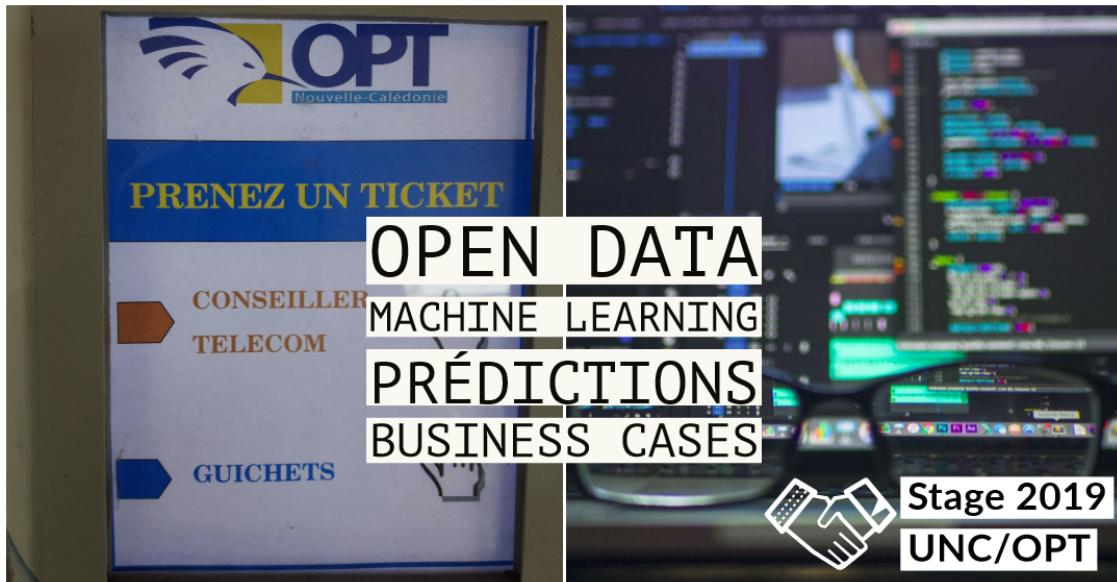


Rapport

December 16, 2019



1 PRESENTATION

1.1 Sujet du stage

L'objectif initial du stage est d'analyser des données de Scal'Air NC, obtenue 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'innovation à l'OPT.

- Lien Sujet du stage
- Lien Présentation reveal.js
- Lien Rapport sur Git

1.2 Apprentissage avec OpenClassRooms et autres

Section 2.1

1.3 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**.

1.3.1 Comment l'indice Atmo est-il calculé ?

Les concentrations maximales horaires d'ozone, de dioxyde de soufre et de dioxyde d'azote, ainsi que les concentrations maximales journalières de particules fines, sont mesurées durant 24 heures. La moyenne de ces concentrations permet ensuite de déterminer les sous-indices des quatre polluants. À chaque sous-indice de polluant correspond une valeur d'indice Atmo. L'indice Atmo retenu est celui correspondant au plus élevé des quatre sous-indices de polluant atmosphérique. Il est possible de faire des prévisions concernant la qualité de l'air du lendemain.

1.3.2 Première observations

Un exemple du jeu de données :

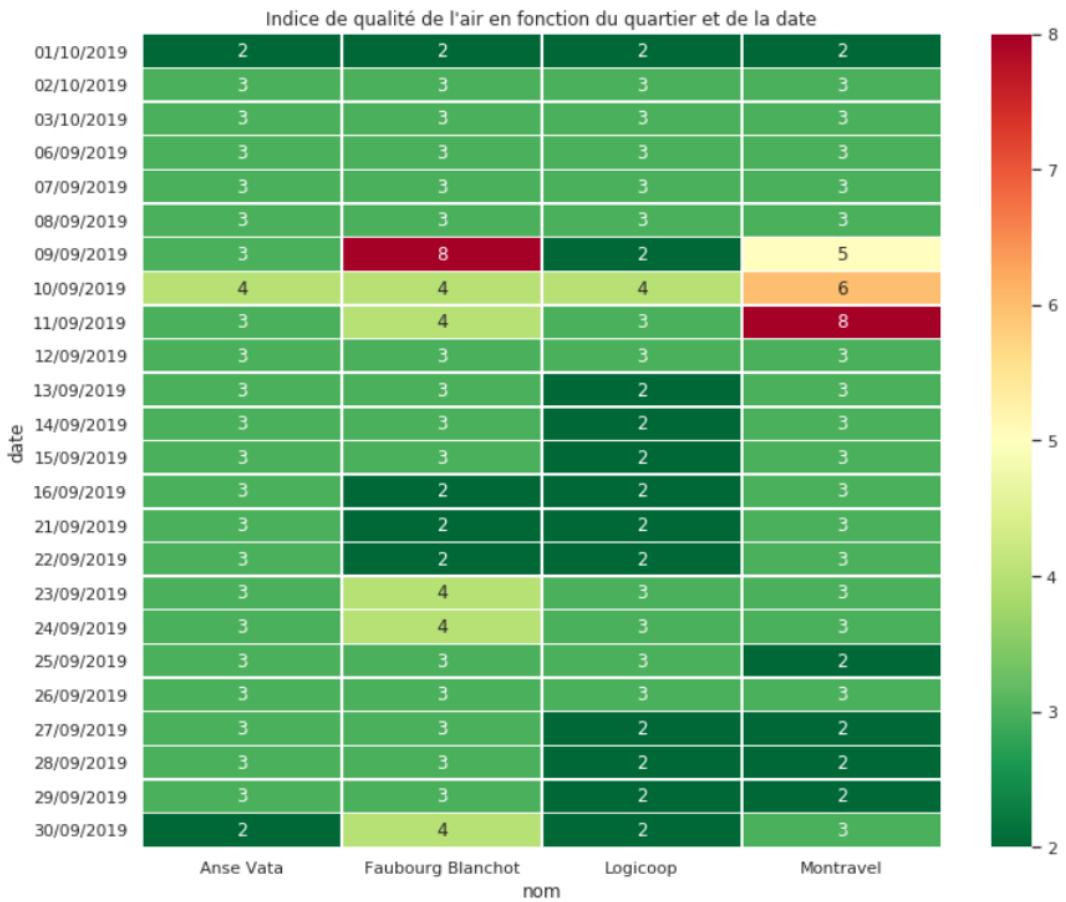
On voit ici un DataFrame créé par python.

	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 premiers graphiques :

Ces premiers graphiques, que l'on peut retrouver en Section ??, nous montre une visualisation de l'indice ATMO par date. On peut y voir une légère augmentation de ce dernier pour 2 des quartiers pour une date précise. Rien qu'avec cela on peut émettre hypothèse, notamment sur l'alignement de certains quartiers ainsi que l'influence du vent sur l'indice ATMO de ces quartiers.

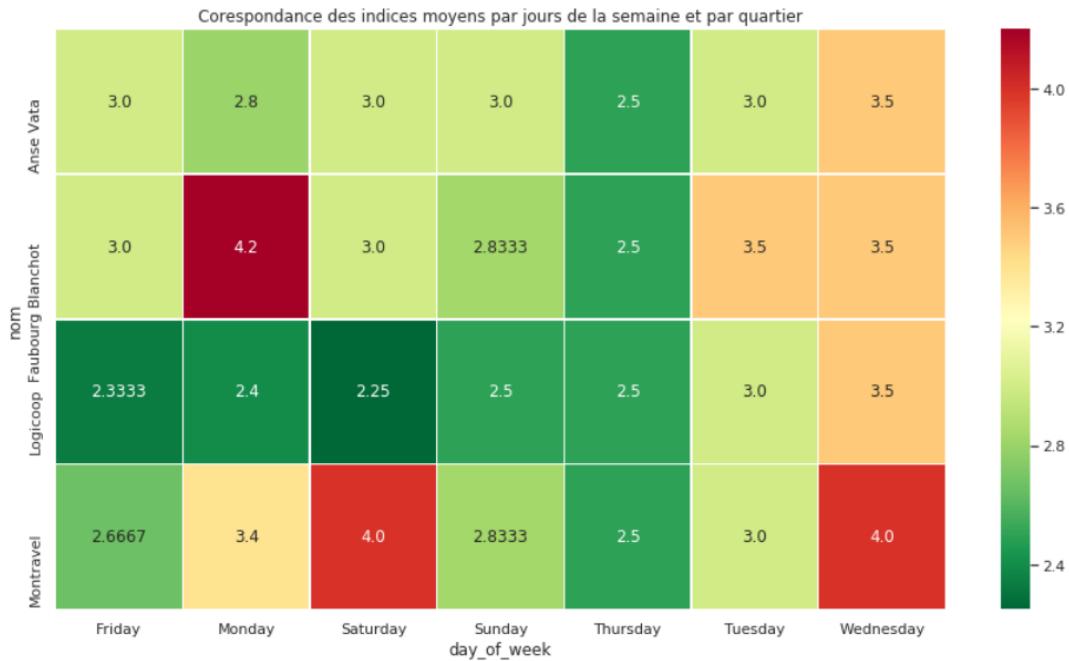
1.3.3 Modification et ajout de données



Suppression de la ligne *GÃ¢l'nÃ¢l'râle*

Day of The Week En rajoutant les jours de la semaine, cela nous permet par la suite de classer les indices en fonctions des jours et peut-Ãªtre trouver une corrÃ©lation entre ces derniers et l'indice de pollution. Par exemple, avec plus de donnÃ©es il se pourrait qu'on trouve un indice plus Ã‰levÃ© le mardi si la SLN nettoie ses cheminÃ©es (ceci est seulement un exemple imagÃ©).

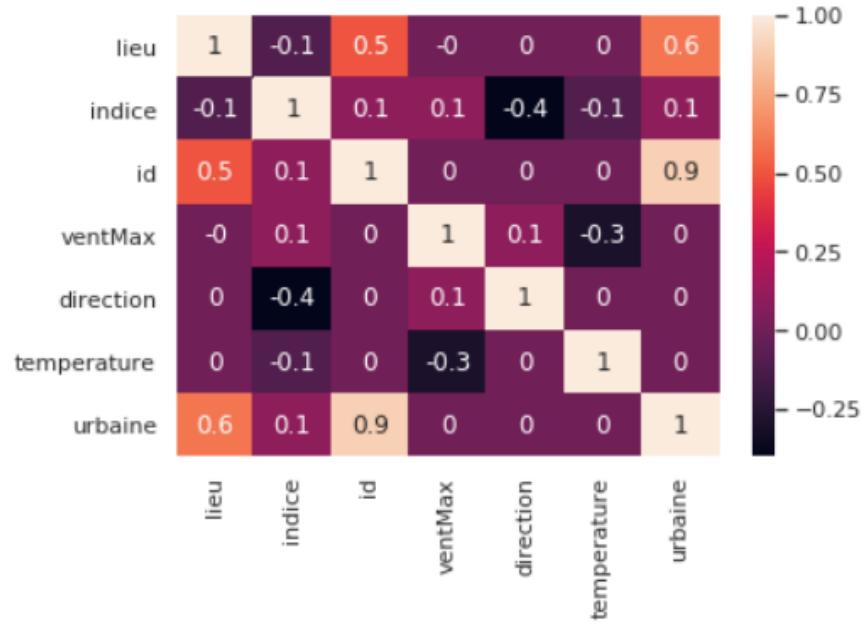
DiffÃ©rente perception de la donnÃ©e :



Vent & Température Ces données seront pour le moment rajouter sur le .xlsx à la main, nous verrons plus loin comment automatiser cette rentrée de donnée via une API

	typologie	lieu	nom	indice	message	color	id	date	ventMax	direction	temperature	urbaine
91	URBaine_INDUSTRIELLE	2	Montravel	3	Bon	GREEN	98002	03/10/2019	15	135	20	0
92	INDUSTRIELLE	5	Logicoop	3	Bon	GREEN	98001	04/10/2019	15	135	20	0
93	URBAINE	4	Faubourg Blanchot	3	Bon	GREEN	98003	04/10/2019	15	135	20	1
94	URBAINE	7	Anse Vata	3	Bon	GREEN	98004	04/10/2019	15	135	20	1
95	URBaine_INDUSTRIELLE	2	Montravel	3	Bon	GREEN	98002	04/10/2019	15	135	20	0

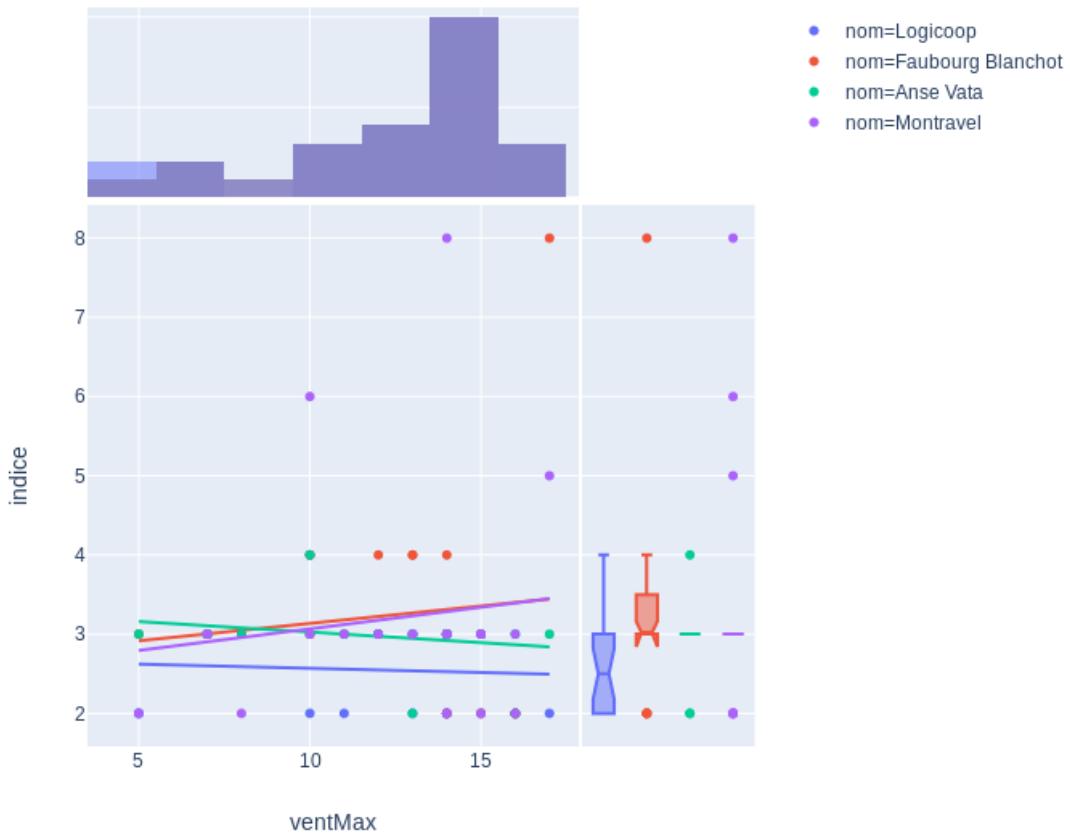
Malgré la faible quantité de données que nous possédons pour le moment, la matrice de corrélation montre quand même une certaine corrélation entre nos attributs



Et grÃ¢ce Ã ce schÃ¢ma la on peut distinguer un lÃ¶ger changement de la qualitÃ¶ de l'air en fonction de la force du vent

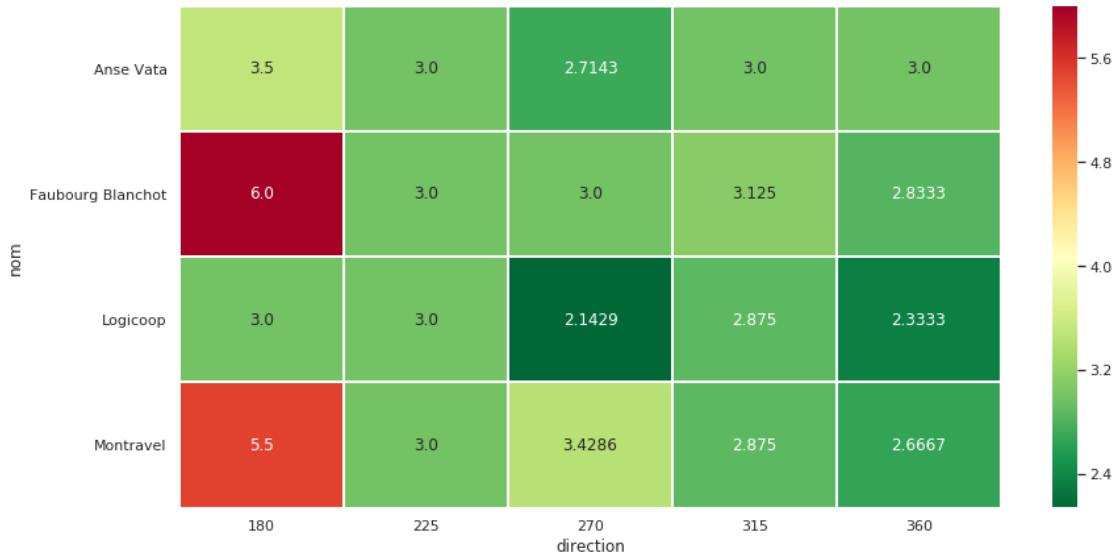
On peut donc supposer que la ou la droite est croissante, les quartiers sont alignÃ©s par rapport Ã la direction moyenne du vent ou trÃ©s proche.

On peut aussi l'observer clairement grÃ¢ce au heatmap suivant



```
[3]: import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

sns.set()
dataScal2_df = pd.read_csv('ScalairNum.csv')
dataScal2_df_Pivot = dataScal2_df.pivot_table('indice', index='nom', ↴
    columns='direction')
fig, ax = plt.subplots(figsize=(14,7))
sns.heatmap(dataScal2_df_Pivot, annot=True, fmt=".5", linewidths=.5, ↴
    linecolor='white', cbar=True, cmap="RdYlGn_r")
# To display the heatmap
import matplotlib.pyplot as plt
plt.show()
```



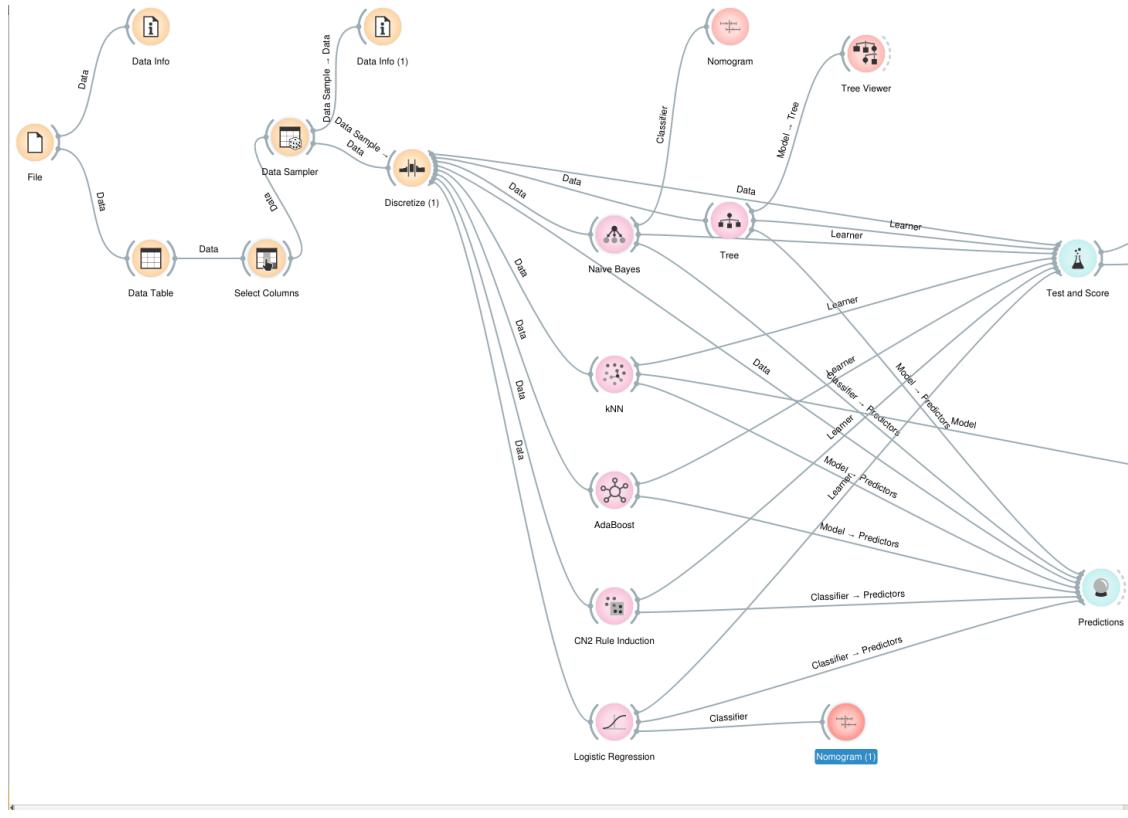
1.3.4 Des informations plus précises

Le graphique 3D, de la librairie Plotly, permet une meilleures compréhension de ces données et surtout, il permet d'observer le lien entre plus de données à la fois.

Section 2.3

1.4 Premier modèle 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 ne sachant pas coder.



1.4.1 Model Tree

Premièrement avant d'attaquer quoi que ce soit, il faut être bien sur 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 données avec une cible (qui est une variable du jeu de données) et qui va pouvoir s'entraîner sur d'autre données 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 privée-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.

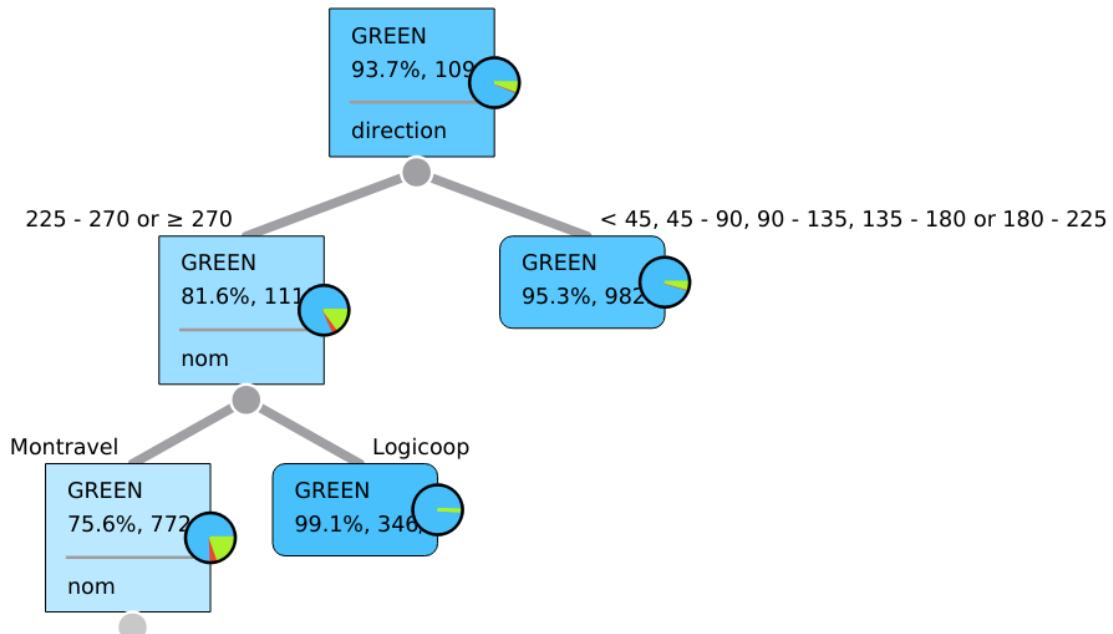
Le modèle Tree est un modèle de classification, l'objectif de notre arbre de décision est donc de classer les couleurs en 3 classes en fonction de leur indice ATMO. A chaque étape, l'algorithme va chercher le critère permettant de séparer au mieux ces 2 populations. Il existe plusieurs critères de séparation, ici la force du vent ainsi que sa direction. Dans notre exemple nous allons utiliser l'entropie.

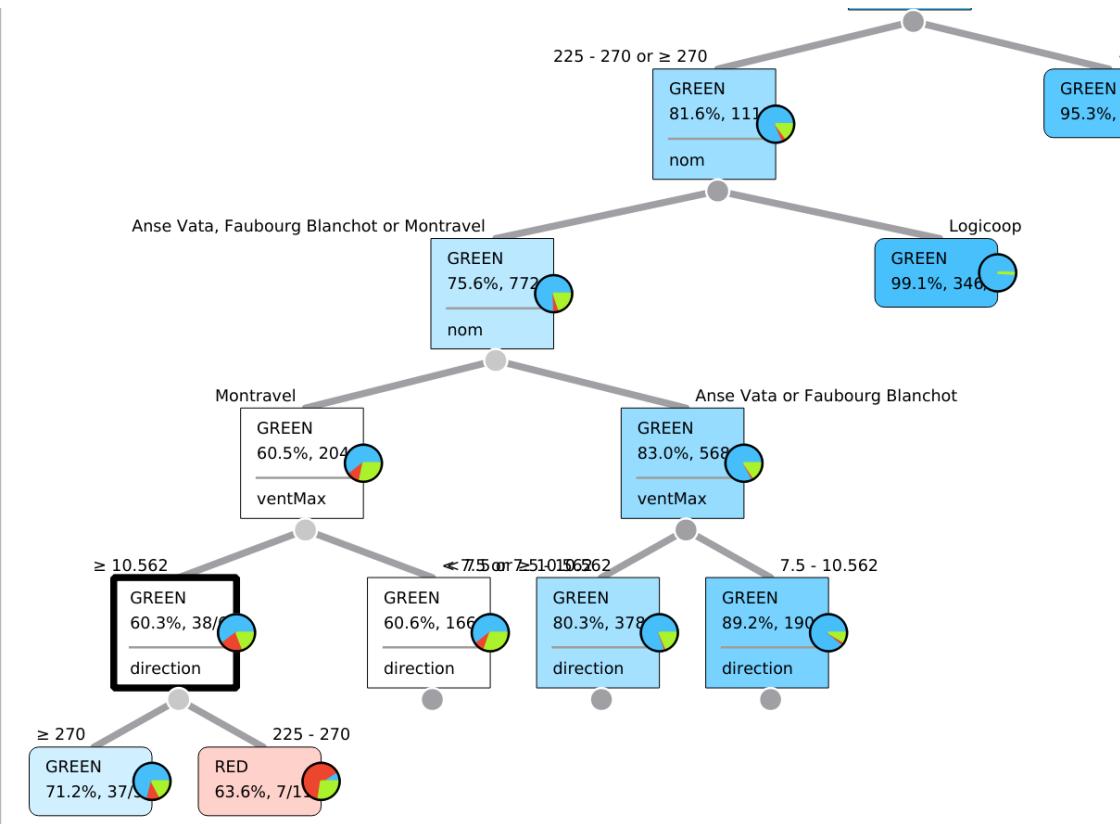
On aurait pu ici prendre un modèle de régression (comme la régression logistique qui est assez précise dans notre cas) mais, grâce à orange, nous avons pu comparer la précision entre les différents modèles et le modèle "Tree" est dans les plus précis, mais surtout il est le plus représentatif, notamment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

Evaluation Results

Method	AUC	CA	F1	Precision	Recall
Naive Bayes	0.756	0.937	0.906	0.877	0.937
Logistic Regression	0.753	0.937	0.906	0.877	0.937
Tree	0.641	0.937	0.907	0.883	0.937
AdaBoost	0.648	0.937	0.907	0.883	0.937
CN2 rule inducer	0.826	0.937	0.907	0.883	0.937
kNN	0.657	0.931	0.908	0.896	0.931

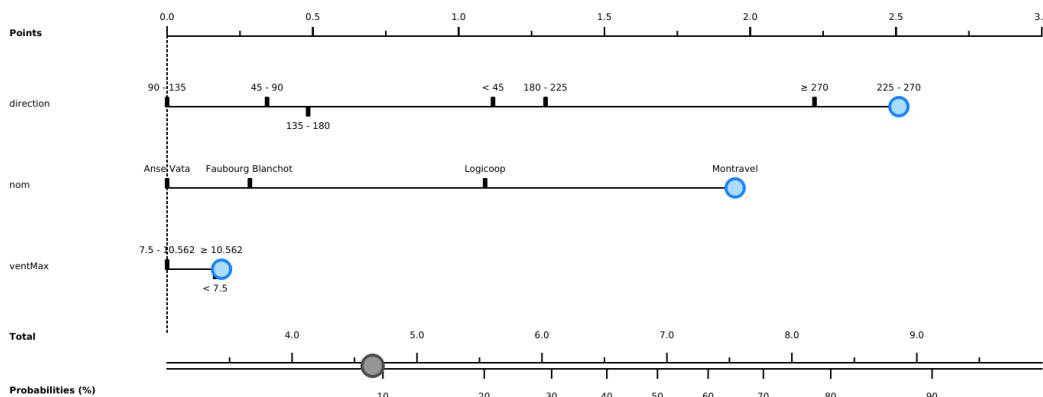
SchÃlma Model Tree Nous avons donc choisi de baser notre modÃle sur la cible "color" qui dÃlpartage donc les indices en 3 catÃgories : green, yellow and red. AprÃs avoir tester avec les indices seul, nous nous sommes rendu compte que la prÃcision Ãltait bien plus ÃlevÃe avec un choix de possibilitÃ plus restreint.





1.4.2 Model Logistic Regression

Ce modèle est à peine plus précis que celui de "Tree" et il a une représentation graphique sur Orange est différente. Elle permet de choisir une "target class" donc soit GREEN, YELLOW ou RED et de faire varier nos paramètres pour observer le pourcentage de chance qu'on a d'avoir cette couleur. Par exemple on peut savoir la probabilité d'avoir un indice dit "RED" sachant qu'on a un vent d'une forceMax supérieure à 11 noeuds, une direction entre Sud SudEst et ceux pour le quartier de Montravel :



Il est notable, que ce soit pour les modèles Orange ou directement créé en python avec Scikit-

Learn, que les modÃÃ©les peuvent Ãtre enregistrer en un fichier particulier et rÃlouvert par la suite dans un nouveau programme, ce que nous verrons dans une autre partie.

1.5 Recuperation de la donnÃÃ©e Ã l'aide d'une API REST

Comme Ãl'noncÃÃ© dans le sujet du stage, la donnÃÃ©e de Scal'Air est accessible via une [API REST](#) sur [Rapide API](#). Il nous a donc suffit d'utiliser une requÃÃ©te http en python pour rÃcupÃÃ©rer la donnÃÃ©e chaque jours. Au dÃÃ©but du stage, nous disposions de seulement 2 mois de donnÃÃ©es, que nous avons donc remplit petit Ã petit grÃÃ©ce Ã cette API. GrÃÃ©ce Ã cette requÃÃ©te HTTP, nous rÃcupÃÃ©rons donc de la donnÃÃ©e sous forme de JSON, il nous Ã donc fallut la traiter et l'enrichir. Pour l'enrichir, nous avons donc utilisÃÃ© une API (rest) d'Open Weather Map disponible aussi sur [RapidAPI](#).

Avec cet ensemble de donnÃÃ©e rÃcupÃÃ©rÃÃ©, il nous restait donc juste Ã assembler le tout et traiter un peu les formats de ces donnÃÃ©es. Rien de plus simple Ã faire que ÃgÃÃ© si l'on est ÃquipÃÃ© des librairies python : pandas et numpy.

```
1 import http.client
2
3 conn = http.client.HTTPSConnection("community-open-weather-map.p.rapidapi.com")
4 headers = {
5     'x-rapidapi-host': "community-open-weather-map.p.rapidapi.com",
6     'x-rapidapi-key': "f587dc6483msh84e28a7c9a650bcpl1b260jsn7c8b7ea489f5"
7 }
8 conn.request("GET", "/forecast?q=Noum%C3%A9e%C2NC&lang=French", headers=headers)
9 res = conn.getresponse()
10 data = res.read()
11 resultat2 = data.decode("utf-8")
```

```

def generationDataFrameWeather(resultat):
    import pandas as pd
    import json
    from pandas.io.json import json_normalize
    import numpy as np
    jsonN = json.loads(resultat)
    for i in range (8):
        normalize = {"ventMax": [jsonN['list'][i]['wind']['speed']],
                     "direction": [jsonN['list'][i]['wind']['deg']],
                     "temperature": [jsonN['list'][i]['main']['temp_max']],
                     }
        if i == 0:
            df = pd.DataFrame(normalize)
        else:
            df_Tmp = pd.DataFrame(normalize)
            df = pd.concat([df, df_Tmp], sort=False)
    np.datetime64('today') # today's date
    return df

```

```

df = generationDataFrameWeather(resultat2)
df['ventMax'] = round((df['ventMax']*(1.943844492)))
df['direction'] = round(df['direction'])
df['temperature'] = round(df['temperature']-273,15)
df['temperature'] = round(df['temperature'])

```

```

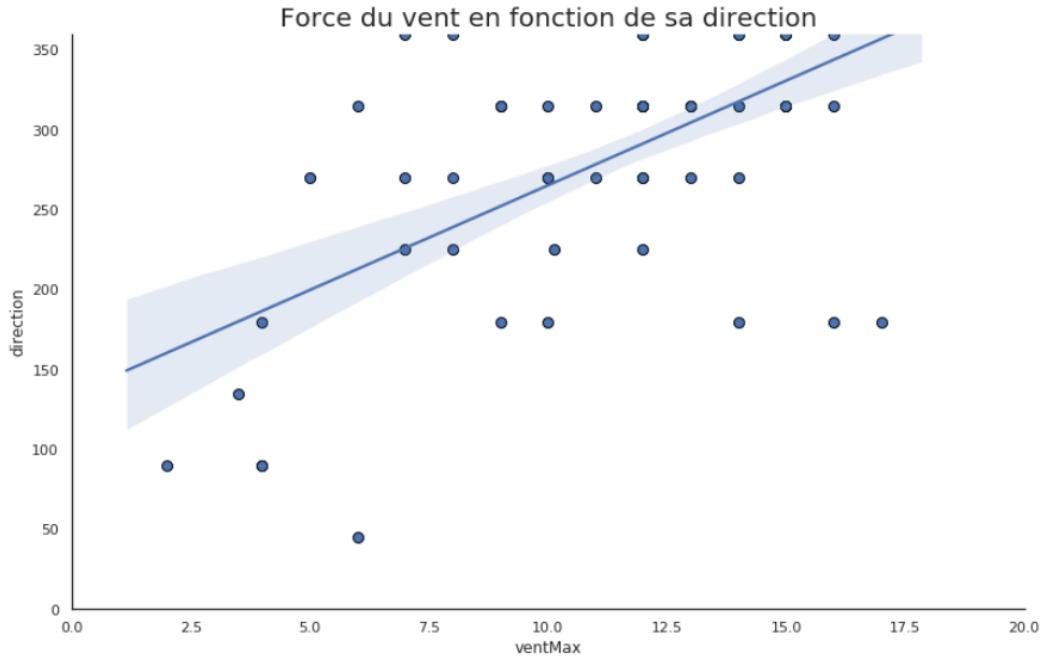
vent = sum(df['ventMax'])/df['ventMax'].size
deg = sum(df['direction'])/df['direction'].size
temp = max(df['temperature'])
normalize = {
    "ventMax": [vent],
    "direction": [deg],
    "temperature": [temp],
}
df_Final = pd.DataFrame(normalize)

```

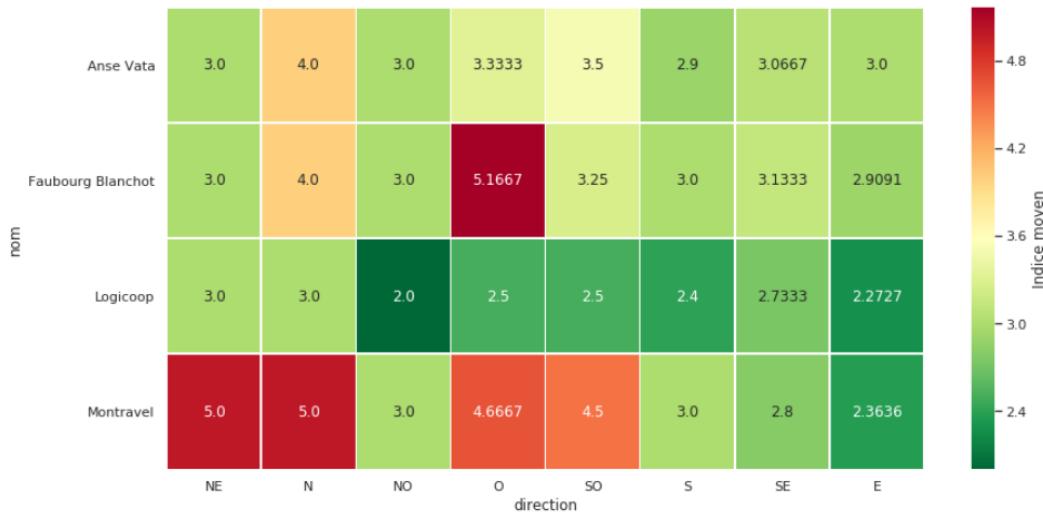
95URBAINE_INDUSTRIELLE	MONTRAVEL	Montravel	3 Bon	GREEN	98002	2019-10-03	15.0	315.0	20.0
96INDUSTRIELLE	LOGICOOP	Logicoop	2 Très bon	GREEN	98001	2019-10-11	3.5	135.0	26.0
97URBAINE	FAUBOURG_BLANCHOT	Faubourg Blanchot	3 Bon	GREEN	98003	2019-10-11	3.5	135.0	26.0
98URBAINE	ANSE_VATA	Anse Vata	3 Bon	GREEN	98004	2019-10-11	3.5	135.0	26.0
99URBAINE_INDUSTRIELLE	MONTRAVEL	Montravel	3 Bon	GREEN	98002	2019-10-11	3.5	135.0	26.0

1.6 Suite de l'Analyse des données de Scal'Air

Avec ces données supplémentaires nous avons pu observer de nouvelles informations et valider ou non des hypothèses précédentes prétendument. On observe à l'aide de nombreux schémas seaborn ou matplotlib que la direction du vent et corrélée avec sa force



De plus on voit aussi que l'indice ATMO est corrélée avec la direction, ce qui semble tout à fait logique compte tenu de la géolocalisation des cartiers étudiés.

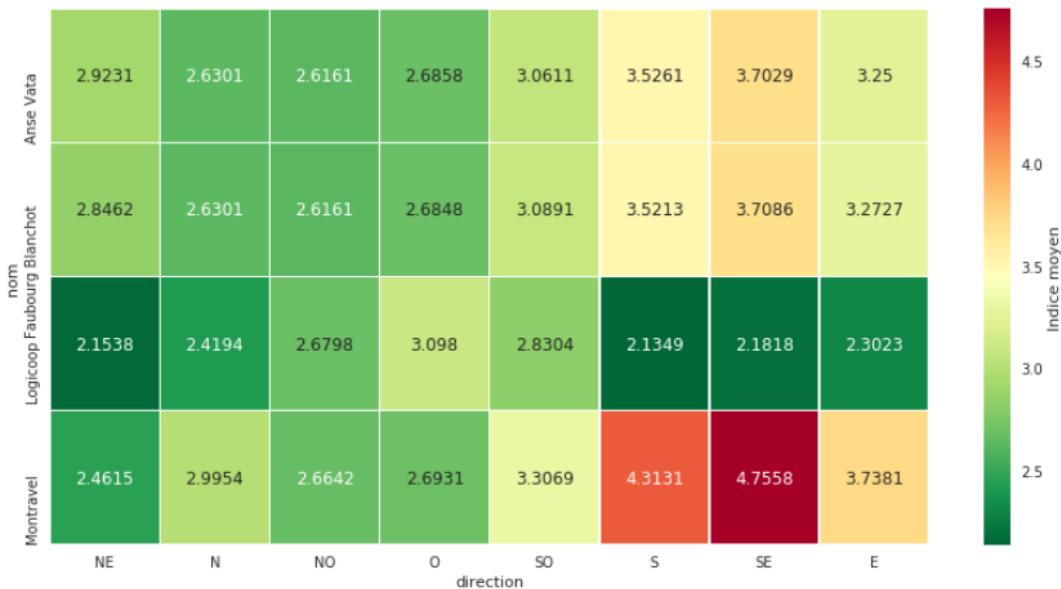


Par exemple nous pouvons émettre l'hypothèse que lorsque le vent vient du NE ou du N, le vent est souvent très faible, ce qui, si l'on part du principe que le principale pollueur de l'air sur Nouméa est la SLN, semble très plausible. Le vent ne portant pas les particules au loin, celles-ci resteraient donc aux alentours de la SLN et donc de Montravel. Pour ce qui est de l'augmentation de l'indice ATMO pour les quartier Faubourg Blanchot et Montravel lors des vents venant d'Ouest et du SudOuest, ces derniers étant généralement assez élevés, il se pourrait que la pollution venue de la SLN soit reportée sur Montravel ainsi qu'au Faubourg Blanchot. Après toutes ces séparations, les hypothèses sont validées, en effet, notre jeu de données est encore très mince, ce qui ne nous permet en aucun cas d'avancer des certitudes. Nous avions aussi, lorsque nous avions

que les deux premiers mois de données, avancerait l'hypothèse qu'il pouvait se passer des activités particulières à la SLN suivant les jours. Et notre premier schéma nous l'exposait bien, en effet l'indice ATMO l'était plus élevé en moyenne le vendredi. Mais grâce aux nouvelles données rentrées via l'API, le nouveau schéma démontre cette hypothèse.

1.6.1 Toujours plus de données

Grace à certaines connaissance, nous avons pu recuperer les données de Scal'Air depuis 2008, ce qui a pu nous permettre de refaire tous nos graphes d'observation avec cette nouvelle donnée pour pouvoir valider ou non nos hypothèses. Ce que nous pouvons en déduire c'est que ces données sont très aléatoires et ne sont pas vraiment corrélées avec le vent et sa direction. Ceci pourrait aussi s'expliquer par notre moyen de récupération des données météorologiques. En effet la seule API trouvée qui permettait d'obtenir l'historique météorologique depuis 2008 nous donnait seulement la moyenne de la direction et de la force du vent par jour. Mais sans se pencher sur cette erreur possible, nos nouveaux graphes nous montrent que non seulement notre hypothèse du jour de la semaine plus pollué que les autres est faux et surtout que ce n'est pas seulement la SLN qui pollue.



En effet ce graphe nous montre bien que lorsque le vent est Sud / SudEst la pollution est plus élevée à Montravel certes, mais aussi sur l'Anse Vata, qui est pourtant dans la direction opposée à la SLN. Ce que nous pouvons en déduire c'est que, lors de vent Sud / SudEst, où le vent est souvent plus fort, la pollution pourrait venir d'ailleurs, comme d'Australie ou de Nouvelle Zelande.

1.7 Création d'une API REST

Le principe de base de cette API est donc, suivant des paramètres passés lors de la requête HTTP (force du vent / direction du vent / lieu), de nous retourner une prédiction de l'indice

ATMO. En testant la création d'un modèle sur orange et de son exportation, nous n'avons pas réussi à utiliser ce modèle en python, alors nous nous sommes penchés sur la question. Et nous avons décidé de créer notre propre modèle en python à l'aide de Scikit-Learn. Malgré la quantité de données relativement importante, notre modèle de prédiction, qu'il soit de classification ou de régression, n'était pas du tout précis (Précision = 0,6). Mais nous l'avons quand même utilisé pour travailler sur le principe, qui sera reproduisible avec un modèle plus précis si un jour nous en avons un, ou sur un jeu de données différent. Malgré tout, nous arrivons quand même à générer un modèle d'une précision de 0,86 mais sur une cible qui est la "color". Le problème étant que pour la dernière, ce modèle ne retourne que "GREEN".

```
In [45]: feature_cols = ['ventMax','nom', 'direction']
X = data[feature_cols] # Features
Y = data['color'] # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=1) # 70% training and 30% test

In [46]: # Create Decision Tree classifier object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)
# Train Decision Tree Classifier
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

In [47]: # Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

Accuracy: 0.859154929577
```

```
In [56]: # Import pandas library
import pandas as pd

# initialize list of lists
data = [[1,1,170.0]]

# Create the pandas DataFrame
test = pd.DataFrame(data, columns = ['ventMax', 'nom','direction'])

In [57]: clf.predict(test)

Out[57]: array(['GREEN'], dtype=object)
```

Pour revenir à la création de notre API REST, nous avons simplement dans un code python importé un modèle déjà créé et entraîné auparavant.

```
Entrée [60]: 1 import pickle
2 # save the model to disk
3 filename = 'finalized_model.sav'
4 pickle.dump(clf, open(filename, 'wb'))
```

Au final, il s'agit d'un script python qui utilise la bibliothèque Flask, qui permet simplement d'assigner une réponse à une requête http. En plus de cela il nous suffit de créer une classe ayant pour attribut nos paramètres du modèle (wind / place / direction) et importer le modèle.

```

class Prevention:
    vent = 0.0
    nom = 0
    direction = 0.0
    def __init__(self,vent,nom,direction):
        self.vent=vent
        self.nom=nom
        self.direction=direction

    def run(self):
        import pandas as pd
        from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
        from sklearn.model_selection import train_test_split # Import train_test_split function
        from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
        import pickle
        # initialize list o Loading...
        data = [[self.vent,self.nom,self.direction]]
        # Create the pandas DataFrame
        test = pd.DataFrame(data, columns = ['ventMax','nom','direction'])
        #filename = 'SaveModelOrange.pkcls'
        filename = 'finalized_model.sav'
        loaded_model = pickle.load(open(filename, 'rb'))
        return loaded_model.predict(test)

```

```

prev = Prevention(wind,place,direction)
tmp = prev.run()
dic={"wind":wind,
      "place":str(request.args['place']),
      "direction":direction,
      "indice":float(tmp[0])}

# Use the jsonify function from Flask to convert our list of
# Python dictionaries to the JSON format.
return jsonify(dic)

```

Et retournant sa réponse en Json.

Une fois ce Script Python lancé, il nous suffit d'ouvrir sur un navigateur le lien local host sur le port 5000 ainsi que les compléments d'URL et les paramètres pour obtenir une prediction et le Json renvoyé par le programme. Mais le but ne s'arrête pas là, ce que nous voulons c'est que cette API tourne h24 et que quiconque puisse y avoir accès et par n'importe quel moyen. Pour cela, nous avons du déployer cette API dans le cloud via heroku, puis sur un market place grâce à Rapide Api. Et nous avons aussi tester de récupérer les données en Java.

1.7.1 Heroku & RapidAPI

Heroku est donc une plateforme en ligne permettant d'héberger gratuitement des petites API. Pour publier la notre, à l'aide du CLI fournit par heroku sous linux, il nous à juste suffit 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 l'était disponible via une URL publique. Le travail restant l'était juste de publier ce lien sur Rapide Api et de documenter un peu notre API en y montrant des exemples. Finalement, grâce à un programme java, nous arrivons à recuperer notre prédiction et cela de n'importe où. De plus Rapide Api propose des exemples de code pour l'utilisation de l'api.

Code Java :

```
HttpRequest request = HttpRequest.newBuilder()
    .GET()
    .header("x-rapidapi-host", "api-predictionpollutionnc.p.rapidapi.com")
    .header("x-rapidapi-key", "f587dc6483mh84e28a7c9a650bcpllb260jsn7cb7ea489f5")
    .uri(URI.create("https://api-predictionpollutionnc.p.rapidapi.com/api/v1/prevision?wind=" + wind + "&place=" + place + "&direction=" + direction))
    .build();

HttpResponse<String> response = httpClient.send(request, HttpResponse.BodyHandlers.ofString());
```

Réponse du Java :

```
Wind value :
2
Place value :
Montravel
Direction value :
325,0
Testing 1 - Send Http GET request
200
{
  "direction": 325.0,
  "indice": 3.0,
  "place": "Montravel",
  "wind": 2.0
}

Process finished with exit code 0
```

1.8 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 5min 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.

1.8.1 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 organisation 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...

1.8.2 Recuperation de la donnée

Pour récupérer cette donnée sur les temps d'attentes, il nous a fallu 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 fallu l'exploiter en python, en faisant 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 ??

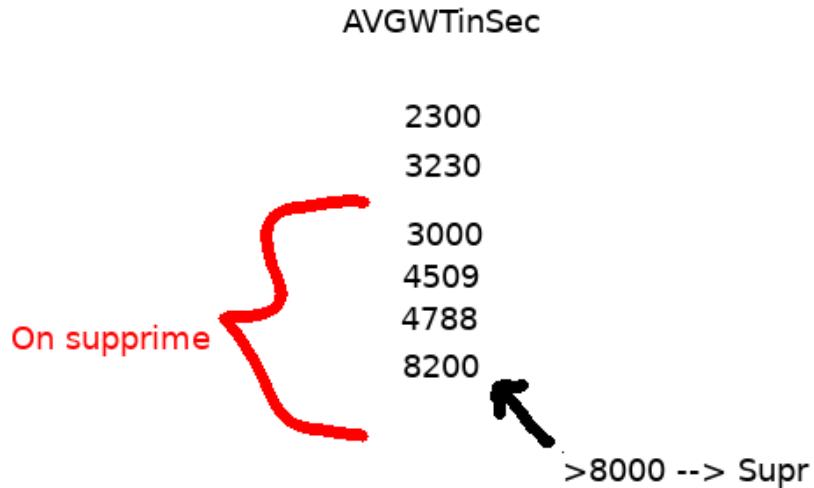
Autrement dit :

	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.

1.8.3 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. 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 près d'entres 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 à l'était un heatmap représentant le nombre de seconde d'attente moyenne pour chaque agences et chaque horaires.

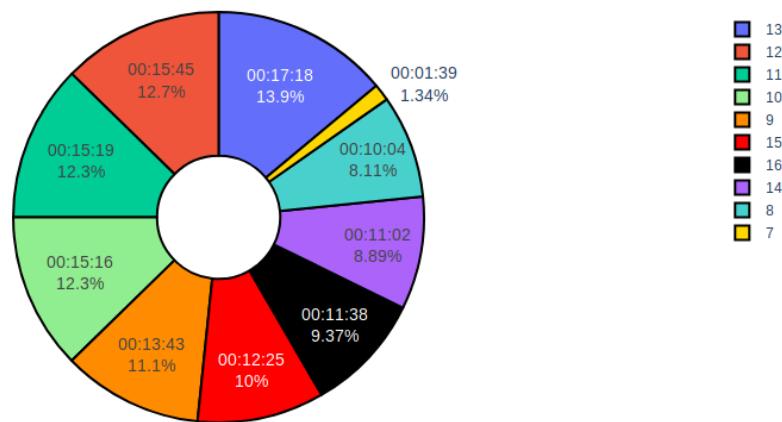
Section 2.5

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'histogramme. 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.

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 également 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



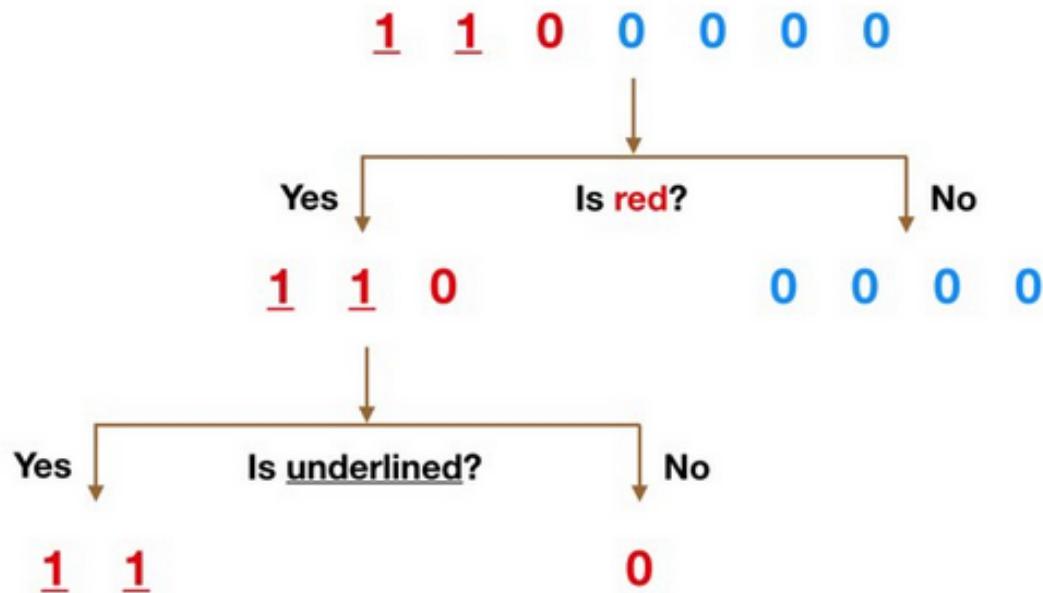
1.8.4 Algorithme de prédiction

Comme vu avec le jeu de donnée précédent, il est préférable, premièrement, de tester la validité des modèles de régression et de classification sur Orange. Comme expliquent les auteurs, cette étape a été faite et couplée à l'essai de plusieurs algorithmes sur Python avec Scikit-Learn, le modèle retenu et le 'RandomForestClassifier'. Avec notre petit jeu de données, 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)
```

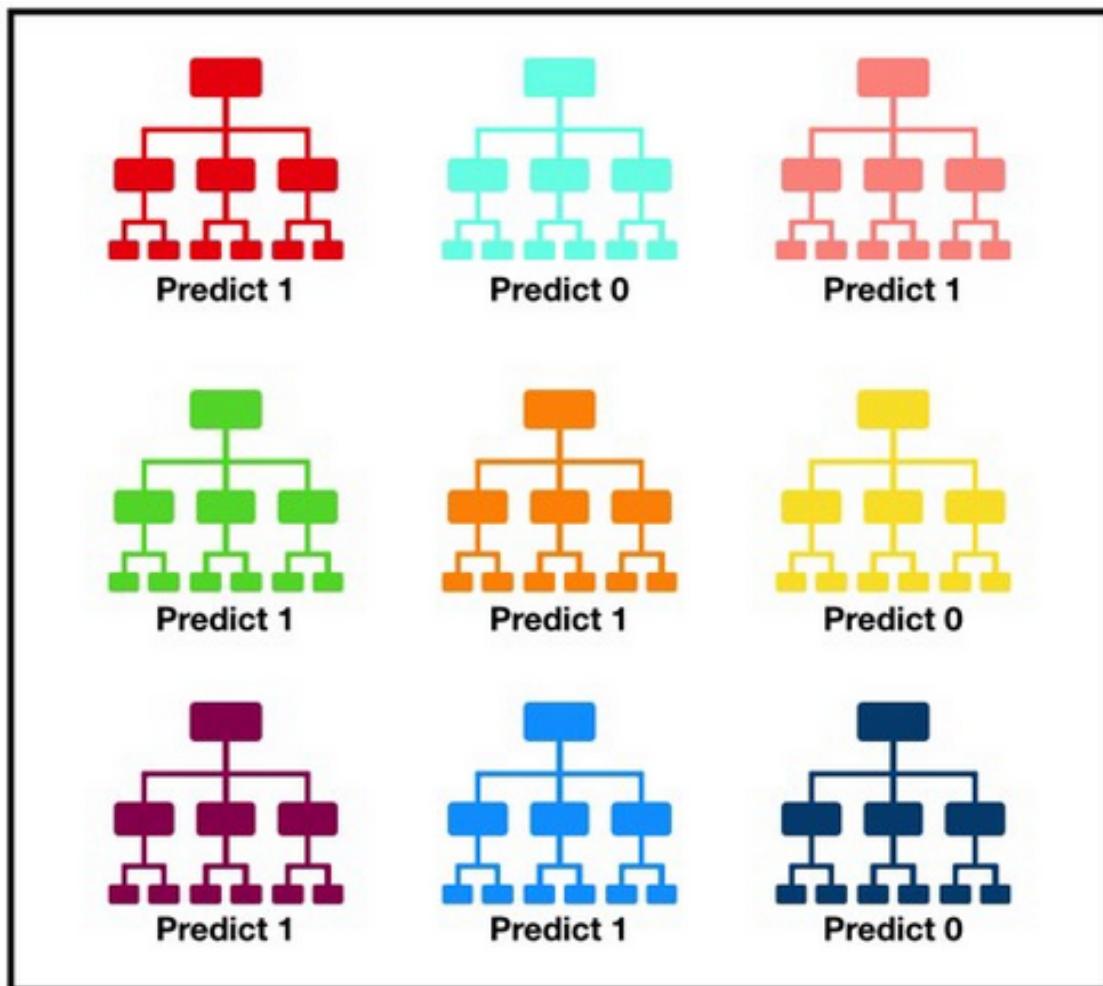
0.84038564542

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



Simple Decision Tree Example

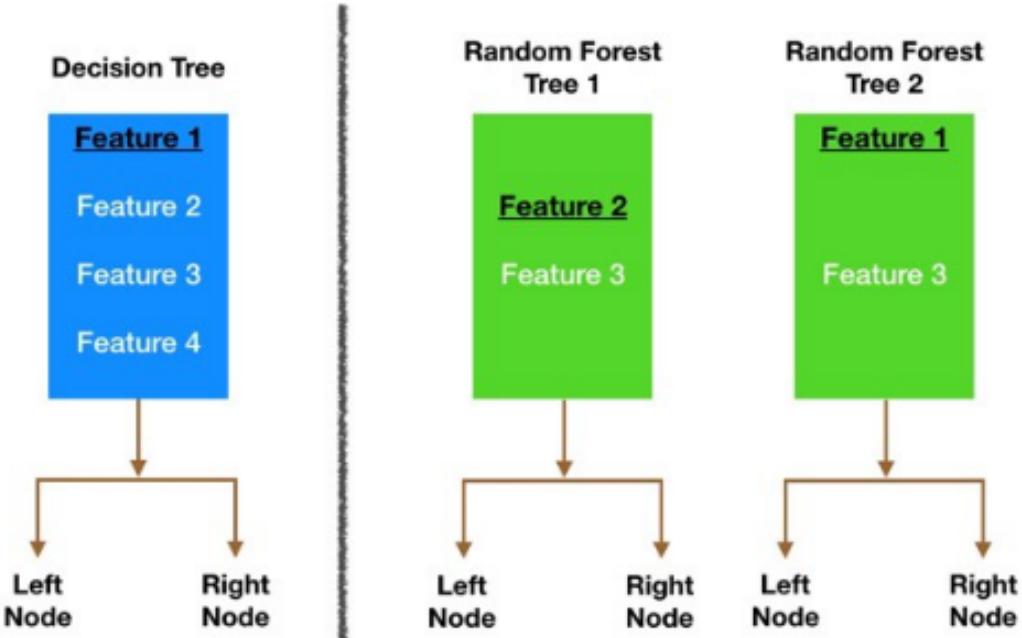
Pourquoi avoir choisi cet algorithme ? Parce que le modèle 'TreeClassifier' a tendance à sur apprendre et que pour pallier à ce problème le modèle Random Forest Classifier a l'avantage de créer plusieurs arbres avec pour moyen de séparation des échantillons aléatoires, et il prend ensuite comme résultat, 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.

1.8.5 CrÃ©ation 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. Et nous ajoutons l'API sur Rapide API ainsi que sa documentation. 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 relancer l'upload de 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.

1.8.6 CrÃ©ation d'une interface Web

Une interface web à était créée pour montrer que l'API est utilisable partout. Ce site web à été créé en PHP/HTML/CSS/JS avec comme framework Bootstrap 4 pour le côté responsive. Ce site web est donc conçu sur un modèle MVC "from scratch" et est facilement utilisable sur tout téléphone.

<https://optapi2.000webhostapp.com/MDB-Free/index.php>
Section 2.6

1.8.7 CrÃ©ation d'une application Mobile

Une application Mobile Ã Ã lait crÃ e dans le mÃ¢me 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).
Section ??

Cet application a Ã lait dÃ velopper 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 trÃ s suivit et possÃ de un trÃ s grand nombre de widget utilisables. Ce langage est un langage objet, il ma donc Ã lait assez facile de l'apprendre, sachant dÃ jÃ  coder en Java. PremiÃ rement, l'application comprenait seulement l'affichage 'live' des temps d'attentes.

[← Live](#)**Agence ENTREPRISES**

Attente moyenne Attente max
⌚ 00:00:00 ⌚ 00:00:00



(9 RUE DU GÉNÉRAL GALLIENI)

Centre de Traitement Postal

Attente moyenne Attente max
⌚ 00:00:00 ⌚ 00:00:00



(15 RUE FERNAND FOREST)

Agence de NOUMEA RIVIERE SALEE

Attente moyenne Attente max
⌚ 00:02:34 ⌚ 00:04:57



(53 AVENUE BONAPARTE)

Agence de NOUMEA VALLEE DES COLONS

Attente moyenne Attente max
⌚ 00:06:01 ⌚ 00:07:33



(54 RUE AUGUSTE BENEBIG)

Agence de NOUMEA BELLE VIE

Attente moyenne Attente max
⌚ 00:07:09 ⌚ 00:11:00



(224 RUE JACQUES IEKAWE)

Agence de NOUMEA SUD

Attente moyenne Attente max
⌚ 00:08:08 ⌚ 00:08:08



(19 RUE MARCELLIN LACABANNE)



[← Live](#)**Agence de NOUMEA SUD****[2]**

Attente moyenne Attente max
⌚ 00:08:08 ⌚ 00:08:08



(19 RUE MARCELLIN LACABANNE)

Agence de NOUMEA VALLEE DU TIR**[2]**

Attente moyenne Attente max
⌚ 00:08:21 ⌚ 00:24:32



(3 RUE BERTHELOT)

Agence de NOUMEA DUCOS CENTRE**[3]**

Attente moyenne Attente max
⌚ 00:10:59 ⌚ 00:23:29



(30 Route de la BAIE DES DAMES - RUE JOSE CASAROLI)

Agence de NOUMEA SUD N'GEA**[3]**

Attente moyenne Attente max
⌚ 00:11:49 ⌚ 00:37:10



(28 RUE HENRI GASPARD)

Agence de NOUMEA DUCOS LOGICOOP**[3]**

Attente moyenne Attente max
⌚ 00:13:44 ⌚ 00:14:25



(139 Route de la BAIE DES DAMES - RUE JOSE CASAROLI)

Agence de NOUMEA AGENCE PRINCIPALE**[3]**

Attente moyenne Attente max
⌚ 00:17:54 ⌚ 00:36:05



(7 RUE EUGÈNE PORCHERON)



Sur le coté droit 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 vidéo 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 'prÃlvision', reprenant le mÃlme principe que celui du site Web. Nous avons 3 'pageView' (3 pages avec une gestion de switch sur les cotÃ's de lÃcran pour passer d'une page Ã l'autre) avec sur chacune un formulaire dÃlidiÃl a la prÃldiction voulue (Endpoint). Le reste des Screenshots pourra Ãltre retrouver en Section [2.7](#).

7:58 | 0,1 Ko/s ☼ ☼



← Prédictions Globales —

Horaire : Choissir une horaire ▼

Jour : Choissir un jour ▼

Agence : Choissir une Agence ▼

Valider



Meilleures horaires



Global



Meilleures agences

2 ANNEXES

2.1 Apprentissage avec OpenClassRooms

Section 1.2

2.1.1 Technologies utilisÃ©s



- Anaconda3



- JupiterNoteBook



- Python3.7



- GitLab

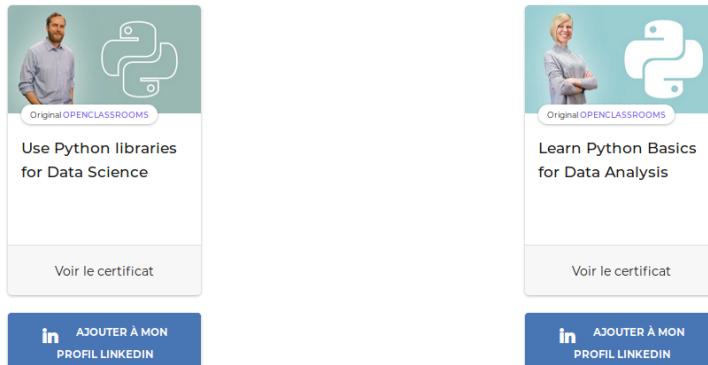


- GitKraken

2.1.2 Fonctionnement d'OpenClassRooms

- Le cours est structuré et il est volontif, 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

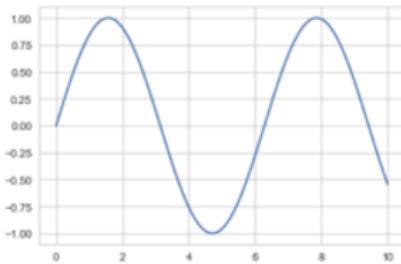


De plus

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));
```

python



Sin curve

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 validées 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 élèves. C'est un très bon exercice pour apprendre et pour voir différents moyens d'arriver à un même but.

2.1.3 Sujets abordés

Anaconda et Python Le cours [Learn Python Basics for Data Analysis](#) a servi à se remettre en 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'autre application). Ce cours nous initie donc au Python avant de toucher à 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 :

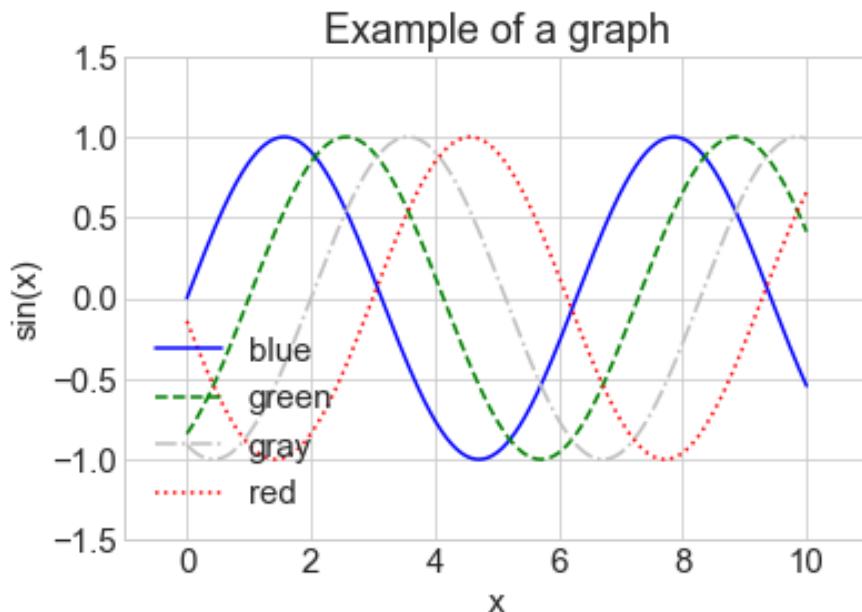
- Numpy
- Matplotlib
- Seaborn
- Pandas

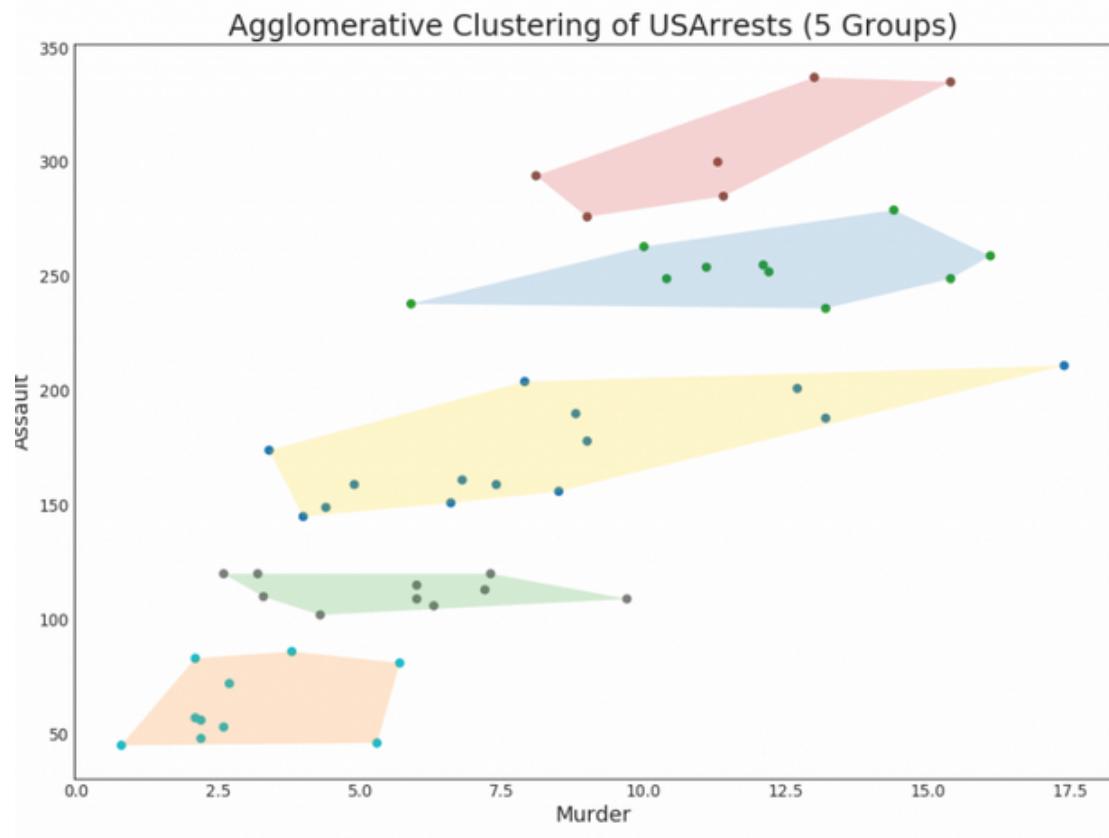
Numpy est une librairie très utilisée en python, elle est la base de toutes les autres citées ci-dessus. Elle permet de créer des tableaux aux dimensions voulues, d'accéder aux cases voulues mais aussi à appliquer toutes sortes de calculs sur les données de ces derniers.

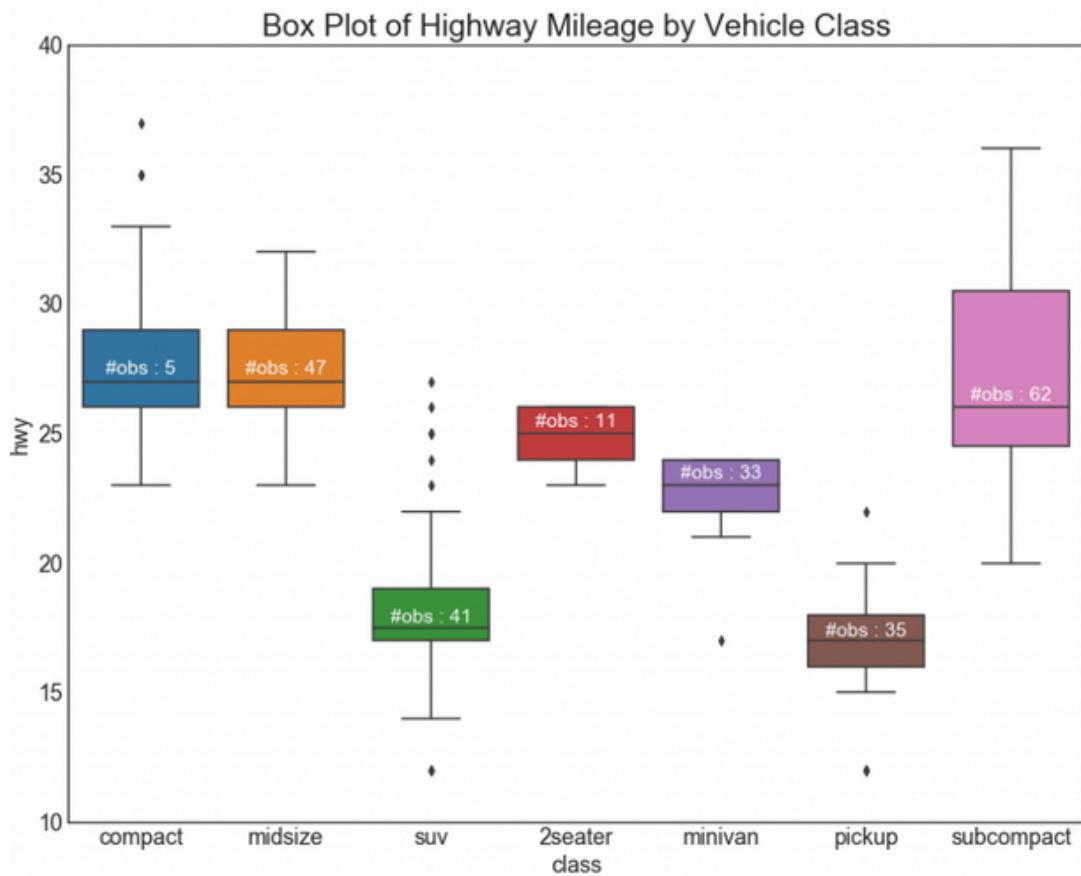
To apply functions on NumPy arrays :

```
python
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)))
```

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.







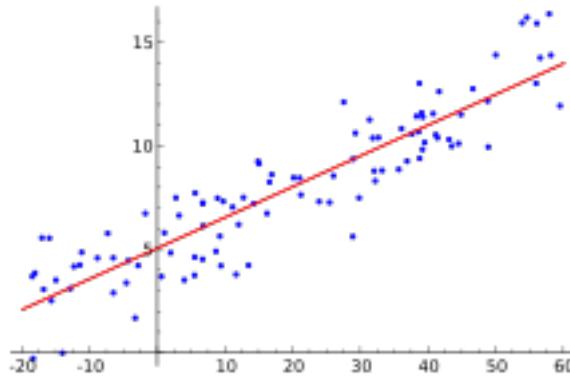
Il existe de nombreux types de graphiques possibles, ce site rÃ©alise les 50 "meilleurs" : [Top50 Matplotlib visualizations](#)

Plus de graphique avec Plotly et HoloViews [Plotly](#) et [HoloView](#) sont d'autres librairies python servants Ã visualiser des donnÃ©es, elles viennent compenser Matplotlib, notamment grÃ¢ce Ã la possibilitÃ© de crÃ©er des graphiques et schÃ¢ma dynamique voir mÃªme interactif.

Regression linÃ©aire avec Scikit-Learn Qu'est-ce que la rÃ©gression linÃ©aire ?

La rÃ©gression linÃ©aire est utilisÃ©e en statistiques pour dÃ©terminer l'influence d'une ou plusieurs donnÃ©es variables sur une Ã©valuation qui se veut linÃ©aire, en prenant en compte des paramÃ©tres additionnels. Dans le cadre d'un modÃ©le linÃ©aire simple, on peut reprÃ©senter graphiquement la relation entre x et y Ã‡a traverse un nuage de points.

L'estimation du modÃ©le linÃ©aire permet de tracer la droite de rÃ©gression, d'Ã©quation $y = \beta_0 + \beta_1 x$. Le paramÃ©tre β_0 reprÃ©sente l'ordonnÃ©e Ã‡a l'origine et β_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'autre processus statistique (Machine Learning, etc).

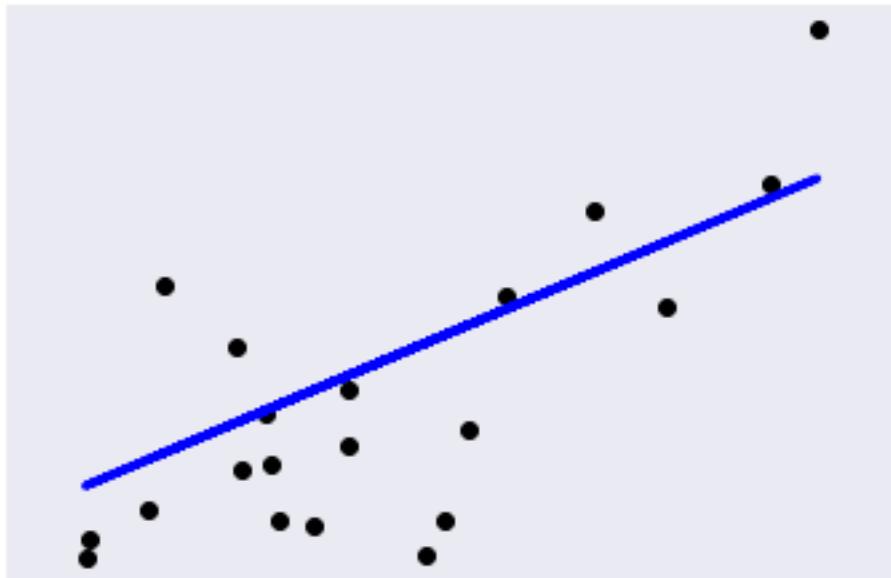
```
# Create linear regression object
regr = linear_model.LinearRegression()
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
```

```
[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))
```

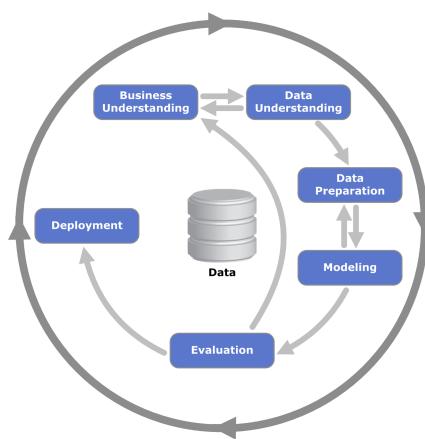
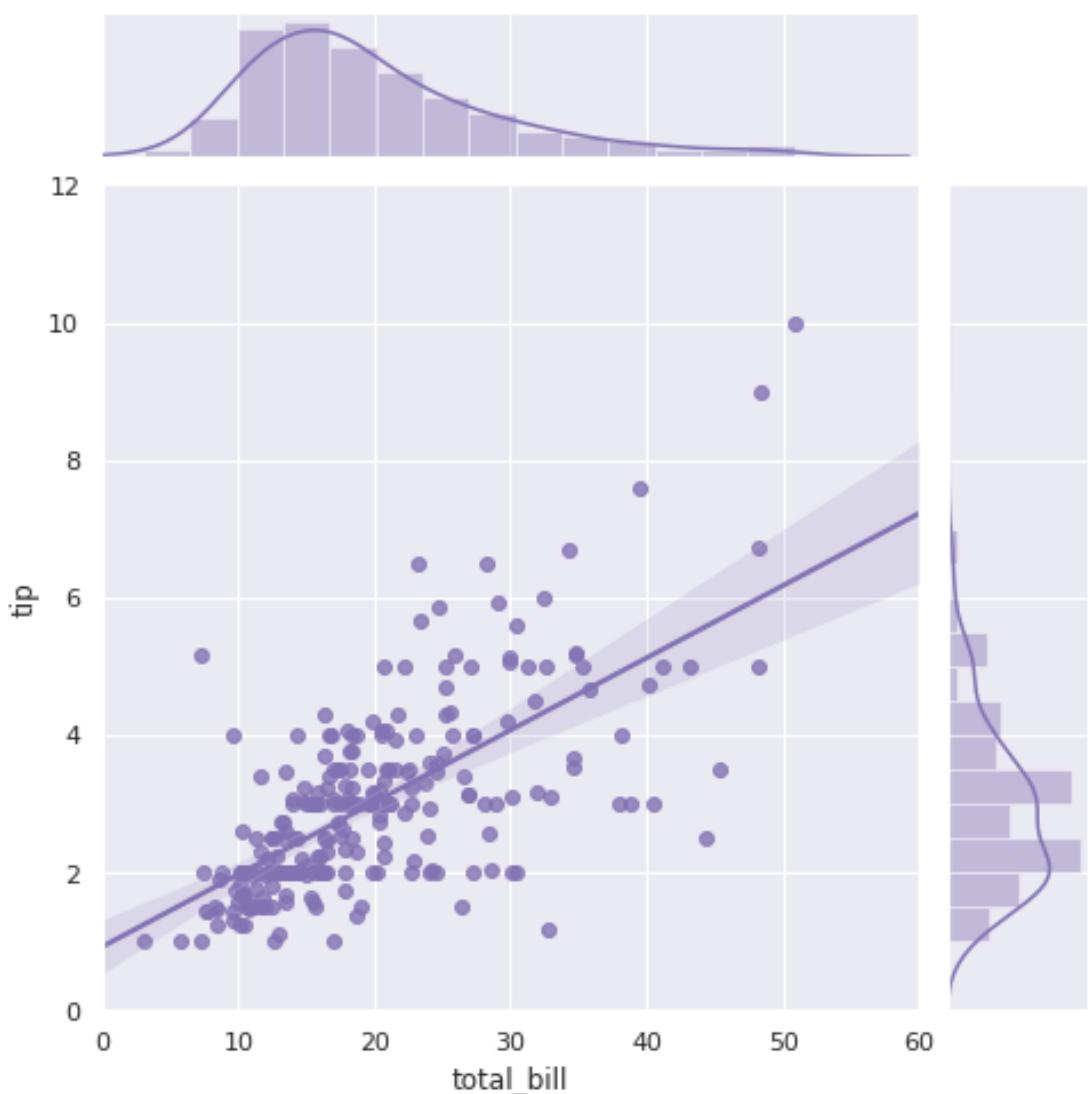
```
Coefficients:  
[938.23786125]  
Mean squared error: 2548.07  
Variance score: 0.47
```

```
[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 intégrée. Une méthode un peu plus simple pour une simple régression linéaire.

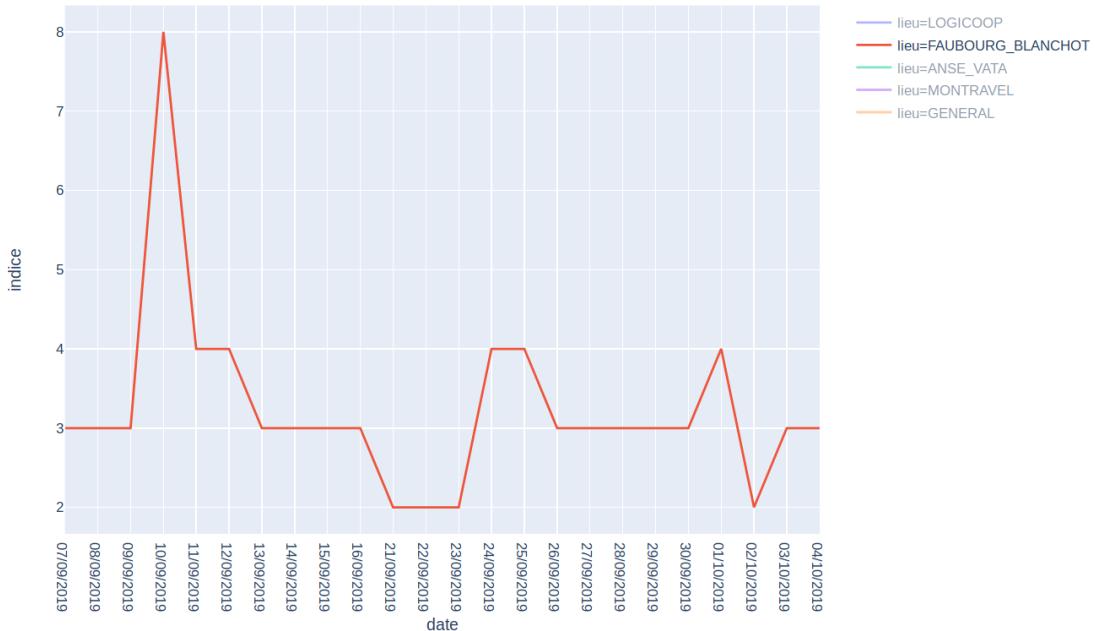
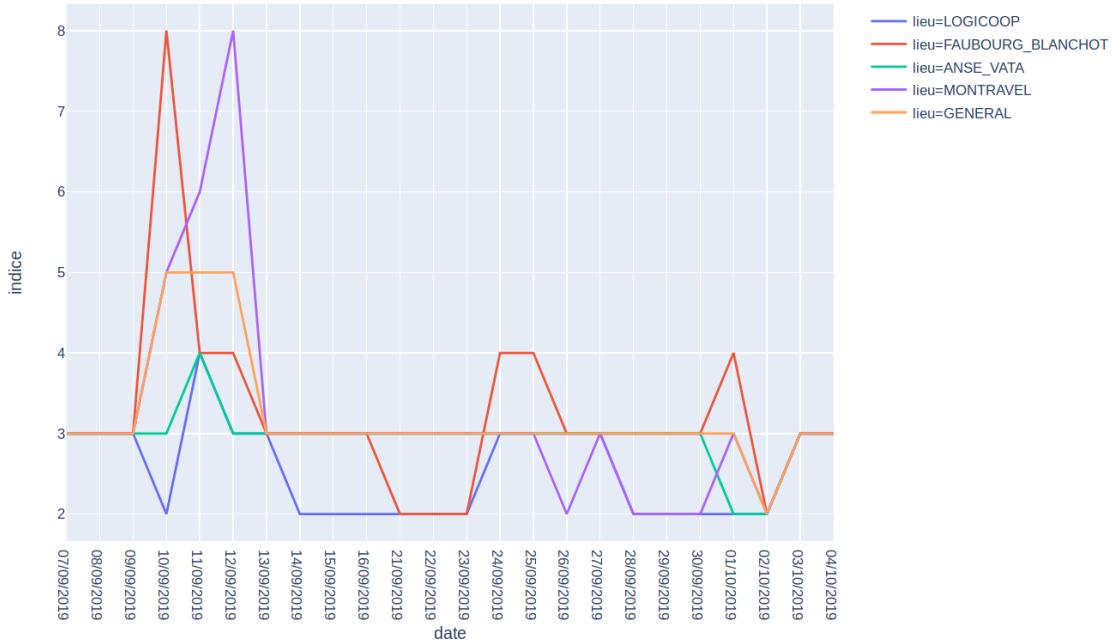
```
[7]: import seaborn as sns  
sns.set(style="darkgrid")  
tips = sns.load_dataset("tips")  
g = sns.jointplot("total_bill", "tip", data=tips, kind="reg",  
                  xlim=(0, 60), ylim=(0, 12), color="m", height=7)
```



Model de régression linéaire

2.2 Graphiques, Premières analyses

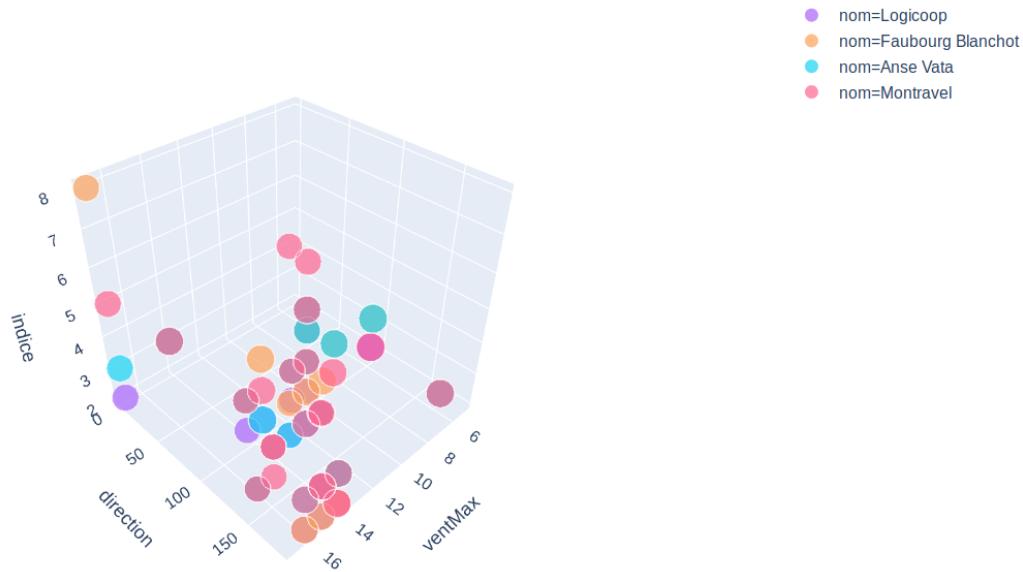
Section ??



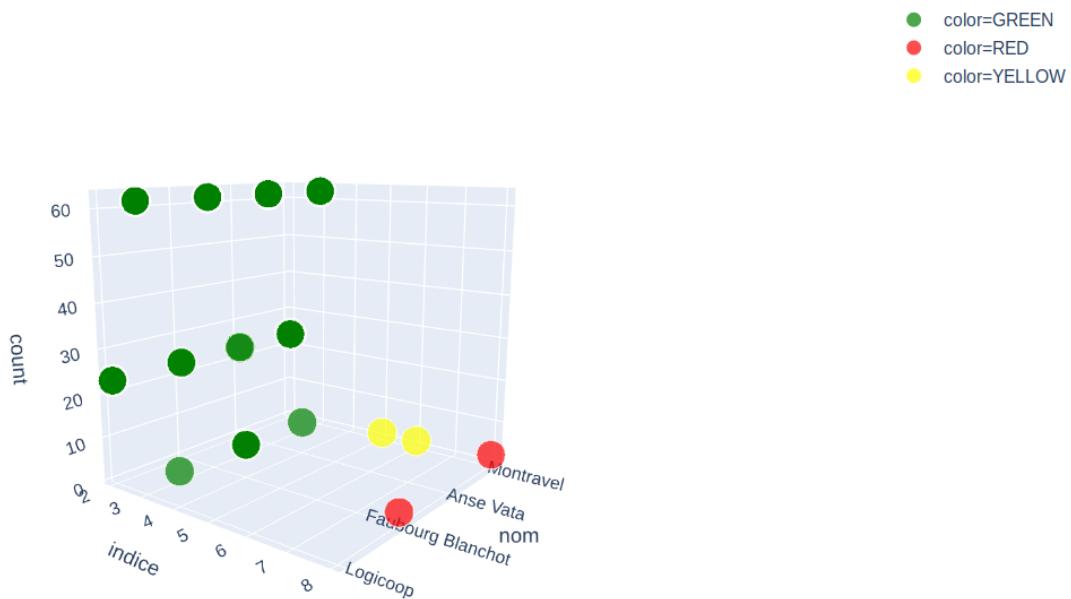
2.3 Graphique 3D avec Plotly

Section ??

Le graphique 3D, de la librairie Plotly, permet une meilleure compréhension de ces données et surtout, il permet d'observer le lien entre plus de données à la fois



Comme par exemple le nombre de fois où un certain indice a été calculé par quartier :



2.4 RÃ©cupÃ©ration de la donnÃ©e 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%A9o+de+lieu)&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']['hits'][0]['_source']['designation']],
             "type": [jsonN['hits']['hits'][0]['_source']['type']],
             "id": [jsonN['hits']['hits'][0]['_id']],
             "adresse": [jsonN['hits']['hits'][0]['_source']['pointAdresse']],
             "AVGWaitingTime": [jsonN['hits']['hits'][0]['_source']['borneEsirius']['meanWaitingTime']],
             "MAXWaitingTime": [jsonN['hits']['hits'][0]['_source']['borneEsirius']['maxWaitingTime']],
             "#UPD": [jsonN['hits']['hits'][0]['_source']['updatedEsiriusDate']]}

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

```

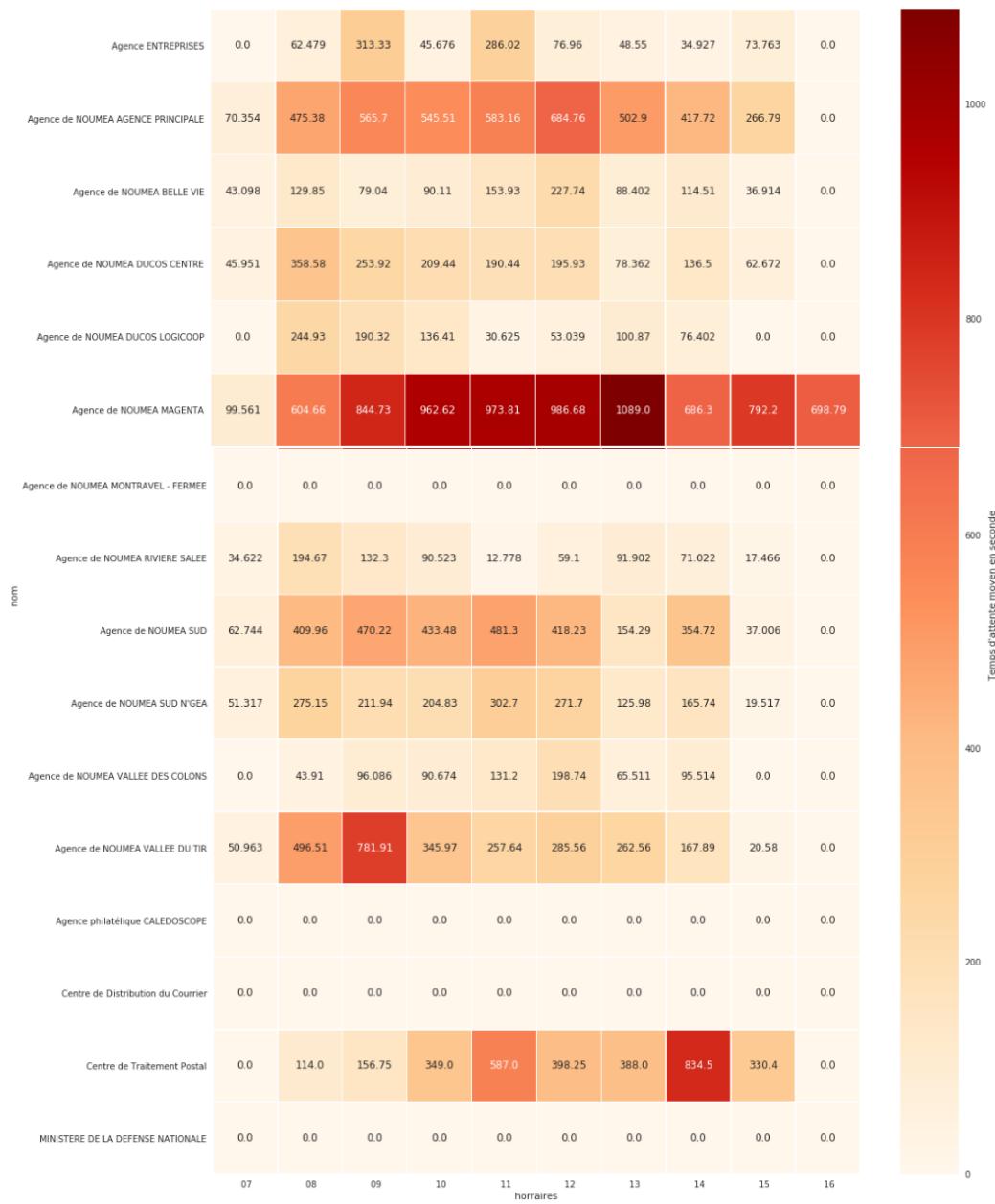
try:
    data = pd.read_csv('OPTWaitingTime.csv')
    print('OPTWaitingTime.csv is here !')
    #data[data=="] <- NA
    res = False
    for i in range (df['date'].size-1,0,-1):
        if data['date'].iloc[i] == strftime("%Y-%m-%d %H:%M:%S", time.localtime()):
            print("Point break")
            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)

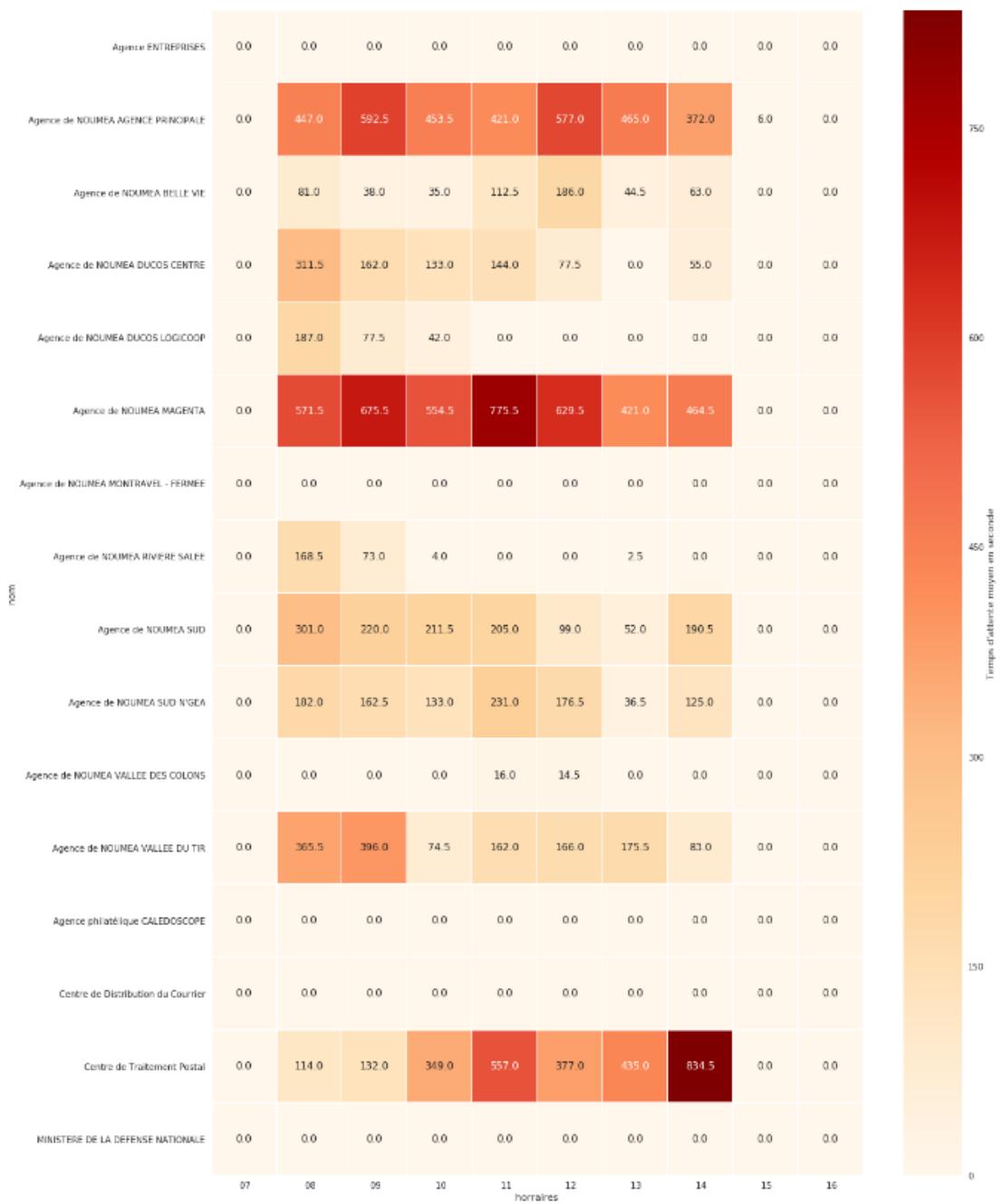
```

2.5 Heatmap OPT

Section ??



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



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.



2.6 QR Code Site Web

Section ??



2.7 ScreenShots Application Mobile

7:58 | 0,1 Ko/s 🔍 ⏴



Meilleures horaires



Jour : Choissir un jour



Agence : Choissir une Agence



Valider



Meilleures horaires



Global



Meilleures agences

49



7:58 | 0,0 Ko/s ⚡ ☀



Prédictions Globales



Horaire : 11h



Jour : Mercredi



Agence : NOUMEA VALLEE DES COLONS



Valider

Meilleures horaires



Global



Meilleures agences
50



← Prévisions

bon

Nom : Agence_Vallee_des_colons

Adresse : 54 RUE AUGUSTE BENEBIG

Jour : Mercredi

Horaire : 11.0 h

Temps d'attente moyen : bon

Avec une précision de : 0.842542503...





TÃ¢lÃ¢charger l'app