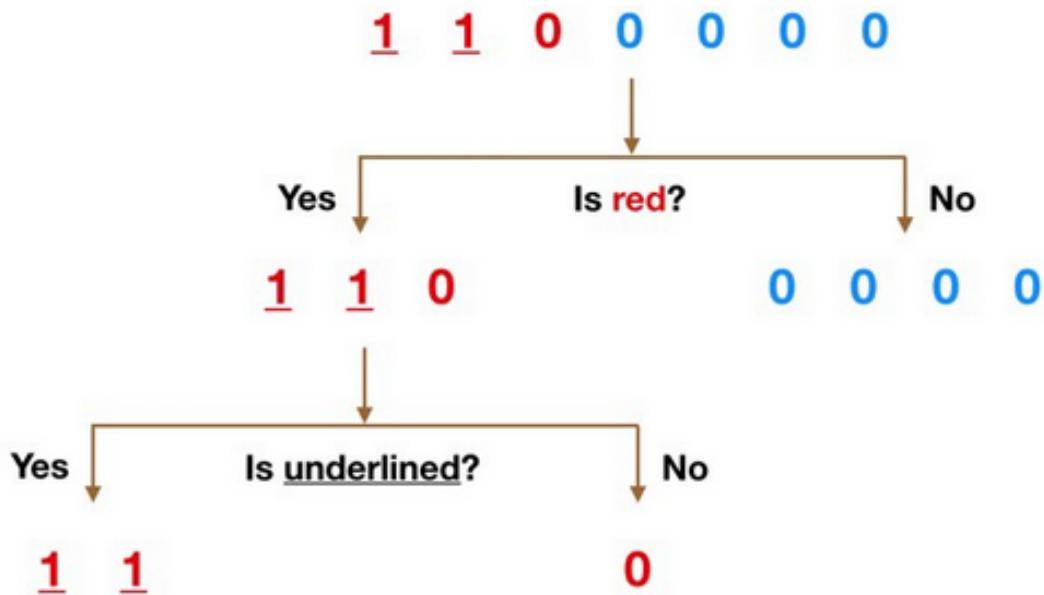
On aurait pu ici prendre un modèle de régression ( comme la régression logistique qui est assez précise dans notre cas ) mais, grâce à orange, nous avons pu comparer la précision entre les différents modèles et le modèle "Tree" est dans les plus précis, mais surtout il est le plus représentatif, notamment avec la possibilité d'afficher un arbre de prédiction.

Comme vu avec le jeu de donnée précédent, il est préférable, premièrement, de tester le validité des modèles de régression et de classification sur Orange. Comme expliqué ci-dessus, cette étape a été faite et couplé à l'essai de plusieurs algorithmes sur Python avec Scikit-Learn, le modèle retenu et le 'RandomForestClassifier'. Avec notre petit jeu de donnée, on arrive à une précision d'environ 0,84.

```
: 1 from sklearn.ensemble import RandomForestClassifier
 2 clf = RandomForestClassifier(n_estimators=10)
 3 clf=clf.fit(X_train,y_train)
 4 result = clf.score(X_test, y_test)
 5 print(result)
```

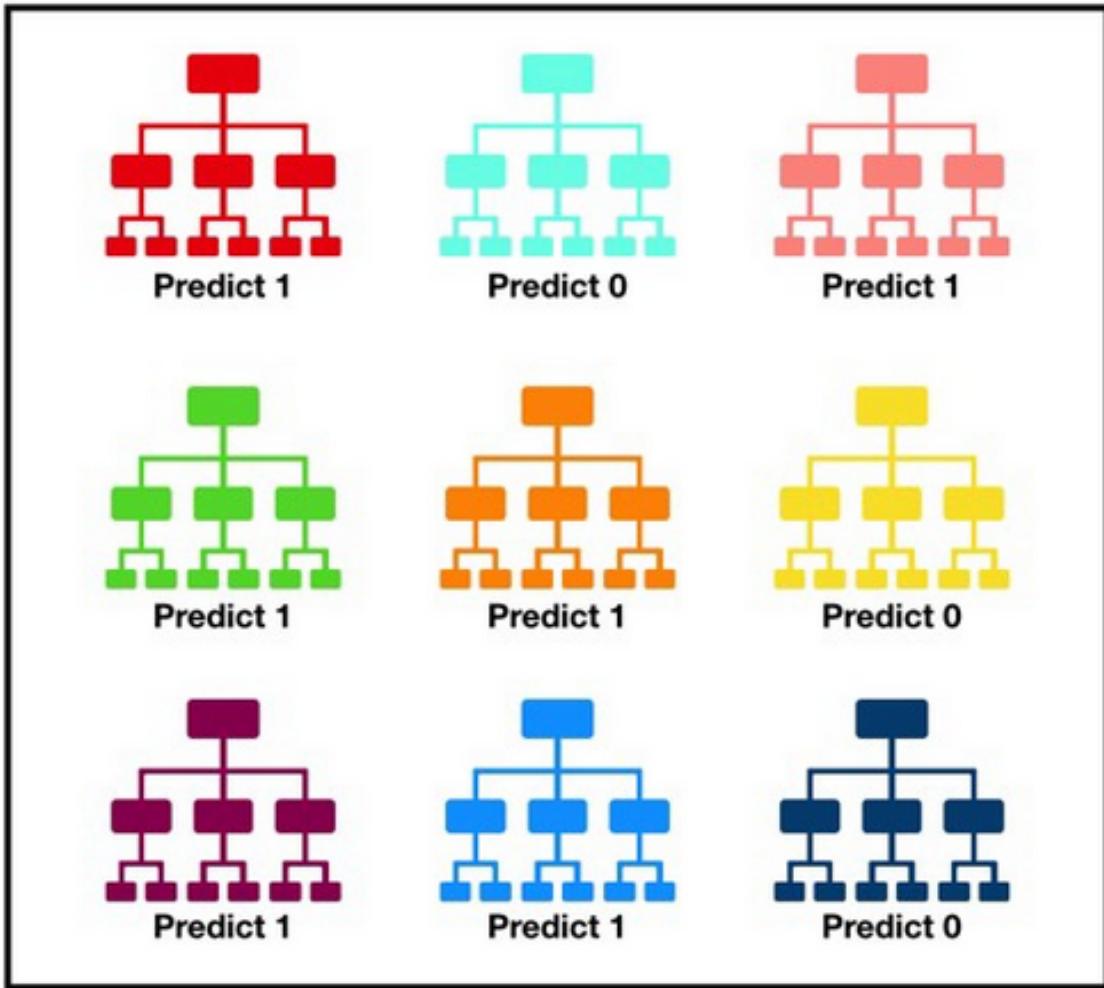
**0.84038564542**

Cet algorithme est principalement basé sur celui du 'TreeClassifier' :



Simple Decision Tree Example

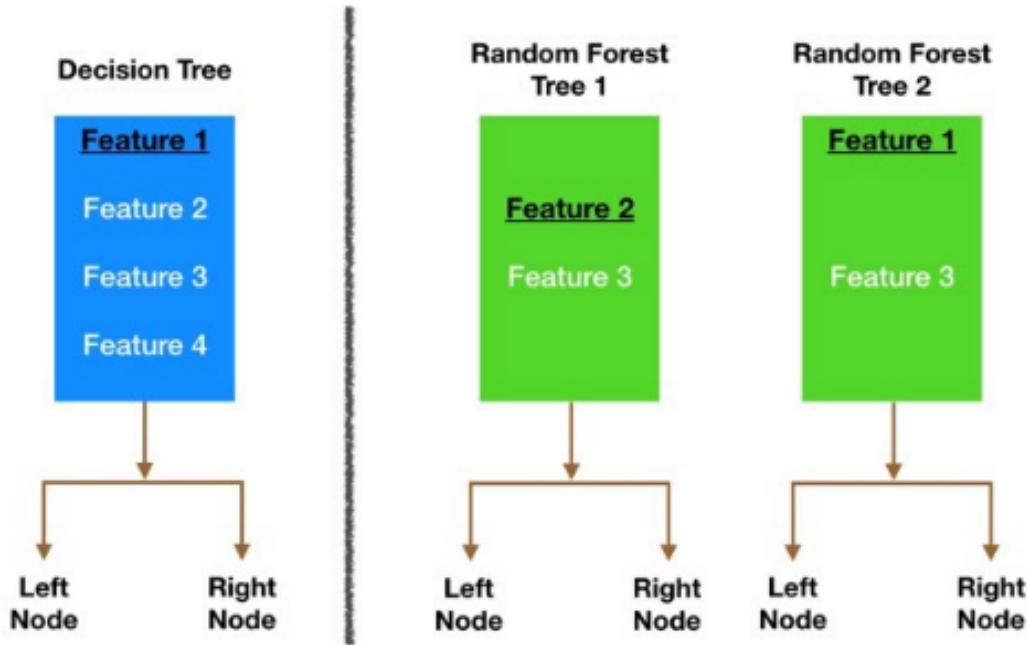
**Pourquoi avoir choisi cet algorithme ?** Parce que le modèle ‘TreeClassifier’ a tendance à sur apprendre et que pour palier à ce problème le modèle RandomForestClassifier a été créé. Il permet de créer plusieurs arbres avec pour moyen de séparation des éléments aléatoires, et il prend ensuite comme résultat, le plus grand nombre d’étiquette identique retourné.



Tally: Six 1s and Three 0s

**Prediction: 1**

Visualization of a Random Forest Model Making a Prediction



Node splitting in a random forest model is based on a random subset of features for each tree.

Encore une fois ce modèle est exportable et peut être utilisé avec n'importe quel code. Nous allons encore une fois l'utiliser dans une API faite en python.

#### 2.4.5 Crédit de l'API

Nous reprendrons les mêmes étapes que pour les données de scal'Air, il suffit, grâce à python et la librairie Flask de faire un petit script important le modèle et renvoyant, suivant les endpoints, le json voulu. Une fois fait, nous reproduirons encore une fois la même méthode, autrement dit, nous publions l'API sur heroku via le CLI d'heroku et un dockerfile. Heroku est une plateforme en ligne (une PAAS) permettant d'héberger gratuitement des API. Pour publier la notre, à l'aide du CLI fourni par heroku sous linux, il nous a juste suffi de Dockeriser notre environnement et de créer un fichier requirement.txt qui indique toutes les dépendances de notre script python. Une fois nos fichiers envoyés sur heroku, et après maintes phases de test, notre api était disponible via une URL publique. Le travail restant était juste de publier ce lien sur Rapide API et de documenter un peu notre API en y montrant des exemples.

Par contre, nous avons mis en place, cette fois ci, un pipeline entre heroku et la repo contenant tous les fichiers pour faire tourner l'API. Ce pipeline permet de ré upload l'API sur heroku si il survient un quelconque changement sur la repo. Ce qui nous permet de mettre continuellement le modèle à jour avec les nouvelles données qui arrivent tous les jours. C'est un déploiement continu. Finalement, cette fois ci, nous n'essayerons pas d'appeler l'API en Java mais en PHP sur un site web.

#### **2.4.6 Creation d'une interface Web**

Une interface web a tait cree pour montrer que l'API est utilisable partout. Ce site web a tait cree en PHP/HTML/CSS/JS avec comme framework Bootstrap 4 pour le cote responsive. Ce site web est donc conus sur un modele MVC "from scratch" et est facilement utilisable sur telephone.

<https://optapi2.000webhostapp.com/MDB-Free/index.php>

Section [3.5](#)

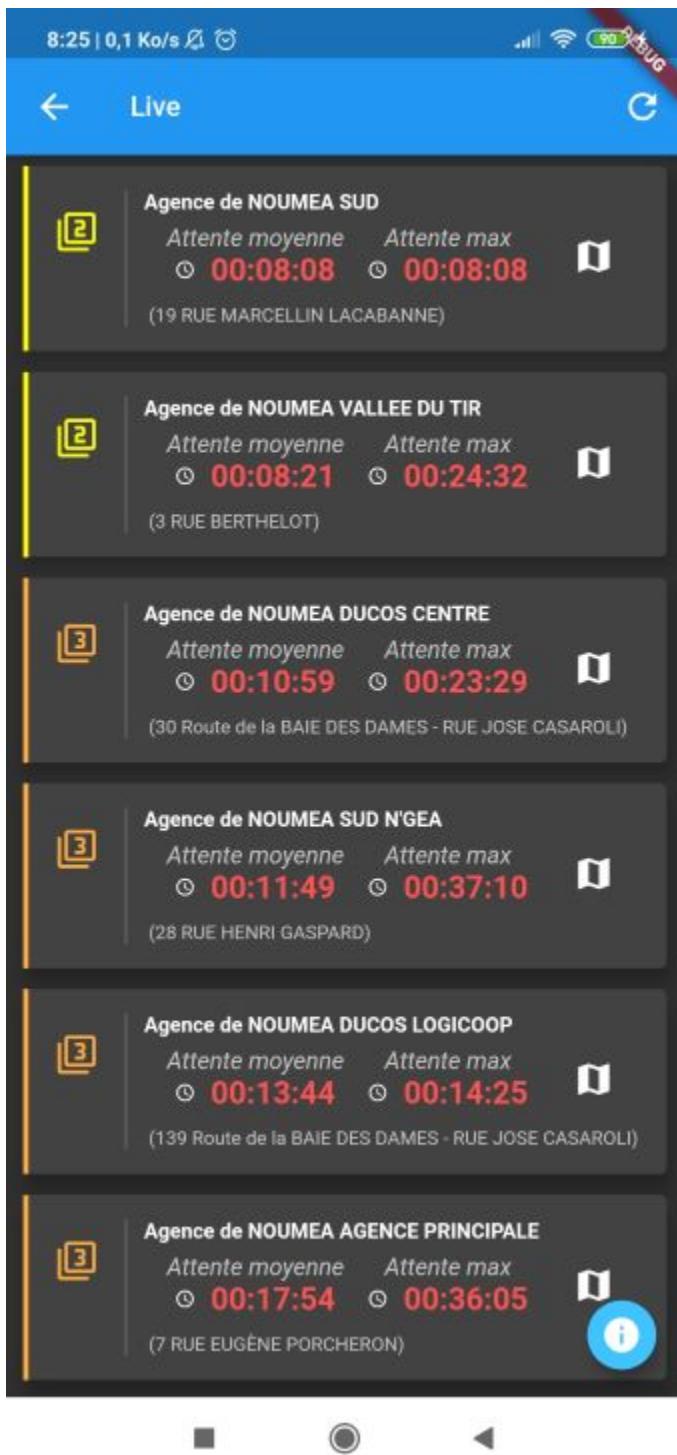
#### **2.4.7 Creation d'une application Mobile**

Une application Mobile a tait cree dans le meme principe que le front Web, avec en petit plus l'affichage de ce que l'on a sur le site Web actuel de l'OPT (les temps d'attentes en direct). [Code sur github](#)

Section ??

Cette application a tait developper en Dart et sous le framework google : Flutter. Ce langage permet de compiler le code sous Android et sous IOS sans aucune modification du code. De plus le framework flutter est trs suivit et possede un trs grand nombre de widget utilisables. Ce langage est un langage objet, il ma donc tait assez facile de l'apprendre, sachant djà coder en Java. Premièrement, l'application comprenait seulement l'affichage 'live' des temps d'attentes.

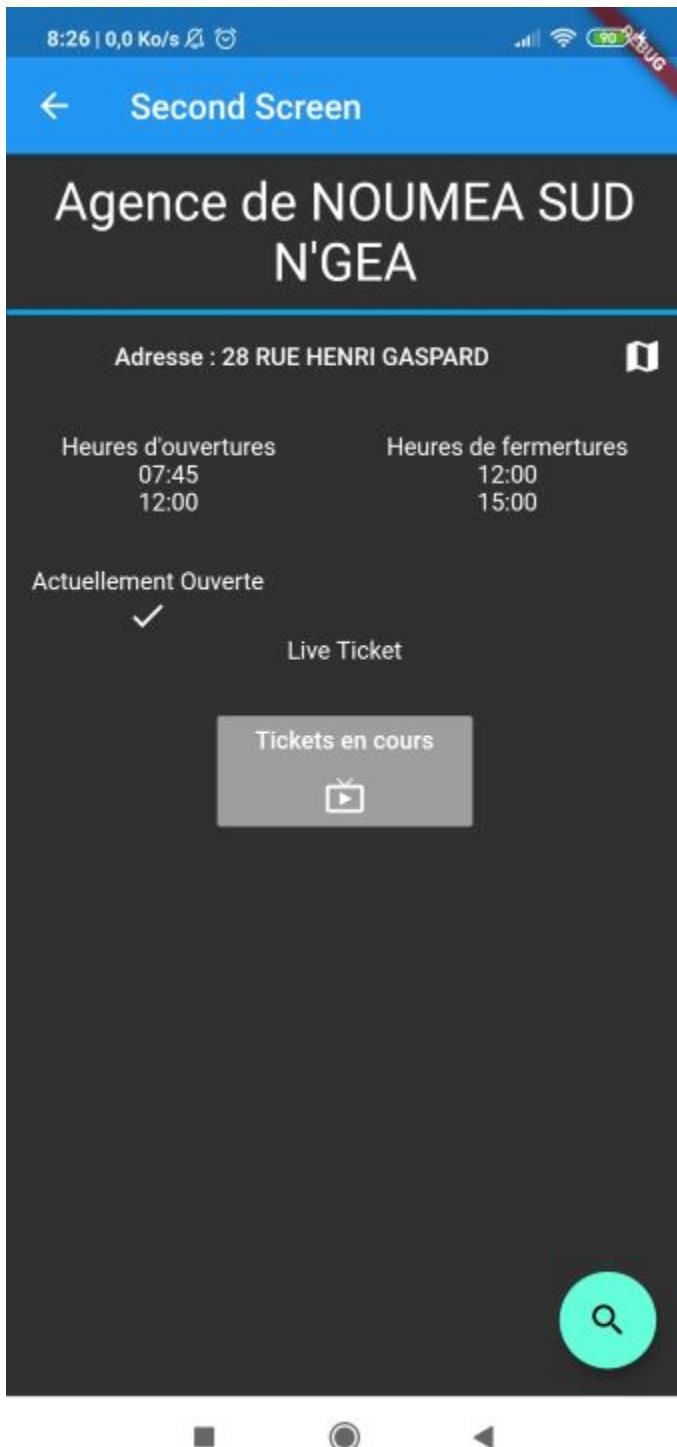




Sur le coté gauche on retrouve un code couleur et une icon permettant de voir rapidement si une agence possède beaucoup d'attente ou non. De plus les agences sont triées dans l'ordre croissant suivant leur temps d'attente moyen.

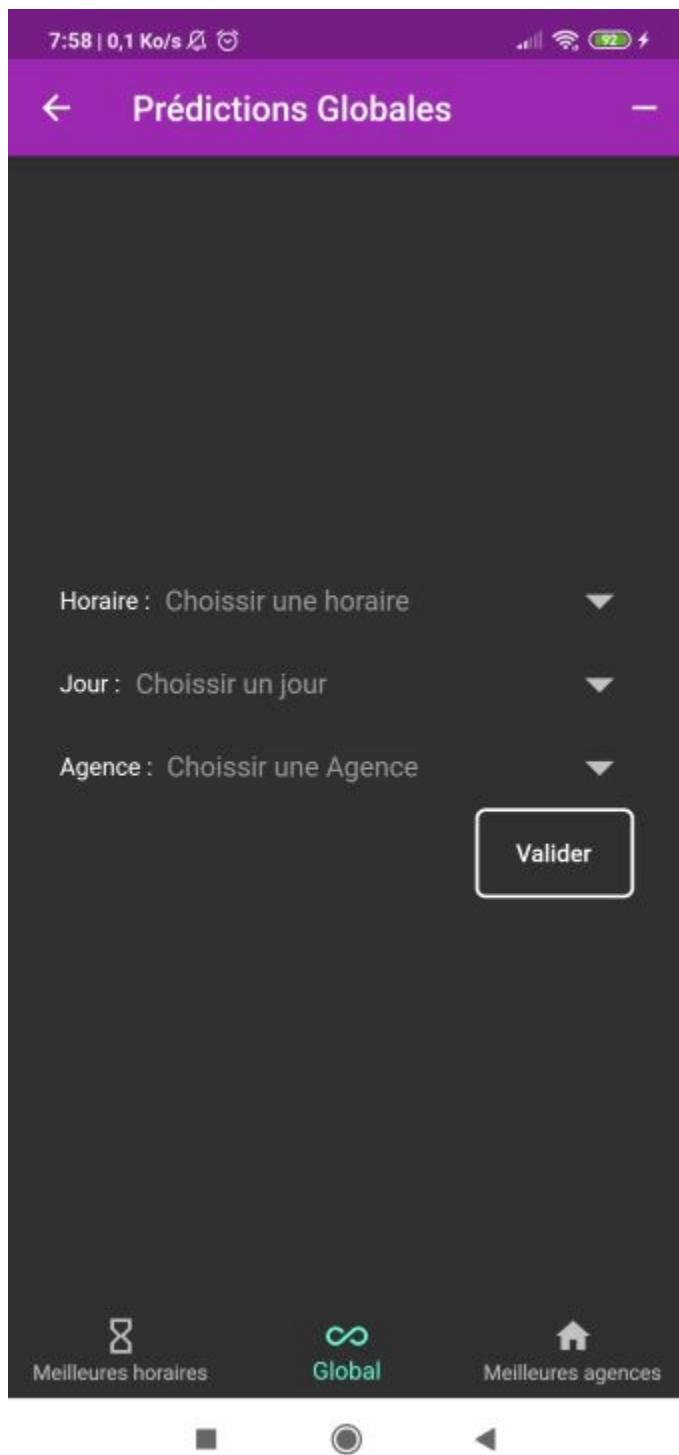
Une fois que l'on clic sur une agence, on peut avoir quelques informations supplémentaires, comme les heures d'ouvertures et de fermetures, le lien google map de l'adresse et un bouton 'live ticket'. Ce bouton redirige vers une vidéo youtube qui montre un écran avec les numéros des tickets en

cours. L'idée serait, dans un futur proche, de soit avoir directement la donnée du ticket en cours dans l'api OpenData de l'opt, ou un flux vidéos continu qui filmerait les écrans des tickets. Avec ça, il serait possible d'aller chercher son ticket à l'agence et de repartir faire une quelconque course en attendant une notification qui nous dirait que c'est bientôt notre tour par exemple. On pourrait même imaginer qu'il serait possible de récupérer un ticket directement sur l'application.



Cette application contient aussi une partie ‘prévision’, reprenant le même principe que celui du site

Web. Nous avons 3 ‘pageView’ (3 pages avec une gestion de switch sur les cotés de l’écran pour passer d’une page à l’autre) avec sur chacune un formulaire dédié a la prédition voulue (Endpoint). Le reste des Screenshots pourra être retrouver en Section 3.6.



## 2.5 Conclusion

Ce stage a été très enrichissant pour moi car il m'a permis de découvrir dans le détail le monde de l'entreprise notamment avec la méthode agile Scrum. De plus il m'a permis d'apprendre l'utilisation de nombreuses technologies comme évoqué dans le corps du rapport. Il m'a aussi permis de rencontrer des personnes travaillant dans le domaines de l'informatique et de la programmation, ce qui m'a apporté une meilleure clairvoyance sur le métier que je voudrais exercer dans un futur proche. Ce stage m'a donc était très important, que ce soit pour la suite de mes études supérieurs ou pour ma future vie professionnelle. Il était question d'appliquer un cheminement d'utilisation de données open source, en passant par l'analyse de données, modèle prédictif, création d'api à l'utilisation de cette dernière dans un programme. Puis le sujet à était approfondi pour passer sur des données liées à l'opt et même sur la création d'un site web ainsi qu'une application mobile. Un cheminement complet accès sur l'utilisation du PAAS et du cloud, et surtout indépendant des structures interne à l'opt. Autrement dit ce stage aurait pu être fait par nimporte qui sur son canapé grâce à ses outils gratuit et cette donnée openSource. Et pour finir, n'oublions pas que ce rapport de stage, qu'il soit sous forme de pdf ou html, a été généré en Latext grâce à du Markdown sous Jupyter NoteBook. Ce qui explique les quelques problèmes d'affichage sous la version pdf, (le htm est beaucoup plus esthétique).

---

## 3 ANNEXES

### 3.1 Index annexes

1. Section [3.2](#)
2. Section ??
3. Section [3.2.2](#)
4. Section ??
5. Section ??
6. Section ??
7. Section ??
8. Section [3.4](#)
9. Section [3.5](#)
10. Section [3.6](#)

### 3.2 Apprentissage avec OpenClassRooms

Section [2.2](#)

#### 3.2.1 Technologies utilisés



- Anaconda3



- JupiterNoteBook



- Python3.7



- GitLab



- GitKraken

### 3.2.2 Fonctionnement d'OpenClassRooms

- Le cours est structuré et il est évolutif, et propose même des évaluations pour se perfectionner et ce tester. Malheureusement, même en ayant validé ses évaluations, le certificat de réussite n'est disponible que si le compte est premium.

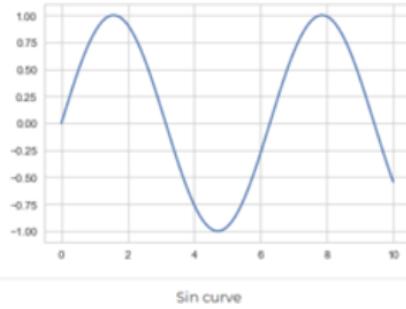
#### Certificats de réussite

The image shows two examples of certificates from the OpenClassRooms platform. Both cards feature a profile picture of a person, the Python logo, and the text "Original OPENCLASSROOMS".  
The left card is titled "Use Python libraries for Data Science" and the right card is titled "Learn Python Basics for Data Analysis". Both cards have a "Voir le certificat" button at the bottom and a "AJOUTER À MON PROFIL LINKEDIN" button at the bottom right.

De plus

les cours sont remplis d'exemples :

```
1 fig = plt.figure()
2 ax = plt.axes()
3 x = np.linspace(0, 10, 1000)
4 ax.plot(x, np.sin(x));
```



We could have simply typed `plt.plot(x, np.sin(x))`.

So, here is our very first plot. Not bad. While this plot is nice, it is not quite ready for use in a professional setting. There are many aesthetic elements that need to be changed including:

- The font size
- The color of the graph
- Line style of the curve
- Excess white space

- Les évaluations sont validés par des élèves et pour réussir le cours complet et obtenir le certificat (seulement si le compte est premium ) il faut aussi corriger 3 évaluations d'autres élève. C'est un très bon exercice pour apprendre et pour voir différents moyens d'arriver à un même but.

### 3.2.3 Sujets abordés

**Anaconda et Python** Le cours [Learn Python Basics for Data Analysis](#) a servi à se remémorer le fonctionnement de base de Python, et surtout à l'utilisation de Jupyter NoteBook. Pour obtenir Jupyter NoteBook, il suffit d'installer l'environnement Anaconda3, qui fournit Python 3.7 et l'Anaconda Navigator ( qui contient Jupyter NoteBook et d'autres applications ). Ce cours nous initie donc au Python avant de toucher à de l'analyse de données.

**Librairie pour la Data Science** Ce cours suivant [Use Python librairies for Data Science](#) a quant à lui servi pour l'apprentissage de la mise en valeur des données. Notamment grâce à de nombreuses librairies qui permettent d'afficher de la données dans des tableaux ou dans toutes sortes de graphiques différents et d'appliquer des traitements sur cette dernière. Dans l'ordre du cours les librairies utilisées sont :

- Numpy
- Matplotlib

- Seaborn
- Pandas

**Numpy** est une librairie très utilisée en python, elle est la base de toutes les autres cité ci-dessus. Elle permet de créer des tableaux aux dimensions voulues, d'accéder aux cases voulues mais aussi à appliquer toutes sortes de calculs sur les données de ces derniers.

To apply functions on NumPy arrays :

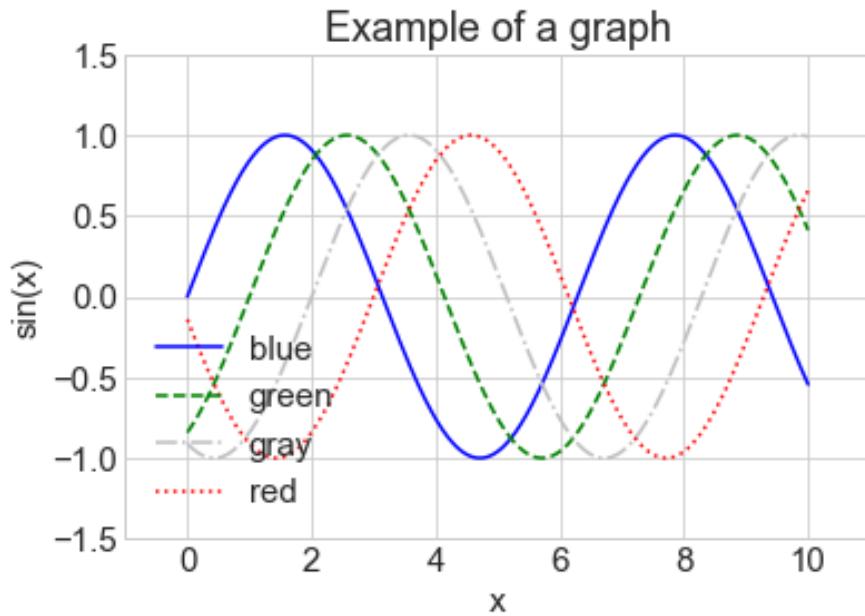
```

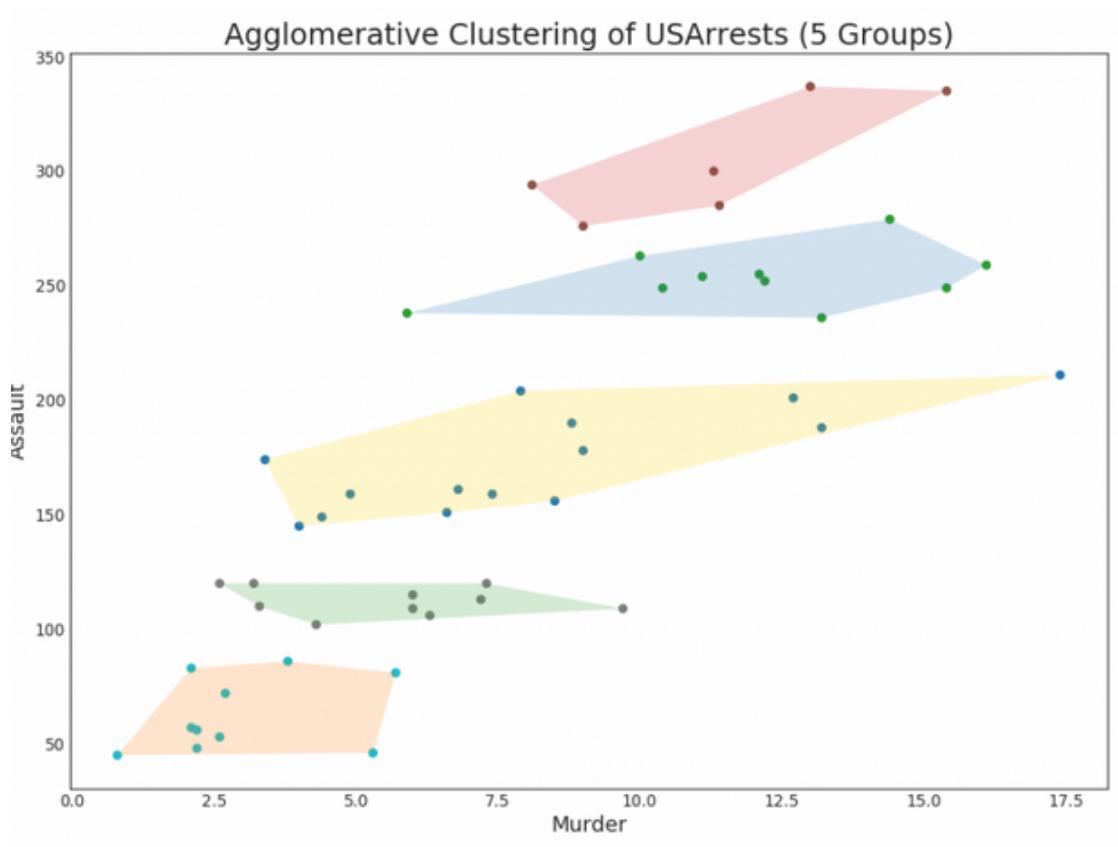
1 x = [-2, -1, 1, 2]
2
3 print("Absolute value: ", np.abs(x))
4 print("Exponential: ", np.exp(x))
5 print("Logarithm: ", np.log(np.abs(x)))

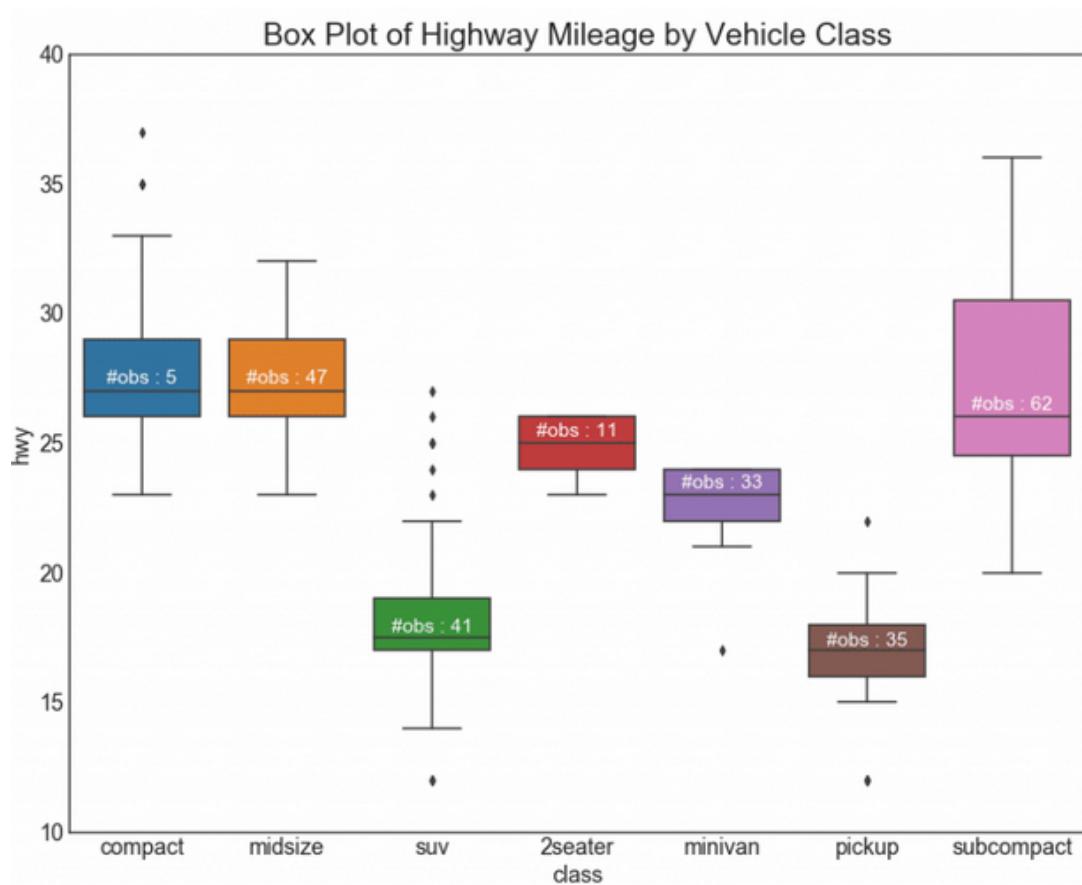
```

python

**Matplotlib** est une librairie qui sert à générer des graphiques directement depuis python. On peut créer toutes sortes de graphiques statiques, de quoi trouver son bonheur et ce qui explique le mieux possible nos données.







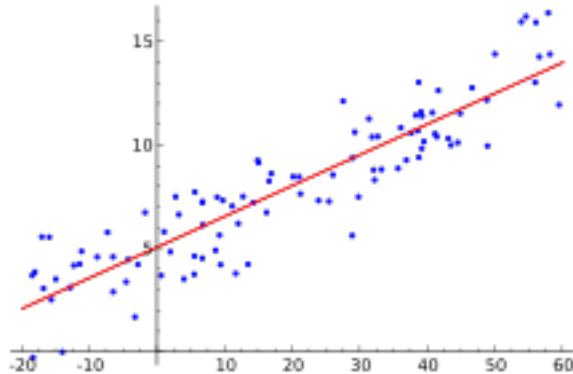
Il existe de nombreux types de graphiques possibles, ce site référence les 50 “meilleurs” : [Top50 Matplotlib visualizations](#)

**Plus de graphique avec Plotly et HoloViews** [Plotly](#) et [HoloView](#) sont d’autres librairies python servant à visualiser des données, elles viennent compenser Matplotlib, notamment grâce à la possibilité de créer des graphiques et schémas dynamiques voir même interactifs.

### Regression linéaire avec Scikit-Learn Qu'est-ce que la régrression linéaire ?

La régression linéaire est utilisée en statistiques pour déterminer l'influence d'une ou plusieurs données variables sur une évaluation qui se veut linéaire, en prenant en compte des paramètres aléatoires. Dans le cadre d'un modèle linéaire simple, on peut représenter graphiquement la relation entre x et y à travers un nuage de points.

L'estimation du modèle linéaire permet de tracer la droite de régression, d'équation  $y = \beta_0 + \beta_1 x$ . Le paramètre  $\beta_0$  représente l'ordonnée à l'origine et  $\beta_1$  le coefficient directeur de la droite.



**Scikit-Learn** Cette librairie permet de classifier des données, d'appliquer des regroupements, des régressions et plein d'autres méthodes statistique (Machine Learning, etc).

```
# Create linear regression object
regr = linear_model.LinearRegression()
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
```

[4]:

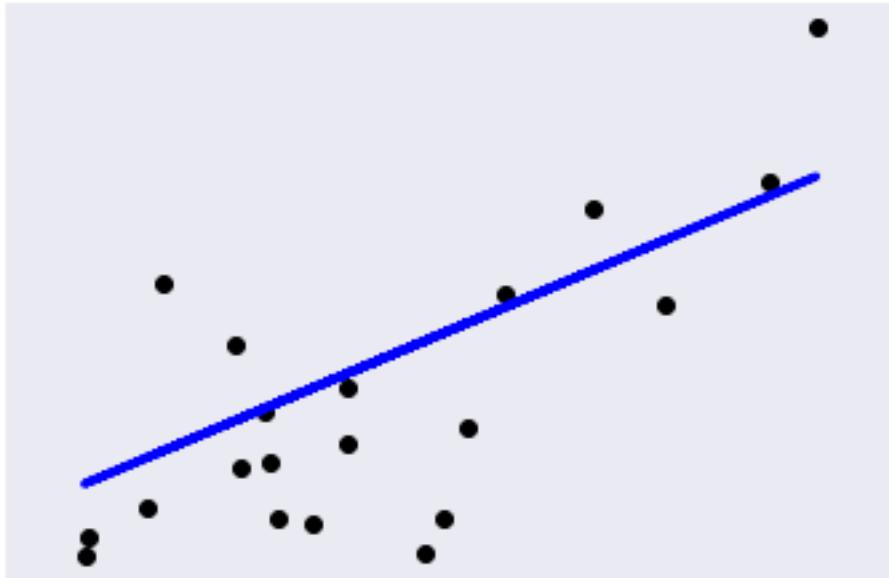
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
# Load the diabetes dataset
diabetes = datasets.load_diabetes()
# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
```

[5]:

```
# Create linear regression object
regr = linear_model.LinearRegression()
# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(diabetes_y_test, diabetes_y_pred))
```

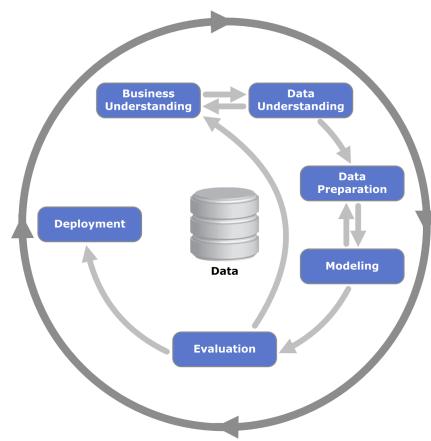
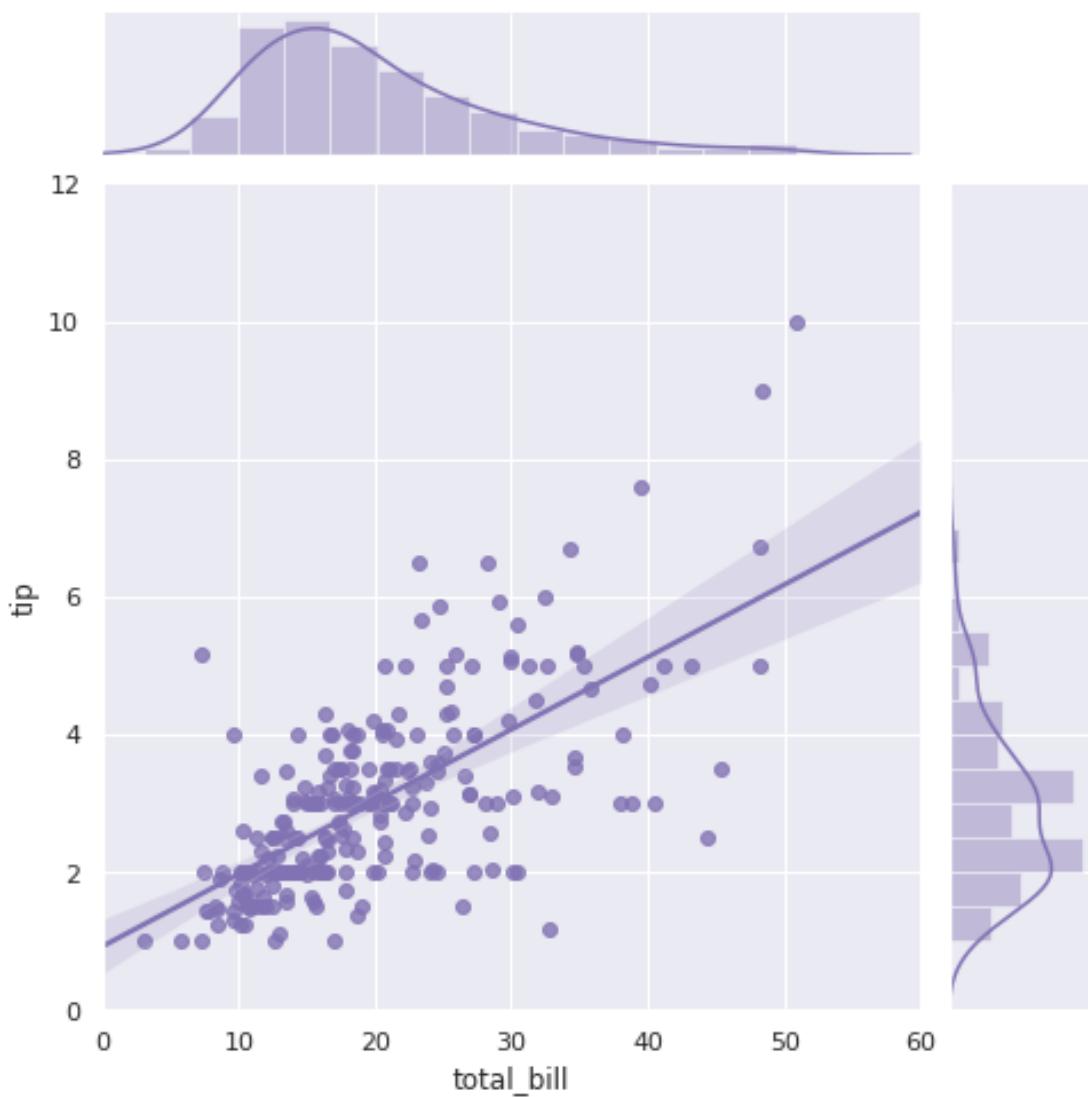
```
Coefficients:  
[938.23786125]  
Mean squared error: 2548.07  
Variance score: 0.47
```

```
[6]: # Plot outputs  
plt.scatter(diabetes_X_test, diabetes_y_test, color='black')  
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)  
plt.xticks()  
plt.yticks()  
plt.show()
```



La régression linéaire, peut plus simplement se faire avec seaborn, qui propose des graphiques avec régression intégré. Une méthode un peu plus simple pour une simple régression linéaire.

```
[7]: import seaborn as sns  
sns.set(style="darkgrid")  
tips = sns.load_dataset("tips")  
g = sns.jointplot("total_bill", "tip", data=tips, kind="reg",  
                  xlim=(0, 60), ylim=(0, 12), color="m", height=7)
```



Model de régression linéaire

### 3.3 Récupération de la donnée du temps d'attente

Section ??

```
import http.client
import pandas as pd
import json
from pandas.io.json import json_normalize
import numpy as np
from time import gmtime, strftime
import time

conn = http.client.HTTPSConnection("open-data.opt.nc")
conn.request("GET", "https://open-data.opt.nc/agences/_search?size=1000&q=pointAdresse:(Noum%C3%A8re+de+station)&sort=pointAdresse")
res = conn.getresponse()
data = res.read()
resultat = data.decode("utf-8")
jsonN = json.loads(resultat)
df = pd.DataFrame()
normalize = {}

normalize = {"nom": [jsonN['hits'][0]['_source']['designation']] ,
             "type": [jsonN['hits'][0]['_source']['type']] ,
             "id": [jsonN['hits'][0]['_id']] ,
             "adresse": [jsonN['hits'][0]['_source']['pointAdresse']] ,
             "AVGWaitingTime": [jsonN['hits'][0]['_source']['borneEsirius']['avgWaitingTime']] ,
             "MAXWaitingTime": [jsonN['hits'][0]['_source']['borneEsirius']['maxWaitingTime']] ,
             "#UPD": [jsonN['hits'][0]['_source']['updatedEsiriusDate']]}

for i in range (1,jsonN['hits'][0]['total']):
    normalize['nom'].append(jsonN['hits'][i]['_source']['designation'])
    normalize['type'].append(jsonN['hits'][i]['_source']['type'])
    normalize['id'].append(jsonN['hits'][i]['_id'])
    normalize['adresse'].append(jsonN['hits'][i]['_source']['pointAdresse'])
    try:
        normalize['AVGWaitingTime'].append(jsonN['hits'][i]['_source']['borneEsirius']['avgWaitingTime'])
        normalize['MAXWaitingTime'].append(jsonN['hits'][i]['_source']['borneEsirius']['maxWaitingTime'])
        #normalize['UPD'].append(jsonN['hits'][i]['_source']['updatedEsiriusDate'])
    except:
        print("An exception occurred, no borneEsurius for the ",i," station")
        normalize['AVGWaitingTime'].append('')
        normalize['MAXWaitingTime'].append('')
        #normalize['UPD'].append('null')

df = pd.DataFrame(normalize)
df['date'] = strftime("%Y-%m-%d %H:%M:%S", time.localtime())
```

```

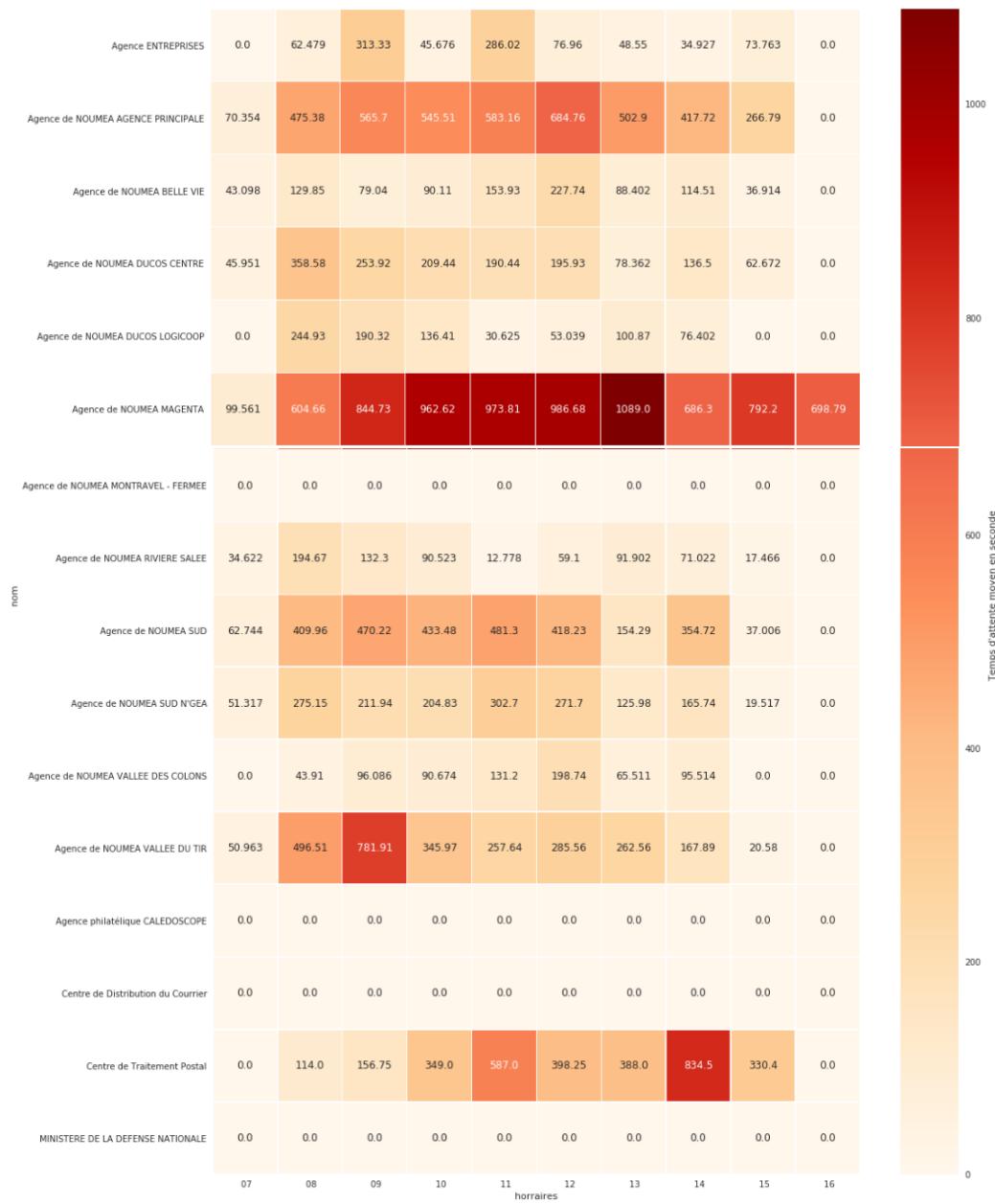
try:
    data = pd.read_csv('OPTWaitingTime.csv')
    print('OPTWaitingTime.csv is here !')
    #data[data==""] <- NA
    res = False
    for i in range (df['date'].size-1,0,-1):
        if data['date'].iloc[i] == strftime("%Y-%m-%d %H:%M:%S", time.localtime()):
            print("Point break")
            res = True
            break
    if not res:
        df = data.append(df)
except ValueError:
    print("no file OPTWaitingTime.csv")

df.to_excel(r'OPTWaitingTime.xlsx', index=False)
df.to_csv(r'OPTWaitingTime.csv', index=False)

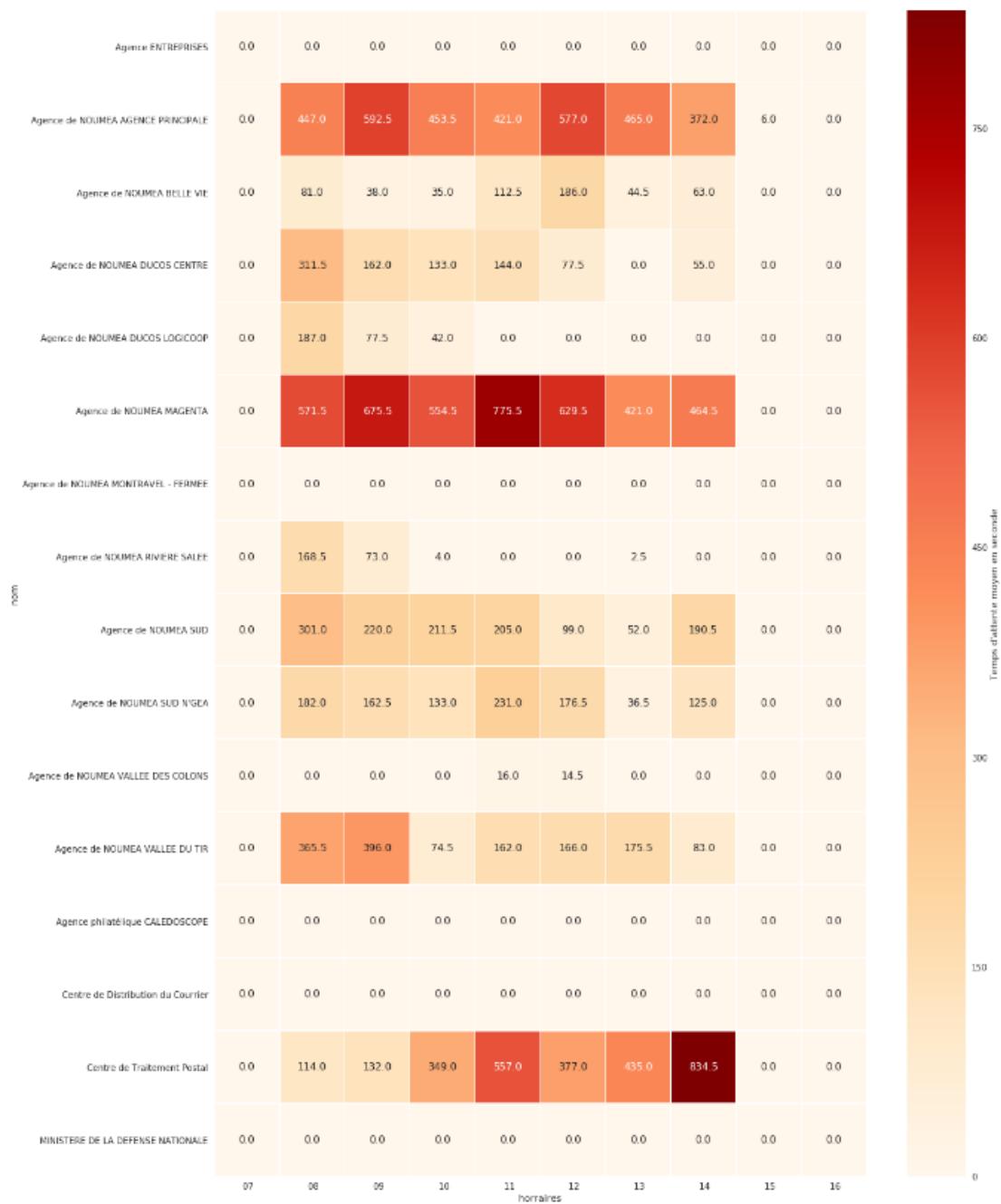
```

### 3.4 Heatmap OPT

Section ??



Bien entendu toutes ces moyennes sont plutôt bien représentative, mais tout de même à prendre à la légère et observer la médiane ainsi que l'écart type. Grâce à ce schéma, on observe très clairement quelles sont les agences les plus visitées et sur quels créneaux horaires. Cela nous permet de voir aussi quelles agences sont ouvertes au public et celles qui ne le sont pas. A contrario, la médiane nous montre aussi qu'à certaines horaires, il peut y avoir beaucoup d'attente, comme aucune attente : par exemple MAGENTA A 15H. Mais la médiane nous montre aussi que nos données ne sont pas faussé par de faux résultats.



Ce schéma suivant nous permet d'observer le temps d'attente moyen en seconde suivant les agences et le jour de la semaine. On y voit un temps d'attente plus long le lundi et mardi pour l'agence de magenta par exemple.



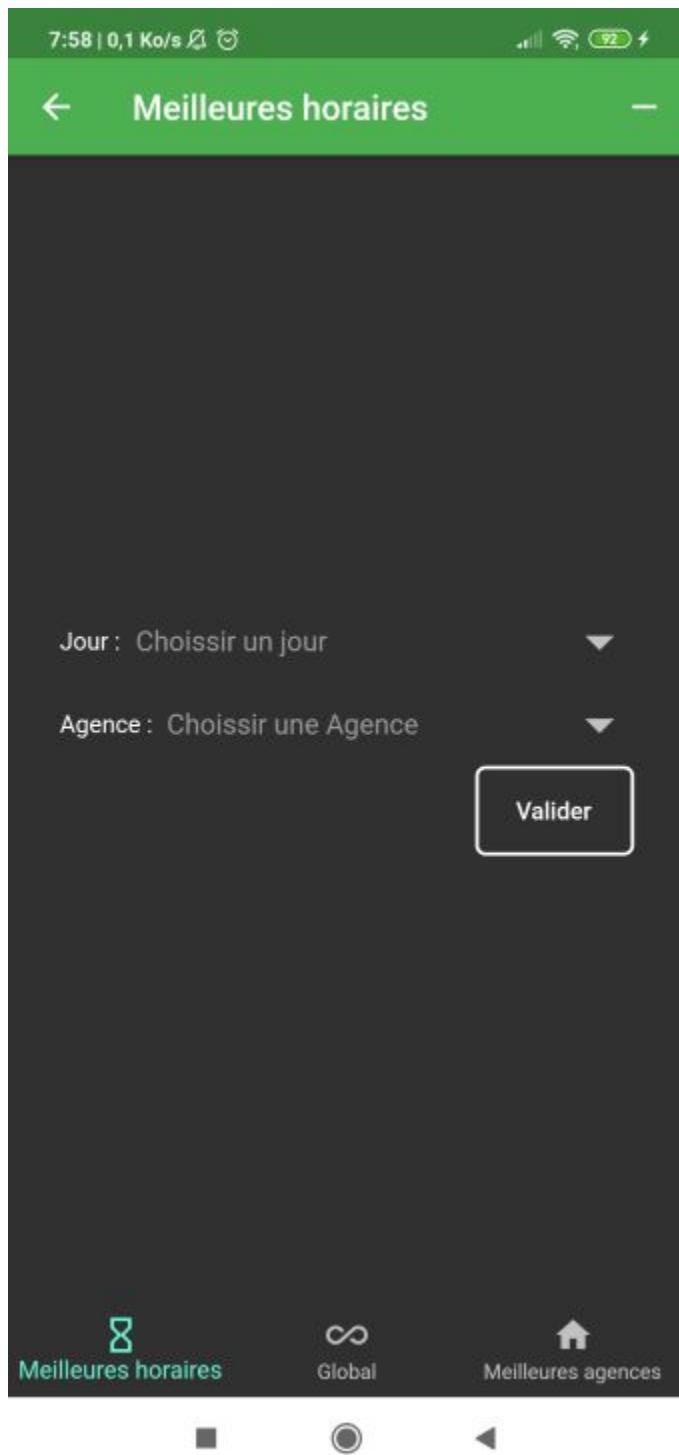
### 3.5 QR Code Site Web

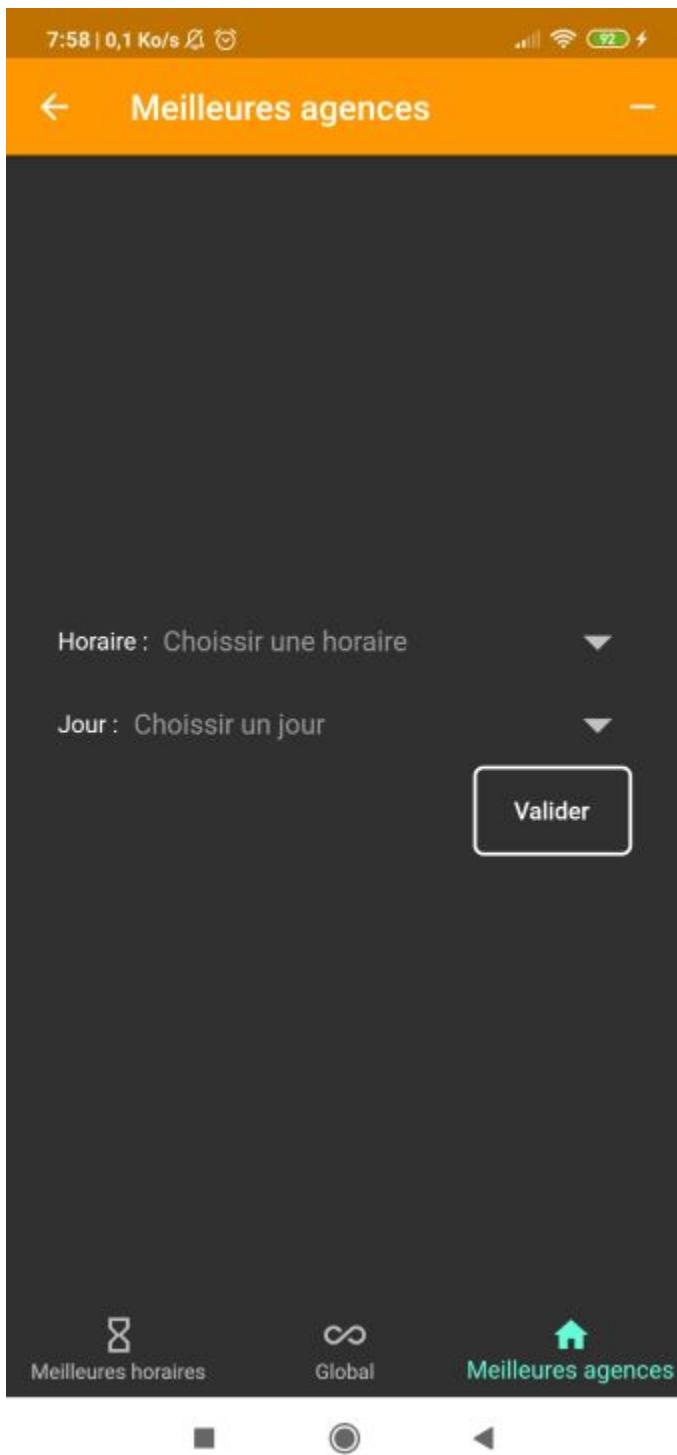
Section ??

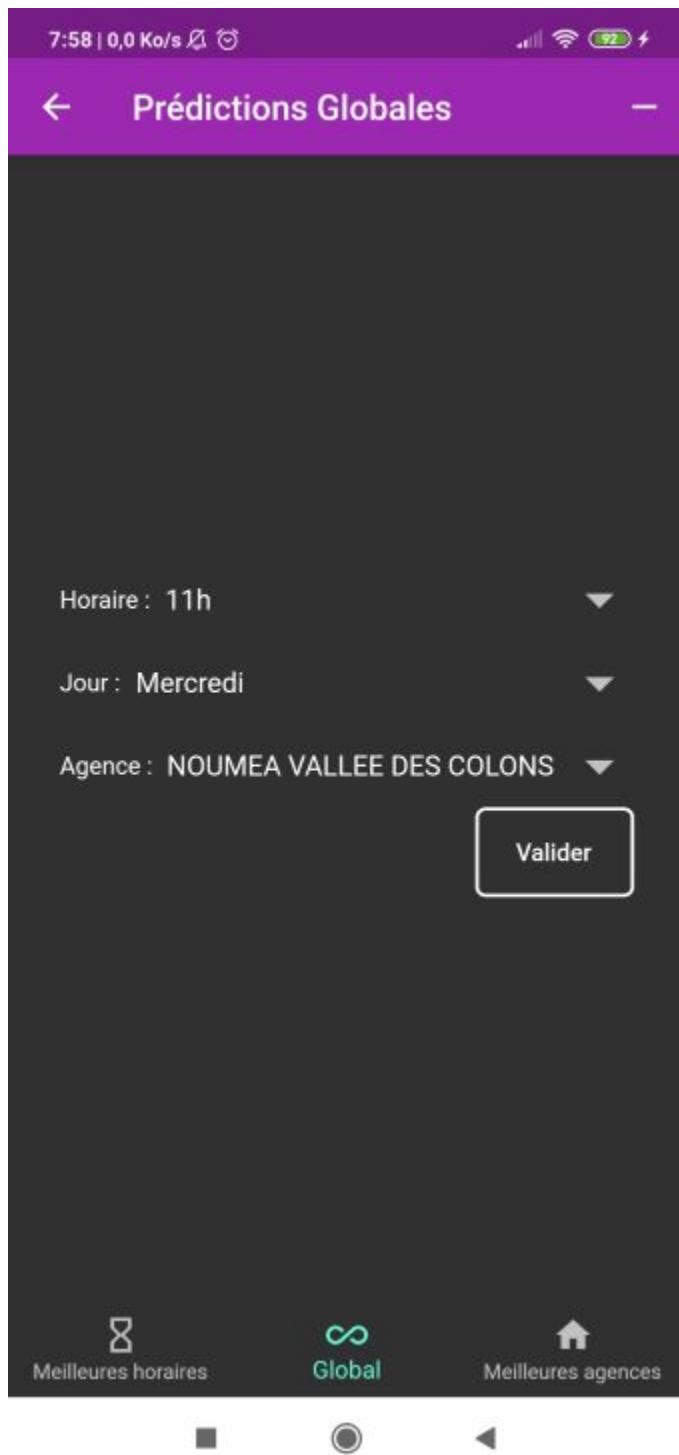


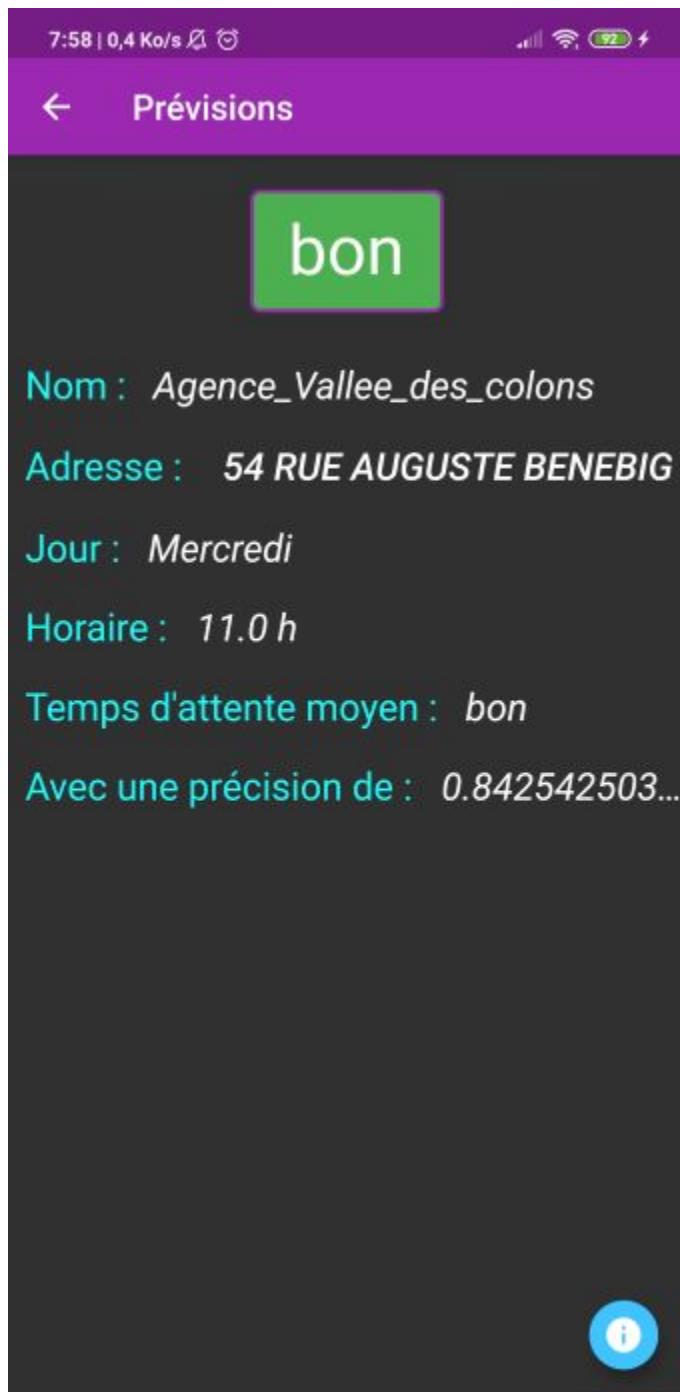
### 3.6 ScreenShots Application Mobile

Section ??











Télécharger l'app