Projet Recherche 2016/2017

Méthodes de segmentation statistiques et Machine Learning

Auteur :
David Zerah

Tuteur :
M. Pascal Yim

Ecole Centrale de Lille

27 avril 2017

Table des matières

Remerciements

Ce projet recherche a pour but de constituer une première approche sur ce qu'un étudiant de première année à l'Ecole Centrale de Lille pourra réaliser lors du Start and Go se déroulant en début de G1.
Durant dix semaines, j'ai tenté de me mettre dans la peau d'un "Data Scientist".
Grâce à cete étude, je suis maintenant à même de proposer une série de tutoriels disponibles dans ce rapport et accessibles à un élève sortant de classes préparatoires aux grandes écoles avec des légères connaissances en Python et en algorithmes.

En premier lieu, je tiens à être reconnaissant envers mon tuteur de projet : M. Pascal YIM pour la proposition de ce sujet et je tiens à le remercier pour la confiance qu'il m'a accordée. Je tenais aussi à féliciter sa capacité d'écoute et j'aimerai souligner sa disponibilité. Ce projet a été pour moi une véritable expérience dans un domaine qui me passionne.

Mes remerciements vont également à M. Egon HEUSON, ingénieur de recherche en Biotechnologies sur Realcat, pour sa patience. Je lui témoigne toute ma gratitude pour son temps, ses explications et ses données qu'il a récoltées à l'aide de la plateforme Realcat.

Je souhaiterais aussi mentionner le livre : "Machine Learning Mastery with Python" de J. Brownlee et la plateforme web "datacamp" qui m'ont notamment aidées pour la rédaction de ce rapport.

Ce rapport a été rédigé en LaTeX.

Introduction

Aujourd'hui, à l'ère du numérique, des données représentants nos goûts, nos envies, nos habitudes, notre manière de parler sont présentes dans les bases de données de différents serveurs : "dans le cloud". Quiconque surfant sur la toile est susceptible de déposer des données dans le cloud le définissant. Face à cette immense quantité de données, un nouveau terme est apparu : "le Big Data".
Le Big Data est donc simplement un terme pour définir toutes ces données figurant dans le cloud, mais imaginons qu'on mélange le Big Data aux statistiques ?
Dans une approche statistique, nous savons que plus l'échantillon prélevé est important, plus nous serons à même d'approximer des données à l'aide d'un modèle qui possède une pertinence la plus exacte possible. Cet echantillon prélevé pourrait donc être ces grosses quantités de données.
Ce mélange conduit donc à l'essort d'une nouvelle discipline : l'apprentissage anticipée ou Machine learning. Le machine learning permet donc de "prévoir le futur" en appliquant des algorithmes qui utilisent des statistiques de haut niveau sur des grandes quantités de données : le big data. Ces algorithmes ont déjà été développés par une série d'ingénieurs et regroupés dans des bibliothèques ouvertes à tous (bibliothèque "open source") dans principalement deux langages : "Python" (Scikit-learn) et "R". Le travail du scientifique des données "data scientist" est donc d'utiliser ces algorithmes sur une batterie de données "dataset" et de sélectionner celui qui obtient la meilleure pertinence (modèle se rapprochant de la réalité).
Ici nous allons seulement nous focaliser sur le langage "Python".

Il est nécessaire qu'un nouvel ingénieur diplomé d'une école généraliste dispose de plusieurs connaissances dans ce domaine, c'est notamment le but du Start and Go se déroulant en début de cursus ingénieur à l'Ecole Centrale de Lille.
Ce rapport, destiné aux étudiants aura pour but de constituer une première approche de ce qu'un : "data scientist" doit réaliser.
L'étudiant devra néanmoins avoir des notions d'algorithmique et connaître la syntaxe du langage "Python".

1 Un premier exemple : le Titanic

1.1 Préparation avant exécution

Aujourd'hui, les données sont partouts, nous pouvons aisément en recenser des dizaines de milliards que des sociétés peuvent collecter et demander à des scientifiques d'analyse "data analyst" de les analyser ou bien nous pouvons demander à les traiter de manière intelligente et autonome à des "data scientists".
Cette batterie de données pouvant figurer dans un seul fichier se nomme : "dataset". Afin de s'entrainer à faire fonctionner des algorithmes d'analyse et prédiction, le site "Kaggle" `https://www.kaggle.com` fournit des exemples de dataset.
Une fois ces données rassemblées, celles-ci doivent suivre le processus suivant :

- Prétraitement des données : c'est le nettoyage des données afin de filtrer les données qui nous intéressent. Nous utiliserons la bibliothèque "Pandas" pour la réalisation de cette étape
- Analyse des données : cela a pour but de répondre à la question : quelles sont les données intéressantes qui ressortent ?
- Application de l'algorithme de prédiction "le machine learning" sur les données intéressantes. Nous utiliserons la bibliothèque "Scikit-learn" pour prélever les algorithmes déjà existant.

Nous utiliserons le language Python.

1.1.1 Installation d'Anacondas et des bibliothèques Pandas et Scikit-Learn

Afin d'utiliser Python, il est nécessaire d'utiliser un logiciel qui permet d'encoder Python et Jupyter et le rendre exécutable par une machine : Anacondas (Télécharger Anacondas : `https://www.continuum.io/downloads`)
Anacondas fournit une bibliothèque : Jupyter qui permet d'encoder et exécuter Python.
Attention : Un bug récurrent apparaissant sur Mac OS Yosémite (impossibilité de l'ouvrir) peut être très facilement résolvable en tapant la commande dans la console (Terminal) :

```
Code Terminal :
sudo rm -rf ~/.continuum/
```

Une fois Anacondas installé, il est nécessaire d'installer la bibliothèque Pandas, Seaborn et Scikit-Learn, de ce fait il suffit de taper la commande dans la console (Terminal) :

```
Code Terminal :
conda install pandas
conda install seaborn
conda install scikit-learn
```

Pour avoir plus de précisions sur Pandas et Scikit-Learn, je conseille le lecteur de visiter la documentation en anglais de ces deux bibliothèques

- Pandas : `http://pandas.pydata.org/pandas-docs/stable/`
- Scikit-Learn : `http://scikit-learn.org/stable/documentation.html`

Vous le découvrerez très vite, ces deux bibliothèques constituent les prérequis fondammentaux pour tous les data scientists.

1.1.2 Pré-requis avant exécution

Une fois Anaconda-Navigator lancé et les bibliothèques Pandas et Scikit learn installées, il reste à lancer Jupyter pour exécuter le code Python. Jupyter est un logiciel qui permet de tester et modifier facilement des commandes en interactif, d'afficher des graphiques, ajouter des titres et des commentaires. Pour ma part, je conseille de placer les datasets téléchargés dans un dossier nommé "Jupyter" qu'on ira chercher avec Jupyter en parcourant les fichiers de notre disque dur. Jupyter s'exécute à travers une page web et s'ouvrant dans un navigateur web. En clair, nous disposons d'un fichier .ipynb qui comprend les algorithmes pour traiter les fichiers de données Excel .csv.
Pour exécuter les algorithmes, il faut alors ouvrir le fichier .ipynb, on retrouvera ainsi les commandes et les commentaires. Cela devient très facile de les modifier pour réaliser des choses intéressantes.

L'exécution se fait bloc par bloc en cliquant sur le bouton :  (ou bien en appuyant simultannément sur les touches SHIFT + Entrer)

1.2 Titanic : Machine learning from disaster

Nous sommes désormais prêt à lancer notre premier algorithme d'analyse de données.
Un des premiers algorithmes les plus intéressants est le Titanic, il permet d'analyser les chances de survies à bord du Titanic en fonction de plusieurs paramètres (Age, sexe, catégories des passagers, le lieu d'embarquement, le type de place et la corrélation entre les proches présents à bords ou non).
Pour télécharger les données : `https://www.kaggle.com/c/titanic/data`
Pour télécharger le code sur Github : `https://github.com/davzer27/datascience/blob/master/Titanic.ipynb`

Les premières lignes ne sont pas très intéressantes dans la compréhension, il s'agit juste de l'importation des bibliothèques pour des simulations et traiter des données, on retrouvera ainsi l'importation des bibliothèques pandas, seaborn et sklearn.

1.2.1 Lecture et classification

Le but est de classifier les données, en effet dans l'Excel, nous avions en une seule ligne toutes les données collées, par exemple : 1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21171,7.25,„S
Grâce à Pandas, on peut rapidement les classifier et les mettre sous forme de matrice colonne :

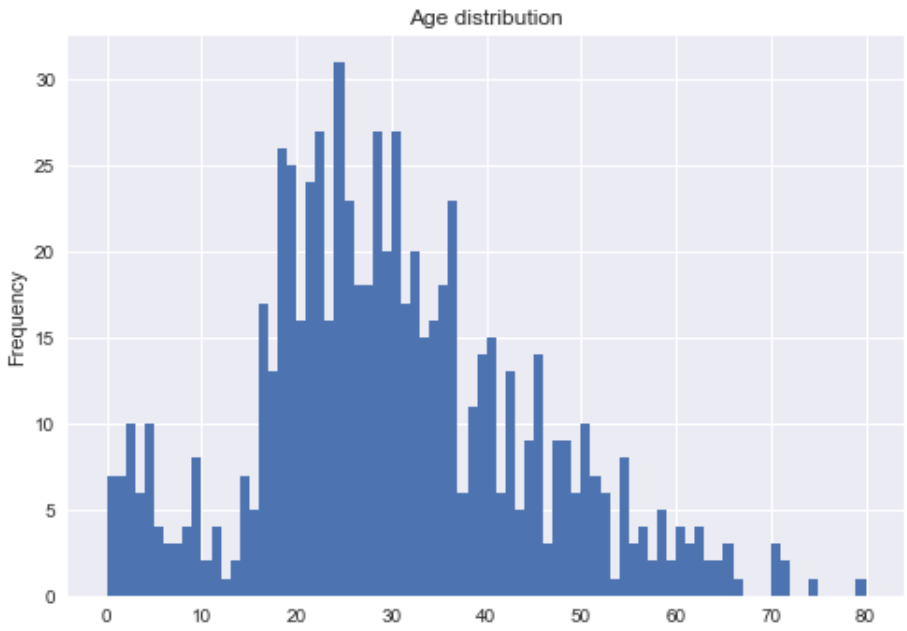
$$\begin{pmatrix} PassengerId & Survived & Pclass & Name & Sexe & Age \end{pmatrix}$$

On pourra ainsi de se souvenir de la ligne $data_train = pd.readcsv('train.csv')$, cela a permis de convertir nos données sous forme de dataframe dont nous verrons plus tard la signification et l'utilisation. La fonction « **head()** » va permettre de nous montrer le résultat de ce qu'on a fait en exhibant les premières valeurs. La fonction « **.info()** » nous donnera des informations sur les données dont on dispose, le nombre et les types de données classifiées.

1.2.2 Spécificités des données

Le travail d'analyse "humain" des données (deuxième phase dans le traitement des données) a abouti sur la conclusion suivante : les chances de survie à bord du Titanic sont indépendantes du numéro de chaque passager, du nom et de son ticket, on les supprime donc dans notre analyse à l'aide de la fonction **drop**(['PassengerID, ..'], axis = 1).
De plus, nous avons remarqué que les colonnes : tarif et l'âge du passager semblaient avoir une influence sur les chances de survies, ainsi nous allons analyser leur corrélation.
Les survivants : Grâce à la fonction **value_counts**, nous pouvons aisément compter le nombre d'éléments de la colonne. Intéressons nous au nombre de survivants (0 pour mort, 1 pour survivant). Sur 891 personnes recensées, 549 sont mortes et 341 ont survécues.
La catégorie des passagers : Nous avons recensé trois catégories de passagers (1 : 216 pers, 2 : 184 pers, 3 : 491 pers). Pour chaque catégorie, on trace les graphiques selon si le passager a survécu ou non. Nous en déduisons que la classe 1 a le plus de chance de s'en sortir, suivi de la classe 2 et enfin la classe 3.
Le sexe : Analysons ensuite les chances de survies en fonction du sexe, on compte simplement le nombre de survivants et on l'affiche sur un graphe

(On compte que les survivants car s'il est mort, sa valeur est 0 donc on ne le comptabilise pas). Nous en déduisons que les femmes ont le plus de chance de s'en sortir vivantes.
L'âge : On suit le même principe, on établit la fréquence des âges dont les passagers ont survécu. Les personnes entre 20 et 30 ans ont eu le plus de chance de s'en sortir.
Certaines personnes n'ont pas d'âge et dispose d'une valeur « NaN » (Not a Number), par conséquent pour avoir un échantillon complet, nous remplissons les valeurs « NaN » avec un échantillon aléatoire en considérant la moyenne des âges.
Et en refaisant le calcul, on en déduit la même conclusion.
Pour avoir un graphique plus clair, nous pouvons afficher la densité à l'aide de la fonction kind= « density ». On en déduit la même chose, les personnes qui ont environ 23 ans ont le plus de chance de survivre au drame.



Sisb : Nombre de frère/soeur, époux/épouse à bord du Titanic (Number of siblings/spouses aboar) :
On en déduit six catégories différentes allant de 0 à 5.

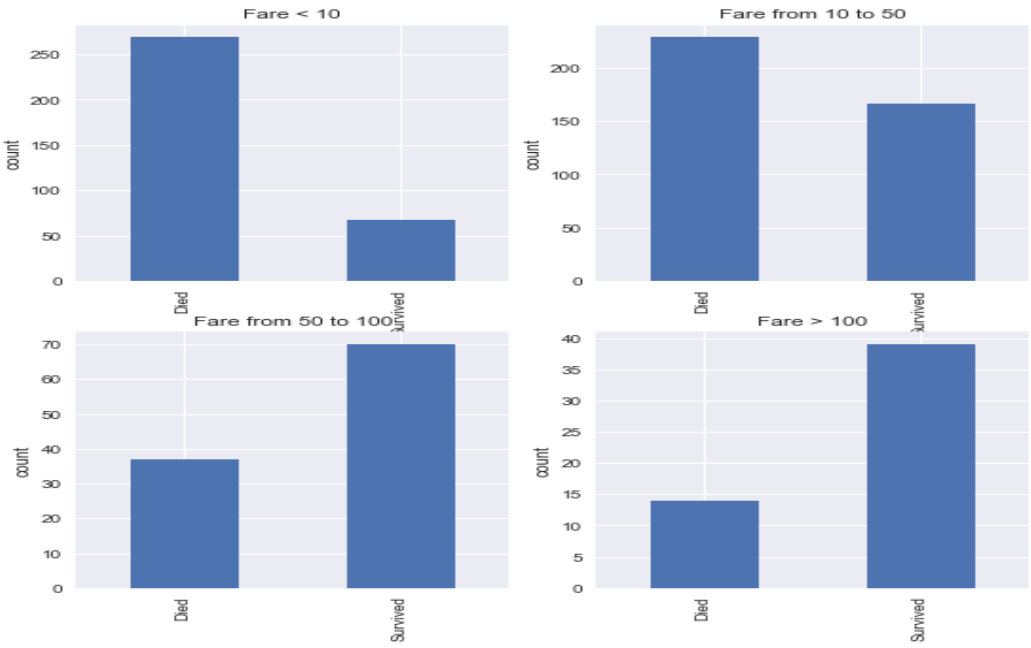
- 0 : 608 passagers
- 1 : 209 passagers
- 2 : 28 passagers
- 4 : 18 passagers
- 3 : 16 passagers
- 8 : 7 passagers

— 5 : 5 passagers

On trace ensuite le graphique pour avoir le nombre de survivants en fonction du nombre de frères/sœurs.
On en déduit que plus de 200 personnes ont survécu avec 0 frère et sœur et 400 sont mortes également. La corrélation est difficile à établir.

Parch : Nombre de parents/enfants à bord (number of parents/children aboard) : C'est le même principe que précédemment, nous avons distingué 7 catégories allant de 0 à 6. Plus de 200 personnes ont survécu alors qu'ils n'avaient 0 enfant/parent et plus de 400 sont mortes alors qu'ils n'avaient 0 enfant/parent.
La corrélation est difficile à établir.

Tarif (Fare) : On suit le même raisonnement que précédemment. On utilisera les fonctions kind = « density » pour avoir la densité en fonction des tarifs. Cela nous fait exhiber quatre catégories : Tarif < 10, Tarif $\in [10, 50]$ tarif $\in [50, 100]$, Tarif > 100.



Ainsi, on en déduit que ceux qui ont des tarifs le plus élevé ont le plus de chance de survivre.

Le fichier test constituera l'échantillon d'apprentissage pour établir une prédiction (nous le verrons dans la suite du rapport). Aisni dans celui-ci, certaines valeurs de "Tarif" ne sont pas des nombres, on les remplie en considérant une moyenne pondérée avec le fichier train.csv.

La formule est : $\sum_{i=0}^{n}(F_{itrain} + F_{itest})/(nb_{tariftrain} + nb_{tariftest})$
Cabine : En faisant un calcul, il existe 687 valeurs non nul sur 891, on se débarrasse de cette valeur avec la fonction drop.
Port d'embarquement : On trouvera 3 valeurs distinctes :

- S(Southampton) : 644 passagers
- C(Cherbourg) : 168 passagers
- Q(Queentown) : 77 passagers

Il existe seulement deux personnes dont on ne connaît pas la provenance d'embarquement (pour connaître le nombre, nous avons simplement utilisé la formule : $difference = data_train.PassengerId.value_counts().sum() - data_train['Embarked'].value_counts().sum()$)

1.2.3 Modèles et prédiction

L'idée est d'appliquer quatre modèles d'apprentissage automatique pour pouvoir faire des prédictions. On retiendra notamment les méthodes : Random Forest, SVC, KNN, Logistic regression qui nous permettent de nous donner des estimateurs.
Nous les appliquons sur un échantillon de données : le dataframe "test" sur lequel l'algorithme va apprendre.
Les bibliothèques installées permettent de faire tourner ces modèles d'apprentissage automatique et d'en mesurer la pertinence en les comparant avec les données récoltées. Le modèle SVC est celui qui est le plus adapté dans cette situation et sera donc utilisé pour faire des prédictions sur d'autres passagers (passager numéro 892, 893 pour savoir s'ils vont survivre ou non.)

1.2.4 Conclusion

Malgré des algorithmes très perfectionnés et une automatisation du traitement, il est nécessaire d'avoir un regard humain afin de supprimer les données inutiles qui perturberaient l'élaboration d'un modèle.

1.3 Conclusion

Une fois ce premier exemple étudié, il est maintenant temps d'entrer dans le vif du sujet et comprendre les différentes notions abordées dans cet exemple.
Il faut savoir que toutes les données sont stockés dans des "dataframes", plus simplement il s'agit d'une matrice composant les données du fichier Excel. Pour simplifier les choses, nous considérons que ces "dataframes" sont des "nouveaux types de tableaux" définies par la bibliothèque **Pandas**.
Sur celles-ci, nous allons effectuer des opérations comme isoler des colonnes,

prélever un échantillon d'une colonne, sommer des colonnes...
Grâce à ces opérations déjà définies dans la bibliothèque "Pandas", nous serons dans un premier temps capable d'analyser plusieurs dizaines de millions de données de manière très efficaces puis dans un second utiliser la bibliothèque "Scikit-learn" pour réaliser des prédictions.

2 Pandas et ses Dataframes sous Python

Vous vous souvenez de la ligne de code "import pandas as pd" ? C'était l'appel d'une bibliothèque open source écrite pour le langage Python qui permet l'analyse et la manipulation des données. Ainsi, toutes les commandes faisant appel à la bibliothèques Pandas commenceront par "pd" (as pd).
Dans notre exemple du Titanic, il permettait de dissocier les données en différentes colonnes pour mieux les analyser, interpreter en réalisant des opérations et ensuite élaborer des prédictions sur celles-ci. Nous allons maintenant découvrir leur fonctionnement.

Ce tutoriel a été élaboré grâce à la plateforme de cours très complet de datacamp : https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python
Astuce : Pour connaître les solutions des exercices, il suffit de tourner prendre la page à l'envers.

2.1 Notion de dataframe

Un dataframe est un objet fréquemment utilisé dans le traitement des données. Il s'agit d'une vulgarisation d'une matrice $\mathcal{M}_{n\times m}$ comportant un certain nombre de données.

Par exemple :

	Colonne 1	**Colonne 2**
Ligne 1	1	2
Ligne2	3	4

Pandas est un puissant outil qui permet de travailler avec des structures de données expressives. L'une des plus intéressantes est le dataframe. Mais pourquoi utiliser un dataframe dans nos données ?
En transformant nos données sous forme de matrice, cela permettra d'utiliser d'autres fonctions de pandas pour manipuler très rapidement ces données comme par exemple la fonction .head() qui nous permettait d'afficher un échantillon des premières données. Pandas s'occupe automatiquement de transformer le fichier .csv (comportant nos tableaux de données) sous forme dataframe traitable grâce à la fonction $pd.read\_csv('fichier.csv')$
On avait importé la bibliothèque Pandas "import pandas as pd", la fonction pd.read va se charger automatiquement de transformer le fichier en dataframe. En réalité Pandas permet de modifier l'en-tête de nos matrice contrairement à une autre bibliothèque : NumPy (on pourra se souvenir de l'importation de la bibliothèque par la commande : *import numpy as np*). NumPy permet donc d'élaborer des tableaux que nous nommerons "array". Les en-têtes caractéristiques des dataframes de Pandas se nomment "des frames".

Numpy : $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ Pandas : $\begin{bmatrix} & \textbf{Colonne 1} & \textbf{Colonne 2} \\ \textbf{Ligne 1} & 1 & 2 \\ \textbf{Ligne2} & 3 & 4 \end{bmatrix}$

Ces dataframes sont très importantes pour traiter nos données, comme nous l'avons vu dand l'exemple du Titanic, les élaborer était la partie préliminaire. En effet, cela nous permettait de compter le nombre d'entrée, d'élaborer des graphes et des corrélations entre les différents champs ou bien de tester des modèles d'estimateur pour prédire les futurs données qui pourraient apparaître. De plus les dataframes peuvent très facilement se manipuler, en effet on peut déterminer le type de données, supprimer des lignes, des colonnes etc.. *Mais comment crée et manipuler des dataframes ?*

2.2 Crée et manipuler des dataframes

Nous pouvons commencer cette partie par une question : *comment crée des dataframes à partir d'un fichier de donnée .csv ?*

Réponse :

Avec la commande pd.read csv, Pandas se charge automatiquement d'élaborer les dataframes.

Pandas nous simplifie donc les choses mais il est intéressant de voir comment élaborer et manipuler ces dataframes.

2.2.1 Création de dataframes

Nous partons d'un tableau (array) que la bibliothèque Numpy nous fournit : $\begin{bmatrix} "" & Colonne1 & Colonne2 \\ Ligne1 & 1 & 2 \\ Ligne2 & 3 & 4 \end{bmatrix}$ Il s'agit d'un tableau à plusieurs dimensions qui peut être interprété comme une matrice. On peut le générer avec la commande :

```
Code Python :
        ma_var = np.array([['','Fruit1','Fruit2'],
                ['Couleur1','Jaune','Vert'],
                ['Couleur2','Orange','Rouge']]);
```

Nous venons de crée un tableau à deux dimensions modélisable par une matrice : $\begin{bmatrix} & Fruit1 & Fruit2 \\ Couleur1 & Jaune & Vert \\ Couleur2 & Orange & Rouge \end{bmatrix}$

Pour pouvoir aisément manipuler des données et élaborer des prévisions futures, il est utile de le transformer sous forme de dataframe.

Un dataframe est composé de

— Un ensemble de données (dans notre exemple : Jaune, Vert, Orange) : on les nommera data
— Des lignes (dans notre exemple, une seule ligne : Couleur) : on les nommera index
— Des colonnes (dans notre exemple précédent : Banane, Poire et Pêche) : on les nommera columns

```
Code Python pour passer d'un tableau Numpy à un dataframe :
       ma_var_numpy = np.array([['','Fruit1','Fruit2'],
                ['Couleur1','Jaune','Vert'],
                ['Couleur2','Orange','Rouge']]);
       ma_var_pandas = pd.DataFrame(data=ma_var_numpy[1:,1:],
                  index=ma_var_numpy[1:,0],
                  columns=ma_var_numpy[0,1:]);
```

Le code est assez simple à comprendre, on utilise la fonction pd.DataFrame en indiquant la plage de l'ensemble des données (data), de l'ensemble des lignes (index) et de l'ensemble des colonnes (columns), cependant la syntaxe de la section des plages reste à éclaircir.
Il faut partir de notre tableau type "numpy" et assimiler plusieurs choses.
En effet, il faut déjà savoir que la numérotation du tableau commence à 0, par exemple pour afficher à Fruit 1, il faudra écrire

```
print(ma_var_numpy[0][1]);
```

La deuxième chose à retenir est d'assimiler la syntaxe ":", par exemple 1 : signifie qu'on sélectionnera toute la partie du tableau à partir de l'indice 1 ainsi :

```
ma_var_numpy[0:][1]; // On sélectionne toute la ligne "0"
// cela affichera ma_var_numpy[0][1], ma_var_numpy[1][1] etc..
```

Après toutes ces notions éclaircies (ou rappelées !), le code devient très simple à comprendre, on prélève simplement les valeurs du tableau NumPy et on remplie le dataframe. On pourrait aussi très facilement crée une DataFrame à partir d'un tableau NumPy sans personnalisation des lignes et des colonnes, ce serait simplement de la transposition d'un tableau en matrice.

```
ma_matrice = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6]]));
```

Nous n'avons pas crée cette structure de donnée pour rien, de ce fait je vous donne deux fonctions assez utiles pour nos données :
La fonction **shape()** : qui retourne la dimension de notre matrice DataFrame. Elle s'utilise de cette manière

```
ma_matrice.shape // Retournera (n, m)
// où n et m sont le nombre de lignes et colonnes
```

La fonction **len** peut nous retourner le nombre de ligne. Pour avoir le nombre de ligne, elle s'utilise de cette manière :

```
len(ma_matrice.index); // retournera le nombre de ligne
```

La fonction **count** est aussi très utile, elle permet de compter le nombre de ligne (en excluant les valeurs NaN), elle s'utilise de cette manière :

```
ma_var[0].count(); // retournera le nombre d'entité
```

La fonction **list** permet de nous renvoyer la liste des colonnes, ou ligne, elle s'utilise de cette manière :

```
list(ma_var.columns.values);
```

2.2.2 Exercice : Création de DataFrame à partir de tableau NumPy

On considère les données de ce tableau :

Sexe	Nom	Prénom	E-mail
H	N1	P1	E1
F	N2	P2	E2

1- À partir de ce tableau, donner le code Python pour obtenir le tableau NumPy (array).

Solution :

```
np.array([['Sexe', 'Nom', 'Prénom', 'E-mail'], ['H', 'N1', 'P1', 'E1'], ['F', 'N2', 'P2', 'E2']])
```

2- À partir du tableau type NumPy, donner le code Python pour obtenir le dataframe correspondant.

Solution :

```
pd.DataFrame(data=VAR[1 :,1 :], index=VAR[1 :,0], columns=VAR[0,0 :])
```

2.3 Sélection de données

Nous avons appris à construire des DataFrames, l'intérêt est de pouvoir aisément les manipuler, ainsi la première manipulation consiste en la selection de plage d'une DataFrame.
Souvenez-vous, il était très simple d'accéder aux valeurs de notre tableau NumPy (un array !)

```
Code Python :
       ma_var_numpy = np.array([['','Fruit1','Fruit2'],
```

```
['Couleur1','Jaune','Vert'],
['Couleur2','Orange','Rouge']]);
print(ma_var_numpy[0][2]); // Affichera "Fruit2"
```

Nous avons aussi vu comment accéder à l'ensemble des colonnes ($list(mavarDF.columns.values)$).

Considérons le DataFrame suivant :

Sexe	Nom	Prénom	E-mail
H	N1	P1	E1
F	N2	P2	E2

Nous souhaitons accéder à la valeur N1 de notre matrice, ici nous avons un dataframe et non un tableau NumPy, nous ne pouvons donc pas utiliser : $ma_var[1][1]$. On utilise la fonction **iloc ou loc**, celle-ci nous permet d'accéder à la valeur d'un élément : $ma\_var.iloc[0][0]$ ou bien $ma\_var.loc[0]['Nom']$ avec le nom de la colonne.
Pour sélectionner la ligne numéro i correspondante à une colonne on peut utiliser le code $df.iloc[i]$ où $i \in [1, n]$.
Petite question : Comment sélectionner toute la colonne 'Nom' avec la fonction loc ?
Solution : df.loc[:,'Nom']
Ca y est, nous connaissons maintenant tout pour accéder aux données des dataFrames.

2.4 Remplissage d'un DataFrame déjà existant

Pour ajouter une ligne, cela est très simple en utilisant la fonction loc (pour accéder aux lignes)

Code Python :
```
df.loc[i] = [0, 1, 2];
```

Cette ligne se nommera automatiquement i et sera remplie par 0, 1 et 2.
Pour ajouter une colonne, même chose sans utiliser loc :

Code Python :
```
df.['Nom Colone'] = [0, 1, 2];
```

On aurait aussi pu utiliser la fonction de Pandas : $pd.Series$ pour ajouter une colonne :

Code Python :
```
df.loc[:, j] = pd.Series(['0', '1'], index=df.index)
#On ajoute à la colonne j, les nombres 0 et 1
```

2.5 Suppression de DataFrame

Nous avons déjà vu qu'une des approches du travail de la science des données fut l'analyse des données afin d'éliminer celles qui n'ont pas d'importance dans l'élaboration de notre modèle. Il est maintenant temps d'apprendre à supprimer des lignes ou des colonnes particulières.
Nous allons simplifier les choses et ne pas trop rentrer dans les détails. Pour supprimer une colonne, cela est très simple et il existe deux manières différentes de le faire.
Le plus simple reste en indiquant seulement le titre de la colonne (dans notre exemple précédent Fruit1).

```
Code Python :
df.drop('Fruit1', axis=1, inplace=True) # supprimera la colonne Fruit1
```

Mais à quoi servent les arguments axis et inplace ?
$Axis = 1$ permet de renseigner que nous sommes en train de supprimer une colonne.
$Inplace = true$ permet de supprimer le dataframe sans l'avoir réinitialisé (avec les dataframes de Pandas, il faut toujours les initiatliser !).
Pour supprimer une ligne, cela est le même raisonnement avec la fonction drop.

```
df.drop(df.index[0]) # supprimera la première ligne
```

2.6 Renommage de DataFrame

Pour renommer un DataFrame, cela est très facile, on utilise la fonction rename en stipulant les modifications. La fonction rename fonctionne de cette manière :

```
df.rename(columns={'Fruit1' :'Aliment1'}, inplace=True)
```

On stipule l'ancien élément en indiquant le nouveau et l'argument inplace permet de réaffecter le dataframe.

Exercice : Comment renommer une ligne ?
Il suffit de remplacer columns par rows

2.7 Applications

Maintenant que nous sommes capable de manipuler des dataframes, il est temps de nous en servir et de comprendre le fonctionnement.

2.7.1 Supprimer les occurrences dans une chaine de caractère

Supprimer les occurrences dans une phrase est un exercice typique dans tout langage de programmation, il est très facilement résolvable mais qu'en est-il de remplacer chaque occurrence par un caractère différent ?
Cela complique légèrement les choses dans un autre langage de programmation mais très facilement résolvable dans des DataFrames. Dans notre cas, nous allons utiliser la méthode replace, cela va rechercher dans toutes les données du DataFrame et remplacer par un caractère ou bien une chaîne de caractère. On lui précisera le paramètre regex.
Par exemple :

```
df.replace({'b': 'p'}, regex=True) #cela remplacera les lettres b par p.
```

2.7.2 Supprimer une partie de chaîne de caractère

Ici, l'idée est d'apprendre à formater très rapidement les données. Si nous souhaitons supprimer une partie d'une chaine de caractère qui ne nous intéresse pas.
Par exemple, si nous disposons d'une suite d'adresse et que nous souhaitons supprimer une virgule ou un point virgule, nous pouvons aisément le faire avec la fonction : « *map* » et *lstrip* (pour le début d'une chaine), *rstrip* (pour la fin d'une chaîne). Nous choisissons simplement la colonne (cols) à étudier et nous appliquons la fonction, par exemple :

```
df['result'] = df['result'].map(lambda x: x.lstrip('+-').rstrip('aAbBcC'))
#on supprimera ainsi toutes les occurrences au début + - et aAbBcC.
```

2.7.3 Appliquer des fonctions à nos données

Il est souvent très utile de faire des opérations sur les données, avec la fonction applymap, rien de plus simple.
Nous rappelons la syntaxe pour définir une fonction :

```
Code Python
def puissancecarree(x)
return x * x
```

Maintenant, pour l'appliquer à notre dataframe, il suffit simplement d'utiliser la fonction *applymap* :

```
df.applymap(puissancecarre(df));
```

2.7.4 Copier dans un fichier .csv (Executable par Excel)

Nous avons appris à traiter des dataframes avec Pandas, maintenant il est utile de savoir comment les importer dans un fichier Excel.
Nous utilisons simplement la fonction $df.to\_csv(`nomDuFichier.csv')$ ou bien $df.to\_excel(`nomDuFichier.xlsx')$ pour un fichier Excel

2.8 Exercice final

L'exercice final permet de vérifier si nous avons assimiler les bases de Pandas
1- Télécharger un dataset disponible sur Kaggle
2- Transformer ce fichier en DataFrame Pandas
3- Ajouter une colonne qui s'appelle : « test »
4- Appliquer une puissance n à la colonne numéro 2
5- Transposer dans un fichier .csv

2.9 Conclusion

Nous savons maintenant comment manipuler cet outil qui est primordial dans les sciences des données en Python.
Afin de découvrir la bibliothèque Scikit-learn, je propose maintenant de manipuler un autre exemple de dataset disponible sur Kaggle : "la détection de fraude de Carte Bancaire".

3 La prédiction autour d'un exemple

Afin d'avoir un second exemple d'analyse, traitement et prédiction de données, j'ai choisi le dataset **Credit Card Fraud Detection**, une batterie de données qui recense plus de 250 000 transactions frauduleuses ou non.
Lien Kaggle du dataset : `https://www.kaggle.com/dalpozz/creditcardfraud`
Je propose aussi une analyse que j'ai réalisé en étant à ce stade de la recherche.
Lien GitHub de mon analyse : `https://github.com/davzer27/datascience/blob/master/creditcard.ipynb`

3.1 Introduction

Ce dataset recense exactement 284 807 transactions qui se sont déroulées en deux jours en Europe.
Il ne comporte pas les numéros de carte bleu de chaque transaction, mais 28 champs (V1, .. V28) qui résultent d'une transformation en Analyse en Composantes Principales (PCA en anglais) qui a permis de transformer par une méthode statistique les variables corrélées en ces 28 variables décorrélées les une des autres. Cela permet de réduire le nombre de variable en ne gardant que les plus pertinentes.
Trois autres champs sont intéressants, Time qui stipulent l'intervalle de temps en seconde qui s'est écoulé entre deux transactions, Amount qui est le montant de la transaction et Class qui indique s'il y a eu une fraude ou non (1 si fraude, 0 sinon).

3.2 Algorithme de traitement et prédiction

Exécuter le fichier : creditcard.ipnyb à l'aide de Jupyter.
Attention : le fichier dataset creditcard.csv doit se trouver dans le même dossier !

3.3 Analyse et explications

3.3.1 Classification des données

La première étape est de classifier les données sous forme de DataFrame. Pour se faire, il suffit d'utiliser la commande : « $pd\_read\_csv('fichier.csv')$ », cela va permettre d'aller directement chercher le fichier *.csv* et de l'afficher sous forme de Dataframe (la bibliothèque Pandas s'occupe de le faire !), on pourra ainsi afficher les 40 premières données avec la fonction $.head(x)$ où x

est le nombre des premières données que l'on souhaite afficher. Les colonnes où « Class » vaut 1 sont les fraudes avérées.

3.3.2 Quelques statistiques simples

La fonction $.info$ nous donne des informations sur les données à analyser, nous disposons de 284801 données avec 31 colonnes représentant $69,5$ MB. De plus avec la fonction $value\_counts$, il est très facile de déterminer le nombre de données où la fraude est avérée : 492 fraudes (0,17% des fraudes sur un échantillon de 284 801 transactions).
Le total des transactions se chiffrent à 25162590 euros (à l'aide de la fonction sum()), ce qui représente environ 88 euros par personne.

3.3.3 Analyse dans le but de supprimer des données

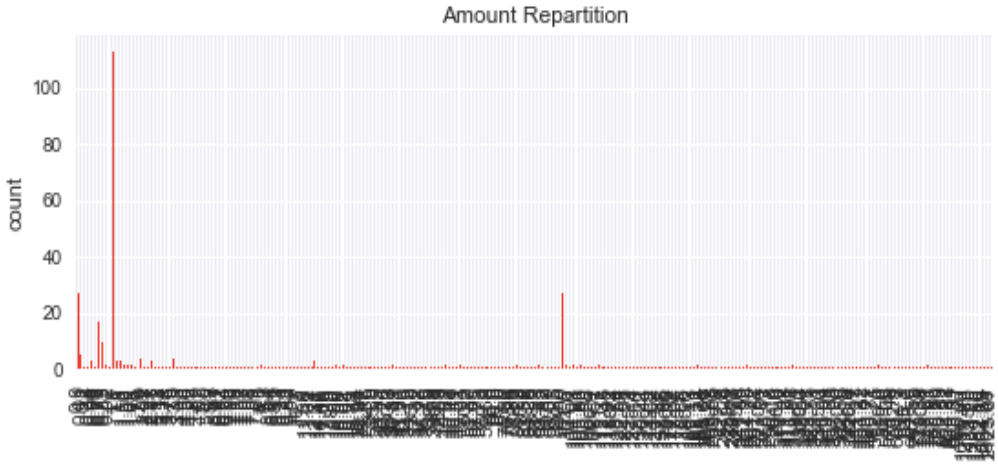
Les colonnes V_i résultent d'une transformation en Analyse en composantes principales (PCA), ainsi il est très difficile de les exploiter, nous avons donc décidé de les supprimer avec la fonction drop.
On se retrouve maintenant avec un dataframe composé de trois colonnes : Amount (Montant), Time (Temps) et Class (Fraude ou non)

3.4 Corrélation avec la fraude

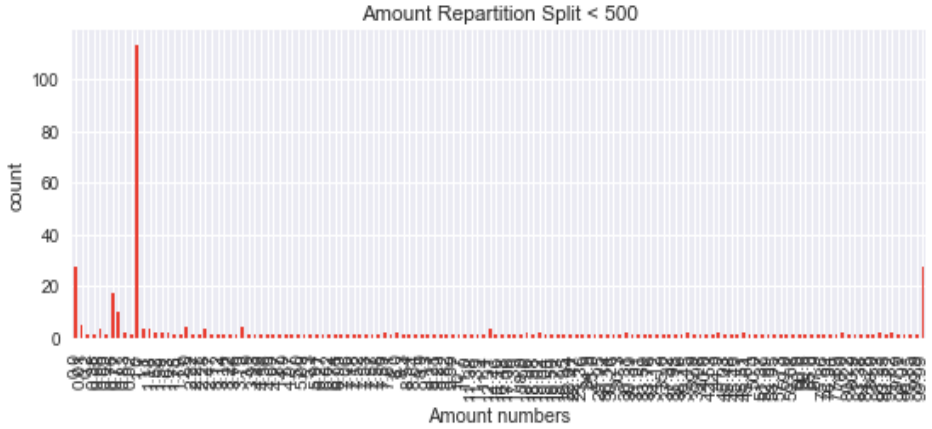
En éliminant les données comportants les V_i, nous nous retrouvons avec le montant et le temps, existe t-il un lien entre la fraude, le montant ou le temps ?

3.4.1 Corrélation avec le montant

On utilise la fonction plot en traçant tous les montants avec le nombre de fraude associé.



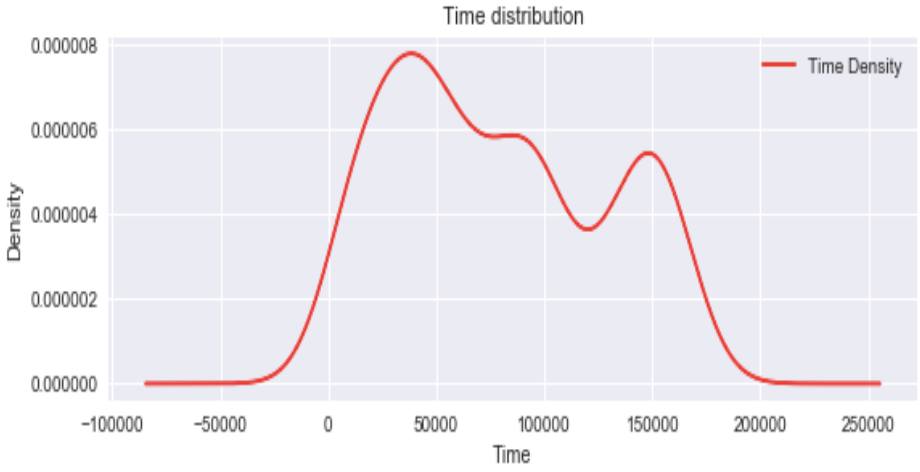
Mais du fait d'un trop grand nombre de montant différent, il a fallu supprimer les montants qui ne nous intéressaient pas (les transactions non frauduleuses !).



Ainsi les montants où la fraude a été avérée sont les montants compris entre $[0, 90]$ euros.

3.4.2 Corrélation avec le temps

En utilisant la fonction plot, nous obtenons ce graphique :



Cela montre clairement que la plus forte densité des fraudes avérées se situent autour d'un intervalle compris entre $[10000s, 50000s]$

3.5 Création d'un échantillon test

Pour faire des prédictions, il est nécessaire d'établir un échantillon test. Ainsi, nous allons prélever 600 valeurs « intéressantes » (où il y a des fraudes), nous avons donc prélevé les 600 valeurs où le temps était supérieur à 93800 s.
La création d'un fichier test devient ainsi réalisable avec la fonction : *.to_csv*.

3.6 Prédiction

La bibliothèque Scikit-Learn fournit des algorithmes de prédiction, cela permet de prédire une colonne à partir de plusieurs autres colonnes.
Les algorithmes principaux sont :

- Forêt aléatoire (Random forest) : Il est basé sur la technique des arbres de décision (chacune des décisions constituent une feuille de l'arbre et on parcourt l'arbre au fil des branches en fonction des décisions à chaque étape).
- Régression logistique (Logistic Regression), utilisé pour prédire des valeurs booléennes 0 ou 1). Elle est la plus simple, elle implique que les futurs variables aléatoires : $X_1, ...X_n \sim f(X))$, f est appelée la fonction Sigmoid, elle a pour forme :
  $f(x) = \frac{1}{1+\exp(x)}$
- SVM (Support Vector Machine) : Ce modèle parcourt toutes les données à la recherche d'une ligne qui sépare le mieux deux classes (co-

lonnes). Les instances de données qui sont les plus proches de cette ligne sont appelées **vecteurs de support**.

— ...

Chacun de ces algorithmes possède ses spécificités qui conduisent à une pertinence différentes sur une même batterie de données. Il n'existe donc pas un "algorithme parfait" qui fonctionne tout le temps.

3.6.1 Le fonctionnement

Ici, je vais illustrer le fonctionnement de la prédiction à l'aide de l'exemple de la fraude à carte de crédit.
Nous disposions d'un jeu de données de plus de 230 000 lignes. La prévision est donc à effectuer sur la colonne Class (Fraude ou pas), en effet nous devons nous demander si pour un montant et un temps donné, est-ce qu'il y aura une fraude ? Pour répondre à cette question, il faut chercher un estimateur (i.e un modèle "connu") qui s'adapte à nos données.
C'est à ce moment que l'on applique les algorithmes préétablis dans la bibliothèque Scikit-learn (LR, RF, SVM). Cependant, afin d'assurer une pertinence maximale des données et un modèle le plus correct possible, il est donc impératif d'établir l'estimateur sur un dataset le plus complet possible pour en assurer une pertinence maximale.
Exemple avec LR :

```
Code Python :
X = df_card.iloc[:,[0,29]]
y = df_card[df_card.columns[30]]
param_grid = {
              'C': [0.01, 0.05, 0.1, 0.5, 1],
              'penalty': ['l1', 'l2']
             }
estimator = linear_model.LogisticRegression()
lr_gs = grid_search.GridSearchCV(estimator, param_grid, cv=4)
lr_gs.fit(X, y)
lr_estimator = lr_gs.best_estimator_
```

La fonction **GridSearch** permet d'élaborer les meilleurs paramètres corectives pour le modèle en fonction de différentes valeurs.
Ensuite, l'appel à la fonction "fit" va permettre à l'algorithme "**d'apprendre**" sur X et y (y est la colonne qu'on voudra prédire donc la fraude et X comporte les données Temps et Montant dans un échantillon les plus complets possible 284 807 entrées).
L'algorithme nous retournera alors un **estimateur** en fonction des valeurs apprises. Celles-ci sont établies à partir d'une série de test que l'algorithme réalise.

Les données qu'on lui rentre sera X (le temps et le montant) et Y (fraude ou pas ?), c'est justement sur Y que la prédiction sera faite pour répondre à la question : "est-ce dans un futur proche avec tel ou tel temps ou montant, une fraude sera possible ?".

3.6.2 Simulation de la prédiction

Disons que maintenant nous disposons du "bon" estimateur qui modélise au mieux notre modèle (nous avons fais le bon choix entre LR, RF, SVM). Nous disposions d'un jeu de données de plus de 230 000 données, nous allons donc établir un échantillon test (disons 600 lignes) où la fraude est la plus présente (nous allons nous focaliser sur un échantillon où le temps est supérieur à 93800 s, d'après l'observation de la densité de fraude en fonction du temps). Cette échantillon test permettra de tester notre prédiction (et même de savoir si elle était bonne et à quelle fréquence !).
Après traitement, échantillonnage et retrait des colonnes comportant les V_i, nous nous retrouvons donc avec un tableau qui représente un échantillon prélevé :

Time	Amount	Class
93800	31.76	0

... 599 autres lignes.
Considérons maintenant pour le même tableau (ou dataframe), la suppression de la colonne "Class" que nous allons prédire !
Nous nous retrouvons avec un dataframe :

Time	Amount
93800	31.76

... 599 autres lignes.
Maintenant, il suffit d'essayer si pour ces données, l'estimateur établit précédemment est correct.
On prédit la colonne Class pour les données test avec le modèle établi à l'aide de :

```
lr_estimator.predict(X_test);
```

Nous obtenons finalement la colonne "Class" prédit sous cette forme

Class
0
0

... 598 autres lignes.

3.6.3 Pertinence

Nous nous retrouvons donc avec un modèle établi testé sur un échantillon de nos données dont on connaît déjà les réponses, une des questions

intéressantes à se poser est donc : "Quelle est la pertinence de notre prédiction" ?
Il est très facile d'y répondre, il suffit simplement de comparer la colonne Class prédit avec la colonne Class "vrai" (celle qui constitue notre dataset) et d'utiliser une fonction dans les metrics Scikit-Learn.
On importe la bibliothèque correspondante :

```
from sklearn.metrics import accuracy_score
```

Puis en utilisant la fonction $accuracy\_score(y\_true, y\_test\_lr)$, on obtient le pourcentage de pertinence, avec la carte de crédit, le modèle de la régression linéaire semble très pertinent car nous obtenons 95% de pertinence.
En effet, il s'agit juste d'une comparaison de la prédiction SUR les mêmes données de Class et ce qui a vraiment eu lieu.
Les algorithmes de la bibliothèque Scikit-Learn permettent aussi de renvoyer le pourcentage de valeurs exactes déterminées à l'aide de l'estimateur donnée.
Il suffit de rajouter dans les paramètres : $oob\_score = True$

```
lr_gs = grid_search.GridSearchCV(estimator, param_grid, cv=4, oob_score = True)
```

Puis taper la ligne :

```
lr_gs.oob_score_
```

La principale préoccupation du "Data Scientist" est donc d'arriver constamment à augmenter ce pourcentage pour tendre vers un modèle le plus exact possible.

Le troisième exemple se focalise exclusivement sur la prédiction, nous allons utiliser les données fournies par un équipement unique à l'Ecole Centrale de Lille : le Realcat, une plateforme de criblage catalytique.

4 Utilisation des données du Realcat

4.1 Préambule

E. Heuson a élaboré une thèse sur les enzymes, ainsi il a pu collecter des milliers de données sur les différentes spécificités de celles-ci à travers plusieurs tests réalisés par des robots.
Afin de comprendre les données que l'on va analyser et prédire, il est nécessaire d'assimiler des notions de bases les concernant.
Ces données représentent une multitude d'enzymes auquels nous avons effectués des tests à l'aide de différents substrats.
Afin de comprendre les termes employés, nous allons réaliser une courte introduction sur les enzymes et les tests réalisés.

4.1.1 Les enzymes

Une enzyme est une protéine dotée de propriétés catalytiques. Celle-ci permet de transformer une molécule chimique par une autre molécule chimique.
Cela suit la réaction suivante : $(Reactif) \overset{Enzyme}{\longrightarrow} (Produit)$. Pour former une enzyme, on part d'un plasmide (molécule d'ADN) numérotés de $[A; n]$ où $n \in R$, celui-ci contient le gêne à traduire et par l'action d'une bactérie, on obtient l'enzyme.
Cela peut se traduire de cette manière : $Plasmide \overset{Bacterie}{\longrightarrow} Enzyme.$
L'idée des expériences est de déterminer l'enzyme formée à partir de différents plasmide et de connaître l'activité de celle-ci (i.e la vitesse qu'elle met pour fabriquer le produit).
Cette activité est mesurée en $UI = \mu mol/min$
Pour connaître cette vitesse, l'idée est d'utiliser différents substrats et d'observer l'action des enzymes sur ceux-là.
Un substrat est une molécule utilisée comme réactif.
Différents tests ont par conséquent été réalisés, nous allons en choisir un pour constituer un dataset "intéressant".

4.1.2 Réalisation des tests

Un des tests réalisé a permis de déterminer si un substrat est accepté ou non. S'il est accepté par l'enzyme, cela nous fournira un produit conformémant à la reaction précédente.

4.2 Le dataset

Les données accumulées par la plateforme de test Realcat étant conséquentes, j'ai décidé de me focaliser sur un exemple simple : l'accept ou le rejet d'un substrat à travers une enzyme.
Ainsi, l'idée est de vérifier si les techniques du machine learning appliquées à un échantillon permettent d'aboutir à des résultats semblables. Ici, il ne s'agit pas d'utiliser du Big Data au vu du nombre de données que nous possédons (inférieur à 500 entrées).

— Dataset du Realcat : `https://github.com/davzer27/datascience/blob/master/realcat_data.xlsx`
— Code réalisé : `https://github.com/davzer27/datascience/blob/master/realcat.ipynb`

4.3 Prétraitement des données

Nous pouvons observer plusieurs données, si la valeur n'existe pas, le substrat n'est pas accepté. Plus la valeur est importante, plus le substrat sera potentiellement accepté.
Ainsi, pour appliquer notre prédiction, il a fallu remplacer les valeurs "NaN" par des valeurs égales à 0 grâce à la fonction "fillna", elle s'utilise de cette manière :

```
df = df.fillna(value=0);
```

Afin d'établir un modèle prédictive, il n'est plus utile de prélever un échantillon des datas, ici nous disposons d'une quantité très limitée de données, prélever un échantillon fausserait d'avantage le modèle.
Nous pouvons donc appliquer les trois algorithmes de prédictions (SVM, LR et RF) et observer lequel donne la meilleure pertinence.

4.4 Prédiction

Il est désormais possible de réaliser de la prédiction sur les données, je vais maintenant illustrer la manière de la réaliser.

```
import pandas as pd
import numpy as np
from sklearn import grid_search, metrics, linear_model, svm, ensemble
from sklearn.ensemble import RandomForestRegressor
```

On importe d'abord toutes les bibliothèques que l'on va utiliser, pandas pour les dataframes, scikit-learn pour les prédictions.

```
df = pd.read_excel('realcat_data.xlsx')
```

Ensuite, nous plaçons simplement toutes les données dans un dataframe.
À noter : ici c'est la fonction a changé car on charge un fichier Excel.
Nous devons nous poser la question : quel substrat est-il intéressant de prédire ?

```
df.info()
```

Cette fonction nous donnera les valeurs non nul, l'enzyme **2OPAA.1** comporte 152 valeurs non nulles, c'est celui-ci qui va nous intéresser le plus.
Préparons nos données pour la prédiction :

```
X_train = df.drop('2OPAA.1', axis=1)
Y_train = df['2OPAA.1']
```

La dataframe Y_train comporte donc la colonne 2OPAA.1 et X_train toutes les autres colonnes.
Il faut se poser la question : quel modèle utilise t-on ? Random Forest ? Logistic Regression ? SVM ?
Ici la régression logistique ne fonctionnera pas, car celle-ci prédit des valeurs booléennes (0 ou 1), cependant nous aurions pu le faire en remplaçant les valeurs supérieur à 0 par 1 mais nous allons perdre de l'information (intensité de la couleur jaune).

4.4.1 Prédiction 1 : Random Forest

```
forest = RandomForestRegressor(criterion='mse', max_depth=None,
                               min_samples_split=2, min_samples_leaf=1,
                               max_features='auto',
                               max_leaf_nodes=None,
                             min_impurity_split=1e-07,
                             min_weight_fraction_leaf=0.0,
                             n_estimators=251, n_jobs=1,
                               bootstrap=True, oob_score=True)
forest = forest.fit(X_train[0:250], Y_train[0:250])
y_chap = forest.predict(X_train[251:452])   #valeurs qu'on predit
print(forest.oob_score_)
```

Quelques explications : Les paramètres du forêt ont été trouvés grâce à l'emploi d'une fonction : GridSearch que je détaillerai plus tard.
D'abord nous devons stipuler à l'algorithme son seuil d'apprentissage, ici j'ai fixé (fit) 251 valeurs, les 251 premières (plus celui-ci augmente plus le score

(ou la pertinence) sera meilleur), l'algorithme va apprendre sur ces valeurs. Ensuite, l'algorithme du forêt aléatoire va simuler les prochaines valeurs $\hat{y}$: les 201 valeurs suivantes.
Finalement, l'algorithme nous retourne sa pertinence : "son score", c'est-à-dire le pourcentage de "bonnes réponses" qu'il a pu trouver. Nous avons pu trouver des pertinences oscillant entre 79 et 84%.

4.4.2 Prédiction 2 : SVM

```
svr = svm.SVR()
svr = forest.fit(X_train[0:250], Y_train[0:250])
y_chap = svr.predict(X_train[251:452])
print(svr.oob_score_)
```

Le principe est exactement le même, nous trouvons une pertinence d'environ 83%

4.4.3 Trouver les paramètres de la prédiction

On se souvient du code :

```
criterion='mse', max_depth=None,
                              min_samples_split=2, min_samples_leaf=1,
                              max_features='auto',
                              max_leaf_nodes=None,
                            min_impurity_split=1e-07,
                            min_weight_fraction_leaf=0.0,
                            n_estimators=251, n_jobs=1,
                              bootstrap=True, oob_score=True
```

Il existe une fonction dans la bibliothèque Scikit-learn permettant de déterminer "les meilleurs paramètres" de chacun des algorithmes pour la prédiction. Celle-ci s'emploie de cette manière :

```
param_grid = {
              'n_estimators': [250],
              'min_samples_leaf': [1, 3, 5]
             }
estimator = ensemble.RandomForestRegressor()
rf_gs = grid_search.GridSearchCV(estimator, param_grid, cv=4)
```

sans oublier d'importer les bibliothèques correspondantes :

```
from sklearn import grid_search, ensemble
```

4.5 Conclusion sur la prédiction appliquée aux données REALCAT

Que ce soit avec l'algorithme SVM ou le Random Forest, nous obtenons une pertinence d'environ 83%.
De plus, nous pouvons avoir accès aux échantillons faussement prédits en tapant la commande :

```
Y_train[251:452]==y_chap
```

Le machine learning a donc un très bon champ d'action même avec une batterie de données relativement limitée.

Conclusion

Durant dix semaines, j'ai appris les différentes techniques d'analyse et prédiction d'une grande quantité de données et j'ai tenté de transmettre mes connaissances à des élèves sortant de classes préparatoires à travers ces tutoriels.
Ce rapport constitue un premier avant goût de ce que le métier du "Data Scientist" consiste. Le lecteur est ainsi désormais capable de réaliser un travail de traitement et de prédiction sans réellement comprendre la complexité des algorithmes mis au point par des ingénieurs et recensés dans la bibliothèque Scikit-learn.
À travers le nombre de données transitant dans le cloud et augmentant de manière exponentielle, l'utilisation du Big Data a énormément de potentiel. Celui-ci permet de crée des intelligences artificielles de plus en plus plus pertinentes, qui apprenent et réfléchissent comme l'homme. Cependant, même si l'évolution et l'apprentissage sont des critères de l'intelligence artificielle, la capacité de réflexion propre à l'homme n'est pas transmissible.
Nous avons vu dans ce rapport des tutoriels et des applications dans le thème du machine learning.
Nous nous sommes focalisés sur des données "compréhensibles par l'humain" (des chiffres ayant une signification), mais depuis plusieurs années, une nouvelle discipline sous jacente du machine learning se développe plus en plus : le deep learning (apprentissage profond), il s'agit de travailler avec des données à haut niveaux d'abstraction (reconnaissance d'image, séquencage ADN...).

L'étude du deep learning serait donc un exemple de projet recherche qui pourrait poursuivre ce premier avant goût des applications du Big Data.

Références

[1] Jason Brownlee *Machine Learning Mastery with Python : Understand your data, create accurate models and work projects end-to-end*. 2016.

[2] Datacamp *Learn R, Pythn and Data Science online*. `http://www.datacamp.com`