المدرسة الوطنية للذكاء الاصطناعي والرقمنة بركان

ⵜⵉⵎⵙⵙⵓⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵏ ⵜⵣⵣⵓⵔⵜ ⵜⴰⵎⴰⵡⴰⵙⵜ ⴰⵜⵓⵎⵜⵉⵣⵉⵏ ⴷ ⵜⵓⵎⵓⵔⵜ

École Nationale de l'Intelligence Artificielle et du Digital Berkane

# Earth quake prediction


**Réalisé Par :**

Anas DARRAZ

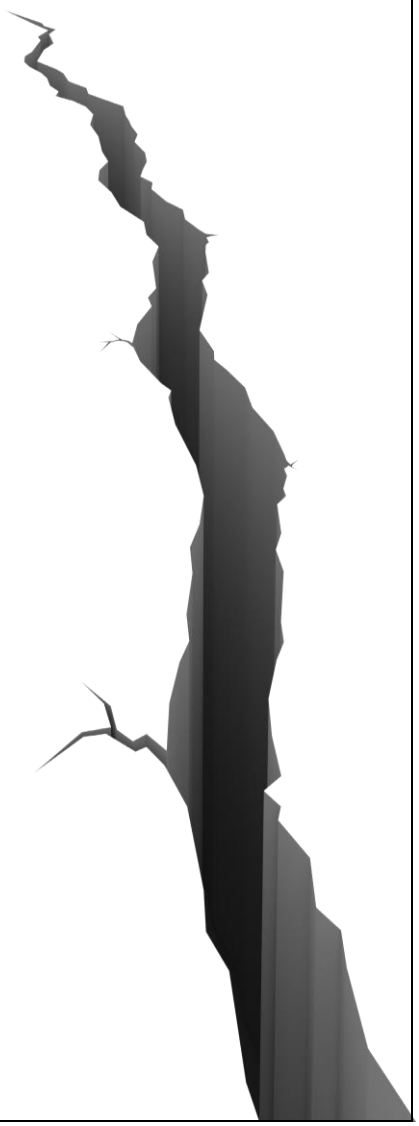
Rachid EL MAGROUA

Ilyas MAJDOUBI

Nabil EL HILALI

**Encadré par :**

Pr. Asmae BENTALEB

**Filière :** Ingénierie réseaux et sécurité informatique

**Module :** Maching Learning

2024/2025

# Remerciement

Nous tenons à adresser nos sincères remerciements à notre professeur, **Asmae BENTALEB**, pour son accompagnement exceptionnel tout au long de ce projet. Sa disponibilité, son expertise et ses conseils avisés ont été d'une aide précieuse pour surmonter les difficultés rencontrées et pour garantir la qualité des travaux réalisés. Elle a su nous guider avec rigueur et bienveillance, nous motivant à donner le meilleur de nous-mêmes. Sa passion pour l'enseignement et son approche pédagogique ont été des sources d'inspiration pour chacun d'entre nous.

Nous exprimons également notre gratitude à tous les membres de notre équipe, dont le dévouement, l'engagement et la collaboration ont été les clés du succès de ce projet. Chaque membre a apporté une contribution essentielle, que ce soit à travers des idées novatrices, des efforts dans l'analyse des données ou le développement des modèles. L'esprit d'équipe qui a prévalu tout au long de ce projet a permis de surmonter les obstacles, de partager des connaissances et d'apprendre ensemble dans un environnement stimulant.

Enfin, nous reconnaissons que ce projet est le fruit d'un véritable effort collectif, où chacun a joué un rôle unique et indispensable. C'est grâce à cette synergie entre l'encadrement de qualité de notre professeur et l'investissement des membres de l'équipe que nous avons pu aboutir à des résultats enrichissants et conformes à nos objectifs initiaux. Nous adressons à tous un grand merci pour leur précieuse contribution

# Contenu :

# Introduction

Les tremblements de terre, ou séismes, sont des secousses soudaines de la croûte terrestre causées par le relâchement d'énergie accumulée, principalement le long des limites des plaques tectoniques. Ces phénomènes naturels, imprévisibles et parfois dévastateurs, représentent un défi majeur pour la compréhension scientifique et la prévention des risques.

C'est dans ce contexte que la **machine learning**, une branche de l'intelligence artificielle, joue un rôle croissant. En analysant de vastes quantités de données sismiques recueillies par des réseaux de capteurs, ces algorithmes permettent de détecter des schémas complexes et subtils dans l'activité tectonique. Machine learning offre ainsi des outils pour prédire la probabilité de futures secousses, identifier les zones à haut risque et améliorer les systèmes d'alerte précoce.

Grâce à cette approche, la science des séismes évolue vers une meilleure compréhension des processus sous-jacents et une gestion plus efficace des impacts, transformant les données massives en solutions concrètes pour la sécurité des populations.

# Objectif du projet

Ce projet a pour objectif d'utiliser des modèles d'apprentissage automatique pour estimer la magnitude des tremblements de terre. En exploitant des ensembles de données sismiques, nous chercherons à comprendre les relations entre différents facteurs géophysiques et la magnitude des séismes.

Dans un premier temps, nous effectuerons une exploration approfondie des données grâce à des visualisations sous forme de cartes et de graphiques. Ces représentations permettront d'identifier des tendances, des anomalies et des relations potentielles entre les variables. Cette étape est cruciale pour orienter les étapes suivantes du projet.

Ensuite, plusieurs modèles de régression seront développés et testés. Chaque modèle sera évalué en fonction de ses performances, telles que la précision, l'efficacité et la capacité à généraliser sur de nouvelles données. L'objectif final sera de comparer ces modèles afin de sélectionner celui qui offre les meilleures prédictions pour estimer la magnitude des tremblements de terre.

Ce projet combine ainsi l'analyse de données, la modélisation prédictive et l'évaluation de modèles, tout en s'appuyant sur des outils de visualisation pour faciliter l'interprétation et la communication des résultats. Les conclusions pourront contribuer à une meilleure compréhension des phénomènes sismiques et à l'amélioration des outils de gestion des risques.

# I. Identification des données :

## 1. Bibliothèques Python pour la Régression en Machine Learning et l'Analyse des Données :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,r2_score

from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

**pandas** : Permet de manipuler des données tabulaires sous forme de DataFrame pour charger, nettoyer et analyser les données.

**train_test_split** : Divise les données en ensembles d'entraînement et de test pour évaluer les performances du modèle sur des données non vues.

**RandomForestRegressor** : Implémente l'algorithme Random Forest pour les tâches de régression en combinant plusieurs arbres de décision pour améliorer la précision et réduire le surapprentissage.

**sklearn.metrics** : Fournit des outils pour mesurer les performances du modèle avec des métriques comme MAE, MSE et $R^2$.

**LabelEncoder** : Encode les variables catégoriques en valeurs numériques pour préparer les données à l'entraînement.

**matplotlib.pyplot** : Crée des visualisations statiques pour explorer les données ou interpréter les résultats.

**seaborn** : Produit des graphiques stylés et intuitifs pour analyser les relations entre les variables.

**numpy** : Fournit des outils pour manipuler des tableaux et effectuer des calculs numériques avancés.

## 2. Informations du Dataset :

- Chargement des données :

```
df =pd.read_csv('earthquakes.csv')
```

- Décrire les données : `df.describe()`

| | magnitude | time | updated | felt | cdi | mmi | tsunami | sig | nst | dmin | rms | gap |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 1137.000000 | 1.137000e+03 | 1.137000e+03 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 |
| mean | 4.856675 | 1.712005e+12 | 1.716957e+12 | 414.408091 | 2.925242 | 4.320141 | 0.059807 | 432.698329 | 115.094107 | 1.342604 | 0.585974 | 55.055286 |
| std | 1.047840 | 1.151499e+10 | 1.108033e+10 | 5746.971362 | 2.562707 | 1.453949 | 0.237232 | 256.177844 | 91.877870 | 1.704364 | 0.308556 | 37.609237 |
| min | 3.000000 | 1.690000e+12 | 1.690000e+12 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 138.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 3.800000 | 1.700000e+12 | 1.710000e+12 | 0.000000 | 0.000000 | 4.000000 | 0.000000 | 234.000000 | 37.000000 | 0.100000 | 0.300000 | 30.000000 |
| 50% | 5.300000 | 1.710000e+12 | 1.720000e+12 | 2.000000 | 3.000000 | 4.000000 | 0.000000 | 449.000000 | 102.000000 | 0.680000 | 0.630000 | 49.000000 |
| 75% | 5.600000 | 1.720000e+12 | 1.730000e+12 | 24.000000 | 5.000000 | 5.000000 | 0.000000 | 518.000000 | 157.000000 | 2.061000 | 0.780000 | 68.000000 |
| max | 7.600000 | 1.730000e+12 | 1.730000e+12 | 183786.000000 | 9.000000 | 9.000000 | 1.000000 | 2419.000000 | 619.000000 | 12.457000 | 2.520000 | 256.000000 |

- Informations avec info() :

```
RangeIndex: 1137 entries, 0 to 1136
Data columns (total 43 columns):
 #    Column              Non-Null Count  Dtype
---   ------              --------------  -----
 0    id                  1137 non-null   object
 1    magnitude           1137 non-null   float64
 2    type                1137 non-null   object
 3    title               1137 non-null   object
 4    date                1137 non-null   object
 5    time                1137 non-null   float64
 6    updated             1137 non-null   float64
 7    url                 1137 non-null   object
 8    detailUrl           1137 non-null   object
 9    felt                1137 non-null   int64
 10   cdi                 1137 non-null   int64
 11   mmi                 1137 non-null   int64
 12   alert               764 non-null    object
 13   status              1137 non-null   object
 14   tsunami             1137 non-null   int64
 15   sig                 1137 non-null   int64
 16   net                 1137 non-null   object
 17   code                1137 non-null   object
 18   ids                 1137 non-null   object
 19   sources             1137 non-null   object
 20   types               1137 non-null   object
 21   nst                 1137 non-null   int64
 22   dmin                1137 non-null   float64
 23   rms                 1137 non-null   float64
 24   gap                 1137 non-null   float64
 25   magType             1137 non-null   object
 26   geometryType        1137 non-null   object
```

## 3. Prétraitement des données :

## a. Suppression des colonnes :

```
#droping unused columns
# we will use 'longitude' and 'latitude' instead

df = df.drop(columns=['locality','placeOnly','location','place','timezone','locationDetails'])

#constant columns
constant_columns = [col for col in df.columns if df[col].nunique() == 1]
df = df.drop(columns=constant_columns)

#Droping description columns
df=df.drop(columns=['id','ids','title','date','time','updated', 'url','code','detailUrl','sources', 'types','what3words','status','tsunami'])
```

- ***Colonnes inutilisées :***
  Colonnes supprimées : Les colonnes telles que *locality*, *placeOnly*, *location*, *place*, *timezone*, et *locationDetails* sont considérées comme inutiles pour l'analyse, car seules les informations de longitude et latitude seront utilisées pour géolocaliser les tremblements de terre.
- ***Colonnes Constantes :***
  Colonnes supprimées : Les colonnes constantes n'apportent aucune information utile à l'analyse. Elles sont donc supprimées du DataFrame pour améliorer l'efficacité de l'analyse et de l'entraînement du modèle.
- ***Colonnes Descriptives :***
  Colonnes supprimées : Les colonnes telles que *id*, *title*, *date*, *url*, et autres sont liées à des descriptions des tremblements de terre et ne sont pas nécessaires pour l'analyse géospatiale ou la prédiction des magnitudes. Ces colonnes sont donc supprimées pour alléger le DataFrame et éviter d'inclure des informations non pertinentes.

## b. Gestion des valeurs manquantes :

```
#columns with messing values
missing_values = df.isnull().sum()
missing_columns = missing_values[missing_values > 0]
missing_columns
```

- La variable `missing_columns` contient les colonnes du DataFrame qui ont des valeurs manquantes. Ce résultat peut être utilisé pour décider des actions à entreprendre, comme le remplacement des valeurs manquantes ou la suppression des lignes/colonnes concernées.

```
alert          373
continent      270
country        338
subnational    421
city           463
postcode       940
dtype: int64
```

## c. Prédire les valeurs manquantes pour alert :

Dans ce projet, les valeurs manquantes de la colonne "alert" ont été prédites en utilisant un modèle de Random Forest. Ce choix s'explique par la capacité de cet algorithme à gérer des données complexes et à capturer les relations non linéaires entre les variables. Les autres colonnes du jeu de données ont été utilisées comme caractéristiques pour entraîner le modèle, afin de prédire les valeurs manquantes de "alert". Cette colonne est essentielle car elle peut fournir des informations supplémentaires pour affiner la prédiction de la magnitude des tremblements de terre. Grâce à cette approche, nous avons pu compléter les données tout en conservant leur cohérence et leur pertinence pour les analyses ultérieures.

```python
# Predict Missing Alerts Using Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Columns to use
columns = ['alert', 'longitude', 'latitude', 'depth', 'sig', 'nst', 'rms', 'dmin', 'mmi', 'cdi']
df_alert = df[columns]
# Separate rows with and without missing 'alert'
df_known = df_alert.dropna(subset=['alert'])
df_missing = df_alert[df_alert['alert'].isnull()]
# Features and target
X = df_known.drop(columns=['alert'])
y = df_known['alert']
# Train a classifier
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = RandomForestClassifier()
model.fit(X_train, y_train)
# Predict missing 'alert' values
missing_alerts = model.predict(df_missing.drop(columns=['alert']))
# Update the 'alert' column in the original DataFrame
df.loc[df['alert'].isnull(), 'alert'] = missing_alerts
# The df DataFrame now contains the predicted values for the missing 'alert' column
# Evaluate the model accuracy
y_pred = model.predict(X_test)  # Predictions on the test set
accuracy = accuracy_score(y_test, y_pred)  # Calculate accuracy
print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

***• Sélection des colonnes utilisées :***

- `columns = ['alert', 'longitude', 'latitude', 'depth', 'sig', 'nst', 'rms', 'dmin', 'mmi', 'cdi']` : Ce tableau définit les colonnes pertinentes pour l'entraînement du modèle, où **`alert`** est la variable cible (à prédire), et les autres colonnes sont utilisées comme caractéristiques (features).

***• Séparation des lignes avec et sans valeur manquante pour 'alert' :***

- `df_known = df_alert.dropna(subset=['alert'])` : Cette ligne conserve les lignes où la valeur de **`alert`** est connue.
- `df_missing = df_alert[df_alert['alert'].isnull()]` : Cette ligne isole les lignes avec des valeurs manquantes pour **`alert`**.

***• Séparation des données en caractéristiques (features) et cible (target) :***

- `X = df_known.drop(columns=['alert'])` : Cette ligne récupère toutes les colonnes sauf **`alert`** comme caractéristiques pour l'entraînement.
- `y = df_known['alert']` : Cette ligne récupère la colonne **`alert`** comme cible (label) pour l'entraînement du modèle.

***• Entraînement du modèle Random Forest :***

- `train_test_split()` : Divise les données en ensembles d'entraînement (80%) et de test (20%).
- `RandomForestClassifier()` : Un classificateur d'ensemble qui utilise plusieurs arbres de décision pour faire des prédictions plus robustes.

- `model.fit(X_train, y_train)` : Entraîne le modèle en utilisant les données d'entraînement.

***• Prédiction des valeurs manquantes de 'alert' :***

- `model.predict(df_missing.drop(columns=['alert']))` : Le modèle prédit les valeurs manquantes de la colonne **`alert`** en utilisant les autres caractéristiques disponibles.

***• Mise à jour du DataFrame :***

- `df.loc[df['alert'].isnull(), 'alert'] = missing_alerts` : Remplace les valeurs manquantes de **`alert`** par les prédictions effectuées par le modèle.

***• Évaluation de l'exactitude du modèle :***

- `accuracy_score(y_test, y_pred)` : Calcule l'exactitude (accuracy) du modèle en comparant les prédictions sur l'ensemble de test avec les valeurs réelles.

## Résultat :

```
Model Accuracy: 98.04%
```

### d. Suppression des colonnes inutilisées avec des valeurs manquantes :

```
#droping inused columns with messing values
# we will use 'longitude' and 'latitude' instead
columns_to_drop = ['continent','country', 'subnational', 'city', 'postcode']
df = df.drop(columns=columns_to_drop)
```

- `columns_to_drop = ['continent','country', 'subnational', 'city', 'postcode']` : Cette liste contient les noms des colonnes qui ne sont pas nécessaires pour l'analyse. Ces colonnes représentent des informations géographiques redondantes.

- **Pourquoi utiliser `longitude` et `latitude` ?**

  - Les colonnes `longitude` et `latitude` fournissent une localisation précise sous forme de coordonnées géographiques.
  - Les colonnes comme `continent`, `country`, OU `city` offrent des informations similaires mais moins précises et souvent inutiles pour l'analyse de modèles de prédiction.

- **Méthode utilisée** :

  - `df = df.drop(columns=columns_to_drop)` : Supprime les colonnes spécifiées dans `columns_to_drop` du DataFrame `df`. Cela allège les données et les rend plus adaptées à l'analyse ou à l'entraînement d'un modèle.

## e. Visualisation des colonnes numériques avec des boxplots :

```
 numerical columns to plot
numeric_cols = ['magnitude', 'felt', 'cdi', 'mmi', 'sig', 'nst', 'dmin','rms', 'gap', 'depth', 'latitude', 'longitude','distanceKM']

# Set up the figure size
plt.figure(figsize=(20, 15))

# Create subplots for each column
for i, col in enumerate(numeric_cols):
    plt.subplot(4, 4, i + 1)  # 4x4 grid of subplots
    sns.boxplot(y=df[col], color='lightblue')
    plt.title(f"Boxplot of {col}")
    plt.tight_layout()  # Adjust layout to avoid overlap

plt.show()
```

- **Définition des colonnes numériques** :

  - ***numeric_cols*** : Une liste de colonnes contenant des données numériques pertinentes pour l'analyse, telles que **magnitude**, **depth**, **latitude**, **longitude**, et d'autres. Ces colonnes représentent les mesures clés liées aux tremblements de terre.

- **Configuration de la figure globale** :

  - ***plt.figure(figsize=(20, 15))*** : Définit la taille globale de la figure pour afficher plusieurs graphiques avec suffisamment d'espace pour éviter la surcharge visuelle.

- **Création des sous-graphiques (subplots)** :

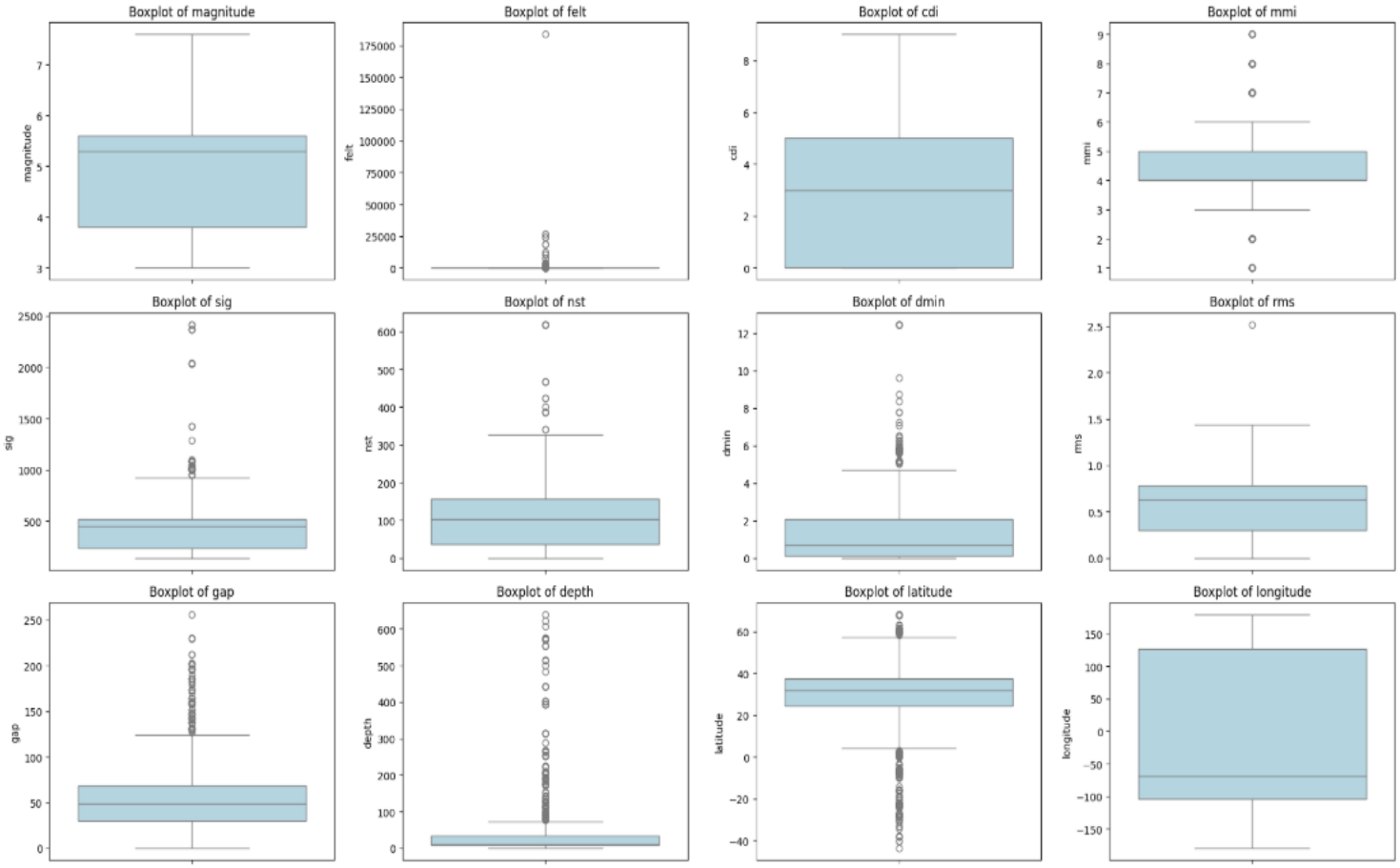
  - ***plt.subplot(4, 4, i + 1)*** : Divise la figure en une grille de 4x4 sous-graphiques (16 graphiques au maximum). Chaque graphique est positionné en fonction de l'index **i**.
  - ***sns.boxplot(y=df[col], color='lightblue')*** : Crée un graphique en boîte (boxplot) pour visualiser la distribution des valeurs et détecter les valeurs aberrantes (outliers) pour chaque colonne.
  - ***plt.title(f"Boxplot of {col}")*** : Ajoute un titre dynamique pour identifier chaque graphique par le nom de la colonne.
  - ***plt.tight_layout()*** : Ajuste automatiquement l'espacement entre les sous-graphiques pour éviter qu'ils ne se chevauchent.

- **Affichage final des graphiques** :

  - ***plt.show()*** : Affiche la figure contenant tous les boxplots.

- **Utilité**

  - **Détection des valeurs aberrantes** : Les boxplots mettent en évidence les valeurs anormalement élevées ou basses (outliers) dans les colonnes numériques.
  - **Analyse des distributions** : Visualiser les tendances, symétries, et plages de valeurs des colonnes numériques.




# 4. Feature Engineering :

## A. Détection des Valeurs Aberrantes à l'aide d'IQR :

La détection des valeurs aberrantes (outliers) est une étape essentielle pour garantir la qualité des données avant leur utilisation dans les modèles de machine learning.

```
# Loop through each column to detect outliers

for col in numeric_cols:
    Q1 = df[col].quantile(0.25)   # 25th percentile
    Q3 = df[col].quantile(0.75)   # 75th percentile
    IQR = Q3 - Q1                 # Interquartile range

    # Define the upper and lower bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Filter and display rows with outliers
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    print(f"Column '{col}' has {len(outliers)} outliers.")
```

**Calcul des Quartiles** :

- **Q1 (1er quartile)** : Correspond à la valeur en dessous de laquelle se trouve 25% des données de la colonne.
- **Q3 (3e quartile)** : Correspond à la valeur en dessous de laquelle se trouve 75% des données.
- **IQR (Interquartile Range)** : Représente l'étendue de la plage moyenne des données, calculée comme **IQR = Q3 - Q1**.

- **Définition des bornes pour les valeurs aberrantes** :

  - **lower_bound = Q1 - 1.5 * IQR** : Toute valeur inférieure à cette limite est considérée comme une valeur aberrante.
  - **upper_bound = Q3 + 1.5 * IQR** : Toute valeur supérieure à cette limite est également une valeur aberrante.

- **Filtrage des outliers** :

  - **df[(df[col] < lower_bound) | (df[col] > upper_bound)]** : Identifie les lignes dans lesquelles la valeur de la colonne est inférieure à la borne inférieure ou supérieure à la borne supérieure.
  - **len(outliers)** : Compte le nombre de valeurs aberrantes détectées pour la colonne en question.

- **Affichage des résultats** :

  - **print(f"Column '{col}' has {len(outliers)} outliers.")** : Affiche le nom de la colonne et le nombre de valeurs aberrantes détectées.

```
Column 'magnitude' has 0 outliers.
Column 'felt' has 188 outliers.
Column 'cdi' has 0 outliers.
Column 'mmi' has 186 outliers.
Column 'sig' has 40 outliers.
Column 'nst' has 16 outliers.
Column 'dmin' has 59 outliers.
Column 'rms' has 1 outliers.
Column 'gap' has 65 outliers.
Column 'depth' has 130 outliers.
Column 'latitude' has 165 outliers.
Column 'longitude' has 0 outliers.
Column 'distanceKM' has 112 outliers.
```

## B. Suppression des valeurs aberrantes avec la méthode de l'IQR

```
#Removing Outliers

for col in numeric_cols:
    Q1 = df[col].quantile(0.25)  # Premier quartile
    Q3 = df[col].quantile(0.75)  # Troisième quartile
    IQR = Q3 - Q1                # Écart interquartile
    lower_bound = Q1 - 1.5 * IQR  # Limite inférieure
    upper_bound = Q3 + 1.5 * IQR  # Limite supérieure
    # Filtrer les outliers
    df[col] = np.where(df[col] < lower_bound, lower_bound, df[col])
    df[col] = np.where(df[col] > upper_bound, upper_bound, df[col])
```

- **Détection des bornes des valeurs aberrantes (outliers)** :

- ***Q1 et Q3*** : Calcul des quartiles pour déterminer la plage interquartile (IQR), qui correspond à la plage centrale des données.
- ***IQR = Q3 - Q1*** : Mesure de l'étendue entre le 1er et le 3e quartile, excluant les valeurs extrêmes.

**Bornes inférieure et supérieure** :

- ***lower_bound = Q1 - 1.5 * IQR*** : Toute valeur inférieure à cette limite est un outlier.
- ***upper_bound = Q3 + 1.5 * IQR*** : Toute valeur supérieure à cette limite est également un outlier.

**Remplacement des valeurs aberrantes** :

- ***np.where(df[col] < lower_bound, lower_bound, df[col])*** : Remplace les valeurs inférieures à la borne inférieure par cette borne.
- ***np.where(df[col] > upper_bound, upper_bound, df[col])*** : Remplace les valeurs supérieures à la borne supérieure par cette borne.

**Raisons du remplacement plutôt que de la suppression** :

- La suppression peut entraîner une perte importante de données, surtout dans des ensembles de données déjà limités.
- Le remplacement maintient la cohérence tout en réduisant l'impact des valeurs aberrantes sur les modèles ou analyses ultérieures.

## C. Vérification des valeurs aberrantes après suppression

```
# Vérificaiton

# Loop through each column to detect outliers
for col in numeric_cols:
    Q1 = df[col].quantile(0.25)  # 25th percentile
    Q3 = df[col].quantile(0.75)  # 75th percentile
    IQR = Q3 - Q1                # Interquartile range

    # Define the upper and lower bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Filter and display rows with outliers
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    print(f"Column '{col}' has {len(outliers)} outliers.")
```

```
Column 'magnitude' has 0 outliers.
Column 'felt' has 0 outliers.
Column 'cdi' has 0 outliers.
Column 'mmi' has 0 outliers.
Column 'sig' has 0 outliers.
Column 'nst' has 0 outliers.
Column 'dmin' has 0 outliers.
Column 'rms' has 0 outliers.
Column 'gap' has 0 outliers.
Column 'depth' has 0 outliers.
Column 'latitude' has 0 outliers.
Column 'longitude' has 0 outliers.
Column 'distanceKM' has 0 outliers.
```

❖ **Analyse Descriptive avec df.describe() Après la Suppression des Valeurs Aberrantes**

| | magnitude | felt | cdi | mmi | sig | nst | dmin | rms | gap | depth | latitude | longitude |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 | 1137.000000 |
| mean | 4.856675 | 15.542656 | 2.925242 | 4.291117 | 419.161829 | 113.376429 | 1.267328 | 0.585077 | 52.735585 | 23.368253 | 29.859733 | -3.930635 |
| std | 1.047840 | 22.622596 | 2.562707 | 1.172657 | 191.696430 | 85.595187 | 1.435717 | 0.304383 | 30.685343 | 23.703139 | 13.963580 | 118.043697 |
| min | 3.000000 | 0.000000 | 0.000000 | 2.500000 | 138.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | -0.250000 | 4.242100 | -179.807000 |
| 25% | 3.800000 | 0.000000 | 0.000000 | 4.000000 | 234.000000 | 37.000000 | 0.100000 | 0.300000 | 30.000000 | 7.550000 | 24.195400 | -104.452000 |
| 50% | 5.300000 | 2.000000 | 3.000000 | 4.000000 | 449.000000 | 102.000000 | 0.680000 | 0.630000 | 49.000000 | 10.000000 | 31.667700 | -68.682000 |
| 75% | 5.600000 | 24.000000 | 5.000000 | 5.000000 | 518.000000 | 157.000000 | 2.061000 | 0.780000 | 68.000000 | 34.723000 | 37.497600 | 126.628000 |
| max | 7.600000 | 60.000000 | 9.000000 | 6.500000 | 944.000000 | 337.000000 | 5.002500 | 1.500000 | 125.000000 | 75.482500 | 57.450900 | 179.972000 |

## D. Encodage Ordinal pour la Colonne "alert"

```python
#alert (ordinale encoding)
from sklearn.preprocessing import OrdinalEncoder

# Define the order of the categories
categories = [['green', 'yellow', 'orange', 'red']]

# Initialize the OrdinalEncoder with the categories
encoder = OrdinalEncoder(categories=categories)

# Reshape the 'alert' column and fit the encoder
df['alert'] = encoder.fit_transform(df[['alert']])

# Display the updated
df['alert']
```

**Objectif** :
La colonne **alert** contient des valeurs catégorielles ordinales (par exemple, green, yellow, orange, red) représentant des niveaux d'alerte croissants.

- **Problème** : Les algorithmes de machine learning ne peuvent pas utiliser directement les données textuelles.
- **Solution** : Convertir ces catégories en valeurs numériques tout en respectant l'ordre logique.

**Utilisation de OrdinalEncoder** :

- *categories* : Permet de spécifier l'ordre des catégories pour garantir une correspondance correcte.
  Exemple :
  - **green** → 0
  - **yellow** → 1
  - **orange** → 2
  - **red** → 3
- *fit_transform* :
  - Entraîne l'encodeur sur les données (dans ce cas, sur la colonne alert).
  - Transforme les catégories en valeurs numériques selon l'ordre défini.

**Mise en forme des données** :

- *df[['alert']]* : La méthode fit_transform attend une entrée en deux dimensions. C'est pourquoi la colonne alert est convertie en DataFrame (et non en Series).
- La colonne mise à jour est ensuite réassignée au DataFrame principal.

```
0          0.0
1          0.0
2          0.0
3          0.0
4          0.0
          ...
1132       2.0
1133       1.0
1134       2.0
1135       0.0
1136       0.0
Name: alert, Length: 1137, dtype: float64
```

## E. Matrice de corrélation :

```
# Calcul de la matrice de corrélation pour les colonnes numériques
correlation_matrix = df[['magnitude'] + columns].corr()
```

**Objectif** :
Calculer la matrice de corrélation pour évaluer les relations linéaires entre les colonnes numériques dans le DataFrame.

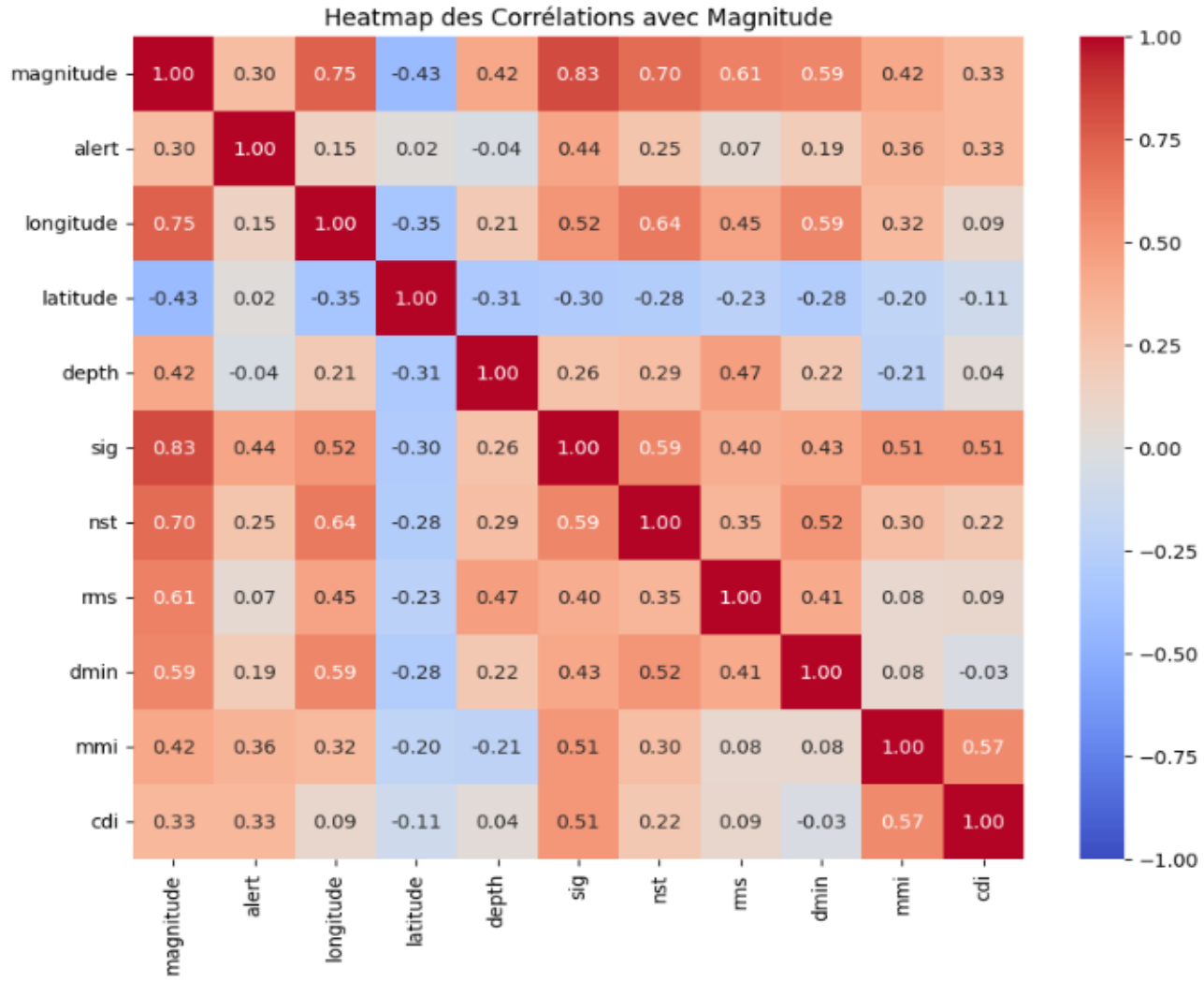
**Étapes principales** :

- **Sélection des colonnes** :
  - Les colonnes utilisées pour le calcul comprennent **magnitude** et toutes les colonnes listées dans la variable **columns**.
  - Exemple : **['magnitude', 'felt', 'cdi', ...]**
- **Matrice de corrélation** :
  - **corr()** : Fonction qui calcule le coefficient de corrélation de Pearson pour chaque paire de colonnes numériques.
  - Résultat : Une matrice carrée où chaque cellule représente le coefficient de corrélation entre deux colonnes.

```
# Création de la heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', vmin=-1, vmax=1)
plt.title('Heatmap des Corrélations avec Magnitude')
plt.show()
```

- **Objectif** :

Visualiser la matrice de corrélation à l'aide d'une **heatmap** (carte thermique) pour mieux comprendre les relations entre les colonnes numériques.




# II. Entraînement du Modèle :

L'étape d'entraînement du modèle est cruciale dans le processus de machine learning, car elle permet au modèle d'apprendre à partir des données disponibles. En utilisant un ensemble de données d'entraînement, le modèle ajuste ses paramètres internes pour minimiser l'erreur entre les prédictions et les valeurs réelles. Cette étape établit les bases nécessaires pour que le modèle généralise efficacement ses prédictions sur de nouvelles données. Dans le contexte de ce projet, l'entraînement du modèle permet de prédire avec précision la magnitude des tremblements de terre en exploitant les relations entre les différentes caractéristiques des données.

## a- Division des données en ensembles d'entraînement et de test :

```
# 5. Division des données en ensembles d'entrainement et de test
X = df.drop('magnitude', axis=1)
y = df['magnitude']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- **Objectif** :
  Diviser les données en deux ensembles :
  - **Ensemble d'entraînement** pour entraîner le modèle.
  - **Ensemble de test** pour évaluer les performances du modèle.
- **Étapes principales** :
  - **Séparation des caractéristiques (features) et de la cible (target)** :
    - *X = df.drop('magnitude', axis=1)* : contient toutes les colonnes sauf **magnitude**, qui représente les caractéristiques utilisées pour prédire la cible.
    - *y = df['magnitude']* : contient uniquement la colonne **magnitude**, qui est la variable cible.
  - **Division des données** :
    - *train_test_split* : Fonction utilisée pour diviser les données.
    - *test_size=0.2* : 20 % des données sont réservées pour l'ensemble de test, les 80 % restants sont utilisés pour l'entraînement.
    - *random_state=42* : Assure que la division est reproductible (les mêmes données seront assignées à chaque ensemble à chaque exécution).
- **Résultat** :
  - *X_train* : Caractéristiques pour l'entraînement.
  - *X_test* : Caractéristiques pour le test.
  - *y_train* : Cible pour l'entraînement.
  - *y_test* : Cible pour le test.

## b- Construction et entraînement des modèles :

```
# 6. Construction et entrainement des modèles

# Initializing the RandomForestRegressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Fitting the model to the training data
rf_model.fit(X_train, y_train)

from sklearn.ensemble import GradientBoostingRegressor
# Initializing the GradientBoostingRegressor
gb_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)

# Fitting the model to the training data
gb_model.fit(X_train, y_train)
```

- **Objectif** :
  Construire et entraîner deux modèles de régression pour prédire la magnitude des tremblements de terre :
  - *RandomForestRegressor*.
  - *GradientBoostingRegressor*.

- **Détails des modèles** :
  - ***Random Forest Regressor :***
    - **RandomForestRegressor(n_estimators=100, random_state=42)** :
      - Utilise une forêt aléatoire composée de **100 arbres de décision**.
      - **random_state=42** : Garantit la reproductibilité des résultats.
    - **rf_model.fit(X_train, y_train)** : Entraîne le modèle sur l'ensemble d'entraînement.
  - ***Gradient Boosting Regressor*** :
    - ***GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)*** :
      - Utilise 100 itérations pour optimiser la performance avec un **taux d'apprentissage** de **0.1**.
      - Combine plusieurs modèles faibles (arbres de décision) en un modèle robuste.
    - ***gb_model.fit(X_train, y_train)*** : Entraîne le modèle sur l'ensemble d'entraînement.
- **Différences entre les modèles** :
  - ***Random Forest*** : Entraîne chaque arbre indépendamment et moyenne leurs prédictions.
  - ***Gradient Boosting*** : Entraîne les arbres de manière séquentielle, chaque nouvel arbre corrigeant les erreurs des précédents.

### c- Évaluation des modèles avec les métriques modèles :

```
# Make predictions on the test set
rf_y_pred = rf_model.predict(X_test)
gb_y_pred = gb_model.predict(X_test)

# Calcul des métriques finales
rf_mae = mean_absolute_error(y_test, rf_y_pred)
rf_mse = mean_squared_error(y_test, rf_y_pred)

gb_mae = mean_absolute_error(y_test, gb_y_pred)
gb_mse = mean_squared_error(y_test, gb_y_pred)

print("Random Forest  - MAE:", rf_mae, "MSE:", r f_mse)
print("Gradient Boosting - MAE:",gb_mae, "MSE:", gb_mse)
```

- **Mean Absolute Error (MAE)** :
  - Mesure la moyenne des écarts absolus entre les prédictions et les valeurs réelles.
  - Une faible valeur indique une meilleure précision du modèle.
- **Mean Squared Error (MSE)** :
  - Évalue l'écart moyen quadratique entre les prédictions et les valeurs réelles.
  - Donne plus de poids aux grandes erreurs.

- **Affichage des résultats** :
  - Les métriques pour chaque modèle sont imprimées pour interprétation.

```
Random Forest  - MAE: 0.028050877192984644 MSE: 0.004458192280701777
Gradient Boosting - MAE: 0.051924132102625926 MSE: 0.007401914631568397
```

# III. Ajustement les paramètres du modèle :

**Objectif de l'optimisation des hyperparamètres** :

- Identifier la meilleure combinaison d'hyperparamètres pour améliorer les performances du modèle.
- Réduire le surapprentissage (overfitting) ou le sous-apprentissage (underfitting).

Pour l'optimisation :

**Validation croisée avec GridSearchCV** :

- Explore systématiquement toutes les combinaisons des hyperparamètres spécifiés .
- Évalue chaque combinaison à l'aide de la validation croisée pour trouver les paramètres optimaux.

## 1- Hyperparamètres pour Random Forest :

```
# 9. Optimisation des hyperparamètres avec validation croisée
from sklearn.model_selection import GridSearchCV
# Hyperparamètres pour Random Forest
rf_param_grid = {
    'n_estimators': [100, 150, 200],        # Nombre d'arbres
    'max_depth': [10, 15, 20],              # Profondeur maximale des arbres
    'min_samples_split': [2, 5, 10],        # Minimum d'échantillons pour diviser un noeud
    'min_samples_leaf': [1, 2, 4]           # Minimum d'échantillons dans une feuille
}
```

**Hyperparamètres définis** :

- **`n_estimators`** : Nombre d'arbres dans la forêt.
- **`max_depth`** : Limite la profondeur maximale de chaque arbre pour contrôler la complexité du modèle.
- **`min_samples_split`** : Nombre minimal d'échantillons requis pour diviser un nœud. Plus la valeur est élevée, plus l'arbre sera contraint.
- **`min_samples_leaf`** : Nombre minimal d'échantillons requis dans une feuille. Cela régule la taille minimale des feuilles.

## 2- Optimisation avec GridSearch pour Random Forest :

```
# Modèle Random Forest
rf_grid_search = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=rf_param_grid,
    scoring='neg_mean_squared_error',     # Critère d'évaluation
    cv=5,                                 # Validation croisée à 5 plis
    n_jobs=-1,                            # Utilisation de tous les cœurs disponibles
    verbose=2                             # Affichage des étapes
)
rf_grid_search.fit(X_train, y_train)
```

- ***RandomForestRegressor(random_state=42) :***
  - Modèle de régression utilisant des forêts aléatoires, initialisé avec une graine pour la reproductibilité.
- ***param_grid :***
  - Définit la grille des hyperparamètres à tester pour optimiser la performance du modèle.
- ***scoring='neg_mean_squared_error' :***
  - Évalue le modèle en calculant l'erreur quadratique moyenne (MSE), puis utilise sa valeur négative comme score (car GridSearchCV maximise le score).
- ***cv=5 :***
  - Effectue une validation croisée à 5 plis, divisant les données en 5 sous-ensembles pour évaluer la robustesse des hyperparamètres.
- ***n_jobs=-1 :***
  - Active l'utilisation de tous les cœurs du processeur pour accélérer les calculs.
- ***verbose=2 :***
  - Ajoute un niveau de détail aux messages d'exécution pour suivre la progression.
- ***rf_grid_search.fit(X_train, y_train) :***
  - Entraîne le modèle pour chaque combinaison d'hyperparamètres dans la grille, en utilisant la validation croisée pour sélectionner la meilleure.

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=   1.2s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=   1.4s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=   1.3s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=   1.2s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=   1.3s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=150; total time=   2.1s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=150; total time=   1.7s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=150; total time=   2.1s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=150; total time=   2.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=150; total time=   2.1s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time=   1.2s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time=   2.2s
```

```python
# Meilleurs hyperparamètres pour Random Forest
print("Meilleurs hyperparamètres pour Random Forest:", rf_grid_search.best_params_)
```

```
Meilleurs hyperparamètres pour Random Forest: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
```

## 3- Hyperparamètres pour Gradient Boosting:

```python
# Hyperparamètres pour Gradient Boosting
gb_param_grid = {
    'n_estimators': [100, 150, 200],       # Nombre d'arbres
    'learning_rate': [0.01, 0.05, 0.1],  # Taux d'apprentissage
    'max_depth': [3, 4, 5],              # Profondeur maximale des arbres
    'subsample': [0.8, 0.9, 1.0]         # Fraction d'échantillons utilisée pour chaque arbre
}
```

## 4- Grid Search pour le Modèle Gradient Boosting :

```
from sklearn.ensemble import GradientBoostingRegressor

# Modèle Gradient Boosting
gb_grid_search = GridSearchCV(
    estimator=GradientBoostingRegressor(random_state=42),
    param_grid=gb_param_grid,
    scoring='neg_mean_squared_error',     # Critère d'évaluation
    cv=5,                                 # Validation croisée à 5 plis
    n_jobs=-1,                            # Utilisation de tous les cœurs disponibles
    verbose=2                             # Affichage des étapes
)
gb_grid_search.fit(X_train, y_train)
```

Ce code permet d'optimiser les hyperparamètres du modèle **Gradient Boosting** en utilisant une recherche par grille avec validation croisée. Cela aide à trouver les meilleurs paramètres pour améliorer la performance du modèle.

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
[CV] END learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time=   0.4s
[CV] END learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time=   0.4s
[CV] END learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time=   0.4s
[CV] END learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.9; total time=   0.4s
[CV] END learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.9; total time=   0.4s
[CV] END learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time=   0.5s
[CV] END learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.8; total time=   0.5s
[CV] END learning_rate=0.01, max_depth=3, n_estimators=100, subsample=0.9; total time=   0.5s
```

```
# Meilleurs hyperparamètres pour Gradient Boosting
print("Meilleurs hyperparamètres pour Gradient Boosting:", gb_grid_search.best_params_)
```

```
Meilleurs hyperparamètres pour Gradient Boosting: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 200, 'subsample': 0.8}
```

## 5- Évaluation finale des modèles avec les meilleurs hyperparamètres :

```
best_rf_model = rf_grid_search.best_estimator_
best_gb_model = gb_grid_search.best_estimator_
```

- ***rf_grid_search.best_estimator_*** :

Après avoir effectué la recherche par grille pour le modèle **Random Forest**, cette commande permet de récupérer le modèle avec les meilleurs hyperparamètres trouvés lors de la recherche. Il est maintenant prêt à être utilisé pour prédire sur de nouvelles données ou évaluer sa performance finale.

- ***gb_grid_search.best_estimator_*** :

De manière similaire, cette commande récupère le meilleur modèle **Gradient Boosting** trouvé par la recherche par grille, en utilisant les hyperparamètres optimaux pour améliorer la performance du modèle

## 6- Prédictions finales:

```
# Prédictions finales
best_rf_preds = best_rf_model.predict(X_test)
best_gb_preds = best_gb_model.predict(X_test)
```

- ***best_rf_model.predict(X_test)*** :

Cette ligne effectue des **prédictions finales** sur l'ensemble de test (X_test) en utilisant le modèle **Random Forest** optimisé (le meilleur modèle trouvé par la recherche par grille). Les résultats sont stockés dans **best_rf_preds**.

 ***best_gb_model.predict(X_test)*** :

De manière similaire, cette ligne génère des prédictions à l'aide du modèle **Gradient Boosting** optimisé. Les résultats sont stockés dans **best_gb_preds.**

## 7- Calcul des métriques finales pour les modèles optimisés:

```
# Calcul des métriques finales
best_rf_mae = mean_absolute_error(y_test, best_rf_preds)
best_rf_mse = mean_squared_error(y_test, best_rf_preds)

best_gb_mae = mean_absolute_error(y_test, best_gb_preds)
best_gb_mse = mean_squared_error(y_test, best_gb_preds)

print("Random Forest après optimisation - MAE:", best_rf_mae, "MSE:", best_rf_mse)
print("Gradient Boosting après optimisation - MAE:", best_gb_mae, "MSE:", best_gb_mse)
```

 ***mean_absolute_error(y_test, best_rf_preds) et mean_squared_error(y_test, best_rf_preds)*** :

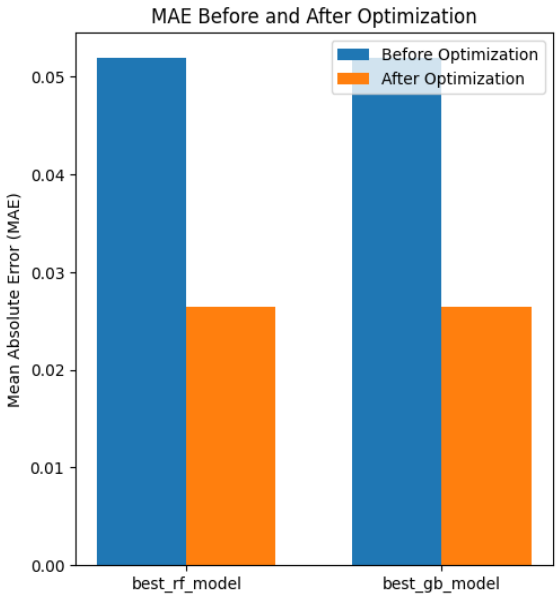
Ces fonctions calculent respectivement l'**erreur absolue moyenne (MAE)** et l'**erreur quadratique moyenne (MSE)** pour les prédictions de **Random Forest** (best_rf_preds) par rapport aux vraies valeurs de l'ensemble de test (y_test).

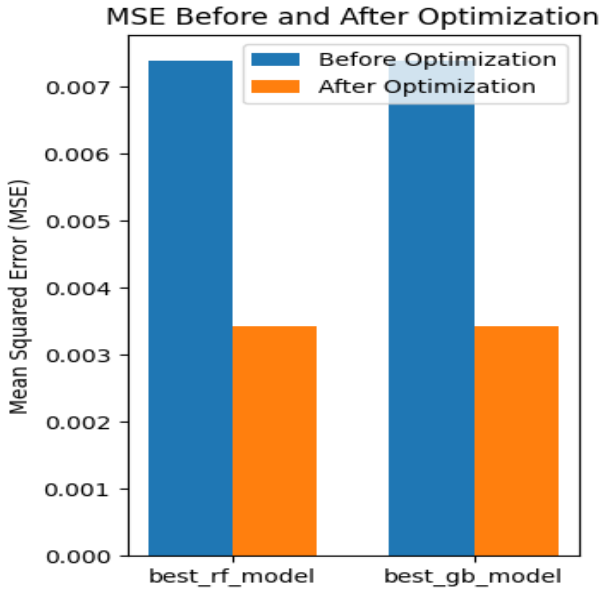
 ***mean_absolute_error(y_test, best_gb_preds) et mean_squared_error(y_test, best_gb_preds)*** :

De même, ces fonctions calculent la **MAE** et la **MSE** pour les prédictions du modèle **Gradient Boosting** (best_gb_preds).

```
Random Forest après optimisation - MAE: 0.028050877192984644 MSE: 0.004458192280701777
Gradient Boosting après optimisation - MAE: 0.026507854790826253 MSE: 0.0034243183263576583
```

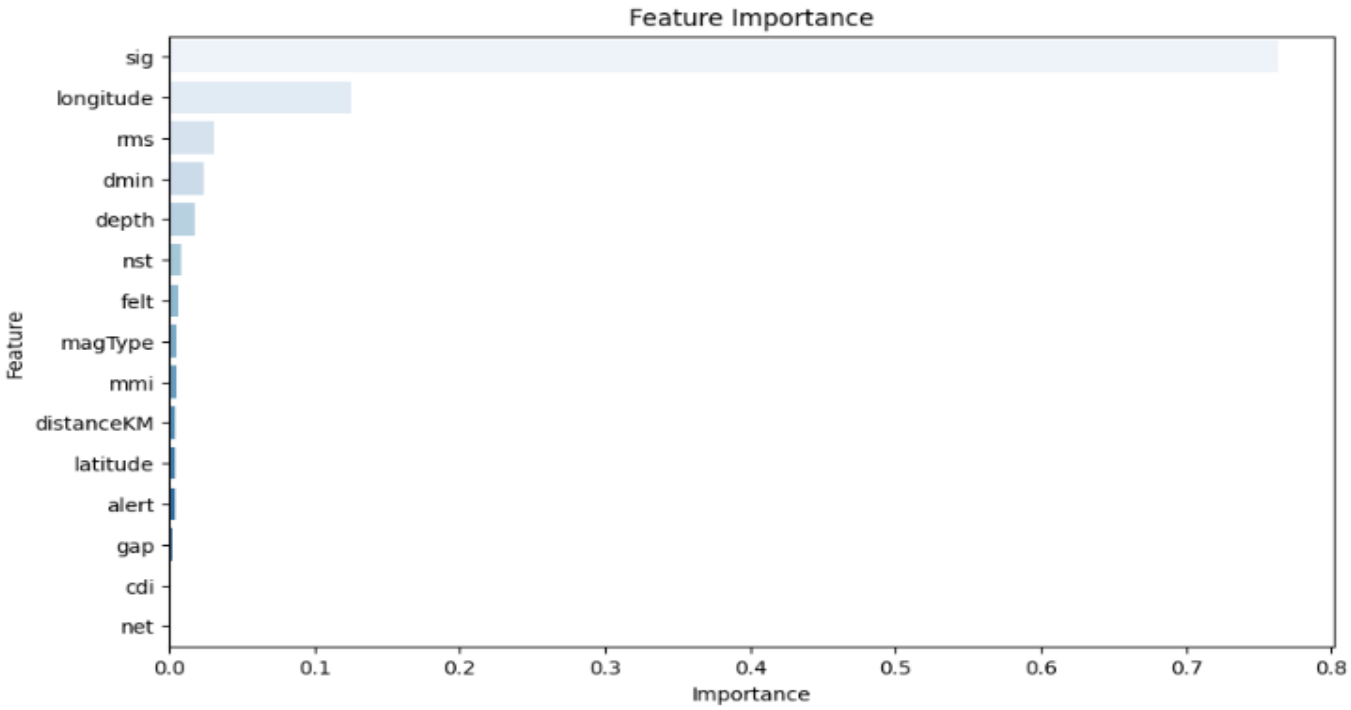
## 8- Comparaison entre MAE et MSE avant et après l'optimisation:



# IV. Choisir l'algorithme convenable :

## 1. Feature Importance :

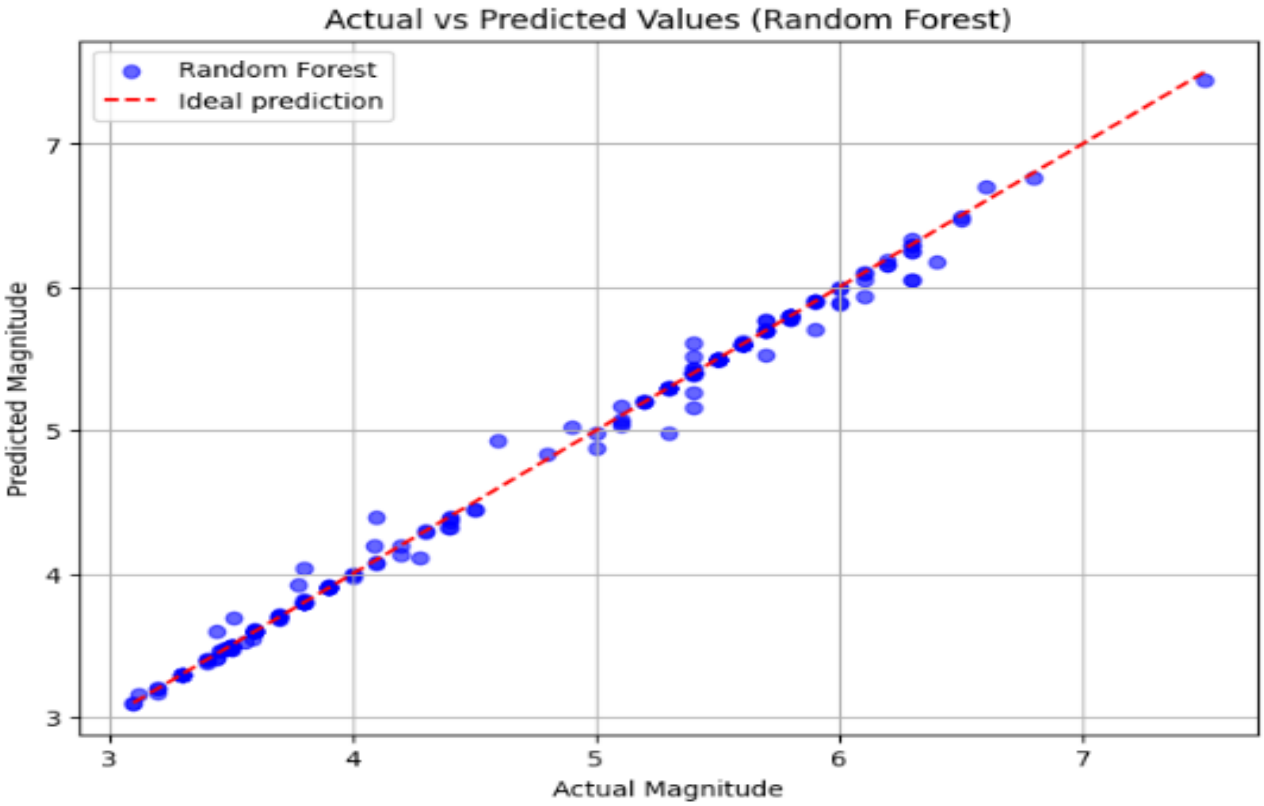



## 2. Graphiques de comparaison entre valeurs réelles et prédites :

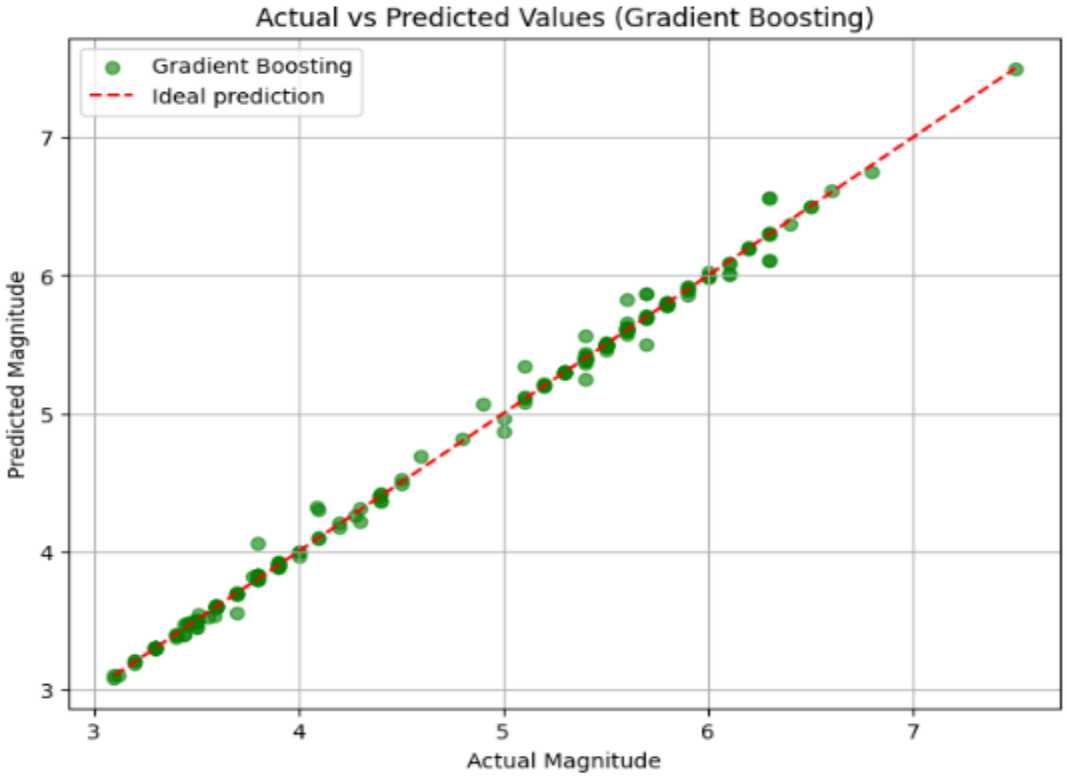
```
# Scatter plot for Random Forest model
plt.figure(figsize=(8, 6))
plt.scatter(y_test, best_rf_preds, color='blue', label='Random Forest', alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', label='Ideal prediction')

plt.title('Actual vs Predicted Values (Random Forest)')
plt.xlabel('Actual Magnitude')
plt.ylabel('Predicted Magnitude')
plt.legend()
plt.grid(True)
plt.show()

# Scatter plot for Gradient Boosting model
plt.figure(figsize=(8, 6))
plt.scatter(y_test, best_gb_preds, color='green', label='Gradient Boosting', alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', label='Ideal prediction')

plt.title('Actual vs Predicted Values (Gradient Boosting)')
plt.xlabel('Actual Magnitude')
plt.ylabel('Predicted Magnitude')
plt.legend()
plt.grid(True)
plt.show()
```

Les graphiques de type **scatter plot** (nuage de points) permettent de visualiser la performance des modèles en comparant les valeurs réelles (magnitude réelle) aux valeurs prédites (magnitude prédite).

- **Pour Random Forest :**




- **Pour Gradient Boosting :**




## 3. L'algorithme convenable :

```python
# Sélection du meilleur modèle
if best_rf_mae < best_gb_mae:
    print("Le modèle Random Forest optimisé est le meilleur avec des hyperparamètres :", rf_grid_search.best_params_)
else:
    print("Le modèle Gradient Boosting optimisé est le meilleur avec des hyperparamètres :", gb_grid_search.best_params_)
```

```
Le modèle Gradient Boosting optimisé est le meilleur avec des hyperparamètres : {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 200, 'subsample': 0.8}
```