L2 MI - Mini Projet

Challenge **XPorter** - Groupe Scooter
Membres :
Fedy Ben Naceur, Hakim Kasdi,
Xuewen Guo, Kevin Tran,
Hamza Manita Jelidi, Hugo Penicand.
URL du challenge : XPorter
Repo Github du projet : Notre Github
URL of the Youtube video : Notre Video

Contexte et description du problème

Le challenge que nous avons choisie est le challenge XPorter. Notre mission est de prédire le nombre de voitures qui passeront à une date, une heure et des informations météorologiques supplémentaires.

Pour réaliser ce challenge, nous avons un dataset qui s'appelle "all-data", qui contient des informations indiquant la qualité de trafic entre différents jours avec des conditions météorologique, comme par exemple la neige, la pluie etc...

Nous avons séparé le travail à réaliser pour ce projet en 3 parties, chacune traitée par un binôme.
Les 3 parties sont :

- Le Pre-processing, c'est-à-dire la transformation de données brutes en données utilisables par l'étape suivante ;
- Le Model, qui sera chargé de trouver le plus performant model d'apprentissage mais aussi les hyper-paramètres correspondants pour notre projet ;
- La Visualisation, dont la mission est d'affichage des résultats de manière compréhensible et pertinente via une interface graphique claire.

Team Model :
Xuewen Guo & Kevin Tran

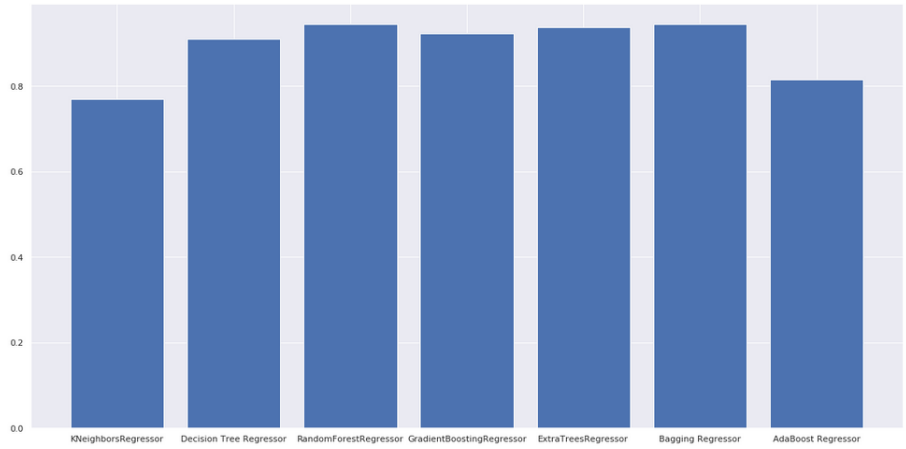
L'objectif de notre groupe était d'entraîner plusieurs modèles de régression, de les tester via la méthode de cross-validation. La première partie de notre tâche a été chercher dans la librairie de *scikit-learn* [1] les modèles que l'on voulait entraîner.

Les modèles sont des méthodes de cette librairie qui lors de son appel à des paramètres (qui sont les hyper-paramètres de notre modèle) prédéfinis. Pour améliorer les scores de notre modèle nous avons dû lire et sélectionner les hyper-paramètres que nous voulions optimiser.

Le choix a dû se faire en tenant en compte le temps d'entraînement de notre modèle, le risque de sur-apprentissage (overfitting), etc..

Les différents modèles testés

```
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
import numpy as np
import matplotlib.pyplot as plt
scores = []
for i in range(len(model_list)):
    scores.append(cross_val_score(model_list[i], X_train, Y_train,
    cv=5, scoring=make_scorer(scoring_function)))
score = np.array(scores)
score=score[:,0]
fig,ax = plt.subplots(1,1,figsize=(20,10))
plt.bar(model_name,score)
```



RandomForest

Lors de notre test, le meilleur modèle de notre liste était le RandomForestRegressor [2]. Dans cette section nous vous expliquerons comment procède le modèle en vous présentant son algorithme [4].

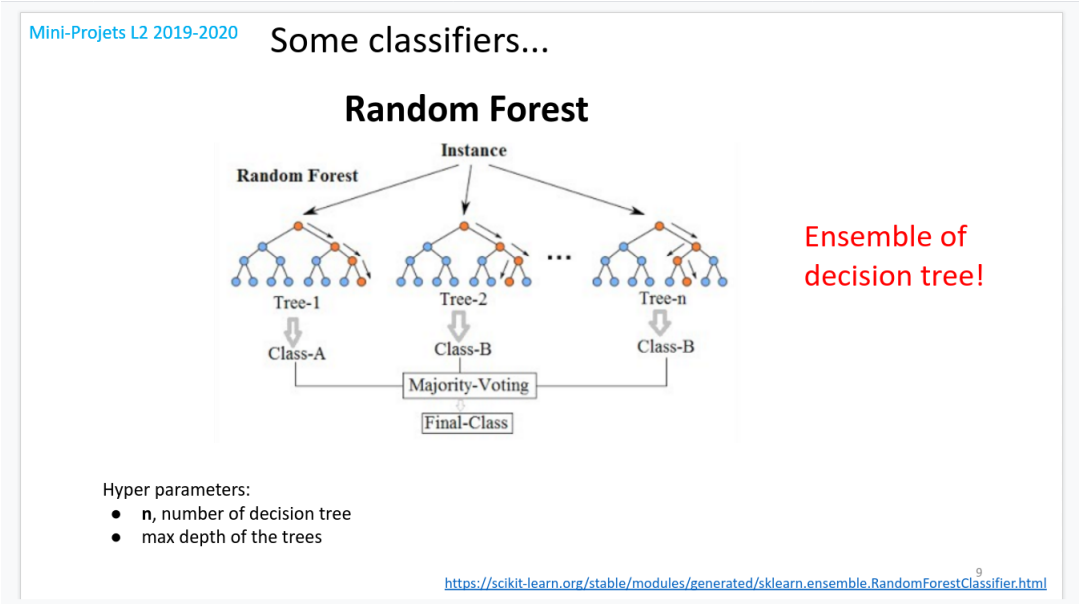
Algorithm 1 Random Forest

Precondition: A training set $S := (x_1, y_1), \ldots, (x_n, y_n)$, features F, and number of trees in forest B.

function RandomForest(S, F)
  $H \leftarrow \emptyset$
  for $i \in 1, \ldots, B$ **do**
    $S^{(i)} \leftarrow$ A bootstrap sample from S
    $h_i \leftarrow$ RandomizedTreeLearn($S^{(i)}, F$)
    $H \leftarrow H \cup \{h_i\}$
  end for
  return H
end function
function RandomizedTreeLearn(S, F)
  At each node:
    $f \leftarrow$ very small subset of F
    Split on best feature in f
  return The learned tree
end function

Cet algorithme permet de créer notre modèle selon un ensemble de données et de features.

Pour un nombre d'arbre fixé dans la forêt (B dans l'algorithme), l'algorithme va créer une liste d'arbres binaires de décision. Pour cela il sélectionne une partie des données $S^{(i)}$. Et appelle la fonction **RandomizedTreeLearn**, cette fonction va créer un arbre en plaçant à chaque noeuds de l'arbre une des meilleurs features testé sur la partie $S^{(i)}$.



De cette façon chaque donnée parcourra les B arbres créer par l'algorithme. Chaque arbre aura la même longueur mais des noeuds différents car chaque partie des données sera différentes. Cela est particulièrement efficace lorsque l'ensemble de donnée S est grand pour pouvoir définir des parties disjoints et ainsi avoir des arbres très différents. L'image ci-dessus donne une idée de ce qu'est notre modèle mais il s'agit ici du modèle non regressor.

Résultats sur le Codalab

Voyant les résultats sur le modèle du RandomForest, nous nous sommes finalement tourné vers le modèle Bagging qui avait un meilleur score sur le ensemble d'entrainement selon le r2 metric.

Team Model :
Fedy Ben Naceur & Hakim Kasdi

Notre groupe était chargé de la tâche de "pre-processing". Notre rôle consiste a créer des méthodes qui nous aideront a avoir des meilleurs performances. Nous avons procéder de la manière suivante :

- On utilise au début les méthodes fournies par sklearn pour détecter les anomalies dans les données et les éliminer ;
- On visualise les données à travers l'algorithme T-SNE (t-distributed stochastic neighbor embedding) ;
- On utilise PCA (Principal Component Regression) pour observer le nombre de "features" optimal qui représente le mieux nos données ;
- On selectionne ensuite les features qui performent le mieux.

0.1 Outliers Detection

0.1.1 Qu'est ce qu'une anomalie dans les données ?

C'est une observation qui diffère du motif global de l'échantillon, qui peut etre de deux type : univariées et multivariées. Outliers univariées peuvent être trouvées en examinant une distribution de valeurs dans un seul espace de caractéristiques. Les outliers multivariées peuvent être trouvées dans un espace à n dimensions (de n caractéristiques). Il peut être très difficile pour le cerveau humain d'examiner des distributions dans des espaces à n dimensions, c'est pourquoi nous devons former un modèle pour le faire à notre place

0.1.2 Code

```
from sklearn.ensemble import IsolationForest
import numpy as np
clf = IsolationForest(behaviour = 'new',
max_samples='auto', random_state = 1,
contamination= 'auto')
preds = clf.fit_predict(data)
preds = np.where( preds<1)
for i in preds:
    data = data.drop(i,axis=0)
```

0.2 Réduction de dimension

0.2.1 T-SNE

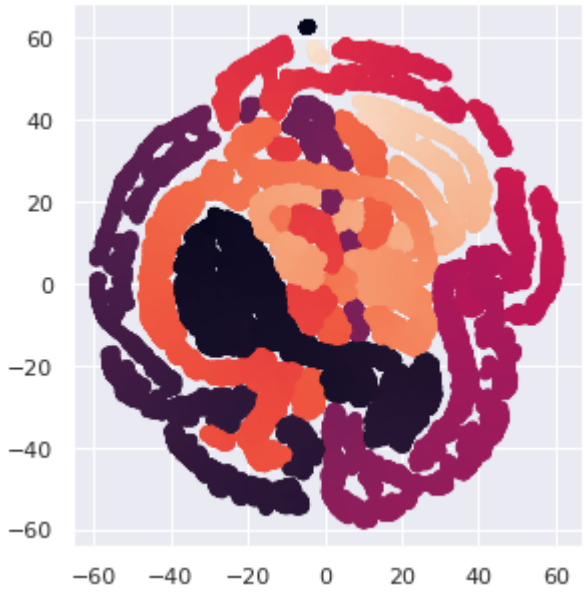
Explication de l'algorithme

T-distributed stochastic neighbor embedding est une technique qui permet d'analyser des données décrites dans des espaces à forte dimensionnalité pour les représenter dans des espaces à deux ou trois dimensions.

Le code

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
X_tsne = TSNE(learning_rate=100).fit_transform(data)
plt.figure(figsize=(10, 5))
plt.subplot(121)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=data.target)
```

Les résultats



0.2.2 PCA

Explication de l'algorithme

Principal Component Analysis pour des données numériques en n dimensions est un algorithme non supervisé d'identification des dimensions de variance décroissante et de changement de base pour ne conserver que les k dimensions de plus grande variance.

Le code

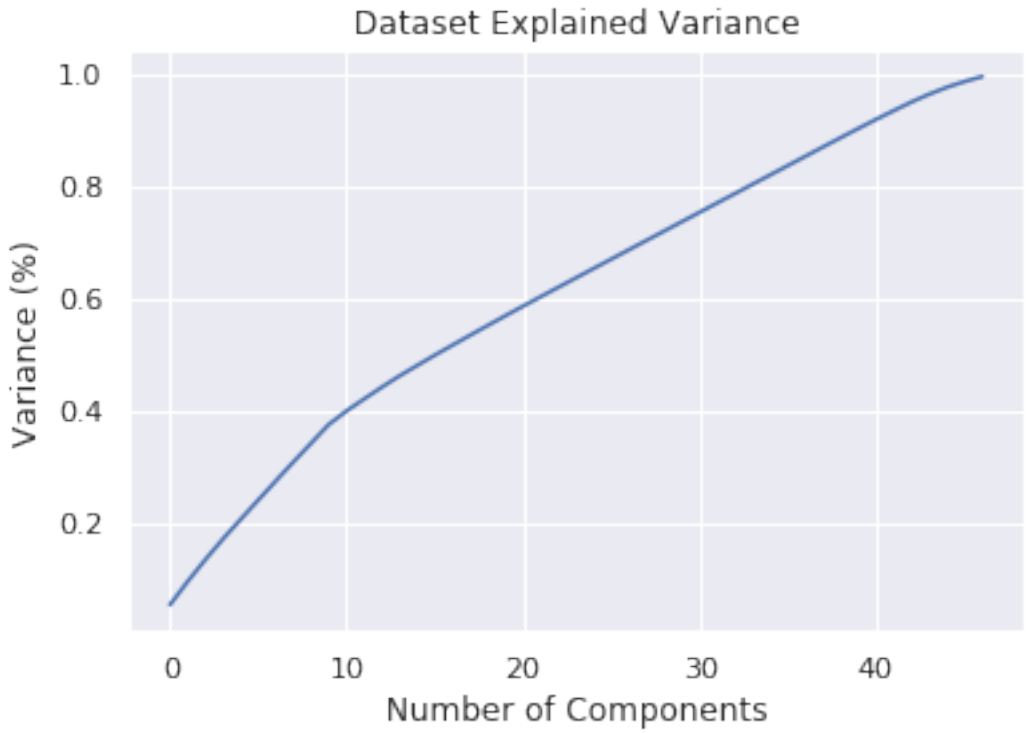
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

#features = data.columns[0:len(data.columns)-1]
#x = data.loc[:, features].values
#y = data.loc[:,['target']].values

scaler = StandardScaler()
scaler.fit(data)
StandardScaler(copy=True ,with_mean=True ,
                                with_std =True)
scaled_data =scaler.transform(data)

pca = PCA(0.99)
pca.fit(scaled_data)

x_pca =pca. transform(scaled_data)
print(x_pca.shape)
pca.explained_variance_ratio_
```

Les résultats

En appliquant la méthode de PCA on a arrivé à observer la participation de chaque "feature" dans la representation des données.Pour concrétiser ce résultat on a fait un plot de nombre de "features" en fonction de de la variance expliqué. Voici le code utilisé et le plot obtenu.

```
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
```

```
plt.ylabel('Variance (%)') #for each component
plt.title('Dataset Explained Variance')
plt.show()
```



0.3 Feature selection

0.3.1 Explication des methodes utilisées

SelectKBest est une class qui note les caractéristiques à l'aide d'une fonction (f regression dans ce cas) et supprime ensuite toutes les caractéristiques sauf les k les plus élevées.

Le code

```
from sklearn.feature_selection import SelectKBest,
                                      f_regression
X = data[data.columns[0:len(data.columns)-1]].values
Y = data[data.columns[-1]].values
X=SelectKBest(score_func=f_regression,k=47).
                                     fit_transform(X,Y)
initial_feature_names = data.columns
for i in initial_feature_names:
    for j in range(47):
        a = np.equal(X[:,j],data[i].values)
    if a[j] :
        print(i)
X.shape()
```

Résultat sur le score final

Après l'utilisation de la méthode montré ci dessus sur le modèle créer par le groupe Model, on remarque que la performance pour certain modèle augmente et pour d'autres diminue. Voici les scores obtenu par le groupe Model :

- KNeighborsRegressor score : 0.77 (+/- 0.01)
- Decision Tree Regressor score : 0.91 (+/- 0.01)
- RandomForestRegressor score : 0.94 (+/- 0.01)
- GradientBoostingRegressor score : 0.92 (+/- 0.01)

— ExtraTreesRegressor score : 0.94 (+/- 0.00)
— Bagging Regressor score : 0.95 (+/- 0.00)
— AdaBoost Regressor score : 0.83 (+/- 0.02)
Voici les scores qu'on a obtenu après la sélection des "features" :
— KNeighborsRegressor score : 0.90 (+/- 0.01)
— Decision Tree Regressor score : 0.89 (+/- 0.01)
— RandomForestRegressor score : 0.94 (+/- 0.01)
— GradientBoostingRegressor score : 0.92 (+/- 0.00)
— ExtraTreesRegressor score : 0.92 (+/- 0.01)
— Bagging Regressor score : 0.93 (+/- 0.01)
— AdaBoost Regressor score : 0.83 (+/- 0.01)

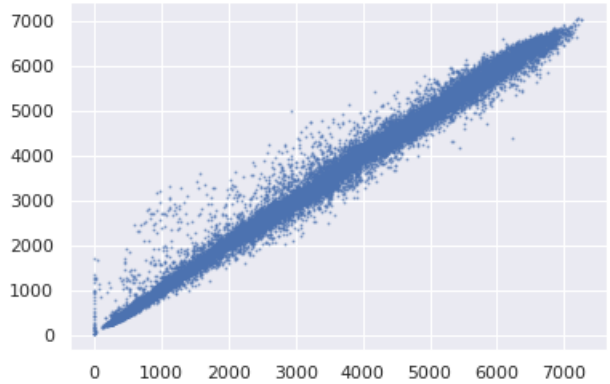
0.3.2 Amelioration et beugs :(nouveau)

Pour améliorer notre modèle on a essayé d'utiliser un pipeline plus complexe en utilisant la standardisation de données puis la PCA et cela en créant deux fonctions preprocess pca preprocess standard que nous utiliseront dans les fonctions fit et predict au cours de l'amélioration on a rencontrer plusieurs bug, et le bug qui nous a causé le plus de problème c'était celui de number of features doesn't match
Après avoir testé plusieurs possibilités pour contourner le problème, on a compris que le problème venait du fait qu'on fit transform les données deux fois
la meilleure méthode qu'on a trouvée c'était de standarizer et d'utiliser la Pca directement dans la méthode fit et predict pour pouvoir contrôler le fit transform en plus de ça on appliqué la methode de SelectKbest.

Team Visualisation :
Hamza Manita Jelidi & Hugo Penicand

L'objectif de notre partie est de fournir une représentation lisible et précise des différentes étapes du projet afin qu'elles soient accessibles et compréhensibles par tous. Cela inclue la mise en évidence des données de départ et la représentation des résultats obtenues par notre model. Il s'agit également de proposer des explications concise et claire, ainsi qu'une interprétation de ce qui est observable à partir de nos interfaces graphiques. Pour cela il est important de varier les types d'affichage et d'user autant que possible des techniques permettant une visibilité accrue (couleur, forme, courbes, etc.).

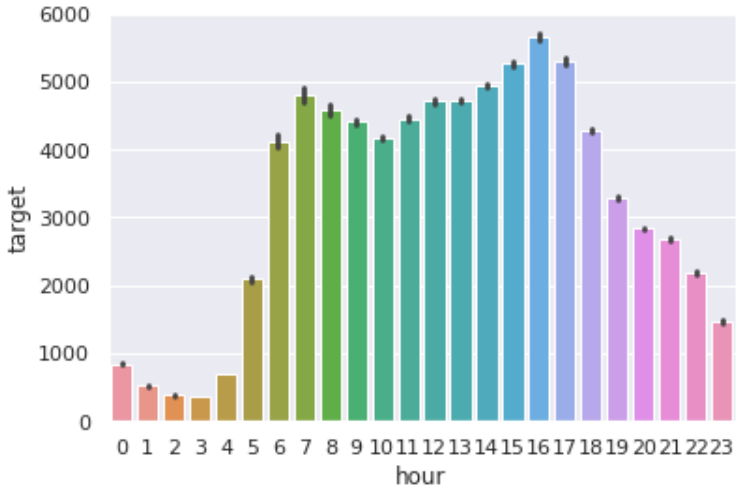
Scatterplot




```
#graphe qui correspond valeurs theoriques en fonction des valeurs observees
plt.scatter(Y_train, Y_hat_train, alpha ='0.5', s = 1 )
plt.show()
```

La régression linéaire simple permet d'évaluer la significativité du lien linéaire entre deux variables. Ci-dessus nous avons Hat train qui correspond aux valeurs théoriques de nos données d'entrainement sur l'axe des abscises (les valeur prédictive), et train qui correspond aux vraies valeurs de nos données d'entrainement sur l'axe des ordonnées (les valeur observée). On peut voir très clairement qu'il y a une corrélation linéaire entre les variables. De plus la courbe observée croit ce qui indique le bon fonctionnement de la régression. Une telle visualisation permet une lecture immédiate des résultats de la régression mais également une bonne compréhension des erreurs éventuelles.

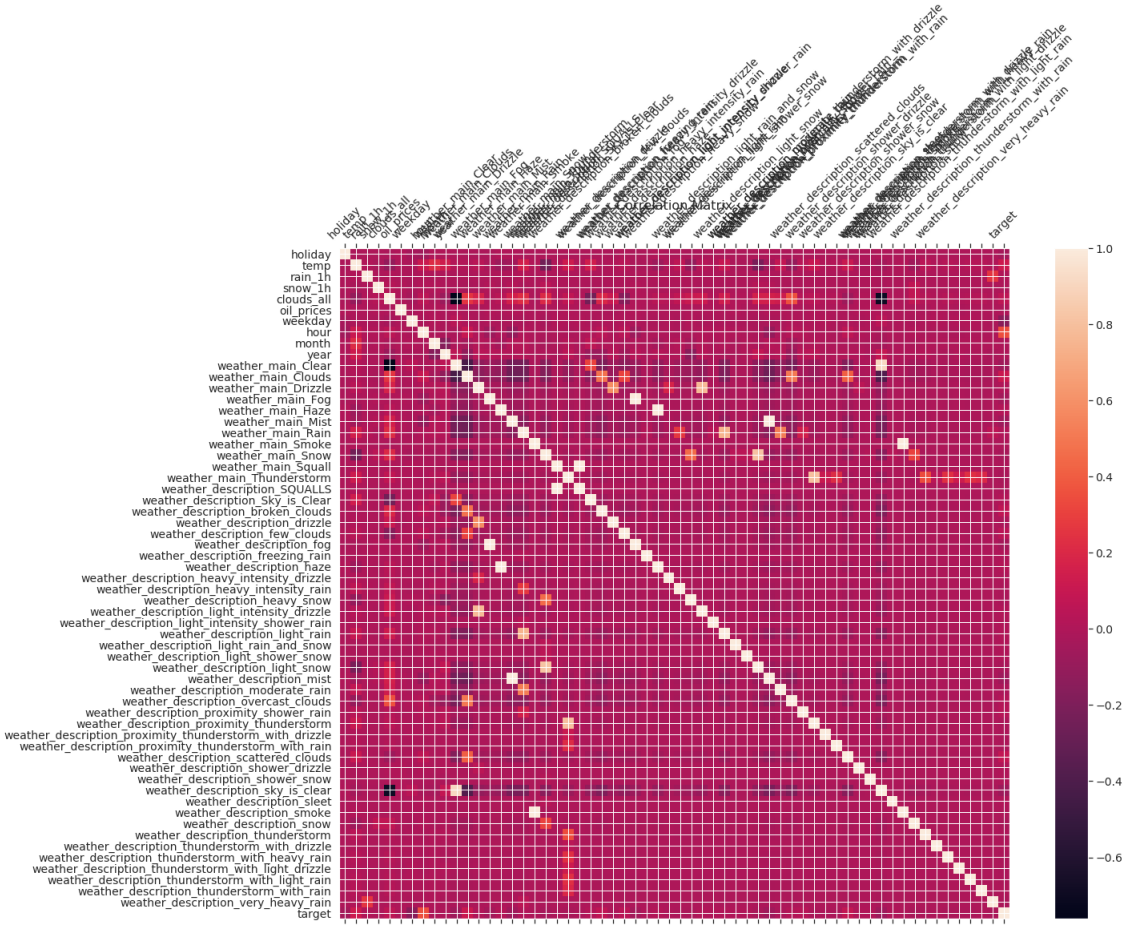
Barplot (Hour,target)




```
#graphe qui compare nos donees en fonction des heures
ax = sns.barplot(x="hour", y="target",data=data)
```

On peut voir ici le volume du trafic en fonction des heures d'une journée analyse : On peut constater qu'il y a 2 piques d'affluences un autour de 7 heure ainsi qu'un autre autour de 16h. Entre ces 2 horraires le trafic restes trés dense. Enfin entre minuit est 5 heure le trafic est trés faible. On peut intérpréter cela comme la période de travail (entre 6h et 18h) qui constitue une sources de deplacement constante.

Correlation Matrix




```
f = plt.figure(figsize=(20, 15))

#on cree la matrice de confusion lie a nos donnee
```

```
plt.matshow(data.corr(), fignum=f.number)
plt.xticks(range(data.shape[1]), data.columns, fontsize=14, rotation=45)
plt.yticks(range(data.shape[1]), data.columns, fontsize=14)
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
plt.title('Correlation Matrix', fontsize=16);
```

Une matrice de corrélations attribue un coefficient de corrélation (un poids) a une paire de paramètre. Plus le coefficient est élevé plus les paramètres sont corrélés.

Analyse : Évidemment chaque paramètre est parfaitement corrélé a lui-même (anti-diagonale blanche). Cependant on peut observer quelque point clair en dehors de cette diagonale nous indiquant une corrélation entre deux paramètres différents.

Bibliographie

[1] Lien vers les différents modèles utilisés :
https://scikit-learn.org/stable/index.html

[2] Lien de la méthode RandomForestRegressor utilisée :
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor

[3] Image du cours sur les différents classifiers :
https://docs.google.com/presentation/d/1vfdS9ttwvK3mzGcoRBoe-n8iVzYOzSKUZsemXjDnPmA/edit#slide=id.g6e8cea48c5_0_65

[4] Lien vers l'algorithme du modèle étudié :
http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/ensembles/RandomForests.pdf

[5] Lien vers la méthode gridSearchCV :
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html