Nantes Université

IQSPOT
Ensemble vers l'immobilier durable

**Master 2 Ingénierie Statistiques**

# Prédiction de la durée de vie restante des batteries de capteurs

**Rapport de Stage de fin d'études dans l'entreprise iQspot**


***Auteur :***
Pauline DUGRÉ

***Encadrant :***
Myriam LOPEZ

***Professeur référent:***
Aymeric STAMM


28 Août 2024

# Remerciements

Je voudrais dans un premier temps remercier, mon responsable de stage STAMM Aymeric, enseignant chercheur à l'université de Nantes, ainsi que toute l'équipe enseignante pour leur encadrement tout au long de cette formation.

Je tiens à remercier tout particulièrement ma tutrice de stage, LOPEZ Myriam, pour sa bienveillance, sa disponibilité, son accompagnement et surtout pour ses conseils, qui ont contribué à alimenter ma réflexion et m'ont permis de progresser. Je lui suis aussi très reconnaissante pour tout le temps qu'elle a accordé à la lecture de mon rapport de stage. Ses remarques ont été très précieuses.

Je remercie également toute l'équipe iQspot qui m'a donné l'opportunité d'effectuer ce stage de fin d'étude. J'ai apprécié son accueil convivial et sa bonne humeur de tous les jours.

Je tiens également à exprimer ma reconnaissance au pôle développement avec qui j'ai partagé le bureau et qui n'a pas hésité à m'apporter de l'aide.

Et enfin, je souhaite adresser mes remerciements à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce stage.

# Table des matières

# Table des sigles et des abréviations

1. **SVR** : Support Vector Regression
2. **MSE** : Mean Square Error
3. **MAE** : Mean Absolute Error
4. $\mathbf{R}^2$ : Coefficient de détermination
5. **RSSI** : Received Signal Strength Indicator
6. **SNR** : Signal Noise Ratio
7. **SF** : Spread Factor

# Introduction

Au fil des années, les villes ont vu émerger une multitude d'infrastructures diverses, allant des bureaux et commerces aux entrepôts logistiques, hôtels et services. Cette variété d'infrastructures, essentielle au développement urbain et économique, est regroupée sous le terme "Immobilier Tertiaire". Bien que ces infrastructures soient utiles dans la vie de tous les jours elles sont malheureusement aussi la cause de grandes consommations énergétiques.

En effet d'après le Ministère de la transition écologique [1], aujourd'hui l'immobilier tertiaire représente 40% des consommations énergétiques et 20% des émissions de gaz à effet de serre de l'immobilier global. Au-delà du fait d'avoir un impact écologique, en 2019 le secteur tertiaire a dépensé près de 22,5 milliards d'euros pour sa consommation finale d'énergie.[2] Ainsi l'immobilier tertiaire soulève deux problématiques principales : une problématique écologique et économique.

Il est primordial de répondre aux impératifs de réduction des consommations énergétiques et des émissions de gaz à effet de serre des bâtiments, tout en facilitant la mise en conformité avec les réglementations environnementales. Cet objectif inclut le respect des normes en matière d'efficacité énergétique, de gestion des déchets et de réduction des émissions de carbone. Il est également essentiel de valoriser les actifs des entreprises en investissant dans des infrastructures spécifiques et des technologies adaptées, qui permettent de rendre les bâtiments plus économes en énergie et de réduire la pollution, contribuant ainsi à diminuer les coûts d'exploitation.

Pour répondre à ces objectifs de réduction des consommations énergétiques, des émissions de gaz à effet de serre et de mise en conformité avec les réglementations environnementales, l'entreprise iQspot, dans laquelle j'ai effectué mon stage de fin d'études, propose une solution basée sur l'utilisation de la technologie. Concrètement, l'entreprise installe des capteurs sur les compteurs d'énergie des bâtiments, permettant ainsi de suivre leur consommation en temps réel. Ces capteurs, au cœur du système, jouent un rôle essentiel dans le suivi des performances énergétiques des bâtiments. Ma mission lors de ce stage a été de développer un moyen d'anticiper la fin de vie des batteries de ces capteurs, en analysant leur évolution dans le temps pour fournir à l'entreprise des prévisions précises sur leur durée de vie restante.

Le plan que je vais suivre lors de ce rapport est le suivant : Après une présentation de l'entreprise et une explication détaillée du fonctionnement des capteurs, nous aborderons les objectifs spécifiques de ce projet. Ensuite, nous décrirons les sources de données et les étapes de prétraitement appliquées. L'analyse se poursuivra avec l'application de méthodes d'apprentissage supervisé, ainsi que la présentation des différents modèles utilisés, pour enfin conclure par une évaluation des résultats obtenus.

# 1 Contexte et Objectifs

## 1.1 Présentation de l'entreprise

Située à Bordeaux, iQspot est une entreprise créé en 2015 par deux ingénieurs, Julien Bruneau et Quentin Enard, voulant mettre au profit de l'écologie leurs connaissances technologiques afin d'accélérer la transition immobilière durable.

L'entreprise s'articule autour de quatre pôles principaux. Voici un organigramme récapitulatif de ces pôles Figure 1 :

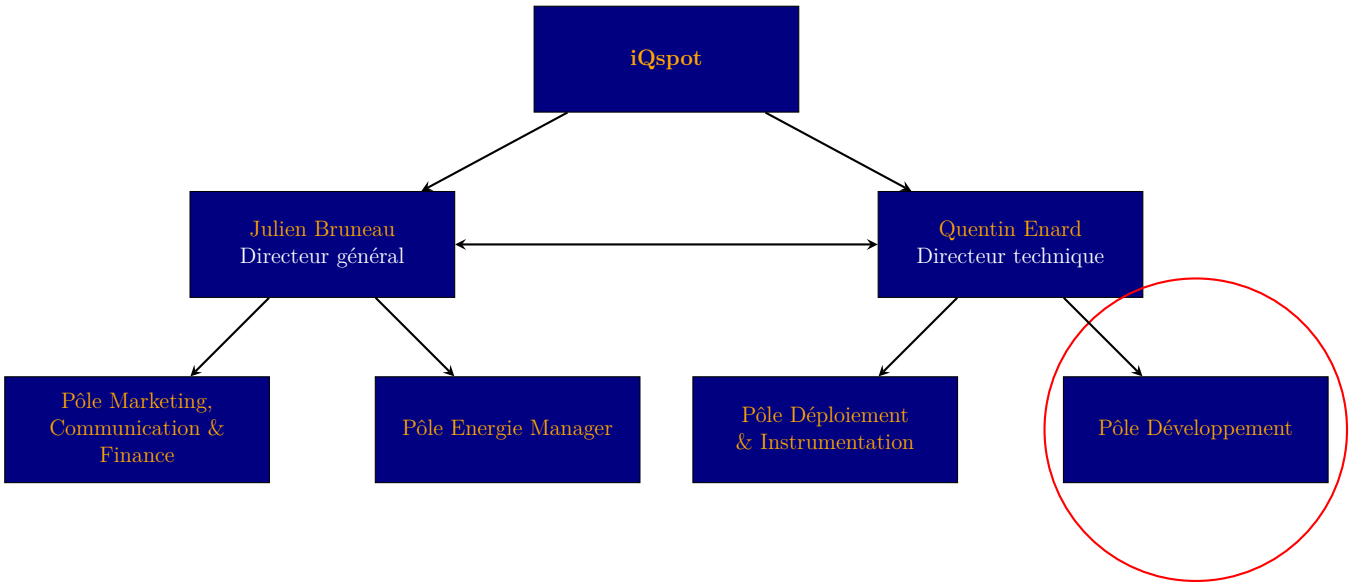



Figure 1: Organigramme d'iQspot

- Le pôle marketing, communication et finance a pour rôle de rendre l'entreprise visible auprès des clients actuels mais surtout auprès de futurs clients potentiels.
- Le pôle énergie management analyse les données des consommations des différents bâtiments en temps réel et accompagne les clients pour réduire leurs consommations grâce à des réglages de leurs équipements et ainsi décarboner leur parc immobilier.
- Le pôle déploiement et instrumentation s'occupe d'installer les capteurs sur les compteurs des bâtiments et d'assurer leur maintenance.
- Le pôle développement, travaille à l'élaboration de la plateforme fournie ainsi qu'à sa maintenance

Bien que le siège de l'entreprise se trouve à Bordeaux, l'entreprise possède aussi une antenne sur Paris. Ces 4 pôles sont donc répartis dans ces deux villes. Pour ma par j'ai pour ce stage rejoint le "pôle développement" supervisé par Quentin Enard, en tant que stagiaire data scientist sous la direction de ma tutrice Myiam Lopez.

Pour répondre à ces enjeux, iQspot a développé et commercialisé non seulement un outil de suivi en temps réel des consommations énergétiques des bâtiments, mais également un service d'accompagnement personnalisé. Ce service, assuré par une équipe d'experts en efficacité énergétique, aide les clients à interpréter les données collectées et à mettre en œuvre des actions concrètes pour réduire l'impact environnemental de leurs bâtiments, tout en maintenant le confort des locataires. Ce double dispositif apporte une solution complète aux investisseurs immobiliers, leur permettant de collecter, analyser, et optimiser la gestion énergétique de leur parc immobilier. On appelle parc immobilier l'ensemble des propriétés immobilières qui appartiennent à la même société. iQspot compte aujourd'hui une trentaine de clients et pilote en temps réel 535 bâtiments correspondant à plus de 4,1 millions de m² de parc immobilier, répartis dans cinq pays en Europe. Mais surtout iQspot a permis d'éviter l'émission de plus de 24 609 tonnes de CO2.

Ainsi, iQspot collecte les données énergétiques en temps réel depuis les compteurs d'énergie des bâtiments mais peut aussi agréger des données antérieures sur la base de factures ou encore de données des fournisseurs d'énergie. Pour collecter ces données depuis les compteurs, iQspot installe des capteurs sur les bâtiments afin de suivre les consommations énergétiques (électricité, gaz, réseau urbain) en eau et des indicateurs de confort tels que le CO2, la température intérieure, l'humidité, l'éclairage, le bruit et même le suivi de la production de déchets.

Ces données sont ensuite consultables sur la plateforme App iQspot, représentée par la Figure 2. L'application est accessible aux clients, leur permettant de valoriser leurs données sous forme de courbes de suivi en temps réel et de répondre aux exigences réglementaires.

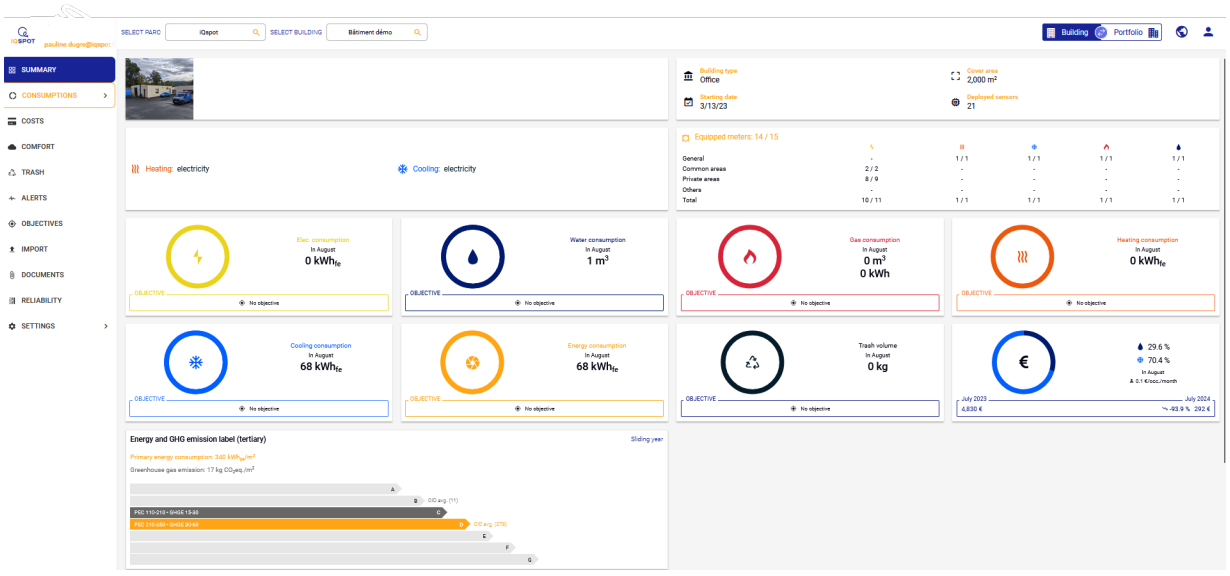



Figure 2: Illustration d'une partie de l'application représentant la synthèse du suivi énergétique d'un bâtiment

Sur cette figure nous avons une représentation de l'onglet synthèse de l'application App iQspot. Cet onglet permet d'avoir une visualisation globale sur tous les types de consommations énergétiques présentes pour un bâtiment et la consommation relevée de chaque fluide par mois. Sur cette page nous avons aussi des renseignements sur le type de bâtiment, la surface couverte de celui-ci et le nombre de capteurs qui y sont déployés.

## 1.2 Problématique

Les capteurs sont au coeur de l'entreprise iQspot, un capteur dysfonctionnant ou n'ayant plus de batterie peut entraîner de mauvaises informations sur les données ou même tout simplement plus de données du tout. Si cela ce produit, la promesse de suivre les consommations énergétiques des bâtiment en temps réel faites par iQspot à ses clients est compromise. En moyenne un capteur vie 5 ans, toutefois il est possible que certain capteurs arrêtent de fonctionner avant.

Ainsi une solution doit être trouvée pour empêcher l'arrêt inattendu de la batterie des capteurs et la perte de données entraînée par la même occasion. Une seule solution est possible:
"prédire l'épuisement de la batterie des capteurs et leur temps de vie restant".

Actuellement au sein de la boîte deux modèles sont déjà mis en place pour apporter des informations sur l'état actuel des batteries des capteurs. Un premier modèle constructeur qui calcul la batterie restante en faisant une différence normalisée entre la tension aux bornes de la pile du capteur, et la tension minimale nécessaire. Et un second modèle physique proposé par l'équipe instrumentation qui a mis en place un calcul de la durée de vie de la batterie en fonction de plusieurs variables comme la capacité de la pile ou du courant de décharge. Toutefois les résultats obtenus par ces deux modèles sont loin d'être fiables à 100%. De nombreux résultats sont aberrants et ne peuvent pas être pris en compte.

## 1.3 Objectifs

La mission qui m'a été donnée pour ce stage a pour objectif d'apporter une nouvelle solution qui sera de préférence meilleure que les deux déjà proposés, sinon complémentaire. En effet mon rôle de stagiaire data scientist va permettre d'aider l'entreprise à avoir une idée de la durée de vie restante des capteurs ainsi que plus d'informations sur l'évolution de la batterie dans le temps. Trouver un troisième modèle qui pourra répondre à ces attentes va donc permettre:

- d'éviter au maximum la perte de données
- optimiser les déplacements prévu pour changer les capteurs

En effet cela permettrait de prendre en charge les équipements en fin de vie, et d'anticiper leur remplacement, avant que ces derniers soient définitivement obsolètes. Ceci entraîne donc un gain de temps, un gain d'argent et une baisse de pollution apporté par les déplacements en transports.

# 2 Présentation des capteurs

## 2.1 Modèles de capteurs

Comme expliqué précédemment les capteurs sont installés sur les différents compteurs des bâtiments afin de collecter leurs consommations énergétiques, que ce soit pour l'eau, l'électricité, le gaz, ou encore l'énergie thermique. Toutefois la manière dont les capteurs récupèrent les données est différente selon le type de compteur. Pour cela il existe différents modèles de capteurs, mais trois ressortent tout particulièrement : les modèles Flash'O, Fludia et Sens'O.

- Les capteurs **Flash'O** sont utilisés pour des compteurs électriques équipés de LED, et captent les impulsions lumineuses émises. Pour en déduire les consommations, il faut alors multiplier chaque impulsion lumineuse par le poids d'impulsion du compteur qui est en Watt heure (Wh) correspondant au nombre de Wh consommé par l'impulsion lumineuse.
- Les capteurs **Fludia** sont d'autres modèles utilisés pour la collecte des consommations électriques, cependant ils s'adaptent aussi bien aux compteurs à LED qu'aux compteurs à roue.
- Les capteurs **Sens'O** représenté sur la figure ci-dessous Figure 3, sont des capteurs utilisés pour relever les consommations sur les compteurs d'eau, de gaz ou d'énergie thermique. Ces capteurs sont waterproof car ils sont installés dans des zones humides voire immergées, ou près des compteurs de gaz. Ils peuvent aussi être utilisés pour lire les consommations électriques. La particularité de ces capteurs c'est qu'ils peuvent relever plusieurs index, et donc les données de plusieurs compteurs à la fois. Nous reviendrons plus tard sur cette notion d'index. De plus il existe 3 types de Sens'O : Sens'O IP68, Pulse Sens'O et Sens'O ATEX.


Figure 3: Exemple d'un capteur Sens'O

## 2.2 Différents Réseaux

Une fois que les capteurs ont relevé les données de consommation elles sont transmises vers des serveurs centraux ou des services cloud où elles peuvent être stockées, analysées puis utilisées. Cette transmission des données est gérée via des réseaux "IoT" : *Internet of things*, publics comme LoRa, Orange et Objenious. Toutefois dans certaines zones des bâtiments, le signal de ces réseaux peut être faible ou inexistant. Pour pallier ce problème, des passerelles réseau nommées *gateways* disposant d'une antenne externe, sont utilisées pour permettre aux équipements de bénéficier d'un bon signal dans les zones mal couvertes et garantir une transmission fiable des données, même dans ces zones. Ainsi le rôle des passerelles est d'envoyer les données des capteurs vers un serveur central ou un service cloud où elles sont analysées et stockées. Les capteurs peuvent alors utiliser les réseaux "IoT" publics, mais aussi le réseau LoRa privé dans le cadre des *gateways*. De plus, il est possible d'associer aux gateways un nombre quasi infini de capteurs. Toutefois même s'il existe plusieurs réseaux les données des capteurs ne sont pas nécessairement envoyées sur un seul, elles peuvent être envoyées simultanément sur deux voire trois réseaux.

## 2.3 Collecte des données

Ainsi avant que iQspot n'ai accès à l'ensemble des données des capteurs il faut passer par plusieurs étapes. La première étape est l'audit effectué par les "Energy managers".
En effet, l'installation d'un nouveau bâtiment ne se résume pas simplement à la pose de capteurs. L'audit consiste à répertorier l'ensemble des compteurs à suivre, ainsi qu'à définir un plan de comptage. On appelle plan de comptage une stratégie visant à mesurer, analyser et représenter les zones et les consommations à suivre dans le bâtiment. Il sert par la suite de base pour définir l'installation des capteurs. En effet comme expliqué précédemment un capteur peut être relié à un seul ou plusieurs compteurs, ceci est déterminé grâce au plan de comptage. Une fois les capteurs installés sur les compteurs, reste à collecter les données de ceux-ci. La collecte de données des compteurs par les capteurs ce fait à chaque impulsions lumineuses des compteurs. L'index dont on a parlé avant, permet de représenter quel compteur a émis une impulsion et indique le nombres de pulsions entre la dernière fois que le compteur a été allumé jusqu'au moment t. Une fois les données collectées, elles sont ensuite centralisées via les différents réseaux sur une base de données qui va être décrite dans la suite de ce rapport.

# 3 Matériel

## 3.1 Environnement de travail

Pendant mon stage, tous mes développements ont été réalisés en Python. J'ai utilisé l'environnement Anaconda pour organiser efficacement mon travail. Anaconda permet de créer des environnements indépendants, évitant ainsi les conflits de packages et facilitant l'installation des bibliothèques nécessaires telles que *NumPy, Pandas*, et *Matplotlib*, grâce à son gestionnaire de packages, conda. Pour le développement en Python, j'ai principalement travaillé sur l'IDE (*environnement de développement intégré*) Spyder. De plus, pour la gestion du code source et la collaboration avec l'équipe, j'ai utilisé la plateforme Bitbucket. Bitbucket est une plateforme de gestion de code source qui permet aux développeurs de collaborer sur des projets de développement logiciel de manière efficace et structurée.

Pour accéder aux données des différents capteurs, l'entreprise utilise MongoDB, une base de données NoSQL orientée documents. Les bases de données NoSQL sont conçues pour gérer de grandes quantités de données diversifiées, non structurées ou semi-structurées, et offrent une grande flexibilité. Contrairement aux bases de données SQL, elles n'ont pas de schéma fixe, ce qui signifie que chaque document peut avoir une structure différente. De plus, elles peuvent facilement s'étendre en ajoutant des serveurs. En effet, les bases de données SQL utilisent des tables pour stocker des données de manière relationnelle, avec des structures organisées en lignes et colonnes et des schémas rigides.

MongoDB stocke donc les données sous forme de documents analogue à une ligne dans une base de données relationnelle, permettant des structures de données variées au sein d'une même collection. Les collections, équivalentes aux tables dans une base de données relationnelle, sont des ensembles de documents de structures flexibles contenant des paires clé-valeur. Chaque paire clé-valeur se compose d'une clé, un identifiant unique référenciant la valeur associée, et d'une valeur, l'information ou les données associées à la clé.

Dans le cas de l'entreprise une collection représente donc un réseau sur lequel les données des capteurs ont été envoyées et un document contient l'ensemble des données d'un capteur (voir Figure 4 ).

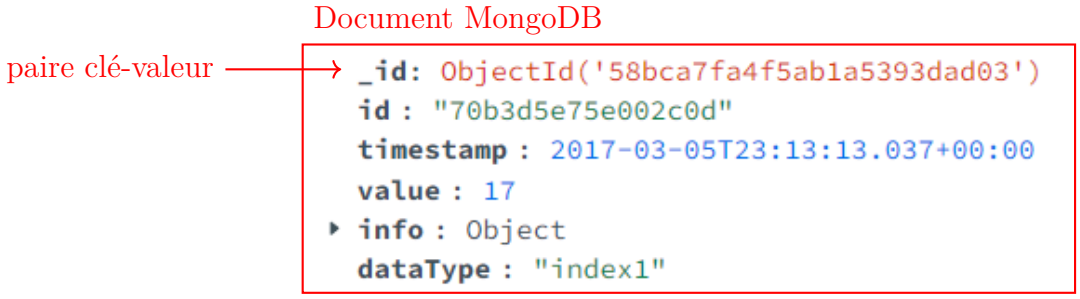



Figure 4: Visualisation d'un document MongoDB

Pour mon sujet de stage se sont donc les collections NkeLora, Objenious et Orange qui vont m'intéresser.

## 3.2 Description de la base de donnée

Chaque capteur utilisé dans cette étude est capable d'émettre plusieurs indicateurs différents et à différentes fréquences. Cependant, en fonction du modèle de capteur, ces indicateurs ne sont pas représentés de manière uniforme dans l'ensemble de la population de capteurs. En d'autres termes, la présence ou la fréquence de certains indicateurs peut varier en fonction du modèle de capteur ou des conditions d'enregistrement. Face à cette hétérogénéité, nous avons choisi de réduire le champ des indicateurs à exploiter. Les indicateurs sélectionnés, qui sont représentatifs et suffisamment disponibles dans notre ensemble de données ont été retenus pour la construction de mon jeu de données. Ils seront détaillés dans les sections suivantes.

- L'indicateur de type datetime : horodatage

  → L'**horodatage** indique la date, l'heure précise auxquelles une mesure a été enregistrée par le capteur, ainsi que le fuseau horaire car certains bâtiments peuvent être à l'étranger.

- Les indicateurs de type chaîne de caractère : message et données brutes.

  → L'indicateur **message** indique le type de message envoyé par le capteur

  → L'indicateur **données brutes** représente les informations originales, non traitées et non transformées, collectées directement à partir de la source. Les valeurs de l'indicateur données brutes sont des chaînes hexadécimales dont la taille dépend du type d'information envoyée par le capteur.

- Les indicateurs de type flottant : batterie et SNR

  → L'indicateur **batterie** représente le niveau de batterie du capteur au moment de la mesure.

  → L'indicateur **SNR** *Signal Noise Ratio*, correspond au résulat du rapport $\frac{signal}{bruit}$ et s'exprime en décibel (dB).

- Les indicateurs de type entier : index, RSSI et SF

  → L'indicateur **index** permet de distinguer les différents compteurs associés à un capteur, si celui-ci surveille plusieurs compteurs.

  → L'indicateur **RSSI** *Received Signal Strength Indicator*, utilisé pour mesurer la puissance de réception du signal radio en décibels-milliwatts (dBm) issu d'une antenne, et fournit l'intensité du signal reçu sur une échelle de 0 à 5V. Ainsi plus le niveau de puissance du signal est élevé, plus le RSSI est fort, ce qui indique que la qualité du signal est meilleure. À l'inverse, un niveau de puissance de signal plus faible se traduit par un RSSI plus faible, indiquant une qualité de signal plus faible. Pour les capteurs alimentés par batterie, il est cependant conseillé d'éviter les valeurs RSSI inférieures à -97 dBm et de viser la valeur RSSI la plus élevée possible afin d'éviter les pertes de paquets de données par interférences et de prolonger la durée de vie de la batterie autant que possible. En effet lorsqu'un paquet de données est perdu par exemple lorsque le RSSI est faible,le capteur doit retransmettre ce paquet jusqu'à ce qu'il soit correctement reçu ou jusqu'à ce que le nombre maximum de tentatives de retransmission soit atteint. Chaque retransmission nécessite l'utilisation de l'énergie de la batterie car il doit rester actif plus longtemps

  → L'indicateur **SF** *Spreading Factor*, correspond au facteur d'étalement du signal, il est lié au mécanisme qui adapte la puissance d'émission et la vitesse de transmission aux conditions du réseau dans lesquelles se trouve l'objet. Le SF peut varier entre 7 et 12, ainsi, plus il sera grand, plus la portée du signal sera élevée, mais plus la transmission des données sera lente et donc plus il fera consommer de la batterie

Ainsi, dans mon analyse, j'exploite ces différents indicateurs afin de construire mon jeu de données. L'indicateur données brutes est exploité en extrayant la taille en octets des chaînes hexadécimales, sachant qu'un caractère correspond à un demi-octet. Par exemple, une chaîne typique de 26 caractères hexadécimaux équivaut à 13 octets. En général, plus il y a de données à traiter (en octets), plus la consommation de batterie du capteur augmente.

Concernant les indicateurs de qualité de signal (RSSI, SNR, SF), ils m'ont permis de comprendre comment la qualité de la communication affecte la consommation de la batterie. Par exemple, un RSSI faible ou un SF élevé entraînent une plus grande consommation d'énergie, ce qui réduit la durée de vie du capteur.

## 3.3 Existant

Pour rappel, l'objectif de mon stage chez iQspot est de prédire la durée de vie restante des batteries des capteurs. Actuellement, un indicateur fournis avec les capteurs et un modèle physique proposé par l'équipe instrumentation existent déjà pour répondre à cette problématique, mais aucun des deux n'est véritablement fiable.

Les capteurs sont équipés d'un indicateur par défaut, défini par le constructeur, qui mesure la tension de la pile. Cependant, cet indicateur s'est révélé peu fiable. En effet, les valeurs affichées ne permettent pas de suivre l'évolution progressive de l'état de la batterie. La tension reste stable au maximum, puis chute soudainement à 0 sans transition, ce qui rend impossible l'anticipation de la fin de vie de la batterie.

L'équipe a également proposé une solution basée sur un modèle physique, dont les paramètres sont fixés par des *a priori* fournis par le constructeur concernant la consommation en mAh. Cependant, ces paramètres fixes ne reflètent pas les conditions réelles d'utilisation des capteurs, qui varient selon l'environnement et la fréquence d'envoi des données.

Ainsi, face à ces limitations, l'objectif de mon stage a été de m'intéresser à une approche *data-driven*, qui repose principalement sur les données collectées par les capteurs pour orienter les décisions, stratégies, et actions. Pour que cette approche soit efficace, il est important de constituer un jeu de données cohérent et fiable. C'est sur la base de ces données que nous pourrons construire un modèle de prédiction capable de mieux anticiper la durée de vie restante des batteries, en tenant compte des conditions réelles d'utilisation.

# 4 Méthodologie

## 4.1 Création du jeu de données

L'objectif étant de prédire la durée de vie restante des batteries de ces capteurs, il est essentiel que le modèle s'entraîne sur des capteurs dont la batterie est déjà épuisée, en apprenant de leur historique. Cependant, pour rendre cette prédiction possible, nous avons besoin d'un indicateur de l'état de santé du capteur, une sorte de jauge telle que le processus de déclin ou de dégradation du capteur au fil du temps, permettant ainsi de mesurer la santé de la batterie. Malheureusement, nous ne disposons pas de données concrètes sur cet indicateur de santé. Pour surmonter cette contrainte, nous nous concentrons sur un groupe de capteurs pour lesquels nous sommes sûrs que leur état de santé est actuellement nul. En d'autres termes, ces capteurs sont considérés comme non fonctionnels, ce qui nous permet d'estimer leur état de santé à chaque moment de leur vie. Ces capteurs serviront donc de base pour l'entraînement du modèle.

Pour cela, l'équipe instrumentation m'a fourni une liste de 65 capteurs identifiés comme étant non fonctionnels. Les collections Nkelora, Objenious et Orange sont composées de 5 103 capteurs qui envoient des informations toutes les heures. Afin de pouvoir décrire la vie de ces 65 capteurs, j'ai constitué un jeu de données basé sur les mensualités. C'est-à-dire que chaque instance du jeu de données correspond à la représentation de la vie d'un capteur jusqu'à un instant t, t étant exprimé en mois. Le modèle est donc entraîné sur cette population uniquement, en se basant sur l'historique de leurs performances. Nous avons choisi de nous baser sur des données mensuelles plutôt que journalières, car en situation de production, il est peu probable qu'une prédiction quotidienne soit nécessaire.

Ainsi mon jeu de données sera constitué d'un ensemble X et de la variable à prédire Y. X représente l'ensemble des instances, où chaque instance, notée x_n, est un vecteur qui décrit l'historique de vie d'un capteur jusqu'à un instant t, t étant exprimé en mois. Chaque instance x_n peut donc être vue comme un instantané de l'historique de vie d'un capteur à un moment donné. La variable Y est la variable à prédire, correspondant à la durée de vie restante du capteur, en jours. Autrement dit, Y est l'étiquette des instances qui représente le nombre de jours restant avant que la batterie du capteur ne soit complètement épuisée, à partir du moment t. Plus t avance dans le temps, plus Y diminue, car le capteur se rapproche de la fin de sa vie. En conclusion, le modèle apprend à partir des instances dans X (notées x_n) pour prédire Y, en fonction de l'historique du capteur jusqu'à l'instant t.

| **Identifiant** | **Instance avec la valeur y associée** |
|---|---|
| ID1 | $x_1 \rightarrow Y_1 = 120$ jours |
| ID1 | $x_2 \rightarrow Y_2 = 90$ jours |
| \| | ... |
| ID1 | $x_n \rightarrow Y_n = 10$ jours |
| ID2 | $x_1^{'} \rightarrow Y_1 = 150$ jours |
| \| | ... |
| ID2 | $x_n^{'} \rightarrow Y_n = 5$ jours |

Table 1: Tableau représentatif de deux instances et la valeur y associée.

Cette Table 1 représente, de manière simplifiée, l'historique de vie d'un capteur, représenté par les instances, associé à une durée de vie restante, représentée par la variable Y.

## 4.2 Apprentissage supervisé

L'apprentissage supervisé consiste à utiliser des jeux de données étiquetés pour entraîner un modèle basé sur des algorithmes. Un algorithme est une suite d'instructions ou de règles précises qu'un ordinateur suit pour résoudre un problème ou effectuer une tâche. Dans le contexte de l'apprentissage supervisé, ces algorithmes sont conçus pour apprendre à partir des données, en classant les informations ou en prédisant des résultats.
L'apprentissage supervisé permet de traiter deux types de problèmes :

- **La classification** : Cette méthode produit une réponse dont les valeurs appartiennent à un ensemble fini de catégories. Elle consiste à regrouper des données en catégories distinctes. Au lieu de prédire une valeur continue, comme le fait la régression, la classification assigne chaque donnée à un groupe spécifique en fonction de ses variables.
- **La régression** : Cette méthode permet d'établir la relation entre les variables prédictives et la variable à prédire. Il est couramment utilisé pour générer des projections ou des prédictions basées sur l'apprentissage supervisé.

Dans le cadre de mon stage, l'objectif est de prédire le moment où les capteurs cesseront de fonctionner en raison de la décharge de la batterie. Bien que l'on aurait pu aborder ce problème en termes de classification, nous avons choisi d'utiliser une approche de régression. La régression est plus appropriée ici, car elle permet de prédire une valeur continue, c'est-à-dire le temps de vie restant avant l'arrêt du capteur.

Pour procéder à l'analyse en apprentissage supervisé, le jeu de données est divisé en un ensemble d'apprentissage et un ensemble de test. L'ensemble d'apprentissage est utilisé pour que les différents modèles s'entraînent et s'améliorent afin de produire le résultat souhaité. L'ensemble de test, quant à lui, est un jeu de données indépendant du jeu de données d'apprentissage utilisé pour évaluer la performance du modèle et vérifier sa fiabilité. L'ensemble de test sert donc principalement à évaluer les performances du modèle ajusté sur le jeu de données d'apprentissage. L'évaluation est réussie lorsque le modèle performe bien sur des données qu'il n'a jamais vues auparavant.

L'objectif est que la performance du modèle, une fois ajusté sur l'ensemble d'apprentissage, soit comparable à celle obtenue sur l'ensemble de test. En d'autres termes, nous visons à ce que l'erreur relative soit faible et que le modèle généralise bien, montrant des résultats similaires sur les données d'entraînement et de test. Pour cela, la performance du modèle est mesurée à l'aide de scores de performance, qui évaluent le nombre d'erreurs faites par le modèle sur les ensembles d'apprentissage et de test. Celle-ci permet de mesurer à quel point les prédictions du modèle s'éloignent des valeurs réelles. Pour minimiser cet écart, le modèle est ajusté jusqu'à ce que l'erreur soit suffisamment réduite.

Comme expliqué précédemment, l'apprentissage supervisé utilise des jeux de données étiquetés. Dans le cas d'un problème de prédiction, le jeu de données d'entraînement se compose de x_train, qui contient les variables de prédictions et de y_train, qui est la variable cible à prédire. Le même découpage est effectué sur le jeu de données de test. Ainsi, le modèle ajusté sur l'ensemble d'apprentissage (x_train, y_train) apprend à établir une relation mathématique entre les instances de l'ensemble X et la variable à prédire Y.

Cette relation mathématique est composée de paramètres induits par les données. Une fois que le modèle est formé, il est testé sur les variables prédictives de l'ensemble de test, x_test. Le modèle utilise les connaissances acquises lors de l'entraînement pour prédire la variable nommée y_pred, dont on souhaite les valeurs proches de celles de la variable y_test, la variable cible réelle du jeu de test. Pour vérifier cette similitude, on compare y_pred à y_test à l'aide de scores, qui seront développés dans la partie Évaluation.

De plus, afin d'obtenir des résultats plus stables et d'utiliser toutes les données pour l'entraînement, le jeu de données peut être divisé plusieurs fois en plusieurs sous-ensembles ou "folds" de même taille, qui sont ensuite utilisés pour l'entraînement du modèle. Cette approche est connue sous le nom de validation croisée.

La validation croisée est particulièrement utile lorsque les données sont hétérogènes. En effet, une simple division des données peut entraîner des sous-ensembles qui ne sont pas représentatifs de l'ensemble du jeu de données. Par exemple, 80% d'un sous-ensemble peut être très différent des 80% d'un autre sous-ensemble, ce qui peut conduire à des scores de performance très divergents en fonction de l'échantillonnage. Cette variation peut également se répercuter sur les résultats en ensemble test et, à terme, sur les performances en production. En utilisant la validation croisée, on obtient une estimation plus fiable des performances du modèle, car elle permet de réduire la variance due à une division particulière des données.

La validation croisée est employée de deux manières :

→ Pour trouver les hyperparamètres les plus optimaux du modèle. La méthode *GridSearchCV* du package *scikit-learn* est utilisé avec le paramètre "cv", qui définit le nombre de folds pour la validation croisée. Cette approche permet de tester différentes combinaisons d'hyperparamètres en utilisant la validation croisée pour chaque combinaison, afin de sélectionner celle qui maximise les performances du modèle.

→ Pour l'évaluation des performances du modèle, la fonction *cross_val_score* du même package a été employée en utilisant la validation croisée. Cette fonction calcule les scores du modèle sur chaque fold du jeu de données et fournit une estimation plus stable des performances en prenant la moyenne des scores obtenus. Contrairement à *GridSearchCV*, qui explore différentes combinaisons d'hyperparamètres pour optimiser le modèle, *cross_val_score* évalue les performances avec les paramètres fixes du modèle. Ainsi, elle permet d'obtenir une mesure fiable de la performance du modèle sur différents sous-ensembles des données, sans optimiser les hyperparamètres.

Une fois l'entraînement et la prédiction terminés, un jeu de données supplémentaire, qui n'est pas soumis à la validation croisée, est normalement utilisé pour une évaluation finale. Ce jeu de données est le jeu de données d'évaluation, obtenus à partir de capteurs, encore en fonctionnement à ce jour. Il permet de vérifier la performance du modèle en conditions réelles.

Maintenant que le protocole à effectuer lors de la partie entraînement et prédiction a été présenté, je vais passer à la description des différents modèles d'apprentissage supervisé que j'ai utilisés au cours de ce stage.

## 4.3 Modèles

Pour commencer les premiers modèles que j'ai testé durant ce stage sont les modèles d'analyse de survie : Kaplan Meier ainsi que le modèle de Cox.

### 4.3.1 Modèle Analyse de Survie

- **Kaplan Meier** :
  Le modèle de Kaplan-Meier est une méthode statistique utilisée principalement en analyse de survie pour estimer la fonction de survie à partir de données de durée de vie. En résumé, il permet de mesurer la fraction d'individus encore vivants après un certain moment t. La courbe représentant cette fonction de survie est constituée de marches horizontales qui diminuent à chaque événement.

  L'avantage de ce modèle est sa capacité à gérer les données censurées, c'est-à-dire les données pour lesquelles l'événement d'intérêt (dans notre cas, le décès des capteurs) n'a pas été observé pour certains individus durant la période d'étude. Ce phénomène est appelé « censure par la droite ».

  La fonction de survie S(t) est la probabilité qu'un individu survive au-delà d'un certain temps t. En général, cette probabilité diminue avec le temps. L'estimateur de Kaplan-Meier fournit une estimation de cette fonction. Cette courbe de survie est en escalier, où chaque diminution correspond à un décès observé. Elle permet de comparer les distributions de survie entre différents groupes.

- **Cox** :
  Le deuxième modèle d'analyse de survie que j'ai utilisé durant mon stage est le modèle de Cox. Ce modèle est l'une des méthodes les plus couramment utilisées pour analyser des données de survie. Le modèle de Cox, également connu sous le nom de modèle des risques proportionnels, est un modèle de régression qui cherche à expliquer comment les variables explicatives sont liées à un événement, dans notre cas, l'arrêt de la batterie.

  Ce modèle peut être appliqué à toute situation où l'on étudie le délai avant que l'événement d'intérêt ne survienne. Pour chaque individu, on connaît la date de la dernière observation ainsi que son état par rapport à l'événement étudié. Cependant, pour certains individus, l'état à la date de fin de l'étude n'est pas connu ; ces individus sont donc considérés comme censurés. L'avantage du modèle de Cox est qu'il permet de prendre en compte ces données même si elles sont incomplètes. Deux variables sont cruciales dans l'analyse de survie :

  - **La variable de survie T** : qui représente le temps de survie jusqu'à la survenue de l'événement.
  - **La variable d'état S** : qui indique si l'événement s'est produit ou non pour chaque individu à un moment donné.

### 4.3.2 Régression Linéaire multiple

La régression linéaire permet de prédire la valeur d'une variable dépendante y en fonction d'une ou plusieurs variables indépendantes. Cette méthode établit donc une relation linéaire entre x (l'entrée) et y (la sortie).

Dans le cas d'une régression linéaire multiple, le modèle établit une relation entre plusieurs variables indépendantes, appelées variables explicatives x_i, et une variable de sortie continue y, que l'on cherche à expliquer.

Contrairement à la régression linéaire simple, qui n'utilise qu'une seule variable explicative, la régression linéaire multiple permet d'intégrer plusieurs variables indépendantes. Cela peut potentiellement améliorer la précision du modèle, à condition que les variables ajoutées soient pertinentes et apportent une réelle contribution à la prédiction.
L'objectif est de trouver une fonction qui prédit y tout en minimisant l'erreur de prédiction.

Pour cet algorithme, il n'est pas nécessaire de spécifier de paramètres en entrée, les coefficients $\beta_i$ sont ajustés par le modèle au cours de l'entraînement.

### 4.3.3 SVR

Le modèle SVR (Support Vector Regression) est un algorithme de régression basé sur les principes du Support Vector Machine (SVM), mais adapté aux problèmes de régression. L'objectif de cet algorithme est de trouver une fonction qui prédit y la variable cible, à partir des vecteurs d'entrée.

Le principe de SVR est de positionner cette fonction de manière à ce qu'elle sépare les données tout en maximisant la distance, appelée « marge », entre les points de données et cette fonction. Contrairement à la régression linéaire classique, où le but est de minimiser l'erreur de prédiction pour chaque point de données, SVR ignore les erreurs qui se trouvent à l'intérieur de cette marge $\epsilon$, car les prédictions dans cette zone sont considérées comme suffisamment précises. Seules les erreurs situées en dehors de cette marge sont pénalisées, ce qui aide à maximiser la marge entre les points de données et la fonction prédictive.

Cependant, dans de nombreux cas, il n'est pas possible de séparer linéairement les données dans leur espace d'origine. Pour surmonter cette difficulté, SVR utilise une technique appelée « noyau » pour transformer les données dans un espace de dimension supérieure où une séparation linéaire devient possible. Le choix du noyau est crucial, car il détermine la manière dont les données sont transformées.

L'algorithme SVR comporte donc plusieurs hyperparamètres clés qui doivent être ajustés pour obtenir des performances optimales. Les principaux hyperparamètres de SVR sont :

- **Kernel** : qui représente le noyau. Les choix courants de noyaux sont : linear (pour problèmes linéaires), poly (noyau polynomial), rbf (noyau gaussien) et sigmoid (noyau sigmoïde)
- **C** : qui représente le paramètre de régularisation
- **Epsilon** $\epsilon$ : qui définie la marge
- **Gamma** $\gamma$ : Applicable pour les noyaux rbf, poly et sigmoid, ce paramètre définit l'influence d'un seul exemple d'entraînement

### 4.3.4 Random Forest

Le modèle de Random Forest, ou "Forêt aléatoire", est un algorithme de classification ou de régression qui repose sur l'assemblage de plusieurs arbres de décision indépendants. Un arbre de décision est une méthode qui divise les données en fonction de certaines conditions ou règles. Chaque nœud de l'arbre représente une question basée sur une variable du jeu de données, et chaque branche correspond à un résultat de cette question. Les feuilles de l'arbre représentent les décisions finales ou les prédictions.

Bien qu'un seul arbre de décision puisse offrir des résultats précis, sa performance dépend fortement de l'échantillon de données d'origine. Par exemple, l'ajout de nouvelles données à l'échantillon peut modifier le modèle et les résultats de manière significative. Le modèle Random Forest améliore la précision et la robustesse des prédictions en combinant les résultats de plusieurs arbres de décision, chacun ayant une vision partielle et unique du problème. Le modèle repose sur le principe que la combinaison de multiples avis (arbres) est généralement plus fiable qu'un seul avis.

Le terme "random" (aléatoire) dans Random Forest provient du double tirage aléatoire appliqué lors de la construction de chaque arbre, à la fois sur les observations et sur les variables. Les deux tirages aléatoires sont les suivants :

- Tree bagging : Cette technique est basée sur un tirage avec remplacement. Pour chaque arbre, une nouvelle base de données est créée en tirant aléatoirement des observations (lignes) de la base de données d'origine. Certaines lignes peuvent être sélectionnées plusieurs fois, tandis que d'autres peuvent ne pas être sélectionnées du tout. Ce principe de diversification des arbres permet de réduire la variance du modèle.
- Feature Sampling : À chaque nœud de l'arbre, un sous-ensemble aléatoire de variables (colonnes) est sélectionné. Seules les variables de ce sous-ensemble sont considérées pour déterminer la meilleure division du nœud. Cela réduit la corrélation entre les arbres, rendant le modèle plus diversifié et résilient.

Au final, tous ces arbres de décision indépendants sont assemblés. Pour un problème de régression, la prédiction du Random Forest pour des données inconnues est la moyenne des prédictions de tous les arbres. Pour la classification, la classe finale est celle qui a été prédite le plus souvent.

L'algorithme Random Forest nécessite plusieurs hyperparamètres qui doivent être définis avant l'entraînement. Voici les principaux hyperparamètres :

- **n_estimators** : qui représente le nombre d'arbres à utiliser dans la fôret. Plus le nombre d'arbres est élevé, plus la performance peut être stable, mais cela augmente aussi le temps de calcul
- **max_depth** : qui correspond à la profondeur maximale de chaque arbre, ce paramètre permet de contrôler le surapprentissage (overfitting) en limitant la profondeur des arbres
- **min_samples_split** : qui représente le nombre minimum d'échantillons requis pour diviser un nœud

- **min_samples_leaf** : le nombre minimum d'échantillons qu'une feuille peut contenir

  Dans les deux cas un nombre élevés de min_samples_split ou min_samples_leaf peut empêcher la complexité de l'arbre.

- **max_features** : qui prend comme valeur : 'auto', sqrt', 'log2', permet de définir le nombre de caractéristiques à considérer lors de la recherche de la meilleure scission

Il existe d'autres hyperparamètres, mais ils n'ont pas été pris en compte ici car ils sont jugés moins importants. Une fois ces hyperparamètres définis, le modèle Random Forest peut être utilisé pour résoudre des problèmes de régression ou de classification.

### 4.3.5 Réseaux de Neurones

Un réseau de neurones est un modèle d'apprentissage inspiré du fonctionnement du cerveau humain. Il est composé de multiples couches de neurones artificiels, qui sont des unités de calcul simples connectées entre elles. Un réseau de neurones typique est constitué de trois types de couches :

– **Couche d'entrée** (input layer) : Elle reçoit les données brutes, telles que les pixels d'une image ou les valeurs numériques d'un tableau. Chaque neurone de cette couche représente une caractéristique de l'entrée.

– **Couches cachées** (hidden layers) : Ce sont les couches intermédiaires entre l'entrée et la sortie. Un réseau peut avoir une ou plusieurs couches cachées. Chaque neurone d'une couche cachée reçoit des signaux des neurones de la couche précédente, effectue une somme pondérée, applique une fonction d'activation (une transformation mathématique), puis le résultat est transmis aux neurones de la couche suivante.

– **Couche de sortie** (output layer) : Elle produit le résultat final du réseau, qui peut être une classe dans un problème de classification, une valeur numérique pour un problème de régression, ou encore une probabilité.

Les neurones des couches cachées et de sortie sont connectés aux neurones des couches précédentes par des poids, qui sont des paramètres ajustés pendant l'entraînement. Chaque connexion a un poids associé qui détermine l'importance d'un neurone pour un autre.

Lors de l'entraînement d'un réseau de neurones, l'objectif est de déterminer les poids optimaux pour minimiser l'erreur entre les prédictions du réseau et les valeurs réelles. Ce processus utilise une méthode appelée propagation arrière : *backpropagation.*

- Propagation Avant *Forward Propagation* :
  Les données d'entrée sont passées à travers les différentes couches du réseau pour générer une prédiction. Cette prédiction est ensuite comparée à la valeur réelle pour calculer l'erreur.

- Propagation Arrière *backpropagation* :
  L'erreur calculée est rétropropagée (l'erreur est propagée en sens inverse) à travers le réseau, depuis la couche de sortie vers les couches précédentes. Pour chaque poids, le gradient est calculé en utilisant la dérivée de la fonction de perte par rapport à ce poids et en tenant compte de l'activation du neurone dans la couche précédente. Les poids sont ajustés en suivant ces gradients pour réduire l'erreur.

L'algorithme de descente de gradient est une méthode utilisée pour minimiser l'erreur en ajustant les poids du réseau. Il ajuste les poids dans la direction opposée au gradient de l'erreur pour réduire cette perte.

Ce processus est répété sur plusieurs itérations, ce qui permet au réseau d'améliorer progressivement ses prédictions en ajustant les poids de manière optimale.
Le réseau de neurones comporte plusieurs hyperparamètres qui doivent être ajustés pour obtenir de bonnes performances. Les principaux hyperparamètres sont :

- Le **nombre de couches cachées et de neurones par couche** : Plus le réseau est profond (avec de nombreuses couches) et large (avec de nombreux neurones par couche), plus il est capable de capturer des relations complexes, mais cela augmente également le risque de surapprentissage (overfitting).
- Le **taux d'apprentissage** (learning rate) : Ce paramètre contrôle la vitesse à laquelle les poids sont mis à jour. Un taux d'apprentissage trop élevé peut entraîner une convergence instable, tandis qu'un taux trop faible peut rendre l'entraînement trop long.
- Les **fonctions d'activation** : Ces fonctions introduisent des non-linéarités dans le réseau, permettant au modèle de capturer des relations non linéaires dans les données. Les fonctions d'activation courantes incluent : Relu, Sigmoid, Tanh
- **L'optimiseur** : Les algorithmes d'optimisation ajustent les poids et les biais pour minimiser la fonction de perte. Quelques optimiseurs populaires sont :
  SGD (Stochastic Gradient Descent), Adam, Adadelta
- Le **nombre d'itérations** (epochs) : Il s'agit du nombre de fois que le réseau passe sur l'ensemble des données d'entraînement. Un plus grand nombre d'époques permet au réseau d'apprendre davantage, mais au risque de surapprentissage.
- La **taille du lot** (batch size) : Nombre d'échantillons traités avant la mise à jour des poids.

Pour une approche plus théorique de chacun de ces modèles, une section plus détaillée est disponible en **Annexe**.

## 4.4 Évaluation

Comme expliqué précédemment, l'apprentissage supervisé se divise en deux étapes essentielles : l'entraînement/prédiction et l'évaluation. Après avoir abordé la phase d'entraînement/prédiction, il est maintenant important de se pencher sur la phase d'évaluation.

L'évaluation des modèles consiste à mesurer la capacité d'un modèle à prédire correctement sur des données qu'il n'a jamais vues auparavant. Elle permet de comparer plusieurs modèles pour sélectionner celui qui offre les meilleures performances selon des critères spécifiques. De plus, l'évaluation aide à identifier et corriger des problèmes tels que le sur-apprentissage ou sous-apprentissage, nous reviendrons sur ce point dans la partie **4.4.2**.

### 4.4.1 Métriques d'évaluation

Pour évaluer les performances d'un modèle, on utilise des métriques d'évaluation, qui sont des mesures de l'erreur entre la valeur de la prédiction et la valeur de la variable Y. Le choix de la métrique appropriée dépend notamment du type de problème (classification ou régression) et des objectifs spécifiques du projet. Dans le cas de la régression, les métriques d'évaluation les plus couramment utilisées sont : l'erreur absolue moyenne (MAE) et le coefficient de détermination ($R^2$).

- **MAE** : mesure l'erreur moyenne absolue entre les valeurs prédites et les valeurs réelles.
  Mathématiquement elle se note : MAE : $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$.
  Elle est facile à interpréter car elle est dans la même unité que la variable de réponse.

- **$R^2$** : mesure la proportion de la variance totale des variables dépendantes expliquée par le modèle.
  Mathématiquement elle se note : $R^2$=1 - $\frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y}_i)^2}$

  où n est le nombre total d'observations, $y\_i$ la valeur réelle de la i-ème observation et $\hat{y}_i$ la valeur prédite pour la i-ème observation.

Ainsi, la métrique MAE mesure l'erreur entre le y prédit et le y réel. Plus l'erreur est faible, c'est-à-dire proches de zéro, plus nous serons satisfaits des prédictions obtenues par notre modèle. L'erreur obtenue avec la MAE est facilement interprétable car elle est exprimée dans la même unité que la variable cible, soit dans notre cas en nombre de jours restants avant le décès.

En ce qui concerne l'interprétation de la métrique $R^2$, une valeur proche de 1 indique un bon ajustement du modèle aux données, tandis qu'une valeur proche de 0 suggère un ajustement médiocre.

### 4.4.2 Sur-apprentissage

Avoir des métriques avec des scores élevés est généralement attendu pour considérer qu'un modèle est performant. Toutefois, si ces scores sont trop élevés, cela peut soulever des questions concernant un possible sur-apprentissage ou sur-ajustement du modèle. Ce phénomène survient lorsque le modèle semble fournir des prédictions précises, mais uniquement pour les données d'entraînement, et non pour de nouvelles données. En effet, un modèle sur-appris peut être trop adapté aux données d'entraînement, ayant mémorisé les détails spécifiques et les bruits présents dans ces données, ce qui le rend inefficace pour prédire des résultats sur des ensembles de données inconnus. Ainsi, un modèle en situation de sur-apprentissage pourrait produire des prévisions inexactes sur des données nouvelles, compromettant ainsi sa performance globale.

Des scores élevés des métriques peuvent indiquer que le modèle a mémorisé les données d'entraînement plutôt que d'avoir appris les relations générales applicables à de nouvelles données. Pour détecter le sur-apprentissage, il existe plusieurs méthodes. Dans le cadre de mon sujet, deux méthodes se sont avérées pertinentes : la matrice de confusion et la courbe d'apprentissage (learning curve).

### 4.4.3 Matrice de Confusion

La matrice de confusion également appelée tableau de contingence, est un outil essentiel pour évaluer la performance d'un modèle de classification. Elle permet de comparer les résultats prédits par le modèle avec les valeurs réelles afin de mesurer sa précision. Dans ce tableau, chaque colonne représente une classe prédite par l'algorithme, tandis que chaque ligne représente une classe réelle. La matrice de confusion offre une vue d'ensemble des prédictions correctes et incorrectes, et fournit des indices précieux sur les types d'erreurs commises. Elle se décompose en quatre catégories principales :

Vrais Positifs (VP) : la prédiction et la valeur réelle sont positives.
Vrais Négatifs (VN) : la prédiction et la valeur réelle sont négatives.
Faux Positifs (FP) : la prédiction est positive alors que la valeur réelle est négative.
Faux Négatifs (FN) : la prédiction est négative alors que la valeur réelle est positive.

La matrice de confusion est un outil précieux pour détecter les signes de sur-apprentissage. Pour ce faire, il est utile de comparer les performances du modèle sur les données d'entraînement et de test.

Sur les données d'entraînement, un modèle ayant sur-appris affichera généralement une matrice de confusion avec un grand nombre de vrais positifs et de vrais négatifs, et peu de faux positifs et de faux négatifs. En revanche, sur les données de test, un modèle sur-appris présentera souvent une augmentation significative des faux positifs et des faux négatifs. Cela indique que le modèle n'a pas bien généralisé aux nouvelles données. Ainsi, des écarts marqués entre les ratios de prédictions correctes et incorrectes sur les données d'entraînement par rapport aux données de test peuvent signaler que le modèle a mémorisé les données d'entraînement au lieu d'apprendre des caractéristiques généralisables.

### 4.4.4 Courbes d'apprentissages

Les courbes d'apprentissage ou *learning curves* sont des outils essentiels pour évaluer la performance des modèles. Elles permettent d'observer comment un modèle apprend et généralise en fonction de la quantité de données d'entraînement ou le nombre d'itérations, selon qu'il s'agit de modèles traditionnels ou de réseaux de neurones.

Courbes d'apprentissage pour les modèles traditionnels :

Les courbes d'apprentissage pour les modèles traditionnels, tels que les forêts aléatoires et les régressions linéaires, sont généralement tracées en fonction de la taille de l'ensemble d'entraînement. L'axe des abscisses représente le nombre d'échantillons de données d'entraînement utilisés pour entraîner le modèle, tandis que l'axe des ordonnées correspond au score de performance de la métrique $R^2$. Pour évaluer la généralisation d'un modèle, il est essentiel de comparer la courbe d'entraînement et la courbe de validation. Le sur-apprentissage se produit lorsque le score d'entraînement est élevé, mais que le score de validation est significativement plus bas, indiquant que le modèle a trop bien appris les particularités des données d'entraînement. Le sous-apprentissage, quant à lui, se manifeste lorsque les scores d'entraînement et de validation sont tous deux bas et proches l'un de l'autre, suggérant que le modèle n'a pas encore suffisamment appris.

Courbes d'apprentissage pour les modèles de réseaux de neurones :

Les courbes d'apprentissage pour les réseaux de neurones diffèrent de celles des modèles traditionnels, car elles sont tracées en fonction du nombre d'épochs plutôt que de la taille de l'ensemble d'entraînement. L'axe des abscisses représente ici le nombre d'épochs, c'est-à-dire le nombre de fois que l'ensemble complet de données a été utilisé pour entraîner le modèle. Les réseaux de neurones nécessitent souvent beaucoup plus d'itérations pour converger et peuvent continuer à améliorer leur performance avec l'ajout d'épochs supplémentaires. L'axe des ordonnées, quant à lui, représente l'erreur. Pour identifier le sur-apprentissage, on compare la courbe d'erreur d'entraînement (train loss) et la courbe d'erreur de validation (validation loss). Une train loss qui diminue indique que le modèle apprend bien sur les données d'entraînement. Cependant, si la validation loss commence à augmenter alors que la train loss continue de diminuer, cela peut indiquer un surapprentissage. Un modèle bien équilibré est généralement caractérisé par des courbes d'erreur d'entraînement et de validation qui convergent vers des valeurs similaires à mesure que le nombre d'épochs augmente. Idéalement, la validation loss doit être légèrement au-dessus de la train loss, et une différence significative entre les deux peut signaler des problèmes de surapprentissage ou d'autres difficultés dans le modèle ou les données.

## 4.5 Évaluation en conditions réelles

Une fois que le modèle a été évalué à l'aide des métriques d'évaluation et que l'on a vérifié l'absence de sur-apprentissage, et si les résultats des méthodes de scoring sont satisfaisants, nous considérons alors que la phase d'entraînement et d'évaluation sur notre découpage train/test est terminée.

Cependant, dans le contexte de notre projet, où l'objectif est de prédire la durée de vie restante des batteries des capteurs, il est crucial de tester notre modèle sur un jeu de données indépendant. Cette étape est essentielle pour obtenir un indicateur de performance représentatif de l'efficacité du modèle en conditions de production. En effet, l'évaluation actuelle est insuffisante en raison de l'échantillon réduit utilisé, qui se concentrait uniquement sur les capteurs non fonctionnels en raison de batteries usées. Un test sur un ensemble de données indépendant permettrait de mieux évaluer la capacité du modèle à généraliser et à fournir des prévisions fiables lorsqu'il sera déployé en production. On se propose donc de s'appuyer sur les données des capteurs actuellement en fonctionnement. En effet, les données utilisées pour l'évaluation initiale proviennent de capteurs qui ont déjà atteint leur état de défaillance. Tester sur des capteurs encore actifs nous permettra de valider la capacité du modèle à faire des prédictions sur des capteurs dont l'état est encore en cours de changement, offrant ainsi une meilleure estimation de la performance du modèle dans des conditions réelles d'utilisation.

Cette étape est cruciale pour garantir que le modèle fonctionne non seulement sur les données historiques, mais qu'il peut également prédire la durée de vie des capteurs encore en activité. Par exemple, il est possible que le modèle fonctionne bien pour identifier les capteurs dont les batteries sont déjà usées, mais pour évaluer sa véritable efficacité, il doit aussi pouvoir anticiper avec précision la durée de vie restante des capteurs qui n'ont pas encore montré de signes de défaillance. Tester uniquement sur des capteurs défectueux ne donne pas une image complète de sa performance. Il est donc essentiel d'évaluer le modèle sur un jeu de données indépendant, incluant des capteurs à différents stades de leur cycle de vie, afin de vérifier qu'il peut généraliser et fournir des prédictions fiables en production.

# 5 Résultats et discussion

Maintenant que les différentes phases de l'apprentissage supervisé ont été présentées et que tous les modèles utilisés durant mon stage ont été décrits, je vais aborder les résultats obtenus. Cette section détaillera les performances des modèles, les observations clés, ainsi que les analyses effectuées pour évaluer leur efficacité. Les courbes d'apprentissage, les métriques de performance et les comparaisons entre les modèles seront mises en avant pour illustrer la pertinence des choix méthodologiques effectués.

## 5.1 Prétraitement

### 5.1.1 Présentation des variables

Pour commencer cette section, je vais présenter en détail chacune des variables qui constitue mon jeu de données. D'une part, il y a les variables quantitatives. Ces variables ont été calculées sous forme de moyennes ou sous forme de compteurs. Puis ensuite, les variables catégorielles.

Parmis les quantitatives de type compteurs nous avons les variables nommées : Index_total, Info_total, Info_batterie_total et Info_message_total.
En effet concernant l'index, il nous a semblé intéressant de savoir sur combien de compteur un capteur a été associé : 1, 2 ou 3. De plus chaque champ émet son information plusieurs fois par mois. La variable Info_total correspond donc au nombre de fois que toutes les différentes informations ont été émises, Info_batterie_total et Info_message_total, le nombre de fois que la variable batterie a émis une information et de la même manière pour la variable message. Mes travaux préliminaires m'ont pas permis de démontrer un réel intérêt à conserver le nombre d'information émise par variable indépendamment, sauf pour les variables batterie et message.

En termes de quantitatives moyennées nous avons cette fois-ci les variables nommées : Moyenne_index_1, Moyenne_index_2, Moyenne_index_3 associé respectivement à chacun des trois compteurs, Moyenne_rawddata, Moyenne_rssi, Moyenne_snr et Moyenne_sf.
La valeur de la variable index correspondant aux nombres d'impulsions entre la dernière fois que le compteur a été allumé jusqu'à un moment t, il nous a semblé intéressant d'avoir une moyenne de cette valeur. Concernant les données rawdata, il est aussi nécessaire d'avoir une variable qui informe du nombre d'octets moyen utilisé. Comme expliqué, un RSSI faible et un SF élevé peuvent être la cause d'une perte de batterie plus rapide. Ainsi, il est important d'avoir des variables correspondant aux valeurs moyennes mensuelles de ces deux variables ainsi que du SNR.

Concernant les variables catégorielles, il y en a seulement une présente dans ce jeu de données. Elle correspond aux différents modèles des capteurs. Le modèle du capteur étant une variable catégorielle, elle a été encodée numériquement pour des besoins algorithmiques par des valeurs appartenant à l'ensemble [0, 1, 2, 3, 4]. Certains capteurs n'ont cependant pas de modèle approprié, pour cela c'est alors le chiffre -1 qui leur a été attribué. J'ai décidé de ne pas supprimer ces capteurs car lors d'analyses préliminaires j'ai pu constater qu'il n'y avait pas de très grandes différences statistiques sur les autres variables en fonction du modèle. Le fait de numériser tous les capteurs qui n'ont pas de modèle attribué par -1, ne devrait pas entraîner de biais dans la suite de l'étude.

Mon jeu de données est constitué de 12 variables explicatives. Reste la variable à prédire, qui est la durée de vie restante d'un capteur (en jours). Pour définir cette variable, je me suis basée sur l'activité du capteur sur les trois réseaux combinés. Cela m'a permis de déduire l'âge du capteur le jour où il a cessé de fonctionner, ainsi que son âge à intervalles mensuels. Cette variable, notée Y, représente donc la variable cible que les modèles d'apprentissage supervisé devront prédire.

En résumé, le jeu de données sur lequel j'ai travaillé est constitué de 13 variables (12 variables explicatives et la variable Y à prédire) et 3 149 instances pour 65 capteurs. À titre d'exemple, la table Table 2 représente une partie du jeu de données.

| | **Instance 1** | **Instance 2** | **Instance 3** |
|---|---|---|---|
| **Info_batterie_total** | 597 | 624 | 650 |
| **Info_message_total** | 14 | 14 | 14 |
| **Index_total** | 1 | 1 | 1 |
| **Info_total** | 185 333 | 193 694 | 201 972 |
| **Moyenne_index1** | 342 763.043143 | 385 557.650298 | 417 957.202703 |
| **Moyenne_index2** | 0 | 0 | 0 |
| **Moyenne_index3** | 0 | 0 | 0 |
| **Moyenne_rawdata** | 22.1540 | 22.1545 | 22.1502 |
| **Moyenne_RSSI** | -112.88 | -112.06 | -112.05 |
| **Moyenne_SF** | 9.83 | 8.70 | 9.86 |
| **Moyenne_SNR** | - 5.06 | -2.08 | -3.56 |
| **Modèle_capteur** | -1 | -1 | -1 |
| **Durée_jours** | 114 | 83 | 53 |

Table 2: Trois instances appartenant au même capteur

### 5.1.2 Données manquantes

Certains capteurs utilisés pour l'entraînement sont soumis à des problème de données manquantes. Afin d'y remédier, les données manquantes sont remplacées par une estimation basée sur la moyenne des valeurs non manquantes.

Concernant la variable SF, le traitement doit s'effectuer différemment. En effet, la valeur du *Spread Factor* se situe toujours entre 7 et 12. Nous avons donc convenu de remplacer les valeurs manquantes par une valeur moyenne de 10. Une étude préliminaire sur le SF a été effectuée avant mon arrivée dans l'entreprise, notamment pour la construction du modèle physique présenté en amont. Il en est ressorti qu'un SF trop élevé entraîne des résultats biaisés, c'est pourquoi la valeur de 10 a été considérée comme la plus juste.

### 5.1.3 Normalisation

Mon jeu de données comprend des variables avec des types de données différents, où certaines valeurs numériques sont plus grandes que d'autres. Cela peut introduire un biais dans les prédictions, car les grandes valeurs numériques pourraient dominer la modélisation.
Dans ce contexte, il est important d'utiliser la normalisation. La normalisation des données est une technique qui standardise les variables en supprimant leur moyenne et en les redimensionnant

à une variance de 1. En d'autres termes, elle transforme les données pour que chaque variables aient une distribution avec une moyenne de 0 et un écart-type de 1. Cette méthode est particulièrement utile pour certains modèles qui sont sensibles à l'échelle des variables. Dans le cas de mon étude j'ai normalisé le jeu de données pour tous modèles traditionnels, soit le modèle de Régression Multiple, SVR, Random Forest et le modèle de Réseau de neurones. En normalisant, on améliore la convergence des algorithmes, la performance du modèle, et on obtient des résultats plus fiables et comparables entre les variables.

### 5.1.4 Découpage entraînement/test

Comme expliqué précédemment, mon jeu de données contient des informations sur différents capteurs dont les batteries sont épuisées, enregistrées à différents moments de leur cycle de vie. L'objectif est d'apprendre à partir de l'historique de vie de ces capteurs pour pouvoir prédire, sur des capteurs encore en fonctionnement, le temps restant avant leur perte d'usage. La variable à prédire dans ce cas est le temps de vie restant de chaque capteur à partir d'un moment t, ce qui correspond à la variable y.

Pour procéder au découpage train/test, étant donné que nous avons plusieurs instances par capteur et un nombre limité de capteurs distincts, il est crucial d'éviter que le modèle soit testé sur des capteurs ayant servis pour la phase d'entraînement du modèle. Si le découpage n'est pas effectué correctement, il y a un risque que le modèle apprenne des caractéristiques spécifiques aux capteurs, ce qui pourrait entraîner un sur-apprentissage de sa performance et une mauvaise généralisation sur de nouveaux capteurs inconnus.

La solution est de découper les données en tenant de l'appartenance à chaque capteur, plutôt que d'échantillonner naïvement sur l'ensemble des instances. Ainsi, la méthode *GroupShuffleSplit* du package *sklearn* permet de diviser un ensemble de données en ensembles d'entraînement et de test tout en respectant les groupes, ce qui signifie que les données des capteurs sont regroupées par mois depuis le début de leur vie. Contrairement à la méthode *TrainTestSplit*, du même package, qui effectue une simple division aléatoire, *GroupShuffleSplit* garantit que les groupes (dans ce cas, les capteurs) ne sont pas partagés entre l'ensemble d'entraînement et l'ensemble de test. 80% des capteurs sont utilisés d'un côté pour l'entraînement et 20% de l'autre pour le test, assurant que les tests sont effectués sur des capteurs qui n'ont pas été vus durant l'entraînement.

Ainsi mon jeu de données d'entraînement sera constitué de 2603 instances pour 52 capteurs et mon jeu de test de 546 instances pour 13 capteurs.

Comme indiqué dans la partie Apprentissage supervisé, toutes les analyses sont effectuées sous validation croisée (cross-validation). Ainsi, j'ai opté pour la méthode k-fold cross-validation avec k=4. Cela signifie que le modèle est entraîné quatre fois, chaque fois sur k-1 des sous-ensembles, et testé sur le sous-ensemble restant. Pour chaque itération, un sous-ensemble différent est utilisé comme ensemble de test, tandis que les k-1 autres sont utilisés pour l'entraînement. Les performances du modèle sont ensuite calculées pour chaque itération, et la performance finale est obtenue en prenant la moyenne des performances sur les quatre itérations. Ainsi, toutes les analyses ont été effectuée en validation croisée de 4 itérations. J'ai choisi cv=4, compte tenu du faible volume de données dont on dispose. De plus, au-delà de 4 ou 5 folds, les résultats ne seront certainement pas différents au vu de la faible variabilité du jeu de donnée.

## 5.2 Ajustement des paramètres des modèles traditionnels et du réseau de neurones

Après avoir détaillé le découpage du jeu de données pour les phases d'entraînement et de prédiction, il est essentiel de se pencher sur l'entraînement et l'ajustement des différents modèles que nous avons utilisés. Avant de comparer les performances de ces modèles, nous devons examiner en profondeur la manière dont les hyperparamètres ont été déterminés et optimisés.

Je vais seulement expliciter les hyperparamètres des modèles traditionnels, car les modèles de survie, comme ceux utilisés dans cette étude, sont généralement plus simples et ne nécessitent pas d'ajustement complexe des hyperparamètres.

### 5.2.1 Modèles traditionnels

Trois modèles traditionnels ont été utilisés lors de ce stage : la régression multiple, le Support Vector Regression (SVR) et la Random Forest. Ces modèles proviennent de la bibliothèque *Scikit learn*, avec les fonctions *LinearRegression*, *SVR*, *RandomForestRegressor*.

**Régression Multiple** :
Le premier modèle utilisé est le modèle de Régression multiple. Comme précisé dans la présentation du modèle dans la partie méthodologie, ce modèle est directement implémenté sans nécessiter d'optimisation des hyperparamètres via *GridSearchCV*, car il n'y a pas de paramètres à ajuster.

**SVR (Support Vector Regression)** :
Le modèle SVR utilise les distances dans l'espace des caractéristiques pour optimiser les prédictions. Si les caractéristiques ne sont pas sur des échelles comparables, les caractéristiques à grande échelle peuvent dominer celles à petite échelle, ce qui peut conduire à des résultats biaisés. Ainsi pour une meilleure performance et une meilleure stabilité des résultats, la normalisation des données est essentielle pour éviter les biais liés aux différences d'échelle des variables. Les principaux paramètres à ajuster sont :

- Le Noyau, qui prend ses valeurs dans [linear, poly, rbf, sigmoid]
- Le paramètre de régularisation, qui prend ses valeurs dans [0.1, 1.0, 10, 100]
- La marge $\epsilon$, qui prend ses valeurs dans [0.001, 0.005, 0.01, 0.05, 0.1]
- Le paramètre d'influence $\gamma$, qui prend ses valeurs dans ['scale', 'auto']

Ainsi, le modèle ayant donné le meilleur résultat est celui avec seulement les hyperparamètres suivants :

- Noyau : rbf
- Régularisation : 100
- $\epsilon$ : 0.005
- $\gamma$ : scale

**Random Forest** :

Le dernier modèle traditionnel utilisé est le modèle de Random Forest. Ce modèle utilise plusieurs paramètres importants pour ajuster ses performances :

- Le nombre d'arbres, qui prend ses valeurs entre [10,100] avec un pas de 10
- Le paramètre de profondeur, qui prend ses valeurs dans [5, 10, 20, 30]
- Le paramètre de division de noeud, qui prend ses valeurs dans [2, 5, 10]
- Le paramètre d'échantillon d'une feuille, qui prend ses valeurs dans [1, 2, 4]
- Le paramètre de meilleur scission, qui prend ses valeurs dans ['auto', 'sqrt', 'log2']

Le modèle ayant donné le meilleur résultat est celui avec seulement les hyperparamètres suivants:

- Nombre d'arbres : 40
- Profondeur : 30
- Meilleure scission : sqrt

Les modèles ont d'abord été entraînés sur un jeu de données non normalisé, mais le SVR nécessite impérativement une normalisation pour de meilleures performances. Ainsi, afin de comparer les performances entre les trois modèles et d'éviter les biais liés aux échelles de variables, la régression multiple et la Random Forest ont également été entraînées sur des données normalisées.

Pour évaluer l'influence des variables, trois approches ont été utilisées : les coefficients de régression, la fonction *feature_importance* du Random Forest, et la fonction *permutation_importance* du SVR.

- Coefficients de Régression : les coefficients montrent l'impact direct de chaque variable sur la prédiction du modèle de régression, avec des coefficients plus élevés indiquant une plus grande influence.
- *feature_importance*(Random Forest) : cette fonction calcule l'importance d'une variable en fonction de combien elle contribue à la réduction de l'impureté dans les arbres de décision. L'importance est la moyenne des contributions de chaque variable à travers tous les arbres du modèle.
- *permutation_importance* (SVR) : cette méthode évalue l'importance des variables en observant la perte de performance du modèle lorsque les valeurs d'une caractéristique sont mélangées aléatoirement. Une grande diminution de performance indique une grande importance.

Les variables les moins influentes, comme Moyenne_index_2, Moyenne_index_3 et Index_total, ont été supprimées pour tester leur impact sur les performances du modèle. Cependant, cette suppression n'a pas amélioré les métriques d'évaluation. Par conséquent, le jeu de données complet avec ses 12 variables a été maintenu pour l'analyse.

### 5.2.2 Réseau de Neurones

Le dernier modèle utilisé durant ce stage est un réseau de neurones, pour lequel j'ai utilisé la librairie PyTorch. PyTorch est une librairie flexible et puissante pour construire, entraîner et déployer des réseaux de neurones. Elle offre des outils essentiels pour la création de modèles, comme la classe *torch.nn.Module*, qui permet de définir l'architecture du réseau. PyTorch facilite également la gestion des gradients et l'optimisation, tout en supportant l'exécution sur GPU. Les GPU (unités de traitement graphique) sont des processeurs spécialement conçus pour effectuer de nombreux calculs en même temps, ce qui rend l'entraînement des réseaux de neurones beaucoup plus rapide. Cela permet de développer des solutions d'apprentissage profond efficaces et personnalisables.

Tout comme pour le modèle SVR, la normalisation des données est une étape cruciale lors de l'utilisation des réseaux de neurones. En effet, normaliser les données avant d'entraîner un réseau de neurones permet de stabiliser et d'accélérer l'apprentissage, d'améliorer l'efficacité des fonctions d'activation, de traiter les caractéristiques de manière équilibrée et d'éviter les problèmes liés aux gradients. Cela se traduit par un modèle plus performant et un processus d'apprentissage plus efficace. Pour cette normalisation, j'ai utilisé la méthode *MinMaxScaler* de la librairie *Scikit learn*

Ainsi, le modèle de réseau de neurones possède une architecture spécifique, et plusieurs paramètres doivent être définis en amont pour sa construction. Ces paramètres incluent le nombre de neurones par couche, le nombre de couches cachées, le taux d'apprentissage, les fonctions d'activation, l'optimiseur, le nombre d'itérations et la taille du lot. Pour que le modèle soit le plus performant possible, il est crucial de choisir les valeurs optimales pour chacun de ces paramètres.

Contrairement aux modèles traditionnels, la méthode *GridsearchCV* ne peut pas être utilisée directement pour ajuster les paramètres des réseaux de neurones. Il est donc nécessaire de tester plusieurs valeurs pour chaque paramètre. Pour évaluer l'efficacité des différentes configurations, il faut analyser les *learnings curves*. Ces courbes permettent de comparer la perte d'entraînement (train loss) et la perte de test (test loss) de chaque modèle, afin de déterminer lequel est le mieux ajusté.

Voici les paramètres utilisés pour l'architecture du réseau de neurones ainsi que les valeurs choisies pour les ajuster :

- Nombre de couches cachées et de neurones par couche :
  Pour mon étude, j'ai testé différentes configurations en variant le nombre de neurones par couche de trois manières : en gardant le même nombre de neurones sur toutes les couches, en doublant le nombre de neurones par rapport à la configuration de base, en réduisant le nombre de neurones en faisant une différence de 4 par couche.
  La couche d'entrée a un nombre de neurones équivalent au nombre de variables explicatives, soit 12, et la couche de sortie contient 1 neurone pour la variable réponse.
  Ainsi, j'ai exploré les variations suivantes pour les couches cachées :
  Nombre de couche cachées entre : [2, 4, 6]
  Nombre de neurones par couches cachées : [12], [12, 24, 48, 96, 192, 384, 768], [12, 8, 4]

- Taux d'apprentissage : 0.001
- Fonction d'activation : [Sigmoide, Relu, Tanh]
- Optimiseur : [Adam, Adadelta]
- Taille du lot : [32, 64, 96]
- Nombre d'itérations : [100, 200, 500]

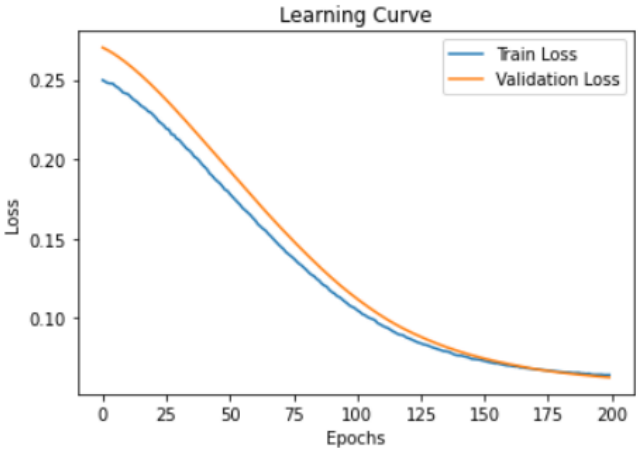



Figure 5: Learning-curves du meilleur modèle de réseau de neurones

La Figure 5 illustre les learning_curves du modèle ayant donné le meilleur résultat. On observe que les deux courbes (d'entraînement et de validation) commencent avec une erreur élevée, car le modèle n'a pas encore appris les caractéristiques des données. Elles convergent ensuite vers une valeur plus basse, indiquant que l'erreur de prédiction diminue et que le modèle apprend correctement sans sur-apprentissage excessif. De plus, la courbe de validation (en orange) est légèrement au-dessus de la courbe d'entraînement (en bleu). Une validation loss légèrement supérieure à la train loss est en effet souhaitable, car elle indique une bonne capacité de généralisation. Un écart trop important pourrait signaler des problèmes de surapprentissage ou un manque de généralisation du modèle. On note également que la courbe de validation semble passer sous la courbe d'entraînement à partir de 175 itérations, ce qui suggère qu'il pourrait y avoir un risque de sous-apprentissage si le nombre d'itérations est augmenté davantage.
Ainsi, le modèle ayant donné le meilleur résultat est celui avec les hyperparamètres suivants :

- 12 neuronnes pour toutes les couches cachées
- 64 comme taille de lot
- 4 couches cachées
- 200 itérations
- Relu comme fonction d'activation
- Adadelta comme optimiseur

Avant de passer à la partie résultat voici avec la Table 3 un récapitulatif des hyperparamètres les plus optimaux pour chaque modèle.

| Modèle | Hyperparamètres les plus optmiaux |
|---|---|
| **SVR** | Noyau : rbf \| Régularisation : 100 \| $\epsilon$ : 0.005 \| $\gamma$ : scale |
| **Random Forest** | Nombre d'arbres : 40 \| Profondeur : 30 \| Meilleure scission : sqrt |
| **Réseau de neurones** | Neurones: 12 pour toutes les couches cachées \| 4 couches cachées \| 200 itérations \| Relu comme fonction d'activation \| Adadelta comme optimiseur |

Table 3: Hyperparamètres les plus optimaux pour chaque modèle

## 5.3 Résultats

À présent, je vais présenter les résultats obtenus par les différents modèles. Cette analyse comparative permettra de déterminer lequel des modèles s'est avéré le plus performant sur ce jeu de données en termes de précision de prédiction et de généralisation. Les performances seront évaluées à l'aide de métriques appropriées, et les résultats seront discutés.

### 5.3.1 Résultats des modèles d'analyse de survie

Pour rappel, dans le cadre de ce stage, l'objectif principal est de prédire la durée de vie restante des batteries des capteurs. L'analyse de la fonction de survie est essentielle pour cette prédiction car elle permet de mieux comprendre le comportement des capteurs en termes de durée de vie. À titre d'information, la durée de vie moyenne des capteurs chez iQspot est de 5 ans.

Pour ce faire, nous avons utilisé le modèle de Kaplan-Meier, une méthode qui fournit une estimation de la probabilité qu'un capteur survive au-delà d'un certain temps. Pour construire cette fonction deux variables sont cruciales, la variable de survie T, qui représente le temps de survie jusqu'à la survenue de l'événement, soit la variable à prédire Y. Ainsi que la variable d'état S, qui indique si l'événement s'est produit ou non pour chaque individu à un moment donné. Dans le jeu de donnée cette variable correspond à la variable nommée "status", qui a été créée seulement pour les modèles d'analyse de survie. Cette variable est une variable catégorielle qui a été numérisée. Les capteurs pour lesquels nous avons une valeur de la batterie finale le jour du décès ont pour valeur status=2 et les autres ont pour valeur status=1.
Ce modèle a été implémenté avec la fonction *KaplanMeierFitter* de la bibliothèque *lifelines*.

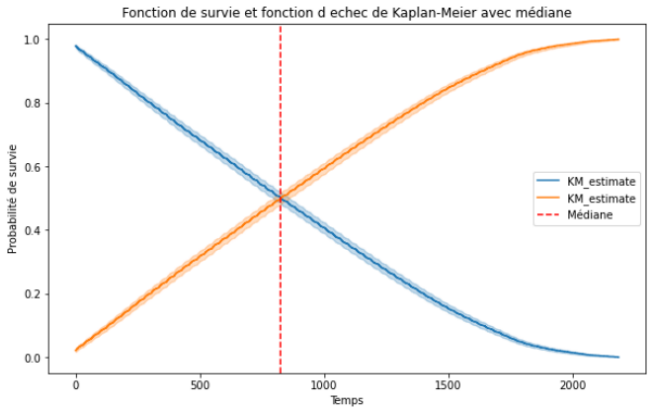



Figure 6: Probabilité de Survie des capteurs du jeu de données

Sur la Figure 6 nous pouvons observer la fonction de survie et la fonction d'échec du jeu de données constitué des 65 capteurs qui ont cessé de fonctionner. La fonction de survie est représentée en bleu, tandis que la fonction d'échec est montrée en orange. L'analyse de ces courbes révèle plusieurs points :

- Fin de la fonction de survie : La fonction de survie s'étend jusqu'à 2000 jours soit environ 5 ans et 6 mois, ce point temporel signifie qu'il reste une probabilité que certains capteurs fonctionnent encore au-delà de 2000 jours. Cette information fournit une vue d'ensemble générale de la durée de vie des capteurs dans notre échantillon.
- Médiane de la Durée de Vie : La médiane, indiquée par la ligne pointillée rouge, est de 827 jours, ce qui correspond à environ 2 ans et 3 mois. Cela signifie que 50% des capteurs ont cessé de fonctionner avant cette période, et 50% ont continué à fonctionner au-delà de cette durée.

Ces informations fournissent un aperçu utile du comportement de la durée de vie des batteries des capteurs et nous aident à comprendre la répartition des durées de vie au sein de notre jeu de données.

Avec ces premières informations sur la probabilité de survie et la répartition des durées de vie, nous allons passer à l'étape suivante : l'évaluation des modèles de prédiction. L'objectif est de déterminer quel modèle offre les meilleures performances pour prédire avec précision la durée de vie des capteurs.

Le premier modèle que j'ai entraîné est le modèle de Cox. Dans le contexte de ce stage, le modèle de Cox implémenté à l'aide de la fonction *CoxPHFitter* de la bibliothèque *lifelines*, a été utilisé pour analyser la durée de vie restante des batteries des capteurs, en prenant en compte divers facteurs pouvant influencer cette durée de vie. En effet, les résultats du modèle se présentent sous forme de ratios de risques (*hazard ratios*), qui indiquent l'effet de chaque variable sur le risque d'événement. Toutefois, lors de son entraînement, le modèle a rencontré des problèmes de convergence. Ces problèmes de convergence surviennent lorsque l'algorithme de maximisation de la vraisemblance, utilisé pour estimer les coefficients associés à chaque variable explicative, ne parvient pas à identifier des paramètres stables pour le modèle. En d'autres termes, le modèle a du mal à "apprendre" correctement des relations entre les variables explicatives et la survie.

Dans le cas de cette étude, ces difficultés peuvent être attribuées à plusieurs facteurs, notamment une quantité insuffisante de données, une forte colinéarité entre certaines variables explicatives, ou encore la présence de variables avec des effets faibles ou complexes qui ne sont pas bien assimilés par le modèle de Cox.

Malheureusement, les résultats obtenus avec ce modèle de survie n'ont pas été satisfaisants. J'ai donc décidé d'explorer d'autres approches en utilisant des modèles plus traditionnels. Ces modèles sont généralement moins complexes que les modèles de survie et peuvent offrir une perspective différente pour la prédiction de la durée de vie restante des batteries.

### 5.3.2 Résultats des modèles traditionnels et du réseau de neurones

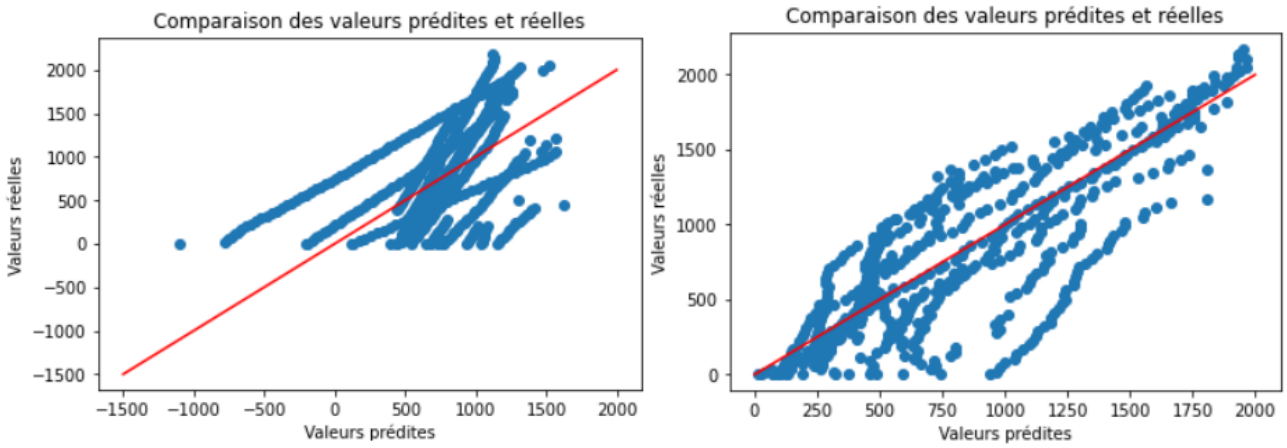
Dans cette partie nous allons donc comparer les scores obtenus pour chacun des modèles en utilisant les différentes méthodes d'évaluation.

| | **MAE (en jours)** | **$R^2$ (en %)** |
|---|---|---|
| **Régression multiple** | 437 | 0.17 |
| **SVR** | 475 | -0.007 |
| **Random Forest** | 230 | 0.68 |
| **Réseau de Neurones** | 486 | -0.04 |

Table 4: Résultats des différentes méthodes de scoring pour les différents modèles

La Table 4 représente seulement les scores les plus élevés pour chaque modèle. Nous pouvons constater qu'avec une MAE de 230 jours, la plus faible parmi les modèles, et un $R^2$ de 0,68, le modèle RandomForest est le plus performant sur ce jeu de données. Ce résultat a été obtenu avec le jeu de données non normalisé, toutefois il n'y avait pas de grande différence entre les performances du modèle sur le jeu normalisé ou non.

Comme précisé dans la partie méthodologie, ces méthodes d'évaluation permettent de mesurer la capacité d'un modèle à prédire correctement sur des données indépendantes des données de l'entraînement. La métrique MAE mesure l'erreur entre la valeur prédite y_pred et la valeur réelle y. Ainsi, avec une MAE de 230 jours, le modèle Random Forest prédit la durée de vie restante avec une erreur d'environ 230 jours, ce qui correspond à environ 7 mois. En d'autres termes, le modèle permet de prédire la durée de vie des capteurs avec une erreur de moins d'un an. Rappelons que la batterie d'un capteur a une autonomie de 5 ans en moyenne. Ainsi, cette information en parallèle de nos résultats montre que notre solution telle quelle promet d'être relativement efficace.




Régression Multiple

Random Forest

Figure 7: Comparaison des valeurs prédites et réelles des deux modèles

Sur la Figure 7 nous pouvons comparer visuellement les prédictions y_pred par rapport aux valeurs réelles pour le modèle de Régression Multiple et le modèle Random Forest. Chaque point bleu représente une prédiction à un instant t (t étant un indicateur mensuel dans la vie du capteur). Si les prédictions étaient exactes, les points bleus formeraient une droite bleue superposée à la droite rouge, représentant une droite affine y=x, qui représente la ligne idéale où les valeurs prédites et réelles seraient parfaitement alignées. Nous pouvons observer que les prédictions sont mieux ajustées à la droite y_réel pour le modèle Random Forest.

J'ai choisi de représenter uniquement le modèle de Régression Multiple en plus de celui du Random Forest, car ce dernier s'est révélé être le plus performant selon les métriques d'évaluation. Il est à noter que ce "classement" peut sembler surprenant, étant donné que les réseaux de neurones sont généralement connus pour offrir des performances élevées. On peut supposer que cette situation pourrait être due à la taille relativement réduite de mon jeu de données, ce qui pourrait limiter la capacité des réseaux de neurones à montrer leur véritable potentiel.

### 5.3.3 Sur-apprentissage

Le modèle qui a le mieux performé sur mon jeu de données est le modèle Random Forest. Dans certains cas, les scores de performance semblent suffisamment élevés pour attester de l'efficacité du modèle. Pourtant ces chiffres peuvent cacher un biais si les ensembles de test et d'entraînement sont très différents. Pour vérifier ce biais comme expliqué dans la partie **4.4.2**, nous examinerons le comportement des courbes d'apprentissage et de validation de ce modèle. Nous réaliserons également une matrice de confusion sur les données de test ainsi que sur les données d'entraînement pour comparer les performances du modèle.

Pour évaluer la performance du modèle, nous avons tracé les courbes d'apprentissage, qui montrent comment le score du modèle évolue en fonction de la taille de l'ensemble d'entraînement. Les courbes d'apprentissage ont été générées en utilisant la fonction *learning curve* de la librairie *scikit learn*, avec une validation croisée de 5 segments et des tailles d'échantillon allant de 10% à 100% de l'ensemble d'entraînement.

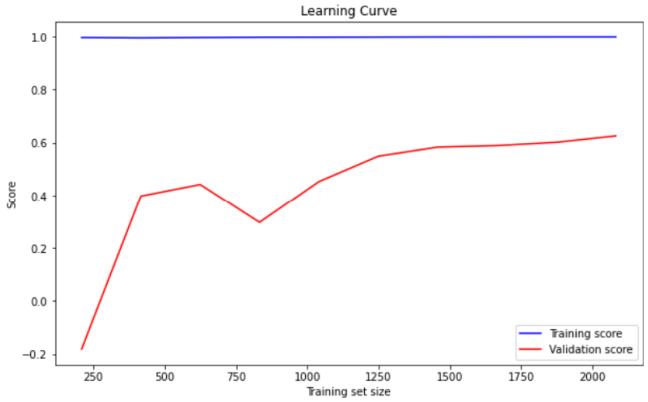
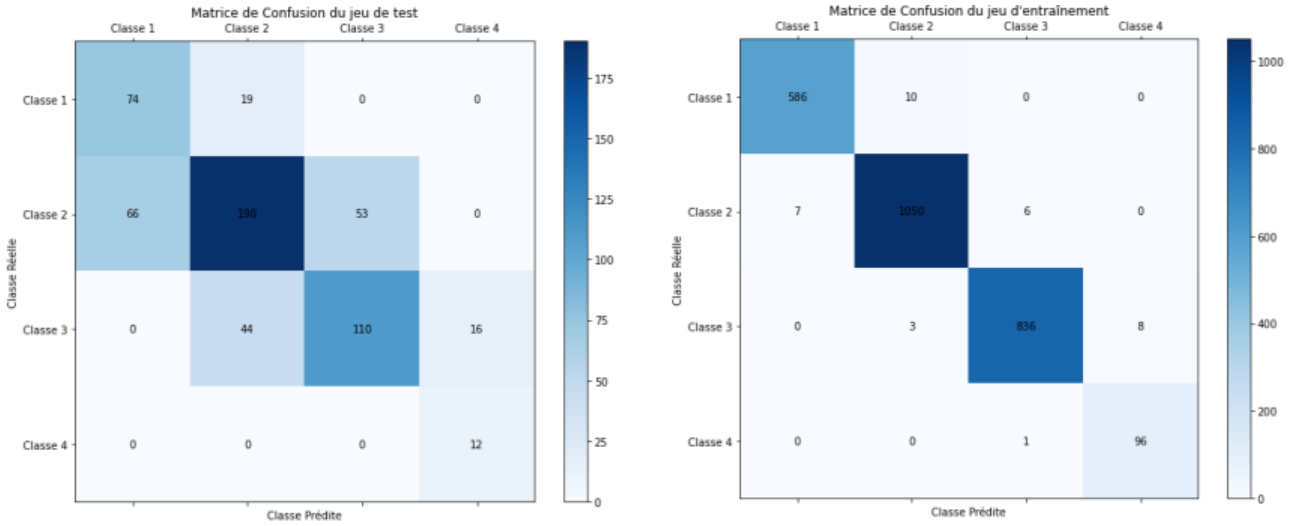



Figure 8: Comparaison des courbes d'apprentissage du jeu d'entraînement et du jeu de test

Sur la Figure 8 la courbe bleue représente le score d'entraînement. Une performance élevée et stable suggère que le modèle apprend bien à partir des données d'entraînement. La courbe rouge, quant à elle, montre la performance du modèle sur l'ensemble de validation. Une performance élevée et proche de celle de l'entraînement indique une bonne généralisation du modèle. Nous pouvons observer que le score de validation augmente avec l'échantillon. En effet, à mesure que l'échantillon croît, le modèle devient plus performant, atteignant un score final de 60% pour la métrique $R^2$. Cependant, le score d'entraînement est quand même supérieur au score de validation, ce qui suggère un petit sur-apprentissage. Cela signifie que le modèle pourrait s'adapter de manière trop spécifique aux données d'entraînement.

Afin de d'étayer cette conclusion par une autre méthode, nous allons à présent étudier les matrices de confusion. Cette approche est couramment utilisée dans les méthodes de classification, où elle permet de comparer les résultats prédits par le modèle avec les valeurs réelles pour chaque catégorie, ce qui est souvent plus révélateur que de comparer simplement la MAE dans chaque ensemble d'entraînement et de test. Comparer les matrices de confusion des jeux de données d'entraînement et de test nous aidera à déterminer si le modèle semble sur-apprendre.

Le principe de la matrice de confusion est de comparer les résultats prédits par le modèle avec les valeurs réelles par catégories de valeurs. Cette méthode fournit une vue plus détaillée de la performance du modèle en termes de classification. Pour ce faire, j'ai scindé mon jeu de données en quatre catégories distinctes basées sur la durée de vie des capteurs : moins d'un an, entre 1 an et 3 ans, entre 3 ans et 5 ans, et plus de 5 ans. La matrice de confusion fournit ainsi une vue d'ensemble des prédictions correctes et incorrectes, et offre des indications précieuses sur les types d'erreur commis. Si la matrice de confusion pour les données d'entraînement ne montre que des vrais positifs, cela pourrait indiquer un sur-apprentissage, où le modèle s'adapte trop spécifiquement aux données d'entraînement et performe moins bien sur des données non vues.




Matrice confusion du jeu de test

Matrice confusion du jeu d'entraînement

Figure 9: Matrices de confusions du jeu de test et d'entraînement

- **Classe 1** : moins d'un an d'historique de vie
- **Classe 2** : entre 1 ans et 3 ans d'historique de vie
- **Classe 3** : entre 3 ans et 5 ans d'historique de vie
- **Classe 4** : plus de 5 ans d'historique de vie

La matrice de confusion à gauche sur la Figure 9 représente les données de test. On observe que le modèle prédit globalement bien les durées de vie des capteurs. Il classe correctement 74 capteurs dans la classe 1, 190 capteurs dans la classe 2, 110 capteurs dans la classe 3, et 12 capteurs dans la classe 4. Le modèle fonctionne particulièrement bien pour les catégories 2 et 3, avec un grand nombre de bonnes prédictions (190 et 110, respectivement). Toutefois, les erreurs les plus courantes se produisent entre les catégories adjacentes. Par exemple, des capteurs avec une durée de vie dans la classe 2 sont souvent classés à tort dans la classe 1 ou classe 3. Bien que le modèle montre une performance raisonnable, il présente des faiblesses dans la distinction entre les catégories proches, ce qui est cohérent avec les scores d'évaluation modérés obtenus.

La matrice de confusion à droite, représentant les données d'entraînement, montre très peu de mauvaises prédictions. Cela peut suggérer que le modèle a légèrement sur-appris, s'adaptant trop spécifiquement aux données d'entraînement.

En conclusion, bien que le modèle soit efficace pour prédire les durées de vie des capteurs dans les classes intermédiaires, ses difficultés à distinguer les catégories proches doivent être prises en compte pour améliorer la précision des prévisions en conditions réelles. En effet les erreurs de classification pourraient entraîner des remplacements prématurés ou tardifs des capteurs et impacterait l'efficacité des opérations de maintenance.

### 5.3.4 Approfondissement

Malgré la présence d'un léger sur-apprentissage du modèle, le modèle Random Forest demeure le mieux ajusté pour mon jeu de données. Toutefois, une erreur de prédiction d'au moins 7 mois n'est pas suffisamment précise pour permettre à iQspot d'anticiper efficacement l'arrêt des batteries de leurs capteurs.

Comme nous avons pu le constater avec la Figure 6, la médiane de la fonction de survie du modèle Kaplan-Meier est de 2 ans et 3 mois, ce qui indique que les 65 capteurs n'ont pas la même durée de vie. De plus, les résultats de la matrice de confusion sur les données de test montrent que le modèle fonctionne mieux pour certaines durées de vie, notamment "entre 1 et 3 ans" et "entre 3 et 5 ans".

Cela soulève une question : bien que le modèle Random Forest semble offrir des performances moyennes sur l'ensemble des capteurs, pourrait-il avoir une meilleure performance si on se concentre uniquement sur un sous-ensemble de capteurs ayant plus de 3 ans de vie ? Ce sous-ensemble aurait potentiellement un historique de vie plus dense que les capteurs plus jeunes, avec seulement un an de vie. De plus, il y a très peu de capteurs dans le jeu de données ayant uniquement 1 ou 2 ans de vie, alors que les capteurs avec une durée de vie moyenne de 5 ans sont plus nombreux.

Ainsi, j'ai décidé de comparer les performances du modèle sur différents échantillons spécifiques de capteurs présentant des historiques de vie variés.

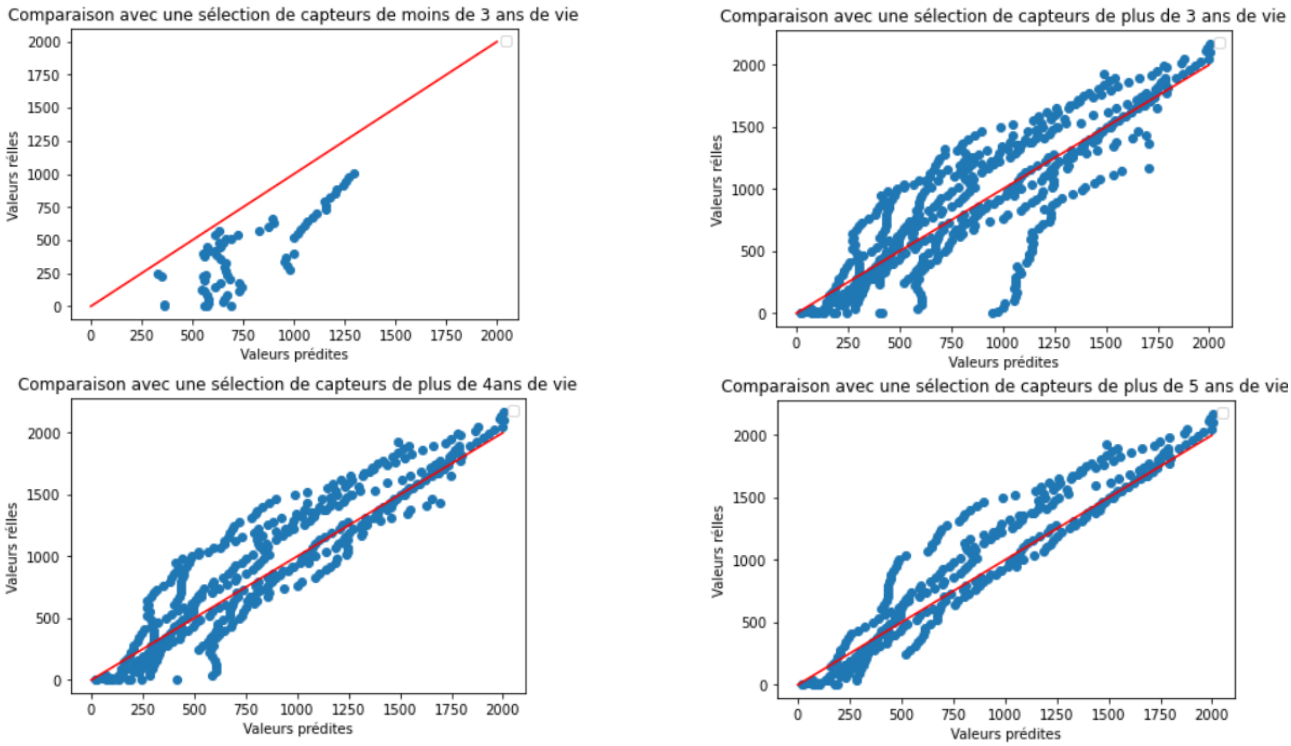



Figure 10: Comparaison des valeurs prédites et des valeurs réelles pour différent historique de vie

| Historique de vie | MAE (en jours) | $R^2$ (en %) |
|---|---|---|
| **moins de 3 ans** | 375 | -1.26 |
| **plus de 3 ans** | 219 | 0.72 |
| **plus de 4 ans** | 183 | 0.83 |
| **plus de 5 ans** | 158 | 0.87 |

Table 5: Résultats des scocres d'évaluations sur des modèles contenant des capteurs de différentes durées de vies

Ainsi, nous pouvons voir avec la Figure 10 et la Table 5 que plus les capteurs ont un historique de vie long, plus le modèle est performant. La Figure 10 montre que les prédictions des instances sont mieux ajustées à la droite y=x, et la Table 5 que les scores d'évaluations de la MAE et du coefficient de détermination $R^2$ sont meilleurs. En effet, pour un jeu de données ne comprenant que des capteurs ayant plus de 5 ans de vie, la MAE est de 158 jours et le coefficient de détermination est de 87%. Cela signifie que le modèle a une précision de 87% et qu'il prédit avec une marge d'erreur d'environ 5 mois, ce qui représente une meilleure performance par rapport aux prédictions faites sur un jeu de données contenant des capteurs ayant des durées de vie allant de moins d'un an à plus de 5 ans.

### 5.3.5 Évaluation en conditions réelles

Comme nous l'avons vu dans la partie **5.1.4**, le jeu de test utilisé pour présenter les résultats précédents contient 546 instances réparties sur 13 capteurs. Toutefois, cette quantité étant insuffisante pour garantir une bonne qualité en production, nous avons décidé d'évaluer notre modèle sur un autre jeu indépendant, composé uniquement de capteurs actuellement en fonctionnement. En effet, jusqu'à présent, nous avons évalué l'efficacité du modèle uniquement sur des capteurs dont la batterie est épuisée. Il est donc pertinent d'obtenir une estimation des faux positifs sur des capteurs encore en fonctionnement.

Cette phase d'évaluation permettra de tester la robustesse du modèle sur des données en temps réel et de vérifier sa capacité à prédire avec précision la durée de vie restante des capteurs encore en activité. Le jeu de données d'entraînement ne contenant que 65 capteurs, il est possible que le modèle n'ait pas eu suffisamment de données pour généraliser avec certitude. En testant le modèle dans des conditions réelles, nous pourrons mieux évaluer sa capacité de généralisation et déterminer s'il est prêt à être déployé en production.

Pour poursuivre cette étude, j'ai sélectionné une centaine de capteurs encore en fonctionnement. J'ai construit un jeu de données pour ces 100 capteurs en suivant la même méthode que celle utilisée pour les 65 capteurs hors service, afin de garantir que les deux jeux de données contiennent exactement les mêmes variables, construites de la même manière. Il est crucial que le modèle, ayant appris à partir de 12 variables explicatives, soit testé sur un jeu de données ayant également 12 variables pour assurer la cohérence et la validité des prédictions.

Une fois le jeu de données des capteurs fonctionnels préparé, l'étape suivante consiste à appliquer le modèle entraîné pour prédire la durée de vie restante de ces capteurs. Ces prédictions seront ensuite analysées et comparées aux données réelles.

Cependant, évaluer les performances du modèle sur ce jeu de données est moins trivial que sur le jeu de données composé de capteurs défaillants. En effet, les métriques d'évaluation classiques ne sont pas applicables ici, car les capteurs n'ont pas encore atteint la phase de défaillance, ce qui rend impossible la comparaison avec les valeurs réelles.

Pour remédier à cette situation, j'ai décidé de conserver les prédictions de durées de vie pour des intervalles de six mois, jusqu'à six ans de vie pour les 100 capteurs. Par exemple, si le modèle a prédit qu'à la date des deux ans de vie d'un capteur, que ce dernier n'a plus de batterie alors qu'il est toujours actif aujourd'hui, la prédiction est clairement erronée. En revanche, si le modèle prédit que le capteur cessera de fonctionner dans un an à partir de la date actuelle, cela montre une bonne capacité de prédiction. Cette comparaison permettra d'évaluer la précision du modèle en conditions réelles et d'identifier toute divergence entre les prédictions et la réalité.

Voici un exemple de cette explication avec la Table 6.

| Identifiant | Date à chaque 6 mois de vie | Date actuelle | Date d'Arrivée Prédite (Y) | Évaluation |
|---|---|---|---|---|
| ID1 | 01/08/2020 | 01/08/2024 | 22/07/2022 | - 2 ans et 10 jours |
| ID1 | 01/02/2021 | 01/08/2024 | 15/06/2023 | - 1 an 1 mois et 17 jours |
| ID1 | 01/08/2021 | 01/08/2024 | 21/12/2023 | - 7 mois et 11 jours |
| \| | ... | ... | ... | ... |
| ID1 | 01/08/2024 | 01/08/2024 | 09/09/2026 | + 2 ans 1 mois et 8 jours |
| ID2 | 01/01/2022 | 01/08/2024 | 02/06/2023 | - 7 mois et 11 jours |
| ID2 | 01/07/2022 | 01/08/2024 | 17/01/2024 | - 6 mois et 15 jours |
| ID2 | 01/01/2023 | 01/08/2024 | 09/04/2025 | + 8 mois et 8 jours |
| \| | ... | ... | ... | ... |
| ID2 | 01/08/2024 | 01/08/2024 | 11/01/2027 | + 2 ans 5 mois et 11 jours |

Table 6: Résumé des prédictions de durée de vie pour les capteurs en fonctionnement

Ainsi, je me suis intéressée à évaluer combien de fois le modèle a fourni une prédiction plausible, c'est-à-dire une date de fin de vie estimée pour les capteurs qui soit postérieure à la date actuelle. Ce critère est crucial, car une prédiction réaliste doit nécessairement indiquer que les capteurs sont encore en fonctionnement pour que la durée de vie restante soit significative. Les résultats obtenus sont les suivants :

| entre 1 et 3 ans | entre 3 ans et 6 ans |
|---|---|
| 57 % | 81 % |

Table 7: Pourcentages de prédictions possibles en fonction de la durée de vie des Capteurs

Comme on peut le voir sur la Table 7, seulement 57% des prédictions pour les instances ayant un historique de vie compris entre 1 et 3 ans (avec un intervalle de 6 mois) sont jugées possibles, c'est-à-dire que les dates prédites pour ces instances sont postérieures à la date actuelle. En revanche, pour les instances ayant un historique de vie entre 3 et 6 ans, 81% des prédictions sont possibles. Ces résultats confirment que notre modèle semble mieux performer sur des capteurs ayant un historique de vie plus long.

# 6 Conclusion et perspectives

L'objectif de cette étude était de développer une solution de prédiction de la durée de vie restante des batteries de capteurs, dans le but d'optimiser la maintenance prédictive. Pour atteindre cet objectif, nous avons exploré des modèles d'apprentissage supervisé par régression.

Parmi ces modèles, le Random Forest s'est distingué par ses performances. L'évaluation initiale sur un jeu de test de taille réduite a montré que ce modèle fournissait des prédictions satisfaisantes. En effet, le modèle est capable de prédire, avec une erreur de plus ou moins 230 jours, la date d'arrêt de la batterie des capteurs, ce qui, pour une moyenne de vie de 5 ans, est un résultat satisfaisant. De plus, il s'est avéré que le modèle a montré plus de précision dans ses prédictions lorsqu'il s'agissait de capteurs ayant un historique de vie de plus de 3 ans. En effet, lors des tests sur des capteurs en fonctionnement, le modèle a démontré sa capacité à prédire avec une certaine précision dans un pourcentage significatif des cas, en particulier pour les capteurs avec une durée de vie plus longue. Cependant, certaines incertitudes demeurent quant à la précision des prédictions, principalement en raison de la quantité limitée de données disponibles, des incertitudes liées à leur qualité, ainsi que de l'absence de données labellisées, notamment en ce qui concerne les états spécifiques de la batterie.

De nouvelles analyses seront menées dans la suite de ce stage afin d'atteindre un niveau de précision suffisant pour une utilisation opérationnelle. Les prochaines étapes de ce projet consisteront à affiner le modèle, à tester son déploiement dans des environnements réels, et à explorer d'autres approches de prédiction. Nous accorderons également une attention particulière à l'analyse non supervisée, qui, bien qu'entamée durant ce stage, n'a pas produit de résultats satisfaisants.

L'intérêt concret de ce travail aura résidé dans la capacité à anticiper les défaillances de la batterie des capteurs. Si un modèle de prédiction de la durée de vie des batteries précis est obtenu, l'objectif d'éviter au maximum la perte de données sera atteint. Cela permettra également aux déployeurs d'iQspot de localiser, dans une même zone, les capteurs risquant de s'arrêter simultanément, réduisant ainsi les déplacements nécessaires pour leur remplacement.

En conclusion, ce projet représente une avancée significative vers une solution de maintenance prédictive efficace. Les prochaines phases d'analyse et de développement viseront à renforcer cette solution, avec l'objectif ultime de déployer un outil fiable et opérationnel.

# Annexe

Dans cette partie annexe, je vais présenter plus en détail chaque modèle d'apprentissage supervisé. N'ayant pas approfondi la partie théorique des modèles dans la pratique de mon stage, j'ai choisi de ne pas entrer dans les détails dans le corps principal de ce rapport.

## Annexe 1 : Kaplan Meier

Le modèle de Kaplan-Meier est une méthode statistique utilisée principalement en analyse de survie pour estimer la fonction de survie à partir de données de durée de vie. En résumé, il permet de mesurer la fraction d'individus encore vivants après un certain moment tt. La courbe représentant cette fonction de survie est constituée de marches horizontales qui diminuent à chaque événement.

L'avantage de ce modèle est sa capacité à gérer les données censurées, c'est-à-dire les données pour lesquelles l'événement d'intérêt (dans notre cas, le décès des capteurs) n'a pas été observé pour certains individus durant la période d'étude. Ce phénomène est appelé "censure par la droite".

La fonction de survie S(t) est la probabilité qu'un individu survive au-delà d'un certain temps t. En général, cette probabilité diminue avec le temps. L'estimateur de Kaplan-Meier fournit une estimation de cette fonction. Il est calculé à l'aide de la formule suivante :

$$\widehat{S} = \prod_{t_i<t} \frac{n_i - d_i}{n_i}$$

- $t_i$ représente le temps au moment du i-ème événement.
- $d_i$ est le nombre d'événements (nombre de décès) au temps $t_i$
- $n_i$ est le nombre de sujets "à risque" juste avant le temps $t_i$ (c'est-à-dire les individus qui n'ont pas encore eu l'événement d'intérêt ou été censurés).

L'estimateur de Kaplan-Meier produit une courbe de survie en escalier, où chaque diminution correspond à un décès observé. Cette courbe permet de comparer les distributions de survie entre différents groupes.

En analyse de survie, il existe également la fonction d'échec, qui est complémentaire à la fonction de survie. Elle est définie comme suit :

$$F(t) = 1 - S(t)$$

où F(t) est la fonction d'échec.

Elle représente la probabilité cumulative qu'un individu ait expérimenté l'événement d'intérêt (le décès) avant ou à un temps t. En d'autres termes, plus le temps passe, plus la probabilité de décès ou d'échec augmente.

## Annexe 2 : Modèle de Cox

Le deuxième modèle d'analyse de survie que j'ai utilisé durant mon stage est le modèle de Cox. Ce modèle est l'une des méthodes les plus couramment utilisées pour analyser des données de survie. Le modèle de Cox, également connu sous le nom de modèle des risques proportionnels, est un modèle de régression qui cherche à expliquer comment les variables explicatives sont liées à un événement, dans notre cas, l'arrêt de la batterie.

Ce modèle peut être appliqué à toute situation où l'on étudie le délai avant que l'événement d'intérêt ne survienne. Pour chaque individu, on connaît la date de la dernière observation ainsi que son état par rapport à l'événement étudié. Cependant, pour certains individus, l'état à la date de fin de l'étude n'est pas connu ; ces individus sont donc considérés comme censurés. L'avantage du modèle de Cox est qu'il permet de prendre en compte ces données même si elles sont incomplètes.

Deux variables sont cruciales dans l'analyse de survie :

- **La variable de survie T** : qui représente le temps de survie jusqu'à la survenue de l'événement.
- **La variable d'état S** : qui indique si l'événement s'est produit ou non pour chaque individu à un moment donné.

Le modèle de Cox est considéré comme un modèle semi-paramétrique et repose sur l'hypothèse des risques proportionnels. Il permet d'exprimer la fonction de risque instantané d'arrêt de la batterie, notée $\lambda(t, X_1, X_2, ...., X_n)$, qui représente la probabilité que l'événement survienne en fonction de l'instant t, sachant que l'individu est encore vivant juste avant t, et en fonction des variables explicatives Xj, appelées facteurs de risque. La probabilité qu'un sujet k décède au temps t, sachant qu'il est vivant juste avant, est donnée par la formule suivante :

$$\lambda(t, X_1, X_2, ...., X_n) = \lambda_0(t) \exp(\sum_{i=1}^{n} \beta_i X_i)$$

Le risque instantané se décompose en deux termes : l'un dépend du temps t , $\lambda_0(t)$ et l'autre des variables $X_j$, $\exp(\sum_{i=1}^{n} \beta_i X_i)$. Si toutes les variables explicatives sont nulles, alors $\lambda(t, X_1, X_2, ...., X_n) = \lambda_0(t)$ et $\lambda_0(t)$ est appelé le risque de base, c'est-à-dire le risque en l'absence de tout facteur de risque.

On dit que le modèle de Cox est semi-paramétrique car il ne cherche pas à estimer directement la fonction $\lambda_0(t)$, qui est identique pour tous les individus à un instant donné. L'objectif est plutôt d'estimer les rapports de risques instantanés de décès entre deux individus ayant des facteurs de risque différents (c'est-à-dire des valeurs de variables explicatives différentes). Le modèle repose sur l'hypothèse des risques proportionnels, qui spécifie que le rapport des risques instantanés reste constant dans le temps. En d'autres termes, l'effet des variables sur le risque de décès ne change pas au fil du temps et est multiplicatif.

Par exemple, si l'on considère deux individus j1 et j2 qui ne diffèrent que par la valeur d'une seule variable $X_k$, avec $X_k$=0 pour j1 et $X_K$=1 pour j2, alors quel que soit le temps t:

$$\frac{\lambda(t,j2)}{\lambda(t,j1)} = \frac{\lambda_0(t)\exp(\beta_1 X_1' + ... + \beta_{k-1}X_{k-1}' + \beta_k \times 1 + \beta_{k+1}X_{k+1}' + ... + \beta_n X_n')}{\lambda_0(t)\exp(\beta_1 X_1' + ... + \beta_{k-1}X_{k-1}' + \beta_k \times 0 + \beta_{k+1}X_{k+1}' + ... + \beta_n X_n')} = \exp(\beta_k)$$

Le rapport est donc indépendant du temps, ce qui signifie que, quel que soit le temps t, l'individu j2 a un risque instantané de mourir $\exp(\beta_k)$ fois supérieur à celui de l'individu j1. Il s'agit d'une hypothèse forte du modèle de Cox, et il est donc nécessaire de vérifier que cette hypothèse est respectée pour chaque variable avant d'appliquer le modèle. Si l'effet est constant, il peut être bénéfique, nocif ou nul. La dernière étape consiste à estimer les coefficients $\beta_k$ en utilisant le principe du maximum de vraisemblance.

## Annexe 3 : Régression Linéaire multiple

La régression linéaire permet de prédire la valeur d'une variable dépendante y en fonction d'une ou plusieurs variables indépendantes. Cette méthode établit donc une relation linéaire entre x (l'entrée) et y (la sortie).
Dans le cas d'une régression linéaire multiple, le modèle établit une relation entre plusieurs variables indépendantes, appelées variables explicatives x_i, et une variable de sortie continue y, que l'on cherche à expliquer. Le modèle de régression linéaire multiple s'exprime sous la forme suivante :

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_n x_n + \epsilon_i$$

où :

- $\beta_0$ est l'ordonnée à l'origine (intercept)
- $\beta_i$ sont les coefficients associés à chaque variable explicative x_i, $\epsilon_i$ représente l'erreur du modèle, c'est-à-dire la différence entre la valeur prédite et la valeur réelle.

Contrairement à la régression linéaire simple, qui n'utilise qu'une seule variable explicative, la régression linéaire multiple permet d'intégrer plusieurs variables indépendantes. Cela peut potentiellement améliorer la précision du modèle, à condition que les variables ajoutées soient pertinentes et apportent une réelle contribution à la prédiction.
L'objectif est de trouver une fonction qui prédit y tout en minimisant l'erreur de prédiction.

Pour cet algorithme, il n'est pas nécessaire de spécifier de paramètres en entrée, les coefficients $\beta_i$ sont ajustés par le modèle au cours de l'entraînement.

## Annexe 4 : SVR

Le modèle SVR (Support Vector Regression) est un algorithme de régression basé sur les principes du Support Vector Machine (SVM), mais adapté aux problèmes de régression. L'objectif de cet algorithme est de trouver une fonction $f(x) = w^T x + b$ qui prédit y la variable cible, à partir des vecteurs d'entrée $x$, où $w$ est le vecteur poids et $b$ le bais.

Le principe de SVR est de positionner cette fonction de manière à ce qu'elle sépare les données tout en maximisant la distance, appelée « marge », entre les points de données et cette fonction. Contrairement à la régression linéaire classique, où le but est de minimiser l'erreur de prédiction pour chaque point de données, SVR ignore les erreurs qui se trouvent à l'intérieur de cette marge $\epsilon$, car les prédictions dans cette zone sont considérées comme suffisamment précises. Seules les erreurs situées en dehors de cette marge sont pénalisées, ce qui aide à maximiser la marge entre les points de données et la fonction prédictive.

Cependant, dans de nombreux cas, il n'est pas possible de séparer linéairement les données dans leur espace d'origine. Pour surmonter cette difficulté, SVR utilise une technique appelée « noyau » pour transformer les données dans un espace de dimension supérieure où une séparation linéaire devient possible. Le choix du noyau est crucial, car il détermine la manière dont les données sont transformées.

L'objectif de SVR reste de minimiser l'erreur, tout en évitant les erreurs en dehors de la marge. La fonction de coût du SVR, qui inclut une régularisation pour minimiser la complexité du modèle, est généralement formulée comme suit :

$$\min_{(w,b)} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\varepsilon_i + \varepsilon_i^*)$$

où

- C est le paramètre de régularisation qui contrôle le compromis entre la minimisation de l'erreur et la régularisation du modèle
- $\varepsilon_i$ et $\varepsilon_i^*$ sont des variables qui pénalisent les erreurs situées en dehors de la marge.

L'algorithme SVR comporte donc plusieurs hyperparamètres clés qui doivent être ajustés pour obtenir des performances optimales. Les principaux hyperparamètres de SVR sont :

- **Kernel** : qui représente le noyau. Les choix courants de noyaux sont : linear (pour problèmes linéaires), poly (noyau polynomial), rbf (noyau gaussien) et sigmoid (noyau sigmoïde)
- **C** : qui représente le paramètre de régularisation
- **Epsilon** $\epsilon$ : qui définie la marge
- **Gamma** $\gamma$ : Applicable pour les noyaux rbf, poly et sigmoid, ce paramètre définit l'influence d'un seul exemple d'entraînement

## Annexe 5 : Random Forest

Le modèle de Random Forest, ou « Forêt aléatoire », est un algorithme de classification ou de régression qui repose sur l'assemblage de plusieurs arbres de décision indépendants. Un arbre de décision est une méthode qui divise les données en fonction de certaines conditions ou règles. Chaque nœud de l'arbre représente une question basée sur une variable du jeu de données, et chaque branche correspond à un résultat de cette question. Les feuilles de l'arbre représentent les décisions finales ou les prédictions.

Bien qu'un seul arbre de décision puisse offrir des résultats précis, sa performance dépend fortement de l'échantillon de données d'origine. Par exemple, l'ajout de nouvelles données à l'échantillon peut modifier le modèle et les résultats de manière significative. Le modèle Random Forest améliore la précision et la robustesse des prédictions en combinant les résultats de plusieurs arbres de décision, chacun ayant une vision partielle et unique du problème. Le modèle repose sur le principe que la combinaison de multiples avis (arbres) est généralement plus fiable qu'un seul avis.

Le terme "random" (aléatoire) dans Random Forest provient du double tirage aléatoire appliqué lors de la construction de chaque arbre, à la fois sur les observations et sur les variables. Les deux tirages aléatoires sont les suivants :

- Tree bagging : Cette technique est basée sur un tirage avec remplacement. Pour chaque arbre, une nouvelle base de données est créée en tirant aléatoirement des observations (lignes) de la base de données d'origine. Certaines lignes peuvent être sélectionnées plusieurs fois, tandis que d'autres peuvent ne pas être sélectionnées du tout. Ce principe de diversification des arbres permet de réduire la variance du modèle.
- Feature Sampling : À chaque nœud de l'arbre, un sous-ensemble aléatoire de variables (colonnes) est sélectionné. Seules les variables de ce sous-ensemble sont considérées pour déterminer la meilleure division du nœud. Cela réduit la corrélation entre les arbres, rendant le modèle plus diversifié et résilient.

Au final, tous ces arbres de décision indépendants sont assemblés. Pour un problème de régression, la prédiction du Random Forest pour des données inconnues est la moyenne des prédictions de tous les arbres. Pour la classification, la classe finale est celle qui a été prédite le plus souvent.

L'algorithme Random Forest nécessite plusieurs hyperparamètres qui doivent être définis avant l'entraînement. Voici les principaux hyperparamètres :

- **n_estimators** : qui représente le nombre d'arbres à utiliser dans la fôret. Plus le nombre d'arbres est élevé, plus la performance peut être stable, mais cela augmente aussi le temps de calcul
- **max_depth** : qui correspond à la profondeur maximale de chaque arbre, ce paramètre permet de contrôler le surapprentissage (overfitting) en limitant la profondeur des arbres
- **min_samples_split** : qui représente le nombre minimum d'échantillons requis pour diviser un nœud

- **min_samples_leaf** : le nombre minimum d'échantillons qu'une feuille peut contenir

  Dans les deux cas un nombre élevés de min_samples_split ou min_samples_leaf peut empêcher la complexité de l'arbre.

- **max_features** : qui prend comme valeur : 'auto', sqrt', 'log2', permet de définir le nombre de caractéristiques à considérer lors de la recherche de la meilleure scission

Il existe d'autres hyperparamètres, mais ils n'ont pas été pris en compte ici car ils sont jugés moins importants. Une fois ces hyperparamètres définis, le modèle Random Forest peut être utilisé pour résoudre des problèmes de régression ou de classification.

## Annexe 6 : Réseaux de Neurones

Un réseau de neurones est un modèle d'apprentissage inspiré du fonctionnement du cerveau humain. Il est composé de multiples couches de neurones artificiels, qui sont des unités de calcul simples connectées entre elles. Un réseau de neurones typique est constitué de trois types de couches :

– **Couche d'entrée** (input layer) : Elle reçoit les données brutes, telles que les pixels d'une image ou les valeurs numériques d'un tableau. Chaque neurone de cette couche représente une caractéristique de l'entrée.

– **Couches cachées** (hidden layers) : Ce sont les couches intermédiaires entre l'entrée et la sortie. Un réseau peut avoir une ou plusieurs couches cachées. Chaque neurone d'une couche cachée reçoit des signaux des neurones de la couche précédente. Il effectue une somme pondérée de ces signaux tel que :

$$z_i = \sum_j w_j x_j + b_i$$

où $z_i$ est l'entrée pondérée pour le neurone i, $w_j$ est le poids de la connexion entre le neurone j et le neurone i, $x_j$ est la sortie du neurone j, et $b_i$ est le biais du neurone i.
Et applique une fonction d'activation (une transformation mathématique) tel que :

$$a_i = \phi(z_i)$$

où $a_i$ est la sortie du neurone i, et $\phi$ est la fonction d'activation. Puis le résultat est transmis aux neurones de la couche suivante.

– **Couche de sortie** (output layer) : Elle produit le résultat final du réseau, qui peut être une classe dans un problème de classification, une valeur numérique pour un problème de régression, ou encore une probabilité.

Les neurones des couches cachées et de sortie sont connectés aux neurones des couches précédentes par des poids, qui sont des paramètres ajustés pendant l'entraînement. Chaque connexion a un poids associé qui détermine l'importance d'un neurone pour un autre.

Lors de l'entraînement d'un réseau de neurones, l'objectif est de déterminer les poids optimaux pour minimiser l'erreur entre les prédictions du réseau et les valeurs réelles. Ce processus utilise une méthode appelée propagation arrière : *backpropagation*.

- Propagation Avant *Forward Propagation* :
  Les données d'entrée sont passées à travers les différentes couches du réseau pour générer une prédiction. Cette prédiction est ensuite comparée à la valeur réelle pour calculer l'erreur.

- Propagation Arrière *backpropagation* :
  L'erreur calculée est rétropropagée ( l'erreur est propagée en sens inverse) à travers le réseau en sens inverse, depuis la couche de sortie vers les couches précédentes. Pour chaque poids, le gradient est calculé en utilisant la dérivée de la fonction de perte par rapport à ce poids et en tenant compte de l'activation du neurone dans la couche précédente. Les poids sont ajustés en suivant ces gradients pour réduire l'erreur.

L'algorithme de descente de gradient est une méthode utilisée pour minimiser l'erreur en ajustant les poids du réseau. Il ajuste les poids dans la direction opposée au gradient de l'erreur pour réduire cette perte.

Ce processus est répété sur plusieurs itérations, ce qui permet au réseau d'améliorer progressivement ses prédictions en ajustant les poids de manière optimale.

Le réseau de neurones comporte plusieurs hyperparamètres qui doivent être ajustés pour obtenir de bonnes performances. Les principaux hyperparamètres sont :

- Le **nombre de couches cachées et de neurones par couche** : Plus le réseau est profond (avec de nombreuses couches) et large (avec de nombreux neurones par couche), plus il est capable de capturer des relations complexes, mais cela augmente également le risque de surapprentissage (overfitting).

- Le **taux d'apprentissage** (learning rate) : Ce paramètre contrôle la vitesse à laquelle les poids sont mis à jour. Un taux d'apprentissage trop élevé peut entraîner une convergence instable, tandis qu'un taux trop faible peut rendre l'entraînement trop long.

- Les **fonctions d'activation** : Ces fonctions introduisent des non-linéarités dans le réseau, permettant au modèle de capturer des relations non linéaires dans les données. Les fonctions d'activation courantes incluent :
  $\rightarrow$ Sigmoïde : $\sigma(x) = \frac{1}{1+e^{-x}}$
  $\rightarrow$ ReLU (Rectified Linear Unit) : $\mathrm{ReLU}(x) = \max(0, x)$
  $\rightarrow$ Tanh : $\tanh(\mathrm{x}) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

- **L'optimiseur** : Les algorithmes d'optimisation ajustent les poids et les biais pour minimiser la fonction de perte. Quelques optimiseurs populaires sont :
  $\rightarrow$ Adam, Adadelta

- Le **nombre d'itérations** (epochs) : Il s'agit du nombre de fois que le réseau passe sur l'ensemble des données d'entraînement. Un plus grand nombre d'époques permet au réseau d'apprendre davantage, mais au risque de surapprentissage.

- **Taille du lot** (batch size) : Nombre d'échantillons traités avant la mise à jour des poids.

Une fois le réseau de neurones bien configuré et entraîné, il peut être utilisé pour effectuer des prédictions sur de nouvelles données, offrant souvent des performances impressionnantes sur des tâches complexes où les relations entre les variables ne sont pas évidentes.

# Bibliographie

# References

[1] Ministère de la Transition Écologique, *Statistiques de l'immobilier tertiaire en 2024*, Revue de l'Écologie Urbaine, 2024.

[2] Ministère de la Transition Écologique, *Dépenses énergétiques du secteur tertiaire en 2019*, Revue de l'Écologie Urbaine, 2019.