

Charlotte Kruzic & Zoé Marquis
UE Entreposage et protection des données

✓ Projet naiades

NB: Cela peut paraître long, mais nous avons dû télécharger le notebook en PDF depuis Google Colab. Colab utilise par défaut une police d'écriture relativement grande, ce qui explique le nombre de pages élevé dans le PDF généré.

Nous vous fournissons également la version `.ipynb` car elle est plus agréable à lire. Si vous n'avez pas Jupyter Notebook installé, vous pouvez l'ouvrir directement depuis votre navigateur avec Google Colab.

Ce projet s'intéresse à la relation entre les propriétés physico-chimiques de l'eau et son état biologique, avec pour objectif de mieux comprendre les interactions entre ces deux dimensions à partir de données existantes.

Nous avons cherché à déterminer si ces données permettent de révéler des modèles spatiaux pertinents, comme les hydroécorégions, et à explorer comment les différents paramètres influencent l'état des écosystèmes aquatiques.

Pour cela, nous avons appliqué des méthodes de clustering à des données collectées à diverses saisons, tout en considérant un éventuel décalage temporel entre les mesures physico-chimiques et biologiques, afin de refléter la dynamique des écosystèmes. Ces données ont été soigneusement nettoyées, préparées et agrégées par saison, ce qui nous a permis d'identifier des regroupements cohérents d'hydro-eco-stations et d'en analyser les similitudes écologiques.

Par la suite, avec le décalage temporel qui nous paraît le meilleur, nous avons également exploré une approche de régression pour prédire l'état biologique de l'eau (I2M2) à partir des paramètres physico-chimiques, des caractéristiques des stations et de leur temporalité. Cette approche vise à évaluer la capacité de ces facteurs à expliquer et prédire l'état des écosystèmes aquatiques (hydrobiologiques dans ce cas), en tenant compte de la complexité et de la variabilité de ces interactions.

✓ Importation des bibliothèques

```
import numpy as np
import pandas as pd
import geopandas as gpd
import shapely.geometry as geom
from shapely import geometry as geom
from matplotlib.patches import Patch
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.colors as mcolors
import plotly.express as px
import plotly.graph_objects as go
import seaborn as sns
from scipy.stats import zscore
import zstandard as zstd
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics import davies_bouldin_score
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
from sklearn.preprocessing import LabelEncoder
```

✓ Chargement des données

```
# stations
df_load_stations = pd.read_csv('data/stations_hb.csv.zst', sep=';', escapechar =
df_stations = df_load_stations.copy() # copy pour nettoyer etc mais garder l'or
```

```
# pc : physicochimie
f="data/donnees_physicochimie.csv.zst"
pc_sample = pd.read_csv(f,nrows=1)
pc_list_cols = pc_sample.columns
pc_list_cat = pc_list_cols[pc_list_cols.str.startswith((
    'Lb','Nom','Mnemo',
    'Cd','Sym','Com'))]
pc_dict_cat = {col: 'category' for col in pc_list_cat}
df_load_pc = pd.read_csv(
    f,
    sep=',',
    engine='c',
    escapechar='\\',
    dtype=pc_dict_cat,
    parse_dates=[7],
    iterator=False)
df_pc = df_load_pc.copy()
```

```
# hydrobio
df_load_hydrobio = pd.read_csv('data/donnees_hydrobio.csv.zst',sep=',',escapech
df_hydrobio = df_load_hydrobio.copy()
```

```
# hydroecoregion
df_load_hydroregions = gpd.read_file("data/Hydroecoregion1-shp.zip")
df_hydroregions = df_load_hydroregions.copy()
```

✓ Analyse exploratoire

✓ Objectifs

L'objectif principal de cette analyse exploratoire est de comprendre le contenu de chaque jeu de données utilisé dans le projet, d'identifier les colonnes pertinentes ainsi que celles qui peuvent être supprimées, et de déterminer comment remodeler les données pour les rendre exploitables pour l'analyse.

✓ Description des jeux de données

Le projet repose sur plusieurs jeux de données, comprenant des mesures physico-chimiques, des informations hydrobiologiques, des données sur les stations de mesure, et des données géographiques des hydroécorégions.

1. Les **données des stations de mesure** fournissent des informations nécessaires pour localiser chaque station de mesure (latitude et longitude), ainsi que des identifiants uniques permettant de relier les stations entre les différents jeux de données.
2. Les **données physico-chimiques** contiennent les mesures physico-chimiques des eaux (par exemple, nitrates, phosphates, pH...). Chaque mesure est associée à une station, à une date, ainsi qu'à un support et une fraction d'analyse spécifique.
3. Les **données hydrobiologiques** comprennent l'indice écologique I2M2 évaluant la qualité biologique des eaux. Ces indices sont liés aux stations et aux dates de prélèvement.
4. Les **données des hydroécorégions** sont des unités spatiales définies sur la base de critères écologiques similaires, et permettent de classer les différents milieux aquatiques à travers la France.

✓ Exploration des données géographiques

```
df_stations.head(3)
```



	CdStationMesureEauxSurface	LbStationMesureEauxSurface	DurStationMesureEauxSurface
0	01000477	LA SLACK À RINXENT (62)	
1	01000602	COLOGNE à BUIRE COURCELLES (80)	
2	01000605	L'OMIGNON À DEVISE (80)	

3 rows × 39 columns

La table station est utile pour situer les stations ainsi que pour extraire l'identifiant pour joindre les tables sur les stations.

```
df_hydroregions.head(3)
```



	gid	CdHER1	NomHER1	geometry
0	1	16	CORSE	POLYGON ((9.43319 43.00468, 9.4357 42.99999, 9...
1	2	12	ARMORICAIN	POLYGON ((-2.61068 48.55022, -2.61268 48.54898...
2	3	13	LANDES	MULTIPOLYGON (((-1.04228 45.54443, -1.03836 45...

```
# On vérifie qu'un code d'hydroécorégion (CdHER1) correspond bien à un seul nom
print(df_hydroregions['CdHER1'].value_counts())
```

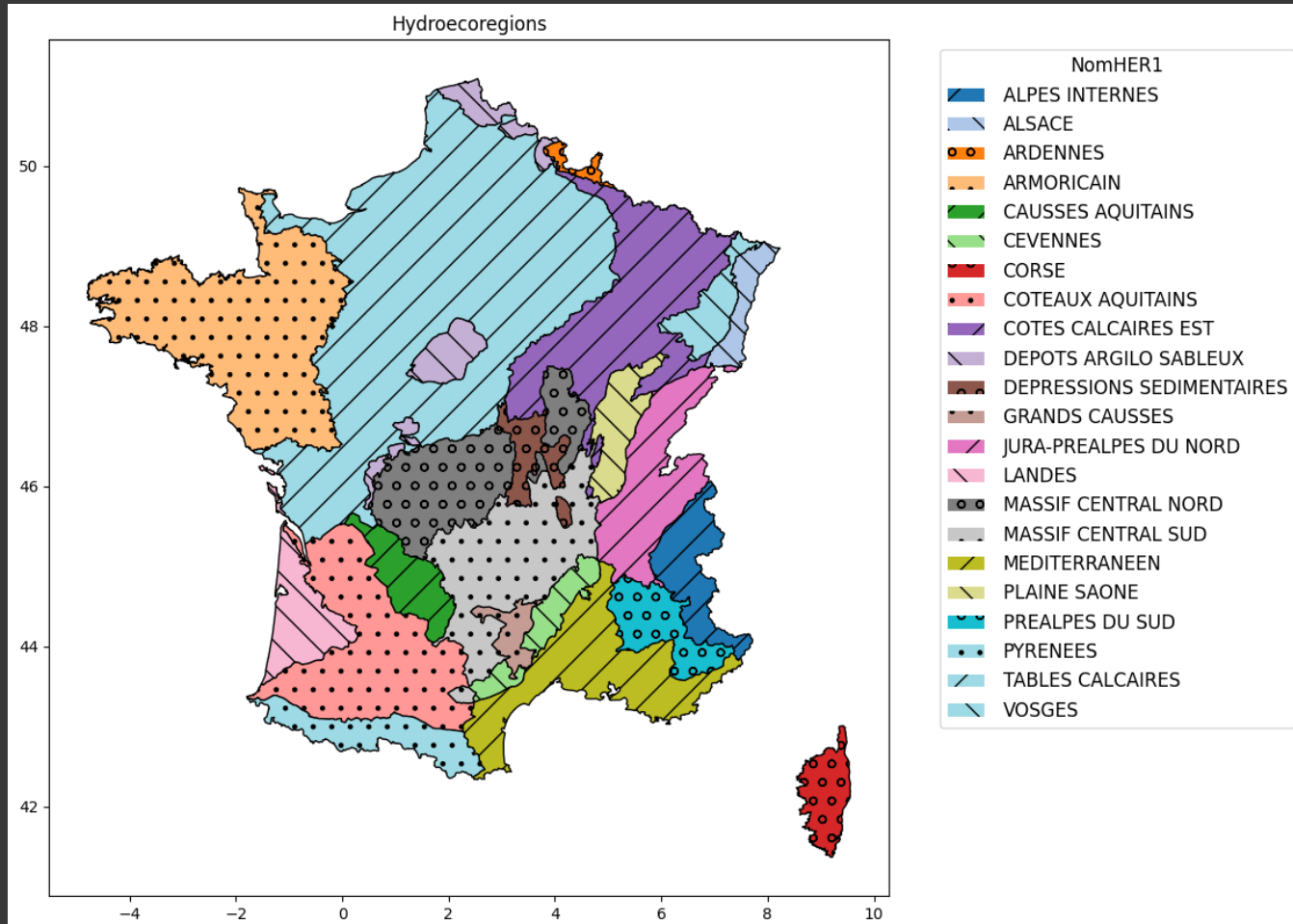


```
CdHER1
16      1
12      1
10      1
9       1
3       1
17      1
6       1
5       1
19      1
8       1
15      1
4       1
18      1
20      1
7       1
2       1
21      1
11      1
1       1
14      1
13      1
22      1
Name: count, dtype: int64
```

On a bien un CdHER1 pour un NomHER1, donc on va afficher la carte avec NomHER1 car cela est plus parlant pour l'utilisateur.

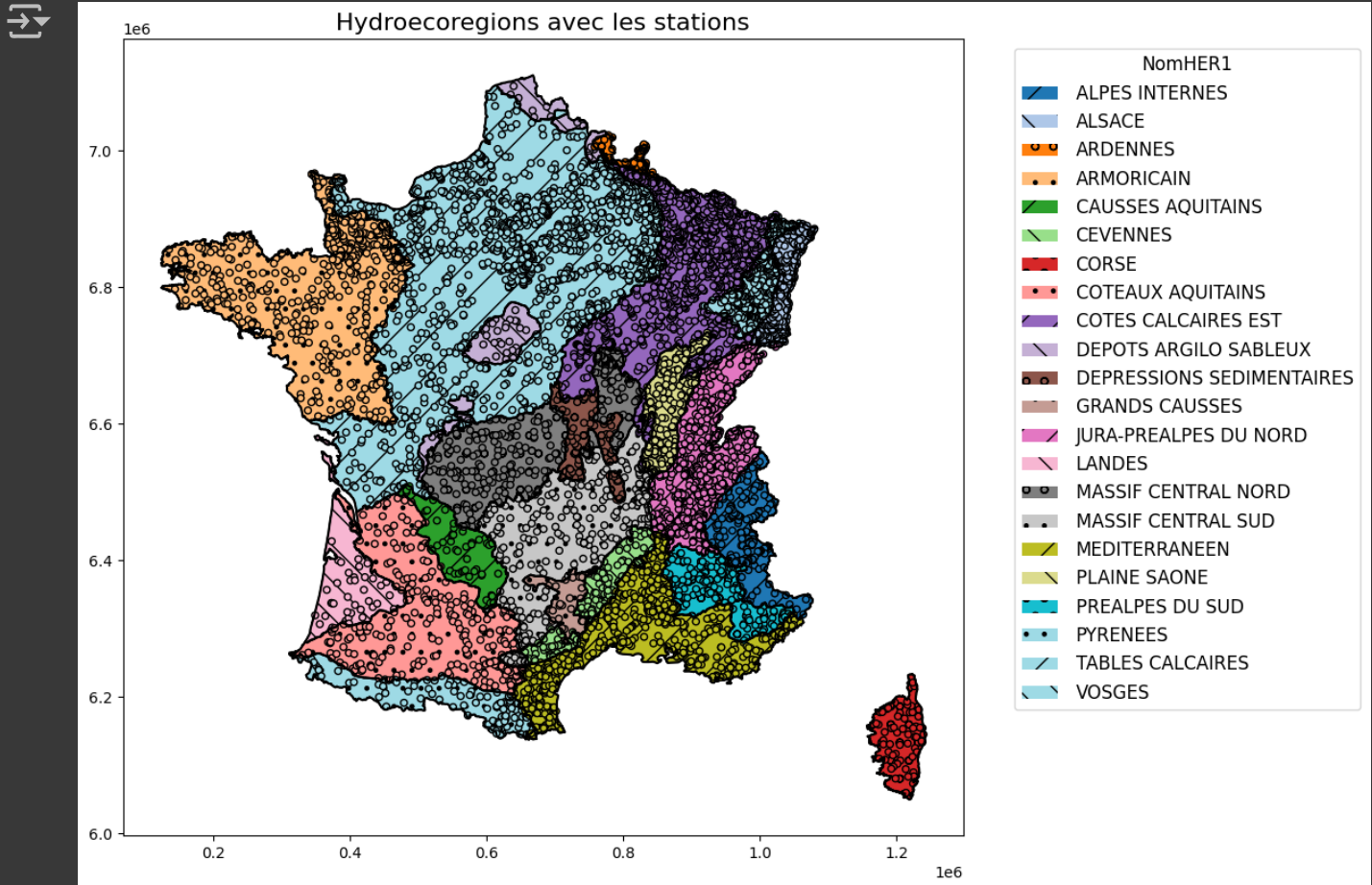
```
# Affichage des hydroeco regions
fig, ax = plt.subplots(figsize=(20, 10))
colors = plt.colormaps['tab20']
hatches = ['/', '\\', 'o', '.']
legend_elements = []
for i, (name, region) in enumerate(df_hydroregions.groupby('NomHER1')):
    patch = region.plot(ax=ax, color=colors(i), hatch=hatches[i % len(hatches)])
```

```
    legend_elements.append(Patch(facecolor=colors(i), hatch=hatches[i % len(hat
ax.set_title('Hydroecoregions')
ax.legend(handles=legend_elements, title='NomHER1', bbox_to_anchor=(1.05, 1), 1
plt.show()
```



Nous allons maintenant situer les différentes stations sur la carte des hydroécorégions.

```
# Affichage des stations sur les hydroecoregions
crs_lambert = 'PROJCS["RGF_1993_Lambert_93",GEOGCS["GCS_RGF_1993",DATUM["D_RGF_
x_col = 'CoordXStationMesureEauxSurface'
y_col = 'CoordYStationMesureEauxSurface'
carto_i2m2 = gpd.GeoDataFrame(df_stations,crs=crs_lambert, geometry = gpd.GeoSe
HER_stations=carto_i2m2.sjoin(df_hydroregions.to_crs(crs_lambert),predicate='wi
fig, ax = plt.subplots(1, 1, figsize=(10, 30))
colors = plt.colormaps['tab20']
hatches = ['/', '\\', 'o', '.']
legend_elements = []
color_mapping = {}
for i, (name, region) in enumerate(df_hydroregions.groupby('NomHER1')):
    region = region.to_crs(crs_lambert)
    patch = region.plot(ax=ax, color=colors(i), hatch=hatches[i % len(hatches)])
    legend_elements.append(Patch(facecolor=colors(i), hatch=hatches[i % len(hat
    color_mapping[name] = colors(i)
station_colors = HER_stations['NomHER1'].map(color_mapping)
HER_stations.plot(ax=ax, color=station_colors, markersize=20, edgecolor='black')
HER_lambert = df_hydroregions.to_crs(crs_lambert)
HER_lambert.boundary.plot(ax=ax, color='black')
ax.legend(handles=legend_elements, title='NomHER1', bbox_to_anchor=(1.05, 1), 1
ax.set_title('Hydroecoregions avec les stations', fontsize=16)
plt.show()
```

Nous observons maintenant la répartition des stations dans les différentes hydroécorégions.

```
stations_par_hydroecoregion = HER_stations.groupby('NomHER1').size().reset_index()
print(stations_par_hydroecoregion)
```



	NomHER1	nombre_de_stations
0	ALPES INTERNES	156
1	ALSACE	352
2	ARDENNES	63
3	ARMORICAIN	445
4	CAUSSES AQUITAINS	47
5	CEVENNES	106
6	CORSE	63
7	COTEAUX AQUITAINS	265
8	COTES CALCAIRES EST	1008
9	DEPOTS ARGILO SABLEUX	47
10	DEPRESSIONS SEDIMENTAIRES	63
11	GRANDS CAUSSES	25
12	JURA-PREALPES DU NORD	632
13	LANDES	38
14	MASSIF CENTRAL NORD	221
15	MASSIF CENTRAL SUD	264
16	MEDITERRANEEN	574
17	PLAINE SAONE	223
18	PREALPES DU SUD	167
19	PYRENEES	116
20	TABLES CALCAIRES	1360
21	VOSGES	313

```
stations_par_hydroecoregion = stations_par_hydroecoregion.sort_values(by='nombre_de_stations')
fig = px.bar(stations_par_hydroecoregion, x='NomHER1', y='nombre_de_stations')
fig.update_layout(xaxis_tickangle=45, width=900, height=600, xaxis_title="Hydroecoregion")
fig.show()
```



Nous pouvons voir que la répartition des stations n'est pas uniforme, et que le nombre de stations par hydroécorégion peut varier de 25 à 1360. Cela pourrait rendre la caractérisation plus complexe, d'autant plus que nous ne savons pas encore combien de stations seront disponibles après la préparation des données.

- ✓ Exploration des données physico-chimiques et hydrobiologiques
- ✓ Physico-chimiques

```
df_pc.info()
```

```

↔ <class 'pandas.core.frame.DataFrame'>
RangeIndex: 8917443 entries, 0 to 8917442
Data columns (total 49 columns):
#   Column                                Dtype
---  -
0   CdStationMesureEauxSurface            category
1   LbStationMesureEauxSurface            category
2   CdSupport                             category
3   LbSupport                             category
4   CdFractionAnalysee                   category
5   LbFractionAnalysee                   category
6   CdPrelevement                        category
7   DatePrel                             datetime64[ns]
8   HeurePrel                            object
9   CdParametre                          category
10  LbLongParamètre                       category
11  RsAna                                float64
12  CdUniteMesure                         category
13  SymUniteMesure                       category
14  CdRqAna                              category
15  MnemoRqAna                           category
16  CdInsituAna                          category
17  LbInsituAna                          category
18  ProfondeurPrel                       float64
19  CdDifficulteAna                      category
20  MnemoDifficulteAna                  category
21  LdAna                                float64
22  LqAna                                float64
23  LsAna                                float64
24  IncertAna                            float64
25  CdMetFractionnement                  category
26  NomMetFractionnement                 category
27  CdMethode                            category
28  NomMethode                           category
29  RdtExtraction                        float64
30  CdMethodeExtraction                  category
31  NomMethodeExtraction                 category
32  CdAccreAna                           category
33  MnemoAccredAna                       category
34  AgreAna                              float64
35  CdStatutAna                          category
36  MnemoStatutAna                       category
37  CdQualAna                            category
38  LbQualAna                            category
39  CommentairesAna                      category
40  ComResultatAna                       category
41  CdRdd                                category
42  NomRdd                               category
43  CdProducteur                         category
44  NomProducteur                        category
45  CdPreleveur                          category
46  NomPreleveur                         category
47  CdLaboratoire                       category

```

```

48 NomLaboratoire          category
dtypes: category(39), datetime64[ns](1), float64(8), object(1)
memory usage: 1.1+ GB

```

```
df_pc.shape
```

```
↗ (8917443, 49)
```

▼ Hydro-biologiques

```
df_hydrobio.info()
```

```

↗ <class 'pandas.core.frame.DataFrame'>
RangeIndex: 43535 entries, 0 to 43534
Data columns (total 21 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   Unnamed: 0                               43535 non-null  int64
1   CdStationMesureEauxSurface               43535 non-null  int64
2   LbStationMesureEauxSurface               43535 non-null  object
3   CdPointEauxSurf                         43115 non-null  float64
4   DateDebutOperationPrelBio               43535 non-null  object
5   CdSupport                                43535 non-null  int64
6   LbSupport                                43535 non-null  object
7   DtProdResultatBiologique                14829 non-null  object
8   CdParametreResultatBiologique           43535 non-null  int64
9   LbLongParametre                         43535 non-null  object
10  ResIndiceResultatBiologique              43522 non-null  float64
11  CdUniteMesure                            43535 non-null  object
12  SymUniteMesure                           43535 non-null  object
13  CdRqIndiceResultatBiologique             43535 non-null  int64
14  MnemoRqAna                              43535 non-null  object
15  CdMethEval                              28373 non-null  object
16  RefOperationPrelBio                     43535 non-null  object
17  CdProducteur                            43535 non-null  int64
18  NomProducteur                           43531 non-null  object
19  CdAccredRsIndiceResultatBiologique       43343 non-null  float64
20  MnAccredRsIndiceResultatBiologique       43343 non-null  object
dtypes: float64(3), int64(6), object(12)
memory usage: 7.0+ MB

```

```
df_hydrobio.shape
```

```
↗ (43535, 21)
```

✓ Préparation des données physicochimiques

✓ Démarche

Pour notre analyse des hydroécorégions, nous devons exploiter à la fois les données temporelles et spatiales à notre disposition. Notre principal objectif est d'utiliser les données temporelles pour effectuer un clustering, afin de déterminer si nous pouvons retrouver les hydroécorégions existantes à partir des caractéristiques physico-chimiques et biologiques disponibles.

Les données physicochimiques sont composées d'environ de 9 millions de points de mesure, ce qui nécessite une réduction importante pour les rendre exploitables, compte tenu des limites de nos ressources informatiques. Nous avons procédé en 2 étape :

- **Nettoyage des données** : La première étape a été le nettoyage des données. Les duplicatas et les valeurs aberrantes ont été supprimés. Cependant, compte tenu la quantité de données, nous ne pouvions pas visualiser l'intégralité ni effectuer des analyses sur chaque point individuellement. Nous avons donc mis en place des stratégies pour nettoyer les données sans pour autant perdre des informations significatives.
- **Agrégation des données** : Nous avons ensuite agrégé les données par stations, par saison et par année, afin de réduire la variabilité et de faciliter l'analyse des tendances. Pour chaque paramètre, nous avons calculé des statistiques résumées telles que la médiane et la moyenne, dans le but de conserver un maximum d'information en réduisant la taille de notre jeu de données. Nous avons également étudié les supports, les fractions, les unités de mesures, et les paramètres mesurés afin de filtrer et ne garder que les informations pertinentes pour l'étude. Et enfin, nous avons remodelé les données afin d'obtenir une table avec pour chaque station, année et saison, des valeurs agrégées pour chaque paramètre physicochimique.

✓ Nettoyage des données

Hypothèse : Nous faisons l'hypothèse que certaines colonnes dans notre jeu de données, telles que le laboratoire, le producteur, le préleveur, et le responsable de la collecte, ne sont pas directement pertinentes pour l'analyse des caractéristiques physicochimiques de l'eau. Ce jeu de données contient essentiellement des métadonnées relatives à la gestion des prélèvements qui ne renseignent pas sur l'état physique ou chimique de l'eau elle-même. De plus, puisque nous prévoyons d'agréger les données par saison et par année, la granularité temporelle de l'heure n'est pas pertinente, nous allons donc éliminer cette colonne.

```
# combien de données dupliquées ?
print(df_pc.duplicated().sum())
```

↗ 11836

```
# supprimer les doublons
df_pc.drop_duplicates(inplace=True)
```

✓ Caractériser une analyse physico chimique

Comme vu dans le TD, ce qui caractérise une analyse c'est :

- le support
- le paramètre
- la fraction analysée
- l'unité

```
print("Supports :", df_pc['LbSupport'].nunique(), "valeurs ->", df_pc['LbSupport'].nunique())
print("Paramètres :", df_pc['LbLongParamètre'].nunique(), "valeurs ->", df_pc['LbLongParamètre'].nunique())
print("Fractions :", df_pc['LbFractionAnalysee'].nunique(), "valeurs ->", df_pc['LbFractionAnalysee'].nunique())
print("Unités :", df_pc['SymUniteMesure'].nunique(), "valeurs ->", df_pc['SymUniteMesure'].nunique())
```

↗ Supports : 5 valeurs -> ['Eau', 'Air', 'Sédiments', 'Diatomées benthiques',
Paramètres : 16 valeurs -> ['Matières en suspension', 'Demande Biochimique',
Fractions : 2 valeurs -> ['Eau brute', "Phase aqueuse de l'eau (filtrée, ce
Unités : 15 valeurs -> ['mg/L', 'mg(O2)/L', '°C', 'unité pH', 'µS/cm', '%',

Pour mieux comprendre nos données physicochimiques, nous allons analyser la distribution des paramètres mesurés ainsi que leurs relations avec les fractions, supports, et unités de mesure. Nous examinerons également les relations entre fractions et supports, afin d'identifier les combinaisons les plus pertinentes et de garantir la cohérence des données pour l'analyse.

✓ Fractions et Supports

```
test = df_pc[['LbSupport', 'LbFractionAnalysee']].drop_duplicates()
```

```
print('Nombre de fractions par support')
test.groupby(['LbSupport']).count()
```

→ Nombre de fractions par support

LbFractionAnalysee	
LbSupport	
Eau	2
Air	2
Sédiments	2
Diatomées benthiques	1
Gammarès	1

Pour un support donné plusieurs fractions peuvent être analysées.

```
print('Nombre de supports par fraction')
test.groupby(['LbFractionAnalysee']).count()
```

→ Nombre de supports par fraction

LbSupport	
LbFractionAnalysee	
Eau brute	5
Phase aqueuse de l'eau (filtrée, centrifugée...)	3

Pour une fraction donnée plusieurs supports peuvent être analysés.

✓ Fractions et paramètres

```
test = df_pc[['LbLongParamètre', 'LbFractionAnalysee']].drop_duplicates()
```

```
print('Nombre de fractions par paramètre')
test.groupby(['LbLongParamètre']).count()
```



Nombre de fractions par paramètre

	LbFractionAnalysee
LbLongParamètre	
Azote Kjeldahl	1
Conductivité à 25°C	1
Demande Biochimique en oxygène en 5 jours (D.B.O.5)	1
Diuron	1
Matières en suspension	1
Oxygène dissous	1
Phosphore total	1
Potentiel en Hydrogène (pH)	1
Taux de saturation en oxygène	1
Température de l'Eau	1
Turbidité Formazine Néphélométrique	1
Carbone Organique	1
Ammonium	1
Nitrates	1
Nitrites	1
Orthophosphates (PO4)	1

Pour un paramètre donné un seul type de fraction est analysé.


```
print('Nombre de paramètres par fraction')
test.groupby(['LbFractionAnalysee']).count()
```

↩ Nombre de paramètres par fraction

LbLongParamètre	
LbFractionAnalysee	
Eau brute	11
Phase aqueuse de l'eau (filtrée, centrifugée...)	5

Pour une fraction donnée plusieurs paramètres peuvent être analysés.

✓ Supports et paramètres

```
test = df_pc[['LbLongParamètre', 'LbSupport']].drop_duplicates()
```

```
print('Nombre de supports par paramètre')
test.groupby(['LbLongParamètre']).count()
```

↔ Nombre de supports par paramètre

	LbSupport
LbLongParamètre	
Azote Kjeldahl	3
Conductivité à 25°C	4
Demande Biochimique en oxygène en 5 jours (D.B.O.5)	3
Diuron	3
Matières en suspension	3
Oxygène dissous	4
Phosphore total	3
Potentiel en Hydrogène (pH)	5
Taux de saturation en oxygène	4
Température de l'Eau	5
Turbidité Formazine Néphélométrique	1
Carbone Organique	1
Ammonium	3
Nitrates	3
Nitrites	3
Orthophosphates (PO4)	3

Pour un paramètre donné plusieurs supports peuvent être utilisés.

```
print('Nombre de paramètres par support')
test.groupby(['LbSupport']).count()
```



Nombre de paramètres par support

LbLongParamètre	
LbSupport	
Eau	16
Air	14
Sédiments	14
Diatomées benthiques	2
Gammares	5

Pour un support donné plusieurs paramètres peuvent être analysés.

✓ Unités de mesure et paramètres

```
test = df_pc[['LbLongParamètre', 'SymUniteMesure']].drop_duplicates()
```

```
print("Nombre d'unités par paramètre")
test.groupby(['LbLongParamètre']).count()
```

↔ Nombre d'unités par paramètre

	SymUniteMesure
LbLongParamètre	
Azote Kjeldahl	1
Conductivité à 25°C	1
Demande Biochimique en oxygène en 5 jours (D.B.O.5)	1
Diuron	1
Matières en suspension	1
Oxygène dissous	1
Phosphore total	1
Potentiel en Hydrogène (pH)	1
Taux de saturation en oxygène	1
Température de l'Eau	1
Turbidité Formazine Néphélométrique	1
Carbone Organique	1
Ammonium	1
Nitrates	1
Nitrites	1
Orthophosphates (PO4)	1

Pour un paramètre donné une seule unité de mesure est utilisée.

```
print("Nombre de paramètres par unité")
test.groupby(['SymUniteMesure']).count()
```

↔ Nombre de paramètres par unité

SymUniteMesure	LbLongParamètre
----------------	-----------------

SymUniteMesure	LbLongParamètre
%	1
NFU	1
mg(N)/L	1
mg(O2)/L	2
mg(P)/L	1
mg/L	1
unité pH	1
°C	1
μS/cm	1
μg/L	1
mg(C)/L	1
mg(NH4)/L	1
mg(NO2)/L	1
mg(NO3)/L	1
mg(PO4)/L	1

Pour une unité de mesure donnée plusieurs paramètres peuvent être analysés.

✓ Quelles informations pouvons-nous en extraire ?

Double-cliquez (ou appuyez sur Entrée) pour modifier

Pour chaque paramètre analysé, il n'existe qu'une seule unité de mesure associée. Nous n'avons donc pas besoin de conserver la colonne `SymUniteMesure` une fois que nous aurons intégré ces données dans notre dataframe pour l'analyse.

D'après les informations fournies dans le cours, un résultat d'analyse (`RsAna`) se caractérise principalement par :

- Un paramètre physico-chimique (`CdParametre`), comme les nitrates, phosphates, température, pH, ...
- Une unité de mesure (`CdUniteMesure`).
- Une fraction d'analyse (`CdFractionAnalysee`) réalisée à partir d'un support de prélèvement (`CdSupport`).

Étant donné que chaque paramètre est lié à une seule unité de mesure, nous allons pivoter notre table de données afin que chaque résultat d'analyse intègre simultanément le paramètre, le support, et la fraction. Cette organisation facilitera l'analyse et garantira une meilleure cohérence dans nos données.

```
# Pour un paramètre donné, combien y a t il d'unité de mesure, de support et de
print(df_pc.groupby('LbLongParametre')['SymUniteMesure'].nunique().value_counts)
print(df_pc.groupby('LbLongParametre')['LbSupport'].nunique().value_counts(), '
print(df_pc.groupby('LbLongParametre')['LbFractionAnalysee'].nunique().value_cc
```

```
↔ SymUniteMesure
1      16
Name: count, dtype: int64

LbSupport
3       9
4       3
5       2
1       2
Name: count, dtype: int64

LbFractionAnalysee
1      16
Name: count, dtype: int64
```

Il n'existe qu'un seul type de fraction analysée pour chaque paramètre.

Pour confirmer cela, nous allons regarder les combinaisons de paramètres, supports et fractions.

```
# Pour un paramètre et un support donné, combien de fraction différentes sont a
grouped = df_pc.groupby(['LbLongParamètre', 'LbSupport'])['LbFractionAnalysee'].
multiple_fractions = grouped[grouped > 1]
print(multiple_fractions)
```

```
Series([], Name: LbFractionAnalysee, dtype: int64)
```

Il n'y a au maximum qu'une seule fraction par combinaison de paramètre et de support. Cela signifie que la fraction peut être déduite directement à partir du couple (paramètre, support). Nous allons donc faire un regroupement par tuple (**paramètre, support**).

```
params = df_pc.groupby(["LbLongParamètre", 'LbSupport'])
```

```
# nombre de valeurs pour chaque tuple
params.size().sort_values(ascending=False)
```

```
LbLongParamètre      LbSupport      size
Potentiel en Hydrogène (pH)      Eau      675940
Température de l'Eau      Eau      675906
Conductivité à 25°C      Eau      668744
Oxygène dissous      Eau      632623
Taux de saturation en oxygène      Eau      613947
...
Turbidité Formazine Néphélométrique      Diatomées benthiques      0
      Gammares      0
Carbone Organique      Air      0
      Sédiments      0
Orthophosphates (P04)      Gammares      0
Length: 80, dtype: int64
```

```
# combien de résultats d'analyse pour chaque tuple ?
params_size = params.agg({'RsAna' : ['size']})
```

```
# regardons combien de lignes sont vides (RsAna = 0)
params_size[params_size[('RsAna', 'size')]==0]
```

```
RsAna
size

LbLongParamètre      LbSupport      size
Azote Kjeldahl      Diatomées benthiques      0
      Gammares      0
Conductivité à 25°C      Diatomées benthiques      0
```

Conductivité à 25 °C	Diatomées benthiques	0
Demande Biochimique en oxygène en 5 jours (D.B.O.5)	Diatomées benthiques	0
	Gammares	0
Diuron	Diatomées benthiques	0
	Gammares	0
Matières en suspension	Diatomées benthiques	0
	Gammares	0
Oxygène dissous	Diatomées benthiques	0
Phosphore total	Diatomées benthiques	0
	Gammares	0
Taux de saturation en oxygène	Diatomées benthiques	0
Turbidité Formazine Néphélométrique	Air	0
	Sédiments	0
	Diatomées benthiques	0
	Gammares	0
Carbone Organique	Air	0
	Sédiments	0
	Diatomées benthiques	0
	Gammares	0
Ammonium	Diatomées benthiques	0
	Gammares	0
Nitrates	Diatomées benthiques	0
	Gammares	0
Nitrites	Diatomées benthiques	0
	Gammares	0
Orthophosphates (PO4)	Diatomées benthiques	0
	Gammares	0

```
params_size[params_size[('RsAna','size')]!=0].sort_values(('RsAna','size'),asce
```


		size
LbLongParamètre	LbSupport	
Température de l'Eau	Gammare	1
Conductivité à 25°C	Gammare	1
Oxygène dissous	Gammare	1
Taux de saturation en oxygène	Gammare	1
Potentiel en Hydrogène (pH)	Gammare	1
	Diatomées benthiques	28
Température de l'Eau	Diatomées benthiques	28
Diuron	Sédiments	35
Azote Kjeldahl	Sédiments	60
Potentiel en Hydrogène (pH)	Sédiments	78
Phosphore total	Sédiments	78
Température de l'Eau	Sédiments	78
Ammonium	Sédiments	78
Nitrates	Sédiments	78
Nitrites	Sédiments	78
Taux de saturation en oxygène	Sédiments	78
Oxygène dissous	Sédiments	78
Orthophosphates (PO4)	Sédiments	78
Matières en suspension	Sédiments	78
Demande Biochimique en oxygène en 5 jours (D.B.O.5)	Sédiments	78
Conductivité à 25°C	Sédiments	78
Diuron	Air	1046
Azote Kjeldahl	Air	2136
Ammonium	Air	3059
Phosphore total	Air	3061
Demande Biochimique en oxygène en 5 jours (D.B.O.5)	Air	3145
Matières en suspension	Air	3148
Nitrates	Air	3149

Nitrates	Air	3149
Nitrites	Air	3149
Orthophosphates (PO4)	Air	3149
Taux de saturation en oxygène	Air	3169
Oxygène dissous	Air	3395
Température de l'Eau	Air	3402
Potentiel en Hydrogène (pH)	Air	3428
Conductivité à 25°C	Air	3432
Diuron	Eau	279124
Turbidité Formazine Néphélométrique	Eau	420400
Ammonium	Eau	509158
Azote Kjeldahl	Eau	525943
Nitrites	Eau	531995
Carbone Organique	Eau	535869
Demande Biochimique en oxygène en 5 jours (D.B.O.5)	Eau	542836
Orthophosphates (PO4)	Eau	543972
Nitrates	Eau	550019
Phosphore total	Eau	569020
Matières en suspension	Eau	587151
Taux de saturation en oxygène	Eau	613947
Oxygène dissous	Eau	632623
Conductivité à 25°C	Eau	668744
Température de l'Eau	Eau	675906
Potentiel en Hydrogène (pH)	Eau	675940

En examinant les mesures, nous pouvons voir que certains supports sont très faiblement représentés : une seule mesure pour les Support Gammares, 28 pour les diatomées benthiques, et maximum 78 pour les sédiments.

En comparaison avec les milliers de mesures pour l'air, et les centaines de milliers de mesures pour l'eau, nous risquons une sous représentation de ces 3 différents supports.

Pour gérer ce déséquilibre, il existe plusieurs stratégies :

- le **suréchantillonnage (over sampling)** pour augmenter artificiellement la présence des supports sous-représentés afin d'équilibrer les données.
- la **suppression des supports faiblement représentés** (ce qu'on a décidé de réaliser : plus tard dans l'analyse, nous supprimerons les supports.)

```
# Conversion en chaînes de caractères de notre colonne caractérisant les paramè
param_series = df_pc['LbLongParamètre'].astype(str) + ' - ' + df_pc['LbSupport']
```

✓ Traiter les seuils (de quantification, de saturation, de détection...)

```
df_pc[['CdRqAna', 'MnemoRqAna']].value_counts()
```

```

CdRqAna  MnemoRqAna
1        Résultat > seuil de quantification et < au seuil de saturation ou
Résultat = 0    7764758
10       Résultat < au seuil de quantification
1125177
0        Analyse non faite
10400
2        Résultat < seuil de détection
3157
7        Traces (< seuil de quantification et > seuil de détection)
1812
3        Résultat > seuil de saturation
237
8        Dénombrement > Valeur
64
9        Dénombrement < Valeur
1
11       Echelle Absente
1
Name: count, dtype: int64
```

d'après la doc :

- 1 = ok : rien besoin de faire
- 0 : analyse non faite -> le résultat doit être vide : nous on va supprimer ces lignes
- 2 : le résultat prend alors la valeur du seuil de détection ou du seuil de quantification suivant qu'il est inférieur à l'un de ces deux seuils
- 3 : le résultat donne alors la valeur du seuil de saturation
- 7 : le résultat prend alors la valeur du seuil de détection ou du seuil de quantification suivant qu'il est inférieur à l'un de ces deux seuils. (comme 2)
- 8 : Les codes remarque 8 et 9 doivent être utilisés pour qualifier des résultats fournis par des méthodes de type qualitatif, décrits par rapport à un seuil bien que compris dans la plage d'utilisation courante des méthodes (supérieur au seuil de quantification et inférieur au seuil de saturation).
- 9 : Les codes remarque 8 et 9 doivent être utilisés pour qualifier des résultats fournis par des méthodes de type qualitatif, décrits par rapport à un seuil bien que compris dans la plage d'utilisation courante des méthodes (supérieur au seuil de quantification et inférieur au seuil de saturation).
- 10 : Le résultat quant à lui prend la valeur du seuil de quantification.
- 11 : pas de doc -> supprimer cette valeur

```
print("Avant suppression:", df_pc.shape)
df_pc = df_pc[~df_pc['CdRqAna'].isin(['0', '8', '9', '11'])]
print("Après suppression:", df_pc.shape)
print(df_pc[['CdRqAna', 'MnemoRqAna']].value_counts())
```

```
↩ Avant suppression: (8905607, 49)
Après suppression: (8895141, 49)
CdRqAna MnemoRqAna
1 Résultat > seuil de quantification et < au seuil de saturation ou
10 Résultat < au seuil de quantification
2 Résultat < seuil de détection
7 Traces (< seuil de quantification et > seuil de détection)
3 Résultat > seuil de saturation
Name: count, dtype: int64
```

Nous créons maintenant le dataframe `df_pc_light`, contenant seulement les colonnes pertinentes pour associer un résultat d'analyse à une station, une date de prélèvement et un paramètre.

```
df_pc_light = df_pc[['CdStationMesureEauxSurface', 'DatePrel', 'RsAna']]
df_pc_light['param'] = param_series
df_pc_light
```

 /var/folders/p1/4yqk4cyx3058m3j04rtkr56m0000gn/T/ipykernel_53919/3914632016


A value is trying to be set on a copy of a slice from a DataFrame.
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs>

	CdStationMesureEauxSurface	DatePrel	RsAna	param
0	05005600	2005-07-06	3.000	Matières en suspension - Eau
1	05200115	2005-09-28	0.700	Demande Biochimique en oxygène en 5 jours (D.B...
2	05001800	2005-01-19	8.100	Température de l'Eau - Eau
3	05001800	2005-01-19	7.800	Potentiel en Hydrogène (pH) - Eau
4	05001800	2005-01-19	845.000	Conductivité à 25°C - Eau
...
8917438	06133330	2022-10-25	0.010	Nitrites - Eau
8917439	06133330	2022-10-25	0.960	Nitrates - Eau
...

```
# supprimer les lignes où pour un paramètre donné on a moins de 1000 valeurs (c
df_pc_light = df_pc_light.groupby('param').filter(lambda x: x['RsAna'].count()
```

```
# afficher les lignes où RsAna est vide
df_pc_light[df_pc_light['RsAna'].isnull()]
```

 CdStationMesureEauxSurface DatePrel RsAna param

```
# afficher les doublons
df_pc_light[df_pc_light.duplicated()]
```

	CdStationMesureEauxSurface	DatePrel	RsAna	param
21474	05215100	2005-07-04	16.3	Température de l'Eau - Eau
35195	02018000	2005-05-03	15.5	Température de l'Eau - Eau
35196	02018000	2005-05-03	7.9	Potentiel en Hydrogène (pH) - Eau
35199	02018000	2005-05-03	9.0	Oxygène dissous - Eau
35581	02025500	2005-05-09	11.3	Température de l'Eau - Eau
...
8878024	06213370	2022-11-15	13.0	Température de l'Eau - Eau
8878932	06002058	2022-11-15	483.0	Conductivité à 25°C - Eau
...

```
# supprimer les doublons (de nouveau, on avait déjà supprimé les doublons mais
print("Avant suppression:", df_pc_light.shape)
df_pc_light.drop_duplicates(inplace=True)
print("Après suppression:", df_pc_light.shape)
```

```
Avant suppression: (8894049, 4)
Après suppression: (8810625, 4)
```

```
# pour un paramètre donné à une station donné à une date donnée, y a t il plusi
duplicates = df_pc_light[df_pc_light.duplicated(subset=['CdStationMesureEauxSur
duplicates.shape
```

```
(40987, 4)
```

Il est possible d'avoir plusieurs résultats d'analyse pour un même paramètre à une station donnée, à une date donnée. Cette situation se produit dans environ 41 000 cas. Plutôt que de supprimer ces doublons potentiels, nous choisissons de les conserver, car nous allons agréger les données par date et par station par la suite.

✓ Traiter les valeurs aberrantes dans les données physicochimiques

Lors de l'exploration des données physicochimiques, nous avons constaté la présence de valeurs aberrantes pour certains paramètres, par exemple, une température de l'eau enregistrée à -9999.0. Compte tenu du volume important des données, afficher tous les outliers s'est avéré trop lourd et inefficace. Cependant, il est clair que ces cas doivent être traités pour garantir la qualité de l'analyse.

Nous avons étudié plusieurs manières pour identifier et traiter ces valeurs aberrantes :

- La méthode de l'intervalle interquartile (IQR)
- Le z-score avec différents seuils

La méthode retenue est le z-score avec un seuil de 3.

```
df_pc_cleaned = df_pc_light.copy()
outliers_summary_before = []
for param in df_pc_cleaned['param'].unique():
    df_param = df_pc_cleaned[df_pc_cleaned['param'] == param].copy()
    df_param['zscore'] = zscore(df_param['RsAna'])
    outliers_before = df_param[abs(df_param['zscore']) > 3]
    outliers_summary_before.append({
        "param": param,
        "total_values": len(df_param),
        "outliers_count_before": len(outliers_before),
        "outliers_percent_before": len(outliers_before) / len(df_param) * 100,
    })
# Imputation des outliers avec la médiane
median_value = df_param['RsAna'].median()
df_param.loc[abs(df_param['zscore']) > 3, 'RsAna'] = median_value
df_pc_cleaned.loc[df_pc_cleaned['param'] == param, 'RsAna'] = df_param['RsAna']
outliers_summary_before_df = pd.DataFrame(outliers_summary_before)
print("Résumé des outliers avant imputation :")
print(outliers_summary_before_df)
```

➡ Résumé des outliers avant imputation :

	param	total_values	\
0	Matières en suspension – Eau	584610	
1	Demande Biochimique en oxygène en 5 jours (D.B...	541160	
2	Température de l'Eau – Eau	656418	
3	Potentiel en Hydrogène (pH) – Eau	652628	
4	Conductivité à 25°C – Eau	649121	
5	Oxygène dissous – Eau	629463	
6	Taux de saturation en oxygène – Eau	610779	
7	Phosphore total – Eau	565122	

8	Turbidité Formazine Néphélométrique – Eau	419605
9	Azote Kjeldahl – Eau	524334
10	Diuron – Eau	278229
11	Carbone Organique – Eau	534038
12	Ammonium – Eau	507401
13	Nitrites – Eau	530105
14	Nitrates – Eau	545844
15	Orthophosphates (P04) – Eau	539900
16	Ammonium – Air	3059
17	Azote Kjeldahl – Air	2136
18	Taux de saturation en oxygène – Air	3169
19	Oxygène dissous – Air	3395
20	Température de l'Eau – Air	3402
21	Potentiel en Hydrogène (pH) – Air	3428
22	Conductivité à 25°C – Air	3432
23	Matières en suspension – Air	3148
24	Demande Biochimique en oxygène en 5 jours (D.B...) – Air	3145
25	Nitrites – Air	3149
26	Nitrates – Air	3149
27	Orthophosphates (P04) – Air	3149
28	Phosphore total – Air	3061
29	Diuron – Air	1046

	outliers_count_before	outliers_percent_before
0	1460	0.249739
1	535	0.098862
2	4	0.000609
3	386	0.059145
4	3651	0.562453
5	100	0.015887
6	11424	1.870398
7	2275	0.402568
8	801	0.190894
9	3126	0.596185
10	605	0.217447
11	3287	0.615499
12	1513	0.298186
13	3338	0.629687
14	5782	1.059277
15	3855	0.714021
16	47	1.536450
17	40	1.872659
18	50	1.577785
19	27	0.795287
20	2	0.058789
21	33	0.962660
22	36	1.048951
23	46	1.461245
24	51	1.621622

Pour chaque paramètre, nous avons calculé les z-scores des résultats d'analyse et identifié les valeurs dont le z-score dépassait 3. Les outliers identifiés ont été remplacés par la médiane des valeurs du paramètre correspondant.

Moins de 2% des données ont été identifiées comme des outliers pour la majorité des paramètres, nous pensons donc avoir limité la perte d'information.

Seul un paramètre, le Diuron - Air, a présenté un taux d'outliers légèrement supérieur à 2% (2,29%). Ce paramètre est également celui avec le moins de valeurs (environ 1 000).

Nous estimons que cette approche permet de conserver la fiabilité des données tout en traitant efficacement les valeurs aberrantes.

✓ Aggrégation des données physicochimiques par saison

✓ Première visualisation des données agrégées

```
df_pc_season = df_pc_cleaned.copy()
df_pc_season['année'] = df_pc_season['DatePre1'].dt.year + (df_pc_season['DatePre1'].dt.month.map({
    12: 'Hiver', 1: 'Hiver', 2: 'Hiver',
    3: 'Printemps', 4: 'Printemps', 5: 'Printemps',
    6: 'Été', 7: 'Été', 8: 'Été',
    9: 'Automne', 10: 'Automne', 11: 'Automne'
}))
df_counts = df_pc_season.groupby(['année', 'saison']).size().reset_index(name='')
```

`df_counts`

	année	saison	nombre de résultats d'analyse par saison
0	2005	Automne	56550
1	2005	Hiver	27418
2	2005	Printemps	47644
3	2005	Été	56389
4	2006	Automne	62823
...
68	2022	Automne	59629
69	2022	Hiver	98052
70	2022	Printemps	82245
71	2022	Été	65427
72	2023	Hiver	19532

73 rows × 3 columns

```
season_colors = {  
    "Printemps": "#77DD77",  
    "Été": "#FFB347",  
    "Automne": "#FF6961",  
    "Hiver": "#AEC6CF"  
}
```

```
fig = px.bar(df_counts,
             x='année',
             y='nombre de résultats d\'analyse par saison',
             color='saison',
             barmode='stack',
             labels={'Année': 'année', 'Nombre de valeurs': 'Nombre de Mesures'},
             title='Nombre de Mesures par Année et Saison',
             color_discrete_map=season_colors
            )
fig.update_layout(
    xaxis_title='année',
    yaxis_title='nombre de résultats d\'analyse',
    legend_title='saison'
)
fig.show()
```



Nous avons plus de 300 000 résultats par an à partir de 2007 jusqu'en 2021, avec une répartition relativement équilibrée entre les saisons. Pour les années 2005, 2006 et 2022, le nombre de résultats est plus faible, autour de 200 000 résultats par an.

Pour simplifier l'analyse tout en conservant les tendances, nous avons choisi d'agréger les données par station, année et saison.

Pour chaque combinaison de station, année, saison, et paramètre (paramètre - support), nous calculons des statistiques représentatives : médiane et moyenne.

Nous créons un nouveau DataFrame composé des valeurs agrégées. Chaque ligne représentera une station, une année, une saison, et contiendra les valeurs médianes et moyennes pour chaque paramètre mesuré dans les colonnes `median_RsAna` et `mean_RsAna`.

Cela permet de réduire la complexité des données tout en conservant leur représentativité.

✓ Aggrégation par station, année et saison

```
df_pc_season.rename(columns={'CdStationMesureEauxSurface': 'station', 'DatePre1
```

```
agg_functions = { 'RsAna': ['mean', 'median'], }
df_aggregated = df_pc_season.groupby(['station', 'param', 'saison', 'année']).agg(agg_functions)
df_aggregated.columns = ['station', 'param', 'saison', 'année', 'mean_RsAna', 'median_RsAna']
print(f"Taille du DataFrame après l'agrégation avec différentes fonctions : {df_aggregated.shape}")
```

➞ Taille du DataFrame après l'agrégation avec différentes fonctions : 2008680

```
df_aggregated.isnull().sum()
```

```
➞ station      0
   param      0
   saison      0
   année      0
   mean_RsAna  15972975
   median_RsAna 15972975
   dtype: int64
```

Après l'agrégation, le nombre total de valeurs a augmenté de manière significative (environ 14 millions de lignes). Cela suggère qu'il y a de nombreuses valeurs nulles, et nous allons les supprimer.

```
# supprimer toutes les lignes avec des valeurs nulles
df_aggregated.dropna(inplace=True)
df_aggregated.shape
```

➞ (4113825, 6)

```
# calculer combien de données en moins ça fait par rapport à avant aggregation
print("Nombre de valeurs en moins par rapport à avant agrégation :", df_pc_season.shape[0] - df_aggregated.shape[0])
print("Ce qui représente une perte de :", (df_pc_season.shape[0] - df_aggregated.shape[0]) / df_pc_season.shape[0] * 100)
```

➞ Nombre de valeurs en moins par rapport à avant agrégation : 4696800
Ce qui représente une perte de : 53.308363481591826 %

Nous avons alors 4 millions de lignes, ce qui est bien plus gérable pour réaliser nos analyses.

✓ Création du dataframe agrégé par saison des données physicochimiques

Nous effectuons une nouvelle transformation du dataframe pour obtenir un résumé agrégé des résultats. L'objectif est de calculer, pour chaque station et chaque saison, les statistiques représentatives des paramètres physicochimiques mesurés.

Nous créons 2 dataframes distincts :

- un contenant les moyennes des résultats d'analyse pour chaque paramètre
- un autre contenant les médianes des résultats d'analyse pour chaque paramètre

Ce pivotement permet de structurer les données de manière claire et synthétique, facilitant ainsi les analyses ultérieures.

```
params = df_aggregated['param'].unique()
```

```
df_pc_agg_saison_median = df_aggregated[['station', 'année', 'saison', 'param',  
df_pc_agg_saison_mean = df_aggregated[['station', 'année', 'saison', 'param', '  
# Pivot des données : station, année, saison en index, param en colonnes  
df_pc_pivot_saison_median = df_aggregated.pivot_table(index=['station', 'année'  
                                                    columns='param',  
                                                    values='median_RsAna',  
                                                    aggfunc='median')  
df_pc_pivot_saison_mean = df_aggregated.pivot_table(index=['station', 'année',  
                                                    columns='param',  
                                                    values='mean_RsAna',  
                                                    aggfunc='mean')  
df_pc_pivot_saison_median.reset_index(inplace=True)  
df_pc_pivot_saison_mean.reset_index(inplace=True)
```

✓ Analyse des données agrégées avec la médiane

```
df_pc_pivot_saison_median.head(5)
```



	param	station	année	saison	Ammonium - Air	Ammonium - Eau	Azote Kjeldahl - Air	Azote Kjeldahl - Eau	Ca Orga
0		05001800	2005	Automne	NaN	NaN	NaN	NaN	
1		05001800	2005	Hiver	NaN	NaN	NaN	NaN	
2		05001800	2005	Printemps	NaN	NaN	NaN	NaN	
3		05001800	2005	Été	NaN	NaN	NaN	NaN	
4		05001800	2007	Automne	NaN	0.025	NaN	1.0	

5 rows × 33 columns

```
print(f"Taille du DataFrame pivoté : {df_pc_pivot_saison_median.shape[0]}")
```



Taille du DataFrame pivoté : 292196

▼ Gestion des valeurs nulles

```
# afficher les valeurs nulles
df_pc_pivot_saison.median.isnull().sum()
```

```
↔ param
station 0
année 0
saison 0
Ammonium - Air 290188
Ammonium - Eau 44641
Azote Kjeldahl - Air 290809
Azote Kjeldahl - Eau 42131
Carbone Organique - Eau 36462
Conductivité à 25°C - Air 289987
Conductivité à 25°C - Eau 6910
Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Air 290115
Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau 30477
Diuron - Air 291423
Diuron - Eau 157613
Matières en suspension - Air 290114
Matières en suspension - Eau 12243
Nitrates - Air 290114
Nitrates - Eau 31344
Nitrites - Air 290114
Nitrites - Eau 35222
Orthophosphates (P04) - Air 290114
Orthophosphates (P04) - Eau 31970
Oxygène dissous - Air 289987
Oxygène dissous - Eau 12305
Phosphore total - Air 290187
Phosphore total - Eau 22900
Potentiel en Hydrogène (pH) - Air 289987
Potentiel en Hydrogène (pH) - Eau 5512
Taux de saturation en oxygène - Air 290154
Taux de saturation en oxygène - Eau 18128
Température de l'Eau - Air 289985
Température de l'Eau - Eau 5620
Turbidité Formazine Néphélométrique - Eau 95299
dtype: int64
```

```
# pourcentage de valeurs nulles pour chaque paramètre
null_percentages = df_pc_pivot_saison_median.isnull().mean() * 100
null_percentages
```

```
⇒ param
station 0.000000
année 0.000000
saison 0.000000
Ammonium - Air 99.312790
Ammonium - Eau 15.277759
Azote Kjeldahl - Air 99.525319
Azote Kjeldahl - Eau 14.418746
Carbone Organique - Eau 12.478610
Conductivité à 25°C - Air 99.244001
Conductivité à 25°C - Eau 2.364851
Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Air 99.287807
Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau 10.430328
Diuron - Air 99.735452
Diuron - Eau 53.940848
Matières en suspension - Air 99.287465
Matières en suspension - Eau 4.189996
Nitrates - Air 99.287465
Nitrates - Eau 10.727046
Nitrites - Air 99.287465
Nitrites - Eau 12.054238
Orthophosphates (P04) - Air 99.287465
Orthophosphates (P04) - Eau 10.941286
Oxygène dissous - Air 99.244001
Oxygène dissous - Eau 4.211214
Phosphore total - Air 99.312448
Phosphore total - Eau 7.837205
Potentiel en Hydrogène (pH) - Air 99.244001
Potentiel en Hydrogène (pH) - Eau 1.886405
Taux de saturation en oxygène - Air 99.301154
Taux de saturation en oxygène - Eau 6.204055
Température de l'Eau - Air 99.243316
Température de l'Eau - Eau 1.923367
Turbidité Formazine Néphélométrique - Eau 32.614752
dtype: float64
```



```
# Représentation des pourcentages de valeurs nulles par paramètre
null_percentages = df_pc_pivot_saison_median.isnull().mean() * 100
null_percentages_df = null_percentages.reset_index()
null_percentages_df.columns = ['Paramètre', 'Pourcentage de valeurs nulles']
null_percentages_df = null_percentages_df.sort_values(by='Pourcentage de valeur
fig = px.bar(null_percentages_df, x='Paramètre', y='Pourcentage de valeurs null
fig.update_traces(texttemplate='%{text:.2f}%', textposition='outside')
fig.update_layout(xaxis_tickangle=45, width=1000, height=600, showlegend=False)
fig.show()
```



```
# liste des paramètres avec plus de 90% de valeurs nulles
params_with_most_nulls = null_percentages[null_percentages > 90].index.tolist()
params_with_most_nulls
```

```
↔ ['Ammonium - Air',
    'Azote Kjeldahl - Air',
    'Conductivité à 25°C - Air',
    'Demande Biochimique en oxygène en 5 jours (D.B.0.5) - Air',
    'Diuron - Air',
    'Matières en suspension - Air',
    'Nitrates - Air',
    'Nitrites - Air',
    'Orthophosphates (P04) - Air',
    'Oxygène dissous - Air',
    'Phosphore total - Air',
    'Potentiel en Hydrogène (pH) - Air',
    'Taux de saturation en oxygène - Air',
    "Température de l'Eau - Air"]
```

Nous avons décidé de supprimer les colonnes contenant plus de 90% de valeurs nulles, car un pourcentage aussi élevé de valeurs manquantes indique que ces paramètres sont rarement mesurés pour une station donnée à une saison donnée.

```
# supprimer les colonnes avec plus de 90% de valeurs nulles
df_pc_pivot_saison_median.drop(columns=params_with_most_nulls, inplace=True)
```

```
# Affichage du nombre de lignes où il n'y a que la saison et l'année qui ne sor
# Et donc où toutes les valeurs des paramètres sont nuls
exclude_columns = ['station', 'année', 'saison']
columns_to_check = [col for col in df_pc_pivot_saison_median.columns if col not
rows_with_all_nulls = df_pc_pivot_saison_median[df_pc_pivot_saison_median[column
rows_with_all_nulls.head(3)
```



param	station	année	saison	Ammonium - Eau	Azote Kjeldahl - Eau	Carbone Organique - Eau	Conductivité à 25°C - Eau
35512	03017000	2005	Automne	NaN	NaN	NaN	NaN
35513	03017000	2005	Hiver	NaN	NaN	NaN	NaN
35656	03024392	2005	Hiver	NaN	NaN	NaN	NaN

```
rows_with_all_nulls.shape
```



```
(517, 19)
```

Il y a environ 500 lignes ne contenant aucune information, nous les supprimons.


```
# supprimer les lignes où il n'y a que param saison et année qui ne sont pas n
print("Nombre de lignes avant suppression :", df_pc_pivot_saison_median.shape[0]
df_pc_pivot_saison_median = df_pc_pivot_saison_median[~df_pc_pivot_saison_medie
print("Nombre de lignes après suppression :", df_pc_pivot_saison_median.shape[0]
```



```
Nombre de lignes avant suppression : 292196
Nombre de lignes après suppression : 291679
```

✓ Analyse des données agrégées avec la moyenne

```
df_pc_pivot_saison_mean.head(5)
```




	param	station	année	saison	Ammonium - Air	Ammonium - Eau	Azote Kjeldahl - Air	Azote Kjeldahl - Eau	Ca Orga
0		05001800	2005	Automne	NaN	NaN	NaN	NaN	
1		05001800	2005	Hiver	NaN	NaN	NaN	NaN	
2		05001800	2005	Printemps	NaN	NaN	NaN	NaN	
3		05001800	2005	Été	NaN	NaN	NaN	NaN	
4		05001800	2007	Automne	NaN	0.025	NaN	1.0	

5 rows × 33 columns

Nous pouvons faire le même constat que pour les données agrégées par médiane, et nous procédons donc au même nettoyage pour les données agrégées par moyenne.

```
# supprimer les colonnes avec plus de 90% de valeurs nulles
df_pc_pivot_saison_mean.drop(columns=params_with_most_nulls, inplace=True)
```

```
# supprimer les lignes où il n'y a que param saison et année qui ne sont pas nu
print("Nombre de lignes avant suppression :", df_pc_pivot_saison_mean.shape[0])
df_pc_pivot_saison_mean = df_pc_pivot_saison_mean[~df_pc_pivot_saison_mean[coli
print("Nombre de lignes après suppression :", df_pc_pivot_saison_mean.shape[0])
```



```
Nombre de lignes avant suppression : 292196
Nombre de lignes après suppression : 291679
```

✓ Analyse exploratoire des données hydrobiologiques

✓ Démarche

Dans cette partie, nous allons analyser et traiter les données hydrobiologiques afin de les préparer pour les intégrer correctement avec les données physicochimiques.

Nous avons conservé seulement les informations utiles pour notre analyse, notamment l'indicateur biologique I2M2, les identifiants des stations pour permettre la liaison avec les autres tables, et les dates de prélèvement.

Ensuite, nous avons structuré les données en les agrégeant par station, saison et année. Nous avons créé deux ensembles de données : l'un avec les médianes des paramètres pour chaque station sur une saison et une année données, et l'autre avec les moyennes. Cette agrégation a permis de réduire la taille globale des données, tout en diminuant la variabilité et en équilibrant le nombre d'enregistrements entre les périodes.

Et enfin, pour tenir compte du délai entre les changements physicochimiques et leur impact sur les indices biologiques, nous avons introduit des décalages temporels (lags) de 1 mois, 3 mois, 6 mois et 1 an. Pour chaque décalage, nous avons généré de nouveaux ensembles de datasets.

✓ Nettoyage des données

```
df_hydrobio.head(3)
```



	Unnamed: 0	CdStationMesureEauxSurface	LbStationMesureEauxSurface	CdPoint
0	0	2000990	LE LERTZBACH À HEGENHEIM	
1	1	2001000	L'AUGRABEN À BARTENHEIM	
2	2	2001000	L'AUGRABEN À BARTENHEIM	

3 rows × 21 columns

```
df_hydrobio.shape
```

```
(43535, 21)
```

Nous pouvons déjà voir qu'il y a beaucoup moins de données hydrobiologiques par rapport aux données physicochimiques (8917443 lignes dans le dataset physicochimiques).

```
# Valeurs manquantes
df_hydrobio.isnull().sum()
```

```

Unnamed: 0      0
CdStationMesureEauxSurface      0
LbStationMesureEauxSurface      0
CdPointEauxSurf      420
DateDebutOperationPrelBio      0
CdSupport      0
LbSupport      0
DtProdResultatBiologique      28706
CdParametreResultatBiologique      0
LbLongParametre      0
ResIndiceResultatBiologique      13
CdUniteMesure      0
SymUniteMesure      0
CdRqIndiceResultatBiologique      0
MnemoRqAna      0
CdMethEval      15162
RefOperationPrelBio      0
CdProducteur      0
NomProducteur      4
CdAccredRsIndiceResultatBiologique      192
MnAccredRsIndiceResultatBiologique      192
dtype: int64

```

```
# Nombre de valeurs uniques pour chaque colonne
df_hydrobio.nunique()
```

```
↳ Unnamed: 0          43535
   CdStationMesureEauxSurface    9489
   LbStationMesureEauxSurface    9389
   CdPointEauxSurf              26
   DateDebutOperationPrelBio     2366
   CdSupport                    1
   LbSupport                    1
   DtProdResultatBiologique      54
   CdParametreResultatBiologique 1
   LbLongParametre              1
   ResIndiceResultatBiologique   9111
   CdUniteMesure                 3
   SymUniteMesure                3
   CdRqIndiceResultatBiologique  2
   MnemoRqAna                   2
   CdMethEval                   7
   RefOperationPrelBio          43535
   CdProducteur                 262
   NomProducteur                248
   CdAccredRsIndiceResultatBiologique 3
   MnAccredRsIndiceResultatBiologique 3
   dtype: int64
```

```
# Recherche des correspondances multiples entre deux colonnes
def afficher_correspondances_multiples(df, col_code, col_libelle):
    multiples = df.groupby(col_code)[col_libelle].nunique()
    codes_avec_multiples_libelles = multiples[multiples > 1].index

    if len(codes_avec_multiples_libelles) > 0:
        print(f"Codes dans '{col_code}' avec plusieurs valeurs dans '{col_libelle}'")
        print(df[df[col_code].isin(codes_avec_multiples_libelles)][[col_code, col_libelle]])
    else:
        print(f"Aucune correspondance multiple trouvée entre '{col_code}' et '{col_libelle}'")
```

```
# Vérification des unités de mesure
print("Valeurs uniques CdUniteMesure : ", df_hydrobio['CdUniteMesure'].unique())
print("Valeurs uniques SymUniteMesure : ", df_hydrobio['SymUniteMesure'].unique())
afficher_correspondances_multiples(df_hydrobio, 'CdUniteMesure', 'SymUniteMesure')
```

```
↳ Valeurs uniques CdUniteMesure : ['X' '214' '0']
   Valeurs uniques SymUniteMesure : ['X' 'n' 'Unité inconnue']
   Aucune correspondance multiple trouvée entre 'CdUniteMesure' et 'SymUniteMesure'
```

```
afficher_correspondances_multiples(df_hydrobio, 'CdStationMesureEauxSurface', '

```

```
⇒ Codes dans 'CdStationMesureEauxSurface' avec plusieurs valeurs dans 'LbStat
      CdStationMesureEauxSurface  LbStationMesureEauxSurface
2588                6073500    LEYSSE A LE-BOURGET-DU-LAC
2986                6135500                ARLY A FLUMET
4066                6580830    DEISSE A GRESY-SUR-AIX
42528               6073500    LEYSSE A LE-BOURGET-DU-LAC 1
42750               6135500                ARLY A FLUMET 1
43397               6580830    DEISSE A GRESY-SUR-AIX 1

```

```
# Renommage des stations ayant le même CdStationMesureEauxSurface et des libellés
codes_a_corriger = [6073500, 6135500, 6580830]
df_hydrobio.loc[df_hydrobio['CdStationMesureEauxSurface'].isin(codes_a_corriger)
df_hydrobio.loc[df_hydrobio['CdStationMesureEauxSurface'].isin(codes_a_corr

```

```
# Vérification
afficher_correspondances_multiples(df_hydrobio, 'CdStationMesureEauxSurface', '

```

```
⇒ Aucune correspondance multiple trouvée entre 'CdStationMesureEauxSurface' e

```

Une grande partie des colonnes de ce jeu de données ne sont pas pertinentes pour l'analyse et nous les supprimerons par la suite.

Voici les colonnes identifiées comme inutiles :

- **Unnamed: 0** : Colonne d'index sans valeur analytique.
- **DtProdResultatBiologique** : Date (jour, mois, année) de calcul du résultat biologique, non utile pour notre analyse et contenant des valeurs manquantes dans 1/4 des cas.
- **CdSupport**, **LbSupport**, **CdParametreResultatBiologique**, **LbLongParametre** : Colonne avec une seule valeur possible et aucune donnée manquante, donc non informatives.
- **RefOperationPrelBio** : Sert uniquement à des jointures avec des tables externes que nous n'utilisons pas.
- **CdAccredRsIndiceResultatBiologique**, **CdProducteur**, **NomProducteur**, **MnAccredRsIndiceResultatBiologique** : Métadonnées administratives sans utilité pour notre analyse.
- **CdUniteMesure**, **SymUniteMesure** : Colonne contenant trois valeurs possibles, mais toutes indiquant l'absence d'unité, donc sans intérêt analytique.

```
df_hydrobio.rename(columns={'CdStationMesureEauxSurface': 'station', 'DateDebut

```

```
df_hydrobio['date'] = pd.to_datetime(df_hydrobio['date'], format='%Y-%m-%d')
df_hydrobio['date'].dtype
```

```
dtype('<M8[ns]')
```

```
# supprimer les I2M2 manquants et les doublons
print("avant suppression :", df_hydrobio.shape)
df_hydrobio.dropna(subset=['I2M2'], inplace=True)
df_hydrobio.drop_duplicates(inplace=True)
print("après suppression :", df_hydrobio.shape)
```

```
avant suppression : (43535, 21)
après suppression : (43522, 21)
```

```
# Gestion des seuils
df_hydrobio['MnemoRqAna'].value_counts()
```

```
MnemoRqAna
Résultat > seuil de quantification et < au seuil de saturation    43522
Name: count, dtype: int64
```

Le seuil est respecté pour l'ensemble des données.

```
# Statistiques descriptives de I2M2
desc_stats = df_hydrobio['I2M2'].describe()
desc_stats.drop(['count'], inplace=True)
fig = go.Figure()
fig.add_trace(go.Bar(x=desc_stats.index, y=desc_stats.values))
fig.update_layout(height=500, width=400, font_size=10, title="Statistique descr")
fig.show()
```

Le paramètre I2M2 ne comporte pas d'outliers, toutes les valeurs sont comprises entre 0 et 1.

✓ Aggrégation par saison


```
df_hydrobio_saison = df_hydrobio.copy()
df_hydrobio_saison['année'] = df_hydrobio_saison['date'].dt.year + (df_hydrobio_saison['date'].dt.month.map({
    12: 'Hiver', 1: 'Hiver', 2: 'Hiver',
    3: 'Printemps', 4: 'Printemps', 5: 'Printemps',
    6: 'Été', 7: 'Été', 8: 'Été',
    9: 'Automne', 10: 'Automne', 11: 'Automne'
}))
df_hydrobio_saison = df_hydrobio_saison[['station', 'année', 'saison', 'I2M2']]
df_hydrobio_saison
```



	station	année	saison	I2M2
0	2000990	2010	Été	0.4726
1	2001000	2010	Automne	0.3481
2	2001000	2011	Été	0.4253
3	2001000	2012	Été	0.2460
4	2001025	2010	Été	0.1137
...
43530	6999107	2009	Été	0.1820
43531	6999125	2008	Été	0.1180
43532	6999125	2009	Été	0.1580
43533	6999180	2008	Été	0.3530
43534	6999180	2009	Été	0.4150

43522 rows × 4 columns

```
df_counts = df_hydrobio_saison.groupby(['année', 'saison']).size().reset_index()
```

✓ Observation des données

```
fig = px.bar(df_counts, x='année', y="nombre de résultats d'analyse par saison")
fig.update_layout(xaxis_title='année', yaxis_title="nombre de résultats d'analyse")
fig.show()
```



Nous pouvons voir que la répartition des prélèvements est très différente que celle des données physicochimiques qui étaient uniformément réparties sur toutes les saisons. Ici, nous constatons une forte dominance des prélèvements réalisés en été. Les autres saisons sont beaucoup moins représentées, avec très peu de résultats enregistrés en hiver, un nombre légèrement supérieur au printemps, et encore légèrement supérieur en automne. Avant 2008, le nombre de prélèvements était inférieur à 1 000 par an. Jusqu'en 2014, ce nombre a progressivement augmenté à moins de 2 000 par an, pour finalement doubler après 2014, avec une concentration importante des prélèvements en été.

```
# Seuils de l'I2M2
seuils_I2M2 = {
    "Très bon": 0.665,
    "Bon": 0.443,
    "Moyen": 0.295,
    "Médiocre": 0.148,
    "Mauvais": 0.0
}
# trouvés ici : https://www.labocea.fr/indice-invertebres-multi-metriques-i2m2/
```

```
# Statistiques descriptives
desc_stats_saison = df_hydrobio_saison.groupby('saison')['I2M2'].describe()
desc_stats_saison = desc_stats_saison.drop(columns=['count']).transpose()
fig = go.Figure()
for saison in desc_stats_saison.columns:
    fig.add_trace(go.Bar(x=desc_stats_saison.index, y=desc_stats_saison[saison]
for etat, valeur in seuils_I2M2.items():
    fig.add_shape(type="line", x0=-0.5, x1=len(desc_stats)-0.5, y0=valeur, y1=valeur)
    fig.add_annotation(x=len(desc_stats)-0.5, y=valeur, text=etat, showarrow=False)
fig.update_layout(height=500, width=700, font_size=10, title="Statistiques descriptives")
fig.show()
```



Nous pouvons voir que les saisons influencent clairement l'indice biologique I2M2, avec une qualité biologique généralement plus favorable en été et en automne. Ce résultat pourrait être lié à des variations naturelles des conditions environnementales, des paramètres physicochimiques, ou à des variations des activités humaines impactant la qualité de l'eau.

✓ Ajout d'un décalage temporelle

Pour notre analyse, nous avons introduit un décalage temporel initial arbitraire de 1 mois entre les données physicochimiques et hydrobiologiques.

Ce choix repose sur l'hypothèse que les variations des paramètres physicochimiques influencent l'indice I2M2, mais que cet impact n'est pas immédiat. En effet, les changements physicochimiques, comme une hausse de la température ou une variation des nutriments, nécessitent un certain temps pour se refléter dans la composition biologique de l'eau.

Nous prévoyons également de tester des décalages supplémentaires de 3 mois, 6 mois et 1 an, afin d'explorer l'effet différé des variations physicochimiques sur la qualité biologique, et d'identifier le délai optimal reflétant le mieux ces interactions.

```
# Puisque les données physicochimiques sont censées avoir un impact sur les dor
# on simule un "retour dans le temps" pour les données hydrobiologiques
# c'est à dire que si on fait un relevé en décembre des données hydrobiologique
# on veut l'associer aux données de novembre des données physicochimiques
# si le lag est de 1 mois.
df_hydrobio['date_lag_1_month'] = df_hydrobio['date'] - pd.Timedelta(days=30)
df_hydrobio['date_lag_3_month'] = df_hydrobio['date'] - pd.Timedelta(days=90)
df_hydrobio['date_lag_6_month'] = df_hydrobio['date'] - pd.Timedelta(days=180)
```

```

df_hydrobio_lag_1_month = df_hydrobio.copy()
df_hydrobio_lag_3_month = df_hydrobio.copy()
df_hydrobio_lag_6_month = df_hydrobio.copy()

df_hydrobio_lag_1_month['année'] = df_hydrobio_lag_1_month['date_lag_1_month'].
df_hydrobio_lag_1_month['saison'] = df_hydrobio_lag_1_month['date_lag_1_month'].
    12: 'Hiver', 1: 'Hiver', 2: 'Hiver',
    3: 'Printemps', 4: 'Printemps', 5: 'Printemps',
    6: 'Été', 7: 'Été', 8: 'Été',
    9: 'Automne', 10: 'Automne', 11: 'Automne'
})

df_hydrobio_lag_3_month['année'] = df_hydrobio_lag_3_month['date_lag_3_month'].
df_hydrobio_lag_3_month['saison'] = df_hydrobio_lag_3_month['date_lag_3_month'].
    12: 'Hiver', 1: 'Hiver', 2: 'Hiver',
    3: 'Printemps', 4: 'Printemps', 5: 'Printemps',
    6: 'Été', 7: 'Été', 8: 'Été',
    9: 'Automne', 10: 'Automne', 11: 'Automne'
})

df_hydrobio_lag_6_month['année'] = df_hydrobio_lag_6_month['date_lag_6_month'].
df_hydrobio_lag_6_month['saison'] = df_hydrobio_lag_6_month['date_lag_6_month'].
    12: 'Hiver', 1: 'Hiver', 2: 'Hiver',
    3: 'Printemps', 4: 'Printemps', 5: 'Printemps',
    6: 'Été', 7: 'Été', 8: 'Été',
    9: 'Automne', 10: 'Automne', 11: 'Automne'
})

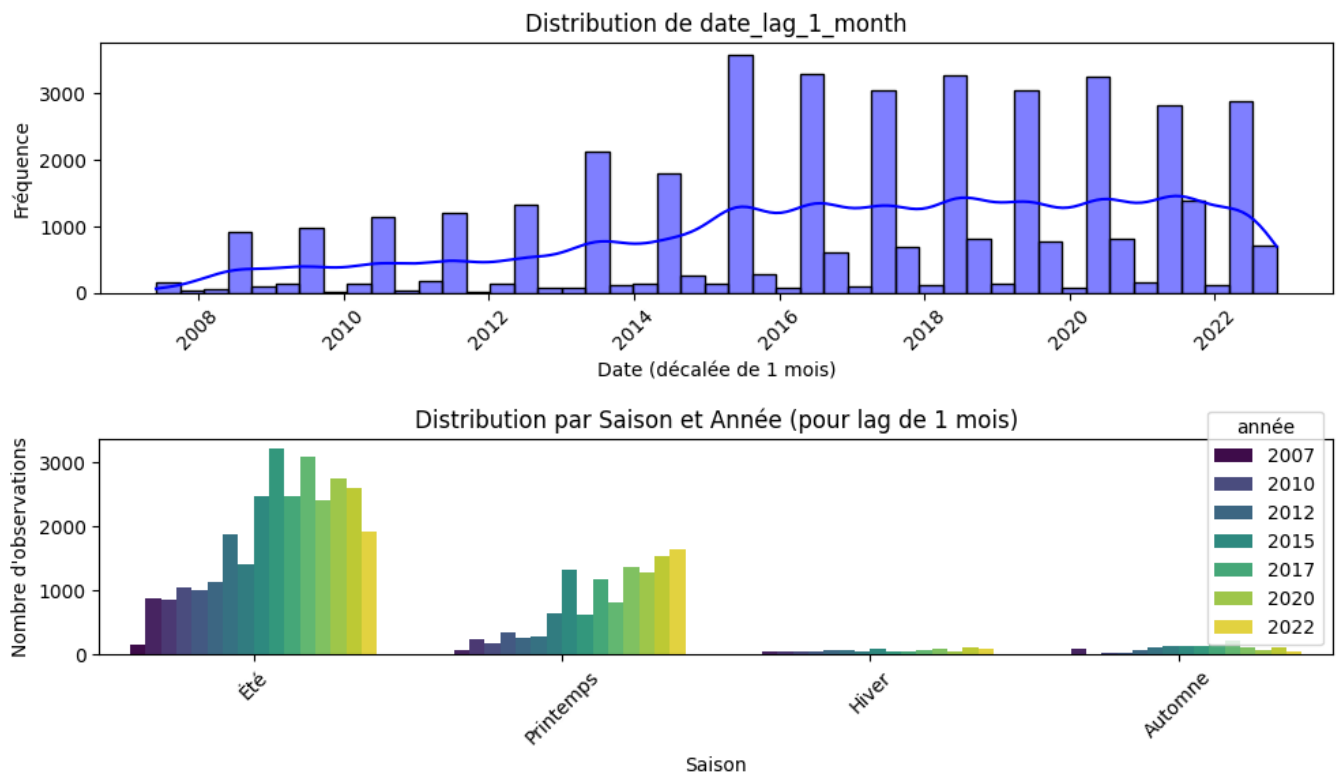
```

```

# Distribution de df_hydrobio['date_lag_1_month']
plt.figure(figsize=(10, 3))
sns.histplot(df_hydrobio['date_lag_1_month'], kde=True, color='blue')
plt.title('Distribution de date_lag_1_month')
plt.xlabel('Date (décalée de 1 mois)')
plt.ylabel('Fréquence')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Distribution par saison et année
plt.figure(figsize=(10, 3))
sns.countplot(x='saison', hue='année', data=df_hydrobio_lag_1_month, palette='v')
plt.title('Distribution par Saison et Année (pour lag de 1 mois)')
plt.xlabel('Saison')
plt.ylabel('Nombre d\'observations')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

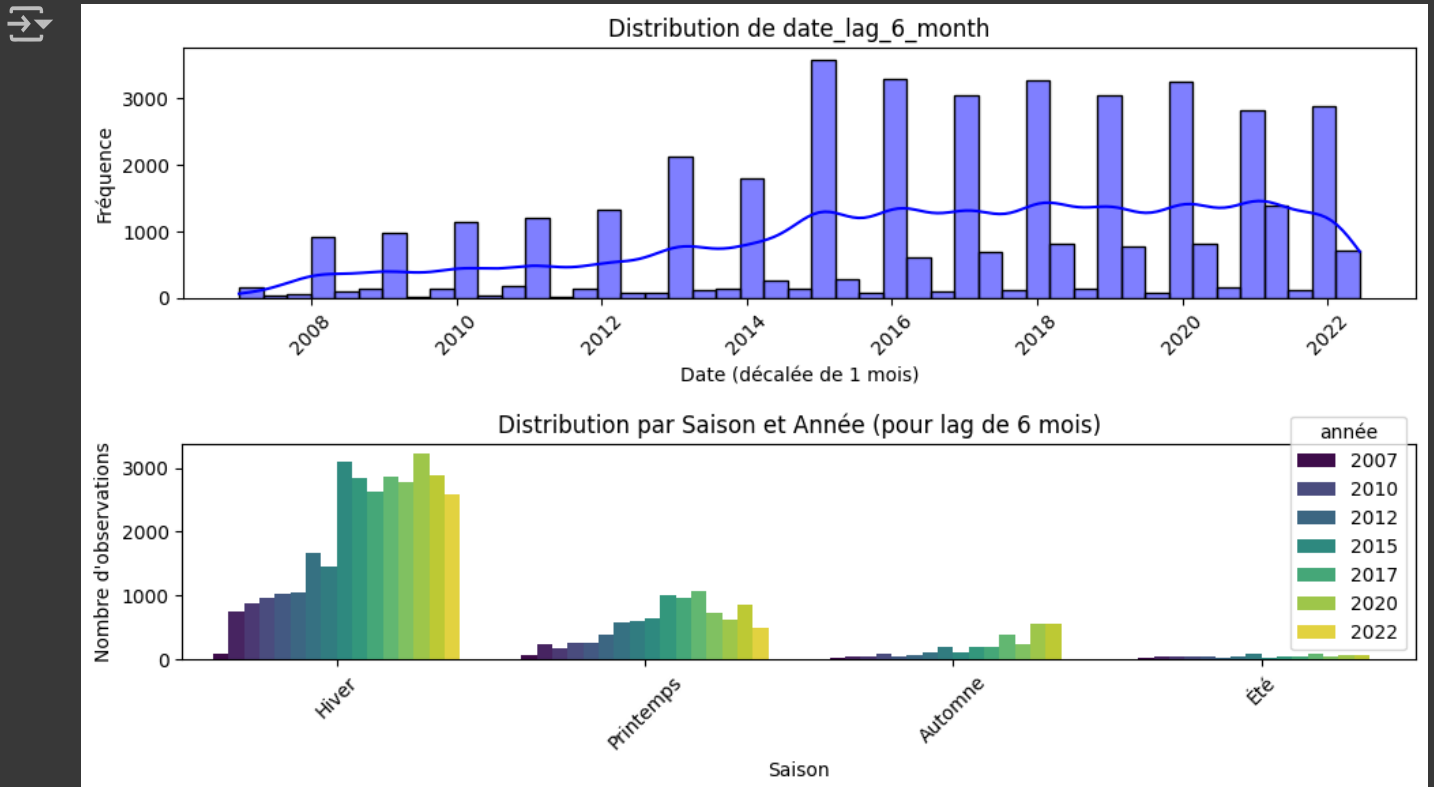
```



```
# Distribution de df_hydrobio['date_lag_1_month']
plt.figure(figsize=(10, 3))
sns.histplot(df_hydrobio['date_lag_6_month'], kde=True, color='blue')
plt.title('Distribution de date_lag_6_month')
plt.xlabel('Date (décalée de 1 mois)')
plt.ylabel('Fréquence')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Distribution par saison et année
plt.figure(figsize=(10, 3))
sns.countplot(x='saison', hue='année', data=df_hydrobio_lag_6_month, palette='
plt.title('Distribution par Saison et Année (pour lag de 6 mois)')
plt.xlabel('Saison')
plt.ylabel('Nombre d\'observations')
```

```
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Les lags ont bien l'air effectifs.

```
df_hydrobio_lag_1_month_median = df_hydrobio_lag_1_month.groupby(['station', 'a
df_hydrobio_lag_3_month_median = df_hydrobio_lag_3_month.groupby(['station', 'a
df_hydrobio_lag_6_month_median = df_hydrobio_lag_6_month.groupby(['station', 'a
df_hydrobio_lag_1_month_mean = df_hydrobio_lag_1_month.groupby(['station', 'anr
df_hydrobio_lag_3_month_mean = df_hydrobio_lag_3_month.groupby(['station', 'anr
df_hydrobio_lag_6_month_mean = df_hydrobio_lag_6_month.groupby(['station', 'anr
```

```
df_hydrobio_lag_1_month_median.head(5)
```



	station	année	saison	I2M2
0	1000274	2016	Été	0.243
1	1000274	2017	Printemps	0.231
2	1000274	2018	Été	0.205
3	1000274	2019	Printemps	0.236
4	1000274	2020	Été	0.096

✓ Intégration des données : Jointure des tables physicochimiques, hydrobiologiques, stations

✓ Démarche

Nous avons joint les différentes tables de données (physicochimiques, hydrobiologiques, stations et hydroécorégions) pour constituer un dataset complet destiné aux analyses.

La première étape a consisté à nettoyer les colonnes relatives aux stations (`CdStationMesureEauxSurface` et `LbStationMesureEauxSurface`) dans chaque dataset, en corrigeant les identifiants et en vérifiant la cohérence des libellés. Ce nettoyage a été appliqué à l'ensemble des tables pour garantir leur compatibilité.

Nous avons ensuite utilisé le dataset `df_pc_median_bio_median_1_month`, qui regroupe les données physicochimiques et hydrobiologiques agrégées par médiane, avec un décalage temporel de 1 mois pour les indices hydrobiologiques, comme base pour toutes les étapes suivantes de l'analyse.

Enfin, le dataset nettoyé `df_pc_median_bio_median_1_month` a été joint aux données des hydroécorégions en fonction des coordonnées géographiques des stations, permettant ainsi de localiser les stations dans leurs hydroécorégions respectives.

✓ Observation et nettoyage des colonnes relatives aux stations

```
# copie des dataframes pour nettoyer les colonnes station
# on garde juste cd et lb pour les stations
df_stations_join = df_stations.copy()
df_stations_join = df_stations_join[['CdStationMesureEauxSurface', 'LbStationMe
df_pc_join = df_pc.copy()
df_pc_join = df_pc[['CdStationMesureEauxSurface', 'LbStationMesureEauxSurface']]
df_hydrobio_join = df_hydrobio.copy()
df_hydrobio_join = df_hydrobio[['station', 'LbStationMesureEauxSurface']].copy(
df_hydrobio_join.rename(columns={'station': 'CdStationMesureEauxSurface'}, inplace=True)
```

```
# afficher un échantillon des station pour chaque dataset
print("Stations dans df_stations")
print(df_stations_join['CdStationMesureEauxSurface'].dtype)
print(df_stations_join['CdStationMesureEauxSurface'].unique())
print()
print("Stations dans df_pc")
print(df_pc_join['CdStationMesureEauxSurface'].dtype)
print(df_pc_join['CdStationMesureEauxSurface'].unique())
print()
print("Stations dans df_hydrobio")
print(df_hydrobio_join['CdStationMesureEauxSurface'].dtype)
print(df_hydrobio_join['CdStationMesureEauxSurface'].unique())
```

```
⇒ Stations dans df_stations
object
['01000477' '01000602' '01000605' ... 'Y9205023' 'Y9715083' 'Y9905043']

Stations dans df_pc
category
['05005600', '05200115', '05001800', '05004000', '05005000', ..., '06001219'
Length: 8809
Categories (8810, object): ['05001800', '05004000', '05005000', '05005350',

Stations dans df_hydrobio
int64
[2000990 2001000 2001025 ... 6580040 6710037 6820126]
```

Ces colonnes doivent être modifiées pour être compatibles.

Selon la documentation, les codes des stations devraient être des entiers composés de 8 chiffres.

Nous avons donc décidé de nettoyer les identifiants pour respecter ce format.


```
def clean_station_column(df, column_name):
    df[column_name] = df[column_name].astype(str).str.extract(r'(\d+)')[0]
    df[column_name] = pd.to_numeric(df[column_name], errors='coerce')
    return df
df_stations_join = clean_station_column(df_stations_join, 'CdStationMesureEauxS
df_pc_join = clean_station_column(df_pc_join, 'CdStationMesureEauxSurface')
df_hydrobio_join = clean_station_column(df_hydrobio_join, 'CdStationMesureEauxS
```

```
# on crée un dataframe avec l'identifiant de chaque station, et pour les 3 data
# on affiche le libellé associé, de cette manière on peut vérifier si les stati
merged_data = pd.merge(df_pc_join, df_hydrobio_join, on='CdStationMesureEauxSur
merged_all = pd.merge(merged_data, df_stations_join, on='CdStationMesureEauxSur
# caster tous les Lb en object
merged_all['LbStationMesureEauxSurface_pc'] = merged_all['LbStationMesureEauxSu
merged_all['LbStationMesureEauxSurface'] = merged_all['LbStationMesureEauxSurfa
merged_all['LbStationMesureEauxSurface_hydrobio'] = merged_all['LbStationMesure
merged_all.dtypes
```

```
⇒ CdStationMesureEauxSurface      int64
   LbStationMesureEauxSurface_pc    object
   LbStationMesureEauxSurface_hydrobio  object
   LbStationMesureEauxSurface        object
   dtype: object
```

✓ Étude des cas où les libellés sont différents

```
label_diff_pc_hydrobio = merged_all[merged_all['LbStationMesureEauxSurface_pc']]
label_diff_pc_hydrobio = label_diff_pc_hydrobio.drop_duplicates()
label_diff_pc_hydrobio
```



	CdStationMesureEauxSurface	LbStationMesureEauxSurface_pc	LbStationMesureEauxSurface
142208	5197200	Le Majesq à Azur	
468800	6083000	BOURBRE A CHAVANOZ	
490122	6800003	EYGUES A ST-MAURICE/EYGUES - LES CIVARDIERES	EYGUES A ST-MAURICE
645026	6208900	MOURACHONNE A PEGOMAS	
1980639	6580640	BIEF D'ENFER A ST ETIENNE SUR REYSSOUZE	
2024300	6139405	BUGEON A LA-CHAMBRE	
2595740	5068920	La Véronne en aval de Riom-ès- Montagnes (Amont...)	La Véronne en aval de Riom-ès-Montagnes
6148279	6830030	TORRENSON A ST-CYR	
10095956	6080995	AGNY A NIVOLAS-VERMELLE	
13028690	6014250	SANSFOND A SAULON-LA-RUE	
15703808	6213230	RUISSEAU LE THOUX A CURIS AU MONT D'OR	RUISSEAU LE THOUX A CURIS AU MONT D'OR
16150583	3250952	LE COURS D'EAU NUMÉRO 01 DE LA BÉLINIÈRE A CON...	LE COURS D'EAU NUMÉRO 01 DE LA BÉLINIÈRE A CON...


Les labels sont différents mais ont l'air de correspondre à la même chose.

```
label_diff_pc_station = merged_all[merged_all['LbStationMesureEauxSurface_pc']]
label_diff_pc_station = label_diff_pc_station.loc[:, ['CdStationMesureEauxSurface', 'LbStationMesureEauxSurface_pc']]
label_diff_pc_station = label_diff_pc_station.drop_duplicates()
label_diff_pc_station
```



	CdStationMesureEauxSurface	LbStationMesureEauxSurface_pc	LbStationMesureEauxSurface_pc
6156	5010000	Le Pharaon à St-Pardon	
11396	5015100	Le Charreau à St-Michel	
77052	5116100	La Séoune à Montjoi	
475147	6580578	AIGUE NOIRE A DOMESSIN	RUIS
505611	6010000	OGNON A PESMES 1	
...	
24098987	1000602	COLOGNE À BUIRE COURCELLES (80)	
24105377	1001131	HELPE MINEURE À GRAND FAYT (59)	HELPE M
26676374	4108492	LONG À DISSAY-SOUS-COURCILLON	F
26687107	4406063	LE BONSON A PERIGNEUX	

```
label_diff_pc_station.head(20)
```

	CdStationMesureEauxSurface	LbStationMesureEauxSurface_pc	LbStatic
6156	5010000	Le Pharaon à St-Pardon	
11396	5015100	Le Charreau à St-Michel	
77052	5116100	La Séoune à Montjoi	
475147	6580578	AIGUE NOIRE A DOMESSIN	RUISS
505611	6010000	OGNON A PESMES 1	
545763	6143950	ROMANCHE A BOURG D'OISANS - LE PONT ROUGE 2	D'OIS,
597663	6135350	PLANAY A MEGEVE 1	
598125	6070460	NANT AILLON A AILLON-LE-VIEUX	NANT I
611335	6144900	ROMANCHE A JARRIE 1	
644341	6078500	TIER A BELMONT-TRAMONET 1	TIER A
668593	6106935	DORNE A DORNAS 2	
677174	6580563	BONNARD A SAINT-BERON 1	RUISSE
700205	6055000	BREVENNE A SAIN-BEL 1	E
706232	6202100	GAPEAU A SIGNES 1	
723439	6065675	COPPY A MAXILLY-SUR-LEMAN 1	F
1082462	5015250	Le Charreau à Torsac	
1629659	6085500	BIENNE A JEURRE 1	
1667982	6047500	PETITE GROSNE A MACON 1	PFT

```
# afficher les données de 20 à 40
label_diff_pc_station[20:40]
```



	CdStationMesureEauxSurface	LbStationMesureEauxSurface_pc	LbStatic
1733672	6178800	ORBIEL A LES-MARTYS	OR
2029742	6176130	AUDE A LIMOUX 1	
2595740	5068920	La Véronne en aval de Riom-ès-Montagnes (Amont...	La Véro
2606716	5074920	Le Gat-Mort à Villagrains	
3071988	6115080	IBIE A LAGORCE 1	IBIE /
3174351	6084360	AIN A MESNOIS 1	
3299210	6182120	HERAULT A PUECHABON 1	HE
3327085	6179550	ARGENT DOUBLE A AZILLE 1	ARG
3334156	6464800	CLAUGE A LA-LOYE 1	
3346565	6580960	GRESSE A VARCES-ALLIERES-ET-RISSET 3	GRESSE
3528228	6178050	SOUPEX A SOUILHE	
3534277	6830180	RISSE A ST-JEOIRE 1	F
3634331	6168200	MASSANE A ARGELES-SUR-MER 1	MASS
3635010	6148850	SAVASSE A ROMANS-SUR-ISERE 2	SAVASSE
3969301	3096650	LA CHEE A MERLAUT 1	
5600749	6107760	DUNIERE A SILHAC 1	
5869307	6165700	EZE A PERTUIS 3	
5933441	6830800	MORGE A VAILLIERES 2	I

```
# 40 à 60
label_diff_pc_station[40:60]
```



	CdStationMesureEauxSurface	LbStationMesureEauxSurface_pc	LbStationMesureEauxSurface
6175174	6041700	BRENNE A SENS-SUR-SEILLE 1	BRENNE A SENS-SUR-SEILLE 1
7416343	6820139	GIER A L'HORME 1	GIER A L'HORME 1
7422432	6580794	JANON A ST-CHAMOND 1	JANON A ST-CHAMOND 1
7550442	6820144	MORNANTE A ST-CHAMOND 1	MORNANTE A ST-CHAMOND 1
7718952	6580793	RICOLIN A ST-CHAMOND 1	RICOLIN A ST-CHAMOND 1
7837573	6436900	CORCELLE A RIGNEY 1	CORCELLE A RIGNEY 1
10399011	6084210	DROUVENANT A BOISSIA 1	DROUVENANT A BOISSIA 1
10627070	6070750	DADON A RUMILLY 1	DADON A RUMILLY 1
10660253	6041280	LEMME A LAC DES ROUGES TRUITES 1	LEMME A LAC DES ROUGES TRUITES 1
10674197	6830122	FILLIERE A FILLIERE 1	FILLIERE A FILLIERE 1
12499089	6491650	SEDAN A VILLEVIEUX 1	SEDAN A VILLEVIEUX 1
12500544	6440850	GRAVELLON A THERVAY 1	GRAVELLON A THERVAY 1
12996658	6440770	FONTAINE DE MAGNEY A SORNAY 1	FONTAINE DE MAGNEY A SORNAY 1
13050103	6440750	FONTAINE DE DOUIS RU A MARNAY 1	FONTAINE DE DOUIS RU A MARNAY 1
13071231	6440950	VEZE A VITREUX 1	VEZE A VITREUX 1
14008955	5211550	La Lèze à Monein	La Lèze à Monein
16150583	3250952	LE COURS D'EAU NUMÉRO 01 DE LA BÉLINIÈRE A CON...	LE COURS D'EAU NUMÉRO 01 DE LA BÉLINIÈRE A CON...

```
# 60 à la fin
label_diff_pc_station[60:]
```



	CdStationMesureEauxSurface	LbStationMesureEauxSurface_pc	LbStati
17816874	6028100	MARAIS A SAONE 1	RI
18204892	6461520	BREVILLIERS A HERICOURT 2	RUISS
22555554	6083710	LEMME A ENTRE DEUX MONTS 1	LEMME
23433166	6068410	DORCHE A CHANAY 1	L
24028092	1002228	LA TERNOISE À TILLY CAPELLE (62)	LA TERI
24097129	1002222	LA RIVIERETTE À LE FAVRIL (59)	LA RIVIE
24098987	1000602	COLOGNE À BUIRE COURCELLES (80)	
24105377	1001131	HELPE MINEURE À GRAND FAYT (59)	HELPE M
		LONG À DISSAY-SOUS-	F

Les différences entre les labels des stations semblent être uniquement dues à des fautes de frappe ou à des abréviations différentes.

Pour vérifier, nous avons localisé les stations sur une carte afin de confirmer leur correspondance :

- 16580582 : vérifié, correspond bien.
- 6115080 : vérifié, correspond bien.
- 6178050 : vérifié, correspond bien.
- 6830800 : vérifié, correspond bien.
- 6830122 : vérifié, correspond bien.

Cependant, pour le code 4406063, nous n'avons pas pu vérifier, nous avons donc décidé de la supprimer par précaution.

✓ Appliquer le nettoyage aux différents dataframes

```
# On applique à df_stations, au dataframe de physicochimique et au dataframe d'
df_stations.rename(columns={'CdStationMesureEauxSurface': 'station'}, inplace=True)
df_stations = clean_station_column(df_stations, 'station')
df_pc_pivot_saison_mean = clean_station_column(df_pc_pivot_saison_mean, 'station')
df_pc_pivot_saison_median = clean_station_column(df_pc_pivot_saison_median, 'station')
df_hydrobio_lag_1_month_median = clean_station_column(df_hydrobio_lag_1_month_median, 'station')
df_hydrobio_lag_3_month_median = clean_station_column(df_hydrobio_lag_3_month_median, 'station')
df_hydrobio_lag_6_month_median = clean_station_column(df_hydrobio_lag_6_month_median, 'station')
df_hydrobio_lag_1_month_mean = clean_station_column(df_hydrobio_lag_1_month_mean, 'station')
df_hydrobio_lag_3_month_mean = clean_station_column(df_hydrobio_lag_3_month_mean, 'station')
df_hydrobio_lag_6_month_mean = clean_station_column(df_hydrobio_lag_6_month_mean, 'station')
# supprimer les lignes avec 4406063
df_stations = df_stations[df_stations['station'] != 4406063]
df_pc_pivot_saison_mean = df_pc_pivot_saison_mean[df_pc_pivot_saison_mean['station'] != 4406063]
df_pc_pivot_saison_median = df_pc_pivot_saison_median[df_pc_pivot_saison_median['station'] != 4406063]
df_hydrobio_lag_1_month_median = df_hydrobio_lag_1_month_median[df_hydrobio_lag_1_month_median['station'] != 4406063]
df_hydrobio_lag_3_month_median = df_hydrobio_lag_3_month_median[df_hydrobio_lag_3_month_median['station'] != 4406063]
df_hydrobio_lag_6_month_median = df_hydrobio_lag_6_month_median[df_hydrobio_lag_6_month_median['station'] != 4406063]
df_hydrobio_lag_1_month_mean = df_hydrobio_lag_1_month_mean[df_hydrobio_lag_1_month_mean['station'] != 4406063]
df_hydrobio_lag_3_month_mean = df_hydrobio_lag_3_month_mean[df_hydrobio_lag_3_month_mean['station'] != 4406063]
df_hydrobio_lag_6_month_mean = df_hydrobio_lag_6_month_mean[df_hydrobio_lag_6_month_mean['station'] != 4406063]
```

✓ Aggrégation des données physicochimiques et hydrobiologiques

Nous avons fusionné les données physicochimiques et hydrobiologiques (avec un décalage d'un mois) agrégées par médiane dans un seul dataset, en utilisant les colonnes communes `station`, `année`, et `saison`.

💡 À partir de cette étape, toutes les analyses seront effectuées sur ce dataset fusionné `df_pc_median_bio_median_1_month`.

```
df_pc_median_bio_median_1_month = pd.merge(df_pc_pivot_saison_median, df_hydrobio_lag_1_month_mean, on=['station', 'année', 'saison'], how='inner')
```



```
df_pc_median_bio_median_1_month.head(5)
```



	station	année	saison	Ammonium - Eau	Azote Kjeldahl - Eau	Carbone Organique - Eau	Conductivité à 25°C - Eau	D Bioch en o en 5 (D.B.
0	5001800	2007	Été	0.04	1.0	3.5	765.0	
1	5001800	2008	Été	0.04	1.0	2.7	765.0	
2	5001800	2009	Été	0.02	1.0	2.2	736.0	
3	5004000	2007	Été	0.04	1.0	2.8	683.0	
4	5004000	2008	Été	0.03	1.0	2.1	689.0	

```
df_pc_median_bio_median_1_month.shape
```



```
(37966, 20)
```

```
# Statistiques descriptives
desc_stats_saison = df_pc_median_bio_median_1_month.groupby('saison')['I2M2'].c
desc_stats_saison = desc_stats_saison.drop(columns=['count']).transpose()
fig = go.Figure()
for saison in desc_stats_saison.columns:
    fig.add_trace(go.Bar(x=desc_stats_saison.index, y=desc_stats_saison[saison]
for etat, valeur in seuils_I2M2.items():
    fig.add_shape(type="line", x0=-0.5, x1=len(desc_stats)-0.5, y0=valeur, y1=valeur)
    fig.add_annotation(x=len(desc_stats)-0.5, y=valeur, text=etat, showarrow=False)
fig.update_layout(height=500, width=700, font_size=10, title="Statistiques descriptives")
fig.show()
```



Après la fusion des datasets, les statistiques descriptives de l'I2M2 restent assez similaires. La fusion n'a donc pas altéré les tendances principales de l'I2M2.

✓ Aggrégation du dataset avec les données des stations et des hydroécorégions

Nous allons maintenant fusionner notre dataset avec celui des stations afin de récupérer les coordonnées géographiques des stations et déterminer leur appartenance aux hydroécorégions.

Ensuite, nous vérifierons le nombre de stations restantes dans les hydroécorégions après la fusion de tous les datasets. Cette étape de fusion a réduit le nombre total de stations en raison des différences dans les codes de stations, qui n'étaient pas toujours compatibles pour faire une jointure.

```
# Fusion avec le dataset des stations pour récupérer les coordonnées des stations
df_pc_median_bio_median_1_month_with_coords = pd.merge(
    df_pc_median_bio_median_1_month,
    df_stations[['station', 'CoordXStationMesureEauxSurface', 'CoordYStationMesureEauxSurface']],
    on='station',
    how='inner'
)
```

✓ Observation de la répartition des stations dans les hydroécorégions

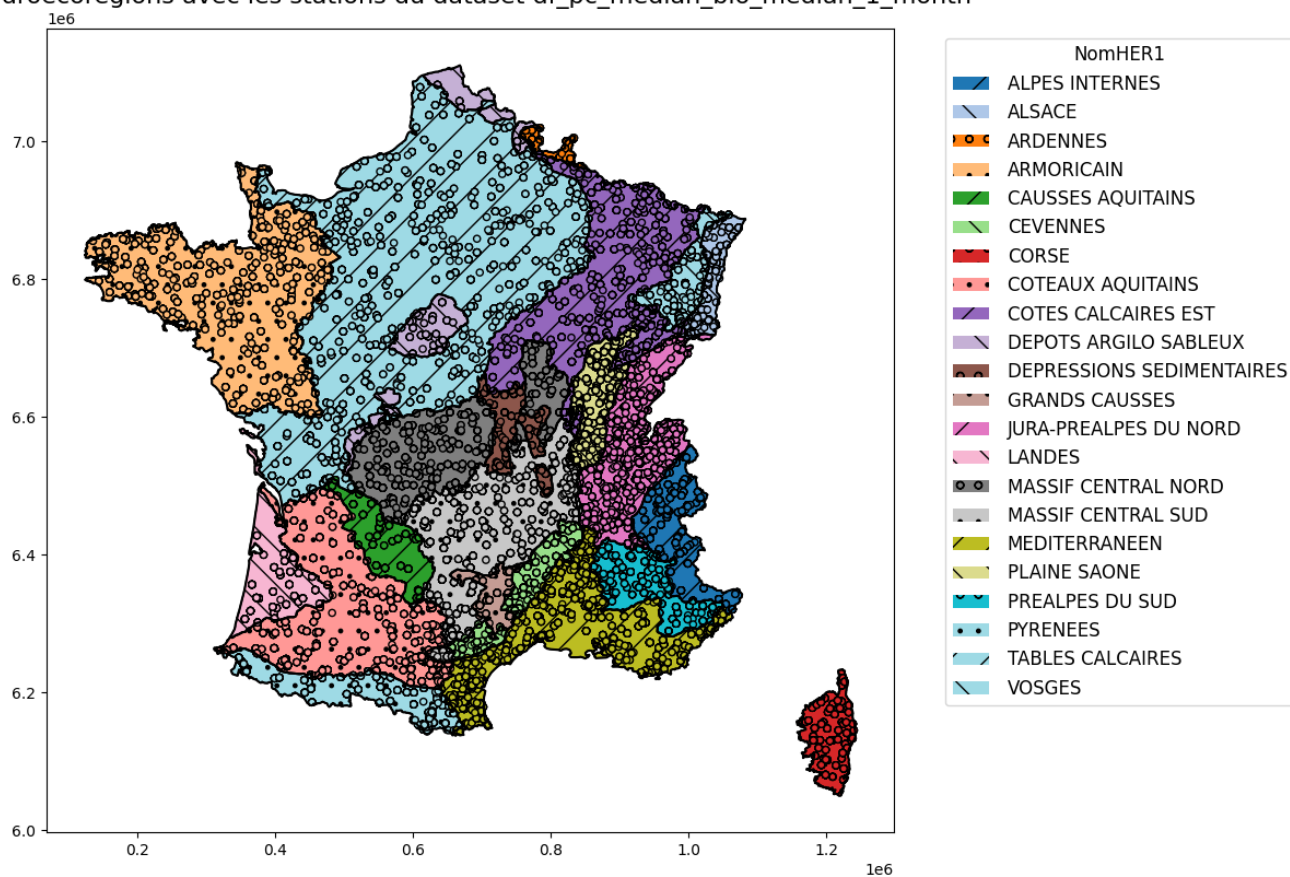
```
df_to_count = df_pc_median_bio_median_1_month_with_coords.copy()
```

```
# Représentation des stations sur la carte des hydroécorégions
crs_lambert = 'PROJCS["RGF_1993_Lambert_93",GEOGCS["GCS_RGF_1993",DATUM["D_RGF_1993"],PRIMARIES["PRIMARIES_RGF_1993"],AUTHORITY="EPSG:31466",PROJECTION="Lambert_93",UNIT="Meter",AXES="XYZ"]']
x_col = 'CoordXStationMesureEauxSurface'
y_col = 'CoordYStationMesureEauxSurface'
carto_i2m2 = gpd.GeoDataFrame(df_to_count,crs=crs_lambert ,
                              geometry = gpd.GeoSeries(df_to_count.agg(lambda x:geom.
                              # Récupération des hydroécorégions pour les stations
                              HER_stations=carto_i2m2.sjoin(df_hydroregions.to_crs(crs_lambert),predicate='within'))
                              fig, ax = plt.subplots(1, 1, figsize=(10, 30))
                              colors = plt.colormaps['tab20']
                              hatches = ['/', '\\\\', 'o', '.']
                              legend_elements = []
                              color_mapping = {}
                              for i, (name, region) in enumerate(df_hydroregions.groupby('NomHER1')):
                                  region = region.to_crs(crs_lambert)
                                  patch = region.plot(ax=ax, color=colors(i), hatch=hatches[i % len(hatches)])
                                  legend_elements.append(Patch(facecolor=colors(i), hatch=hatches[i % len(hatches)]))
                                  color_mapping[name] = colors(i)
                              station_colors = HER_stations['NomHER1'].map(color_mapping)
                              HER_stations.plot(ax=ax, color=station_colors, markersize=20, edgecolor='black')
                              HER_lambert = df_hydroregions.to_crs(crs_lambert)
```

```
HER_lambert.boundary.plot(ax=ax, color='black')
ax.legend(handles=legend_elements, title='NomHER1', bbox_to_anchor=(1.05, 1), loc='right')
ax.set_title('Hydroecoregions avec les stations du dataset df_pc_median_bio_mec')
plt.show()
```



Hydroecoregions avec les stations du dataset df_pc_median_bio_median_1_month



```
print(f"Nombre de doublons dans df_stations : {df_stations.duplicated(subset='s
print(f"Nombre de doublons dans df_pc_median_bio_median_1_month_with_coords : {
```

```
↗ Nombre de doublons dans df_stations : 7
  Nombre de doublons dans df_pc_median_bio_median_1_month_with_coords : 18909
```


```
df_stations.drop_duplicates(subset='station', inplace=True)
df_to_count.drop_duplicates(subset='station', inplace=True)
```

```
# Récupération des hydroécorégions pour les stations dans df_stations
carto_i2m2_1 = gpd.GeoDataFrame(df_stations, crs=crs_lambert, geometry = gpd.Geo
HER_stations_1=carto_i2m2_1.sjoin(df_hydroregions.to_crs(crs_lambert), predicate
# Récupération des hydroécorégions pour les stations dans df_pc_median_bio_medi
carto_i2m2_2 = gpd.GeoDataFrame(df_to_count, crs=crs_lambert, geometry = gpd.Geo
HER_stations_2=carto_i2m2_2.sjoin(df_hydroregions.to_crs(crs_lambert), predicate
```

```
print(f"Nombre de doublons après jointure spatiale dans HER_stations_1 : {HER_s
print(f"Nombre de doublons après jointure spatiale dans HER_stations : {HER_sta
```

```
↗ Nombre de doublons après jointure spatiale dans HER_stations_1 : 0
  Nombre de doublons après jointure spatiale dans HER_stations : 0
```

```
# Nombre de stations par région hydroécologique dans df_stations et dans df_pc_
# df_stations
stations_count_by_region_1 = HER_stations_1.groupby('NomHER1').size().reset_in
# df_pc_median_bio_median_1_month
stations_count_by_region_2 = HER_stations_2.groupby('NomHER1').size().reset_in
# comparaison du nombre de stations entre les deux dataframes
comparaison = pd.merge(stations_count_by_region_1, stations_count_by_region_2, c
print(comparaison)
```



	NomHER1	count_df_stations	count_df_pc_median
0	ALPES INTERNES	156	84
1	ALSACE	352	97
2	ARDENNES	63	15
3	ARMORICAIN	445	249
4	CAUSSES AQUITAINS	47	31
5	CEVENNES	106	81
6	CORSE	63	47
7	COTEAUX AQUITAINS	265	188
8	COTES CALCAIRES EST	1008	269
9	DEPOTS ARGILLO SABLEUX	47	35
10	DEPRESSIONS SEDIMENTAIRES	63	43
11	GRANDS CAUSSES	25	18
12	JURA-PREALPES DU NORD	628	390
13	LANDES	38	30
14	MASSIF CENTRAL NORD	221	137
15	MASSIF CENTRAL SUD	264	215
16	MEDITERRANEEN	571	376
17	PLAINE SAONE	222	170
18	PREALPES DU SUD	167	85
19	PYRENEES	116	84
20	TABLES CALCAIRES	1360	431
21	VOSGES	313	64

```
# Affichage du nombre de stations par hydroécorégion dans les 2 dataframes
comparaison_sorted = comparaison.sort_values('count_df_pc_median', ascending=True)
fig = go.Figure(data=[
    go.Bar(x=comparaison_sorted['NomHER1'], y=comparaison_sorted['count_df_stations']),
    go.Bar(x=comparaison_sorted['NomHER1'], y=comparaison_sorted['count_df_pc_median'])
])
fig.update_layout(title="Comparaison du nombre de stations par hydroécorégion",
fig.show()
```



Certaines hydorrégions sont très sous représentées, et les plus représentées ont au maximum 430 stations ce qui reste peu.

```
# Calcul de la perte
comparison['percentage_loss'] = ((comparison['count_df_stations'] - comparison['count_df_pc_median']) / comparison['count_df_stations']) * 100
comparison['percentage_loss'] = comparison['percentage_loss'].fillna(0)
comparison.sort_values('percentage_loss', inplace=True)
```

```
# Affichage du pourcentage de perte du nombre de stations par hydroécocorégion
fig = go.Figure(data=[go.Bar(x=comparison['NomHER1'], y=comparison['percentage_loss'])])
fig.update_layout(title="Pourcentage de perte des stations par hydroécocorégion",
fig.show()
```



```
# Calcul du nombre de stations perdues au total
total_stations_initial = comparison['count_df_stations'].sum()
total_stations_final = comparison['count_df_pc_median'].sum()
total_stations_perdue = total_stations_initial - total_stations_final
print(f"Nombre total de stations perdues : {total_stations_perdue}")
print(f"Pourcentage de stations perdues : {round((total_stations_perdue / total_stations_initial) * 100, 1)}%")
```

```
➡ Nombre total de stations perdues : 3401
   Pourcentage de stations perdues : 52.0%
```

Suite à la fusion des différents datasets, nous constatons une perte significative de stations par rapport au dataset initial. Au total, plus de 3000 stations, soit environ 50 % des stations, ont été perdues.

La distribution des pertes par hydroécocorégion montre que certaines régions, comme les Vosges et les Ardennes, ont perdu une proportion importante de leurs stations, tandis que d'autres, comme le Massif Central Sud et la Plaine de la Saône, en ont conservé une plus grande partie, avec cependant environ 20% de perte.

Ces disparités régionales pourraient influencer les résultats de nos analyses.

✓ Analyse des corrélations entre les caractéristiques

```
df_correl = df_pc_median_bio_median_1_month_with_coords.copy()
```

```
# Ajout des hydrorégions pour voir si des caractéristiques y sont corrélées
carto_i2m2_correl = gpd.GeoDataFrame(df_correl, crs=crs_lambert, geometry = gpd.
HER_stations_correl=carto_i2m2_correl.sjoin(df_hydroregions.to_crs(crs_lambert))
df_correl = HER_stations_correl.drop(columns=['geometry', 'index_right', 'NomHE
```

```
df_correl['saison'] = df_correl['saison'].factorize()[0]
```

```
# Mapping des colonnes pour pouvoir afficher la matrice de corrélation correcte
column_mapping = {
    'Ammonium - Eau': 'NH4',
    'Azote Kjeldahl - Eau': 'NKjeldahl',
    'Carbone Organique - Eau': 'CO',
    'Conductivité à 25°C - Eau': 'Cond25',
    'Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau': 'DB05',
    'Matières en suspension - Eau': 'MES',
    'Nitrates - Eau': 'NO3',
    'Nitrites - Eau': 'NO2',
    'Orthophosphates (P04) - Eau': 'P04',
    'Oxygène dissous - Eau': 'O2',
    'Phosphore total - Eau': 'Ptot',
    'Potentiel en Hydrogène (pH) - Eau': 'pH',
    'Taux de saturation en oxygène - Eau': 'O2_sat',
    'Température de l\'Eau - Eau': 'TempEau',
    'Turbidité Formazine Néphélométrique - Eau': 'Turbidité',
    'I2M2': 'I2M2',
    'CoordXStationMesureEauxSurface': 'CoordX',
    'CoordYStationMesureEauxSurface': 'CoordY'
}
df_correl = df_correl.rename(columns=column_mapping, index=column_mapping)
```

```
# Création de la matrice de corrélation
correlation_matrix = df_correl.corr()
np.fill_diagonal(correlation_matrix.values, np.nan)
fig = px.imshow(correlation_matrix, text_auto=".1f", color_continuous_scale='
fig.update_layout(xaxis=dict(tickangle=45), autosize=True, title_x=0.5, width
fig.show()
```



Cette matrice de corrélation révèle des relations entre plusieurs caractéristiques. Notamment :

I2M2 : Corrélé négativement à Cond25, NO2, PO4 et Ptot, indiquant une dégradation biologique lorsque ces paramètres augmentent. Positivement corrélié à l'oxygène dissous et son taux de saturation, soulignant leur importance pour la qualité biologique.

✓ Clustering des stations

✓ Démarche

Dans cette partie, nous cherchons à répondre à notre problématique initiale :

Est-il possible de retrouver les hydroécorégions à partir des données physicochimiques et hydrobiologiques ?

Pour cela, nous avons réalisé un clustering pour regrouper les stations en fonction de leurs caractéristiques. Cette approche nous permet d'identifier des regroupements naturels basés sur les données, susceptibles de refléter les hydroécorégions, tout en capturant des relations complexes et non linéaires que les corrélations simples ne révèlent pas.

Nous avons commencé par un nettoyage des données, incluant la gestion des valeurs manquantes et des variables non numériques. Les données ont été normalisées, et nous avons supprimé les colonnes inutiles ou susceptibles d'influencer artificiellement le clustering, comme les identifiants de stations.

Ensuite, nous avons déterminé le nombre optimal de clusters en utilisant deux méthodes : la méthode du coude (basée sur la diminution de l'inertie) et le coefficient Davies-Bouldin (évaluant la séparation et la compacité des clusters).

Une fois le nombre de clusters défini, nous avons appliqué l'algorithme K-means pour attribuer un cluster à chaque station.

Enfin, nous avons analysé les résultats en comparant les clusters obtenus avec les hydroécorégions afin d'évaluer leur cohérence.

✓ Préparation des données

▼ Gestion des données manquantes

```
df_pc_median_bio_median_1_month_with_coords.isnull().sum()
```

```

↔ station      0
   année      0
   saison      0
   Ammonium - Eau      2109
   Azote Kjeldahl - Eau      2512
   Carbone Organique - Eau      1748
   Conductivité à 25°C - Eau      351
   Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau      2015
   Diuron - Eau      9367
   Matières en suspension - Eau      371
   Nitrates - Eau      2022
   Nitrites - Eau      2019
   Orthophosphates (P04) - Eau      1957
   Oxygène dissous - Eau      407
   Phosphore total - Eau      1633
   Potentiel en Hydrogène (pH) - Eau      329
   Taux de saturation en oxygène - Eau      1351
   Température de l'Eau - Eau      284
   Turbidité Formazine Néphélométrique - Eau      7048
   I2M2      0
   CoordXStationMesureEauxSurface      0
   CoordYStationMesureEauxSurface      0
   dtype: int64

```

```
missing_percentage = df_pc_median_bio_median_1_month_with_coords.isnull().mean()
print(missing_percentage)
```

```
station 0.000000
année 0.000000
saison 0.000000
Ammonium - Eau 9.557257
Azote Kjeldahl - Eau 11.383514
Carbone Organique - Eau 7.921330
Conductivité à 25°C - Eau 1.590610
Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau 9.131282
Diuron - Eau 42.447999
Matières en suspension - Eau 1.681243
Nitrates - Eau 9.163004
Nitrites - Eau 9.149409
Orthophosphates (P04) - Eau 8.868446
Oxygène dissous - Eau 1.844383
Phosphore total - Eau 7.400190
Potentiel en Hydrogène (pH) - Eau 1.490914
Taux de saturation en oxygène - Eau 6.122264
Température de l'Eau - Eau 1.286990
Turbidité Formazine Néphélométrique - Eau 31.939095
I2M2 0.000000
CoordXStationMesureEauxSurface 0.000000
CoordYStationMesureEauxSurface 0.000000
dtype: float64
```

```
# plus de 40 % de données manquantes pour Diuron - Eau
# on peut supprimer cette colonne
df_pc_median_bio_median_1_month_with_coords.drop(columns=['Diuron - Eau'], inplace=True)
```

```
# Calculer le nombre de valeurs manquantes par ligne
missing_values_per_row = df_pc_median_bio_median_1_month_with_coords.isnull().sum()
missing_summary = missing_values_per_row.value_counts().reset_index()
missing_summary.columns = ['Nombre de valeurs manquantes', 'Nombre de lignes']
print(missing_summary.sort_values('Nombre de valeurs manquantes'))
```

	Nombre de valeurs manquantes	Nombre de lignes
0	0	11776
1	1	7137
3	2	823
5	3	197
9	4	97
4	5	261
8	6	127
7	7	144
2	8	1087
12	9	42
6	10	193
11	11	81
15	12	1
13	13	8
10	14	87
14	15	6

Nous avons imputé les données manquantes par leur médiane. Nous avons également testé d'imputer ces données par leur moyenne mais les résultats de clusterings étaient légèrement moins bons.

```
# Imputation des valeurs manquantes avec la médiane
imputer = SimpleImputer(strategy='median') # ici on pourrait aussi mettre mean
colonnes = df_pc_median_bio_median_1_month_with_coords.columns
colonnes = colonnes.drop('station')
colonnes = colonnes.drop('année')
colonnes = colonnes.drop('saison')
colonnes = colonnes.drop('I2M2')
df_pc_median_bio_median_1_month_with_coords[colonnes] = imputer.fit_transform(c
df_pc_median_bio_median_1_month_with_coords.isnull().sum()
```

station	0
année	0
saison	0
Ammonium – Eau	0
Azote Kjeldahl – Eau	0
Carbone Organique – Eau	0
Conductivité à 25°C – Eau	0
Demande Biochimique en oxygène en 5 jours (D.B.O.5) – Eau	0
Matières en suspension – Eau	0
Nitrates – Eau	0
Nitrites – Eau	0
Orthophosphates (P04) – Eau	0
Oxygène dissous – Eau	0
Phosphore total – Eau	0
Potentiel en Hydrogène (pH) – Eau	0
Taux de saturation en oxygène – Eau	0
Température de l'Eau – Eau	0
Turbidité Formazine Néphélométrique – Eau	0
I2M2	0
CoordXStationMesureEauxSurface	0
CoordYStationMesureEauxSurface	0
dtype: int64	

✓ Gestion des données non numériques

```
# Mapper chaque saison à un trimestre
saison_to_quarter = {
    "Hiver": 1,
    "Printemps": 2,
    "Été": 3,
    "Automne": 4
}
df_pc_median_bio_median_1_month_with_coords['trimestre'] = df_pc_median_bio_mec
# drop saison
df_pc_median_bio_median_1_month_with_coords.drop(columns=['saison'], inplace=True)
df_pc_median_bio_median_1_month_with_coords.head(5)
```



	station	année	Ammonium - Eau	Azote Kjeldahl - Eau	Carbone Organique - Eau	Conductivité à 25°C - Eau	Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau
0	5001800	2007	0.04	1.0	3.5	765.0	0.5
1	5001800	2008	0.04	1.0	2.7	765.0	0.6
2	5001800	2009	0.02	1.0	2.2	736.0	1.0
3	5005350	2007	0.04	1.0	1.6	600.0	1.9
4	5005350	2008	0.03	1.0	1.6	610.0	1.0

5 rows × 21 columns

```
# Convertir l'identifiant de station en variable catégorielle (cela crée un enc
df_pc_median_bio_median_1_month_with_coords['station'] = df_pc_median_bio_media
df_pc_median_bio_median_1_month_with_coords['trimestre'] = df_pc_median_bio_mec
df_pc_median_bio_median_1_month_with_coords['année'] = df_pc_median_bio_median_
df_pc_median_bio_median_1_month_with_coords.dtypes
```

```
↔ station object
   année int64
Ammonium – Eau float64
Azote Kjeldahl – Eau float64
Carbone Organique – Eau float64
Conductivité à 25°C – Eau float64
Demande Biochimique en oxygène en 5 jours (D.B.O.5) – Eau float64
Matières en suspension – Eau float64
Nitrates – Eau float64
Nitrites – Eau float64
Orthophosphates (P04) – Eau float64
Oxygène dissous – Eau float64
Phosphore total – Eau float64
Potentiel en Hydrogène (pH) – Eau float64
Taux de saturation en oxygène – Eau float64
Température de l'Eau – Eau float64
Turbidité Formazine Néphélométrique – Eau float64
I2M2 float64
CoordXStationMesureEauxSurface float64
CoordYStationMesureEauxSurface float64
trimestre int64
dtype: object
```

✓ Suppression des colonnes

Nous supprimons les colonnes des coordonnées, afin de ne pas influencer le clustering.

```
df_pc_median_bio_median_1_month_c = df_pc_median_bio_median_1_month_with_coords
```

```
df_pc_median_bio_median_1_month_c = df_pc_median_bio_median_1_month_with_coords
df_clustering = df_pc_median_bio_median_1_month_c.copy()
```


✓ Normalisation et encodage des données

Pour préserver l'information relative aux stations, essentielle pour suivre l'évolution de leurs caractéristiques physicochimiques et hydrobiologiques sur plusieurs trimestres, nous avons choisi de conserver cette colonne dans le processus de clustering.

Pour minimiser tout biais lié aux identifiants de stations, nous avons encodé cette colonne avec des valeurs numériques arbitraires. Ensuite, comme pour les autres caractéristiques, nous avons normalisé ses valeurs afin d'assurer une contribution équilibrée au clustering.

Nous avons également testé la suppression de la colonne station, mais cela a légèrement dégradé les résultats. Par ailleurs, perdre le lien temporel qu'offre cette information entre les enregistrements nous semblait non pertinent. Nous avons donc décidé de conserver la colonne station, en sachant qu'elle introduit un léger biais.

```
df_clustering.head(5)
```




	station	année	Ammonium - Eau	Azote Kjeldahl - Eau	Carbone Organique - Eau	Conductivité à 25°C - Eau	Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau
0	5001800	2007	0.04	1.0	3.5	765.0	0.5
1	5001800	2008	0.04	1.0	2.7	765.0	0.6
2	5001800	2009	0.02	1.0	2.2	736.0	1.0
3	5005350	2007	0.04	1.0	1.6	600.0	1.9
4	5005350	2008	0.03	1.0	1.6	610.0	1.0

Nous utilisons LabelEncoder plutôt que One-Hot Encoding en raison du volume important de données. One-Hot Encoding aurait entraîné une explosion du nombre de colonnes, ce qui aurait considérablement alourdi le traitement et augmenté la complexité du modèle. LabelEncoder permet de représenter les catégories sous forme d'entiers, ce qui est plus efficace en termes de mémoire et de calcul.

Étant donné que nous travaillons avec des données dont la distribution est bornée et sans connaître précisément leur forme, nous utilisons le MinMaxScaler. Ce dernier est plus adapté pour redimensionner les données dans une plage spécifique lorsque celles-ci sont déjà bornées.

```
# Encodage des stations
le = LabelEncoder()
df_clustering['station_encoded'] = le.fit_transform(df_clustering['station'])
df_clustering.drop(columns=['station'], inplace=True)
# Normalisation des données pour qu'elles aient toutes la même importance
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(df_clustering)
normalized_data = pd.DataFrame(normalized_data, columns=df_clustering.columns)
```

```
normalized_data.head(5)
```



	année	Ammonium - Eau	Azote Kjeldahl - Eau	Carbone Organique - Eau	Conductivité à 25°C - Eau	Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau	Matière suspensée - Eau
0	0.000000	0.006169	0.190283	0.220588	0.423398	0.000000	0.000000
1	0.066667	0.006169	0.190283	0.167112	0.423398	0.007407	0.000000
2	0.133333	0.002742	0.190283	0.133690	0.407242	0.037037	0.010000
3	0.000000	0.006169	0.190283	0.093583	0.331476	0.103704	0.040000
4	0.066667	0.004455	0.190283	0.093583	0.337047	0.037037	0.080000

✓ Recherche du nombre de clusters optimal

```
nb_hydroecoregions = df_hydroregions['CdHER1'].nunique()
print("Nombre d'hydroécorégions : ", nb_hydroecoregions)
```

```
↗ Nombre d'hydroécorégions : 22
```

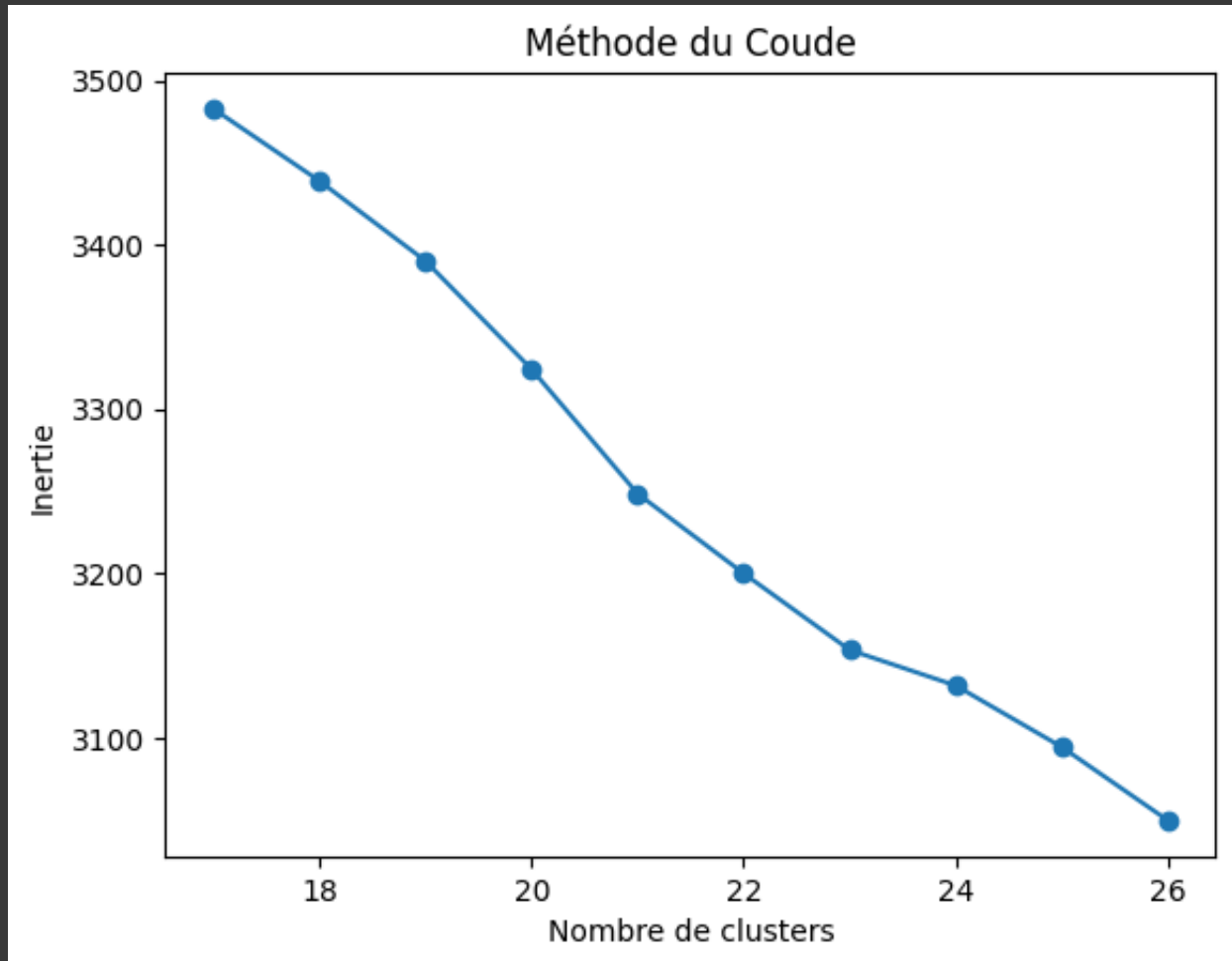
```
# Plage des valeurs de k à tester
k_values = range(nb_hydroecoregions-5, nb_hydroecoregions+5)
```

```
# Calcule de l'inertie et du score de Davies-Bouldin pour chaque valeur de k
davies_bouldin_scores = []
inertia = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(normalized_data)
    inertia.append(kmeans.inertia_)
    cluster_labels = kmeans.fit_predict(normalized_data)
    davies_bouldin_scores.append(davies_bouldin_score(normalized_data, cluster_
```

✓ Méthode du coude

La méthode du coude consiste à observer la diminution de l'inertie (somme des erreurs quadratiques intra-clusters) en fonction du nombre de clusters. Le point où la diminution devient moins significative peut indiquer le nombre optimal de clusters.

```
# Tracer l'inertie pour observer le coude  
plt.plot(k_values, inertia, marker='o')  
plt.title('Méthode du Coude')  
plt.xlabel('Nombre de clusters')  
plt.ylabel('Inertie')  
plt.show()
```

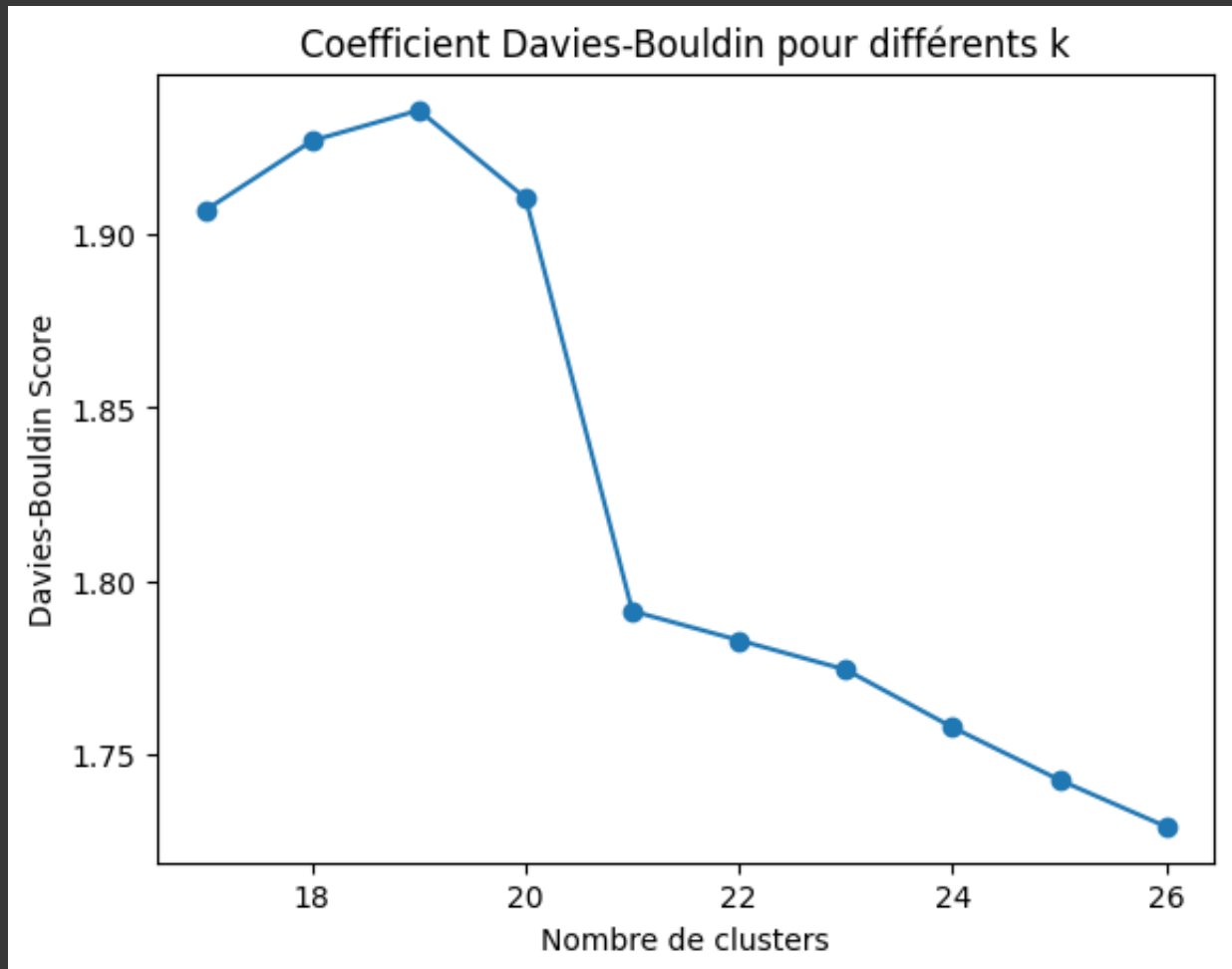


Nous n'arrivons pas à voir de coude, cela rend impossible à déterminer le nombre optimal de clusters. Nous allons donc utiliser le coefficient Davies-Bouldin pour faire notre choix.

✓ Coefficient Davies-Bouldin

Le coefficient Davies-Bouldin évalue la qualité des clusters en mesurant leur compacité et leur séparation. Une valeur plus faible indique des clusters mieux définis.

```
plt.plot(k_values, davies_bouldin_scores, marker='o')  
plt.title('Coefficient Davies-Bouldin pour différents k')  
plt.xlabel('Nombre de clusters')  
plt.ylabel('Davies-Bouldin Score')  
plt.show()
```



Nous pouvons voir clairement que le score de Davies-Bouldin est minimal pour 26 clusters (pour k testé entre 2 et 40).

On voit tout de même une nette amélioration autour de 21. Nous allons donc utiliser $k = 22$ pour notre clustering, car le but est de retrouver les hydroécorégions.

- ✓ Réalisation du clustering avec K-Means
- ✓ Clustering

```
nb_clusters = 22 # c'est le nombre d'hydroecoregions
```

```
kmeans = KMeans(n_clusters=nb_clusters, random_state=42)
kmeans.fit(normalized_data)
```



KMeans

```
KMeans(n_clusters=22, random_state=42)
```

✓ Analyse des caractéristiques des clusters

```
# Analyse des centres des clusters
cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=df_clustering.c
print("Centres des clusters :")
print(cluster_centers)
# Variabilité des caractéristiques par cluster
influence = cluster_centers.std(axis=0)
print("\nInfluence des caractéristiques :")
print(influence.sort_values(ascending=False))
```



Centres des clusters :

	année	Ammonium – Eau	Azote Kjeldahl – Eau	Carbone Organique – Eau
0	0.707556	0.003651	0.102529	0.090147
1	0.249482	0.017077	0.160705	0.176372
2	0.262536	0.014060	0.175244	0.183344
3	0.762536	0.006252	0.109104	0.174547
4	0.209027	0.011974	0.199577	0.091468
5	0.733449	0.003293	0.106691	0.040919
6	0.748257	0.008057	0.104491	0.146719
7	0.638630	0.006876	0.107028	0.188756
8	0.665855	0.014854	0.129444	0.157090
9	0.247924	0.009221	0.191398	0.090527
10	0.717722	0.027798	0.133328	0.198282
11	0.807137	0.006971	0.114634	0.204066
12	0.526005	0.005426	0.115513	0.098832
13	0.652447	0.018494	0.206541	0.481690
14	0.762981	0.009198	0.118056	0.178326
15	0.192699	0.009375	0.172560	0.134697
16	0.894674	0.006302	0.107940	0.158099
17	0.556588	0.007803	0.109166	0.120521
18	0.700532	0.021140	0.157779	0.245159
19	0.826806	0.004704	0.101173	0.076801
20	0.276536	0.009791	0.191287	0.082039
21	0.355509	0.018202	0.176868	0.122463
22	0.814249	0.015142	0.120401	0.155828
23	0.260950	0.023928	0.205889	0.205656
24	0.868313	0.012506	0.134622	0.246087

```
25 0.480108 0.152589 0.290769 0.257444
```

```
Conductivité à 25°C – Eau \
```

```
0 0.083917
1 0.273867
2 0.351371
3 0.120180
4 0.213629
5 0.205073
6 0.295166
7 0.064199
8 0.266297
9 0.277629
10 0.403002
11 0.241226
12 0.142874
13 0.154753
14 0.315303
15 0.108710
16 0.110466
17 0.196895
18 0.346247
19 0.269281
20 0.207648
21 0.345499
22 0.366028
23 0.269317
24 0.224428
25 0.465191
```

```
Demande Biochimique en oxygène en 5 jours (D.B.0.5) – Eau \
```

```
0 0.065462
```

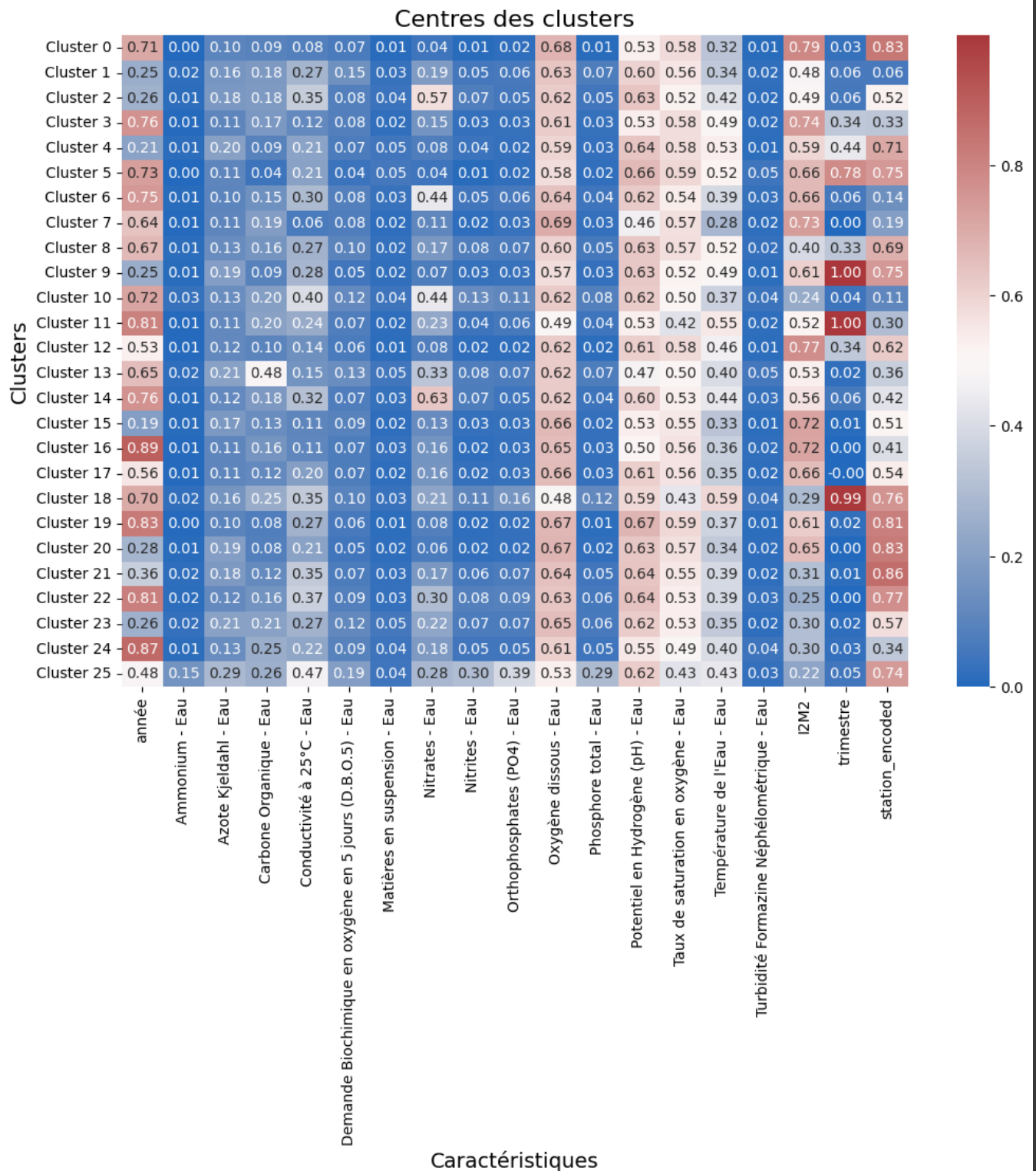
```
# Affichage des caractéristiques avec les plus grandes variabilités entre clust
# = Caractéristiques les plus influentes
fig = px.bar(influence.sort_values(ascending=False),
             title="Influence des caractéristiques par cluster",
             labels={'index': 'Caractéristiques', 'value': "Écart-type des cent
fig.update_layout(xaxis_tickangle=45, width=800, height=600, showlegend=False)
fig.show()
```



En dehors des données spatio-temporelles, les paramètres ayant le plus d'impact sont l'indice I2M2, les Nitrates, la Température de l'eau et la Conductivité à 25°C.

```
# Matrice des centres des clusters
plt.figure(figsize=(12, 8))
sns.heatmap(cluster_centers, annot=True, fmt=".2f", cmap="vlag", xticklabels=df
plt.title("Centres des clusters", fontsize=16)
```

```
plt.xlabel("Caractéristiques", fontsize=14)
plt.ylabel("Clusters", fontsize=14)
plt.show()
```



La matrice des centres des clusters révèle des valeurs disparates pour l'indice I2M2, les Nitrates, le Carbone organique et la Conductivité de l'eau. En revanche, pour certaines colonnes, nous obtenons des valeurs identiques, avec une ou deux valeurs dominantes dans l'ensemble des clusters.

✓ Attribution des clusters

```
df_pc_median_bio_median_1_month_with_clusters = df_pc_median_bio_median_1_month_with_clusters  
df_pc_median_bio_median_1_month_with_clusters['cluster'] = kmeans.labels_
```



```
df_pc_median_bio_median_1_month_with_clusters.head(5)
```



	station	année	Ammonium - Eau	Azote Kjeldahl - Eau	Carbone Organique - Eau	Conductivité à 25°C - Eau	Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau
0	5001800	2007	0.04	1.0	3.5	765.0	0.5
1	5001800	2008	0.04	1.0	2.7	765.0	0.6
2	5001800	2009	0.02	1.0	2.2	736.0	1.0
3	5005350	2007	0.04	1.0	1.6	600.0	1.9
4	5005350	2008	0.03	1.0	1.6	610.0	1.0

✓ Calcul du score de silhouette

```
# Calculer le score de la silhouette
score = silhouette_score(normalized_data, df_pc_median_bio_median_1_month_with_
print(f"Silhouette score: {score}")
```



Silhouette score: 0.1251427074415718

Le score de silhouette obtenu est plutôt faible, ce qui indique que les stations ne sont pas clairement regroupées dans leurs clusters respectifs, et que certaines stations sont plus proches des centres d'autres clusters.

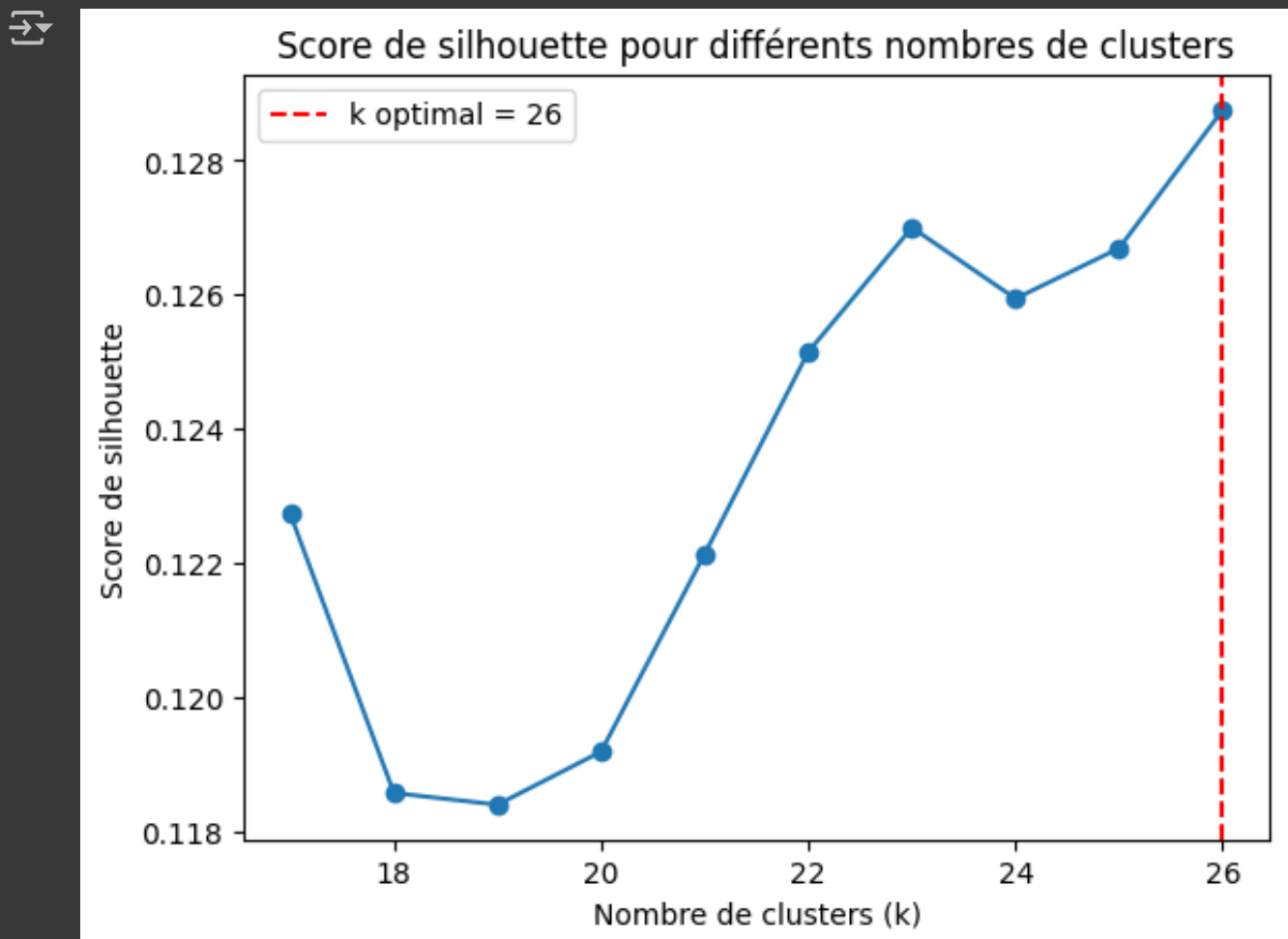
```
# Recherche du meilleur nombre de clusters selon le score de silhouette
silhouette_scores = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    cluster_labels = kmeans.fit_predict(normalized_data)

    score = silhouette_score(normalized_data, cluster_labels)
    silhouette_scores.append(score)

best_k = k_values[silhouette_scores.index(max(silhouette_scores))]
print(f"Le meilleur nombre de clusters est : {best_k}")
```

➞ Le meilleur nombre de clusters est : 26

```
plt.plot(k_values, silhouette_scores, marker='o')
plt.title("Score de silhouette pour différents nombres de clusters")
plt.xlabel("Nombre de clusters (k)")
plt.ylabel("Score de silhouette")
plt.axvline(x=best_k, color='r', linestyle='--', label=f"k optimal = {best_k}")
plt.legend()
plt.show()
```



Le score de silhouette est le meilleur quand $k = 26$. Nous conservons $k = 22$, afin de rester alignée avec notre problématique géographique.

- ✓ Etude des résultats obtenus
- ✓ Chaque station a-t-elle été assignée à un seul cluster ?

```
# Créer un DataFrame pour calculer le pourcentage d'appartenance des stations à
cluster_distribution = df_pc_median_bio_median_1_month_with_clusters.groupby('s
print(cluster_distribution)
```

```
⇒ cluster      0      1      2      3      4      5      6      7  \
station
1000477  16.666667  0.0    0.000000  0.0    0.0    0.000000  0.0    0.0
1000602   0.000000  0.0    0.000000  0.0    0.0    0.000000  0.0    0.0
1000605   0.000000  0.0    0.000000  0.0    0.0    0.000000  0.0    0.0
1001122   0.000000  0.0    0.000000  0.0    0.0    0.000000  0.0    0.0
1001131   0.000000  0.0    0.000000  0.0    0.0    0.000000  0.0    0.0
...      ...      ...      ...      ...      ...      ...      ...      ...
6999125   0.000000  0.0    7.142857  0.0    0.0   28.571429  0.0    0.0
6999137   0.000000  0.0   20.000000  60.0    0.0   20.000000  0.0    0.0
6999153   0.000000  0.0   60.000000  0.0    0.0   40.000000  0.0    0.0
6999176   0.000000  0.0  100.000000  0.0    0.0    0.000000  0.0    0.0
6999178   0.000000  0.0  100.000000  0.0    0.0    0.000000  0.0    0.0

cluster      8      9      ...     12     13     14     15     16      1
station      ...
1000477  16.666667  0.0    ...    0.000000  0.000000  0.0    0.0    0.0  66.66666
1000602  100.000000  0.0    ...    0.000000  0.000000  0.0    0.0    0.0  0.00000
1000605  16.666667  0.0    ...   50.000000  0.000000  0.0    0.0    0.0  0.00000
1001122   0.000000  0.0    ...   16.666667  0.000000  0.0    0.0    0.0  0.00000
1001131   0.000000  0.0    ...    0.000000  0.000000  0.0    0.0    0.0  66.66666
...      ...      ...      ...      ...      ...      ...      ...      ...
6999125   0.000000  0.0    ...    0.000000  7.142857  0.0    0.0    0.0  0.00000
6999137   0.000000  0.0    ...    0.000000  0.000000  0.0    0.0    0.0  0.00000
6999153   0.000000  0.0    ...    0.000000  0.000000  0.0    0.0    0.0  0.00000
6999176   0.000000  0.0    ...    0.000000  0.000000  0.0    0.0    0.0  0.00000
6999178   0.000000  0.0    ...    0.000000  0.000000  0.0    0.0    0.0  0.00000

cluster      18      19      20      21
station
1000477   0.000000  0.000000  0.000000  0.0
1000602   0.000000  0.000000  0.000000  0.0
1000605  33.333333  0.000000  0.000000  0.0
1001122   0.000000  83.333333  0.000000  0.0
1001131   0.000000  33.333333  0.000000  0.0
...      ...      ...      ...      ...
6999125   0.000000  0.000000  21.428571  0.0
6999137   0.000000  0.000000  0.000000  0.0
6999153   0.000000  0.000000  0.000000  0.0
6999176   0.000000  0.000000  0.000000  0.0
6999178   0.000000  0.000000  0.000000  0.0
```

[3158 rows x 22 columns]

→ 22067

3158

```

➡ Stations à 100% dans un seul cluster : 778
   Pourcentage de stations 100% dans le même cluster : 24.64 %

```

Nous voyons qu'un quart des stations sont classées dans le même cluster pour tous les trimestres et toutes les années. Bien que cela ne soit pas beaucoup, cela montre que le clustering prend en compte la station et s'en sert pour relier les enregistrements et suivre les variations trimestrielles.

✓ Combien d'enregistrements classés dans la bonne hydorrégion ?

```
# Attribution du cluster dominant à chaque station
station_cluster_counts = df_pc_median_bio_median_1_month_with_clusters_with_coc.groupby('station').value_counts().reset_index()
dominant_cluster_per_station = station_cluster_counts.loc[station_cluster_counts.groupby('station')['count'].idxmax()]
df_pc_median_bio_median_1_month_with_clusters_with_coord = df_pc_median_bio_median_1_month_with_clusters_with_coc.merge(dominant_cluster_per_station, on='station')
df_pc_median_bio_median_1_month_with_clusters_with_coord.rename(columns={'cluster': 'dominant_cluster'})
```

Page 93 sur 137

```
# Ajout de l'hydroécorégion réelle pour chaque station
carto_i2m2_her = gpd.GeoDataFrame(df_analyse, crs=crs_lambert,
                                  geometry=gpd.GeoSeries(df_analyse.agg(lambda
HER_stations_her = carto_i2m2_her.sjoin(df_hydroregions.to_crs(crs_lambert), pr
df_analyse_etude = HER_stations_her.drop(columns=['geometry', 'index_right', 'c
df_analyse_etude.rename(columns={'CdHER1': 'hydroecoregion'}, inplace=True)
```

```
# Pour chaque cluster, nous déterminons quelle est l'hydrorégion dominante
dominant_class_per_cluster = df_analyse_etude.groupby('cluster_dominant_station
dominant_class_per_cluster.columns = ['cluster_dominant_station', 'dominant_hyc
df_analyse_etude = df_analyse_etude.merge(dominant_class_per_cluster, on='clust
```

```
df_analyse_etude.head(5)
```

	station	année	CoordXStationMesureEauxSurface	CoordYStationMesureEauxSu
0	5001800	2007	399856.0	653
1	5001800	2008	399856.0	653
2	5001800	2009	399856.0	653
3	5005350	2007	450151.0	657
4	5005350	2008	450151.0	657

```
# On garde qu'une ligne par station
df_analyse_etude.drop_duplicates(subset='station', inplace=True)
```

```
# Quel cluster est associé à quelle hydroécorégion ?
clusters_hydroregions = df_analyse_etude.groupby('cluster_dominant_station')['h
clusters_hydroregions.columns = ['cluster_dominant_station', 'dominant_hydroec
clusters_hydroregions['dominant_hydroecoregion_cluster'] = clusters_hydroregior
df_hydroregions['CdHER1'] = df_hydroregions['CdHER1'].astype(str)
clusters_hydroregions_with_names = clusters_hydroregions.merge(df_hydroregions|
clusters_hydroregions_with_names = clusters_hydroregions_with_names.rename(colu
print(clusters_hydroregions_with_names[['cluster_dominant_station', 'dominant_h
```

	cluster_dominant_station	dominant_hydroecoregion_cluster	\
0	0	6	
1	1	6	

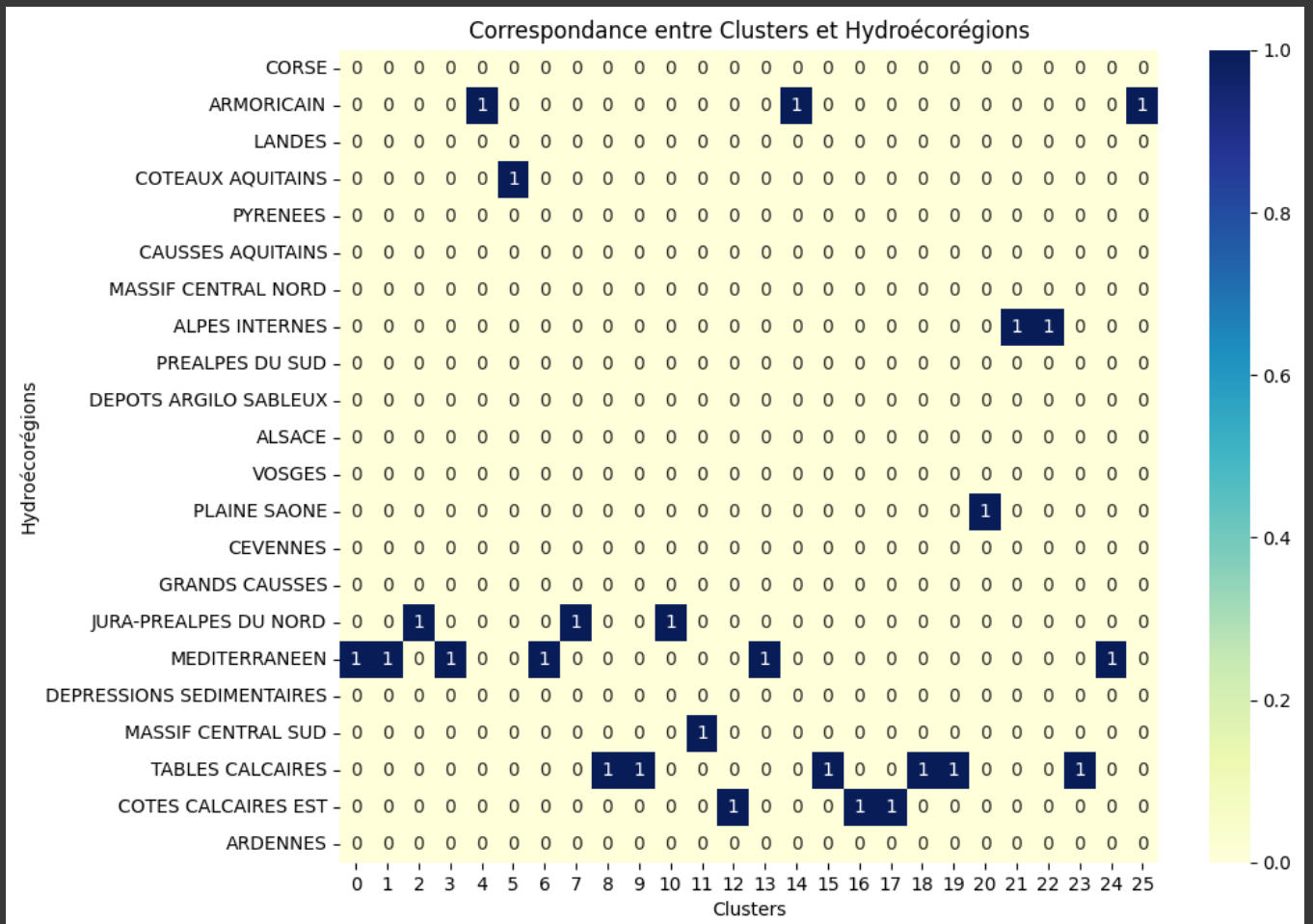
2	2	5
3	3	6
4	4	12
5	5	14
6	6	6
7	7	5
8	8	9
9	9	9
10	10	5
11	11	3
12	12	10
13	13	6
14	14	12
15	15	9
16	16	10
17	17	10
18	18	9
19	19	9
20	20	15
21	21	2
22	22	2
23	23	9
24	24	6
25	25	12

	nom_dominant_hydroecoregion_cluster
0	MEDITERRANEEN
1	MEDITERRANEEN
2	JURA-PREALPES DU NORD
3	MEDITERRANEEN
4	ARMORICAIN
5	COTEAUX AQUITAINS
6	MEDITERRANEEN
7	JURA-PREALPES DU NORD
8	TABLES CALCAIRES
9	TABLES CALCAIRES
10	JURA-PREALPES DU NORD
11	MASSIF CENTRAL SUD
12	COTES CALCAIRES EST
13	MEDITERRANEEN
14	ARMORICAIN
15	TABLES CALCAIRES
16	COTES CALCAIRES EST
17	COTES CALCAIRES EST
18	TABLES CALCAIRES
19	TABLES CALCAIRES
20	PLAINE SAONE
21	ALPES INTERNES
22	ALPES INTERNES
23	TABLES CALCAIRES
24	MEDITERRANEEN
25	ARMORICAIN

```

all_hydroecoregions = df_hydroregions['NomHER1'].unique()
cross_tab = pd.crosstab(clusters_hydroregions_with_names['nom_dominant_hydroecoregion'], all_hydroecoregions)
plt.figure(figsize=(10, 8))
sns.heatmap(cross_tab, annot=True, fmt='d', cmap='YlGnBu', cbar=True)
plt.title("Correspondance entre Clusters et Hydroécorégions")
plt.xlabel("Clusters")
plt.ylabel("Hydroécorégions")
plt.show()

```



Nous observons que certaines hydroécorégions sont associées à plusieurs clusters, tandis que d'autres ne sont pas représentées dans les résultats du clustering. Cela peut s'expliquer par des similarités entre les caractéristiques physicochimiques et hydrobiologiques de stations appartenant à des hydroécorégions différentes, ou par des variations internes importantes au sein d'une même hydroécorégion.

De plus, nous voyons que parmi les 22 clusters définis, seuls 9 des hydroécorégions sont identifiées dans les données.

```
# Ajouter une colonne qui indique si l'enregistrement a bien été classée
# On vérifie pour chaque ligne si le cluster correspond à l'hydroécorégion domi
df_analyse_etude['correct_classification'] = df_analyse_etude['hydroecoregion']
```

```
num_correctly_classified = df_analyse_etude['correct_classification'].sum()
total_stations = df_analyse_etude.shape[0]
accuracy = num_correctly_classified / total_stations
print(f"Nombre de stations bien classées: {num_correctly_classified}/{total_stations}")
print(f"Taux de classification correcte: {accuracy * 100:.2f}%")
```

```
➡ Nombre de stations bien classées: 1091/3139
Taux de classification correcte: 34.76%
```

✓ Quels trimestres présentent le plus de stations bien classées ?

```
df_analyse_etude['annee_trimestre'] = df_analyse_etude['année'].astype(str) + 'trimestre'
```

```
stations_correct = df_analyse_etude[df_analyse_etude['correct_classification']]
stations_correct_count = stations_correct.groupby('annee_trimestre')['station'].count()
stations_correct_count.columns = ['annee_trimestre', 'stations_correctement_classées']
```

```
total_stations_count = df_analyse_etude.groupby('annee_trimestre')['station'].count()
total_stations_count.columns = ['annee_trimestre', 'stations_totales']
```

```
merged_counts = pd.merge(total_stations_count, stations_correct_count, on='annee_trimestre')
merged_counts['stations_correctement_classées'] = merged_counts['stations_correctement_classées']
merged_counts['pourcentage_correct'] = (merged_counts['stations_correctement_classées'] / merged_counts['stations_totales']) * 100
```

```
fig = px.bar(merged_counts, x='annee_trimestre', y='pourcentage_correct', title=
fig.update_layout(xaxis_tickangle=45, width=900, height=600, showlegend=False,
fig.show()
```

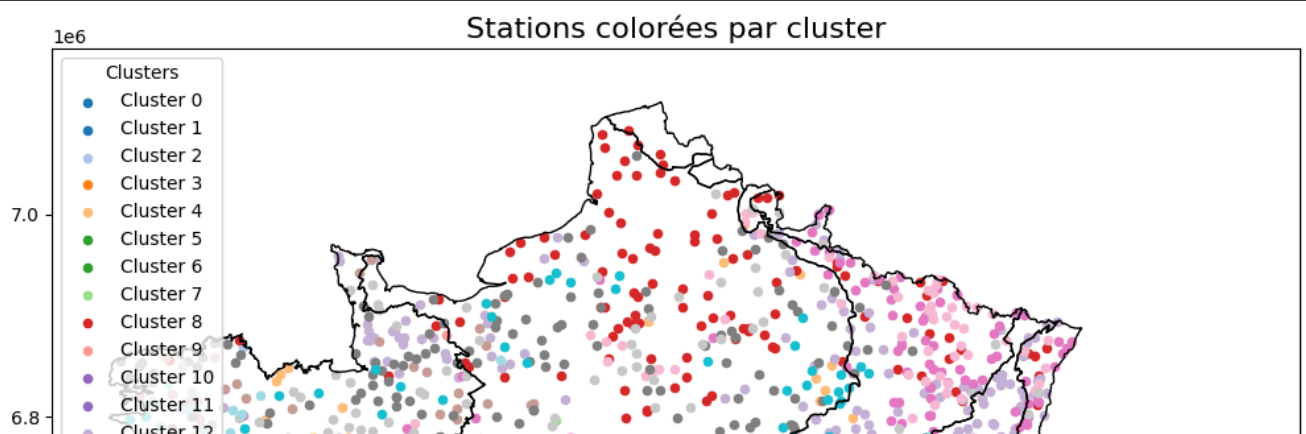


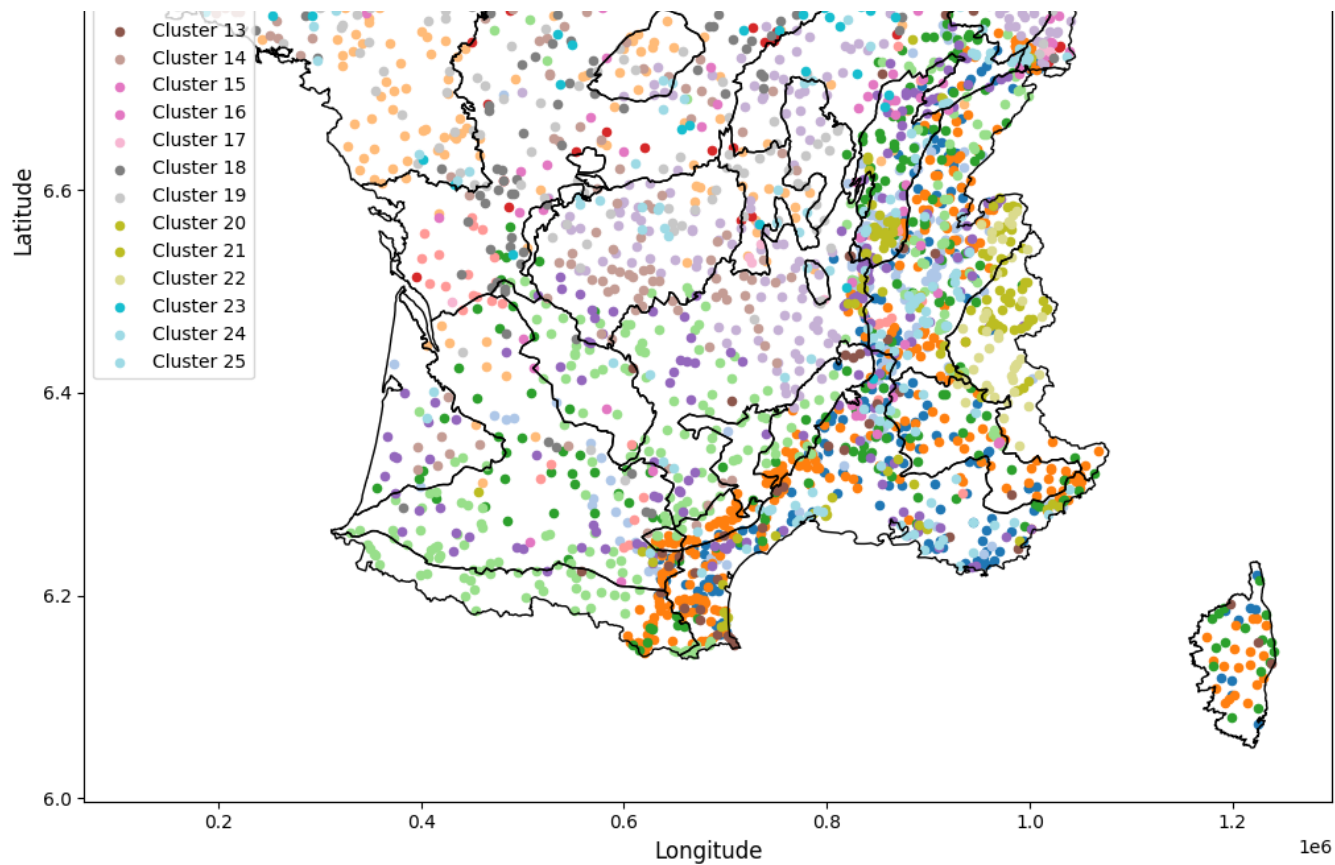
✓ Visualisation du clustering sur les stations

```
# Représentation des stations avec la couleur de leur cluster dominant
crs_lambert = 'PROJCS["RGF_1993_Lambert_93",GEOGCS["GCS_RGF_1993",DATUM["D_RGF_
x_col = 'CoordXStationMesureEauxSurface'
y_col = 'CoordYStationMesureEauxSurface'
cluster_col = 'cluster_dominant_station'
carto_i2m2 = gpd.GeoDataFrame(df_analyse_etude, crs=crs_lambert, geometry=gpd.C
HER_lambert = df_hydroregions.to_crs(crs_lambert)
unique_clusters = carto_i2m2[cluster_col].unique()
unique_clusters = sorted(unique_clusters)
cmap = cm.get_cmap('tab20', len(unique_clusters))
cluster_colors = {cluster: mcolors.to_hex(cmap(i)) for i, cluster in enumerate(
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
HER_lambert.boundary.plot(ax=ax, color='black', linewidth=1)
for cluster, color in cluster_colors.items():
    cluster_data = carto_i2m2[carto_i2m2[cluster_col] == cluster]
    cluster_data.plot(ax=ax, color=color, markersize=20, label=f"Cluster {clust
ax.legend(loc='upper left', fontsize='medium', title='Clusters')
plt.title('Stations colorées par cluster', fontsize=16)
plt.xlabel('Longitude', fontsize=12)
plt.ylabel('Latitude', fontsize=12)
plt.tight_layout()
plt.show()
```

/var/folders/p1/4yqk4cyx3058m3j04rtkr56m0000gn/T/ipykernel_53919/1842718937

The `get_cmap` function was deprecated in Matplotlib 3.7 and will be removed





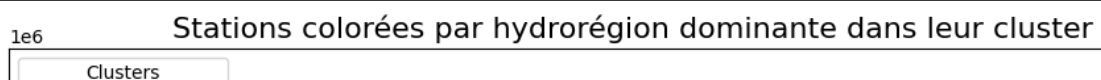
```
df_analyse_etude.head(5)
```

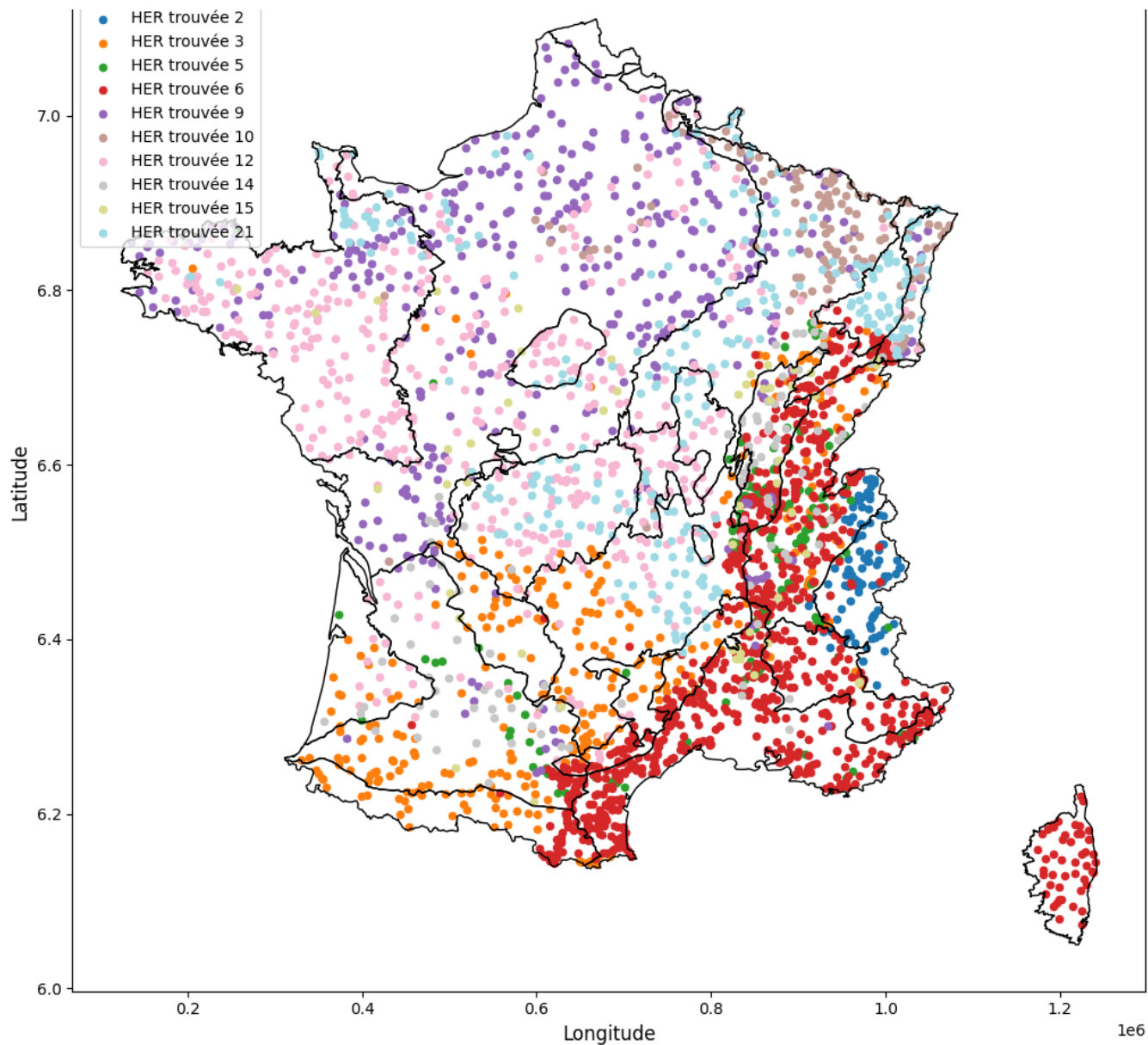
	station	année	CoordXStationMesureEauxSurface	CoordYStationMesureEauxS
0	5001800	2007	399856.0	65
3	5005350	2007	450151.0	65
17	5005400	2007	446087.0	65
31	5005950	2007	451708.0	65
45	5006100	2007	459757.0	65

```
# Représentation des stations avec la couleur de l'hydorrégion dominante dans l
crs_lambert = 'PROJCS["RGF_1993_Lambert_93",GEOGCS["GCS_RGF_1993",DATUM["D_RGF_
x_col = 'CoordXStationMesureEauxSurface'
y_col = 'CoordYStationMesureEauxSurface'
cluster_col = 'dominant_hydroecoregion_cluster'
carto_i2m2 = gpd.GeoDataFrame(df_analyse_etude, crs=crs_lambert, geometry=gpd.C
HER_lambert = df_hydroregions.to_crs(crs_lambert)
unique_clusters = carto_i2m2[cluster_col].unique()
unique_clusters = sorted(unique_clusters)
cmap = cm.get_cmap('tab20', len(unique_clusters))
cluster_colors = {cluster: mcolors.to_hex(cmap(i)) for i, cluster in enumerate(
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
HER_lambert.boundary.plot(ax=ax, color='black', linewidth=1)
for cluster, color in cluster_colors.items():
    cluster_data = carto_i2m2[carto_i2m2[cluster_col] == cluster]
    cluster_data.plot(ax=ax, color=color, markersize=20, label=f"HER trouvée {c
ax.legend(loc='upper left', fontsize='medium', title='Clusters')
plt.title('Stations colorées par hydorrégion dominante dans leur cluster', font
plt.xlabel('Longitude', fontsize=12)
plt.ylabel('Latitude', fontsize=12)
plt.tight_layout()
plt.show()
```

```
/var/folders/p1/4yqk4cyx3058m3j04rtkr56m0000gn/T/ipykernel_53919/811036617.
```

The get_cmap function was deprecated in Matplotlib 3.7 and will be removed

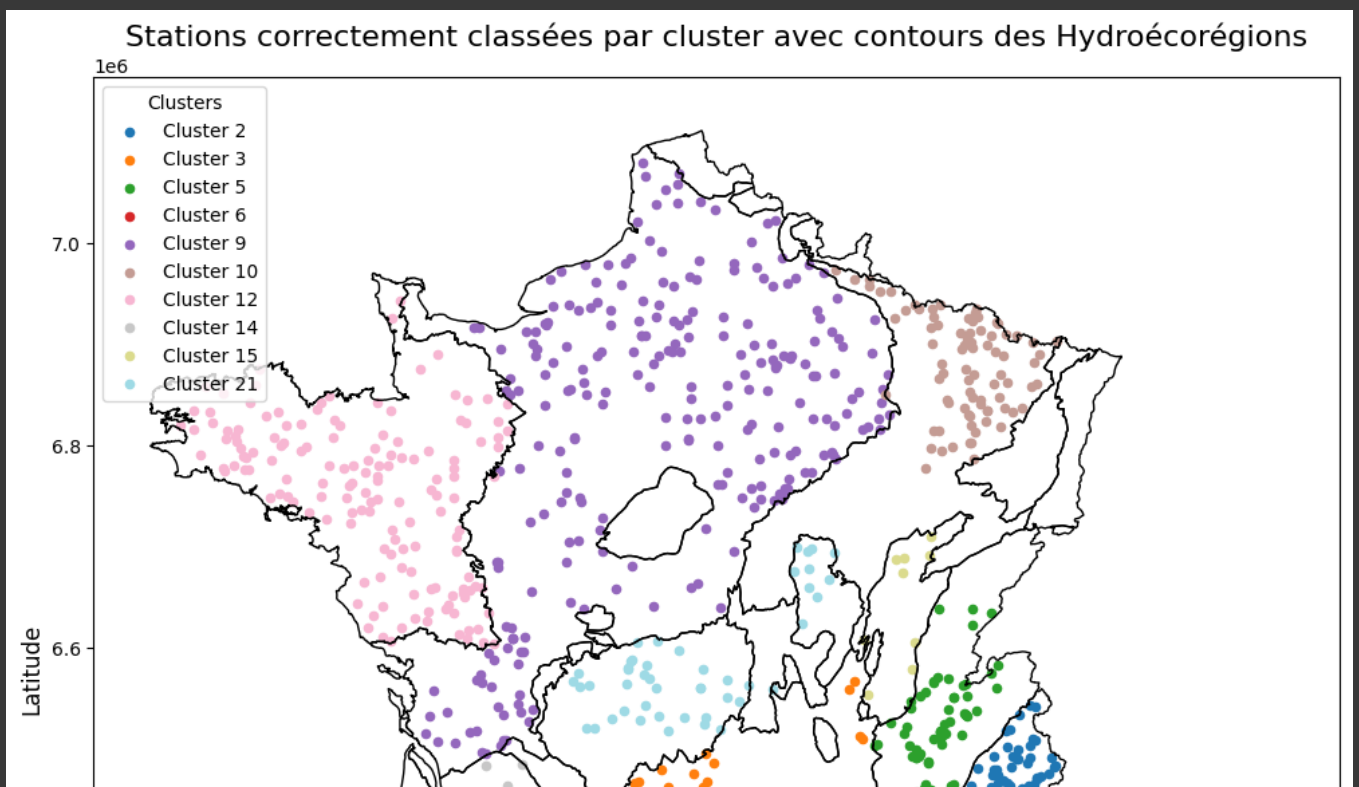


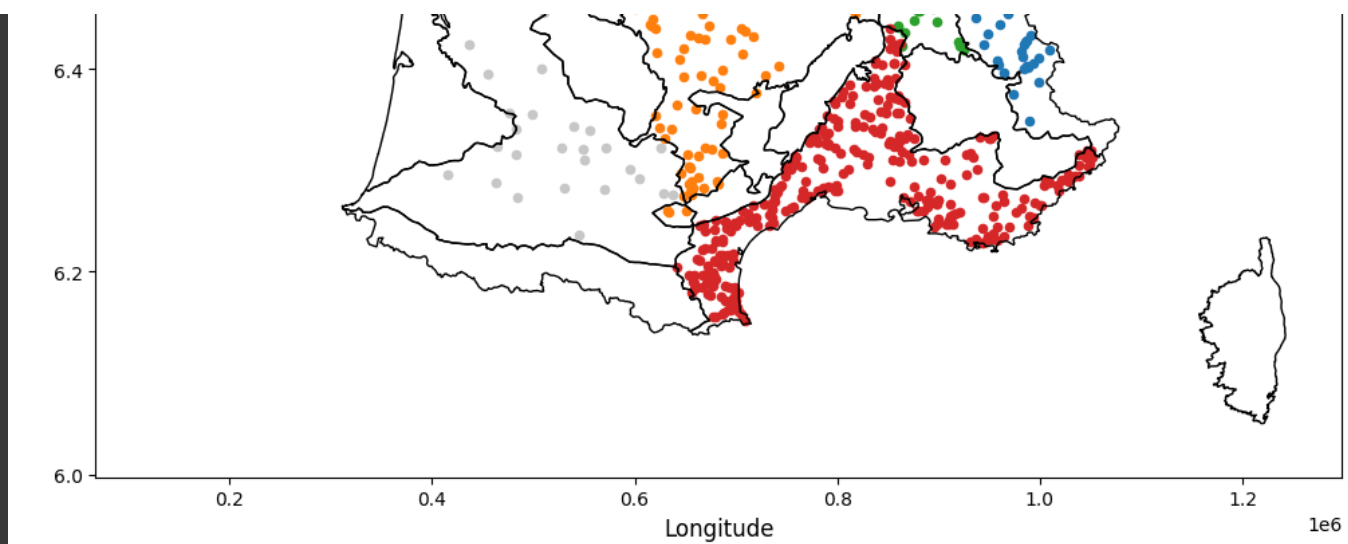


```
# Représentation des stations bien classées
correct_stations = df_analyse_etude[df_analyse_etude['correct_classification']]
carto_correct_i2m2 = gpd.GeoDataFrame(correct_stations, crs=crs_lambert,
                                      geometry=gpd.GeoSeries(correct_stations.a
HER_lambert = df_hydroregions.to_crs(crs_lambert)
unique_clusters_correct = carto_correct_i2m2[cluster_col].unique()
unique_clusters_correct = sorted(unique_clusters_correct)
cmap = cm.get_cmap('tab20', len(unique_clusters_correct))
cluster_colors_correct = {cluster: mcolors.to_hex(cmap(i)) for i, cluster in enumerate(unique_clusters_correct)}
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
HER_lambert.boundary.plot(ax=ax, color='black', linewidth=1)
for cluster, color in cluster_colors_correct.items():
    cluster_data = carto_correct_i2m2[carto_correct_i2m2[cluster_col] == cluster]
    cluster_data.plot(ax=ax, color=color, markersize=20, label=f"Cluster {cluster}")
ax.legend(loc='upper left', fontsize='medium', title='Clusters')
plt.title('Stations correctement classées par cluster avec contours des Hydroéc
plt.xlabel('Longitude', fontsize=12)
plt.ylabel('Latitude', fontsize=12)
plt.tight_layout()
plt.show()
```

→ /var/folders/p1/4yqk4cyx3058m3j04rtrkr56m0000gn/T/ipykernel_53919/1846449924

The `get_cmap` function was deprecated in Matplotlib 3.7 and will be removed





✓ Décalage temporel de 6 mois

Après avoir effectué le clustering avec un décalage temporel de 1 mois, les premiers résultats nous ont permis d'évaluer les regroupements d'hydroécostations en fonction des données physico-chimiques et biologiques. Pour approfondir cette analyse, nous allons explorer un décalage temporel plus important, de 6 mois, afin d'examiner si un lag différent peut améliorer la qualité des clusters, mieux refléter le délai entre les variations physico-chimiques et leurs répercussions biologiques, et enrichir notre compréhension des relations spatio-temporelles entre les hydroécostations.

```
df_pc_median_bio_median_6_month = pd.merge(df_pc_pivot_saison_median, df_hydrok
```

```
df_pc_median_bio_median_6_month.shape
```

```
↔ (35275, 20)
```

```
# Fusion avec le dataset des stations
df_pc_median_bio_median_6_month_with_coords = pd.merge(
    df_pc_median_bio_median_6_month,
    df_stations[['station', 'CoordXStationMesureEauxSurface', 'CoordYStationMes
on='station',
    how='inner'
)
```

```
df_correl_6 = df_pc_median_bio_median_6_month_with_coords.copy()
```



```
# Ajout des hydrorégions pour voir si des caractéristiques y sont corrélées
carto_i2m2_correl_6 = gpd.GeoDataFrame(df_correl_6, crs=crs_lambert, geometry =
HER_stations_correl_6=carto_i2m2_correl_6.sjoin(df_hydroregions.to_crs(crs_lambert), how='inner')
df_correl_6 = HER_stations_correl_6.drop(columns=['geometry', 'index_right', 'index_left'])
df_correl_6['saison'] = df_correl_6['saison'].factorize()[0]
# Mapping des colonnes pour pouvoir afficher la matrice de corrélation correctement
column_mapping = {
    'Ammonium - Eau': 'NH4',
    'Azote Kjeldahl - Eau': 'NKjeldahl',
    'Carbone Organique - Eau': 'CO',
    'Conductivité à 25°C - Eau': 'Cond25',
    'Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau': 'DB05',
    'Matières en suspension - Eau': 'MES',
    'Nitrates - Eau': 'NO3',
    'Nitrites - Eau': 'NO2',
    'Orthophosphates (P04) - Eau': 'P04',
    'Oxygène dissous - Eau': 'O2',
    'Phosphore total - Eau': 'Ptot',
    'Potentiel en Hydrogène (pH) - Eau': 'pH',
    'Taux de saturation en oxygène - Eau': 'O2_sat',
    'Température de l\'Eau - Eau': 'TempEau',
    'Turbidité Formazine Néphélométrique - Eau': 'Turbidité',
    'I2M2': 'I2M2',
    'CoordXStationMesureEauxSurface': 'CoordX',
    'CoordYStationMesureEauxSurface': 'CoordY'
}
df_correl_6 = df_correl_6.rename(columns=column_mapping, index=column_mapping)
```

```
correlation_matrix_6 = df_correl_6.corr()
np.fill_diagonal(correlation_matrix_6.values, np.nan)
fig = px.imshow(correlation_matrix_6, text_auto=".1f", color_continuous_scale=
fig.update_layout(xaxis=dict(tickangle=45), autosize=True, title_x=0.5, width=
fig.show()
```



Les mêmes observations sont présentes qu'auparavant : **I2M2** reste négativement corrélé avec Cond25, NO2, PO et Ptot, et ce, malgré le décalage de 6 mois. Cela suggère que ces paramètres sont probablement les principaux facteurs influençant l'état hydrobiologique de l'eau, indépendamment de l'impact des saisons.

✓ Clustering pour 6 mois de lag

```
df_pc_median_bio_median_6_month_with_coords.isnull().sum()
```

```
station 0
année 0
saison 0
Ammonium - Eau 1942
Azote Kjeldahl - Eau 2117
Carbone Organique - Eau 1602
Conductivité à 25°C - Eau 195
Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau 1843
Diuron - Eau 7569
Matières en suspension - Eau 382
Nitrates - Eau 1791
Nitrites - Eau 1803
Orthophosphates (P04) - Eau 1784
Oxygène dissous - Eau 199
Phosphore total - Eau 1491
Potentiel en Hydrogène (pH) - Eau 190
Taux de saturation en oxygène - Eau 1126
Température de l'Eau - Eau 139
Turbidité Formazine Néphélométrique - Eau 6503
I2M2 0
CoordXStationMesureEauxSurface 0
CoordYStationMesureEauxSurface 0
dtype: int64
```

```
missing_percentage_6 = df_pc_median_bio_median_6_month_with_coords.isnull().mean()
print(missing_percentage_6)
```

```
↔ station 0.000000
année 0.000000
saison 0.000000
Ammonium – Eau 9.141835
Azote Kjeldahl – Eau 9.965636
Carbone Organique – Eau 7.541308
Conductivité à 25°C – Eau 0.917949
Demande Biochimique en oxygène en 5 jours (D.B.O.5) – Eau 8.675799
Diuron – Eau 35.630561
Matières en suspension – Eau 1.798239
Nitrates – Eau 8.431013
Nitrites – Eau 8.487502
Orthophosphates (P04) – Eau 8.398061
Oxygène dissous – Eau 0.936779
Phosphore total – Eau 7.018783
Potentiel en Hydrogène (pH) – Eau 0.894412
Taux de saturation en oxygène – Eau 5.300570
Température de l'Eau – Eau 0.654333
Turbidité Formazine Néphélométrique – Eau 30.612437
I2M2 0.000000
CoordXStationMesureEauxSurface 0.000000
CoordYStationMesureEauxSurface 0.000000
dtype: float64
```

```
# on supprime aussi Diuron – Eau
df_pc_median_bio_median_6_month_with_coords.drop(columns=['Diuron – Eau'], inplace=True)
```

```
# Calculer le nombre de valeurs manquantes par ligne
missing_values_per_row_6 = df_pc_median_bio_median_6_month_with_coords.isnull()
missing_summary_6 = missing_values_per_row_6.value_counts().reset_index()
missing_summary_6.columns = ['Nombre de valeurs manquantes', 'Nombre de lignes']
print(missing_summary_6.sort_values('Nombre de valeurs manquantes'))
```

	Nombre de valeurs manquantes	Nombre de lignes
0	0	11844
1	1	6833
3	2	552
7	3	137
9	4	64
5	5	204
6	6	157
8	7	133
2	8	903
10	9	47
4	10	277
12	11	34
14	12	1
15	13	1
11	14	46
13	15	10

```
# Imputation des valeurs manquantes par la médiane
imputer = SimpleImputer(strategy='median') # ici on pourrait aussi mettre mean
colonnes = df_pc_median_bio_median_6_month_with_coords.columns
colonnes = colonnes.drop('station')
colonnes = colonnes.drop('année')
colonnes = colonnes.drop('saison')
colonnes = colonnes.drop('I2M2')
df_pc_median_bio_median_6_month_with_coords[colonnes] = imputer.fit_transform(df_pc_median_bio_median_6_month_with_coords[colonnes])
df_pc_median_bio_median_6_month_with_coords.isnull().sum()
```

```
station 0
année 0
saison 0
Ammonium – Eau 0
Azote Kjeldahl – Eau 0
Carbone Organique – Eau 0
Conductivité à 25°C – Eau 0
Demande Biochimique en oxygène en 5 jours (D.B.O.5) – Eau 0
Matières en suspension – Eau 0
Nitrates – Eau 0
Nitrites – Eau 0
Orthophosphates (P04) – Eau 0
Oxygène dissous – Eau 0
Phosphore total – Eau 0
Potentiel en Hydrogène (pH) – Eau 0
Taux de saturation en oxygène – Eau 0
Température de l'Eau – Eau 0
Turbidité Formazine Néphélométrique – Eau 0
I2M2 0
CoordXStationMesureEauxSurface 0
CoordYStationMesureEauxSurface 0
dtype: int64
```

```
df_pc_median_bio_median_6_month_with_coords_for_regression = df_pc_median_bio_n
```

```
# Mapper chaque saison à un trimestre
saison_to_quarter = {
    "Hiver": 1,
    "Printemps": 2,
    "Été": 3,
    "Automne": 4
}

df_pc_median_bio_median_6_month_with_coords['trimestre'] = df_pc_median_bio_mec
# drop saison
df_pc_median_bio_median_6_month_with_coords.drop(columns=['saison'], inplace=True)
df_pc_median_bio_median_6_month_with_coords.head(5)
```



	station	année	Ammonium - Eau	Azote Kjeldahl - Eau	Carbone Organique - Eau	Conductivité à 25°C - Eau	Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau
0	5001800	2007	0.04	1.0	6.60	895.0	0.50
1	5001800	2008	0.04	1.0	5.50	803.5	0.65
2	5001800	2009	0.06	1.0	4.60	833.5	0.50
3	5005350	2007	0.12	1.0	1.65	593.0	0.90
4	5005350	2008	0.04	1.0	1.50	632.5	1.60

5 rows × 21 columns

```
# Convertir l'identifiant de station en variable catégorielle (cela crée un enc
df_pc_median_bio_median_6_month_with_coords['station'] = df_pc_median_bio_media
df_pc_median_bio_median_6_month_with_coords['trimestre'] = df_pc_median_bio_mec
df_pc_median_bio_median_6_month_with_coords['année'] = df_pc_median_bio_median_
df_pc_median_bio_median_6_month_with_coords.dtypes
```

```
↔ station object
   année int64
Ammonium – Eau float64
Azote Kjeldahl – Eau float64
Carbone Organique – Eau float64
Conductivité à 25°C – Eau float64
Demande Biochimique en oxygène en 5 jours (D.B.O.5) – Eau float64
Matières en suspension – Eau float64
Nitrates – Eau float64
Nitrites – Eau float64
Orthophosphates (P04) – Eau float64
Oxygène dissous – Eau float64
Phosphore total – Eau float64
Potentiel en Hydrogène (pH) – Eau float64
Taux de saturation en oxygène – Eau float64
Température de l'Eau – Eau float64
Turbidité Formazine Néphélométrique – Eau float64
I2M2 float64
CoordXStationMesureEauxSurface float64
CoordYStationMesureEauxSurface float64
trimestre int64
dtype: object
```

```
# drop les coordonnées
df_pc_median_bio_median_6_month_c = df_pc_median_bio_median_6_month_with_coords
df_pc_median_bio_median_6_month_c = df_pc_median_bio_median_6_month_with_coords
df_clustering_6 = df_pc_median_bio_median_6_month_c.copy()
```

```
# Encodage des stations
le = LabelEncoder()
df_clustering_6['station_encoded'] = le.fit_transform(df_clustering_6['station']
df_clustering_6.drop(columns=['station'], inplace=True)
# Normalisation des données pour qu'elles aient toutes la même importance
scaler = MinMaxScaler()
normalized_data_6 = scaler.fit_transform(df_clustering_6)
normalized_data_6 = pd.DataFrame(normalized_data_6, columns=df_clustering_6.col
```

On teste ici directement avec 22 clusters, qui est le nombre d'hydroécroégions.

```
kmeans = KMeans(n_clusters=22, random_state=42)
kmeans.fit(normalized_data_6)
```



KMeans

```
KMeans(n_clusters=22, random_state=42)
```

```
# Analyse des centres des clusters
cluster_centers_6 = pd.DataFrame(kmeans.cluster_centers_, columns=df_clustering)
print("Centres des clusters :")
print(cluster_centers_6)
# Variabilité des caractéristiques par cluster
influence_6 = cluster_centers_6.std(axis=0)
print("\nInfluence des caractéristiques :")
print(influence_6.sort_values(ascending=False))
```



Centres des clusters :

	année	Ammonium – Eau	Azote Kjeldahl – Eau	Carbone Organique – Eau
0	0.698467	0.003703	0.103479	0.090523
1	0.328105	0.015226	0.141999	0.165752
2	0.485039	0.153096	0.295034	0.259995
3	0.726435	0.006011	0.106410	0.155199
4	0.295109	0.009563	0.167491	0.111694
5	0.728114	0.003733	0.102967	0.053176
6	0.765597	0.007173	0.108304	0.156238
7	0.565142	0.007076	0.108252	0.128290
8	0.622577	0.016792	0.143201	0.164365
9	0.225716	0.010606	0.196300	0.060238
10	0.718125	0.026329	0.130423	0.197367
11	0.797656	0.008145	0.120576	0.217816
12	0.290601	0.014211	0.171691	0.186506
13	0.657866	0.019224	0.209163	0.484424
14	0.886146	0.011175	0.127999	0.216051
15	0.193605	0.009852	0.177127	0.134025
16	0.809513	0.005938	0.107138	0.179006
17	0.791328	0.014402	0.118397	0.151644
18	0.730698	0.012129	0.135354	0.168549
19	0.833558	0.005049	0.101595	0.079183
20	0.301445	0.009649	0.185023	0.084360
21	0.281307	0.024189	0.198796	0.165406

Conductivité à 25°C – Eau \

0	0.088203
1	0.235228
2	0.464355
3	0.125860
4	0.169859
5	0.188359
6	0.291556
7	0.187297


```

8      0.277654
9      0.238012
10     0.398769
11     0.249009
12     0.352056
13     0.159425
14     0.221966
15     0.124263
16     0.073163
17     0.368012
18     0.305563
19     0.268921
20     0.234057
21     0.306888

```

Demande Biochimique en oxygène en 5 jours (D.B.O.5) – Eau \

```

0      0.065298
1      0.130716
2      0.193447
3      0.079933
4      0.071575
5      0.045762
6      0.077588
7      0.074638
8      0.104098

```

Après les paramètres de spatio-temporalité, les caractéristiques les plus influentes sont :

- I2M2
- Nitrates
- Conductivité à 25°C
- Carbone Organique
- Température de l'eau

Ces paramètres suggèrent des indicateurs clés de l'état de l'eau, permettant ainsi de détecter les hydroécorégions.

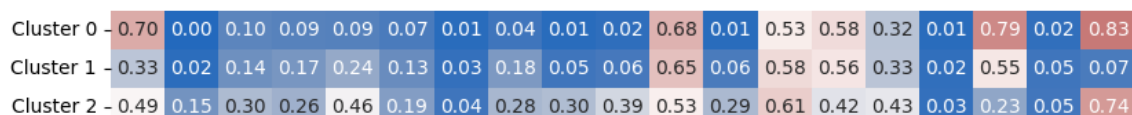
```

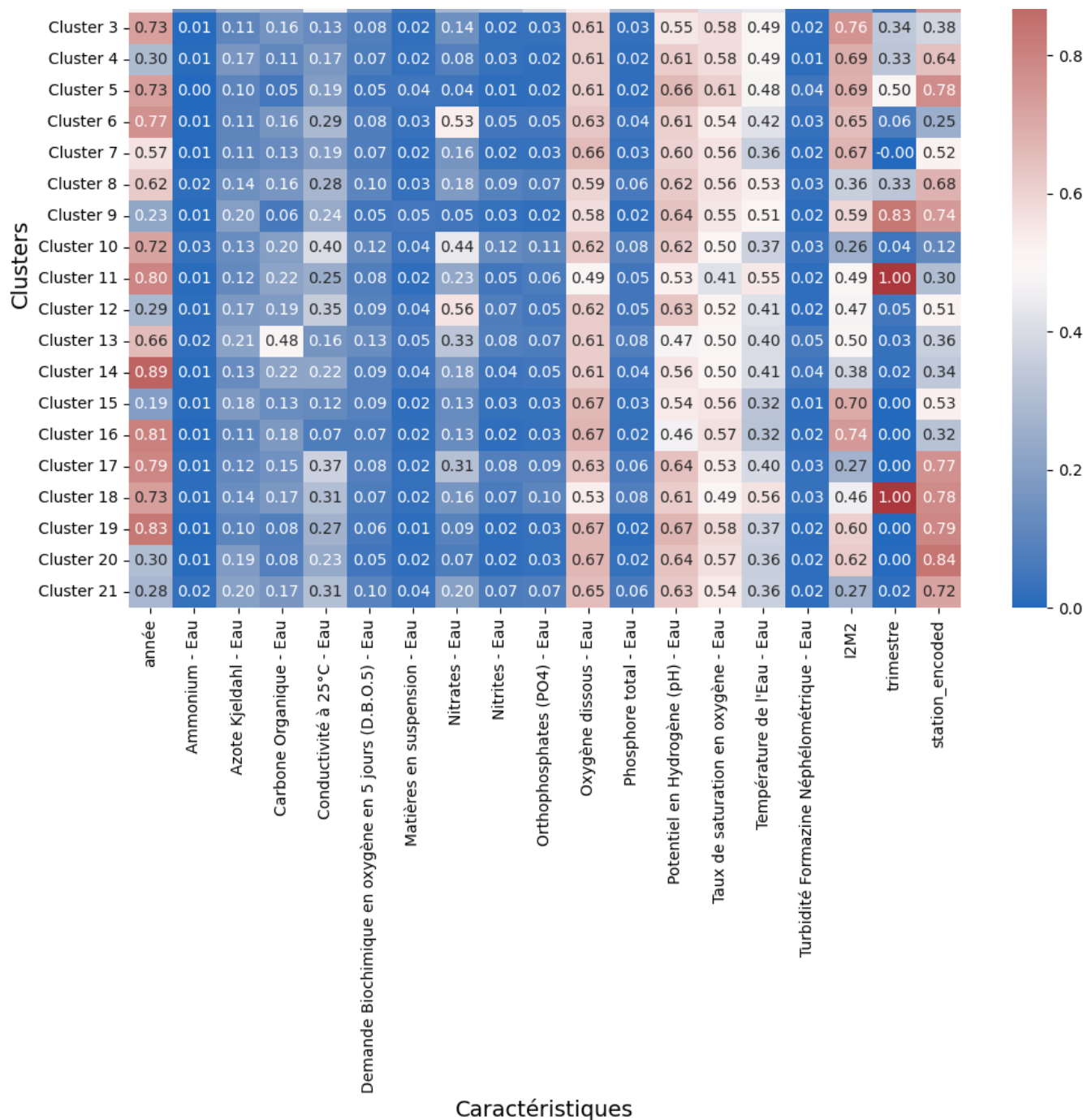
# Matrice des centres des clusters
plt.figure(figsize=(12, 8))
sns.heatmap(cluster_centers_6, annot=True, fmt=".2f", cmap="vlag", xticklabels=
plt.title("Centres des clusters", fontsize=16)
plt.xlabel("Caractéristiques", fontsize=14)
plt.ylabel("Clusters", fontsize=14)
plt.show()

```



Centres des clusters





```
df_pc_median_bio_median_6_month_with_clusters = df_pc_median_bio_median_6_month_with_clusters  
df_pc_median_bio_median_6_month_with_clusters['cluster'] = kmeans.labels_
```

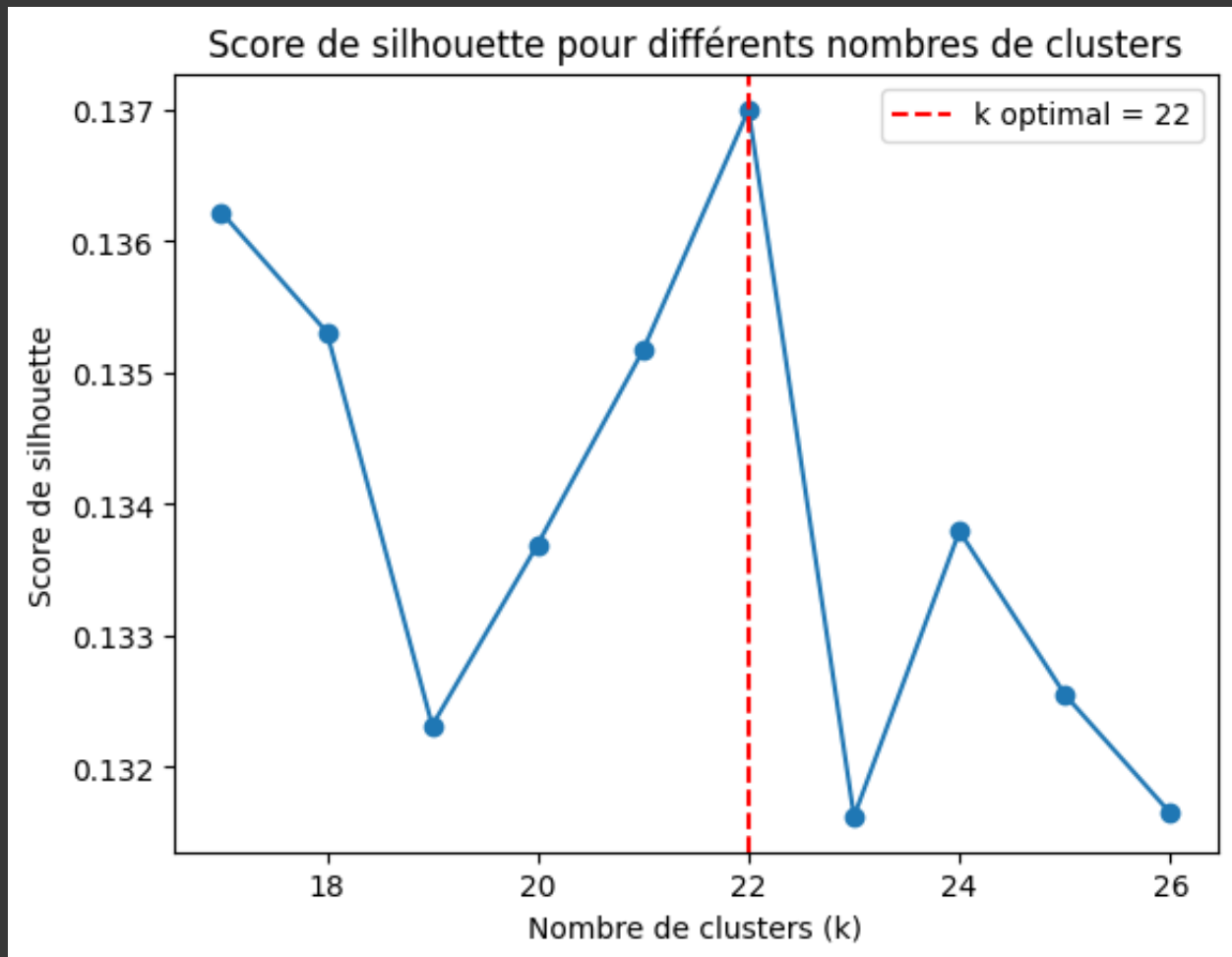
```
# Calcul du score de silhouette  
score_6 = silhouette_score(normalized_data_6, df_pc_median_bio_median_6_month_with_clusters['cluster'])  
print(f"Silhouette score: {score_6}")
```

➡ Silhouette score: 0.1370000820490834

```
# Recherche du meilleur nombre de clusters selon le score de silhouette  
silhouette_scores_6 = []  
for k in k_values:  
    kmeans = KMeans(n_clusters=k, random_state=42)  
    cluster_labels = kmeans.fit_predict(normalized_data_6)  
    score = silhouette_score(normalized_data_6, cluster_labels)  
    silhouette_scores_6.append(score)  
best_k_6 = k_values[silhouette_scores_6.index(max(silhouette_scores_6))]  
print(f"Le meilleur nombre de clusters est : {best_k_6}")
```

➡ Le meilleur nombre de clusters est : 22

```
plt.plot(k_values, silhouette_scores_6, marker='o')
plt.title("Score de silhouette pour différents nombres de clusters")
plt.xlabel("Nombre de clusters (k)")
plt.ylabel("Score de silhouette")
plt.axvline(x=best_k_6, color='r', linestyle='--', label=f"k optimal = {best_k_6}")
plt.legend()
plt.show()
```



Cette fois ci, le score de silhouette est exactement le nombre d'hydroécorégions.

```
cluster_distribution = df_pc_median_bio_median_6_month_with_clusters.groupby('s')
print(cluster_distribution)
stations_100_percent = cluster_distribution[cluster_distribution.eq(100.0).sum(
print("Stations à 100% dans un seul cluster :", end=' ')
print(stations_100_percent.shape[0])
print("Pourcentage de stations 100% dans le même cluster : ", round(stations_100_percent.mean(), 2))
```

```

↔ cluster      0      1      2      3      4      5      6
station
1000477 0.0    0.000000  0.000000  0.000000  0.0    0.000000  14.285714  0
1000602 0.0    0.000000  0.000000  0.000000  0.0    0.000000  0.000000  0
1000605 0.0    0.000000  0.000000  16.666667  0.0    0.000000  50.000000  0
1001122 0.0    28.571429  0.000000  0.000000  0.0    0.000000  14.285714  0
1001131 0.0    28.571429  0.000000  0.000000  0.0    0.000000  0.000000  0
...
6999125 0.0    0.000000  16.666667  0.000000  0.0    0.000000  0.000000  0
6999137 0.0    0.000000  0.000000  0.000000  0.0    16.666667  0.000000  0
6999153 0.0    0.000000  0.000000  0.000000  0.0    0.000000  0.000000  0
6999176 0.0    0.000000  0.000000  0.000000  0.0    0.000000  0.000000  0
6999178 0.0    0.000000  0.000000  0.000000  0.0    0.000000  0.000000  0

cluster      8      9      ...    12      13      14      15      16
station
1000477 0.000000  0.0    ...    0.0    0.000000  0.000000  0.0    0.000000  0
1000602 0.000000  0.0    ...    0.0    0.000000  0.000000  0.0    0.000000  0
1000605 0.000000  0.0    ...    0.0    0.000000  0.000000  0.0    0.000000  0
1001122 0.000000  0.0    ...    0.0    0.000000  14.285714  0.0    14.285714  0
1001131 0.000000  0.0    ...    0.0    14.285714  14.285714  0.0    0.000000  0
...
6999125 0.000000  0.0    ...    0.0    0.000000  0.000000  0.0    0.000000  25
6999137 16.666667  0.0    ...    0.0    0.000000  0.000000  0.0    0.000000  0
6999153 25.000000  0.0    ...    0.0    0.000000  0.000000  0.0    0.000000  75
6999176 50.000000  0.0    ...    0.0    0.000000  0.000000  0.0    0.000000  50
6999178 50.000000  0.0    ...    0.0    0.000000  0.000000  0.0    0.000000  0

cluster      18      19      20      21
station
1000477 0.000000  0.000000  0.0    0.0
1000602 0.000000  0.000000  0.0    0.0
1000605 0.000000  0.000000  0.0    0.0
1001122 0.000000  0.000000  0.0    0.0
1001131 0.000000  0.000000  0.0    0.0
...
6999125 16.666667  16.666667  0.0    25.0
6999137 0.000000  66.666667  0.0    0.0
6999153 0.000000  0.000000  0.0    0.0
6999176 0.000000  0.000000  0.0    0.0
6999178 0.000000  50.000000  0.0    0.0

```

[2853 rows x 22 columns]

Stations à 100% dans un seul cluster : 692

Pourcentage de stations 100% dans le même cluster : 24.26 %

```
df_pc_median_bio_median_6_month_with_clusters_with_coord = df_pc_median_bio_med
df_pc_median_bio_median_6_month_with_clusters_with_coord['cluster'] = kmeans.la
# Attribution du cluster dominant à chaque station
station_cluster_counts = df_pc_median_bio_median_6_month_with_clusters_with_coc
dominant_cluster_per_station = station_cluster_counts.loc[station_cluster_count
df_pc_median_bio_median_6_month_with_clusters_with_coord = df_pc_median_bio_med
df_pc_median_bio_median_6_month_with_clusters_with_coord.rename(columns={'clust
df_analyse_6 = df_pc_median_bio_median_6_month_with_clusters_with_coord[['stati
```

```
carto_correct_i2m2_6 = gpd.GeoDataFrame(df_analyse_6, crs=crs_lambert,
                                         geometry=gpd.GeoSeries(df_analyse_6.agg
HER_stations_correct_6 = carto_correct_i2m2_6.sjoin(df_hydroregions.to_crs(crs_
df_analyse_etude_6 = HER_stations_correct_6.drop(columns=['geometry', 'index_r
df_analyse_etude_6.rename(columns={'CdHER1': 'hydroecoregion'}, inplace=True)
# hydroécorégion dominante par cluster
dominant_class_per_cluster_6 = df_analyse_etude_6.groupby('cluster_dominant_sta
dominant_class_per_cluster_6.columns = ['cluster_dominant_station', 'dominant_h
df_analyse_etude_6 = df_analyse_etude_6.merge(dominant_class_per_cluster_6, on=
df_analyse_etude_6.drop_duplicates(subset='station', inplace=True)
```

```
# association cluster - hydroécorégion
clusters_hydroregions_6 = df_analyse_etude_6.groupby('cluster_dominant_station'
clusters_hydroregions_6.columns = ['cluster_dominant_station', 'dominant_hydroec
clusters_hydroregions_6['dominant_hydroecoregion_cluster'] = clusters_hydroregi
df_hydroregions['CdHER1'] = df_hydroregions['CdHER1'].astype(str)
clusters_hydroregions_with_names_6 = clusters_hydroregions_6.merge(df_hydroregi
clusters_hydroregions_with_names_6 = clusters_hydroregions_with_names_6.rename(
print(clusters_hydroregions_with_names_6[['cluster_dominant_station', 'dominant
```

```
↔ cluster_dominant_station dominant_hydroecoregion_cluster \
0 0 16
1 1 10
2 2 14
3 3 21
4 4 5
5 5 2
6 6 9
7 7 21
8 8 5
9 9 6
10 10 9
11 11 12
12 12 1
13 13 12
14 14 9
15 15 3
16 16 12
17 17 5
```

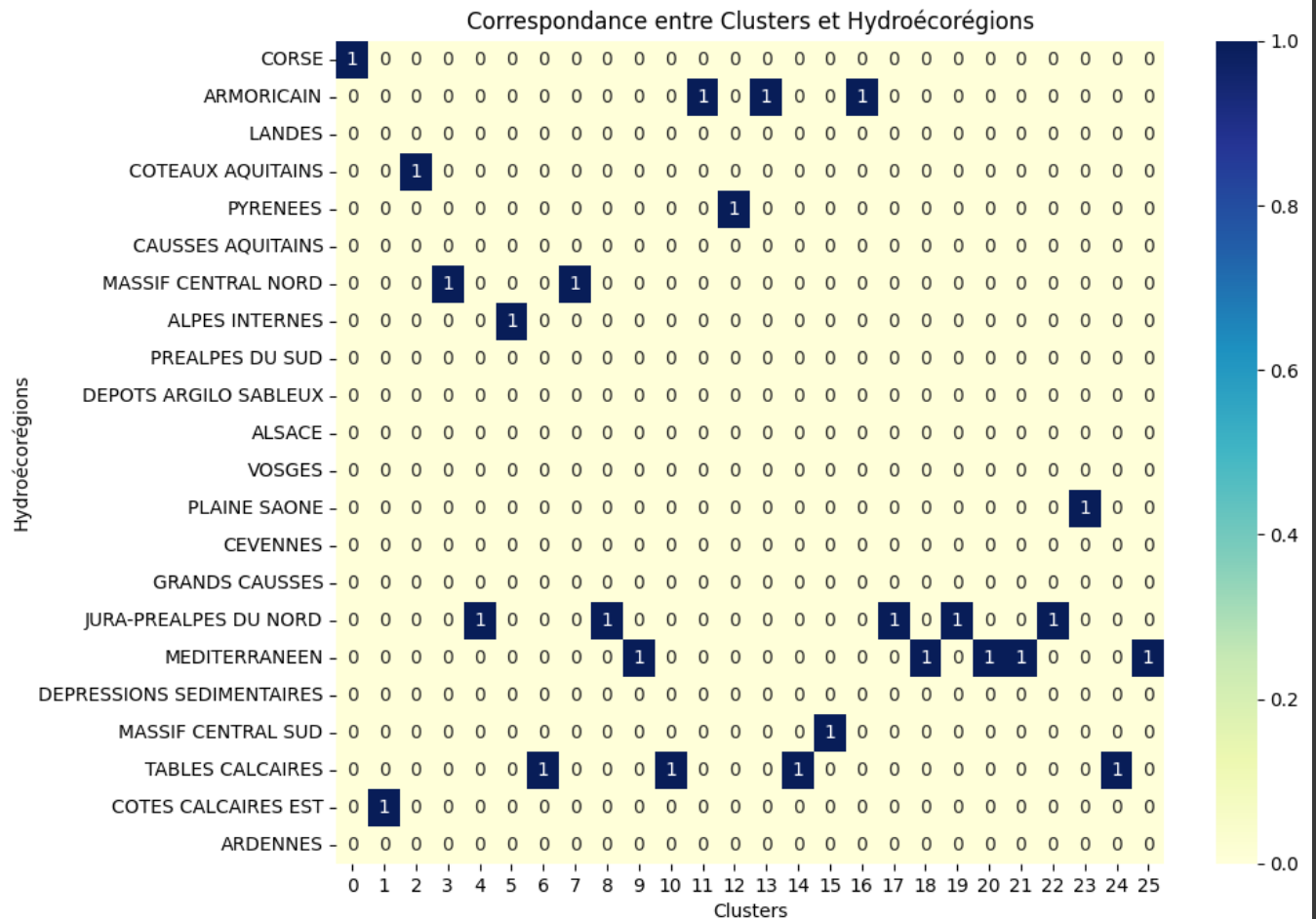
18	18	6
19	19	5
20	20	6
21	21	6
22	22	5
23	23	15
24	24	9
25	25	6

	nom_dominant_hydroecoregion_cluster
0	CORSE
1	COTES CALCAIRES EST
2	COTEAUX AQUITAINS
3	MASSIF CENTRAL NORD
4	JURA-PREALPES DU NORD
5	ALPES INTERNES
6	TABLES CALCAIRES
7	MASSIF CENTRAL NORD
8	JURA-PREALPES DU NORD
9	MEDITERRANEEN
10	TABLES CALCAIRES
11	ARMORICAIN
12	PYRENEES
13	ARMORICAIN
14	TABLES CALCAIRES
15	MASSIF CENTRAL SUD
16	ARMORICAIN
17	JURA-PREALPES DU NORD
18	MEDITERRANEEN
19	JURA-PREALPES DU NORD
20	MEDITERRANEEN
21	MEDITERRANEEN
22	JURA-PREALPES DU NORD
23	PLAINE SAONE
24	TABLES CALCAIRES
25	MEDITERRANEEN

```

all_hydroecoregions = df_hydroregions['NomHER1'].unique()
cross_tab_6 = pd.crosstab(clusters_hydroregions_with_names_6['nom_dominant_hydroecoregion'], all_hydroecoregions)
plt.figure(figsize=(10, 8))
sns.heatmap(cross_tab_6, annot=True, fmt='d', cmap='YlGnBu', cbar=True)
plt.title("Correspondance entre Clusters et Hydroécorégions")
plt.xlabel("Clusters")
plt.ylabel("Hydroécorégions")
plt.show()

```



Cette fois ci 12 hydroécorégions sont indentifiés.

Comme pour le clustering avec décalage temporel d'1 mois, on observe des hydroécorégions associés à plusieurs clusters, et les mêmes que pou le précédent clustering, notamment l'hydroécorégion méditerranéen.

```
df_analyse_etude_6['correct_classification'] = df_analyse_etude_6['hydroecoregi
num_correctly_classified_6 = df_analyse_etude_6['correct_classification'].sum()
total_stations_6 = df_analyse_etude_6.shape[0]
accuracy_6 = num_correctly_classified_6 / total_stations_6
print(f"Nombre de stations bien classées: {num_correctly_classified_6}/{total_s
print(f"Taux de classification correcte: {accuracy_6 * 100:.2f}%")
```

➞ Nombre de stations bien classées: 1071/2836
Taux de classification correcte: 37.76%

On a 3% de stations mieux classés qu'avec le lag d'1 mois.

```
# Représentation des stations bien classés
correct_stations_6 = df_analyse_etude_6[df_analyse_etude_6['correct_classificat
carto_correct_i2m2_6 = gpd.GeoDataFrame(correct_stations_6, crs=crs_lambert,
                                         geometry=gpd.GeoSeries(correct_stations
HER_lambert = df_hydroregions.to_crs(crs_lambert)
unique_clusters_correct_6 = carto_correct_i2m2_6[cluster_col].unique()
unique_clusters_correct_6 = sorted(unique_clusters_correct_6)
cmap = cm.get_cmap('tab20', len(unique_clusters_correct_6))
cluster_colors_correct_6 = {cluster: mcolors.to_hex(cmap(i)) for i, cluster in
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
HER_lambert.boundary.plot(ax=ax, color='black', linewidth=1)
for cluster, color in cluster_colors_correct_6.items():
    cluster_data = carto_correct_i2m2_6[carto_correct_i2m2_6[cluster_col] == cl
    cluster_data.plot(ax=ax, color=color, markersize=20, label=f"Cluster {clust
ax.legend(loc='upper left', fontsize='medium', title='Clusters')
plt.title('Stations correctement classées par cluster avec contours des Hydroéc
plt.xlabel('Longitude', fontsize=12)
plt.ylabel('Latitude', fontsize=12)
plt.tight_layout()
plt.show()
```

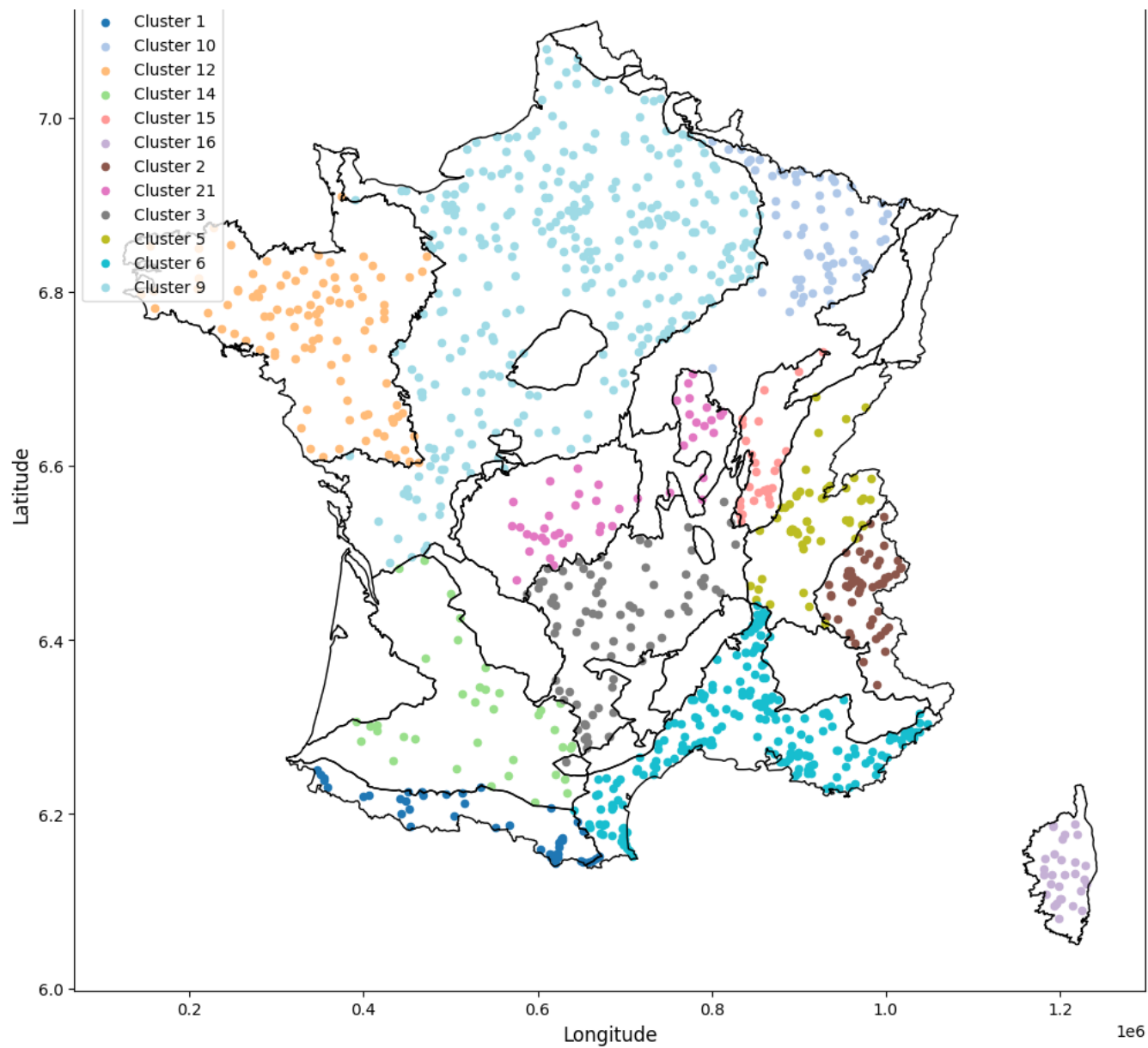
➞ /var/folders/p1/4yqk4cyx3058m3j04rtkr56m0000gn/T/ipykernel_53919/3498149462

The get_cmap function was deprecated in Matplotlib 3.7 and will be removed

Stations correctement classées par cluster avec contours des Hydroécorégions

1e6

Clusters



On observe que dans les deux clusterings, certaines régions sont plus facilement identifiables, notamment à l'ouest, dans la région de Lyon, autour de la Méditerranée, et dans le nord de la Lorraine. La Corse n'est détectée qu'avec le lag de 6 mois, tout comme la région au sud-ouest.

Les zones couvertes avec le lag de 6 mois montrent une amélioration de la précision dans la détection des hydroécorégions. En effet, le lag de 6 mois semble plus pertinent que celui de 1 mois pour étudier l'impact de I2M2 dans la caractérisation des hydroécorégions. Nous espérons également que cette amélioration se reflète dans la régression qui sera réalisée par la suite, en posant l'hypothèse que 6 mois offre une meilleure base pour l'analyse.

✓ Régression : prédiction de I2M2

Dans cette section, nous explorons la possibilité de prédire l'état biologique de l'eau, représenté par l'indice I2M2, à partir des paramètres physico-chimiques, des informations spatiales (hydroécorégions) et de la dimension temporelle (saisons et décalages temporels).

En particulier, nous utilisons un lag de 6 mois pour les paramètres physico-chimiques, car les résultats du clustering précédemment réalisés avec cette configuration se sont révélés légèrement meilleurs. Cela nous conduit à supposer qu'un décalage temporel de 6 mois reflète plus fidèlement une relation sous-jacente entre les propriétés physico-chimiques et biologiques de l'eau qu'un lag d'1 mois.

Ainsi, en réutilisant les données préalablement préparées et enrichies avec l'identifiant des hydroécorégions, nous construisons un modèle de régression pour évaluer dans quelle mesure ces variables permettent de prédire efficacement I2M2. Cette démarche vise à approfondir notre compréhension des interactions entre les propriétés physico-chimiques et biologiques, tout en testant la pertinence de ces paramètres comme facteurs explicatifs de l'état des écosystèmes aquatiques.

```
df_pc_median_bio_median_6_month_with_coords
df_for_regression = df_pc_median_bio_median_6_month_with_coords.copy()
```

```
# Pour rappel :
df_for_regression
```



	station	année	Ammonium - Eau	Azote Kjeldahl - Eau	Carbone Organique - Eau	Conductivité à 25°C - Eau	Deman Biochimiq en oxygène en 5 jou (D.B.O.5) E
0	5001800	2007	0.040	1.0	6.60	895.0	0.
1	5001800	2008	0.040	1.0	5.50	803.5	0.
2	5001800	2009	0.060	1.0	4.60	833.5	0.
3	5005350	2007	0.120	1.0	1.65	593.0	0.
4	5005350	2008	0.040	1.0	1.50	632.5	1.
...
21238	6453450	2022	0.010	0.5	2.25	456.0	0.
21239	6446401	2021	0.025	0.5	1.50	503.0	0.
21240	6446401	2022	0.020	0.5	2.50	463.0	1.
21241	6446330	2021	0.020	0.5	1.30	529.5	0.
21242	6446330	2022	0.010	0.5	1.60	541.0	0.

21243 rows × 21 columns

```
# Ajouter le Code HER1
test= df_for_regression.copy()
carto_her = gpd.GeoDataFrame(test, crs=crs_lambert, geometry=gpd.GeoSeries(test
HER_stations_her = carto_her.sjoin(df_hydroregions.to_crs(crs_lambert), predic
test = HER_stations_her.drop(columns=['geometry', 'index_right', 'gid', 'NomHEF
test.rename(columns={'CdHER1': 'hydroecoregion'}, inplace=True)
test.head(5)
```



	station	année	Ammonium - Eau	Azote Kjeldahl - Eau	Carbone Organique - Eau	Conductivité à 25°C - Eau	Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau
0	5001800	2007	0.04	1.0	6.60	895.0	0.50
1	5001800	2008	0.04	1.0	5.50	803.5	0.65
2	5001800	2009	0.06	1.0	4.60	833.5	0.50
3	5005350	2007	0.12	1.0	1.65	593.0	0.90
4	5005350	2008	0.04	1.0	1.50	632.5	1.60

5 rows x 22 columns

```
# drop les coordonnées
df_for_regression = test.copy()
df_for_regression.drop(columns=['CoordXStationMesureEauxSurface', 'CoordYStatic
df_for_regression.head(5)
```



	station	année	Ammonium - Eau	Azote Kjeldahl - Eau	Carbone Organique - Eau	Conductivité à 25°C - Eau	Demande Biochimique en oxygène en 5 jours (D.B.O.5) - Eau
0	5001800	2007	0.04	1.0	6.60	895.0	0.50
1	5001800	2008	0.04	1.0	5.50	803.5	0.65
2	5001800	2009	0.06	1.0	4.60	833.5	0.50
3	5005350	2007	0.12	1.0	1.65	593.0	0.90
4	5005350	2008	0.04	1.0	1.50	632.5	1.60

```
# valeur min et max de I2M2 : vérifier que c'est bien 0 et 1
print(df_for_regression['I2M2'].min())
print(df_for_regression['I2M2'].max())
```

```
0.0
1.0
```

```
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
# Encodage des stations
le = LabelEncoder()
df_for_regression['station_encoded'] = le.fit_transform(df_for_regression['stat
df_for_regression["hydroecoregion_encoded"] = le.fit_transform(df_for_regressio
df_for_regression.drop(columns=['station', 'hydroecoregion'], inplace=True)
# Normalisation des données pour qu'elles aient toutes la même importance
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(df_for_regression)
normalized_data = pd.DataFrame(normalized_data, columns=df_for_regression.colun
# df_for_regression = pd.concat([df_for_regression[['station_encoded', 'hydroec
```

```
from sklearn.model_selection import train_test_split
X = normalized_data.drop(columns=['I2M2'])
y = normalized_data['I2M2']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
model = LinearRegression()
cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_sc
mean_cv_score = -cv_scores.mean()
print(f"Mean CV score: {mean_cv_score}")
```

↔ Mean CV score: 0.03133060488047301

Le fait que y soit compris entre 0 et 1 est à prendre en compte.

Un score moyen de validation croisée (Mean CV score) dans une régression représente l'erreur quadratique moyenne (MSE) qui mesure la différence entre les valeurs réelles et les valeurs prédites par le modèle. Le MSE est exprimé dans les mêmes unités que la variable cible.

Puisque la plage de y est de 0 à 1, un MSE de 0.03 signifie que, en moyenne, l'écart quadratique entre les valeurs réelles et prédites est de 0.03, ce qui est relativement faible par rapport à l'échelle totale (0 à 1).

```

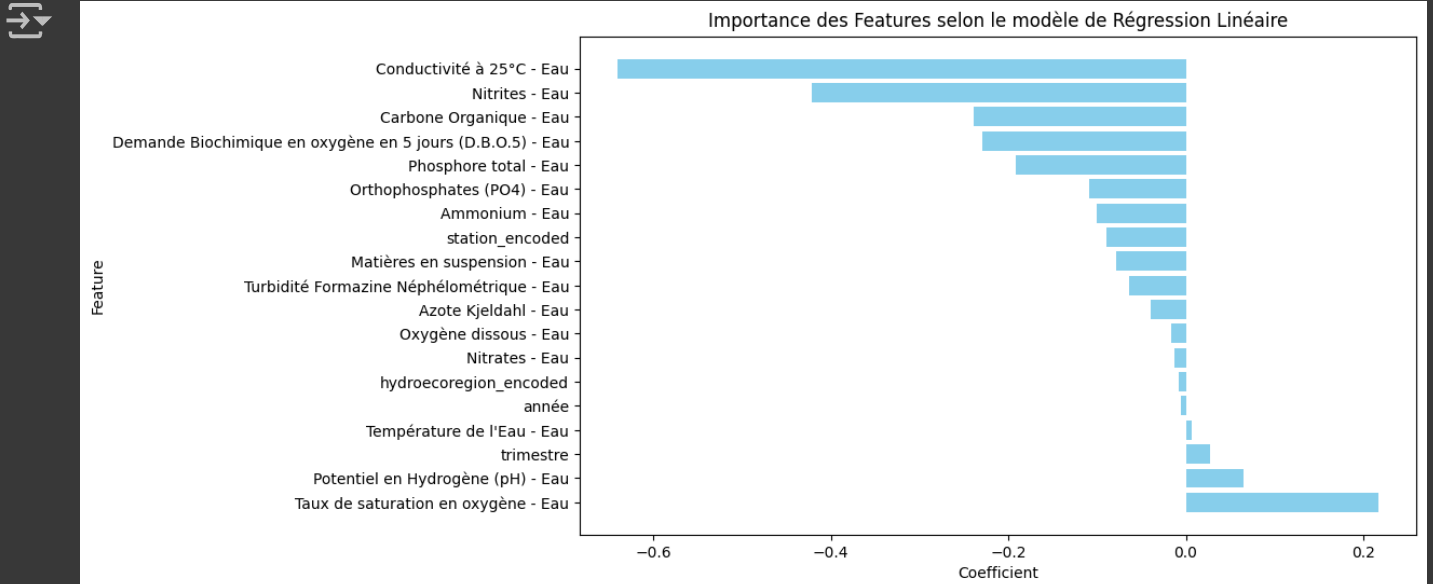
model.fit(X_train, y_train)
coefficients = model.coef_
features = X_train.columns
coef_df = pd.DataFrame(coefficients, index=features, columns=['Coefficient'])
coef_df = coef_df.sort_values('Coefficient', ascending=False)
print(coef_df)

```



	Coefficient
Taux de saturation en oxygène - Eau	0.216832
Potentiel en Hydrogène (pH) - Eau	0.064725
trimestre	0.027391
Température de l'Eau - Eau	0.007042
année	-0.005469
hydroecoregion_encoded	-0.008531
Nitrates - Eau	-0.013270
Oxygène dissous - Eau	-0.015859
Azote Kjeldahl - Eau	-0.039886
Turbidité Formazine Néphélométrique - Eau	-0.063309
Matières en suspension - Eau	-0.078234
station_encoded	-0.089366
Ammonium - Eau	-0.100480
Orthophosphates (P04) - Eau	-0.108816
Phosphore total - Eau	-0.191791
Demande Biochimique en oxygène en 5 jours (D.B....	-0.228816
Carbone Organique - Eau	-0.238602
Nitrites - Eau	-0.420908
Conductivité à 25°C - Eau	-0.640112


```
# Visualisation des coefficients
plt.figure(figsize=(10, 6))
plt.barh(coef_df.index, coef_df['Coefficient'], color='skyblue')
plt.xlabel('Coefficient')
plt.ylabel('Feature')
plt.title('Importance des Features selon le modèle de Régression Linéaire')
plt.show()
```



On regarde les caractéristiques qui participent le plus à la détermination de l'indice I2M2 en regardant les valeurs absolues des différents paramètres.

```
coef_df_sorted_abs = coef_df.abs().sort_values('Coefficient', ascending=False)
print(coef_df_sorted_abs)
```

	Coefficient
Conductivité à 25°C - Eau	0.640112
Nitrites - Eau	0.420908
Carbone Organique - Eau	0.238602
Demande Biochimique en oxygène en 5 jours (D.B....	0.228816
Taux de saturation en oxygène - Eau	0.216832
Phosphore total - Eau	0.191791
Orthophosphates (P04) - Eau	0.108816
Ammonium - Eau	0.100480
station_encoded	0.089366
Matières en suspension - Eau	0.078234
Potentiel en Hydrogène (pH) - Eau	0.064725
Turbidité Formazine Néphélométrique - Eau	0.063309
Azote Kjeldahl - Eau	0.039886
trimestre	0.027391
Oxygène dissous - Eau	0.015859
Nitrates - Eau	0.013270
hydroecoregion_encoded	0.008531
Température de l'Eau - Eau	0.007042
année	0.005469

Résultats

D'après les coefficients les plus éloignés de zéro, les six caractéristiques les plus influentes pour la prédiction de I2M2 sont, dans l'ordre :

- Conductivité à 25°C - Eau
- Nitrites - Eau
- Carbone Organique - Eau
- Demande Biochimique en oxygène en 5 jours
- Taux de saturation en oxygène - Eau
- Phosphore total - Eau

✓ Régularisation

Nous utilisons la régression Lasso, qui applique une régularisation L1, pour identifier les caractéristiques ayant un véritable impact sur la prédiction de l2M2. Cette régularisation L1 permet de réduire à zéro les coefficients des variables les moins influentes, ce qui nous aide à déterminer quelles caractéristiques physicochimiques ont le plus d'impact sur les données hydrobiologiques.

```
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.005)
cv_scores = cross_val_score(lasso, X_train, y_train, cv=5, scoring='neg_mean_score')
mean_cv_score = -cv_scores.mean()

print(f"Mean CV score: {mean_cv_score}")
```

➡ Mean CV score: 0.03731374537661418

```
lasso.fit(X_train, y_train)
lasso_coef = lasso.coef_
lasso_coef_df = pd.DataFrame(lasso_coef, index=features, columns=['Coefficient'])
lasso_coef_df = lasso_coef_df.sort_values('Coefficient', ascending=False)
print(lasso_coef_df)
```

	Coefficient
année	0.000000
Oxygène dissous - Eau	0.000000
station_encoded	-0.000000
trimestre	0.000000
Turbidité Formazine Néphélométrique - Eau	-0.000000
Température de l'Eau - Eau	-0.000000
Taux de saturation en oxygène - Eau	0.000000
Potentiel en Hydrogène (pH) - Eau	0.000000
Phosphore total - Eau	-0.000000
Orthophosphates (P04) - Eau	-0.000000
Ammonium - Eau	-0.000000
Nitrites - Eau	-0.000000
Matières en suspension - Eau	-0.000000
Demande Biochimique en oxygène en 5 jours (D.B....	-0.000000
Azote Kjeldahl - Eau	-0.000000
hydroecoregion_encoded	0.000000
Nitrates - Eau	-0.001346
Carbone Organique - Eau	-0.092639
Conductivité à 25°C - Eau	-0.528447

```
print(f"Nombre de features gardées : {sum(lasso_coef != 0)}")
```

➞ Nombre de features gardées : 3

```
print("Résultats : les caractéristiques les plus importantes selon le modèle Las  
df_lasso_coef = lasso_coef_df[df_lasso_coef['Coefficient'] != 0]  
df_lasso_coef_sorted = df_lasso_coef.abs().sort_values('Coefficient', ascending  
print(df_lasso_coef_sorted)
```

➞ Résultats : les caractéristiques les plus importantes selon le modèle Lasso

	Coefficient
Conductivité à 25°C – Eau	0.528447
Carbone Organique – Eau	0.092639
Nitrates – Eau	0.001346

On retrouve des paramètres précédemment mentionnés.

✓ Conclusion

Notre travail a exploré la relation entre les données physicochimiques et hydrobiologiques dans le but de mieux comprendre les interactions entre ces paramètres et d'identifier des modèles spatiaux significatifs, tels que les hydroécorégions.

L'objectif principal était de déterminer si ces régions écologiques pouvaient être détectées à partir des données physicochimiques et hydrobiologiques, et comment les différents paramètres interagissent pour influencer l'état des écosystèmes aquatiques.

Nous avons choisi de prendre en compte différents laps de temps, à savoir 1 mois et 6 mois, afin d'étudier les relations de causalité entre les paramètres physicochimiques et les indicateurs hydrobiologiques. Cette approche nous a permis d'analyser les corrélations entre les variables et de nous interroger sur l'influence persistante de certains paramètres physicochimiques, malgré les variations saisonnières, sur l'état hydrobiologique. Ainsi, nous avons exploré dans quelle mesure la temporalité impacte ces interactions et si les paramètres physicochimiques peuvent prédire ou expliquer les dynamiques des écosystèmes aquatiques sur différentes périodes.

L'application du clustering K-means a introduit une nouvelle dimension dans notre étude : celle de l'identification des hydroécorégions à partir des données. Ce processus nous a permis de tester si des regroupements naturels des stations, basés sur leurs caractéristiques physicochimiques et hydrobiologiques, étaient observables. Nous avons également comparé l'impact des deux laps de temps (1 mois vs 6 mois) pour déterminer lequel offrait de meilleurs résultats dans la détection des hydroécorégions, en fonction de la dynamique des paramètres et de l'évolution des conditions environnementales.

Après avoir identifié le laps de temps optimal, nous avons réintégré les hydroécorégions dans notre dataset, puis entraîné un modèle de régression pour prédire I2M2. Cela nous a permis de déterminer les trois paramètres les plus influents pour la prédiction de I2M2. Enfin, nous avons appliqué une approche de régularisation (Lasso) pour identifier les caractéristiques physicochimiques ayant le plus de poids dans la prédiction de I2M2, nous permettant ainsi de mieux comprendre quels paramètres sont véritablement significatifs et présentent un lien concret avec les dynamiques écologiques sur le terrain.

Notre travail s'est structuré autour de plusieurs étapes clés :

- Préparation et analyse des données physicochimiques et hydrobiologiques
- Fusion des différents jeux de données avec les traitements préliminaires nécessaires
- Préparation des données pour le clustering des stations : choix du nombre de clusters, encodage des variables catégorielles, et normalisation
- Réalisation du clustering des stations en utilisant la méthode K-Means pour différents lags temporels
- Analyse des résultats obtenus pour chaque lag et comparaison des performances pour déterminer le laps de temps le plus cohérent
- Visualisation des résultats
- Réalisation d'une régression pour prédire I2M2 à partir des données, avec l'intégration des hydroécorégions, et application d'une régularisation afin d'identifier les variables physicochimiques ayant un impact réel sur la prédiction de I2M2

En résumé :

- Nous avons pu identifier les paramètres physicochimiques et hydrobiologiques qui permettent de différencier les différentes régions au cours du temps
- Nous avons constaté qu'un lag de 6 mois est plus pertinent qu'un lag de 1 mois, ce qui suggère que l'impact des données physicochimiques sur l'état hydrobiologique est plus prononcé à long terme
- Nous avons déterminé les paramètres physicochimiques les plus influents dans l'explication des dynamiques hydrobiologiques.

Résultats

Les résultats du clustering des stations ont montré que, bien que prometteurs, seulement un tiers des stations ont été correctement classées dans leurs hydroécorégions respectives. Cela suggère que certaines hydroécorégions partagent des caractéristiques physicochimiques et hydrobiologiques trop similaires, rendant leur distinction difficile. En effet, les résultats obtenus avec les deux approches de clustering (1 mois et 6 mois) étaient relativement proches. Toutefois, l'approche avec un lag de 6 mois a permis de classer légèrement plus de stations correctement, avec une amélioration de 3 % par rapport à celle avec un lag de 1 mois. De plus, l'indice de silhouette a révélé que, pour un lag de 6 mois, 22 clusters ont été détectés contre 26 pour un lag de 1 mois.

En ce qui concerne les paramètres influençant le clustering des hydroécorégions, nous avons

observé que, quel que soit le lag (1 mois ou 6 mois), les paramètres les plus discriminants sont :

- I2M2
- Nitrates
- Conductivité à 25°C
- Carbone organique
- Température de l'eau

La majorité de ces paramètres se sont distingués dans les centres des clusters, ce qui indique qu'ils sont les principaux facteurs influençant la formation des clusters. En particulier, I2M2, les nitrates, le carbone organique et la conductivité de l'eau apparaissent comme les indicateurs les plus significatifs pour différencier les hydroécorégions.

Concernant les résultats de la régression, les trois paramètres les plus importants pour la caractérisation de l'indice I2M2 sont la conductivité à 25°C, les nitrites et le carbone organique dans l'eau. Les résultats du modèle Lasso confirment cette tendance, avec la conductivité à 25°C, le carbone organique et les nitrates apparaissant comme les facteurs les plus significatifs.

Nous avons réutilisé le dataset avec un décalage temporel de 6 mois car nous avons observé que cette approche semble plus pertinente pour capturer la différence de temps entre les changements dans l'eau et leur impact sur l'hydrobiologie, comme expliqué précédemment.

Il est important de noter que les coefficients négatifs de la conductivité, des nitrites et du carbone organique dans la régression indiquent que ces paramètres sont associés à une diminution de l'indice I2M2. En d'autres termes, à mesure que ces paramètres augmentent, l'indice I2M2 a tendance à diminuer, ce qui pourrait suggérer que des concentrations plus élevées de ces éléments ont un impact négatif sur l'état hydrobiologique de l'eau.

Les matrices de corrélation ont également révélé des corrélations négatives entre I2M2, la conductivité à 25°C et les nitrates, renforçant ainsi l'idée que ces paramètres jouent un rôle central dans l'état de l'hydrobiologie de l'eau.

NB : À noter que nous avons également testé l'approche sans décalage temporel (comme détaillé dans ce notebook), mais les résultats se sont avérés clairement moins bons que ceux obtenus avec un décalage de 1 mois.

Limitations et biais

Pour le clustering des hydroécorégions :

- Surreprésentation des données hydrobiologiques en été et sous-représentation en dehors de cette période.
- Les données physicochimiques sont régulières, mais l'hydrobiologie est biaisée par la saisonnalité.
- Lors de la jointure des données, perte de 50 % des stations, réduisant ainsi la taille de l'échantillon en raison du faible nombre de relevés.
- Déséquilibre dans le nombre de stations par région, ce qui introduit des biais dans l'analyse.
- Suppression de certaines combinaisons de paramètres physicochimiques en raison de leur faible représentation.
- Beaucoup d'imputation des valeurs manquantes par la médiane, ce qui peut affecter la précision des résultats.
- Suppression des outliers via la médiane, en raison du nombre limité de données, pour éviter l'impact de valeurs extrêmes. La médiane a été choisie car elle est moins sensible aux valeurs aberrantes et reflète mieux la tendance centrale dans des ensembles de données hétérogènes.

Pour la régression :

- Une méthode de clustering améliorée, permettant de mieux ajuster le lag temporel entre les données physicochimiques et hydrobiologiques, pourrait potentiellement conduire à des résultats plus représentatifs. En effet, on aurait pu extraire un autre dataset (avec le lag de temps "optimal").

Cependant :

- Nous avons réussi à prédire l'I2M2 avec une erreur moyenne raisonnable en régression.
- Les mêmes paramètres importants apparaissent régulièrement entre la régression, la régularisation et les corrélations, ce qui renforce la robustesse des résultats obtenus, et suggère que l'analyse est globalement fiable.

Pistes

Pour le clustering des hydroécorégions :

- Tester d'autres lags de temps pour mieux capter les variations saisonnières (1 an serait

très intéressant car on pourrait capturer les impacts saisonniers tout en tenant compte du décalage temporel entre les données physicochimiques et hydrobiologiques).

- Explorer différentes méthodes d'agrégation des valeurs, comme l'utilisation de la moyenne.
- Essayer d'autres agrégations temporelles (par exemple, par mois ou par année).

Une étude plus approfondie, incluant une répartition plus homogène des données et une exploration d'autres périodes temporelles, pourrait affiner la détection des hydroécorégions et améliorer la compréhension des facteurs physicochimiques et hydrobiologiques qui influencent les écosystèmes aquatiques.

Concernant la méthode de clustering, des approches alternatives comme le K-Means avec Dynamic Time Warping (DTW) ou l'utilisation de K-Medoids pourraient être envisagées. Ces techniques pourraient permettre une meilleure gestion des séries temporelles et une meilleure distinction des groupes d'hydroécorégions, en tenant compte de la nature dynamique des données.

Ces différentes perspectives ouvrent la voie à une meilleure compréhension des interactions complexes entre les facteurs physicochimiques et hydrobiologiques, et à une amélioration des méthodes d'analyse utilisées pour détecter et caractériser les hydroécorégions.