ik9jle0ut

October 15, 2025

[1]:
```
!pip install pandas numpy matplotlib seaborn scikit-learn plotly folium␣
↪pingouin openpyxl

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.cluster import KMeans
import folium
import plotly.express as px
import pingouin as pg
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages
(2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages
(2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-
packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-
packages (0.13.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-
packages (1.6.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.12/dist-packages
(5.24.1)
Requirement already satisfied: folium in /usr/local/lib/python3.12/dist-packages
(0.20.0)
Collecting pingouin
  Downloading pingouin-0.5.5-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: openpyxl in /usr/local/lib/python3.12/dist-
packages (3.1.5)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-
packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-
packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in
```

```
/usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-
packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.5)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-
packages (from scikit-learn) (1.16.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-
packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.12/dist-packages (from plotly) (8.5.0)
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.12/dist-
packages (from folium) (0.8.2)
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.12/dist-
packages (from folium) (3.1.6)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-
packages (from folium) (2.32.4)
Requirement already satisfied: xyzservices in /usr/local/lib/python3.12/dist-
packages (from folium) (2025.4.0)
Collecting pandas-flavor (from pingouin)
  Downloading pandas_flavor-0.7.0-py3-none-any.whl.metadata (6.7 kB)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.12/dist-
packages (from pingouin) (0.14.5)
Requirement already satisfied: tabulate in /usr/local/lib/python3.12/dist-
packages (from pingouin) (0.9.0)
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.12/dist-
packages (from openpyxl) (2.0.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2>=2.9->folium) (3.0.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-
packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: xarray in /usr/local/lib/python3.12/dist-packages
(from pandas-flavor->pingouin) (2025.10.1)
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests->folium) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-
packages (from requests->folium) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
```

```
/usr/local/lib/python3.12/dist-packages (from requests->folium) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.12/dist-packages (from requests->folium) (2025.10.5)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.12/dist-
packages (from statsmodels->pingouin) (1.0.1)
Downloading pingouin-0.5.5-py3-none-any.whl (204 kB)
                         204.4/204.4 kB
3.8 MB/s eta 0:00:00
Downloading pandas_flavor-0.7.0-py3-none-any.whl (8.4 kB)
Installing collected packages: pandas-flavor, pingouin
Successfully installed pandas-flavor-0.7.0 pingouin-0.5.5
```

[2]:
```python
# 2 Lecture des données Excel
from google.colab import files

# Upload du fichier Excel
uploaded = files.upload()

# Lire le fichier Excel (remplacer 'data.xlsx' par le nom réel)
df = pd.read_excel(list(uploaded.keys())[0])

# Aperçu des données
df.head()
```

```
<IPython.core.display.HTML object>

Saving Couche_join.xls to Couche_join.xls
```

[2]:
```
   ID_GEOFLA  CODE_COMM  INSEE_COM                    NOM_COMM  \
0         43         46       4046  LE CHAFFAUT-SAINT-JURSON
1        133        156       4156                  PUIMICHEL
2        523        140       4140                LES OMERGUES
3        533        244       4244                     VOLONNE
4        541        231       4231                    VALERNES

             STATUT  X_CHF_LIEU  Y_CHF_LIEU  X_CENTROID  Y_CENTROID  Z_MOYEN  \
0    Commune simple        9524       63315        9517       63299      657
1    Commune simple        9423       63240        9426       63246      684
2    Commune simple        9084       63448        9070       63426     1085
3  Chef-lieu canton        9412       63392        9428       63407      685
4    Commune simple        9361       63558        9370       63547      689

   …  PM25_2022  PM25_2021   O3_aot_2021  PM10_2021  PM10_2022 O3_pic_2022  \
0  …   7.115596   6.215596  14382.893218  10.710019  12.676659  108.485551
1  …   6.864815   5.964815  15071.822266  10.417789  12.317789  110.160999
2  …   5.357932   5.084126  12463.222824   9.068824   9.632363  107.692051
3  …   7.041551   6.103935  13890.483358  10.454592  12.307749  107.351556
4  …   7.863844   6.570470  12792.700027  10.730546  12.872127  105.842179
```

```
   NO2_2021  NO2_2022  O3_PIC_2021 count_dep-ha
0  2.940701  3.335649    98.095726          0.0
1  2.621400  3.021063    99.150369          0.0
2  2.600000  3.000000    99.789954          0.0
3  2.908847  3.302569    96.750547          0.0
4  2.800878  3.198216    95.525842          0.0

[5 rows x 33 columns]
```

[7]:

```python
from sklearn.preprocessing import MinMaxScaler

polluants = [
    'ICAIR_2021', 'O3_AOT40_2022', 'ICAIR_2022', 'PM25_2022', 'PM25_2021',
    'O3_aot_2021', 'PM10_2021', 'PM10_2022', 'O3_pic_2022',
    'NO2_2021', 'NO2_2022', 'O3_PIC_2021'
]

# Colonnes à normaliser : polluants + count_dep-ha
cols_to_normalize = polluants + ['count_dep-ha']

# Initialiser le scaler Min-Max
scaler = MinMaxScaler(feature_range=(0, 100))

# Copier le DataFrame original
df_norm = df.copy()

# Appliquer la normalisation seulement sur les colonnes sélectionnées
df_norm[cols_to_normalize] = scaler.fit_transform(df[cols_to_normalize])

# Vérifier
print(df_norm[cols_to_normalize].head())
```

```
   ICAIR_2021  O3_AOT40_2022  ICAIR_2022  PM25_2022  PM25_2021  O3_aot_2021  \
0   47.663075      63.320461   16.780389  33.574603  20.102485    42.435415
1   46.636889      68.973153   15.414995  29.594695  15.955282    46.725105
2   44.152526      52.219903    6.975053   5.680400   1.391208    30.482383
3   46.837983      59.200352   15.729716  32.399492  18.255917    39.369372
4   47.776320      51.329673   18.850566  45.449319  25.971071    32.533908

   PM10_2021  PM10_2022  O3_pic_2022  NO2_2021  NO2_2022  O3_PIC_2021  \
0  14.867282  26.640274    63.283671  1.171568  1.185432    51.720319
1  12.326571  24.039980    70.971894  0.073589  0.074390    58.145598
2   0.598373   4.581962    59.642487  0.000000  0.000000    62.042185
3  12.646546  23.967236    58.080040  1.062034  1.068601    43.524983
4  15.045751  28.056594    51.153876  0.690760  0.700050    36.063624
```

```
   count_dep-ha
0           0.0
1           0.0
2           0.0
3           0.0
4           0.0
```

[8]:
```
# Matrice de corrélation Pearson
corr_pearson = df_norm[polluants + ['count_dep-ha']].corr(method='pearson')

# Matrice de corrélation Spearman
corr_spearman = df_norm[polluants + ['count_dep-ha']].corr(method='spearman')

# Matrice de corrélation Kendall
corr_kendall = df_norm[polluants + ['count_dep-ha']].corr(method='kendall')
```

[21]:
```
df_stats
```

[21]:

	polluant	r	p-val
3	PM25_2022	-0.013376	0.681324
2	ICAIR_2022	-0.048592	0.135522
4	PM25_2021	-0.052487	0.106859
7	PM10_2022	-0.055451	0.088444
9	NO2_2021	-0.058603	0.071757
10	NO2_2022	-0.058626	0.071644
0	ICAIR_2021	-0.062306	0.055537
6	PM10_2021	-0.063890	0.049596
5	O3_aot_2021	-0.065234	0.044981
8	O3_pic_2022	-0.080031	0.013859
1	O3_AOT40_2022	-0.080287	0.013556
11	O3_PIC_2021	-0.082993	0.010701

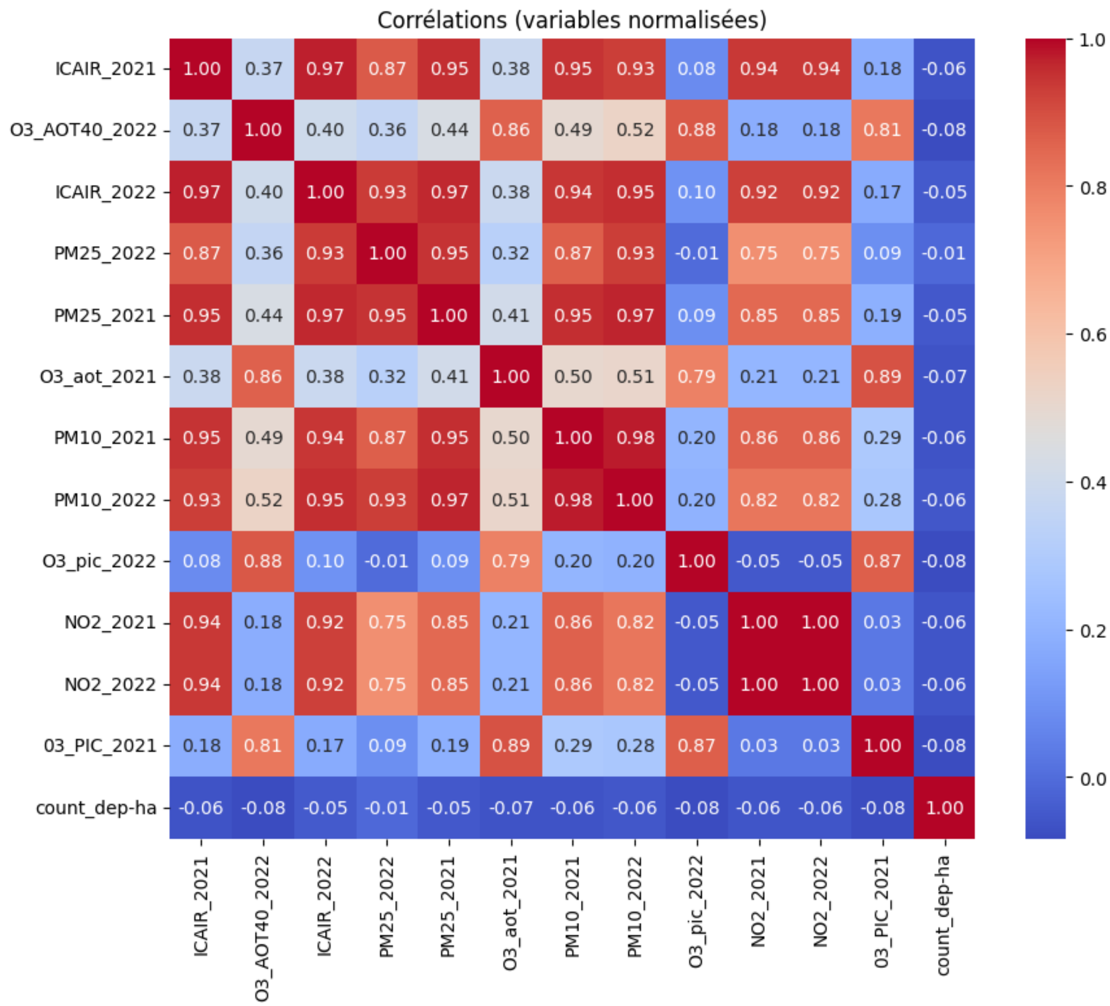
[75]:
```

```

[26]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import seaborn as sns
import matplotlib.pyplot as plt

# --------------------
# 1. Variables polluants + cible
# --------------------
polluants = [
    'ICAIR_2021', 'O3_AOT40_2022', 'ICAIR_2022', 'PM25_2022', 'PM25_2021',
    'O3_aot_2021', 'PM10_2021', 'PM10_2022', 'O3_pic_2022',
    'NO2_2021', 'NO2_2022', 'O3_PIC_2021'
```

```
]
cols_to_use = polluants + ['count_dep-ha']

# --------------------
# 2. Vérifier les NaN/inf
# --------------------
print("Valeurs manquantes :")
print(df_norm[cols_to_use].isna().sum())

# Supprimer ou imputer les lignes avec NaN si nécessaire
df_model = df_norm[cols_to_use].replace([np.inf, -np.inf], np.nan).dropna()

# --------------------
# 3. Corrélation (Pearson)
# --------------------
corr_matrix = df_model.corr()

plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Corrélations (variables normalisées)")
plt.show()

# Corrélation spécifique avec le dépérissement
corr_target = corr_matrix["count_dep-ha"].sort_values(ascending=False)
print("Corrélations avec le dépérissement :\n", corr_target)

# --------------------
# 4. Régression multiple OLS
# --------------------
X = df_model[polluants]
y = df_model["count_dep-ha"]

X = sm.add_constant(X)  # constante
model = sm.OLS(y, X).fit()
print(model.summary())
```

```
Valeurs manquantes :
ICAIR_2021        0
O3_AOT40_2022     0
ICAIR_2022        0
PM25_2022        33
PM25_2021        33
O3_aot_2021      33
PM10_2021         0
PM10_2022         0
O3_pic_2022       0
NO2_2021          0
```

```
NO2_2022         0
O3_PIC_2021      0
count_dep-ha    33
dtype: int64
```




```
Corrélations avec le dépérissement :
 count_dep-ha     1.000000
PM25_2022       -0.013376
ICAIR_2022      -0.048592
PM25_2021       -0.052487
PM10_2022       -0.055451
NO2_2021        -0.058603
NO2_2022        -0.058626
ICAIR_2021      -0.062306
PM10_2021       -0.063890
O3_aot_2021     -0.065234
O3_pic_2022     -0.080031
```

```
O3_AOT40_2022   -0.080287
03_PIC_2021     -0.082993
Name: count_dep-ha, dtype: float64
                            OLS Regression Results
==============================================================================
Dep. Variable:           count_dep-ha   R-squared:                       0.027
Model:                            OLS   Adj. R-squared:                  0.014
Method:                 Least Squares   F-statistic:                     2.139
Date:                Wed, 15 Oct 2025   Prob (F-statistic):             0.0128
Time:                        07:43:16   Log-Likelihood:                -2561.8
No. Observations:                 945   AIC:                             5150.
Df Residuals:                     932   BIC:                             5213.
Df Model:                          12
Covariance Type:            nonrobust
================================================================================
=
                   coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-
const            0.9530      3.428      0.278      0.781      -5.775
7.681
ICAIR_2021      -0.0035      0.082     -0.043      0.965      -0.164
0.156
O3_AOT40_2022   -0.0067      0.024     -0.282      0.778      -0.053
0.040
ICAIR_2022      -0.5054      0.455     -1.112      0.266      -1.397
0.387
PM25_2022        0.2413      0.143      1.689      0.092      -0.039
0.522
PM25_2021       -0.0650      0.034     -1.939      0.053      -0.131
0.001
O3_aot_2021      0.0350      0.022      1.577      0.115      -0.009
0.079
PM10_2021        0.0826      0.044      1.895      0.058      -0.003
0.168
PM10_2022       -0.1020      0.051     -1.986      0.047      -0.203
-0.001
O3_pic_2022      0.0456      0.046      1.001      0.317      -0.044
0.135
NO2_2021         3.7639      7.829      0.481      0.631     -11.600
19.128
NO2_2022        -3.4297      7.832     -0.438      0.662     -18.800
11.940
03_PIC_2021     -0.0276      0.020     -1.364      0.173      -0.067
0.012
==============================================================================
Omnibus:                     2222.013   Durbin-Watson:                   2.032
```

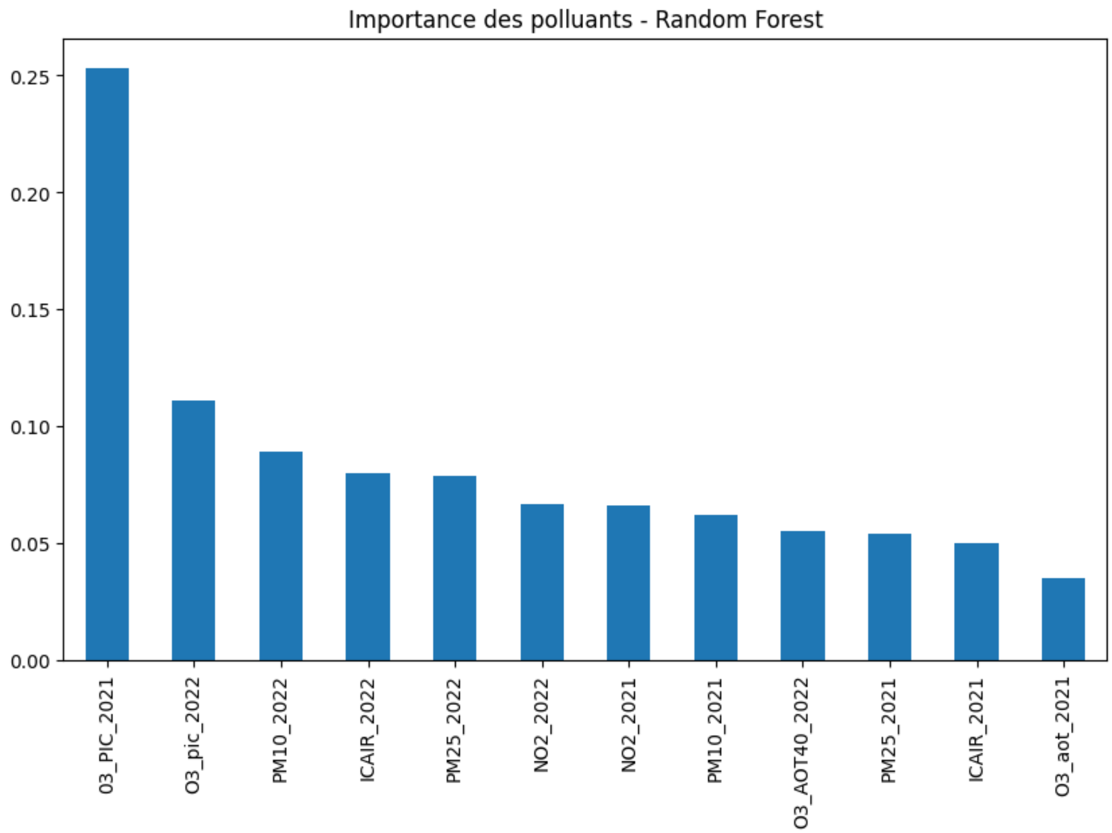
```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         12357571.552
Skew:                          21.594   Prob(JB):                         0.00
Kurtosis:                     561.550   Cond. No.                     1.36e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.36e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[47]:

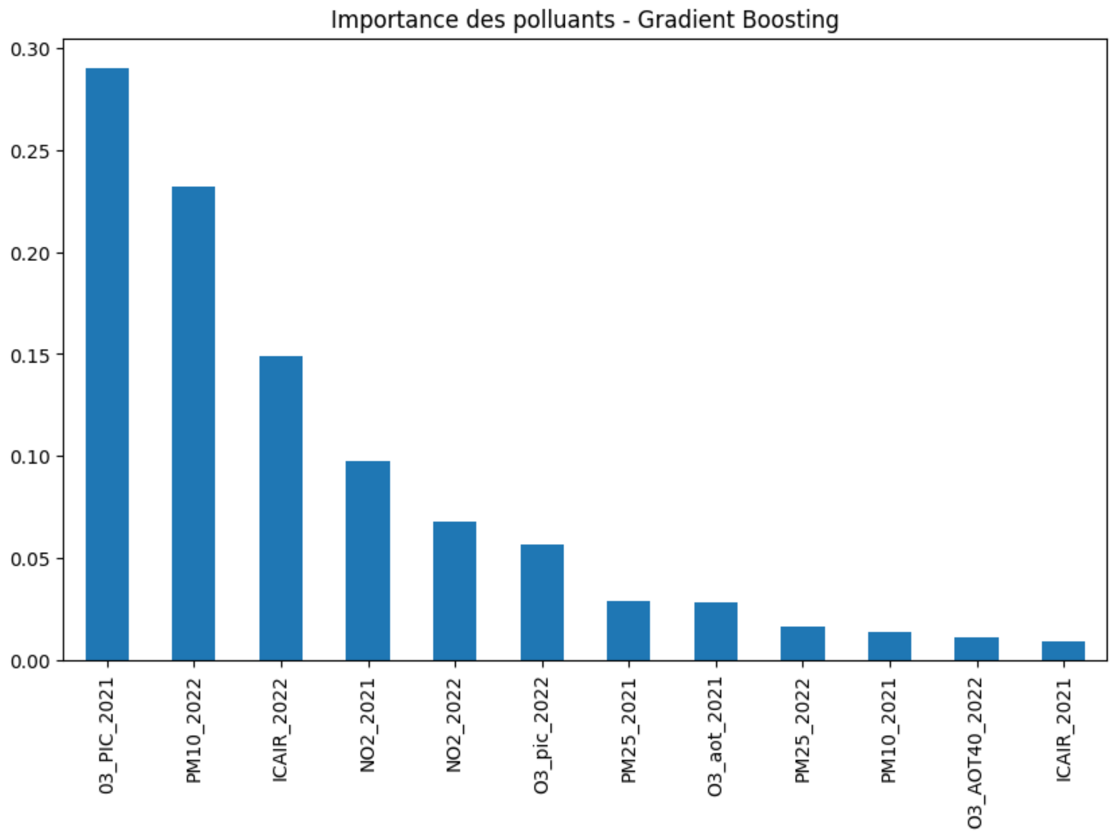
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import matplotlib.pyplot as plt

# ----------------------------
# Colonnes polluants
# ----------------------------
polluants = [
        'ICAIR_2021', 'O3_AOT40_2022', 'ICAIR_2022', 'PM25_2022', 'PM25_2021',
    'O3_aot_2021', 'PM10_2021', 'PM10_2022', 'O3_pic_2022',
    'NO2_2021', 'NO2_2022', 'O3_PIC_2021'
]

cols_to_normalize = polluants + ['count_dep-ha']

# ----------------------------
# Normalisation MinMax
# ----------------------------
scaler = MinMaxScaler(feature_range=(0, 100))
df_norm = df.copy()
df_norm[cols_to_normalize] = scaler.fit_transform(df[cols_to_normalize])

# ----------------------------
# Features & Target (nettoyage NaN)
# ----------------------------
df_model = df_norm[polluants + ['count_dep-ha']].dropna()
X = df_model[polluants]
y = df_model['count_dep-ha']

# ----------------------------
# Split train/test
# ----------------------------
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ----------------------------
# Fonction pour tester un modèle
# ----------------------------
def run_model(name, model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    print(f"{name} - RMSE: {rmse:.3f}, R²: {r2:.3f}")

    # Importance des variables si disponible
    if hasattr(model, "feature_importances_"):
        importances = pd.Series(model.feature_importances_, index=X_train.
 ↪columns)
        importances.sort_values(ascending=False).plot(kind='bar',␣
 ↪figsize=(10,6))
        plt.title(f"Importance des polluants - {name}")
        plt.show()

# ----------------------------
# Lancer les modèles
# ----------------------------
run_model(
    "Random Forest",
    RandomForestRegressor(n_estimators=500, random_state=42),
    X_train, X_test, y_train, y_test
)

run_model(
    "Gradient Boosting",
    GradientBoostingRegressor(n_estimators=500, learning_rate=0.05,␣
 ↪max_depth=3, random_state=42),
    X_train, X_test, y_train, y_test
)
```

```
Random Forest - RMSE: 1.918, R²: -1.974
```




```
Gradient Boosting - RMSE: 1.864, R²: -1.809
```



[30]:
```
# @title Titre par défaut
# RMSE (~1.91) : c'est l'erreur moyenne de prédiction. Comme tes données sont␣
↪normalisées entre 0 et 100, une erreur ~1.9 est relativement faible en␣
↪valeur absolue, mais il faut regarder le R² pour juger la qualité du modèle.

# R² (~ -1.97) : un R² négatif indique que le modèle fait pire qu'une simple␣
↪moyenne.

# R² = 1 → modèle parfait

# R² = 0 → modèle prédit aussi bien que la moyenne

# R² < 0 → modèle pire que la moyenne, donc le modèle n'explique rien du␣
↪dépérissement avec ces variables.

# Interprétation

# La pollution seule n'explique pas le dépérissement : même avec des modèles␣
↪non-linéaires, les polluants disponibles (PM10, PM2.5, O , NO , ICAIR) ne␣
↪permettent pas de prédire le count_dep-ha.
```

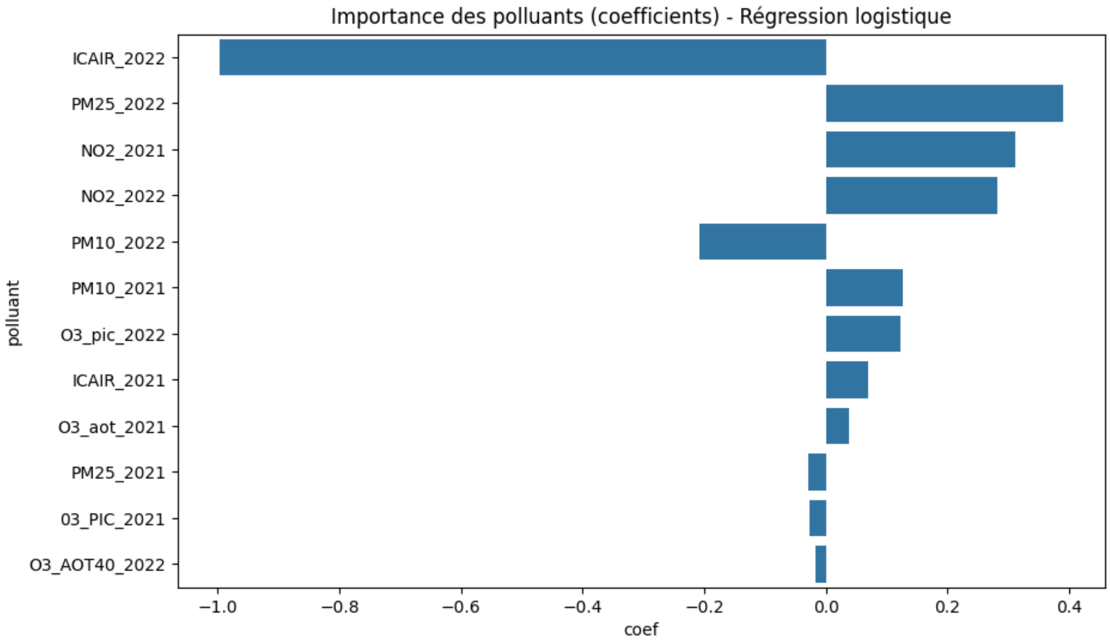
```
# Relations complexes ou facteurs manquants :

# Le dépérissement peut dépendre d'autres facteurs : climat, sols, humidité,␣
↪stress hydrique, pathogènes, interventions humaines.

# Même si Random Forest ou Gradient Boosting capturent des non-linéarités, ils␣
↪ne peuvent rien apprendre si les variables pertinentes ne sont pas présentes.

# Multicolinéarité et bruit : les polluants sont corrélés entre eux et il peut␣
↪y avoir beaucoup de bruit dans les mesures, ce qui empêche le modèle de␣
↪généraliser.

# Conclusion

# Tes modèles montrent que la pollution seule, telle que mesurée, n'est pas un␣
↪bon prédicteur du dépérissement forestier.

# Il faudrait :

# ajouter d'autres variables environnementales (climat, sol, altitude, stress␣
↪hydrique…),

# éventuellement combiner les polluants en indices synthétiques (PCA ou scores␣
↪de pollution),

# ou étudier des sous-ensembles de données plus homogènes (ex. par région ou␣
↪type de forêt).
```

[48]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report,␣
↪roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns

# ----------------------------
# 1. Colonnes polluants
# ----------------------------
polluants = [
         'ICAIR_2021', 'O3_AOT40_2022', 'ICAIR_2022', 'PM25_2022', 'PM25_2021',
    'O3_aot_2021', 'PM10_2021', 'PM10_2022', 'O3_pic_2022',
    'NO2_2021', 'NO2_2022', 'O3_PIC_2021'
```

```
]

cols_to_normalize = polluants + ['count_dep-ha']

# ----------------------------
# 2. Normalisation MinMax
# ----------------------------
scaler = MinMaxScaler(feature_range=(0, 100))
df_norm = df.copy()
df_norm[cols_to_normalize] = scaler.fit_transform(df[cols_to_normalize])

# ----------------------------
# 3. Convertir la cible en binaire
# ----------------------------
threshold = df_norm['count_dep-ha'].median()  # seuil : médiane
df_norm['dep_class'] = (df_norm['count_dep-ha'] > threshold).astype(int)

# ----------------------------
# 4. Préparer les features et la cible
# ----------------------------
X = df_norm[polluants].replace([np.inf, -np.inf], np.nan).dropna()
y = df_norm.loc[X.index, 'dep_class']

# Split train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# ----------------------------
# 5. Entraîner la régression logistique
# ----------------------------
logreg = LogisticRegression(class_weight='balanced', max_iter=(1000))

logreg.fit(X_train, y_train)

# Prédictions
y_pred = logreg.predict(X_test)
y_proba = logreg.predict_proba(X_test)[:,1]

# ----------------------------
# 6. Évaluation du modèle
# ----------------------------
print("Matrice de confusion :")
print(confusion_matrix(y_test, y_pred))

print("\nClassification report :")
print(classification_report(y_test, y_pred))
```

```python
roc_auc = roc_auc_score(y_test, y_proba)
print("ROC-AUC :", roc_auc)

# ----------------------------
# 7. Importance des variables (coefficients)
# ----------------------------
coef_df = pd.DataFrame({
    'polluant': polluants,
    'coef': logreg.coef_[0]
}).sort_values(by='coef', key=abs, ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(x='coef', y='polluant', data=coef_df)
plt.title("Importance des polluants (coefficients) - Régression logistique")
plt.show()
```

```
Matrice de confusion :
[[141 111]
 [  8  24]]

Classification report :
              precision    recall  f1-score   support

           0       0.95      0.56      0.70       252
           1       0.18      0.75      0.29        32

    accuracy                           0.58       284
   macro avg       0.56      0.65      0.50       284
weighted avg       0.86      0.58      0.66       284

ROC-AUC : 0.7072172619047619
```



[ ]:
```
# Vrais Négatifs (TN) : 128  - Bonnes prédictions de classe 0

# Faux Positifs (FP) : 135 - Beaucoup d'erreurs de type I

# Faux Négatifs (FN) : 8 - Peu d'erreurs de type II

# Vrais Positifs (TP) : 24 - Bonnes prédictions de classe 1
```