Projet en Machine Learning

NEW YORK TAXIS KAGGLE CHALLENGE

20 avril 2020

ISMGH Hatim - MAFTOUL Omar - OUHEB Massinissa
Université Paris Saclay

Table des matières

Table des figures

1 Introduction

Le but de ce rapport est de décrire les démarches effectuées pour la réalisation de ce projet, qui ce dernier consiste de construire un modèle qui prédit la durée totale des trajets en taxi à New York City en utilisant un ou plusieurs algorithmes d'apprentissage. Le jeu de données principal est celui publié par la NYC Taxi and Limousine Commission, qui comprend l'heure de ramassage, les coordonnées géographiques, le nombre de passagers et plusieurs autres variables.

Les prédictions obtenues seront mis à la fin dans un fichier *csv* et sera envoyé sous Kaggle qui nous donnera le score de nos prédictions.

La métrique d'évaluation est appelée la racine de l'erreur logarithmique quadratique moyenne. Le RMSLE (*Root Mean Squared Logarithmic Error*) est calculé comme suite :

$$e = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(log(p_i + 1) - log(a_i + 1))^2}$$

Où :

— e est la valeur RMSLE (score).

— n est le nombre total d'observations dans l'ensemble de données (public / privé).

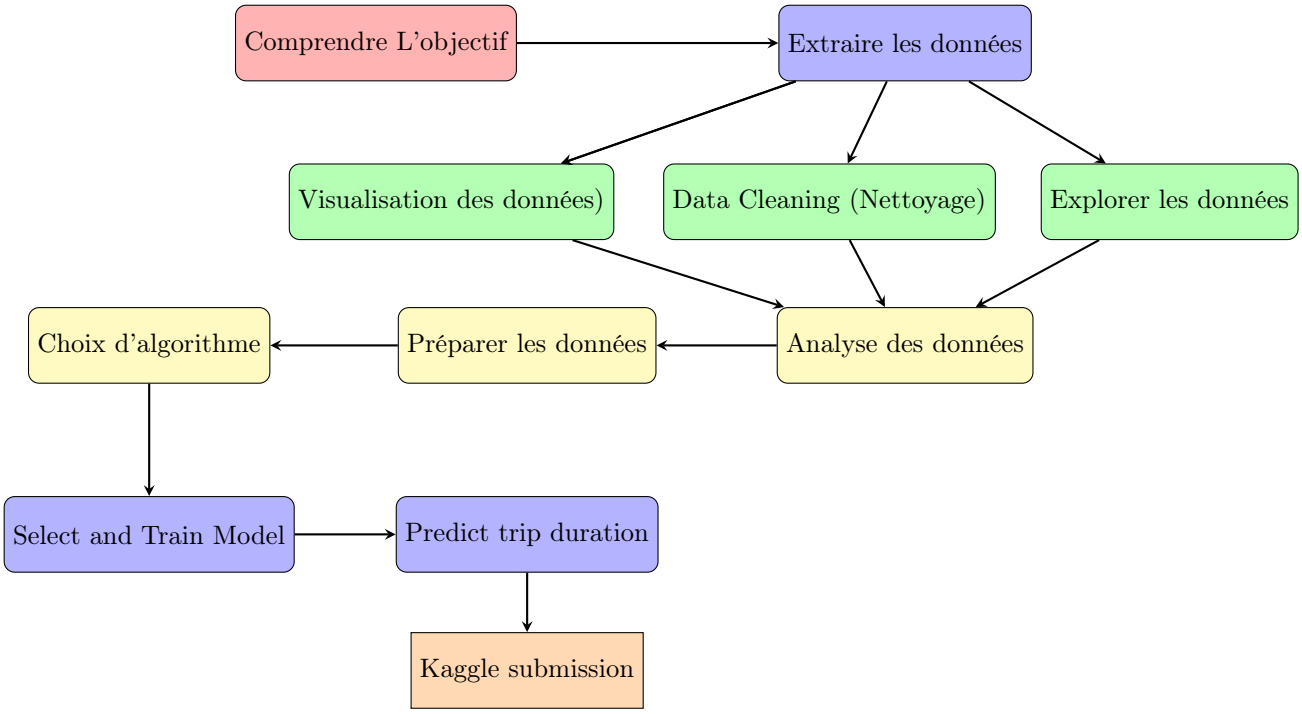
— p_i est votre prévision de la durée du voyage.

— a_i est la durée réelle du trajet pour i.

— $log(x)$ est le logarithme naturel de x.

2 Étapes à suivre tout au long du projet

Le schéma ci-dessous récapitule et décrit les étapes suivies tout au long de ce projet.



3 Pré-traitement des données

Dans cette section, on va s'intéresser à découvrir les données qui nous ont été fournies par Kaggle.
Selon cette dernière, les données des trajets ont été mise à la disposition du monde par ***NYC Taxi and Limousine Commission (TLC)***, les données sont divisées sur deux fichiers :
train.csv qui contient 1,458,644 de trajets et qui servira par la suite à entraîner nos algorithmes d'apprentissage, et *test.csv* qui contient 625,134 trajets en excluant l'attribut de **durée du trajet(trip duration)** qu'on doit évidemment prédire.
De plus, Kaggle nous informe que les données ont été nettoyées (aucune ligne de donnée ne contient des valeurs null ou vide), et ceci ne nous empêchera pas d'explorer ces données, et de s'assurer de ces cohérences et filtrer les données nuisibles si c'est nécessaire.

3.1 Exploration des données

Sous **Python**, nous allons commencer à explorer le fichier *train.csv* qui nous intéresse le plus. Nous essayerons d'acquérir une idée globale sur les attributs de la table des données (explorer leurs types, générer les statistiques liées à chaque attribut).

3.1.1 Dictionnaire de données

Dans cette section, nous allons décrire les attributs ainsi que leurs types en utilisant *Panda* (Bibliothèque connue pour l'exploitation et le traitement des données sous Python).
En utilisant la méthode *info()* de *Panda*, on obtient les types de données des différents attributs, ainsi, nous rajoutant de plus une colonne qui indique la description correspondante à chaque attribut dans le tableau ci-dessous :

Attribut	**Type**	**Description**
id	object (string)	l'identifiant associé a chaque trajet
vendor_ id	int64	le code de la machine utilisée pour enregistrer les données (1 : Creative Mobile Technologies, LLC ; 2 : VeriFone Inc.).
pickup_ datetime	object (string)	la date et l'heure exacte du début du trajet.
dropoff_ datetime	object (string)	la date et l'heure exacte de la fin du trajet.
passenger_ count	int64	le nombre de passagers pris par le taxi.
pickup_ longitude	float64	la longitude du point de départ du trajet.
pickup_ latitude	float64	la latitude du point de départ.
dropoff_ longitude	float64	la longitude du point d'arrivée.
dropoff_ latitude	float64	la latitude du point d'arrivée.
store_ and_ fwd_ flag	object (string)	cet attribut désigne si les données sont enregistrées dans la mémoire du véhicule avant d'être envoyées aux serveur.
trip_ duration	int64	la durée du trajet en secondes (l'attribut qu'on cherche à prédire).

3.1.2 Statistiques et visualisation des données

Jusqu'ici , nous avions jeté un coup d'oeil rapide sur l'ensemble des données mise à notre disposition, cependant, dans cette section, nous irons plus loin dans l'exploration des données, notamment l'affichage des statistiques liées aux attributs numérique ainsi que la visualisation de ces derniers dans des graphes. En revanche, en mentionnant les *attributs numérique*, certaines méta données ne sont pas de type numérique (*float, int*), tel que l'attribut ***pickup-dropoff_ datetime*** et ***store_ and_ fwd_ flag***, ceci pourrait causer des problèmes dans nos prédictions que nous ferons

ultérieurement, c'est ainsi que nous procéderons à convertir ces attributs, car la plus part des algorithmes d'apprentissage préfèrent de prendre des données numériques plutôt que des labels ou textes.
Commençant par convertir ***store_and_fwd_flag***, qui peut être considéré comme un attribut catégorique puisqu'on rencontre des valeurs répétitives, ainsi que pour chaque trajet, il prend l'une des deux valeurs : *Y (Oui)* et *N (Non)*.

```
train.store_and_fwd_flag.value_counts()
# N : 1,450,599 rows contains N
# Y : 8045 rows contains Y
```

On pourrait donc faire correspondre ces deux valeurs à des chiffres binaires, ***1*** correspondra à ***Y***, et ***0*** correspondra à ***N***, de plus, comme convenu, on convertira l'attribut ***pickup-dropoff_datetime*** en *TimeStamp* (qui nous aidera ultérieurement grâce aux fonctions intégrées à extraire facilement le jour, l'année, l'heure et le mois).

```
    import pandas as pd
    train['date']  = pd.to_datetime(train[['pickup_datetime','dropoff_datetime']])
    dummy = pd.get_dummies(train_data['store_and_fwd_flag'])
    train_taxi_2 = pd.concat((train_data, dummy), axis=1)
    train_taxi_2 = train_taxi_2.drop(['store_and_fwd_flag'], axis=1)
    train_taxi_2 = train_taxi_2.drop(['N'], axis=1)
    train_taxi_2 = train_taxi_2.rename(columns={"Y": "store_and_fwd_flag"})
    train = train_taxi_2
    train.store_and_fwd_flag.value_counts()
    # 0 : 1,450,599 rows contains 0
    # 1 : 8045 rows contains 1
```

A posteriori, on pourra afficher un récapitulatif des attributs numérique, ce dernier comporte des données statistiques tel que la moyenne, le min/max, et le std (qui mesure la dispersion des valeurs d'un même attribut).

```
IN :
        train.describe()
OUT :
```

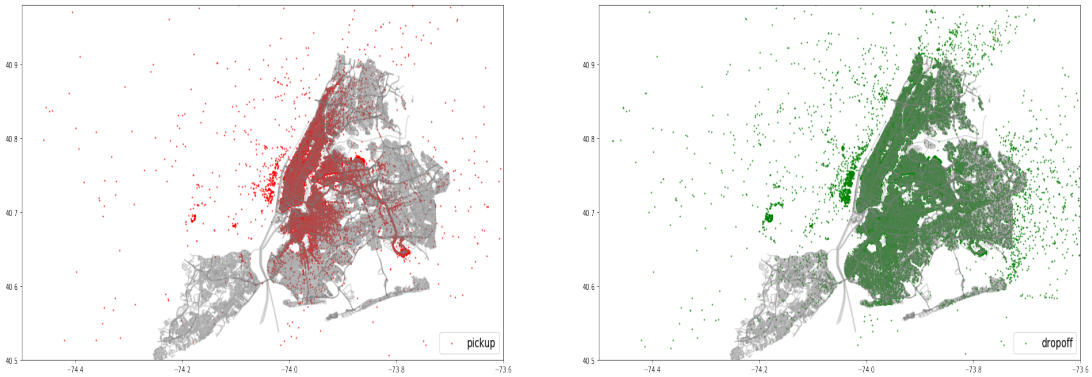
	vendor_id	passenger_count	pickup_longitude	pickup_latitude
count	1.458644e+06	1.458644e+06	1.458644e+06	1.458644e+06
mean	1.534950e+00	1.664530e+00	-7.397349e+01	4.075092e+01
std	4.987772e-01	1.314242e+00	7.090186e-02	3.288119e-02
min	1.000000e+00	0.000000e+00	-1.219333e+02	3.435970e+01
25%	1.000000e+00	1.000000e+00	-7.399187e+01	4.073735e+01
50%	2.000000e+00	1.000000e+00	-7.398174e+01	4.075410e+01
75%	2.000000e+00	2.000000e+00	-7.396733e+01	4.076836e+01
max	2.000000e+00	9.000000e+00	-6.133553e+01	5.188108e+01

FIGURE 1 – *describe() of train Dataframe part 1*

	dropoff_longitude	dropoff_latitude	trip_duration	store_and_fwd_flag
count	1.458644e+06	1.458644e+06	1.458644e+06	1.458644e+06
mean	-7.397342e+01	4.075180e+01	9.594923e+02	5.515396e-03
std	7.064327e-02	3.589056e-02	5.237432e+03	7.406066e-02
min	-1.219333e+02	3.218114e+01	1.000000e+00	0.000000e+00
25%	-7.399133e+01	4.073588e+01	3.970000e+02	0.000000e+00
50%	-7.397975e+01	4.075452e+01	6.620000e+02	0.000000e+00
75%	-7.396301e+01	4.076981e+01	1.075000e+03	0.000000e+00
max	-6.133553e+01	4.392103e+01	3.526282e+06	1.000000e+00

FIGURE 2 – *describe() of train Dataframe part 2*

`count, mean, min, max` indiquent respectivement la valeur du nombre total, la valeur de la moyenne, la valeur du minimum et la valeur du maximum, de plus, ces colonnes sont intuitivement compréhensible. Les lignes **`25%, 50% 75%`** représentent respectivement le quart, la médiane, et le trois quart des données, ces derniers sont des pourcentages sous lesquels les données sont découpées, par exemple, 25% des durées de trajets dans l'ensemble de notre *Dataframe* sont en dessous ou égale à 397 secondes.
Ce résumé en chiffres est assez important, car cela servira ultérieurement à procéder au traitement de notre *DataFrame*, notamment à se débarrasser des trajets qui peuvent être nuisibles pour les autres données (*Noisy Data*).
Comme on l'a pu constaté dans la section précédente, on a remarqué la présence de méta-données géographique dans le dictionnaire de données comme **`dropoff_longitude-latitude`** et **`pickup _longitude-latitude`**, ces deux derniers attributs expriment la localisation où le client a pris le taxi, et la localisation où le taxi a déposé le client. Donc à l'aide de **`Geopandas`**, un package sous Python qui facilite l'affichage des données qui correspondent le plus au monde réel (Données démographiques, économiques ...), nous allons attacher les deux attributs en question à des localisations spécifique sur la carte de la ville de New York (que nous avions pu récupérer sous l'extension *`.shp`* à partir du site Web gouvernementale de New York *nyc.gov*).



(a) Pickup Locations (b) Dropoff Locations

FIGURE 3 – Dropoff and Pickup Locations in New York City

On peut maintenant avoir une vision meilleur en explorant les cartes ci-dessus. Nous apercevons donc en visualisant ces données géographiques plusieurs points dispersés partout dans la ville de New York, en revanche, certaines points se trouvent en dehors des limites de la ville.

4 Traitement et analyse des données

A ce moment, après la visualisation et l'exploration des données, on pourra commencer à les traiter et analyser, nous retournerons ainsi vers les Figures 1 et 2 pour les mêmes raisons.

4.1 Interception des données nuisibles

Dans la section qui a été dédiée à la visualisation et les statistiques, nous avions introduit la méthode `describe()`, donc nous allons de nouveau l'utiliser pour afficher le même récapitulatif en gardant seulement les attributs qui reflètent plus d'importance et qui peuvent influencer les prédictions, on parle de `trip_duration, passenger_count, pickup_location et dropoff_location`. Le but de cette procédure est d'intercepter les chiffres suspicieux à partir du `min, max, mean, count` , nous entendons par les chiffres suspicieux les données qui peuvent sortir du contexte de l'ensemble des données, tel qu'une durée de trajet qui dure des heures, ou un trajet où le taxi a pris un nombre inhabituel de personnes.

	passenger_count	trip_duration
count	1.458644e+06	1.458644e+06
mean	1.664530e+00	9.594923e+02
std	1.314242e+00	5.237432e+03
min	0.000000e+00	1.000000e+00
25%	1.000000e+00	3.970000e+02
50%	1.000000e+00	6.620000e+02
75%	2.000000e+00	1.075000e+03
max	9.000000e+00	3.526282e+06

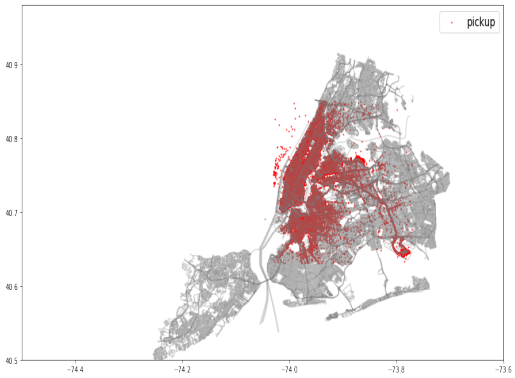
FIGURE 4 – Interception des données nuisibles sur trip_duration et passenger_count

Pour l'attribut **`passenger_count`**, les colonnes `min et max` indiquent qu'un taxi pourrait contenir entre 0 et 9 passagers pour un trajet donné. De l'autre côté, pour l'attribut **`trip_duration`**, on remarque qu'un trajet pourrait durer entre 1 seconde et
$\frac{3.526282 \times 10^6}{3600 \times 24} = 41$ jours.
Ce que l'on peut en déduire de ces remarques, c'est qu'un trajet pour un taxi en ville ne pourrait pas durer 1 seconde et de même ne pourrait pas durer plusieurs jours, de plus, un trajet ne pourrait être valide que si au moins un passager est à bord du taxi. Sachant que 75% des trajets comptent avoir au plus 2 passagers par taxi, les trajets qui comportent un nombre assez élevé tel que le **`max`** qui est égale à 9 semblent moins intéressants pour notre modèle d'apprentissage.

Donc afin de garantir des prédictions qui seront proches de la réalité, il faudra qu'on se débarrasse de tout les trajets qui contiennent des données nuisibles, tel que les trajets qui durent des journées, ou entre autre, les trajets qui contiennent des points de départ/dépôt en dehors de la ville de New

York. Cette partie du traitement est primordiale et empêchera nos algorithmes d'apprentissage à s'entraîner sur des modèles qui contiennent des données nuisibles, car si le cas se présente, on risque de voir un **Overfitting / Overtraining (sur apprentissage)**, où notre algorithme essayera d'apprendre plus que nécessaire en incluant les détails de données et les données nuisibles, ces derniers auront un impacte négatif de sorte que l'algorithme n'aura pas la possibilité de bien généraliser.
A l'aide de la fonction **quantile()** fournie par `Panda`, on va garder seulement les valeurs des durées de trajets qui représentent 99% de l'ensemble des données de départ, donc on limitera l'attribut `trip_duration` à une valeur x en seconde, où tout les trajets auront une durée inférieure ou égale à x, **quantile(.99)** a le même principe des pourcentages `25%, 50% et 75%` de la fonction **describe()** qu'on a vu auparavant. Ceci permettra de se débarrasser des *Noisy Data* vis à vis de l'attribut `trip_duration`. De même, on va exclure les trajets où il y' a 0 passagers. Assurément, les **pickup-dropoff_location** se trouvant hors limite de la ville vont être éliminés du *DataFrame*.

```
train = train[train.trip_duration < train.trip_duration.quantile(0.99)]
train = train[(train['passenger_count'] > 0)]
xlim = [-74.03, -73.77]
ylim = [40.63, 40.85]
train = train[(train.pickup_longitude> xlim[0]) & (train.pickup_longitude < xlim[1])]
train = train[(train.dropoff_longitude> xlim[0]) & (train.dropoff_longitude < xlim[1])]
train = train[(train.pickup_latitude> ylim[0]) & (train.pickup_latitude < ylim[1])]
train = train[(train.dropoff_latitude> ylim[0]) & (train.dropoff_latitude < ylim[1])]
```



(a) Pickup Locations excluding locations outside the city

	passenger_count	trip_duration
count	1.444051e+06	1.444051e+06
mean	1.663301e+00	8.074555e+02
std	1.313507e+00	5.754200e+02
min	1.000000e+00	1.000000e+00
25%	1.000000e+00	3.950000e+02
50%	1.000000e+00	6.560000e+02
75%	2.000000e+00	1.058000e+03
max	9.000000e+00	3.439000e+03

(b) describe()

FIGURE 5 – Données après traitement

Après la suppression de toutes les localisations qui sont approximativement non inclus dans la ville de New York, nous avions limiter le nombre de trajets à `1,425,132` au lieu de `1,444,051` , donc on a supprimé `18,919` trajets ce qui représente `1.31%` des données du début.

4.2 Relations entre attributs et corrélation

Après le traitement de données, il est préférable d'explorer la corrélation qui permet de mesurer les dépendances entre les attributs, et plus spécifiquement, la corrélation entre les attributs et la variable objective `trip_duration`, cette étape est appréciable pour la construction d'un bon modèle d'apprentissage, ainsi aidera au choix des algorithmes qui correspondrons le mieux au modèle. Par la suite, à l'aide de la fonction `coor()`, on va afficher la matrice de corrélation, chaque ligne de cette matrice correspondra à un attribut et son coefficient de corrélation qui est définit entre -1 et 1, ce coefficient encadrera la manière et le pourcentage sous lesquels l'attribut est corrélé avec un autre, dans notre cas, on affichera la corrélation entre tout les attributs et l'attribut d'objectif qu'on veut prédire par la suite (`trip_duration`).

```
IN :
    matrix_correlation = train.corr()
    matrix_correlation["trip_duration"].sort_values(ascending=False)
OUT :
```

trip_duration	1.000000
pickup_longitude	0.306657
dropoff_longitude	0.230599
store_and_fwd_flag	0.021760
passenger_count	0.014956
vendor_id	0.007264
dropoff_latitude	-0.177400
pickup_latitude	-0.189843

FIGURE 6 – Correlation coeffcients between trip_duration and other variables

La matrice de corrélation de la figure à droite dévoile plusieurs indications. Premièrement, on distingue deux types de corrélation : **positive** et **négative**, par exemple, la corrélation entre l'attribut `trip_duration` et lui même est une corrélation fortement positive, son coefficient est égale à 1 (ce qui est évident). Prenant (`pickup_latitude`), cette dernière a une corrélation négative avec `trip_duration`, ce qui signifie que la valeur de `trip_duration` diminue quand la valeur de `pickup_latitude` diminue aussi. Autre attribut tel que `vendor_id` est faiblement corrélé avec `trip_duration` car son coefficient est très proche de 0.
Remarque : Le coefficient de corrélation mesure uniquement les corrélations linéraires, c'est à dire, si x augmente/diminue, y augmentera/ diminuera aussi. Apparemment, et à titre d'exemple, les attributs `vendor_id et trip_duration` montrent la présence d'une relation non-linéaire entre les deux, puisque l'une est faiblement dépendante de l'autre.
En récapitulatif, plus le coefficient est proche de 1 ou -1, plus les attributs ont une forte liaison linéaire. Plus le coefficient est proche de 0, plus il y' a une faible relation linéaire.

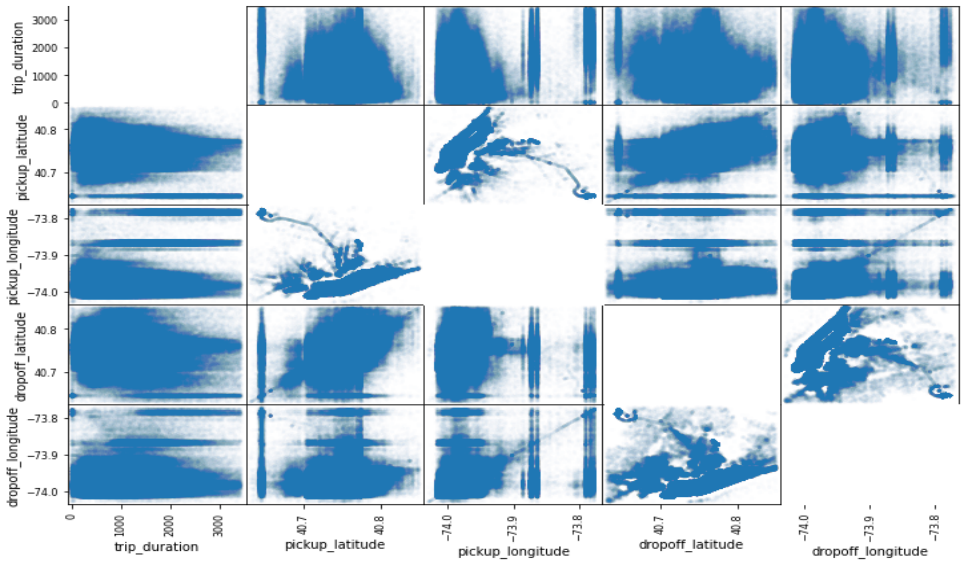


FIGURE 7 – Corrélation et linéarité entre attributs

5 Combinaisons entre attributs et la création des nouvelles variables

Avant de mettre en oeuvre notre modèle, il est considérablement important d'expérimenter la création de nouvelles variables à partir des attributs de base, ceci peut aider à améliorer et enrichir le modèle d'apprentissage avec des données plus intéressantes que celles du départ. Donc dans cette section, nous allons essayer des combinaisons entre attributs afin de donner naissance à des nouveaux, puis nous allons étudier leurs importances qui décideront par la suite si on peut les garder dans notre modèle ou les abandonner.

5.1 Qu'est ce qui peut influencer un trajet ?

`La Distance` . En effet, on dispose d'éléments géographiques qui peuvent nous permettre de savoir la distance d'un trajet. Le couple **`Longitude , Latitude`** peuvent former un point donné dans la carte de la ville, donc **`pickup`**`(Longitude , Latitude)` et **`dropoff`**`(Longitude , Latitude)` seront respectivement le point de départ et d'arrivée, et effectivement, entre les deux points se formera une distance. Puisqu'on doit calculer la distance [1] entre deux points dans un ***ellipsoïde aplati*** (une forme sphérique qui distingue approximativement la surface de la planète Terre [2]) et non pas dans une simple surface (on ne procédera pas donc à calculer une distance euclidienne), on utilisera la formule de **haversine** [3] qui calcule la plus petite distance entre deux points dans une sphère. On aura pas à programmer par nous même les équations de calcule, la

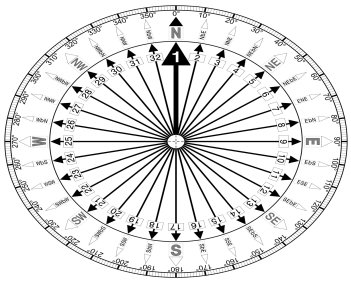
1. $a = hav(\Delta latitude) + cos(latitide_1) \times cos(latitude_2) \times hav(\Delta longitude)$
   $distance = 2 \times 6371 \times arctan(\sqrt{a}, \sqrt{1-a})$
2. Le rayon de la terre est 6371 Km.
3. $hav(\theta) = \sin^2(\frac{\theta}{2})$

bibliothèque **Geod de pyproj** le fera à notre place. On rajoutera donc une nouvelle colonne qu'on nommera **approx_distance** qui combinera les quatre attributs **pickup_long-lat et dropoff_long-lat**. Notons que la **distance** se calculera en **mètre**, ainsi que chaque nouveau attribut ajouté s'ajoutera aussi au données de **test**.

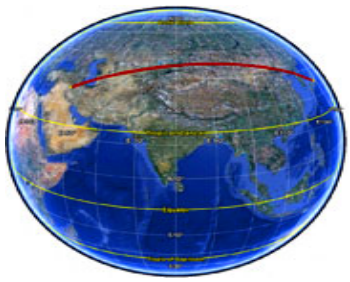
```
import pandas as pd
import numpy as np
from pyproj import Geod

wgs84_geod = Geod(ellps='WGS84')

def Distance(lat1,lon1,lat2,lon2):
    az12,az21,dist = wgs84_geod.inv(lon1,lat1,lon2,lat2)
    return dist

train['approx_distance'] = Distance(train['pickup_latitude'].tolist(),
    train['pickup_longitude'].tolist(),
    train['dropoff_latitude'].tolist(),
    train['dropoff_longitude'].tolist())
```

5.2 Le nouvel attribut 'Direction'

En parlant toujours du trafic qui peut jouer un rôle à l'influence de la durée du trajet, on a eu l'idée de rajouter la **direction** [4] du trajet. Évidemment, on croit que la densité des trafics changent selon les zones et directions, et donc les durées sont impactées aussi. Comme pour la `distance`, la `direction` sera la combinaison des quatre variables géographique `pickup et dropoff`. Cet attribut sera basé sur le calcule d'une équation [5] qui déterminera la valeur d'un angle [6] sur un cercle (un Relèvement). La figure ci dessous montrera la valeur que chaque trajet va prendre pour la nouvelle variable `direction`.



(a) Angles de directions



(b) Exemple d'une direction, la terre correspondra au cercle à gauche

FIGURE 8 – Direction dans un plan sphérique

```
def bearing_array(lat1, lng1, lat2, lng2):
    lng_delta_rad = np.radians(lng2 - lng1)
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
```

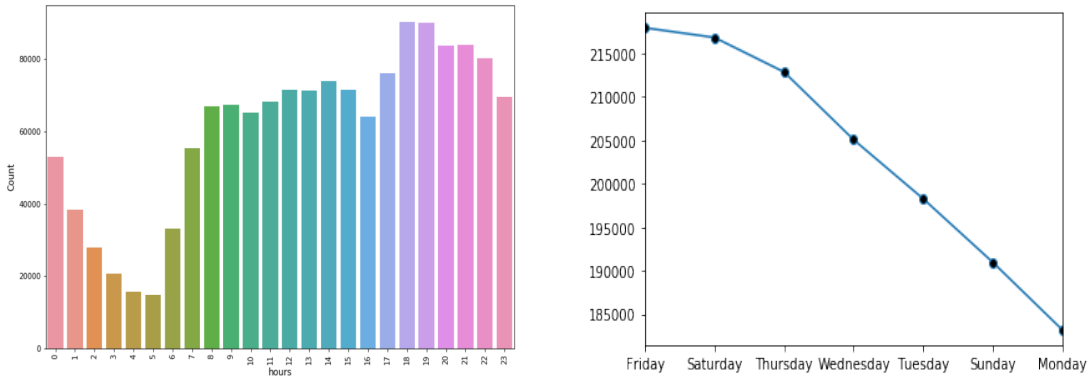
4. Nord Sud Est West
5. $\theta = arctan^2(sin(\Delta longitude) \times cos(latitude_2), cos(latitude_1) \times sin(latitude_2) - sin(latitude_1) \times cos(latitude_2) \times cos(\Delta longitude)$
6. L'angle sera compris entre -180° et 180°

```
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) * np.cos(lng_delta_rad)
    return np.degrees(np.arctan2(y, x))

train['direction'] = bearing_array(train['pickup_latitude'].values,
                                   train['pickup_longitude'].values,
                                   train['dropoff_latitude'].values,
                                   train['dropoff_longitude'].values)
```

5.3 L'ajout des heures de la journée et les jours

Le trafic routier peut à son tour impacter la durée de trajet non seulement pour les taxis mais pour tout type de transport, certes, ce sont les parties de la journées qui peuvent impacter le trafic, notamment les heures de pointes et les heures où la majorité des personnes partent au travail ou écoles. A titre d'exemple, en peut intuitivement prédire qu'il est fortement probable que le trafic soit plus dense entre 17h00 et 19h00 que entre 1h00 et 2h00, et donc le trafic peut influencer la durée. C'est le même principe pour les jours de la semaine, on peut aussi imaginer que le trafic et la durée dépendront des jours (week-ends, vacances [7]).

```
train['hours'] = train['pickup_datetime'].map(lambda x: int(x.hours))
train['day'] = train['pickup_datetime'].map(lambda x: int(x.day))
```



(a) Nombres de trajet en fonction de l'heure

(b) Nombres de trajet en fonction des jours

FIGURE 9 – Nombre de trajets en fonction des jours et heures de la journée (format 24 H)

7. On a essayé de rajouté une colonne ***Holiday***, qui indique si les jours des trajets font parties des jours de vacances, cependant, aucun jour n'est inclus dans le calendrier des vacances américaine, par conséquent, l'idée du rajout de cet attribut est mise à l'écart

5.4 Données géographiques et le 'Principal Component Analysis (PCA)'

Nous avons constaté que `pickup_longitude , pickup_latitude, dropoff_latitude, dropoff_longitude` ont été les variables protagonistes d'où on a été inspiré pour créer des nouvelles variables. Ceci, ne nous empêchera pas de les utiliser aussi dans notre modèle d'apprentissage. En parlant du modèle, parfois les algorithmes prennent un temps considérable pour l'entraîner, et surtout si les dimensions des variables d'entrée sont grandes. C'est ainsi qu'on va procéder à l'utilisation du `PCA` pour les variables géographiques, afin de diminuer leurs dimensions et par conséquent leurs valeurs. Ce choix raisonnable parmi d'autres alternatives aidera à faire fonctionner nos algorithmes d'apprentissage plus vite et d'une manière plus performante.
On peut donc remarquer que le point de départ est deux dimensionnel, car on possède deux variables qui le représentent (`pickup_longitude , pickup_latitude`), et c'est le même principe pour le point d'arrivée `dropoff_latitude, dropoff_longitude`. Le but du `PCA` dans notre cas est de diminuer la dimension de 2 vers 1, et par conséquent, la création de deux nouvelles variables uni-dimensionnels : (`pickup_pca , dropoff_pca`). L'enchaînement de cette procédure est réalisable avec la bibliothèque **`Scikit Learn`** qui prédéfinit la fonction `PCA` [8].

```
from sklearn.decomposition import PCA
pca = PCA(n_components=1)
train['pickup_pca'] = pca.fit_transform(train[['pickup_latitude','pickup_longitude']])
train['dropoff_pca'] = pca.fit_transform(train[['dropoff_latitude','dropoff_longitude']])
```

5.5 Variables supplémentaires à des fins expérimentale

Dans cette section, et à l'égard des attributs qu'on a rajouté et qui semblent avoir une grande influence sur notre objectif de prédiction, nous allons discuter à propos des variables que nous avons rajouté mais qui nous ont pas permis d'obtenir des scores importants pour la simple raison qu'ils ont été moins importantes que les autres variables.

- `manhattan_distance` : moins précise que la distance calculée à partir du *haversine*, le trajet entre deux points dans un plan est une suite de passages horizontaux et verticaux uniquement.
- `precipitation` : Une valeur en millimètre qui indique le nombre de précipitation [9] (s'il pleuvait) pour chaque jour qui correspond au jour du trajet, ces données météorologique on été récupérées à partir du site gouvernementale de New York.
- `pickup_businessday` : Indique si le jour où le trajet a été effectué est un jour de travail (entre lundi et vendredi inclus).
- `ride_count` : le nombre de trajets qui se produisent à chaque heure spécifique.
- `speed` : la vitesse avec laquelle le taxi roulait, elle est la division de la distance sur la durée pour le *DataFrame* d'entraînement, et la moyenne de l'attribut lui même du même *Dataframe* dans les données de *test*, puisque ce dernier ne contient pas de durée (c'est à nous de la prédire).

8. l'hyperparameter *n_ components* indique le nombre de dimensions sous lequel on veut transformer la dimension du début.
9. l'arrtibut precipitation a été utilisé dans l'un de nos modèles d'apprentissage et a été testé sous la régression polynomiale

6 Les modèles de prédiction et algorithmes d'apprentissage

6.1 La Regression

L'objectif de la régression linéaire est de prédire au mieux les valeurs de y à partir des variables de prédication $(x_1, x_2..., x_n)$. Formellement, elle s'écrit : $y = a_0 + a_1x_1 + a_2x_2 + ... + a_nx_n + \epsilon$

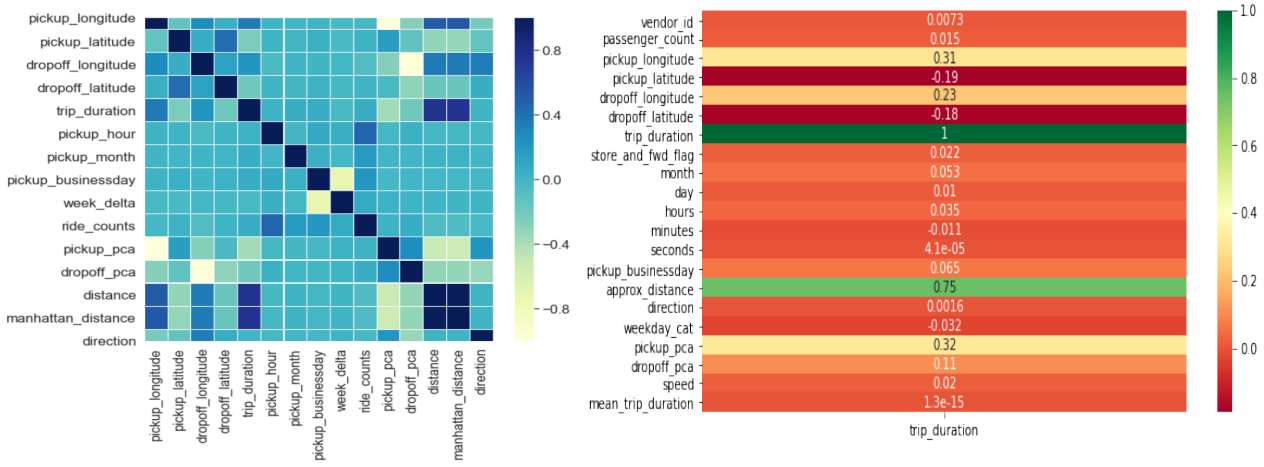
l'erreur ϵ résume les insuffisances du modèle.

Nous allons procéder à entraîner notre modèle en utilisant une régression Polynomiale (Une forme que peut prendre un régression linéaire), elle consiste à augmenter la complexité du modèle en ajoutant à chaque attribut un degré. Ceci a pour but de surpasser le sur/sous apprentissage , cette tâche ne peut être effectuée avec une régression linéaire simple. Par conséquent, l'équation de la régression polynomiale devient : $y = a_0 + a_1x_i + a_2x_i^2 + \ldots + a_nx_1^m + \varepsilon_i (i = 1, 2, \ldots, n)$.

Le modèle utilisée est composée des attributs suivants :

`'distance','precipitation','mean_trip_duration',`
`'weekday_cat','hours','passenger_count','speed', 'day_part'.`

Pour cela nous allons exploiter la matrice de corrélation pour évaluer la dépendance entre plusieurs variables en même temps. Un modèle linéaire est créé en combinant les instances a forte corrélation.



(a) Matrice de corrélation (entre tout les attributs)

(b) Coefficients de corrélation entre l'objectif et tous les attributs

FIGURE 10 – Matrice et coefficients de corrélation après l'ajout des nouvelles variables

6.1.1 Métrique d'évaluation

Nous avons utilisé la même métrique que le site de kaggle dont la notre est fournie par *scikit-learn*, le score obtenu est de 0.2659

6.1.2 Score Kaggle

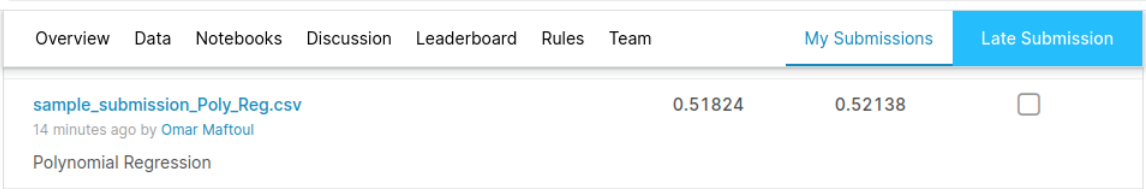


FIGURE 11 – Score Kaggle pour Polynomial Regression (Score privée à gauche)

6.2 SVR

SVR (Support Vector Regression) est une extension du SVM (Support Vector Machine) qui permet de prédire les variables continues à partir des données fournies en entrée de l'algorithme. SVR a été proposé par Vladimir N. Vapnik et ses collèges en 1996.

6.2.1 l'algorithme SVR

Le but de SVR est de trouver la fonction de l'hyperplan qui maximise le nombre de points dans un intervalle de taille [-ε , +ε] (dans la figure 12, on cherche la fonction de la droite continue noire) car on ne peut pas trouver toujours un hyperplan qui engendre tous les points (figure 12) , la fonction de l'Hyperplan ne va pas être utilisées pour séparer les points en deux classes comme dans SVM, mais plutôt pour la prédiction.

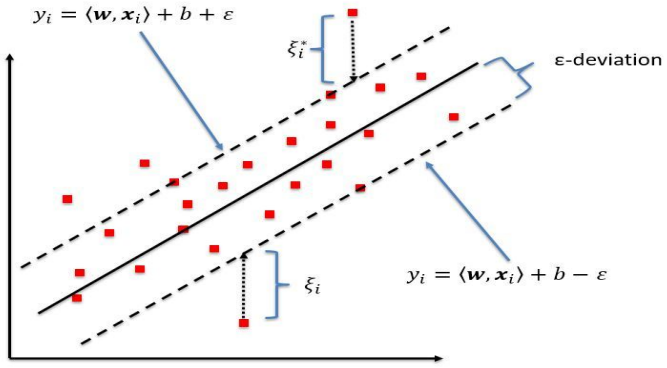


FIGURE 12 – Le modèle SVR

On suppose qu'on a un ensemble d'exemples d'apprentissage $\{(x_1, y_1), ..., (x_l, y_l)\} \subset \mathcal{X} \times \mathbb{R}$ ou $\mathcal{X}$ est le vecteur des entrées du modèle (Ex. $\mathcal{X} = \mathbb{R}^d$), SVR vise à trouver la fonction :
$f(x) = x^T \cdot w + \beta$ Tel que $w \in \mathcal{X}$ et $\beta \in \mathbb{R}$ prédisent la variable de sortie y_i avec une erreur de $\pm\varepsilon$. L'algorithme va résoudre un problème d'optimisation quadratique pour calculer w et β et c'est la raison pour laquelle sa complexité est quadratique.

$$\text{Minimiser} \quad \frac{1}{2}w^T \cdot w + \mathcal{C}\sum_{i=1}^{\mathfrak{l}}(\xi_i + \xi_i^*)$$

$$\text{Suit.à} \quad \begin{cases} y_i - x_i^T \cdot w - \beta \leqslant \varepsilon + \xi_i & \forall i \in \{1, ..., \mathfrak{l}\} \\ x_i^T \cdot w + \beta - y_i \leqslant \varepsilon + \xi_i^* & \forall i \in \{1, ..., \mathfrak{l}\} \\ \xi_i, \xi_i^* \geqslant 0 & \forall i \in \{1, ..., \mathfrak{l}\} \end{cases}$$

ξ_i et ξ_i^* sont des variables qu'on associe à chaque exemple d'apprentissage pour accepter une erreur de prédiction ($|f(x_i) - y_i|$) plus que ε pour certains points.
L'algorithme résolu le problème de dualité [10] associée en utilisant la méthode de **Lagrange**, il associe à chaque exemple d'apprentissage x_i deux coefficients positifs α_i et α_i^* et donc on obtient après la résolution :

$w = \sum_{i=1}^{\mathfrak{l}}(\alpha_i - \alpha_i^*)x_i$ et donc la fonction de prédiction devient : $\boxed{f(x) = \sum_{i=1}^{\mathfrak{l}}(\alpha_i - \alpha_i^*)x_i^T \cdot x + \beta}$

On peut utiliser des fonctions de produits des vecteurs $\mathcal{K}(x_i^T, x)$ pour rendre l'algorithme SVR non-linéaire (noyau) à la place du produit scalaire $x_i^T \cdot x$, mais vu la complexité de l'algorithme on peut pas tester ces noyaux car le nombre d'exemples d'apprentissages dépasse 1 million.

6.2.2 Les paramètres

SVR nécessite la définition de quelques paramètres avant l'exécution de l'algorithme.

Paramètre	Description
ε	l'intervalle d'erreur de prédiction qu'on souhaite obtenir (L'épaisseur du margine)
$\mathcal{C}$	Une constante qui définit le dégrée de tolérance pour les exemples d'apprentissage qui ont le taux d'erreur plus que ε, elle sert aussi à pénaliser le Sur-Apprentissage
L	La fonctionne d'évaluation d'erreur on va utiliser tous le temps la fonction L_ε définit par : $L_\varepsilon = \begin{cases} 0 & Si \quad \lvert f(x_i) - y_i \rvert \leqslant \varepsilon \\ \lvert f(x_i) - y_i \rvert - \varepsilon & Si \quad \lvert f(x_i) - y_i \rvert \geqslant \varepsilon \end{cases}$
max_iter	Le nombre maximum d'itérations

6.2.3 Exécution

On va utiliser la classe ***sklearn.svm.LinearSVR*** définie dans la bibliothèque ***sklearn*** pour exécuter l'algorithme SVR.

```
from sklearn.svm import LinearSVR
SVR = LinearSVR(epsilon = 0 , C=1 , L = 'epsilon_insensitive', max_iter = 2 000 000 )
SVR.fit(X_train, Y_train)
```

Les résultats qu'on a obtenu lors des tests en utilisant différent paramètres et attributs sont représentées par le tableau si dessous :

10. problème d'optimisation pusiqu'on cherche à maximiser une valeur

Paramètres			Attributs	Résultats	
ε	$\mathcal{C}$	max_iter	les attributs utilisés	score (SK-$LEARN$)	score ($KAGGLE$)
0	1	6000	Distance	+0.0028	0.53529
1	1	10000	Distance	+0.0028	0.53948
1	1	1000	Distance	−0.00085	0.54487
0	1	1000	Distance	−0.00085	0.54518
100	1	8000	Distance, Direction	−0.00085	0.55240
1	50	5000	Distance	−0.0077	0.94606

6.2.4 la complexité

SVR est très performant mais le temps d'exécution et l'espace occupé par l'algorithme augmente rapidement en fonction des exemples d'apprentissages et des attributs
$\mathcal{O}(n_{attributs} \times n^2_{exemples})$.

6.3 Random forest (forêts d'arbres décisionnels)

Une forêt aléatoire est basée sur des arbres de décision. Cependant, ces derniers ont tendance à être des classificateurs non biaisés avec une forte variance ce qui conduit dans la plus part des cas au sur-apprentissage. Pour palier à ces différents problèmes nous avons opté pour les forêts aléatoires.

6.3.1 Comment ça fonctionne

Une forêt aléatoire est une technique d'ensemble capable d'effectuer à la fois des tâches de régression et de classification à l'aide de plusieurs arbres de décision. Elle peut aussi utiliser une une technique appelée **Bootstraping ou Bagging**, qui consiste en la formation de chaque arbre de décision sur un échantillon de données différent où l'échantillonnage est effectué avec remplacement (remise).

6.3.2 Algorithme de Random Forest Regressor

- Sélectionnez **N** enregistrements aléatoires dans l'ensemble de données avec remise (methode de bootstrap).
- Pour chaque enregistrement $i \in [1, N]$ Construisez un arbre de décision
- Pour un problème de régression tel que celui ci, pour un nouvel enregistrement i, chaque arbre de la forêt prédit une valeur pour $T(x) = Y_i$ (sortie). La valeur finale peut être calculée en prenant la moyenne de toutes les valeurs prédites par tous les arbres en forêt (Agrégation par la moyenne).

$$F(x) = \frac{1}{N} \sum_{i=1}^{N} Y_i$$

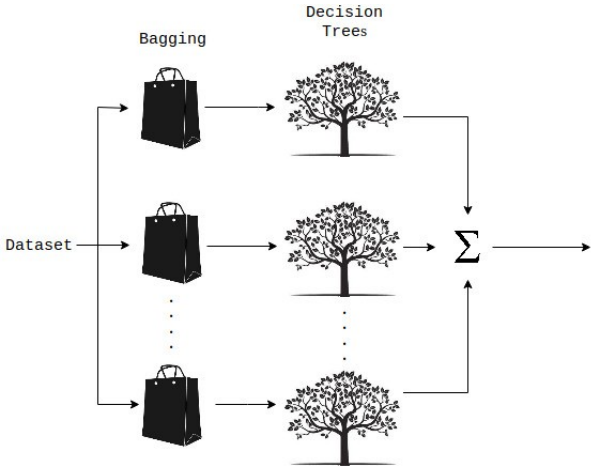


FIGURE 13 – Random forest regressor process

Complexité

Ce qui rend désavantageux les forêts aléatoires c'est leurs complexité en temps qui nécessite plus par rapport aux autres algorithmes comparable. De ce fait, ça nécessite beaucoup plus de ressources de calcul, en raison du grand nombre d'arbres de décision réunis.
Complexité : $O(nTree \times nVar \times Nlog(N))$.
Où :

- nTree : le nombre d'arbres choisis
- nVar : nombre de variables d'entraînement
- N : nombre d'échantillons

6.3.3 Sur le plan pratique

Après avoir traité nos données dans les premières parties, nous allons exploiter la classe **RandomForestRegressor** de la bibliothèque python **scikit-learn** afin de construire un modèle de prédiction des durées des taxis dont l'essentiel est représenté par ce morceau de code :

```
from sklearn.ensemble import RandomForestRegressor
#... autres traitements
rfr = RandomForestRegressor(n_estimators=50, criterion='mse', max_depth=35)
model_rfr = rfr.fit(X_train, y_train)
y_test = model_rfr.predict(X_test)
```

Importance des variables

Random forest est utilisé pour mesurer l'ordre d'importance des variables de prédiction, pour cela dans un premier temps nous avons utilisé toutes nos variables pour entraîner le modèle précédemment paramétré.

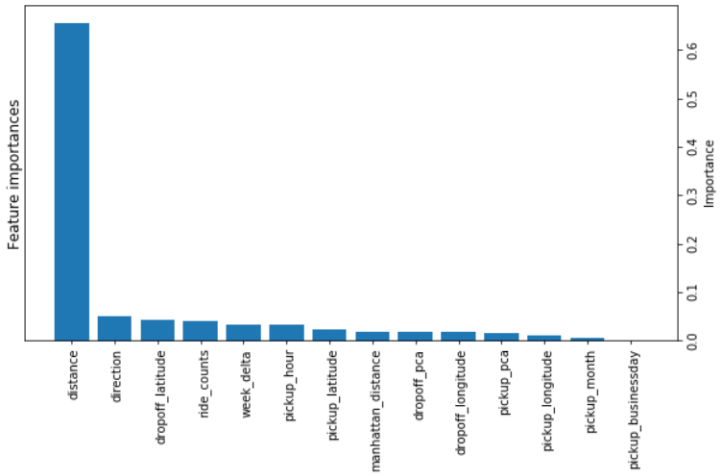


FIGURE 14 – Importance des variables

Choix des hyperparametres

Les hyperparamètres doivent être définis avant le début du processus d'apprentissage et jouent un rôle important dans la performance du modèle. Pour entraîner le modèle nous avons utilisé les hyperparamètres suivants :

— n_estimators : spécifie le nombre d'arbres à construire dans le modèle, sa valeur par défaut est de 10 arbres.

— max_depth : spécifie la profondeur maximale des arbres et a comme valeur par défaut 10.

Afin de trouver les meilleures valeurs qui seront associée à ces deux hyperparamètres nous avons spécifié un ensemble de valeurs pour chacune des valeurs des deux hyperparamètres **max_depth = (10,20,30,35,40)** et **n_estimator = (10,20,30,40,50)**, ce qui forme 25 modèles différents et représenter graphiquement l'erreur quadratique moyenne logarithmique de chaque modèle.

Remarque : *En raison du temps que prenait le traitement en variant les valeurs des hyperparametres nous nous somme arrêté à 25 combinaisons et nous avons fait les test sur des échantillons de 100000 lignes. Pour cela les scores dans le graphe paraissent plus grands.*

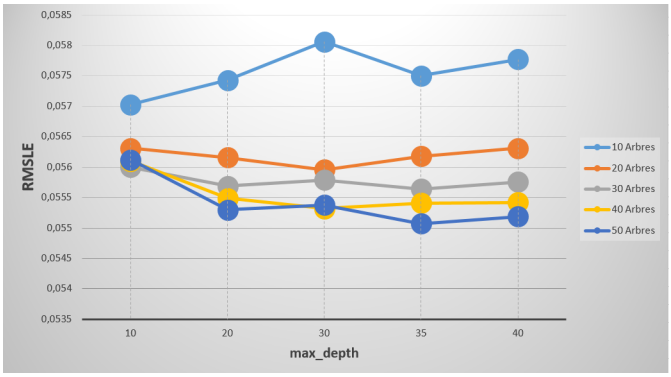


FIGURE 15 – Graphique represantant la recherche des hyperparametres

Évaluation du modèle

Nous nous somme basé pour évaluer ce modèle sur la métric *Root Mean Squared Logarithmic Error (RMSLE)* qui est fournit par la bibliothèque scikit-learn.

```
1 from sklearn.metrics import mean_squared_log_error
2 print('RMSLE', mean_squared_log_error(y_test, rfr.predict(X_train))**0.5)
```

$$RMSLE = 0.018701716851113104$$

6.3.4 Score obtenu sur Kaggle

Jusqu'a présent c'est le meilleur score que nous avons pu obtenir avec la technique de Random Forest malgré les difficultés de performance rencontrées au niveau de la RAM.

Submission and Description	Private Score	Public Score
submissionRFR.csv a few seconds ago by Massinissa rfr	0.40368	0.40647

FIGURE 16 – Score Kaggle pour Random Forest Regressor

6.3.5 Comparatif des scores Kaggle

Dans la figure suivante nous avons fait une petite comparaison graphique entre les 3 modèle que nous avons utilisé précédemment.

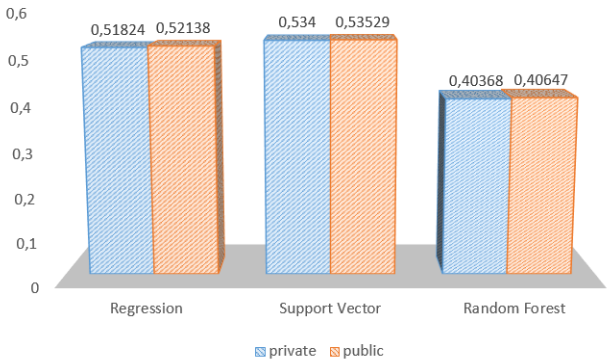


FIGURE 17 – Graphique de comparaison incluant le score public et privé

7 Conclusion

À travers ce projet, nous avons mis en oeuvre différents modèles d'apprentissage automatique qui sont plus ou moins performants les un que les autres.
Ce projet nous a permis aussi de s'initier au vaste domaine du machine learning et d'acquérir de nouvelles compétences dans l'analyse de données et d'exploiter les différentes bibliothèques de *python 3* qui ont permis par la suite d'accentuer nos compétences en programmation et pu réaliser ce projet.
Dans l'avenir, nous espérons également explorer les méthodes de Boosting tel que la librarie *XGBoost*, aussi l'utilisation des *réseaux de neurones*, dont ces derniers constituent un élément essentiel dans les techniques dites de Deep learning (ou Apprentissage profond) et un outil largement répandu pour la prédiction de données tel que le temps.

A
Annexe

A.1 Random Forest Code

```
from sklearn.ensemble import RandomForestRegressor

do_not_use_for_training = [ 'trip_duration','vendor_id', 'store_and_fwd_flag',
                            'pickup_is_weekend', 'passenger_count',
                           'pickup_weekday', 'pickup_holiday', 'kmeans_dropoff_loc']
trainFinal = df[:-len(test)]
testFinal = df[-len(test):]
feature_names = [f for f in trainFinal.columns if f not in do_not_use_for_training]
print('We have %i features.' % len(feature_names))
y_train = np.log(trainFinal['trip_duration'].values + 1)
X_train = trainFinal[feature_names].values

X_test = testFinal[feature_names].values

rfr = RandomForestRegressor(n_estimators=70, criterion='mse', max_depth=35)
model_rfr = rfr.fit(X_train, y_train)

y_pred = model_rfr.predict(X_test)
# Enregistrer les resultat dans un fichier de submission
test['trip_duration'] = np.exp(y_pred) - 1
test.reset_index()[['id','trip_duration']].to_csv('submissionRFR.csv', index=False)
```

A.2 Linear Regression Code

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import r2_score, mean_squared_error


prepared = train_taxi[['approx_distance','precipitation','mean_trip_duration','weekday_cat','hours',
'passenger_count','speed', 'day_part']]
labels = train_taxi['trip_duration'].values.reshape(-1,1)

poly_features = PolynomialFeatures(degree=2, include_bias=False)
prepared_poly = poly_features.fit_transform(prepared)

model_poly = LinearRegression()
model_poly.fit(prepared_poly, labels)


X = test_taxi[['approx_distance','precipitation','mean_trip_duration','weekday_cat','hours',
'passenger_count','speed', 'day_part']]
test_poly = poly_features.fit_transform(X)

my_prediction_poly = model_poly.predict(test_poly)


Submission_poly = pd.DataFrame()
Submission_poly["id"]= test_taxi["id"]
Submission_poly["trip_duration"] = my_prediction_poly
Submission_poly.to_csv("/content/drive/My Drive/Data/sample_submission_Poly_Reg.csv",index=False)
```

A.3 SVR Code

```
from sklearn.svm import LinearSVR
regr = LinearSVR(random_state=0, tol=1e-5, epsilon = 1 ,max_iter = 10000)
```

```
regr.score(data_train['approx_distance'].values.reshape(-1, 1), data_train['trip_duration'])
Z=regr.predict(data_test['approx_distance'].values.reshape(-1, 1))
S = pd.DataFrame()
S["id"]= data_test["id"]
S["trip_duration"] = Z
S.to_csv("submittionsvm.csv",index=False)
```

La totalité des Notebooks sont disponibles sur le repo github suivant : https://github.com/logarithm27/NYC_Kaggle_Challenge