

MACHINE LEARNING PAR LA PRATIQUE AVEC PYTHON

PROJETS REELS DANS LES FINANCES,
L'IMMOBILIER, LE TRADING, LA
SANTE, LE MARKETING, ETC.



Parfois j'ai un
Superviseur et
parfois j'en ai
pas.

JOSUÉ AFouda

MACHINE LEARNING PAR LA PRATIQUE AVEC PYTHON :

PROJETS REELS DANS LES FINANCES, L'IMMOBILIER,
LE TRADING, LA SANTE, LE MARKETING, ETC.

Par [Josué AFOUDA](#)

Août 2020

A travers ce livre, vous apprendrez à :

- Utiliser la librairie Scikit-Learn destinée à l'apprentissage automatique ;
- Résoudre des problèmes de Classification et de Régression ;
- Construire un modèle de prévision des futures valeurs d'une série temporelle ;
- Utiliser les algorithmes populaires dans le monde réel comme *Linear Regression*, *Random Forest*, *Logistic Regression*, *KMeans*, etc. ;
- Choisir le(s) métrique(s) convenable(s) pour l'évaluation de la performance d'un modèle de Machine Learning ;
- Valider efficacement un modèle à l'aide de techniques comme *cross-validation* ;
- Rechercher les hyperparamètres optimaux d'un modèle de Machine Learning ;
- Réduire la dimension d'un ensemble de données avec l'Analyse en Composantes Principales ;
- Automatiser le flux de travail d'un projet de Machine Learning ;
- Utiliser des outils d'assistance intelligente comme TPOT dans la construction d'un modèle de Machine Learning.

A qui est destiné ce livre ?

Ce livre est écrit dans un langage simple et facile à comprendre. Il peut être utilisé, en toute autonomie, par n'importe quelle personne souhaitant utiliser les outils de Data Science et de Machine Learning pour résoudre des problèmes réels en entreprise. Le livre est destiné :

- Aux personnes en reconversion professionnelle intéressées par la Science des données qui regorge d'innombrables opportunités d'emploi ;
- Aux personnes ayant des bases en programmation avec le langage Python et qui souhaitent comprendre le Machine Learning ;
- Aux étudiants dans des filières DATA qui souhaitent pratiquer les enseignements théoriques reçus à l'Université ;
- Aux Formateurs pour les aider dans la conception de leurs cours sur le Machine Learning ;
- À tout le monde 😊

Comment tirer le meilleur parti de ce livre ?

La méthodologie de travail en Data Science peut être appliquée dans la résolution de n'importe quel problème dans la vie réelle qui nécessite le traitement de données. De ce fait, tout le monde peut utiliser les outils de la Data Science et ceci dans n'importe quel domaine. L'apprentissage automatique offre la possibilité de régler des problèmes de la vie quotidienne sans pour autant être un « EXPERT » en Mathématiques ou en codage informatique. Vous pouvez aussi comprendre le Machine Learning et l'utiliser dans vos travaux. Comment ?

Que vous ayez le livre en version broché ou numérique, il est primordial de créer vous-mêmes vos notebooks et de saisir les codes afin de les exécuter et d'analyser les résultats. Ce livre est pratique à 100%. Ce sont de réels projets qui y sont traités. Donc pour chaque projet, récupérer les données et créez votre notebook pour écrire les codes. Analysez les résultats.

L'auteur Josué AFOUDA :

Josué AFOUDA est un passionné de programmation informatique et d'analyse de données. Après l'obtention d'un Baccalauréat option Mathématiques et Sciences Physiques, il obtient successivement un Master 2 en Hydrologie spécialisation Modélisation des écoulements d'eaux de surface puis un autre Master 2 en Qualité et Traitement des eaux. Il a beaucoup travaillé dans le monde de la recherche scientifique sur des sujets tels que l'analyse fréquentielle des précipitations, la modélisation stochastique, le Changement Climatique, l'Eau, l'Assainissement, le Machine Learning, etc. Ces missions de travail lui ont permis d'être toujours en contact avec les données. Autodidacte, il continue de se former chaque jour sur les dernières avancées en Data Science, Machine Learning, Deep Learning, etc. Josué AFOUDA aime aussi beaucoup écrire, enseigner les Mathématiques et la Programmation. Aujourd'hui, en

tant que Formateur Professionnel, il souhaite attirer des personnes, même sans compétences, en programmation dans le domaine de la Data Science pour les aider à prendre part aux multiples opportunités de carrière dans ce domaine. De plus, en tant que Consultant il aide, les chercheurs, les Associations, ainsi que les entreprises à faire parler leurs données afin de les aider à prendre de meilleures décisions pour leurs Projets.

DONNEES UTILISEES DANS CE LIVRE

Vous pouvez télécharger toutes les données utilisées dans ce livre ainsi que d'autres données pour vous exercer à travers ce lien :

<https://github.com/JosueAfouda/Machine-Learning-par-la-pratique-avec-Python/archive/master.zip>

Par ailleurs, vous pouvez utiliser directement les URL de chaque fichier de données dans vos notebooks ce qui vous dispense de les télécharger.

Table des matières

GENERALITES	1
INTRODUCTION.....	1
UN MONDE TRES COMPLEXE ET CONFUS	1
LE MACHINE LEARNING.....	2
PROJET 0 : ANALYSE D'UNE CAMPAGNE MARKETING AVEC PANDAS.....	7
PROJET 1 : PREDICTION DU PRIX D'UNE VOITURE EN FONCTION DE SES CARACTERISTIQUES	31
PROJET 2 : PREDICTION DU PRIX D'UNE MAISON EN FONCTION DE SES CARACTERISTIQUES.....	52
PROJET 3 : MODELISATION DU RISQUE DE CREDIT (MODELE DE SCORING)	65
PROJET 4 : CONSTRUCTION D'UN MODELE DE PREDICTION DU CANCER DU SEIN.....	91
PROJET 5 : PREDICTION DES PRIX DES MAISONS	98
PROJET 6 : MODELE DE PREDICTION DES INTENTIONS D'ACHATS DES UTILISATEURS D'UNE BOUTIQUE EN LIGNE	103
PROJET 7 : CLUSTERING AVEC KMEANS.....	114
PROJET 8 : SEGMENTATION DE LA CLIENTELE D'UNE ENTREPRISE	132
PROJET 9 : REDUCTION DE LA DIMENSION D'UN ENSEMBLE DE DONNEES PAR L'ANALYSE EN COMPOSANTES PRINCIPALES.....	137
PROJET 10 : CONSTRUCTION D'UN MODELE DE PREDICTION DES PUBLICITES SUR UN SITE WEB	152
PROJET 11 : COMBINAISON DES TECHNIQUES DE CLUSTERING ET D'ACP POUR LA SEGMENTATION DE CLIENTELE.....	162
PROJET 12 : MODELISATION DES COURS BOURSIERS.....	178
PROJET 13 : UTILISATION DE L'OUTIL TPOT COMME ASSISTANT INTELLIGENT DANS UN PROJET DE MACHINE LEARNING.....	200
PROJET 14 : AUTOMATISATION DU WORKFLOW D'UN PROJET DE MACHINE LEARNING AVEC LA FONCTION PIPELINE	210

GENERALITES

INTRODUCTION

Vous avez sûrement déjà entendu parler au moins une fois dans votre vie d'Intelligence artificielle, de Machine Learning, de Big Data, de Data Science. Ces concepts, comme des phénomènes de mode, sont mentionnés partout parfois à tort ou à raison. La communauté scientifique n'est même pas unanime sur les appellations et chacun développe sa propre philosophie derrière chacun de ces concepts.

L'objectif de cette introduction générale à ce livre ne sera donc pas de s'attarder sur la forme, les termes, mais sur le fond et le contenu de ces mots. Je parlerai surtout de Machine Learning qu'on peut traduire littéralement par Apprentissage automatique. Dans la suite j'emploierai très souvent le terme « Machine Learning ».

Que faut -il comprendre par Machine Learning ? Comment ce terme est-il apparu ? Quels sont les types de Machine Learning ? A travers des exemples concrets et sans aller en profondeur dans l'aspect technique, je vous aiderai à comprendre ce concept. De plus, je vous aiderai à comprendre trois tâches classiques de Machine Learning : La Régression, La Classification et la Segmentation (Clustering).

UN MONDE TRES COMPLEXE ET CONFUS

L'informatique a connu une évolution fulgurante. De la règle à calculer, en passant par les gros ordinateurs, nous en sommes aujourd'hui à des machines hyperpuissantes qui tiennent dans notre main c'est-à-dire les smartphones. De ce fait, notre perception de l'informatique a aussi évolué et on ne fournit pratiquement aucun effort pour accéder à l'informatique. Aujourd'hui, on n'a pas besoin d'être un surdoué en Mathématiques ou en codage pour profiter pleinement de toutes les possibilités de l'outil Informatique. D'ailleurs, le "purement logique et mathématique" n'a pratiquement plus d'adeptes. Il est vrai que les Mathématiciens ont su résoudre de grandes équations. Certains ont même écrit de grands algorithmes pour approcher numériquement les solutions de certaines équations complexes. Mais quel intérêt pour le commun des mortels ? Très vite, les scientifiques se sont rendu compte que le vrai défi était de résoudre des problèmes de la vie quotidienne et non d'écrire des millions de ligne de code. Mais notre monde est tellement complexe, désordonné, multiforme, confus qu'il est incapable de l'appréhender par les seules règles logiques.

Alors que faire ?

LE MACHINE LEARNING

Dès lors qu'on s'est rendu compte que les problèmes réels de la vie ne pouvaient être réglé par les seules règles mathématiques et logiques, on a imaginé un système capable d'apprendre. De là est né le Machine Learning (Apprentissage automatique) qui est un domaine de l'Intelligence artificielle.

Selon Wikipédia, L'[apprentissage automatique](#)¹ (*Machine Learning* en anglais) est un champ d'étude de l'[Intelligence artificielle](#)² qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d' apprendre à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés pour chacune de ces tâches. Plus largement, il concerne la conception, l'analyse, l'optimisation, le développement et l'implémentation de telles méthodes. En clair, le système apprend à partir des exemples qu'on lui montre. Ces exemples sont des données qui sont rentrées dans le système par des hommes.

Il existe déjà plusieurs cas d'apprentissage automatique que nous expérimentons déjà au quotidien et d'autres qui sont en cours de développement et de vulgarisation. On peut citer entre autres :

- **Le classement de nos mails** : Par exemple dans la messagerie de Google, nous avons des mails qui viennent dans notre boîte principale, d'autres sont rangés dans la catégorie "Réseaux sociaux" ou "Promotions". D'autres vont systématiquement dans la catégorie "spam". Ce n'est pas nous-mêmes qui faisons ce classement. Google a entraîné son système de messagerie à faire ce classement de manière automatique.
- **Les systèmes de reconnaissance d'image** : On a entraîné les systèmes à reconnaître une image spécifique. Par exemple, dans les voitures autonomes, le système peut distinguer un panneau stop, une bande piétonne, un feu rouge, etc. Il y a aussi des systèmes reconnaissance faciale qui peuvent reconnaître le visage d'une personne parmi une multitude d'images.
- **Le diagnostic médical** : Sur la base des données de patients, la machine peut dire si un individu est atteint par exemple du cancer.

Il y a d'autres exemples comme les logiciels de traduction de langues, reconnaissance vocale, etc.

Les exemples de Machine Learning sont légion et d'autres applications sont créées chaque jour.

Mais comment la Machine apprend -elle et "réussit" à faire ce qu'on attend d'elle ?

Dans tous les exemples cités ci-dessus, ce sont les hommes qui ont entraîné le système à faire une tâche spécifique. Par exemple, pour apprendre à un ordinateur à reconnaître un chat, on lui montre plusieurs photos de chat. Le système intègre ces informations en son sein. On le teste ensuite afin de s'assurer qu'il soit capable de distinguer un chat d'un chien par exemple. Pour ce faire il faut lui montrer des centaines voire des milliers d'images. En effet, plus on dispose de données d'entrées, plus le système apprend mieux et sa performance est élevée. Heureusement, aujourd'hui nous sommes à l'ère du [Big Data](#)³. Le monde génère chaque jour des milliards de milliards de données de tout type. L'exploitation de ces données permet le développement d'intelligences artificielles de plus en plus performantes qui nous aident à régler des problèmes de la vie quotidienne.

L'apprentissage automatique révolutionne notre façon de résoudre les problèmes quotidiens et améliore nos vies dans plusieurs domaines tels que la santé, l'industrie, l'environnement, la finance, etc. La machine apprend de plusieurs exemples qu'on lui montre. Le plus difficile pour le commun des mortels est peut-être l'écriture des algorithmes d'apprentissage. Mais heureusement, des grandes sociétés comme Google développent ces algorithmes d'apprentissage en mode Open Source de telle façon que le Machine Learning est accessible à n'importe quelle personne ayant la volonté d'apprendre. Vous n'êtes pas obligé d'avoir un Doctorat en Mathématiques et en Informatique pour faire du Machine Learning. Il y a plein de ressources sur Internet, dont ce livre 😊 qui peuvent vous aider à démarrer et à être acteur de cette révolution.

Je vous invite à regarder cette [vidéo](#)⁴ intéressante sur l'apprentissage automatique. Dans la suite, je vous aiderai à comprendre trois techniques classiques de Machine Learning. En effet, il existe trois (02) types majeurs d'apprentissage automatique :

- **Apprentissage supervisé** (*Supervised Learning* en anglais) : l'apprentissage supervisé fait les prédictions sur la base de données étiquetées. En apprentissage supervisé on a deux (02) types de problème : les problèmes de **Régression**, lorsqu'il s'agit de prédire une variable quantitative et les problèmes de **Classification**, lorsqu'il s'agit de prédire une variable qualitative.
- **Apprentissage non supervisé** (*Unsupervised Learning* en anglais) : Contrairement à l'apprentissage supervisé, le but d'un algorithme d'apprentissage non supervisé est de trouver lui-même des structures (ou relations) sous-jacentes dans les données non étiquetées.

REGRESSION

La Régression est une technique classique d'apprentissage automatique dont le but est de prédire une variable quantitative en fonction d'autres variables. Partons d'un exemple pour comprendre les problèmes de Régression. Imaginez que vous ayez un échantillon de personnes avec les données sur leur poids et leur taille. L'objectif est de trouver une relation c'est-à-dire une fonction $y = a*x + b$ qui donne la taille d'une personne (y) en fonction de son poids (x). Cette fonction est un modèle de [Régression⁵](#). Dans cet exemple, il s'agit d'une régression linéaire.

Vous pouvez ensuite utiliser votre modèle pour prédire la taille d'une personne (qui n'était pas dans votre jeu de données) connaissant son poids. Il est très important de garder à l'esprit que dans les problèmes de Régression, la variable-cible (y) est une variable quantitative et que vous avez besoin de données étiquetées pour construire votre modèle. Dans le cas de notre modèle qui donne la taille en fonction du poids, nous avons utilisé un jeu de données où nous avons pour chaque observation (chaque personne est une observation) son poids et sa taille. Il s'agit donc bel et bien de données étiquetées.

On peut citer d'autres exemples d'application de Régression :

- La prédiction du prix des maisons en fonction de leurs caractéristiques (superficie, localisation, parking ou non, etc.) ;
- Prédiction du taux de change de l'euro en dollar en Janvier 2020 ;
- Prédiction du chiffre d'affaire d'une société ;
- Etc.

CLASSIFICATION

La Classification est une technique d'apprentissage automatique dont le but est de prédire des classes ou catégories de nouvelles observations. Ici, on prédit une variable qualitative discrète contrairement à un problème de régression où on prédit une variable quantitative.

Partons encore d'un exemple. Supposons que vous avez un jeu de données avec par exemple comme variables le taux de cholestérol, le poids, le sexe et une variable "malade" qui est égale à oui si le patient est atteint d'une maladie cardio-vasculaire et non dans le cas contraire. Vous pouvez construire un modèle de classification qui s'ajuste bien à vos données. Et à partir de ce modèle, vous pouvez prédire si un nouveau patient sera malade ou pas. Il est clair que c'est une manière simpliste d'expliquer la chose et que le Machine Learning n'a pas vocation à remplacer les médecins ! L'apprentissage automatique est déjà en train de révolutionner le domaine de la

Santé. Plusieurs études montrent qu'il améliore les diagnostics, l'infrastructure clinique et les soins préventifs.

Nous pouvons citer d'autres exemples de problème de classification :

- Utilisation de données financières étiquetées pour prédire si la valeur d'une action augmentera ou diminuera la semaine prochaine. Dans cet exemple, il y a deux résultats qualitatifs distincts : le marché boursier à la hausse et le marché boursier à la baisse. Cela peut être représenté à l'aide d'une variable binaire et constitue une application parfaitement adaptée à la classification.
- Prédire si un candidat à un prêt bancaire sera en défaut de paiement ou pas est un autre exemple de problème de classification.
- Analyser des images d'animaux et prédire si l'image est celle d'un chien, d'un chat ou d'un panda.
- Etc.

Il est important de garder à l'esprit que pour les problèmes de classification, la variable-cible est une variable qualitative et qu'on utilise des données étiquetées comme pour la Régression.

La Régression et la Classification constituent l'apprentissage automatique supervisé. C'est le type d'apprentissage automatique qui fait les prédictions à partir de données étiquetées contrairement à l'apprentissage automatique non supervisé.

L'apprentissage non supervisé englobe une variété de techniques d'apprentissage automatique comme la segmentation (ou clustering) et la réduction de dimension.

SEGMENTATION (OU CLUSTERING)

Comprenez le mot Clustering comme Regroupement en classes. Le Clustering est une technique de Machine Learning dont l'objectif est de regrouper les observations. Ici on ne dispose pas de données étiquetées. L'algorithme regroupe les objets similaires en termes de leurs caractéristiques. Dans chaque groupe, on a donc des observations similaires mais chaque groupe est différent d'un autre groupe.

Cette technique est beaucoup utilisée en Marketing. On groupe les clients similaires entre eux afin de faire des publicités ciblées.

CONCLUSION

Le Machine Learning (apprentissage automatique) est un outil révolutionnaire et efficace qui est aussi bien utile dans les entreprises pour le business que pour les citoyens ordinaires. De plus grâce aux données massives que nous générerons, les champs d'application et les opportunités de développement grâce à cet outil sont énormes. Chaque personne peut prendre part à cette révolution et gagner en efficacité et en rentabilité.

Pour finir, je vous invite à regarder cette [vidéo](#)⁶. Si vous êtes un débutant et que vous aspirez à devenir un **praticien** du machine Learning, ce livre est parfaitement fait pour vous.

Le premier projet est un projet n°0 (comme l'indexation dan Python 😊). Ce projet vous permettra de revoir quelques fonctionnalités de Pandas, Numpy, Matplotlib et Seaborn utiles dans une analyse de données.

N.B : Il est indispensable d'avoir les bases en programmation avec le langage Python appliquée à l'analyse des données avant de pouvoir apprendre le Machine Learning. Si ce n'est pas votre cas, je vous conseille de vous mettre à jour à travers ce livre de formation :

[**SAVOIR PROGRAMMER AVEC LE LANGAGE PYTHON APPLIQUE A L'ANALYSE DES DONNEES : Cours, Exercices corrigés et Projets réels pour comprendre les fondamentaux de la programmation informatique avec le langage Python appliqué à la Science des données**](#)⁷.

PROJET 0 : ANALYSE D'UNE CAMPAGNE MARKETING AVEC PANDAS

INTRODUCTION

Si avoir des compétences techniques telles que la programmation, les statistiques, le nettoyage de données, le Machine Learning, ..., est absolument incontournable pour un Data Scientist, le plus grand défi est de comprendre comment ces compétences et concepts se traduisent dans le monde réel et de pouvoir les appliquer pour résoudre des problèmes en entreprises. L'un des domaines majeurs où la Data Science peut s'appliquer dans une entreprise est le Marketing. En effet dans une équipe marketing, un Data Scientist a pour rôle d'aider l'entreprise à comprendre l'impact de ses campagnes marketing. Il peut donc effectuer plusieurs tâches dont :

- L'analyse d'indicateurs clés de performance ;
- L'analyse du fonctionnement des différents canaux marketing (Ex : Combien de nouveaux utilisateurs s'abonnent par suite de l'envoi de courriels ; Compte tenu des taux de conversion et des revenus actuels, est-ce que l'entreprise doit continuer à investir dans tel ou tel autre canal et combien doit-elle dépenser ? etc.) ;
- L'exécution d'expériences (A/B Tests) pour comprendre l'impact d'un changement particulier.

A travers ce projet, vous serez entraîné à traduire les questions commerciales courantes en résultats mesurables, notamment : "Comment une campagne marketing a-t-elle fonctionné ?", "Quel canal d'acquisition réfère le plus d'abonnés ?", "Pourquoi un canal particulier est-il sous-performant ?" Nous utiliserons les données marketing d'une entreprise d'abonnement en ligne.

LIBRAIRIES

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

DONNEES

```
# Importation des données
marketing_df = pd.read_csv('https://raw.githubusercontent.com/
JosueAfouda/Marketing-Pandas/master/marketing.csv')
```

```
# Affichage des données

print(marketing_df.info())

print('\n')

marketing_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10037 entries, 0 to 10036
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
--- 
 0   user_id         10037 non-null   object 
 1   date_served    10021 non-null   object 
 2   marketing_channel 10022 non-null   object 
 3   variant        10037 non-null   object 
 4   converted      10022 non-null   object 
 5   language_displayed 10037 non-null   object 
 6   language_preferred 10037 non-null   object 
 7   age_group       10037 non-null   object 
 8   date_subscribed 1856 non-null   object 
 9   date_canceled   577 non-null    object 
 10  subscribing_channel 1856 non-null   object 
 11  is_retained     1856 non-null   object 
dtypes: object(12)
memory usage: 941.1+ KB
None

user_id  date_served  marketing_channel  variant  converted  language_displayed  language_preferred  age_group  date_subscribed  date_canceled  subscribing_channel  is_retained
0  a100000029  1/1/18  House Ads  personalization  True  English  English  0-18 years  1/1/18  NaN  House Ads  True
1  a100000030  1/1/18  House Ads  personalization  True  English  English  19-24 years  1/1/18  NaN  House Ads  True
2  a100000031  1/1/18  House Ads  personalization  True  English  English  24-30 years  1/1/18  NaN  House Ads  True
3  a100000032  1/1/18  House Ads  personalization  True  English  English  30-36 years  1/1/18  NaN  House Ads  True
4  a100000033  1/1/18  House Ads  personalization  True  English  English  36-45 years  1/1/18  NaN  House Ads  True
```

Les variables ***date_served***, ***date_subscribed*** et ***date_canceled*** sont des dates et ne sont pas au bon format. Nous mettrons ces variables au format 'Datetime'.

```
# Résumé statistique
```

```
marketing_df.describe().T
```

	count	unique	top	freq
user_id	10037	7309	a100000882	12
date_served	10021	31	1/15/18	789
marketing_channel	10022	5	House Ads	4733
variant	10037	2	control	5091
converted	10022	2	False	8946
language_displayed	10037	4	English	9793
language_preferred	10037	4	English	9275
age_group	10037	7	19-24 years	1682
date_subscribed	1856	31	1/16/18	163
date_canceled	577	115	4/2/18	15
subscribing_channel	1856	5	Instagram	600
is_retained	1856	2	True	1279

En utilisant les données, répondrons à des questions très importantes pour l'entreprise.

CANAUX MARKETING UTILISES PAR L'ENTREPRISE

```
marketing_df['marketing_channel'].value_counts()
```

```
House Ads      4733
Instagram     1871
Facebook      1860
Push          993
Email          565
Name: marketing_channel, dtype: int64
```

Pour avoir plutôt les proportions :

```
marketing_df['marketing_channel'].value_counts(normalize = True)
House Ads      0.472261
Instagram     0.186689
Facebook      0.185592
Push          0.099082
Email          0.056376
Name: marketing_channel, dtype: float64
```

L'entreprise utilise majoritairement (un peu plus de 47%) les annonces auto-promotionnelle.

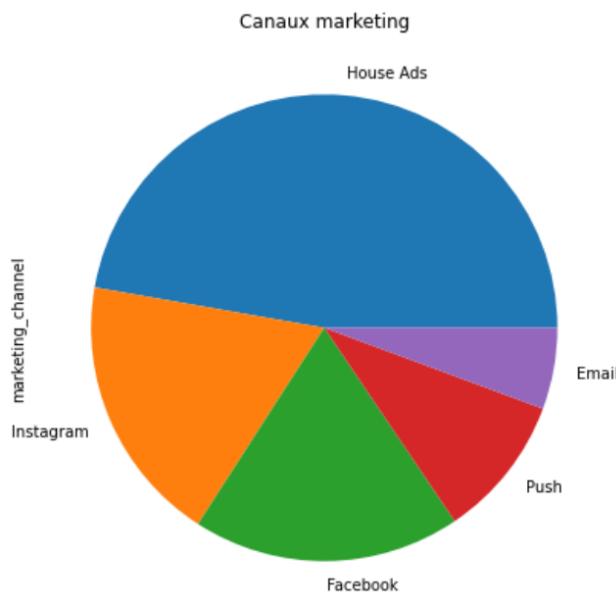
```
# Canaux marketing
```

```
plt.figure(figsize=(7, 7))
```

```
marketing_df['marketing_channel'].value_counts(normalize = True).plot(kind = 'pie')
```

```
plt.title("Canaux marketing")
```

```
plt.show()
```



VARIABLES DE DATES AU FORMAT DATETIME

```
# Changement du type de certaines colonnes
for col in ['date_served', 'date_subscribed', 'date_canceled']
:
    marketing_df[col] = pd.to_datetime(marketing_df[col])

# Vérification

marketing_df.dtypes

user_id                  object
date_served              datetime64[ns]
marketing_channel        object
variant                  object
converted                object
language_displayed       object
language_preferred       object
age_group                object
date_subscribed          datetime64[ns]
date_canceled            datetime64[ns]
subscribing_channel     object
is_retained              object
dtype: object
```

Les variables **date_served**, **date_subscribed** et **date_canceled** sont maintenant dans un bon format ('datetime').

Quel est le nombre d'utilisateurs retenus par la campagne marketing ?

```
# Nombre d'utilisateurs retenus
marketing_df['is_retained'].sum()
```

1279 utilisateurs ont été retenus par la campagne marketing. En proportion, cela équivaut à près de 0,69% de tous les utilisateurs.

```
marketing_df['is_retained'].value_counts(normalize=True)

True      0.689116
False     0.310884
Name: is_retained, dtype: float64
```

CREATION DE NOUVELLES COLONNES

Lors d'une analyse de données, on peut être amené à créer de nouvelles variables à partir de celles existantes.

Nous voulons créer une nouvelle variable par encodage des modalités de la variable ***subscribing_channel*** :

```
# Dictionnaire des chaînes distinctes de souscription avec leurs codes
dict_channels = {'House Ads':1, 'Instagram':2, 'Facebook':3, 'Push':4, 'Email':5}

# Création de la colonne 'channel_code'
marketing_df['channel_code'] = marketing_df['subscribing_channel'].map(dict_channels)
```

Ajoutons une nouvelle colonne ***is_correct_lang*** qui est égale à 'Yes' si l'utilisateur a vu l'annonce marketing dans sa langue préférée et 'No' dans le cas contraire.

```
marketing_df['is_correct_lang'] = np.where(marketing_df['language_displayed']==marketing_df['language_preferred'],
                                             'Yes', 'No')

# Ajout d'une colonne 'DoW' qui représente le jour de la semaine
marketing_df['DoW'] = marketing_df['date_subscribed'].dt.dayofweek
```

UTILISATEURS UNIQUES QUI VOIENT LES ANNONCES MARKETING CHAQUE JOUR

Ceci est crucial pour comprendre l'efficacité des efforts marketing au cours du dernier mois.

Pour répondre à cette question, il faut grouper les observations (lignes de la dataframe) par la variable ***date_served*** et compter le nombre d'utilisateurs (variable ***user_id***).

```
daily_users = marketing_df.groupby('date_served')['user_id'].nunique()

# Visualisation des résultats
plt.figure(figsize=(15, 7))

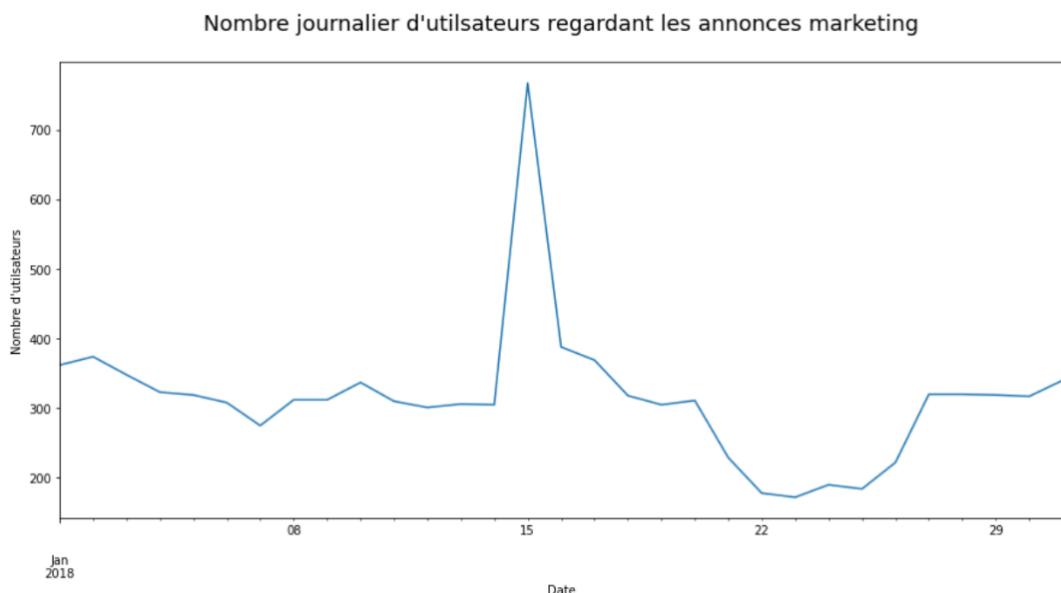
daily_users.plot.line()

plt.xlabel('Date')

plt.ylabel("Nombre d'utilisateurs")
```

```
plt.title("Nombre journalier d'utilisateurs regardant les annonces marketing\n", fontsize = 18)
```

```
plt.show()
```



Alors que la première moitié du mois enregistre entre 300 et 400 utilisateurs par jour, il y a eu un énorme pic au milieu du mois. Cela peut être dû au fait que l'entreprise a déployé une grosse campagne par mail qui a atteint plusieurs utilisateurs qui n'étaient pas des visiteurs quotidiens du site.

QUELQUES INDICATEURS CLES DE PERFORMANCE

Une campagne marketing a-t-elle réussi ? Il y a plusieurs manières de mesurer le succès d'une campagne. Le taux de conversion et le taux de rétention sont très souvent utilisés.

❖ TAUX DE CONVERSION

Parmi toutes les personnes qui sont entrées en contact avec la campagne, combien ont acheté le produit ou combien se sont abonnés au service de l'entreprise (cela dépend du type de business et de l'objectif fixé au départ) ? C'est le taux de conversion qui permet de répondre à cette question. Le taux de conversion est le nombre de personnes converties sur le nombre total de personnes touchées par la campagne. On peut parler de conversion en termes d'achats ou en termes d'abonnement à un service par exemple. En termes d'abonnement à un service, le taux de conversion est le pourcentage des utilisateurs qui ont vu les annonces marketing et se sont ensuite abonnés.

❖ TAUX DE RETENTION

Une fois que l'utilisateur s'est abonné, l'est-il toujours après 1 mois, 3 mois ou 1 an ? Le taux de rétention est le pourcentage de personnes qui restent abonnées après une certaine période de temps.

Calculons le taux de conversion global.

```
# Calcul du Taux de conversion global
    # Nombre d'utilisateurs uniques
n_unique_users = marketing_df['user_id'].nunique()
print("Le nombre d'utilisateurs uniques est :", n_unique_users)

    # Nombre d'utilisateurs uniques ayant souscrit au service
n_souscripteurs = marketing_df[marketing_df['converted']==True
] ['user_id'].nunique()
print('\n')
print("Le nombre d'utilisateurs uniques ayant souscrit au service est :", n_souscripteurs)

    # Taux de conversion
taux_conv_global = n_souscripteurs/n_unique_users
print('\n')
print("Le taux de conversion global est égal à :", round(taux_conv_global*100, 2), "%")
```

Le nombre d'utilisateurs uniques est : 7309

Le nombre d'utilisateurs uniques ayant souscrit au service est : 1015

Le taux de conversion global est égal à : 13.89 %

Vous vous demandez peut-être, est-ce un bon taux de conversion ? Cela dépendra fortement de votre entreprise. Il n'y a pas de nombre particulier que toutes les équipes marketing tentent d'atteindre. Au lieu de cela, lorsque vous travaillez dans une équipe marketing, il est utile d'examiner les données historiques pour déterminer si un taux de conversion correspond à ce dont vous pouvez vous attendre.

Vous calculerez le taux de rétention ou le nombre d'abonnés restants des utilisateurs qui se sont convertis à votre produit. Cela vous permettra de savoir si votre campagne marketing a converti des abonnés réellement intéressés par le produit.

Il est possible de créer une entreprise avec un taux de conversion élevé en offrant aux utilisateurs un essai gratuit, mais avoir un faible taux de rétention une fois que les utilisateurs sont facturés pour vos services. Ce n'est pas intrinsèquement une mauvaise chose, mais il est important de fournir aux parties prenantes (décideurs et actionnaires) de votre entreprise un aperçu du pourcentage d'utilisateurs qui restent abonnés.

Calculons donc le taux de rétention global !

```
# Calcul du taux de rétention global d'un mois
    # nombre de personnes qui sont restées abonnées
retained = marketing_df[marketing_df['is_retained']==True]['user_id'].nunique()

    # Taux de rétention
taux_retention_global = retained / n_souscripteurs
print(round(taux_retention_global*100, 2), "%")

66.8 %
```

Tout comme avec le taux de conversion, il n'y a pas de taux de rétention standard qui s'appliquera aux entreprises et aux industries. Regardez les taux de rétention historiques ou les taux de rétention d'entreprises dans un secteur similaire pour interpréter vos résultats.

SEGMENTATION DE CLIENTELE

Au lieu de calculer par exemple les taux de conversion et de rétention de l'ensemble, on peut les calculer par groupe d'âge par exemple. Le fait de grouper les clients selon certaines de leurs caractéristiques similaires est une technique appelée segmentation. Elle permet de faire des analyses plus approfondies et d'effectuer des campagnes marketing beaucoup plus ciblées.

❖ TAUX DE RETENTION PAR CANAL DE SOUSCRIPTION

Pour obtenir le taux de rétention pour un canal de souscription, il faudra calculer le nombre total d'utilisateurs retenus par ce canal divisé par le nombre total d'utilisateurs ayant souscrit à un abonnement via ce canal.

```
# Nombre total d'utilisateurs retenus par canal de marketing

n_retained_per_canal = marketing_df[marketing_df['is_retained']==True].groupby('subscribing_channel')['user_id'].nunique()

print(n_retained_per_canal)
```

```

subscribing_channel
Email           141
Facebook        152
House Ads       173
Instagram       158
Push             54
Name: user_id, dtype: int64

```

Pour chaque canal de souscription, calculons le nombre total d'utilisateurs du site web ayant souscrit à un abonnement.

```

# Nombre total d'utilisateurs ayant souscrit à l'abonnement via
# chaque canal

nConvertedPerCanal = marketing_df[marketing_df['converted']
                                    ==True].groupby('subscribing_channel')['user_id'].nunique()

print(nConvertedPerCanal)

subscribing_channel
Email           161
Facebook        221
House Ads       298
Instagram       232
Push             77
Name: user_id, dtype: int64

```

Nous pouvons maintenant calculer le taux de rétention pour chacun des canaux de souscription :

```

# Taux de rétention par canal de souscription

tauxRetentionParCanal = (nRetainedPerCanal / nConverted
                           _per_canal) * 100

tauxRetentionParCanal.sort_values(ascending=False)

subscribing_channel
Email           87.577640
Push            70.129870
Facebook        68.778281
Instagram       68.103448
House Ads       58.053691
Name: user_id, dtype: float64

# Visualisation des résultats

plt.figure(figsize=(10, 6))

```

```

taux_retention_par_canal.sort_values(ascending=False).plot(kind='bar')

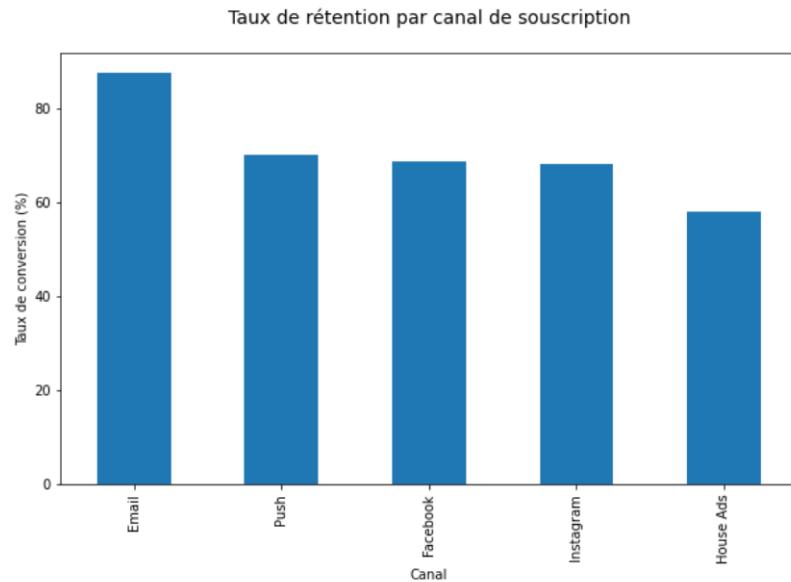
plt.title("Taux de rétention par canal de souscription\n", fontsize=14)

plt.xlabel('Canal')

plt.ylabel('Taux de conversion (%)')

plt.show()

```



Les courriers électroniques présentent le taux de rétention le plus élevé parmi les canaux de souscription au service de cette entreprise.

❖ AUTOMATISATION DU CALCUL DE TAUX DE RETENTION

Puisque nous allons devoir calculer à plusieurs reprises le taux de rétention pour différents segments, définissons une fonction qui nous permettra d'automatiser ce calcul afin de ne pas recopier du code.

```

# Fonction de calcul du taux de rétention

def retention_rate(dataframe, column_names):

    retained = dataframe[dataframe['is_retained'] == True].groupby(column_names) ['user_id'].nunique()

    converted = dataframe[dataframe['converted'] == True].groupby(column_names) ['user_id'].nunique()

```

```

    retention_rate = retained/converted

    return retention_rate

```

Nous allons vérifier que la fonction marche très bien en l'appliquant pour recalculer le taux de rétention par canal de souscription.

```

# Appel de la fonction retention_rate() pour vérification

retention_rate(marketing_df, ['subscribing_channel'])

subscribing_channel
Email      0.875776
Facebook   0.687783
House Ads   0.580537
Instagram  0.681034
Push       0.701299
Name: user_id, dtype: float64

```

La fonction a donné le même résultat que précédemment donc elle marche bien. Appliquons cette fonction pour calculer le taux de rétention selon la langue d'affichage de l'annonce.

```

# Taux de rétention par langue affichée

taux_retention_par_langue = retention_rate(marketing_df, ['languageDisplayed'])

taux_retention_par_langue

languageDisplayed
Arabic     0.750000
English    0.668467
German     0.773585
Spanish    1.000000
Name: user_id, dtype: float64

# Visualisation des résultats

plt.figure(figsize=(10, 5))

taux_retention_par_langue.sort_values(ascending=False).plot(kind='bar')

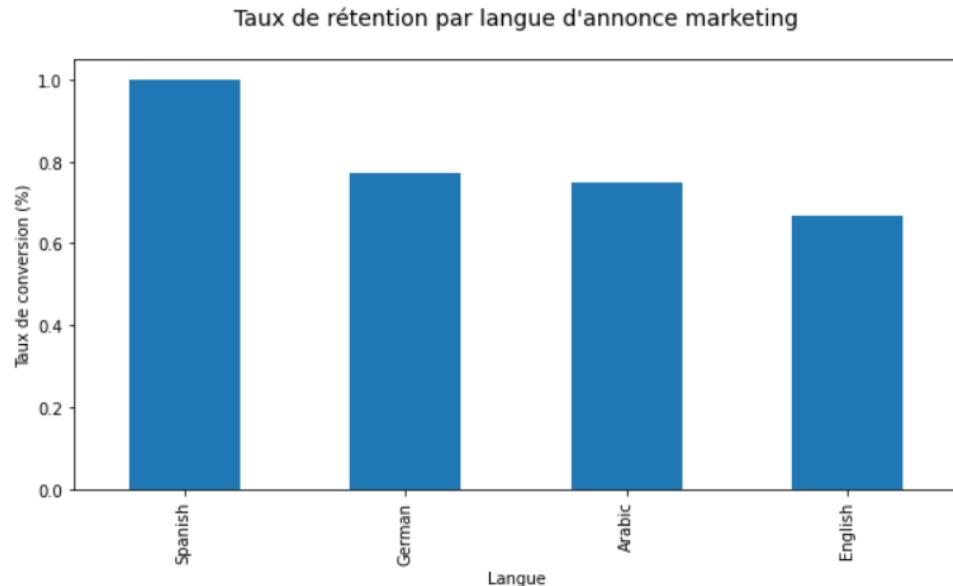
plt.title("Taux de rétention par langue d'annonce marketing\n",
          fontsize=14)

plt.xlabel('Langue')

```

```
plt.ylabel('Taux de conversion (%)')
```

```
plt.show()
```



L'Espagnol offre le meilleur taux de rétention (environ 100%).

❖ TAUX DE CONVERSION PAR CANAL D'ACQUISITION

```
# Nombre d'utilisateurs uniques ayant souscrit au service
```

```
n_souscripteurs_par_canal = marketing_df[marketing_df['converted'] == True].groupby('subscribing_channel')['user_id'].nunique()
```

```
# Nombre total de personnes atteintes par la campagne pour chaque canal
```

```
total_per_canal = marketing_df.groupby('subscribing_channel')['user_id'].nunique()
```

```
taux_conv_par_canal = (n_souscripteurs_par_canal / total_per_canal) * 100
```

```
taux_conv_par_canal.sort_values(ascending=False)
```

```
subscribing_channel
Push           100.000000
Instagram      100.000000
House Ads       100.000000
Facebook        100.000000
Email           83.854167
```

```
Name: user_id, dtype: float64
```

❖ AUTOMATISATION DU CALCUL DU TAUX DE CONVERSION

Puisque nous allons devoir calculer à plusieurs reprises le taux de conversion pour différents segments, définissons une fonction qui nous permettra d'automatiser ce calcul afin de ne pas recopier du code.

```
# Définition d'une fonction de calcul du taux de conversion

def conversion_rate(dataframe, column_names):

    # Nombre total d'utilisateurs converties

    column_conv = dataframe[dataframe['converted'] == True].groupby(column_names) ['user_id'].nunique()

    # Nombre total d'utilisateurs

    column_total = dataframe.groupby(column_names) ['user_id'].nunique()

    # Taux de conversion

    conversion_rate = column_conv/column_total

    # Remplacement des valeurs manquantes par 0

    conversion_rate = conversion_rate.fillna(0)

    return conversion_rate

# Vérification de la fonction

conversion_rate(marketing_df, ['subscribing_channel'])

subscribing_channel
Email      0.838542
Facebook   1.000000
House Ads  1.000000
Instagram  1.000000
Push       1.000000
Name: user_id, dtype: float64
```

La fonction donne le même résultat que précédemment dont elle marche bien.

❖ TAUX DE CONVERSION PAR LANGUE DE L'ANNONCE

On veut calculer le taux de conversion pour chacune des langues d'annonce afin de savoir laquelle est la plus efficace pour les campagnes.

```
taux_conv_par_langue = conversion_rate(marketing_df, ['language_displayed'])

print(taux_conv_par_langue)

languageDisplayed
Arabic      0.500000
English     0.129167
German      0.716216
Spanish     0.200000
Name: user_id, dtype: float64
```

L'Allemand offre le meilleur taux de conversion (environ 72%) et un taux de rétention de 77%.

Nous avions vu que l'espagnol a le meilleur taux de rétention (100%). Ici, on voit que pour les annonces, cette langue présente un taux de conversion de 20%.

```
# Visualisation des résultats
```

```
plt.figure(figsize=(10, 5))

taux_conv_par_langue.sort_values(ascending=False).plot(kind='bar')

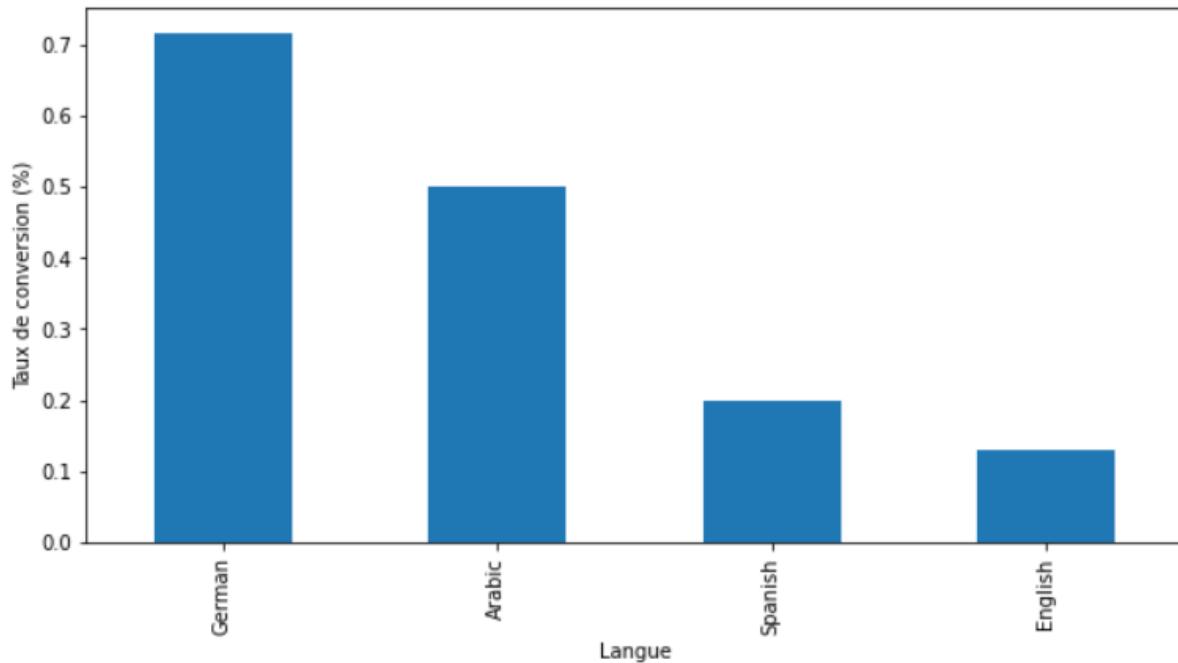
plt.title("Taux de conversion par langue d'annonce marketing\n", fontsize=14)

plt.xlabel('Langue')

plt.ylabel('Taux de conversion (%)')

plt.show()
```

Taux de conversion par langue d'annonce marketing



Les campagnes marketing en Allemand semblent être plus efficaces que celles dans les autres langues en termes de conversion des utilisateurs.

❖ TAUX DE CONVERSION JOURNALIER

```
# Calcul du taux de conversion journalier
```

```
taux_conv_journalier = conversion_rate(marketing_df, ['date_served'])

print(taux_conv_journalier)
      date_served
2018-01-01    0.099448
2018-01-02    0.098930
2018-01-03    0.103448
2018-01-04    0.108359
2018-01-05    0.125392
2018-01-06    0.113636
2018-01-07    0.141818
2018-01-08    0.115385
2018-01-09    0.125000
2018-01-10    0.118694
2018-01-11    0.080645
2018-01-12    0.076412
2018-01-13    0.084967
2018-01-14    0.085246
2018-01-15    0.113429
2018-01-16    0.255155
2018-01-17    0.219512
2018-01-18    0.091195
2018-01-19    0.059016
2018-01-20    0.067524
2018-01-21    0.087336
2018-01-22    0.123596
2018-01-23    0.122093
2018-01-24    0.115789
2018-01-25    0.125000
2018-01-26    0.090090
2018-01-27    0.065625
2018-01-28    0.062500
2018-01-29    0.059561
2018-01-30    0.066246
2018-01-31    0.052941
Name: user_id, dtype: float64
```

```
# Transformation du résultat en dataframe

taux_conv_journalier = pd.DataFrame(taux_conv_journalier.reset_index())

taux_conv_journalier.columns = ['date_subscribed', 'conversion_rate']

taux_conv_journalier.head()
```

	date_subscribed	conversion_rate
0	2018-01-01	0.099448
1	2018-01-02	0.098930
2	2018-01-03	0.103448
3	2018-01-04	0.108359
4	2018-01-05	0.125392

Jusqu'à présent, nous avons calculé des taux de rétention et de reconversion en fonction d'une seule variable. Sachez qu'il est possible de considérer plus d'une variable.

❖ TAUX DE CONVERSION JOURNALIER PAR GROUPE D'AGES

```
daily_conv_rate_age_group = conversion_rate(marketing_df, ['date_served', 'age_group'])

print(daily_conv_rate_age_group)

      date_served  age_group    conversion_rate
2018-01-01    0-18 years     0.155172
                  19-24 years     0.196721
                  24-30 years     0.105263
                  30-36 years     0.040816
                  36-45 years     0.042553
                  ...
2018-01-31    24-30 years     0.057692
                  30-36 years     0.000000
                  36-45 years     0.035088
                  45-55 years     0.023256
                  55+ years       0.026316
Name: user_id, Length: 217, dtype: float64
```

Transformons ces données en une dataframe.

```
daily_conv_rate_age_group = pd.DataFrame(daily_conv_rate_age_group.unstack(level=1))
# level = 1 car c'est la variable age_group (position 1) qu'on veut désemballer
# l'indexation dans python débute par 0

daily_conv_rate_age_group.head()
```

date_served	age_group 0-18 years	19-24 years	24-30 years	30-36 years	36-45 years	45-55 years	55+ years
2018-01-01	0.155172	0.196721	0.105263	0.040816	0.042553	0.022222	0.086957
2018-01-02	0.180328	0.164384	0.114754	0.040000	0.043478	0.043478	0.024390
2018-01-03	0.102041	0.208955	0.150943	0.042553	0.060000	0.047619	0.043478
2018-01-04	0.117647	0.200000	0.183673	0.045455	0.043478	0.061224	0.073171
2018-01-05	0.142857	0.250000	0.136364	0.090909	0.068182	0.046512	0.088889

```
# Visualisation
```

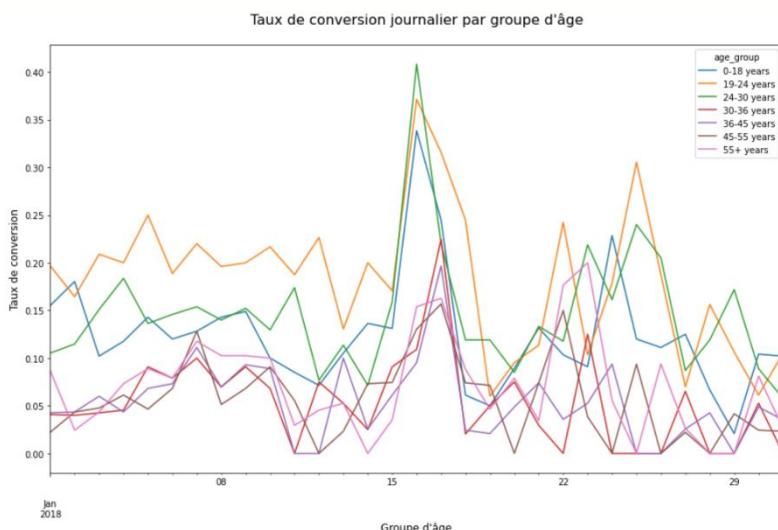
```
daily_conv_rate_age_group.plot(kind='line', figsize = (15, 9))

plt.title("Taux de conversion journalier par groupe d'âge\n",
size = 20, fontsize=16)

plt.ylabel('Taux de conversion', size = 15, fontsize=12)

plt.xlabel("Groupe d'âge", size = 15, fontsize=12)

plt.show()
```



❖ TAUX DE CONVERSION JOURNALIER PAR CANAL MARKETING

```
daily_conv_rate_canal = conversion_rate(marketing_df, ['date_served', 'marketing_channel'])

daily_conv_rate_canal = pd.DataFrame(daily_conv_rate_canal.unstack(level=1))

daily_conv_rate_canal.head()
```

marketing_channel	Email	Facebook	House Ads	Instagram	Push
date_served					
2018-01-01	1.0	0.117647	0.084656	0.106667	0.083333
2018-01-02	1.0	0.098361	0.077982	0.129032	0.055556
2018-01-03	0.0	0.080645	0.088542	0.171875	0.083333
2018-01-04	0.5	0.138462	0.089820	0.126984	0.058824
2018-01-05	1.0	0.112903	0.126582	0.159420	0.027778

```
# Visualisation
```

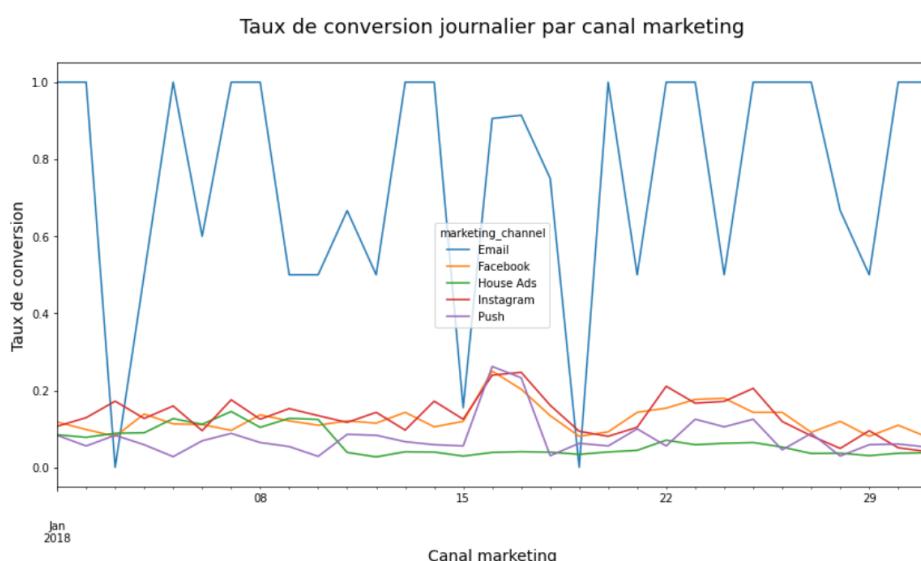
```
daily_conv_rate_canal.plot(kind='line', figsize = (14, 7))

plt.title("Taux de conversion journalier par canal marketing\n",
          size = 20, fontsize=18)

plt.ylabel('Taux de conversion', size = 16, fontsize=14)

plt.xlabel("Canal marketing", size = 16, fontsize=14)

plt.show()
```



❖ TAUX DE CONVERSION PAR JOUR DE SEMAINE ET PAR CANAL MARKETING

Les utilisateurs sont -ils plus susceptibles de convertir le week-end par rapport aux autres jours de la semaine ?

```
# Colonne Jour de la semaine où l'annonce a été diffusée

marketing_df['DoW_served'] = marketing_df['date_served'].dt.dayofweek

# Taux de conversion par jour de la semaine et par canal marketing

DoW_conversion = conversion_rate(marketing_df, ['DoW_served',
'marketing_channel'])

DoW_df = pd.DataFrame(DoW_conversion.unstack(level=1))

DoW_df
```

marketing_channel	Email	Facebook	House Ads	Instagram	Push
DoW_served					
0.0	0.162621	0.119601	0.062660	0.122517	0.064516
1.0	0.906250	0.147887	0.070312	0.151943	0.115854
2.0	0.837209	0.127036	0.075269	0.150160	0.105882
3.0	0.727273	0.133333	0.059034	0.143498	0.067797
4.0	0.666667	0.110132	0.062278	0.129870	0.055556
5.0	0.818182	0.109375	0.057566	0.088710	0.069767
6.0	0.750000	0.116071	0.065217	0.127193	0.065574

```
# Taux de conversion en fonction du jour de la semaine

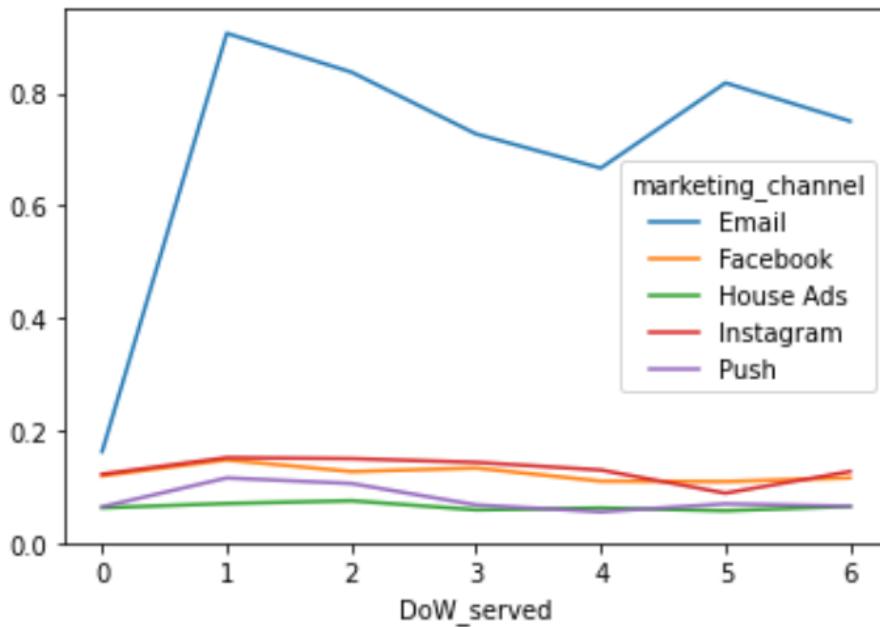
DoW_df.plot()

plt.title('Taux de conversion en fonction du jour de la semaine\n')

plt.ylim(0)
```

```
plt.show()
```

Taux de conversion en fonction du jour de la semaine



Nous avons calculé les taux de conversion et de rétention pour divers segments ou combinaison de segments. Il y a d'autres types d'analyses que vous pouvez effectuer.

❖ PREFERENCES JOURNALIERES LINGUISTIQUES DES UTILISATEURS

```
# Nombres d'utilisateurs pour chaque langue préférée à chaque date
```

```
n_lang_per_date = marketing_df.groupby(['date_served', 'language_preferred'])['user_id'].nunique()
```

```
print(n_lang_per_date)
```

date_served	language_preferred	
2018-01-01	Arabic	4
	English	342
	German	5
	Spanish	11
2018-01-02	Arabic	4
		...
2018-01-30	Spanish	19
2018-01-31	Arabic	8
	English	310
	German	5
	Spanish	17

Name: user_id, Length: 121, dtype: int64

```
# Transformation du résultat ci-dessus en une dataframe

n_lang_per_date = pd.DataFrame(n_lang_per_date.unstack(level=1))
#level=1 car c'est le deuxième index ('language_preferred')
# qu'on veut déempiler

n_lang_per_date.head()
```

language_preferred	Arabic	English	German	Spanish
date_served				
2018-01-01	4.0	342.0	5.0	11.0
2018-01-02	4.0	355.0	5.0	10.0
2018-01-03	3.0	334.0	3.0	8.0
2018-01-04	2.0	305.0	2.0	14.0
2018-01-05	1.0	303.0	2.0	14.0

```
# Visualisation du résultat
n_lang_per_date.plot(figsize=(14, 7))

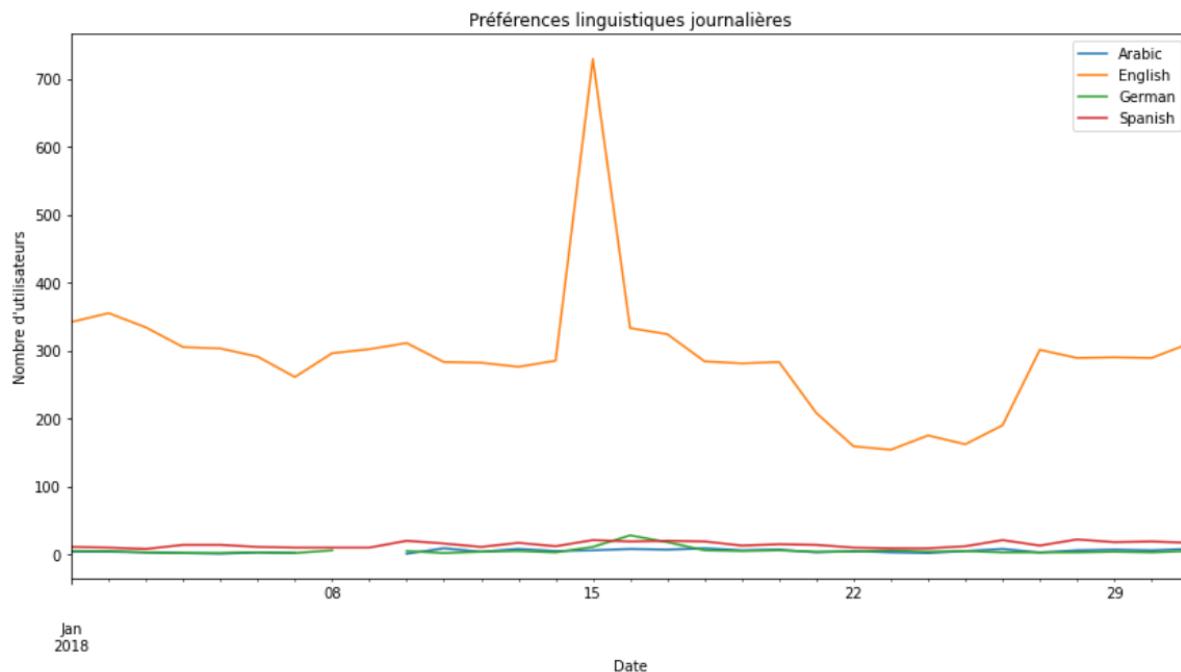
plt.title('Préférences linguistiques journalières')

plt.xlabel('Date')

plt.ylabel("Nombre d'utilisateurs")

plt.legend(loc = 'upper right', labels = n_lang_per_date.columns.values)

plt.show()
```



Comme nous l'avions vu précédemment, la langue la plus populaire est de loin l'Anglais.

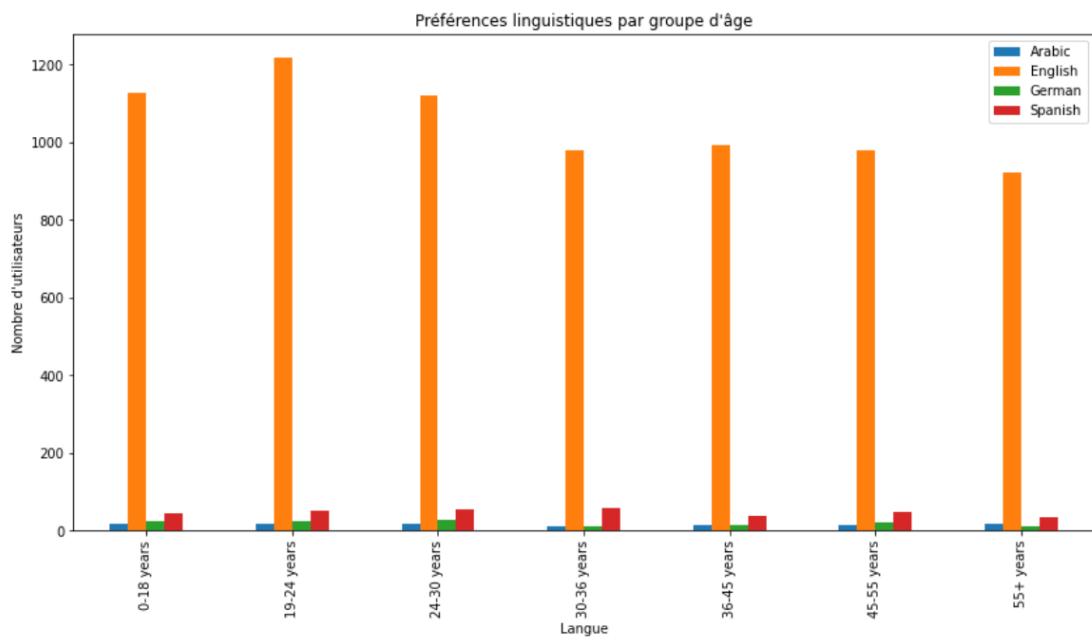
PREFERENCES LINGUISTIQUES PAR GROUPE D'AGE

```
# Nombre d'utilisateurs par groupe d'âges et langues préférées

n_lang_per_age = marketing_df.groupby(['age_group', 'language_preferred'])['user_id'].nunique()

n_lang_per_age = pd.DataFrame(n_lang_per_age.unstack(level=1))

n_lang_per_age.head()
language_preferred  Arabic  English  German  Spanish
age_group
0-18 years          16    1126     24      43
19-24 years         18    1218     23      51
24-30 years         16    1121     29      54
30-36 years         12     978     12      57
36-45 years         13    993      14      39
```



L'Anglais demeure la langue préférée quel que soit le groupe d'âges.

CANAUX MARKETING SELON LES GROUPES D'AGES

Supposons que les décideurs de l'entreprise veulent savoir si les canaux marketing atteignent tous les utilisateurs de manière égale ou s'il y a une différence. Nous allons donc créer un graphique indiquant le nombre de personnes atteintes par chaque canal marketing par âge.

```
# Nombre d'utilisateurs par groupe d'âges et canal d'acquisition
```

```
canal_age = marketing_df.groupby(['age_group', 'marketing_channel'])['user_id'].nunique()
```

```
canal_age = pd.DataFrame(canal_age.unstack(level=1))
```

```
canal_age.head()
```

	marketing_channel	Email	Facebook	House Ads	Instagram	Push
age_group						
0-18 years	91	256	585	292	184	
19-24 years	107	331	643	301	144	
24-30 years	116	289	587	268	180	
30-36 years	60	238	600	252	83	
36-45 years	63	217	505	268	192	

```

# Visualisation

canal_age.plot(kind='bar', figsize = (14, 7))

plt.title("Canal d'acquisition par groupe d'âge")

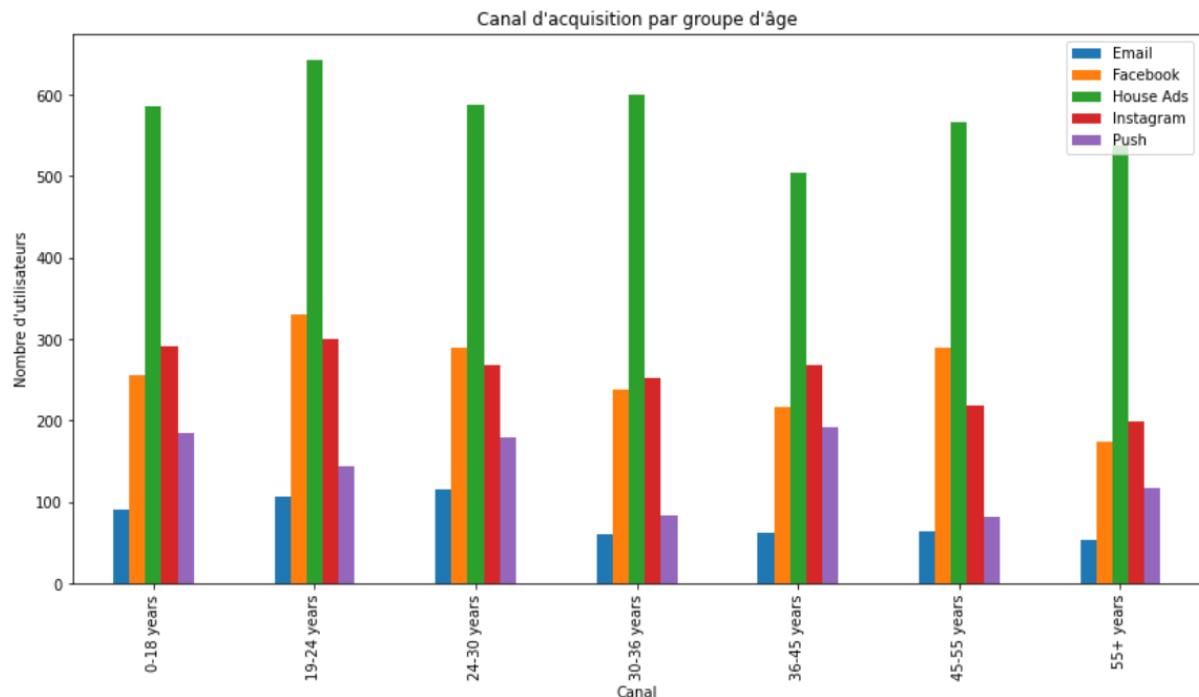
plt.xlabel("Canal")

plt.ylabel("Nombre d'utilisateurs")

plt.legend(loc = 'upper right', labels = canal_age.columns.values)

plt.show()

```



CONCLUSION

Le Marketing est l'un des départements stratégiques les plus importants pour une entreprise. Pourvoir analyser les données clients afin d'en tirer des informations utiles est une compétence hautement importante. Pandas est une librairie très puissante pour faire l'analyse des données avec Python.

A travers ce projet, nous avons utilisé quelques fonctionnalités de Pandas afin d'effectuer une analyse descriptive de la campagne marketing d'une entreprise de services en ligne.

PROJET 1 : PREDICTION DU PRIX D'UNE VOITURE EN FONCTION DE SES CARACTERISTIQUES

INTRODUCTION

Une voiture doit être vendue à un prix raisonnable aussi bien pour les potentiels acheteurs mais aussi pour le vendeur. Comment déterminer alors le meilleur ou le juste prix d'une voiture ? Quelles sont les caractéristiques d'une voiture qui affectent son prix.

Ce projet nous permettra de découvrir un type d'apprentissage automatique supervisé : La **régression**. Dans les tâches de régression, la variable cible est une variable continue. Par exemple, si on veut prédire le prix d'une maison, le bénéfice d'une entreprise, le PIB (Produit Intérieur Brut) d'un pays, le nombre de vélos loués à une heure donnée, ..., on utilisera des algorithmes de régression étant donné que ces variables sont quantitatives.

L'objectif ici est de construire un modèle de régression linéaire qui prédit le prix de vente d'une voiture en fonction de ses caractéristiques.

LIBRAIRIES

```
#Importation des librairies nécessaires à la réalisation du projet

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from scipy import stats

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error
```

DONNEES

Pour répondre à notre problématique, nous disposons d'un fichier csv de données de plusieurs voitures avec leurs caractéristiques et leurs prix.

```

#Chemin du fichier

filename = "https://raw.githubusercontent.com/JosueAfouda/Prix
-Voiture/master/imports-85.data"

#Lecture du fichier

df = pd.read_csv(filename, header = None)

#Cinq premières lignes

df.head()

```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823	ohcv	six	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337	ohc	four	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824	ohc	five	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450

Les noms des colonnes ne sont pas définis dans ce jeu de données. Heureusement que nous avons retrouvé ces noms de colonnes dans un autre [fichier](#) dans le même dépôt que les données.

```
#Noms des colonnes
```

```

headers = ["symboling", "normalized-losses", "make", "fuel-
type", "aspiration", "num-of-doors", "body-style",
           "drive-wheels", "engine-location", "wheel-
base", "length", "width", "height", "curb-weight", "engine-type",
           "num-of-cylinders", "engine-size", "fuel-
system", "bore", "stroke", "compression-ratio", "horsepower",
           "peak-rpm", "city-mpg", "highway-mpg", "price"]

```

```
#Relecture des données
```

```

df = pd.read_csv(filename, header = None, names = headers)

df.head()

```

Les valeurs manquantes sont ici représentées par des "?". Ceci est courant dans les données qu'on retrouve sur Internet.

```
# Structure de la dataframe

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   symboling        205 non-null    int64  
 1   normalized-losses 205 non-null    object  
 2   make             205 non-null    object  
 3   fuel-type        205 non-null    object  
 4   aspiration       205 non-null    object  
 5   num-of-doors     205 non-null    object  
 6   body-style       205 non-null    object  
 7   drive-wheels     205 non-null    object  
 8   engine-location   205 non-null    object  
 9   wheel-base       205 non-null    float64 
 10  length           205 non-null    float64 
 11  width            205 non-null    float64 
 12  height           205 non-null    float64 
 13  curb-weight      205 non-null    int64  
 14  engine-type      205 non-null    object  
 15  num-of-cylinders 205 non-null    object  
 16  engine-size      205 non-null    int64  
 17  fuel-system      205 non-null    object  
 18  bore              205 non-null    object  
 19  stroke            205 non-null    object  
 20  compression-ratio 205 non-null    float64 
 21  horsepower        205 non-null    object  
 22  peak-rpm          205 non-null    object  
 23  city-mpg          205 non-null    int64  
 24  highway-mpg        205 non-null    int64  
 25  price             205 non-null    object  
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
```

Il est très important de comprendre le sens de chaque variable dans un jeu de données. Vous trouverez [ici](#)⁸ la description des variables.

NETTOYAGE DES DONNEES

❖ GESTION DES VALEURS MANQUANTES

Il y a des valeurs manquantes représentées par des *"?* dans les données. Comment gérer ces données manquantes ?

```
#Remplacement de "?" par NaN

df.replace("?", np.nan, inplace = True)
```

```
#Evaluation du nombre de valeurs manquantes par colonne
```

```
df.isnull().sum()
```

```
symboling          0
normalized-losses 41
make              0
fuel-type          0
aspiration         0
num-of-doors       2
body-style          0
drive-wheels        0
engine-location     0
wheel-base          0
length              0
width              0
height              0
curb-weight         0
engine-type         0
num-of-cylinders    0
engine-size          0
fuel-system          0
bore                 4
stroke               4
compression-ratio    0
horsepower           2
peak-rpm              2
city-mpg              0
highway-mpg            0
price                 4
dtype: int64
```

D'après la description du jeu de données, les variables ***normalized-losses***, ***bore***, ***stroke***, ***horsepower*** et ***peak-rpm*** sont des variables continues. Nous remplacerons donc les valeurs manquantes de chacune de ces colonnes par la moyenne de la colonne. La variable ***num-of-doors*** est une variable catégorielle donc nous allons remplacer ses deux valeurs manquantes par le mode de la colonne.

La variable ***price*** est notre variable cible donc nous allons tout simplement supprimer les valeurs manquantes de cette colonne !

```

#Imputation par la moyenne des colonnes normalized-
losses, bore, stroke, horsepower et peak-rpm

def imput_avg(x):

    avg = x.astype('float').mean(axis = 0)

    return x.replace(np.nan, avg, inplace = True)

for colonne in ['normalized-
losses', 'bore', 'stroke', 'horsepower', 'peak-rpm']:

    imput_avg(df[colonne])

#Imputation par le mode de la variable num-of-doors

mode = df['num-of-doors'].value_counts().idxmax()

df['num-of-doors'].replace(np.nan, mode, inplace = True)

#Suppression des lignes de df où il y a des données manquantes
dans la variable price

df.dropna(subset = ['price'], axis = 0, inplace = True)

```

Après avoir supprimé des lignes dans une dataframe, il faut toujours réinitialiser les indices.

```

#Réinitialisation des indices

df.reset_index(drop = True, inplace = True)

```

Vérifions que les diverses imputations effectuées aient bien fonctionnées.

```

#Nombre de valeurs manquantes dans df

df.isnull().sum()

```

```

symboling          0
normalized-losses 0
make              0
fuel-type         0
aspiration        0
num-of-doors      0
body-style        0
drive-wheels      0
engine-location   0
wheel-base        0
length            0
width             0
height            0
curb-weight       0
engine-type       0
num-of-cylinders 0
engine-size       0
fuel-system       0
bore              0
stroke            0
compression-ratio 0
horsepower        0
peak-rpm          0
city-mpg          0
highway-mpg       0
price             0
dtype: int64

```

Ce résultat confirme qu'il n'y a plus de données manquantes dans notre dataframe.

❖ FORMAT CORRECT DES DONNEES

```
# Format des données
```

```
df.dtypes
```

```

symboling          int64
normalized-losses object
make              object
fuel-type         object
aspiration        object
num-of-doors      object
body-style        object
drive-wheels      object
engine-location   object
wheel-base        float64
length            float64
width             float64
height            float64
curb-weight       int64
engine-type       object
num-of-cylinders object
engine-size       int64
fuel-system       object
bore              object
stroke            object
compression-ratio float64
horsepower        object
peak-rpm          object
city-mpg          int64
highway-mpg       int64
price             object
dtype: object

```

On remarque que certaines variables ne sont pas stockées dans le bon format. Par exemple, il y a des variables de type numérique comme *bore* et *stroke* qui sont stockées dans le format 'object' (format des chaînes de caractères). En se basant sur la [description](#)⁸ des données, nous allons convertir les types de données dans le format approprié pour chaque colonne.

```
#Conversion
```

```
df[['bore', 'stroke', 'peak-
rpm', 'price']] = df[['bore', 'stroke', 'peak-
rpm', 'price']].astype("float")

df[['normalized-losses', 'horsepower']] = df[['normalized-
losses', 'horsepower']].astype("int")
```

```
#Vérification de la conversion
```

```
df.dtypes
```

```
symboling          int64
normalized-losses  int64
make               object
fuel-type          object
aspiration         object
num-of-doors       object
body-style         object
drive-wheels       object
engine-location    object
wheel-base         float64
length              float64
width               float64
height              float64
curb-weight         int64
engine-type         object
num-of-cylinders   object
engine-size         int64
fuel-system         object
bore                float64
stroke              float64
compression-ratio   float64
horsepower          int64
peak-rpm             float64
city-mpg             int64
highway-mpg          int64
price                float64
dtype: object
```

❖ SAUVEGARDE DES DONNEES NETTOYEEES

Avant de passer à l'analyse exploratoire des données, nous sauvegardons le fichier de données issu de cette étape de nettoyage.

```
df.to_csv('df_clean.csv')
```

ANALYSE EXPLORATOIRE DES DONNEES

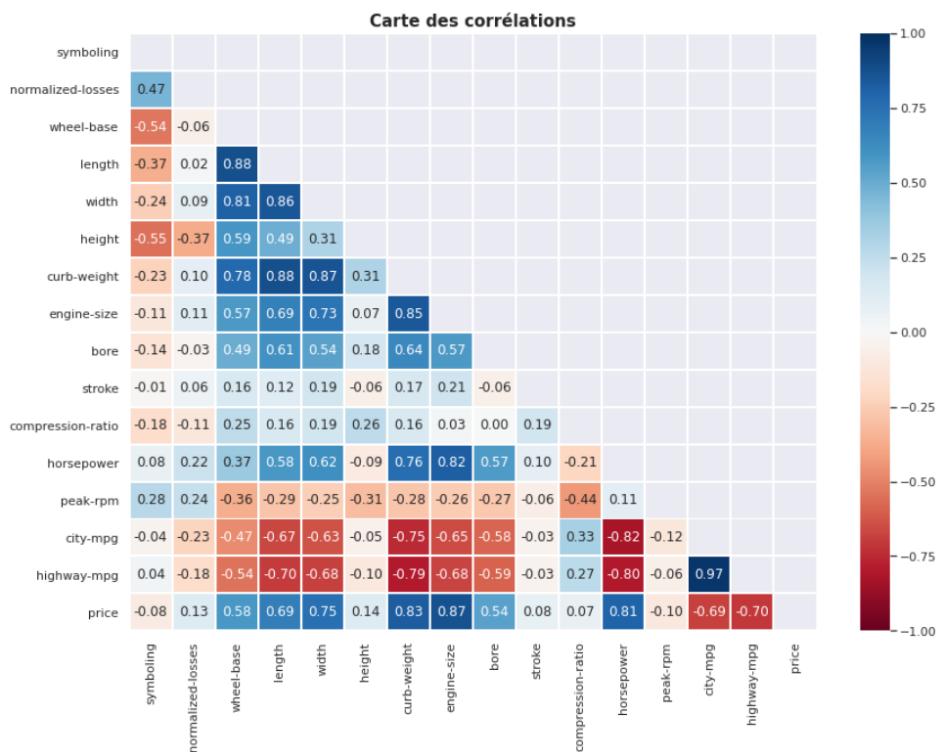
Le but de cette partie est de mieux comprendre les données en tirant des informations à partir de ces données. Principalement, nous voulons déterminer les variables qui influencent le prix d'une voiture.

❖ CORRELATION

Nous voulons mesurer la force des relations linéaires entre variables. Pour ce faire, nous calculerons le coefficient de corrélation de Pearson et nous afficherons les résultats.

```
# Carte des corrélations
```

```
sns.set(rc={'figure.figsize':(14,10)})  
  
mask = np.triu(np.ones_like(df.corr(), dtype=bool))  
  
sns.heatmap(df.corr(), mask=mask, center=0, cmap='RdBu', linewidths=1, annot=True, fmt=".2f", vmin=-1, vmax=1)  
  
plt.title('Carte des corrélations', fontsize=15, fontweight="bold")  
  
plt.show()
```



En examinant les coefficients de corrélation, nous détectons les potentiels prédicteurs du prix d'une voiture. Pour chaque variable numérique potentiellement prédicteur du prix, nous ferons une régression linéaire simple entre elle et la variable *price*, afficher le coefficient de corrélation de Pearson et ainsi que sa p-valeur.

Le but de la régression linéaire est d'ajuster une droite aux données. Une droite, en 02 dimensions, est représentée par l'équation : $y = ax + b$. y est la variable dépendante (encore appelée variable d'intérêt ou variable cible) et x est la variable indépendante (encore appelée prédicteur). a et b sont les paramètres du modèle (a est la pente de la droite d'ajustement et b est l'ordonnée à l'origine) que nous voulons apprendre.

Commençons d'abord par définir une fonction qui automatise cette tâche afin de ne pas copier-coller du code.

```
#Définition d'une fonction slrViz_corr

def slrViz_corr(var):

    sns.regplot(x = var, y = df['price'])

    plt.ylim(0,)

    Pearson_coef, P_value = stats.pearsonr(var, df['price'])

    return print('Le coefficient de corrélation de Pearson est',
    ', Pearson_coef, 'avec une P-valeur de', P_value)
```

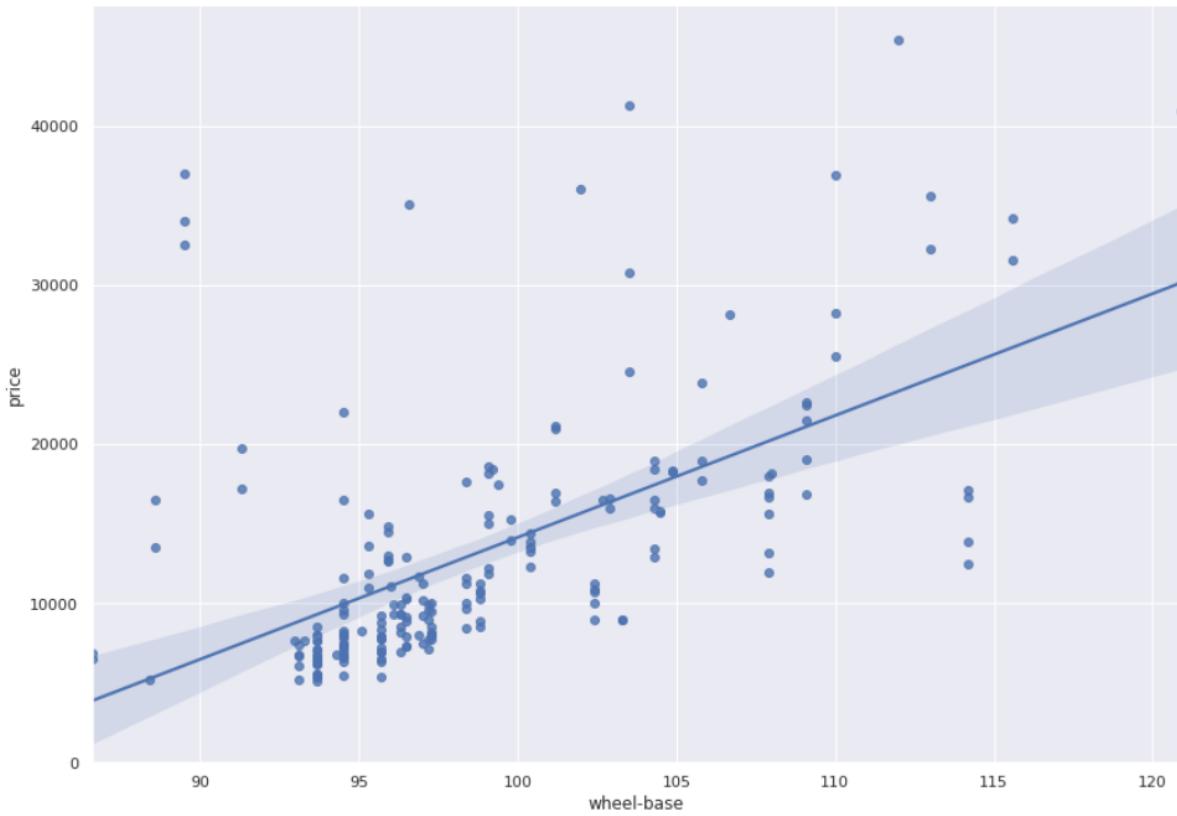
wheel-base VS price

La variable *wheel-base* représente la distance entre les essieux⁹ avant et arrière d'un véhicule.

```
#Régression linéaire simple de 'price' en fonction de 'wheel-
base'

slrViz_corr(df['wheel-base'])
```

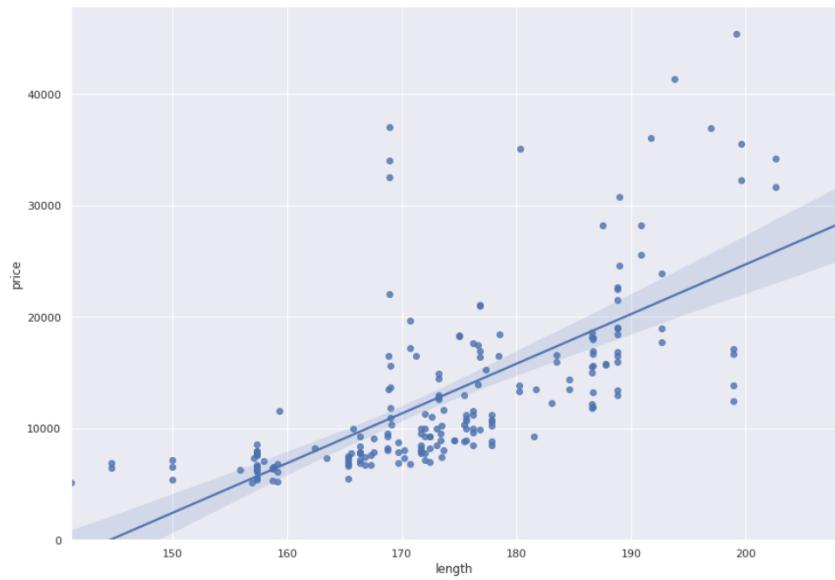
Le coefficient de corrélation de Pearson est 0.584641822265508 avec une P-valeur de 8.076488270733218e-20



La corrélation entre les variables **wheel-base** et **price** est statistiquement significative (p-value inférieure à 0,001) bien que la relation linéaire ne soit pas très forte.

length VS price

Le coefficient de corrélation de Pearson est 0.6906283804483639 avec une P-valeur de 8.016477466159328e-30

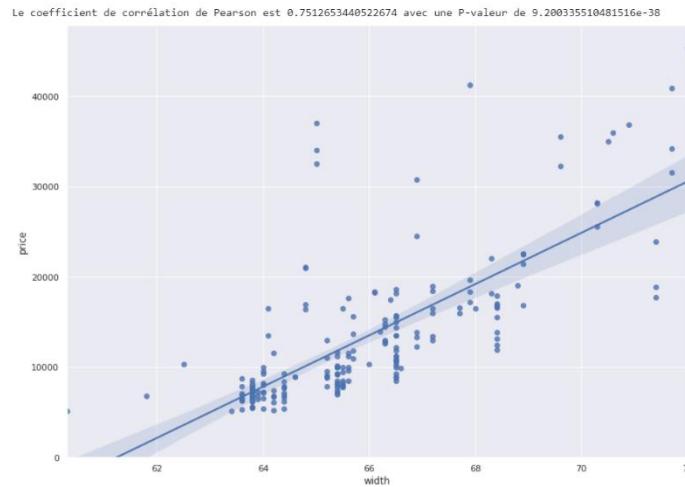


Il y a une forte corrélation positive et statistiquement significative (p-value inférieure à 0,001) entre les variables **length** et **price**.

width VS price

```
#Régression linéaire simple de 'price' en fonction de 'length'
```

```
slrViz_corr(df['width'])
```



Il y a une très forte corrélation positive et statistiquement significative (p-value inférieure à 0,001) entre les variables **width** et **price**.

curb-weight VS price

La variable **curb-weight** représente le poids à vide d'une voiture c'est-à-dire son poids sans occupants ni bagages.

```
#Régression linéaire simple de 'price' en fonction de 'curb-weight'
```

```
slrViz_corr(df['curb-weight'])
```



Il y a une corrélation positive extrêmement forte et statistiquement significative (p-value inférieure à 0,001) entre les variables ***curb-weight*** et ***price***.

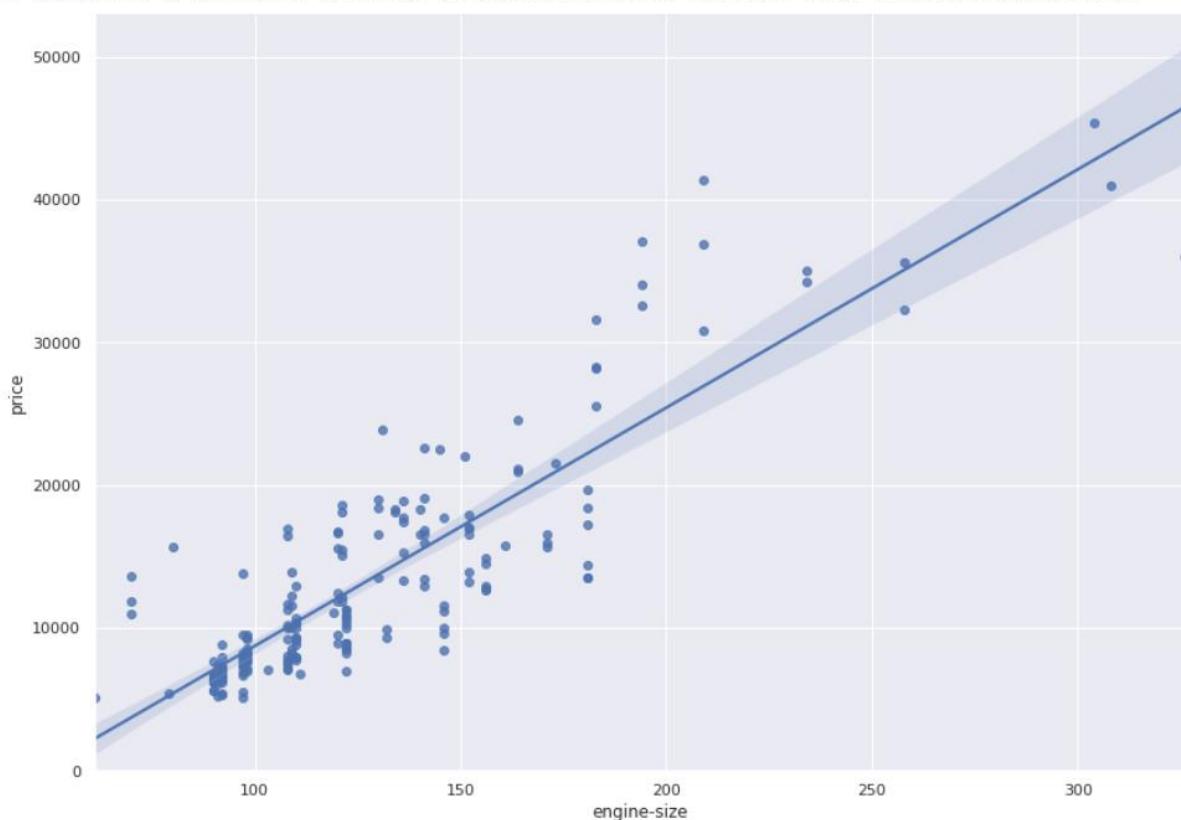
engine-size VS price

La variable ***engine-size*** représente la taille du moteur.

```
#Régression linéaire simple de 'price' en fonction de 'engine-size'
```

```
slrViz_corr(df['engine-size'])
```

Le coefficient de corrélation de Pearson est 0.8723351674455185 avec une P-valeur de 9.265491622198389e-64



Il y a une corrélation positive extrêmement forte et statistiquement significative (p-value inférieure à 0,001) entre les variables ***engine-size*** et ***price***.

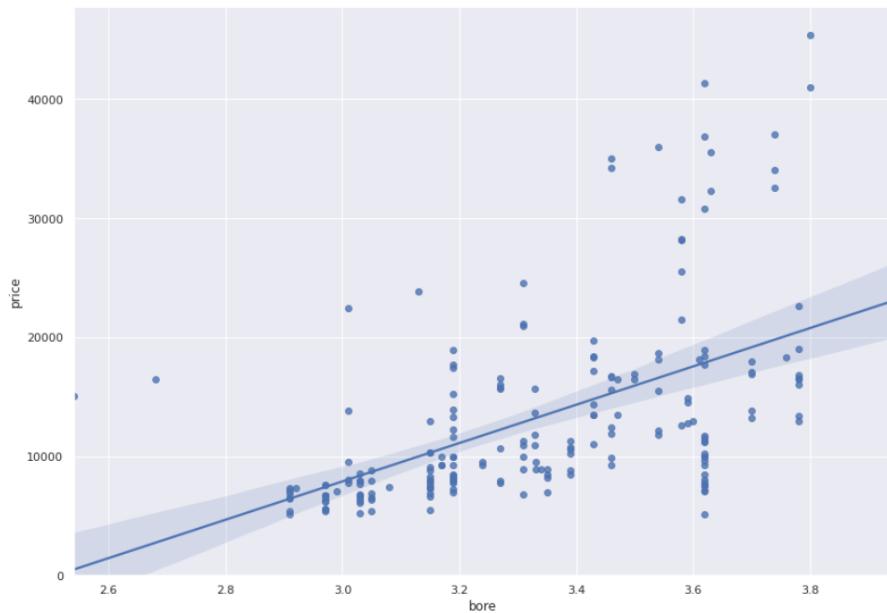
bore VS price

La variable ***bore*** représente le diamètre du cylindre de la jante.

```
#Régression linéaire simple de 'price' en fonction de 'bore'
```

```
slrViz_corr(df['bore'])
```

Le coefficient de corrélation de Pearson est 0.5431553832626602 avec une P-valeur de 8.049189483935489e-17



La corrélation entre les variables **bore** et **price** est statistiquement significative (p-value inférieure à 0,001) bien que la relation linéaire ne soit pas très forte.

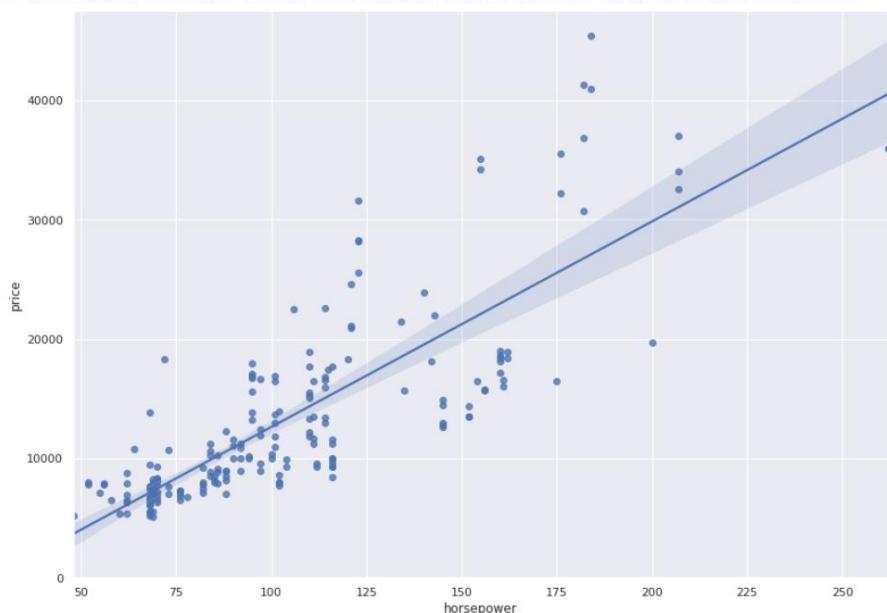
horsepower VS price

horsepower représente la puissance du moteur (une unité de puissance est égale à 745,7 watts).

```
#Régression linéaire simple de 'price' en fonction de 'horsepower'
```

```
slrViz_corr(df['horsepower'])
```

Le coefficient de corrélation de Pearson est 0.8096068016571054 avec une P-valeur de 6.273536270650504e-48



Il y a une très forte corrélation positive et statistiquement significative (p-value inférieure à 0,001) entre la puissance du moteur et le prix de la voiture. Autrement dit, plus le moteur est puissant, plus le prix de la voiture est élevé.

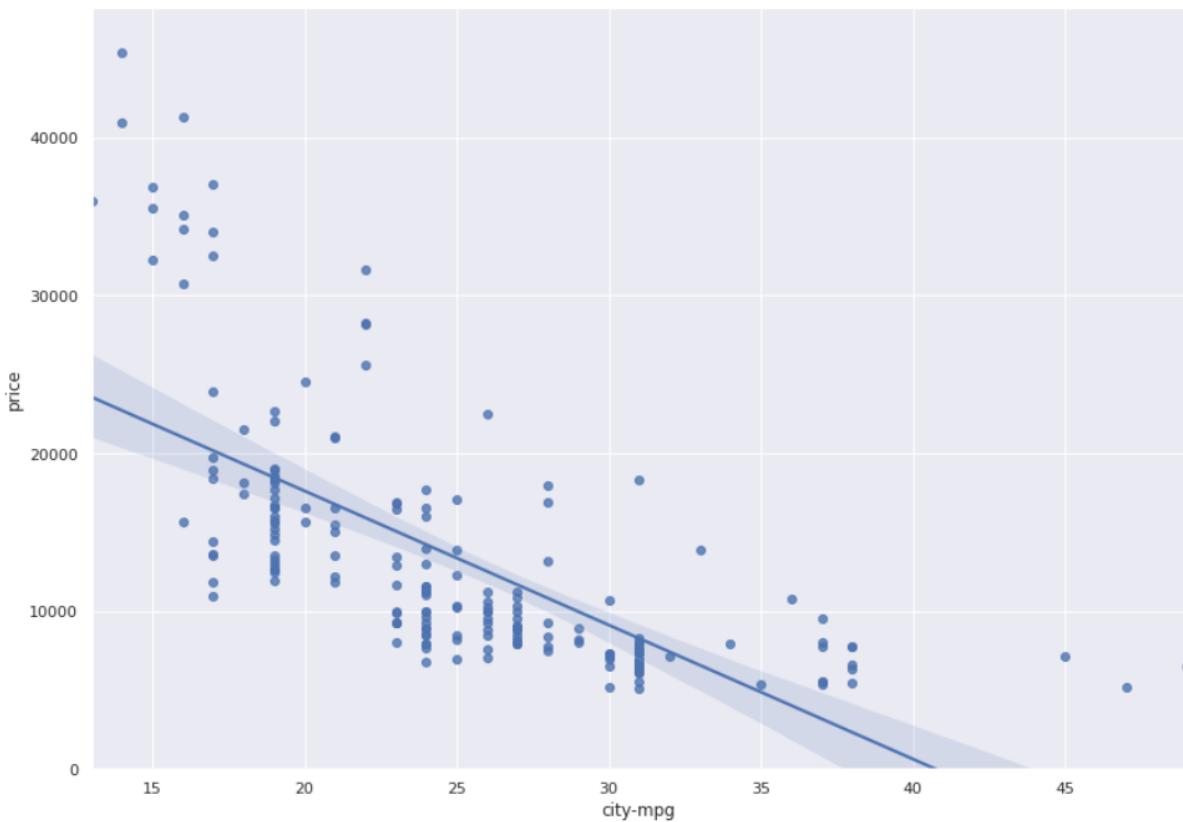
city-mpg VS price

La variable ***city-mpg*** représente la consommation en carburant de la voiture en agglomération. Elle est donnée en miles par gallon. Pour trouver la consommation en Litres/100 Km, il faut diviser 235 par la consommation en miles/gallon).

```
#Régression linéaire simple de 'price' en fonction de 'city-mpg'
```

```
slrviz_corr(df['city-mpg'])
```

Le coefficient de corrélation de Pearson est -0.6865710067844678 avec une P-valeur de 2.321132065567641e-29



Il y a une forte corrélation négative et statistiquement significative (p-value inférieure à 0,001) entre les variables ***city-mpg*** et ***price***. Autrement dit, plus la consommation en carburant (en ville) de la voiture est importante, plus sa valeur (prix) baisse.

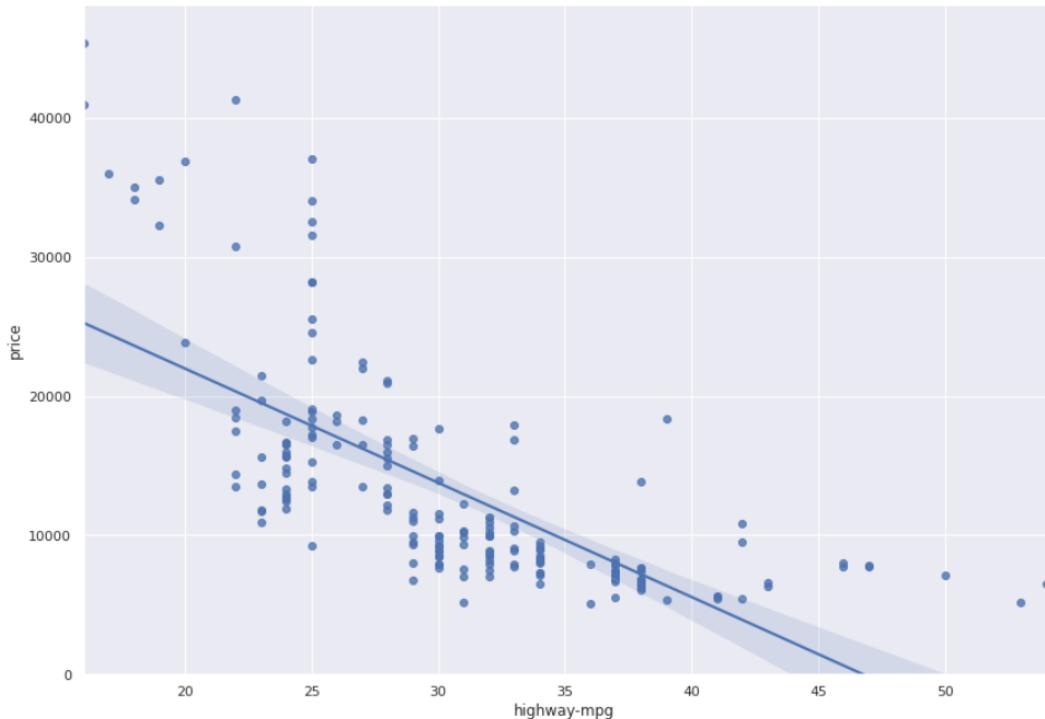
highway-mpg VS price

La variable **highway-mpg** représente la consommation en carburant de la voiture sur l'autoroute.

```
#Régression linéaire simple de 'price' en fonction de 'highway-mpg'
```

```
slrViz_corr(df['highway-mpg'])
```

Le coefficient de corrélation de Pearson est -0.704692265058953 avec une P-valeur de 1.7495471144476358e-31



Il y a une très forte corrélation négative et statistiquement significative (p-value inférieure à 0,001) entre les variables **highway-mpg** et **price**. Autrement dit, plus la consommation en carburant (sur l'autoroute) de la voiture est importante, plus son prix baisse.

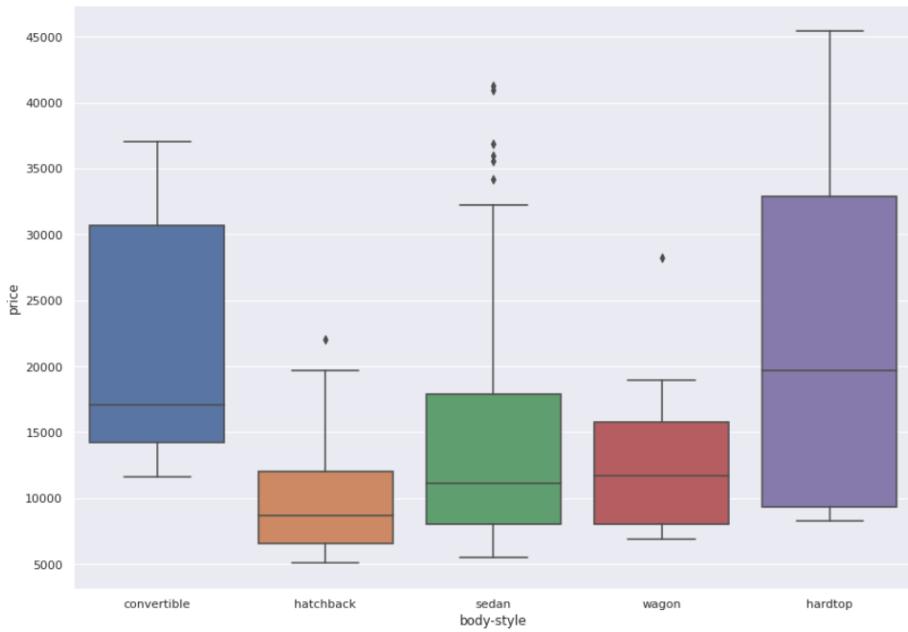
A travers les analyses ci-dessus, nous avons déterminé les variables numériques qui influencent le prix d'une voiture. Qu'en est-il de la relation entre chacune des valeurs catégorielles et la variable d'intérêt ? Nous pouvons visualiser ces relations par des boîtes à moustaches.

❖ Variables catégorielles VS variable cible

body-style et price

```
sns.boxplot(x = 'body-style', y = 'price', data = df)
```

```
plt.show()
```

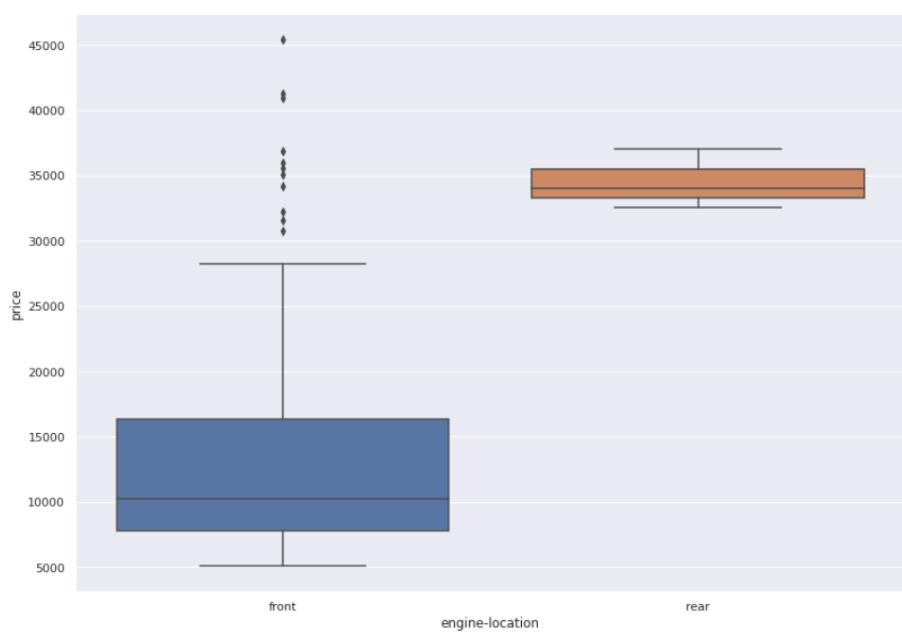


Nous voyons que les distributions de prix entre les différentes catégories de ***body-style*** ont un chevauchement important, et donc cette variable ne serait pas un bon prédicteur de prix.

engine-location VS price

La variable ***engine-location*** représente l'emplacement (avant ou arrière) du moteur.

```
sns.boxplot(x = 'engine-location', y = 'price', data = df)
plt.show()
```

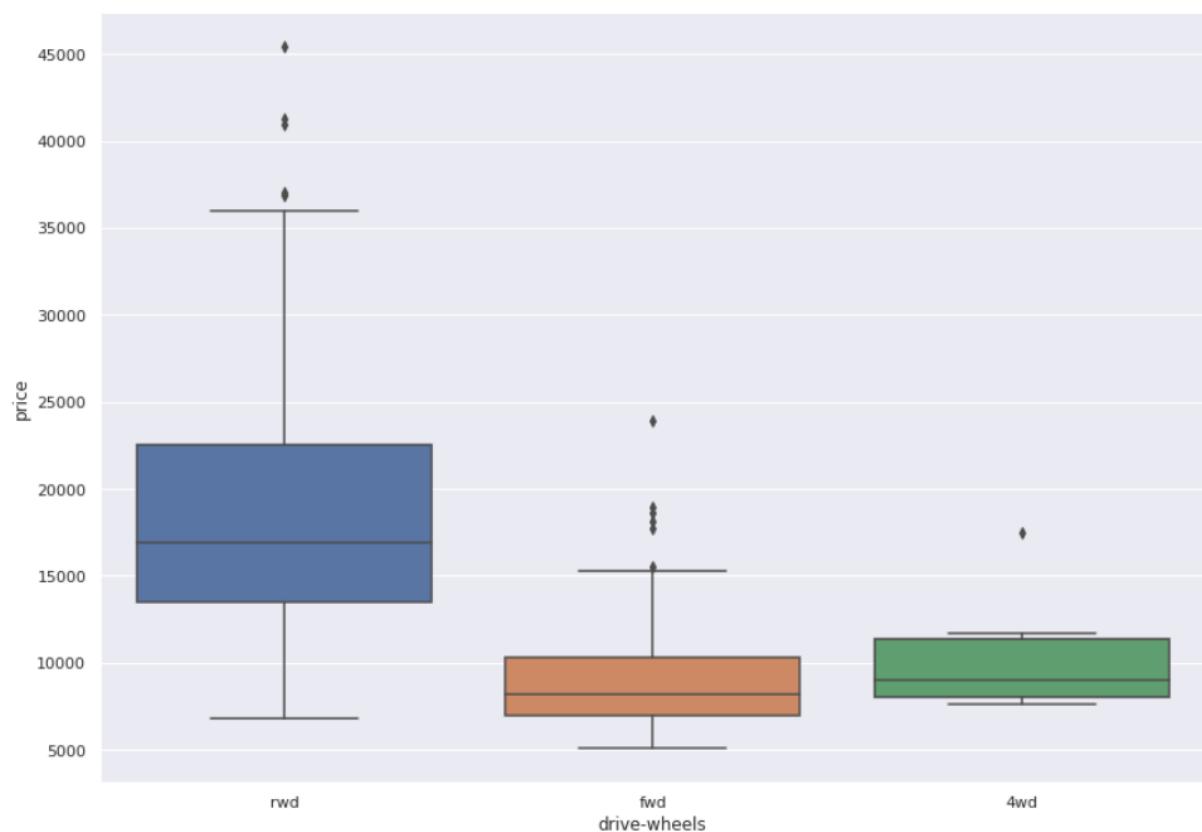


Les voitures avec des moteurs à l'arrière sont plus chères que les voitures avec des moteurs à l'avant. Par ailleurs, il y a seulement trois voitures qui ont un moteur à l'arrière. Ce problème de déséquilibre de classe disqualifie la variable engine-location comme variable prédicteur du prix.

```
df['engine-location'].value_counts()
```

drive-wheels VS price

La variable ***drive-wheels*** définit les roues motrices de la voiture.



On voit ici que la répartition des prix entre les différentes catégories de roues motrices est différente. La variable ***drive-wheels*** pourrait potentiellement être une variable prédicteur de prix d'une voiture.

Cette analyse exploratoire des données nous a permis d'avoir une meilleure idée des variables qu'il est important de prendre en compte lors de la construction du modèle de prédiction du prix d'une voiture. Il s'agit des variables suivantes :

Variables numériques continues : ***'length'*, *'width'*, *'curb-weight'*, *'engine-size'*, *'horsepower'*, *'city-mpg'*, *'highway-mpg'*, *'wheel-base'* et *'bore'*.**

Variable catégorielle : 'drive-wheels'.

MODELISATION DU PRIX D'UNE VOITURE

❖ CONSTRUCTION DU MODELE

Nous construirons un modèle de régression linéaire multiple pour prédire le prix de vente d'une voiture en fonction de certaines de ses caractéristiques. Ces caractéristiques ont été déterminé lors de l'Analyse exploratoire des données.

```
#Variables indépendantes
```

```
X = df[['wheel-base', 'length', 'width', 'curb-
weight', 'engine-size', 'bore', 'horsepower', 'city-
mpg', 'highway-mpg']]
```

```
#Variable dépendante
```

```
y = df['price']
```

```
#Création d'un modèle de régression linéaire
```

```
lm = LinearRegression()
```

```
#Entraînement du modèle
```

```
lm.fit(X, y)
```

```
LinearRegression(copy_X=True,          fit_intercept=True,      n_jobs=None,
normalize=False)
```

```
#Coefficients
```

```
lm.coef_
```

```
array([ 111.78344803,   -72.26084531,    634.95267368,     3.11828375,
       79.05419995,  -1026.27963187,    59.42633928,   -170.06836451,
      184.10851533])
```

```
#Ordonnée à l'origine
```

```
lm.intercept_
```

```
-49178.735733150395
```

❖ EVALUATION DU MODELE

```
#Coefficient de détermination
```

```
lm.score(X, y)
```

```
0.819067025194617
```

Avec ce modèle, nous avons un coefficient de détermination (R^2) égal à environ 0,82. Nous pouvons donc dire que le modèle capture bien les relations linaires dans les données. En effet, plus R^2 est proche de 1, plus on a un bon modèle (R^2 varie entre 0 et 1).

Pour évaluer la qualité d'ajustement du modèle aux valeurs réelles, nous allons examiner la distribution des valeurs ajustées qui résultent du modèle et la comparer à la distribution des valeurs réelles.

Analysons la distribution des prix calculés par le modèle et celle des prix réels.

```
#Prix prédicts à partir du modèle
```

```
Y_hat = lm.predict(X)
```

```
#Distributions des valeurs ajustées et des valeurs réelles
```

```
plt.figure(figsize=(10, 8))
```

```
ax1 = sns.distplot(y, hist=False, color="b", label="Prix réels")
```

```
sns.distplot(Y_hat, hist=False, color="r", label="Prix calculé")
```

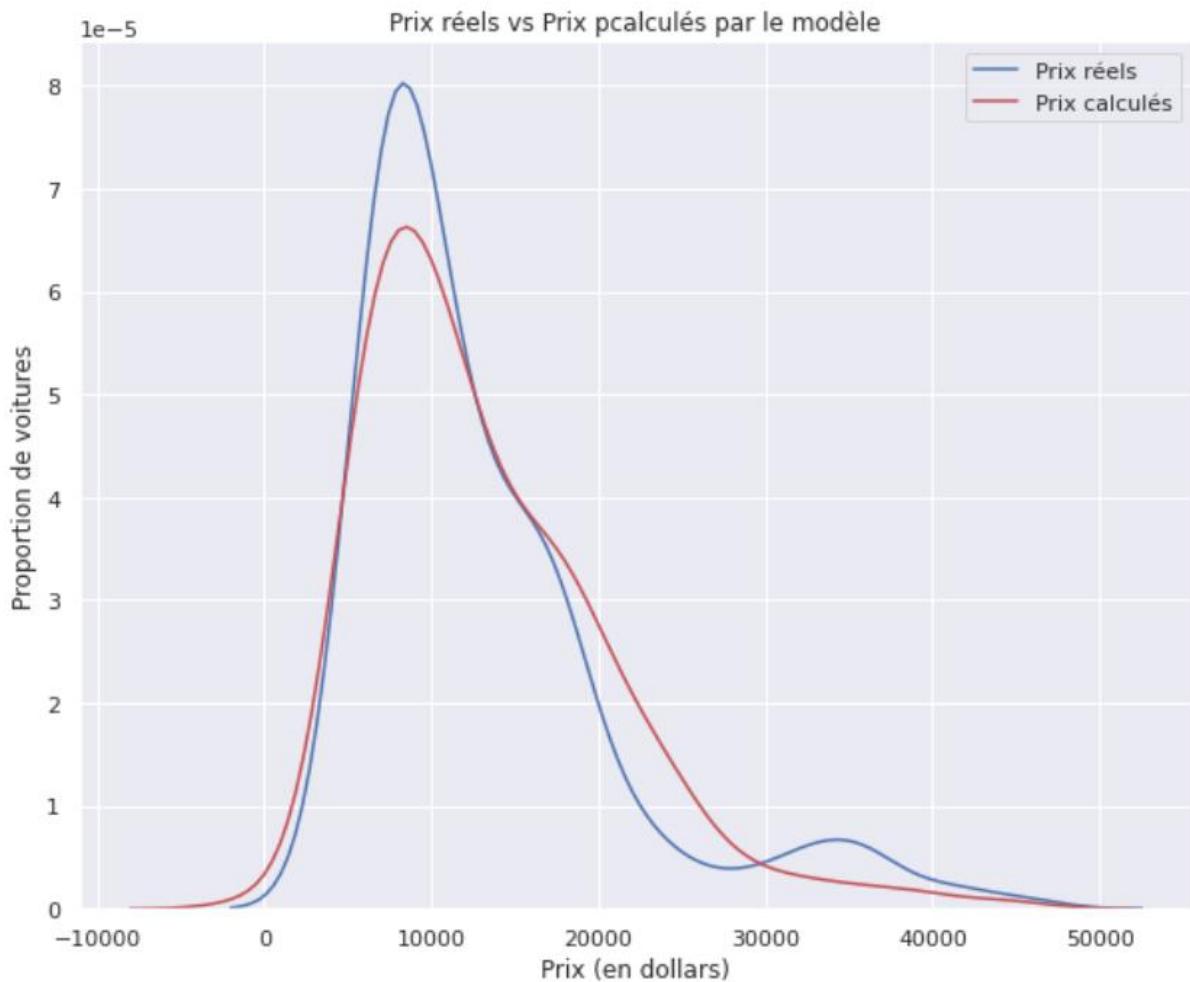
```
plt.title('Prix réels vs Prix calculés par le modèle')
```

```
plt.xlabel('Prix (en dollars)')
```

```
plt.ylabel('Proportion de voitures')
```

```
plt.show()
```

```
plt.close()
```



La métrique d'évaluation par défaut de la performance d'un modèle de régression linéaire est le coefficient de détermination. Le coefficient de détermination est égal à 0,82 c'est-à-dire que 82% de la variation de prix est expliquée par le modèle. Par ailleurs, on remarque que les deux courbes sont pratiquement superposées ce qui veut dire que les prix calculés par le modèle sont proches des prix réels. Globalement, nous pouvons donc dire que le modèle est de bonne qualité. Néanmoins, selon le graphique ci-dessus il y a une zone que le modèle ne couvre pas ce qui indique que nous pouvons essayer d'améliorer ce modèle.

CONCLUSION

Ce projet est une initiation à la Régression qui est une tâche de Machine Learning supervisé. Dans le but de déterminer à quel prix 'juste' on peut vendre ou acheter une voiture, nous avons effectué la présente étude. D'abord, nous avons collecté des données pertinentes et procéder à leur nettoyage. Ensuite, ce furent les phases de Nettoyage puis d'Analyse Exploratoire des données. Au cours de cette dernière phase, nous avons pu déterminer les variables qui influencent vraiment la valeur d'une voiture c'est-à-dire son prix. Enfin, nous avons construit

un modèle de régression linéaire puis terminé par l'évaluation de ce modèle. Le modèle a donné un bon résultat sur les données ayant servi à son entraînement. Mais le fait d'évaluer un modèle seulement sur les données d'entraînement ne nous dit pas s'il est capable de bien se comporter avec des nouvelles données. Or le but ultime de la création d'un modèle est de pouvoir l'utiliser pour effectuer des prédictions avec de nouvelles données. Un modèle peut avoir un très bon score d'entraînement et avoir une mauvaise performance sur de nouvelles données : problème de surapprentissage (*overfitting*). Dans les projets suivants, nous verrons plusieurs techniques permettant de vérifier la capacité d'un modèle de Machine Learning à se généraliser et aussi des techniques d'amélioration de sa qualité.

PROJET 2 : PREDICTION DU PRIX D'UNE MAISON EN FONCTION DE SES CARACTERISTIQUES

INTRODUCTION

Le marché de l'immobilier est l'un des secteurs économiques les plus importants au monde. Chaque année, ce sont des centaines de milliards de dollars de transactions immobilières qui sont effectuées. Les investisseurs les plus avertis, utilisent les outils de Data Science et de Machine Learning pour analyser les données et en tirer des informations importantes. Ces informations leur permettent de prendre de meilleures décisions par rapport à leurs investissements.

Dans ce projet, nous traiterons d'un cas d'application de l'apprentissage automatique supervisé à l'industrie immobilière. Il s'agira de construire un modèle de régression linéaire capable de prédire le prix d'une maison en fonction de certains de ses attributs.

LIBRAIRIES

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from scipy.stats import pearsonr

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error

from sklearn.model_selection import cross_val_score
```

DONNEES

L'ensemble de données que nous utiliserons contient des informations sur un ensemble de maison dans la région de Boston.

```
df = pd.read_csv('https://raw.githubusercontent.com/JosueAfouda/Boston-Housing/master/Boston.csv')
```

```
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	36.2

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   CRIM        506 non-null    float64
 1   ZN          506 non-null    float64
 2   INDUS       506 non-null    float64
 3   CHAS         506 non-null    int64  
 4   NOX         506 non-null    float64
 5   RM           506 non-null    float64
 6   AGE          506 non-null    float64
 7   DIS          506 non-null    float64
 8   RAD          506 non-null    int64  
 9   TAX          506 non-null    int64  
 10  PTRATIO     506 non-null    float64
 11  LSTAT        506 non-null    float64
 12  MEDV         506 non-null    float64
dtypes: float64(10), int64(3)
memory usage: 51.5 KB
```

La dataframe contient 506 lignes (chaque ligne représente une maison) et 13 variables. Pour comprendre la signification de chaque variable, veuillez-vous reporter à la [page descriptive](#)¹⁰ de ces données. La variable ***MEDV*** est notre variable d'intérêt. Elle indique la valeur médiane des logements occupés par leur propriétaire en 1000 \$.

Apparemment, il n'y a pas de valeurs manquantes dans la dataframe.

Nous pouvons aussi vérifier s'il y a des lignes complètes dupliquées et les supprimer.

```
# Nombre de lignes entières dupliquées  
  
df.duplicated().sum()
```

Il n'y a pas de lignes entières dupliquées.

ANALYSE EXPLORATOIRE

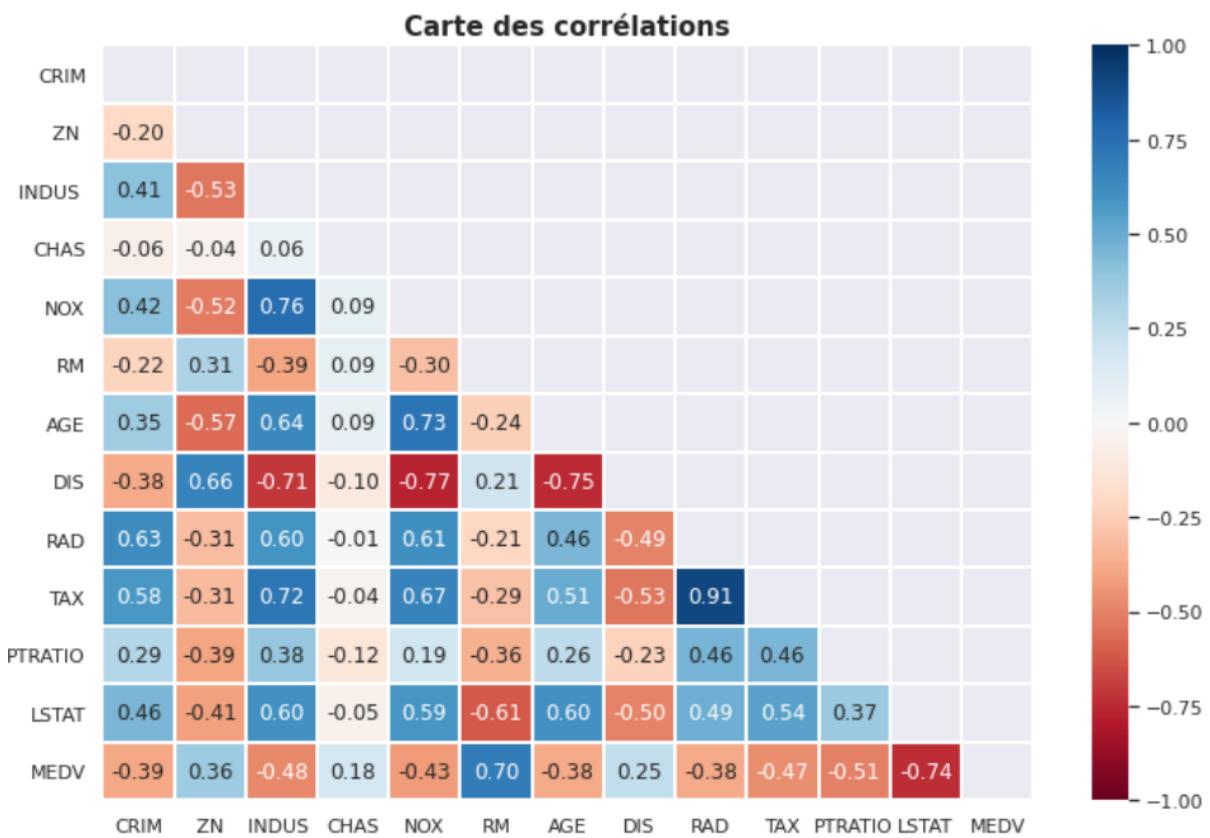
❖ RESUME STATISTIQUE DES DONNEES

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677082	88.9762
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.0000
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.7400
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.0000
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.8710
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8.7800
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100.0000
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12.1265
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24.0000
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.0000
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.0000
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.9700
MEDV	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50.0000

❖ CORRELATIONS ENTRE VARIABLES

```
# Carte des corrélations  
  
sns.set(rc={'figure.figsize':(12,8)})  
mask = np.triu(np.ones_like(df.corr(), dtype=bool))  
  
sns.heatmap(df.corr(), mask=mask, center=0, cmap='RdBu', linewidths=1, annot=True, fmt=".2f", vmin=-1, vmax=1)  
  
plt.title('Carte des corrélations', fontsize=15, fontweight="bold")  
  
plt.show()
```



Cette carte nous indique la force des relations linéaires entre les variables indépendantes d'une part et entre chacune de ces variables avec la variable cible. Plus le carré est rouge, plus la corrélation entre les deux variables est forte et négative. Plus le carré est bleu, plus la corrélation entre les deux variables est forte et positive. Par exemple, il y a une forte corrélation positive entre la variable cible (*MEDV*) et la variable *RM* qui indique le nombre moyen de pièces par logement. Autrement dit plus la maison a de pièces, plus son prix est élevé (ce qui est tout à fait logique).

❖ DISTRIBUTION DES VARIABLES

Nous pouvons visualiser d'un seul coup la distribution de chaque variable :

```
sns.pairplot(df)
```

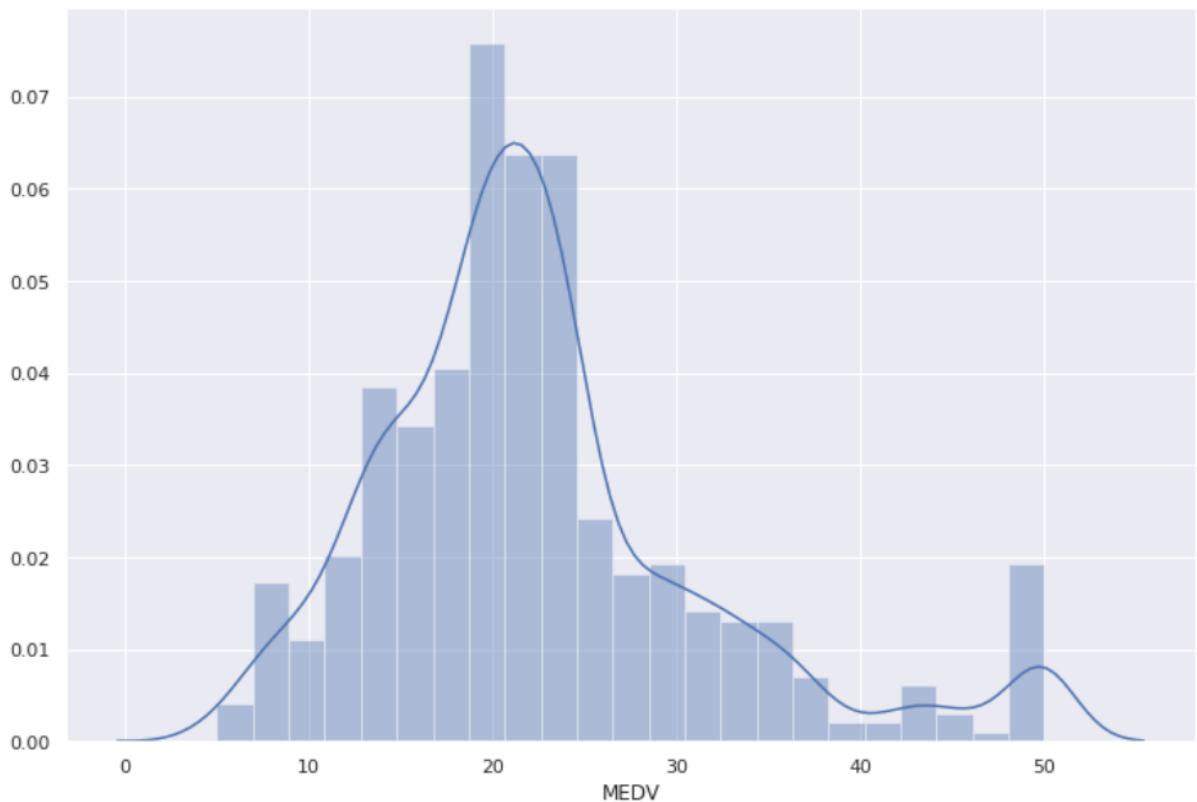
```
plt.show()
```

La fonction [pairplot\(\)](#)¹¹ trace des relations par paires dans un jeu de données. Par défaut, cette fonction crée une grille d'axes de telle sorte que chaque variable numérique dans les données sera partagée dans l'axe des y sur une seule ligne et dans l'axe des x sur une seule colonne. Les

axes diagonaux sont traités différemment, dessinant un tracé pour montrer la distribution univariée des données pour la variable dans cette colonne.

Nous pouvons aussi tracer séparément chaque histogramme. Traçons par exemple la distribution de la variable d'intérêt :

```
# Histogramme de la variable cible  
  
sns.distplot(df['MEDV'])  
  
plt.show()
```



Selon ce graphique, la distribution de la variable cible suit quasiment une loi normale.

Pourquoi s'intéresse-t-on si tant aux distributions des variables ?

La plupart des algorithmes de Machine Learning sont conçus selon l'hypothèse que les variables d'entrées suivent une loi normale. Si ce n'est pas le cas, cela peut affecter énormément la qualité du modèle et conduire à de mauvaises prédictions. Il existe plusieurs techniques de transformations de variables pour normaliser les variables (la transformation logarithmique par exemple). Afin d'être plus confiant sur la distribution de chaque variable par rapport à la loi

normale, nous allons calculer les degrés d'asymétrie. Une distribution normale a un degré d'asymétrie égal à 0. Donc une variable suit la loi normale lorsque son degré d'asymétrie est proche de 0.

```
# Calcul des degrés d'asymétrie

df.skew().sort_values()

      PTRATIO    -0.802325
      AGE        -0.598963
      INDUS      0.295022
      RM          0.403612
      TAX         0.669956
      NOX         0.729308
      LSTAT       0.906460
      RAD         1.004815
      DIS         1.011781
      MEDV        1.108098
      ZN          2.225666
      CHAS        3.405904
      CRIM        5.223149
      dtype: float64
```

D'après ces résultats, les variables ayant les plus forts degrés d'asymétrie sont : *ZN* (proportion de terrains résidentiels zonés pour les lots de plus de 25 000 *sq.ft.*), *CHAS* (Variable fictive de Charles River égale à 1 si la zone délimite la rivière et 0 sinon) et *CRIM* (taux de criminalité par habitant par ville).

```
# Transformation logarithmique des variables 'CRIM', 'RAD', 'DIS',
# 'MEDV', 'CHAS' et 'ZN'

df['CRIM'] = np.log(df['CRIM'])

df['RAD'] = np.log(df['RAD'])

df['DIS'] = np.log(df['DIS'])

df['MEDV'] = np.log(df['MEDV'])

df['CHAS'] = np.log(df['CHAS'] + 1)

df['ZN'] = np.log(df['ZN'] + 1)
```

```
# (Re)calcul des coefficients d'asymétrie

df.skew().sort_values()

      PTRATIO    -0.802325
      AGE        -0.598963
      MEDV       -0.330321
      DIS         0.152730
      RAD         0.286617
      INDUS      0.295022
      RM          0.403612
      CRIM      0.405934
      TAX         0.669956
      NOX         0.729308
      LSTAT      0.906460
      ZN          1.193451
      CHAS        3.405904
      dtype: float64
```

Supprimons la variable 'CHAS' à cause de son coefficient d'asymétrie trop élevé par rapport à celui des autres variables :

```
df.drop('CHAS', axis = 1, inplace = True)
```

Traçons une nouvelle carte des corrélations afin de voir si les relations linéaires ont été renforcées ou pas après les transformations logarithmiques de certaines variables :

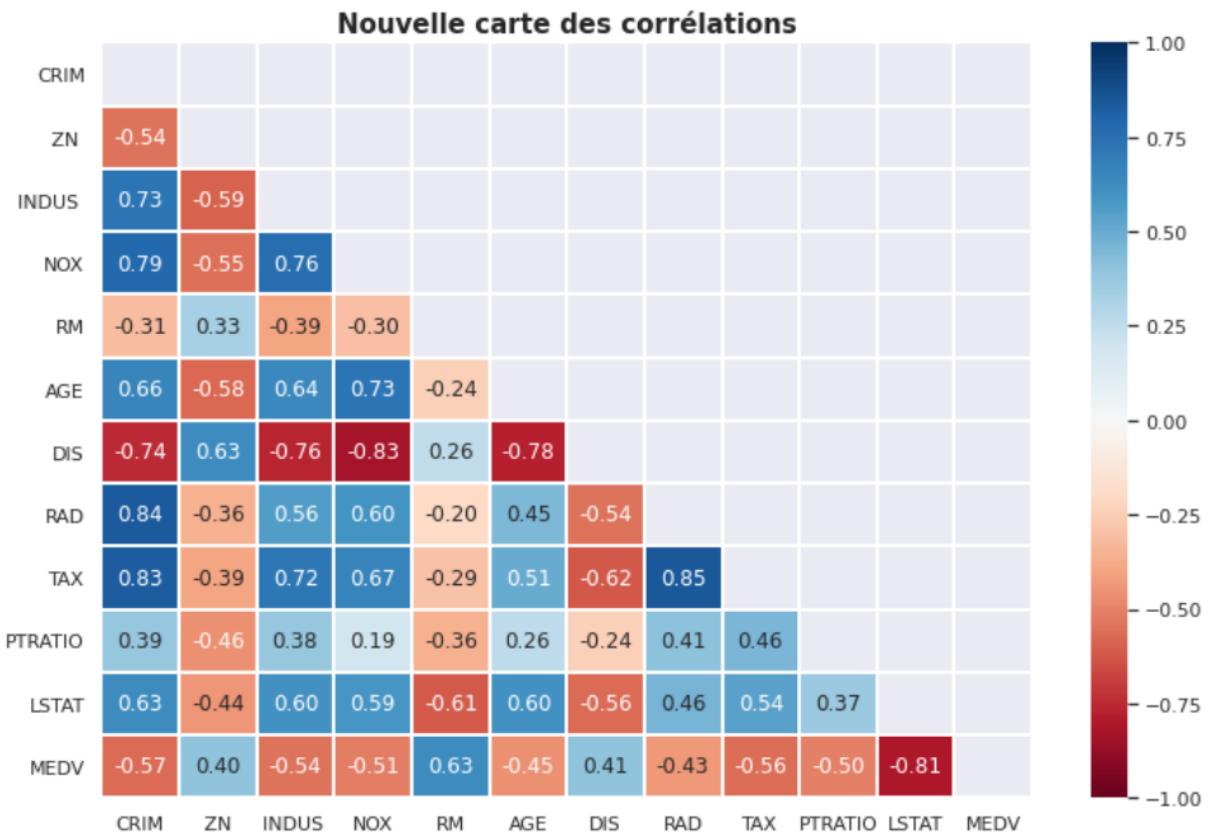
```
# Nouvelle carte des corrélations
sns.set(rc={'figure.figsize':(12,8)})

mask = np.triu(np.ones_like(df.corr()), dtype=bool)

sns.heatmap(df.corr(), mask=mask, center=0, cmap='RdBu', linewidths=1, annot=True, fmt=".2f", vmin=-1, vmax=1)

plt.title('Nouvelle carte des corrélations', fontsize=15, fontweight="bold")

plt.show()
```



En comparant cette nouvelle carte des corrélations à l'ancienne, nous remarquons effectivement un renforcement de la relation linéaire entre plusieurs variables. Par exemple, la relation linéaire entre la variable ‘CRIM’ et la variable cible s'est nettement améliorée (-0,39 à -0,57).

MODELISATION

Avant de passer à la construction du modèle, divisons d'abord la dataframe en données qui serviront à entraîner le modèle (*train data*) et en données d'évaluation de la performance du modèle (*test data*). Ainsi, nous pourrons mesurer la capacité de notre modèle à s'ajuster à de nouvelles données. Cette division se fait avec la fonction [train_test_split\(\)](#).¹²

```
# Dataframe des variables indépendantes
```

```
X = df.drop('MEDV', axis = 1)
```

```
# Variable cible
```

```
y = df['MEDV']
```

Pour obtenir la dataframe des variables indépendantes, nous avons juste supprimer la variable cible de df.

```

# Train/Test data

seed = 111

test_size = 0.2

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = test_size, random_state = seed)

# Affichage des dimensions

print("X_train :", X_train.shape)

print("y_train :", y_train.shape)

print("X_test :", X_test.shape)

print("y_test :", y_test.shape)

X_train : (404, 12)
y_train : (404,)
X_test : (102, 12)
y_test : (102,)

```

Lorsqu'on divise les données, la grande partie est utilisée comme données d'entraînement et le reste comme données d'évaluation. Ici, nous avons choisi une taille de données de test égale à 20% du total des observations. Le paramètre *random_state* permet d'obtenir une division reproductible à chaque fois que nous exécutons ce code.

Passons à présent à la construction du modèle.

```

# Création du modèle

model = LinearRegression()

# Entraînement de l'algorithme

model.fit(X_train, y_train)

```

Nous avons d'abord défini un modèle de régression linéaire (variable *model*) puis nous l'avons ajusté aux données d'entraînement. La méthode d'ajustement *fit()* de l'objet modèle a été appelée pour ajuster le modèle de régression linéaire aux données. L'ajustement consiste à

estimer les coefficients de régression (paramètres) en utilisant la [méthode des moindres carrés ordinaires](#).¹³

Une fois que le modèle est ajusté aux données, nous pouvons calculer certains paramètres comme le coefficient de détermination (R^2), l'erreur quadratique moyenne (MSE) et afficher les coefficients ainsi que l'ordonnée à l'origine.

```
# Coefficient de détermination
```

```
model.score(X_train, y_train)
```

```
0.7599308321478287
```

Le coefficient de détermination est égal à 0,76.

```
# Coefficients du modèle
```

```
model.coef_
```

Connaissant ces coefficients, nous pouvons écrire une équation de la variable cible égale à une combinaison linéaire des prédicteurs.

Avant de calculer l'erreur quadratique moyenne, nous devons générer les prédictions c'est-à-dire les valeurs de la variable *MEDV* calculées par notre modèle.

```
# Prédictions sur le train data
```

```
y_pred = model.predict(X_train)
```

```
# Erreur quadratique moyenne
```

```
mean_squared_error(y_train, y_pred)
```

```
0.03970543844863186
```

L'erreur quadratique moyenne est égale à 0,04

Afin de voir si notre modèle est capable de bien s'ajuster à de nouvelles données, calculons le coefficient de détermination et l'erreur quadratique moyenne cette fois-ci sur les données de test.

```
# R2 sur le test data  
  
model.score(X_test, y_test)  
  
0.7335793246239515
```

Le R² des données de test (0,73) est inférieur à celui des données d'entraînement (0,76).

```
# Prédictions sur le test data  
  
y_hat = model.predict(X_test)  
  
# Erreur quadratique moyenne  
  
mean_squared_error(y_test, y_hat)  
  
0.04568839722859577
```

L'erreur quadratique moyenne du modèle sur les données de test est égale à environ 0,04 comme celui des données de test. Nous pouvons donc conclure qu'il n'y a pas un problème de surajustement et que notre modèle est capable de bien se généraliser.

❖ VALIDATION CROISEE DU MODELE

En divisant nos données, nous avons pu calculer certaines métriques sur les données de test afin de nous assurer que le modèle est capable de bien s'ajuster à de nouvelles données. Mais, il y a un écueil dans ce processus. Les données de test peuvent avoir certaines particularités qui font que le R² (ou l'erreur quadratique moyenne) ait donné telle ou telle autre valeur. Donc la métrique calculée n'est pas bien représentative de la capacité du modèle à généraliser sur de nouvelles données. Pour pallier à ce problème dû essentiellement à la division arbitraire de la fonction *train_test_split()*, nous utiliserons la technique de la validation croisée ([cross-validation](#)).¹⁴

Pour comprendre la technique de cross-validation :

- Commençons par diviser la dataframe en 5 groupes ou plis ;
- Ensuite, le premier pli est pris comme ensemble de test et l'algorithme est entraîné avec les 4 plis restants. On fait les prédictions sur l'ensemble de test puis on calcule la métrique d'évaluation du modèle ;
- Le deuxième pli est maintenant utilisé comme ensemble de test et les 4 autres plis comme données d'entraînement ;

- On fait de même jusqu'à ce que chacun des 5 plis ait été utilisé comme ensemble de test.

Finalement, on se retrouve avec 5 modèles donc 5 métriques (R^2 par exemple) et on peut calculer des statistiques comme la moyenne ou la médiane de ces métriques. Au lieu de 5 plis, vous pouvez choisir k plis (**k -fold cross validation**) avec k un nombre entier naturel non nul. Mais attention au nombre de plis car plus vous avez de plis, plus cela est coûteux en termes de calcul. La méthode de validation croisée nous permet d'éviter que la métrique d'évaluation choisie ne dépende du fractionnement arbitraire de la fonction `train_test_split()`.

Appliquons la technique de validation croisée avec 5 plis.

```
# Création d'un modèle

linear_model = LinearRegression()

# 5-fold cross-validation

cv_scores = cross_val_score(linear_model, X, y, cv = 5)

cv_scores

array([0.63039073, 0.7542449 , 0.59337479, 0.4822631 , 0.45586857])
```

Par défaut, [`cross_val_score\(\)`](#)¹⁵ utilise R^2 comme métrique de choix pour la régression. Donc nous avons 5 R^2 et nous pouvons calculer la moyenne et la médiane.

```
np.mean(cv_scores)

0.5832284181249279

np.median(cv_scores)

0.5933747925719739
```

Le R^2 médian est égal à 0,59. Bien que cette valeur soit plus faible que les 0,73 obtenu en faisant un simple `train_test_split()`, elle est beaucoup plus vérifique. Nous pouvons valider le modèle et être plus confiant sur sa qualité.

```
# Modèle final

lm = LinearRegression()
lm.fit(X, y)
```

```

# Coefficients

lm.coef_


array([-2.43563529e-02,  7.31048069e-04,  3.06055544e-03, -7.79681053e-01,
       9.64873383e-02,  1.89928201e-04, -2.12488554e-01,  1.02839484e-01,
      -6.27712028e-04, -3.70149683e-02, -3.29179525e-02])

# Ordonnée à l'origine

lm.intercept_


4.210068806396677

```

Nous pouvons écrire l'équation du modèle qui servira pour le calcul de la valeur d'une nouvelle maison dont on connaît les caractéristiques :

$$\begin{aligned}
\text{MEDV} = & -2.43563529e-02 \times \text{CRIM} + 7.31048069e-04 \times \text{ZN} \\
& + 3.06055544e-03 \times \text{INDUS} - 7.79681053e-01 \times \text{NOS} \\
& + 9.64873383e-02 \times \text{RM} + 1.89928201e-04 \times \text{AGE} \\
& - 2.12488554e-01 \times \text{DIS} + 1.02839484e-01 \times \text{RAD} \\
& - 6.27712028e-04 \times \text{TAX} - 3.70149683e-02 \times \text{PTRATIO} \\
& - 3.29179525e-02 \times \text{LSTAT}
\end{aligned}$$

Attention aux transformations logarithmiques. Pour prédire la valeur d'une nouvelle maison, il faudra nécessairement effectuer les mêmes transformations logarithmiques. Le MEDV de l'équation ci-dessus est une valeur log. Pour avoir la vraie valeur, il faut appliquer la fonction [`numpy.exp\(\)`](#)¹⁶

CONCLUSION

La régression linéaire est très utilisée dans les problèmes de Machine Learning. Dans ce projet, nous avons vu comment construire et évaluer un modèle de régression linéaire dans Python.

Les deux premiers projets de ce livre nous ont permis d'être plongé dans la pratique du Machine Learning. De façon précise, nous avons vu comment traiter un problème de régression. Le flux de travail sera pratiquement le même pour des problèmes de classification.

Dans le prochain projet, nous traiterons un problème de classification.

PROJET 3 : MODELISATION DU RISQUE DE CREDIT (MODELE DE SCORING)

COMPREHENSION DU PROBLEME

Lorsqu'une banque prête de l'argent à une personne, elle prend le risque que cette dernière ne rembourse pas cet argent dans le délai convenu. Ce risque est appelé Risque de Crédit. Alors avant d'octroyer un crédit, les banques vérifient si le client (ou la cliente) qui demandent un prêt sera capable ou pas de le rembourser. Cette vérification se fait grâce à l'analyse de plusieurs paramètres tels que les revenus, les biens, les dépenses actuelles du client, etc. Cette analyse est effectuée manuellement par plusieurs banques. Ainsi, elle est très consommatrice en temps et en ressources financières.

Grâce au Machine Learning, il est possible d'automatiser cette tâche. Dans ce projet, nous allons construire un algorithme capable de prédire si une personne sera en défaut de paiement ou pas (1 : défaut, 0 : non-défaut). Nous sommes donc face à un problème de classification car nous voulons prédire une variable discrète (binaire pour être précis).

LIBRAIRIES

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_report

from sklearn.model_selection import RandomizedSearchCV
```

DONNEES

Deux (02) types de données peuvent être utilisés pour modéliser la probabilité de défaut de paiement :

- Données liées à la demande de crédit ;
- Données comportementales décrivant le bénéficiaire du prêt.

Dans la pratique, les banques utilisent un mélange de ces deux types de données pour construire leur modèle de *scoring* appliqué au risque de crédit.

Commençons par l'importation des données :

```
df = pd.read_csv('https://github.com/JosueAfouda/Credit-Risk-  
Modeling/raw/master/data_credit.txt')  
  
df.head()  
  
# Informations  
  
df.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 32581 entries, 0 to 32580  
Data columns (total 12 columns):  
 #   Column           Non-Null Count  Dtype    
 ---  --     
 0   person_age      32581 non-null   int64   
 1   person_income    32581 non-null   int64   
 2   person_home_ownership 32581 non-null   object   
 3   person_emp_length 31686 non-null   float64   
 4   loan_intent     32581 non-null   object   
 5   loan_grade      32581 non-null   object   
 6   loan_amnt       32581 non-null   int64   
 7   loan_int_rate   29465 non-null   float64   
 8   loan_status     32581 non-null   int64   
 9   loan_percent_income 32581 non-null   float64   
 10  cb_person_default_on_file 32581 non-null   object   
 11  cb_person_cred_hist_length 32581 non-null   int64  
dtypes: float64(3), int64(5), object(4)  
memory usage: 3.0+ MB
```

L'ensemble des données comporte 12 variables et 32581 observations (lignes) historiques. Chaque observation correspond à une personne ayant contracté un prêt. On a des variables qui décrivent le crédit (montant, statut, taux d'intérêt, etc.) et d'autres variables qui décrivent la personne ayant contracté ce crédit (âge, revenu, etc.). Nous allons donc utiliser ces données historiques afin de construire le modèle de *scoring* qui va prédire le statut des nouveaux candidats à un crédit.

Il est très important de comprendre les variables de notre jeu de données :

- **person_age** : variable indiquant l'âge de la personne ;
- **person_income** : variable indiquant les revenus (ou encore le salaire) de la personne ;
- **person_home_ownership** : variable indiquant le statut de la personne par rapport à son lieu d'habitation (propriétaire, locataire, etc.) ;
- **person_emp_length** : variable indiquant la durée (en mois) depuis laquelle la personne est en activité professionnelle ;
- **loan_intent** : variable indiquant le motif du crédit ;
- **loan_grade** : variable indiquant le grade du prêt ;
- **loan_amnt** : variable indiquant le montant du prêt ;
- **loan_int_rate** : variable indiquant le taux d'intérêt du crédit ;
- **loan_status** : c'est la variable d'intérêt. Elle indique si la personne est en défaut de paiement (1) ou pas (0) ;
- **loan_percent_income** : variable indiquant le pourcentage des revenus par rapport au crédit ;
- **cb_person_default_on_file** : variable indiquant si la personne ait à découvert ou pas ;
- **cb_person_cred_hist_length** : variable indiquant la durée des antécédents de crédits.

Passons à présent à l'analyse exploratoire des données qui nous permettra de mieux les comprendre.

ANALYSE EXPLORATOIRE DES DONNEES

```
# Résumé statistique des données
```

```
df.describe()
```

Nous remarquons que les moyennes et les écart-types sont très différents d'une variable à une autre. Cela indique que les données ne sont pas à la même échelle. Selon l'algorithme que nous

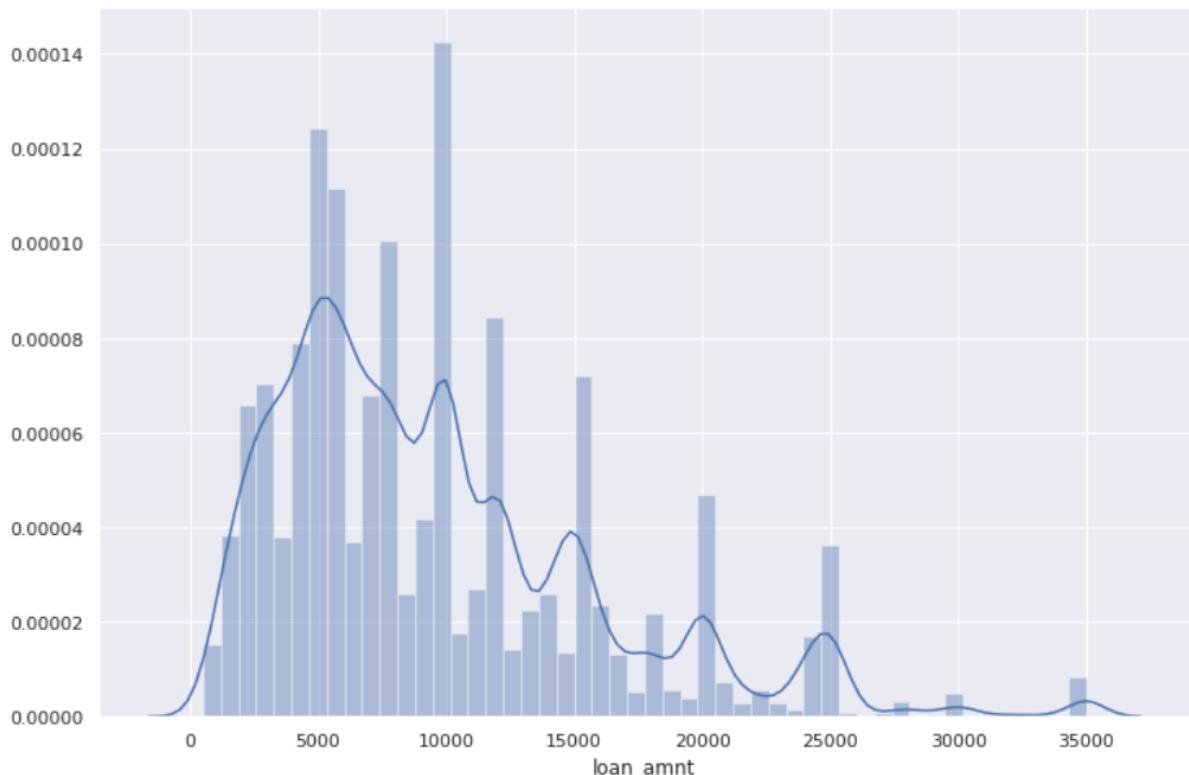
utiliserons, il faudra probablement normaliser les données avant de les modéliser. En effet, certains algorithmes de Machine Learning nécessitent une normalisation des données pour un meilleur résultat.

Une manière très rapide de visualiser les distributions des variables ainsi que les nuages de points est d'utiliser la fonction ***pairplot()*** de Seaborn :

```
# Distribution des variables et nuages de points  
  
sns.pairplot(df)  
  
plt.show()
```

Nous pouvons mieux visualiser les histogrammes avec la fonction ***distplot()*** de Seaborn :

```
# Distribution de la variable 'loan_amnt'  
  
sns.set(rc={'figure.figsize':(12,8)})  
  
sns.distplot(df['loan_amnt'])  
  
plt.show()
```



```
# Coefficients d'asymétrie

df.skew()

person_age           2.581393
person_income         32.865349
person_emp_length    2.614455
loan_amnt            1.192477
loan_int_rate         0.208550
loan_status           1.364888
loan_percent_income   1.064669
cb_person_cred_hist_length 1.661790
dtype: float64
```

Le coefficient de la variable *person_income* est très élevé. Pour corriger cela, faisons une transformation logarithmique de cette variable :

```
# Transformation log de la variable 'person_income'

df['person_income'] = np.log(df['person_income'])

# Coefficient d'asymétrie de la variable transformée

df['person_income'].skew()

0.1559408016162584
```

Voilà qui est bien mieux !

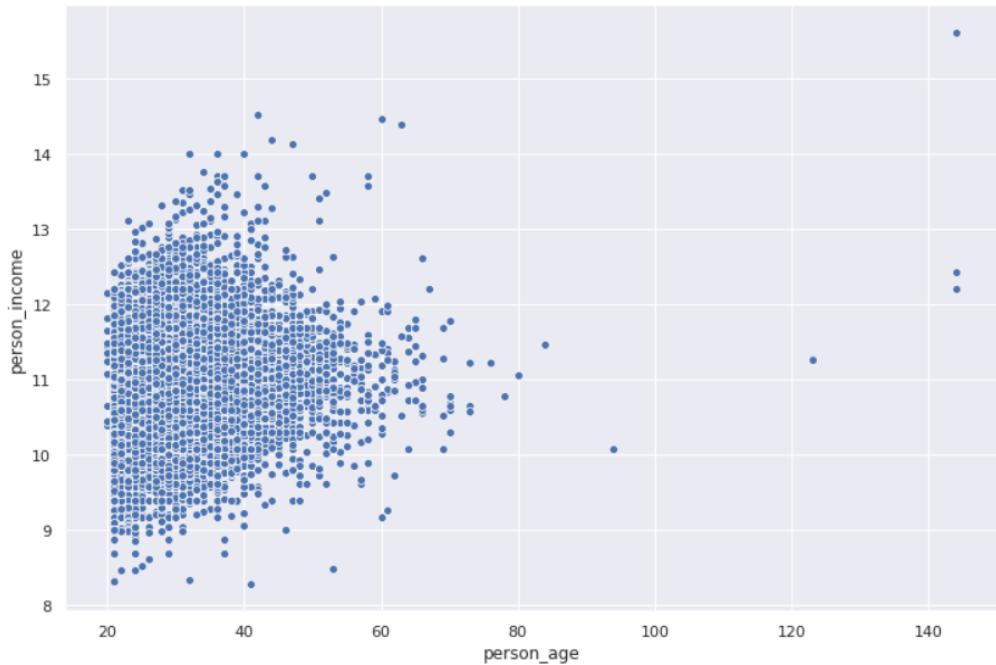
Traçons à présent le nuage des points des revenus en fonction de l'âge :

```
# Pour tracer un nuage de point

sns.set(rc={'figure.figsize':(12,8)})

sns.scatterplot(x='person_age', y='person_income', data=df) # Revenu en fonction de l'âge

plt.show()
```



Le graphique ci-dessus nous montre qu'il y une très faible corrélation positive entre les revenus et l'âge. Nous remarquons aussi des *outliers* (ce sont les valeurs 'aberrantes', c'est-à-dire des valeurs qui sortent du lot).

Analysons à présent la fréquence des modalités de chaque variable catégorielle :

```
# Faisons une boucle for pour construire un diagamme à barre pour toutes les variables qualitatives

for col in df.select_dtypes(include=['object']).columns.to_list() + ['loan_status']:

    sns.countplot(df[col])

    plt.xlabel(col, fontweight="bold")

    plt.ylabel('Nombre', fontweight="bold")

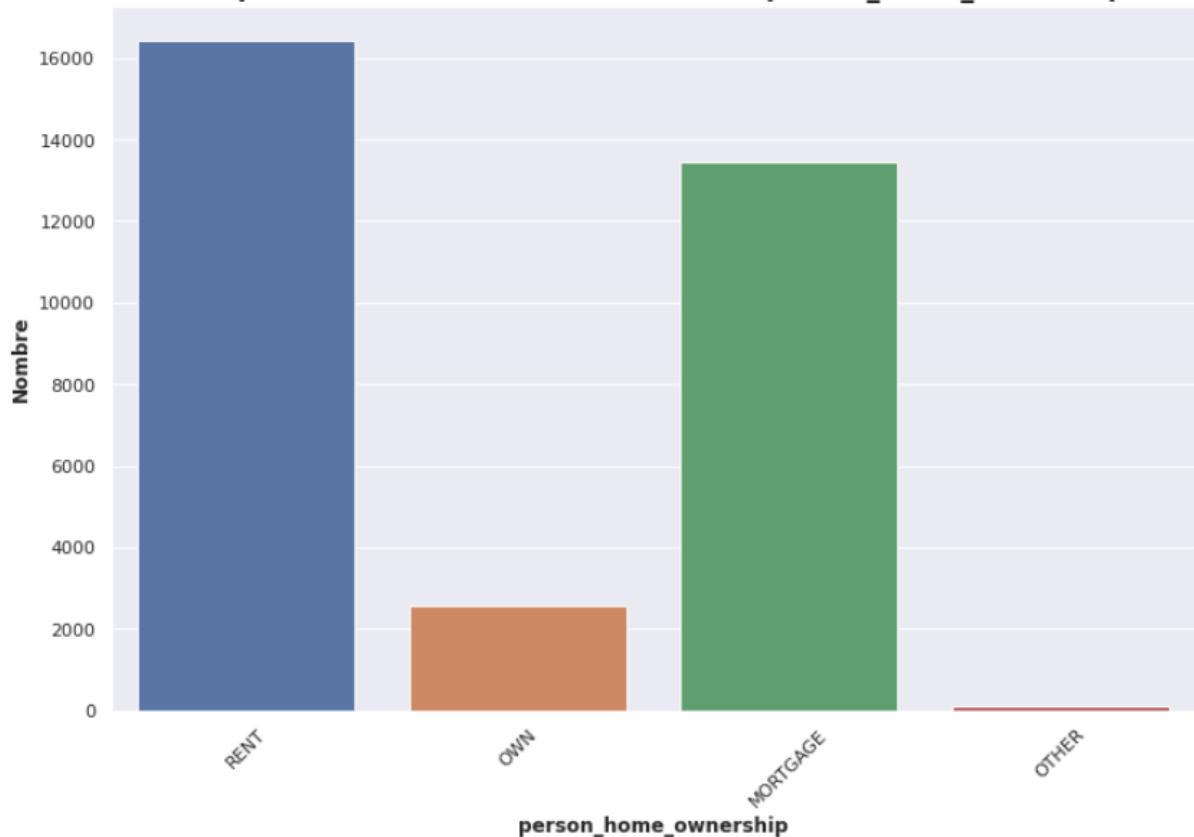
    plt.xticks(rotation = 45)

    plt.title('Fréquence des modalités de la variable ' + str(col), fontsize=16, fontweight="bold")

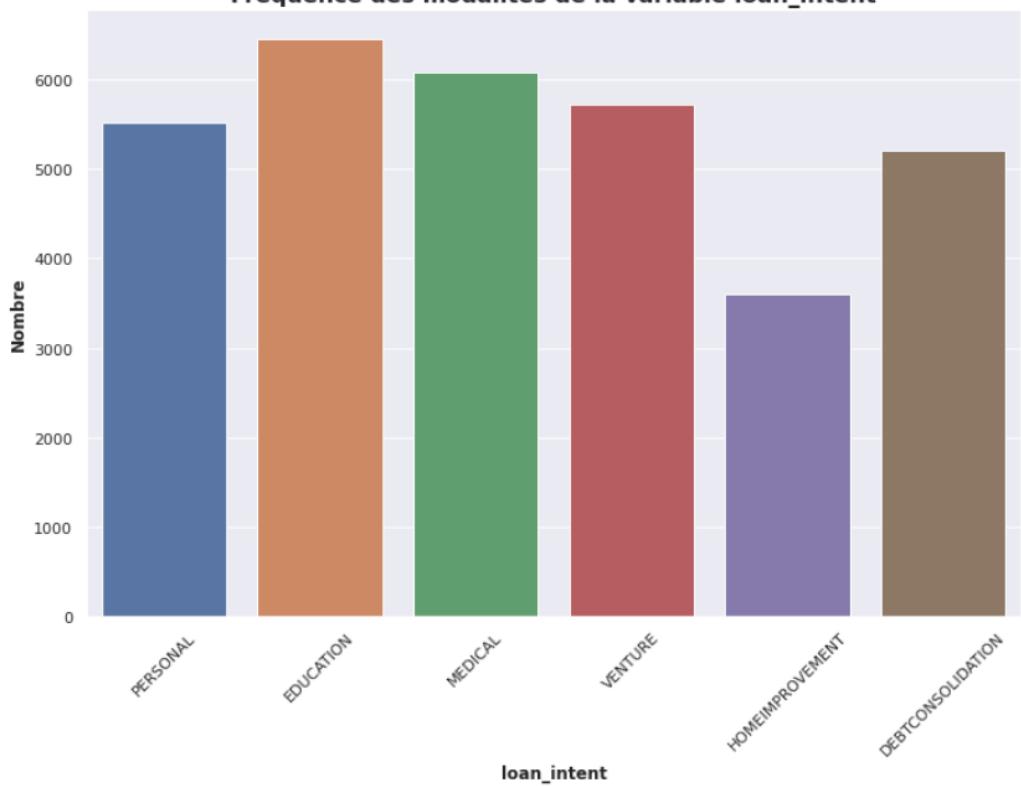
    plt.show()

    print('\n')
```

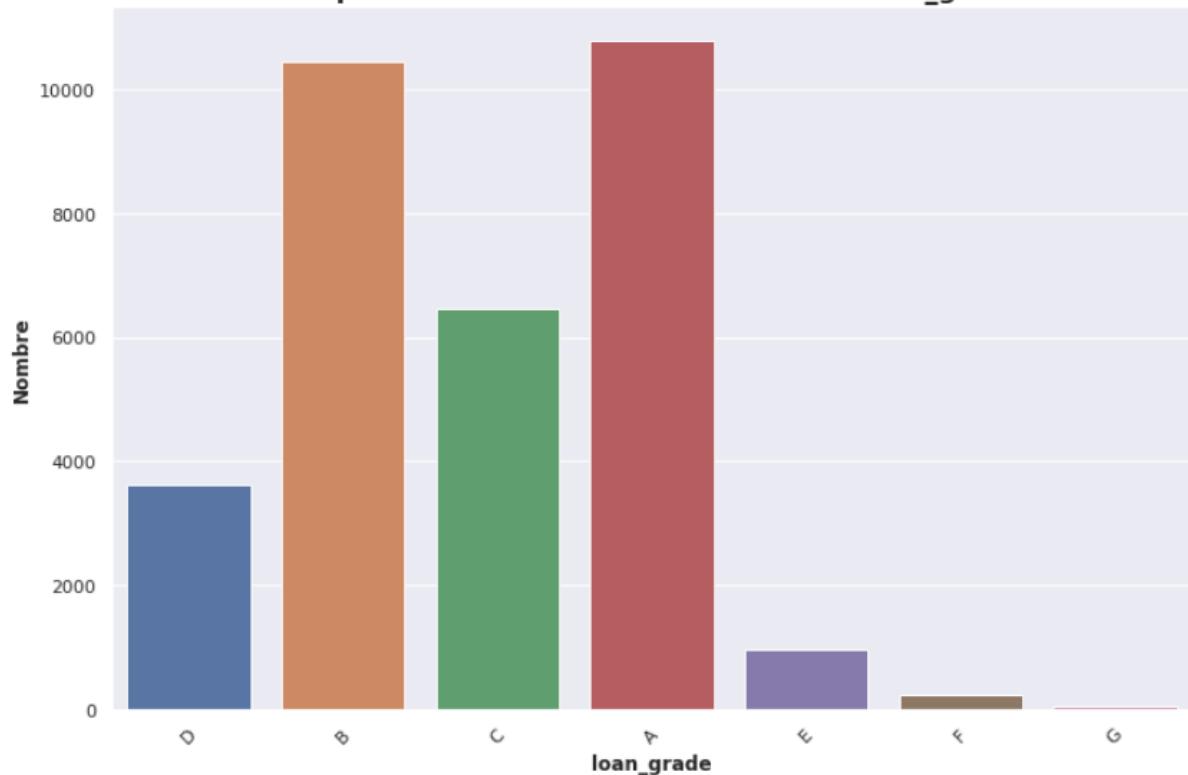
Fréquence des modalités de la variable person_home_ownership



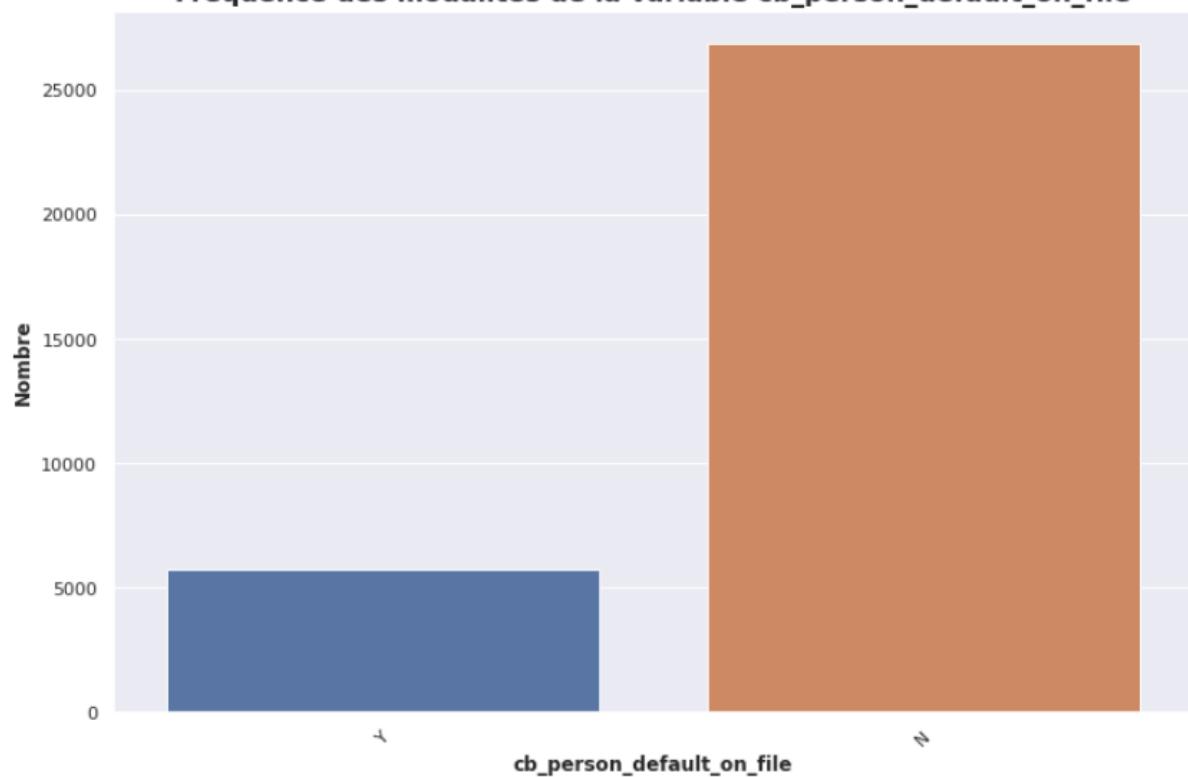
Fréquence des modalités de la variable loan_intent

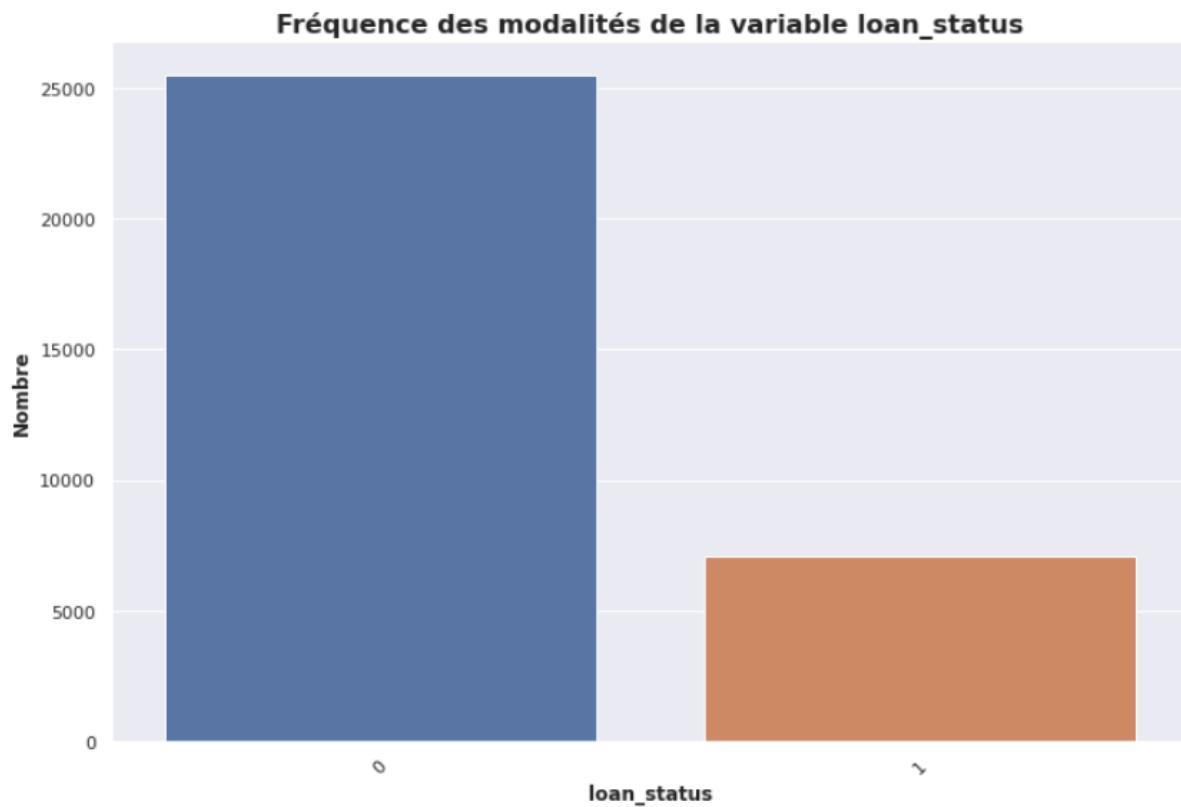


Fréquence des modalités de la variable loan_grade



Fréquence des modalités de la variable cb_person_default_on_file





Nous remarquons un déséquilibre de classe au niveau de la variable d'intérêt (*loan_status*). Il y a beaucoup plus de personnes qui n'ont pas été en défaut de paiement que de personnes en défaut de paiement. Ce problème peut impacter la qualité de notre modèle car ce dernier pourrait avoir tendance à prédire que tous les nouveaux demandeurs de prêts ne seront pas en défaut de paiement. Ceci causerait d'énormes pertes d'argent à la banque. Il existe plusieurs méthodes pour traiter les problèmes de déséquilibre de classe. Nous utiliserons l'une de ces méthodes dans la partie prétraitement des données avant la modélisation.

Très souvent, les données financières sont souvent analysées avec des tableaux croisés dynamiques comme dans Excel. Construisons alors quelques tableaux croisés dynamiques :

```
# Tableau croisé dynamique entre le motif de prêt et le statut du prêt
```

```
pd.crosstab(df["loan_intent"], df["loan_status"], margins = True)
```

	loan_status	0	1	All
	loan_intent			
DEBTCONSOLIDATION	3722	1490	5212	
EDUCATION	5342	1111	6453	
HOMEIMPROVEMENT	2664	941	3605	
MEDICAL	4450	1621	6071	
PERSONAL	4423	1098	5521	
VENTURE	4872	847	5719	
All	25473	7108	32581	

```
# Tableau croisé dynamique de l'accès à la propriété groupé par le statut ainsi que le grade du crédit
```

```
pd.crosstab(df["person_home_ownership"], [df["loan_status"], df["loan_grade"]])
```

	loan_status	0							1						
	loan_grade	A	B	C	D	E	F	G	A	B	C	D	E	F	G
	person_home_ownership														
MORTGAGE	5219	3729	1934	658	178	36	0	239	324	321	553	161	61	31	
OTHER	23	29	11	9	2	0	0	3	5	6	11	6	2	0	
OWN	860	770	464	264	26	7	0	66	34	31	18	31	8	5	
RENT	3602	4222	2710	554	137	28	1	765	1338	981	1559	423	99	27	

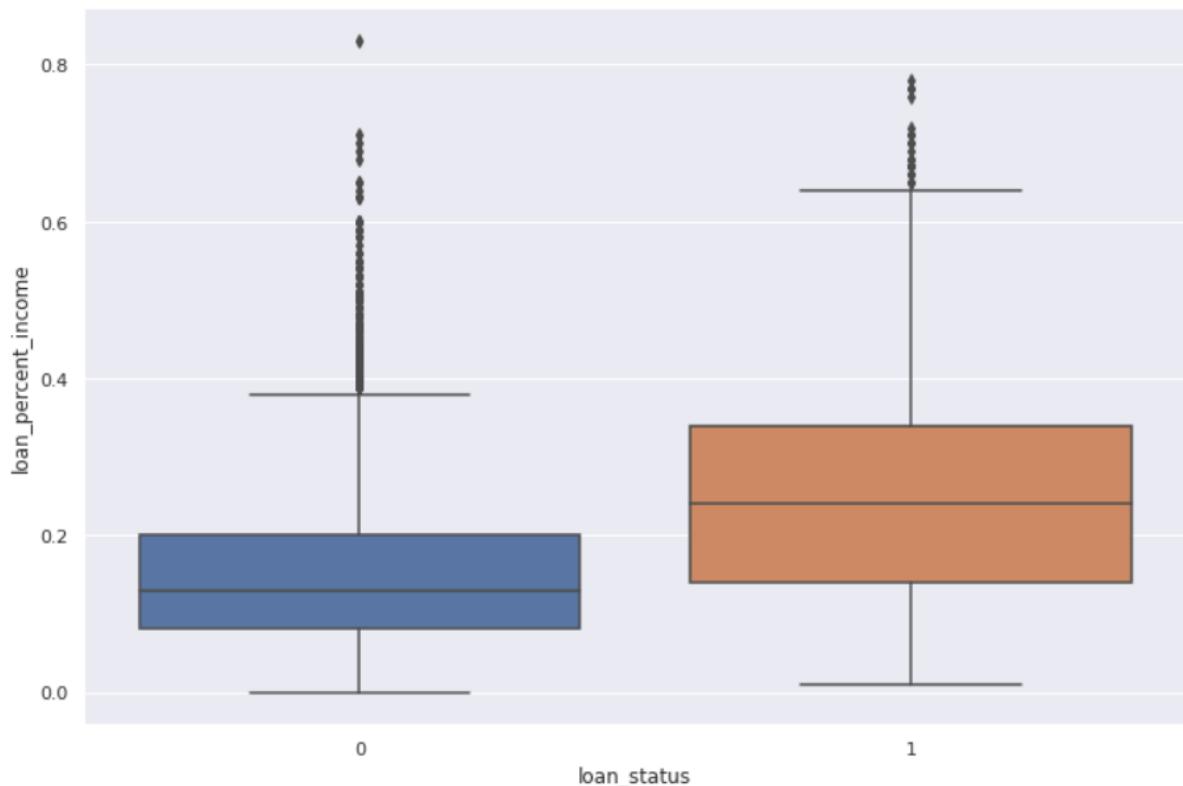
```
# Tableau croisé dynamique du % moyen du revenu par rapport au montant du prêt groupé par les variables "person_home_ownership" et 'loan_status'
```

```
pd.crosstab(df['person_home_ownership'], df['loan_status'], values=df['loan_percent_income'], aggfunc='mean')
```

loan_status	0	1
person_home_ownership		
MORTGAGE	0.146504	0.184882
OTHER	0.143784	0.300000
OWN	0.180013	0.297358
RENT	0.144611	0.264859

Par exemple, le revenu de ceux qui ont leur propre maison (donc les propriétaires) et qui sont en défaut de paiement est égal en moyenne à 29,73% du montant de leur crédit.

Quelle est la relation entre le pourcentage du revenu et le statut du crédit ?



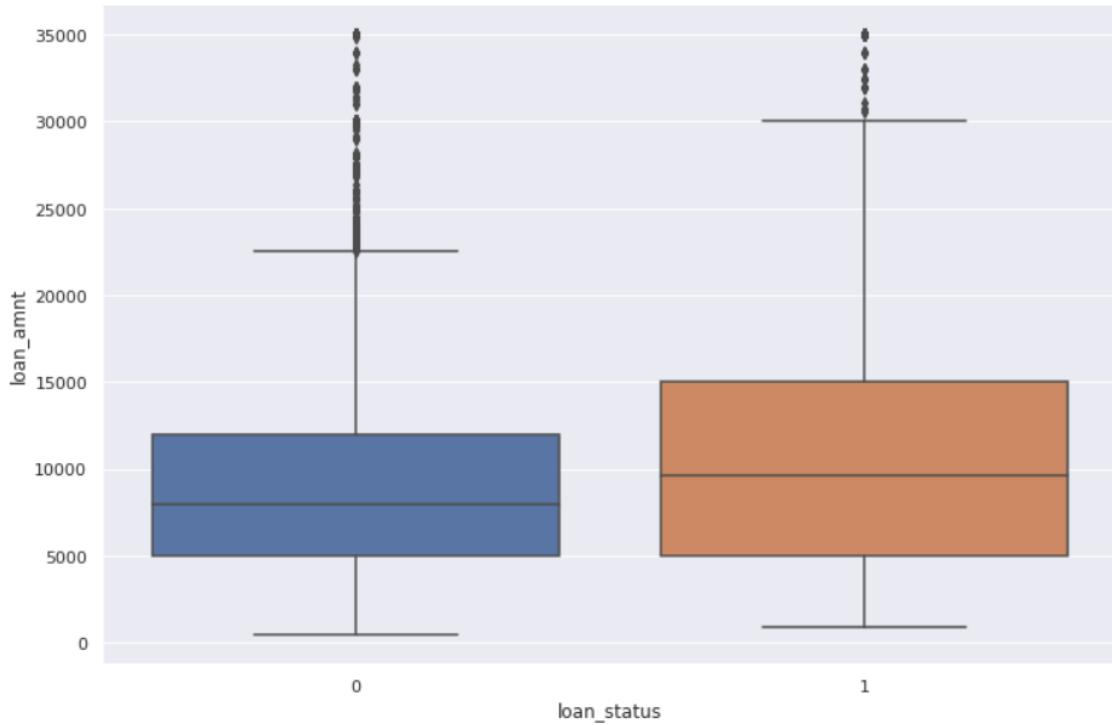
Selon ce graphique, le pourcentage moyen de revenu (par rapport au montant du prêt) pour les personnes en défaut de paiement est plus élevé. Cela indique que ces bénéficiaires ont un ratio dette / revenu déjà trop élevé. En effet, lorsque le ratio dette / revenu est très important, cela

veut dire tout simplement que le crédit est trop élevé par rapport aux revenus de la personne. Et lorsqu'on a un crédit trop élevé par rapport à ses revenus, le risque d'être en défaut de paiement est logiquement et naturellement élevé.

```
# loan_amnt vs loan_status

sns.boxplot(x='loan_status', y='loan_amnt', data=df)

plt.show()
```



Globalement, les personnes en défaut de paiement ont un crédit supérieur à celui des personnes qui ont remboursé leurs prêts.

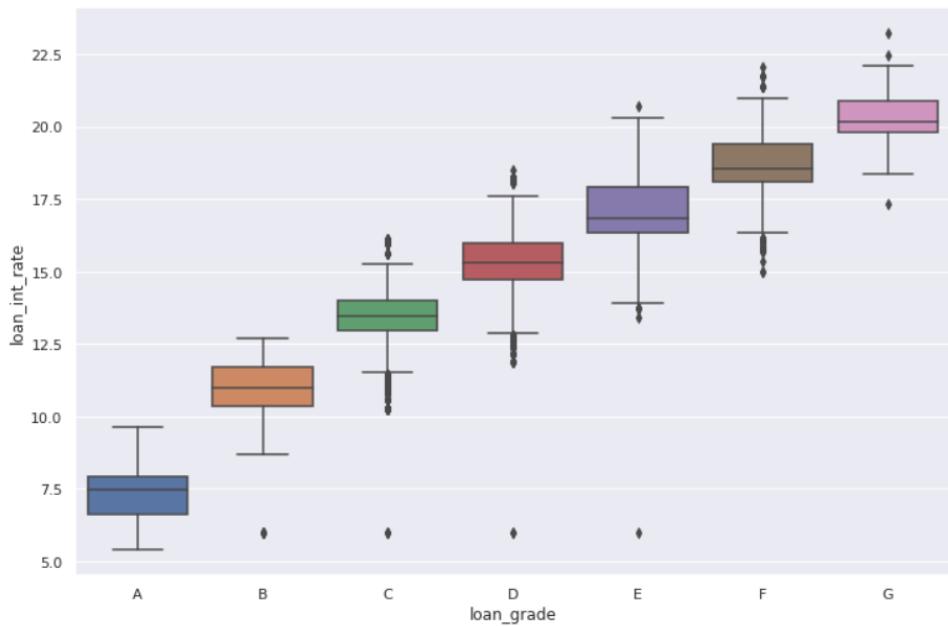
```
# loan_int_rate vs loan_grade

credit_grade = ["A", "B", "C", "D", "E", "F", "G"]

sns.set(rc={'figure.figsize':(12,8)})

sns.boxplot(x='loan_grade', y='loan_int_rate', data=df, order=
credit_grade)

plt.show()
```



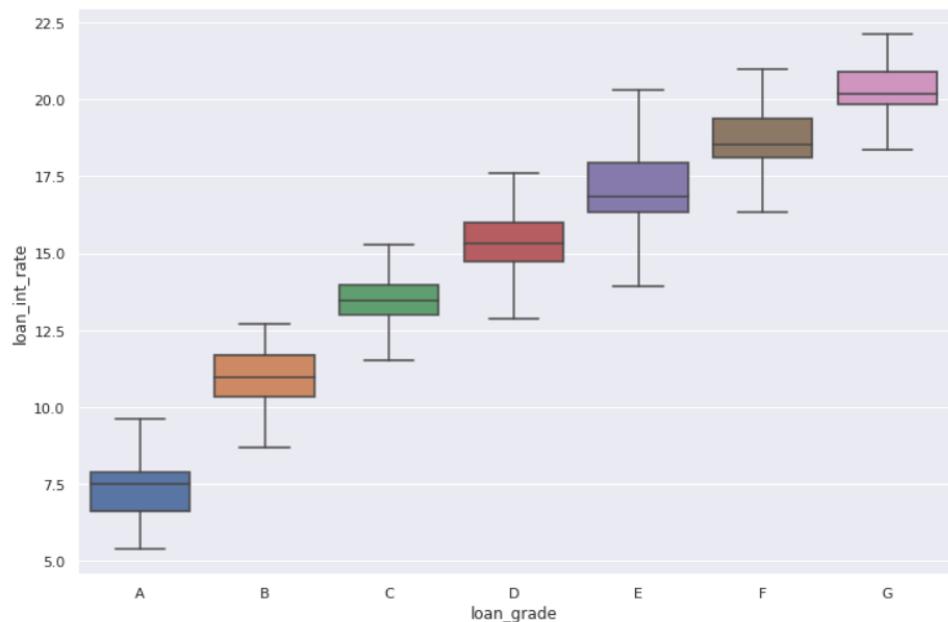
Plus le crédit est important (grade élevé), plus le taux d'intérêt est élevé.

Construisons le même graphique sans les *outliers* pour voir si on la même tendance :

```
# Même graphique sans les outliers
```

```
sns.set(rc={'figure.figsize':(12,8)})

sns.boxplot(x='loan_grade', y='loan_int_rate', sym='', data=df
, order=credit_grade)
plt.show()
```



❖ CORRELATIONS

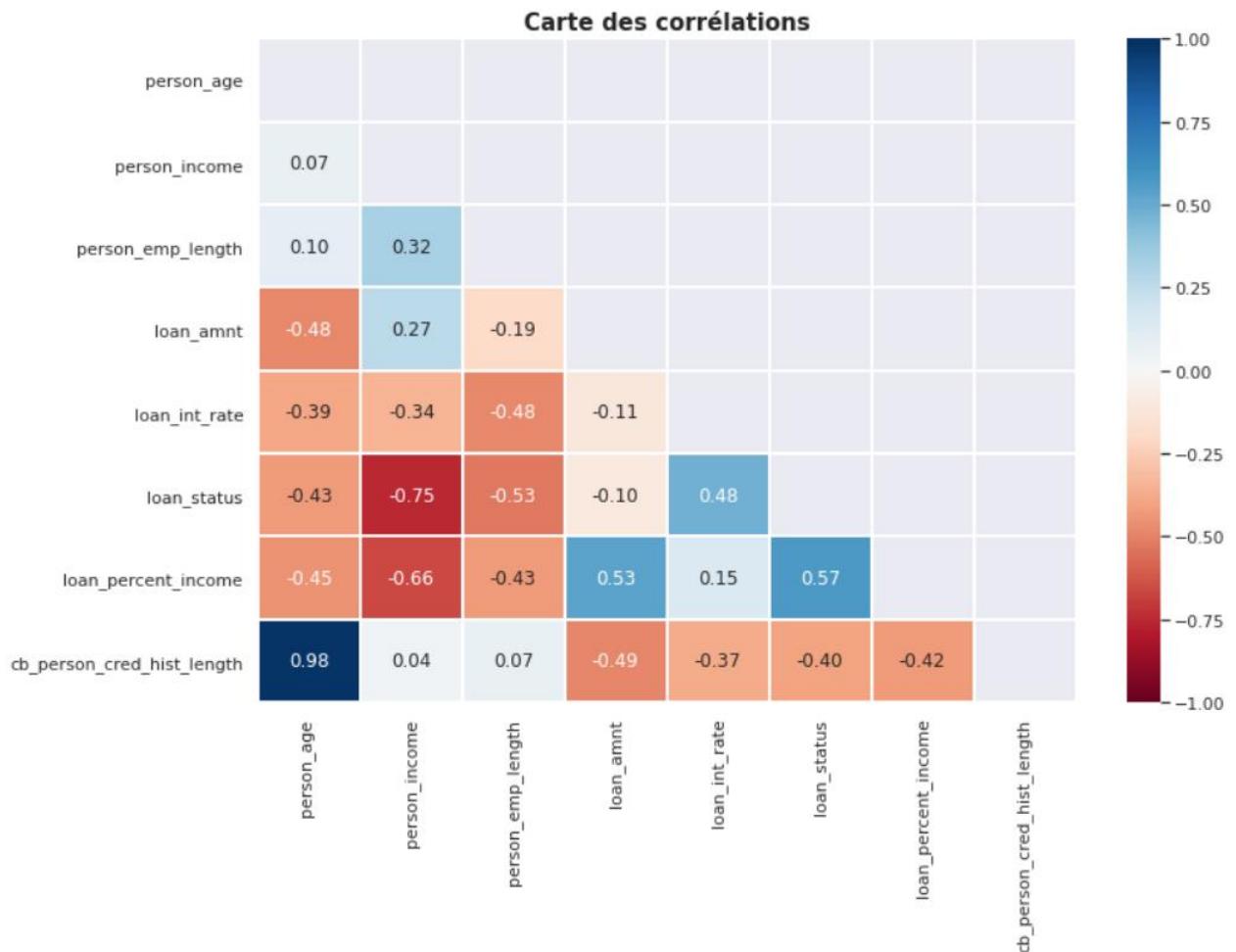
```
# Carte des corrélations

mask = np.triu(np.ones_like(df.select_dtypes(exclude=['object']),
                           corr(), dtype=bool))

sns.heatmap(df.select_dtypes(exclude=['object']).corr().corr(),
            mask=mask, center=0, cmap='RdBu', linewidths=1, annot=True,
            fmt=".2f", vmin=-1, vmax=1)

plt.title('Carte des corrélations', fontsize=15, fontweight="bold")

plt.show()
```



NETTOYAGE DES DONNEES

Lors de l'analyse exploratoire des données, nous avons remarqué la présence d'*outliers*. Ces valeurs aberrantes peuvent affecter la qualité d'un modèle de Machine Learning. Nous allons donc les retirer des données.

Avant de retirer les *outliers*, il faut d'abord les détecter. L'analyse exploratoire nous a permis de savoir que cette banque avait par exemple des client(e)s de plus de 100 ans, même de plus de 140 ans. Nous pouvons utiliser le 'bon sens' pour dire qu'il est très peu probable qu'une personne qui demande un crédit ait plus de 100 ans. Ainsi on supprimerait toutes les lignes où la variable *person_age* a des valeurs supérieures à 100.

Il existe aussi des méthodes statistiques, plus ou moins rigoureuses, pour détecter les *outliers*. Nous utiliserons l'une de ces méthodes à savoir l'inter-quartile.

❖ METHODE DE DETECTION DES VALEURS ABERRANTES BASEE SUR L'INTER-QUARTILE (IQR)

Selon cette méthode, une valeur est aberrante si elle est inférieure à $Q1 - 1.5 \text{IQR}$ ou supérieure à $Q3 + 1.5 \text{IQR}$. A partir de cette définition, créons une définition de détection des *outliers* :

```
def detection_outliers_by_iqr(y):  
  
    Q1, Q3 = np.quantile(y, 0.25), np.quantile(y, 0.75)  
  
    iqr = Q3 - Q1  
  
    borne_inf = Q1 - iqr * 1.5  
  
    borne_sup = Q3 + iqr * 1.5  
  
    return np.where((y > borne_sup) | (y < borne_inf))
```

❖ SUPPRESSION DES OUTLIERS

```
# Application de la fonction detection_outliers_by_iqr()  
  
indices_outliers_age = detection_outliers_by_iqr(df['person_age'])  
  
print(indices_outliers_age)  
  
print(type(indices_outliers_age))
```

```
(array([    81,    183,    575, ..., 32578, 32579, 32580]),)  
<class 'tuple'>
```

Nous avons un tuple contenant un tableau Numpy donnant les indices de lignes des valeurs aberrantes de la variable **person_age** détectées par la fonction **detection_outliers_by_iqr()** précédemment créée. Nous allons maintenant supprimer ces observations (lignes) :

```
df.drop(indices_outliers_age[0], inplace=True) # Le tableau numpy est le premier élément du tuple (donc d'indice 0)
```

```
df.shape
```

```
(31087, 12)
```

En supprimant les *outliers*, l'ensemble des données est passé de 32580 à 31087 observations.

❖ LES VALEURS MANQUANTES

```
# Nombre de valeurs manquantes par colonne
```

```
df.isna().sum()
```

person_age	0
person_income	0
person_home_ownership	0
person_emp_length	861
loan_intent	0
loan_grade	0
loan_amnt	0
loan_int_rate	2953
loan_status	0
loan_percent_income	0
cb_person_default_on_file	0
cb_person_cred_hist_length	0
dtype:	int64

Il existe deux principales techniques pour traiter les valeurs manquantes :

- Suppression des lignes contenant les valeurs manquantes (à seulement envisager lorsque les valeurs manquantes font moins de 5% des données) ;
- Remplacement des valeurs manquantes en faisant des imputations par la moyenne, la médiane, ...etc de la variable contenant ces valeurs manquantes. Il existe également d'autres méthodes d'imputation plus ou moins sophistiquées comme la méthode des k plus proches voisins (KNN).

Dans la pratique, il est conseillé de choisir une méthode puis de construire et d'évaluer le modèle. Puis de changer la méthode d'imputation et reconstruit le modèle ainsi de suite afin de finalement choisir une méthode qui donne les meilleurs résultats.

Commençons par une imputation par la médiane :

```
# Imputation par la moyenne

df['person_emp_length'] = df['person_emp_length'].fillna(df['person_emp_length'].median())

df['loan_int_rate'] = df['loan_int_rate'].fillna(df['loan_int_rate'].median())

# Vérification

df.isna().sum()
```

❖ TRANSFORMATION DES VARIABLES CATEGORIELLES EN VARIABLES NUMERIQUES

Dans Python, la plupart des algorithmes de Machine Learning ne fonctionnent pas directement avec les variables catégorielles. La dernière étape de préparation des données avant la modélisation sera donc consacrée à la transformation de ces variables qualitatives en variables numériques. Nous utiliserons la fonction [pandas.get_dummies\(\)](#)¹⁶

```
# dataframe des variables numériques

vars_num = df.select_dtypes(exclude=['object'])

# dataframe des variables non-numériques

vars_cat = df.select_dtypes(include=['object'])
```

```

# Transformation des variables non-numériques

vars_cat_dummies = pd.get_dummies(vars_cat)

# nouvelle dataframe

df_new = pd.concat([vars_num, vars_cat_dummies], axis = 1)

print(df_new.shape)

df_new.head()

(31087, 27)

```

L'opération de transformation des variables qualitatives en variables numérique a porté à 27 le nombre de variables.

MODELISATION

Nous voulons construire un modèle de Machine Learning capable de prédire si une personne sera en défaut de paiement (1) ou non (0) en fonction de certaines variables. Il s'agit donc d'un problème de Classification.

Commençons par scinder la dataframe en données d'entraînement (*train data*) et en données d'évaluation (*test data*) du modèle :

```

# Train/test data

X = df_new.drop('loan_status', axis = 1) # X est une dataframe
# constituée uniquement des variables indépendantes de df

y = df_new['loan_status'] # y est la variable dépendante (variable d'intérêt)

seed = 123

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.3, random_state = seed)

print(X_train.shape)

print(X_test.shape)
print(y_train.shape)

```

```
print(y_test.shape)
```

```
(21760, 26)
(9327, 26)
(21760,)
(9327,)
```

Dans la fonction *train_test_split()* :

- L'argument **stratify = y** est très important surtout quand on a un déséquilibre de classe dans les données. Il permet de créer des données d'entraînement et des données de test ayant les mêmes proportions d'étiquettes de classe que l'ensemble de données originel ;
- **test_size = 0.3** veut dire que 30% des données serviront à évaluer le modèle et donc que 70% des données serviront à entraîner l'algorithme ;
- **random_state** est défini afin de s'assurer de la reproductibilité de l'opération. En effet, en définissant l'argument *random_state*, on est assuré qu'à chaque fois que le code sera exécuté, ce seront les mêmes données d'entraînement et de test qui seront générées.

Construisons d'abord un modèle de base avec l'algorithme des forêts aléatoires (*RandomForestClassifier*) :

```
# Création du modèle

rf = RandomForestClassifier(random_state = seed)

# Entraînement du modèle

rf.fit(X_train, y_train)
```

EVALUATION DES PERFORMANCES DU MODELE

L'évaluation de la performance d'un modèle est une tâche qu'il faut effectuer avec soin. Il existe plusieurs métriques permettant d'évaluer cette performance. Le choix de la métrique doit être faite rigoureusement en fonction du problème business qu'on veut résoudre.

Commençons d'abord par calculer la précision globale du modèle de forêts aléatoires.

```
# Précision globale du modèle

print("Le score sur les données d'entraînement est :", rf.score(X_train, y_train))
```

```
print("Le score sur les données d'évaluation est :", rf.score(X_test, y_test))
```

```
Le score sur les données d'entraînement est : 0.9999540441176471  
Le score sur les données d'évaluation est : 0.9356706336442586
```

Ces valeurs nous indiquent que pratiquement 100% des statuts de prêts ont été correctement prédis au niveau des données d'entraînement et qu'environ 94% des statuts de prêts ont été correctement prédis au niveau des données de test. Cet écart entre score d'entraînement et score de test indique qu'il y a peut-être un problème de surapprentissage (*overfitting*).

Une autre fonction utilisée pour évaluer les modèles de classification est la fonction *classification_report()*. Cette fonction permet de générer d'un seul coup plusieurs métriques.

```
# Prédictions sur le test data  
  
y_pred = rf.predict(X_test)  
  
# Rapport de classification  
  
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.93	0.99	0.96	7290
1	0.97	0.73	0.83	2037
accuracy			0.94	9327
macro avg	0.95	0.86	0.90	9327
weighted avg	0.94	0.94	0.93	9327

Deux métriques sont vraiment utiles dans le rapport de classification : ***precision*** et ***recall***.

Le ***recall*** des prêts en défaut de paiement est égal à 0,73 ce qui signifie que 73% des véritables prêts en défaut de paiement ont été correctement prédis donc 27% des véritables prêts en défaut de paiement ont été mal prédis par le modèle comme étant des prêts qui ne sont pas en défaut de paiement. Ceci constitue un grand risque de perte d'argent pour la banque.

Imaginons que la banque ait 10000 prêts et que le montant de chaque prêt est 3000 Euros. Si tous ces 10000 prêts étaient en défaut de paiement, la perte d'argent serait estimée à $10000 \times 0.27 \times 3000$ soit 8 100 000 Euros. C'est une grosse perte d'argent pour la banque. Bien que le modèle donne une précision globale d'environ 94%, la banque perdrait énormément d'argent si elle implémentait ce modèle dans son système de crédit.

En revanche, le modèle prédit bien les prêts qui ne sont pas en défaut paiement. 99% des véritables prêts qui ne sont pas en défaut de paiement ont été correctement prédicts donc seulement 1% ont été mal prédicts par le modèle comme étant des prêts en défaut de paiement. Ceci constitue un petit risque de perte d'opportunité d'affaires pour la banque.

Notre modèle a deux problèmes majeurs : *overfitting* et mauvaise prédition des prêts en défaut de paiement. Ce dernier problème est certainement dû au déséquilibre de classe observé dans l'analyse exploratoire des données.

Il peut être très difficile de jongler entre la précision et le *recall* lorsqu'on tente d'augmenter la performance d'un modèle de classification. Heureusement, il existe une métrique qui combine les deux : [**F1-Score**](#)¹⁷.

Dans la suite, nous essayerons d'améliorer la performance de notre modèle en cherchant à augmenter le *F1-Score* des défauts de paiement.

DETERMINATION DE L'IMPORTANCE DES PREDICTEURS

Nous allons exploiter quand même le modèle de forêts aléatoires afin de déterminer l'importance de chaque variable indépendante dans la prédition du statut de prêt.

```
# Importance des variables
```

```
vars_imp = pd.Series(rf.feature_importances_, index = X.columns).sort_values(ascending = False)
```

```
vars_imp
```

loan_percent_income	0.227448
person_income	0.143809
loan_int_rate	0.109570
loan_amnt	0.076076
person_emp_length	0.060354
person_home_ownership_RENT	0.050525
loan_grade_D	0.048754
person_age	0.044365
cb_person_cred_hist_length	0.034412
person_home_ownership_MORTGAGE	0.028716
loan_intent_DEBTCONSOLIDATION	0.023212
loan_intent_MEDICAL	0.022241
loan_grade_C	0.019900
person_home_ownership_OWN	0.018639
loan_intent_HOMEIMPROVEMENT	0.014150
loan_grade_E	0.011795
loan_grade_A	0.010900
loan_intent_EDUCATION	0.010697
loan_intent_PERSONAL	0.008946
loan_intent_VENTURE	0.008513
cb_person_default_on_file_N	0.007469
cb_person_default_on_file_Y	0.006634
loan_grade_B	0.005134
loan_grade_F	0.004265
loan_grade_G	0.002675
person_home_ownership_OTHER	0.000804

```

# Visualisation des variables importantes

sns.barplot(x=vars_imp, y=vars_imp.index)

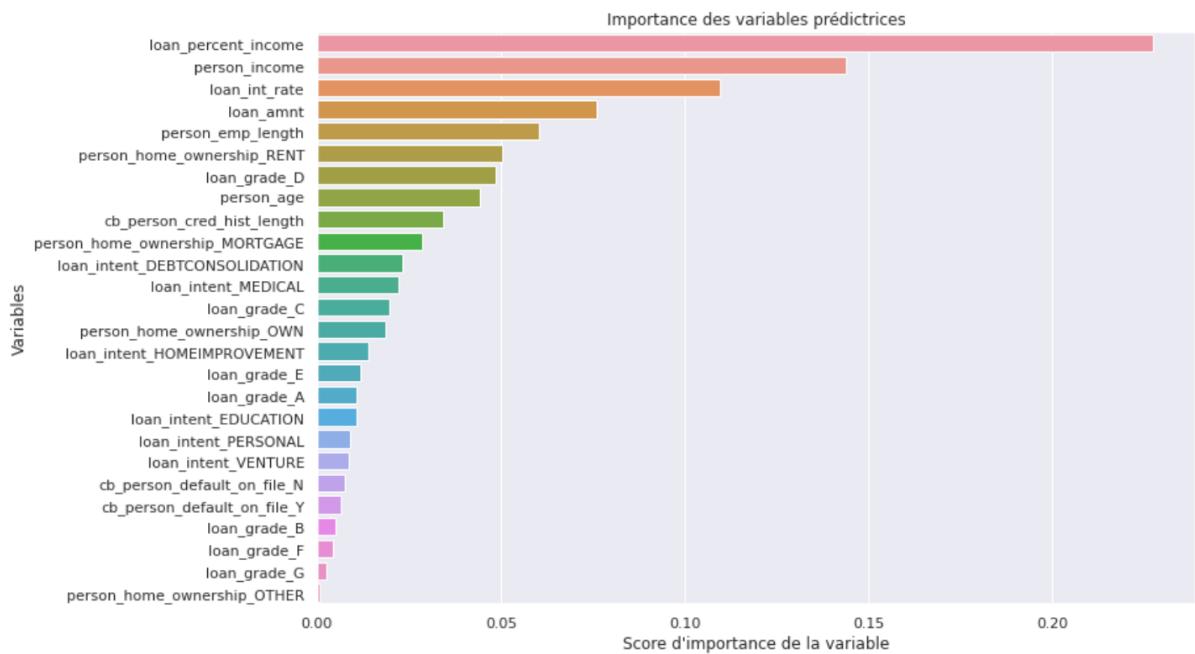
plt.xlabel("Score d'importance de la variable")

plt.ylabel('Variables')

plt.title("Importance des variables prédictrices")

plt.show()

```



(RE)CONSTRUCTION D'UN MODELE DE FORETS ALEATOIRES

Remodélisons les données en ne considérant cette fois-ci que les variables dont le score d'importance est supérieur à 0,010.

```
# variables dont le score d'importance est supérieur à 0,010 (Essayez plusieurs seuils et regardez les valeurs des métriques)
```

```
vars_selected = vars_imp[vars_imp > 0.010].index.to_list()

vars_selected
```

```

['loan_percent_income',
 'person_income',
 'loan_int_rate',
 'loan_amnt',
 'person_emp_length',
 'person_home_ownership_RENT',
 'loan_grade_D',
 'person_age',
 'cb_person_cred_hist_length',
 'person_home_ownership_MORTGAGE',
 'loan_intent_DEBTCONSOLIDATION',
 'loan_intent_MEDICAL',
 'loan_grade_C',
 'person_home_ownership_OWN',
 'loan_intent_HOMEIMPROVEMENT',
 'loan_grade_E',
 'loan_grade_A',
 'loan_intent_EDUCATION']

# Nouvelle division des données

X_train, X_test, y_train, y_test = train_test_split(X[vars_selected], y, stratify = y, test_size = 0.3, random_state = seed)

print(X_train.shape)
print(X_test.shape)

print(y_train.shape)

print(y_test.shape)

(21760, 18)
(9327, 18)
(21760,)
(9327,)

# Création du modèle

rf2 = RandomForestClassifier(random_state = seed)

# Entraînement du modèle

rf2.fit(X_train, y_train)

# Prédictions

y_pred2 = rf2.predict(X_test)

```

```
# Rapport de classification

print(classification_report(y_test, y_pred2))
```

	precision	recall	f1-score	support
0	0.93	0.99	0.96	7290
1	0.97	0.73	0.83	2037
accuracy			0.93	9327
macro avg	0.95	0.86	0.89	9327
weighted avg	0.94	0.93	0.93	9327

En ne sélectionnant que les variables indépendantes dont le score d'importance est supérieur à 0,010, les performances du modèle (voir rapport de classification ci-dessus) n'ont pas été amélioré. Cette opération nous a quand même permise de réduire la complexité du modèle car nous sommes passés de 26 à 18 variables indépendantes tout en gardant les mêmes performances qu'avec le premier modèle.

Dans ce qui suit, nous appliquerons la méthode de sous-échantillonnage afin de régler le problème de déséquilibre de classe.

METHODE DE SOUS-ECHANTILLONNAGE

Imaginons qu'on ait 100 prêts dont 20 sont en défaut de paiement et 80 ne le sont pas. La méthode de sous-échantillonnage consiste à échantillonner de façon aléatoire 20 prêts non-défauts et à les combiner avec les 20 prêts défauts de paiement pour former un ensemble de données équilibré.

```
# Concaténation de X_train, y_train

df_train = pd.concat([X_train.reset_index(drop = True),
                      y_train.reset_index(drop = True)],
                     axis = 1)

# Nombre de prêts dans chaque classe

n_nondefauts, n_defauts = df_train['loan_status'].value_counts()
```

```

# dataframe des prêts en défaut et dataframe des prêts en non
défauts

defauts_df = df_train[df_train['loan_status'] == 1]

nondefauts_df = df_train[df_train['loan_status'] == 0]

# Sous-
échantillonnage des non défauts pour qu'ils soient en même nom
bre que les défauts

nondefauts_df2 = nondefauts_df.sample(n_defauts)

# Concaténation

df_train2 = pd.concat([nondefauts_df2.reset_index(drop = True),
                       defauts_df.reset_index(drop = True)],
                      axis = 0)

# Vérification du nombre de prêts de chaque classe

df_train2['loan_status'].value_counts()

1    4752
0    4752
Name: loan_status, dtype: int64

```

Construisons maintenant un modèle de forêts aléatoires sur ce nouvel ensemble de données sous-échantillonné.

```

X = df_train2.drop('loan_status', axis = 1)

y = df_train2['loan_status']

X_train, X_test, y_train, y_test = train_test_split(X, y, stra
tify = y, test_size = 0.3, random_state = seed)

# Création du modèle
rf3 = RandomForestClassifier(random_state = seed)

# Entraînement du modèle
rf3.fit(X_train, y_train)

```

```

# Prédictions
y_pred3 = rf3.predict(X_test)

# Rapport de classification
print(classification_report(y_test, y_pred3))

```

	precision	recall	f1-score	support
0	0.81	0.91	0.86	1426
1	0.90	0.78	0.84	1426
accuracy			0.85	2852
macro avg	0.85	0.85	0.85	2852
weighted avg	0.85	0.85	0.85	2852

Ce rapport de classification indique le modèle rf3 prédit les défauts de paiement de manière beaucoup plus précise que les précédents modèles. En effet, 78% des prêts en défauts de paiement ont été correctement prédit contre 73% pour les deux modèles précédents.

CONCLUSION

Dans ce projet, nous avons traité un problème de classification. De plus, ce projet nous a permis d'ajouter plusieurs techniques utiles à notre boîte à outils de Machine Learning. Notre objectif était de construire un modèle qui prédit si le débiteur d'une banque sera en défaut de paiement ou pas. Un tel modèle est une véritable aide à la décision et permet aux banques de mieux gérer le risque de crédit.

Comment rechercher les hyperparamètres optimaux d'un modèle ? Ce sera l'objet principal du prochain projet.

PROJET 4 : CONSTRUCTION D'UN MODELE DE PREDICTION DU CANCER DU SEIN



INTRODUCTION

Selon l'[Organisation Mondiale de la Santé](#)¹⁸ (OMS), le cancer du sein est le cancer le plus fréquent chez les femmes dans les pays développés et les moins développés. En 2018, près de 2 millions de nouveaux cas de cancer du sein ont été diagnostiqués. Les médecins savent que le cancer du sein survient lorsque certaines cellules mammaires commencent à se développer anormalement. Ces cellules se divisent plus rapidement que les cellules saines et continuent de s'accumuler, formant une masse. Le problème avec ces cellules cancérigènes est qu'elles ne causent pas de douleur ou d'inconfort jusqu'à ce qu'elles se soient propagées aux tissus voisins. Dès lors, la détection d'un cancer du sein à un stade peu avancé de son développement peut permettre de soigner plus facilement mais aussi de limiter les séquelles liées à certains traitements.

Dans cette optique d'amélioration des diagnostics et des soins préventifs, l'Intelligence Artificielle peut aider les professionnels de Santé dans leur prise de décisions et permettre aux

patients d'être traités rapidement. Les hôpitaux collectent de plus en plus de données dont l'analyse est un défi et une opportunité. Plusieurs Organisations (Centres de recherche, Start-ups, Entreprises, etc.) à travers le monde travaillent sur le développement d'algorithme s d'apprentissage automatique qui sont entraînés par les données collectées. Les modèles ainsi créés sont ensuite utilisés pour prédire par exemple quand une insuffisance respiratoire, une septicémie ou d'autres problèmes tels que le cancer du sein sont susceptibles de se produire.

Dans ce projet, nous verrons un cas d'utilisation du Machine Learning dans la Santé. Notre objectif est de construire un modèle capable de prédire si une patiente, sur la base de ses données, sera atteinte d'un cancer du sein ou pas.

LIBRAIRIES

```
import pandas as pd

import numpy as np

from sklearn.datasets import load_breast_cancer

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import cross_val_score, GridSearchCV
```

DONNEES

Les données utilisées dans le cadre de ce projet, proviennent de [UCI Machine Learning Repository](#).¹⁹ Il s'agit de la base de données diagnostic du cancer du sein du Wisconsin aux USA. Ces données sont également disponibles dans le module [datasets.load_breast_cancer de sklearn](#).²⁰ Veuillez consulter la [page descriptive](#)¹⁹ de ces données pour une meilleure compréhension.

```
# Importation des données

df = load_breast_cancer()

# Affichage de df

print(df)
```

df est un dictionnaire contenant plusieurs paires clés-valeurs. La clé « data » a pour valeurs un tableau numpy des données des variables indépendantes. La clé « target » a pour valeur un tableau numpy représentant la variable cible. Cette variable cible a deux modalités : 'malignant' (1) et 'benign' (0). La clé « features_names » contient un tableau numpy des noms des variables indépendantes.

```
# Variable cible
```

```
y = df.target
```

```
print(y)
```

variables indépendantes

```
x = df.data
```

```
print(X)
```

```
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
```

x.shape

(569, 30)

MODELISATION

Nous sommes face à un problème de classification. Parmi les algorithmes de classification les plus utilisés en Data Science, il y a l'algorithme des k plus proches voisins ([K Nearest Neighbor](#)²¹ ou **KNN** en Anglais). KNN est un algorithme d'apprentissage automatique supervisé particulièrement simple et facile à implémenter.

Commençons d'abord par l'initialisation de l'algorithme KNN avec le module [sklearn.neighbors.KNeighborsClassifier](#)²² et ses paramètres par défaut :

```
# Initialisation de KNN avec ses hyperparamètres par défaut
```

```
knn = KNeighborsClassifier()
```

Réalisons ensuite une validation croisée avec 5 plis :

```
# 5-fold cross-validation
```

```
cv_scores = cross_val_score(knn, X, y, cv = 5, scoring = 'precision')
```

```
# Affichage des scores
```

```
print(cv_scores)
```

```
[0.8625 0.93243243 0.94520548 0.94594595 0.95652174]
```

Calculons le score moyen :

```
# Score moyen arrondi à deux chiffres après la virgule
```

```
round(np.mean(cv_scores), 2)
```

Le score moyen est égal à 0,93.

GRID SEARCH

Pour effectuer notre validation croisée, nous avons considéré les paramètres par défaut de l'algorithme de KNN : ce sont les hyperparamètres. Les hyperparamètres sont définis avant le début du processus de modélisation. Il ne faut pas les confondre avec les paramètres d'un modèle obtenus après l'entraînement de l'algorithme par des données. Pour l'algorithme de classification KNN dans Scikit-Learn, les hyperparamètres sont :

```

# Affichage de knn

print(knn)

KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
metric_params=None, n_jobs=None, n_neighbors=5,
p=2,
weights='uniform')

```

Veuillez consulter cette [documentation](#)²² pour les définitions de ces algorithmes. Par exemple *n_neighbors* indique le nombre de points voisins que l'algorithme considère dans ses calculs. Comme vous le voyez bien, des valeurs par défaut sont définies pour chaque hyperparamètre.

Comment pouvons-nous être sûr que ces valeurs par défaut sont optimales pour notre modèle ? Quelle est la meilleure combinaison des hyperparamètres qui donne la meilleure performance du modèle ? Imaginons que nous voulions essayer plusieurs valeurs pour les hyperparamètres *algorithm*, *leaf_size*, et *n_neighbors*. Il faudra donc essayer chaque combinaison de ces valeurs, construire le modèle et l'évaluer par une validation croisée pour finalement choisir l'hyperparamétrage qui donne la meilleure métrique d'évaluation. Ce processus peut être effectué automatiquement avec une recherche de grille (*Grid Search*). Voici comment nous l'implémentons dans Python :

```

# Dictionnaire des hyperparamètres avec leurs valeurs

grid = {'n_neighbors':[1,3,5,7,9,10],
        'weights': ['uniform', 'distance'],
        'algorithm':['auto', 'ball_tree', 'kd_tree', 'brute'],
        'metric':['euclidean', 'manhattan'],
        'leaf_size':[10,20,30,40,50]
       }

# Estimateur

model = GridSearchCV(estimator = KNeighborsClassifier(),
                      param_grid = grid,
                      scoring = 'precision',
                      cv = 5)

```

Nous avons d'abord défini un dictionnaire avec comme clés les hyperparamètres et leurs valeurs que nous voulons essayer (veiller à bien écrire le nom de chaque hyperparamètre). Nous avons

ensuite défini un objet [GridSearchCV](#)²³ qui nous permettra de rechercher le meilleur hyperparamétrage. Afin de mieux comprendre, calculons le nombre total de modèles qui seront construit. Chaque combinaison de valeur sera essayée. Pour une combinaison, 5 modèles seront construits (`cv = 5`) afin de choisir celui qui donne la meilleure métrique (ici ‘precision’). Pour chaque valeur d’un hyperparamètre, nous testons chaque valeur de chaque autre hyperparamètre : c’est ce qui forme une combinaison. Pour trouver le nombre de combinaisons possibles, il faudra donc multiplier les longueurs des listes qui définissent les valeurs des hyperparamètres. Alors, dans notre cas, le nombre de combinaisons possibles est égal à $6 \times 2 \times 4 \times 2 \times 5$ soit 480 combinaisons. 5 modèles sont créés pour chaque combinaison. Alors le nombre total de modèles créés est égal à 480×5 soit 2400 modèles.

```
# Recherche du meilleur modèle
```

```
model.fit(X, y)
```

L’objet GridSearchCV a un attribut « `cv_results_` » qui stocke dans un dictionnaire les résultats de la recherche. Nous allons afficher ces résultats dans un format plus lisible qu’un dictionnaire : une dataframe.

```
# Conversion du dictionnaire des résultats en une dataframe
results = pd.DataFrame(model.cv_results_)

# 'params' et 'mean_test_score' sont les attributs qui nous intéressent

best = results[['params', 'mean_test_score']].sort_values(by =
    'mean_test_score', ascending = False).head()

best
```

	params	mean_test_score
142	{'algorithm': 'ball_tree', 'leaf_size': 10, 'm...	0.932089
406	{'algorithm': 'brute', 'leaf_size': 20, 'metri...	0.932089
286	{'algorithm': 'kd_tree', 'leaf_size': 20, 'met...	0.932089
262	{'algorithm': 'kd_tree', 'leaf_size': 10, 'met...	0.932089
310	{'algorithm': 'kd_tree', 'leaf_size': 30, 'met...	0.932089

Les 5 meilleurs modèles ont le même score avec des hyperparamètres différents.

```
# Meilleur hyperparamétrage  
  
best['params'].iloc[0]  
  
# Meilleur score  
  
best['mean_test_score'].iloc[0]
```

Nous avons déterminé les valeurs des hyperparamètres ayant donné le meilleur score. Malheureusement, ils n'ont pas amélioré le score trouvé avec les hyperparamètres par défaut qui était aussi égal à 0,93. Nous pouvons utiliser notre objet GridSearchCV (« model ») directement comme modèle de classification et l'utiliser pour effectuer des prédictions. Nous n'avons donc pas besoin d'entraîner à nouveau un algorithme de KNN avec les meilleurs hyperparamètres trouvés.

CONCLUSION

Dans ce projet, nous avons construit un modèle de de prédition du cancer de sein. De plus, nous avons appris comment rechercher les hyperparamètres optimaux du modèle. La méthode de *Grid Search CV* telle que décrite ici, peut aussi être appliquée pour n'importe quel projet de machine Learning aussi bien pour des tâches de classification que pour des tâches de régression et même en apprentissage non supervisé. Si cette méthode a l'avantage d'être un processus automatisé, facile à comprendre et à implémenter, il n'en demeure pas moins qu'elle a quelques inconvénients. L'inconvénient majeur de la méthode *Grid Search* est qu'elle est très coûteuse en calcul, c'est-à-dire que lorsque le nombre d'hyperparamètres à essayer augmente considérablement et que la dataframe est volumineuse, les temps de traitement peuvent être très lents. De plus, lorsque vous définissez votre grille, vous pouvez omettre par inadvertance un hyperparamétrage qui serait en fait optimal. S'il n'est pas spécifié dans votre grille, il ne sera jamais essayé !

Pour surmonter ces inconvénients, nous examinerons la recherche aléatoire (***Random Search***) dans le projet suivant.

PROJET 5 : PREDICTION DES PRIX DES MAISONS

INTRODUCTION

Nous plongeons à nouveau dans le secteur de l'immobilier. Notre objectif ici est de construire un modèle de régression capable de prédire les prix des maisons dans la ville de Ames de l'Etat de l'Iowa aux USA. Nous utiliserons un algorithme d'arbre de décision puis nous rechercherons les hyperparamètres optimaux en utilisant la méthode de recherche aléatoire.

LIBRAIRIES

```
import pandas as pd

import numpy as np

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean_squared_error as mse

from sklearn.model_selection import train_test_split, RandomizedSearchCV
```

DONNEES

Les données utilisées dans ce projet proviennent de [Kaggle](#)²⁴. L'ensemble originel comporte 81 variables. Pour aller vite à l'essentiel, nous avons procédé à un prétraitement de ces données ce qui nous a réduit la dimension de l'ensemble d'origine à 29 variables.

```
# Importation des données

df = pd.read_csv('https://raw.githubusercontent.com/JosueAfouda/Ames-House-Prices/master/house_price_preprocessed', index_col = 0)

df.head()

# Structure des données

df.info()
```

Il n'y a pas de valeurs manquantes.

DIVISION DES DONNEES

Nous utiliserons 25% des données comme données de test.

```
# Dataframe des variables indépendantes

X = df.drop('SalePrice', axis = 1)

# Variable cible

y = df['SalePrice']

# Train/Test data

test_size = 0.25

seed = 111

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = test_size, random_state = seed)

# Dimension du train et du test data

print(X_train.shape)

print(X_test.shape)

(1095, 28)
(365, 28)
```

MODELISATION

L'algorithme d'arbre de décision pour la régression est très utilisé en Data Science. L'un de ses avantages est qu'il ne requiert aucune transformation dans les variables si nous avons affaire à des données non linéaires. Pour l'implémenter dans Python, nous utiliserons le module [*sklearn.tree.DecisionTreeRegressor*](#)²⁵.

```
# Création d'un modèle d'arbre de décision

dt = DecisionTreeRegressor(random_state = seed)

# Entraînement du modèle

dt.fit(X_train, y_train)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort='deprecated',
                     random_state=111, splitter='best')
```

Cet algorithme a plusieurs hyperparamètres dont les plus importants et les plus utilisés sont : *criterion*, *max_depth*, *min_samples_split*, *min_samples_leaf* et *max_features*. Vous pouvez consulter la définition de ces hyperparamètres sur cette [page de Scikit-Learn](#)²⁶.

```
# Prédictions
```

```
y_pred = dt.predict(X_test)

# Calcul de l'erreur quadratique moyenne

mse_dt = mse(y_test, y_pred)

print(mse_dt)
```

```
2013265454.4273973
```

Avec la méthode de recherche aléatoire, nous essayerons de trouver un meilleur modèle. Autrement dit, nous cherchons un modèle dont l'erreur quadratique moyenne sera inférieure à 2013265454,4273973.

La méthode *Random Search* n'essaie pas toutes les combinaisons possibles dans la grille des hyperparamètres comme c'était le cas avec la méthode *Grid Search*. On essaie plutôt de manière aléatoire un certain nombre de combinaisons. Plusieurs études sur cette méthode ont montré qu'avec relativement peu d'essais, il est probable (probabilité relativement élevée) de se rapprocher du meilleur score. L'implémentation de cette méthode dans Python se fait avec le module [RandomizedSearchCV](#)²⁷.

```
# Dictionnaire des hyperparamètres
```

```
param_dists = {'criterion':['mse', 'friedman_mse', 'mae'],
               'splitter':['best', 'random'],
               'max_depth':[3,5,7,None],
               'min_samples_split':np.arange(0.1, 1.1, 0.1),
               'min_samples_leaf':list(range(1,21)),
               'max_features':['auto', 'sqrt', 'log2', None]}
```

```
# Estimateur (objet RandomizedSearchCV)

model = RandomizedSearchCV(estimator = DecisionTreeRegressor(r
andom_state = seed),
                           param_distributions = param_dists,
                           n_iter = 200,
                           scoring = 'neg_mean_squared_error',
                           cv = 5,
                           random_state = seed)
```

En spécifiant $n_iter = 200$, cela veut dire que nous voulons essayer de manière aléatoire 200 combinaisons. Comme score, nous avons spécifié l'erreur quadratique moyenne (rigoureusement l'opposé de l'erreur quadratique moyenne).

```
# Démarrage de la recherche

model.fit(X, y)

# Les 5 Meilleurs hyperparamètres avec leurs scores

results = pd.DataFrame(model.cv_results_)

best = results.loc[:, ['params', 'mean_test_score']].sort_values
('mean_test_score', ascending = False).head()

best
```

	params	mean_test_score
147	{'splitter': 'best', 'min_samples_split': 0.1,...}	-1.714097e+09
141	{'splitter': 'best', 'min_samples_split': 0.1,...}	-1.843539e+09
114	{'splitter': 'best', 'min_samples_split': 0.1,...}	-1.843579e+09
145	{'splitter': 'best', 'min_samples_split': 0.1,...}	-1.962182e+09
134	{'splitter': 'best', 'min_samples_split': 0.1,...}	-2.063297e+09

Le signe « - » ne doit en aucun cas nous tromper ici (Faites comme s'il n'était pas là). On cherche bel et bien la plus petite valeur absolue de $mean_test_score$.

```
# Meilleur hyperparamétrage
```

```
best['params'].iloc[0]
```

```

{'criterion': 'mae',
 'max_depth': None,
 'max_features': 'auto',
 'min_samples_leaf': 10,
 'min_samples_split': 0.1,
 'splitter': 'best'}

# Comparaison entre le meilleur score trouvé et mse_dt

np.abs(best['mean_test_score'].iloc[0]) < mse_dt

```

True

La recherche du meilleur hyperparamétrage a permis de trouver un modèle qui est de loin meilleur à celui construit avec les hyperparamètres par défaut.

```
mse_dt - np.abs(best['mean_test_score'].iloc[0])
```

```
299168082.90102744
```

CONCLUSION

L'implémentation de la recherche aléatoire pour améliorer la qualité d'un modèle est très simple. Etant donné, que c'est l'utilisateur lui-même qui définit le nombre de combinaisons aléatoires à essayer (paramètre n_iter), Il est possible d'étendre la plage de votre recherche d'hyperparamètres au-delà de ce qui serait pratique avec une recherche de grille. Par ailleurs, lorsque vous avez un grand volume de données et un algorithme avec beaucoup d'hyperparamètres, il est conseillé d'utiliser une recherche aléatoire à cause des temps de calcul à moins que vous disposiez d'un ordinateur hyper puissant 😊 Dans ce cas, vous pouvez toujours essayer la méthode *Grid Search*.

PROJET 6 : MODELE DE PREDICTION DES INTENTIONS D'ACHATS DES UTILISATEURS D'UNE BOUTIQUE EN LIGNE

CONTEXTE

Une boutique en ligne dispose des données sur chacun des utilisateurs de son site web. Parmi les différentes variables, il y en a une qui indique si l'utilisateur a terminé sa session par un achat ou pas.

Un des objectifs de l'analyse de ce jeu de données pourrait être de construire un modèle qui prédit si un utilisateur donné achètera des produits sur le site web : c'est un cas d'étude de prédition des intentions d'achats. Le bénéfice de cette étude est qu'avec un tel modèle, on peut identifier les variables importantes qui peuvent prédire l'intention d'achat d'un utilisateur sur le site web. Les résultats permettront ainsi aux décideurs de prendre des mesures concrètes et efficaces afin d'augmenter les achats sur ce site web.

LIBRAIRIES

```
#Librairies fondamentales

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
```

DONNEES

```
#Données

df = pd.read_csv('https://github.com/JosueAfouda/Shoppers-
Intention/raw/master/online_shoppers_intention.csv')

df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Administrative    12330 non-null   int64  
 1   Administrative_Duration 12330 non-null   float64 
 2   Informational     12330 non-null   int64  
 3   Informational_Duration 12330 non-null   float64 
 4   ProductRelated    12330 non-null   int64  
 5   ProductRelated_Duration 12330 non-null   float64 
 6   BounceRates       12330 non-null   float64 
 7   ExitRates         12330 non-null   float64 
 8   PageValues        12330 non-null   float64 
 9   SpecialDay        12330 non-null   float64 
 10  Month            12330 non-null   object  
 11  OperatingSystems  12330 non-null   int64  
 12  Browser          12330 non-null   int64  
 13  Region           12330 non-null   int64  
 14  TrafficType      12330 non-null   int64  
 15  VisitorType      12330 non-null   object  
 16  Weekend          12330 non-null   bool   
 17  Revenue          12330 non-null   bool  
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB

```

La variable Revenue est celle qui indique si l'utilisateur a réalisé un achat (*True*) ou pas (*False*).

Les modèles de Machine Learning utilisent uniquement des données numériques donc nous allons devoir transformer les variables catégorielles en variables numériques.

Créons d'abord une dataframe des variables indépendantes et une dataframe de la variable cible et sauvegardons-les.

```

# Variables indépendantes

features = df.drop('Revenue', axis=1)

# Variable dépendante

target = df['Revenue']

# Sauvegarde de données au format csv

features.to_csv('feats.csv', index=False)

target.to_csv('target.csv', header='Revenue', index=False)

```

Générons les statistiques descriptives :

```
# Résumé statistique (variables numériques)
```

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Administrative	12330.0	2.315166	3.321784	0.0	0.000000	1.000000	4.000000	27.000000
Administrative_Duration	12330.0	80.818611	176.779107	0.0	0.000000	7.500000	93.256250	3398.750000
Informational	12330.0	0.503569	1.270156	0.0	0.000000	0.000000	0.000000	24.000000
Informational_Duration	12330.0	34.472398	140.749294	0.0	0.000000	0.000000	0.000000	2549.375000
ProductRelated	12330.0	31.731468	44.475503	0.0	7.000000	18.000000	38.000000	705.000000
ProductRelated_Duration	12330.0	1194.746220	1913.669288	0.0	184.137500	598.936905	1464.157213	63973.522230
BounceRates	12330.0	0.022191	0.048488	0.0	0.000000	0.003112	0.016813	0.200000
ExitRates	12330.0	0.043073	0.048597	0.0	0.014286	0.025156	0.050000	0.200000
PageValues	12330.0	5.889258	18.568437	0.0	0.000000	0.000000	0.000000	361.763742
SpecialDay	12330.0	0.061427	0.198917	0.0	0.000000	0.000000	0.000000	1.000000
OperatingSystems	12330.0	2.124006	0.911325	1.0	2.000000	2.000000	3.000000	8.000000
Browser	12330.0	2.357097	1.717277	1.0	2.000000	2.000000	2.000000	13.000000
Region	12330.0	3.147364	2.401591	1.0	1.000000	3.000000	4.000000	9.000000
TrafficType	12330.0	4.069586	4.025169	1.0	2.000000	2.000000	4.000000	20.000000

ENCODAGE DES VARIABLES CATEGORIELLES

Les variables à transformer numériquement sont : *Weekend* (binaire), *VisitorType*, *Month* et *Revenue* (binaire).

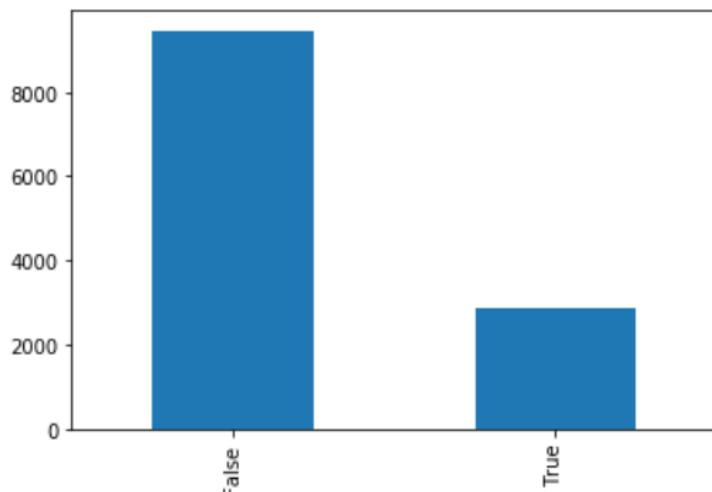
```
# Lecture de la dataframe des variables indépendantes
```

```
data = pd.read_csv('feats.csv')
```

Voyons la répartition de la variable binaire *Weekend* :

```
data['Weekend'].value_counts().plot(kind='bar')
```

```
plt.show()
```



Les utilisateurs achètent beaucoup plus en semaine qu'en weekend.

```
# Encodage
```

```
data['is_weekend'] = data['Weekend'].apply(lambda row: 1 if row == True else 0)

# Vérifions que 1 a été mis pour True et que 0 a été mis pour False

data[['Weekend', 'is_weekend']].tail()
```

	Weekend	is_weekend
12325	True	1
12326	True	1
12327	True	1
12328	False	0
12329	True	1

L'encodage est réalisé avec succès. On peut maintenant supprimer la colonne *Weekend* :

```
data.drop('Weekend', axis=1, inplace=True)
```

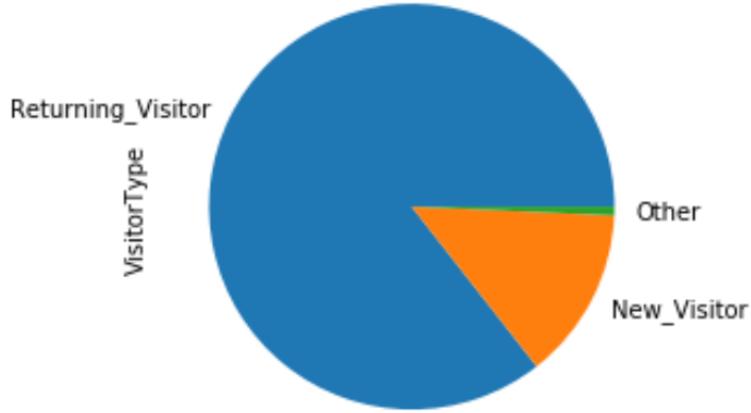
Passons maintenant à la transformation de la variable *VisitorType*. Cette variable n'est pas binaire donc nous ne pouvons appliquer la méthode précédente pour la transformer numériquement. Nous allons plutôt lui appliquer la fonction [pd.get_dummies\(\)](#)¹⁶.

Commençons par visualiser la répartition des modalités de cette variable.

```
# Modalités de la variable 'VisitorType'

data['VisitorType'].value_counts().plot(kind='pie')

plt.show()
```



Le site web attire très faiblement de nouveaux visiteurs.

```
# Application de la méthode pd.get_dummies()

visitor_type_dummies = pd.get_dummies(data['VisitorType'], prefix='VisitorType')

pd.concat([data['VisitorType'], visitor_type_dummies], axis=1)
.tail()
```

	VisitorType	VisitorType_New_Visitor	VisitorType_Other	VisitorType_Returning_Visitor
12325	Returning_Visitor	0	0	1
12326	Returning_Visitor	0	0	1
12327	Returning_Visitor	0	0	1
12328	Returning_Visitor	0	0	1
12329	New_Visitor	1	0	0

Attention : Quand on applique la méthode `pd.get_dummies()` sur une colonne, il apparaît une redondance de l'information. Par exemple, si on a trois modalités dans la variable et que pour une ligne donnée, deux des trois colonnes créées prennent la valeur 0 alors la troisième colonne restante a forcément pour valeur 1. Donc il faut supprimer l'une des colonnes créées afin d'éliminer la redondance de l'information. On élimine la colonne qui a la plus faible occurrence.

La modalité de plus faible occurrence dans la variable `VisitorType` est : `Other`.

```
# Suppression de 'VisitorType_Other'

visitor_type_dummies.drop('VisitorType_Other', axis=1, inplace=True)
```

On colle (concaténation) la dataframe créée avec la dataframe originelle :

```
# Concaténation

data = pd.concat([data, visitor_type_dummies], axis=1)

# Suppression de l'ancienne variable 'VisitorType'

data.drop('VisitorType', axis=1, inplace=True)
```

Appliquons le même processus à la colonne *Month* :

```
data['Month'].value_counts().plot(kind='bar')

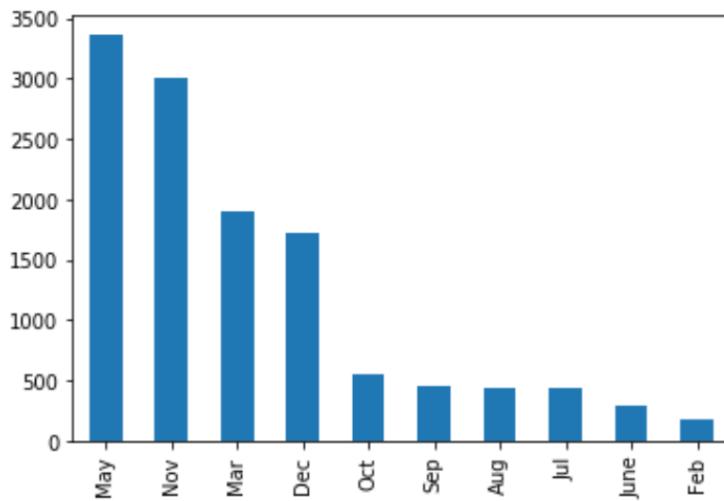
month_dummies = pd.get_dummies(data['Month'], prefix='Month')

pd.concat([data['Month'], month_dummies], axis=1).tail()

month_dummies.drop('Month_Feb', axis=1, inplace=True)

data = pd.concat([data, month_dummies], axis=1)

data.drop('Month', axis=1, inplace=True)
```



Passons maintenant à la transformation de la colonne *Revenue*.

```
target = pd.read_csv('target.csv')

# Encodage de 'Revenue'

target['Revenue'] = target['Revenue'].apply(lambda row: 1 if row==True else 0)
```

```
#Sauvegarde
```

```
target.to_csv('target_clean.csv', index=False)
```

Les variables catégorielles ont été transformé avec succès en variables numériques.

TYPES APPROPRIES ?

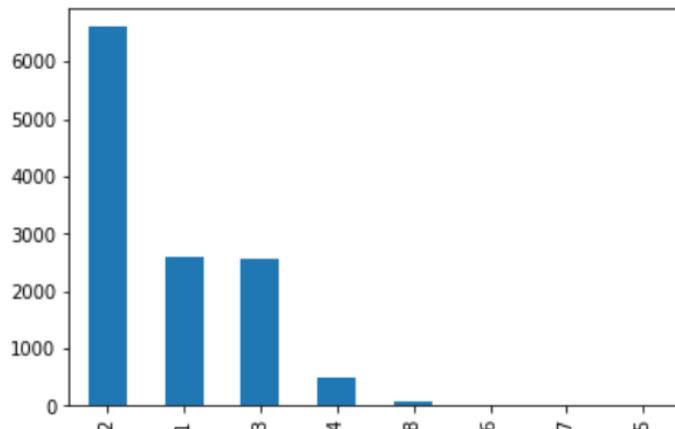
Il faut toujours vérifier que les données de chaque colonne soient bien stockées dans le format approprié. Si des colonnes, bien qu'étant numériques, ne sont pas dans le type approprié, alors il faudra leur appliquer aussi la fonction pd.get_dummies().

Dans notre cas ici, il s'agit des colonnes *OperatingSystems*, *Browser*, *TrafficType* et *Region*. L'identification des colonnes qui ne sont pas stockées dans le format approprié est conditionnée par une bonne compréhension du jeu de données et du Business auquel vous faites face. Le Data Analyst ou Data Scientist doit donc être en parfaite collaboration avec les experts du domaine.

Passons maintenant à l'encodage des variables citées ci-dessus.

```
# OperatingSystems
```

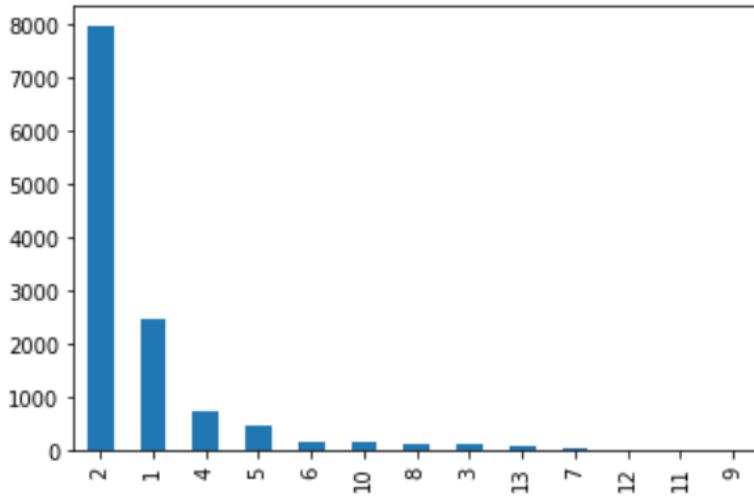
```
data['OperatingSystems'].value_counts().plot(kind='bar')
```



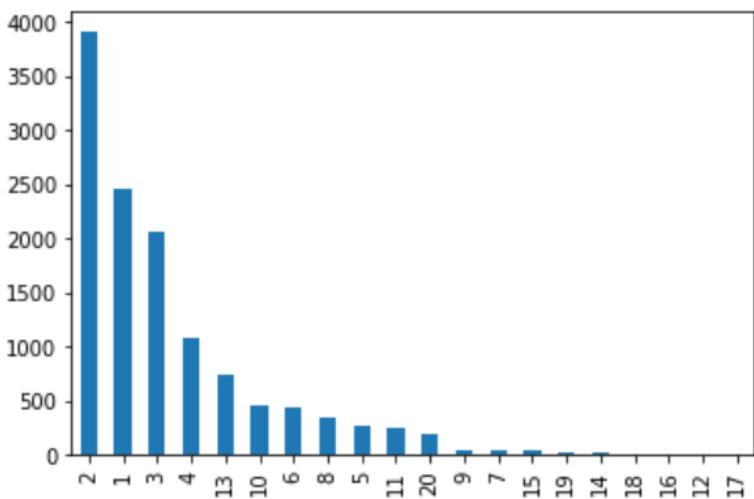
```
operatings_systems_dummies = pd.get_dummies(data['OperatingSystems'], prefix='OperatingSystems')
operatings_systems_dummies.drop('OperatingSystems_5', axis=1, inplace=True)
```

```
data = pd.concat([data, operatings_systems_dummies], axis=1)
```

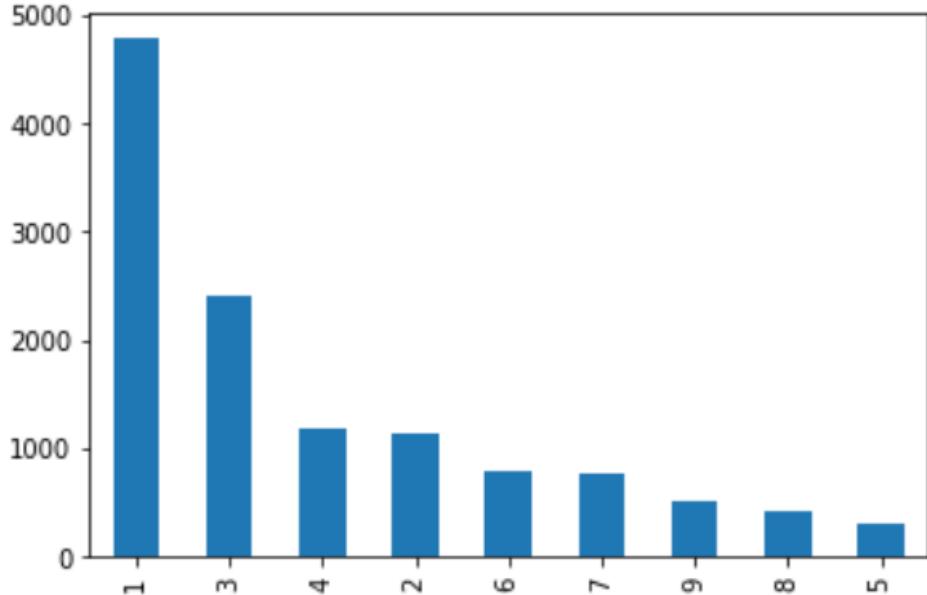
```
# Browser  
  
data['Browser'].value_counts().plot(kind='bar')
```



```
browser_dummies = pd.get_dummies(data['Browser'], prefix='Browser')  
  
browser_dummies.drop('Browser_9', axis=1, inplace=True)  
  
data = pd.concat([data, browser_dummies], axis=1)  
  
# TrafficType  
  
data['TrafficType'].value_counts().plot(kind='bar')
```



```
# Region  
  
data['Region'].value_counts().plot(kind='bar')
```



```
region_dummies = pd.get_dummies(data['Region'], prefix='Region')  
  
region_dummies.drop('Region_5', axis=1, inplace=True)  
  
data = pd.concat([data, region_dummies], axis=1)
```

Toutes nos variables sont à présent bien codées et dans le format approprié. Avant de passer à la modélisation, faisons une sauvegarde de nos données.

```
# Sauvegarde des variables indépendantes  
  
data.to_csv('feats_clean.csv', index=False)
```

MODELISATION

Commençons par importer les données :

```
features_clean = pd.read_csv('feats_clean.csv')  
  
target_clean = pd.read_csv('target_clean.csv')
```

Procémons à une division des données en données d'entraînement (80%) du modèle et données pour l'évaluation du modèle (20%).

```
from sklearn.model_selection import train_test_split

test_size = 0.2

random_state = 42

X_train, X_test, y_train, y_test = train_test_split(features_clean, target_clean, test_size = test_size, stratify = target_clean, random_state = random_state)

# Affichage de la dimension des données d'entraînement et de test

print(X_train.shape)

print(y_train.shape)

print(X_test.shape)

print(y_test.shape)

(9864, 72)
(9864, 1)
(2466, 72)
(2466, 1)
```

Construisons un modèle de régression logistique pour prédire l'intention d'achat d'un utilisateur.

```
# Importation du module LogisticRegression (écrire paragraphe)

from sklearn.linear_model import LogisticRegression

# Création du modèle

model = LogisticRegression(solver='liblinear',
                           max_iter = 500,
                           C = 10,
                           penalty = 'l1',
                           random_state = random_state)
```

```
#Entraînement du modèle

model.fit(X_train, y_train['Revenue'])

# Prédictions sur le test set

yhat = model.predict(X_test)
```

Passons maintenant à l'évaluation du modèle. Il y a plusieurs paramètres qu'on peut calculer pour évaluer un modèle de Machine Learning. Veuillez consulter la page de [sklearn.metrics](#)²⁸ pour de plus amples informations sur ces paramètres.

```
print("Score d'entraînement du modèle :", model.score(X_train,
y_train))
```

Score d'entraînement du modèle : 0.8860502838605029

Est-ce que le modèle est capable de bien se généraliser ? Pour répondre à cette question, nous devons évaluer le modèle avec les données de Test.

```
from sklearn.metrics import accuracy_score, precision_recall_f
score_support, classification_report
```

```
#Evaluation du modèle
```

```
accuracy = accuracy_score(yhat, y_test)
```

```
print(f'La précision globale du modèle est {accuracy*100:.2f}%
')
```

La précision globale du modèle est 88.12%

Au vu des scores d'entraînement et d'évaluation, nous pouvons conclure qu'il n'y a pas un problème de surajustement au niveau du modèle. Le modèle est donc capable de bien se généraliser.

PROJET 7 : CLUSTERING AVEC KMEANS

INTRODUCTION

Comprenez le mot *clustering* comme regroupement en classes. Le Clustering est une technique d'apprentissage automatique non-supervisé dont l'objectif est de découvrir les groupes sous-jacents (ou "clusters") dans un ensemble de données. Ici on ne dispose pas de données étiquetées comme c'est le cas dans l'apprentissage automatique supervisé. L'algorithme regroupe les objets similaires en termes de leurs caractéristiques. Dans chaque groupe, on a donc des observations similaires mais les groupes sont distincts les uns des autres.

Cette technique est très utilisée dans le domaine du Marketing pour réaliser la segmentation de clientèle. La Segmentation est une pratique de partitionnement des clients en groupes d'individus ayant les mêmes caractéristiques (On utilise souvent les données démographiques des clients car elles sont plus faciles à collecter). Cette stratégie permet de comprendre le comportement des clients, sur la base de leurs données historiques, et donc d'utiliser efficacement les ressources allouées au Marketing.

Dans cette section, vous apprendrez à utiliser l'algorithme *KMeans*²⁹ qui est le plus populaire pour les tâches de clustering.

INITIATION A KMEANS

Le but de cette section est d'explorer l'algorithme de partitionnement automatique *KMeans* sur des jeux de données synthétiques 2D.

Cet exercice est inspiré en partie de cette [page de la documentation scikit-learn](#).³⁰ La théorie mathématique derrière l'algorithme *KMeans* est décrite dans cette [page](#)³¹.

```
# Données synthétiques 2D

from sklearn import datasets

import numpy as np

n_samples = 500

# blobs isotropes
random_data = datasets.make_blobs(n_samples=n_samples, random_
state=8)
```

Visualisons le jeu de données à partitionner à l'aide d'un graphique de nuage de points :

```
X,y = random_data

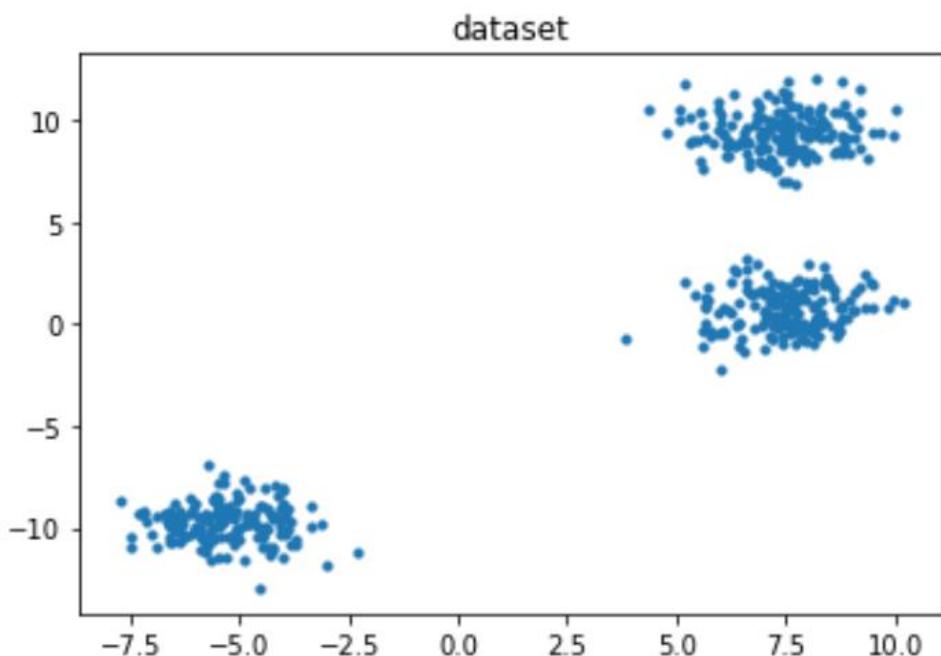
import matplotlib.pyplot as plt

plt.figure()

plt.scatter(X[:, 0], X[:, 1], s=10)

plt.title('dataset')

plt.show()
```



Le nuage de points ci-dessus nous suggère trois groupes distincts. Appliquons alors l'algorithme *KMeans* avec 3 clusters.

EXECUTION DE KMEANS

```
# Importation du module KMeans

from sklearn.cluster import KMeans

# Création d'un modèle KMeans avec 3 clusters
model = KMeans(n_clusters=3)
```

```
# Entraînement du modèle
```

```
model.fit(X)
```

```
# Clusters (labels)
```

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

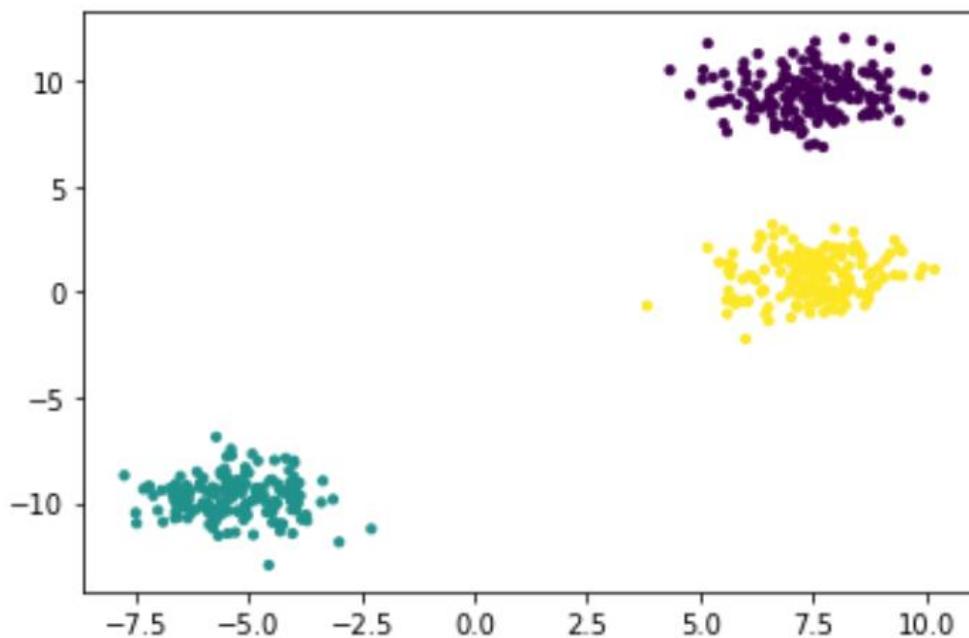
```
print(labels)
```

```
[1 0 0 2 2 2 0 2 0 0 0 2 2 1 2 1 2 1 1 2 0 1 0 0 1 2 2 1 0 0 2 1 1 0 2 2 0  
1 1 0 1 0 2 0 2 2 0 1 0 0 2 0 1 0 1 0 2 0 1 0 0 2 0 1 2 1 2 2 2 2 0 2 0  
2 2 2 1 2 2 2 0 0 0 0 2 1 2 1 2 2 0 0 2 2 1 1 2 0 0 1 2 1 0 2 0 1 0 2 0 0  
0 2 0 2 0 1 0 0 2 0 1 2 0 2 1 0 0 1 2 2 1 2 2 2 0 1 1 0 1 0 2 1 1 1 2 0 1  
1 0 1 2 1 1 1 0 2 2 0 0 0 1 2 1 1 2 0 0 2 1 1 1 0 1 1 1 1 0 0 2 0 2 1 1 2  
0 1 2 1 2 2 2 2 1 1 2 2 0 0 2 0 1 1 2 2 1 0 1 1 0 1 0 0 2 0 0 1 1 0 2 1 1  
1 0 2 1 2 2 1 2 1 0 2 0 0 0 2 0 1 0 1 1 2 1 2 2 2 0 0 0 1 2 1 1 0 0 0 2 2  
0 2 0 2 1 1 0 0 1 1 2 0 1 1 0 0 2 1 0 0 2 2 2 2 0 1 2 1 2 0 0 1 2 0 1 0 2  
1 2 2 1 0 0 1 2 2 1 2 2 2 2 0 1 1 2 1 1 2 1 1 0 0 1 0 2 0 2 0 2 1 0 2 2 2  
0 1 2 2 0 2 1 0 1 1 0 2 1 2 0 0 1 2 2 0 2 2 2 2 0 0 2 1 0 0 2 2 0 0 1 2 2  
0 2 2 0 2 2 0 0 0 2 2 2 1 1 2 2 1 0 0 1 0 1 1 1 0 1 0 0 2 1 1 1 1 2 0 2 2  
1 0 1 2 2 0 1 1 1 0 1 2 1 1 1 0 1 0 0 2 0 0 1 1 2 1 2 2 0 1 2 2 0 2 1 1 0  
1 1 2 0 2 1 1 1 2 2 0 1 2 1 0 0 0 1 0 1 1 1 1 2 0 1 0 0 0 0 1 1 1 0 2  
0 1 1 1 1 0 0 0 1 1 2 1 1 0 2 2 0 0 1]
```

Nous pouvons mieux visualiser les clusters créés :

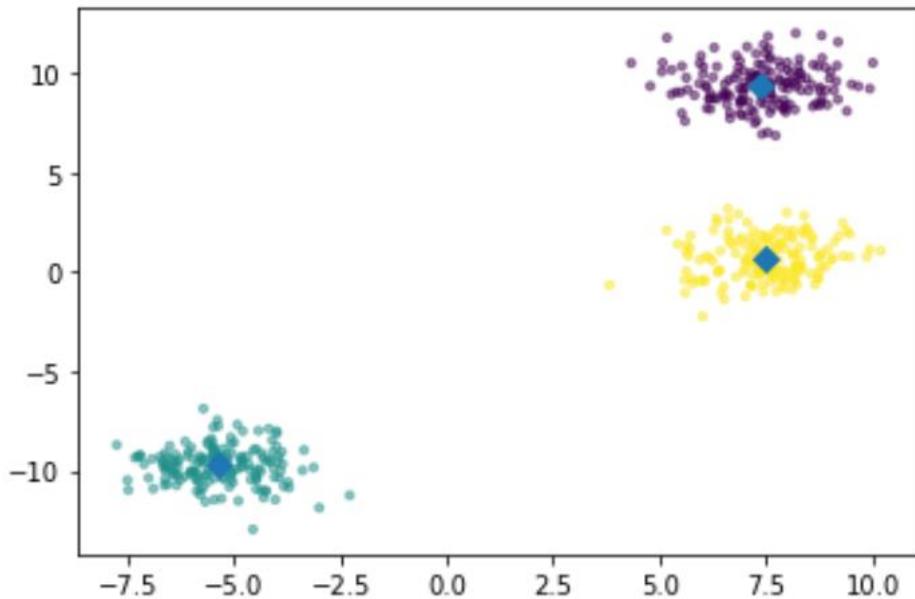
```
plt.scatter(X[:, 0], X[:, 1], s=10, c=labels)
```

```
plt.show()
```



On peut aussi afficher les centroïdes (centre de chaque cluster) dans le graphique :

```
# Valeurs des centroïdes  
  
centroids = model.cluster_centers_  
  
# Centroïde du premier groupe (cluster 0)  
  
centroid_x = centroids[:,0]  
  
# Centroïde du deuxième groupe (cluster 1)  
  
centroid_y = centroids[:,1]  
  
# Nuage de points avec les centroïdes  
  
plt.scatter(X[:, 0], X[:, 1], s=10, c=labels, alpha=0.5)  
  
plt.scatter(centroid_x, centroid_y, marker='D', s=50)  
  
plt.show()
```



Les centroïdes sont bel et bien au centre des clusters ce qui montre que la segmentation est de bonne qualité. Ceci est naturellement dû au fait que le nombre de clusters à choisir était très évident vu qu'il s'agit de données synthétiques. Dans la réalité, on est confronté à de réelles données et le nombre de clusters n'est pas toujours évident à déterminer.

Les centroïdes sont les représentants de chaque cluster. Si vous faites par exemple une segmentation de la clientèle de votre entreprise, vous considérerez que les caractéristiques de chaque centroïde sont représentatives des caractéristiques de tous les clients du cluster de ce centroïde. Puisque vous ne pouvez pas connaître personnellement chacun de vos clients, la segmentation vous donne une idée des différents groupes de clients que vous avez. Ainsi, vous pouvez adresser des campagnes marketing ciblées pour chacun de ces groupes.

SEGMENTATION DES DONNEES D'IRIS

L'ensemble de données [Iris](#)³² contient 3 classes de 50 instances chacune, chaque classe se référant à un type de plante d'iris. Une classe est séparable linéairement des 2 autres ; ces dernières ne sont pas linéairement séparables les unes des autres (Source : Michael Marshall (MARSHALL%PLU '@' io.arc.nasa.gov)).

```
import pandas as pd

iris_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']

iris = pd.read_csv('https://raw.githubusercontent.com/JosueAfo
uda/KMeans-
Lesson/master/iris.data', header=None, names=iris_names)

iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Supposons qu'on n'ait pas la colonne *class* dans l'ensemble des données et qu'on veuille les segmenter.

```
# Retrait de la colonne class

species = iris.pop('class')

iris.head()
```

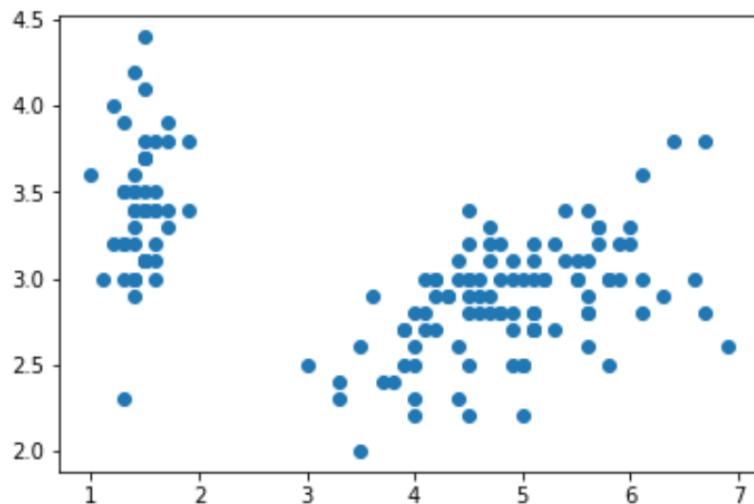
	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Comme pour les données synthétiques, nous allons visualiser l'ensemble des données par un nuage de points afin d'essayer d'identifier le nombre de clusters.

```
# Nuage de points entre la longueur des pétales et la largeur des sépales
```

```
plt.scatter('petal_length', 'sepal_width', data=iris)
```

```
plt.show()
```



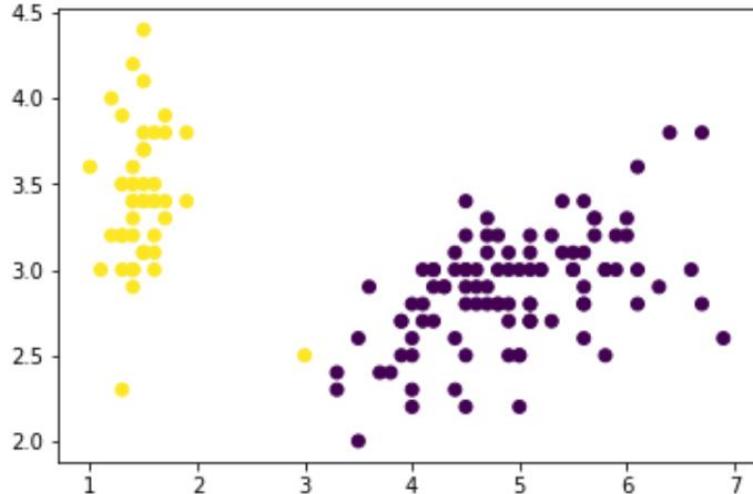
Le nuage de points ci-dessus nous suggère 2, 3 ou 4 clusters ? Contrairement aux données synthétiques précédentes, le nombre de clusters ici n'est pas évident.

```
# Mettons les données en tableau numpy
```

```
DATA = iris[['petal_length', 'sepal_width']].values
```

```
# Création d'un modèle KMeans avec 2 clusters
```

```
model_iris = KMeans(n_clusters=2)
```



Contrairement aux données synthétiques, la segmentation en deux groupes des données d'iris n'est pas de très bonne qualité. Il y a deux points jaunes qui sont carrément hors de leur groupe. Effectuons une autre segmentation avec 3 clusters.

```
# Cr ation d'un mod le KMeans avec 3 clusters

Model_iris = KMeans(n_clusters=3)

# Entra nement du mod le

Model iris.fit(DATA)
```

```

# Clusters (labels)

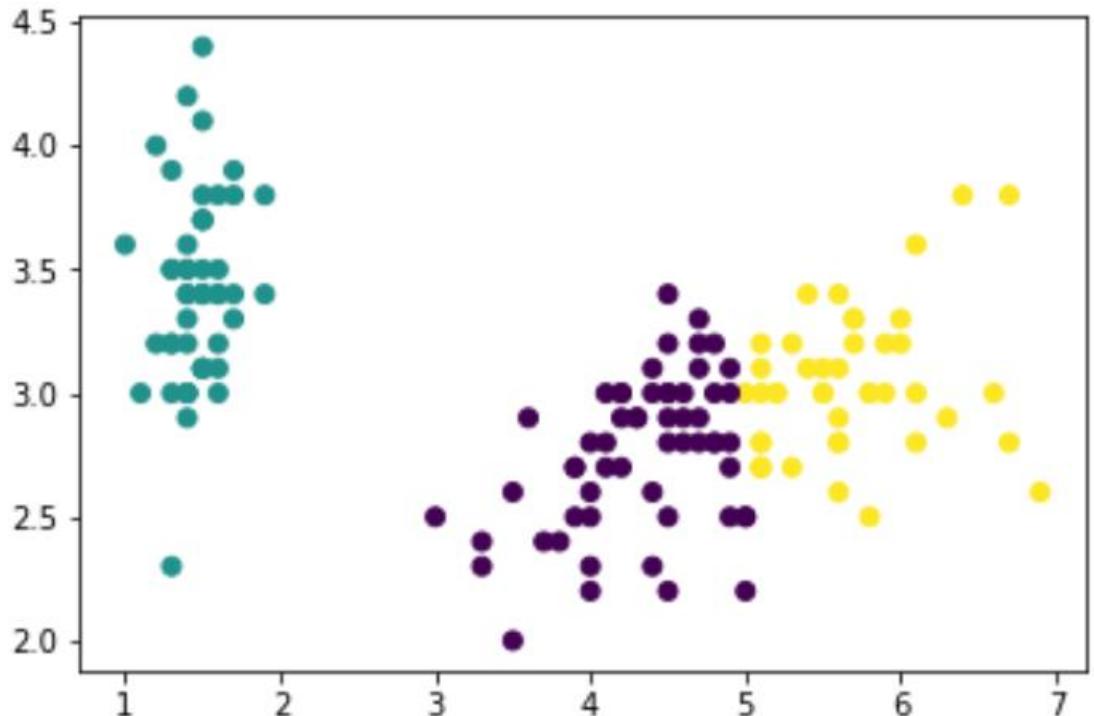
Labels_iris = Model_iris.predict(DATA)

# Visualisation des clusters

plt.scatter(DATA[:,0], DATA[:,1], c=Labels_iris)

plt.show()

```



```

# Centroïdes

Centroids = Model_iris.cluster_centers_

Centroid_x = Centroids[:,0]

Centroid_y = Centroids[:,1]

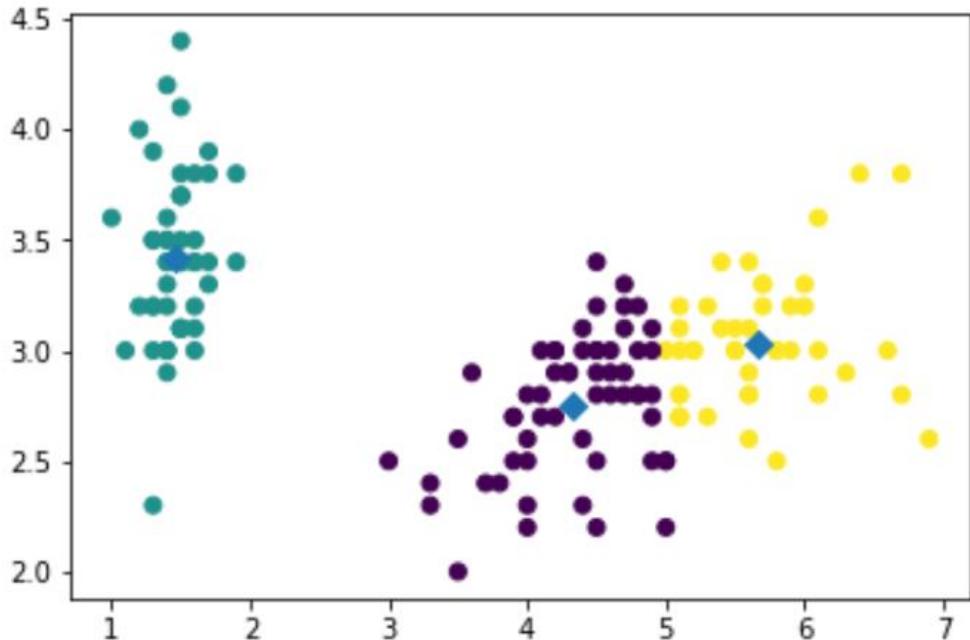
# Nuage de points avec les centroïdes

plt.scatter(DATA[:,0], DATA[:,1], c=Labels_iris)

plt.scatter(Centroid_x, Centroid_y, marker='D', s=50)

plt.show()

```



Le résultat est meilleur avec 3 clusters bien qu'il y ait encore quelques écarts. Pourquoi on n'essayerait pas avec 4, 5, 6, ..., n clusters ?

En matière de clustering avec KMeans, le choix du nombre de clusters revient à l'utilisateur de l'algorithme. Alors :

Comment être sûr du nombre de clusters ?

Comment évaluer la qualité d'une segmentation ?

Il n'y a malheureusement pas de solution miracle quant au choix du nombre de clusters. Le choix peut être effectué en fonction de notre connaissance des données et du problème business que l'on veut résoudre. Par exemple, dans le cas des données iris, on savait qu'il y a trois types d'espèces d'où le choix d'une segmentation en trois groupes. Nous allons quand même vérifier la qualité du modèle.

```
# Création d'une dataframe avec les clusters et les espèces d'iris
df = pd.DataFrame({'Labels':Labels_iris, 'Species':species})

# Tableau croisé pour compter le nombre de fois où chaque espèce coïncide avec chaque cluster (Matrice de confusion).

pd.crosstab(df['Labels'], df['Species'])
```

	Species	Iris-setosa	Iris-versicolor	Iris-virginica
Labels				
0	0	48	9	
1	50	0	0	
2	0	2	41	

Ce tableau croisé nous montre qu'on a effectivement une bonne segmentation des données d'iris avec $k = 3$. En effet, le cluster 1 correspond à 100% à l'espèce *Iris-setosa*. Le cluster 2 et le cluster 0 correspondent majoritairement à *Iris-virginica* et à *Iris-versicolor* respectivement.

Comment déterminer le meilleur nombre de clusters lorsqu'on a des données qui ne sont pas étiquetées ? Rappelons que l'objectif d'un algorithme d'apprentissage automatique non supervisé est de déterminer par lui-même des relations dans les données non-étiquetées. Il faut donc un moyen pour mesurer la qualité d'une segmentation et ce moyen doit utiliser uniquement les clusters et les échantillons eux-mêmes. C'est ce que nous verrons dans la section suivante.

SEGMENTATION D'ESPECES DE POISSONS

L'ensemble des données de cette partie est un tableau d'échantillons donnant des mesures (le poids en grammes, la longueur en centimètres, le rapport en pourcentage de la hauteur à la longueur, etc.) de 7 espèces de poissons codées de 1 à 7. Chaque ligne représente un poisson individuel. Ces données proviennent de [Journal of Statistics Education](#).³³ Consultez la [description](#)³⁴ de ces données que vous pouvez télécharger via cette [page](#).³⁴

```
columns = ['Species', 'Weight', 'Length1', 'Length2', 'Length3',
           'Height%', 'Width%', 'Sex']

fishes = pd.read_csv('http://jse.amstat.org/datasets/fishcatch
.dat.txt', delimiter= '\s+', header=None, names=columns)

fishes.head()
```

	Species	Weight	Length1	Length2	Length3	Height%	Width%	Sex
1	1	242.0	23.2	25.4	30.0	38.4	13.4	NaN
2	1	290.0	24.0	26.3	31.2	40.0	13.8	NaN
3	1	340.0	23.9	26.5	31.1	39.8	15.1	NaN
4	1	363.0	26.3	29.0	33.5	38.0	13.3	NaN
5	1	430.0	26.5	29.0	34.0	36.6	15.1	NaN

```

# Informations sur les données

fishes.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 159 entries, 1 to 159
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   Species   159 non-null    int64  
 1   Weight    158 non-null    float64 
 2   Length1   159 non-null    float64 
 3   Length2   159 non-null    float64 
 4   Length3   159 non-null    float64 
 5   Height%   159 non-null    float64 
 6   Width%    159 non-null    float64 
 7   Sex        72 non-null    float64 
dtypes: float64(7), int64(1)
memory usage: 11.2 KB

# Suppression de la colonne 'Sex' qui contient beaucoup de valeurs manquantes

fishes.drop(['Sex'], axis=1, inplace=True)

# Suppression de la ligne de données manquantes dans la colonne 'Weight'

fishes.dropna(axis=0, inplace=True)

# Supposons qu'on a pas la colonne Species

Species = fishes.pop('Species')

# Création du tableau d'échantillons

fishes_array = fishes.values

```

METHODE DU COUDE (*ELBOW METHOD*)

Puisqu'on n'a pas une idée du nombre de clusters adéquat, on exécutera l'algorithme KMeans pour plusieurs nombres de clusters et on choisira le meilleur nombre grâce à la méthode du coude (*Elbow method*³⁵ Voir graphique ci-dessous).

```

n = range(1,11)

inerties = []

for k in n:

    # Modèle KMeans avec k clusters

    kmeans = KMeans(n_clusters=k)

    # Entraînement du modèle

    kmeans.fit(fishes_array)

    # Inerties de chacun des n modèles

    inerties.append(kmeans.inertia_)

# Graphique d'inertie en fonction du nombre de clusters

plt.plot(n, inerties, '-o')

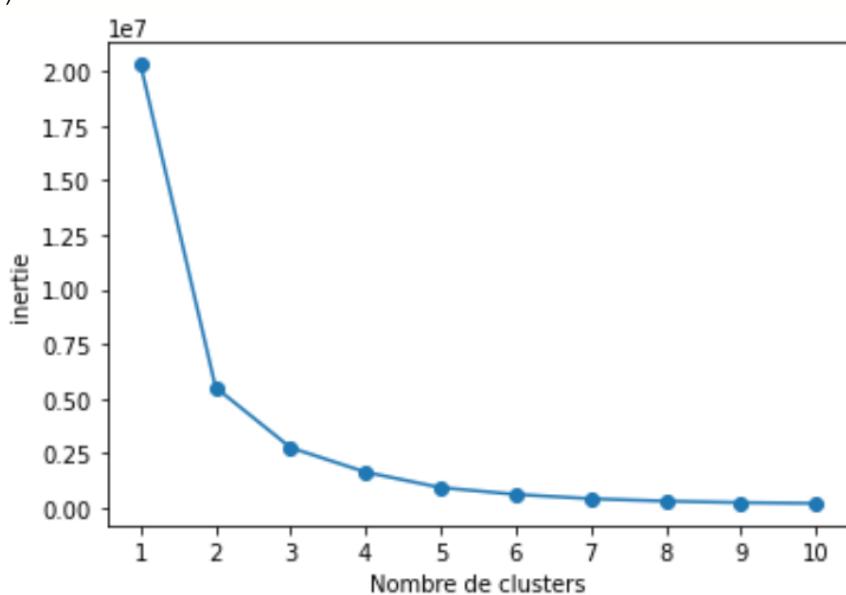
plt.xlabel('Nombre de clusters')

plt.ylabel('inertie')

plt.xticks(n)

plt.show()

```



La méthode du 'coude' consiste à choisir le k (nombre de clusters) pour lequel on observe un 'coude' dans la courbe de l'inertie en fonction des k. En d'autres termes, on choisit le k à partir duquel l'inertie commence à diminuer plus lentement. En effet, un bon clustering a des clusters serrés (c'est-à-dire que la distance intra-cluster est faible et la distance inter-cluster est grande) donc de faible inertie.

En appliquant donc cette méthode, on choisit k = 4. Visualisons à présent les clusters.

```
# Construction du modèle KMeans avec 4 clusters

model_fishes = KMeans(n_clusters=4) # Vous pouvez mettre d'autres valeurs pour n_clusters et observez le graphique

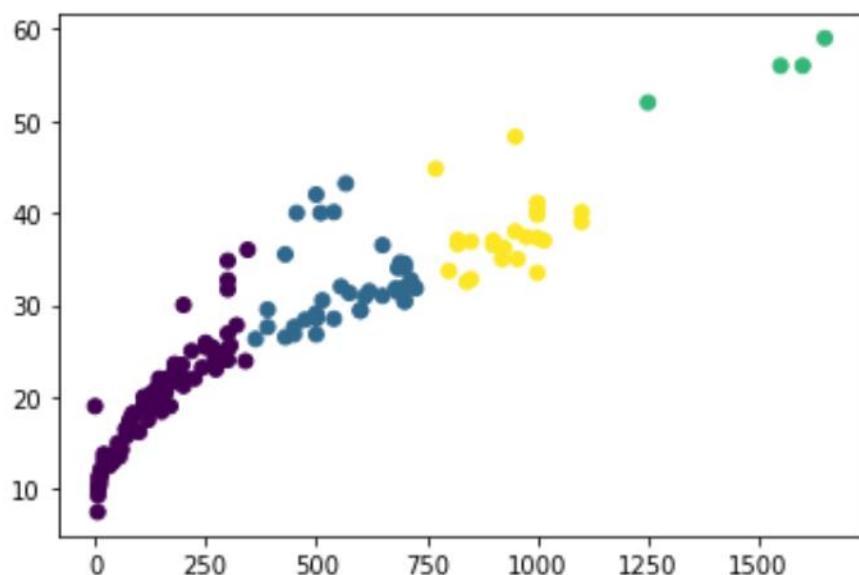
# Entraînement + Predictions

labels_fishes = model_fishes.fit_predict(fishes_array)

# Visualisation des clusters

plt.scatter(fishes_array[:,0], fishes_array[:,1], c=labels_fishes)

plt.show()
```



SEGMENTATION DES VINS

Ces données sont le résultat d'une analyse chimique de vins cultivés dans la même région d'Italie mais issus de trois cultures différentes. L'analyse a déterminé les quantités de 13 constituants trouvés dans chacun des trois types de vins (Source : [UCI Machine Learning Repository](#))³⁶.

```
wines_cols = ['class_label', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium',  
             'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',  
             'Color intensity', 'Hue', 'OD280/OD315 of dilute  
             wines', 'Proline']  
  
wines = pd.read_csv('https://raw.githubusercontent.com/JosueAfuoda/KMeans-  
Lesson/master/wine.data', header=None, names=wines_cols, index  
_col=False)  
  
wines.head()
```

	class_label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline	
0	1	14.23	1.71	2.43		15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14		11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67		18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1165
3	1	14.37	1.95	2.50		16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87		21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

```
wines.info()
```

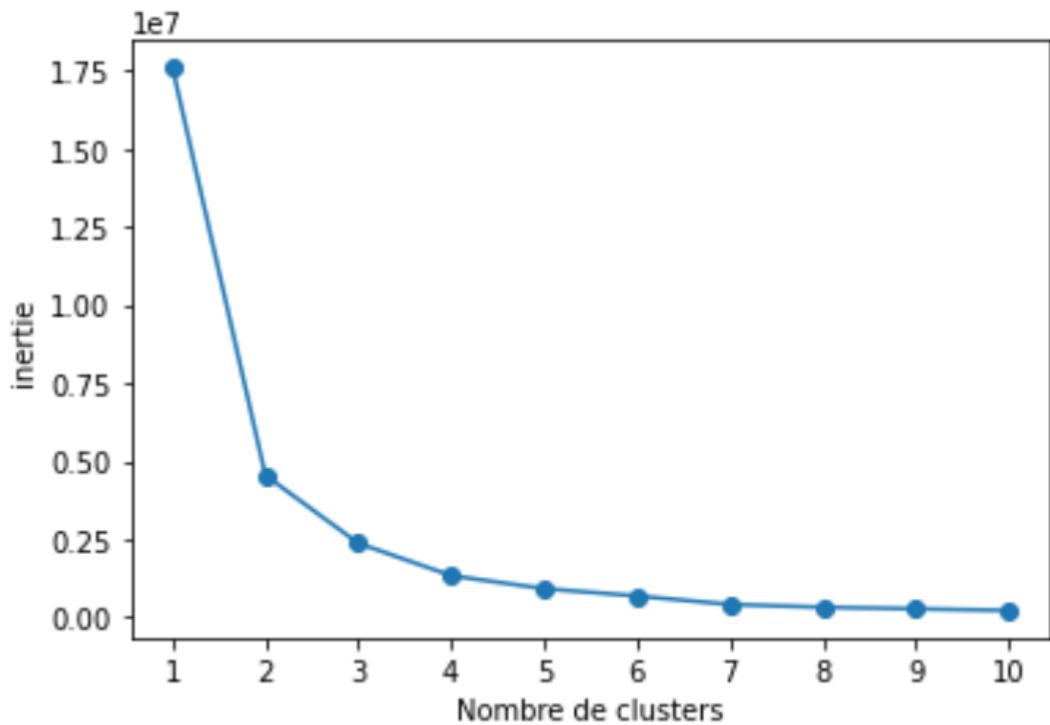
```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 178 entries, 0 to 177  
Data columns (total 14 columns):  
 #   Column           Non-Null Count  Dtype     
 ---  --  
 0   class_label      178 non-null    int64    
 1   Alcohol          178 non-null    float64  
 2   Malic acid       178 non-null    float64  
 3   Ash              178 non-null    float64  
 4   Alcalinity of ash 178 non-null    float64  
 5   Magnesium        178 non-null    int64    
 6   Total phenols    178 non-null    float64  
 7   Flavanoids       178 non-null    float64  
 8   Nonflavanoid phenols 178 non-null    float64  
 9   Proanthocyanins 178 non-null    float64  
 10  Color intensity 178 non-null    float64  
 11  Hue              178 non-null    float64  
 12  OD280/OD315 of diluted wines 178 non-null    float64  
 13  Proline          178 non-null    int64    
dtypes: float64(11), int64(3)  
memory usage: 19.6 KB
```

```
wines['class_label'].unique()  
  
array([1, 2, 3], dtype=int64)
```

Il y a trois catégories de vins (numérotées 1, 2 et 3) dans l'ensemble des données. Supposons que nous n'avions pas la colonne *class_label*. L'objectif sera donc de regrouper ces vins en des catégories distinctes en utilisant leurs caractéristiques physico-chimiques.

Dans un premier temps, appliquons la méthode du coude pour déterminer le nombre de clusters adéquat et dans un second temps, construisons le modèle *KMeans* avec le nombre de clusters déterminé.

```
class_labels = wines.pop('class_label')  
  
wines_array = wines.values  
  
inertias = []  
  
for k in n: # n = range(1,11)  
  
    # Modèle KMeans avec k clusters  
  
    kmeans = KMeans(n_clusters=k)  
  
    # Entrainement du modèle  
  
    kmeans.fit(wines_array)  
  
    # Inerties de chacun des n modèles  
  
    inertias.append(kmeans.inertia_)  
  
# Graphique d'inertie en fonction du nombre de clusters  
  
plt.plot(n, inertias, '-o')  
  
plt.xlabel('Nombre de clusters')  
  
plt.ylabel('inertie')  
  
plt.xticks(n)  
  
plt.show()
```



Ce graphique nous suggère 3 clusters.

```
wines_model = KMeans(n_clusters=3)

labels_wines = wines_model.fit_predict(wines_array)

wines_df = pd.DataFrame({'Labels':labels_wines, 'Varieties':class_labels})

pd.crosstab(wines_df['Labels'], wines_df['Varieties'])
```

Varieties	1	2	3
Labels			
0	13	20	29
1	46	1	0
2	0	50	19

L'examen de ce tableau croisé nous montre qu'on n'a pas une bonne segmentation. Et pourtant, il y a bel et bien 3 variétés de vins dans l'ensemble des données. Que s'est-il donc passé ?

Dans notre ensemble de données, on a des variables qui ne sont pas à la même échelle. Ceci affecte la qualité de la segmentation étant donné que l'algorithme *KMeans* calcule les distances entre les points et regroupe les k points les plus proches. Il est recommandé alors de standardiser les données avant application de l'algorithme.

Nous pouvons combiner les deux étapes (Standardisation et Modélisation) dans un pipeline :

```
from sklearn.preprocessing import StandardScaler  
  
from sklearn.pipeline import make_pipeline  
  
scaler = StandardScaler()  
  
wines_sc_model = KMeans(n_clusters=3)  
  
# Création d'un objet pipeline  
  
pipeline = make_pipeline(scaler, wines_sc_model)  
  
labels_sc_wines = pipeline.fit_predict(wines_array)  
  
wines_sc_df = pd.DataFrame({'Labels':labels_sc_wines, 'Varieties':class_labels})  
  
pd.crosstab(wines_sc_df['Labels'], wines_sc_df['Varieties'])
```

Varieties 1 2 3

Labels

0	59	3	0
1	0	3	48
2	0	65	0

La standardisation, en réduisant la variance dans les données, a augmenté considérablement la qualité de notre segmentation.

CONCLUSION

En définitive, l'algorithme KMeans est un algorithme de Machine Learning non supervisé dont l'implémentation dans Python est très simple. Dans ce projet, nous avons vu le flux de travail dans un projet de clustering. Bien que le choix du nombre de clusters revienne à l'utilisateur de l'algorithme, il doit être fait soigneusement. La visualisation des données ainsi que la méthode du coude permettent d'effectuer un bon choix. Par ailleurs, la compréhension du business est aussi un grand atout.

PROJET 8 : SEGMENTATION DE LA CLIENTELE D'UNE ENTREPRISE

INTRODUCTION

La segmentation de la clientèle d'une entreprise est une pratique de partitionnement des clients en groupes d'individus de mêmes caractéristiques. Cette stratégie permet de comprendre le comportement des clients, sur la base de leurs données historiques, et donc d'utiliser de manière efficace les ressources allouées au Marketing.

La segmentation se fait par une technique de Machine Learning non supervisé appelée Clustering. L'algorithme de clustering le plus populaire est KMeans. Dans ce projet, nous appliquerons cet algorithme pour segmenter la clientèle d'une entreprise.

LIBRAIRIES FONDAMENTALES

```
#Importation des librairies

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
```

DONNEES

```
# Importation des données

customers = pd.read_csv('https://raw.githubusercontent.com/Jos
ueAfouda/Customer-Segmentation/master/Cust_Segmentation.csv')

# Affichage des 5 premières lignes de la dataframe

customers.head()
```

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	Address	DebtIncomeRatio
0	1	41	2	6	19	0.124	1.073	0.0	NBA001	6.3
1	2	47	1	26	100	4.582	8.218	0.0	NBA021	12.8
2	3	33	2	10	57	6.111	5.802	1.0	NBA013	20.9
3	4	29	2	4	19	0.681	0.516	0.0	NBA009	6.3
4	5	47	1	31	253	9.308	8.908	0.0	NBA008	7.2

```
# Structure des données

customers.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 850 entries, 0 to 849
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  --  
 0   Customer Id     850 non-null    int64  
 1   Age              850 non-null    int64  
 2   Edu              850 non-null    int64  
 3   Years Employed  850 non-null    int64  
 4   Income           850 non-null    int64  
 5   Card Debt        850 non-null    float64 
 6   Other Debt       850 non-null    float64 
 7   Defaulted        700 non-null    float64 
 8   Address          850 non-null    object  
 9   DebtIncomeRatio  850 non-null    float64 
dtypes: float64(4), int64(5), object(1)
memory usage: 66.5+ KB
```

Chaque ligne représente un client et chaque colonne représente une variable caractéristique des clients (Age, Revenu, Adresse, etc.). Toutes les entreprises peuvent disposer facilement de ce genre de données sur leur clientèle.

L'algorithme K-means ne traite pas directement les variables qualitatives (comme la plupart des algorithmes de Machine Learning de [Scikit-Learn](#))³⁷ puisqu'il est question ici de calcul de distance. Supprimons donc la colonne *Address*.

```
#Suppression de 'Address'

customers = customers.drop("Address", axis = 1)
```

STANDARDISATION DES DONNEES

Pour donner le même poids à chaque point, nous allons normaliser les données :

```

# Importation de StandardScaler

from sklearn.preprocessing import StandardScaler

# Standardisation des données

X = customers.values[:, 1:] #Toutes les colonnes sauf la colonne 'Customer Id'

X = np.nan_to_num(X) #Remplacer les Nan par 0 et les infinis par les plus grands nombres finis

cluster_data = StandardScaler().fit_transform(X)

cluster_data

array([[ 0.74291541,  0.31212243, -0.37878978, ..., -0.59048916,
       -0.52379654, -0.57652509],
       [ 1.48949049, -0.76634938,  2.5737211 , ...,  1.51296181,
       -0.52379654,  0.39138677],
       [-0.25251804,  0.31212243,  0.2117124 , ...,  0.80170393,
        1.90913822,  1.59755385],
       ...,
       [-1.24795149,  2.46906604, -1.26454304, ...,  0.03863257,
        1.90913822,  3.45892281],
       [-0.37694723, -0.76634938,  0.50696349, ..., -0.70147601,
       -0.52379654, -1.08281745],
       [ 2.1116364 , -0.76634938,  1.09746566, ...,  0.16463355,
       -0.52379654, -0.2340332 ]])

```

EXECUTION DE K-MEANS

Faisons un partitionnement en 3 groupes.

```

# Importation de KMeans

from sklearn.cluster import KMeans

k = 3

k_means = KMeans(init="k-
means++", n_clusters=k, n_init=12) #Création d'un objet de la classe KMeans

```

```

k_means.fit(cluster_data) #Application de l'algorithme aux données standardisées

labels = k_means.labels_ #Obtention des étiquettes

#Assignment de chaque client à un groupe (cluster)

customers["Label"] = labels

# Affichage de la dataframe avec les étiquettes

customers.head()

```

En faisant la moyenne des variables au niveau de chaque cluster (0, 1 et 2) on aura les caractéristiques du centroïde (représentant) de chaque groupe.

```
customers.groupby("Label").mean()
```

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio
Label									
0	426.122905	33.817505	1.603352	7.625698	36.143389	0.853128	1.816855	0.000000	7.964991
1	424.408163	43.000000	1.931973	17.197279	101.959184	4.220673	7.954483	0.162393	13.915646
2	424.451807	31.891566	1.861446	3.963855	31.789157	1.576675	2.843355	0.993939	13.994578

Nous avons donc le représentant de chacun des groupes de clients ainsi que leurs caractéristiques. Dans chaque cluster, les clients sont similaires les uns aux autres et on peut donc considérer comme caractéristiques communes dans chaque cluster, celles de son représentant c'est-à-dire le centroïde. Ainsi, on peut créer 3 profils de clients :

- Profil 1 (Cluster 0) : Âges moyens (34 ans), Salaires moyens (environ 36000 dollars par an en moyenne) et financièrement responsables.
- Profil 2 (Cluster 1) : Clients les plus âgés (43 ans en moyenne), ayant les plus gros salaires (102000 dollars par an en moyenne) et légèrement endettés.
- Profil 3 (Cluster 2) : Les plus jeunes (32 ans en moyenne), Salaires les plus bas (environ 32000 dollars par an en moyenne) et très endettés.

CONCLUSION

L'implémentation de l'algorithme KMeans dans Python est très simple pour réaliser une segmentation. La segmentation des clients est très importante pour toute entreprise et permet d'analyser les comportements des clients. L'algorithme KMeans peut être aussi utilisé dans d'autres domaines tel que la santé, la recherche scientifique, etc.

PROJET 9 : REDUCTION DE LA DIMENSION D'UN ENSEMBLE DE DONNEES PAR L'ANALYSE EN COMPOSANTES PRINCIPALES

INTRODUCTION

En tant que Data Scientist / Data Analyst, vous serez amenés à travailler avec de grands ensembles de données possédant des centaines, milliers voir des millions de variables. Ce genre de données, se retrouvent facilement dans tous les domaines comme la Santé, l'Ingénierie, la Finance, le Web, etc. Les ensembles de données à grande dimension posent de nombreux défis dont :

- Le problème de stockage et des ressources de calcul très coûteux en termes d'analyse ;
- Le problème de surapprentissage dans des tâches de modélisation ;
- Le fait que certains algorithmes, très utilisés en Machine Learning, ne peuvent pas traiter ce genre de données ;
- Etc.

Alors, que faire pour traiter et analyser les ensembles de données de grande dimension ? On fait appel aux techniques de réduction de dimensionnalité.

Le but de la réduction de la dimensionnalité est de réduire les dimensions des ensembles de données pour surmonter les défis cités ci-dessus posés par les données de grande dimension. Il existe plusieurs techniques de réduction de dimensionnalité parmi lesquelles la plus populaire est l'Analyse en Composantes Principales (ACP). A travers ce projet, vous apprendrez ce qu'est l'ACP et comment l'utiliser dans Python pour réduire la dimensionnalité d'un ensemble de données.

ACP : Définition

L'idée générale derrière l'ACP est la décorrélation des variables. En effet l'ACP transforme les variables d'un jeu de données en un nouvel ensemble de variables appelées composantes principales. Chacune de ces composants représente (explique) un certain pourcentage de la variabilité totale des données le long d'un axe orthogonal les uns aux autres. Le premier axe (donc la première composante principale) est la direction dans laquelle les données varient le plus.

La réduction de dimensionnalité avec la technique d'ACP se fait suivant le principe selon lequel les composantes principales avec une faible variance constituent le 'bruit' (c'est ce bruit qui est l'une des causes majeures de surapprentissage des algorithmes de Machine Learning) et les composantes principales avec une forte variance constituent l'information contenue dans les données. Donc on retient les premières composantes qui expliquent la majorité de la variance (90% par exemple) totale des données.

L'ACP peut être utilisée avant la construction d'un modèle d'apprentissage supervisé pour améliorer les performances et la généralisation de ce modèle. Il peut également être utile pour un apprentissage non supervisé. Par exemple, on peut combiner l'ACP avec une technique de clustering pour segmenter efficacement la clientèle d'une entreprise.

LIBRAIRIES

```
import pandas as pd

import numpy as np

from scipy.stats import pearsonr

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline
```

EXERCICE D'INITIATION A L'ACP

On dispose d'un jeu de données de mesures (largeur et longueur) sur les grains de maïs.

```
# Importation des données

grains_df = pd.read_csv('https://raw.githubusercontent.com/Jos
ueAfouda/PCA-Lesson/master/seeds-width-vs-
length.csv', header=None, names=['largeur', 'longueur'])

print(grains_df.info())

print('\n')
```

```
grains_df.head()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210 entries, 0 to 209
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype  
---  --          -----          ----- 
 0   largeur    210 non-null    float64 
 1   longueur   210 non-null    float64 
dtypes: float64(2)
memory usage: 3.4 KB
None
```

	largeur	longueur
0	3.312	5.763
1	3.333	5.554
2	3.337	5.291
3	3.379	5.324
4	3.562	5.658

Calculons le coefficient de corrélation de Pearson des deux variables :

```
# Création d'un tableau numpy 2D à partir de grains_df
grains_array = grains_df.values

l = grains_array[:, 0]
L = grains_array[:, 1]

# Coefficient de corrélation entre la largeur et la longueur des grains

coef_cor, pvalue = pearsonr(l, L)
```

```

print("Le coefficient de corrélation entre la largeur et la longueur des grains est égal à ", coef_cor, "pour une p_valeur égale à ", pvalue)
print('\n')

print('La figure ci-dessous montre bien cette forte corrélation entre les variables :')

# Visualisation du nuage de points

plt.scatter(l, L)

plt.xlabel('Largeur des grains')

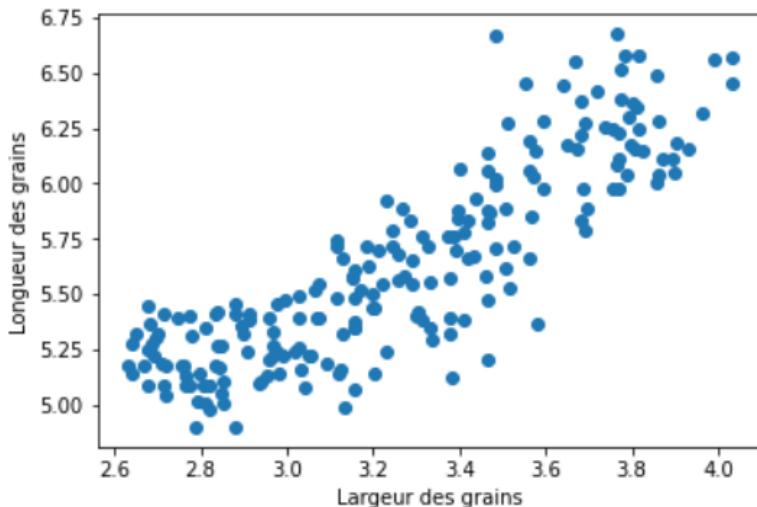
plt.ylabel('Longueur des grains')

plt.show()

```

Le coefficient de corrélation entre la largeur et la longueur des grains est égal à 0.8604149377143466 pour une p_valeur égale à 8.121332906193694e-63

La figure ci-dessous montre bien cette forte corrélation entre les variables :



Il existe une forte corrélation positive et statistiquement significative entre la largeur et la longueur des grains. Appliquons donc la technique de l'ACP pour décorrélérer ces variables.

```

# Création d'une instance de la classe PCA

model_acp = PCA()

```

```

# Application du modèle aux données (Entrainement du modèle par les données et transformation des données par le modèle)

grains_acp = model_acp.fit_transform(grains_array)

print(type(grains_acp))

<class 'numpy.ndarray'>

```

L'argument *n_components* de la fonction *PCA()* est par défaut égal au nombre de variables de l'ensemble des données. Le résultat de l'application de l'ACP sur les données est un tableau numpy 2D composée de 02 composantes principales. Visualisons ce résultat :

```

# Première composante principale

comp1 = grains_acp[:, 0]

# Deuxième composante principale

comp2 = grains_acp[:, 1]

# Calcul du coefficient de corrélation entre comp1 et comp2

cor_comp, pvalue_comp = pearsonr(comp1, comp2)

print("Le coefficient de corrélation entre les 02 composantes principales est égal à ", cor_comp)
print('\n')

print('La figure ci-dessous montre bien cette très faible corrélation entre les composantes principales :')

# Nuage de points entre comp1 et comp2

plt.scatter(comp1, comp2)

plt.xlabel('Composante 1')

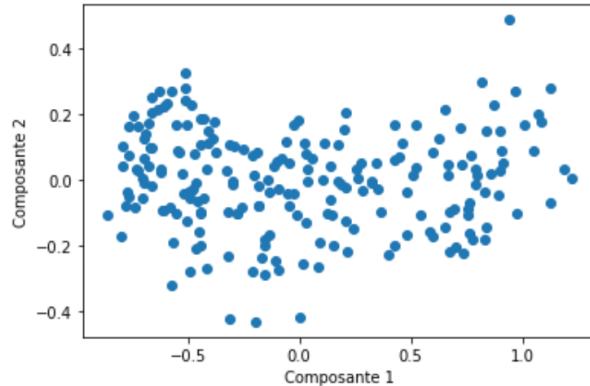
plt.ylabel('Composante 2')

plt.show()

```

Le coefficient de corrélation entre les 02 composantes principales est égal à 2.5478751053409354e-17

La figure ci-dessous montre bien cette très faible corrélation entre les composantes principales :



Il n'y a pas de corrélation linéaire entre les deux composantes principales. L'ACP transforme les variables corrélées d'un jeu de données en un nouvel ensemble de variables non corrélées.

Quelle est la variance expliquée par chacune des 02 composantes principales ?

```
# Attribut du modèle PCA pour retrouver la quantité de variance expliquée par chaque composante
```

```
model_acp.explained_variance_
```

```
array([0.31595462, 0.02301882])
```

```
# Attribut du modèle PCA pour retrouver le pourcentage de variance expliquée par chaque composante
```

```
model_acp.explained_variance_ratio_
```

```
array([0.93209254, 0.06790746])
```

```
# Pour voir le pourcentage de variance qu'on peu expliquer en utilisant un certain nombre de composantes :
```

```
model_acp.explained_variance_ratio_.cumsum()
```

```
array([0.93209254, 1.          ])
```

```

# Attribut du modèle PCA pour retrouver le nombre de composantes principales

model_acp.n_components_

2

# Visualisation du pourcentage de variance expliquée par chaque composante

plt.bar(range(model_acp.n_components_), model_acp.explained_variance_ratio_) # Ici range(model_acp.n_components_) est égale à range(2)

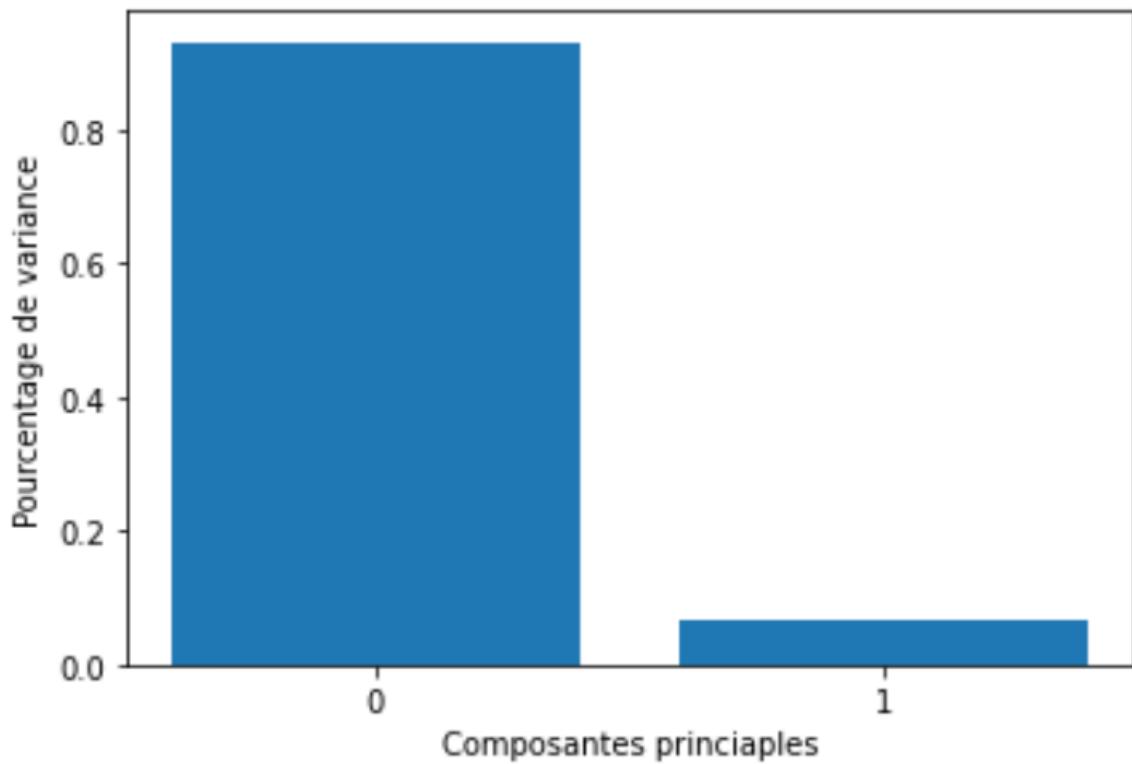
plt.xlabel('Composantes principales')

plt.ylabel('Pourcentage de variance')

plt.xticks(range(model_acp.n_components_))

plt.show()

```



La première composante principale explique un peu plus de 93% de la variance des données tandis que la deuxième composante est plutôt anecdotique (moins de 7% de la variance totale). On peut donc dire que la **dimension intrinsèque** de ce jeu de données est 1.

```
# Attribut pour retrouver la moyenne de chaque variable originelle
```

```
model_acp.mean_
```

```
array([3.25860476, 5.62853333])
```

Les deux valeurs de ce tableau numpy 1D représentent respectivement l'abscisse et l'ordonnée du repère orthogonal formé par les 02 axes principaux.

```
# Vérification de la moyenne de chaque variable originelle
```

```
grains_df.mean(axis=0)
```

```
largeur      3.258605
longueur     5.628533
dtype: float64
```

```
# Attribut pour retrouver les composants principaux au niveau de chaque axe
```

```
model_acp.components_
```

```
array([[ 0.63910027,  0.76912343],
       [-0.76912343,  0.63910027]])
```

Le tableau Numpy 2D résultant de l'attribut components_ d'un modèle d'ACP nous indique dans quelle mesure le vecteur de chaque composante principale est affectée par chaque variable originelle.

Pour cet exercice, on a :

- Composante principale 1 = 0.63910027 largeur + 0.76912343 longueur
- Composante principale 2 = -0.76912343 largeur + 0.63910027 longueur

Cela peut aider à l'interprétation des composantes principales.

```
# Nuage de points des données originelles
```

```
plt.scatter(l, L)
```

```

# Moyenne de chaque composante

mean = model_acp.mean_

# Tracé du premier axe sous forme de flèche

axe1 = model_acp.components_[0, :]

plt.arrow(mean[0], mean[1], axe1[0], axe1[1], color='red', width=0.01)

# Tracé du deuxième axe sous forme de flèche

axe2 = model_acp.components_[1, :]

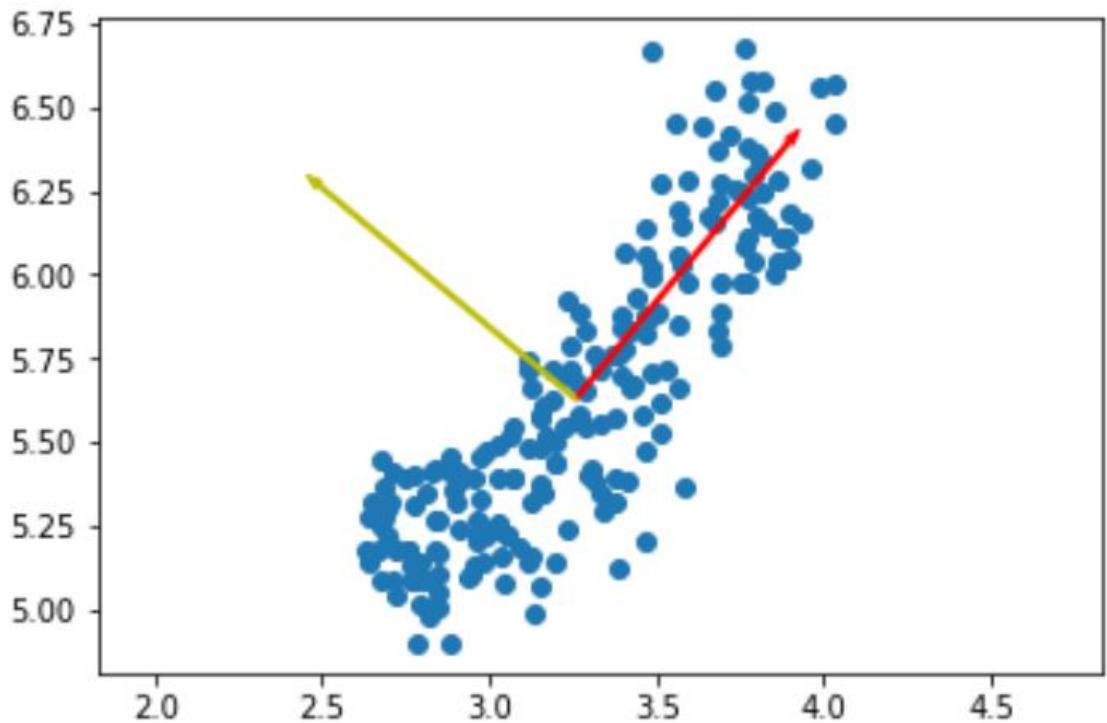
plt.arrow(mean[0], mean[1], axe2[0], axe2[1], color='y', width=0.01)

# Mettre les axes dans la même échelle

plt.axis('equal')

plt.show()

```



Nous pouvons décrire chaque point de l'ensemble des données comme une combinaison linéaire de ces deux vecteurs pondérés par un poids. En effet, un point peut être repéré dans le repère (Largeur, Longueur) comme dans le repère formé par les deux axes principaux (axe1 en rouge, axe2 en jaune).

Le graphique nous montre clairement que les points varient majoritairement suivant le premier axe principal (en rouge). La dimension intrinsèque de nos données est donc bel et bien 1.

ACP DANS UN PIPELINE

L'ensemble des données de cette section est un tableau d'échantillons donnant des mesures (le poids en grammes, la longueur en centimètres, le rapport en pourcentage de la hauteur à la longueur, etc.) de 7 espèces de poissons codées de 1 à 7. Chaque ligne représente un poisson individuel. Ces données proviennent de [*Journal of Statistics Education*](#)³³. Consultez la [description](#)³⁴ de ces données que vous pouvez télécharger via cette [page](#)³⁴.

```
# Importation des données

columns = ['Species', 'Weight', 'Length1', 'Length2', 'Length3',
           'Height%', 'Width%', 'Sex']

fishes = pd.read_csv('http://jse.amstat.org/datasets/fishcatch
.dat.txt', delimiter= '\s+', header=None, names=columns)

print(fishes.info())

print('\n')

fishes.head()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 159 entries, 1 to 159
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   Species    159 non-null    int64  
 1   Weight     158 non-null    float64 
 2   Length1   159 non-null    float64 
 3   Length2   159 non-null    float64 
 4   Length3   159 non-null    float64 
 5   Height%   159 non-null    float64 
 6   Width%    159 non-null    float64 
 7   Sex        72 non-null    float64 
dtypes: float64(7), int64(1)
memory usage: 11.2 KB
None
```

	Species	Weight	Length1	Length2	Length3	Height%	Width%	Sex
1	1	242.0	23.2	25.4	30.0	38.4	13.4	NaN
2	1	290.0	24.0	26.3	31.2	40.0	13.8	NaN
3	1	340.0	23.9	26.5	31.1	39.8	15.1	NaN
4	1	363.0	26.3	29.0	33.5	38.0	13.3	NaN
5	1	430.0	26.5	29.0	34.0	36.6	15.1	NaN

```
# Suppression des colonnes 'Species' et 'Sex'

fishes.drop(['Species', 'Sex'], axis=1, inplace=True)

print(fishes.head(3))

# Remplacement de la valeur manquante au niveau de la variable
# 'Weight'

fishes['Weight'] = fishes['Weight'].fillna(fishes['Weight'].me
an(axis=0))

# Vérifions les données manquantes

print('\n')

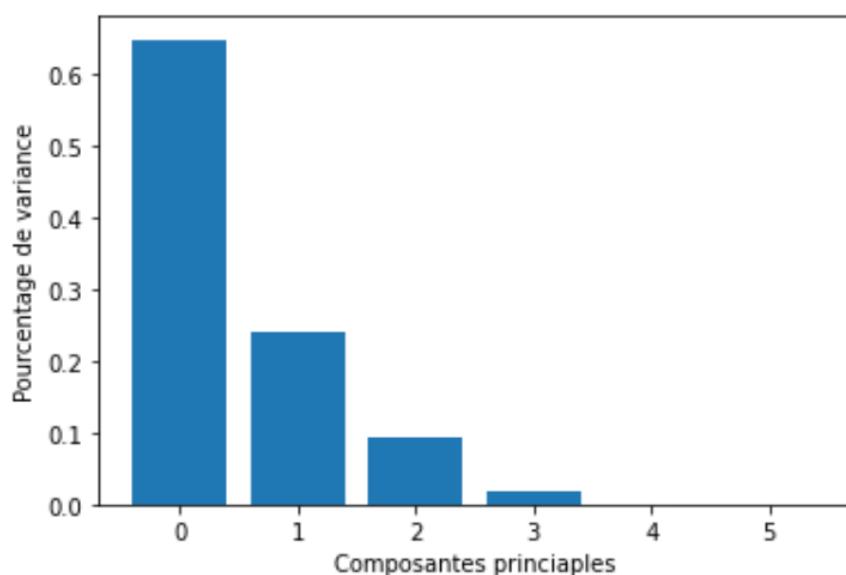
print(fishes.isna().sum())
```

	Weight	Length1	Length2	Length3	Height%	Width%
1	242.0	23.2	25.4	30.0	38.4	13.4
2	290.0	24.0	26.3	31.2	40.0	13.8
3	340.0	23.9	26.5	31.1	39.8	15.1

Weight	0
Length1	0
Length2	0
Length3	0
Height%	0
Width%	0
dtype:	int64

L'ensemble des données ci-dessus est de 6 dimensions (6 variables). A l'aide de la technique de l'ACP, nous réduirons sa dimension afin de trouver sa dimension intrinsèque.

```
# Création d'un tableau numpy 2D à partir de la dataframe  
  
fishes_array = fishes.values  
  
pipe = Pipeline([('scaler', StandardScaler()),  
                 ('pca', PCA())])  
  
# Application de l'objet pipe aux données  
  
pipe.fit(fishes_array)  
  
# Visualisation du pourcentage de variance expliquée par chaque composante  
  
plt.bar(range(pipe[1].n_components_),  
        pipe[1].explained_variance_ratio_) # Ici range(pipe[1].n_components_) est égale à range(6)  
  
plt.xlabel('Composantes principales')  
  
plt.ylabel('Pourcentage de variance')  
  
plt.xticks(range(pipe[1].n_components_))  
  
plt.show()
```



```
# Cumul du pourcentage de variance expliquée

pipe[1].explained_variance_ratio_.cumsum()

array([0.64945036, 0.89047222, 0.98237637, 0.99955204, 0.99995958,
       1.        ])
```

Les deux premières composantes expliquent 89% de la variance totale des données. On peut donc dire que la dimension intrinsèque de cet ensemble de données est raisonnablement égale à 2. Transformons donc la dataframe originelle de 6 variables en une dataframe de 2 variables.

```
# Création d'un pipeline

pipe2 = Pipeline([('scaler', StandardScaler()),

                  ('pca', PCA(n_components=2))])

# Transformation des données

fishes_transformed = pipe2.fit_transform(fishes_array)

fishes_transformed.shape

(159, 2)
```

Le nouveau jeu de données comporte 159 lignes et 2 colonnes. La réduction de dimensionnalité a été parfaitement opérée 😊

SELECTION DES COMPOSANTS PRINCIPAUX SELON LE POURCENTAGE DE VARIANCE EXPLIQUEE

Au lieu de créer d'abord un modèle ACP avec comme nombre de composantes égal au nombre de variables puis de déterminer ensuite la dimension intrinsèque raisonnable et enfin recréer un autre modèle ACP avec cette dimension intrinsèque pour transformer les données, on peut directement spécifier le pourcentage de la variance qu'on désire garder.

Dans cette section, nous utiliserons le jeu de données Iris. L'ensemble de données Iris contient 3 classes de 50 instances chacune, chaque classe se référant à un type de plante d'iris. Une classe est séparable linéairement des 2 autres ; ces derniers ne sont pas linéairement séparables les uns des autres (Source : Michael Marshall (MARSHALL%PLU '@' io.arc.nasa.gov)).

```

# Importation des données

import pandas as pd

iris_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']

iris = pd.read_csv('https://raw.githubusercontent.com/JosueAfo
uda/PCA-
Lesson/master/iris.data', header=None, names=iris_names)

print(iris.info())

print('\n')

iris.head()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   sepal_length  150 non-null   float64 
 1   sepal_width   150 non-null   float64 
 2   petal_length  150 non-null   float64 
 3   petal_width   150 non-null   float64 
 4   class        150 non-null   object  
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None

```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Quel est le nombre de composantes principales qu'il faut pour expliquer 90% de la variance totale contenue dans les données d'iris ?

```
# Suppression de la colonne 'class'

iris.drop(['class'], axis=1, inplace=True)

# Création d'un tableau numpy 2D à partir de la dataframe iris

iris_array = iris.values

# Création d'un pipeline

pipe_iris = Pipeline([('scaler', StandardScaler()),
                      ('pca', PCA(n_components=0.9))])

# Transformation des données

pipe_iris.fit(iris_array)

# Nombre de composantes nécessaire pour expliquer 90% de la variance

n_90 = len(pipe_iris.steps[1][1].components_)

print("Le nombre de composantes principales qu'il faut pour expliquer 90% de la variance totale contenue dans les données d'iris est :", n_90)
```

Le nombre de composantes principales qu'il faut pour expliquer 90% de la variance totale contenue dans les données d'iris est : 2

CONCLUSION

L'Analyse en Composantes Principales est une technique efficace pour réduire la dimensionnalité des grands ensembles de données. Elle permet de manipuler aisément ce genre de données. Dans ce projet, vous avez appris à implémenter l'ACP dans Python. Dans le projet suivant, nous appliquerons l'ACP afin de réduire la dimension d'un ensemble de données du web qui contient plus de 1000 variables. Dans le projet n°11, nous verrons comment l'ACP peut être utilisée pour améliorer la segmentation de la clientèle d'une entreprise.

PROJET 10 : CONSTRUCTION D'UN MODELE DE PREDICTION DES PUBLICITES SUR UN SITE WEB

INTRODUCTION

Les entreprises, de tout domaine, font face à une concurrence accrue et mondiale. De plus, Internet fait partie intégrale de notre vie quotidienne. Grâce à cet outil, on peut acheter en ligne, se former, vendre des produits, contacter des personnes dans le monde entier, etc. Dans ce contexte, il est absolument primordial pour une entreprise de développer sa présence sur le net. Elle doit aussi faire en sorte que les utilisateurs aient une très bonne expérience sur ses plateformes en ligne en général et sur son site web en particulier. Malheureusement, certains sites web ne permettent pas aux utilisateurs d'avoir une bonne expérience de navigation. Cela peut être dû à plusieurs facteurs comme :

- Des problèmes de rendus de pages ;
- Un mauvais graphisme qui ne donne pas envie de continuer sur le site ;
- Des informations qui ne sont pas placées au bon endroit et sont donc difficiles d'accès ;
- Un modèle de page trop complexe ;
- Une monétisation excessive ;
- Un contenu pauvre ;
- Des publicités agressives et des Pop-ups trop fréquents et mal placés qui irritent ;
- Etc.

Dans ce projet, nous allons nous focaliser sur l'un des facteurs favorisant la mauvaise expérience des utilisateurs d'un site internet : les publicités agressives et les pop-ups trop fréquents et mal placés. Nous allons appliquer des techniques de Machine Learning pour aider une entreprise à identifier les publicités potentielles et les pop-ups, puis à les éliminer avant même qu'ils n'apparaissent afin d'améliorer l'expérience des utilisateurs de son site web.

CONTEXTE ET OBJECTIFS DE L'ETUDE

Vous êtes Data Scientist dans une entreprise de commerce qui possède une boutique en ligne. Le Directeur Marketing vous fait part des nombreuses plaintes reçues d'utilisateurs par rapport à leur mauvaise expérience de navigation sur le site internet de la boutique due aux nombreuses publicités et à des pop-ups trop fréquents et mal placés qui apparaissent pendant la navigation. Lors de la réunion avec les différents collaborateurs intervenant sur le site, il a été décidé de

créer un moteur sur le serveur Web qui identifie les publicités potentielles ainsi que les pop-ups, puis les élimine avant même qu'ils n'apparaissent. La création de ce moteur passe par la construction d'un modèle qui prédit si quelque chose est une publicité ou non. Il s'agit donc d'une tâche de classification binaire.

En tant que Data Scientist de l'entreprise, vous êtes chargé de créer ce modèle. Pour ce faire, vous disposez d'un [ensemble de données](#)³⁸ qui contient un ensemble de publicités possibles sur une variété de pages Web du site de la boutique. Cet ensemble de données a également été étiqueté, chaque annonce possible ayant une étiquette indiquant s'il s'agit réellement d'une publicité ou non (voir [description de la base de données](#))³⁹.

LIBRAIRIES

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split

from sklearn.decomposition import PCA

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_report
```

DONNEES

```
# Importation et affichage des données

df = pd.read_csv('https://raw.githubusercontent.com/JosueAfoud
a/Internet-advertisements/master/ad.data', header=None)

print(df.info())

print('\n')

df.head()
```

La dataframe contient 3279 observations (lignes) et 1559 variables (colonnes). La dernière colonne est notre variable cible et les autres les variables indépendantes.

Vous pouvez consulter [ici](#)⁴⁰ les noms des variables. Le jeu de données contient 3279 lignes et 1559 variables.

```
# Variables indépendantes  
  
X = df.loc[:, 0:1557]  
  
# Variable dépendante  
  
y = df[1558]  
  
# Valeurs distinctes de la variable dépendante  
  
y.unique()  
  
array(['ad.', 'nonad.'], dtype=object)
```

La variable dépendante est une variable binaire qui indique s'il s'agit d'une publicité (*ad*) ou non (*nonad*).

NETTOYAGE DES DONNEES

```
# Regardons un échantillon aléatoire de 10 lignes de X  
  
X.sample(10)
```

On remarque la présence des symboles '?' indiquant certainement des valeurs manquantes. De plus, on dirait que ces '?' apparaissent uniquement au niveau des 3 premières colonnes. Vérifions le type des données dans chaque variable.

```
# Type des données de chaque variable  
  
df.dtypes
```

```

0      object
1      object
2      object
3      object
4      int64
       ...
1554    int64
1555    int64
1556    int64
1557    int64
1558    object
Length: 1559, dtype: object

```

C'est normal que les valeurs de la variable dépendante soient de type *object* mais les variables indépendantes devraient toutes être de type numérique (*float*). Les variables indépendantes de type *object* le sont à cause de la présence des symboles '?'. Pour être sûr de ne rien oublier, nous allons remplacer ces symboles par *NaN* au niveau de toutes les colonnes pour spécifier qu'elles sont des valeurs manquantes. Dans un premier temps, le remplacement sera effectué au niveau des 4 premières colonnes et dans un second temps au niveau du reste des colonnes.

```

# Remplacement des '?' par des NaN au niveau des 4 premières colonnes
for i in range(4):
    X[i] = X[i].str.replace('?', 'nan').values.astype('float')

# Remplacement des '?' par des NaN au niveau du reste des colonnes
for i in range(4, 1558):
    X[i] = X[i].replace('?', 'nan').values.astype('float')

```

```
# Affichage de X
```

```
X.sample(10)
```

Y a-t-il des valeurs manquantes dans la dataframe ?

```
# Nombre de valeurs manquantes par colonne
```

```
X.isna().sum()
```

```
0          903
1          901
2          910
3         1246
4            0
...
1553        0
1554        0
1555        0
1556        0
1557        0
Length: 1558, dtype: int64
```

Remplaçons maintenant les valeurs manquantes (NaN) au niveau de chaque variable par la moyenne de la variable

```
# Remplacement des valeurs manquantes
```

```
for i in range(1558):
```

```
    X[i] = X[i].fillna(X[i].mean(axis=0)) # Par défaut axis=0
```

```
# Vérification
```

```
X.isna().sum()
```

```

0      0
1      0
2      0
3      0
4      0
..
1553   0
1554   0
1555   0
1556   0
1557   0
Length: 1558, dtype: int64

```

Il n'y a plus de valeurs manquantes au niveau des variables.

PREPARATION DES DONNEES POUR LA MODELISATION

❖ NORMALISATION

```

# Création d'un tableau numpy à partir de la dataframe X
X_array = X.values

# Normalisation des données
scaler = MinMaxScaler()

X_array_scaled = scaler.fit_transform(X_array)

X_array_scaled.shape

(3279, 1558)

```

❖ REDUCTION DE LA DIMENSIONNALITE

Le véritable défi que pose notre dataframe est sa très grande dimensionnalité. Nous allons donc appliquer l'ACP afin de réduire cette dimensionnalité avant de passer les données dans un algorithme de classification binaire.

```

# Création d'une instance de la classe PCA

model_pca = PCA()

# Application aux données

model_pca.fit(X_array_scaled)

# Visualisation du pourcentage cumulé de variance expliquée

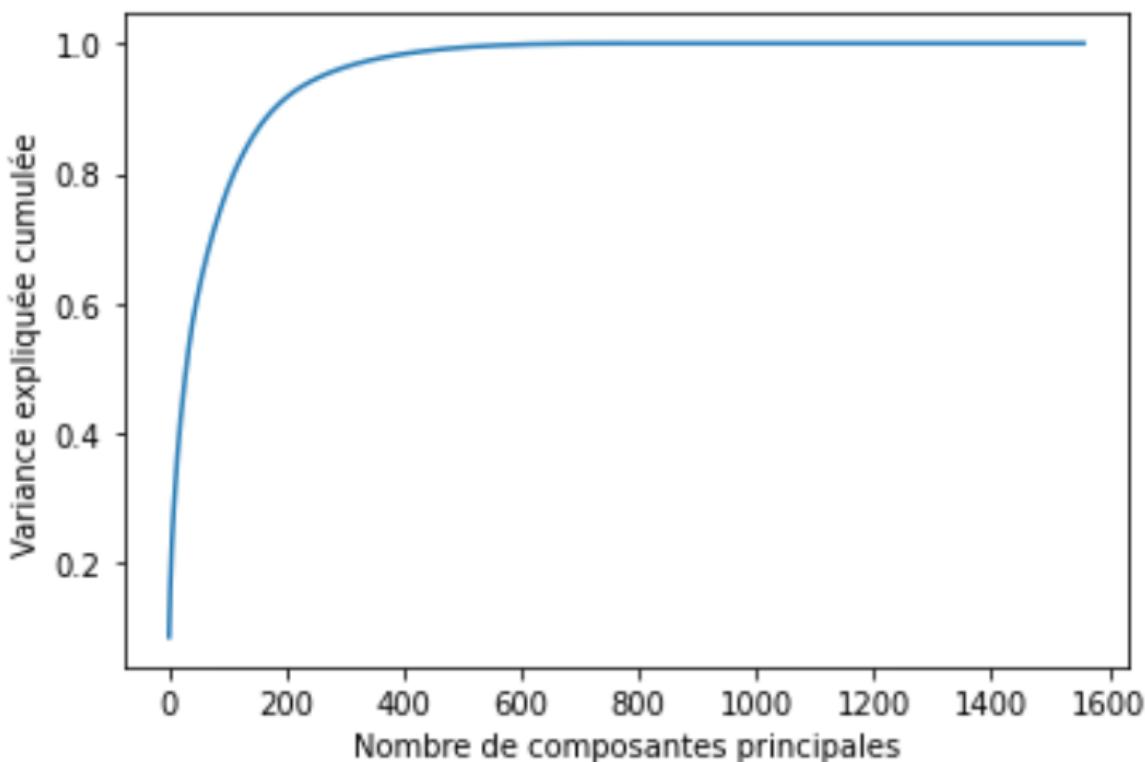
plt.plot(np.cumsum(model_pca.explained_variance_ratio_))

plt.xlabel('Nombre de composantes principales')

plt.ylabel('Variance expliquée cumulée')

plt.show()

```



Selon le graphique ci-dessus, les 300 premières composantes principales expliquent pratiquement la totalité de la variance des données. Les 200 premières composantes principales expliquent 80% de la variance totale des données. Nous décidons donc de réduire la dimension de notre ensemble de données en prenant 250 comme nombre de composantes principales.

```

# PCA avec 250 composantes principales

pca = PCA(n_components=250)

# Transformation des données

X_array_reduced = pca.fit_transform(X_array_scaled)

# Dimension du nouveau jeu de données

X_array_reduced.shape

(3279, 250)

```

❖ DIVISION DES DONNEES EN DONNEES D'ENTRAINEMENT ET DE D'EVALUATION DU MODELE

```

# Données d'entraînement et d'évaluation du modèle

seed = 1111

test_size = 0.3

y_array = y.values

X_train, X_test, y_train, y_test = train_test_split(X_array_reduced, y_array, stratify = y_array, test_size = test_size, random_state = seed)

print(X_train.shape)

print(y_train.shape)

print(X_test.shape)

print(y_test.shape)

```

MODELISATION PAR REGRESSION LOGISTIQUE

Passons maintenant à l'étape tant attendue de la création d'un modèle qui prédit si quelque chose est une publicité ou non.

```

# Création d'une instance de la classe LogisticRegression()

lr = LogisticRegression(random_state=seed)

# Entraînement du modèle

lr.fit(X_train, y_train)

#Score du modèle

print("Score sur les données d'entraînement :", round(lr.score(X_train, y_train), 2))

print("Score sur les données d'évaluation :", round(lr.score(X_test, y_test), 2))

```

Score sur les données d'entraînement : 0.98
 Score sur les données d'évaluation : 0.97

Le modèle donne une précision globale de 98% sur les données d'entraînement et semble bien se généraliser (pas de phénomène de surapprentissage) avec un score de 97% sur les données de test. Passons maintenant aux prédictions.

```

# Prédictions

yhat = lr.predict(X_test)

# Rapport de classification

print(classification_report(yhat, y_test))

```

	precision	recall	f1-score	support
ad.	0.80	0.96	0.87	114
nonad.	1.00	0.97	0.98	870
accuracy			0.97	984
macro avg	0.90	0.97	0.93	984
weighted avg	0.97	0.97	0.97	984

Selon ce rapport de performance globale du modèle, 97% (*accuracy*) des observations des données d'évaluation ont été correctement classées. 96% des publicités (classe *ad*) ont été correctement classés comme tel (paramètre *recall*) ce qui veut dire que seulement 4% des publicités ont été mal classés comme étant des non-publicités : c'est un risque de mauvaise expérience de navigation dû aux publicités de l'ordre de 3%.

Par ailleurs 97% des non-publicités (classe *nonad*) ont été classé comme tel ce qui veut dire qu'environ 3% des non-publicités ont été mal classés comme étant des publicités : c'est un risque de voir certaines informations, qui ne sont pas des publicités, être supprimé par le modèle alors qu'elles étaient destinées aux utilisateurs. Ce risque est acceptable (3%).

CONCLUSION

A travers ce projet, nous avons montré comment le Machine Learning peut être mis en œuvre pour aider une entreprise à être efficace dans sa présence en ligne. Précisément, nous avons dans un premier temps réduit la dimensionnalité d'un ensemble de données puis créer un modèle qui prédit si une chose est une publicité ou pas. Le modèle obtenu est très performant.

Pour des fins de comparaison, vous pouvez construire un modèle avec les variables originelles et/ou avec d'autres algorithmes de classification (à essayer avec des ordinateurs puissants 😊).

PROJET 11 : COMBINAISON DES TECHNIQUES DE CLUSTERING ET D'ACP POUR LA SEGMENTATION DE CLIENTELE

INTRODUCTION

La segmentation de la clientèle d'une entreprise est une pratique de division des clients en groupes d'individus ayant les mêmes caractéristiques. Cette stratégie permet de comprendre le comportement des clients, sur la base de leurs données historiques, et donc d'utiliser efficacement les ressources allouées au Marketing. La segmentation est effectuée par une technique d'apprentissage automatique non supervisée appelée Clustering. L'algorithme de clustering le plus populaire est KMeans. Dans ce projet, nous montrerons comment utiliser cet algorithme pour segmenter les clients d'une entreprise. Nous montrerons également comment combiner KMeans et l'Analyse en Composantes Principales afin d'améliorer la qualité de la segmentation.

LIBRAIRIES

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

import pickle
```

DONNEES

```
# Importation des données

df = pd.read_csv('https://raw.githubusercontent.com/JosueAfoud
a/KMeans-and-PCA/master/segmentation%20data.csv', index_col=0)
```

```
df.head()
```

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
ID							
1000000001	0		0	67	2	124670	1
1000000002	1		1	22	1	150773	1
1000000003	0		0	49	1	89210	0
1000000004	0		0	45	1	171565	1
1000000005	0		0	53	1	149031	1

```
# Dimension des données
```

```
print('La dataframe contient', df.shape[0], 'ligne (chaque ligne représente un client) et', df.shape[1], 'colonnes')
```

DESCRIPTION DE CHAQUE VARIABLE

- *Sex*

1: Female, 0: Male

- *Marital Status*

0: single, 1: non-single

- *Education*

0: other/unknown, 1: high-school, 2:university, 3:graduate school

- *Occupation*

0: unemployed, 1: skilled, 2: highly qualified

- *settlement size*

0: small, 1: mid-sized, 2:big

```
# Données manquantes
```

```
df.isna().sum()
```

```
Sex          0
Marital status 0
Age          0
Education     0
Income         0
Occupation    0
Settlement size 0
dtype: int64
```

Il n'y a pas de valeurs manquantes dans les données. Vérifions que les formats de stockage des valeurs au niveau de chaque variable :

```
df.dtypes
```

```
Sex            int64
Marital status int64
Age            int64
Education      int64
Income          int64
Occupation     int64
Settlement size int64
dtype: object
```

Les valeurs sont stockées dans le bon format.

ANALYSE EXPLORATOIRE DES DONNEES

```
# Résumé statistique
```

```
df.describe()
```

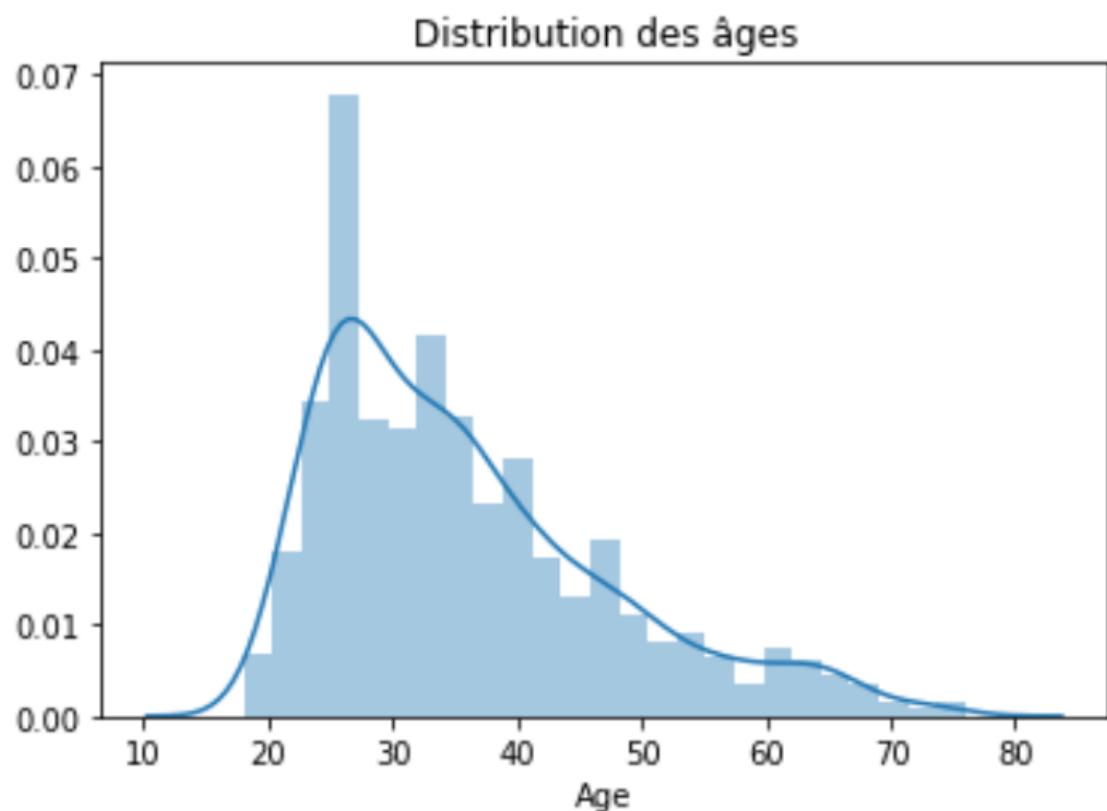
L'âge est compris entre 18 et 76 ans. L'âge médian est de 33 ans. Le revenu est compris entre 35832 et 309364 dollars. Le revenu médian est de 115548,5 \$.

```
# Distribution de la variable 'Age'
```

```
sns.distplot(df["Age"])
```

```
plt.title('Distribution des âges')
```

```
plt.show()
```

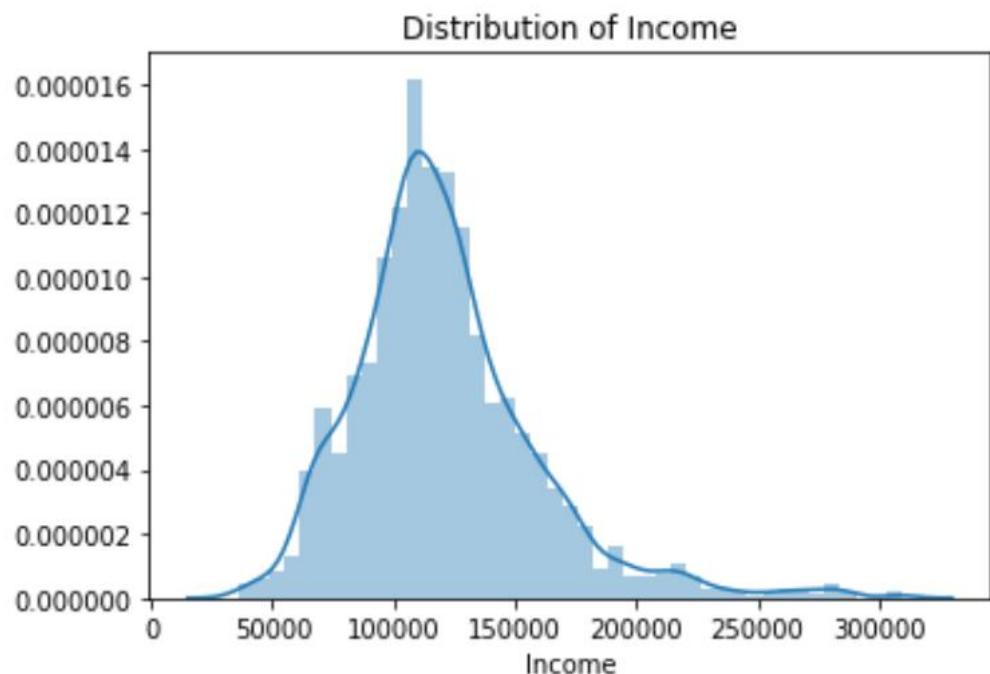


```
# Distribution de la variable 'Income'
```

```
sns.distplot(df["Income"])
```

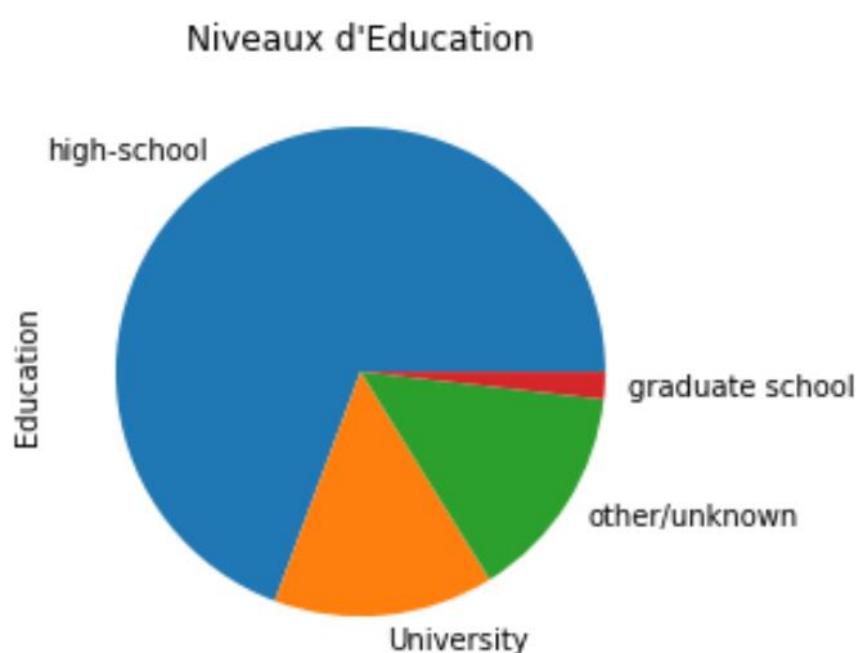
```
plt.title('Distribution of Income')
```

```
plt.show()
```



```
# Education
```

```
df[\"Education\"].replace({0:'other/unknown', 1:'high-\nschool', 2:'University', 3:'graduate school'}).value_counts(no\nrmalize=True).plot(kind='pie')\n\nplt.title("Niveaux d'Education")\n\nplt.show()
```



```
# Occupation

df["Occupation"].replace({0:'unemployed', 1:'skilled', 2:'highly-qualified'}).value_counts(normalize=True).plot(kind='pie')

plt.title("Niveaux d'occupations")

plt.show()
```

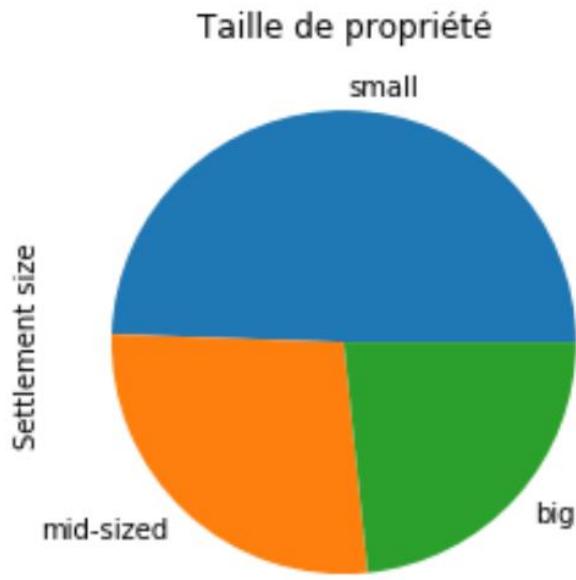


```
# Settlement size

df["Settlement size"].replace({0:'small', 1:'mid-sized', 2:'big'}).value_counts(normalize=True).plot(kind='pie')

plt.title('Taille de propriété')

plt.show()
```



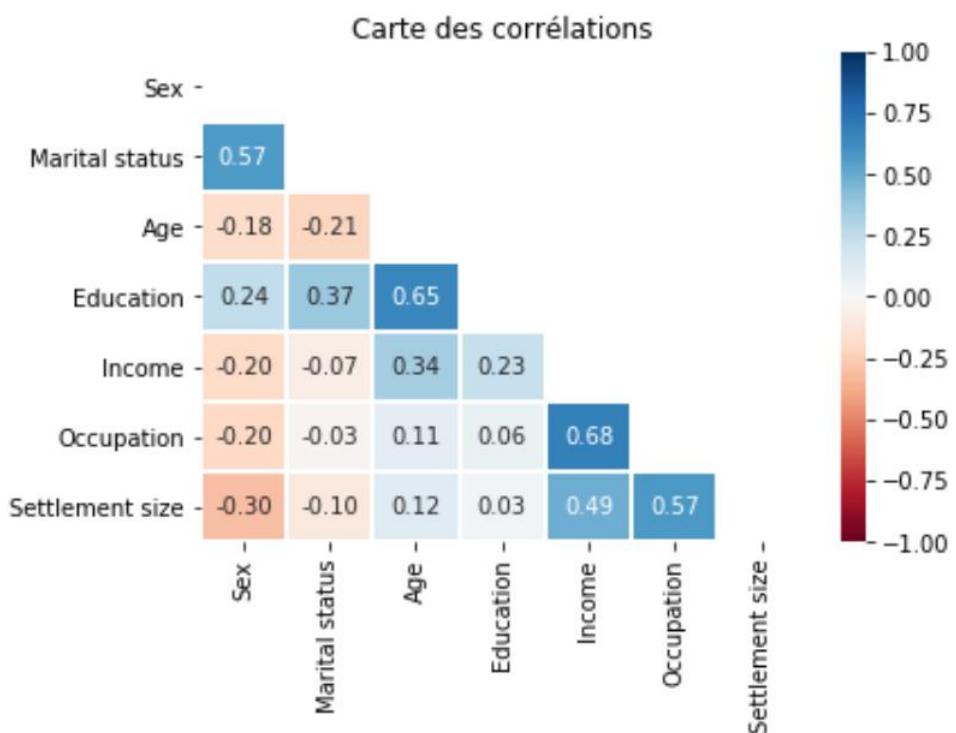
Visualisons à présent les corrélations entre variables :

```
mask = np.triu(np.ones_like(df.corr(), dtype=bool))

sns.heatmap(df.corr(), mask=mask, center=0, cmap='RdBu', linewidths=1, annot=True, fmt=".2f", vmin=-1, vmax=1)

plt.title('Carte des corrélations')

plt.show()
```



SEGMENTATION AVEC KMEANS

```
# Standardisation des données

scaler = StandardScaler()

df_std = scaler.fit_transform(df)
```

Afin de trouver le meilleur nombre de cluster par la méthode du coude, nous allons exécuter l'algorithme pour k allant de 1 à 10. De plus, nous choisirons KMeans++ comme [méthode d'initialisation](#).⁴¹ En effet KMeans++ sélectionne de manière intelligente les centres de cluster initiaux pour le clustering afin d'accélérer la convergence de l'algorithme.

WCSS est la somme des carrés des distances de chaque point de données dans tous les clusters à leurs centres de gravité respectifs. L'idée est de pouvoir minimiser cette somme (Inertie du système).

```
# Clustering

seed = 111

wcss = []

for i in range(1,11):

    kmeans = KMeans(n_clusters = i, init = 'k-
means++', random_state = seed)

    kmeans.fit(df_std)

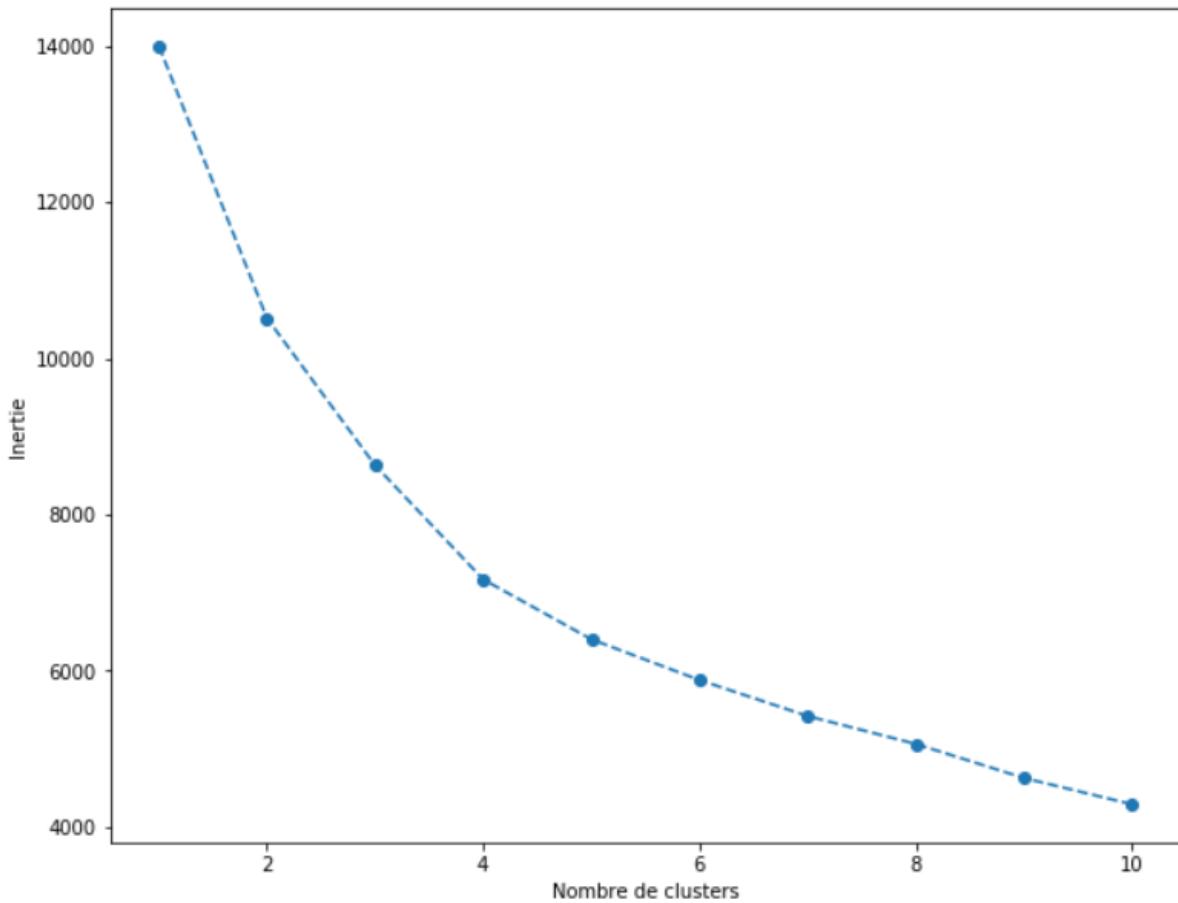
    wcss.append(kmeans.inertia_)

# Méthode du coude pour le choix du nombre de clusters

plt.figure(figsize = (10,8))

plt.plot(range(1, 11), wcss, marker = 'o', linestyle = '--')

plt.xlabel('Nombre de clusters')
plt.ylabel('Inertie')
plt.show()
```



Nous choisirons 4 comme nombre de clusters. Exécutons à nouveau KMeans pour cette valeur fixe de k.

```
kmeans = KMeans(n_clusters = 4, init = 'k-means++', random_state = seed)

kmeans.fit(df_std)
```

Il est temps d'explorer les résultats de notre segmentation. Tout d'abord, créons un nouveau bloc de données avec les entités d'origine et ajoutons une nouvelle colonne avec les clusters attribués pour chaque point.

```
df_segm_kmeans = df.copy()

df_segm_kmeans['Cluster'] = kmeans.labels_

df_segm_kmeans.head()
```

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size	Cluster	
ID									
100000001	0		0	67	2	124670	1	2	3
100000002	1		1	22	1	150773	1	2	0
100000003	0		0	49	1	89210	0	0	2
100000004	0		0	45	1	171565	1	1	1
100000005	0		0	53	1	149031	1	1	1

Centroïde de chaque cluster

```
centroid = df_segm_kmeans.groupby(['Cluster']).mean()
```

```
centroid
```

Cluster	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
0	0.853901	0.997163	28.963121	1.068085	105759.119149	0.634043	0.422695
1	0.029825	0.173684	35.635088	0.733333	141218.249123	1.271930	1.522807
2	0.352814	0.019481	35.577922	0.746753	97859.852814	0.329004	0.043290
3	0.501901	0.692015	55.703422	2.129278	158338.422053	1.129278	1.110266

Sur la base des caractéristiques de chaque centroïde, l'équipe Marketing peut attribuer un nom à chaque groupe de clients. Par exemple, nous remarquons que :

- Le dernier groupe (cluster 3) concerne les clients qui ont en moyenne 55 ans (les plus âgés), un diplôme universitaire, un emploi de direction, les revenus les plus élevés et qui vivent dans des villes de taille moyenne. On peut donc dire que c'est le groupe des personnes aisées.
- Le deuxième groupe (cluster 2) concerne les clients qui ont une moyenne d'âge de 36 ans, qui n'ont pas de diplôme d'études secondaires, sont au chômage, aux revenus les plus faibles et vivent dans les petites villes. Ce groupe est tout le contraire du cluster 3. On peut donc dire que c'est le groupe de personnes ayant le moins d'opportunités.

Visualisons les clusters :

```

# Chaque point de notre ensemble de données est tracé avec la
couleur du cluster auquel il a été attribué.

x_axis = df_segm_kmeans['Age']

y_axis = df_segm_kmeans['Income']

hue = df_segm_kmeans['Cluster']

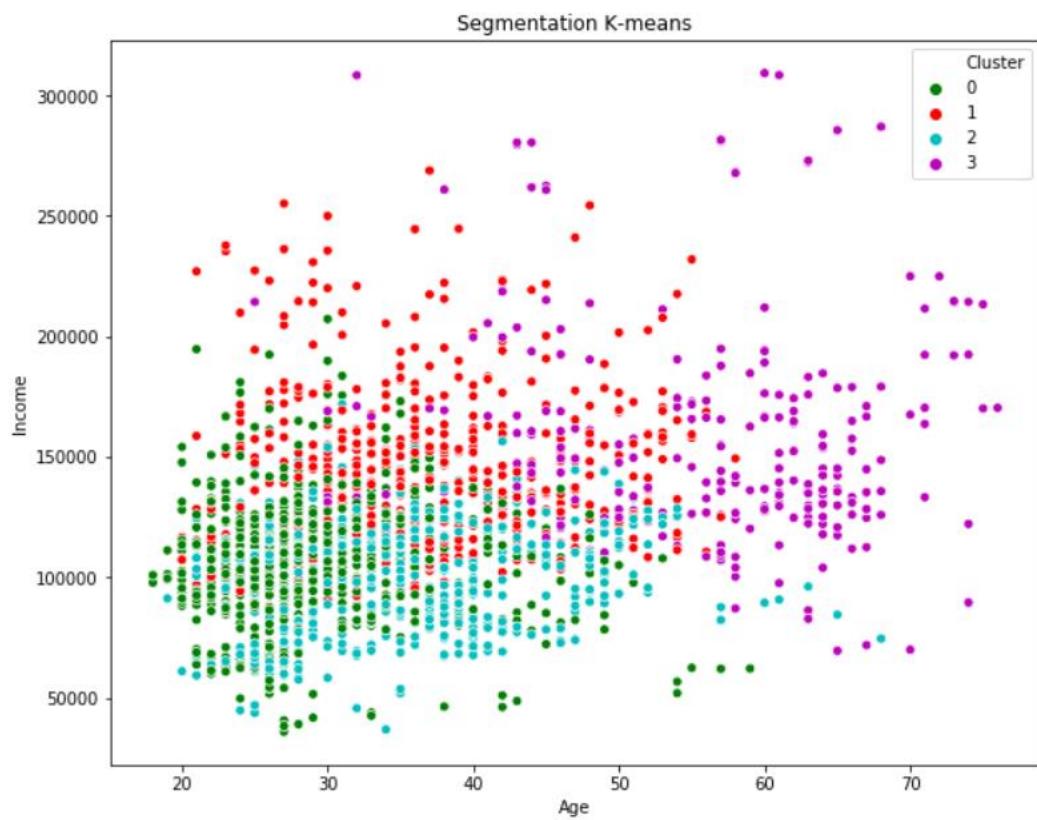
plt.figure(figsize = (10, 8))

sns.scatterplot(x_axis, y_axis, hue = hue, palette = ['g', 'r',
, 'c', 'm'])

plt.title('Segmentation K-means')

plt.show()

```



Comme vous le voyez bien sur le graphique, les points sont très mélangés et on ne distingue pas bien chaque cluster. Dans la section suivante, nous essaierons d'améliorer notre segmentation en combinant Analyse en Composantes Principales et KMeans.

SEGMENTATION AVEC ACP + KMEANS

```
# Création d'un modèle PCA (Principal Component Analysis)

pca = PCA()

# Ajustement du modèle PCA aux données standardisées

pca.fit(df_std)

# Variance expliquée par chaque composante

print(pca.explained_variance_ratio_)

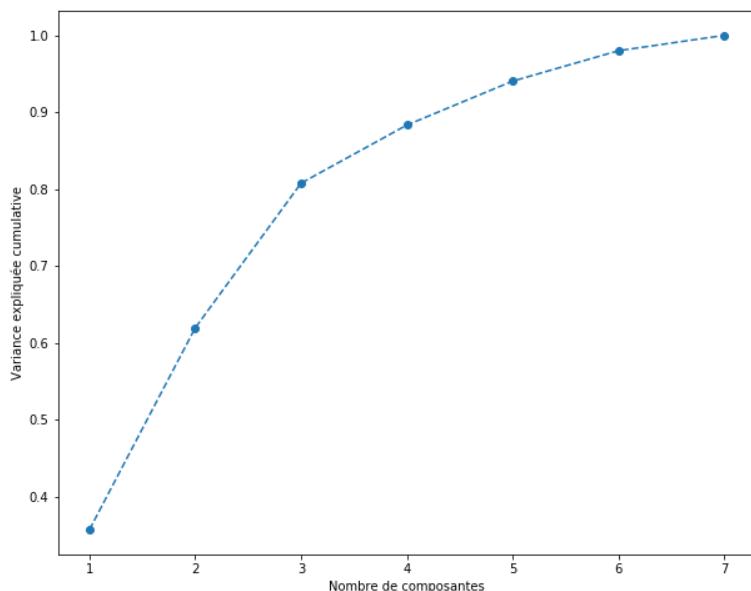
[0.35696328 0.26250923 0.18821114 0.0755775  0.05716512 0.03954794
 0.02002579]
```

Par défaut, le nombre de composantes dans l'algorithme d'ACP est égal au nombre de variables de la datafram originelle. Traçons un graphique de la variance cumulée expliquée en fonction du nombre total de composantes. Notre objectif est de garder 80% de toute la variance.

```
plt.figure(figsize = (10,8))

plt.plot(range(1,8), pca.explained_variance_ratio_.cumsum(), marker = 'o', linestyle = '--')

plt.xlabel('Nombre de composantes')
plt.ylabel('Variance expliquée cumulative')
plt.show()
```



Selon le graphique précédent, les trois premières composantes expliquent 80% de la variance.

```
# ACP avec 3 composantes

pca3 = PCA(n_components = 3)

pca3.fit(df_std)

df_pca3_comp = pd.DataFrame(data = pca3.components_,
                             columns = df.columns.values,
                             index = ['Component 1', 'Component
2', 'Component 3'])
df_pca3_comp
```

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
Component 1	-0.314695	-0.191704	0.326100	0.156841	0.524525	0.492059	0.464789
Component 2	0.458006	0.512635	0.312208	0.639807	0.124683	0.014658	-0.069632
Component 3	-0.293013	-0.441977	0.609544	0.275605	-0.165662	-0.395505	-0.295685

Nous pouvons maintenant refaire une segmentation avec les données transformées par l'ACP.

```
scores_pca3 = pca3.transform(df_std)

wcss = []

for i in range(1,11):

    kmeans_pca3 = KMeans(n_clusters = i, init = 'k-
means++', random_state = seed)
    kmeans_pca3.fit(scores_pca3)

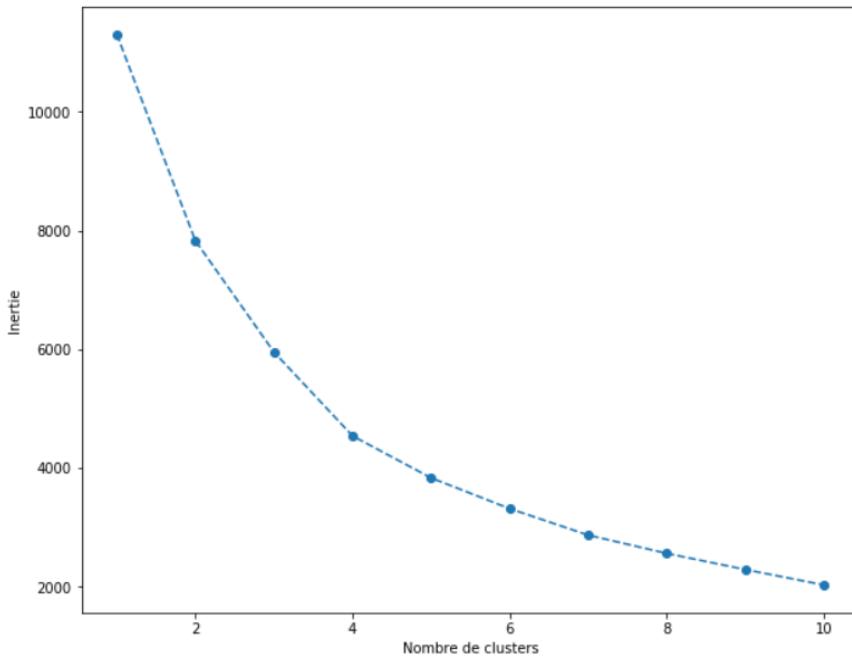
    wcss.append(kmeans_pca3.inertia_)

# Méthode du coude pour le choix du nombre de clusters

plt.figure(figsize = (10,8))

plt.plot(range(1, 11), wcss, marker = 'o', linestyle = '--')

plt.xlabel('Nombre de clusters')
plt.ylabel('Inertie')
plt.show()
```



Selon la méthode du coude, nous choisissons 4 comme nombre de clusters.

```
kmeans_pca3 = KMeans(n_clusters = 4, init = 'k-
means++', random_state = seed)

kmeans_pca3.fit(scores_pca3)

# Création d'une dataframe avec les variables d'origines,
# les composantes et les étiquettes de clusters

df_segm_pca3_kmeans = pd.concat([df.reset_index(drop = True),
pd.DataFrame(scores_pca3)], axis = 1)

df_segm_pca3_kmeans.columns.values[-
3:] = ['Component 1', 'Component 2', 'Component 3']

df_segm_pca3_kmeans['Cluster PCA'] = kmeans_pca3.labels_

df_segm_pca3_kmeans.head()
```

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size	Component 1	Component 2	Component 3	Cluster PCA
0	0		0	67	2	124670	1	2	2.514746	0.834122	2.174806
1	1		1	22	1	150773	1	2	0.344935	0.598146	-2.211603
2	0		0	49	1	89210	0	0	-0.651063	-0.680093	2.280419
3	0		0	45	1	171565	1	1	1.714316	-0.579927	0.730731
4	0		0	53	1	149031	1	1	1.626745	-0.440496	1.244909

```
# Centroides
```

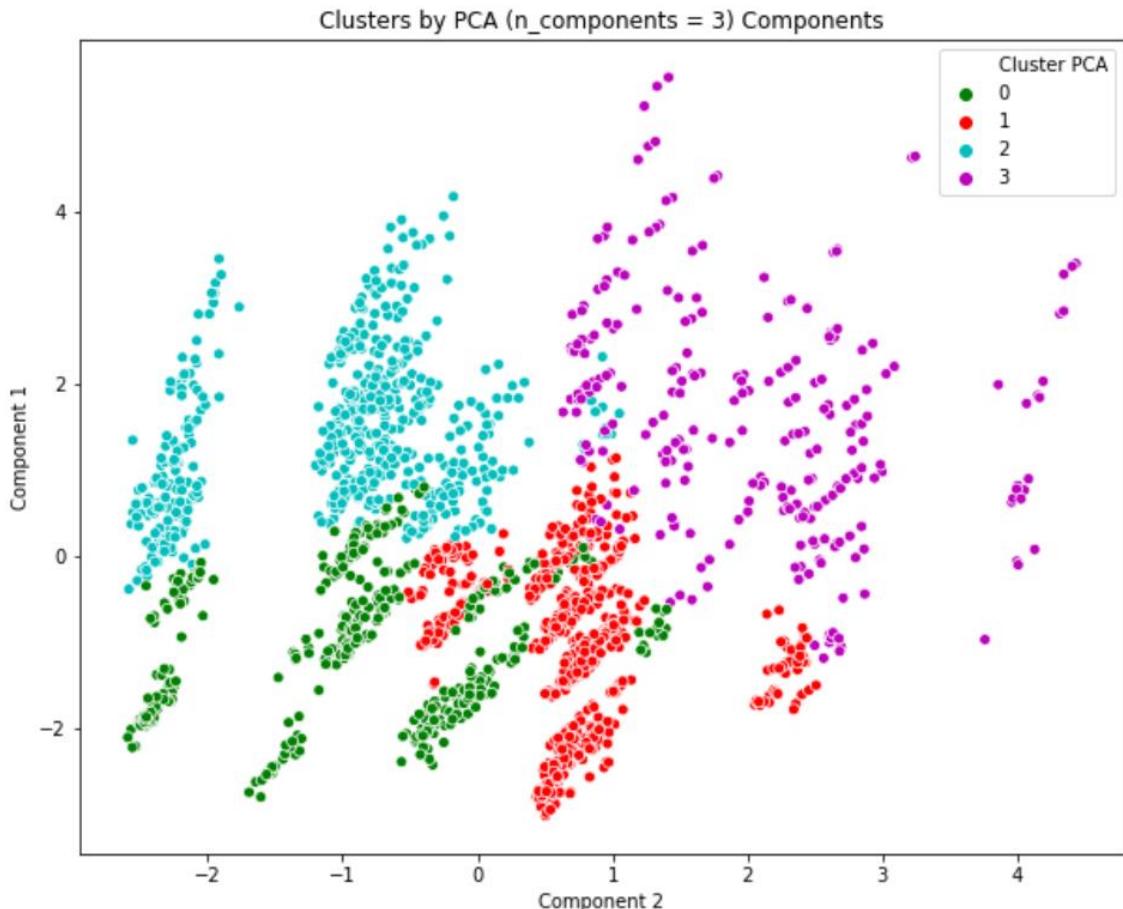
```
centroid_pca3 = df_segm_pca3_kmeans.groupby(['Cluster PCA']).mean()
```

```
centroid_pca3
```

Cluster PCA	Sex	Marital status	Age	Education	Income	Occupation	Settlement size	Component 1	Component 2	Component 3
0	0.306522	0.095652	35.313043	0.760870	93692.567391	0.252174	0.039130	-1.046406	-0.902963	1.003644
1	0.900289	0.965318	28.878613	1.060694	107551.500000	0.677746	0.440751	-1.107019	0.703776	-0.781410
2	0.027444	0.168096	35.737564	0.734134	141525.826758	1.267581	1.480274	1.372663	-1.046172	-0.248046
3	0.505660	0.690566	55.679245	2.128302	158019.101887	1.120755	1.101887	1.687328	2.031200	0.844039

Visualisons les clusters :

```
x_axis = df_segm_pca3_kmeans['Component 2']  
  
y_axis = df_segm_pca3_kmeans['Component 1']  
  
hue = df_segm_pca3_kmeans['Cluster PCA']  
  
plt.figure(figsize = (10, 8))  
  
sns.scatterplot(x_axis, y_axis, hue = hue, palette = ['g', 'r',  
'c', 'm'])  
  
plt.title('Clusters by PCA (n_components = 3) Components')  
  
plt.show()
```



Les points sont mieux regroupés au niveau de chaque cluster ce qui permet de mieux distinguer chaque cluster. Cette segmentation est meilleure que la première faite avec uniquement l'algorithme KMeans.

CONCLUSION

La segmentation des clients est très importante pour toute entreprise et permet d'analyser le comportement des clients. L'algorithme K-means combiné avec PCA est un outil puissant et facile à utiliser pour effectuer une bonne segmentation. Il peut également être utilisé dans des domaines tels que la santé, la recherche scientifique, etc.

PROJET 12 : MODELISATION DES COURS BOURSIERS

INTRODUCTION

Avez-vous l'intention d'acheter des actions dans le marché boursier ? Si vous possédez des actions d'une entreprise dont les profits augmentent alors vous gagnerez de l'argent. En revanche, si cette entreprise a une mauvaise performance et que ses bénéfices chutent alors vous perdrez de l'argent. Investir en bourse est tout un art. Il faut pouvoir acheter les actions de la 'bonne entreprise' au bon moment. Heureusement, il est possible de prédire les futurs prix des actions d'une société sur la base de données historiques. Cela permet aux investisseurs d'avoir une idée sur les entreprises intéressantes et le moment favorable pour acheter des actions.

Dans ce projet, nous construirons un modèle [ARIMA](#)⁴² de prévisions des futurs prix de l'indice du [Dow Jones](#)⁴³ en fonction des prix historiques de cet indice. Il s'agit ici d'un projet de Modélisation d'une [série temporelle](#)⁴⁴.

LIBRAIRIES

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from statsmodels.tsa.stattools import adfuller

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

from statsmodels.tsa.statespace.sarimax import SARIMAX

import datetime

import warnings
warnings.filterwarnings("ignore")
```

DONNEES

Pour réaliser ce projet, nous utiliserons les indices du Dow Jones. Pour collecter les données, nous utiliserons [Yahoo Finance](#)⁴⁵. Nous avons téléchargé au format csv les prix historiques

journaliers allant du 03 décembre 2007 au 03 décembre 2017 soit 10 ans de données sur les indices du Dow Jones. Vous pouvez retrouver directement les données utilisées dans ce projet via ce [lien](#)⁴⁶.

```
# Importation des données
```

```
dow_jones = pd.read_csv('https://raw.githubusercontent.com/Jos  
ueAfouda/Stock-Market-Price-  
Prediction/master/dow_jones_2007_2017.csv')
```

```
dow_jones.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2007-12-03	13368.219727	13407.240234	13296.280273	13314.570313	13314.570313	212170000
1	2007-12-04	13311.240234	13316.280273	13237.589844	13248.730469	13248.730469	204940000
2	2007-12-05	13244.009766	13460.240234	13244.009766	13444.959961	13444.959961	256800000
3	2007-12-06	13445.849609	13632.900391	13426.179688	13619.889648	13619.889648	197270000
4	2007-12-07	13618.269531	13668.099609	13601.360352	13625.580078	13625.580078	179420000

```
# Structure de la dataframe dow_jones
```

```
dow_jones.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 2519 entries, 0 to 2518  
Data columns (total 7 columns):  
 #   Column      Non-Null Count  Dtype     
 ---  --          --          --          --            
 0   Date        2519 non-null    object    
 1   Open         2519 non-null    float64  
 2   High         2519 non-null    float64  
 3   Low          2519 non-null    float64  
 4   Close        2519 non-null    float64  
 5   Adj Close    2519 non-null    float64  
 6   Volume       2519 non-null    int64  
dtypes: float64(5), int64(1), object(1)  
memory usage: 137.9+ KB
```

L'ensemble des données contient 2519 observations et 7 variables :

- **Date** : date ;
- **Open** : prix de l'indice du Dow Jones à l'ouverture du marché ;
- **High** : prix le plus élevé de l'indice atteint dans la journée ;
- **Low** : prix le plus bas de l'indice atteint dans la journée ;
- **Close** : prix de l'indice à la clôture de la journée ;
- **Adj Close** : prix de clôture ajusté. Le prix de clôture ajusté tient compte de divers facteurs et, sur la base de ceux-ci, génère la valeur réelle des actions de la société. Ici, nous regardons la valeur de l'indice DJIA (*Dow Jones Industrial Average*) donc, la plupart du temps, le prix de clôture et le prix de clôture ajusté sont les mêmes ;
- **Volume** : nombre d'indices négociés dans la journée.

Le prix de l'indice à l'ouverture d'une journée est proche du prix de clôture au jour précédent. Donc, nous n'allons pas considérer la variable *Open*. De plus, avec les données dont nous disposons, nous ne pouvons pas connaître les moments de la journée où le prix le plus élevé et le prix le plus bas ont été atteint. Donc nous ne pourrons pas prédire ces prix. Notre objectif sera alors de prédire les prix ajustés de clôture pour les prochains jours de trading ce qui nous permettra d'avoir une compréhension globale de la tendance.

L'affichage des premières lignes de la datafarme *dow_jones* indique que les prix de clôture sont quasiment égaux aux prix ajustés de clôture. Nous partons du principe selon lequel les prix ajustés de clôture prennent en compte plusieurs facteurs important et donnent la valeur réelle de l'action de la société. C'est pour cela que nous choisissons de travailler avec la variable *Adj Close*.

```
# Dataframe simplifiée

df = dow_jones[['Date', 'Adj Close']]

# Colonne 'Date' au format DateTime

df['Date'] = pd.to_datetime(df['Date'])

# Définir la colonne 'Date' en tant qu'indices

df.set_index('Date', inplace = True)
```

```
# Affichage des 5 premières lignes de df
```

```
df.head()
```

Date	Adj Close
2007-12-03	13314.570313
2007-12-04	13248.730469
2007-12-05	13444.959961
2007-12-06	13619.889648
2007-12-07	13625.580078

```
# Affichage des 5 dernières lignes de df
```

```
df.tail()
```

Date	Adj Close
2017-11-27	23580.779297
2017-11-28	23836.710938
2017-11-29	23940.679688
2017-11-30	24272.349609
2017-12-01	24231.589844

ANALYSE DE LA SERIE TEMPORELLE DES PRIX AJUSTES DE CLOTURE DU DOW JONES

```
# Résumé statistique
```

```
df.describe()
```

Adj Close	
count	2519.000000
mean	14389.746939
std	3806.313389
min	6547.049805
25%	11411.595215
50%	13471.219727
75%	17525.764649
max	24272.349609

Au cours de la période 2007-2017, le prix de l'indice a évolué entre 6547,05 et 24272,35 dollars. La moyenne et la médiane sont relativement proches ce qui indique que la distribution des prix suit probablement une loi normale

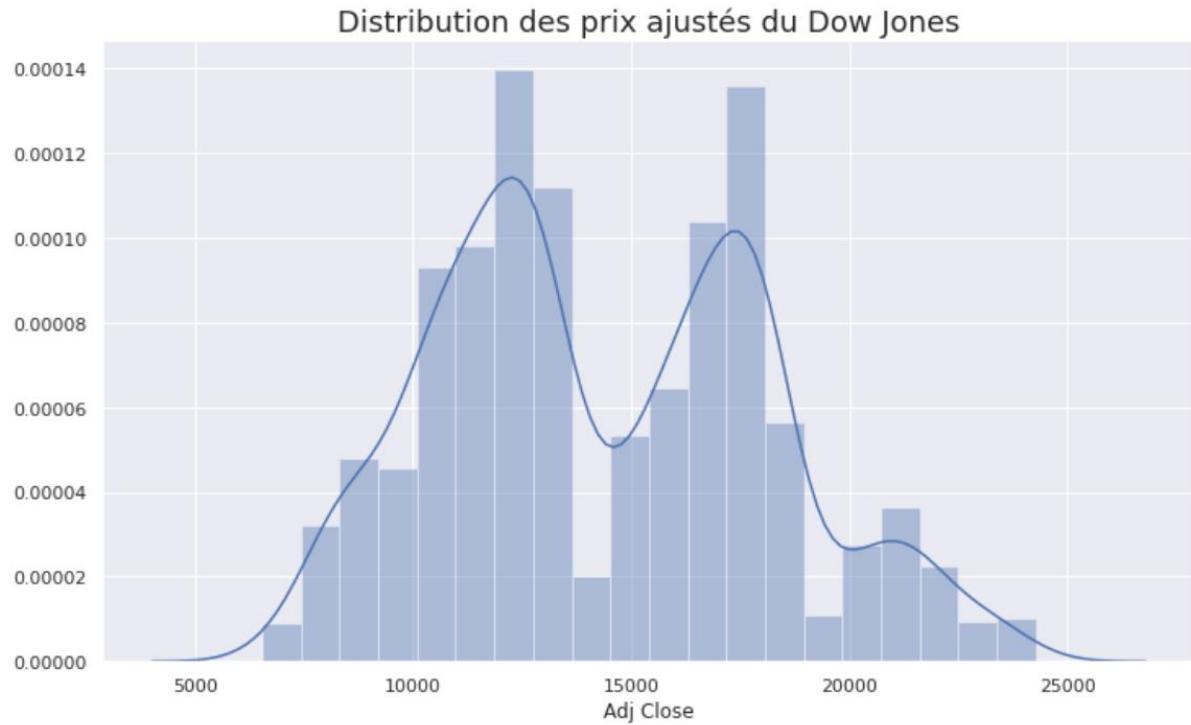
```
# Distribution des prix
```

```
sns.set(rc={'figure.figsize':(12,7)})
```

```
sns.distplot(df['Adj Close'])
```

```
plt.title("Distribution des prix ajustés du Dow Jones", fontsize = 18)
```

```
plt.show()
```



La distribution des prix du Dow Jones est bimodale indiquant qu'il y a deux pics avec une chute au milieu. La courbe de densité est sinusoïdale montrant une alternance de montée et de chute du prix de l'indice ce qui est plutôt commun aux cours des actions qui sont majoritairement des séries non-stationnaires. En effet les gens attendent que les prix des actions chutent pour acheter et espèrent que les prix augmentent dans un avenir plus ou moins proche afin de les vendre et de gagner un bénéfice.

```
# Visualisation de l'évolution du prix
```

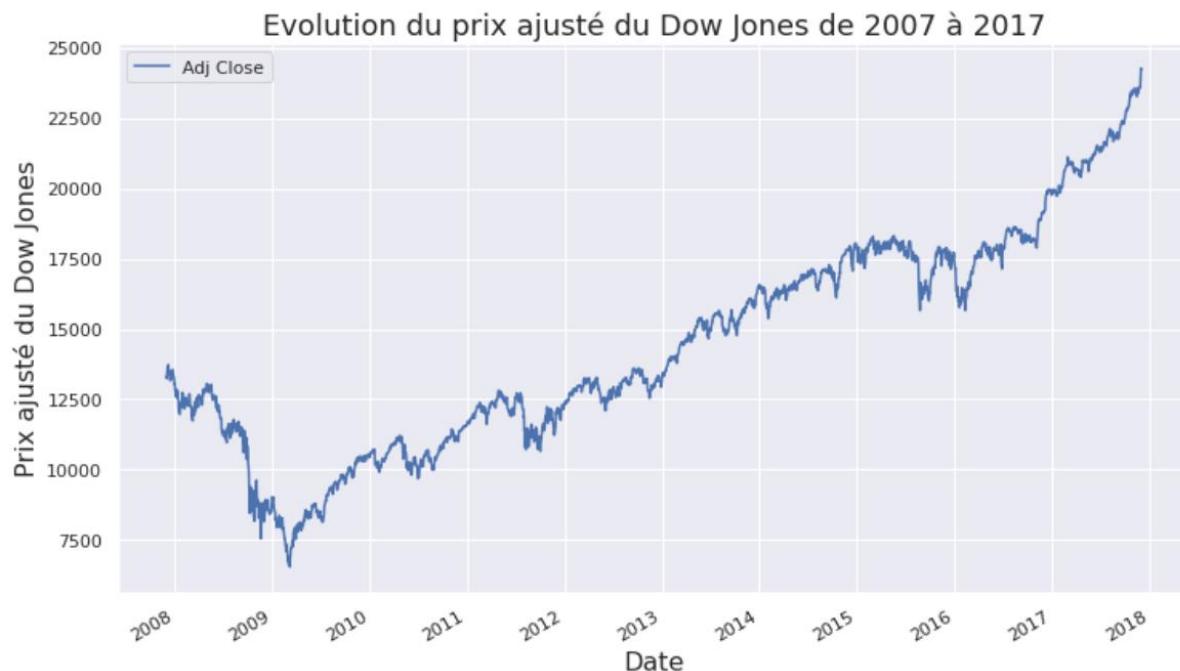
```
df.plot()
```

```
plt.title("Evolution du prix ajusté du Dow Jones de 2007 à 2017", fontsize = 18)
```

```
plt.xlabel ('Date', fontsize = 16)
```

```
plt.ylabel('Prix ajusté du Dow Jones', fontsize = 16)
```

```
plt.show()
```



Globalement, l'évolution du prix de l'indice entre 2007 et 2017 a une tendance haussière. Ceci nous indique également que la série n'est pas stationnaire.

STATIONNARITE

Est-ce que notre série temporelle est stationnaire ou non ? Avant de répondre à cette question, il faut comprendre pourquoi vérifie-t-on la stationnarité d'une série chronologique. En effet, si une série temporelle est stationnaire alors on peut utiliser les modèles [ARMA](#)⁴² pour prédire ses prochaines valeurs. Si la série n'est pas stationnaire alors il faudra la transformer d'abord en une série stationnaire avant de pouvoir utiliser un modèle ARMA.

Pour qu'une série temporelle soit stationnaire, il ne doit y avoir aucune tendance, ni aucun changement évident de variance ou de dynamique (la série doit avoir une variance constante et une dynamique de temps constante).

L'analyse de l'histogramme et de l'évolution de notre série temporelle nous montre qu'elle présente un changement de dynamique et aussi une tendance haussière. Alors elle n'est pas stationnaire. Nous devons donc la transformer en une série stationnaire. Mais avant de le faire, nous allons vérifier la non-stationnarité de notre série temporelle par un test statistique : le test de [Dicky-Fuller](#)⁴⁷. L'hypothèse nulle du test de Dicky-Fuller est que la série temporelle n'est pas stationnaire en raison de la tendance.

```

# Test de Dicky-Fuller

results = adfuller(df['Adj Close'])

print(type(results))

print(results)

<class 'tuple'>

(1.2543740484097001, 0.9963408448678397, 20, 2498, {'1%': -
3.4329705094097114, '5%': -2.8626977267304357, '10%': -
2.5673863028421136}, 31617.41265211667)

```

Comme vous le voyez, les résultats du test de Dicky-Fuller sont stockés dans un tuple.

- Le premier élément de ce tuple est la statistique de test. Plus ce nombre est négatif, plus il est probable que la série soit stationnaire.
- Le deuxième élément du tuple est la p-valeur du test. Si la p-valeur est inférieure à 0,05, nous rejetons l'hypothèse nulle et supposons donc que la série temporelle est stationnaire.
- Le dictionnaire qu'il y a dans le tuple stocke les valeurs critiques (seuils) qui correspondent à différentes p-valeur. Par exemple dans notre cas, si nous considérons une p-valeur inférieure ou égale à 0,05 (5%), la statistique de test devra être inférieure à -2,8627 avant de pouvoir rejeter l'hypothèse nulle et donc supposer que la série est stationnaire

La statistique de test est égale à 1,25 avec une p-valeur de 0,996. Nous acceptons donc l'hypothèse nulle selon laquelle la série n'est pas temporelle en raison de la tendance.

A partir de la visualisation et du résultat du test de Dicky-Fuller, nous pouvons conclure que notre série temporelle est non-stationnaire. Il est très important de toujours visualiser la série chronologie et de faire aussi le test avant de tirer une conclusion définitive.

Transformons à présent nos données en une série stationnaire. Une façon très courante de rendre une série chronologique stationnaire est de prendre sa différence. C'est-à-dire, de chaque valeur de la série soustraire la valeur précédente.

```

# Transformation en une série stationnaire

df_stat = df.diff()

```

```
# Affichage de df_stat
```

```
df_stat.head()
```

Date	Adj Close
2007-12-03	NaN
2007-12-04	-65.839844
2007-12-05	196.229492
2007-12-06	174.929687
2007-12-07	5.690430

```
df_stat.tail()
```

Date	Adj Close
2017-11-27	22.789063
2017-11-28	255.931641
2017-11-29	103.968750
2017-11-30	331.669921
2017-12-01	-40.759765

C'est tout à fait normal que la première valeur soit *NaN* (valeur manquante) car il n'y a pas de valeur précédente à soustraire.

Supprimons la ligne contenant cette valeur manquante :

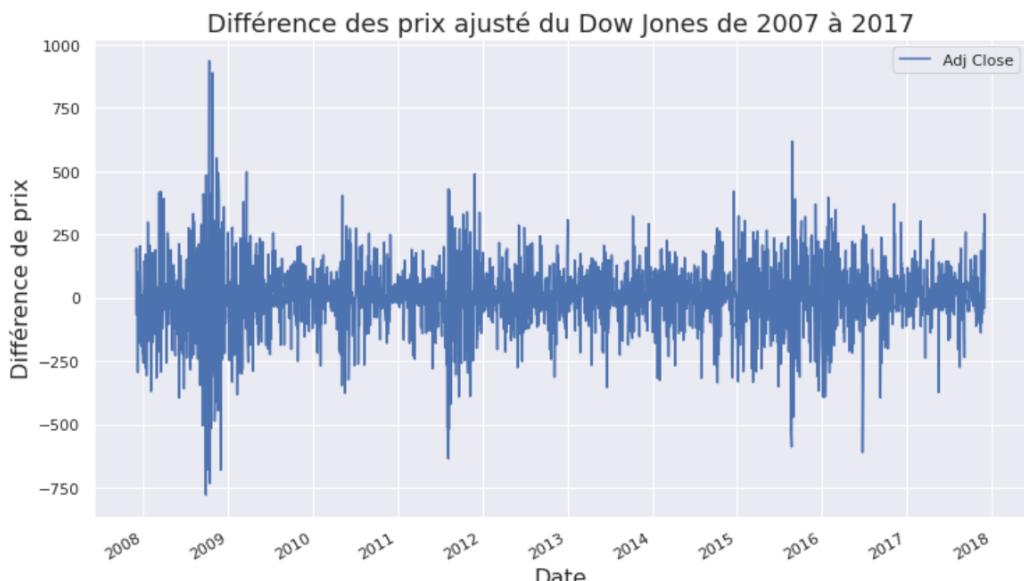
```
df_stat = df_stat.dropna()
```

Faisons encore le test statistique de Dicky-Fuller pour vérifier la stationnarité de la nouvelle série :

```
results2 = adfuller(df_stat['Adj Close'])  
  
print(results2)  
  
(-11.029880558890577, 5.703548257817322e-20, 19, 2498, {'1%': -  
3.4329705094097114, '5%': -2.8626977267304357, '10%': -  
2.5673863028421136}, 31604.126162811706)
```

La statistique de test est cette fois-ci égale à -11,03 avec une p-valeur pratiquement nulle ($5,70 \times 10^{-20}$). Nous rejetons donc l'hypothèse nulle. La série est donc stationnaire.

```
# Visualisation de la nouvelle série stationnaire  
  
df_stat.plot()  
  
plt.title("Différence des prix ajusté du Dow Jones de 2007 à 2017", fontsize = 18)  
  
plt.xlabel ('Date', fontsize = 16)  
  
plt.ylabel('Différence de prix', fontsize = 16)  
  
plt.show()
```



Nous remarquons qu'il n'y a plus de tendance dans la nouvelle série. La nouvelle série est donc bel et bien stationnaire.

Parfois, il est nécessaire de prendre la différence plus d'une fois avant de pouvoir rendre une série non-stationnaire en une série stationnaire. Après chaque opération de différenciation, vous devez visualiser la série et refaire le test statistique. Cela vous permet de connaître l'ordre de différenciation d nécessaire pour transformer votre série non stationnaire en une série stationnaire. Par exemple, si vous avez dû faire deux fois la différenciation avant que la série ne devienne stationnaire, alors l'ordre de différenciation est égal à 2. Il ne faut pas non plus différencier plus qu'il n'en faut. Connaître l'ordre de différenciation est très important pour la phase de modélisation.

Il existe aussi d'autres types de transformation qu'on peut effectuer comme la transformation logarithmique, racine carrée, etc.

Passons maintenant à la l'étape de modélisation.

MODELISATION

Nous sommes face à un problème de modélisation d'une série temporelle où nous devons effectuer des prévisions. Il s'agit de prédire les prix du Dow Jones dans les prochains jours après le 3 décembre 2017 (date de fin de notre série). Les prix sont des valeurs continues donc nous devons utiliser un algorithme de régression. De plus, puisque nous voulons prédire les futures valeurs de la série chronologique par rapport aux valeurs précédentes de cette même série chronologique, nous utiliserons alors un [modèle autorégressif](#)⁴⁸.

La modélisation d'une série temporelle n'est pas une tâche simple. Elle passe par plusieurs étapes plus ou moins complexes. Voici en résumé ces étapes :

- Sélection de l'ordre de différenciation d au cas où la série chronologique n'est pas stationnaire ;
- Sélection du bon ordre du modèle $ARMA(p, q)$. Il s'agit de pouvoir déterminer p (nombre de décalages autorégressifs) et q (nombre de décalages moyens mobiles). Nous ne verrons pas ici les détails mathématiques mais nous vous montrerons comment déterminer de manière pratique ces paramètres.
- Entraînement du modèle $ARMA(p, q)$ avec les bons paramètres p et q . Une fois que vous avez construit le modèle, vous pouvez maintenant effectuer vos prévisions. Cependant, votre modèle prédit la valeur de la différence de la série et non la valeur réelle de la série

chronologique. Pour obtenir les prévisions des valeurs réelles, vous devez passer par une étape appelée Intégration.

- L'intégration des prévisions. Cette étape complexe vous permettra de transformer la prévision des différences en une prévision des valeurs réelles.

Comme vous le voyez bien, le travail à faire est fastidieux ! Mais heureusement, il y a une méthode pour regrouper les étapes de différenciation, de modélisation et d'intégration dans un seul objet modèle [ARIMA](#)⁴⁹ que vous pouvez appliquer directement à une série non stationnaire. Cette méthode est très simple et elle facilite vraiment la vie 😊.

L'objectif de ce projet n'est pas de vous apprendre les théories mathématiques qui sous-tendent la modélisation des séries temporelles. Sur Internet, vous trouverez plein de documents à ce sujet. L'idée ici est de vous montrer de manière très pratique, comment modéliser une série temporelle en écrivant un code Python que vous pouvez utiliser dans un autre projet du même genre.

CHOIX DU BON ORDRE DU MODELE ARMA (p,q)

L'ordre du modèle est très important pour la qualité des prévisions. Sans rentrer dans les détails mathématiques, vous apprendrez comment choisir les bons paramètres p et q de votre modèle ou tout au moins avoir une idée de leurs intervalles. L'un des principaux moyens d'identifier le bon ordre du modèle est de tracer la [fonction d'autocorrélation](#)⁵⁰ et la [fonction d'autocorrélation partielle](#)⁵¹. La série temporelle doit être stationnaire avant de tracer ces deux fonctions.

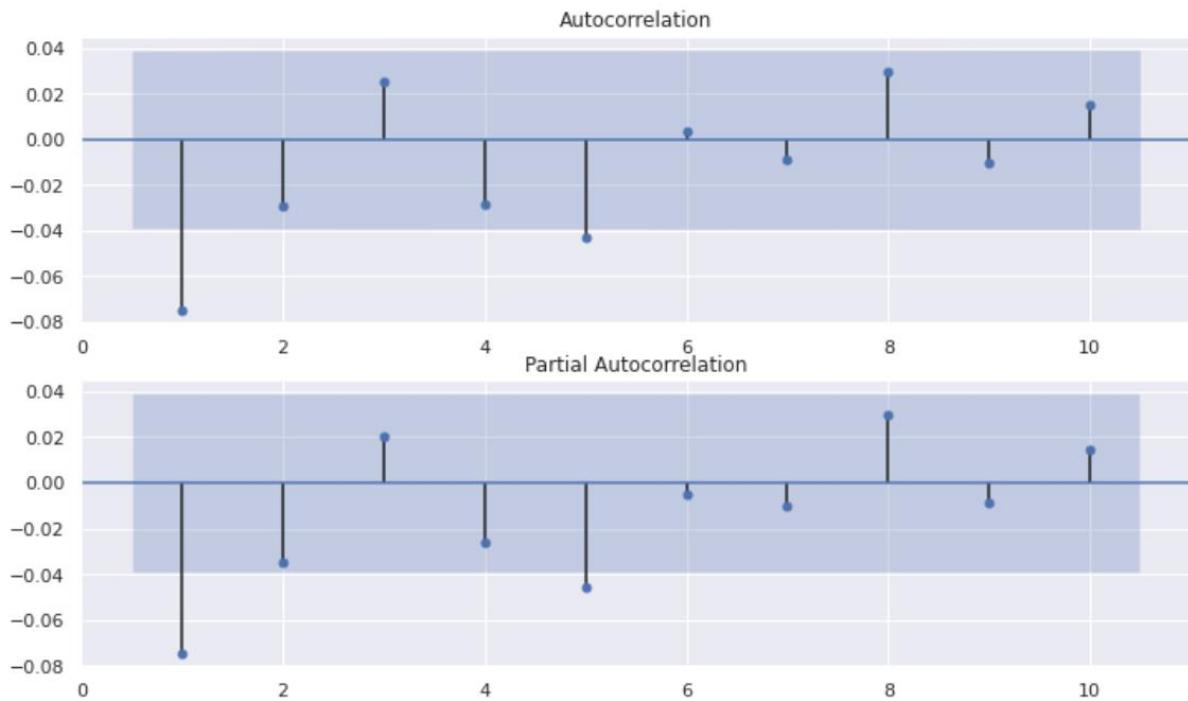
```
# Autocorrélation et Auto corrélation partielle

fig, (ax1, ax2) = plt.subplots(2,1)

plot_acf(df_stat, lags=10, zero=False, ax=ax1)

plot_pacf(df_stat, lags=10, zero=False, ax=ax2)

plt.show()
```



La fonction d'autocorrélation indique le paramètre p et la fonction d'autocorrélation partielle indique le paramètre q . En analysant ces deux fonctions, nous pouvons déduire l'ordre du modèle. Les barres qui se trouvent à l'intérieur de la zone ombrée en bleu ne sont pas statistiquement significatives.

Pour les deux paramètres, nous remarquons que les barres statistiquement significatives sont les barres 1 et 5. La barre 5 est dans les deux tracés est pratiquement à l'intérieur de la zone ombrée bleue. Donc, les deux fonctions nous suggèrent un modèle $ARMA(1,1)$.

Il existe aussi d'autres manières d'identifier le bon ordre du modèle : le [critère d'information d'Akaike](#)⁵² (**AIC** en anglais) et le [critère d'information bayésien](#)⁵³ (**BIC** en Anglais). Plus les valeurs de ces deux critères sont faibles, plus le modèle est bon. L'AIC et le BIC pénalisent les modèles complexes c'est-à-dire avec un ordre trop élevé. Très souvent les deux critères indiquent le choix du même modèle. Si ce n'est pas le cas, il faudra choisir nous-mêmes. Si l'objectif est d'identifier un bon modèle prédictif, il faut considérer l'AIC. Si l'objectif est d'identifier un bon modèle explicatif, il faut considérer le BIC.

Voyons maintenant comment implémenter tout ceci dans Python :

```
aic_bic = [ ]  
  
for p in range(6):  
  
    for q in range(6):  
  
        try:  
  
            model = SARIMAX(df, order=(p,1,q))  
  
            resultats = model.fit()  
  
            print(p, q, resultats.aic, resultats.bic)  
  
            aic_bic.append((p, q, resultats.aic, resultats.bic  
))  
  
        except:  
  
            print(p, q, None, None)
```

```

0 0 31986.546772382884 31992.37799259749
0 1 31973.82870169945 31985.49114212866
0 2 31974.28875143764 31991.782412081455
0 3 31975.274969389746 31998.599850248163
0 4 31974.51382602103 32003.66992709405
0 5 31971.936660634823 32006.92398192245
1 0 31974.65933793529 31986.321778364498
1 1 31974.66670146612 31992.160362109935
1 2 31975.19405965985 31998.51894051827
1 3 31976.72555751686 32005.881658589882
1 4 31974.275383167696 32009.262704455323
1 5 31973.774628807078 32014.59317030931
2 0 31973.93434328514 31991.42800392895
2 1 31974.741676888698 31998.066557747115
2 2 31975.239134279444 32004.395235352466
2 3 31973.376183551198 32008.363504838824
2 4 31969.77309099154 32010.59163249377
2 5 31972.96633418274 32019.616095899575
3 0 31974.64593967188 31997.970820530296
3 1 31976.310981849918 32005.46708292294
3 2 31977.165020049615 32012.15234133724
3 3 31966.30241308547 32007.1209545877
3 4 31971.791859675846 32018.44162139268
3 5 31971.45695385455 32023.93793578599
4 0 31975.05559526384 32004.21169633686
4 1 31974.333322215418 32009.320643503044
4 2 31974.17333347148 32014.99187497371
4 3 31971.91369159723 32018.563453314066
4 4 31973.702514379354 32026.183496310794
4 5 31976.14209916943 32034.454301315473
5 0 31972.11839351003 32007.105714797657
5 1 31974.021806578785 32014.840348081016
5 2 31973.257658081624 32019.90741979846
5 3 31974.542280504986 32027.023262436425
5 4 31973.86360671772 32032.175808863765
5 5 31976.450998178887 32040.594420539535

```

Le code ci-dessus nous permet de rechercher le meilleur modèle pour notre série temporelle. Les paramètres p et q varient entre 0 et 5 inclus (valeurs suggérées par les fonctions d'autocorrélation et d'autocorrélation partielle). Un modèle $ARIMA(p,d,q)$ est ajusté directement à notre série temporelle non stationnaire pour chaque combinaison possible de p et q . Veillez toujours à spécifier l'ordre de différenciation d . Ici d est égal à 1. Pour finir, nous affichons les valeurs de AIC et de BIC pour chaque combinaison de p et q .

Il peut arriver qu'une combinaison de p et q ne fonctionne pas et résulte donc en une erreur. Pour pallier cela, nous avons mis les instructions à l'intérieur d'une syntaxe *try...except*.

C'est très difficile de se retrouver avec tous ces chiffres. Puisque nous avons stocker ce résultat dans une liste, nous allons pouvoir afficher les valeurs dans un ordre bien déterminé ce qui nous

permettra de vite connaître la meilleure combinaison de p et q. Notre objectif est d'identifier un bon modèle prédictif donc nous choisirons la combinaison p et q qui donne le plus petit AIC.

```
# Construction d'une dataframe à partir de la liste aic_bic  
  
order_df = pd.DataFrame(aic_bic, columns=['p', 'q', 'AIC', 'BIC'])  
  
# Affichage de order_df dans l'ordre croissant des AIC  
order_df.sort_values('AIC').head()
```

	p	q	AIC	BIC
21	3	3	31966.302413	32007.120955
16	2	4	31969.773091	32010.591632
23	3	5	31971.456954	32023.937936
22	3	4	31971.791860	32018.441621
27	4	3	31971.913692	32018.563453

D'après ce résultat, le meilleur modèle est celui d'ordre (3,3).

A présent, nous allons construit un modèle **ARIMA(3,1,3)** de prévision des futurs prix des cours du Dow Jones. Mais avant cela, divisons la série en données d'entraînement et d'évaluation du modèle.

```
# Création d'un modèle ARIMA(3,1,3)  
  
dow_jones_model = SARIMAX(df, order=(3,1,3))  
  
# Entraînement du modèle  
  
dow_jones_results = dow_jones_model.fit()
```

EVALUATION DU MODELE

Après la construction du modèle, nous devons évaluer sa qualité. Nous analyserons les résidus c'est-à-dire les différences entre les valeurs prédictes par le modèle et les valeurs réelles.

```
# Résidus

residus = dow_jones_results.resid

print(residus)
```

```
Date
2007-12-03    13314.570313
2007-12-04    -48.964152
2007-12-05    201.314635
2007-12-06    185.104554
2007-12-07    30.752848
...
2017-11-27    15.946922
2017-11-28    264.083364
2017-11-29    131.192852
2017-11-30    340.025048
2017-12-01    -15.060549
Length: 2519, dtype: float64
```

Pour un modèle idéal, les résidus doivent être normalement distribués. Nous allons vérifier cela à l'aide de 04 graphiques.

Calculons d'abord l'erreur absolue moyenne, c'est-à-dire la moyenne des valeurs absolues des résidus :

```
# Erreur absolue moyenne

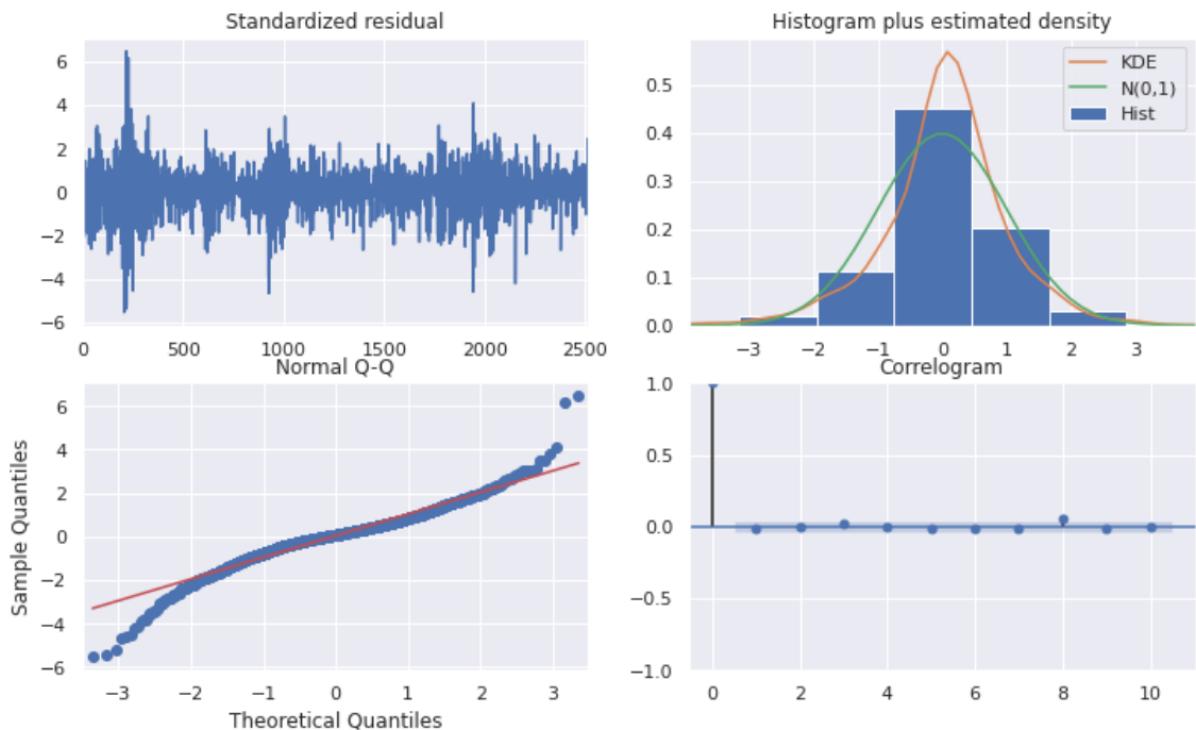
mae = np.mean(np.abs(residus))

print("L'erreur absolue moyenne est égale à :", mae)

L'erreur absolue moyenne est égale à : 102.55577178633419
```

Plus l'erreur absolue moyenne est faible, plus le modèle est performant.

Il y a 04 graphiques très souvent utilisés dans l'évaluation d'un modèle de régression d'une série temporelle :



Le graphique donnant les résidus standardisés montre qu'il n'y a pas de structure évidente dans ces résidus ce qui est un bon signe que notre modèle fonctionne correctement.

L'histogramme donne la distribution des résidus. La courbe verte est une loi normale et la courbe rouge est une version lissée de l'histogramme des résidus du modèle. Pour un modèle idéal, ces deux courbes sont parfaitement superposées donc identiques ce qui n'est pas tellement le cas ici. Dans la réalité il n'y a pas de modèle totalement parfait.

Le graphique Normal Q-Q montre aussi comment la distribution des résidus de notre modèle (en bleu) se compare avec la loi normale (en rouge). Nous remarquons que les points se retrouvent le long de la droite rouge sauf les points extrêmes.

Le dernier graphique est le corrélogramme (tracé de la fonction d'autocorrélation) des résidus. Pour un modèle idéal, 95% des corrélations pour un décalage supérieur à zéro ne doivent pas être significatives en d'autres termes les barres doivent se retrouver à l'intérieur de la zone ombrée bleue. Dans notre cas, toutes les barres ne sont pas significatives donc les résidus ne sont pas corrélés.

A partir de ces 04 graphiques, nous pouvons conclure que notre modèle fonctionne correctement.

En plus de ces 04 graphiques, nous pouvons nous servir du résumé statistique généré par notre modèle de prévision :

```
# Résumé statistique du modèle
```

```
print(dow_jones_results.summary())
```

Statespace Model Results						
Dep. Variable:	Adj Close	No. Observations:	2519			
Model:	SARIMAX(3, 1, 3)	Log Likelihood	-15976.151			
Date:	Sat, 25 Jul 2020	AIC	31966.302			
Time:	17:42:30	BIC	32007.121			
Sample:	0 - 2519	HQIC	31981.116			
Covariance Type:	opg					
	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.0353	0.132	0.269	0.788	-0.223	0.293
ar.L2	-0.5962	0.081	-7.326	0.000	-0.756	-0.437
ar.L3	0.6100	0.126	4.857	0.000	0.364	0.856
ma.L1	-0.1033	0.124	-0.835	0.404	-0.346	0.139
ma.L2	0.5701	0.074	7.713	0.000	0.425	0.715
ma.L3	-0.6484	0.114	-5.670	0.000	-0.873	-0.424
sigma2	1.907e+04	323.928	58.869	0.000	1.84e+04	1.97e+04
Ljung-Box (Q):		45.52	Jarque-Bera (JB):		1622.44	
Prob(Q):		0.25	Prob(JB):		0.00	
Heteroskedasticity (H):		0.74	Skew:		-0.37	
Prob(H) (two-sided):		0.00	Kurtosis:		6.86	

Considérons les valeurs de **Prob(Q)** et **Prob(JB)** dans le tableau ci-dessus :

- **Prob(Q)** est la p-valeur associée à l'hypothèse nulle selon laquelle les résidus du modèle ne sont pas corrélés ;
- **Prob(JB)** est la p-valeur associée à l'hypothèse nulle selon laquelle les résidus sont normalement distribués ;

Dans le cas présent $\text{Prob}(Q) = 0.25$ donc supérieure à 0.05. Alors on accepte l'hypothèse selon laquelle les résidus de notre modèle ne sont pas corrélés. $\text{Prob}(JB) = 0.00$ donc inférieure à 0.05. Alors on rejette l'hypothèse nulle selon laquelle les résidus de notre modèle sont

normalement distribués. Ceci est sûrement dû aux valeurs extrêmes (Voir graphique Normal Q-Q)

Alors les résidus de notre modèle ne sont pas corrélés (ce qui est bien) et sont pratiquement normalement distribués sauf pour les valeurs extrêmes.

PREVISIONS

Passons à présent à la prévision des prix de l'indice dans les 365 prochains jours de trading.

```
# Prévision des futurs 365 prix de l'indice

mean_price_forecast = dow_jones_results.get_forecast(steps = 365).predicted_mean

mean_price_forecast
```

2519	24206.150625
2520	24202.815918
2521	24202.766097
2522	24189.233535
2523	24186.750665
	...
2879	24186.419655
2880	24186.386196
2881	24186.372795
2882	24186.413198
2883	24186.402204
	Length: 365, dtype: float64

Nous avons une série des futurs prix de l'indice du Dow Jones. Calculons les intervalles de confiance des prévisions ci-dessus :

```
# Intervalles de confiance des prévisions

intervals = dow_jones_results.get_forecast(steps = 365).conf_int()
```

intervals

	lower	Adj Close	upper	Adj Close
2519	23935.496554	24476.804696		
2520	23832.822784	24572.809052		
2521	23759.260005	24646.272190		
2522	23682.632294	24695.834776		
2523	23626.924607	24746.576724		
...
2879	19755.171901	28617.667408		
2880	19749.018952	28623.753440		
2881	19742.894881	28629.850709		
2882	19736.833949	28635.992446		
2883	19730.729036	28642.075373		

365 rows × 2 columns

BEAU TRAVAIL 😊 Nous avons calculé les futurs 365 prix ajustés de l'indice du Dow Jones avec leurs intervalles de confiance.

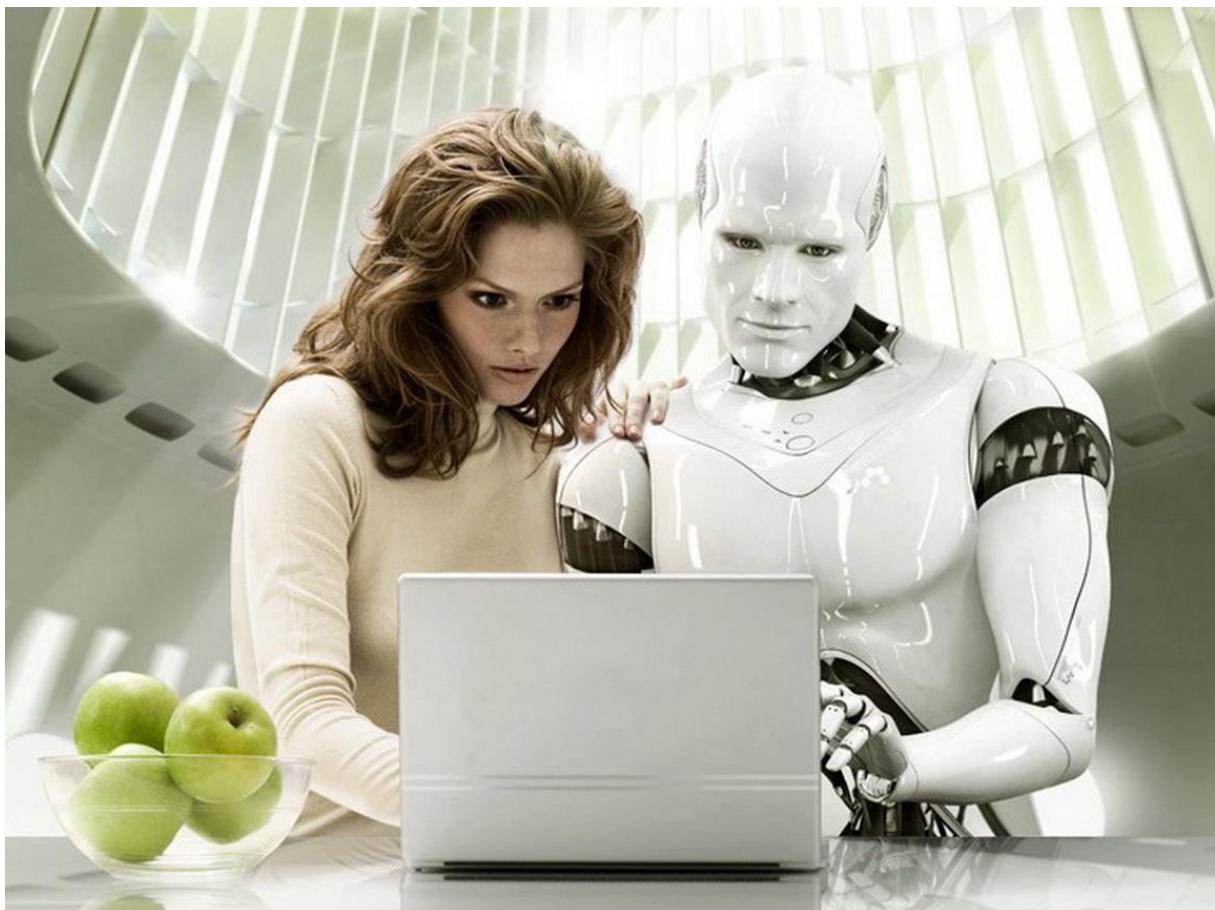
CONCLUSION

Dans ce projet, vous avez appris comment modéliser une série temporelle. Nous sommes partis de la collecte des données de l'indice du Dow Jones de 2007 à 2017 sur Yahoo Finance jusqu'à la prévision des prix ajustés de l'année 2018. Le flux de travail utilisé dans ce projet peut être appliqué à tout projet de modélisation d'une série chronologique et ceci dans n'importe quel domaine (Finance, Climat, etc.).

La modélisation des cours boursiers est un domaine de recherche très actif. Plusieurs organisations financières et Instituts de recherche dans le monde se sont spécialisées dans ce

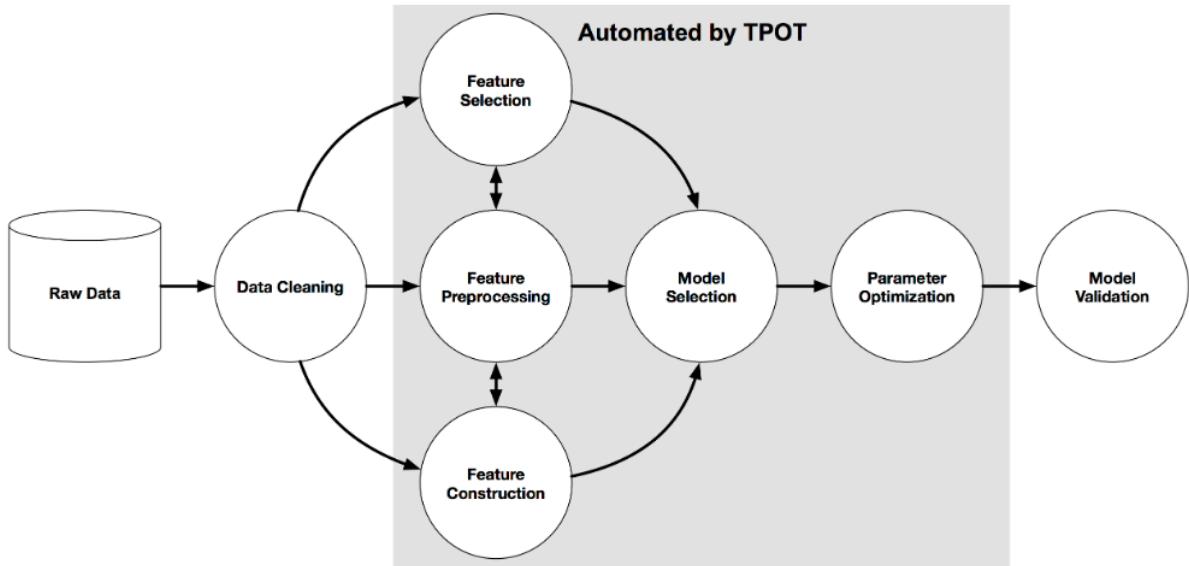
domaine. Vous pouvez aussi contribuer à ces recherches et même gagner de l'argent en passant. En effet, il existe deux communautés particulièrement intéressantes, [Quantopian](#)⁵³ et [Numerai](#)⁵⁴, où vous pouvez soumettre vos algorithmes de prévision des prix des actions dans des compétitions. Si votre algorithme surpassé les algorithmes des autres concurrents, vous gagnerez un prix en espèces, et si vous obtenez la licence pour votre algorithme, vous tirerez un profit des transactions qui seront effectuées via votre algorithme sous licence. Ces deux plateformes disposent également de plusieurs tutoriels vous permettant d'apprendre.

PROJET 13 : UTILISATION DE L'OUTIL TPOT COMME ASSISTANT INTELLIGENT DANS UN PROJET DE MACHINE LEARNING



INTRODUCTION

Que diriez-vous d'un outil qui vous facilite la vie en vous aidant à trouver, de manière automatique et optimisée, le meilleur modèle de Machine Learning pour vos données ? 😊. Eh oui ! Un tel outil existe bel et bien et il est à la portée de tous. Dans ce projet vous apprendrez, à travers un cas d'étude, comment utiliser la librairie TPOT dans Python afin d'automatiser entièrement vos pipelines de Machine Learning et de trouver le meilleur modèle pour ajuster vos données. Le cas d'étude est la prédiction de dons de sang dans les universités de Taiwan. L'objectif est de prédire si un ancien donneur est susceptible de donner à nouveau du sang.



CONTEXTE

Très souvent dans un projet de Machine Learning, on recherche le modèle qui donnera le meilleur score (la métrique d'évaluation du modèle est définie en fonction du problème qu'on traite). Malheureusement, il existe une multitude de combinaisons de techniques possibles pour les étapes de construction. Ces étapes concernent le nettoyage des données, la sélection de variables, la réduction de dimension, la transformation des variables catégorielles en variables numériques, la standardisation, la modélisation, le réglage des hyperparamètres, etc. Dès lors, le défi qui se pose est de trouver la meilleure combinaison de techniques pour minimiser les erreurs de prédictions. Une très bonne compréhension des données et aussi une bonne expérience peuvent être de grands atouts. Mais dans certains cas, on ne sait vraiment pas par où commencer et/ou ce qu'on pourrait faire de plus pour améliorer le modèle. Ceux qui ont l'habitude des compétitions en Data Science comprendront très bien. Dans ces compétitions, le détail compte beaucoup car les scores sont souvent affichés à 4 chiffres après la virgule et même plus. En tant qu'humain, il est normal qu'après avoir essayé une dizaine de combinaisons possibles qu'on soit fatigué et qu'on veuille passer à autre chose. Heureusement, il existe un puissant outil qui peut réaliser automatiquement des milliers de combinaisons de techniques et vous générer le code du meilleur pipeline qui a donné le meilleur score pour l'ajustement de vos données. La librairie TPOT peut être considérée comme votre assistant en Data Science. Elle vous permettra d'aller prendre du café, de papoter avec vos collègues, jouer au baby-foot, et même dormir si vous voulez 😊 pendant qu'elle travaille pour vous.



N'est-ce pas génial ! En lisant ce projet, l'automatisation de vos pipelines Machine Learning avec l'outil TPOT n'aura plus aucun secret pour vous.

PRESENTATION DE L'OUTIL TPOT

[TPOT](#)⁵⁵ est une librairie de Python qui optimise les pipelines de Machine Learning à l'aide de la programmation génétique. Le but de ce projet n'est pas d'expliquer la théorie sur la programmation génétique. Ce qui nous intéresse ici est de comprendre, de manière très pratique, l'utilisation de l'outil TPOT. Grâce à cet outil, vous pouvez explorer automatiquement des centaines de pipelines possibles pour trouver le meilleur pour votre dataframe. Pour installer TPOT, vous pouvez suivre les instructions dans cette [page](#)⁵⁵. Pour le présent cas d'étude, nous utiliserons TPOT pour nous aider à trouver un modèle que nous pouvons ensuite explorer et optimiser davantage. Après avoir préparé vos données, vous les utilisez pour entraîner l'algorithme que vous avez choisi puis vous régler ses hyperparamètres afin d'optimiser le score du modèle. La data Science étant un processus itératif, il n'est pas rare de revenir en arrière afin de revoir telle ou telle autre technique dans la préparation des données, refaire une autre modélisation puis un nouveau réglage des hyperparamètres. De plus, plusieurs algorithmes sont entraînés puis évalués. Au lieu de faire tout cela par vous-même, l'outil TPOT automatise toutes ces étapes de recherche de la meilleure combinaison de techniques sous le principe de la programmation génétique. Mieux encore, TPOT vous génère le code du meilleur pipeline qui a donné le meilleur score pour l'ajustement de vos données. Passons à la pratique !

ETUDE DE CAS D'APPLICATION DE L'OUTIL TPOT : PRÉDICTION DES DONS DE SANG

PROBLEMATIQUE

Selon l'organisation Mondiale de la Santé (OMS), "le sang est le cadeau le plus précieux qu'une personne puisse faire à une autre personne : le cadeau de la vie." Malheureusement, il y a une tendance baissière dans les dons de sang et cela est encore plus grave dans des régions très vulnérables du globe.

Le Machine Learning peut aider dans la prédition des futurs dons de sang et ainsi sauver des vies. L'objectif de cette étude de cas est de construire un modèle qui prédit si un donneur de sang est susceptible de donner à nouveau du sang. Nous utilisons TPOT pour nous aider à trouver un modèle que nous essayerons d'optimiser davantage.

Commençons par importer les librairies nécessaires pour cette étude.

LIBRAIRIES

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.metrics import roc_auc_score

from tpot import TPOTClassifier

from sklearn.linear_model import LogisticRegression
```

DONNEES

Le jeu de données se compose d'un échantillon aléatoire de 748 donneurs du centre de services de transfusion sanguine de la ville de Hsin-Chu à Taiwan. Vous pouvez télécharger les données via le répertoire [UCI Machine Learning](#)⁵⁶ et consulter leur description détaillée.

```
#Importation et affichage des données
```

```
df = pd.read_csv('https://raw.githubusercontent.com/JosueAfouda/TPOT/master/transfusion.data')

df.head()
```

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	whether he/she donated blood in March 2007	
0	2	50	12500	98		1
1	0	13	3250	28		1
2	1	16	4000	35		1
3	2	20	5000	45		1
4	1	24	6000	77		0

Renommons la variable cible :

```
#Renommer la variable d'intérêt 'whether he/she donated blood
in March 2007'
df.rename(columns={'whether he/she donated blood in March 2007
':'Is a donor'}, inplace=True)

#Informations sur les données
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 748 entries, 0 to 747
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   Recency (months)    748 non-null   int64  
 1   Frequency (times)   748 non-null   int64  
 2   Monetary (c.c. blood) 748 non-null   int64  
 3   Time (months)       748 non-null   int64  
 4   Is a donor          748 non-null   int64  
 dtypes: int64(5)
 memory usage: 29.3 KB
```

La variable d'intérêt *'Is a donor'* est binaire. 0 : le donneur ne donnera pas de sang ; 1 : le donneur donnera du sang. Par ailleurs, il n'y a pas de valeurs manquantes et les variables sont stockées dans les types adéquat (entiers naturels). On peut donc dire que le jeu de données est propre et peut être directement utilisé pour la modélisation. Très souvent, il faut passer par une étape plus ou moins fastidieuse de nettoyage de données avant l'entraînement des algorithmes. J'ai choisi un jeu de données propre et léger pour vous permettre de vous focaliser sur l'utilisation de l'outil TPOT. Cependant, ce que vous apprendrez ici peut être très bien appliqué pour des jeux de données beaucoup plus importants en taille et nécessitant une bonne phase de préparation. Avant de passer à la modélisation proprement dite, divisons les données en ensemble d'entraînement et de validation.

```

#Train/Test split
seed = 123

target = df.pop('Is a donor')

X_train, X_test, y_train, y_test = train_test_split(df, target
, test_size=0.2, stratify=target)

print(X_train.shape)

print(y_train.shape)

print(X_test.shape)

print(y_test.shape)

(598, 4)
(598,)
(150, 4)
(150,)

```

APPLICATION DE TPOT

Voici la partie tant attendue où vous verrez l'outil TPOT en action. Puisqu'il s'agit d'un problème de classification, nous avons importé la classe TPOTClassifier. Pour un problème de régression, il faudra importer la classe TPOTRegressor. Consultez [ici](#)⁵⁵ la signification des paramètres de TPOTClassifier.

```

#Création d'une instance de la classe TPOTClassifier

tpot = TPOTClassifier(generations=10,
                      population_size=10,
                      verbosity=2,
                      offspring_size=10,
                      scoring='roc_auc',
                      config_dict='TPOT light',
                      cv = 5,
                      random_state=seed)

```

- ***generations*** : nombre d'itérations du processus d'optimisation du pipeline d'exécution.
La valeur par défaut est 100.
- ***population_size*** : nombre d'individus à conserver dans la population de programmation génétique à chaque génération. La valeur par défaut est 100.

- ***verbosity*** : La quantité d'informations que TPOT communique pendant son exécution.
- ***offspring_size*** : nombre de descendants à produire dans chaque génération de programmation génétique. La valeur par défaut est 100.
- ***scoring*** : Fonction utilisée pour évaluer la qualité d'un pipeline donné pour le problème de classification. La valeur par défaut est 'accuracy'.
- ***config_dict*** : Un dictionnaire de configuration pour personnaliser les opérateurs et les paramètres que TPOT recherche dans le processus d'optimisation. La valeur par défaut est None. 'TPOT light' indique que TPOT utilisera une configuration intégrée avec uniquement des modèles et préprocesseurs rapides.
- ***cv*** : Stratégie de validation croisée utilisée lors de l'évaluation des pipelines. La valeur par défaut est 5.
- ***random_state*** : Choisissez un entier naturel. Utilisez ce paramètre pour vous assurer que TPOT vous donnera les mêmes résultats chaque fois que vous l'exécuterez avec le même ensemble de données avec le même entier naturel.

#Entraînement

```
tpot.fit(X_train, y_train)
```

Voici l'affichage au cours de l'entraînement ainsi que le résultat :

```
HBox(children=(FloatProgress(value=0.0, description='Optimization Progress', max=110.0, style=ProgressStyle(de...
Generation 1 - Current best internal CV score: 0.7363618083030699
Generation 2 - Current best internal CV score: 0.7373274361641784
Generation 3 - Current best internal CV score: 0.7373274361641784
Generation 4 - Current best internal CV score: 0.7373274361641784
Generation 5 - Current best internal CV score: 0.7373274361641784
Generation 6 - Current best internal CV score: 0.7396700148748232
Generation 7 - Current best internal CV score: 0.7396700148748232
Generation 8 - Current best internal CV score: 0.7396700148748232
Generation 9 - Current best internal CV score: 0.740206932444296
Generation 10 - Current best internal CV score: 0.7412345779621962

Best pipeline: LogisticRegression(MinMaxScaler(input_matrix), C=1.0, dual=False, penalty=12)
TPOTClassifier(config_dict='TPOT light', crossover_rate=0.1, cv=5,
               disable_update_check=False, early_stop=None, generations=10,
               log_file=<ipykernel.iostream.OutStream object at 0x00000219B50E76C8>,
               max_eval_time_mins=5, max_time_mins=None, memory=None,
               mutation_rate=0.9, n_jobs=1, offspring_size=10,
               periodic_checkpoint_folder=None, population_size=10,
               random_state=123, scoring='roc_auc', subsample=1.0,
               template=None, use_dask=False, verbosity=2, warm_start=False)
```

Pour chacune des 10 générations calculées, nous avons le score CV du modèle. Comme vous le voyez, le meilleur pipeline est celui qui a le score de précision CV égal à 74,12%. L'algorithme de classification est la Régression logistique précédée d'une étape de prétraitement

des données avec MinMaxScaler(). Le code ci-dessous permet d'afficher les étapes du meilleur pipeline trouvé :

```
#Affichage des étapes du meilleur pipeline

print('\nEtapes du meilleur pipeline:', end='\n')

for idx, (name, transform) in enumerate(tpot.fitted_pipeline_.steps, start=1):
    print(f'{idx}. {transform}' )
```

Le résultat :

```
Etapes du meilleur pipeline:
1. MinMaxScaler(copy=True, feature_range=(0, 1))
2. LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, l1_ratio=None, max_iter=100,
multi_class='auto', n_jobs=None, penalty='l2',
random_state=123, solver='lbfgs', tol=0.0001, verbose=0,
warm_start=False)
```

Auriez-vous pensé à un tel pipeline ? C'est possible. Le pipeline trouvé est plutôt simple. Cependant, il arrive aussi parfois que TPOT propose un pipeline très complexe. Quel est le score sur les données de validation ?

```
# AUC score du modèle tpot sur les données de validation

tpot_auc_score2 = roc_auc_score(y_test, tpot2.predict_proba(X_
test)[:, 1])

print(f'\nAUC score on test data for the second tpot: {tpot_auc_
score2:.4f}')
```

Le score sur les données de validation est de 82,10%.

En augmentant les valeurs des arguments *generations*, *population_size*, et *offspring_size*, vous augmentez le nombre de combinaisons que l'outil TPOT va essayer et ainsi augmenter votre chance de trouver un modèle avec un meilleur score. Rappelez-vous que la valeur par défaut pour ces 3 paramètres est 100. Mais une telle valeur fera que l'algorithme peut prendre plusieurs heures pour s'exécuter. L'objectif de ce projet est de vous montrer le principe du fonctionnement du TPOT. Vous pouvez essayer vos propres valeurs pour tenter d'améliorer davantage le

modèle. Nous allons quand même augmenter (doubler) la valeur de ces 3 arguments afin de voir ce qui se passera.

```
#Création d'une autre instance de la classe TPOTClassifier

tpot2 = TPOTClassifier(generations=30,
                       population_size=30,
                       verbosity=2,
                       offspring_size=30,
                       scoring='roc_auc',
                       config_dict='TPOT light',
                       cv = 5,
                       random_state=seed)

#Entraînement

tpot2.fit(X_train, y_train)

#Affichage des étapes du meilleur pipeline

print('\nEtapes du meilleur pipeline:', end='\n')

for idx, (name, transform) in enumerate(tpot2.fitted_pipeline_
.steps, start=1):
    print(f'{idx}. {transform}')

# AUC score du modèle tpot sur les données de validation

tpot_auc_score2 = roc_auc_score(y_test, tpot2.predict_proba(X_
test)[:, 1])

print(f'\nAUC score on test data for the second tpot: {tpot_au
c_score2:.4f}')
```

Sur les données d'entraînement, on est passé à un score de 74,12% à 74,58%. En revanche le score sur les données d'évaluation est passé de 82,10% à 81,30%. Nous allons donc garder le premier pipeline. Par ailleurs le pipeline trouvé cette fois-ci est plus complexe que le premier. Une fois que vous êtes satisfait des résultats, vous pouvez exporter le code Python correspondant pour le pipeline optimisé vers un fichier texte avec la fonction d'exportation. Nous allons enregistrer le code du premier pipeline trouvé avec la fonction *tpot.export()*.

```
# Enregistrer votre meilleur pipeline au format .py  
tpot.export('tpot_best_pipeline.py')
```

SUPER ! Nous n'avons pas eu à essayer plusieurs options avant d'obtenir non seulement le meilleur modèle mais aussi son code. TPOT a travaillé pour nous 😊.

LIMITES DE TPOT

Même si l'outil TPOT est très puissant et génial, il ne doit en aucun cas nous plonger dans une paresse de réflexion. Il doit être considéré comme un point de départ pour vous orienter dans votre recherche du meilleur modèle. Par ailleurs, il faut garder à l'esprit ceci : *Garbage in, Garbage out*. Si vous ne nettoyez pas correctement vos données, ne vous attendez pas à avoir de bons résultats même en utilisant TPOT. Les deux inconvénients principaux de TPOT sont :

Il peut prendre plusieurs minutes voire heures et même jours d'exécution puisqu'il essaie plusieurs combinaisons de prétraitement (y compris PCA, Standardisation, etc.) avec plusieurs algorithmes ;

Il peut proposer différentes solutions pour le même jeu de données.

CONCLUSION

Dans ce projet, j'ai montré à travers un simple problème de classification binaire, comment utiliser l'outil TPOT. TPOT automatise toutes les étapes de recherche de la meilleure combinaison de techniques sous le principe de la programmation génétique. Mieux encore, TPOT vous génère le code du meilleur pipeline qui a donné le meilleur score pour l'ajustement de vos données. TPOT dispose de plusieurs autres possibilités. De ce fait je vous conseille de consulter cette excellente [documentation](#)⁵⁵ pour en apprendre davantage.

PROJET 14 : AUTOMATISATION DU WORKFLOW D'UN PROJET DE MACHINE LEARNING AVEC LA FONCTION PIPELINE

INTRODUCTION

Un projet en Data Science nécessite plusieurs étapes pour parvenir aux résultats finaux. Trois étapes sont incontournables : Nettoyage des données, Transformation (Prétraitement) des données et Modélisation. La réalisation de ces étapes n'est pas toujours linéaire. Bien souvent, un projet en Data Science est un processus itératif c'est-à-dire qu'il faut effectuer des *feedbacks* réguliers afin de revoir tel ou tel autre élément pour augmenter la performance de votre modèle. De ce fait, le travail peut vite devenir fastidieux, très consommateur de temps et pénible. De plus, lorsque vous avez effectué séparément chacune de ces étapes pour une dataframe donnée, il faudra encore les reprendre pour un autre jeu de données (Travail répétitif).

Le module [*pipeline*](#)⁵⁷ de *sklearn* avec sa fonction [*Pipeline*](#)⁵⁸ permet de construire un super objet composé de transformateurs de la dataframe et d'un ou plusieurs estimateurs. En clair, vous pouvez avoir par exemple les étapes de prétraitement, de transformation (Réduction de dimensionnalité par exemple) et de modélisation le tout dans un même objet appelé pipeline. Ainsi, cet objet (pipeline) peut être appliqué facilement à d'autres jeux de données. Alors, grâce à l'objet pipeline, vous gagnez beaucoup de temps et vous éliminez d'un coup la pénibilité et la répétitivité du travail.

Après la lecture de ce projet, vous saurez comment automatiser toutes vos tâches de Data Science et de Machine Learning. De plus, la fonction pipeline n'aura plus aucun secret pour vous 😊

ETUDE DE CAS D'APPLICATION DE PIPELINE : MODELISATION DU RISQUE DE CREDIT

CONTEXTE

Vous êtes Data Scientist dans une grande Banque et vous êtes chargé principalement de la modélisation du risque de crédit. Vous devez effectuer plusieurs tâches pour augmenter la performance de vos modèles. Pour chaque nouveau jeu de données, vous devez encore reprendre toutes ces tâches. Cela vous prend énormément de temps et la pression devient de plus en forte par rapport aux deadlines.

Plus d'inquiétude à vous faire. Après la lecture de ce projet, vous saurez comment automatiser toutes vos tâches de Data Science et de Machine Learning. De plus, la fonction pipeline n'aura plus aucun secret pour vous et vous serez toujours en avance sur les délais 😊

Commençons par importer les librairies nécessaires pour cette étude.

LIBRAIRIES

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer

from sklearn.decomposition import PCA

from sklearn.metrics import classification_report

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import GridSearchCV
```

DONNEES

Avant d'importer le jeu de données dans votre notebook, il est conseillé de l'ouvrir d'abord avec Excel ou toute autre application afin de faire une petite inspection visuelle. De plus, il faut lire la documentation sur vos données si celle-ci existe.

C'est ce travail préalable qui nous a permis de savoir qu'il n'y a pas de noms de colonnes dans le jeu de données, que le séparateur est une virgule et que les valeurs manquantes sont représentées par un "?".

```
df_raw = pd.read_csv('https://raw.githubusercontent.com/Josue Afouda/Pipeline-ML/master/crx.data',
                     sep=',', header = None, na_values = "?")
```

```
df_raw.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	202.0	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	43.0	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	280.0	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	100.0	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	120.0	0	+

```
#Informations sur le jeu de données
df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
 ---  --   -----  --    
 0   0       678 non-null    object 
 1   1       678 non-null    float64
 2   2       690 non-null    float64
 3   3       684 non-null    object 
 4   4       684 non-null    object 
 5   5       681 non-null    object 
 6   6       681 non-null    object 
 7   7       690 non-null    float64
 8   8       690 non-null    object 
 9   9       690 non-null    object 
 10  10      690 non-null    int64  
 11  11      690 non-null    object 
 12  12      690 non-null    object 
 13  13      677 non-null    float64
 14  14      690 non-null    int64  
 15  15      690 non-null    object 
dtypes: float64(4), int64(2), object(10)
memory usage: 86.4+ KB
```

PREPARATION DES DONNEES

La préparation des données est une étape très importante dans tout projet en Data Science. Concernant les valeurs manquantes, elles seront tout simplement retirées. Mais il est bon de savoir qu'il existe plusieurs manières de traiter les données manquantes comme les imputations par la moyenne, médiane, valeur la plus fréquente, etc.

Créons une copie de notre jeu de données :

```
df = df_raw.copy()
```

Selon la description de ce jeu de données, au niveau de la colonne 15 le symbole "+" représente un cas où le crédit a été accordé (client solvable) et le symbole "-" représente un cas où le crédit n'a pas été accordé (client insolvable) à une personne. Nous allons remplacer ces symboles : 1 pour "+" et 0 pour "-".

```
df[15].replace({ "+":1, "-":0 }, inplace=True)
```

Nombre de valeurs manquantes par colonne :

```
df.isna().sum()
```

```
0      12  
1      12  
2       0  
3       6  
4       6  
5       9  
6       9  
7       0  
8       0  
9       0  
10      0  
11      0  
12      0  
13     13  
14      0  
15      0  
dtype: int6
```

Retirons toutes les lignes présentant des données manquantes :

```
df.dropna(axis=0, inplace=True)
```

```
df.shape
```

En éliminant les données manquantes, on passe de 690 à 653 lignes.

ETAPE DE PRETRAITEMENT DES DONNEES

Le prétraitement des données (*Data Preprocessing*) est l'étape qui vient juste avant la modélisation proprement dite. Dans le présent cas, cette étape comprendra la transformation des variables catégorielles en variables numériques (*Encoding*) ainsi que la standardisation (*Scaling*).

L'objectif étant de construire un pipeline qui englobe tout le processus de Machine Learning, le prétraitement sera donc la première composante (ou étape) de notre pipeline.

Pour la transformation des variables catégorielles, la fonction [*OneHotEncoder*](#)⁵⁹ a été utilisé.

```
#Création d'un objet OneHotEncoder pour transformer les variables catégorielles
```

```
catTransformer = OneHotEncoder(drop='first')
```

L'argument *drop = 'first'* permet d'éliminer la variable générée à partir de la première modalité. Cet argument est important car il permet de supprimer le problème de colinéarité entre variables générées après un *One Hot Encoding*.

Pour la standardisation au niveau des variables numériques, la fonction [*StandardScaler\(\)*](#)⁶⁰ a été utilisé.

```
#Création d'un objet StandardScaler pour mettre à la même échelle les variables numériques
```

```
numTransformer = StandardScaler()
```

Appliquons maintenant ces transformateurs. Au lieu d'appliquer séparément chaque transformateur, nous allons utiliser la fonction [*ColumnTransformer*](#)⁶¹ qui nous permettra d'appliquer à la fois les deux transformateurs en les regroupant dans un même objet.

```
#Features
```

```
X = df.loc[:, 0:14]
```

```
#Target
```

```
y = df.loc[:, 15]
```

```

#Variables catégorielles

catVariables = X.select_dtypes(include=['object']).columns

#Variables numériques

numVariables = X.select_dtypes(exclude=['object']).columns

#Création d'un objet ColumnTransformer

preprocessor = ColumnTransformer(
    transformers=[

        ('numeric', numTransformer, numVariables),

        ('categoric', catTransformer, catVariables)])

```

Créons l'objet pipeline avec comme étape l'objet *preprocessor* :

```

#Création d'un objet pipeline qui inclut l'objet ColumnTransformer

pipeline_object = Pipeline(steps=[('preprocessing', preprocessor),
                                  ('dim_red', PCA(n_components=10))])

```

ETAPE DE REDUCTION DE DIMENSIONNALITE

Mettons à jour notre objet pipeline en y incluant un objet PCA (*Principal Components Analysis*) comme deuxième étape :

```

pipeline_object = Pipeline(steps=[('preprocessing', preprocessor),
                                  ('dim_red', PCA(n_components=10))])

```

L'étape de modélisation permettra de conférer à cet objet pipeline le caractère d'un estimateur.

ETAPE DE CONSTRUCTION DU MODELE

Avant de passer à la modélisation, divisons la dataframe en données d'entraînement et d'évaluation de modèle.

```
#Train/Test set

seed = 42

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = seed, test_size = 0.2)
```

- **UN SEUL ALGORITHME DE CLASSIFICATION**

Mettions à jour notre objet pipeline en y incluant un modèle de classification utilisant l'algorithme de régression logistique.

```
pipeline_object = Pipeline(steps=[('preprocessing', preprocess),
                                 ('dim_red', PCA(n_components =10)),
                                 ('lr_model', LogisticRegression(random_state = seed))])
```

Le fait d'avoir ajouté un algorithme de classification en dernière étape à l'objet pipeline lui confère les caractéristiques d'un classificateur (ici Régression Logistique)

Maintenant nous pouvons utiliser l'objet pipeline comme estimateur et lui appliquer les méthodes *fit()* et *predict()* et terminer par le calcul de certaines métriques pour évaluer le modèle.

```
#Application du pipeline aux données d'entraînement
```

```
pipeline_object.fit(X_train, y_train)
```

Dans un premier temps, les données d'entraînement subiront le prétraitement (première étape du Pipeline). Dans un second temps elles subiront une réduction de dimensions grâce à l'objet PCA et en troisième temps elles seront passées dans l'algorithme de Régression logistique.

Pour évaluer le modèle, on peut tout simplement utiliser la méthode *score()* comme suit :

```
#Score du modèle
```

```
print("Accuracy on Training Data:", round(pipeline_object.score(X_train, y_train), 2))
```

```
print("Accuracy on Test Data:", round(pipeline_object.score(X_test, y_test), 2))
```

Le modèle donne un score de 89% sur le train data contre 85% pour le test data.

Passons aux prédictions du modèle :

```
# Prédictions  
  
yhat = pipeline_object.predict(X_test)
```

On peut dès lors dresser un rapport complet d'évaluation du modèle :

```
#Rapport de classification  
  
print(classification_report(y_test, yhat))
```

	precision	recall	f1-score	support
0	0.86	0.88	0.87	74
1	0.84	0.81	0.82	57
accuracy			0.85	131
macro avg	0.85	0.84	0.84	131
weighted avg	0.85	0.85	0.85	131

Selon ce rapport de performance du modèle, 85% des clients du test data ont été correctement classé comme solvables ou non. 88% des clients insolubles (classe 0) ont été correctement classés comme tel ce qui veut dire que 12% des clients insolubles ont été mal classés comme étant des clients solvables : c'est un risque de perte d'argent de l'ordre de 12% pour la banque.

Par ailleurs 81% des clients solvables (classe 1) ont été classé comme tel ce qui veut dire qu'environ 19% des clients solvables ont été mal classés comme étant des clients insolubles : c'est une perte d'opportunité pour la banque de l'ordre de 19%.

• PLUSIEURS ALGORITHMES DE CLASSIFICATION

Dans un projet de Data Science, il est très rare d'essayer un seul modèle pour tirer des conclusions. Plusieurs algorithmes sont entraînés par les données. Le modèle le plus performant est sélectionné sur la base du calcul d'un paramètre d'évaluation préalablement défini selon le type de Business.

Grâce à une boucle *for*, nous allons entraîner plusieurs modèles et évaluer leur performance. Voici le code que vous pouvez écrire pour ce genre de tâche :

```

#Création d'une liste d'algorithmes de classification
classifiers = [LogisticRegression(random_state=seed),
                RandomForestClassifier(random_state=seed),
                AdaBoostClassifier(random_state=seed),
                KNeighborsClassifier(n_neighbors=5) ]

#Entraînement et évaluation de chacun des algorithmes de la liste ci-dessus
for algorithm in classifiers:
    pipeline_object = Pipeline(steps=[('preprocessing', preprocessor),
                                       ('dimred', PCA(n_components=10)),
                                       ('model', algorithm)])

    #Fit
    pipeline_object.fit(X_train, y_train)

    #Score
    print(algorithm)
    print("Accuracy on Training Data:", round(pipeline_object.score(X_train, y_train), 2))
    print("Accuracy on Test Data:", round(pipeline_object.score(X_test, y_test), 2))
    print("—" * 90)

    LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                        intercept_scaling=1, l1_ratio=None, max_iter=100,
                        multi_class='auto', n_jobs=None, penalty='l2',
                        random_state=42, solver='lbfgs', tol=0.0001, verbose=0,
                        warm_start=False)
    Accuracy on Training Data: 0.89
    Accuracy on Test Data: 0.85
    -----
    RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                           criterion='gini', max_depth=None, max_features='auto',
                           max_leaf_nodes=None, max_samples=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_jobs=None, oob_score=False, random_state=42, verbose=0,
                           warm_start=False)
    Accuracy on Training Data: 1.0
    Accuracy on Test Data: 0.82
    -----
    AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                       n_estimators=50, random_state=42)
    Accuracy on Training Data: 0.93
    Accuracy on Test Data: 0.82
    -----
    KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                         metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                         weights='uniform')
    Accuracy on Training Data: 0.89
    Accuracy on Test Data: 0.82
    -----

```

Vous pouvez essayer plusieurs autres algorithmes avec leurs hyperparamètres par défaut. L'idée est de pouvoir sélectionner les algorithmes qui sont performants aussi bien sur le train data que sur le test data. La seule contrainte possible est le temps d'exécution qui est directement rattachée à la puissance de l'ordinateur que vous utilisez. Si vous utilisez une machine de guerre :), par exemple un core i9 avec 64 GB de ram 😊, faites libre cours à votre imagination. De toute façon, il y a une multitude d'algorithmes de Machine Learning.

En une seule fois, nous avons entraîné 04 algorithmes de classification ce qui fait un gain de temps énorme (On pouvait même inclure d'autres algorithmes). Nous retrouvons les mêmes scores pour la régression logistique.

Au vu des scores des différents modèles sur le training et le test set, le modèle de **régression logistique** semble être le plus performant.

Les algorithmes ci-dessus ont été entraîné avec leurs paramètres par défaut. L'une des étapes clé d'un projet de Data Science consiste à affiner un modèle en essayant plusieurs hyperparamètres (**Hyperparameters Tuning**). Cette tâche est non seulement fastidieuse mais très consommatrice en temps. De plus, il faut aussi effectuer une [cross-validation](#)⁶² afin de se prémunir contre le surapprentissage ([Overfitting](#)⁶³).

Heureusement, cette fois ci-encore nous avons pipeline 😊. Nous allons implémenter dans un pipeline toutes ces tâches de recherche des meilleurs hyperparamètres et de cross-validation au niveau de l'algorithme de Régression Logistique afin d'essayer d'augmenter sa performance.

```
#Création d'un pipeline avec RandomForestClassifier

pipe = Pipeline(steps=[('preprocessing', preprocessing),
                      ('dimred', PCA()),
                      ('classifier', LogisticRegression(rando
m_state=seed))])

#Création d'un dictionnaire regroupant les paramètres à tourne
r

param_grid ={'dimred_n_components' : range(2, 16),
             'classifier_penalty' : ['l1', 'l2'],
             'classifier_C' : [1,3, 5],
             'classifier_solver' : ['liblinear'],
             'classifier_max_iter' : [100, 500]}
```

```

#Création d'un estimateur avec GridSearchCV (Combinaison de Grid Search et de Cross-Validation)
estimator = GridSearchCV(pipe, cv=10, param_grid=param_grid)

#Entraînement de cet estimateur
estimator.fit(X_train,y_train)

#Meilleur score et meilleurs paramètres
print("Meilleur score obtenu: %f avec les hyperparamètres : %s"
" % (estimator.best_score_, estimator.best_params_ ) )

```

Notez qu'avec le pipeline, on a pu défini un espace de recherche aussi bien pour les hyperparamètres de l'algorithme mais aussi une gamme de nombre de composantes principales.

Puisque l'argument *refit* de la fonction *GridSearchCV* est par défaut égal à *True*, alors le modèle est automatiquement réentraîné avec les meilleurs hyperparamètres trouvés. On peut donc effectuer directement des prédictions et aussi dresser le rapport de classification :

```

#Prédiction sur le test set avec le meilleur estimateur
pred = estimator.predict(X_test)

#Rapport de classification
print(classification_report(pred, y_test))

```

	precision	recall	f1-score	support
0	0.80	0.90	0.85	68
1	0.87	0.76	0.81	63
accuracy			0.83	131
macro avg	0.84	0.83	0.83	131
weighted avg	0.84	0.83	0.83	131

Selon ce rapport de performance du modèle, 84% (contre un score de 85% avec les paramètres par défaut) des clients du test data ont été correctement classé comme solvables ou non . Avec ce modèle tourné, on prédit mieux les clients insolubles. En effet 90% des clients insolubles (classe 0) ont été correctement classés comme tel ce qui veut dire que 10% des clients insolubles ont été mal classés comme étant des clients solvables : c'est un risque de perte d'argent de l'ordre de 10% (contre 12% avec le modèle par défaut) pour la banque.

En revanche, on perd en justesse sur la prédiction des clients solvables. 76% (contre 81% avec le modèle par défaut) des clients solvables (classe 1) ont été classé comme tel ce qui veut dire qu'environ 24% des clients solvables ont été mal classés comme étant des clients insolubles : c'est une perte d'opportunité pour la banque de l'ordre de 24%.

Avec le premier modèle de régression logistique (hyperparamètres par défaut), la Banque a un risque de perte d'argent de l'ordre de 12% contre 10% avec le deuxième modèle (modèle tourné). Par ailleurs avec le premier modèle, la Banque a un risque de perte d'opportunité de gagner de l'argent (intérêts sur les crédits) de l'ordre de 19% contre 24% avec le deuxième modèle.

RESUME DES ETAPES DE CONSTRUCTION DU PIPELINE

- ✓ Préparation du jeu de données ;
- ✓ Première étape du pipeline : Prétraitement (*Preprocessing*) comprenant transformation des variables catégorielles (*Encoding*) et Mise à l'échelle (*Scaling*) ;
- ✓ Deuxième étape du pipeline : Réduction de dimensionnalité (Analyse en Composantes Principales) ;
- ✓ Troisième étape du pipeline : Modélisation ;
- ✓ Quatrième étape : Recherche des meilleurs hyperparamètres (*Grid Search*).

CONCLUSION

Dans ce projet j'ai montré, à travers une étude de cas (Modélisation du risque de crédit), qu'un pipeline peut regrouper trois (03) incontournables étapes d'un projet de Data Science à savoir :

- Pré-traitement et transformation des données ;
- Construction d'un ou de plusieurs modèle(s) de Machine Learning ;
- Recherche des meilleurs hyperparamètres.

Ainsi, le code de ce projet peut être appliqué à un nouveau jeu de données ce qui fait un gain énorme de temps et d'efficacité.

Il y a plusieurs autres manières d'utiliser un Pipeline. Je vous conseille de consulter cette [documentation⁶⁴](#) de Scikit-Learn qui est très riche en exemples pratiques.

La Data Science et le Machine Learning sont accessibles à tout le monde. Je veux que ce livre touche le maximum de personnes. Aidez-moi à réaliser cet objectif en mettant vos avis, commentaires sur le site dans lequel vous l'avez acheté. Vous pouvez aussi le recommander à vos proches qui s'intéressent au domaine de la DATA. Pour les personnes qui veulent bénéficier d'un accompagnement personnalisé, n'hésitez pas à me contacter via cette adresse : j.a.datatech.consulting@gmail.com

Je suis également disposé à collaborer en tant que Consultant Formateur avec des structures intervenant dans le domaine de la formation professionnelle sur des thématiques liées aux données et à la programmation informatique.

Josué AFOUDA

WEBOGRAPHIE

1. https://fr.wikipedia.org/wiki/Apprentissage_automatique
2. https://fr.wikipedia.org/wiki/Intelligence_artificielle
3. https://fr.wikipedia.org/wiki/Big_data
4. https://www.youtube.com/watch?time_continue=138&v=rYGHfAvXHQ&feature=emb_logo
5. [https://fr.wikipedia.org/wiki/R%C3%A9gression_\(statistiques\)](https://fr.wikipedia.org/wiki/R%C3%A9gression_(statistiques))
6. https://www.youtube.com/watch?time_continue=2&v=AfyoSOTB4M&feature=emb_logo
7. https://www.amazon.fr/programmer-langage-appliqu%C3%A9-analyse-donn%C3%A9es-ebook/dp/B08CY2XNXX/ref=sr_1_2?__mk_fr_FR=%C3%85M%C3%85%C5%BD%C3%95%C3%91&dchild=1&keywords=afouda&qid=1596219235&sr=8-2
8. <https://archive.ics.uci.edu/ml/datasets/automobile>
9. <https://fr.wikipedia.org/wiki/Essieu>
10. <https://raw.githubusercontent.com/JosueAfouda/Boston-Housing/master/housing.names>
11. <https://seaborn.pydata.org/generated/seaborn.pairplot.html>
12. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
13. https://en.wikipedia.org/wiki/Least_squares
14. [https://en.wikipedia.org/wiki/Cross-validation_\(statistics\)](https://en.wikipedia.org/wiki/Cross-validation_(statistics))
15. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html
16. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html
17. https://en.wikipedia.org/wiki/F1_score
18. <https://www.who.int/cancer/detection/breastcancer/en/index1.html>
19. [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Diagnostic\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))
20. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html
21. https://fr.wikipedia.org/wiki/M%C3%A9thode_des_k_plus_proches_voisins
22. <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

23. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
24. <https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data>
25. <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html>
26. <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html>
27. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
28. <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics>
29. <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>
30. https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html#sphx-glr-auto-examples-cluster-plot-cluster-comparison-py
31. <https://scikit-learn.org/stable/modules/clustering.html#k-means>
32. <https://archive.ics.uci.edu/ml/datasets/iris>
33. http://jse.amstat.org/jse_data_archive.htm
34. <http://jse.amstat.org/datasets/fishcatch.txt>
35. [https://en.wikipedia.org/wiki/Elbow_method_\(clustering\)#:~:text=In%20cluster%20analysis%2C%20the%20elbow,number%20of%20clusters%20to%20use.](https://en.wikipedia.org/wiki/Elbow_method_(clustering)#:~:text=In%20cluster%20analysis%2C%20the%20elbow,number%20of%20clusters%20to%20use.)
36. <https://archive.ics.uci.edu/ml/datasets/wine>
37. <https://fr.wikipedia.org/wiki/Scikit-learn>
38. <https://raw.githubusercontent.com/JosueAfouda/Internet-advertisements/master/ad.data>
39. https://raw.githubusercontent.com/JosueAfouda/Internet-advertisements/master/ad_doc.DOCUMENTATION
40. <https://raw.githubusercontent.com/JosueAfouda/Internet-advertisements/master/ad.names>
41. <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>
42. https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average
43. https://fr.wikipedia.org/wiki/Dow_Jones_Industrial_Average
44. https://en.wikipedia.org/wiki/Time_series

45. https://finance.yahoo.com/quote/%5EDJI/history?period1=1196706600&period2=1512325800&interval=1d&filter=history&frequency=1d&guccounter=1&guce_referrer=aHR0cHM6Ly9zdWJzY3JpcHRpb24ucGFja3RwdWIuY29tL2Jvb2svYmlnX2RhdGFFYW5kX2J1c2luZXNzX2ludGVsbGlnZW5jZS85NzgxNzg4MzkwMDQwLzIvY2gwMmx2bDFzZWMyMy9jb2xsZWN0aW5nLXRoZS1kYXRhc2V0&guce_referrer_sig=AQAAAJOsCwEGFKsdgIaYSkoBHDORhG5VqZSF_OahJgxWoSekyzIJz_wCbkGewE-3jfUkZZ7QmJG-U4sFrcGsrXj4zsS1X3bCkaQ_bxbG90dVubNhg8_SwRFcN10tdDnti0mnRbkunhnokXTHMCr9-6vsOuR4vIEAvtIzsWSKjpJxrmJ
46. https://raw.githubusercontent.com/JosueAfouda/Stock-Market-Price-Prediction/master/dow_jones_2007_2017.csv
47. https://en.wikipedia.org/wiki/Dickey%20%93Fuller_test
48. https://en.wikipedia.org/wiki/Autoregressive_model
49. https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average
50. <https://towardsdatascience.com/significance-of-acf-and-pacf-plots-in-time-series-analysis-2fa11a5d10a8>
51. https://en.wikipedia.org/wiki/Partial_autocorrelation_function
52. https://fr.wikipedia.org/wiki/Crit%C3%A8re_d'information_d'Akaike
53. <https://www.quantopian.com/>
54. <https://numer.ai/>
55. <http://epistasislab.github.io/tpot/using/>
56. <https://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center>
57. <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.pipeline>
58. <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>
59. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html>
60. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>
61. <https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html>
62. https://scikit-learn.org/stable/modules/cross_validation.html
63. <https://en.wikipedia.org/wiki/Overfitting>

64. <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#examples-using-sklearn-pipeline-pipeline>