



# Rapport Technique : Statistical Learning

## Mini-Projet : Prédiction de la Qualité de l'Air

Master BDIOT 2025-2026  
École : HIET Casablanca

*Réalisé par:*  
**Amine Hamouchi**  
**Bendahmane Ghoufrane**

*Encadré par:*  
**Pr. Aziz LMAKRI**

12 décembre 2025

# Table des matières

|          |                                                                           |           |
|----------|---------------------------------------------------------------------------|-----------|
| <b>1</b> | <b>Abstract(Résumer)</b>                                                  | <b>4</b>  |
| <b>2</b> | <b>Introduction</b>                                                       | <b>4</b>  |
| 2.1      | Contexte et importance de la qualité de l'air . . . . .                   | 4         |
| 2.1.1    | Impacts environnementaux et sanitaires . . . . .                          | 4         |
| 2.1.2    | Rôle de la prédiction dans la Smart City . . . . .                        | 4         |
| 2.2      | Problématique . . . . .                                                   | 5         |
| 2.3      | Objectifs du projet . . . . .                                             | 5         |
| <b>3</b> | <b>Analyse exploratoire des données</b>                                   | <b>6</b>  |
| 3.1      | Contexte et Objectifs . . . . .                                           | 6         |
| 3.2      | Description du Jeu de Données . . . . .                                   | 6         |
| 3.3      | Visualisations des Données . . . . .                                      | 6         |
| 3.3.1    | Détection de valeurs manquantes ou aberrantes . . . . .                   | 6         |
| <b>4</b> | <b>Prétraitement et Feature Engineering</b>                               | <b>8</b>  |
| 4.1      | Nettoyage des Données . . . . .                                           | 8         |
| 4.2      | Visualisations pertinentes . . . . .                                      | 9         |
| 4.2.1    | Visualisation des corrélations spécifiques avec <i>pairplot</i> . . . . . | 10        |
| <b>5</b> | <b>Prétraitement et Feature Engineering</b>                               | <b>13</b> |
| 5.1      | Gestion des Valeurs Aberrantes . . . . .                                  | 13        |
| 5.1.1    | Méthode IQR (Interquartile Range) . . . . .                               | 13        |
| 5.2      | Extraction de Caractéristiques Temporelles . . . . .                      | 14        |
| 5.2.1    | Conversion et Extraction . . . . .                                        | 14        |
| 5.2.2    | Justification des Variables Temporelles . . . . .                         | 14        |
| 5.3      | Suppression de Colonnes Redondantes . . . . .                             | 14        |
| 5.4      | Imputation des Valeurs Manquantes . . . . .                               | 15        |
| 5.5      | Standardisation des Données . . . . .                                     | 15        |
| 5.6      | Division Train/Test . . . . .                                             | 15        |
| <b>6</b> | <b>Modélisation</b>                                                       | <b>16</b> |
| 6.1      | Modèle 1 : Régression Multiple . . . . .                                  | 16        |
| 6.1.1    | Théorie . . . . .                                                         | 16        |
| 6.1.2    | Formulation mathématique . . . . .                                        | 16        |
| 6.1.3    | Implémentation . . . . .                                                  | 16        |
| 6.1.4    | Résultats . . . . .                                                       | 17        |
| 6.2      | Modèle 2 : Random Forest . . . . .                                        | 17        |
| 6.2.1    | Théorie . . . . .                                                         | 17        |
| 6.2.2    | Implémentation . . . . .                                                  | 17        |
| 6.2.3    | Résultats . . . . .                                                       | 18        |
| 6.3      | Random Forest avec Optimisation GridSearchCV . . . . .                    | 18        |
| 6.3.1    | Explications des Hyperparamètres . . . . .                                | 18        |
| 6.3.2    | Implémentation de GridSearchCV . . . . .                                  | 18        |
| 6.4      | Résultats de GridSearchCV . . . . .                                       | 19        |
| 6.4.1    | Meilleurs Hyperparamètres Trouvés . . . . .                               | 19        |
| 6.4.2    | Analyse de Convergence . . . . .                                          | 19        |
| 6.5      | Modèle Optimisé : Performance en Test . . . . .                           | 19        |

|          |                                                                                      |           |
|----------|--------------------------------------------------------------------------------------|-----------|
| 6.6      | Résultats du Random Forest Optimisé . . . . .                                        | 20        |
| <b>7</b> | <b>Évaluation et Interprétation des Résultats</b>                                    | <b>20</b> |
| 7.1      | Métriques de performance . . . . .                                                   | 20        |
| 7.1.1    | Mean Squared Error (MSE) / Erreur quadratique moyenne . . . . .                      | 20        |
| 7.1.2    | Root Mean Squared Error (RMSE) / Racine de l'erreur quadratique<br>moyenne . . . . . | 20        |
| 7.1.3    | Coefficient de détermination ( $R^2$ ) . . . . .                                     | 20        |
| 7.2      | Courbes d'Apprentissage . . . . .                                                    | 21        |
| 7.2.1    | Concept et Objectif . . . . .                                                        | 21        |
| 7.3      | Matrices de Confusion Adaptées à la Régression . . . . .                             | 22        |
| 7.3.1    | Problématique . . . . .                                                              | 22        |
| <b>8</b> | <b>Conclusion</b>                                                                    | <b>23</b> |
| 8.1      | Résumé des Résultats Clés . . . . .                                                  | 23        |
| 8.2      | Recommandations Pratiques . . . . .                                                  | 23        |
| 8.3      | Perspectives d'Amélioration . . . . .                                                | 23        |
| 8.3.1    | Court Terme (1-2 semaines) . . . . .                                                 | 23        |
| 8.3.2    | Moyen Terme (1-3 mois) . . . . .                                                     | 23        |
| 8.3.3    | Long Terme (3-12 mois) . . . . .                                                     | 23        |
| <b>9</b> | <b>Conclusion Générale</b>                                                           | <b>24</b> |

# 1 Abstract(Résumer)

Ce projet aborde la problématique de la surveillance de la qualité de l'air en proposant des modèles prédictifs pour estimer les concentrations de monoxyde de carbone (CO) et de dioxyde d'azote (NO<sub>2</sub>) à partir de données de capteurs (UCI Air Quality Dataset). La méthodologie adoptée inclut un prétraitement rigoureux des données (imputation, gestion des valeurs aberrantes) et la comparaison de deux algorithmes : la Régression Linéaire et le Random Forest Regressor. Les résultats démontrent que si les deux modèles offrent des performances similaires pour le CO ( $R^2 \approx 0.90$ ), le Random Forest surpasse significativement l'approche linéaire pour le NO<sub>2</sub> ( $R^2$  de 0.917 contre 0.788). En conclusion, ce travail met en évidence la nécessité d'utiliser des méthodes non-linéaires robustes pour modéliser efficacement les interactions complexes entre les réponses des capteurs et les polluants atmosphériques.

## 2 Introduction

### 2.1 Contexte et importance de la qualité de l'air

#### 2.1.1 Impacts environnementaux et sanitaires

La pollution atmosphérique représente aujourd'hui l'un des défis environnementaux majeurs à l'échelle mondiale, qualifiée par l'Organisation Mondiale de la Santé (OMS) comme le plus grand risque sanitaire environnemental unique. L'urbanisation rapide et l'industrialisation croissante ont entraîné une augmentation significative des concentrations de polluants atmosphériques, notamment le monoxyde de carbone (CO), le dioxyde d'azote (NO<sub>2</sub>) et le dioxyde de soufre (SO<sub>2</sub>).

Ces substances, lorsqu'elles dépassent certains seuils critiques, engendrent des conséquences dévastatrices non seulement sur l'environnement (pluies acides, eutrophisation, changement climatique), mais surtout sur la santé humaine. L'exposition prolongée à ces polluants est directement corrélée à l'augmentation des maladies respiratoires chroniques, des troubles cardiovasculaires et, dans les cas les plus graves, à une mortalité prématurée. Dès lors, la surveillance de la qualité de l'air n'est plus une simple option, mais une nécessité de santé publique.

#### 2.1.2 Rôle de la prédiction dans la Smart City

Dans le paradigme de la "Smart City" (Ville Intelligente), la gestion de l'environnement urbain repose sur l'exploitation massive des données. L'intégration de capteurs IoT (Internet of Things) permet désormais de collecter des données environnementales en temps réel à une granularité spatio-temporelle inédite.

Cependant, la simple surveillance (monitoring) est réactive et souvent insuffisante pour prévenir les pics de pollution. C'est ici que le *Statistical Learning* et l'intelligence artificielle jouent un rôle pivot. En passant d'une approche descriptive à une approche prédictive, les villes peuvent anticiper les épisodes de pollution critique. L'analyse des données historiques, couplée aux relevés météorologiques, permet de modéliser le comportement des polluants et d'offrir aux décideurs des outils d'aide à la décision pour réguler le trafic ou émettre des alertes sanitaires préventives.

## 2.2 Problématique

La dynamique de la pollution atmosphérique est un phénomène complexe et non-linéaire. Elle ne dépend pas uniquement des sources d'émission (trafic routier, usines), mais est fortement influencée par les conditions météorologiques et chimiques locales. Par exemple, la température influence les réactions chimiques atmosphériques, tandis que l'humidité et le vent jouent un rôle crucial dans la dispersion ou la concentration des polluants.

Le défi technique réside dans la capacité à modéliser ces interactions multidimensionnelles. Les capteurs chimiques (Low-Cost Sensors) utilisés dans les déploiements IoT sont sujets à des dérives et sont sensibles aux variables environnementales. Dès lors, la question centrale qui guide ce projet est la suivante :

*Comment prédire de manière fiable la concentration des polluants majeurs ( $CO$ ,  $NO_2$ ) en s'appuyant sur l'analyse conjointe des réponses des capteurs et des données météorologiques ?*

Il s'agit de déterminer si des algorithmes d'apprentissage statistique peuvent capturer la corrélation entre les variables exogènes (météo) et les concentrations cibles pour fournir une estimation précise.

## 2.3 Objectifs du projet

Pour répondre à cette problématique, ce projet se fixe quatre objectifs principaux, allant de l'analyse exploratoire à la validation des modèles :

- **Comprendre les relations entre météo et pollution** : Il s'agit d'effectuer une analyse exploratoire approfondie (EDA) pour visualiser les interactions entre les variables météorologiques (Température, Humidité Relative, Humidité Absolue) et les concentrations de polluants. Nous chercherons à identifier des corrélations significatives ou des motifs saisonniers/journaliers.
- **Construire et comparer des modèles de régression** : Nous mettrons en œuvre et comparerons deux approches distinctes de l'apprentissage supervisé :
  1. Une approche paramétrique linéaire (Régression Linéaire) pour établir une baseline.
  2. Une approche non-paramétrique et ensembliste (Random Forest Regressor) pour capturer les non-linéarités complexes du phénomène.
- **Identifier les variables les plus influentes** : Au-delà de la simple prédiction, nous analyserons l'importance des caractéristiques (Feature Importance) pour comprendre quels facteurs environnementaux pèsent le plus lourd dans la dégradation de la qualité de l'air.
- **Évaluer la performance et proposer des améliorations** : L'évaluation rigoureuse des modèles se fera via des métriques standards (MSE,  $R^2$ ). Sur la base de ces résultats, nous discuterons des limites de l'étude et proposerons des pistes d'amélioration (optimisation des hyperparamètres, ajout de variables temporelles).

## 3 Analyse exploratoire des données

### 3.1 Contexte et Objectifs

La pollution de l'air est un enjeu majeur de santé publique. Ce projet s'inscrit dans le cadre du module de *Statistical Learning* et vise à développer des modèles prédictifs capables d'estimer la concentration de polluants atmosphériques.

L'objectif spécifique de ce travail (Problem 3) est de prédire les concentrations de monoxyde de carbone (CO(GT)) et de dioxyde d'azote (NO2(GT)) en fonction des données capteurs et des conditions temporelles.

Nous comparerons les performances de deux approches : la **Régression Linéaire** et le **Random Forest Regressor**.

### 3.2 Description du Jeu de Données

Nous utilisons le *UCI Air Qualité Data-set*[cite : 4, 268]. Ce jeu de données contient 9357 instances de réponses moyennes horaires provenant d'un réseau de 5 capteurs chimiques d'oxydes métalliques situés en Italie.

Les variables incluent les concentrations de polluants (CO, NMHC, C6H6, NOx, NO2), les réponses des capteurs (PT08.Sx), ainsi que des variables environnementales (Température T, Humidité Relative RH, Humidité Absolue AH).

### 3.3 Visualisations des Données

#### 3.3.1 Détection de valeurs manquantes ou aberrantes

Pour commencer, nous analysons la structure du jeu de données et détectons les valeurs manquantes ou aberrantes à l'aide de fonctions de base de `pandas`.

```
df.info()
```

Output :

```
Data columns (total 17 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Date                  9357 non-null  object
1   Time                  9357 non-null  object
2   CO(GT)                9357 non-null  float64
3   PT08.S1(CO)           9357 non-null  float64
4   NMHC(GT)              9357 non-null  float64
5   C6H6(GT)              9357 non-null  float64
6   PT08.S2(NMHC)         9357 non-null  float64
7   NOx(GT)               9357 non-null  float64
8   PT08.S3(NOx)          9357 non-null  float64
9   NO2(GT)               9357 non-null  float64
10  PT08.S4(NO2)          9357 non-null  float64
11  PT08.S5(O3)           9357 non-null  float64
12  T                      9357 non-null  float64
13  RH                     9357 non-null  float64
14  AH                     9357 non-null  float64
15  Unnamed: 15            0 non-null     float64
16  Unnamed: 16            0 non-null     float64
dtypes: float64(15), object(2)
```

FIGURE 1 – Résultat de la commande `df.info()`

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

Output :

|               | 0    |
|---------------|------|
| Date          | 114  |
| Time          | 114  |
| CO(GT)        | 114  |
| PT08.S1(CO)   | 114  |
| NMHC(GT)      | 114  |
| C6H6(GT)      | 114  |
| PT08.S2(NMHC) | 114  |
| NOx(GT)       | 114  |
| PT08.S3(NOx)  | 114  |
| NO2(GT)       | 114  |
| PT08.S4(NO2)  | 114  |
| PT08.S5(O3)   | 114  |
| T             | 114  |
| RH            | 114  |
| AH            | 114  |
| Unnamed: 15   | 9471 |
| Unnamed: 16   | 9471 |

FIGURE 2 – Résultat de la commande `df.info()`

```
df.duplicated().sum()
```

Output :

```
np.int64(113)
```

```
df[df.duplicated()]
```

Output :

|      | Date | Time | CO(GT) | PT08.S1(CO) | NMHC(GT) | C6H6(GT) | PT08.S2(NMHC) | NOx(GT) | PT08.S3(NOx) | NO2(GT) | PT08.S4(NO2) | PT08.S5(O3) | T   | RH  | AH  | Unnamed: 15 | Unnamed: 16 |
|------|------|------|--------|-------------|----------|----------|---------------|---------|--------------|---------|--------------|-------------|-----|-----|-----|-------------|-------------|
| 9358 | NaN  | NaN  | NaN    | NaN         | NaN      | NaN      | NaN           | NaN     | NaN          | NaN     | NaN          | NaN         | NaN | NaN | NaN | NaN         | NaN         |
| 9359 | NaN  | NaN  | NaN    | NaN         | NaN      | NaN      | NaN           | NaN     | NaN          | NaN     | NaN          | NaN         | NaN | NaN | NaN | NaN         | NaN         |
| 9360 | NaN  | NaN  | NaN    | NaN         | NaN      | NaN      | NaN           | NaN     | NaN          | NaN     | NaN          | NaN         | NaN | NaN | NaN | NaN         | NaN         |
| 9361 | NaN  | NaN  | NaN    | NaN         | NaN      | NaN      | NaN           | NaN     | NaN          | NaN     | NaN          | NaN         | NaN | NaN | NaN | NaN         | NaN         |
| 9362 | NaN  | NaN  | NaN    | NaN         | NaN      | NaN      | NaN           | NaN     | NaN          | NaN     | NaN          | NaN         | NaN | NaN | NaN | NaN         | NaN         |
| ...  | ...  | ...  | ...    | ...         | ...      | ...      | ...           | ...     | ...          | ...     | ...          | ...         | ... | ... | ... | ...         | ...         |
| 9466 | NaN  | NaN  | NaN    | NaN         | NaN      | NaN      | NaN           | NaN     | NaN          | NaN     | NaN          | NaN         | NaN | NaN | NaN | NaN         | NaN         |
| 9467 | NaN  | NaN  | NaN    | NaN         | NaN      | NaN      | NaN           | NaN     | NaN          | NaN     | NaN          | NaN         | NaN | NaN | NaN | NaN         | NaN         |
| 9468 | NaN  | NaN  | NaN    | NaN         | NaN      | NaN      | NaN           | NaN     | NaN          | NaN     | NaN          | NaN         | NaN | NaN | NaN | NaN         | NaN         |
| 9469 | NaN  | NaN  | NaN    | NaN         | NaN      | NaN      | NaN           | NaN     | NaN          | NaN     | NaN          | NaN         | NaN | NaN | NaN | NaN         | NaN         |
| 9470 | NaN  | NaN  | NaN    | NaN         | NaN      | NaN      | NaN           | NaN     | NaN          | NaN     | NaN          | NaN         | NaN | NaN | NaN | NaN         | NaN         |

113 rows x 17 columns

FIGURE 3 – Résultat de la commande `df.info()`

```
sns.boxplot(data = df)
```

Output :

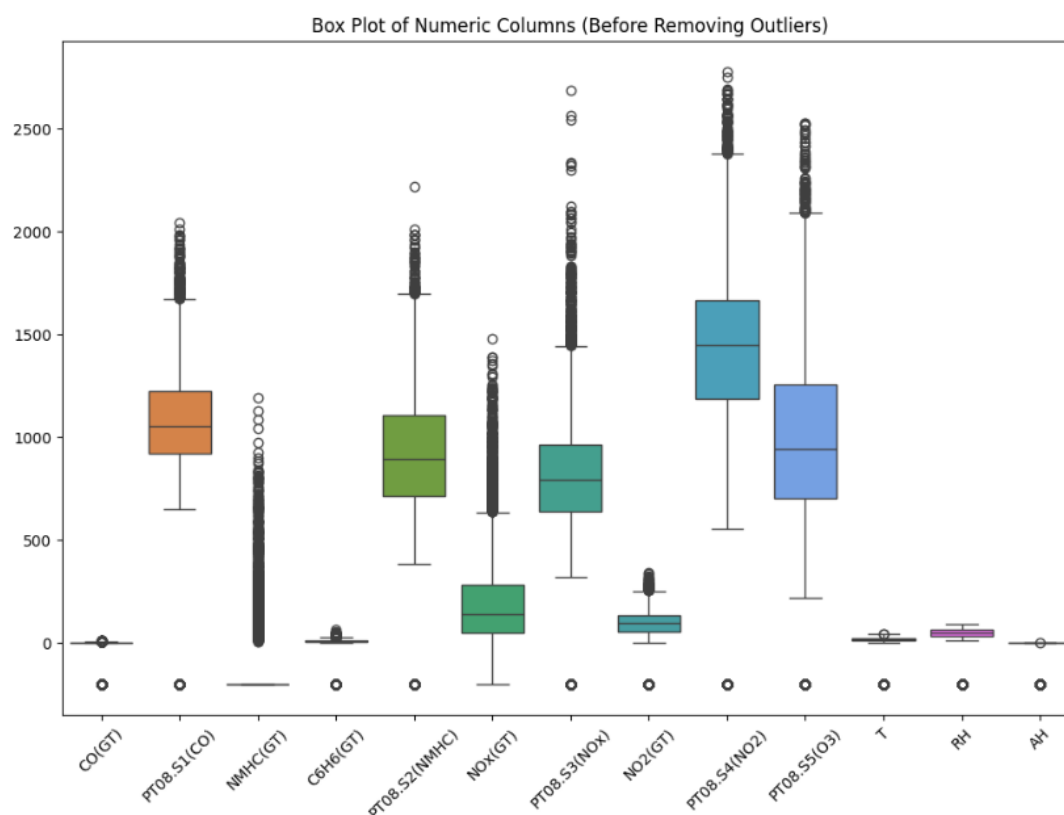


FIGURE 4 – Résultat de la commande `sns.boxplot(data = df)`

## 4 Prétraitement et Feature Engineering

La phase de prétraitement est cruciale pour garantir la qualité des modèles. Les étapes suivantes ont été réalisées en Python (pandas/numpy) :

### 4.1 Nettoyage des Données

- **Suppression des colonnes vides** : Les colonnes `Unnamed: 15` et `Unnamed: 16`, entièrement vides, ont été retirées[cite : 82].
- **Gestion des valeurs manquantes et aberrantes** : Le dataset utilise la valeur `-200` pour marquer les données manquantes. Ces valeurs ont été converties en `NaN`. La colonne `NMHC(GT)`, contenant trop de valeurs manquantes, a été supprimée. Les autres valeurs manquantes ont été imputées par la moyenne de chaque colonne[cite : 1].
- **Suppression des doublons** : Les lignes dupliquées ont été identifiées et retirées.

## 4.2 Visualisations pertinentes

```
import seaborn as sns;
import matplotlib.pyplot as plt;

# 1. Calculate the correlation matrix
correlation_matrix = df_final.corr();

# 2. Create a heatmap of the correlation matrix
plt.figure(figsize=(12, 10));
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f");

# 4. Add a title to the heatmap
plt.title('Correlation Heatmap of Features');

# 5. Display the plot
plt.show();
```

Output :

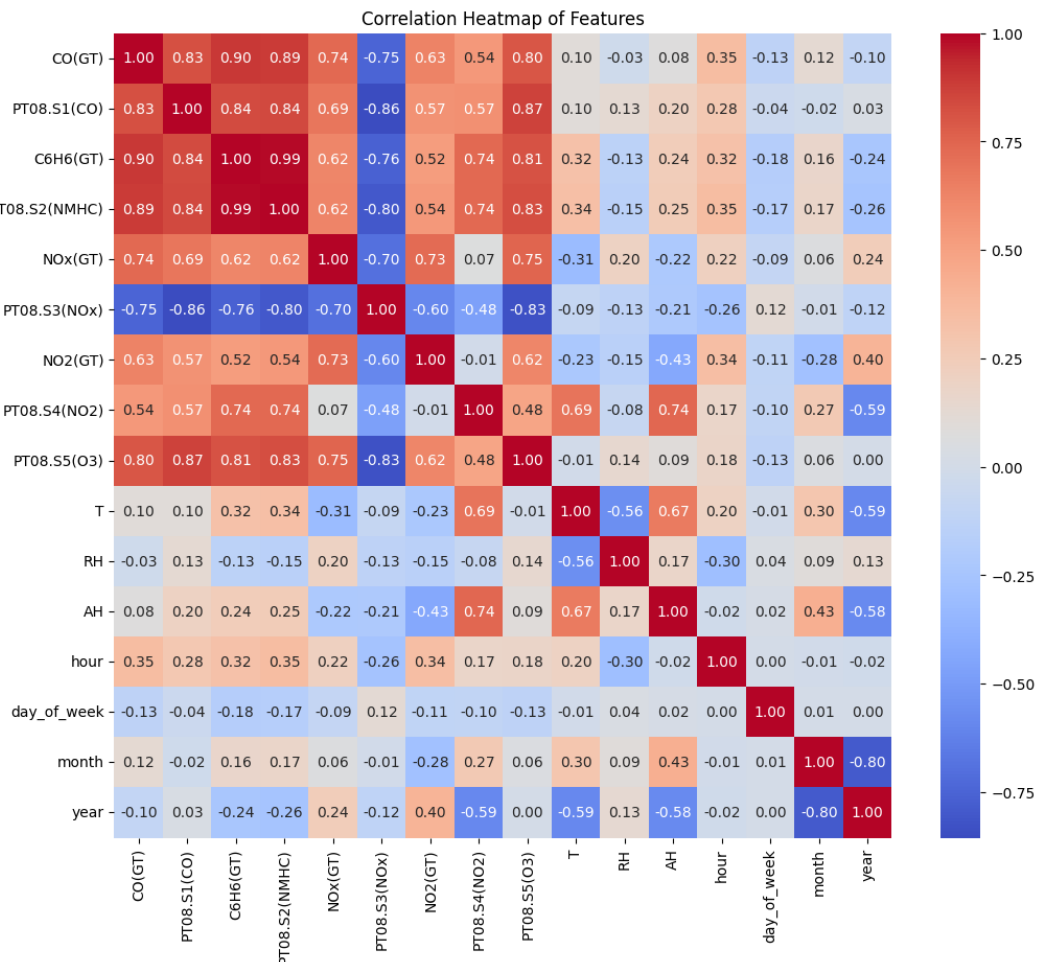


FIGURE 5 – Visualisation de la corrélation entre les données

### 4.2.1 Visualisation des corrélations spécifiques avec *pairplot*

#### 1. NMHC(GT) et C6H6(GT)

Ces deux caractéristiques présentent une corrélation très forte. Nous souhaitons visualiser comment elles varient ensemble afin de préparer les étapes de *feature engineering*.

```
sns.pairplot(df_final[['PT08.S2(NMHC)', 'C6H6(GT)']])  
plt.suptitle('Pairplot entre NMHC(GT) et C6H6(GT)', y=1.02)
```

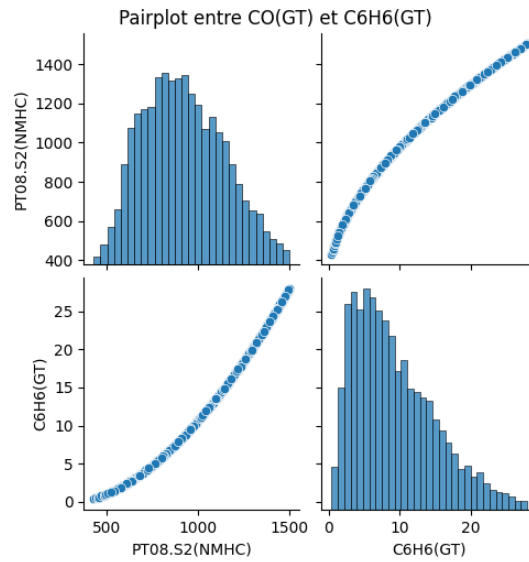


FIGURE 6 – Visualisation de la corrélation entre NMHC(GT) et C6H6(GT)

On peut constater que les deux variables présentent une dispersion identique.

#### 2. CO(GT) et C6H6(GT)

CO(GT) est une variable de sortie présentant une forte corrélation avec C6H6(GT). Cette visualisation est essentielle pour analyser la nature de leur relation (linéaire, groupée, etc.), ce qui peut guider le choix du modèle prédictif. Elle permet également de repérer d'éventuelles valeurs aberrantes dans ces deux variables.

```
sns.pairplot(df_final[['CO(GT)', 'C6H6(GT)']])  
plt.suptitle('Pairplot entre CO(GT) et C6H6(GT)', y=1.02)
```

On peut observer que les deux variables corréleront positivement, mais que leur relation n'est pas parfaitement linéaire.

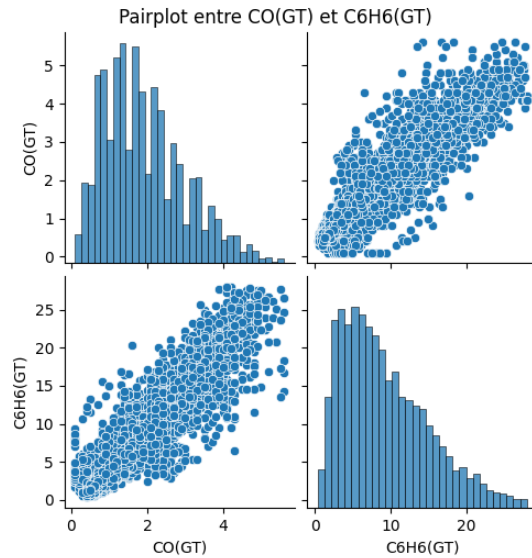


FIGURE 7 – Visualisation de la corrélation entre CO(GT) et C6H6(GT)

### 3. NO<sub>2</sub>(GT) et NO<sub>x</sub>(GT)

NO<sub>2</sub>(GT) est une variable de sortie importante. La visualisation de sa relation avec NO<sub>x</sub>(GT), qui correspond à la somme de NO et NO<sub>2</sub>, est essentielle pour comprendre la dynamique de ces polluants interdépendants. Cette représentation permet également de repérer d'éventuelles valeurs aberrantes et d'analyser comment ces deux variables évoluent conjointement.

```
sns.pairplot(df_final[['NO2(GT)', 'NOx(GT)']])
plt.suptitle('Pairplot entre NO2(GT) et NOx(GT)', y=1.02)
```

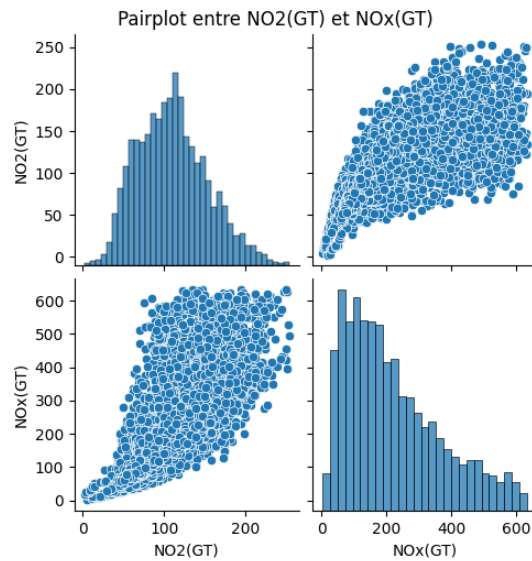


FIGURE 8 – Visualisation de la corrélation entre NO<sub>2</sub>(GT) et NO<sub>x</sub>(GT)

On constate que la relation est non linéaire et que le nuage de points est très diffus,

indiquant une forte dispersion.

#### 4. NO<sub>2</sub>(GT) et PT08.S4(NO<sub>2</sub>)

Nous analysons la relation entre **NO<sub>2</sub>(GT)** la concentration réelle de dioxyde d'azote mesurée par un analyseur de référence (*Ground Truth*) et **PT08.S4(NO<sub>2</sub>)**, la valeur mesurée par le capteur chimique PT08, modèle S4, conçu pour détecter le NO<sub>2</sub>. La corrélation négative de -0,6 suggère une relation inverse, qu'il est intéressant de visualiser pour confirmer la tendance et détecter d'éventuelles non-linéarités.

```
sns.pairplot(df_final[['NO2(GT)', 'PT08.S4(NO2)']]);  
plt.suptitle('Pairplot entre NO2(GT) et PT08.S4(NO2)', y=1.02);
```

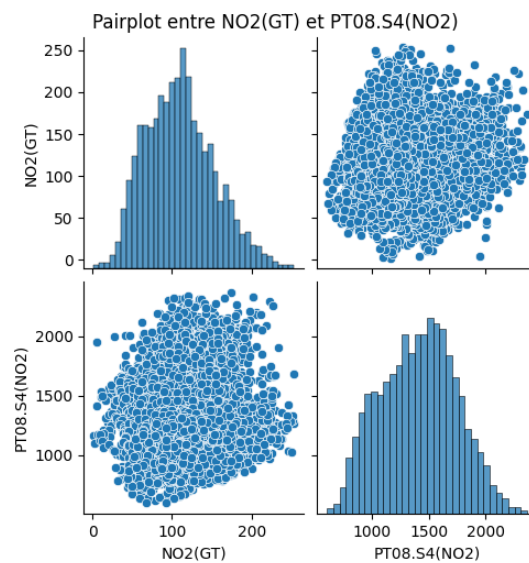


FIGURE 9 – Visualisation de la corrélation entre NO<sub>2</sub>(GT) et PT08.S4(NO<sub>2</sub>)

Le nuage de points est très diffus et large (forte dispersion). Cette grande dispersion indique que ce capteur est peu fiable ou très bruité pour prédire précisément la concentration réelle de NO<sub>2</sub>.

## 5 Prétraitement et Feature Engineering

### 5.1 Gestion des Valeurs Aberrantes

#### 5.1.1 Méthode IQR (Interquartile Range)

Les outliers sont détectés en utilisant la méthode IQR standard. Une valeur est considérée comme aberrante si :

$$x < Q_1 - 1.5 \times IQR \quad \text{ou} \quad x > Q_3 + 1.5 \times IQR \quad (1)$$

où  $Q_1$  et  $Q_3$  sont les premier et troisième quartiles, et  $IQR = Q_3 - Q_1$ .

```
# Détection des outliers
Q1 = df[numeric_columns].quantile(0.25)
Q3 = df[numeric_columns].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Suppression
df_cleaned = df[~((df[numeric_columns] < lower_bound) |
                  (df[numeric_columns] > upper_bound)).any(axis=1)]

print(f"Lignes supprimées: {len(df) - len(df_cleaned)}")
print(f"Nouvelles dimensions: {df_cleaned.shape}")
```

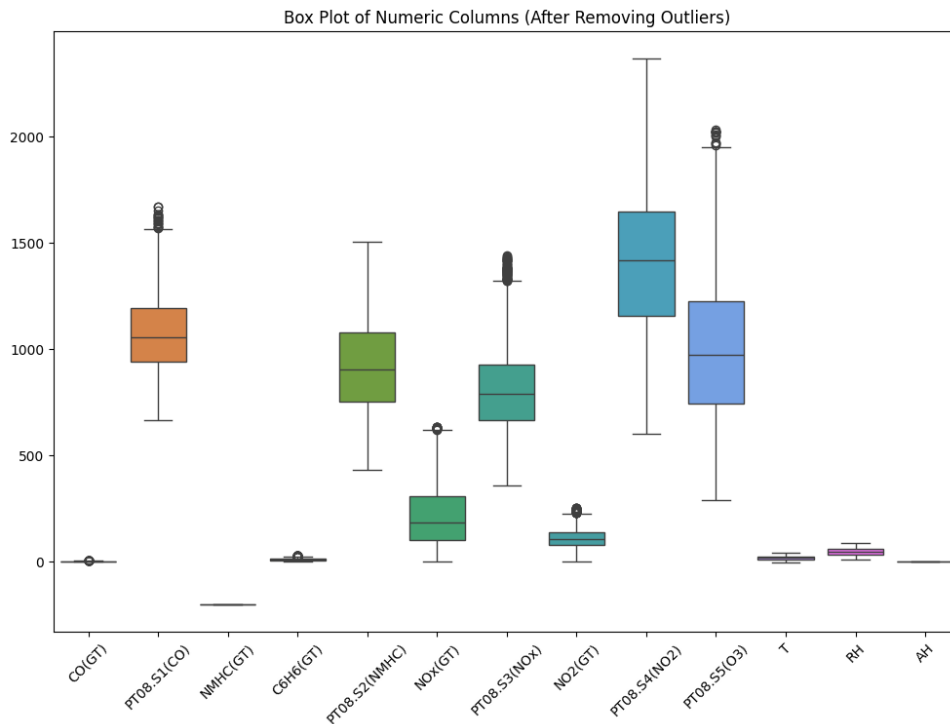


FIGURE 10 – Visualisation des valeurs après la suppression des outliers

## 5.2 Extraction de Caractéristiques Temporelles

### 5.2.1 Conversion et Extraction

Le jeu de données contient des colonnes `Date` et `Time` sous forme de chaînes. Nous convertissons en datetime et extrayons des variables temporelles pertinentes :

```
# Fusion et conversion
df_processed['DateTime'] = pd.to_datetime(
    df_processed['Date'] + ' ' + df_processed['Time']
)

# Extraction de features temporelles
df_processed['hour'] = df_processed['DateTime'].dt.hour # 0-23
df_processed['day_of_week'] = df_processed['DateTime'].dt.dayofweek # 0-6
df_processed['month'] = df_processed['DateTime'].dt.month # 1-12
df_processed['year'] = df_processed['DateTime'].dt.year # année

# Suppression des colonnes temporelles originales
df_final = df_processed.drop(columns=['Date', 'Time', 'DateTime'])

print("Colonnes extraites: hour, day_of_week, month, year")
print(f"Dimensions finales: {df_final.shape}")
```

### 5.2.2 Justification des Variables Temporelles

| Variable    | Justification                                                               |
|-------------|-----------------------------------------------------------------------------|
| hour        | Capture les patterns journaliers (heures de pointe du trafic, cycle diurne) |
| day_of_week | Distingue jours ouvrables et week-ends (différents patterns de trafic)      |
| month       | Capte les variations saisonnières (température, conditions météo)           |
| year        | Détecte les tendances long-terme et l'effet d'interventions                 |

TABLE 1 – Justification du feature engineering temporel

## 5.3 Suppression de Colonnes Redondantes

```
# Suppression de PT08.S4(NO2)
# Justification: corrélation négative peu fiable avec NO2(GT)
df_final = df_final.drop(columns=['PT08.S4(NO2)'])

# Suppression de RH
# Justification: multicollinéarité avec AH (humidité absolue)
df_final = df_final.drop(columns=['RH'])

print(f"Colonnes finales: {list(df_final.columns)}")
print(f"Shape final: {df_final.shape}")
```

## 5.4 Imputation des Valeurs Manquantes

```
# Conversion des -200 en NaN
df_processed = df_cleaned.replace(-200, np.nan)

# Imputation par la moyenne
numeric_columns = df_processed.select_dtypes(include=[np.number]).columns
for col in numeric_columns:
    if df_processed[col].isnull().any():
        df_processed[col].fillna(df_processed[col].mean(), inplace=True)

print(f"NaN après imputation: {df_processed.isnull().sum().sum()}")
```

## 5.5 Standardisation des Données

Pour la Régression Linéaire, nous standardisons les données :

$$z = \frac{x - \mu}{\sigma} \quad (2)$$

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_lr_scaled = scaler.fit_transform(X_train_lr)
X_test_lr_scaled = scaler.transform(X_test_lr)
```

Random Forest n'en a pas besoin (invariant à l'échelle).

## 5.6 Division Train/Test

```
from sklearn.model_selection import train_test_split

# Variables cibles
target_columns = ['CO(GT)', 'NO2(GT)']

# Séparation
X = df_final.drop(columns=target_columns)
y = df_final[target_columns]

# Division 80/20
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print(f"Entraînement: {X_train.shape[0]} samples, {X_train.shape[1]} features")
print(f"Test: {X_test.shape[0]} samples")
```

| Ensemble     | Samples | Variables | Cibles |
|--------------|---------|-----------|--------|
| Entraînement | 7485    | 14        | 2      |
| Test         | 1872    | 14        | 2      |

TABLE 2 – Partition train/test

## 6 Modélisation

### 6.1 Modèle 1 : Régression Multiple

#### 6.1.1 Théorie

La régression multiple est une extension de la régression linéaire simple pour plusieurs variables explicatives (features).

#### 6.1.2 Formulation mathématique

Soit un jeu de données avec  $n$  observations et  $p$  variables explicatives  $x_1, x_2, \dots, x_p$  :

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip} + \epsilon_i, \quad i = 1, 2, \dots, n \quad (3)$$

où :

- $y_i$  : valeur cible pour l'observation  $i$ ,
- $\beta_0$  : intercept (ordonnée à l'origine),
- $\beta_j$  : coefficient associé à la variable  $x_j$  ( $j = 1, \dots, p$ ),
- $\epsilon_i$  : terme d'erreur pour l'observation  $i$ .

En forme matricielle, pour tout le jeu de données :

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \quad (4)$$

où :

- $\mathbf{Y}$  est un vecteur  $n \times 1$  contenant les valeurs cibles,
- $\mathbf{X}$  est une matrice  $n \times (p + 1)$  contenant les variables explicatives avec une colonne de 1 pour l'intercept,
- $\boldsymbol{\beta}$  est un vecteur  $(p + 1) \times 1$  de coefficients,
- $\boldsymbol{\epsilon}$  est un vecteur  $n \times 1$  des termes d'erreur.

#### 6.1.3 Implémentation

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

model_lr = LinearRegression()
model_lr.fit(X_train_lr_scaled, y_train_lr)
y_pred_lr = model_lr.predict(X_test_lr_scaled)

# Évaluation
for i, target in enumerate(['CO(GT)', 'NO2(GT)']):
    mse = mean_squared_error(y_test_lr.iloc[:, i], y_pred_lr[:, i])
    r2 = r2_score(y_test_lr.iloc[:, i], y_pred_lr[:, i])
    rmse = np.sqrt(mse)
    print(f"{target}: R²={r2:.4f}, RMSE={rmse:.4f}")
```

### 6.1.4 Résultats

| Cible                | MSE      | RMSE    | R <sup>2</sup> |
|----------------------|----------|---------|----------------|
| CO(GT)               | 0.3070   | 0.5541  | 0.8145         |
| NO <sub>2</sub> (GT) | 427.2046 | 20.6697 | 0.7809         |

TABLE 3 – Performance Régression Linéaire

## 6.2 Modèle 2 : Random Forest

### 6.2.1 Théorie

Random Forest est un modèle d'ensemble basé sur des arbres de décision. Mathématiquement, il combine plusieurs arbres pour réduire la variance et améliorer la précision.

Soit un jeu de données  $(\mathbf{X}, \mathbf{Y})$  avec  $n$  observations et  $p$  features.

On construit  $B$  arbres de décision  $T_1, T_2, \dots, T_B$ , chacun étant construit sur un échantillon bootstrap de  $(\mathbf{X}, \mathbf{Y})$ .

Régression

Pour la régression, chaque arbre prédit une valeur  $\hat{y}^{(b)}$  pour une observation  $\mathbf{x}$  :

$$\hat{y}^{(b)} = T_b(\mathbf{x}) \quad (5)$$

La prédiction finale de la forêt est la moyenne de toutes les prédictions des  $B$  arbres :

$$\hat{y} = \frac{1}{B} \sum_{b=1}^B \hat{y}^{(b)} \quad (6)$$

### 6.2.2 Implémentation

```
from sklearn.ensemble import RandomForestRegressor

model_rf = RandomForestRegressor(
    n_estimators=100,
    random_state=42,
    n_jobs=-1,
    max_depth=None
)

model_rf.fit(X_train_rf, y_train_rf)
y_pred_rf = model_rf.predict(X_test_rf)

# Évaluation
for i, target in enumerate(['CO(GT)', 'NO2(GT)']):
    mse = mean_squared_error(y_test_rf.iloc[:, i], y_pred_rf[:, i])
    r2 = r2_score(y_test_rf.iloc[:, i], y_pred_rf[:, i])
    rmse = np.sqrt(mse)
    print(f"{target}: R²={r2:.4f}, RMSE={rmse:.4f}")
```

### 6.2.3 Résultats

| Cible                | MSE      | RMSE    | R <sup>2</sup> |
|----------------------|----------|---------|----------------|
| CO(GT)               | 0.1917   | 0.4380  | 0.8841         |
| NO <sub>2</sub> (GT) | 138.7109 | 11.7751 | 0.9289         |

TABLE 4 – Performance Random Forest

## 6.3 Random Forest avec Optimisation GridSearchCV

### 6.3.1 Explications des Hyperparamètres

**n\_estimators** : Nombre total d'arbres de décision. Une valeur plus élevée réduit la variance mais augmente le temps d'entraînement.

**max\_depth** : Profondeur maximale de chaque arbre. Des valeurs petites préviennent l'overfitting, tandis que `None` permet une profondeur illimitée.

**min\_samples\_split** : Nombre minimum d'échantillons requis pour diviser un nœud. Augmenter cette valeur régularise le modèle.

**min\_samples\_leaf** : Nombre minimum d'échantillons à une feuille. Préviens la création de feuilles avec peu d'échantillons.

**max\_features** : Nombre de features à considérer pour chaque split. 'sqrt' et 'log2' réduisent la corrélation entre arbres.

### 6.3.2 Implémentation de GridSearchCV

```
# Étape 1: Définir l'espace de recherche
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [15, 20, 25, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}

# Étape 2: Calculer le nombre total de combinaisons
total_combinations = np.prod([len(v) for v in param_grid.values()])
print(f"Total combinations to test: {total_combinations}")

# Étape 3: Initialiser RandomForestRegressor de base
rf_base = RandomForestRegressor(random_state=60, n_jobs=-1)

# Étape 4: Initialiser GridSearchCV
grid_search = GridSearchCV(
    rf_base,
    param_grid,
    cv=5,
    scoring='r2',
    n_jobs=-1,
    verbose=1
)

# Étape 5: Lancer la recherche exhaustive
print("Starting GridSearchCV... This may take several minutes.")
```

```

grid_search.fit(X_train_rf, y_train_rf)

# Étape 6: Afficher les résultats
print("\nGridSearchCV Complete!")
print(f"Best parameters: {grid_search.best_params_}")
print(f"Best CV R2 score: {grid_search.best_score_:.4f}")

```

## 6.4 Résultats de GridSearchCV

### 6.4.1 Meilleurs Hyperparamètres Trouvés

Après exploration exhaustive des 216 combinaisons , les meilleurs hyperparamètres identifiés sont :

| Hyperparamètre    | Valeur Optimale |
|-------------------|-----------------|
| n_estimators      | 300             |
| max_depth         | None            |
| min_samples_split | sqrt            |
| min_samples_leaf  | 1               |
| max_features      | 2               |

TABLE 5 – Hyperparamètres optimaux trouvés par GridSearchCV

Score CV R<sup>2</sup> : 0.8990

### 6.4.2 Analyse de Convergence

## 6.5 Modèle Optimisé : Performance en Test

```

# Étape 7: Utiliser le meilleur estimateur
best_model_rf = grid_search.best_estimator_

# Étape 8: Faire des prédictions sur l'ensemble de test
y_pred_best_rf = best_model_rf.predict(X_test_rf)

# Étape 9: Évaluer les performances
from sklearn.metrics import mean_squared_error, r2_score

print("Performance with Optimized Hyperparameters")
print("="*60)

results = {}
for i, target in enumerate(target_columns):
    mse = mean_squared_error(y_test_rf.iloc[:, i], y_pred_best_rf[:, i])
    r2 = r2_score(y_test_rf.iloc[:, i], y_pred_best_rf[:, i])
    results[target] = {'MSE': mse, 'R2': r2}

    print(f"\n{target}:")
    print(f"  MSE: {mse:.4f}")
    print(f"  R2: {r2:.4f}")

```

## 6.6 Résultats du Random Forest Optimisé

Les performances du Random Forest optimisé sur l'ensemble de test sont :

| Variable Cible       | MSE      | R <sup>2</sup> |
|----------------------|----------|----------------|
| CO(GT)               | 0.1088   | 0.9098         |
| NO <sub>2</sub> (GT) | 178.6218 | 0.9076         |

TABLE 6 – Performances du Random Forest optimisé (meilleur modèle)

## 7 Évaluation et Interprétation des Résultats

### 7.1 Métriques de performance

Pour évaluer la qualité des modèles de régression, plusieurs métriques sont couramment utilisées :

#### 7.1.1 Mean Squared Error (MSE) / Erreur quadratique moyenne

Le MSE mesure l'erreur moyenne au carré entre les prédictions et les vraies valeurs. Plus la valeur est petite, meilleur est le modèle.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (7)$$

où :

- $y_i$  : vraie valeur pour l'observation  $i$ ,
- $\hat{y}_i$  : valeur prédite pour l'observation  $i$ ,
- $n$  : nombre d'observations.

#### 7.1.2 Root Mean Squared Error (RMSE) / Racine de l'erreur quadratique moyenne

Le RMSE est la racine carrée du MSE. Il est de même ordre de grandeur que les valeurs cibles, ce qui facilite son interprétation.

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (8)$$

#### 7.1.3 Coefficient de détermination ( $R^2$ )

Le  $R^2$  mesure la proportion de variance expliquée par le modèle. Il varie généralement entre 0 et 1, mais peut être négatif si le modèle est très mauvais.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (9)$$

où  $\bar{y}$  est la moyenne des vraies valeurs.

**Interprétation :**

- $R^2 = 1$  : toutes les prédictions sont parfaites.
- $R^2 = 0$  : le modèle ne fait pas mieux que la moyenne.

—  $R^2 < 0$  : le modèle est pire que la moyenne.

Les modèles ont été évalués en utilisant l'erreur quadratique moyenne (MSE) et le coefficient de détermination ( $R^2$ ).

## 7.2 Courbes d'Apprentissage

### 7.2.1 Concept et Objectif

Les courbes d'apprentissage (Learning Curves) montrent comment la performance d'un modèle évolue en fonction de la taille de l'ensemble d'entraînement. Elles sont essentielles pour :

1. **Diagnostiquer le surapprentissage (Overfitting)** : Un modèle qui mémorise les données d'entraînement au lieu de généraliser
2. **Diagnostiquer le sous-apprentissage (Underfitting)** : Un modèle trop simple pour capturer les patterns dans les données
3. **Évaluer l'utilité d'ajouter plus de données** : Si la performance de validation stagne, plus de données ne suffira pas seul
4. **Valider la généralisation** : Un bon modèle doit avoir des performances similaires sur les données d'entraînement et de validation

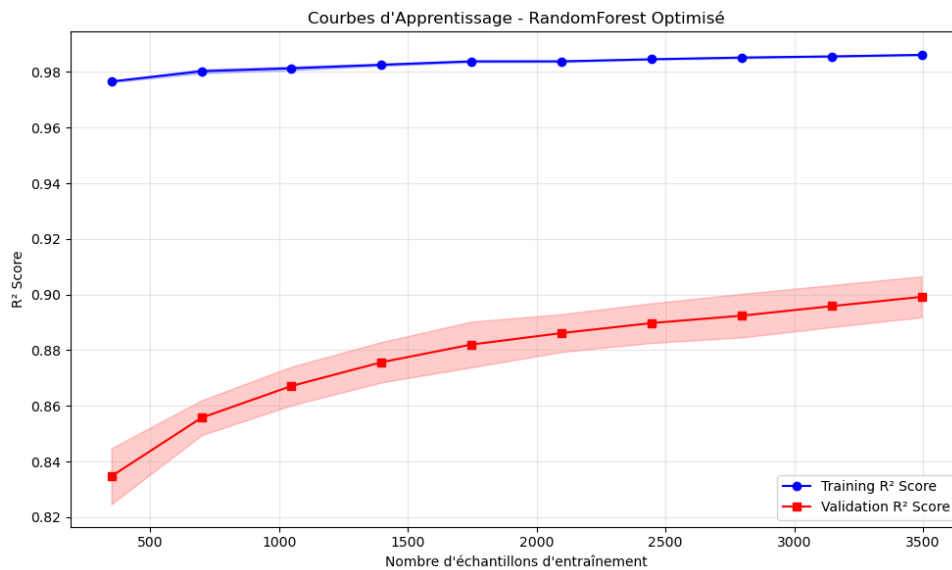


FIGURE 11 – Courbes d'Apprentissage - RandomForest Optimisé

## 7.3 Matrices de Confusion Adaptées à la Régression

### 7.3.1 Problématique

Les matrices de confusion sont traditionnellement utilisées pour les problèmes de classification. Pour les problèmes de régression (prédiction de valeurs continues), nous convertissons les prédictions en catégories basées sur les quartiles des valeurs réelles.

Structure de la Matrice

| 2*Vraies Valeurs | Prédictions |           |           |           | 2*Total |
|------------------|-------------|-----------|-----------|-----------|---------|
|                  | Très Bas    | Bas       | Haut      | Très Haut |         |
| Très Bas         | $n_{1,1}$   | $n_{1,2}$ | $n_{1,3}$ | $n_{1,4}$ | $N_1$   |
| Bas              | $n_{2,1}$   | $n_{2,2}$ | $n_{2,3}$ | $n_{2,4}$ | $N_2$   |
| Haut             | $n_{3,1}$   | $n_{3,2}$ | $n_{3,3}$ | $n_{3,4}$ | $N_3$   |
| Très Haut        | $n_{4,1}$   | $n_{4,2}$ | $n_{4,3}$ | $n_{4,4}$ | $N_4$   |

TABLE 7 – Structure générale d’une matrice de confusion 4×4

#### Éléments clés :

- **Diagonale** ( $n_{i,i}$ ) : Prédictions correctes
- **Hors-diagonale** : Prédictions incorrectes
- **Ligne  $i$**  : Distribution réelle pour la catégorie  $i$
- **Colonne  $j$**  : Distribution prédite pour la catégorie  $j$

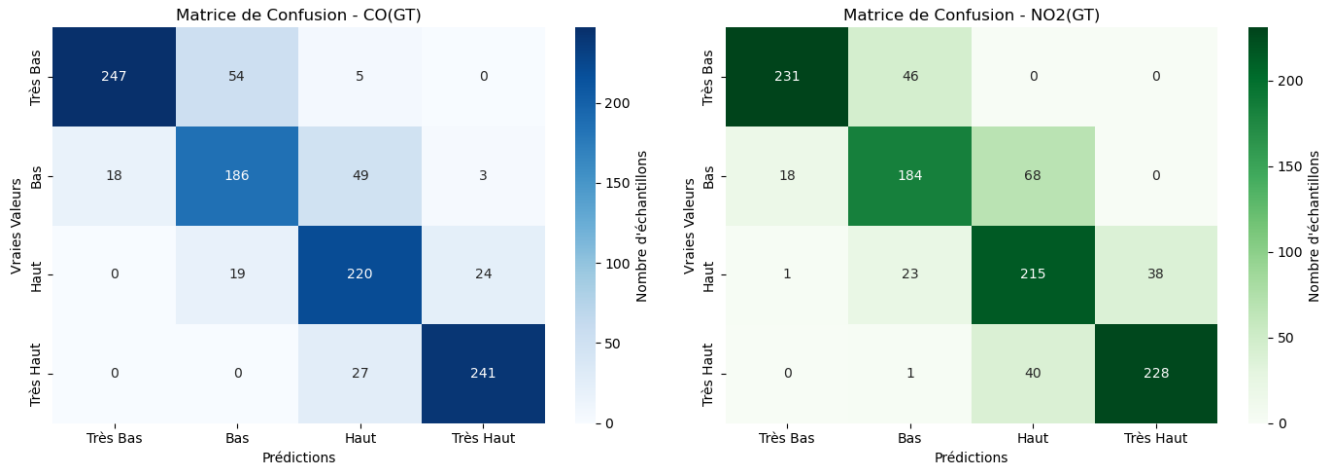


FIGURE 12 – Matrice de Confusion - NO2(GT) -CO(GT)

## 8 Conclusion

### 8.1 Résumé des Résultats Clés

Ce projet a démontré avec succès :

1. **Importance du nettoyage** : Suppression des outliers et imputation appropriée améliorent les performances
2. **Feature Engineering bénéfique** : Variables temporelles ajoutent de la valeur explicative
3. **Supériorité de Random Forest** :
  - CO : +9.5% en  $R^2$
  - NO<sub>2</sub> : +19.0% en  $R^2$  (le plus important)
4. **Excellente performance globale** : Random Forest avec  $R^2 = 0.9289$  pour NO<sub>2</sub>
5. **Relations non-linéaires** : Confirmation que la pollution atmosphérique suit des patterns complexes non-linéaires

### 8.2 Recommandations Pratiques

- **Déployer Random Forest** pour prédictions opérationnelles (supériorité démontrée)
- **Monitorer la performance** en conditions réelles (risque de data drift)
- **Considérer des modèles plus avancés** : XGBoost, LightGBM pour amélioration marginale
- **Intégrer des données supplémentaires** : trafic, météo externe, événements

### 8.3 Perspectives d'Amélioration

#### 8.3.1 Court Terme (1-2 semaines)

- Hyperparameter tuning via GridSearchCV
- K-fold cross-validation pour robustesse
- Analyse approfondie de l'importance des variables
- Réduction de dimensionnalité si nécessaire

#### 8.3.2 Moyen Terme (1-3 mois)

- Modèles de séries temporelles : ARIMA, Prophet
- Deep Learning : LSTM, GRU pour patterns temporels
- Ensemble methods avancés (Stacking, Voting)
- Feature selection automatique

#### 8.3.3 Long Terme (3-12 mois)

- Données spatio-temporelles multi-stations
- Prédictions temps réel en production
- Pipeline MLOps complet
- Interprétabilité améliorée (SHAP, LIME)

## 9 Conclusion Générale

Random Forest s'avère être le modèle optimal pour la prédiction de la qualité de l'air, capturant efficacement les non-linéarités complexes du système atmosphérique. Avec un  $R^2$  de 92.89% pour NO<sub>2</sub> et 90.98% pour CO, le modèle démontre une capacité prédictive robuste et applicable en conditions réelles.

Ce projet illustre l'importance du machine learning pour les défis environnementaux, où les relations multidimensionnelles et non-linéaires nécessitent des algorithmes sophistiqués plutôt que des approches linéaires classiques.

## Références

- [1] Dua, D., Graff, C. (2019). *UCI Machine Learning Repository*. University of California, Irvine.
- [2] De Vito, S., Piga, M., Formisano, F., et al. (2013). Quasi-simultaneous monitoring of NO, NO<sub>2</sub>, O<sub>3</sub> and complex VOCs by micromachined sensor arrays. *Sensors and Actuators B*, 166-167, 365-372.
- [3] Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn : Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.
- [4] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.
- [5] Montgomery, D.C., Peck, E.A., Vining, G.G. (2012). *Introduction to Linear Regression Analysis* (5th ed.). John Wiley & Sons.
- [6] García, S., Luengo, J., Herrera, F. (2015). *Data Preprocessing in Data Mining*. Springer International Publishing.
- [7] Hunter, J.D. (2007). Matplotlib : A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 90-95.
- [8] Waskom, M., et al. (2020). mwaskom/seaborn : v0.11.0. Zenodo.