# Prédiction et Explicabilité de la Dégradation des Batteries : Modèles de Prédiction du SOH basés sur les Données de Batteries du MIT

Projet réalisé par :
- Zoé Marquis
- Charlotte Kruzic

## Introduction

Dans un contexte de prédiction de séries temporelles, ce projet vise à développer un modèle prédictif pour anticiper l'évolution de l'état de santé (State of Health, SOH) des batteries tout en identifiant les facteurs influençant leur dégradation grâce à des techniques d'explicabilité. La complexité des données disponibles et les spécificités des séries temporelles ont nécessité une préparation méthodique des données et une optimisation des paramètres avant la mise en œuvre des modèles.

# Méthodologie

La première étape a consisté à préparer les données, en construisant des fenêtres glissantes (*sliding windows*) adaptées au contexte de séries temporelles. La taille de ces fenêtres, ainsi que le stride (décalage), étant des hyperparamètres cruciaux, ils ont été déterminés de manière expérimentale. Pour cela, nous avons initialement testé différentes configurations avec un modèle de forêt aléatoire (Random Forest), en veillant à équilibrer précision et temps de calcul, tout en écartant les strides trop petits pour limiter les durées de traitement excessives.

Nous avons adapté la préparation des données en fonction des différents types de modèles:

- Pour les modèles capables de gérer des données temporelles en 3D, comme les réseaux de neurones récurrents (RNN), les fenêtres glissantes ont été utilisées directement sous leur forme 3D.
- Pour les modèles basés sur les arbres (Random Forest, XGBoost) ou les régressions (linéaire, logistique, etc.), qui nécessitent des données en 2D, il a fallu "flatten" les fenêtres glissantes. Chaque série temporelle a ainsi été représentée par un vecteur 2D enrichi d'informations supplémentaires extraites des batteries, notamment des caractéristiques globales ajoutées comme colonnes communes à tous les pas de temps.

Enfin, la variable catégorielle a été encodée différemment selon le type de modèle :

- Label encoding pour les modèles capables de gérer des données ordinales ou catégoriques.
- Binary encoding dans les autres cas, pour éviter l'explosion du nombre de colonnes, contrairement à un one-hot encoding qui aurait été trop coûteux en ressources.

Pour garantir une analyse cohérente entre les modèles et permettre une étude approfondie de l'explicabilité, nous avons extrait un sous-échantillon composé de 5 batteries provenant de différents lots (*batches*) et de 5 instants temporels distincts pour chaque batterie. Ce sous-échantillon a été utilisé spécifiquement pour les analyses d'explicabilité, tout en veillant à ce que les mêmes données soient utilisées de manière uniforme entre les différents

Pour les modèles Random Forest, XGBoost, régression linéaire, Lasso, Ridge et Elastic Net, nous avons effectué une recherche par grille (*grid search*) afin de déterminer les hyperparamètres optimaux.

Tous les entraînements des modèles ont été réalisés avec une **validation croisée** à 5 plis, en veillant à ce que la division en plis soit identique pour tous les notebooks. En effet, pour garantir la cohérence et la reproductibilité des résultats, une graine aléatoire commune (random seed) a été fixée dans tous les notebooks et pour tous les modèles. Cela a permis d'évaluer chaque modèle sur les mêmes sous-ensembles de données, assurant une comparaison équitable des performances.

Une fois les modèles entraînés, nous avons identifié le modèle le plus performant parmi Random Forest, XGBoost, régression linéaire, Lasso, Ridge, Elastic Net, RNN, CNN 1D, et

LSTM, en comparant leurs performances selon plusieurs métriques : MSE, MAE, RMSE et MAPE.

Nous avons choisi les modèles CNN 1D et LSTM pour leur capacité à exploiter les relations temporelles dans les données, en capturant à la fois les dépendances locales (avec CNN 1D) et les dépendances longues (avec LSTM).

Nous avons utilisé ces différentes métriques pour obtenir une évaluation complète et équilibrée :

- MSE (Mean Squared Error) : Met en évidence les grandes erreurs en les pondérant plus lourdement, ce qui est utile pour détecter des prédictions éloignées.
- MAE (Mean Absolute Error) : Fournit une mesure intuitive des erreurs moyennes en unités des données, moins sensible aux valeurs aberrantes.
- RMSE (Root Mean Squared Error) : Traduit les erreurs en une échelle comparable aux données d'origine tout en conservant la sensibilité aux grandes erreurs.
- MAPE (Mean Absolute Percentage Error) : Permet d'évaluer les erreurs en pourcentage par rapport aux valeurs réelles, rendant les résultats comparables entre différents ensembles de données.

L'analyse combinée de ces métriques nous a permis d'identifier un modèle offrant une performance robuste, tout en prenant en compte les différentes sensibilités et échelles des erreurs.

Une fois les modèles entraînés et les hyperparamètres optimisés, XGBoost a émergé comme le meilleur modèle en raison de ses bons résultats et de sa robustesse face aux différentes métriques d'évaluation (MSE, MAE, RMSE, et MAPE). Afin d'approfondir l'analyse des prédictions de ce modèle, nous avons appliqué des techniques d'explicabilité pour mieux comprendre l'impact des différentes caractéristiques sur les résultats. Cela a inclus l'examen des importances des features par des critères classiques, l'inspection des arbres de décision pour visualiser l'impact de chaque caractéristique, ainsi que l'utilisation de SHAP pour obtenir des explications plus détaillées à la fois au niveau global et local. Ces analyses ont permis de mettre en lumière les relations complexes entre les features d'entrée et les prédictions du modèle, offrant ainsi une meilleure compréhension des facteurs influençant les décisions de XGBoost. Cette approche d'explicabilité a été essentielle pour valider la performance du modèle tout en fournissant une interprétation transparente et fiable de ses prédictions.

# Arborescence et explication du code

## Organisation des dossiers

L'organisation des fichiers et dossiers permet une gestion claire et efficace des données, modèles et résultats du projet. Voici une description des principaux dossiers :

- **`archive`** : Ce dossier contient d'anciens notebooks utilisés au cours de la phase de développement. Ces fichiers documentent les étapes intermédiaires ou expérimentales du projet.
- **`Data`** : Regroupe les données brutes et les fichiers produits lors des phases de préparation (ex. : fenêtres glissantes). Ce dossier est exclu du dépôt Git via un fichier `.gitignore`, afin d'éviter une surcharge inutile du dépôt.
- **`Data_to_Share`** : Contient les performances des différents modèles, utilisées principalement dans le notebook `7_results` pour les comparaisons.
- **`Images`** : Stocke les graphiques et visualisations générés, notamment ceux liés à l'explicabilité des modèles.
- **`Models`** : Ce dossier contient les paramètres des modèles entraînés ainsi que les sauvegardes générées en fin d'entraînement. Ces fichiers permettent de réutiliser les modèles sans avoir à les réentraîner.

## Préparation de l'environnement

Avant de lancer le projet, un environnement virtuel est nécessaire pour garantir la reproductibilité et l'indépendance des dépendances. Les étapes de configuration sont les suivantes :

1. Création d'un environnement virtuel :

```
python3.11 -m venv mon_venv
source mon_venv/bin/activate
pip install --upgrade pip
pip install -r requirements.txt
```

2. Les fichiers `.mat` nécessaires doivent être placés dans le dossier `Data/`.
3. Exécution du script MATLAB `script_barcode_channel_id.m` pour générer les premières données d'entrée.

## Ordre d'exécution des notebooks

Pour assurer une progression logique et reproductible, les notebooks doivent être exécutés dans l'ordre suivant :

1. **`1_preparation.ipynb`** : Préparation des données, comprenant le nettoyage et la création des fenêtres glissantes.

2. **`2_rf_search_window.ipynb`** : Recherche de la taille optimale des fenêtres à l'aide de Random Forest.
3. **`3_1_rf_gridsearch.ipynb`** : Recherche des meilleurs hyperparamètres pour Random Forest à l'aide d'un grid search.
4. **`3_2_rf.ipynb`** : Entraînement final de Random Forest avec les hyperparamètres optimaux, suivi de l'analyse SHAP pour une explicabilité partielle.
5. **`4_regressions_grid_search.ipynb`** : Grid search pour des modèles de régression (linéaire, Ridge, Lasso, Elastic Net), avec analyse SHAP du meilleur modèle.
6. **`5_1_xgboost_gridsearch.ipynb`** : Recherche des meilleurs hyperparamètres pour XGBoost via un grid search.
7. **`5_2_xgboost.ipynb`** : Entraînement final de XGBoost avec sauvegarde du modèle.
8. **`6_cnn.ipynb`** : Entraînement d'un réseau de neurones convolutionnel (CNN) 1D.
9. **`6_lstm.ipynb`** : Entraînement d'un modèle LSTM.
10. **`6_rnn.ipynb`** : Entraînement d'un modèle RNN.
11. **`7_results.ipynb`** : Comparaison des performances des différents modèles entraînés.
12. **`8_xgboost_XAI.ipynb`** : Analyse d'explicabilité détaillée du meilleur modèle, ici XGBoost.

## Remarque

Le notebook **`9_cnn_timeshap.ipynb`** documente un essai d'implémentation de TimeSHAP. Ce fichier n'est pas essentiel dans le pipeline final mais peut être utile pour explorer des approches alternatives.

# Préparation des données

## Chargement des données

Les données utilisées pour ce projet sont issues de trois fichiers (2017-05-12, 2017-06-30 et 2018-04-12), de données des batteries du MIT, utilisées dans le projet *Data-driven prediction of battery cycle life before capacity degradation* et contiennent des informations issues de tests sur des batteries lithium-ion.

Chacun de ces fichiers contient un batch, c'est à dire une série d'expérimentations spécifiques, et comprend un total combiné de 124 batteries lithium-ion, testées jusqu'à leur défaillance. Ces batchs présentent des particularités propres liées aux conditions de test et aux politiques de charge.

Ces fichiers étant au format MATLAB, un format que nous n'avions jamais utilisé, nous avons dû utiliser les scripts Load Data.ipynb et BuildPkl_BatchX.ipynb, disponible dans le dépôt GitHub associé au projet, pour extraire les données des fichiers et les convertir en un format exploitable, afin de pouvoir les utiliser pour notre projet.

## Exploration des données

Les données comprennent un total de 124 batteries réparties en trois batch, présentant des différences dans les conditions de test et les traitements appliqués aux batteries, comme indiqué dans la documentation.

### Structure des données

Chaque batterie est décrite avec les éléments suivants :

- *charge_policy* : paramètres de charge au format C1(Q1)-C2
    - C1 et C2 : taux de charge en courant
    - Q1 : seuil de capacité atteint (%)
- *cycle_life* : durée de vie en cycles
- *barcode* : identifiant unique de chaque batterie
- *channel_id* : identifiant du canal utilisé pour le test
- *summary* : dictionnaire contenant les données par cycle. Il contient :
    - cycle : numéro de cycle (indicateur de l'ordre dans notre série temporelle)
    - QD : capacité de décharge
    - QC : capacité de charge
    - IR : résistance interne
    - Tmax : températures maximale
    - Tavg : températures moyenne
    - Tmin : températures minimale
    - chargetime : temps de charge.
- *cycles* : dictionnaire contenant les données détaillées pour chaque cycle, avec des séries temporelles pour :
    - Temps (t), capacité de charge (Qc), courant (I), tension (V), température (T).
    - Capacités linéarisées (Qdlin, Tdlin) et dérivée capacité-tension (dQdV).

### Vérifications et justification des choix

Bien que le champ *cycles* contienne des séries temporelles très détaillées, nous avons décidé de ne pas l'utiliser en raison de la complexité de sa structure et du temps disponible limité pour ce projet. Nous avons préféré utiliser le champ *summary*, qui fournit des données sur les cycles plus simples à manipuler.

Nous avons également comparé les données extraites avec les informations de la documentation. Les données correspondaient bien, à l'exception du champ *cycle_life* dans le batch 2 qui présentait des tailles différentes. Aucune explication n'a été trouvée dans la documentation, mais nous pensons que cette différence n'est pas fondamentale dans notre analyse, donc nous ne l'avons pas traitée.

Dans la documentation fournie, il est mentionné que certaines batteries du batch 1 ont été réutilisées dans le batch 2.
Pour pouvoir joindre ces données et identifier les batteries concernées, nous avions besoin des champs *barcode* et *channel_id*. Pour les récupérer, nous avons modifié le code en MATLAB (cf. fichier script_barcode_channel_id.m) pour extraire et formater ces informations. Cependant, nous nous sommes rendu compte que les données correspondants à ces batteries (communes aux batch 1 et 2) avaient été supprimées dans le fichier Load Data.ipynb.

```
del batch2['b2c7'] # c'est batch 2 channel 1 (suite de batch 1 channel 1)
del batch2['b2c8'] # c'est batch 2 channel 2 (suite de batch 1 channel 2)
del batch2['b2c9'] # c'est batch 2 channel 3 (suite de batch 1 channel 3)
del batch2['b2c15'] # c'est batch 2 channel 5 (suite de batch 1 channel 5)
del batch2['b2c16'] # c'est batch 2 channel 6 (suite de batch 1 channel 6)
```

Nous avons donc pris la décision de laisser les choses comme elles étaient, et de ne pas récupérer ces batteries. Cela nous a permis d'éviter toute problématique liée à l'identification de ces batteries ou au recalcul du SOH, qui aurait nécessité de prendre en compte les cycles du batch 1 pour compléter les données du batch 2.

## Création d'un dataframe

Afin d'avoir des données exploitables pour l'analyse et les modèles, nous avons créé un dataframe structuré auquel nous avons ajouté des champs calculés.

### Extraction de données

Nous avons commencé par extraire les composants C1, Q1 et C2 de la politique de charge C1(Q1)-C2 des batteries, et nous avons créé des colonnes dédiées.

Nous avons extrait depuis les identifiants des batteries, le batch duquel elles proviennent afin de pouvoir ajouter, pour nos analyses, les métadonnées qui y sont liées.

Nous avons extrait les données contenues dans *summary* pour les transformer en colonnes distinctes, notamment en créant une colonne pour QD1 (la capacité nominale initiale, au cycle 1) et une autre pour QD2 (QD au cycle 2), utilisée dans les cas où QD1 est égale à 0.

Ensuite, nous avons pivoté les données de manière à ce que chaque cycle corresponde à une ligne, car les informations de summary étaient initialement agrégées sur l'ensemble des cycles.

Nous devons intégrer le SOH dans les données, la formule de l'énoncé est :

$$SOH = \frac{Q_{\text{actuelle}}}{Q_{\text{nominale}}} \times 100$$

D'après la documentation et nos recherches complémentaires, nous avons déterminé que le "Q" mentionné dans l'énoncé fait référence à la capacité de décharge, représentée par QD dans les données.

Le SOC (State of Charge) est calculé à partir de QC, qui correspond à la capacité de charge. Cette valeur indique l'état instantané de la batterie en pourcentage, en fonction de la charge disponible par rapport à sa capacité nominale.

Quant au SOH (State of Health), il est déterminé à partir de QD, car cette métrique reflète directement la capacité de décharge de la batterie par rapport à sa capacité initiale. En effet, le SOH est une mesure de la dégradation de la batterie, souvent définie comme le ratio entre la capacité actuelle de décharge (QD_actuelle) et la capacité de décharge initiale (QD_nominale). Cette relation permet d'évaluer dans quelle mesure la batterie a perdu en performance au fil du temps et des cycles de charge-décharge.

Nous avons donc ajouté une colonne Qnominale représentant la capacité nominale de la batterie. Comme dit précédemment, nous avons constaté que dans certains cas, la première valeur de décharge (QD1) était égale à 0, ce qui aurait rendu le calcul impossible. Pour gérer ces cas, si QD1 = 0, nous avons défini Qnominale = QD2, et sinon, Qnominale = QD1.

## Gestion des valeurs aberrantes

Au cours de l'analyse, nous avons identifié quatre séries temporelles présentant des valeurs de SOH (State of Health) supérieures à 102 %. Ces séries ont été supprimées du jeu de données, représentant seulement 4 cas sur un total de 124.

Les valeurs de SOH supérieures à 102 % sont considérées comme des anomalies pour plusieurs raisons. Bien que des valeurs proches ou légèrement supérieures à 100 % puissent parfois survenir, cela peut être attribué à des variations dans les mesures, des erreurs de calibration ou des conditions spécifiques où la batterie présente temporairement des performances légèrement supérieures à sa capacité nominale. Ce phénomène peut apparaître dans la plupart des batteries, et est généralement tolérable dans une certaine mesure.

Cependant, lorsque les valeurs de SOH dépassent significativement cette limite, comme dans le cas des séries atteignant plus de 102 %, elles deviennent physiquement et logiquement incohérentes. Ces écarts ne peuvent pas être expliqués par des fluctuations

normales et indiquent très probablement des erreurs dans la mesure, la saisie ou le traitement des données.

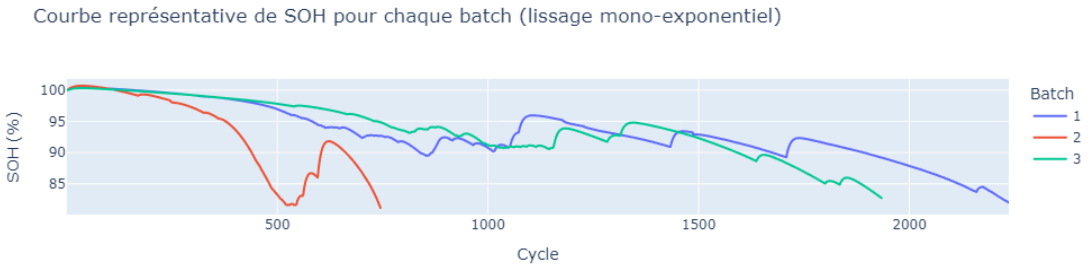
Ainsi, nous avons décidé de supprimer ces séries aberrantes, car elles pourraient biaiser les résultats du modèle et nuire à la qualité des prédictions. En revanche, les valeurs proches de 100 % ont été conservées, car elles restent plausibles dans le contexte des variations naturelles des performances des batteries.

# Observation des batchs

Une fois les données préparées, nous avons généré des courbes représentatives pour chaque série temporelle afin de mettre en évidence les tendances globales et mieux comprendre les dynamiques spécifiques à chaque batch.

Pour analyser les différences entre les batchs, nous avons utilisé différentes méthodes de lissage pour produire ces courbes "représentatives". Ces techniques permettent de réduire le bruit des données et de faire ressortir les tendances principales. Nous avons utilisé les 2 méthodes suivantes :

- Moyenne mobile (Moving Average) : Permet de lisser les fluctuations locales en utilisant une fenêtre glissante. Nous utilisons des fenêtres de 5 et 10 cycles pour capturer respectivement des variations locales rapides et des tendances globales.
- Lissage exponentiel :
    - Mono-exponentiel : plus de poids aux données récentes, adapté aux séries sans saisonnalité.
    - Double-exponentiel : inclut une composante pour la tendance.
    - Triple-exponentiel : ajoute une gestion de la saisonnalité.




Les résultats montrent clairement qu'il existe des différences entre les batchs, en particulier entre le batch 2 et les deux autres.

## Ajout de données de la documentation

Pour mieux comprendre ces variations, nous avons décidé d'ajouter des données sur les batchs issues de la documentation, afin de prendre en compte un maximum d'informations dans nos modèles.

Nous avons donc ajouté les colonnes, spécifiques à chaque batch, suivantes :

- *cycle to x% of nominal capacity*
- *cutoff_currents*
- *charging_rest*

- *after_discharging_rest*
- *before_discharging_rest*
- *IR_rest*
- *pulse_width*

Nous avons ensuite pu supprimer l'identifiant du batch.

### Normalisation / Standardisation des données

Pour éviter tout biais dans les modèles, nous avons procédé à la normalisation des données : toutes les données doivent se trouver entre 0 et 1.
Certaines colonnes n'ont pas été normalisées pour les raisons suivantes :

- *SOH* : Variable cible pour la prédiction.
- *barcode* : Cette variable va être encodée.

### Encodage

Nous avons utilisé 2 méthodes d'encodage, car les modèles que nous devons tester dans ce projet ne nécessitent pas forcément les mêmes manières d'encoder. Certains modèles nécessitent des données Label encodé, pour les autres nous avons préféré utiliser un Binary encoder plutôt qu'un One Hot encoding qui aurait généré trop de colonnes.

## Fenêtres glissantes (sliding windows)

Pour pouvoir utiliser ces données dans nos différents modèles, nous avons créé des fenêtres glissantes pour segmenter les séries temporelles en sous-ensembles. Nous avons créé des fenêtres 2D, avec une extension en 3D pour le modèle séquentiel RNN.

### Création des fenêtres

Afin d'optimiser le stockage et l'utilisation des données pour les fenêtres 2D, avant de créer les fenêtres nous avons identifié les colonnes dont les valeurs restent constantes sur tous les timesteps d'une fenêtre donnée, et nous les avons retirées pour éviter de dupliquer des informations.
Une fois les fenêtres glissantes créées, nous les avons aplaties (2D) pour pouvoir les utiliser dans nos modèles de régressions et arbres de décisions.

Ensuite, nous avons réintégré les informations constantes, extraites précédemment, dans une seule colonne, afin de réduire la redondance tout en conservant l'intégrité des données.

Enfin, nous avons créé des fenêtres spécifiques pour chaque batterie, sans chevauchement. Dans les cas où, en raison des paramètres de taille de fenêtre (window_size) et de pas de décalage (stride), les dernières données d'une batterie n'étaient pas capturées, nous avons ajouté une fenêtre supplémentaire. Cette fenêtre a été incluse uniquement si elle contenait au moins cinq cycles utilisables, afin de garantir que toutes les batteries aient une représentation complète dans les données.

Ainsi, l'objectif est de développer un modèle prédictif performant capable d'anticiper l'évolution de la courbe de SOH des batteries en fonction des données des cycles

précédents. Pour cela, nous avons pris la valeur de SOH associée au dernier cycle de chaque fenêtre comme cible de prédiction (y), permettant au modèle d'apprendre à prédire la santé future de la batterie en s'appuyant sur les valeurs observées dans les cycles précédents.

## Choix de la taille

Pour déterminer la taille optimale des fenêtres et le stride, nous avons cherché un équilibre entre :

- Capturer les dépendances temporelles significatives : Une fenêtre trop courte pourrait ne pas inclure suffisamment d'informations pour identifier les tendances et variations importantes.
- Préserver un volume suffisant de données pour l'entraînement : Une fenêtre trop longue réduit le nombre d'échantillons disponibles, limitant ainsi la diversité et la robustesse des données pour les modèles.
- Impact du stride : Le stride, qui détermine le décalage entre deux fenêtres successives, joue également un rôle clé. Un stride trop grand pourrait entraîner une perte d'information en ne capturant pas toutes les variations pertinentes, tandis qu'un stride trop petit augmenterait considérablement le nombre de fenêtres, ce qui peut rendre le modèle plus coûteux en termes de calculs et de temps d'entraînement. Il est donc important de choisir un stride qui équilibre la couverture temporelle et l'efficacité du modèle.

Nous avons généré plusieurs jeux de données en utilisant différentes tailles de fenêtres et de strides, puis nous les avons enregistrés au format pickle. Cela nous a permis de les réutiliser dans le notebook `2_rf_search_window.ipynb`, où nous avons déterminé la combinaison optimale de taille de fenêtre et de stride. Nous aborderons cette partie dans la prochaine section.

Une fois la combinaison optimale déterminée, nous avons généré des fenêtres glissantes en 3D. Ces fenêtres utilisent les mêmes découpages que celles en 2D, mais conservent la structure temporelle pour exploiter les capacités des modèles RNN à capturer des dépendances séquentielles.

# Recherche de la configuration optimale pour les fenêtres glissantes

Afin de déterminer la taille optimale des fenêtres et du stride, nous avons procédé de manière expérimentale dans le notebook `2_rf_search_window.ipynb`. Nous avons testé différentes tailles de fenêtres en utilisant un modèle **Random Forest** pour évaluer l'impact de ces configurations sur les performances. L'objectif était de trouver un compromis où les résultats sont améliorés, sans que le temps de calcul devienne trop long.

Nous avons observé qu'un stride trop petit entraînait des durées de traitement excessives, ce qui nous a conduits à éviter ces configurations. Nous avons laissé des outputs et des commentaires dans le notebook afin que vous puissiez constater par vous-même les résultats obtenus. Ce principe d'expérimentation et de transparence a été suivi dans tous les autres notebooks.

Nous avons effectué un aller-retour entre le notebook de préparation des données `1_preparation.ipynb` et celui de recherche des meilleurs paramètres pour affiner progressivement les configurations, un processus similaire à un **GridSearch**.

Les tests ont révélé que la combinaison **20** pour la taille de la fenêtre et **5** pour le stride offrait le meilleur compromis : une amélioration notable des performances tout en maintenant un temps de calcul raisonnable. Nous avons choisi ces valeurs comme référence pour tester les modèles.

Observation :
Les fenêtres plus longues ne semblent pas améliorer la performance, ce qui suggère que les variations de SOH sont principalement influencées par des dépendances à court terme. En effet, au-delà d'un certain seuil, la taille de la fenêtre n'apporte pas d'information supplémentaire significative, et les modèles sont capables d'anticiper l'évolution du SOH avec des fenêtres plus courtes.

# Extraction d’un sous-ensemble pour l’explicabilité

Dans la préparation des données, nous avons sélectionné un sous-ensemble spécifique pour tester les techniques d'explicabilité sur différents modèles. Bien que cette étape soit déjà en partie mentionnée dans le processus de préparation des données, il était nécessaire de définir un sous-ensemble de fenêtres et de batteries pour garantir une analyse ciblée.

Nous avons procédé comme suit :

- **Sélection aléatoire de 5 batteries** parmi chaque batch de données.
- **Sélection de 5 fenêtres** pour chaque batterie, assurant une diversité temporelle en capturant des moments différents de l’évolution du SOH.
- Nous avons vérifié les indices de ces fenêtres pour éviter tout chevauchement, afin de garantir que chaque fenêtre correspondait à un échantillon unique dans le temps.

# Modèles développés

Pour cette étude, nous avons mis en place trois catégories de modèles prédictifs afin de comparer leurs performances et d'évaluer leur efficacité sur les données :

## 1. Modèles basés sur des réseaux de neurones récurrents (RNN)

Trois types de réseaux ont été implémentés :

- **CNN 1D** :
  - Couches convolutives : 3 couches `Conv1D` avec 64, 128 et 256 filtres respectivement, une taille de filtre de 3, et une activation `relu`.
  - Pooling : 2 couches `MaxPooling1D` pour réduire la dimensionnalité.
  - Couches fully connected : une couche `Flatten`, suivie de deux couches `Dense` avec 128 et 64 neurones respectivement, et une activation `relu`.
  - Dropout : une couche de régularisation avec un taux de 50 % pour limiter l'overfitting.
- **LSTM** :
  - Une seule couche `LSTM` avec 64 unités et une activation `tanh`, sans retour de séquences (`return_sequences=False`).
- **RNN simple** :
  - Une seule couche `SimpleRNN` avec 64 unités et une activation `tanh`, également sans retour de séquences.

## 2. Modèles basés sur des arbres de décision

Deux algorithmes principaux ont été utilisés :

- **Random Forest** : Un modèle d'ensemble basé sur des arbres de décision, réputé pour sa robustesse mais plus coûteux à entraîner.
- **XGBoost** : Un modèle d'ensemble optimisé par gradient boosting, reconnu pour sa rapidité et ses performances.

## 3. Modèles de régression

Quatre variantes ont été testées :

- **Régression linéaire simple** : Méthode de base pour la prédiction.
- **Ridge Regression** : Une version régularisée avec une pénalité L2, ajustable via l'hyperparamètre `alpha`.
- **Lasso Regression** : Une régularisation L1 pour effectuer une sélection de features.
- **Elastic Net Regression** : Une combinaison des pénalités L1 et L2, contrôlée par les hyperparamètres `alpha` et `l1_ratio`.

Ici, nous avons sélectionné le meilleur modèle parmi ces quatre variantes en utilisant une recherche par grille (*grid search*). Les résultats finaux et les graphiques ne présentent que les performances de la **Ridge Regression**, identifiée comme la version optimale pour ce jeu de données.

Parmi ces modèles, les réseaux de neurones récurrents (notamment LSTM et SimpleRNN) ainsi que XGBoost et les modèles de régression ont offert une rapidité d'entraînement appréciable, contrairement au modèle Random Forest, qui s'est avéré plus coûteux en termes de temps de calcul.

# Optimisation du code et des hyperparamètres pour améliorer les performances

Pour l'optimisation du code et l'amélioration des performances, nous avons principalement utilisé une recherche par grille (**grid search**) pour affiner les hyperparamètres des modèles suivants :

- **Random Forest (RF)**
- **XGBoost**
- Tous les modèles de régression linéaire (régression classique, Lasso, Ridge, Elastic Net)

Ces optimisations ont été centralisées dans un seul notebook pour les modèles de régression, où nous avons retenu le modèle le plus performant avec ses meilleurs hyperparamètres.

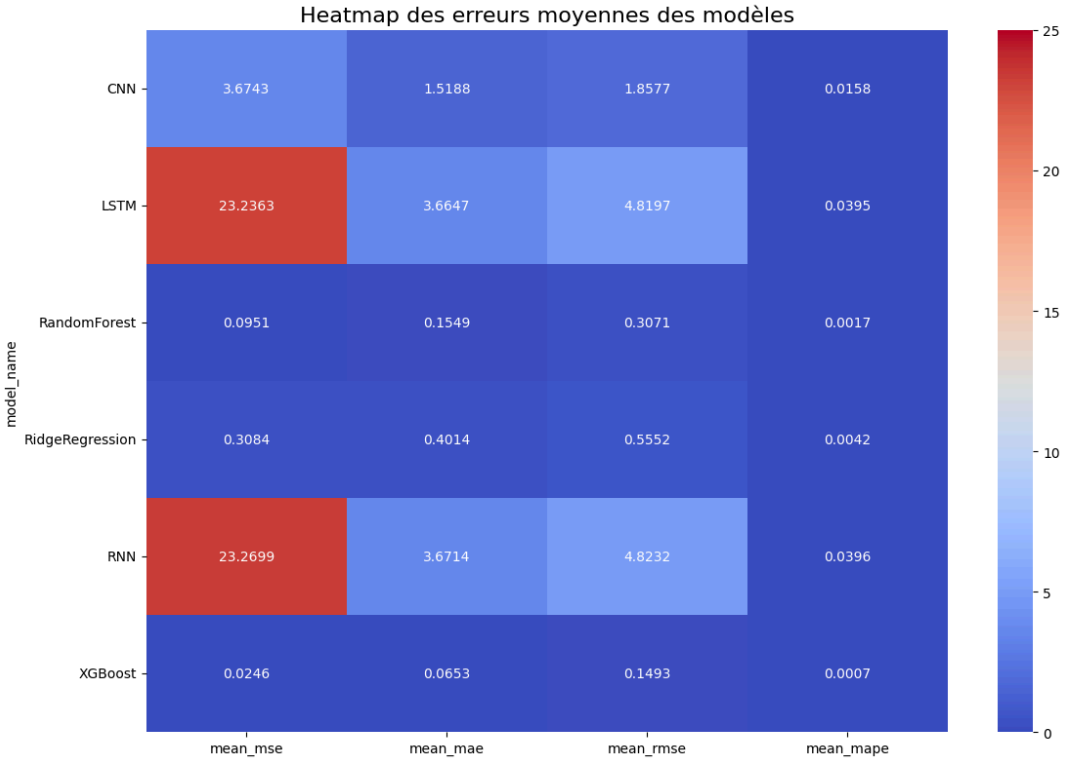
Nous n'avons pas effectué de **grid search** pour les modèles de type RNN (RNN, LSTM, CNN 1D), car leur entraînement est plus coûteux en temps, et la validation croisée aurait rendu l'optimisation impraticable.

Compte tenu des contraintes de temps et de ressources, nous avons restreint le nombre de combinaisons d'hyperparamètres testées. Par exemple, pour **Random Forest**, le temps d'entraînement étant significatif, il n'était pas réaliste d'explorer plusieurs milliers de combinaisons, d'autant plus que chaque combinaison devait être évaluée sur 5 plis pour la validation croisée.

Cette approche nous a permis de maximiser l'efficacité tout en tenant compte des limitations pratiques liées à l'entraînement de modèles complexes.

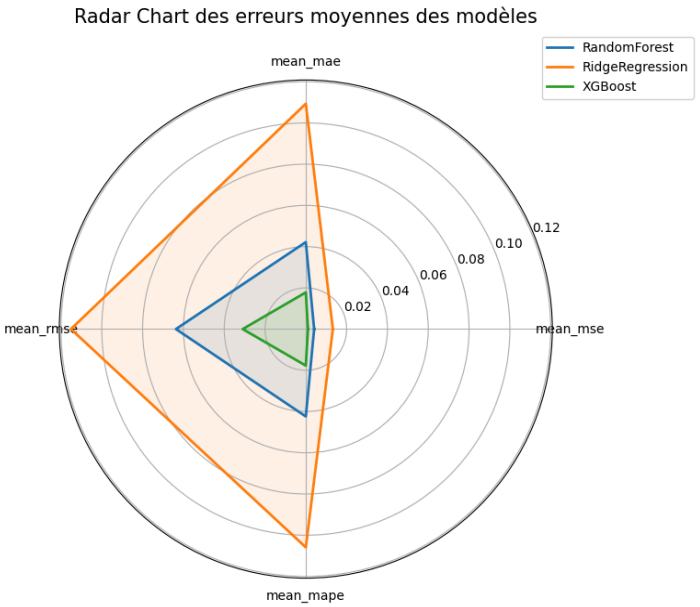
# Comparaison des modèles

Nous avons évalué les performances moyennes de nos six modèles (CNN, LSTM, RandomForest, RidgeRegression, RNN et XGBoost) en utilisant les erreurs moyennes obtenues à partir d'une validation croisée sur 5 folds. Les 4 métriques que nous avons utilisées sont la MSE, la MAE, la RMSE et la MAPE.



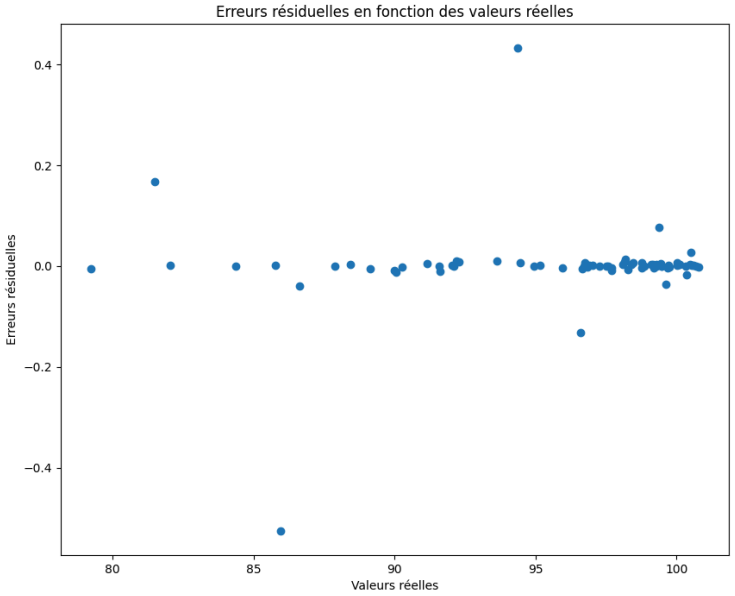

Les résultats obtenus, illustrés dans la heatmap, montrent que les modèles RandomForest, RidgeRegression et XGBoost ont des performances bien meilleures que les autres modèles avec des erreurs moyennes très faibles sur l'ensemble des métriques. À l'inverse, les modèles LSTM et RNN se révèlent peu performants, avec certaines erreurs significativement élevées. Et le modèle CNN, bien qu'il présente de meilleures performances que LSTM et RNN, reste légèrement moins performant que les trois meilleurs modèles.

Nous avons ensuite cherché à départager les trois meilleurs modèles. Pour cela, nous avons utilisé un radar chart en excluant les modèles RNN, CNN et LSTM, afin de comparer leurs performances sur les quatre métriques principales.




Les résultats, représentés sur le radar chart, montrent que XGBoost se distingue comme le meilleur modèle avec des erreurs extrêmement faibles sur toutes les métriques, suivi par Random Forest, puis Ridge Regression.

Pour XGBoost, que nous avons identifié comme le meilleur modèle, nous avons analysé les erreurs de prédiction en utilisant le sous-ensemble spécifique des données d'explicabilité (X_explicabilité, explications dans la section "Extraction d'un sous-ensemble pour l'explicabilité"). Les prédictions générées par le modèle XGBoost optimal (y_pred) ont été comparées aux valeurs réelles (y_explicabilité) correspondantes, et les erreurs résiduelles ont été calculées comme la différence entre ces prédictions et les valeurs observées.




Les résultats montrent que la majorité des prédictions sont très précises, avec des erreurs majoritairement proches de zéro. Seuls cinq points s'écartent légèrement davantage. Cela confirme que XGBoost est très performant sur ce sous-ensemble de données.

# Explicabilité

Dans cette section, nous abordons l'explicabilité du modèle XGBoost, identifié comme le modèle le plus performant dans notre analyse précédente. Par manque de temps, toutes les approches d'explicabilité n'ont pas pu être pleinement implémentées ou testées, ce qui sera détaillé en conclusion. Nous nous concentrons ici uniquement sur l'explicabilité appliquée à XGBoost.

Le modèle XGBoost utilisé dans cette analyse a été entraîné par validation croisée, comme décrit dans le notebook 5_XGBoost_training. Nous avons sélectionné celui correspondant au meilleur fold, en raison de la similitude des performances moyennes entre les folds et du grand volume de données disponibles, ce qui justifie cette approche.

L'analyse a été réalisée dans le notebook **8_xgboost_XAI**.

Après avoir chargé le modèle préalablement entraîné, nous avons extrait les features les plus importantes en utilisant l'attribut `feature_importances_`.
D'après nos recherches, la méthode `feature_importances_` dans XGBoost calcule l'importance des caractéristiques en se basant sur leur fréquence d'utilisation dans les splits des arbres de décision du modèle. Plus précisément, elle compte combien de fois une caractéristique est utilisée pour diviser les données, pondérant cette fréquence par l'amélioration qu'elle apporte à la métrique objective (comme le gain ou la réduction de l'erreur). Cette mesure agrégée sur tous les arbres donne une indication de l'influence globale de chaque caractéristique sur les prédictions du modèle.
L'extraction a produit le classement suivant :

```
Feature ranking:
1. QC_20 (0.663472056388855)
2. pulse_width (0.1249132975935936)
3. QC_19 (0.1107892319560051)
4. after_discharging_rest (0.03708828613162041)
5. QC_18 (0.023017795756459236)
6. cycle_to_x% (0.006016927305608988)
7. chargetime_20 (0.004682956263422966)
8. cycle_normalized_1 (0.0035937586799263954)
9. chargetime_19 (0.0017507510492578149)
10. C2 (0.0017422523815184832)
```

Les trois premières features, **QC_20**, **pulse_width**, et **QC_19**, se démarquent nettement par leur importance. Elles représentent à elles seules plus de 89 % de la pondération totale attribuée par le modèle. Ces features semblent jouer un rôle clé dans les prédictions de XGBoost, ce qui justifie leur analyse approfondie. On peut supposer que :

- **QC_20** représente la capacité de charge au moment du SOH (State of Health) que nous cherchons à prédire, calculée sur une fenêtre de temps de 20 cycles.

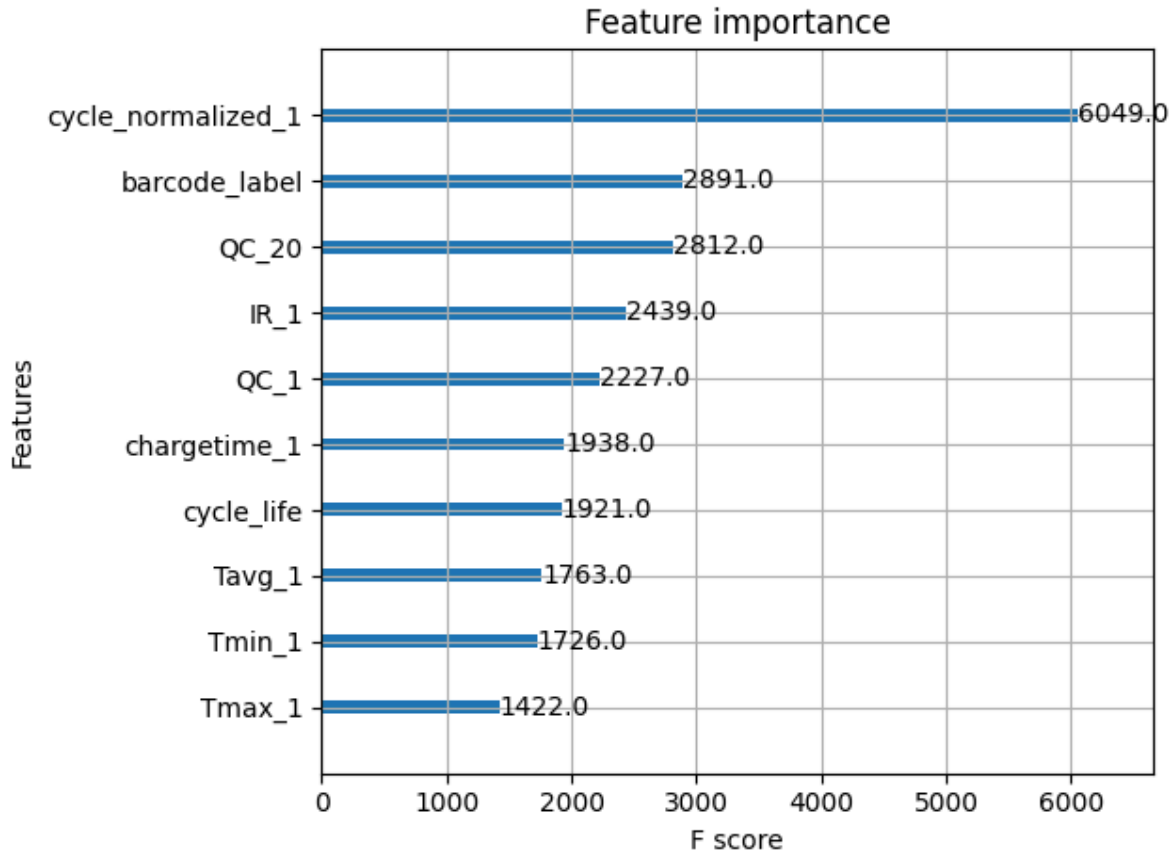
- **pulse_width** reflète des dynamiques temporelles ou des mesures spécifiques associées à la batterie.
- **QC_19** correspond à la capacité de charge au pas de temps précédent (par rapport à QC_20), offrant un complément d'information pour modéliser l'évolution des performances de la batterie.

## Importance des features par critères (weight, gain, cover)

Nous avons calculé l'importance des features sur la performance prédictive données par le F-score en utilisant les trois métriques proposées par XGBoost : **weight**, **gain** et **cover**. Voici une analyse des résultats obtenus :

**1. Weight**

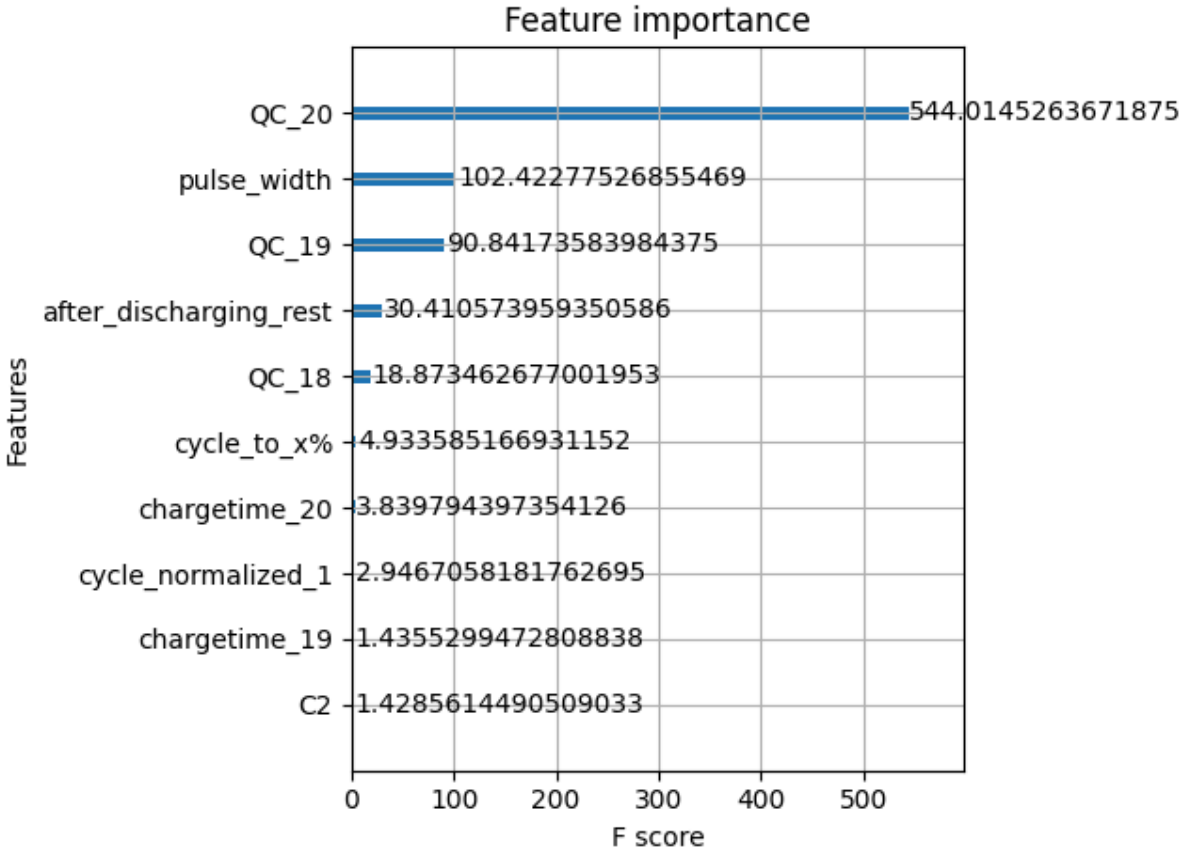
Le critère *weight* correspond au nombre de fois qu'une feature est utilisée pour scinder les données dans l'ensemble des arbres. Les features les plus importantes selon ce critère sont:




Ces résultats montrent une répartition des splits reliant des cycles, des caractéristiques thermiques, et des métriques globales de performance.

**2. Gain**

Le critère *gain* mesure l'amélioration moyenne du modèle apportée par une feature lorsqu'elle est utilisée dans les arbres. Les plus importantes selon ce critère sont :

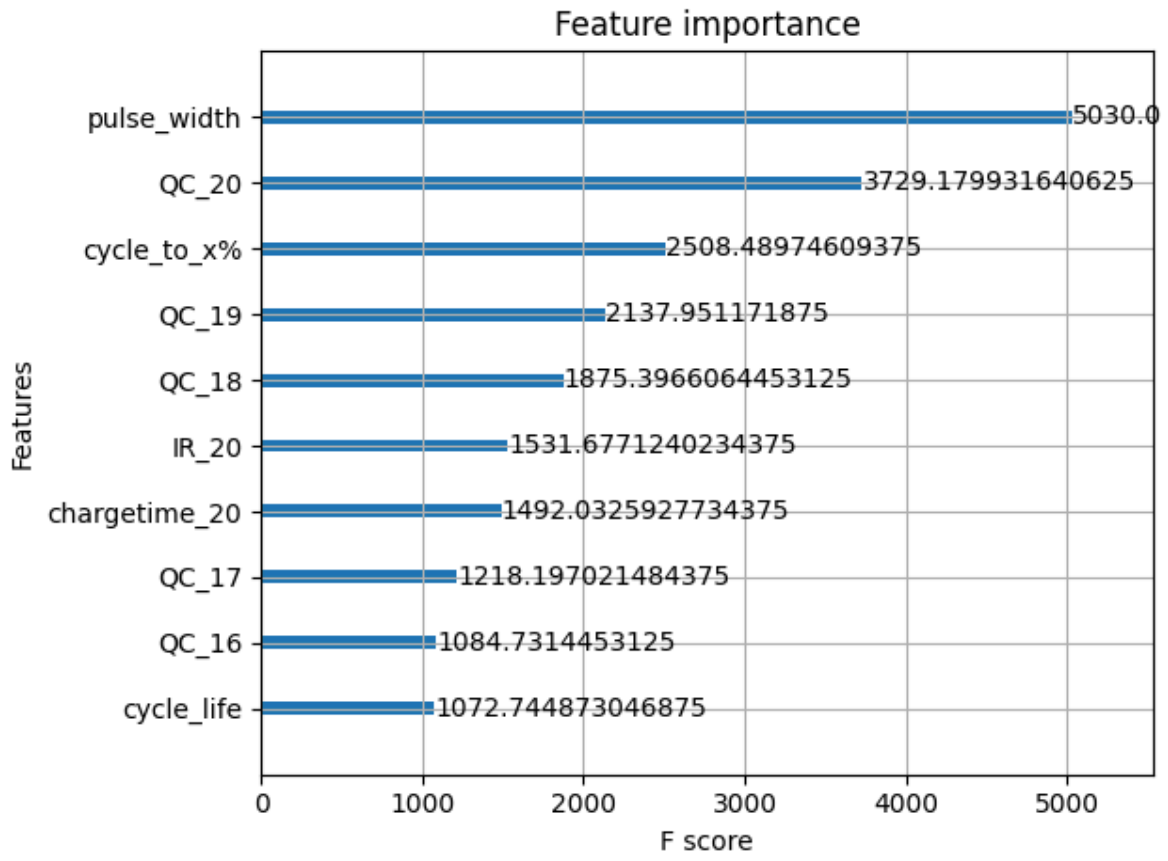



- **QC_20** (très nettement dominante)
- **pulse_width**
- **QC_19**
- **after_discharging_rest**
- **QC_18**

Cela indique que ces features capturent des informations clés qui influencent fortement la qualité des prédictions.

**3. Cover**

Le critère *cover* évalue la couverture moyenne (proportion d'échantillons) des features sur l'ensemble des arbres. Les plus importantes selon ce critère sont :




- **pulse_width**
- **QC_20**
- **cycle_to_x%**
- **QC_19**
- **QC_18**

On remarque également l'apparition dans le top 10 des features **QC_17** et **QC_16**, qui semblent également pertinentes bien que leur impact soit moindre sur les autres critères.

Il est aussi intéressant de noter l'apparition de chargetime au dernier timestep (à la capture de la valeur de SOH actuelle)

## Synthèse

- **QC_20** et **QC_19** apparaissent régulièrement dans le top des trois métriques, confirmant leur importance cruciale.
- La dominance de **QC_20** dans le gain suggère qu'elle fournit des informations déterminantes pour le modèle.

- Les métriques thermiques (**Tavg_1**, **Tmin_1**, **Tmax_1**) au cycle 1 et les cycles associés (**cycle_normalized_1**, **cycle_to_x%**) montrent leur utilité pour caractériser la performance de la batterie à divers niveaux.
- Les **QC** (capacité de charge) des derniers timesteps et les **températures** mesurées au premier temps de la fenêtre sont importants pour évaluer la performance de la batterie.

## Visualisation des arbres de décision

Pour approfondir l'analyse du modèle XGBoost, nous avons exploré sa structure interne en visualisant les arbres de décision. L'objectif était d'identifier des schémas ou des logiques spécifiques capturées par le modèle.

**Visualisation du "meilleur arbre"**

Nous avons utilisé la fonction `plot_tree` pour représenter l'arbre ayant contribué le plus significativement à la performance finale du modèle.

L'arbre visualisé représente le **"meilleur arbre"**, c'est-à-dire celui correspondant à la dernière itération ayant donné la meilleure performance durant l'entraînement.

Cette visualisation permet de :

1. **Comprendre les règles utilisées** : Les splits basés sur les features montrent les décisions prises par le modèle pour partitionner les données.
2. **Identifier les contributions majeures** : Les splits les plus proches de la racine indiquent les features les plus importantes dans cet arbre spécifique.

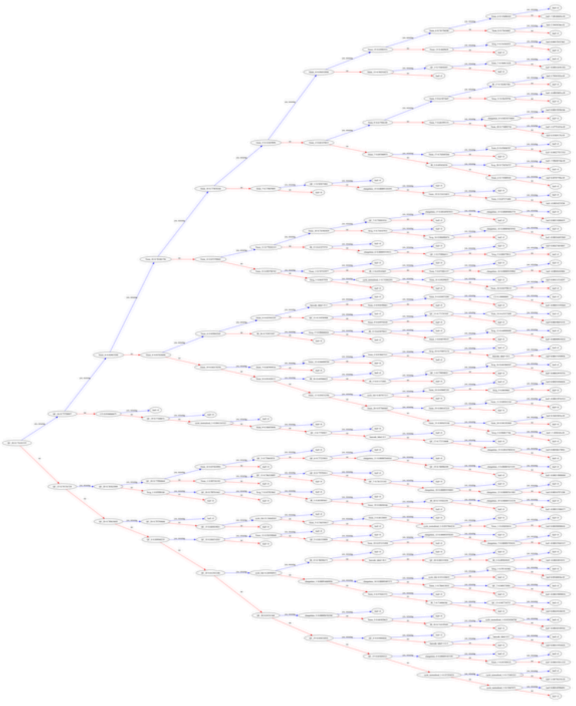
**Résultat obtenu**

Le "meilleur arbre" visualisé, correspondant à la dernière itération ayant produit la meilleure performance, nous révèle les points suivants :

- **Split initial :** L'arbre commence par un split basé sur **QC_20**, ce qui montre l'importance de cette caractéristique dès le début pour la prise de décision.
- **Splits suivants :** Après ce premier split, l'arbre se divise davantage en utilisant des caractéristiques comme **QC_15** ou **QC_16**, indiquant leur rôle secondaire mais significatif dans les décisions du modèle.
- **Temporalité des caractéristiques :** On observe une forte utilisation de différentes valeurs de **QC** à des **timesteps** plus avancés, principalement autour du **timestep 20**, ce qui suggère que ces valeurs sont particulièrement informatives dans les phases plus tardives.
- **Températures :** Des données liées aux températures (**max**, **avg**, **min**) apparaissent fréquemment dans les splits, soulignant leur importance pour prédire le comportement du modèle.
- **Charge time :** La caractéristique **chargetime** est également utilisée à divers **timesteps**, mettant en évidence son impact dans différentes phases du cycle.
- **IR :** Quelques features **IR_13** et **IR_20** apparaissent également, suggérant une influence moindre mais toujours présente.

**Conclusion**

Cette visualisation confirme que les caractéristiques telles que **QC_20 (et les QC capacité de charge à différents timesteps)** les températures et **chargetime** à des moments spécifiques sont déterminantes pour les décisions prises par le modèle XGBoost. Les **timesteps avancés**, en particulier autour du **timestep 20**, semblent jouer un rôle majeur dans les partitions effectuées par l'arbre.



Pour plus de détails, veuillez vous référer à l'arbre dans le dépôt, disponible en haute résolution dans : **Images/XGBoost_best_tree.png**.

## Analyse de l’explicabilité avec SHAP

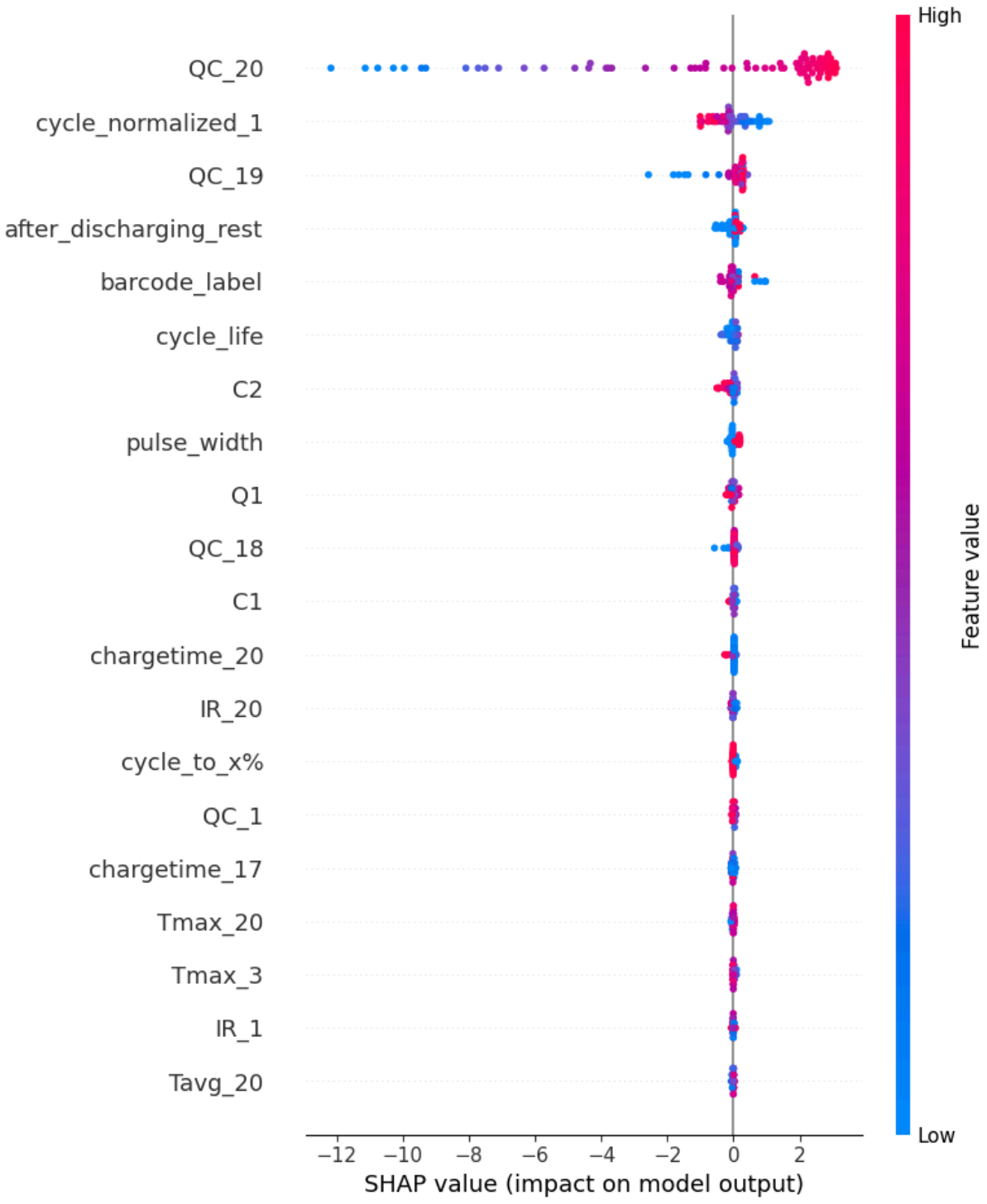
Pour approfondir l’interprétabilité du modèle XGBoost, nous avons utilisé **SHAP (SHapley Additive exPlanations)**, une méthode puissante permettant de quantifier l’impact de chaque feature sur les prédictions du modèle.

Les valeurs de Shapley sont calculées à l’origine pour une instance spécifique. Elles indiquent l’importance positive ou négative de chaque attribut dans la prédiction réalisée pour cette instance particulière, ce qui en fait une méthode d’**explication locale**. Pour obtenir une **explication globale**, représentative de l’ensemble des prédictions, on calcule la moyenne des valeurs de Shapley pour chaque attribut sur un ensemble d’instances. Cette approche permet d’identifier les facteurs ayant, en moyenne, le plus ou le moins d’impact sur les prédictions globales du modèle.

Dans cette analyse, nous avons utilisé ces deux perspectives pour mieux comprendre le comportement de notre modèle et la contribution de chaque feature aux résultats obtenus.

**Visualisation des contributions des features**

1. **Résumé global des impacts des features**



Fonctionnement du graphique Beeswarm SHAP

source : https://medium.com/@anshulgoel991/model-exploitability-using-shap-shapley-additive-explanations-and-lime-local-interpretable-cb4f5594fc1a

- Points de données :
  - Chaque point représente une instance de données. La couleur des points montre la valeur de la caractéristique pour cette instance :
    - Rouge : Valeur élevée de la caractéristique.
    - Bleu : Valeur faible de la caractéristique.
- Étendue de l'importance des caractéristiques (Feature Importance Spread) :
  - La dispersion des points le long de l'axe horizontal indique la plage et la distribution des valeurs SHAP pour chaque caractéristique. Une large dispersion montre que l'importance de cette caractéristique varie beaucoup selon les instances.
- Impact relatif (Relative Impact) :
  - La position des points sur l'axe horizontal montre l'impact de chaque caractéristique sur les prédictions. Les points plus à droite (valeurs SHAP élevées) indiquent des contributions positives importantes, tandis que les points plus à gauche (valeurs SHAP faibles) suggèrent des contributions négatives.
- Consistance de l'influence des caractéristiques (Consistency of Feature Influence) :
  - Si les points d'une caractéristique sont regroupés, cela signifie que son impact sur les prédictions est stable. Des distributions plus dispersées peuvent indiquer une influence moins stable.
- Importance comparative (Comparative Importance) :
  - En comparant la dispersion des points entre les différentes caractéristiques, on peut évaluer leur importance relative. Une caractéristique avec des points plus dispersés ou décalés a généralement plus d'impact.
- Axe vertical (Y-axis) :
  - Les caractéristiques sont listées sur l'axe vertical, triées par ordre d'importance globale. Les caractéristiques situées en haut ont le plus grand impact sur la prédiction.

En résumé :

Pour les combinaisons de couleurs et de position :

- Rouge à gauche et bleu à droite :
  - Cela indique que pour les instances avec une valeur élevée de la caractéristique (rouge), l'impact est négatif(à gauche) et pour les instances avec une valeur faible (bleu), l'impact est positif (à droite). Cela pourrait suggérer que la caractéristique a un impact inverse selon qu'elle soit élevée ou faible (peut-être une relation non linéaire ou une influence opposée).
- Bleu à gauche et rouge à droite :
  - Cela signifie que pour les instances avec une valeur faible de la caractéristique (bleu), l'impact est négatif (à gauche), et pour les instances avec une valeur élevée (rouge), l'impact est positif (à droite). Cela pourrait

suggérer que la caractéristique a un effet inverse entre les faibles et les fortes valeurs.

Analyse des caractéristiques

QC_20 :

- Bleu à gauche (très à gauche) : Les valeurs faibles de QC_20 semblent avoir un impact négatif sur la prédiction, ce qui peut indiquer qu'une faible valeur de cette caractéristique tend à réduire la performance ou à conduire à une prédiction plus basse.
- Rouge à droite (moins loin à droite, entre -12 et 2) : Les valeurs élevées de QC_20 ont un impact positif, mais moins marqué. Cela suggère que QC_20 contribue positivement à la prédiction lorsqu'elle a une valeur élevée, mais l'effet n'est pas aussi fort que pour les faibles valeurs.
- Résumé : QC_20 semble avoir une relation inverse avec la prédiction. Les faibles valeurs de QC_20 ont un impact négatif, tandis que les valeurs élevées ont un impact positif, mais de manière moins prononcée. La dispersion des points montre une variabilité significative de l'impact de cette caractéristique.

cycle_normalized_1 :

- Rouge à gauche : Les valeurs élevées de cycle_normalized_1 semblent avoir un impact négatif sur la prédiction.
- Bleu à droite : Les valeurs faibles de cycle_normalized_1 ont un impact positif sur la prédiction.
- Petite dispersion : La faible dispersion des points suggère que l'impact de cycle_normalized_1 est assez cohérent. Il semble que les valeurs faibles entraînent systématiquement des prédictions plus élevées, et les valeurs élevées entraînent des prédictions plus basses, mais l'effet est relativement stable.
- Résumé : cycle_normalized_1 a une influence inverse sur les prédictions : les valeurs élevées entraînent des prédictions faibles, et les valeurs faibles entraînent des prédictions élevées, avec une relation assez stable (moins de variation dans les données).

QC_19 :

- Bleu à gauche : Comme QC_20, les valeurs faibles de QC_19 semblent également avoir un impact négatif sur la prédiction.
- Gros point bleu et rouge à gauche : Cela peut indiquer que, même si la majorité des instances avec des valeurs faibles de QC_19 ont un impact négatif, il y a quelques instances (représentées par le gros point) où les valeurs faibles de QC_19 n'ont pas un impact aussi négatif, ou même un impact plus fort dans certains cas.
    - Plus éparse que cycle_normalized_1 : Cela suggère qu'il y a plus de variabilité dans l'impact de QC_19, ce qui pourrait signifier que cette caractéristique a une influence moins stable ou plus complexe par rapport à cycle_normalized_1.
- Résumé : QC_19 semble aussi avoir une relation inverse avec la prédiction, mais avec plus de variabilité. Les valeurs faibles de QC_19 ont tendance à avoir un impact négatif, mais certaines instances montrent des effets différents, ce qui entraîne une plus grande dispersion des points.

## Interprétation globale des caractéristiques de SOH à partir du graphique Beeswarm SHAP

Pour rappel, le SOH d'une batterie reflète sa capacité à stocker et fournir de l'énergie par rapport à sa capacité initiale. L'objectif ici est de prédire comment la batterie se comporte au fil du temps, en fonction de plusieurs paramètres.

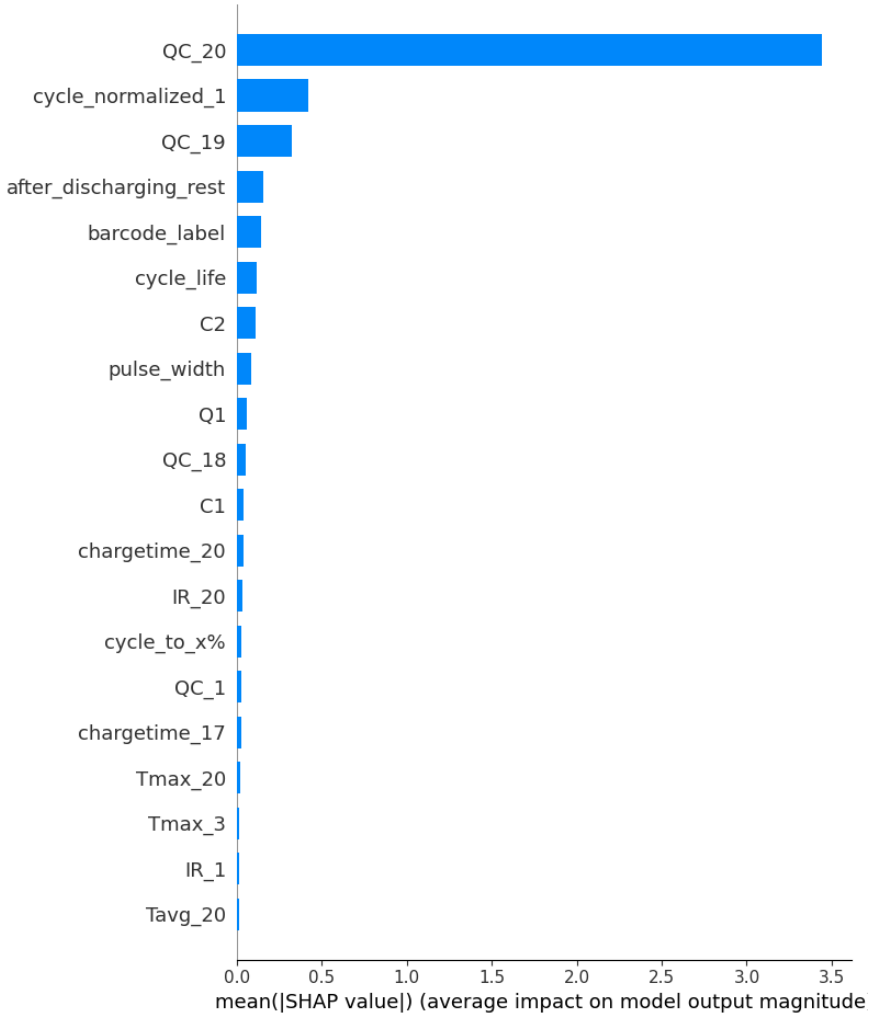
1. **Caractéristiques liées aux derniers timesteps**: Les derniers timesteps (tels que QC_20, QC_19, QC_18, IR_20, Tavg_20) semblent avoir un impact majeur sur la prédiction du SOH (State of Health) de la batterie. Cela semble logique, car l'état de santé d'une batterie est directement lié à ses performances récentes, comme sa capacité de charge, sa résistance interne, et sa température.
    - QC_20, QC_19, et QC_18 : Ces caractéristiques mesurent l'état de charge de la batterie à différents moments et jouent un rôle clé dans l'évaluation du SOH. Leur positionnement sur le graphique montre qu'elles sont parmi les caractéristiques les plus influentes pour prédire le SOH. Cela reflète l'idée que les dernières performances de la batterie (comme la capacité restante) sont cruciales pour prédire son état de santé.
    - IR_20 : La résistance interne est également très significative. Une résistance élevée peut indiquer une dégradation de la batterie, réduisant ainsi sa performance et affectant son SOH. Cela est particulièrement pertinent car une batterie vieillissante présente généralement une résistance interne plus élevée.
    - Tavg_20 : La température moyenne de la batterie est un autre facteur essentiel. Des températures plus élevées accélèrent généralement la dégradation des cellules de la batterie, ce qui peut nuire à son SOH au fil du temps.
2. **Caractéristiques liées aux premiers timesteps**:
    - cycle_normalized_1 : Indique combien de cycles de vie la batterie a déjà endurés. Cela permet d'évaluer l'usure générale de la batterie, ce qui influence son état de santé (SOH). C'est plutôt logique de retrouver cela parmi les caractéristiques les plus influentes.
    - IR_1 et QC_1 : Mesurent l'état initial de la batterie dans les premiers cycles. Ces caractéristiques ont un impact à distance sur le SOH, même après une vingtaine de cycles, car elles fournissent des informations sur l'efficacité de la batterie au début de son utilisation.
3. **Caractéristiques liées à la manière dont la batterie est chargée** :
    - C1 et C2 : Ces paramètres mesurent le taux de charge (courant) de la batterie, et leur impact sur le SOH est significatif. Des charges trop rapides ou trop lentes peuvent affecter la performance de la batterie à long terme. C1et C2 indiquent comment le courant est distribué durant le processus de charge.
    - Q1 : Ce paramètre correspond au seuil de capacité atteint (%). Il est essentiel car il montre si la batterie atteint des niveaux de charge optimaux ou si des surcharges fréquentes risquent d'altérer sa longévité.
    - **Charge policy** : Ces caractéristiques sont liées à la stratégie de charge adoptée pour la batterie (par exemple, le courant et les seuils de capacité). Une gestion inefficace de la charge peut accélérer l'usure de la batterie et affecter son SOH.

4. **Caractéristiques liées au batch et à des facteurs externes (after_discharging_rest, pulse_width) :**
    - Les différences entre les lots de batteries semblent affecter la dégradation. Les caractéristiques liées au batch, telles que after_discharging_rest et pulse_width, montrent l'influence de la manière dont chaque lot de batteries est expérimenté, ce qui peut entraîner des variations dans la performance et la dégradation des cellules. Ces facteurs externes peuvent donc jouer un rôle important dans l'état de santé de la batterie.

**Prise de recul** : Bien que des caractéristiques telles que QC_20, cycle_normalized_1, QC_19, et after_discharging_rest aient un impact direct sur le SOH, il est important de noter que sur le graphique SHAP, toutes les valeurs sont réajustées autour de 0 sur l'axe des Y. Cela signifie que, même si ces caractéristiques apparaissent comme influentes, leur impact est mesuré par rapport à une référence moyenne. En revanche, les autres caractéristiques sont plus regroupées, ce qui suggère un impact moins marqué et ne permet pas de tirer des conclusions très poussées.

**2. Résumé bar chart des contributions moyennes**

Pour visualiser les contributions moyennes de chaque feature sur l'ensemble des données, nous avons généré un graphique de type *bar plot.*




Interprétation du graphique Bar SHAP :

Le graphique bar SHAP montre l'importance moyenne des caractéristiques sur la prédiction du modèle.

- Axe horizontal : Représente la valeur moyenne absolue des valeurs SHAP. Plus la barre est longue, plus l'impact de la caractéristique est important.
- Axe vertical : Liste les caractéristiques triées par ordre décroissant d'importance.

Ce graphique confirme les informations précédentes :

- QC_20 : La caractéristique la plus importante, avec un impact majeur, bien supérieur aux autres.
- cycle_normalized_1 : La deuxième caractéristique la plus influente.
- QC_19 : La troisième caractéristique la plus influente.

## Explication locale basée sur une instance prise aléatoirement dans chaque batch

Pour la suite de l'analyse, nous avons extrait aléatoirement une instance de chaque batch, pour revoir la sélection des instances pour l'explicabilité, veuillez vous référer à la section "Extraction d'un sous-ensemble pour l'explicabilité". Cette méthode nous a permis d'examiner les force plots associés à ces instances. Ces visualisations offrent une perspective détaillée sur l'impact des caractéristiques sélectionnées sur les prédictions du modèle, pour une instance "représentative" de chaque batch.

Les interprétations des résultats sont basées sur les méthodologies expliquées dans les sources suivantes :

- Aquila Data - SHAP : mieux comprendre l'interprétation de modèles
- Aquila Data - Interprétabilité des modèles de machine learning
- Model Exploitability Using SHAP

### Fonctionnement du graphique Force Plot SHAP

Le modèle SHAP calcule les contributions de chaque "variable" ("feature" et "caractéristique" sont des mots plus adéquats) à la prédiction finale, exprimée comme une fonction linéaire des variables considérées. Trois propriétés fondamentales sous-tendent cette méthode pour garantir une explication correcte du modèle:

- Missingness : Les variables manquantes n'ont aucun impact sur la prédiction.
- Consistency : Augmenter l'impact d'une variable dans le modèle ne réduira jamais sa contribution (permet de comparer les importances des variables entre différents modèles).
- Local accuracy : La somme des contributions des variables égale la sortie du modèle pour une instance donnée.

La valeur de base représente la prédiction moyenne du modèle pour l'ensemble du dataset, tandis que la valeur de sortieest la prédiction individuelle pour un cas particulier. Les valeurs SHAP, proportionnelles à la taille des flèches sur le graphique, montrent comment chaque variable "pousse" la prédiction depuis la base vers la valeur finale.

**Contributions des caractéristiques (Feature Contributions)**

- Couleurs :
  - Rouge : Les caractéristiques en rouge ont un impact positif sur la prédiction, c'est-à-dire qu'elles contribuent à ce que la prédiction soit plus élevée que la valeur de base.
  - Bleu : Les caractéristiques en bleu ont un impact négatif sur la prédiction, c'est-à-dire qu'elles contribuent à ce que la prédiction soit plus basse que la valeur de base.
- Base Value :
  - La base value (valeur de base) est la valeur moyenne obtenue comme sortie pour cette classe. C'est la prédiction moyenne du modèle pour l'ensemble du dataset.
- Sens des flèches :
  - Les flèches pointant vers la droite indiquent des contributions positives, augmentant la prédiction.
  - Les flèches pointant vers la gauche indiquent des contributions négatives, diminuant la prédiction.

**Impact des caractéristiques (Feature Impact)**

- Longueur des barres :
    - La longueur de chaque barre indique la magnitude de l'impact de la caractéristique sur la prédiction. Plus la barre est longue, plus l'impact est élevé.

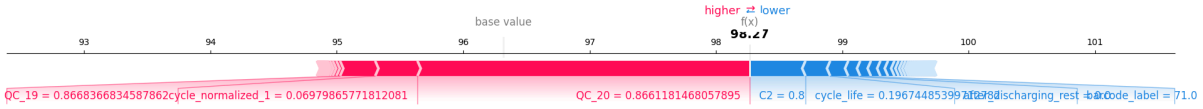
**Direction de l'influence (Direction of Influence)**

- Barres vers la droite : Les caractéristiques avec des barres pointant vers la droite poussent la sortie du modèle vers des valeurs plus élevées.
- Barres vers la gauche : Les caractéristiques avec des barres pointant vers la gauche poussent la sortie du modèle vers des valeurs plus basses.

**Décalage global de la prédiction (Overall Prediction Shift)**

- Distance par rapport à la Base Value : La distance de chaque barre par rapport à la base value montre comment la contribution de chaque caractéristique déplace la prédiction par rapport à la valeur moyenne.

Analyse du graphe de force SHAP pour une instance du batch 1




**Valeur de base (Base Value)**

La valeur de base est d'environ 96, représentant la prédiction moyenne du modèle pour l'ensemble du dataset. Les contributions des caractéristiques modifient cette valeur de base pour arriver à la prédiction finale. Dans ce cas précis, la prédiction finale est 98.27, ce qui indique que la prédiction est légèrement supérieure à la valeur de base. Cela signifie que les contributions positives (représentées par les couleurs rouges) ont un impact légèrement plus important que les contributions négatives (représentées par les couleurs bleues).

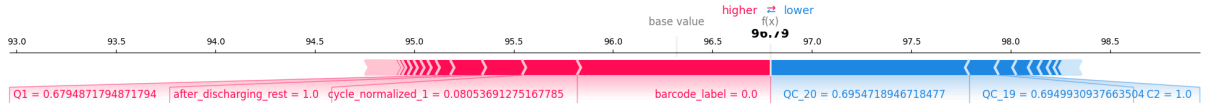
**Contributions des caractéristiques**

- QC_19 et cycle_normalized_1 : Ces caractéristiques, indiquées en rouge et pointant vers la droite, ont un impact positif sur la prédiction, augmentant la valeur par rapport à la base. Leur impact est modéré, ce qui est reflété par la longueur des barres.
- QC_20 : Cette caractéristique a également un impact positif, mais sa barre est beaucoup plus longue, indiquant un impact important. Cela confirme les résultats observés dans le graphique beeswarm.
- C2 : Cette feature apparaît en bleu, ce qui signifie qu'elle a un impact négatif sur la prédiction. Elle tend à faire diminuer la valeur de la prédiction, et son rôle est aussi important que celui de QC_19 et cycle_normalized_1.
- Cycle Life : Cette caractéristique, également en bleu, contribue à réduire la prédiction, comme le montre la longueur de la barre.

**Conclusion**

Les caractéristiques les plus influentes, comme observé dans le graphique beeswarm, sont QC_20, barcode, cycle life, QC_19, cycle_normalized_1, et after_discharging_rest. En particulier, QC_20 reste la feature la plus importante, suivie de C2, cycle_normalized_1, et QC_19.

Analyse du graphe de force SHAP pour une instance du batch 2




La valeur de base pour ce batch est identique à celle observée dans le batch 1, ce qui est logique. La prédiction pour cette instance spécifique du batch 2 est de 96.79, ce qui est plus proche de la valeur observée dans le batch 1.

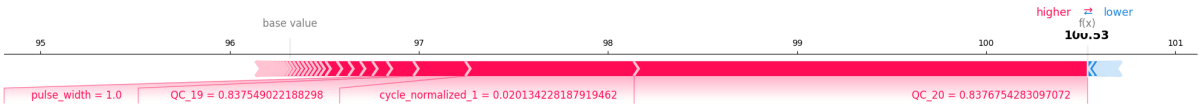
**Contributions des caractéristiques :**

- En rouge (ordre croissant d'importance) :
  - Q1
  - after_discharging_rest
  - cycle_normalized_1
  - Barcode label : Comme les données du batch 2 sont distinctes dans leur préparation, il n'est pas surprenant que le modèle identifie cette batterie différemment grâce à son label. Ce label semble jouer un rôle clé dans cette instance.
- En bleu (ordre croissant d'importance) :
  - C2
  - QC 19
  - QC 20

Il est à noter que barcode label et QC_20 ont une importance similaire, avec des longueurs de barre quasi identiques, ce qui montre un impact similaire sur la prédiction.

**Conclusion** :
Dans ce batch, on observe l'apparition de barcode label, Q1, et C2 comme caractéristiques significatives, en plus des features habituelles comme cycle_normalized_1, QC 20, et QC_19. Cela suggère un impact important des conditions spécifiques du batch 2, ce qui montre que les différences de conditions d'utilisation des batteries (données différentes entre chaque batch) a certainement une importance.

Analyse du graphe de force SHAP pour une instance du batch 3




La valeur de prédiction pour cette instance spécifique du batch 3 est de 100.53, ce qui est plus éloigné de la valeur de base de 96 observée dans les deux exemples précédents.

**Contributions des caractéristiques :**

- En rouge (ordre croissant d'importance) :
  - pulse_width
  - QC_19
  - cycle_normalized_1
  - QC_20 : Comme dans les autres analyses, QC_20 apparaît avec la flèche la plus longue, indiquant son impact majeur sur la prédiction.

**Conclusion** :
Le batch 3 montre une influence plus forte de pulse_width, suivie de QC_19 et cycle_normalized_1. Cependant, QC_20 reste, une fois de plus, la caractéristique la plus déterminante avec une flèche de contribution plus longue, soulignant son rôle clé dans la prédiction.

## Tentative d’explicabilité pour les modèles RNN

Dans le cadre de notre projet, nous avons exploré l’explicabilité des modèles de type RNN en utilisant la bibliothèque **TimeSHAP**, conçue pour traiter les données séquentielles et multidimensionnelles. Cette approche est particulièrement adaptée pour expliquer les prédictions sur des données temporelles ou 3D. Nous avons suivi la documentation officielle et les tutoriels disponibles pour implémenter cette méthode. Cependant, malgré ces efforts, nous avons rencontré des erreurs liées à des problèmes internes de la bibliothèque, notamment un **circular import** signalé par d’autres utilisateurs sur GitHub. Ces limitations techniques n’ont pas permis d’obtenir les résultats escomptés.

Vous pouvez retrouver ce que nous avons tenté d’implémenter dans le notebook `9_cnn_timeshap`. Ce même problème d'erreur est décrit dans l'**issue** suivante sur GitHub: https://github.com/feedzai/timeshap/issues/48.

De plus, nous avons suivi le tutoriel détaillé disponible ici : https://github.com/feedzai/timeshap/blob/main/notebooks/AReM/AReM_TF.ipynb pour mieux comprendre l'implémentation, mais malheureusement, les erreurs persistent.

## Tentative avec PyXAI

PyXAI est une bibliothèque d'explicabilité dédiée aux modèles à base d’arbres (arbre de décision, forêt aléatoire, arbre boosté). Elle permet d’offrir des explications rigoureuses pour des modèles comme **XGBoost**. Nous avons souhaité essayer cette bibliothèque car Zoé en avait entendu parler lors de sa licence à l'Université d'Artois.

Contrairement à SHAP ou LIME, PyXAI garantit que les explications locales qui sont calculées (en particulier les explications abductives) sont correctes par rapport aux modèles. *sources : sites officiels et GitHub*

D'après la documentation de PyXAI, il est possible d'intégrer un modèle XGBoost déjà entraîné directement dans l'outil pour fournir des explications sur les prédictions du modèle, comme expliqué dans leur guide sur les régressions BT. Il est possible de réapprendre dans PyXAI un modèle XGBoost mais nous n’avons pas trouvé comment fournir en entrée les valeurs des hyperparamètres déterminés par grid search précédemment. De ce fait, nous n’avons pas pu poursuivre l’utilisation de PyXAI, car le modèle aurait été moins performant.

Lors de nos tentatives d’intégration de PyXAI, nous avons rencontré une autre difficulté, cette fois-ci d'installation, principalement dues à des problèmes de compatibilité entre l’environnement virtuel (venv) et MacOS.

En raison du manque de temps et de ces obstacles techniques, nous n’avons pas pu insister davantage sur cette solution.

# Conclusion

Ce projet a permis de développer un modèle prédictif performant capable d'anticiper la courbe de SOH (State of Health) des batteries en fonction de leurs données historiques.

Le prétraitement des données a été fondamental pour obtenir des bonnes performances prédictives. Après une exploration et une comparaison approfondies de plusieurs algorithmes, le modèle XGBoost s'est révélé être le plus adapté pour ses performances robustes et sa capacité à capturer les relations complexes entre les features.

Au-delà de la précision prédictive, une attention particulière a été portée à l'interprétation des résultats grâce à des techniques d'explicabilité telles que SHAP. Ces analyses ont permis d'identifier les facteurs clés influençant la dégradation des batteries, notamment certaines caractéristiques comme les temps de charge, les mesures de température, et les capacités de charge à certains temps précédent la valeur de SOH spécifiques. Nous avons constaté que l'importance des caractéristiques peut varier de façon significative selon les instances traitées.

En pratique, ce modèle offre des opportunités concrètes pour guider les décisions liées à la maintenance et à l'optimisation de l'usage des batteries. En anticipant les évolutions de SOH et en mettant en lumière les éléments déterminants de cette dégradation, il devient un outil précieux pour prolonger la durée de vie des batteries et améliorer leur utilisation. Ainsi, ce travail apporte non seulement une solution technique efficace, mais contribue également à une meilleure gestion des ressources énergétiques, répondant à des enjeux industriels et environnementaux cruciaux. Les explications produites sous forme de *feature attribution* peuvent être utiles à un utilisateur expert pour décider de faire confiance ou non aux prédictions réalisées par le modèle selon l'expertise dont il dispose.

# Répartition des tâches

Au début du projet, nous avons travaillé de manière collaborative, souvent en appel, sans plan de répartition strict des tâches. Certaines étapes, comme la préparation des données, nécessitaient une réflexion commune et de nombreux échanges, ce qui en fait un travail réalisé à deux. Progressivement, lorsque Zoé a commencé à implémenter le RNN, le premier modèle testé, nous avons ajusté la phase de préparation des données en la rendant plus propre et adaptée pour les représentations 2D et 3D.

À mesure que le projet avançait et que nos autres engagements se sont accumulés, la répartition des tâches s'est précisée :

- Zoé s'est concentrée sur les entraînements et les optimisations, notamment grâce à un ordinateur plus puissant, ce qui lui a permis d'exécuter les grid search et les modèles les plus exigeants en ressources.
- Charlotte a assuré la synthèse des résultats, en réalisant un notebook qui compare les performances des modèles et met en évidence que XGBoost était le plus performant.

- Enfin, Zoé a pris en charge toute la partie explicabilité, en analysant les importances des features, en étudiant les arbres de décision et en utilisant les techniques SHAP pour fournir une interprétation détaillée des prédictions du modèle.

Cette organisation progressive a permis d'avancer efficacement, en tirant parti de nos points forts respectifs et de nos disponibilités.