

# Stage Orange

Garance Malnoë

Mai 2023

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Livre : Machine Learning. Implémentation en Python avec Scikit-learn</b>	<b>3</b>
2.1	Une vue d'ensemble . . . . .	3
2.1.1	Les formes d'apprentissage du Machine Learning . . . . .	4
2.1.2	La méthodologie CRISP-DM . . . . .	5
2.2	Chargement et analyse des données . . . . .	6
2.2.1	Définir la "carte d'identité des datasets . . . . .	6
2.2.2	La description des champs choisis . . . . .	6
2.2.3	L'analyse statistique sur chaque champ et analyse des données. . . . .	6
2.3	La préparation des données . . . . .	6
2.3.1	Limitation des données . . . . .	6
2.3.2	Préparer les données numériques . . . . .	7
2.3.3	Dataset d'entraînement et dataset d'évaluation . . . . .	8
2.4	La préparation des catégories . . . . .	9
2.4.1	Modification des catégories . . . . .	9
2.4.2	Les données particulières . . . . .	10
2.4.3	Automatisation de la préparation . . . . .	11
2.5	Étapes de modélisation . . . . .	11
2.5.1	Création d'un ensemble de données de validation . . . . .	12
2.5.2	Préparation des datasets . . . . .	12
2.5.3	Création des modèles . . . . .	12
2.5.4	Amélioration des modèles (fine-tuning) . . . . .	13
2.6	Algorithmes d'apprentissage supervisé : Classification . . . . .	14
2.6.1	Mesurer l'erreur d'une classification . . . . .	15
2.6.2	Arbre de décision et algorithmes dérivés . . . . .	18
2.6.3	K-nearest neighbors . . . . .	22
2.6.4	Régression logistique . . . . .	23
2.6.5	Naïve Bayes . . . . .	23
2.6.6	SVM : Support Vector Machine . . . . .	25

2.7	Algorithmes d'apprentissage supervisé : Régression . . . . .	25
2.7.1	La tâche de régression . . . . .	25
2.7.2	Entraînement et évaluation des modèles . . . . .	26
2.7.3	Utilisation des algorithmes de classification . . . . .	28
2.7.4	Régression linéaire et variantes . . . . .	29
2.7.5	Régression polynomiale . . . . .	30
2.7.6	Cas particulier de la prédiction . . . . .	30
2.8	Algorithmes d'apprentissage non supervisés . . . . .	32
2.8.1	Le clustering . . . . .	32
2.8.2	Réduction des dimensions . . . . .	35
2.8.3	Systèmes de recommandation . . . . .	38
2.8.4	Association . . . . .	39
2.9	Évaluation et déploiement . . . . .	40
2.9.1	Phase d'évaluation . . . . .	40
2.9.2	Phase de déploiement . . . . .	41
<b>3</b>	<b>Analyse de texte avec Python</b>	<b>41</b>
3.1	Références . . . . .	41
3.2	Introduction . . . . .	42
3.3	Préparation du sujet, problématisation et récupération des données	42
3.4	Préparation des données . . . . .	43
3.5	Analyse exploratoire des données . . . . .	45
3.5.1	Les mots les plus communs . . . . .	45
3.5.2	Nombre de mots . . . . .	48
3.5.3	Le nombre de grossiertés . . . . .	50
3.6	Techniques d'analyse de texte . . . . .	55
3.6.1	Analyse de sentiments . . . . .	55
3.6.2	Modélisation des sujets . . . . .	59
3.6.3	Génération de texte . . . . .	61
<b>4</b>	<b>Introduction au langage SQL : Structured Query Language</b>	<b>64</b>
4.1	Selectionner de données . . . . .	64
4.2	Modifier les données . . . . .	66

# 1 Introduction

Ce fichier a pour objectif de rescencer toutes les ressources utiles au bon déroulement du stage effectué chez Orange de Mai 2023 à Aout 2023. Il comprend notamment l'ensemble des notes prises sur le livre d'introduction au Machine Learning sur la bibliothèque Scikit-learn, une introduction à la bibliothèque Pandas et un peu de SQL.

## 2 Livre : Machine Learning. Implémentation en Python avec Scikit-learn

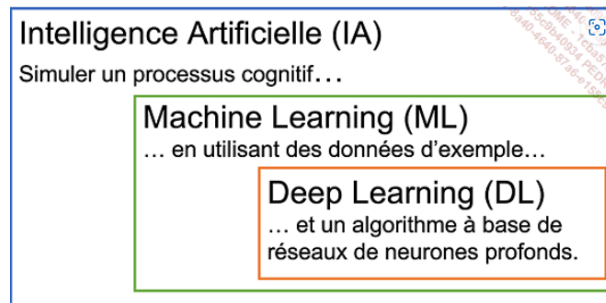
### 2.1 Une vue d'ensemble

Commençons par revenir sur le vocabulaire de la science des données et de l'intelligence artificielle, il existe plusieurs termes qui sont utilisés indifféremment alors qu'ils ne désignent pas tout à fait la même chose : l'intelligence artificielle, le data mining, le machine learning, le deep learning et la data science.

L'intelligence artificielle est un domaine scientifique qui a pour objectif de créer une machine intelligente qui pourrait résoudre tout problème qui se présenterait à elle et plus simplement de simuler un processus cognitif et de donner une impression d'intelligence. On différencie l'intelligence artificielle forte/générale où la machine aurait des émotions et pourrait avoir un raisonnement poussé mais qui n'existe pas et n'existera peut-être jamais et l'intelligence artificielle faible/étroite qui répond à un seul problème par la simulation d'un raisonnement et qui existe déjà.

Le Data Mining, maintenant souvent appelé Machine Learning, est un domaine issu des statistiques où l'on étudie de très grandes quantités de données pour obtenir de nouvelles connaissances. Il n'y a pas vraiment d'intelligence, il s'agit de trouver des motifs. Au fil des années les techniques de Machine Learning ont été incorporées à la notion d'intelligence artificielle.

Le Deep learning est une sous catégorie du Machine Learning qui consiste à créer des algorithmes très "profonds" des réseaux de neurones ou des arbres de décision avec plutôt 500 couches que 4 ou 5.



Enfin, la Data Science consiste à trouver des faits intéressants dans les données et à mettre en oeuvre toutes les étapes pour rendre utilisables ces données : préparation des données, statistiques descriptives, création des modèles, visualisation, création d'un tableau de bord. La data science ne fait pas partie du domaine de l'intelligence mais en est proche.

### 2.1.1 Les formes d'apprentissage du Machine Learning

Le Machine Learning consiste à créer un modèle qui résultera de l'apprentissage sur des données d'entraînement. Le modèle n'est pas codé par un développeur mais créé par un algorithme d'apprentissage. Il existe plusieurs types d'apprentissage et plusieurs modèles pour chaque type.

#### Apprentissage supervisé

C'est le type d'apprentissage le plus couramment utilisé. Chaque donnée est faite de plusieurs variables et d'une variable label, l'objectif va être de créer un modèle pour faire le lien entre les variables (explicatives) et la variable label.

Il existe plusieurs types de modèles : classification, régression, prévision...

#### Apprentissage non supervisé

Dans ce type d'apprentissage on fournit les données sans labélisation et on souhaite que l'algorithme nous renvoie une image globale des données, les groupes et les liens qu'il existe entre les données.

Il existe également plusieurs types de modèles : clustering, recommandations, associations etc...

#### Apprentissage par renforcement

Là aucune donnée d'apprentissage n'est fournie à l'algorithme. Le modèle va apprendre à partir d'évaluations positives ou négatives sur son comportement. Par exemple si l'on souhaite une IA qui marche : on lui donne des bons points si elle alterne les pieds et qu'elle se rapproche d'une cible mais on lui donne des mauvais points si sa tête touche le sol. Après beaucoup d'itérations, on pourra obtenir une IA qui reproduit le déplacement d'un humain. Voici plusieurs ex-

emples de vidéos qui utilisent l'apprentissage par renforcement : [ici](#) et [ici](#).

### Apprentissage semi-supervisé

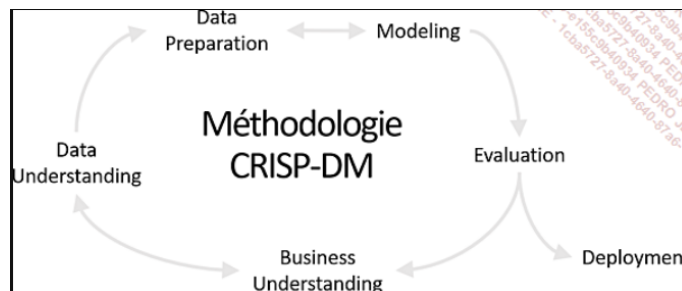
Comme son nom le laisse pensé c'est un mélange d'apprentissage non-supervisé et d'apprentissage supervisé. On l'utilise pour labéliser de gros data-set qui demanderaient trop de temps à labéliser manuellement. Le but est de labéliser quelques données puis d'utiliser le machine learning pour la labélisation. Cette technique peut permettre de n'avoir à labéliser que 10 à 50% des données du dataset.

#### 2.1.2 La méthodologie CRISP-DM

CRISP-DM est l'acronyme de Cross Industry Standard Process for Data Mining. Il s'agit d'un processus standard pour faire du Machine Learning, il est itératif : on le refait autant de fois que nécessaire jusqu'à ce que le projet convienne ou soit abandonné.

Ce processus consiste en 5 phases :

- Business Understanding : Comprendre la demande du client.
- Data Understanding : Comprendre les données, faire des statistiques descriptives.
- Data Preparation : Transformer les données pour pouvoir les utiliser correctement ensuite.
- Modelling : Création de modèles et optimisation des paramètres d'apprentissage.
- Evaluation : Les experts du sujet / le client évaluent si le modèle proposé convient.
- La toute dernière phase qui est en dehors de la boucle d'itération. C'est la mise en production de la solution.



## 2.2 Chargement et analyse des données

### 2.2.1 Définir la "carte d'identité des datasets

- Nom
- Provenance
- Taille
- Formatage
- Description des données

### 2.2.2 La description des champs choisis

- Nom
- Type
- Format
- Description
- Unité
- Absence de données
- Limites potentielles

### 2.2.3 L'analyse statistique sur chaque champ et analyse des données.

Pour chaque variable du data set, il faut produire une analyse statistique sous forme textuelle ou tabulaire et une visualisation des données. On pourra calculer les mesures suivantes : moyenne, médiane, minimum, maximum écart-type, variance et les quartiles + histogramme/boite-moustache

---

```
mean,median,min,max,std,var,  
quantile(alpha) = object[column].describe()  
object.hist(bins=nombre de separation,figsize=taille,by=variable)
```

---

## 2.3 La préparation des données

### 2.3.1 Limitation des données

- Remplir les données vides (bien justifier son choix car il va affecter la modélisation) (feature engineering)
- Supprimer des entrées : erreurs, trop peu de données...
- Supprimer des caractéristiques : pas de rapport avec le sujet, pas exploitable, même valeur à chaque fois, trop uniformes...

---

```

#Enlever une colonne en particulier
new_object = object.drop(columns=['nom de la colonne'])

#Ne garder que les colonnes donnees
new_object = object[['nom de la colonne a garder','classe']]

#Enlever toutes les entees qui n'ont rien pour la
caracterisitique
new_object = object.dropna(axis=0,subset=['caracteristique'])

#Exemples de restrictions
#Ne garde que les personnes de plus de 20 ans
new_object = object[object['Age'] >20]
#Ne garde que les personnes de sexe feminin
new_object = object[object['Sexe'] == 'F']

#Remplir les donnees vides s'il en manque peu :
object['column'] = object['column'].fillna(qqlc type de la
caracteristique)

```

---

### 2.3.2 Préparer les données numériques

On commence par la validation sémantique : Est-ce que les données font sens ? Par exemple un âge négatif ou supérieur à 200 ans doit être enlevé.

On fait ensuite une validation statistique. On supprime les données lorsque l'écart à la moyenne est supérieur à 3/4 fois l'écart-type et quand l'écart au premier quartile est supérieur à 1,5 fois l'écart inter quartile (q3-q1).

On peut dans certains cas procéder à une discrétisation, c'est-à-dire passer de données numériques à des catégories. Il y a plusieurs façons de procéder :

- Intervalles égaux (à la manière d'un histogramme), c'est pertinent pour les données avec une répartition uniforme.
- Répartition par quantile (à choisir q = ? en fonction des données)
- Répartition manuelle Il

---

```

import pandas as pd
object['carac_manu'] =
    pd.cut(object['carac'],bins=[0,11,18],lables=['not ok at
all','weird','mmmh'])

```

---

Enfin, on peut normaliser les données numériques. Si une variable se trouve entre 0 et 1000 alors qu'une autre est entre 0 et 10, l'influence de la première variable pourra avoir tendance à être surévaluée par l'algorithme de ML et donc

donc donner des modèles peu pertinents. La normalisation permet de ramener l'ensemble des variables numériques dans le même ordre de grandeur et d'ainsi éviter ce phénomène. Il y a plusieurs types de normalisations :

- Normalisation min-max , tout ramener entre 0 et 1 : Plusieurs avantages et inconvénients :
  - Tout est dans le même intervalle [0;1]
  - Le min et le max sont dépendants du dataset d'entraînement, il faut être sûr que ce sera pareil pour toutes les autres données que l'on lui donnera.
  - On peut se retrouver avec des valeurs extrêmes qui seront dans une tout petit intervalle.
 La formule de calcul :  $X' = \frac{X - \min}{\max - \min}$
- - Normalisation standard, normalisation autour de 0 et d'écart-type 1: → Les données se retrouvent à 95
  - Toujours un impact mais + faibles des valeurs extrêmes La formule de calcul :  $X' = \frac{X - \text{moyenne}}{\text{ecart-type}}$
- - Normalisation robuste :
  - 50 % des données sont dans l'intervalle [-0.5;0.5]
  - Ne prend pas en compte les valeurs extrêmes ni l'ensemble des données comme la normalisation standard.
  - Elle est à privilégier lorsqu'il y a des valeurs extrêmes dans le data set.
 La formule de calcul :  $X' = \frac{X - \text{mediane}}{Q3 - Q1}$

### 2.3.3 Dataset d'entraînement et dataset d'évaluation

Le plus souvent on suit le ratio 80/20 mais on peut l'adapter en fonction des besoins. Pour choisir les données :

- Séparation aléatoire : En opposition au fait de prendre les 80 premières lignes d'un tableau à 100 entrées. On choisit de manière aléatoire les 80% de données :

---

```
train_object = object.sample(frac=0.8, random = seed)
test_object = object.drop(train_object.index)
#On separe en 2 de maniere aleatoire car les donnees sont
souvent deja rangees.
#La graine n'est pas obligatoire, mais c est plus cool de
prendre 0,42 ou 1337.
```

---

- Séparation stratifiée :



---

```

train_object =
    object.groupby('carac', group_key='valeur').apply(lambda
        x: x.sample(frac=0.8, random_state=42))
test_object = object.drop(train_object.index)

```

---

## 2.4 La préparation des catégories

A nouveau il faut valider les valeurs : sémantiquement (elles ont bien une signification) et statistique (il n'y a pas une catégorie ne revienne qu'une seule fois sur 150).

### 2.4.1 Modification des catégories

- Ordonner ou réordonner

---

```

object['carac'].cat.ordered #Renvoie si la categorie est
                           consideree ordonnee ou pas.

#Pour ordonner les differentes categories.
object['carac'].cat.reorder_categories(['1carc', '2carac', '3carac'], ordered=True,
                                       inplace=True)

```

---

- Modifier la liste des catégories : Si deux catégories désignent la même chose (Monsieur et Mr) alors on modifie pour n'en avoir qu'un seul. Si une catégorie est inutilisée ou que trop peu de personne sont dans une même catégorie (apparaît 1 fois ou 2 sur 1000) proche ou les mettre tous dans une catégorie 'autre'.

---

```

object['carac'].replace('Mr', 'Monsieur', inplace=True) #On garde le
2e.
object['carac'].cat.remove_unused_categories(inplace=True) #On
enleve les inutilisees

#Exemple de déplacement avec les donnees Titanic
nb_values = titanic_df['title'].value_counts()
to_replace = nb_values[nb_values<20].index
titanic_df['title'] = titanic_df['title'].replace({x:'Other' for x
in to_replace}).astype('category')

```

---

- Transformer une donnée catégorielle en données numériques : (S/M/L → 0/1/2) cela marche bien pour les catégories ordonnées mais il faut éviter dans les cas où il n'y a pas d'ordre.
- Dummification (variables muettes) pg 132 : On crée une colonne pour chaque catégorie et on met rempli avec des 0 et des 1. On peut parfois enlever la première ou la dernière colonne (s'il y a des 0 pour toutes les

colonnes alors la dernière sera forcément un 1) mais ça n'est pas obligatoire.

---

```
import pandas as pd

#Exemple avec la base de donnees Titanic
title_df = pd.get_dummies(titanic_df['title'],prefix=t)
new_df = pd.concat([titanic_df,title_df],axis=1)
new_df.drop(columns='title',inplace=True)

#Exemple pour enlever la derniere colonne pour le sexe dans la
base de donnees Titanic
sex_df =
    pd.get_dummies(titanic_df['Sex'],prefix='Sex',drop_first=True)
new_df = pd.concat([titanic_df,sex_df],axis=1)
new_df.drop(columns='Sex',inplace=True)
```

---

#### 2.4.2 Les données particulières

Les dates : pandas permet de bien manipuler ce genre de données avec beaucoup de formats différents// Les textes ://

- Préparation en amont : harmonisation (minuscule/majuscule), suppression des espaces, remplacement de certains caractères.

---

```
#Exemple avec la base de donnees Titanic
titanic_df['Name_lower']=titanic_df['Name'].str.lower()
titanic_df['Name_lower']=titanic_df['Name_lower'].str.replace(' ','')
```

---

- Effectuer une recherche dans les chaînes :

---

```
contains #Contient une sous-chaine
count #Compte le nombre d'apparition de la sous-chaine
startswith ou endswith #Commence ou termine avec une certaine
chaine de caracteres
findall #Renvoie toutes les occurrences de la sous-chaine sur
chacun des enregistrements
match #indique si les valeurs correspondent a un pattern
find #renvoie la premiere occurrence d'une sous-chaine ou -1
index #renvoie une erreur si l'occurrence n'existe pas.
```

---

- Extraire des sous-chaînes :

---

```
slice #permet d'extraire une sous-chaine en indiquant les indices
d'ou on veut couper
extract / extractall #Permet d extraire toutes les occurrences sauf
la premiere / y compris la premiere
```

---

```
split #Separe une chaine sur un separateur fourni et renvoie un
      tableau de sous-chaines
partition #separe la chaine en 3 : avant le separateur, le
          separateur, apres le separateur
apply #Pour appliquer une fonction definie plus tot
```

---

### 2.4.3 Automatisation de la préparation

Ne pas hésiter à proposer plus préparation de données et de les garder en mémoire (git, jupyter). Il y a des préparation qui sont déjà prévue dans la bibliothèque Scikit-learn : les transformers.

- Création d'un transformer :

---

```
init : il s'agit du constructeur du Transformer qui peut contenir
      des parametres
fit : lorsque Transformer est appele sur des donnees
      d'entrainement il faut egalement appeler fit
transform : lorsque le Transformer est appele sur les donnees de
            test ou les donnees reelles c'est la methode transform qui
            sera utilisee. Celle-ci va alors utiliser les parametres
            calcules en amont par fit pour appliquer les bonnes
            transformations a la categorie.
fit_transform : fait fit et transform en meme temps

#Exemple de normalisation min-max :
init : va creer les variables min et max vides
fit : va calculer sur l'ensemble d'apprentissage les deux valeurs
      et les stocker
transform : va utiliser les min et max pour les appliquer aux
            nouvelles donnees.
```

---

- Création d'une pipeline : Cela permet d'appliquer plusieurs transformations à des données et de tout condenser dans une seule méthode plutôt que d'avoir à écrire une grosse ligne de code bien dégueu.
- Il est parfois plus intéressant de juste garder la bibliothèque Pandas pour le traitement des données et de ne passer à scikit-learn que pour la modélisation.

## 2.5 Étapes de modélisation

Il y a des règles de bonne pratique à respecter lorsque l'on modélise pour éviter de faire des erreurs. Elles sont à suivre tout au long des étapes:

- Indiquer comment sera testé le modèle ainsi que les métriques choisies pour l'évaluation
- Préciser la technique choisie et la décrire brièvement

- Indiquer quelles sont les contraintes du modèle choisi
- Préciser quels paramètres (des données) sont utilisés
- Calculer les métriques du modèle (erreur par exemple) et si possible proposer une analyse du modèle
- Indiquer si le modèle est acceptable ou non

Les étapes de la modélisation :

### **2.5.1 Création d'un ensemble de données de validation**

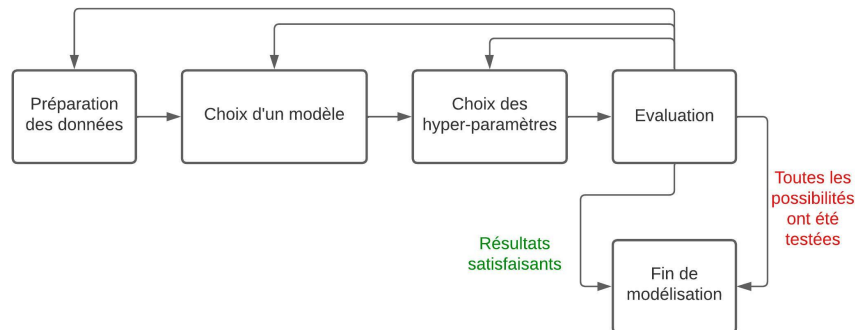
Il permet de valider le modèle et ses hyper-paramètres. Un hyper-paramètre : c'est un paramètre de l'apprentissage du modèle (ex: le pas du gradient pour la méthode du gradient descendant). On retient la méthode des K-folds qui consiste à séparer le dataset d'apprentissage en K sous-ensemble : K-1 servent à l'apprentissage du modèle et le dernier de validation. On répète K fois et on analyse.

### **2.5.2 Préparation des datasets**

On fait en fonction de la bibliothèque de modélisation que l'on va utiliser. Par exemple Scikit-learn n'accepte que les données numériques et n'accepte pas les cases vides il faut donc préparer les données dans cette optique (voir la partie précédente sur la préparation des données).

### **2.5.3 Création des modèles**

C'est un processus itératif qui suit le schéma suivant.



On va créer un modèle (appelé aussi estimator) parmi les types de modèles proposés par scikit-learn. Le mot clé "fit" permet de faire apprendre le modèle et le mot clé "predict" permet à partir de données non-labélisées d'avoir la prédiction du label par le modèle.

On évalue ensuite le modèle à l'aide de techniques qui seront décrites plus loin pour chaque type d'algorithme.

Et on sauvegarde notre modèle avec pickle ou joblib.

#### 2.5.4 Amélioration des modèles (fine-tunning)

1. Optimisation des hyper-paramètres. En théorie on souhaiterait tester toutes les combinaisons d'hyper-paramètres mais c'est rarement possible de le faire. Souvent on utilisera des techniques de grilles (on teste plusieurs combinaisons dans une grille), techniques aléatoires ou de gradient descendant. La recherche en grille et aléatoire sont proposées dans scikit-learn. Pour la première il suffit de donner la liste de valeurs d'hyper paramètres que l'on souhaite tester et pour la seconde de choisir un modèle aléatoire.

2. Sur et sous apprentissage :

- Sur-apprentissage : le modèle est très bon sur les données d'apprentissage mais donne de mauvais résultats sur les données de validation → over-fitting / sur-apprentissage
- Mauvais résultats à la fois sur les données d'apprentissages et l'ensemble de validation alors il y a eu un soucis d'apprentissage : sous-apprentissage.
- Voir cours d'Analyse Appliquées sur l'approximation polynomiale.
- Souvent sur les modèles les + simples il va y avoir un sur-apprentissage et un sous-apprentissage sur les modèles + complexes.
- Une stratégie peut-être de commencer avec un modèle simple avec peu de paramètres et s'il y a un sous-apprentissage de venir complexifier le modèle au fur et à mesure quitte à finir sur du Deep Learning avec des millions de paramètres.

- Il faut faire un compromis entre l'erreur sur l'ensemble d'apprentissage (le biais) et l'erreur entre l'ensemble de validation et l'ensemble d'apprentissage (la variance).

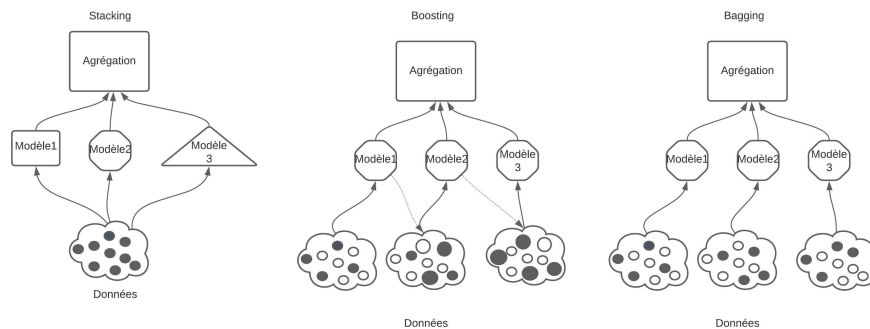
### 3. Les méthodes ensemblistes :

Il peut être intéressant de regarder plusieurs modèles avec des résultats satisfaisant et de faire leur moyennes. Dans le meilleur des cas, les erreurs vont se compenser et les prédictions se lisser.

→ Bagging : Bootstrap Aggregating. A partir de l'ensemble d'apprentissage on crée plusieurs sous data set (tirage avec remise) pour entraîner le même type de modèle puis on fait la “moyenne” des modèles.

→ Boosting : La méthode reprend l'idée du bagging de faire des sous-ensembles et plusieurs modèles mais au lieu d'avoir un tirage uniforme chaque donnée a une probabilité propre d'être tirée. Plus le/les modèles précédents se trompaient pour cette donnée plus sa probabilité augment et inversement pour renforcer le lissage. Les résultats sont souvent bien meilleurs.

→ Stacking : Au lieu d'utiliser plusieurs fois le même types de modèles le stacking va utiliser plusieurs modèles différents et les entraîner sur les mêmes données. Les prédictions des modèles seront ensuite imputer dans un méta modèle qui ressortira ensuite une prédiction finale. Cette méthode permet de pouvoir utiliser les avantages de méthodes différentes en même temps. On peut bien évidemment aller au delà de 2 couches (modèles et méta modèle) et en avoir bien plus.



## 2.6 Algorithmes d'apprentissage supervisé : Classification

La tâche de classification est l'une des deux principales en apprentissage supervisé. Son but est d'associer à chaque donnée une étiquette (ou un label) parmi un ensemble de labels possibles (ou catégories). Il y a 2 types de classification: binaire s'il n'y a que 2 possibilités (ex : A survécu au naufrage / n'a pas

survécu) ou multiclasse s'il y en a d'avantage (ex: pluie, nuageux, ensoleillé, neige...). Les catégories sont décidées avant toute forme d'apprentissage et toutes les données d'apprentissage sont labélisées ce qui permet de connaître la réponse qui est attendue. On est bien dans le cas d'un apprentissage supervisé.

Quelques exemples :


- Industrie : déterminer si une pièce, un produit ... présente un défaut et s'il faut un changement
- Site marchands : Associer directement un produit à une catégorie à partir de sa description
- Sécurité internet : Classification comme spam de mails
- Objets connectés : Déterminer si l'objet sous surveillance est dans un état normal ou pas, s'il y a un besoin de maintenance...

Il n'y a pas besoin d'une préparation particulière a priori, les classifications ne nécessitent pas forcément d'avoir des paramètres uniquement d'un certain type (catégories ou numérique) cependant pour que les modèles produits soient le plus fidèle possible il est préférable d'avoir un nombre important de données de chaque classe finale.

### 2.6.1 Mesurer l'erreur d'une classification

- Matrices de confusion : Elles permettent de connaître les prédictions faites par le modèle pour les différentes classes.

Pour la classification binaire :

	Prédiction 0	Prédiction 1	Total
Réel 0	Vrai Négatif 	Faux Positif (type I) 	xx
Réel 1	Faux Négatif (type II) 	Vrai Positif 	xx
Total	xx	xx	xx

L'erreur de type I, le faux positif (ex: on dit à un patient sain qu'il est malade). Cette erreur est en générale considérée comme moins grave.

L'erreur de type II, le faux négatif (ex: on dit à un patient malade qu'il est sain). Cette erreur est en générale considérée comme plus grave car

c'est celle qui a le plus de conséquences vraiment négatives. Pour la classification multiclass :

	Prédiction 0	Prédiction 1	Prédiction 2	Total
Réel 0	✓	✗	✗	xx
Réel 1	✗	✓	✗	xx
Réel 2	✗	✗	✓	xx
Total	xx	xx	xx	xx

- L'accuracy : Permet de connaître le pourcentage de bonnes prédictions.

$$Accuracy = \frac{VP+VN}{Total}$$

- Le rappel : Permet de compléter l'accuracy en précisant la proportion de la classe positive (ex: le nombre de malades détectés parmi les malades). Un fort rappel indique donc que presque tous les cas de la classe positive ont été détectés et que l'erreur de type 2 est faible.

$$Rappel = \frac{VP}{VP+FN}$$

- La précision : Permet également de compléter l'accuracy et d'indiquer la proportion de vrai positif parmi l'ensemble des positifs détectés (le nombre de patient réellement malades parmi ceux que le test indique comme malades). La précision permet d'estimer l'erreur de type 1.

$$Precision = \frac{VP}{VP+FP}$$

→ Un laboratoire cherchant à développer un test pour détecter une maladie cherche donc avant tout un très bon rappel et moins une bonne précision (le mieux étant d'avoir les 2).

Pour les multiclass : c'est un petit plus compliqué et soulèvent souvent des exceptions. Il y a plusieurs moyen de faire la moyenne. (Voir page 185).

- Le F1-score est un indicateur basé sur le rappel et la précision. Voici sa formule :



$$F1 - Score = \frac{2 * precision * rappel}{precision + rappel}$$

- Le Fbeta-score : Comme les erreurs de type I et de type II n'ont pas toujours la même importance, ce test permet de prendre en compte cela à l'inverse du F1-score qui considère qu'elles ont la même valeur. Beta est un réel.

$$\beta = \frac{\text{erreur type II}}{\text{erreur type I}} = \frac{\text{rappel}}{\text{precision}}$$

$$F_{\beta} = \frac{(1 + \beta^2) * \text{precision} * \text{rappel}}{(\beta^2 * \text{precision}) + \text{rappel}}$$

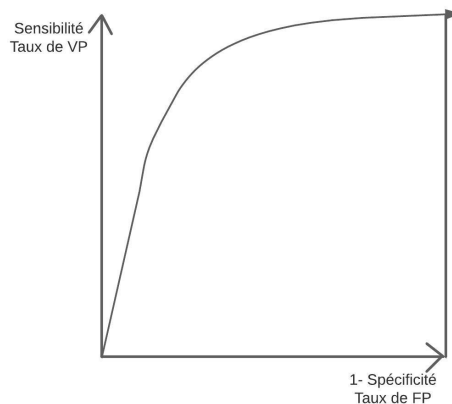
- La sensibilité : c'est la même chose que le rappel.
- La spécificité : Permet de mesurer la capacité du modèle à donner un résultat négatif pour la classe négative (le nombre de personnes saines parmi les personnes prédites comme saines).

$$Specificite = \frac{VN}{VN + FP}$$

→ Si la sensibilité ou la spécificité sont trop faible alors le résultat n'est pas fiable.

- Une sensibilité trop faible ne permet pas de décider d'une résultat positif.
- Une spécificité trop faible ne permet pas de décider d'un résultat négatif

- La courbe ROC et l'AUC. Cette courbe permet de trouver le meilleur compromis entre la spécificité et la sensibilité. Souvent on trace avec un pas de 0.05 et on relie ou alors on le fait avec un logiciel. Le meilleur compromis sera le point le plus proche du coin en haut à gauche qui donne une forte sensibilité (repère bien les malades parmi les malades) et une forte spécificité (repère bien les personnes saines parmi les personnes saines). L'aire sous la courbe ou AUC indique à quel point le test a une bonne performance. 0,5 correspond à un test aléatoire et 1 est l'objectif à atteindre.



AUC	Performance
[0.9;1]	Très bonne perf
[0.8;0.9]	Bonne perf
[0.7;0.8]	Performance moyenne
[0.5;0.7]	Performance médiocre

- Probabilités : Dans de nombreux modèles, le résultat prédit n'est pas vraiment juste une classe mais plutôt une probabilité (ex avec la base de données du Titanic : survie = x% et décès = y%). En général pour choisir la classe on prend un seuil de 50% mais on peut le modifier si on le souhaite. De plus, plutôt d'utiliser `predict` dans `scikit-learn` on peut plutôt demander les probabilités associées à chaque classe avec `predict_proba`.

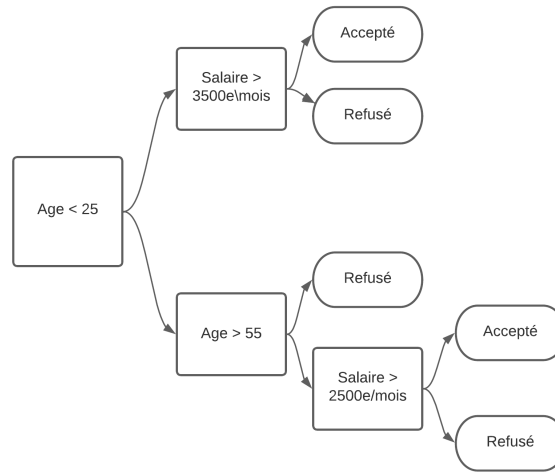
Attention : Il est très important de choisir les indicateurs avant de faire les modèles sinon on peut avoir tendance à engineer notre modèle pour certains indicateurs mais qui ne sont pas du tout pertinent pour notre sujet.

### 2.6.2 Arbre de décision et algorithmes dérivés

Site en anglais expliquant très bien le sujet avec un bon exemple

Les arbres de décisions sont parmi les algorithmes de ML les plus simples néanmoins ils peuvent être très performants. Ils sont performants avec les méthodes ensemblistes présentées plus tôt. C'est un ensemble de noeuds reliés par des arcs. Il y a plusieurs sorties possibles : juste une catégorie ou alors les probabilités associées à chaque catégorie.

Exemple pour accepter un prêt :



Le choix du point de coupure est important. On dit qu'un noeud est "pure" s'il permet de clairement dire "cette donnée est de telle classe" mais le plus souvent les noeuds sont "impurs" (plusieurs classes différentes passent la condition du noeud) et il existe plusieurs façons de mesurer cette impureté :

**L'entropie** : Le terme vient de la physique, il mesure le chaos/le désordre, ou l'impureté d'un nœud dans notre cas. Un noeud avec une composition variée (impureté) est considéré comme ayant une plus forte entropie qu'avec une composition unique (pureté). L'entropie est une mesure comprise entre 0 et 1 le but étant d'avoir des noeuds avec une entropie la plus proche de 0 possible. L'entropie est mesurée avec la formule suivante :

$$E = - \sum_{i=1}^n p_i \log_2(p_i)$$

Avec  $p_i$  la probabilité de choisir un exemple de la classe  $i$ .

Il s'agit ensuite de choisir quel catégorie regarder pour le premier noeud (root). Pour cela on calcule l'entropie pour chaque catégorie et de chacune de ses classes et on compare à la moyenne. Il s'agit du calcul du gain d'information dont la formule est la suivante :

$$\text{Gain d'information} = \text{Entropie}_{\text{parent}} - \text{Entropie}_{\text{enfants}}$$

Prenons l'exemple du site internet. On calcule l'entropie pour toutes les classes des 3 catégories : work\_status (working ou not working), background (maths, CS, other), mode (online or not online). Faisons un exemple avec work\_status :

$$\text{Entropy}_{\text{working}} = 0.9183$$

$$\text{Entropy}_{\text{notworking}} = 0.6500$$

$$\text{Entropy}_{\text{notworking}} = \text{moyenne pondérée des entropie des 2 classes} = 0.8110$$

**Gain d'information** =  $0.9183 - 0.8110 = 0.1858$

Et on répète l'opération avec les 2 autres catégories. On retiendra la catégorie avec le meilleur gain d'information.

Il est important de noter que cet algorithme est plus gourmand que le suivant.

**Gini** : cet indicateur mesure la probabilité qu'une donnée aléatoire soit mal-classée par le noeud, il vaut donc 0.5 dans le cas binaire aléatoire. L'objectif est donc d'avoir un indice de Gini le plus proche de 0 possible. Sa formule est :

$$Gini = 1 - \sum_{i=1}^j P(i)^2$$

Où  $j$  représente le nombre de classes dans la catégorie que l'on étudie dans le noeud et  $P(i)$  le ratio de la classe  $i$  sur le total d'observation.

La création des arbres de décisions suivent des algorithmes dits "greedy" : ils prennent la meilleure pureté possible à chaque étape et on précise dans le code si l'on souhaite utiliser Gini ou l'entropie pour faire notre choix.

Il est aussi important de savoir à quel moment la construction de l'arbre doit s'arrêter pour donner un modèle le plus efficace possible et surtout éviter le sur-apprentissage. Il y a plusieurs critères qui peuvent être implémentés et cumulés :

- La profondeur maximale avec (`max_depth`)
- Le nombre minimal de données d'un noeud pour le couper (`min_samples_leaf`)
- Nombre minimal de données dans une feuille (`min_weight_fraction_leaf`)
- Le nombre maximal de catégories à prendre en compte à chaque split potentiel (`max_features`)
- Le nombre maximal de feuilles (`max_leaf_nodes`)
- Le gain minimum pour qu'une coupure soit faite (`min_impurity_decrease`)
- L'impureté minimale (`min_impurity_split`)

Puisqu'il s'agit de classification, il est possible d'accéder aux probabilités de chaque feuille avec le mot clé `value` mais également à l'importance de chaque catégorie dans l'arbre de décisions (pour Iris, la longueur des pétales détermine la classe à laquelle appartient le pétale dans 93% des cas). Enfin, la dernière étape est l'exportation de l'arbre et sa visualisation. Les arbres de décision peuvent être exportés au format Graphviz (`.dot`).

---

```

from sklearn.tree import DecisionTreeClassifier, export_graphviz
# Creation d'un modele
tree_classifier = DecisionTreeClassifier(max_depth=3,
random_state=42)
tree_classifier.fit(train_X_iris, train_y_iris)
# Export en fichier dot (avec mises en forme)
class_names = ['Setosa', 'Versicolor', 'Virginica']
feature_names = ['sepal_length', 'sepal_width', 'petal_length',
'petal_width']
export_graphviz(tree_classifier, out_file='tree_iris.dot',
class_names=class_names, feature_names=feature_names,
rounded=True, filled=True)

```

---

Ce format est rarement lisible directement. L'utilitaire dot permet de convertir les images en formats plus courants comme PNG. Celui-ci s'appelle en ligne de commande uniquement. À l'intérieur de Jupyter, il suffit de préfixer l'appel par ! pour indiquer une commande système :

---

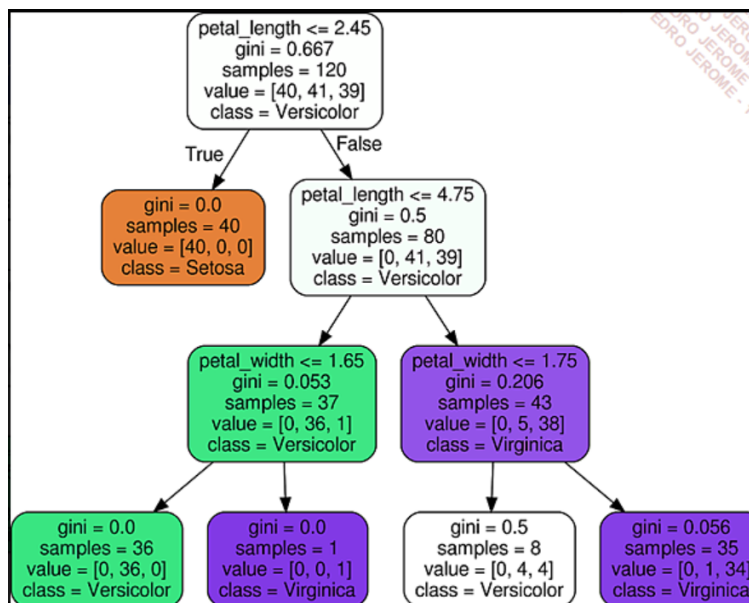
```

!dot -Tpng tree_iris.dot -o tree_iris.png -Gdpi=600
from IPython.display import Image
Image(filename = 'tree_iris.png')

```

---

On obtient l'arbre de décision suivant pour les données d'Iris.



Les forêts aléatoires (random forests), application du bagging ensembliste aux arbres de décision :

Les arbres de décisions sont parfois un peu trop simplistes pour pouvoir modéliser correctement des problèmes complexes avec de bons résultats. Les random forests sont le résultat de l'application d'une méthode ensembliste aux arbres de décisions : le bagging. Le principe est de créer de multiples arbres sur des sous-ensembles de données pour lisser les erreurs induites par un seul arbre en faisant la moyenne des prédictions. Ces arbres peuvent être créés en parallèle si la machine le permet, ce qui permet de gagner du temps.

Cependant, cette solution n'est pas magique. Tout d'abord, elle est plus complexe et donc moins rapide et surtout elle peut entraîner du sur-apprentissage dans les dataset les + simples.

**XCBoost (eXtreme Gradient Boosting)**, application du boosting ensembliste aux arbres de décision :

Cette méthode de construction du modèle repose sur les mêmes problématiques que les forêts aléatoires mais n'a pas le même fonctionnement. Cette fois les arbres sont créés les uns après les autres puisque que cette méthode utilise repose sur le boosting qui influence les données de training en fonction des erreurs des arbres précédents.

Là aussi il faut faire attention au sur-apprentissage et au compromis entre performance et complexité.

### 2.6.3 K-nearest neighbors

Cette méthode un peu particulière ne crée par de modèle à proprement parler. L'ensemble d'apprentissage constitue le modèle. Il est dit non-paramétrique.

En effet, chaque donnée est classée en fonction des K données les plus proches dans l'espace des variables. K est donc l'hyperparamètre à déterminer.

→ Le choisir trop faible favorisera le sur-apprentissage.

→ Le choisir trop grand aura tendance à flouter les frontières entre les groupes. Par convention on choisi K=5 mais on ajustera pour avoir les meilleures performances tout en évitant les nombres paires qui mènent à des égalités.

Le calcul est très sensible à l'amplitude de chaque variable. Il est donc très important pour cette méthode de normaliser les données numériques qui permet de gagner beaucoup d'accuracy.

Il est aussi important de tester les paramètres qui définissent comment est calculée la distance entre 2 points.

Cet algorithme n'est pas très performant pour les prédictions car pour les nouvelles données il va devoir calculer la distance à chaque point des données

d'apprentissage ce qui peut être très long. Il existe donc des optimisations pour séparer l'espace et donc ne calculer qu'une partie des distances, c'est le paramètre algorithme qui permet de choisir la stratégie de parcours de l'espace.

#### 2.6.4 Régression logistique

##### Régression logistique binaire

La régression logistique fonctionne en 2 temps :

- La création d'une fonction linéaire des variables, qui associe des nombres positifs aux données de la classe positive et des valeurs négatives à la classe négative.

$$c = a_0 + a_1x_1 + \dots + a_nx_n$$

- L'application de la fonction logistique sur le résultat obtenu pour ramener la sortie à une valeur entre 0 et 1.

$$y = \frac{e^c}{1+e^c}$$

La régression logistique est simple à mettre en place. L'apprentissage consiste donc à déterminer les coefficients  $a_i$  de la fonction linéaire. Il existe plusieurs algorithmes applicables.

L'avantage principal de cette méthode est qu'elle est simple à comprendre et analyser : un coefficient positif implique une corrélation positive et inversement, on obtient par une simple division le ratio entre deux variables

##### Régression logistique polytomique

La régression logistique avec plusieurs classes est dite polytomique. Pour régler ce problème on va tout simplement se ramener à  $k$  problèmes de régression binaire un pour chaque classe et appliquer le même algorithme que pour les régressions logistiques binaires à chaque fois.

A la sortie de l'algorithme, une normalisation a lieu pour que la somme des probabilités soit égale à 1.

#### 2.6.5 Naïve Bayes

La classification naïve bayésienne (ou naïve Bayes) tire son nom de ses deux principales caractéristiques :

- Elle se base sur les probabilités conditionnelles (théorème de Bayes)

- Elle fait la supposition "naïve" que les variables ne sont pas corrélées entre elles.

Cette technique est simple à mettre en place et donne de bons résultats.

Dans le cas de la classification on cherche à obtenir la probabilité :  $P(y|X)$  avec  $y$  la classe à prédire et  $X$  la donnée, elle est dite **postérieure**.

Grâce au théorème de Bayes et à l'hypothèse d'indépendance des variables entre elles, il est possible de calculer une telle probabilité à partir des probas :

- $P(y)$ , qui correspond généralement à la proportion des données d'apprentissage dans la classe  $y$ . Elle est dite **antérieure**.
- $P(X_i|y)$  La probabilité d'une caractéristique pour une classe donnée. Elle est dite '**vraisemblance**'.
- $P(X)$  la probabilité d'avoir l'entrée  $X$ . Elle est dite **évidence**.

On a la formule suivante par la formule de Bayes :

$$posterieure = \frac{vraisemblance * anterieure}{evidence}$$

Tout comme la méthode des plus proches voisins, cette méthode est dite déterministe, elle n'utilise pas d'aléatoire.

Pour les variables continues c'est un peu plus complexe donc on va utiliser des algorithmes pour nous aider à calculer l'évidence et la vraisemblance. Il existe plusieurs algorithmes de types Naïve Bayes, en fonction des suppositions faites sur les distributions des différents paramètres pour que la prédiction soit plus optimisée :

- Catégoriel : on va se ramener à des catégories comme  $[0;10]$ ,  $[10;20]$  etc.
- Gaussien : Chaque valeur est supposée suivre une loi gaussienne. Un calcul est donc fait avec la moyenne et l'écart-type pour déduire la proba d'une valeur donnée. On aura déterminé au moment de l'analyse des données si c'est applicable.
- Bernouilli : lorsque la variable est binaire. Il est alors calculé la probabilité d'avoir ou pas une variable donnée.

Attention : le choix de l'algorithme se fait sur l'ensemble des variables donc il faut choisir en amont de tout catégoriser ou de tout discrétiser.



### 2.6.6 SVM : Support Vector Machine

Les SVM servent à trouver des droites permettant de séparer deux ensembles de points. Dans les problèmes de classification linéairement séparables il y a une infinité de droites permettant de faire cela.

La marge entre la droite et les 2 points les plus proches de chaque côté est d'autant plus grande que la capacité de généralisation du modèle. Pour choisir cette droite on va essayer de faire un compromis entre la généralisation aux nouveaux points et l'erreur sur le set d'apprentissage : le paramètre  $C$  qui indique l'erreur acceptable. On le choisira en fonction du problème.

Malheureusement la majorité des problèmes ne sont pas linéairement séparables (on ne pas tracer une droite pour séparer les 2 ensembles de données) mais pour régler ce problème on peut tout simplement passer à la dimension supérieure et alors on pourra trouver un hyperplan séparant les 2 sets. Ce sont les kernels qui servent à créer de nouvelles dimensions en fonction des entrées.

Cette méthode est très avantageuse car elle permet d'avoir beaucoup de caractéristique et de quand même pourvoir augmenter le nombre de dimension. De plus, ils sont efficaces niveau calcul et sont simples à sauvegarder.

## 2.7 Algorithmes d'apprentissage supervisé : Régression

### 2.7.1 La tâche de régression

La régression est l'autre principale tâche du Machine Learning en apprentissage supervisé. La régression consiste à associer une valeur numérique (variable cible) à un ensemble de variables explicatives. Contrairement à la classification qui ne pouvait ne prédire que certaines valeurs (catégories), la variable cible d'une régression est une variable continue. De plus, dans la plupart des algorithmes de régression il n'y a pas de bornes donc l'intervalle de définition de la variable cible correspond à l'ensemble des valeurs possibles.

Commençons tout d'abord par plusieurs exemples d'utilisation de la régression :

- Prédire le prix d'un produit à partir de ses caractéristiques (la taille et la couleur d'une tomate au supermarché et son prix).
- Évaluer les risques d'un événement en fonction d'autres événements ou informations.
- Estimer la qualité d'une production en fonction de données physiques.

Pour la plupart des algorithmes, le but sera de trouver une formule reliant les variables explicatives et la variable cible. Une préparation spécifique des données

est donc nécessaire :

- Normalisation des valeurs numériques pour ne pas avoir de biais sur l'importance de telle ou telle variable.
- Transformation des variables catégorielles en variables numériques.

Comme pour les algorithmes de classification, nous n'allons pas voir tous les algorithmes existants mais seulement les plus connus.

### 2.7.2 Entraînement et évaluation des modèles

Tout d'abord intéressons nous à l'entraînement et à l'évaluation des modèles de régression.

Le processus reste similaire à celui des algorithmes de classification. On crée des ensembles d'apprentissage, de test et de validation (potentiellement validation croisée), on crée un modèle avec les paramètres souhaités, on fait l'apprentissage avec le mot-clé *fit* (données  $X + y$ ), on prédit avec le mot clé *predict* (seulement données  $X$ ) et on appelle les différentes métriques avec le mot-clé *metrics*.

#### Notion d'erreur

Le rôle des modèles de régression est de trouver une fonction mathématiques faisant le lien entre les entrées et la sortie mais il est rare de pouvoir avoir une fonction parfaite : il y a toujours une erreur. Cependant, les algorithmes vont essayer de minimiser cette erreur et de l'équilibrer.

$$\text{Erreur} = \text{valeur réelle} - \text{valeur prédite} = y - y'$$

Si l'erreur est négative alors la prédiction est au dessus de la valeur réelle. Si l'erreur est positive alors la prédiction est en-dessous de la valeur réelle. Le but de la droite est d'avoir une erreur totale à 0 : les erreurs positives compensent les erreurs négatives.

$$\sum_{i=1}^n (y_i - y'_i) = 0$$

#### Indicateurs dérivés de la mesure d'erreurs

- Erreur moyenne absolue :  $MAE = \frac{1}{N} \sum |y_i - y'_i|$

---

`sklearn.metrics.mean_absolute_error`

---

- Erreur médiane absolue : ‘

---

```
sklearn.metrics.median_absolute_error
```

---

- Erreur quadratique moyenne (mean squared error) : elle a pour avantage de renforcer l'importance des grandes erreurs.  $MSE = \frac{1}{N} \sum (y_i - y'_i)^2$

---

```
sklearn.metrics.mean_squared_error
```

---

- Racine de l'erreur quadratique moyenne (permet de ramener l'erreur dans l'unité de la variable cible, RMSE).

---

```
np.sqrt(sklearn.metrics.mean_squared_error(test_y, pred_y))
```

---

- Coefficient de détermination et variance expliquée  
Cet indicateur utilise la potentielle corrélation entre les variables explicatives et la variable cible. La variance de la variable cible est "expliquée" par la variance des variables explicatives. Le coefficient de détermination est le carré de la corrélation linéaire (r de Bravais Pearson) et la variance expliquée est le pourcentage associé au coefficient de détermination. Très très grossièrement : plus la variance expliquée est grande plus le modèle est bon MAIS il faut bien retenir que ça n'est pas une vraie définition et que l'augmentation du nombre de variables explicatives a tendance à augmenter le coefficient de détermination sans raison.

$$r^2 = 1 - \frac{\sum (y - y')^2}{\sum (y - \bar{y})^2} \quad EVS = 1 - \frac{Var(y - y')}{Var(y)}$$

---

```
sklearn.metrics.explained_variance_score(test_y, pred_y)  
sklearn.metrics.r2_score(test_y, pred_y)
```

---

- Erreur maximale

---

```
sklearn.metrics.max_error(test_y, pred_y)
```

---

- Taux d'erreur moyen

---

```
sklearn.metrics.mean_absolute_percentage_error(test_y,  
pred_y)
```

---

- Erreur quadratique logarithmique moyenne

---

```
sklearn.metrics.mean_squared_log_error(test_y, pred_y)
```

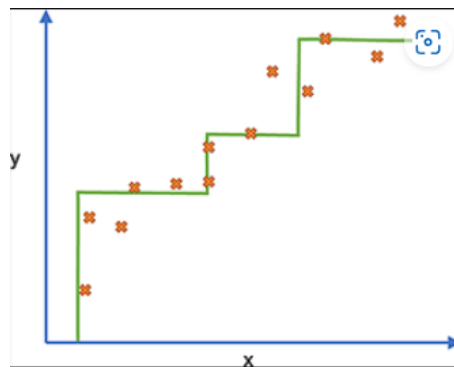
---

L'indicateur le plus utilisé est la RMSE mais on peut en utiliser plusieurs. A nouveau c'est important de définir l'évaluation avant la modélisation pour que l'évaluation reste pertinente.

### 2.7.3 Utilisation des algorithmes de classification

Les algorithmes de classification peuvent prédire la classe (variable cible) à partir de variables explicatives. Le nombre de classes est défini en amont et, dans le cas de la régression, il y a une infinité de valeurs possibles. Les algorithmes de classification ne peuvent donc pas prédire parfaitement des variables continues.

Cependant, on peut quand même les utiliser pour avoir une idée des valeurs prises par la variables sur tel ou tel intervalle en utilisant un nombre défini de classes, l'algorithme associera chaque donnée à la valeur de variable cible la plus proche parmi les potentielles.



On a forcément des résultats moins bons que pour un algorithmes de régression mais les algorithmes de classification on l'avantage d'être simples à mettre en place et d'être peu soumis à l'erreur.

#### Arbres de décisions

Les arbres de décisions sont utilisable pour les régressions. Au lieu d'associer une classe à chaque feuille, c'est la valeur moyenne de la variable cible des éléments de cette feuille qui sera donnée. Le résultat des arbres de décisions est donc une fonction en escalier. On l'appel avec le mot clé *DecisionTreeRegressor*. Pour les problèmes de régression, il est également possible d'utiliser des algorithmes qui utilisent des méthodes ensemblistes comme les forêts aléatoires (random forest) ou l'algorithme XGBoost.

#### K-nearest neighbors

On se rappelle que l'algorithme des K-plus proches voisins consiste à regarder les K points les plus proches d'une donnée à prédire et de prendre la classe majoritaire. Dans le cas d'une régression ce sera la moyenne de la valeur cible des voisins que sera renvoyées.

## SVM

On peut également utiliser les Support Vector Machines qui ont pour but de séparer l'espace par des hyper-plans pour classer les données sur des problèmes de régression. Comme pour les autres algorithmes, chaque classe peut être vue comme une valeur potentielle de la variable cible.

### 2.7.4 Régression linéaire et variantes

La régression linéaire est l'algorithme à la base de la régression. Il s'agit de déterminer une droite en fonction des différentes variables :

$$y = a_0 + \sum a_i x_i$$

Il y a une infinité de droites mais on choisit celle qui donne l'erreur quadratique la plus petite.

---

```
regressor = LinearRegression()
regressor.fit(train_X, train_y)
pred_y = regressor.predict(test_X)

#Evaluation
print(metrics.mean_squared_error(test_y, pred_y))

#Données du modèle (intercept = ordonnée à l'origine:
regressor.coef_
regressor.intercept_
```

---

Avantages : simplicité, vitesse de calcul et interprétation (signe et valeur absolue des coefficients directeurs).

Désavantages : pas toujours le plus performant, il ne donne pas les coefficients optimaux (souvent trop grands) et il est sensible à la colinéarité : l'impact des coefficients ne peut plus être comparé et leur signe non plus (les deux peuvent se compenser).

Pour compenser certains désavantages on peut utiliser la régression rigide : on ajoute une contrainte sur les facteurs de la régression, on parle de régularisation, pour garder les coefficients les plus petits possible.

Pour la mettre en place on ajoute un coefficient  $\alpha$  à la somme des carrés des coefficients. Il ne faut pas le choisir trop petit sinon il n'y aura aucun effet ni trop grand auquel cas il n'y aura pas d'apprentissage.

---

```
from sklearn.linear_model import Ridge
from sklearn import metrics

regressor = Ridge(alpha=1)
```

```
regressor.fit(train_X, train_y)
```

---

Lors d'une régression linéaire il est rare que certains paramètres valent 0 alors que dans la réalité tous les paramètres n'ont pas une influence. L'influence détectée lors de l'apprentissage est uniquement liée aux données d'apprentissage. Pour limiter cet effet on peut limiter le nombre de variables influentes pour améliorer la capacité de généralisation de notre modèle. Pour faire cela on applique la régression dite "lasso" qui reprend le principe de la régression rigide en limitant la valeur absolue des coefficients avec un paramètre  $\alpha$  mais la régularisation ne se fait plus par la somme quadratique mais par la somme des valeurs absolues des coefficients.

---

```
from sklearn.linear_model import Lasso
from sklearn import metrics

regressor = Lasso(alpha=1)
regressor.fit(train_X, train_y)
```

---

Pour choisir le paramètre  $\alpha$  il est possible de faire une recherche sur le meilleur paramètre. On utilise le mot clé *ndalphas* pour indiquer le nombre de  $\alpha$  à tester.

### 2.7.5 Régression polynomiale

Bien souvent, la fonction mathématique faisant le lien entre les variables explicatives et la variable cible n'est pas linéaire (degré  $\geq 1$ ). Les régressions vues précédemment ne s'appliquent qu'aux cas linéaires mais d'autres régressions existent et prennent en compte la non-linéarité du modèle à fournir.

Cependant, il n'existe pas d'algorithmes d'apprentissages faits pour les régressions polynomiales dans la bibliothèque scikit-learn. Mais Scikit propose tout de même un pré-processeur (*PolynomialFeatures, degree*) permettant de calculer toutes les variables de degré supérieur et ensuite il est possible de se ramener à un cas linéaire.

### 2.7.6 Cas particulier de la prédiction

#### Prédiction et séries temporelles

Le problème de prédiction ne repose pas sur des variables explicatives (ex : le prix d'une maison en fonction de sa taille, de son emplacement...) mais de ses valeurs précédentes (ex: augmentation de la valeur sur 10 ans). Il n'y a donc pas de dataset en générales mais une suite de valeurs dite *série temporelle*.

Les séries temporelles ont en général 2 données : l'indice de la donnée (ou une date) et la valeur à cet indice. Les variable cible et explicative sont confondues.

On peut analyser une série temporelle à partir de 4 éléments :

- La tendance générale : croissance, décroissance, stabilité...
- Les saisonnalités : des variations qui reviennent à des fréquences données.
- Les cycles : variations qui reviennent à des fréquences non définies.
- Le bruit : les variations non prévisibles souvent assez faible.

### Préparation des données

Lorsque l'on travaille avec des dates la préparation des données est très importantes et leur compréhension tout autant. On veut par exemple savoir : jour, mois, année, numéro de la semaine, vacances ou non, saison, week-end ... mais seulement si cela est pertinent ! La saison est pertinente pour la vente de glaces (consommation en hausse l'été) mais pas pour la vente de lit (qui reste la même toute l'année).

Il est important de bien préparer l'horizon : quelles valeurs regarder pour faire la prédiction. Cela peut-être les 30 derniers jours ou les valeurs des années précédentes à la même date.

### Application

La bibliothèque Scikit-learn ne propose pas d'algorithme pour traiter les séries temporelles (voir statsmodels) mais on peut les traiter avec les algorithmes de régression linéaire et en créant nous même l'horizon avec la fonction *shift*. On va par exemple créer trois variables avec dans chacune les valeurs à 1, 2, 3 intervalles de temps et faire la régression linéaires dessus.

### Utilisation de modèles spécifiques

L'application présentée ci-dessus ne marche bien que pour les séries temporelles linéaires avec peu de bruit et de saisonnalité. L'erreur devient bien plus grande sur des séries temporelles qui ne rentrent pas dans ce cadre.

Quelques modèles proposés par la librairie statsmodels.

- Auto-régression : des algorithmes basés sur le même fonctionnement que la régression linéaire avec la création d'un horizon puis d'un modèle linéaire.

---

```
import statsmodels.api as sm
#lags -> horizon
#trend -> t pour dependence au temps et c pour stable.
model = AutoReg(train_y, lags=18, trend='t')
model_fit = model.fit()
pred = model_fit.predict(0, 98)
```

---

- Moyenne glissante (Mobile average MA) : permet de prédire la valeur future à partir d'une combinaison linéaire de la moyenne de la série et des n valeurs précédentes d'un processus stochastique. Plusieurs paramètres sont donc à prendre en compte : n et le processus stochastique.
- Intégration : Les modèles ci-dessus ne s'appliquent qu'à des séries temporelles stationnaires (constante aux saisons et bruit près). Par un processus de différenciation on peut prédire la variation et non la valeur brute. On utilisera le terme "ordre de différenciation" pour décrire le degré de la série : 0 pas de différenciation la série est stationnaire, 1 la série suit une variation d'un polynôme de degré 1 on calcule vitesse, 2 la série suit une variation d'un polynôme de degré 2 et dans ce cas on prédit une accélération etc...
- ARIMA (AutoRegressive Integrated Moving Average) reprend les 3 concepts précédents et les prend en paramètres : la taille de l'horizon de l'auto-régression, l'ordre pour l'intégration, l'ordre pour la moyenne glissante.

---

```
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(train_y, order=(18, 1, 4))
model_fit = model.fit()
pred = model_fit.predict(80, 98)
```

---

## 2.8 Algorithmes d'apprentissage non supervisés

### 2.8.1 Le clustering

Le clustering consiste à identifier des données qui ont des caractéristiques similaires et former des clusters. Contrairement aux algorithmes de classification, les classes ne sont pas connues à l'avance, après la création de groupes par l'algorithme c'est à nous d'analyser manuellement les données pour labéliser les groupes.

Quelques exemples :

- Catégorisation automatique sur un site marchand de produits similaires.
- Classification des usages pour repérer les comportements frauduleux.
- Publicité ciblée pour proposer des publicités similaires à celles que l'utilisateur a déjà vues ou cliquées.
- Détection d'anomalie pour les images
- Regrouper des humains pour des réseaux sociaux si ils partagent des similitudes (âge, passions, métier...)



Il y a 2 grandes classes d'algorithmes pour faire du clustering.

### Algorithmes basés sur les distances.

- Algorithme K-Means :

L'utilisateur choisi un x nombre de cluster puis l'algorithme suivant est mis en oeuvre :

1. Choisir x centroïdes pour être un centre de cluster aléatoirement.
2. Associer tous les points à un des centroïde choisi précédemment en fonction de la distance.
3. Calculer le barycentre de chaque groupe crée et déplacer le centroïde à la position du barycentre.
4. Boucler de 2 à 4 jusqu'à stabilisation (déplacement des centroïdes inférieur à une limite passée en paramètre).

Cet algorithme n'est pas parfait :

- L'utilisateur doit préciser le nombre de clusters à l'avance donc il est souvent nécessaire de faire des tests avec plusieurs valeurs.
- Les résultats sont dépendants du placement initial des centroïdes. Il est donc nécessaire de faire plusieurs démarrages avec des graines aléatoires. On fera alors une comparaison des différentes propositions pour voir les clusters qui reviennent le plus ou qui donnent les meilleurs résultats.

Pour comparer 2 modèles on utilise l'inertie. C'est la somme des distances au carré de chaque point du dataset au centroïde le plus proche. Plus l'inertie est faible plus les points sont proches de leur centre donc un modèle avec une inertie faible sera considéré meilleur.

Dans scikit-learn l'implémentation des algorithmes de clustering se fait uniquement sur des dataset avec des données numériques et sans données manquantes.

---

```
from sklearn.cluster import KMeans

clusterAlgo = KMeans(n_clusters=3, random_state=42)
clusterAlgo.fit(titanic_df)

clusterAlgo.cluster_centers_
clusterAlgo.inertia_
```

---

- Variantes des K-Means (mini-batch):

Une autre problématique de l'algorithme des K-means est le temps de calcul de toutes les distances qui peut être très long. La version "mini-batch" permet de régler ce défaut.

Dans cette version, au lieu de calculer la distance pour tous les éléments du dataset on calculera la distance pour seulement une partie choisie aléatoirement des éléments et qui change à chaque nouvelle itération.

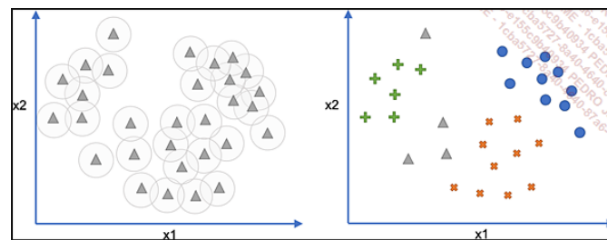
Cet algorithme est surtout à utiliser pour les plus grands dataset où le coût entre les 2 algorithmes est notable. Cependant, il est important de garder en tête que cet algorithme donne des résultats de moins bonne qualité (pour moins de 10 clusters les pertes sont d'environ 2% et plus de 8% pour + de 20 clusters. Voir cette étude pour une comparaison plus détaillée.

### Algorithmes basés sur la densité

Les algorithmes basés sur les distances ont deux défauts majoritaires : il faut indiquer le nombre de clusters et ils ne marchent pas sur les clusters plus allongés. D'autres algorithmes ont donc été créés souvent basé sur la densité.

- DBSCAN

Cet algorithme consiste à construire un voisinage de chaque point (distance  $\epsilon$  donnée en paramètre). Si dans le voisinage d'un point il a au moins  $n$  autres points alors il est considéré comme étant au coeur d'un cluster. Les points ayant au moins 1 voisin mais moins de  $n$  voisins sont considérés être à la frontière du cluster. Les points qui n'ont aucun voisin sont considérés comme étant du bruit.



---

```
from sklearn.cluster import DBSCAN

clusterAlgo = DBSCAN(eps=1, min_samples=15)
clusterAlgo.fit(titanic_df)
clusterAlgo.labels_
```

---

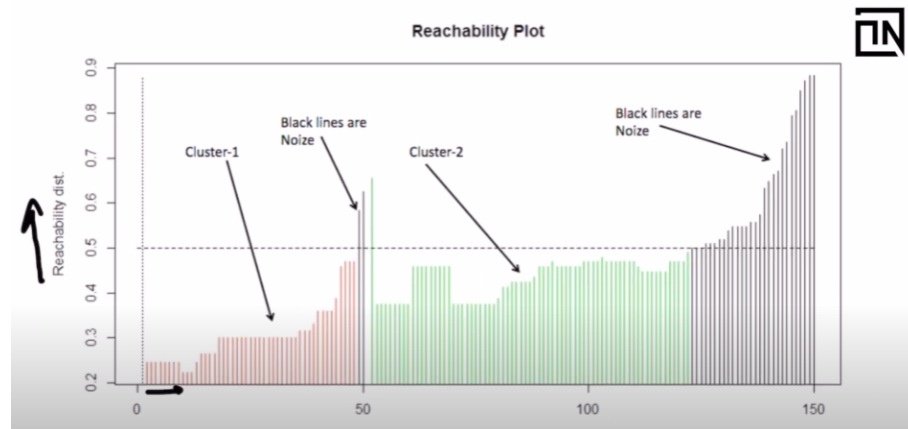
- Variante : OPTICS

Cette variante permet de corriger la problématique du choix d' $\epsilon$  dans l'algorithme DBSCAN. Si il est trop petit alors trop de points sont du bruit et trop grand et on a un seul grand cluster.

OPTICS prend un paramètre un  $\epsilon$  maximum. On peut créer des voisinages de plus petite taille que ce maximum. Ceci est très pratique pour les cluster de différentes densités. Pour cela on introduit 2 notions :

- core distance : la plus petite distance pour laquelle le point considéré soit au coeur d'un cluster.
- reachability distance : la distance entre un point donné et le plus proche point faisant partie d'un cluster (core ou border)

On construit ensuite un diagramme où l'on classe par ordre croissant de reachability distance tous les les points. On décide d'une limite de reachability distance à partir de laquelle les points seront considérés comme du bruit. Les clusters apparaîtrons sur le diagramme ensuite.



## 2.8.2 Réduction des dimensions

Dans beaucoup des cas les dataset comprennent de nombreux paramètres et la visualisation des données est vite limité par le fait que l'on ne puisse visualiser des données que jusqu'à 3 dimensions. La réduction des dimensions est rarement utilisée en temps que modèle de Machine Learning mais plutôt comme outils pour aider à la visualisation des données et ainsi à leur bonne préparation pour l'application d'un autre modèle. Le but est donc de pouvoir faire ressortir les tendances principales, les potentielles limites et d'estimer le bruit présent dans les données pour la partie visualisation et pour la préparation des données l'objectif est de ne retenir que les caractéristiques pertinentes, éliminer la redondance, alléger le dataset en limitant la quantité de données (sans perdre la qualité du dataset pour autant).

### Détection des axes principaux

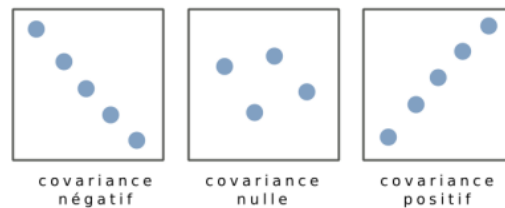
Une première façon de réduire la dimension est de déterminer quels sont les principaux axes. Pour faire cela on peut s'appuyer sur les algorithmes d'apprentissage supervisés notamment les arbres de décisions qui donnent les variables ayant le plus d'importance en peu de temps.

### Création de nouveaux axes

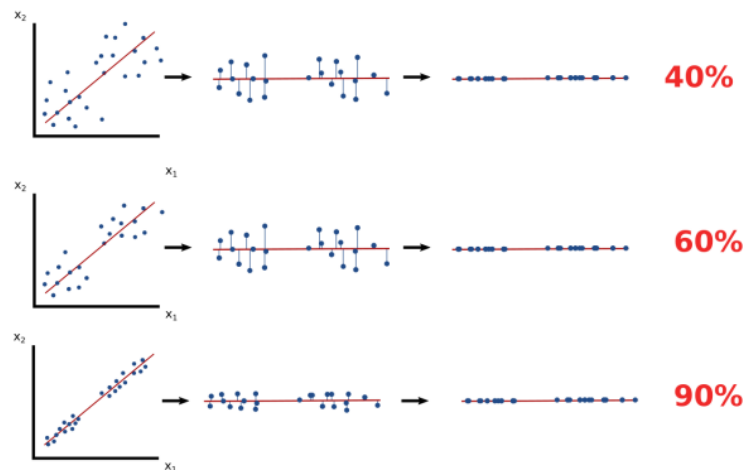
Une approche plus puissante et plus intéressante est de faire un changement de repère pour aller dans un repère avec moins de dimensions mais sans perdre d'information. Il y a plusieurs moyens de faire cela :

- Principal Component Analysis (PCA) ou décomposition en composantes principales :

Lien. Cette méthode consiste à concentrer les informations de deux variables dans une même nouvelle variable : une composante principale. Pour cela nous allons regarder la covariance des variables entre elles. Si deux variables ont une forte covariance en valeur absolue alors cela signifie que les 2 variables sont fortement corrélées (si l'une augmente l'autre aussi ou l'autre diminue proportionnellement).

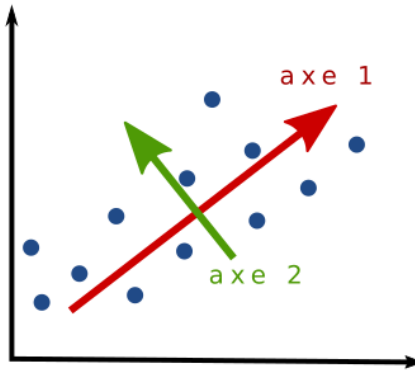


Dans ce cas là, les 2 variables pourraient être transformées en une seule avec une perte d'information faible mais un gain en temps de calcul et en visualisation. Visuellement on veut faire cela :



Comme on le voit, on perd plus ou moins d'information par cette technique. Il est donc important de préciser cette perte car l'oubli de cette étape rendrait impossible l'interprétation des résultats.

L'objectif est donc de pouvoir trouver l'axe sur lequel projeter les données. Pour cela on va utiliser l'algèbre linéaire : On commence tout d'abord par trouver la matrice de covariance pour chaque couple de variable puis on cherche ses valeurs propres et ses vecteurs propres qui vont justement correspondre aux axes de composante principale. On choisira le vecteur propre associé à la plus grande des valeurs propres de la matrice : ce sera celle qui récupérera les plus d'information. Visuellement :



Si l'on part d'un espace à  $n$  dimensions et que l'on veut se ramener à un espace de  $p$  dimensions il suffit d'effectuer le processus  $n-p$  fois en choisissant à chaque fois les 2 variables ayant la plus grande covariance en valeur absolue.

- Linear Discriminant Analysis :  
LDA cherche également un axe sur lequel projeter les données des 2 variables. Cette méthode, plutôt que de maximiser les variations comme le fait PCA, essaye de maximiser la séparabilité entre les catégories. Les critères pour choisir l'axe sur lequel projeter sont :
  - Maximiser la distance entre les moyennes des 2 variables après projection.
  - Minimiser la variance des 2 variables.

$$ratio = \frac{(\mu_1 - \mu_2)^2}{s_1^2 + s_2^2}$$

De la même façon que pour PCA on répète le procédé autant de fois que nécessaire.

L'algorithme LDA est préférable notamment si on souhaite appliquer un apprentissage derrière.

### 2.8.3 Systèmes de recommandation

Il s'agit d'un algorithme qui permet de recommander du contenu ou des produits en fonction de l'historique de l'utilisateur. Ils sont très utilisés par Amazon, Netflix, Youtube, Spotify etc...

Deux cas un peu plus complexes existent :

Le nouvel utilisateur à qui on fera des recommandations aléatoires ou à qui on demandera au moment de l'inscription les sujets qui l'intéresse.

Le nouveau contenu qui sera proposé de façon aléatoire ou selon sa similarité avec d'autres produits déjà existants.

Les principales approches sont :

- Modèles basés sur la popularité  
C'est l'algorithme le plus simple, il consiste à simplement compter le nombre de personnes qui consomment chaque produit et les plus populaires seront mis en avant par la plateforme. Il permet notamment d'avoir quelque chose à proposer même pour une plateforme avec peu d'historique et de pouvoir gérer les nouveaux utilisateurs.

Cependant les nouveaux produits qui n'ont donc jamais été consommé seront montrés à personne donc une part d'aléatoire doit avoir lieu pour qu'il puisse être apporté à la vue de quelques personnes et qu'il devienne populaire ou pas.

- Modèles basés sur le contenu (content-based filtering)  
Cet algorithme repose sur les metadata des données : artiste, album, titre, bpm, producteur, genre pour une musique par exemple, il va proposer aux utilisateurs des contenus similaires à ceux qu'il a déjà consommés en calculant la distance entre 2 produits et en sélectionnant les plus proches.

Cet algorithme a l'avantage d'être plus personnalisé que le tri par popularité mais il a tendance à entraîner un manque de diversité et nécessite que tous les contenus aient des metadonnées correctement remplies. Un autre soucis se pose pour les nouveaux utilisateurs qui n'ont rien consommé, il est souvent d'usage de leur demander leur préférences lors de l'inscription pour commencer la personnalisation.

- Modèles basés sur les autres utilisateurs (collaborative filtering)  
Ce modèle repose sur les relations entre les différents utilisateurs. Si deux utilisateurs ont aimé les mêmes choses alors on peut supposer que les nouveaux produits consommés par l'un des deux intéressera l'autre également. Pour choisir les produits à montrer à un utilisateur, l'algorithme va faire

la moyenne des avis des personnes proches sur ce produit.

Comme pour l'algorithme basé sur le contenu cet algorithme pose problèmes pour les nouveaux utilisateurs et les nouveaux produits et peut être réglé par les mêmes solutions.

- Méthodes hybrides

En réalité ces algorithmes sont rarement utilisés seuls mais sont plutôt des mélanges avec une approche sur les utilisateurs et une autre approche comme la popularité ou le contenu.

Il est également possible de proposer différentes zones de recommandations qui permettent d'augmenter les chances de nouvelle consommation de produit.

#### 2.8.4 Association

Le but des algorithmes d'association est de ... *roulement de tambours* ... faire des associations (*qui l'eût crut ?*) ! Plus généralement, il a pour but de faire des liens entre des objets des produits qui peuvent être consommés en même temps comme des sushi et de la sauce soja ou des pâtes et de la sauce tomate. Le but final est de pouvoir augmenter les ventes, par exemple :

- Faire une promotion sur l'achat des 2 produits.
- Mettre les 2 produits à côté.
- Espacer les 2 produits s'ils sont très souvent consommés ensemble pour faire parcourir de nouveaux rayons aux acheteurs.

L'exemple le plus courant est donc celui d'un supermarché et des produits qui se retrouvent le plus ensemble dans un panier mais c'est aussi utile pour la bio-informatique, la détection de fraude ou la fouille sur le Web.

Les associations proposent des règles sous la forme  $A \Rightarrow B$  avec A et B pouvant contenir plusieurs éléments :

- Si A possède plusieurs éléments alors il faut le cumul des éléments de A pour avoir B.
- Si B possède plusieurs éléments alors A entraîne tous les éléments de B.

#### Évaluation :

- Le support

Il indique la part de A dans la dataset. Plus il est grand plus le nombre de clients concernés est important et donc plus intéressant.

- L'indice de confiance

Il permet d'estimer la probabilité d'avoir B si l'on a A. Il faut qu'il soit suffisamment grand pour ne pas juste correspondre à un tirage aléatoire mais dans le cas d'étude d'aléa climatique on préfère un indice de confiance

plus faible quitte à avoir des fausses alertes qu'un indice de confiance trop élevé et ne pas être alerté sur un danger.

- Le lift

C'est l'indicateur d'information apporté par la règle, il estime la différence entre l'existence d'une corrélation entre 2 objets et le fait que les 2 objets soient aléatoires. Un lift égale à 1 signifie que les 2 objets n'ont aucun lien mais plus il augmente plus l'information apportée par la règle est importante, on retiendra les règles avec un lift supérieur à 2.

### Algorithme "Apriori" :

Le nombre de règles possibles augmentent exponentiellement avec l'ajout de nouveaux produits (parties d'un ensemble), on ne peut donc pas se permettre de tout tester. Pour palier à ce problème on va donc appliquer l'algorithme "apriori" pour choisir quelles règles sont à tester.

On fixe tout d'abord un *minsup* et un *minconf* souvent 0.2 et 0.5 respectivement la plupart du temps.

On va tout d'abord faire des sets de 1 produit puis 2 puis 3... en éliminant ceux n'ayant pas le support minimum et en s'arrêtant quand aucune construction n'est possible.

En suite, on regarde le lift et la confiance de chaque règle issue d'un set de plusieurs éléments et on garde finalement les règles qui ont un bon lift et un bon indice de confiance.

## 2.9 Évaluation et déploiement

### 2.9.1 Phase d'évaluation

Après avoir évalué le modèle avec les données de test et des validation croisées il faut évaluer le modèle avec le métier (le client, des experts du milieu etc...) pour savoir si le modèle correspond bien aux attentes.

Par exemple, un modèle avec une précision de 99% pour détecter les erreurs de français dans une dictée serait considéré comme très bon mais il serait sans doute vu comme très mauvais s'il s'agit de détecter un piéton sur la route. Il faut faire du cas par cas selon les situations et les domaines. Il ne faut surtout ne pas hésiter à revenir aux étapes précédentes : préparation des données, modélisation, travail des hyper-paramètres etc...

En suite, il faut revenir sur le processus pour bien vérifier que l'on a pas eu de data leakage : utiliser les données de test pour l'entraînement. Au quel cas les métriques pour tester la qualité du modèle ne veulent rien dire et il faudra recommencer en faisant bien attention à ne pas utiliser les données de test.



Une fois que le modèle et le processus validé il faut décider de ce que l'on souhaite faire :

- Passer en phase de déploiement.
- Corriger les défauts trouvés et améliorer le modèle.
- Arrêter le projet.

Comme pour tout le reste, la phase d'évaluation et les décisions prises doivent être judicieusement documentées et justifiées.

### **2.9.2 Phase de déploiement**

C'est la dernière phase de la méthode CRISP-DM que l'on présentait au début de cette synthèse.

#### **Planification du déploiement**

Il faut tout d'abord décider quel modèle on va déployer et fournir une documentation complète sur la création du modèle, ses utilisations, les données d'entrée et de sortie, ses limites et l'utilisation des résultats dans le système plus global.

#### **Monitoring et maintenance**

La mise en marche d'un modèle ne signifie par forcément la fin du projet. En effet, pendant potentiellement plusieurs années du monitoring et de la maintenance seront nécessaires pour observer l'évolution du modèle et ses potentiels défauts.

Cela couvre divers aspects :

- Techniques : est-ce que le modèle est rapide, disponible etc...
- Les données passées en entrées : sont-elles toujours les mêmes qu'avant ?
- Les résultats du modèle : sont-ils toujours aussi précis qu'au début ?

#### **Rapport final et maintenance**

Un projet sera considéré comme fini lorsque toute la documentation aura été terminée. Elle comprendra autant les aspects techniques du modèle sur son utilisation, ce qu'il donne, sa création mais aussi les aspects plus généraux du projet comme le respect des délais, du budget, les variations qu'il y aura eu etc... Et les justifications !

## **3 Analyse de texte avec Python**

### **3.1 Références**

Livres :

Natural Language Processing with Python Quick Start Guide  
Natural Language Processing and Computational Linguistics

Hands-On Natural Language Processing with Python  
Hands-On Python Natural Language Processing  
The Applied AI and Natural Language Processing Workshop  
Natural Language Processing: Python and NLTK  
Natural Language Processing with Flair  
The Natural Language Processing Workshop  
Natural Language Processing Fundamentals  
Python Natural Language Processing

Vidéos :

Natural Language Processing in Python

Texte processing en Python tutoriel français complet (NLTK et Spacy)

## 3.2 Introduction

Le traitement du langage naturel est une composante de l'intelligence artificielle. Par langage naturel on entend le français, le chinois ou l'anglais et non pas les langages de programmation tels que Scala, Python ou Java. L'objectif de cette discipline est de pouvoir tirer des informations pertinentes (avis, sentiments, problématiques...) à partir de textes. Les données que l'on peut avoir à traiter sont très variées, elles peuvent aller des tweets d'un utilisateur à des critiques de films à des commentaires de techniciens sur une production pour Orange.

Comme pour la création de modèle en Machine Learning l'analyse de texte demande de suivre des étapes pour obtenir des résultats pertinents :

- Préparation du sujet, problématisation et récupérer les données
- Préparation des données
- Analyse exploratoire des données / statistiques
- Traitement des données par l'application de techniques d'analyse de texte
- Synthétisation des résultats et réponse à la problématique

Pour faire cela on va utiliser de nombreuses bibliothèques python (textblob, numpy, scikit-learn, pandas...) et le notebook Jupyter.

## 3.3 Préparation du sujet, problématisation et récupération des données

C'est la première étape, elle va servir à poser les bases et orienter notre projet. Il ne faut donc pas la négliger. Cette étape consiste à poser les contours de notre problème ou de notre question et le cas échéant d'émettre des hypothèses. Ceci va nous amener à nous questionner sur les données nécessaires pour répondre

et ensuite à récupérer ces données.

Pour l'ensemble de ce chapitre nous allons nous appuyer sur l'exemple d'Alice Zhao présenté lors d'une conférence pour PyOhio. Sa problématique est la suivante : qu'est-ce qui rend le show d'Ali Wang si spécial ? Pourquoi l'apprécie-t-elle tant ?

Pour répondre à sa question elle souhaite récupérer le transcript du show d'Ali Wang et 11 autres comédiens (quelle a choisis selon des critères précis) afin de réaliser une analyse de texte : est-ce que le style d'Ali Wang de celui des autres et quels sont les comédiens susceptibles de l'intéresser (qui ont un style proche de celui d'Ali) ?

Elle crée donc un script Python pour automatiquement aller récupérer les transcripts sur internet. Après avoir tout récupérer il est pertinent de "pickle" nos trouvailles pour éviter d'aller chercher les données à chaque fois et éviter des temps de calcul inutiles.

---

```
#data_df est un dataframe cree avec pandas.  
import pickle  
data_df.to_pickle("corpus.pkl")
```

---

### 3.4 Préparation des données

Après la collecte des données, on passe à l'étape la plus importante de tout le processus : le préparation des données et notamment le nettoyage du texte.

On débute avec un **corpus** : une base de données avec un texte associé à chaque origine de la donnée. Cependant, un corpus est difficile à analyser tel quel. Il faut donc préparer le texte et le rendre le plus simple possible. On va transformer le texte de plusieurs manières. Il est important de garder en tête que l'on a souvent recours à plusieurs nettoyages successifs si l'on se rend compte que l'on a oublié quelque chose, que certains mots sont inutiles... Voici les principales méthodes de nettoyage appliquées aux textes :

- Nettoyer le texte : enlever la ponctuation, mettre en minuscule, enlever les nombres, enlever le texte qui ne fait pas de sens (/n)...

Exemple : "All right, Petunia. Wish me luck out there. You will die on august 2377." devient "all right petunia wish me luck out there you will die on august"

- Résoudre les problèmes de typos.
- Séparer le texte en plus petites parties (tokenize) : mot, deux mots, phrase.

- Enlever les -ation : féminiser et féminisation décrivent la même chose mais seraient comptés comme 2 mots différents.
- Enlever les "stop words" : les mots sans grande importance de la langue comme le, une, tout, ne etc... en français.

Exemple : Dans la phrase précédente on ne gardera que les mots right, petunia, wish, luck, die et august. L'ordre n'a pas d'importance.

- Mettre les mots dans une **matrice**. Cette matrice est construite avec une colonne pour l'origine de la donnée (ex : le comédien ou la production) et une colonne pour chaque mot où les coefficients sont le nombre de fois où le mot apparaît dans la donnée.

Comedian	right	petunia	wish	me	luck	die	august	hello	thank	welcome	wow	exciting
John Mulaney	1	1	1	1	1	1	1	0	0	0	0	0
Ali Wong	0	0	0	0	0	0	0	3	2	0	0	0
Dave Chappelle	0	0	0	0	0	0	0	0	3	0	1	1
...												

Quelques commandes pour faire cela :

---

```

# Première étape de nettoyage
import re
import string

def clean_text_round1(text):
    '''Make text lowercase, remove text in square brackets, remove
    punctuation and remove words containing numbers.'''
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text

round1 = lambda x: clean_text_round1(x)

# 2ème étape de nettoyage
def clean_text_round2(text):
    '''Get rid of some additional punctuation and non-sensical text that
    was missed the first time around.'''
    text = re.sub('[\"']', '', text)
    text = re.sub('\n', '', text)
    return text

round2 = lambda x: clean_text_round2(x)

```

```

data_clean = pd.DataFrame(data_clean.transcript.apply(round1))
data_clean = pd.DataFrame(data_clean.transcript.apply(round2))
data_clean

from sklearn.feature_extraction.text import CountVectorizer

"Enlever les stopwords, en precisant la langue
cv = CountVectorizer(stop_words='english')
data_cv = cv.fit_transform(data_clean.transcript)
data_dtm = pd.DataFrame(data_cv.toarray(),
                        columns=cv.get_feature_names())
data_dtm.index = data_clean.index
data_dtm

```

---

### 3.5 Analyse exploratoire des données

Après avoir nettoyé nos données et les avoir standardisées, il est temps de regarder ce qu'elles contiennent et si elles font du sens. On verra vite que, comme pour le Machine Learning, il sera nécessaire de revenir à l'étape de nettoyage pour obtenir des résultats satisfaisants.

Le but est un peu le même que pour les données numériques, nous allons essayer de trouver les principaux motifs avec des outils simples avant d'appliquer des techniques de Machine Learning pour voir des motifs cachés.

Dans son exemple, Alice Zhao regarde les mots les plus communs, la taille du vocabulaire et ... le nombre d'insultes. C'est une bonne façon de voir que chaque analyse est à faire en fonction du contexte des données !

On notera d'ailleurs que dans son code Alice Zhao affiche régulièrement les données qu'elle obtient. C'est une très bonne façon de comprendre ce que l'on fait.

#### 3.5.1 Les mots les plus communs

---

```

# Read in the document-term matrix
import pandas as pd

data =
    pd.read_pickle('C:/Users/SPML0410/Downloads/nlp-in-python-tutorial-master/
nlp-in-python-tutorial-master/pickle/dtm.pkl')
data = data.transpose()
data.head()

# Find the top 30 words said by each comedian
top_dict = {}

```

```

for c in data.columns:
    top = data[c].sort_values(ascending=False).head(30)
    top_dict[c]= list(zip(top.index, top.values))

top_dict
# Print the top 15 words said by each comedian
for comedian, top_words in top_dict.items():
    print(comedian)
    print(', '.join([word for word, count in top_words[0:14]]))
    print('---')

```

---

On obtient (en partie) le résultat suivant :

```

ali
like, im, know, just, dont, shit, thats, youre, gonna, ok, lot, gotta, oh, wanna
---
anthony
im, like, know, dont, got, joke, thats, said, anthony, day, say, just, guys, people
---
bill
like, just, right, im, know, dont, gonna, got, fucking, yeah, shit, youre, thats, dude
---
bo
know, like, think, love, im, bo, just, stuff, repeat, dont, yeah, want, right, cos
---
dave
like, know, said, just, im, shit, people, didnt, ahah, dont, time, fuck, thats, fucking
---
hasan
like, im, know, dont, dad, youre, just, going, thats, want, got, love, shes, hasan
---
jim
like, im, dont, right, fucking, just, went, know, youre, people, thats, day, oh, think
---
joe
like, people, just, dont, im, fucking, fuck, thats, gonna, theyre, know, youre, think, shit
---

```

On voit assez vite que beaucoup de mots sont communs à tous les comédiens, ça n'est donc pas très pertinent de les comparer sur cette base de mot. Pour trouver des informations pertinentes, on retourne à l'étape de nettoyage du code en ajoutant les mots qui sont partagés par plus de la moitié des comédiens à la liste des stop-words.

---

```

# Look at the most common top words --> add them to the stop word list
from collections import Counter

# Let's first pull out the top 30 words for each comedian
words = []
for comedian in data.columns:
    top = [word for (word, count) in top_dict[comedian]]
    for t in top:
        words.append(t)
# If more than half of the comedians have it as a top word, exclude it from the list

```

```

add_stop_words = [word for word, count in Counter(words).most_common()
                   if count > 6]
# Let's aggregate this list and identify the most common words along
with how many routines they occur in
Counter(words).most_common()

# Let's update our document-term matrix with the new list of stop words
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer

# Read in cleaned data
data_clean =
    pd.read_pickle('C:/Users/SPML0410/Downloads/nlp-in-python-tutorial-master/nlp-in-python-tutorial-r

# Add new stop words
stop_words = text.ENGLISH_STOP_WORDS.union(add_stop_words)

# Recreate document-term matrix
cv = CountVectorizer(stop_words=stop_words)
data_cv = cv.fit_transform(data_clean.transcript)
data_stop = pd.DataFrame(data_cv.toarray(),
                          columns=cv.get_feature_names())
data_stop.index = data_clean.index

# Pickle it for later use
import pickle
pickle.dump(cv, open("cv_stop.pkl", "wb"))
data_stop.to_pickle("dtm_stop.pkl")

```

---

Maintenant que la liste des mots les populaires chez chaque comédien a été mise à jour et est plus personnelle, on peut s'essayer aux nuages de mots qui sont un bon moyen de visualiser rapidement les mots principaux.

---

```

# Let's make some word clouds!
# Terminal / Anaconda Prompt: conda install -c conda-forge wordcloud
from wordcloud import WordCloud

wc = WordCloud(stopwords=stop_words, background_color="white",
               colormap="Dark2",
               max_font_size=150, random_state=42)
# Reset the output dimensions
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [16, 6]

full_names = ['Ali Wong', 'Anthony Jeselnik', 'Bill Burr', 'Bo Burnham',
              'Dave Chappelle', 'Hasan Minhaj',
              'Jim Jefferies', 'Joe Rogan', 'John Mulaney', 'Louis C.K.',

```

```

'Mike Birbiglia', 'Ricky Gervais']

# Create subplots for each comedian
for index, comedian in enumerate(data.columns):
    wc.generate(data_clean.transcript[comedian])

    plt.subplot(3, 4, index+1)
    plt.imshow(wc, interpolation="bilinear")
    plt.axis("off")
    plt.title(full_names[index])

plt.show()

```

---



On voit dans l'exemple d'Alice Zhao qu'Ali Wong utilise beaucoup le mot "shit" tout comme Dave Chappelle et que les mots "fuck" et "fucking" reviennent aussi beaucoup !

Pour résumer : récupérer les mots les plus populaires pour essayer d'observer des tendances en fonction des personnes/groupes/types et qu'il peut être nécessaire de revenir à l'étape de nettoyage des données.

### 3.5.2 Nombre de mots

Après avoir analysé les mots les plus populaires on peut s'intéresser au nombre de mots et de mots unique pour chaque type de donnée (ou chaque comédien dans notre exemple).

---

```

# Find the number of unique words that each comedian uses

# Identify the non-zero items in the document-term matrix, meaning that
# the word occurs at least once
unique_list = []
for comedian in data.columns:
    uniques = data[comedian].to_numpy().nonzero()[0].size
    unique_list.append(uniques)

```



```

# Create a new dataframe that contains this unique word count
data_words = pd.DataFrame(list(zip(full_names, unique_list)),
    columns=['comedian', 'unique_words'])
data_unique_sort = data_words.sort_values(by='unique_words')
data_unique_sort

```

---

	comedian	unique_words
1	Anthony Jeselnik	984
9	Louis C.K.	1098
3	Bo Burnham	1272
6	Jim Jefferies	1313
0	Ali Wong	1341
8	John Mulaney	1391
4	Dave Chappelle	1404
7	Joe Rogan	1435
10	Mike Birbiglia	1494
5	Hasan Minhaj	1559
2	Bill Burr	1633
11	Ricky Gervais	1633

Il peut aussi être pertinent de regarder le nombre de mots à la minutes (puisque l'on a la durée des spectacles).

---

```

# Calculate the words per minute of each comedian

# Find the total number of words that a comedian uses
total_list = []
for comedian in data.columns:
    totals = sum(data[comedian])
    total_list.append(totals)

# Comedy special run times from IMDB, in minutes
run_times = [60, 59, 80, 60, 67, 73, 77, 63, 62, 58, 76, 79]

# Let's add some columns to our dataframe
data_words['total_words'] = total_list
data_words['run_times'] = run_times
data_words['words_per_minute'] = data_words['total_words'] /
    data_words['run_times']

```

```

# Sort the dataframe by words per minute to see who talks the slowest
  and fastest
data_wpm_sort = data_words.sort_values(by='words_per_minute')
data_wpm_sort

# Let's plot our findings
import numpy as np

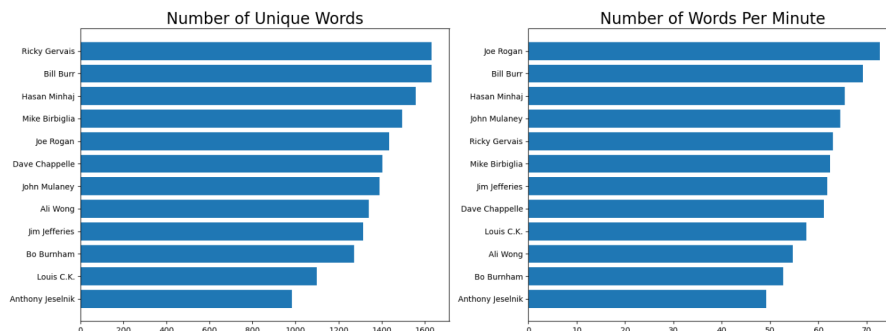
y_pos = np.arange(len(data_words))

plt.subplot(1, 2, 1)
plt.barh(y_pos, data_unique_sort.unique_words, align='center')
plt.yticks(y_pos, data_unique_sort.comedian)
plt.title('Number of Unique Words', fontsize=20)

plt.subplot(1, 2, 2)
plt.barh(y_pos, data_wpm_sort.words_per_minute, align='center')
plt.yticks(y_pos, data_wpm_sort.comedian)
plt.title('Number of Words Per Minute', fontsize=20)

plt.tight_layout()
plt.show()

```



Enfin, on pourra retenir de cette analyse les personnes qui parlent le plus et le moins vite et celles qui disent le plus de mots ou le moins lors d'un show.

De manière générale, il est intéressant de regarder les données sous tous les angles. Parfois, on apprendra pas grand chose d'intéressant mais parfois on pourra découvrir des choses que l'on ne soupçonnait pas. Il ne faut pas se censurer.

### 3.5.3 Le nombre de grossièretés

En parlant de censure ! On a vu avec les nuages de mots que beaucoup de comédiens étaient friands de "shit" et autres "fuck". Alice Zhao décide donc de

faire une analyse sur ces mots.

On remarquera qu'elle fait le choix de ne regarder que "shit", "fuck" et "fucking" et pas d'autres insultes et qu'elle choisi de concaténer les données de "fuck" et de "fucking".

---

```
# Earlier I said we'd revisit profanity. Let's take a look at the most
common words again.
Counter(words).most_common()

# Let's isolate just these bad words
data_bad_words = data.transpose()[['fucking', 'fuck', 'shit']]
data_profanity = pd.concat([data_bad_words.fucking +
    data_bad_words.fuck, data_bad_words.shit], axis=1)
data_profanity.columns = ['f_word', 's_word']
data_profanity

# Let's create a scatter plot of our findings
plt.rcParams['figure.figsize'] = [10, 8]

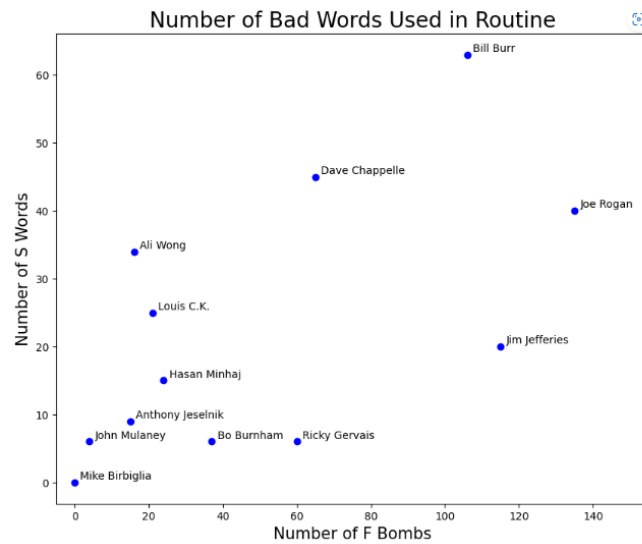
for i, comedian in enumerate(data_profanity.index):
    x = data_profanity.f_word.loc[comedian]
    y = data_profanity.s_word.loc[comedian]
    plt.scatter(x, y, color='blue')
    plt.text(x+1.5, y+0.5, full_names[i], fontsize=10)
    plt.xlim(-5, 155)

plt.title('Number of Bad Words Used in Routine', fontsize=20)
plt.xlabel('Number of F Bombs', fontsize=15)
plt.ylabel('Number of S Words', fontsize=15)

plt.show()
```

---

On obtient le graphique suivant :



J'ai également essayé de faire un histogramme avec le nombre total d'insultes (pas que "fuck", "shit" et "fucking") ainsi que de calculer les pourcentages d'insultes sur le total de mots.

---

```
# Let's get most of the bad words
data_bad_words = data.transpose()[['fucking', 'fuck',
    'shit', 'sluts', 'slut', 'motherfucker', 'jerk', 'cunt', 'bastard', 'fuckface', 'bitch', 'pussy', 'fag', 'fa
data_bad_words = data_bad_words.transpose()
data_bad_words
```

---

	ali	anthony	bill	bo	dave	hasan	jim	joe	john	louis	mike	ricky
fucking	5	6	70	22	32	8	78	69	2	6	0	47
fuck	11	9	36	15	33	16	37	66	2	15	0	13
shit	34	9	63	6	45	15	20	40	6	25	0	6
sluts	0	0	0	17	0	0	0	0	0	0	0	0
slut	0	0	0	6	0	0	1	0	0	0	0	0
motherfucker	1	0	1	0	10	0	0	3	2	1	0	0
jerk	0	0	0	0	0	0	0	1	0	1	0	0
cunt	1	0	0	0	0	0	15	0	0	0	0	6
bastard	0	1	0	0	0	0	2	0	0	0	0	0
fuckface	0	0	0	0	0	0	1	1	0	0	0	0
bitch	3	1	2	2	4	0	2	10	1	0	0	1
pussy	5	0	2	6	2	0	3	2	0	2	0	0
fag	0	0	0	3	0	0	0	0	0	0	0	0
faggot	0	0	0	2	0	0	0	1	0	0	0	1
scum	0	0	0	0	0	0	0	0	0	0	0	3
prick	0	0	0	4	0	0	0	0	2	0	0	0
faggot	0	0	0	2	0	0	0	1	0	0	0	1
whore	0	0	0	0	0	0	0	0	0	2	2	0
shitty	3	0	2	0	0	2	0	2	0	3	0	1
damn	0	1	1	0	3	1	0	4	0	0	0	0

On peut notamment remarquer que Mike ne dit aucune insulte durant tout son show à l'exception de 2 "whore".

---

```

#Now we concatenate everything in one column
insult_count_list = []
for comedian in data_bad_words.columns:
    totals = sum(data_bad_words[comedian])
    insult_count_list.append(totals)

#And we create a new dataframe
data_count_insult = pd.DataFrame(list(zip(full_names,
    insult_count_list)), columns=['comedian', 'insults'])
data_count_insult = data_count_insult.sort_values('insults',axis=0)

# Let's add some columns to our dataframe
data_count_insult['total_words'] = data_words['total_words']
data_words['run_times'] = run_times
data_count_insult['insult per word'] = data_count_insult['insults'] /
    data_count_insult['total_words']
data_count_insult

```

---

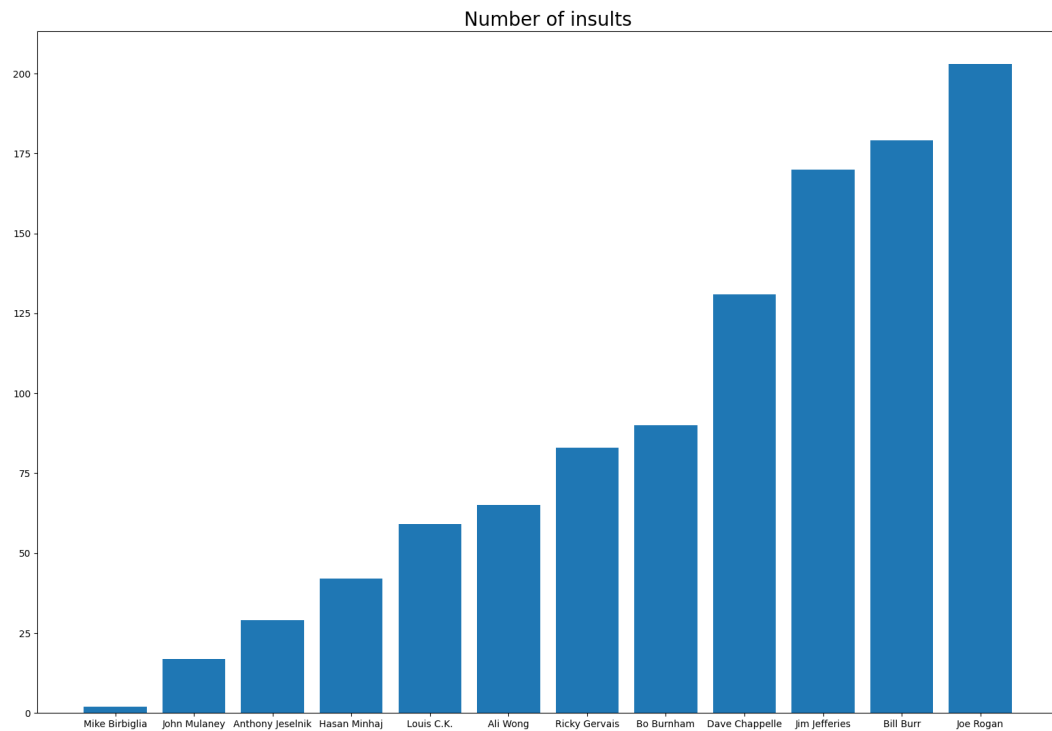
	comedian	insults	total_words	insult per word
10	Mike Birbiglia	2	4741	0.000422
8	John Mulaney	17	4001	0.004249
1	Anthony Jeselnik	29	2905	0.009983
5	Hasan Minhaj	42	4777	0.008792
9	Louis C.K.	59	3332	0.017707
0	Ali Wong	65	3283	0.019799
11	Ricky Gervais	83	4972	0.016693
3	Bo Burnham	90	3165	0.028436
4	Dave Chappelle	131	4094	0.031998
6	Jim Jefferies	170	4764	0.035684
2	Bill Burr	179	5535	0.032340
7	Joe Rogan	203	4579	0.044333

On voit que les pourcentages sont très variés allant de 0.04% pour Mike à plus de 4% pour Joe Rogan !

---

```
#Let's do a bar diagramm
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
comedian = data_count_insult['comedian']
insults = data_count_insult['insults']
ax.bar(comedian,insults)
plt.title('Number of insults', fontsize=20)
plt.show()
```

---



## 3.6 Techniques d'analyse de texte

Jusque là, l'analyse que nous avons faite de ces données est assez basique, nous avons compté des mots et faits quelques statistiques : des choses que l'on pourrait aussi faire avec des données numériques. Mais il existe des méthodes bien plus intéressantes qui sont propres à l'analyse de texte qui peuvent nous en apprendre bien plus. Nous allons voir 3 de ces techniques.

### 3.6.1 Analyse de sentiments

Commençons tout d'abord par l'analyse des sentiments. Pour cela nous allons utiliser la bibliothèque TextBlob créée par des linguistes qui ont labélisé chaque mot et leur ont donné une polarité (négative ou positive) et une subjectivité (0 pour les faits et 1 pour une opinion).

#### Sentiment général du show

On commence par créer et appliquer les 2 fonctions qui permettant d'obtenir la polarité et la subjectivité :

---

```
import pandas as pd
```

```
data =
```

```
pd.read_pickle('C:/Users/SPML0410/Downloads/nlp-in-python-tutorial-master/nlp-in-python-tutorial-')
```

```

# Create quick lambda functions to find the polarity and subjectivity of
  each routine
# Terminal / Anaconda Navigator: conda install -c conda-forge textblob
from textblob import TextBlob

pol = lambda x: TextBlob(x).sentiment.polarity
sub = lambda x: TextBlob(x).sentiment.subjectivity

data['polarity'] = data['transcript'].apply(pol)
data['subjectivity'] = data['transcript'].apply(sub)

```

---

	transcript	full_name	polarity	subjectivity
ali	Ladies and gentlemen, please welcome to the st...	Ali Wong	0.069359	0.482403
anthony	Thank you. Thank you. Thank you, San Francisco...	Anthony Jeselnik	0.055237	0.558976
bill	[cheers and applause] All right, thank you! Th...	Bill Burr	0.016479	0.537016
bo	Bo What? Old MacDonald had a farm E I E I O An...	Bo Burnham	0.074514	0.539368
dave	This is Dave. He tells dirty jokes for a livin...	Dave Chappelle	-0.002690	0.513958
hasan	[theme music: orchestral hip-hop] [crowd roars...	Hasan Minhaj	0.086856	0.460619
jim	[Car horn honks] [Audience cheering] [Announce...	Jim Jefferies	0.044224	0.523382
joe	[rock music playing] [audience cheering] [anno...	Joe Rogan	0.004968	0.551628
john	All right, Petunia. Wish me luck out there. Yo...	John Mulaney	0.082355	0.484137
louis	Intro\nFade the music out. Let's roll. Hold th...	Louis C.K.	0.056665	0.515796
mike	Wow. Hey, thank you. Thanks. Thank you, guys. ...	Mike Birbiglia	0.092927	0.518476
ricky	Hello. Hello! How you doing? Great. Thank you....	Ricky Gervais	0.066489	0.497313

---

```

# Let's plot the results
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [10, 8]

for index, comedian in enumerate(data.index):
    x = data.polarity.loc[comedian]
    y = data.subjectivity.loc[comedian]
    plt.scatter(x, y, color='blue')
    plt.text(x+.001, y+.001, data['full_name'][index], fontsize=10)
    plt.xlim(-.01, .12)

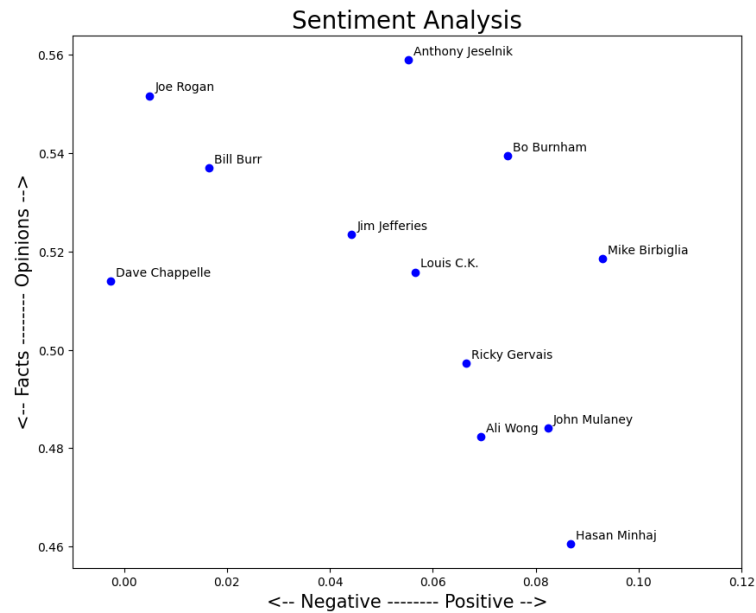
plt.title('Sentiment Analysis', fontsize=20)
plt.xlabel('<-- Negative ----- Positive -->', fontsize=15)
plt.ylabel('<-- Facts ----- Opinions -->', fontsize=15)

```



```
plt.show()
```

---



### Sentiment tout au long de la routine

C'est intéressant d'avoir le sentiment général qui se dégage d'un show mais il peut aussi être intéressant de voir l'évolution sur l'ensemble. S'il y a des périodes plus positives ou négatives.

Pour cela on découpe le texte en  $n=10$  parties et on cherche la polarité sur chaque section pour tous les comédiens.

---

```
# Split each routine into 10 parts
import numpy as np
import math

def split_text(text, n=10):
    '''Takes in a string of text and splits into n equal parts, with a
       default of 10 equal parts.'''

    # Calculate length of text, the size of each chunk of text and the
    # starting points of each chunk of text
    length = len(text)
    size = math.floor(length / n)
    start = np.arange(0, length, size)

    # Pull out equally sized pieces of text and put it into a list
    split_list = []
```

```

    for piece in range(n):
        split_list.append(text[start[piece]:start[piece]+size])
    return split_list

# Let's create a list to hold all of the pieces of text
list_pieces = []
for t in data.transcript:
    split = split_text(t)
    list_pieces.append(split)

# Calculate the polarity for each piece of text
polarity_transcript = []
for lp in list_pieces:
    polarity_piece = []
    for p in lp:
        polarity_piece.append(TextBlob(p).sentiment.polarity)
    polarity_transcript.append(polarity_piece)

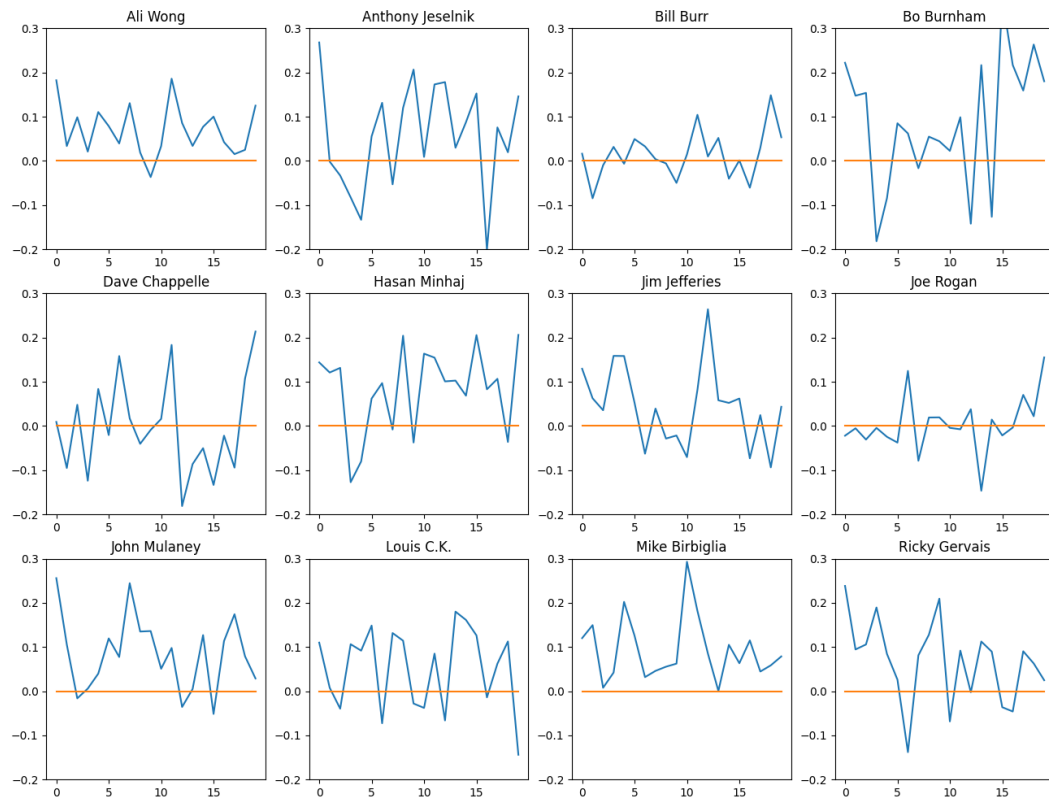
# Show the plot for one comedian
plt.plot(polarity_transcript[0])
plt.title(data['full_name'].index[0])
plt.show()

# Show the plot for all comedians
plt.rcParams['figure.figsize'] = [16, 12]
for index, comedian in enumerate(data.index):
    plt.subplot(3, 4, index+1)
    plt.plot(polarity_transcript[index])
    plt.plot(np.arange(0, len(list_pieces[0]),
        np.zeros(len(list_pieces[0]))))
    plt.title(data['full_name'][index])
    plt.ylim(bottom=-.2, top=.3)

plt.show()

```

---



On peut voir que Mike a un show positive globalement comme Ali mais Dave a des mots considérés comme très négatifs sur la fin de son show.

### 3.6.2 Modélisation des sujets

La deuxième méthode consiste à trouver les sujets abordés de chaque donnée d'un ensemble de données. Pour cela on utilise la méthode "Latent Dirichlet Allocation" (LDA).

Commençons par un exemple avec les données suivantes :

J'aime les bananes; les grenouilles et les poissons vivent dans des étangs; les chatons et les chiots sont poilus; j'ai bu un smoothie épinard pomme; Mon chaton aime le chou.

Naturellement, nous reconnaissons 2 sujets : la nourriture et les animaux. Le but de LDA est de repérer le mix de topic dans chaque donnée et le mix de mots dans chaque topic.

**Comment fonctionne LDA ?**

- L'utilisateur choisi le nombre de sujets qu'il y a sur l'ensemble de ses données
- Assigne aléatoire un sujet à chaque mot
- Regarde chaque mot et son sujet associé de chaque donnée et regarde si le sujet revient souvent dans le donnée et si le mot revient souvent dans le mix du sujet. Si pour les deux c'est plutôt rare alors le mot est associé à un nouveau sujet.
- On itère jusqu'à stabilisation

Cependant, certains mots ne sont pas pertinents. En utilisant LDA sur tous les mots de toutes les données les résultats seront peu pertinents et donc impossible à utiliser.

Pour palier à cette dérive on peut utiliser la bibliothèque NLTK et ne traiter que certains types de mots (verbes, noms, adjectifs...). Voici un exemple où l'on ne s'intéresse qu'aux noms et aux adjectifs.

---

```
# Let's read in our document-term matrix
import pandas as pd

data =
    pd.read_pickle('C:/Users/SPML0410/Downloads/nlp-in-python-tutorial-master/nlp-in-python-tutorial-

# Let's create a function to pull out nouns from a string of text
from nltk import word_tokenize, pos_tag

# Let's create a function to pull out nouns from a string of text
def nouns_adj(text):
    '''Given a string of text, tokenize the text and pull out only the
        nouns and adjectives.'''
    is_noun_adj = lambda pos: pos[:2] == 'NN' or pos[:2] == 'JJ'
    tokenized = word_tokenize(text)
    nouns_adj = [word for (word, pos) in pos_tag(tokenized) if
                  is_noun_adj(pos)]
    return ' '.join(nouns_adj)

# Apply the nouns function to the transcripts to filter only on nouns
data_nouns_adj = pd.DataFrame(data_clean.transcript.apply(nouns_adj))
data_nouns_adj

# Create a new document-term matrix using only nouns and adjectives,
    also remove common words with max_df
cvna = CountVectorizer(stop_words=stop_words, max_df=.8)
data_cvna = cvna.fit_transform(data_nouns_adj.transcript)
```

```

data_dtmna = pd.DataFrame(data_cvna.toarray(),
                           columns=cvna.get_feature_names())
data_dtmna.index = data_nouns_adj.index
data_dtmna

# Create the gensim corpus
corpusna =
    matutils.Sparse2Corpus(scipy.sparse.csr_matrix(data_dtmna.transpose()))

# Create the vocabulary dictionary
id2wordna = dict((v, k) for k, v in cvna.vocabulary_.items())
n=4
# Let's start with n topics
ldana = models.LdaModel(corpus=corpusna, num_topics=n,
                        id2word=id2wordna, passes=10)
ldana.print_topics()

```

---

Et finalement le rendu de LDA se fait sous cette forme :

```

[(0,
  '0.009*"joke" + 0.005*"mom" + 0.005*"parents" + 0.004*"hasan" + 0.004*"jokes" + 0.004*"anthony" + 0.003*"nuts" + 0.003*"dead"
  + 0.003*"tit" + 0.003*"twitter"'),
 (1,
  '0.005*"mom" + 0.005*"jenny" + 0.005*"clinton" + 0.004*"friend" + 0.004*"parents" + 0.003*"husband" + 0.003*"cow" + 0.003*"o
  k" + 0.003*"wife" + 0.003*"john"'),
 (2,
  '0.005*"bo" + 0.005*"gun" + 0.005*"guns" + 0.005*"repeat" + 0.004*"um" + 0.004*"ass" + 0.004*"eye" + 0.004*"contact" + 0.003
  *"son" + 0.003*"class"'),
 (3,
  '0.006*"ahah" + 0.004*"nigga" + 0.004*"gay" + 0.003*"dick" + 0.003*"door" + 0.003*"young" + 0.003*"motherfucker" + 0.003*"stu
  pid" + 0.003*"bitch" + 0.003*"mad"')]

```

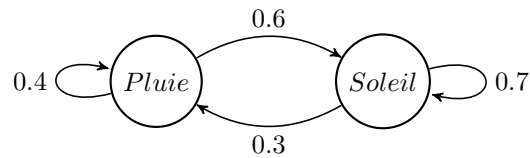
Cette méthode est intéressante mais elle a plusieurs désavantages :

- Le choix par l'utilisateur du nombre de sujet : on devrait sûrement faire plusieurs essais pour trouver le bon nombre
- Choix des classes grammaticales étudiées : il faudra peut-être en enlever et en ajouter pour arriver à l'optimum

### 3.6.3 Génération de texte

Cette méthode se base sur les chaînes de Markov. C'est une suite de variables aléatoires à valeurs dans un espace probabilisé (E,P) où E est l'ensemble des états et P les probabilités. Faisons un exemple avec la météo.

Soit  $E = \{\text{pluie}; \text{soleil}\}$  et  $P = \{V_{\text{pluie}, \text{pluie}}, V_{\text{pluie}, \text{soleil}}, V_{\text{soleil}, \text{pluie}}, V_{\text{soleil}, \text{soleil}}\} = \{0.4, 0.6, 0.3, 0.7\}$ . Cette chaîne de Markov correspond au graphique suivant :



L'objectif est de donner la loi de probabilité d'un état en partant du principe que cet état ne dépend que des états précédents.

Pour l'analyse de texte les états sont des mots et le but est de prédire le mot qui viendra après à l'aide des statistiques du texte. Bien sûr, cette approche n'est pas parfaite et ne fera pas toujours des phrases grammaticalement correctes mais c'est déjà une première approche.

---

```

# Read in the corpus, including punctuation!
import pandas as pd

data = pd.read_pickle('corpus.pkl')

# Extract only Ali Wong's text
ali_text = data.transcript.loc['ali']
ali_text[:200]

from collections import defaultdict

def markov_chain(text):
    '''The input is a string of text and the output will be a dictionary
    with each word as
    a key and each value as the list of words that come after the key
    in the text.'''

    # Tokenize the text by word, though including punctuation
    words = text.split(' ')

    # Initialize a default dictionary to hold all of the words and next
    words
    m_dict = defaultdict(list)

    # Create a zipped list of all of the word pairs and put them in
    word: list of next words format
    for current_word, next_word in zip(words[0:-1], words[1:]):
        m_dict[current_word].append(next_word)

    # Convert the default dict back into a dictionary
    m_dict = dict(m_dict)
    return m_dict

# Create the dictionary for Ali's routine, take a look at it
ali_dict = markov_chain(ali_text)
  
```

```

import random
def generate_sentence(chain, count=15):
    '''Input a dictionary in the format of key = current word, value =
        list of next words
        along with the number of words you would like to see in your
        generated sentence.'''

    # Capitalize the first word
    word1 = random.choice(list(chain.keys()))
    sentence = word1.capitalize()

    # Generate the second word from the value list. Set the new word as
    the first word. Repeat.
    for i in range(count-1):
        word2 = random.choice(chain[word1])
        word1 = word2
        sentence += ' ' + word2

    # End it with a period
    sentence += '.'
    return(sentence)

for i in range(0,10):
    print(generate_sentence(ali_dict))
    print("-----")

```

---

```

Same kind that's why I'm a Asian men are shocked by, because, usually, Asian-American women.
-----
Juicy. You just want their- Glasses always wanted. I'm the stage: Ali Wong. Have a.
-----
Know?
my last boyfriend because I was sliced mango. Know what I'm getting older, because the.
-----
Much. For 30 days. He proposed!" "It came out first. You'll get to all your.
-----
Girl, when it gets very distracting for me. When- When I say whatever you know.
-----
Fucking baby. That baby's a ghost that came out about how much resentment, especially when.
-----
Also takes so emotional. We had a doula is? You owe me out. They have.
-----
Yeah. Where I'm done.'" I'm gonna become this left hand down your own race. The.
-----
Scratch yourself, all of one. I respect you, boom, a miscarriage. It's all day since..
-----
Restaurant deaf and I used to see the third grade, 'cause you're 33, you'll know.
-----

```

---

Les résultats sont loin d'être parfaits, les phrases ne font pas vraiment de sens mais c'est déjà un premier pas !

## 4 Introduction au langage SQL : Structured Query Language

Toutes les applications et sites internet manipulent des données : nom, adresse, numéro de compte, âge, amis etc... Le langage SQL permet de manipuler automatiquement ces larges quantités de données sous forme de table (comment un document Excel) avec dans chaque ligne un donnée et chaque colonne des propriétés sur les données.

Il est assez lourd d'avoir de grosses tables avec toutes les informations de tous les utilisateurs. Pour cela on travaille sur des "relational database" qui sont des tables faites pour être mise en lien avec d'autres tables. Une table avec les informations d'un utilisateur du site Khan Academy, une table avec les informations des badges de chaque utilisateurs et qui pourraient être mise en relation avec la première table pour faire une 3eme table.

### 4.1 Selectionner de données

Comment créer une table ? :

---

```
CREAT TABLE groceries (id INTEGER PRIMARY KEY, name TEXT, quantity
    INTEGER,aisle INTEGER);
INSERT INTO groceries VALUES (1,"Banana",4,1);
INSERT INTO groceries VALUES (2,"Peanut",10,12);
#Rempli automatiquement les colonnes non specifiees
INSERT INTO groceries(name,quantity,aisle) VALUES ("Cherry",90,2);
```

---

Comment récupérer des données :

---

```
#Recup tout
SELECT * FROM groceries;
#Recup dans l'ordre
SELECT * FROM groceries ORDER BY quantity;
#Recup avec condition
SELECT * FROM groceries WHERE quantity > 5 AND (quantity < 12 OR aisle
    <2);
```

---

Comment faire des calculs simples comme compter le nombre d'éléments ?

---

```
#Somme des toutes les quantites
SELECT SUM(quantity) FROM groceries;
#La quantite de chaque ingredient pour chaque allee
SELECT aisle, SUM(quantity) FROM groceries GROUP BY aisle;
```

---

L'opérateur IN et NOT

---



```

SELECT * FROM exercise_logs WHERE type IN ("biking","rowing",tree
      climbing");
#Au lieu de
SELECT * FROM exercise_logs WHERE type = "biking" OR type = "rowing" or
      type = "climbing";
#NOT IN
SELECT * FROM exercise_logs WHERE type NOT IN ("biking","rowing",tree
      climbing");

```

---

Sélection à partir de plusieurs tables : sub-query

```

#Table 1 exercise_logs
#Table 2 dr_favorites
SELECT * FROM exercise_logs WHERE type IN (SELECT type FROM
      dr_favorites);

```

---

L'opérateur LIKE : sélectionner si l'élément apparaît dans le String.

```

SELECT type FROM exercise_logs WHERE reason LIKE "%cardiovascular%"

```

---

Changer le nom d'une colonne

```

SELECT type, SUM(calories) AS total_calories FROM exercise_logs GROUP BY
      type;

```

---

L'opérateur HAVING : la condition se fait sur le groupe et non pas sur chaque donnée.

```

SELECT type, SUM(calories) AS total_calories FROM exercise_logs GROUP BY
      type HAVING total_calories > 150;

```

---

L'opérateur COUNT

```

SELECT type FROM exercise_logs GROUP BY type HAVING COUNT(*) > 2
#rend les activites groupees et seulement celles qu'elle a fait plus de
      2 fois.
#le * indique qu'on regarde tous

```

---

L'opérateur CASE

```

SELECT type, heart_rate,
      CASE
        WHEN heart_rate > 220 - 30 THEN "above_max"
        WHEN heart_rate > ROUND(0.9 * (220-30)) THEN "above_target"
        WHEN heart_rate > ROUND(0.5 * (220-30)) THEN "within target"
        ELSE "below target"
      END as "hr_zone"
FROM exercise_logs;

```

---

Comment rejoindre des données à partir de plusieurs tables ?

---

```
#Cross Join
SELECT * FROM student_grades, students;

#implicit inner join
SELECT * FROM student_grades, students WHERE student_grades.student_id =
    students.id;

#explicit inner join
SELECT students.first_name,
    students.last_name,email,student_grades.test,student_grades.grade
FROM students JOIN student_grades ON students.id =
    student_grades.student_id;

#outer join : permet d ajouter meme si c'est Null
SELECT students.frist_name, students.last_name, student_projects.title
FROM students LEFT OUTER JOIN student_projects ON students.id =
    student_projects.student_id;
```

---

Croiser une table avec elle même

---

```
SELECT students.first_name, students.last_name, students.email,
    buddies.first_name as "buddy_name", FROM students JOIN students
    buddies;
```

---

## 4.2 Modifier les données

La commande UPDATE

---

```
UPDATE diary_logs SET content = "It was an awful day !" WHERE id = 1;
```

---

La commande DELETE

---

```
DELETE FROM diary_logs WHERE id = 1;
```

---

Ajouter une colonne

---

```
ALTER TABLE diary_logs ADD emotion TEXT default "unknown";
```

---

La commande DROP

---

```
DROP TABLE diary_logs
```

---