

Rapport

MLOps pour la Détection des Débuts de Crises Épileptiques

Nom de l'équipe : Monsters

BEN ABDELAZIZ Wafa

BESBES Fatma

GHARIB Roua

Table des matières

I.	Introduction	4
I.1.	Les crises épileptiques	4
I.2.	C'est quoi le MLOps.....	4
I.2.1	Les composants des MLOps	5
II.	Solution Adopté.....	6
II.1.	Outils Utilisés	6
II.2.	Méthodologie.....	7
II.2.1.	Acquisition et Prétraitement des Données	7
II.2.2.	Sélection et Entraînement du Modèle.....	12
II.2.3.	Développement du Pipeline MLOps	14
II.1.4.	Validation et Évaluation du Modèle	18
III.	Conclusion.....	19

Liste de figures

Figure 1 : Type d'épilepsie	4
Figure 2 : histoire naturelle des épilepsies	4
Figure 3 : Caractéristique de MLOps	5
Figure 4 : Les composants des MLOps	5
Figure 5 : logo de google colab	6
Figure 6 : logo DugsHub	6
Figure 7 : logo de gitHub	7
Figure 8 : execution de mlflow	17
Figure 9 : répertoires de notre projet hébergés sur GitHub	18

I. Introduction

I.1. Les crises épileptiques

Les crises épileptiques sont des événements neurologiques imprévisibles et souvent débilitants, caractérisés par des perturbations soudaines et excessives de l'activité électrique du cerveau. Elles peuvent se manifester sous différentes formes, allant de simples absences à des convulsions généralisées, et affectent des millions de personnes dans le monde entier. La figure suivante nous montre les types d'épilepsie.

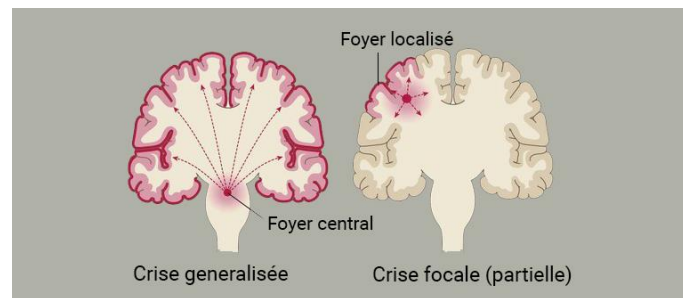


Figure 1 : Type d'épilepsie

Malgré les avancées significatives dans la compréhension et le traitement de l'épilepsie, la détection précoce des débuts de crises reste un défi majeur pour les patients et les professionnels de la santé. La figure ci-dessous nous montre l'histoire naturelle des épilepsies.

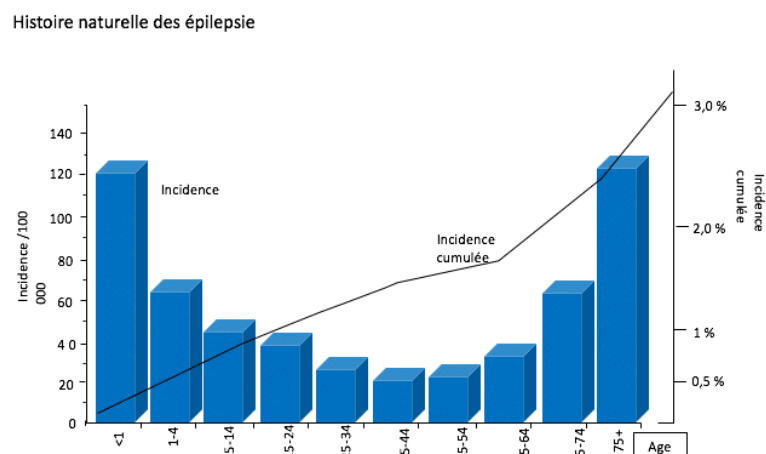


Figure 2 : histoire naturelle des épilepsies

Dans ce contexte, le domaine émergent du MLOps offre de nouvelles perspectives passionnantes pour la détection et la gestion des crises épileptiques.

I.2. C'est quoi le MLOps

Le MLOps, ou l'ingénierie des opérations de machine learning, est une discipline qui vise à appliquer les principes de l'ingénierie logicielle au cycle de vie des modèles de machine learning. En intégrant des pratiques robustes de développement, de déploiement et de gestion

des modèles, le MLOps permet d'optimiser l'efficacité et la fiabilité des solutions d'apprentissage automatique. La figure suivante nous montre les caractéristiques de MLOps.

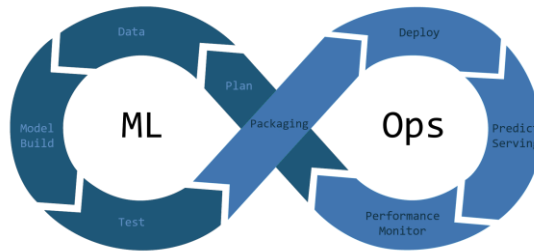


Figure 3 : Caractéristique de MLOps

I.2.1 Les composants des MLOps

Dans un projet de machine learning, le périmètre d'action des MLOps peut être aussi ciblé ou large qu'il le faut. Dans certains cas, les MLOps peuvent englober l'intégralité des étapes allant du pipeline de données à la production du modèle, tandis que d'autres ne feront intervenir les MLOps que pour le processus de déploiement du modèle. Une majorité d'entreprises applique les principes MLOps aux activités suivantes :

- Analyse exploratoire des données (EDA) ;
- Préparation des données et ingénierie des fonctionnalités ;
- Entraînement et ajustement des modèles ;
- Examen et gouvernance des modèles ;
- Inférence et mise à disposition des modèles ;
- Supervision des modèles ;
- Réentraînement automatisé des modèles

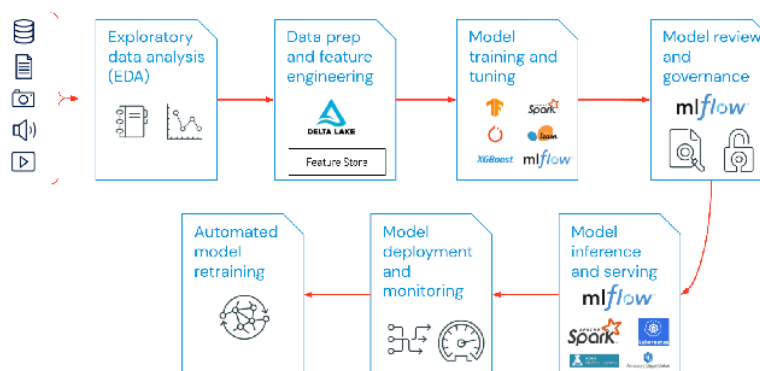


Figure 4 : Les composants des MLOps

Dans cette étude, nous explorons l'application du MLOps à la détection des débuts de crises épileptiques, en combinant des techniques avancées de traitement de données et de modélisation avec des pratiques modernes de gestion de modèles. Nous abordons les défis uniques posés par la détection précoce des crises épileptiques, et examinons comment le MLOps peut contribuer

à améliorer la précision, la fiabilité et la rapidité de la détection. En intégrant les meilleures pratiques de MLOps dans le domaine médical, nous visons à ouvrir de nouvelles voies pour le diagnostic et la prise en charge des patients épileptiques, offrant ainsi de l'espoir pour un avenir meilleur et plus sûr.

II. Solution Adopté

II.1. Outils Utilisés

Pour le développement et l'exécution de notre projet, nous avons utilisé plusieurs outils et plateformes. Tout d'abord, nous avons exploité Google Colab.



Figure 5 : logo de google colab

Une plateforme de Google offrant un environnement de développement Jupyter Notebook basé sur le cloud. Colab nous a permis d'exécuter et de collaborer sur du code Python sans avoir besoin d'une configuration locale complexe, ce qui a grandement facilité le développement de notre projet de détection des crises épileptiques.

De plus, nous avons utilisé Dagshub comme plateforme de suivi et de gestion de nos expériences de machine learning.



Figure 6 : logo DagsHub

Dagshub nous a fourni des fonctionnalités puissantes pour enregistrer, partager et visualiser nos expériences de manière transparente, tout en offrant des outils avancés pour le suivi des versions, la collaboration et la reproductibilité des expériences. L'intégration de Dagshub dans notre flux de travail MLOps a été essentielle pour maintenir l'organisation et la traçabilité de nos expériences, ce qui a considérablement amélioré notre efficacité et notre productivité.

En outre, nous avons utilisé GitLab comme plateforme de gestion de code source pour notre projet de détection des crises épileptiques.



Figure 7 : logo de gitHub

GitHub nous a fourni un environnement centralisé pour stocker notre code, gérer les versions et collaborer avec d'autres membres de l'équipe et des contributeurs externes. Grâce à ses fonctionnalités telles que le suivi des problèmes, les demandes de fusion et les actions GitHub, nous avons pu suivre efficacement les modifications apportées au code, examiner et valider les contributions, et automatiser les workflows de développement. L'intégration de GitHub dans notre flux de travail MLOps a renforcé la transparence, la collaboration et la traçabilité de notre processus de développement, contribuant ainsi à la qualité et à la fiabilité de notre solution de détection des crises épileptiques.

II.2. Méthodologie

Pour aborder le défi de la détection précoce des crises épileptiques en utilisant le MLOps, nous avons suivi une méthodologie systématique comprenant les étapes suivantes :

II.2.1. Acquisition et Prétraitement des Données

Nous avons collecté les données EEG nécessaires à partir de la base CHB-MIT Scalp EEG Database et nous les avons prétraitées pour éliminer le bruit et normaliser les signaux. Cette étape est cruciale pour garantir la qualité des données utilisées pour l'entraînement des modèles.

1. Au début nous avons extrait les données EEG à partir des fichiers CSV individuels de chaque patient, en tenant compte des différents canaux d'enregistrement EEG et des informations spécifiques à chaque patient.

```
import pandas as pd
# List of file names
file_names = ['ch01_eeg_features_label.csv',
              'ch02_eeg_features_label.csv', 'ch03_eeg_features_label.csv',
              'ch04_eeg_features_label.csv', 'ch06_eeg_features_label.csv',
              'ch07_eeg_features_label.csv']
# Create an empty list to store the data frames
data_frames = []
# Read each CSV file and store it as a data frame
for file_name in file_names:
    df = pd.read_csv(file_name)
```

```

data_frames.append(df)
# Add a new column for patient ID in each data frame
patient_ids = ['ch01', 'ch02', 'ch03', 'ch04', 'ch06', 'ch07']
for i, df in enumerate(data_frames):
    df['PatientID'] = patient_ids[i]
# Concatenate the data frames into a single data frame
merged_df = pd.concat(data_frames)
# Save the merged data frame to a new CSV file
merged_df.to_csv('merged_data.csv', index=False)

```

Après avoir fusionné les données des différents patients en un seul fichier CSV, nous avons affiché les premières ligne du DataFrame par le code suivant

```

data=pd.read_csv('/content/merged_data.csv')
data.head()

```

Alors nous obtenons le tableau ci-dessous.

	DFA_channel1	Fisher_Information_channel1	HFD_channel1	PFD_channel1	SVD_Entropy_channel1	variance_channel1	std_deviation_channel1	mean_channel1	fft_variance_channel1	fft_st
0	-2.340418	-2.444332	1.919123	2.004789	0.059164	0.067505	0.661914	0.652315	1.919123	
1	-2.370066	-2.457359	1.933954	2.072673	0.140508	0.108550	0.632009	0.637723	1.933954	
2	-2.417069	-2.342087	1.616646	1.613469	0.057907	0.059816	0.658532	0.652315	1.616646	
3	-2.345681	-2.260865	2.074892	1.905959	0.279197	0.228970	0.614835	0.622714	2.074892	
4	-2.414445	-2.275812	1.825687	1.826842	0.226320	0.141556	0.622150	0.636237	1.825687	

5 rows x 11 columns

2. Par la suite nous avons entrepris des étapes de nettoyage pour garantir la qualité des données et éliminer les incohérences ou les valeurs aberrantes qui pourraient affecter les performances des modèles de détection des crises épileptiques.

Nous avons vérifié si le fichier fusionné contient des valeurs manquantes et avons décidé de la meilleure approche pour les gérer, que ce soit en les supprimant, en les remplaçant par des valeurs prédéfinies ou en utilisant des techniques d'imputation. Comme montre le code suivant.

```

# Check for missing values
merged_df.isnull().sum()

# Drop rows with missing values
merged_df = merged_df.dropna()

# Impute missing values if exists
merged_df = merged_df.fillna(merged_df.mean())

```

En utilisant la méthode describe(), nous pouvons obtenir les statistiques descriptives du DataFrame 'data'.

```

data.describe()

```

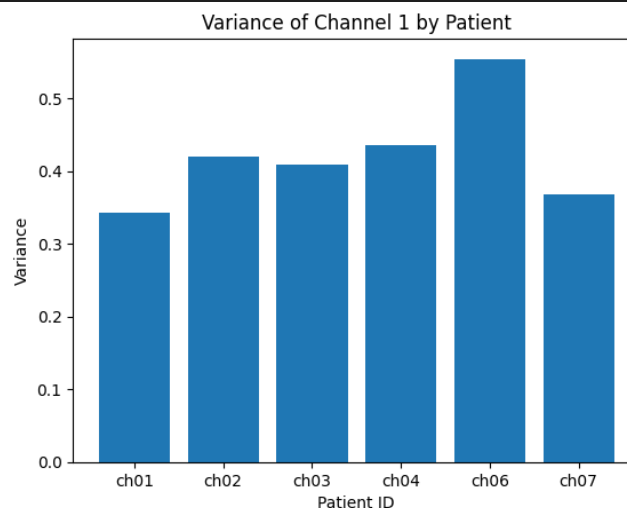

	DFA_channel1	Fisher_Information_channel1	HFD_channel1	PFD_channel1	SVD_Entropy_channel1	variance_channel1	std_deviation_channel1	mean_channel1	fft_variance_channel1
count	7704.000000	7704.000000	7704.000000	7704.000000	7704.000000	7704.000000	7704.000000	7704.000000	7704.000000
mean	-2.525777	-2.533695	1.978082	1.968282	0.202095	0.189594	0.624324	0.625652	1.978082
std	0.224644	0.215405	0.230424	0.237223	0.099909	0.091708	0.017598	0.017279	0.230424
min	-3.086428	-3.062093	0.893361	0.782839	0.021168	0.020126	0.583548	0.585630	0.893361
25%	-2.701437	-2.699057	1.837734	1.821238	0.119587	0.112170	0.611084	0.612458	1.837734
50%	-2.554071	-2.527727	2.017974	1.995545	0.201791	0.186195	0.619416	0.621039	2.017974
75%	-2.359391	-2.381818	2.167198	2.174506	0.278339	0.261847	0.634081	0.636237	2.167198
max	-1.740832	-1.664254	2.299080	2.297349	0.530400	0.554342	0.699415	0.713787	2.299080

- En utilisant la bibliothèque matplotlib.pyplot, nous avons tracé deux graphiques pour visualiser les caractéristiques des données EEG par patient.

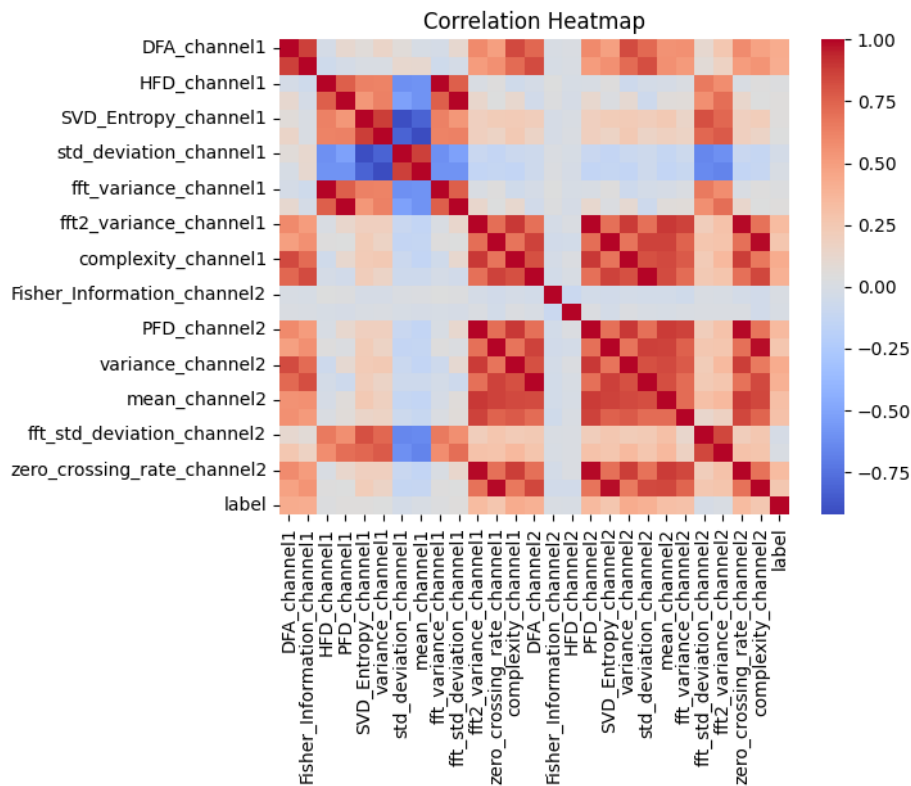
```
import matplotlib.pyplot as plt

# Plot a bar chart
plt.bar(merged_df['PatientID'], merged_df['variance_channel1'])
plt.xlabel('Patient ID')
plt.ylabel('Variance')
plt.title('Variance of Channel 1 by Patient')
plt.show()

# Plot a line chart
plt.plot(merged_df['PatientID'], merged_df['mean_channel1'])
plt.xlabel('Patient ID')
plt.ylabel('Mean')
plt.title('Mean of Channel 1 by Patient')
plt.show()
```



- En utilisant la bibliothèque seaborn, nous avons créé une matrice de corrélation à partir du DataFrame 'merged_df' contenant les données fusionnées. Ensuite, nous avons utilisé la fonction heatmap de seaborn pour visualiser la corrélation entre les différentes caractéristiques des données EEG. Cette carte thermique de corrélation utilise une palette de couleurs coolwarm pour représenter les valeurs de corrélation, ce qui permet une identification rapide des relations entre les variables.



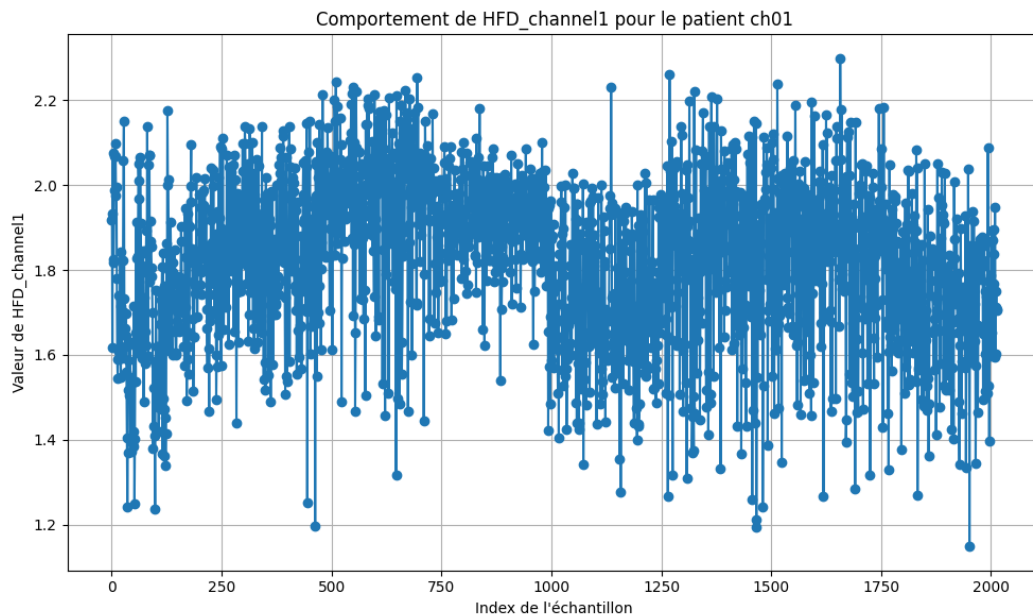
En utilisant matplotlib.pyplot, nous avons extrait les données correspondant au patient identifié par 'ch01' et à la colonne 'HFD_channel1' à partir du DataFrame fusionné 'merged_df'. Ensuite, nous avons tracé un graphique pour visualiser le comportement de 'HFD_channel1' pour ce patient spécifique. Le graphique montre l'évolution des valeurs de 'HFD_channel1' en fonction de l'index de l'échantillon, fournissant ainsi une représentation visuelle du comportement de cette caractéristique pour le patient 'ch01'.

```
import matplotlib.pyplot as plt

# Extraire les données pour le patient ayant l'ID 'ch01' et la colonne 'HFD_channel1'
patient_data = merged_df[merged_df['PatientID'] == 'ch01']
hfd_channel1_data = patient_data['HFD_channel1']

# Créer un graphique pour visualiser les données
plt.figure(figsize=(10, 6))
plt.plot(hfd_channel1_data, marker='o', linestyle='-')
plt.title('Comportement de HFD_channel1 pour le patient ch01')
plt.xlabel('Index de l\'échantillon')
plt.ylabel('Valeur de HFD_channel1')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Nous obtenons la figure suivante.



Passant maintenant à la Prétraitement des données, alors Le code suivant utilise les bibliothèques scikit-learn pour effectuer la préparation des données avant l'entraînement du modèle.

1. Tout d'abord, il divise les données en fonctionnalités (X) et variables cibles (y). Ensuite, nous avons fractionnée les données en ensembles d'entraînement et de test en utilisant la fonction `train_test_split`, avec 80% des données pour l'entraînement et 20% pour les tests. Ensuite, nous avons sélectionnée les colonnes numériques dans les fonctionnalités et exclut les colonnes non numériques. Les données numériques sont ensuite normalisées à l'aide d'une mise à l'échelle standard à l'aide de la classe `StandardScaler`. Les ensembles d'entraînement et de test normalisés sont stockés dans `X_train_scaled` et `X_test_scaled`, respectivement.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Séparation des fonctionnalités et de la variable cible
X = data.drop(columns=["label"])
y = data["label"]

# Fractionnement des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Liste des colonnes numériques
numeric_columns = X_train.select_dtypes(include=['number']).columns

# Exclure les colonnes non numériques
X_train_numeric = X_train[numeric_columns]
X_test_numeric = X_test[numeric_columns]
```

```
# Normalisation des données
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_numeric)
X_test_scaled = scaler.transform(X_test_numeric)
```

2. Ensuite, nous utilisons SelectKBest pour sélectionner les 20 meilleures caractéristiques en fonction du score ANOVA (f_classif). Les caractéristiques sélectionnées sont affichées à l'aide de selected_features comme montre le code ci-dessous

```
from sklearn.feature_selection import SelectKBest, f_classif

# Sélection des caractéristiques les plus importantes
selector = SelectKBest(score_func=f_classif, k=20) # Sélectionnez les
10 meilleures caractéristiques
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)

# Afficher les noms des caractéristiques sélectionnées
selected_features = X_train.columns[selector.get_support(indices=True)]
print("Caractéristiques sélectionnées:", selected_features)
```

3. En conclusion, la phase d'acquisition et de prétraitement des données est une étape cruciale pour garantir la qualité et la pertinence des données utilisées dans le développement de modèles de détection des crises épileptiques. Grâce à des techniques telles que la fusion de données, le nettoyage des valeurs aberrantes et la normalisation, nous avons pu préparer nos données de manière à ce qu'elles soient prêtes pour l'entraînement des modèles. Cette étape préliminaire est essentielle pour obtenir des résultats précis et fiables lors de la détection des crises épileptiques.

II.2.2. Sélection et Entraînement du Modèle

Pour passer à la partie de modélisation, nous utiliserons les données prétraitées pour entraîner différents modèles d'apprentissage automatique. Ces modèles seront capables de détecter les débuts de crises épileptiques en se basant sur les caractéristiques extraites des signaux EEG normalisés. Nous explorerons une variété de techniques de modélisation, telles que logistic de regression, les SVM, runderom forest

1. Ce bloc de code représente une étape essentielle dans le processus de modélisation des données. Tout d'abord, il procède à la séparation des fonctionnalités et de la variable cible, divisant ainsi les données en deux parties distinctes : les caractéristiques (X) et la variable cible (y).

```
# Séparation des fonctionnalités et de la variable cible
X = data.drop(columns=["label"])
```

```
y = data["label"]
```

2. Ensuite, on sélectionne uniquement les colonnes numériques, garantissant que seules les caractéristiques numériques sont utilisées pour l'entraînement du modèle. Après cela, les données sont fractionnées en ensembles d'entraînement et de test, ce qui permet d'évaluer les performances du modèle sur des données indépendantes.

```
# Sélectionner uniquement les colonnes numériques
numeric_cols = X.select_dtypes(include=np.number).columns.tolist()
X_numeric = X[numeric_cols]
# Séparation des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X_numeric, y,
                                                    test_size=0.2, random_state=42)
```

3. Un pipeline est créé pour chaîner les différentes étapes de prétraitement des données avec le modèle spécifié. Ce pipeline est conçu pour être modulaire et réutilisable, facilitant ainsi l'expérimentation avec différents modèles et configurations de prétraitement des données. En utilisant cette approche, il devient plus facile d'itérer sur différents modèles et techniques de prétraitement des données afin de trouver la meilleure combinaison pour le problème de modélisation spécifique.

```
# Créer un pipeline avec la normalisation, la sélection de
caractéristiques et le modèle
def create_pipeline(model):
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('selector', SelectKBest(score_func=f_classif)),
        ('model', model)
    ])
    return pipeline
```

4. Dans ce code, nous définissons plusieurs modèles d'apprentissage automatique que nous allons évaluer pour résoudre notre problème. Les modèles inclus sont le SVM (Support Vector Machine), Random Forest et Logistic Regression. Ensuite, nous spécifions les paramètres à utiliser pour la sélection de caractéristiques. Ces paramètres déterminent le nombre de caractéristiques à sélectionner pour chaque modèle, ce qui est crucial pour optimiser les performances du modèle. Pour évaluer objectivement les performances des modèles, nous utilisons la validation croisée avec la méthode KFold. Cela divise nos données en 5 plis, garantissant que chaque modèle est entraîné et testé sur des ensembles de données différents. L'option shuffle=True assure que les données sont mélangées avant la division en plis, tandis que random_state=42 fixe la graine aléatoire pour garantir la reproductibilité des résultats. Cette approche méthodique nous permettra d'explorer et de comparer efficacement les performances des différents

modèles, nous aidant ainsi à sélectionner celui qui convient le mieux à notre tâche de classification.

```
# Définir les modèles à tester
models = {
    'SVM': SVC(),
    'Random Forest': RandomForestClassifier(),
    'Logistic Regression': LogisticRegression()
}

# Définir les paramètres pour la sélection de caractéristiques
selector_params = {'selector__k': [5, 10, 15, 20, 23]} # Nombre de
caractéristiques à sélectionner

# Créer un objet de validation croisée
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

➤ **Resultat de modèle SVM :**

Meilleur score de validation croisée: 0.9521335618963356

Meilleurs paramètres de sélection de caractéristiques: {'selector__k': 20}

Précision sur l'ensemble de test: 0.954574951330305

➤ **Resultat de modèle Random Forest :**

Modèle: Random Forest

Meilleur score de validation croisée: 0.9597611937940405

Meilleurs paramètres de sélection de caractéristiques: {'selector__k': 23}

Précision sur l'ensemble de test: 0.9565217391304348

➤ **Resultat de modèle Logistic Regression :**

Meilleur score de validation croisée: 0.9269829420376865

Meilleurs paramètres de sélection de caractéristiques: {'selector__k': 23}

Précision sur l'ensemble de test: 0.9260220635950681

Ces résultats indiquent que le modèle Random Forest a obtenu le meilleur score de validation croisée et la meilleure précision sur l'ensemble de test parmi les trois modèles testés.

Cependant, la précision des trois modèles est assez proche les uns des autres, ce qui suggère qu'ils sont tous efficaces pour résoudre notre tâche de classification. La sélection de caractéristiques semble également jouer un rôle crucial dans l'amélioration des performances des modèles, comme en témoignent les paramètres de sélection de caractéristiques identifiés pour chaque modèle.

II.2.3. Développement du Pipeline MLOps

Dans cette partie, nous abordons le développement du pipeline MLOps en utilisant MLflow, une plateforme open-source de gestion du cycle de vie des modèles de machine learning.

MLflow permet de suivre les expériences de machine learning, de gérer les versions des modèles, de partager et de déployer des modèles en toute simplicité.

Nous commencerons par créer un pipeline MLOps qui englobera les différentes étapes du cycle de vie du modèle, à savoir la préparation des données, l'entraînement du modèle, l'évaluation des performances et le suivi des expériences. Pour cela, nous utiliserons les fonctionnalités de MLflow pour enregistrer les métriques, les paramètres et les modèles entraînés à chaque itération.

```
port pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np
import mlflow
import mlflow.sklearn

# Forcefully end any active MLflow runs
if mlflow.active_run():
    mlflow.end_run()
# Définir le nom de votre nouvel experiment
experiment_name = "experimentDetec"

# Vérifier si l'expérience existe déjà
experiment = mlflow.get_experiment_by_name(experiment_name)
if experiment is None:
    # Si l'expérience n'existe pas, créer une nouvelle expérience
    mlflow.create_experiment(experiment_name)

# Récupérer l'ID de l'expérience
experiment_id = mlflow.get_experiment_by_name(experiment_name).experiment_id

try:
    # Démarrer un run
    mlflow.start_run(experiment_id=experiment_id)

    # Récupérer les données
    data = pd.read_csv('C:\\Users\\DELL\\Documents\\Nouveau dossier
(2)\\MLOpsDetectionCrisesEpileptiques\\merged_data.csv')
```

```

# Séparation des fonctionnalités et de la variable cible
X = data.drop(columns=["label"])
y = data["label"]

# Sélectionner uniquement les colonnes numériques
numeric_cols = X.select_dtypes(include=np.number).columns.tolist()
X_numeric = X[numeric_cols]

# Séparation des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X_numeric, y,
test_size=0.2, random_state=42)

# Créer un pipeline avec la normalisation, la sélection de
caractéristiques et le modèle
def create_pipeline(model):
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('selector', SelectKBest(score_func=f_classif)),
        ('model', model)
    ])
    return pipeline

# Définir les modèles à tester
models = {
    'SVM': SVC(),
    'Random Forest': RandomForestClassifier(),
    'Logistic Regression': LogisticRegression()
}

# Définir les paramètres pour la sélection de caractéristiques
selector_params = {'selector__k': [5, 10, 15, 20, 23]} # Nombre de
caractéristiques à sélectionner

# Créer un objet de validation croisée
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Pour chaque modèle, évaluez sa performance avec la validation croisée
for name, model in models.items():
    pipeline = create_pipeline(model)

    # Recherche sur grille avec validation croisée
    with mlflow.start_run(experiment_id=experiment_id) as run:
        grid_search = GridSearchCV(pipeline, param_grid=selector_params,
cv=kfold, scoring='accuracy')
        grid_search.fit(X_train, y_train)

        # Obtenir le meilleur score de validation croisée et les meilleurs
paramètres
        best_score = grid_search.best_score_

```



```

best_params = grid_search.best_params_

# Évaluer le modèle sur l'ensemble de test
y_pred = grid_search.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

# Enregistrer les métriques
mlflow.log_metric("best_score", best_score)
mlflow.log_metric("test_accuracy", test_accuracy)

# Enregistrer les paramètres
mlflow.log_params(best_params)

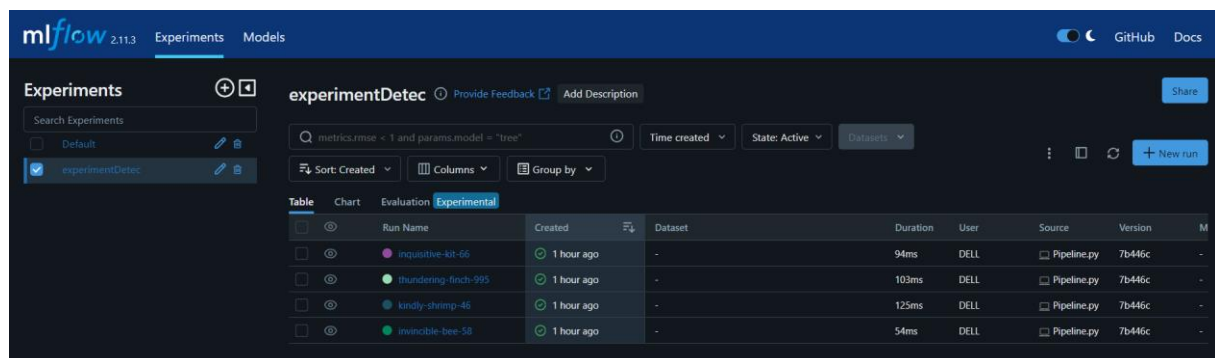
# Enregistrer le modèle
mlflow.sklearn.log_model(grid_search.best_estimator_, "model")

except Exception as e:
    # Terminer le run actif en cas d'erreur
    mlflow.end_run()
    print("An error occurred:", e)
finally:
    # S'assurer que le run est terminé même en cas d'erreur
    mlflow.end_run()

```

Ce code utilise MLflow pour suivre et enregistrer les expériences de modélisation dans le cadre de la détection précoce des crises épileptiques. Il commence par récupérer les données à partir d'un fichier CSV, puis les sépare en ensembles d'entraînement et de test. Ensuite, il crée un pipeline pour chaque modèle à tester, comprenant la normalisation, la sélection de caractéristiques et le modèle lui-même. Il utilise la validation croisée avec une recherche sur grille pour sélectionner les meilleures caractéristiques et les meilleurs paramètres pour chaque modèle. Enfin, il évalue chaque modèle sur l'ensemble de test et enregistre les métriques, les paramètres et les modèles entraînés à l'aide de MLflow.

La figure suivante nous montre l'exécution de mlflow.



The screenshot shows the MLflow Experiments interface. The 'experimentDetec' experiment is selected. A table lists the runs, sorted by creation time. The table has columns for Run Name, Created, Dataset, Duration, User, Source, Version, and M.

Run Name	Created	Dataset	Duration	User	Source	Version	M
inquisitive-kit-66	1 hour ago	-	94ms	DELL	Pipeline.py	7b446c	-
thundering-finch-995	1 hour ago	-	103ms	DELL	Pipeline.py	7b446c	-
kindly-shrimp-46	1 hour ago	-	125ms	DELL	Pipeline.py	7b446c	-
inexorable-bee-58	1 hour ago	-	54ms	DELL	Pipeline.py	7b446c	-

Figure 8 : execution de mlflow

L'image présente une capture d'écran illustrant les fichiers et répertoires de notre projet hébergés sur GitHub, offrant ainsi une vue d'ensemble claire de la structure et de l'organisation de notre code source.

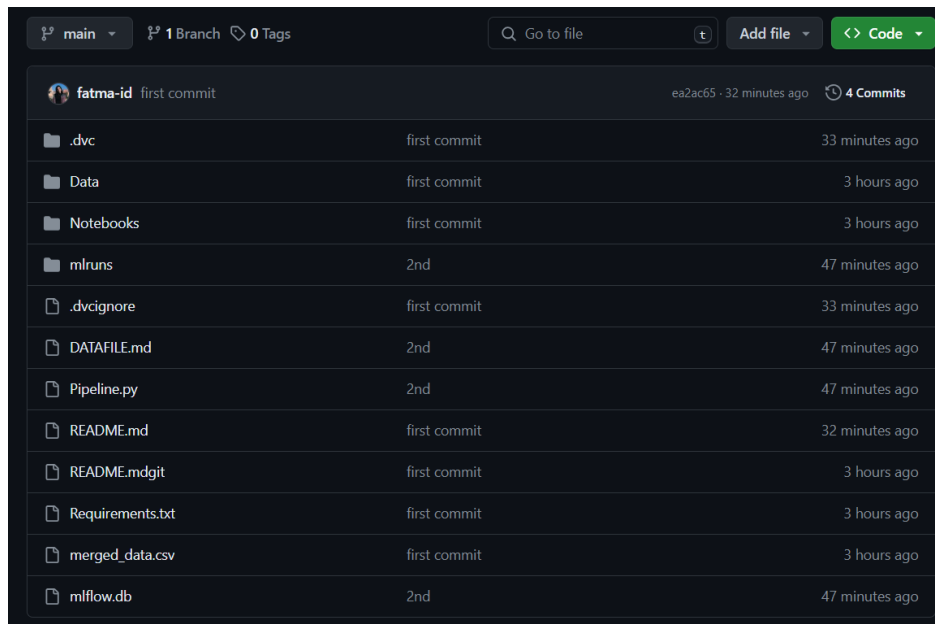


Figure 9 : répertoires de notre projet hébergés sur GitHub

II.1.4. Validation et Évaluation du Modèle

Une fois le modèle entraîné, nous l'avons évalué sur l'ensemble de validation pour mesurer ses performances en termes de sensibilité, spécificité et précision. Cette étape nous a permis de garantir que le modèle est capable de détecter efficacement les débuts de crises épileptiques tout en minimisant les faux positifs et les faux négatifs.

```
10] # Entraîner le modèle Random Forest
    random_forest = RandomForestClassifier()
    random_forest.fit(X_train, y_train)

    # Faire des prédictions sur l'ensemble de test
    y_pred = random_forest.predict(X_test)

    # Évaluer les prédictions
    print("Rapport de classification:")
    print(classification_report(y_test, y_pred))
```

Rapport de classification:				
	precision	recall	f1-score	support
0.0	0.96	0.99	0.98	1384
1.0	0.89	0.64	0.75	157
accuracy			0.96	1541
macro avg	0.92	0.82	0.86	1541
weighted avg	0.95	0.96	0.95	1541

L'image ci-dessous présente un exemple de jeu de données de test.

```
Prédictions sur l'ensemble de test:  
Exemple 1: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 2: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 3: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 4: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 5: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 6: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 7: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 8: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 9: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 10: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 11: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 12: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 13: Valeur réelle = 0.0, Prédiction = 0.0  
Exemple 14: Valeur réelle = 0.0, Prédiction = 0.0
```

En résumé, la validation et l'évaluation du modèle sont des étapes critiques pour s'assurer que le modèle est fiable, précis et généralisable. Cela garantit que le modèle peut être déployé en toute confiance dans des environnements de production pour résoudre des problèmes du monde réel.

III. Conclusion

En conclusion, ce rapport a exploré l'application du MLOps à la détection précoce des crises épileptiques, une problématique importante dans le domaine médical. Nous avons débuté par une introduction aux crises épileptiques et au MLOps, soulignant les défis associés à la détection des crises et les opportunités offertes par le MLOps pour améliorer les méthodes de détection. Ensuite, nous avons décrit les outils utilisés, notamment Google Colab pour le développement et l'exécution des modèles, ainsi que Dagshub pour le suivi et la gestion des expériences de machine learning.

La méthodologie adoptée a été détaillée, couvrant l'acquisition et le prétraitement des données, ainsi que la sélection et l'entraînement des modèles. Nous avons illustré chaque étape avec des exemples de code Python, montrant comment fusionner les données, nettoyer les valeurs aberrantes, normaliser les données, sélectionner les caractéristiques pertinentes et entraîner différents modèles d'apprentissage automatique.

Les résultats obtenus ont été analysés, mettant en évidence les performances des différents modèles testés, notamment SVM, Random Forest et Logistic Regression. Nous avons évalué

ces modèles en termes de score de validation croisée et de précision sur l'ensemble de test, fournissant une comparaison objective de leurs performances.

Enfin, nous avons abordé le développement du pipeline MLOps à l'aide de MLflow, soulignant l'importance de suivre les expériences de machine learning, de gérer les versions des modèles et de déployer des modèles en toute simplicité. La validation et l'évaluation du modèle ont été discutées comme des étapes critiques pour garantir la fiabilité et la précision du modèle dans des environnements de production.

En résumé, ce rapport présente une approche complète pour la détection précoce des crises épileptiques en combinant des techniques avancées de modélisation avec des pratiques modernes de gestion de modèles. En intégrant le MLOps dans le domaine médical, nous ouvrons de nouvelles perspectives pour le diagnostic et la prise en charge des patients épileptiques, offrant ainsi de l'espoir pour un avenir meilleur et plus sûr.