Sommaire

IA & Big Data

- Le Big Data
- Machine Learning
- Projet : Prix des locations

Le Big Data

Intelligence artificielle

- *Ensemble de théories et de techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence.*
- *Une branche de l'informatique en interaction avec plusieurs disciplines : Logique, Recherche opérationnelle, Neurosciences, Linguistique, Vision, Probabilités, Statistiques, Théorie des jeux, Heuristiques.*
- *Un aspect fondamental de l'IA est la conception de méthodes d'apprentissage, d'adaptation ou de reconnaissance qui permettent à un système d'améliorer ses performances.*
- *Afin de faire ressembler la machine à un homme deux grandes catégories d'outils sont développés permettant :*
  - *la reconnaissance : Image (écriture, pilotage automatique, vision …), parole, traitement du langage des signaux…*
  - *l'extraction de connaissances (Data Mining) : Aide à la décision (banque, finance, médical, militaire…), planification (commerce), classement, robotique ….*

Le Big Data

Les mégadonnées

- *Combinaison de données structurées (BdD, fichiers, images, ….) ou non structurées (SMS, données issues de capteurs, clicks …)*
- *Une partie des données que nous produisons sont stockées (ExaOctets par jour), et ce volume va considérablement augmenter dues à des productions automatisées (cartes bancaires, caméras, capteurs…).*
- *Ces données sont stockées sur des serveurs reliés par un réseau « cloud computing », puis exploités dans le but de produire de la connaissance, mais surtout d'en extraire ou de faire émerger les informations qui se cacheraient dans ces documents.*
- *Les mégadonnées sont caractérisées par :*
  - *le volume : Téra (10^{12}), Penta (10^{15}), ou ExaOctets (10^{18})*
  - *la variété : BD(SQL), Fichiers (NoSQL), Journaux serveurs web …*
  - *la vélocité : vitesse à laquelle les données sont traitées (requette)*
  - *la véracité : validité des données (réseaux sociaux, pages web…)*
  - *la valeur ou la variabilité*

Les mégadonnées

- *La gestion d'une architecture de Big Data, nécessite des compétences différentes de celles des administrateurs de Bases de données.*
- *La gestion du Cloud permet de gérer ces données mais demande aux administrateurs de surveiller son évolution et d'assurer les migrations.*
- *Les administrateurs doivent également rendre accessibles les données en particulier si elles sont variées et distribuées sur différentes plates-formes.*
- *Des fonctions de gestion (qualité, cohérence, néttoyage…) et de l'ajout de métadonnées (origine, suivit temporel…) sont généralement ajouté pour en faciliter l'exploitation.*
- *Les lois sur la protection des données (RGPD) imposent la mise en place de procédures spécifiques.*
- *Bien que l'on ne puisse pas donner de définition précise de ce que peut être le Big Data on peut tout de même considérer qu'il correspond à un concept désignant un immense entrepôt de données ainsi que les technologies permettant de l'alimenter et de l'exploiter.*

Le Big Data

Cloud computing

Producteurs

Consommateurs

Analyser

Exploiter

Machine Learning

Apprentissage

- *Dans l'apprentissage supervisé, le plus courant, les données sont étiquetées et le modèle sait ce qu'il doit chercher (Target).*
- *Les modèles chercher à « clusteriser » les données.*
- *Les méthodes évolutionnaires sont fondées sur l'évolution d'une population à partir de deux types d'opération : sélection, croisement.*
- *Dans l'apprentissage par renforcement des essais sont récompensés ou pénalisés en fonction des résultats obtenus.*

Apprentissage	Supervisé	Non-Supervisé	Evolutionnaire	Renforcement
Qu'est-ce qu'on apprend	Relations	Structures	L'optimum d'une fonction	Lois d'action
Quelles informations	Sorties désirées		Une fonction	Récompenses
Les formes d'apprentissage	Par instruction	Par observation	Par évolution et sélection	Par évaluation
Les lois d'apprentissage	Gradient ou Distance	Auto-organisation	Stochastique	Différences temporelles

Machine Learning

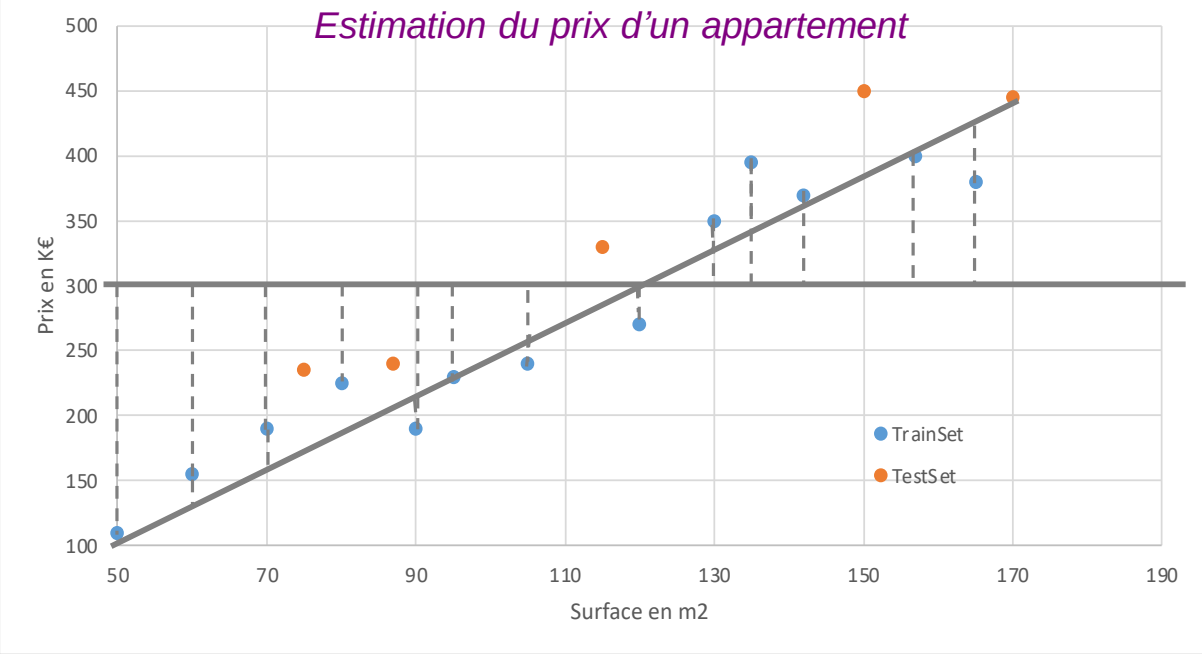
Apprentissage

- *Les algorithmes d'apprentissage automatique recherche des schémas récurrents (modèles) dans l'entreprôt des données.*
- *Les algorithmes sont ensuite en mesure d'apprendre à partir de ces modèles, et peuvent réaliser certaines tâches sans avoir été programmé.*
- *Ces algorithmes utilisent les statistiques, les probabilités et des fonction d'optimisation (descente de gradient) pour trouver et paramétrer les modèles.*
- *Le principe du machine learning consiste donc à trouver un modèle en exploitant une grande quantité de données, puis à appliquer ce modèle pour prédire ce qui pourrait se passer avec de nouvelles données.*
- *Le Deep Learning est une technique d'apprentissage basée sur des couches de neurones qui collaborent pour extraire des informations et faire des prédictions.*
- *Contrairement au machine learning, le Deep Learning est en capacité de travailler sur des données non structurées.*

Machine Learning

Principe de base

- *Les données sont utilisées pour développer des modèles, qui seront ensuite capable de faire des prédictions.*
- *On utilise une partie des données (TrainSet) pour « caler » les paramètres du modèle et une partie (TestSet) pour le valider.*



A partir d'un modèle de type
Prix = ax+b
(x surface – a et b paramètres)

Le but est de trouver a et b qui minimisent les erreurs sur le TrainSet

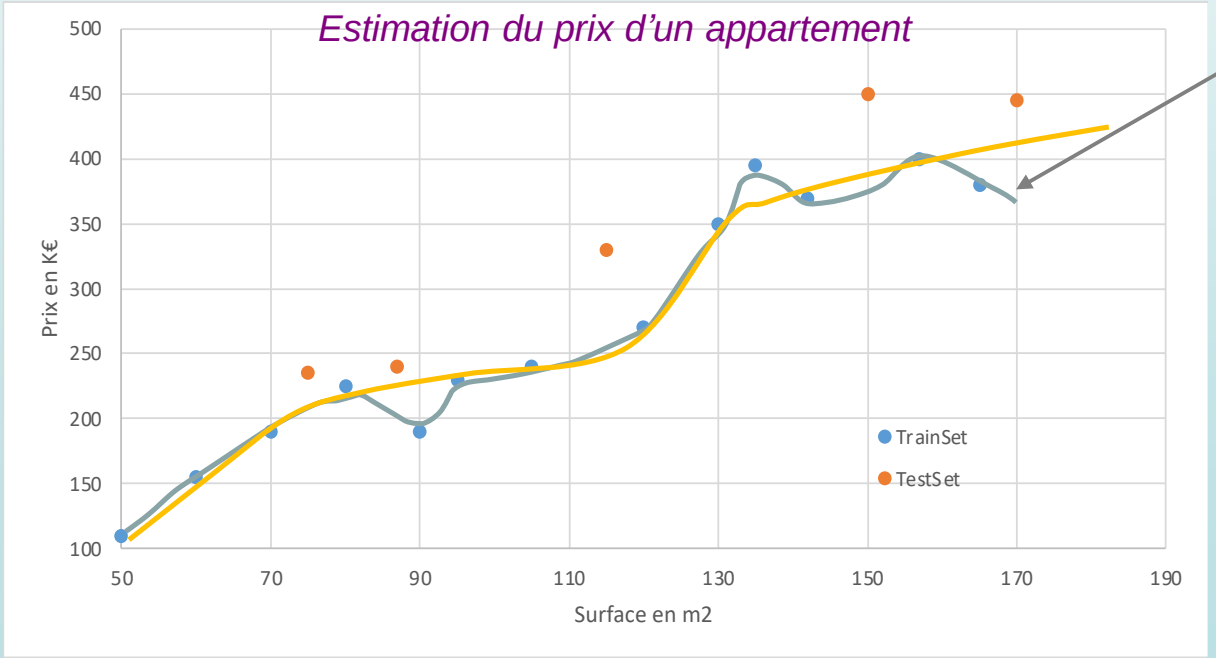
Pour a=0 et b=300 on a des erreurs (190, 145, 110 …, 100, 80) = 1185

Pour a=2,9 et b=-45.8 on a (10, 25.8, 110 …, 12, 55.4) = 320.4

Machine Learning

Principe de base

- *On développe un algorithme d'optimisation qui minimise la somme des erreurs commises par le modèle sur les données du TrainSet.*
- *Si le modèle est plus complexe (équation du second degré) l'erreur sera réduite et tend vers 0 lorsque le degré de l'équation augmente.*



Equation de degré 13

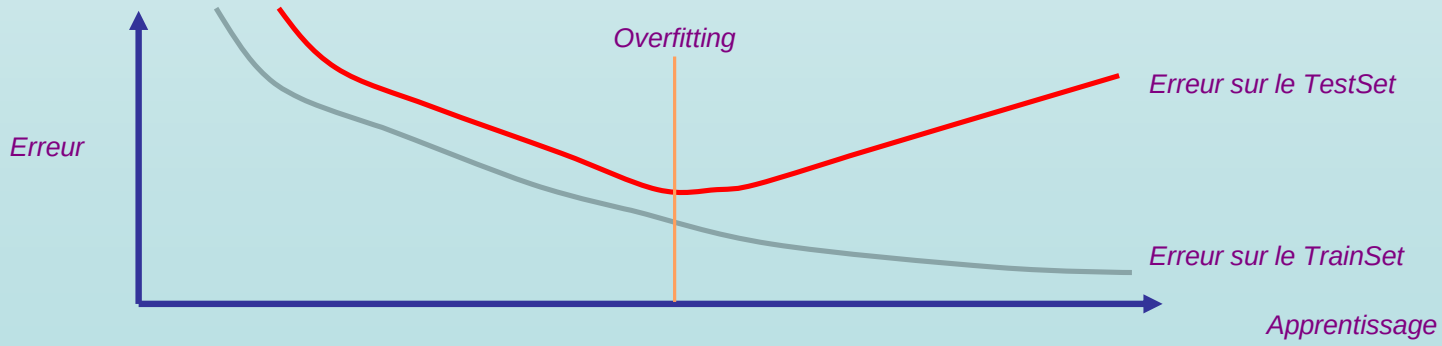
Le modèle à bien appris sur le TrainSet, mais les erreurs commises par rapport à un modèle de degré 4 sur le TestSet est plus grande

Overfitting

Machine Learning

Overfitting

- *Les principales causes de mauvaises performances des modèles en Machine Learning sont dues à du sur ou du sous apprentissage (Overfitting et Underfitting).*
- *L'overfitting se produit lorsqu'un modèle s'adapte très bien au Train Set, il a appris les corrélations générales du modèle mais également le "bruit" dû aux imprécisions sur les données.*
- *Le modèle est alors parfaitement adapté aux données du Train Set mais prédira très mal les sorites des données non encore vues.*



Machine Learning

Les distances

- *La distance (ou erreur) entre deux observations (samples) joue un rôle important en Machine Learning. Elle permet de mesure la différence relative qui peut exister entre les observations.*
- *Les distances se mesures en comparant les features (variables) des observations entre elles.*
- *C'est en minimisant ces distances que la plupart des algorithmes de machine learning fonctionnent.*
- *Il existe plusieurs distances : Euclidienne, Manhattan, Tchebychev …*

Si A et B sont deux observations avec n features

Distance Euclidienne

$$\sqrt{\sum_{i=1}^{n}\left(x_i^A - x_i^B\right)^2}$$

Distance Manhattan

$$\sum_{i=1}^{n}\left|x_i^A - x_i^B\right|$$

Distance Tchebychev

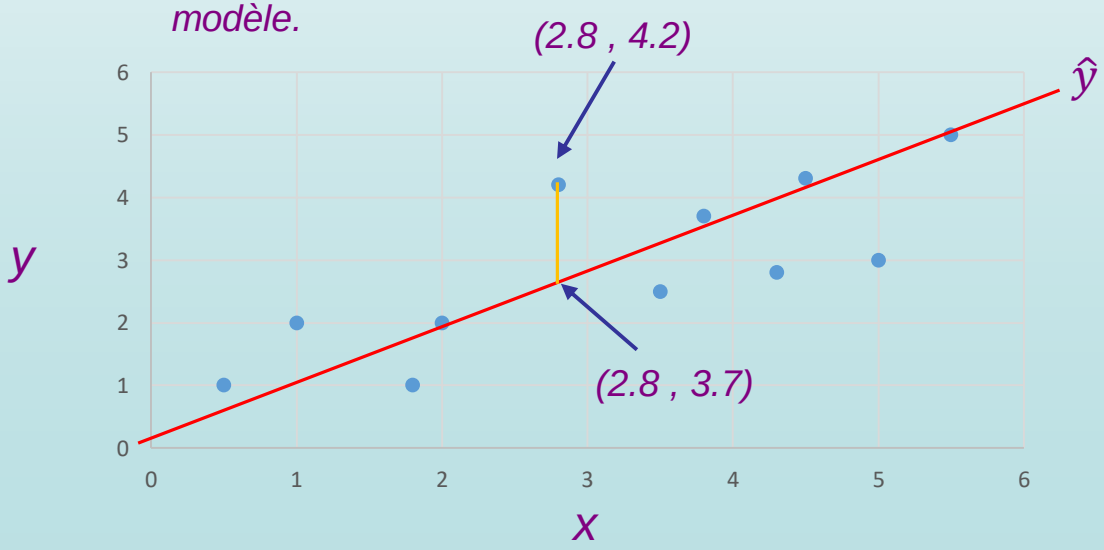
$$sup_{1\leq i\leq n}\left|x_i^A - x_i^B\right|$$

Machine Learning

Les métriques

- *Les métriques permettent d'évaluer le niveau de la prédiction d'un modèle et donner une idée plus ou moins précise de sa qualité.*
- *Pour évaluer la performance des modèles de régression par exemple on mesure la distance qui sépare la prédiction effectuée de la valeur réelle.*
- *Pour un modèle de classification on compte plutôt le nombre d'erreurs.*

Les distances entre les positions réelles et estimées permettent d'évaluer la qualité du modèle.



Erreur Absolue moyenne

$$MAE(Y) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Erreur Quadratique moyenne

$$MSE(Y) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Erreur Absolue médiane

$$MAD(Y) = mediane(|y_i - \hat{y}_i|)$$

Machine Learning

Les métriques

- *On utilise généralement la MSE si l'on souhaite accorder plus importance aux grandes erreurs.*
- *La MAE sera utilisée si l'on accorde la même importance aux erreurs.*
- *La MSE est beaucoup plus sensible aux valeurs aberrantes (outliers). Aussi si les erreurs sont provoquées par des outliers on utilisera la MAE.*
- *Lors de l'évaluation (scoring) d'un modèle de régression, la fonction score() scikit-learn utilise un coefficient de détermination appelé R^2.*
- *R2 mesure l'adéquation entre le modèle et les données observées ou à quel point le modèle de régression est adapté pour décrire les Targets.*

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

y_i valeurs de la target
$\hat{y}_i$ valeurs calculées par le modèle
$\bar{y}$ la moyenne

Plus R^2 est proche de 1 et plus le modèle est performant. Ce coefficient évalue le niveau de performance du modèle par rapport aux variations des données

Machine Learning

Descente de Gradient

- *L'algorithme de descente du gradient est un algorithme d'optimisation, qui permet de trouver le minimum de n'importe qu'elle fonction convexe.*
- *Une fonction f() définie sur un intervalle I est convexe (un seul minima local) si pour tout x et y de I et pour tout t ∈ [0,1] on a :*

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$$

- *En machine learning ou en deep learning cet algorithme est utilisé pour obtenir le minimum de la fonction coût (erreur quadratique).*

f est un modèle β les paramètres du modèle, m les différentes samples et y les targets associées

Fonction coût ou Erreur quadratique

$$E(f(\beta))\frac{1}{2m}\sum_{i=1}^{m}(f(\beta, x^m) - y^m)^2$$

Cette fonction est convexe et n'a qu'un seul minima. L'algorithme de descente du gradient permet de trouver les valeurs des paramètres β qui minimisent l'erreur.

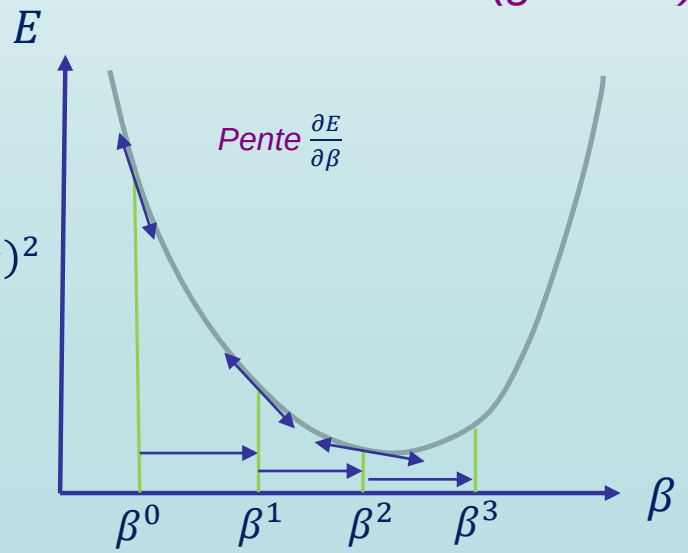
Machine Learning

Algorithme de la descente du gradient

- *La majorité des modèles d'apprentissage repose sur l'optimisation d'une fonction de coût (erreur quadratique).*
- *L'algorithme de descente de gradient est un des nombreux moyens qui permettent de trouver le minimum (ou maximum) d'une fonction.*
- *La formule générale est la suivante :* $\beta^{i+1} = \beta^i - \eta \nabla E(f(\beta^i, x))$
- η *est le taux d'apprentissage,* β *les paramètres de la fonction f (model) à optimiser et* ∇ *la matrice des dérivées (gradient).*

Erreur quadratique convexe

$$E(f(\beta)) = \frac{1}{2m}\sum_{i=1}^{m}(f(\beta, x^m) - y^m)^2$$



En faisant évoluer les valeurs de β *dans la direction inverse des dérivées on diminue progressivement les valeurs de l'erreur quadratique.*

Machine Learning

Projet d'apprentissage

- *Un projet d'apprentissage passe par un certain nombre d'étapes.*
- *(1) La première étape concerne l'importation des données ainsi que la création de structures informatiques permettant de les manipuler.*
- *(2) Le prétraitement est une étape très importante, car il est impossible pour les algorithmes d'utiliser des données brutes, issues de la collecte.*
- *Cette étape inclue des opérations telles que le nettoyage, le traitement des données incomplètes ou erronées, l'analyse des features, l'encodage des données catégorielles, la normalisation, la réduction de dimensionnalité…*
- *(3) La création de jeux de données pour l'entrainement (TrainSet) et le test (TestSet) pour vérifier la validité de notre modèle.*
- *(4) Déterminer le bon modèle et optimiser les hyperparamètres du modèle choisi.*
- *(5) Entrainer et évaluer le modèle puis en fonction des résultats obtenus déployer l'application ou éventuellement retourner à l'étape (3) voir (2).*

Machine Learning

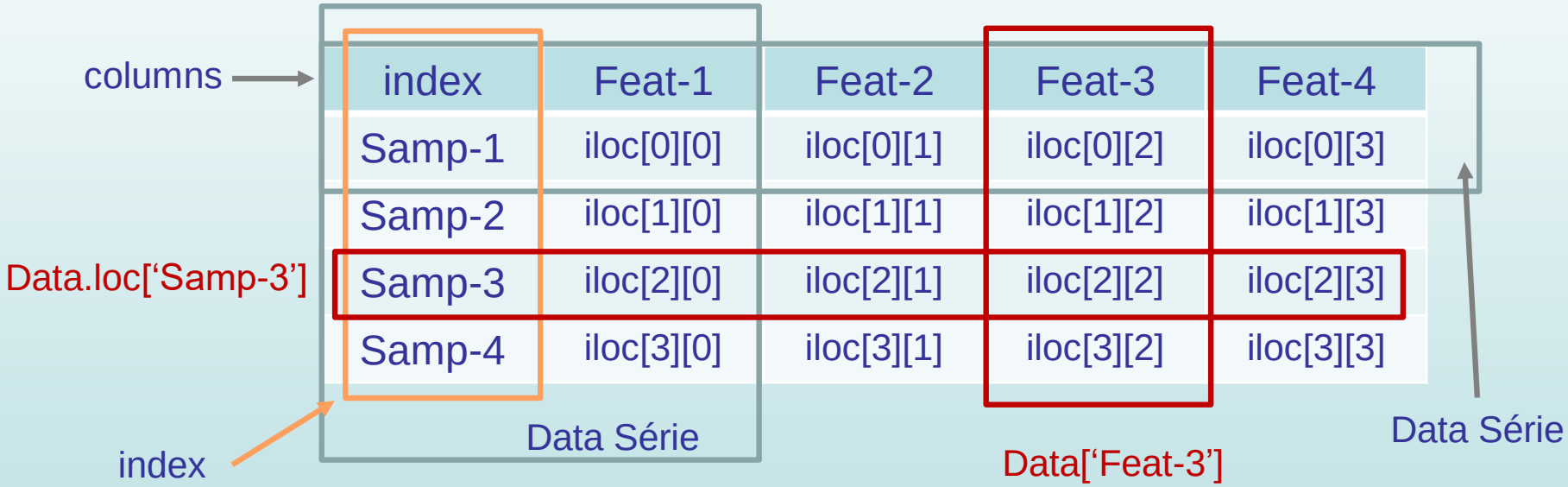
Les libraires python

- *Les programmes de machine learning en python manipules des jeux de données (DataSet) au travers de matrices et de vecteurs.*
- *Les observations sont enregistrées dans des tableaux en ligne ou les colonnes sont traitées comme étant les variables (ou Features).*
- *Parmi ces colonnes une peut être identifiée comme la Target dans le cas des approches par apprentissage supervisé.*
- ***Numpy** est une bibliothèque qui permet de manipuler des matrices ou des tableaux multi-dimensionnels et d'effectuer des opérations mathématiques et statistiques.*
- *Les données des matrices Numpy sont de même type.*
- ***Pandas** est une librairie spécialisée dans l'analyse des données, elle manipule des objets de type (principe d'Excel).*
- *Une DataFrame est une matrice individus-variables où les lignes sont des observations et les colonnes des attributs décrivant les individus.*

Machine Learning

Les libraires python

- *Dans les librairies Pandas les individus et les variables sont indexés.*



- ***MatplotLib*** *outil de Data Visualisation permet de produire des graphes.*
- ***Seaborn*** *(basée sur MatplotLib) est un outil de visualisation simplifié parfaitement intrégé avec les structures de type Pandas*

Machine Learning

Bibliothèque Pandas

```
import pandas as pd
# Lecture d'un fichier csv, données délimitées par ;
Data = pd.read_csv('file.csv', delimiter=";")
# Lecture d'un fichier de txt, séparateur tab
Data = pd.read_table( 'file.txt', sep = '\t', header = 0)
# Accès aux colonnes, types, info
Data.columns ; Data.dtypes ; Data.info()
# Accès aux indexes
Data.index
# Accès à la colonne spécifié
Data['attribut'] ; Data.attribut ; Data[['C1', 'C2' …]]
# Accès Colonne , un élément
Data.loc[0] ; Data.iloc[0][0] ; DataFr.loc[
# Restriction aux données qui ont la column à val
Data.loc[ DatLoc['Column']=val, : ]
# Trie du tableau
Data.sort_values(by= 'Nom d'une colonne')
# Groupement par critères, partitionne la dataframe
# Critère : une feature catégorielle ou une opération
Data.groupby(['Columns',..])
```

```
import matplotlib.pyplot as plt
Data.hist(column='Column')
# Affichage de la densité
Data['Column'].plot.kde()
# Affichage combiné
Data.hist(column='Ci' , by= value)
# Nuage de points
Data.plot.scatter(x='varaiable1',y='variable2')
# Ajout d'une colonne
Data['name'] = "Un tableau de données »
# Création à partir d'un tableau
Data = pd.DataFrame({ 'Nom1' : Tab ;
                      'Nom2' : Tab2 …})
```

Machine Learning

Bibliothèque Pandas

- *La première tâche dans le cas d'un projet d'apprentissage consiste à analyser les données du DataSet.*
- *La plupart des bibliothèques (Pandas, Scikit-Learn …) disposent de jeux de données, qui peuvent être lues suivant différents formats (csv, xlsx …)*
- *kaggle.com est un site qui propose des projets et des DataSet.*
- *Pandas est une bibliothèque, adaptée pour le Big Data, qui permet de réaliser des traitements sur les données (DataFrame), utilisées par Matplotlib, l'analyse statistique en SciPy, ou par Scikit-learn.*

Principales méthodes et fonctions la la bibliothèque Pandas

```
Data.shape()        # dimension du dataFrame (Data)
Data.dtypes.        # type des features
Data.columns        # liste des features
Data.info()         # information, type, null
Data.describe()     # description statistique simple
Data.duplicated()   # identifie les duplications
Data.where()        # recherche
Data.isna() ; Data.isnull()      # données nan ou null
```

```
Data.drop()         # Elimine des colonnes
# les lignes (données manquantes, dupliquées)
Data.dropna(axis=0)    ; Data.drop_duplicated()
Data.value_counts()     # compte les répétitions
Data.groupby()      # Analyse par groupe
Data.corr()         # analyse les corrélations
# opérations statistiques
Data.sum() ; Data.mean() ; Data.median()  …
```

Machine Learning

Bibliothèque Pandas

- *Chargement des données.*

```
import pandas as pd
import seaborn as sns
import numpy as np
Data = pd.read_csv('Jobs salaries.csv', delimiter=",")        # Lecture des données
Data.drop(['Unnamed: 0', 'salary'], axis=1, inplace=True)    # Suppression de colonnes (axe : 1)
```

- *La méthode groupby permet de regrouper les données selon les catégories d'afin d'appliquer une fonction. Les DataFrame peuvent être divisés par rapport a n'importe quel axes.*

```
Data.groupby(['employment_type']).mean()                    # moyennes sur les données numériques
Data.groupby(['employment_type'])['salary_in_usd'].mean()                # salaire moyen
Data.groupby(['employment_type','year_work])['salary_in_usd'].mean() # par type et année
```

- *Il est possible d'appliquer (apply, transform) une fonction à une DataFrame afin de modifier les valeurs dans une DataFrame.*

```
Data['year_work'] = Data.apply(['year_work']).apply(lambda x : x+1)    # Ajoute 1 a l'année
def min( ) : return x-1
Data['year_work'] = Data.apply(['year_work']).apply(min)    # Supprime 1
```

Machine Learning

Bibliothèque Pandas

- *Le boolean indexing permet de sélectionner des données dans une DataFrame à partir d'un masque de booléens.*
- *Le boolean indexing, passe par une dataframe qui contient des valeurs booléennes (True ou False), via des tests booléens sur les données.*

```
# Création d'un masque : Tableau de booleans ou seul les colonnes d'indexe correspondant au type de contrat PT sont à True
(Data['employment_type']=='PT')
(Data['employment_type']=='PT') & (Data['work_year']==2020)     # Combinaison de conditions
```

- *Les résultats des tests permettent d'effectuer des opérations sur les DataFrame.*

```
Data[Data['employment_type']!='FT'].index      # Index des contrats non FT
Data[Data['employment_type']!='FT'] ['salary_in_usd'].mean   # moyenne des salaires non FT
Data[~Data['employment_type']!='FT'] ['salary_in_usd'].mean   # moyenne des salaires FT
# Suppression de toutes les données non FT
Data.drop(Data[Data['employment_type']!='FT'].index, inplace=True)
```

Machine Learning

Bibliothèque Scikit Learn

- *Est une librairie d'apprentissage automatique. Elle contient tous les algorithmes du machine learning.*
- *Scikit-Learn s'utilise conjointement avec : Numpy, Pandas ou Matplotlib.*

#Algorithmes pour la classification
Perceptron multicouche (réseau de neurones)
Machines à vecteur de support (SVM)
K plus proches voisins (KNN)
Arbres de décision (Decision Tree)
Régression linéaire (Linear Regression)
Régression logistique (Logistic Regression)

Classification naïve bayésienne (Naive Bayes)
#Algorithmes pour la clustering
Partition en k moyennes (K-means clustering)
Regroupement hiérarchique (CAH)
Processus Gaussien de classification (GPC)
Processus Gaussien de régression (GPR)
Réduction de dimension

- *Scikit-learn fourni des modules pour préparer les données (séparer, normaliser les données ou traiter les valeurs manquantes, …).*
- *Scikit-Learn contient une bibliothèque pour l'analyse des modèles comme la cross validation, le réglage des hyper-paramètres …*
- *Elle permet également de mesurer la qualité d'un modèle, le chargement de DataSet ou la génération de nouveaux DataSet.*

Machine Learning

Bibliothèque Scikit Learn

- *Les données*

```
# Traitement des données sous la forme de matrices
X : Tableau numpy (array) ou pandas (dataFrame) 2D de taille (m samples)*(n features)
# Pour l'apprentissage supervisé : Traitement de la target sous forme d'une martice
y :  Tableau numpy (array) ou pandas (dataFrame) 1D de taille (m samples)*(1 feature)
```

- *L'apprentissage*

```
# Création d'un estimateur pour le traitement des données
model = 'estimateur.scikit-learn' ( 'hyper-paramètres')
model.fit (X, y)          # Entrainement du modèle pour l'apprentissage supervisé
model.fit (X)             # Entrainement du modèle pour l'apprentissage non supervisé
```

- *Prédiction*

```
ypred = model.predict(X)
```

- *Evaluation*

```
model.score(X, y)                    # Evaluation du score du modele
sklearn.model_selection.GridSearchCV # Calage des hyper-paramètres
```

Pre-processing

Démarche de traitement

- *Etape 1 : Analyse globale des données :*
  - *Chargement des données, taille du dataSet (samples, deatures)*
  - *Problème : supervisé (régression ou classification), non supervisé.*
  - *Informations générales sur les différentes features : données manquantes, statistiques, …*
  - *Données catégorielles : Différentes valeurs*
- *Etape 2 : Nettoyage des données :*
  - *Traitement des données manquantes*
  - *Samples : Analyse des outliers, Détection d'anomalies*
  - *Features : Suppression de features. Réduction de features*
- *Etape 3 : La transformation des données :*
  - *Traitement des données manquantes*
  - *Les imputations via scikit-learn, clusterisation des données*
  - *Encodage des données catégorielles*

Pre-processing

Analyse globale : Accès aux données

- *Lecture d'un jeu données (prix airbnb) via la bibliothèque Pandas.*

```
# Choix d'un DataSet sur le prix Airbnb – Téléchargement au format csv
import pandas as pd
Data = pd.read_csv('Airbnb prediction.csv', delimiter=",")     # DataSet de type DatFrame
```

- *Taille du DataSet et type des features*

```
Data.shape                    # (74111 lignes, 28 colonnes)
Data.dtypes                   # id int64, log_price float64, property_type object …
Data.dtypes.value_counts()    # 18 object, 3 int64, 7 float64, 1 bool
```

- *Identification de la Target (log_price) et affichage de sa répartition*

```
import seaborn as sns
sns.histplot(Data['log_price'], kde=True, bins=30) # affichage de l'histogramme de prix
```

- *Répartition des données manquantes*

```
sns.heatmap(Data.isnull(),yticklabels=False,cbar=False,cmap='viridis')
X = Data.isna().sum()                  # nombre de données manquantes par features
(X[X>0]/Data.shape[0]).sort_values().  # Affichage triées en % (uniquement si sum>0)
```

Pre-processing

Analyse globale : complètes

- *Une connaissance plus approfondie de la Target peut fournir des informations sur le modèle et la stratégie à adopter.*

```
sns.histplot(Data['log_price'], kde=True, bins=30)        # Affichage des prix avec 30 barres
price_distr = pd.DataFrame(np.histogram(Data['log_price'], bins = 30)).T      #Distribution
```

- *Afin d'avoir une idée plus précise on peut afficher les prix réels.*

```
sns.histplot(Data['log_price'].apply(lambda x : math.exp(x)), kde=True, bins=30)
```

- *Dans la plupart des cas il est utile d'avoir plus d'informations sur les données manquantes, pour évaluer leur importance vis-à-vis de la target.*
- *Sur les features ayant le plus de données manquantes on peut calculer le nombre, le prix total le prix moyen …*

```
X = Data['first_review'].isnull()                          # données manquantes de first_review
sns.histplot(Data[X]['log_price'], kde=True, bins=30)      # répartition des prix
Data[Data['first_review'].isnull()]['log_price'].mean()    # prix moyen des données null
Data[Data['first_review'].isnull()==False]['log_price'].mean() # prix moyen des autres données
```

Pre-processing

Analyse globale : features incomplètes

- *La variable first_review représente une date. Il peut être intéressant de connaitre le nombre de locations en fonction de l'année du mois …*
- *Il est possible transformer une série en DateTime en utilisant la fonction to_datetime de pandas,*
- *On peut ensuite créer une DataFrame contenant uniquement les prix et les années correspondant à first_reveiw puis afficher le tout*

```
Data['host_since'] = pd.to_datetime(Data['host_since'])    # transformation du type host_since
# Création d'une dataFrame contenant l'année de la first review et le prix
Data_first_review = pd.DataFrame( {
    'year': pd.DatetimeIndex(Data['first_review']).year,      # création d'une colonne année
    'price': Data['log_price'] })                             # création d'une colonne prix
Data_ first_review.fillna(0, inplace=True)                    # remplacement de valeurs nulles
sns.countplot(x="year", data= Data_ first_review)             # répartition en fonction de l'année
```

- *En combinant les prix et les années il est possible de voir la répartition des données avec les principales statistiques, ainsi que les outliers.*

```
sns.boxplot(x="year", y="price" , data= Data_ first_review)
```

Pre-processing

Analyse globale : Données catégorielles

- *Il est nécessaire de récupérer le nom des features catégorielles.*

```
col_cat = Data.select_dtypes('object').columns        # nom des features de type string
for col in cal_cat :
  print(col, "  : "  , Data[col].nunique())          # nombre de valeurs différentes dans les features
```

```
# property_type : 65 valeurs # – type de location   # room_type : 3 valeurs # – type d'habitation
# amenities : 67122 valeurs # – description des équipements
# bed_type : 5 valeurs différentes – type des lits
# cancellation_policy : 5 valeurs # – type d'annulation   # city : 6 valeurs différentes
# description : 73479 valeurs # – Cette feature peut être supprimée
# first_review : 2554 valeurs # – est en réalité une date – peut être supprimée
# host_has_profile_pic , host_identity_verified: 2 valeurs #, peut être supprimée
# host_response_rate  : 80 valeurs # – en réalité un pourcentage – peut être supprimée
# host_since  : 3087 valeurs # – en réalité une date – peut être supprimée
# instance_bookable : 2 valeurs # – réservation possible
# last_review : 1371 valeurs # – en réalité une date – peut être supprimée
# name : 73359 valeurs # – peut être supprimée
# neighbourhood : 619 valeurs # – indication du quartier
# thumbmail_url  : 65883 valeurs # – peut être supprimée
# zip_code  : 769 valeurs # – peut être supprimée
```

Pre-processing

Nettoyage des données : Données manquantes

- *Données booléennes : pour host_has_profil_pic et host_identity_verified on affecte la valeur faux aux données manquantes.*

```
# Avant d'affecter la valeur faux aux features booléennes il faut voir leur valeurs
Data['host_has_profile_pic'].value_counts()                    # t (73696) et f (226)
Data['host_identity_verified'].value_counts()                  # t (49747) et f (24175)
# Les valeurs faux sont donc à remplacer par de f
Data['host_has_profile_pic'] = Data['host_has_profile_pic'].fillna('f')
Data['host_identity_verified'] = Data['host_identity_verified'].fillna('f')
```

- *Données Dates : On commence par remplacer les dates par l'année et ensuite les données manquantes par la dernière année non renseignée, et les valeurs manquantes par 0.*

```
Data['first_review'] = pd.DatetimeIndex(Data['first_review']).year      # Récupération de l'année
Data['host_since'] = pd.DatetimeIndex(Data['host_since']).year
Data['last_review'] = pd.DatetimeIndex(Data['last_review']).year
# Remplacement des valeurs manquantes par 0
Data['host_has_profile_pic'].fillna(value=0, inplace=True)
Data['host_since'].fillna(value=0, inplace=True)
Data['last_review'].fillna(value=0, inplace=True)
```

Pre-processing

Nettoyage des données : Données numériques

- *last_review est une date qui a été transformée en donné numérique (année). Les données manquantes ont étés remplacées par l'année 0.*
- *On cherche à savoir s'il existe une corrélation entre les features last_review et le prix de location.*

```
# création d'une DataFrame composée seulement des features last_review et log_price
select_ = Data[['last_review', 'log_price']]
select_.drop(select_[select_['last_review']==0].index, inplace=True)     #suppression des 0
select_corr()                                                            #pas de corrélation
Data.drop('[last_review'], axis=1, inplace=True)                 #on peut supprimer last_review
```

- *Le même traitement peut être appliqué aux features last_review et host_since. En constatant qu'il n'y a aucune corrélation avec le prix des locations on peut également supprimer ces deux colonnes.*

```
Data.drop(['last_review', 'host_since'], axis=1, inplace=True)
```

- *On peut supprimer d'autres features qui ne semblent pas liées au prix.*

```
Data.drop(['host_has_profile_pic', 'host_has_profile_pic', 'host_response_rate',
                                'neighbourhood'], axis=1, inplace=True)
```

Pre-processing

Nettoyage des données : Traitement des outliers

- *Au regard de la courbe des prix on constate que certains prix sont soit anormalement bas soit très haut, correspondant à de outliers.*
- *Nous identifions dans un premier temps les outliers correspondant à des prix bas < 2.5 ou très haut > 7.5 avant de les supprimer.*

```
# on identifie les samples qui sont des outliers sur les prix < 2.5 et > 7.5
outliers = Data [(Data ['log_price'] < 2.5) |
                 (Data['log_price'] > 7.5)].sort_values(by = 'log_price')
# Suppression dans le Data des outliers identifiés
Data= Data.drop(index=outliers.index)
```

- *La suppression des outliers devrait se faire sur toutes les données numériques de la même manière, et sur les données catégorielles en suivant la démarche effectué sur le type des locations.*
- *Pour les données numériques il est plus rapide d'utiliser une méthode d'apprentissage.*

Pre-processing

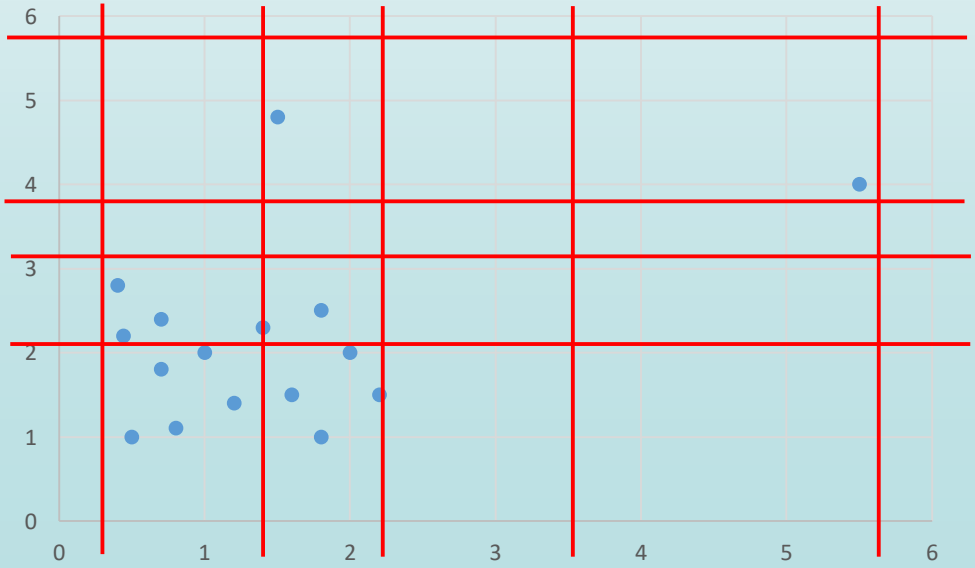
Nettoyage des données : détection d'anomalies

- *Il se peut qu'un sample soit très éloigné des autres, mais que cela ne soit pas perceptible en consultant une seule feature.*
- *Par exemple une image grise dans un DataSet d'animaux ne permet aucun traitement, mais il n'est pas possible de la rejeter en consultant un seul pixel, il faut faire une comparaison globale.*
- *Les techniques d'apprentissage non supervisées permettent de détecter des anomalies dans un DataSet.*
- *Dans Sickit-Learn la méthode isolationForest permet d'isoler des samples qui ont des caractéristiques très différentes des autres enregistrements.*
- *Si l'on précise un pourcentage de samples à isoler, le modèle après apprentissage (fonction fit) sera en mesure (fonction predict) d'affecter la valeur 1 aux samples normaux et -1 aux autres.*
- *Ces valeurs peuvent ensuite être utilisées pour supprimer les anomalies dans le dataSet.*

Pre-processing

Nettoyage des données : Isolation Forest

- *Isolation Forest est une technique de détection d'anomalies. On cherche à identifier les individus dont les caractéristiques sont très éloignées de celles des autres individus.*
- *La méthode consiste à effectuer une série de splits de manière aléatoire, afin d'isoler un individu.*
- *Plus le nombre de splits nécessaire pour isoler un individu est petit et plus il y aura de chance que cet individu soit une anomalie.*



Un premier tirage aléatoire nous donne les splits :
(Y, 5.8) ; (Y, 3.1) ; (X,3.5) …

Au troisième split les individus (1.5, 4.8) et (5.5, 4) sont isolés.

Le second tirage donne les résultats:
(X, 1.4) ; (X, 5.8) ; (Y, 2.1) ; (Y, 3.8) ; (X, 0.2) ; (X, 2,3) …

Au quatrième split l'individu (1.8, 2.5) est isolé

Au sixième split les individus (1.5, 4.8) et (5.5, 4) sont isolés.

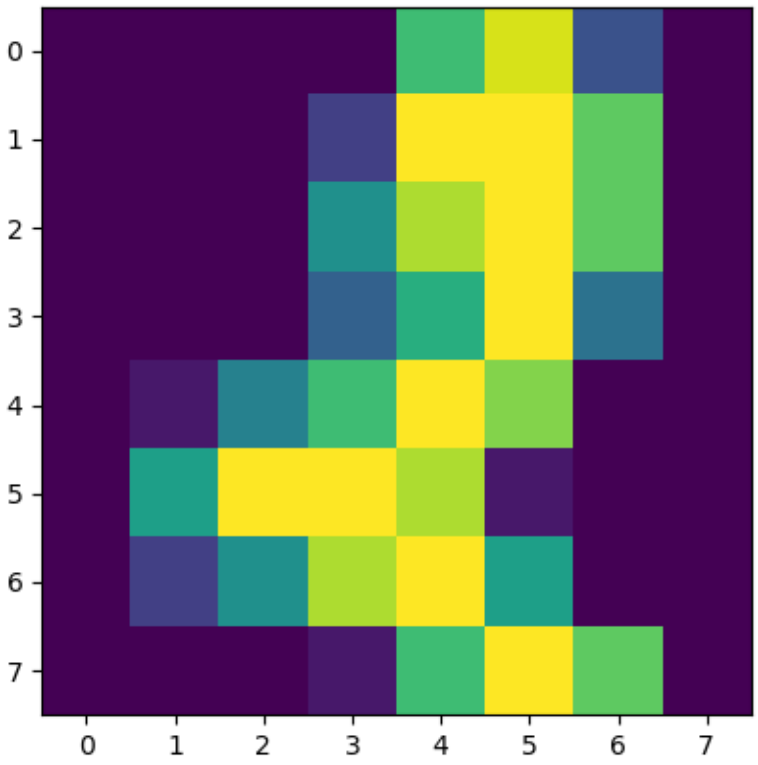
Afin d'éviter de traiter le point (1.8, 2.5) comme une anomalie, l'algorithme effectue un série de splits et conserve la moyenne du nombre de splits nécessaire pour isoler les individus.

Pre-processing

Nettoyage des données : Isolation Forest

```
from sklearn.ensemble
X = np.loadtxt("DataIF.t
model = IsolationFores
model.fit(X)
plt.scatter(X[:,0], X[:,1],
plt.show()
```

```
digits = load_digits()
X = digits.data ; y = dig
images = digits.images
model = IsolationFores
            contaminatio
model.fit(X)
# Predict est un tableau
anomalies = np.where(
plt.imshow(images[ano
plt.title(y[anomalies][0])
```



Pre-processing

Nettoyage des données : Suppression de features

- *Toutes les valeurs de id sont différentes, on peut donc supprimer cette feature qui n'apporte aucune information sur les prix pratiqués.*

```
Data['id'].nunique()                          # 74110 valeurs différentes
Data.drop(['id'], axis=1, inplace=True)       # suppression de la feature id
```

- *La plupart des valeurs de name sont différentes, de même pour 'zipcode', 'thumbnail_url', et 'description'. Elles peuvent être supprimées.*

```
Data.drop(['zipcode', 'thumbnail_url', 'name', 'description'], axis=1, inplace=True)
```

- *En consultant la table des corrélations on constate que la latitude et la longitude ne sont pas corrélés avec les prix, ainsi que number_of_review et review_score_rating, et peuvent donc être supprimées.*

```
plt.figure(figsize=(12,12))
sns.heatmap(Data.corr(), annot=True)                    # matrice de corrélation
Data.drop(['longitude', 'latitude', 'number_of_reviews',  'review_scores_rating'],
          axis=1, inplace=True)
```

Pre-processing

Nettoyage des données : Réduction des features - ACP

- *L'ACP vise à fournir une image simplifiée du nuage des individus la plus fidèle. On cherche un sous espace qui résume au mieux les données*
- *On cherche à transformer des variables liées entre elles ou corrélées en nouvelles variables dé-corrélées les unes des autres.*
- *Il est maintenant possible d'obtenir une base orthogonale de la matrice des corrélations.*

$$Cor(X) \longrightarrow [vec][val][vec]^t$$

[vec] est la matrice des vecteurs propres
[val] la matrice diagonale des valeurs propres λ_k

```
Vp = np.linalg.eig(Cor_X)
```

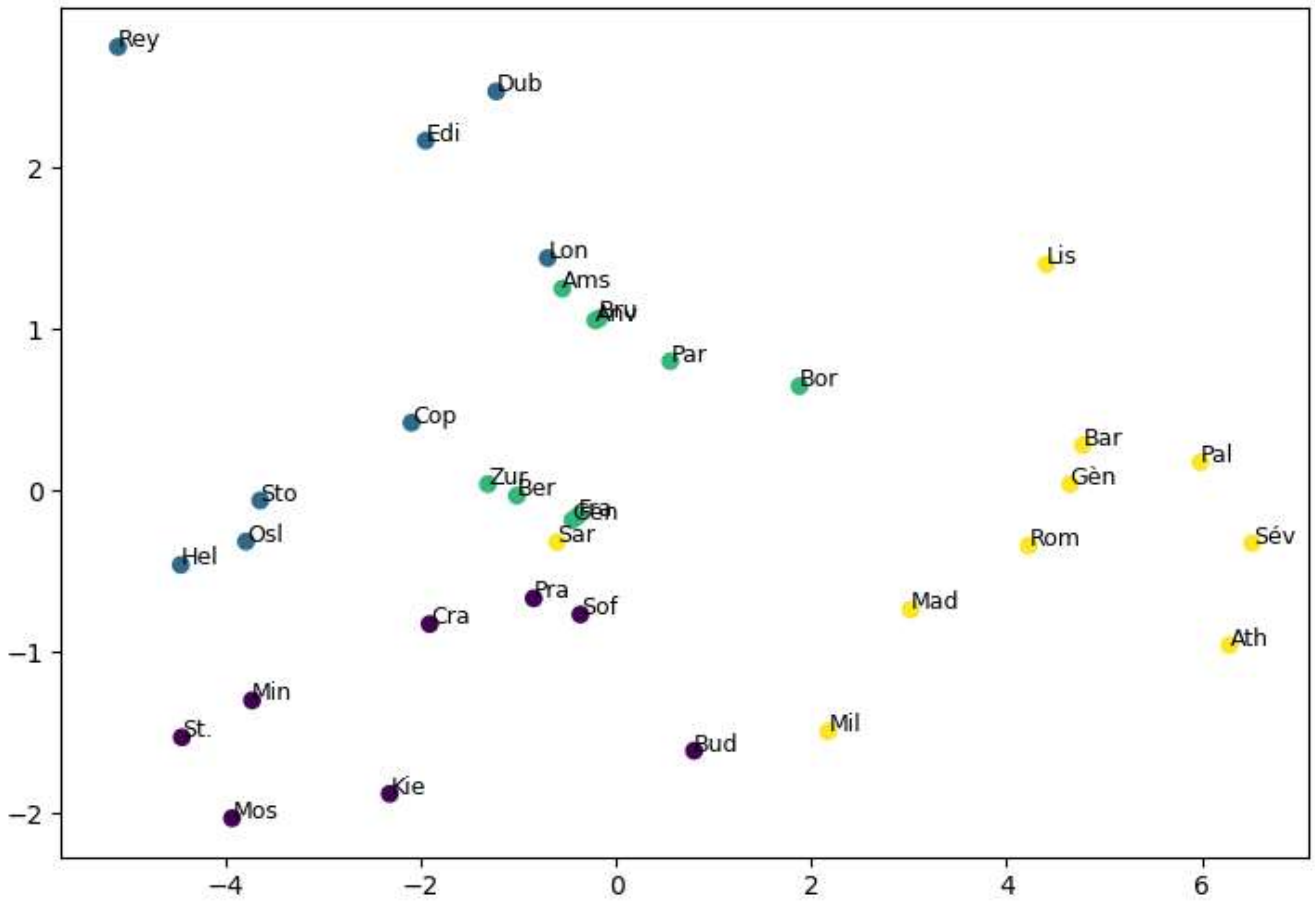
- *Les valeurs propres permettent de connaitre l'importance des dimensions les unes par rapport aux autres : composantes principales.*

```
from sklearn.preprocessing import StandardScaler # Bibliothèque pour la standardisation des données
from sklearn.decomposition import PCA            # Bibliothèque spécifique pour l'ACP
sc = StandardScaler() ; X = sc.fit_transform(DataSet)
acp = PCA(svd_solver='full')
Coord = acp.fit_transform(X)                     # Matrice des coordonnées dans le nouvel espace
print(acp.explained_variance_)                   # affichage des valeurs propres
```

Pre-processing

Nett

```
from sklea
Data_com
Data = Da
X = sc.fit_
X = pd.Da
encode =
codes=en
Data_acp
Data_acp
fig, ax = p
# Affichag
plt.scatter
# Parcours
for ind in D
    ax.annc
plt.show()
```



Pre-processing

Nettoyage des données : Suppression de features

- *Il est possible en analysant les variances de supprimer celles qui ont une variance inférieure à un seuil que l'on se fixe et varient que très peu.*
- *De plus la présence de ces variables peut dégrader les résultats.*
- *Le selecteur VarianceThreshold permet de sélectionner les features en fonction de la variance des variables.*

```
iris = sns.load_dataset('iris').select_dtypes(exclude='object')
from sklearn.feature_selection import VarianceThreshold
iris.var(axis=0)                          # Variance des différentes features
plt.plot(iris)                            # Affichage de l'évolution des paramètres
selector = VarianceThreshold(threshold=0.2)  # sélection des paramètres sur la variance
selector.fit_transform(iris)              #  colonnes avec une variance > 0.2
```

- *Ces sélecteurs fournissent plusieurs fonctions comme celle qui retourne la liste des variables sélectionnées de celles qui ne le sont pas.*

```
selector.get_support()                    #  liste des variables sélectionnées
```

Pre-processing

Nettoyage des données : Suppression de features

- *Il existe d'autres possibilités de sélectionner les features en fonction de différents tests de dépendance (khi2, anova, corrélation …).*
- *Ces tests fournissent un tableau qui contient les scores des tests de dépendance entre les variables X et y.*
- *Les sélecteurs comme selectKBest, basé sur un des tests statistiques permettent de sélectionner les meilleures variables.*
- *De plus la présence de ces variables peut dégrader les résultats.*
- *VarianceThreshold permet de sélectionner les features en fonction de la variance des variables.*

```
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.datasets import load_iris
iris = load_iris
chi2(iris.data, iris.target)                    # affichage des tests de khi2
selector = SelectKBest(chi2, k=2)
selector.fit_transform(iris.data, iris.target)  # Apprentissage et évaluation
selector.get_support()                          # Affichage des variables sélectionnées
```

Pre-processing

Les transformations : Données numériques

- *last_review est une date qui a été transformée en donné numérique (année). Les données manquantes ont étés remplacées par l'année 0.*
- *On cherche à savoir s'il existe une corrélation entre les features last_review et le prix de location.*

```
# création d'une DataFrame composée seulement des features last_review et log_price
select_ = Data[['last_review', 'log_price']]
select_.drop(select_[select_['last_review']==0].index, inplace=True)     #suppression des 0
select_corr()                                                            #pas de corrélation
Data.drop('[last_review'], axis=1, inplace=True)                 #on peut supprimer last_review
```

- *Le même traitement peut être appliqué aux features last_review et host_since. En constatant qu'il n'y a aucune corrélation avec le prix des locations on peut également supprimer ces deux colonnes.*

```
Data.drop(['last_review', 'host_since'], axis=1, inplace=True)
```

- *On peut supprimer d'autres features qui ne semblent pas liées au prix.*

```
Data.drop(['host_has_profile_pic', 'host_has_profile_pic', 'host_response_rate',
                                'neighbourhood'], axis=1, inplace=True)
```

Pre-processing

Les transformations : scikit-learn Imputation

- *Le module impute de sklearn fournit des transformer capables de remplacer les valeurs manquantes d'un dataSet.*
- *Dans la fonction SimpleImputer on doit préciser quelles sont les valeurs manquantes (np.nan, ' ', …) que l'on souhaite remplacer, ainsi que la stratégie : moyenne (mean), médiane (median), plus fréquente (most_frequent) ou constante (constant).*
- *La méthode fit() génère une fonction en analysant le trainSet, la méthode transform() s'applique ensuite à tous les jeux de données.*
- *Les stratégies comme mean, median ou most_frequent doivent être apprises uniquement sur des données du TrainSet*

```
Impor t numpy as np                                                    #
from sklearn.impute import SimpleImputer          # fonction d'imputation
transformer = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value='other')
X = pd.DataFrame(Data['neighbourhood'])
transformer.fit(X)                                  # Apprentissage sur une colonne
X['neighbourhood']=transformer.transform(X)
X[X['neighbourhood']=='other'].value_counts()   # 6872 valeurs other
```

Pre-processing

Les transformations : scikit-learn Imputation

- *Si l'on entraine sur le DataSet, le modèle aura récupérer des informations sur le TrainSet et les résultats seront inévitablement biaisés*

```
transformer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
X = Data[['neighbourhood','host_identity_verified']]
transformer.fit(X)                                  # Apprentissage sur deux colonnes
X=pd.DataFrame(transformer.transform(X), columns=X.columns)
X['host_identity_verified'].value_counts()          # (49748 + 188) 49936 T et 24175 F
X['neighbourhood'].isna().value_counts()            # (2862 + 6872) 9734 Williamburg
```

- *KNNImputer va essayer de remplacer les données manquantes d'un sample par les valeurs des samples qui lui ressemblent le plus.*
- *KNN s'appliquent uniquement sur des données numériques.*

```
from sklearn.impute import KNNImputer
transformer = KNNImputer(n_neighbors=1)
X = Data.copy() ; X=X.select_dtypes(exclude='object') ; X.drop(['id'], axis=1, inplace=True)
transformer.fit(X)                                  # Apprentissage sur deux colonnes
Y=X[X.index==3] ; Y.loc(3,['latitude'])=np.nan ; Y.loc(3,['longitude'])=np.nan
transformer.transform(Y)
```

Pre-processing

Les transformations : Encodage

- *La plupart des algorithmes de machine learning ne peuvent apprendre qu'à partir de jeux de données numériques.*
- *L'encodage consiste à convertir les données catégorielles en valeurs numériques.*
- *Les données catégorielles peuvent être traitées directement à partir des fonctions apply , replace, ou transforme des DataFrames.*

```
# Encodage de la variable catégorielle cleaning_fee avec la fonction apply
Data['cleaning_fee'] = Data['cleaning_fee'].apply(lambda x : 0 if x==True else 1)
# Encodage de la variable catégorielle cleaning_fee avec la fonction apply
Data['cleaning_fee'].replace(['True', 'False'], [0,1], inplace=True)
```

- *Si le nombre de variables à encoder est important, ces méthodes ne sont pas adaptées. On utilise la méthode get_dummies qui crée autant de colonnes qu'il y a de variables avec uniquement les valeurs 0 et 1.*

```
# Encodage des données
Data = pd.get_dummies(Data, columns=['cleaning_fee'])
```

Pre-processing

Les transformations : scikit-learn Transformers

- *Le module pre-processing fournit des 'transformers' catégoriels utilisés pour un traitement identique de tous les jeux de données.*
- *Les transformers apprendrent comment transformer les données sur le trainSet et peuvent l'appliquer sur toutes les données actuelles ou futures.*
- *La méthode fit() génère une fonction en analysant le trainSet, la méthode transform() peut s'appliquer ensuite à toutes les données.*
- *L'encodage ordinal consiste a associer à chaque catégories d'une variable une valeur décimale unique.*
- *LabelEncoder encode uniquement sur une série (souvent une target), OrdinalEncoder peut elle s'appliquer sur tout un dataFrame.*
- *inverse_transform() de LabelEncoder effectue un 'décodage'.*

```
from sklearn.preprocessing import LabelEncoder                  # récupération du transformer
transformer = LabelEncoder                                      # création d'un transformer
transformer.fit(Data['bed_type'])                               # apprentissage
Data['bed_type']=transformer.transform(Data['bed_type'])        # transformation des données
transformer.inverse = transform(Data['bed_type'])               # transformation inverse
```

Pre-processing

Les transformations : scikit-learn Transformers

- *Les encodeurs ordinal peuvent créer des problèmes, en effet un grand nombre de modèle sont basés sur des métriques.*
- *En codant la liste ['Chat', 'Chien', 'Lion', 'Poule', 'Rat'], nous obtenons [0, 1, 2, 3, 4], ce qui reviendrait à dire qu'un lion est plus proche d'une poule que de chat ??? Ces catégories n'ont rien d'ordinale.*
- *L'encodage One Hot (équivalent à get_dummies) crée autant de features binaires qu'il y a de catégories différentes.*
- *Les individus sont alors identifiés par un 1 dans une seule colonne et caractérisés par des variables différentes, plus comparables entre elles.*
- *Trois encodeurs LabelBinarizer, MultiLabelBinarizer et OneHotEncoder.*

```
from sklearn.preprocessing import OneHotEncoder          # récupération du transformer
transformer = OneHotEncoder()
Select_ = Data.select_dtypes('object').columns.drop(['amenities']) # colonnes object
X = transformer.fit_transform(Data[Select_])
X.toarray()
```

Pré-processing : Airbnb prédiction

Bilan : Features à supprimer

- *On déduit que des 15 features peuvent être supprimées.*

Nom	Type	Valeurs	Explication
Id	int64	74111	Id est un identificateur qui n'a aucun lien avec le prix
Description, name	string	73479-74111	Texte descriptif – pas d'interprétation possible
first_review, last_review	date	58247-58248	21% de valeurs manquantes. Pas de lien avec le prix
host_has_profile_pic	bool	73923	Pas de variabilité seules 226 valeurs fausses
host_identity_verified	bool	73923	Pas de différence sur les moyennes entre les valeurs
host_response_rate	int64	55812	Aucune corrélation avec le prix (0.006)
host_since	date	73923	Pas de lien direct avec le prix.
longitude, latitude	float	74111	Aucune corrélation avec le prix
number_of_reviews	int64	74111	Aucune corrélation avec le prix (0.03)
thumbnail_url	string	65895	Toutes les valeurs sont différentes aucun impact
review_scores_rating	float	57389	Pas de corrélation et 23% de valeurs manquantes
zipcode	int64	73145	Les informations sur les quartiers sont plus précises

Pré-processing : Airbnb prédiction

Analyse spécifique : Doublons

- *Il existe 609 doublons dans le dataSet.*
- *Après suppression des doublons il reste 73502 samples*

```
Data = Data.drop_duplicates(['log_price', 'amenities', 'city', 'neighbourhood], inplace=True)
```

Analyse spécifique : log_price - target

- *La feature 'log_price' est la target, le modèle d'apprentissage est un modèle supervisé de régression.*
- *Il n'y a pas de valeurs manquantes et la répartition est gaussienne.*
- *On constate qu'il y des prix très bas et d'autres un peu trop élevés.*

```
sns.histplot(data=Data, x='log_price', bins=30, kde=True)   # Répartition des prix
xns.boxplot(data=Data, x='log_price')                       # identification des outliers
X=pd.DataFrame(np.histogram(Data['log_price'], bins=30)).T  # Liste des bacs et
```

```
outliers = Data[(Data['log_price'<2.5]) | (Data['log_price']>7.5)] # liste des outliers 37 et 36
Data.drop(index=outliers.index, inplace=True)                      # 7429 samples
```

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'property_type'

- *35 types de location, 16 ont moins de 50 samples, soit 107 entrées.*

```
X=Data['property_type'].value_counts()>50          # nombre de samples par type
X[X==False].shape                                  # 107 entrées différentes
```

- ***Stratégie 1** : Afin d'éviter de garder des locations avec des types peu représentés, il est possible de les supprimer.*

```
# Création d'une DataFrame qui ne contient que les types de plus de 50 locations
Data = Data[Data['property_type'].isin(X[X==True].index)]
```

- ***Stratégie 2** : Il peut être envisageable pour ne pas perdre des données de regrouper les types de locations dans une catégorie particulière.*
- *Ce regroupement peut se faire si les moyennes sont équivalentes.*

```
(Data.groupby('property_type')['log_price'].mean()).sort_values()   # prix moyen par type
Data[~Data['property_type'].isin(X[X==True].index)].mean()          # prix : 4.92
Data[Data['property_type']=='Other']['log_price'].mean()            # prix : 4.94
```

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'property_type'

- *Transformation des types de location dans la catégorie Other.*

```
list_property  = X[X==True].index                    # types à modifier
# Transformation de tous les types de locations en Other
Data['property_type']=Data['property_type'].apply(lambda x : 'Other' if x in list_property else x)
# Affichage de la répartition des prix par rapport à la variable property_type
plt.figure(figsize=(24,8))
sns.boxplot(data=Data, x = 'property_type', y = 'log_price')
```

- *Quelque soit la stratégie choisie il reste 19 types de location.*
- *Les locations doivent être transformer via les fonctions get_dummies de pandas ou OneHotEncoder de scikit-learn.*
- *Les types de location ne pouvant pas être comparés entre eux il est nécessaire de les coder en utilisant une colonne binaire pour chacun.*

```
# Découpage des types de location en colonnes binaires avec suppression de la première
Data = pd.get_dummies(data=Data, columns= ['property_type'], drop_first=True)
```

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'room_type'

- *Pour des modèles d'apprentissage (hors arbres de décision, il peut être utile de centrer et réduire ces données.*
- *Ces données traitées par des colonnes binaires.*

```
Data.groupby(['room_type'])['log_price'].mean()          # moyenne de chaque types
Data = pd.get_dummies(data=Data, columns=['room_type'], drop_first=True)
```

Analyse spécifique : Variable 'accommodates'

- *Les valeurs sont comprises entre 1 et 16, avec des répartitions très inégales. La plupart des valeurs sont comprises entre 1 et 8*
- *Les prix moyens sont clairement fonction de la valeur d'accommodate.*
- *Les prix des locations sont fonction de cette variable, il est évident que celle-ci aura un impact important sur l'estimation.*

```
X = pd.DataFrame(Data['accommodates'].value_counts())   # Nombre d'accommodates
sns.barplot(x=X.index, y=X['accommodates'])             # Répartition des prix
Data.groupby(['accommodates'])['log_price'].mean()      # Prix varient 4.1 (1) à 6.0 (16)
```

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'accommodates'

- *La plupart des variables sont des valeurs binaires.*
- *Il peut alors être utile de normaliser les données pour des modèles d'apprentissage qui utilisent la descente de gradient pour l'estimation.*
- *La normalisation Min-Max qui met toutes les valeurs à l'échelle sur une plage de données comprise [0,1].*

```
from sklearn.preprocessing import MinMaxScaler          # moyenne de chaque types
sc = MinMaxScaler()
# Transformation des données accommodates et données réduites comprise entre 0 et 1
Data['accommodates']=sc.fit_transform(Data[['accommodates']])
```

Analyse spécifique : Variable 'bathrooms'

- *Les prix sont liés au nombre de salle de bains excepté pour les locations avec 8 salles de bains.*
- *La plupart sont des locations partagées*

```
Data.groupby('bathrooms')['log_price'].mean()          # prix moyen par nombre de SDB
Data[Data['bathrooms']==8]                             # 34 locations dont 22 partagées
```

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'bathrooms'

- *Il peut être envisageable de supprimer les locations avec 8 salles de bains qui peuvent impacter le résultat final.*
- *Compte tenu du nombre peu important de cette catégorie et du lien avec la variable room_type on peut décider de conserver ces données.*
- *Cette variable est particulière car on constate qu'il y a un grand nombre de locations avec plus de salle de bain que de chambres ou de lits.*
- *Il y a même des nombres de salle de bain non entiers (0.5 à 7.5), il pourrait être nécessaire de traiter ces cas pour améliorer les estimations.*
- *Il y a 195 valeurs manquantes. On constate que le prix moyen de ces locations est égal aux locations avec une SBD.*
- *On décide donc de remplacer les données manquantes par la valeur 1.*

```
X=Data[(Data['bathrooms'].isna())]                    # Données manquantes
X['log_price'].mean()                                 # prix : 4.64
# prix moyen des locations avec 1 SDB = 4.67
Data['accommodates'].fillna(value=1, inplace=True)    # modification des données nan
```

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'bed_type'

- *Il y a 5 types de couchages, les plus nombreux (71351) sont de type 'Real Bed', avec des prix moyens de 4.7 contre 4.3 ou 4.4 pour les autres.*
- *Les prix des locations autre que 'Real Bed' sont similaires.*
- *Compte tenu du nombre peu important de ces locations et leur similarité on peut les regrouper en une seule catégorie.*

```
X=Data[(Data['bed_type']!='Real Bed')==True]                    # Type de lits autre que Real Bed
sns.histplot(data=X, x='log_price', kde=True, bins=30, hue='bed_type')
# Encodage au format binaire
Data['bed_type']=Data['bed_type'].apply(lambda x : 1 if x!='Real Bed' else 0)
```

Analyse spécifique : Variable 'cancelation_policy'

- *5 types d'annulation avec des prix moyens très différents.*
- *Les données doivent être conservées et encodées.*

```
# Encodage binaire avec suppression d'une des colonnes
Data = pd.get_dummies(data=Data, columns=[cancelation_policy'], drop_first=True)
```

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'cleaning_fee'

- *Il ne semble pas y avoir d'influence de cleaning_fee sur les prix.*
- *La variable sera encodée en binaire.*

```
sns.histplot(data=Data, x='log_price', kde=True, bins=30, hue='cleaning_fee')
Data['cleaning_fee'] = Data['cleaning_fee'].apply(lambda x : 1 if x==True else 0)
```

Analyse spécifique : Variable 'city'

- *6 villes différentes avec des locations plutôt à NYC et LA.*
- *Les prix moyens à SF sont légèrement plus élevés.*
- *Ces données doivent être encodées en utilisant des colonnes binaires.*

```
sns.histplot(data=Data, x='log_price', kde=True, bins=30, hue='city')
Data = pd.get_dummies(data=Data, columns=['city'], drop_first=True)
```

Analyse spécifique : Variable 'instant_bookable'

- *Même analyse que pour cleaning_fee*

```
Data['instant_bookable'] = Data['instant_bookable'].apply(lambda x : 1 if x=='t' else 0)
```

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'bedrooms'

- *Il y a 90 valeurs manquantes, avec un prix moyen de 4.77.*
- *Leur répartition est assez irrégulière, il est difficile de leurs affecter toutes une valeur spécifique. On choisit donc de les supprimer.*
- *Comme pour la variable 'accommodates' les valeurs peuvent être centrées et réduite pour certains algorithmes d'apprentissage.*

```
Data.dropna(subset=['bedrooms'],axis=0, inplace=True)
Data['beedrooms']=sc.fit_transform(Data[['beedrooms']])
```

Analyse spécifique : Variable 'beds'

- *Il reste 99 valeurs manquantes, avec un prix moyen de 4.56.*
- *Les locations avec un seul lit ont des prix équivalents : 4,53.*
- *Comme précédemment on va choisir de supprimer ces valeurs.*

```
X = Data[Data['beds'].isna()==True]
sns.histplot(data=X, x='log_price', bins=30, kde=True)
Data.dropna(subset=[beds'],axis=0, inplace=True)
```

Pré-processing : Airbnb prédiction

Analyse spécifique : corrélation

- *L'analyse des corrélations, nous indique que les variables accommodates, room_type et beedrooms sont très corrélées aux prix.*
- *A l'inverse la variable instant_bookable semble très peu corrélée avec les prix, cette variables pourrait être supprimée.*
- *On constate qu'il n'y a pas un impact fort des villes sur les prix mis à part SF. Il pourrait être utile de regrouper les villes autres que SF.*

```
X = Data.copy()
X.drop(['property_type','amenities','neighbourhood'], axis=1, inplace=True)
plt.figure(figsize=(24,24))
sns.heatmap(X.corr(), annot=True)
```

- *En fonction de leur lien Anova avec le prix on obtient.*

```
from sklearn.feature_selection import SelectKBest, f_classif
model = SelectKBest(score_func=f_classif, k=13)
model.fit(X.drop(['log_price'],axis=1), X['log_price'])
model.get_feature_names_out()      # accommodates, batbrooms, cleaning_fee, bedrooms',
'beds', (3) 'room_type_", (4) 'cancellation_policy_', 'city_SF'
```

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'neighbourhood'

- *Il 6637 valeurs manquantes avec une répartition qui correspond à la répartition classique.*
- *Ces valeurs ne semblent pas appartenir à un quartier en particulier.*

```
X=Data[Data['neighbourhood'].isnull()==True]                  # 6637 valeurs manquantes
sns.histplot(Data['log_price'], bins=30, kde=True)            #  variation des prix
Data.dropna(subset=['neighbourhood'],axis=0, inplace=True)    #  suppression des données
```

- *Il a 619 quartiers différents, sur les 66603 samples.*
- *Les prix moyens par quartiers vont de [3.17 à 7.17].*

```
Select_ = pd.DataFrame((Data.groupby(['neighbourhood'])['log_price'].mean()))
sns.histplot(Select_['log_price'], bins=30, kde=True)
```

- *On peut découper les quartiers en 10 groupes.*

```
Select_['Categorie'] = pd.cut(Select_['log_price'], bins=10, labels=False)
sns.histplot(y='log_price', data=Select_)          #  Nombre de locations pour chaque prix
Select_.sort_values(by='log_price', inplace=True) #  réorganisation en fonction du prix
sns.histplot(x='neighbourhood', y='log_price', data=Select_)
```

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'neighbourhood'

- *Transformation des quartiers en fonction du découpage.*

```
for col_ in Select_.index:                # Changement des quartiers en fonction du découpage
    Data['neighbourhood'] = Data['neighbourhood'].apply(
                                lambda x : Select_['Categorie'][col_] if x==col_ else x)
```

Analyse spécifique : Variable 'amenities'

- *Afin de pouvoir identifier les différents aménagements, nous devons commencer par supprimer les caractères inutiles (", {, }).*
- *La bibliothèque re permet le traitement des expressions régulières.*

```
import re                                          # bibliothèque desexpressions régulières
# Suppression de tous les caractères de séparation par rien
Data['amenities']=Data['amenities'].apply(lambda x : re.sub(r'[\"\{\}]',",x))
```

- *On peut maintenant traiter les différentes mots (séparés uniquement par une ,) pour créer une liste de tous les aménagements.*
- *On utilise un ensemble qui contient la liste des aménagements.*

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'amenities'

- *Cette fonction permet de générer la liste de tout les aménagements.*

```
def amenities_list(df) :
    amenities = set()                          # Création d'un ensemble
    for line in df['amenities'] :              # Parcours de toutes les valeurs de amenities
        for word in line.split(','):           # Découpage de la ligne en mots
            if len(word.strip())>0 :           # Si une information est présente
                amenities.add(word.strip())    # Suppression des blancs début/fin puis ajout
    return(list(amenities))
amenities = amenities_list(Data)               # 130 aménagements différents
```

- *On se doit maintenant de vérifier si des valeurs sont mal orthographiées, et les remplacer dans ce cas.*

```
sorted(amenities)                              # Affichage des aménagements
```

- *On crée deux listes d'aménagements, une contenant les valeurs à remplacer et l'autre les valeurs qui les remplaces.*
- *La fonction replace permet d'appliquer les remplacements.*

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'amenities'

- *Le seul moyen est d'identifier les erreurs "manuellement".*

```
to_replace_ = list(['Doorman Entry', 'Elevator in building' , 'Internet' , 'Firm matress',
        'Other pet(s)', 'Smart lock', 'Wide clearance to bed', 'Wide clearance to shower & toilet',
         'Wide clearance to shower and toilet',  'translation missing: en.hosting_amenity_49',
         'translation missing: en.hosting_amenity_50', 'Waterfront'])
replace_ = list(['Doorman', 'Elevator', 'Ethernet connection' , 'Firm mattress', 'Pets allowed',
          'Smartlock' , 'Wide clearance', 'Wide clearance', 'Wide clearance', 'translation missing',
          'translation missing', 'Beachfront'])
Data['amenities'].replace(to_replace= to_replace_, value=replace_, regex=True, inplace=True)
```

- *Les aménagements maintenant bien orthographiés, il est possible de créer une colonne pour chaque aménagement.*

```
def amenities_to_dummies(df, amenities) :      # Dummification des données
   for amenitie in amenities :                 # Parcours des différentes location
       # Création d'une dataframe amenitie composé de 0 ou de 1 lorsque cette commodité est
       #présente dans le descriptif de la location considérée
       dummie = [1 if amenitie in s.split(',') else 0 for s in Data['amenities']]
       dummie = pd.DataFrame(dummie, index=Data.index, columns=[amenitie])
       df = pd.concat([df,dummie], axis=1)      # ajout de la colonnes amenitie
return df                                       # Retour du nouveau DataFrame
```

Pré-processing : Airbnb prédiction

Analyse spécifique : Variable 'amenities'

- *Compte tenu du nombre important de colonnes créés il est nécessaire d'analyser ces colonnes pour voir si certaines peuvent être supprimées.*
- *En particulier celles qui sont trop peu présentes dans les locations.*

```
Select_ = []
for col in amenities :
    if Data[col].sum()<=10 : Select_.append(col)     # Liste des aménagements à supprimer
Data.drop(Select_, axis=1, inplace=True)             # Suppression des aménagements
```

- *L'analyse des corrélations avec le prix des locations permet ensuite de constater qu'elles sont ceux qui peuvent avoir un impact ou pas.*

```
amenities=sorted([amenitie for amenitie in amenities if amenitie not in Select_])
corr_ = []
for amenitie in amenities:  corr_.append(Data[amenitie].corr(Data['log_price']))
for corr, amenitie in sorted(zip(corr, amenities)) :
    print(f'{amenitie : <50} {corr} ({Data[amenitie].sum()})')
```

- *On constate que les aménagements avec peu de données sont celles qui ont le moins d'influence, elle pourrait être supprimées.*

Pré-processing : Airbnb prédiction

Analyse spécifique : Features sélection

- *Avec un grand nombre de features il est souvent nécessaire de sélectionner les variables (Kbest, RFE, Forêts ou Arbres de décision, …)*
- *SelectKBest peut être utilisée avec une suite de différents tests statistiques pour sélectionner un nombre spécifique de fonctionnalités.*

```
from sklearn.feature_selection import SelectKBest, f_classif   # Test Anova
model = SelectKBest(score_func=f_classif, k=20)
model.fit(Data[amenities], Data['log_price'])                  # Apprentissage
X = pd.DataFrame({'name' : model.feature_names_in_, 'score' : model.scores_})
X = X.sort_values(by='score', ascending=False)
for row in X.iterrows():
    print(f'{row[1][0]: <40} {row[1][1]}')
# classement de résultats obtenus
[Family/kid friendly  (11.3), 'TV' (9.4), 'Cable TV' (7.9), Indoor fireplace (6.3), 'translation
missing' (5.9), 'Path to entrance lit at night' (4.5), Table corner guards (4.5), 'Dryer' (4.5),
'Suitable for events' (4.4), 'Washer' (4.3), 'Lock on bedroom door' (4.3), '24-hour check-in' (4),
'Self Check-In' (3.9), 'Stair gates' (), 'Hair dryer' (3.7), 'Doorman' (3.7) …]
```

- *On constate que les features qui ont le meilleur score Anova ne sont pas toutes celles qui ont les meilleurs taux de corrélation.*

Pré-processing : Airbnb prédiction

Analyse spécifique : Features sélection

- *RFE utilise un modèle d'apprentissage pour sélectionner au minimum les n meilleurs variables, par éliminations successives.*

```
from sklearn.feature_selection import RFE        # Test Anova
from sklearn.svm import SVR                      # utilisation de support vector machine
# conservation de 20 features au minimum avec 2 features éliminés à chaque itération
model = RFE(SVR(kernel="linear"), n_features_to_select=20, step=2))
model.fit(Data[X], Data['log_price'])            # Apprentissage
model.class_                                      # Liste des features sélectionnées
model.grid_score_                                 # Scores obtenu par SVR à chaque étapes
```

- *L'analyse en composante principale permet également de réduire la dimensionnalité d'un dataSet.*
- *Si les valeurs des features sont très différentes, il est préalable de centrer et de réduire les données.*

```
from sklearn.decomposition import PCA            # Module la réduction de dimensionnalité
model = PCA(svd_solver='full')
model.fit(Data[X])                               # Application de l'ACP sur les données
np.cumsum(model.explained_variance_ratio_) # Avec 72 variables (116) on explique 99%:
```

Pré-processing : Airbnb prédiction

Analyse spécifique : Features sélection

- *Les arbres de décision, les random forest, adaboost ou extraTrees peuvent être utilisés pour estimer l'importance des features.*
- *L'attibut feature_importances_ retourne quelles sont les features les plus importantes dans l'arbre de décision*

```
from sklearn.tree import DecisionTreeRegressor          # Arbre de décision
model = DecisionTreeRegressor()
model.fit(Data[X], Data['log_price'])                   # Apprentissage
features_importance = pd.DataFrame(model.feature_importances_,
                                   index=Data[X].columns, columns=['importance'])
```

- *L'affichage sous forme de bar.*

```
feature_importance = feature_importance[(feature_importance['importance']>0.02)] # 21/ 116
plt.figure(figsize=(12,10)) ; plt.xticks(rotation=90)   # Caractéristiques pour l'affichage
model.fit(Data[X], Data['log_price'])                   # Apprentissage
sns.barplot(y=features_importance['importance'], x=feature_importance.index)
# ['24-hour check-in', 'Cable TV', 'Air conditioning'), 'Breakfast', 'Buzzer/wireless intercom',
'Carbon monoxide detector', 'Elevator', 'Essentials', 'Family/kid friendly', 'Fire extinguisher',
'First aid kit', 'Free parking on premises', 'Hair dryer', 'Hangers', 'Indoor fireplace', 'Iron', 'Laptop
friendly workspace', 'Lock on bedroom door', 'Pets allowed', 'Safety card', 'Shampoo', 'TV']
```

Apprentissage : Airbnb prédiction

Chargement des données

- *On commence par lire les données, puis on supprime les variables inutiles et les doublons*

```
import pandas as pd
import numpy as np
import seaborn as sns
Data = pd.read_csv('Airbnb prediction.csv', delimiter=',')
```

- *On supprimer ensuite les features inutiles et les enregistrement en double.*

```
def data_cleaning(df) :
    # Suppression des features jugées inutiles (15 features)
    df.drop(['id', 'description', 'name', 'first_review', 'last_review',
          'host_has_profile_pic', 'host_identity_verified', 'host_response_rate',
          'host_since', 'longitude', 'latitude', 'number_of_reviews', 'thumbnail_url',
          'review_scores_rating', 'zipcode'], axis=1, inplace=True)
     # Suppression des samples dupliqués (609 sur la base de seulement 4 features)
     df.drop_duplicates(['log_price','amenities', 'city', 'neighbourhood'], inplace=True)
     return(df)
Data = data_cleaning(Data))
```

Apprentissage : Airbnb prédiction

Traitement des données manquantes ou aberrantes

- *Les outliers ainsi que les données manquantes pouvant générer des interprétations erronées sont supprimées.*

```
def drop_samples(df):
    # Suppression des locations considérées comme des outliers
    outliers = df[(df['log_price']<2.5) | (df['log_price']>7.5)].index
    df.drop(index=outliers, inplace=True)
    # Suppression des données manquantes :dans  beedrooms, beds, neighbourhood
    df.dropna(subset=['bedrooms'],axis=0, inplace=True)
    df.dropna(subset=['beds'],axis=0, inplace=True)
    df.dropna(subset=['neighbourhood'],axis=0, inplace=True)
    return df
Data = drop_samples(Data)
```

Encoder et compléter des features

- *Les données manquantes sur certaines features peuvent être remplacer par des valeurs jugées cohérentes.*
- *Sur les neuf données catégorielles 6 peuvent être encodées directement via la méthode de get_dummies() de pandas.*

Apprentissage : Airbnb prédiction

Encoder et compléter des features

- *Sur les données numériques on remplace les données manquantes.*
- *Pour les données catégorielles certaines doivent être décomposées en tant que variables binaires.*

```
def encodage(df):
    # Remplacement des données manquantes sur les données numériques
    df['accommodates'].fillna(value=1, inplace=True)
    df['bathrooms'].fillna(value=1, inplace=True)
    # Encodage simple des données catégorielles
    df['bed_type']=df['bed_type'].apply(lambda x : 1 if x=='Real Bed' else 0)
    df['cleaning_fee']=df['cleaning_fee'].apply(lambda x : 1 if x==True else 0)
    df['instant_bookable']=df['instant_bookable'].apply(lambda x : 1 if x=='t' else 0)
    # Création de colonnes binaires pour les catégorielles
    df = pd.get_dummies(df, columns=['room_type'])
    df = pd.get_dummies(df, columns=['cancellation_policy'])
    df = pd.get_dummies(df, columns=['city'])
    return df
Data = encodage(Data)
```

- *Il reste à traiter les features 'amenities', 'property_type'. 'neighbourhood'*

Apprentissage : Airbnb prédiction

Encodage 'amenities'

- *L'encodage de la variable amenities nécessite plusieurs étapes.*
- *(1) Supprimer tous les caractères inutiles tels que ", {, }.*
- *(2) Identifier les différents aménagements disponibles.*
- *(3) Corriger les erreurs et modifier les aménagements mal orthographiés*
- *(4) Créer des variables pour chaque aménagement, et les compléter avec les données de la feature amenities*
- *(5) Supprimer les aménagements trop peu présents dans les locations.*
- *(6) Enfin, il peut être nécessaire de réduire sa dimensionnalité.*

```
def amenities_list(df) :
    # Identification de tous les aménagements disponibles
    amenities = set()
    for line in df :
        for word in line.split(',') :
            if (len(word.strip())>0) :
                amenities.add(word.strip())
    return list(amenities)
```

Apprentissage : Airbnb prédiction

Encodage 'amenities'

```
def encode_amenities(df) :
    # (1) Suppression des caratères '',{, et } dans amenities
    df['amenities'] = df['amenities'].apply(lambda x : re.sub(r'[\"\{\}]','',x))
    # (2) Liste des aménagements disponibles
    amenities = amenities_list(df['amenities'])
    # (3) Remplacement des aménagements mal orthographiés
    to_replace_ = list(['Doorman Entry', 'Elevator in building' , 'Internet', 'Firm matress', 'Other
        pet(s)', 'Smart lock', 'Wide clearance to bed', 'Wide clearance to shower & toilet',
        'Wide clearance to shower and toilet', 'translation missing: en.hosting_amenity_49',
        'translation missing: en.hosting_amenity_50', 'Waterfront'])
    replace_ = list(['Doorman', 'Elevator', 'Ethernet connection' , 'Firm mattress', 'Pets allowed',
        'Smartlock' , 'Wide clearance', 'Wide clearance', 'Wide clearance', 'translation missing',
        'translation missing', 'Beachfront'])
    df.replace(to_replace= to_replace_, value=replace_, regex=True, inplace=True)
    # Caclul de la nouvelle liste des aménagements et suppression de amenities
    amenities = amenities_list(df['amenities'])
    # (4) Création d'une colonne pour chaque catégorie
```

Apprentissage : Airbnb prédiction

Encodage 'amenities'

```
    # (4) Remplissage des colonnes aménities à partir des données de chaque samples
    for amenitie in amenities :
        dummie = [1 if amenitie in s.split(',') else 0 for s in Data['amenities']]
        dummie = pd.DataFrame(dummie, index=Data.index, columns=[amenitie])
        df = pd.concat([df,dummie], axis=1)       # ajout de la colonnes amenitie
    df.drop(['amenities'], axis=1, inplace=True)
    return df, amenities
Data, amenities = encode_amenities(Data)
```

- *Il est ensuite nécessaire de réduire la dimensionnalité.*

```
def reduce_amenities(df, amenities)
    # (5) Suppression des aménagement peu représentés
    Select_ = []
    for amenitie in amenities:
      if (Data[amenitie].sum()<=10): Select_.append(amenitie)
    df.drop(Select_, axis=1, inplace=True)
    # Récupération de la liste  des aménagements restants
    amenities=sorted([amenitie for amenitie in amenities if amenitie not in Select_])
    # (6) Aucune réduction en l'état
    return df, amenities
Data, amenities = encode_amenities(Data, amenities)
```

Apprentissage : Airbnb prédiction

Encodage 'property_type'

- *Dans la partie analyse nous avons identifié deux stratégies possibles compte tenu du nombre peu important de certains type de propriétés.*
- *La première consiste à supprimer les types inférieurs à 50 valeurs.*
- *La seconde à les regrouper dans la catégorie 'Other'.*

```
def encodage_property_type(df) :
    #  Identification des types avec peu de valeurs
    Select_ = (df['property_type'].value_counts()<50)
    Select_ = list(Select_[Select_==True].index)
    #  Suppression types identifiés (solution 1)
    df.drop(df[df['property_type'].isin(Select_)].index, axis=0, inplace=True)
    #  Transformation en Other (solution 2)
    # df['property_type']=df['property_type'].apply(lambda x : 'Other' if x in Select_ else x)
    df=pd.get_dummies(df, columns=['property_type'], drop_first=True)
    return df
Data = encodage_property_type(Data)
```

Apprentissage : Airbnb prédiction

Encodage 'neighbourhood'

- *Nous pouvons soit supprimer les quartiers pour l'évaluation des prix soit les regrouper en fonction de catégories.*
- *Dans une analyse par villes il pourrait être utile de traiter les quartiers différemment.*

```
def encodage_neighbourhood(df) :
    #  Suppression des quartiers
    df.drop('neighbourhood', axis=1, inplace=True) #  suppression de la colonne neighbourhood
    return(df)
```

- *A ce stade un premier modèle d'apprentissage peut être utilisé pour analyser les résultats.*

Apprentissage : Airbnb prédiction

Modèle d'apprentissage

- Pour commencer on découpe le dataSet en un trainSet et un testSet, avant de choisir un modèle d'apprentissage.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(Data.drop('log_price', axis=1),
        Data['log_price'], test_size=0.2, random_state=0)        # Découpe du dataSet
```

- On commence par un arbre de décision, car ils sont simples à interpréter, peuvent traiter des données avec peu de préparation, acceptent des données numériques et catégorielles …
- De plus l'attribut feature_importances_ permet de mesurer l'importance des features par rapport à la target pour d'éventuelles simplifications.

```
from sklearn.tree import DecisionTreeRegressor
model.fit(X_train, y_train)
def evaluation(model, X_train, X_test, y_train, y_test):
    y_pred = model.predict(X_test)
    print(model.score(X_test, y_test))
model = DecisionTreeRegressor (random_state=0)        # Model d'arbre de décision
evaluation(model, X_train, X_test, y_train, y_test)   # score R2 : 0.23
```

Apprentissage : Airbnb prédiction

Amélioration du modèle

- *Pour améliorer le modèle nous pouvons commencer par régler les hyperparamètres, pour cela nous devons utiliser uniquement le trainSet.*
- *Le trainSet est alors découpé en n parties égales (n-1 train et 1 de valSet), et le modèle est entrainé sur n-1 train et testé sur la partie valSet.*
- *La cross validation consiste évaluer le modèle en utilisant toutes les configurations possibles d'une découpe du tainSet en n parties.*

```
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
score= []
for n in range(1,50) :                    #  on teste le modèle en fonction de la profondeur de l'arbre
    model = DecisionTreeRegressor(max_depth=n, random_state=0)
    score.append( cross_val_score(model, X_train, y_train, cv=10)
plt.plot([x.mean() for x in score])      #  la meilleur profondeur de l'arbre est de 9
```

- *Le meilleur score est obtenu pour une profondeur d'arbre de 9*

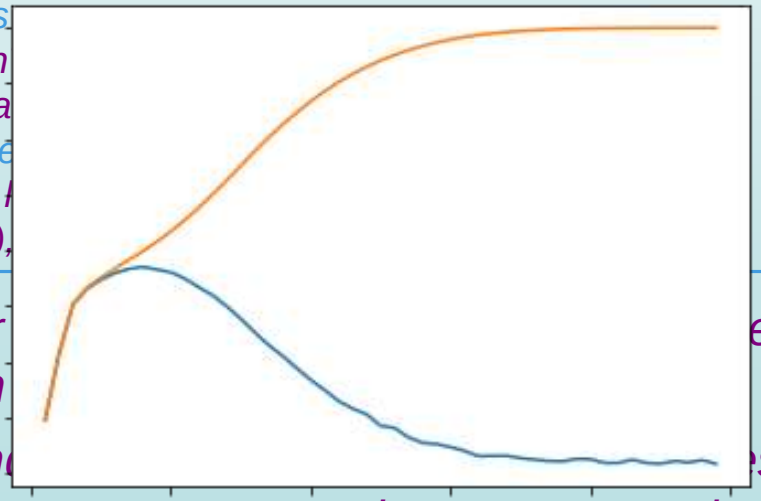
```
model = DecisionTreeRegressor (random_state=0, max_depth=9)
evaluation(model, X_train, X_test, y_train, y_test)          #  score R2 : 0.58
```

Apprentissage : Airbnb prédiction

Amélioration du modèle

- Un des principaux problèmes des arbres de décision vient de leur sensibilité au sur-apprentissage, lorsque le découpage devient trop fin.
- La fonction validation_curve permet d'évaluer les scores d'un modèle pour évaluer le réglage le plus performant.
- Cette fonction teste toutes les valeurs d'un hyper-paramètre.

```
from sklearn.model_selection import validation_curve
#  Calage du paramètre max_depth sur 50 valeurs
train_score, val_score = validation_curve(Decision
                         param_name='max_depth', pa
#  Affichage de résultats sur les données du train e
plt.plot(np.arange(1,50), val_score.mean(axis=1),
plt.plot(np.arange(1,50), train_score.mean(axis=1),
```



- Comme on peut le constater sur … et le validation set que le modèle est en …
- En peut en conclure qu'un gran… …s et correspondent à du "bruit", un découpage trop grand est contre productif.

Apprentissage : Airbnb prédiction

Importance des variables

- Si l'on regarde l'importance des variables pour le modèle, on constate qu'a partir partir de la 10ieme feature l'importance est inférieurs à 0.005.

```
feature_importances_  = pd.DataFrame(data=model.feature_importances_,
                                     index=X_train.columns, columns=['importance'])
feature_importances_=feature_importances_.sort_values(by='importance', ascending=False)
#  room_type_Entire home/apt (0.59), bathrooms (0.18), accommodates , bedrooms (0.04),
city_SF (0.03), city_LA (0.02), Elevator , city_Chicago, room_type_Private room (0.01)
```

Réduction de dimensionnalité

- On peut tester quel serait l'impact d'un apprentissage sur des données composée de seulement d'une partie des features les plus importantes.

```
def reduce_features(Data, feature_importances_) :
    #  Sélection des 20 meilleurs features
    features_select_ = list(feature_importances_.index[:20])
    features_select_.append('log_price')
    df = Data[features_select_]
    X_train, X_test, y_train, y_test = train_test_split(df.drop('log_price', axis=1),
          df['log_price'], test_size=0.2, random_state=0)
    evaluation(model, X_train, X_test, y_train, y_test)          #  score R2 : 0.58
```

Apprentissage : Airbnb prédiction

Amélioration du modèle : variable neighbourhood

- Pour cela nous modifions l'encodage de 'neighbourhood' en conservant la variable et en regroupant en fonction des prix moyens par quartiers.

```
def encodage_neighbourhood(df) :
    #  Création de 10 groupes pour les quartiers en fonction des prix moyens
    Select_ = pd.DataFrame((Data.groupby(['neighbourhood'])['log_price'].mean()))
    Select_['Categorie'] = pd.cut(Select_['log_price'], bins=10, labels=False)
    for col_ in Select_.index:          #  Changement des quartiers en fonction du découpage
        df['neighbourhood'] = df['neighbourhood'].apply(
                              lambda x : Select_['Categorie'][col_] if x==col_ else x)
    return(df)
Data = encodage_neighbourhood(Data)
model = DecisionTreeRegressor (random_state=0, max_depth=9)
evaluation(model, X_train, X_test, y_train, y_test)          #  score R2 : 0.64
```

Influence des différentes features

- Le quartier est la troisième feature la plus importante, et en réduisant le Data au 20 features les plus importantes le score est identique 0.644.

```
#  room_type_Entire home/apt (0.54), bathrooms (0.17), neighbourhood (0.15), bedrooms
(0.04), accommodates (0.03) city_NYC (0.001),
```

Apprentissage : Airbnb prédiction

Les courbes d'apprentissage

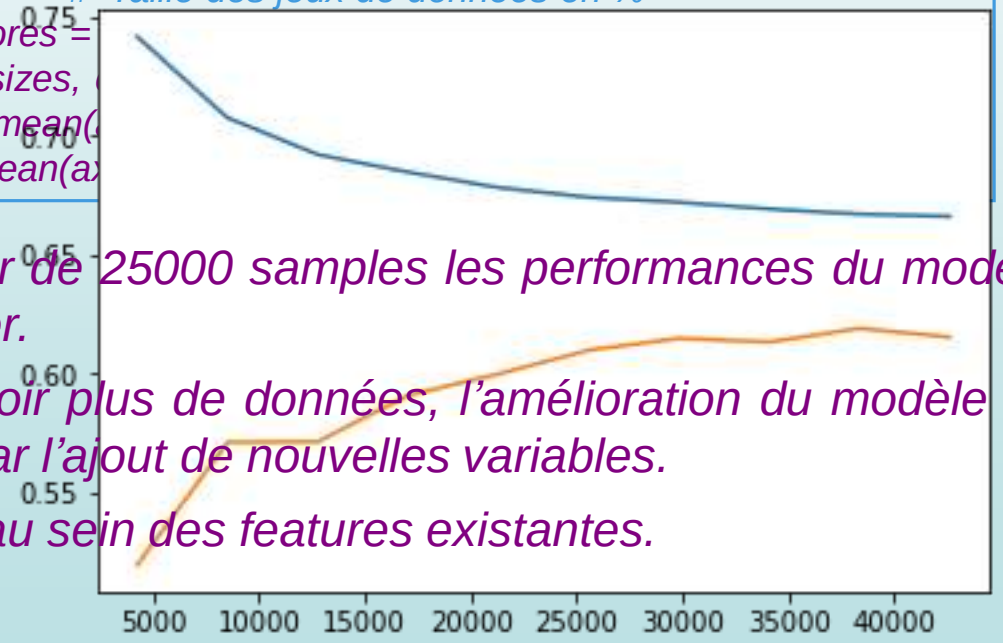
- *Les courbes d'apprentissage permettent de voir comment les erreurs sur la prédiction évoluent en fonction de la taille de l'ensemble.*
- *Ces courbes donnent des informations sur le biais et la variance dans un algorithme d'apprentissage.*
- *Le biais correspond à la différence entre la prédiction moyenne d'un modèle et la valeur réelle.*
- *La variance mesure à les écarts entre d'erreurs en fonction de la variation des jeux de données d'apprentissage.*
- *Le but étant d'obtenir un modèle avec une faible variance et un faible biais. Pour diminuer le biais il faudrait un jeu d'entrainement élevé, mais alors la variance du modèle augmentera.*
- *Il faut donc trouver un compromis au travers des courbes de validation.*
- *la fonction learning_curve entraine le modèle sur des jeux de données de tailles différents et retourne trois variables les tailles des différents jeux de test, les scores sur la partie train et sur la partie validation.*

Apprentissage : Airbnb prédiction

Les courbes d'apprentissage

- *Les courbes d'apprentissage sont un excellent outil de diagnostic pour déterminer le biais et la variance dans un algorithme d'apprentissage.*

```
from sklearn.model_selection import learning_curve
def courbes d'apprentissage(model, X_train, y_train) :
    sizes = np.linspace(0.1, 1, 10)          #  Taille des jeux de données en %
    train_sizes, train_scores, val_scores =
                                train_sizes=sizes, (
    plt.plot(train_sizes, train_scores.mean(
    plt.plot(train_sizes, val_scores.mean(ax
```



- *On constate qu'à partir de 25000 samples les performances du modèle ne semblent plus évoluer.*
- *Il est donc inutile d'avoir plus de données, l'amélioration du modèle ne peut donc passer que par l'ajout de nouvelles variables.*
- *Ou plus de variabilité au sein des features existantes.*

Apprentissage : Airbnb prédiction

Changement de modèle

- *Les RandomForest découpent un dataset en plusieurs groupes, puis à proposer un modèle d'entraînement à chacun de ses groupes.*
- *Les résultats sont recombinés pour obtenir des prévisions solides.*

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(max_depth = 20, n_jobs = -1, n_estimators = 700)
evaluation(model, X_train, X_test, y_train, y_test)          #  score R2 : 0.68
```

- *Le modèle est amélioré, l'idée maintenant consiste à sélectionner uniquement les meilleurs features pour voir si les résultats sont meilleurs.*
- *On utilise ici la notion de pipeline qui consiste à créer des modèles qui combinent différentes actions entre elles.*

```
from sklearn.feature_selection import SelectKBest, f_classif  #  test anova pour la sélection
from sklearn.pipeline import Pipeline        #  Pipeline pour combiner la sélection et le model
model = Pipeline( steps= [('select features', SelectKBest(f_classif, k=30)),
                          ('RandomForest', RandomForestRegressor(max_depth = 20, n_jobs = -1,
                           random_state = 0, n_estimators = 700))])
evaluation(model, X_train, X_test, y_train, y_test)          #  score R2 : 0.62
```

Apprentissage : Airbnb prédiction

Recherche des meilleurs paramètres d'un modèle

- *Tous les modèles disposent d'un certain nombre d'hyper-paramètres.*
- *Pour trouver le meilleur réglage en fonction du jeu de données, il est nécessaire de tester un grand nombre de combinaisons possibles.*
- *GridSearchCV teste ces différentes combinaisons.*
- *A partir d'un dictionnaire contenant les différents paramètres à régler associés aux différentes valeurs à tester.*
- *GridSearchCV est construit à partir du modèle, du dictionnaire et de la taille de la cross validation*

```
from sklearn.model_selection import GridSearchCV
#  Paramètres à régler soit : 7 * 2 * 10 * 4
param_grid = { 'n_estimators' : np.arange(100, 800, 100)  ,  'max_features' : ['auto', 'sqrt']
            'max_depth' : np.aragce(10, 110, 10) , 'min_samples_split' : np.arance(2, 25, 5) }
model = GridSearchCV(RandomForestRegressor(), param_grid, cv=5)
model.fit(X_train, y_train)                    #  Entrainement du model
model.best_score_                              #  Meilleur score
model.best_params_                             #  Meilleurs paramètres
bestModel = model.best_estimator_              #  Meilleur model
```

Apprentissage : Airbnb prédiction

Autres modèles

- *Les modèles de la bibliothèque linear régression donnent des résultats aux alentours de 0.66 qui n'améliorent pas vraiment les résultats*
- *L'utilisation d'un réseau de neurone ne fournit pas de meilleurs résultats.*

```
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
model = Pipeline( steps= [('scalar', StandardScaler()), ('select features', SelectKBest(f_classif,
        k=70)), ('Neural Network', MLPRegressor(solver='sgd', activation = 'logistic',
        random_state = 0, max_iter = 300, hidden_layer_sizes=5))])
evaluation(model, X_train, X_test, y_train, y_test)          # # score R2 : 0.65
```

- *En choisissant un support vector machine les résultats obtenus sont meilleurs.*

```
from sklearn.svm import SVR
model = SVR(degree=3)
evaluation(model, X_train, X_test, y_train, y_test)          # # score R2 : 0.69
model = SVR(degree=5)
evaluation(model, X_train, X_test, y_train, y_test)          # # score R2 : 0.69
```

Compléments

Mise à l'échelle des données

- *La plupart des modèles sont sensibles aux variations d'échelle des valeurs entre les différentes features.*
- *En particulier ceux dont les techniques d'apprentissage sont basées sur la descente de gradient.*
- *Plusieurs techniques de normalisation des données peuvent être utilisées : MaxAbsScalar, MinMaxScalar, StandardScalar ou RobustScalar*
- *MaxAbsScalar : bien adaptée si les données contiennent des outliers.*
- *MinMaxScalar : est adaptée pour des données dont les distributions ne sont pas gaussiennes, mais est sensible aux outliers.*
- *StandardScalar : à utiliser si les données sont normalement distribuées.*
- *RobustScalar : utilise la même technique que MinMaxScalar mais est moins sensible aux outliers.*

$$X_{MaSc} = \frac{X}{X_{max}} \qquad X_{MMSc} = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad X_{StSc} = \frac{X - \bar{X}}{S} \qquad X_{RoSc} = \frac{X - X_{mediane}}{IQR}$$

Compléments

Variance

- *Mesure la moyenne des écarts au carré entre les valeurs d'une série et leur moyenne. Elle mesure la dispersion des valeurs entre elles.*

Variance $s^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$

N est le nombre d'éléments de la série
x_i *la ième valeur de la série*
$\bar{x}$ *la moyenne de la série*
s est l'écart type

Covariance

- *Evalue le sens de variation de deux variables et donc leur dépendance.*

$$\sigma_{xy} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y}) = \overline{xy} - \bar{x}\bar{y}$$

Coefficient de corrélation

- *Quantifie la force de la relation linéaire entre deux variables.*
- *Les valeurs varient entre -1 et 1 (0 => aucune corrélation)*

$$\rho_{XY} = \frac{\sigma_{xy}}{\sqrt{s^2(x) * s^2(y)}}$$

Compléments

Tests Anova

- *Le test est utilisé pour comparer la moyenne de plusieurs variables afin de mesurer si elles sont liées.*
- *La valeur du teste Anova correspond au rapport entre la somme des erreurs inter variables et la somme des erreurs intra.*
- *Si m est le nombre d'individus, et n le nombre de variables*

$$F = \frac{SCE_{inter}/DDL_{inter}}{SCE_{intra}/DDL_{intra}}$$

$$SCE_{inter} = \sum_{j}^{n} m_j \left(\bar{x_j} - \bar{x}\right)^2 \qquad DDL_{inter} = n - 1$$

$$SCE_{intra} = \sum_{j}^{n} \sum_{i=1}^{m} \left(x_{i,j} - \bar{x_j}\right)^2 \qquad DDL_{intra} = n * (m - 1)$$

- *On doit fixer un seuil d'acceptabilité généralement α =5 % = 0.05*
- *On compare ensuite la valeur de F avec la valeur donnée par la table de fisher pour un seuil de 5%.*
- *Si la valeur de F est supérieur à cette valeur les variables sont liées.*

Compléments

Tests Anova : exemple

- *Les prix dépendent-ils du resto. n=3 (catégories) et m=6 (individus)*

Resto 1	Resto 2	Resto 3
18	51	24
23	45	29
41	42	23
62	59	11
32	47	32
34	27	28

Moyennes des variables

$\overline{x_1}$	$\overline{x_2}$	$\overline{x_3}$
35.0	45.2	24.5

Moyenne globale $\bar{x} = 34.9$

$DDL_{intra} = n * (m-1) = 15$

$DDL_{inter} = n - 1 = 2$

$$SCE = SCE_{inter} + SCE_{intra} = \sum_{j}^{n}\sum_{i=1}^{m}\left(x_{i,j} - \bar{x}\right)^2$$

Coefficient d'Anova $$F = \frac{SCE_{inter}/DDL_{inter}}{SCE_{intra}/DDL_{intra}}$$

$$SCE_{inter} = \sum_{j}^{3} 6 * \left(\overline{x_j} - \bar{x}\right)^2 = 1201.0$$

$F = 4.69$

$$SCE_{intra} = \sum_{j}^{n}\sum_{i=1}^{m}\left(x_{i,j} - \overline{x_j}\right)^2 = 1985.0$$

Il y a une probabilité de 5% que la valeur soit supérieur à 3.68. F=4.69 permet de rejeter l'hypothèse H_0 (toutes les variables ont la même moyenne) et donc on a l'hypothèse H_1 qui implique que les prix dépendent du restaurant.

Tests Statistique F1

- *La matrice de confusion ou matrice d'erreur est un tableau qui permet de visualiser les performances d'un algorithme d'apprentissage supervisé.*
- *Comme souvent il y à plus de valeurs d'une classe que d'une autre, une étude directe va biaiser les résultats vers la valeur la plus haute.*

$valeurs\ réelles$ →

$valeurs\ prédites$ ↕

	Positifs	Négatifs	
Positifs	Vrais-Positifs	Faux-Positifs	Recall VP/(VP+FP)
Négatifs	Faux-Négatifs	Vrais-Négatifs	Sensitivity VN/(FN+VN)
	Precision VP/(VP+FN)	Specificity VN/(FP+VN)	Accuracy VP+VN/(total)

Si nous avons plus de deux classes à prédire – trois (A,B,C) il faut faire trois tableau A / (B+C) ; B /(A+C) et C / (A+B)

$$Score\ F1 = 2 * \frac{Precision(i) * Recall(i)}{Precision(i) + Recall(i)}$$