

# Documentation: Pipeline ELK sur les taux de pollution

CORNEJO GUILLEN Oscar, OLIVEIRA PEREIRA Diogo

20 février 2025

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Partie 1 : Collecte des Données via une API Publique</b>	<b>2</b>
2.1	Choix de l'API . . . . .	2
2.2	Extraction et Transmission des Données . . . . .	2
<b>3</b>	<b>Partie 2 : Transmission des Données avec Kafka</b>	<b>3</b>
3.1	Création du Topic Kafka . . . . .	3
3.2	Configuration du Producteur Kafka . . . . .	4
3.3	Configuration du Consommateur Kafka . . . . .	4
<b>4</b>	<b>Partie 3 : Transformation et Indexation des Données</b>	<b>5</b>
4.1	Utilisation de Logstash . . . . .	5
4.1.1	Configuration de Logstash . . . . .	5
4.2	Mapping Elasticsearch . . . . .	7
<b>5</b>	<b>Partie 4 : Requêtes</b>	<b>7</b>
5.1	Requête textuelle . . . . .	7
5.2	Requêtes d'agrégation . . . . .	8
5.3	Requête N-gram . . . . .	9
5.4	Requête floue (fuzzy) . . . . .	9
5.5	Série temporelle . . . . .	11
<b>6</b>	<b>Partie 5 : Analyse et Visualisation avec Kibana</b>	<b>12</b>
6.1	Application de la requête textuelle . . . . .	12
6.2	Application des requêtes d'agrégation . . . . .	13
6.3	Application de la requête N-gram . . . . .	13
6.4	Application de la requête floue (fuzzy) . . . . .	14
6.5	Application de la série temporelle . . . . .	15
<b>7</b>	<b>Partie 6 : Traitement des Données</b>	<b>16</b>
7.1	Spark . . . . .	16

# 1 Introduction

## 2 Partie 1 : Collecte des Données via une API Publique

### 2.1 Choix de l'API

L'étude de la qualité de l'air est essentielle pour préserver la santé publique et protéger l'environnement. La pollution de l'air, causée par les émissions industrielles, les transports et d'autres activités humaines, peut entraîner des maladies respiratoires, cardiovasculaires et même des effets neurologiques à long terme. En surveillant et en analysant les niveaux de polluants, il est possible de mettre en place des politiques efficaces pour réduire leur impact et améliorer la qualité de vie. De plus, une meilleure compréhension des dynamiques atmosphériques permet d'anticiper les épisodes de pollution et de sensibiliser la population aux bonnes pratiques pour limiter leur exposition aux substances nocives.

L'API sélectionnée pour la collecte des données est <https://www.geodair.fr/>, qui fournit des mesures de qualité de l'air en temps quasi réel. Cette API permet d'obtenir des données sur différents polluants ainsi que leur concentration. L'API ainsi que sa documentation sont disponibles au lien : <https://www.geodair.fr/donnees/api>

### 2.2 Extraction et Transmission des Données

En référence à l'API, la première étape consiste à demander une API-Key pour l'authentification dans l'en-tête des requêtes effectuées.

En parlant de **rafraîchissement des données** dans l'API, comme décrit ces données sont mises à jour presque en temps réel, cependant un utilisateur n'a accès qu'à 24 requêtes par jour.

En revanche, l'API se limite uniquement à collecter séparément un polluant de l'air pour une journée donnée, pour laquelle il faut passer par deux étapes :

- **Appel à un endpoint (/MoyJ/export)** .- Pour générer une ressource (dans ce cas le fichier csv) prenant en arguments l'ID du polluant et la date d'acquisition des données.
- **Appel à un endpoint (/download)** .- Pour télécharger les données au format csv en prenant en entrée l'ID de la ressource obtenu dans la requête précédente.

Ainsi, ayant 13 polluants dans l'air relevés par <https://www.geodair.fr/> (Voir l'Annexe 1), nous sommes limités à ne faire l'appel qu'une fois par jour à l'API pour la collecte des données, donc une requête est effectuée quotidiennement pour récupérer les données du jour précédent. Les données sont téléchargées sous forme de fichiers CSV, mais uniquement pour le mois de février. Ces fichiers sont ensuite traités par morceaux (chunks) pour une gestion efficace des données. Une fois transformées, les données sont envoyées au pipeline Kafka via un producteur Kafka.

De cette façon, nous pouvons montrer, par exemple, ces données extraites :

Date de début	Date de fin	Organisme	code zas	Zas	code site	nom site	type d'implan	Polluant	type d'influer	discriminant	
#####	#####	ATMO GRAND	FR44ZAG02	ZAG METZ	FR01011	Metz-Centre	Urbaine	NOX as NO2	Fond	1	
#####	#####	ATMO GRAND	FR44ZAG02	ZAG METZ	FR01012	Metz-Borny	Urbaine	NOX as NO2	Fond	1	
#####	#####	ATMO GRAND	FR44ZRE01	ZR GRAND-ES	FR01016	Atton	Rurale près d	NOX as NO2	Industrielle	1	
#####	#####	ATMO GRAND	FR44ZAG02	ZAG METZ	FR01018	Scy-Chazelles	Périurbaine	NOX as NO2	Fond	1	
#####	#####	ATMO GRAND	FR44ZAG02	ZAG METZ	FR01020	Thionville-Ce	Urbaine	NOX as NO2	Fond	1	
Réglementaire	type d'évalua	procédure de	type de valeu	valeur	valeur brute	unité de mesu	taux de saisie	couverture te	couverture de	code qualité	validité
Oui	mesures fixes	Auto	NO2_NC	Moyenne jou	20	20.3958333	µg-m3	100	100	100 R	1
Oui	mesures fixes	Auto	NO2_NC	Moyenne jou	35	35.2833333	µg-m3	100	100	100 R	1
Oui	mesures fixes	Auto	NO2_NC	Moyenne jou	19	19.1791667	µg-m3	100	100	100 A	1
Oui	mesures fixes	Auto	NO2_NC	Moyenne jou	21	21.325	µg-m3	100	100	100 A	1
Oui	mesures fixes	Auto	NO2_NC	Moyenne jou	26	25.5583333	µg-m3	100	100	100 R	1

FIGURE 1 – Exemple de données acquises depuis l'API de <https://www.geodair.fr/>

Ces données ont le schéma suivant :

```

1  {
2    "title": "schema_name",
3    "type": "array",
4    "items": {
5      "type": "object",
6      "properties": {
7        "Date de debut": {
8          "type": "string",
9          "format": "date-time"
10       },
11       "Date de fin": {
12         "type": "string",
13         "format": "date-time"
14       },
15       "Organisme": {
16         "type": "string"
17       },
18       "code zas": {
19         "type": "string"
20       },
21       "Zas": {
22         "type": "string"
23       },
24       "code site": {
25         "type": "string"
26       },
27       "nom site": {
28         "type": "string"
29       },
30       "type d'implantation": {
31         "type": "string"
32       },
33       "Polluant": {
34         "type": "string"
35       },
36       "type d'influence": {
37         "type": "string"
38       },
39       "discriminant": {
40         "type": "string"
41       },
42     },
43     "Reglementaire": {
44       "type": "string"
45     },
46     "type d'evaluation": {
47       "type": "string"
48     },
49     "procedure de mesure": {
50       "type": "string"
51     },
52     "type de valeur": {
53       "type": "string"
54     },
55     "valeur": {
56       "type": "number"
57     },
58     "valeur brute": {
59       "type": "number"
60     },
61     "unite de mesure": {
62       "type": "string"
63     },
64     "taux de saisie": {
65       "type": "number"
66     },
67     "couverture temporelle": {
68       "type": "number"
69     },
70     "couverture de donnees": {
71       "type": "number"
72     },
73     "code qualite": {
74       "type": "string"
75     },
76     "validite": {
77       "type": "number"
78     }
79   },
80 }

```

## 3 Partie 2 : Transmission des Données avec Kafka

### 3.1 Création du Topic Kafka

Pour assurer la transmission des données collectées vers les consommateurs, un topic Kafka nommé *air-quality* est créé. La commande suivante est utilisée pour créer le *topic* :

```

1  docker exec -it kafka kafka-topics --create --topic air-quality --partitions 1
2    --replication-factor 1 --bootstrap-server localhost:9092

```

On peut vérifier la création du *topic* en exécutant la commande :

```

1  docker exec -it kafka kafka-topics --list --bootstrap-server localhost:9092

```

Où on montre la présence du *topic* *air-quality* qu'on vient de créer :

```
C:\Users\OSCAR>docker exec -it kafka kafka-topics --list --bootstrap-server localhost:9092
__consumer_offsets
air-quality
kafka-logs

What's next:
Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug kafka
Learn more at https://docs.docker.com/go/debug-cli/
```

FIGURE 2 – Vérification de la création du *topic* Kafka

## 3.2 Configuration du Producteur Kafka

Le producteur Kafka est configuré pour extraire les données depuis l'API `geodair.fr` et les envoyer au topic Kafka `air-quality`, où :

- Les données sont récupérées au format CSV.
- Ils passent par une fonction pour les transformer de CSV en JSON.
- Après on applique une fonction de normalisation qui est particulièrement utile pour éliminer les caractères spéciaux français dans un JSON. Elle transforme les caractères accentués et les ligatures en leurs équivalents sans accent. Par exemple, "é" devient "e", "ç" devient "c", et "œ" devient "oe". Cette normalisation facilite le traitement des données françaises dans des systèmes qui ne gèrent pas bien les caractères spéciaux, tout en conservant la lisibilité du texte.
- Enfin les données sont transmises

Le producteur utilise la bibliothèque `kafka-python` pour établir la connexion et envoyer les messages. Voici un extrait de code illustrant cette configuration :

```
1 # Kafka Configuration
2 KAFKA_BROKER = "localhost:9092"
3
4 # Initialize Kafka Producer
5 producer = KafkaProducer(
6     bootstrap_servers=KAFKA_BROKER,
7     value_serializer=lambda v: json.dumps(v).encode("utf-8"),
8     max_request_size=10485760
9 )
```

Comme on regarde, le `kafka broker` (qui est un nœud ou un serveur individuel au sein d'un cluster Kafka) est un arguments de configuration. En plus `value_serializer` convertit les valeurs des messages en JSON puis en UTF-8 octets avant l'envoi, et `max_request_size` définit la taille maximale de la requête à 10 Mo (10485760 octets), permettant l'envoi de messages supérieurs à la limite par défaut de 1 Mo.

Finalement, les données sont envoyées en prenant le `KAFKA TOPIC` comme argument avec la ligne de code suivante :

```
1 producer.send(KAFKA_TOPIC, all_data)
```

## 3.3 Configuration du Consommateur Kafka

Le consommateur Kafka se connecte au topic `air-quality` pour recevoir les messages JSON. Les données reçues sont désérialisées et stockées dans un fichier pour un traitement ultérieur. Le consommateur utilise également `kafka-python` pour gérer la réception des messages. Voici un extrait de code de configuration pour le consommateur :

```
1 KAFKA_TOPIC = "air-quality"
2 KAFKA_BROKER = "localhost:9092"
3
4 consumer = KafkaConsumer(
5     KAFKA_TOPIC,
6     bootstrap_servers=KAFKA_BROKER,
7     auto_offset_reset="earliest",
```

```

8     #enable_auto_commit=False,
9     value_deserializer=lambda x: json.loads(x.decode("utf-8")),
10    max_partition_fetch_bytes=10485760
11 )

```

Où cette configuration permet au consommateur de lire les messages JSON depuis le début de la rubrique, de les désérialiser automatiquement et de gérer des messages jusqu'à 10 Mo par partition. Ces données peuvent être traitées ou enregistrées dans un fichier, dans ce cas nous l'avons fait dans un fichier JSON

Dans ces configurations, le producteur et le consommateur sont conçus pour fonctionner de manière continue, assurant ainsi une transmission fluide des données entre l'API et le système de stockage.

## 4 Partie 3 : Transformation et Indexation des Données

### 4.1 Utilisation de Logstash

Logstash est un outil puissant pour la collecte, la transformation et l'envoi de données. Dans ce contexte, il est utilisé pour lire les données du topic Kafka `air-quality`, les transformer, et les indexer dans Elasticsearch. Les principales fonctionnalités incluent :

- **Lecture des données** depuis Kafka.
- **Transformation des données** : conversion des formats de date, renommage des champs, typage des valeurs numériques.
- **Ajout d'indicateurs de qualité** basés sur le code de qualité.
- **Indexation des données** dans Elasticsearch sous l'index `air_quality_pol`.

#### 4.1.1 Configuration de Logstash

La configuration de Logstash est définie dans un fichier de configuration. Voici le contenu du fichier `/etc/logstash/conf.d/logstash_kafka.conf` :

```

1  input {
2    kafka {
3      bootstrap_servers => "localhost:9092"
4      topics => ["air-quality"]
5      codec => "json"
6      client_id => "logstash-air-quality"
7      group_id => "logstash-air-quality-group"
8      auto_offset_reset => "latest"
9      consumer_threads => 3
10   }
11 }
12
13 filter {
14   date {
15     match => ["Date de debut",
16              "yyyy/MM/dd HH:mm:ss"]
17     target => "timestamp_debut"
18   }
19   date {
20     match => ["Date de fin",
21              "yyyy/MM/dd HH:mm:ss"]
22     target => "timestamp_fin"
23   }
24   mutate {
25     convert => {
26       "valeur" => "float"
27       "valeur brute" => "float"
28       "taux de saisie" => "float"
29       "couverture temporelle" => "float"
30       "couverture de donnees" => "float"
31       "validite" => "integer"
32     }
33   }
34 }
35
36 mutate {
37   remove_field => ["event", "filename"]
38 }
39
40 if [code_qualite] == "A" {
41   mutate {
42     add_field => { "quality_level" => "high" }
43   }
44 } else if [code_qualite] == "R" {
45   mutate {
46     add_field => { "quality_level" => "medium" }
47   }
48 } else {
49   mutate {
50     add_field => { "quality_level" => "low" }
51   }
52 }
53
54 output {
55   elasticsearch {
56     hosts => ["http://localhost:9200"]
57     index => "air_quality"
58     document_id => "%{@timestamp}"
59   }
60   stdout { codec => rubydebug }
61 }

```

Logstash traite les données en trois étapes : l'entrée (**Input**), le filtrage (**Filter**), et la sortie (**Output**). Dans la configuration Kafka, plusieurs paramètres optimisent la consommation des données. Le `client_id` et le `group_id` identifient de manière unique le client Logstash et son groupe de consommateurs, essentiel pour gérer plusieurs instances et assurer un suivi précis. Le paramètre `auto_offset_reset` est défini sur `latest` pour que Logstash lise les messages depuis le début du topic en l'absence d'un offset précédent, évitant toute perte de données. Enfin, `consumer_threads` est configuré à 3 pour paralléliser la consommation, améliorant les performances en répartissant la charge sur plusieurs threads.

Premièrement, on lit les données JSON depuis Kafka. La configuration spécifie le serveur Kafka, le topic (`air-quality`), et d'autres paramètres tels que l'identifiant du client et le groupe de consommateurs.

L'étape du **Filter** effectue plusieurs transformations sur les données. D'abord, les dates sont converties dans un format compatible avec Elasticsearch. Ensuite, les champs numériques sont typés (par exemple, conversion en 'float' ou 'integer'). Les noms des champs sont renommés pour une meilleure compatibilité avec Elasticsearch, en utilisant des noms standardisés. Enfin, un indicateur de qualité (`quality_level`) est ajouté en fonction du code de qualité (`code_qualite`), qui peut être `high`, `medium`, ou `low`.

Au final, les données transformées sont envoyées vers Elasticsearch. La configuration spécifie l'hôte Elasticsearch, l'index (`air_quality`), et un identifiant unique pour chaque document. Un débogage est également activé pour afficher les données traitées dans la console, ce qui est utile pour le développement et les tests.

## 4.2 Mapping Elasticsearch

Le mapping Elasticsearch définit la structure des données indexées, y compris les types de champs, les analyzers et les filtres pour optimiser les recherches.

Notre mapping permet de prendre en charge une variété de formats de date, pour maintenir une cohérence temporelle entre les documents ; étape cruciale pour garantir que les informations restent exploitables, quelle que soit leur origine.

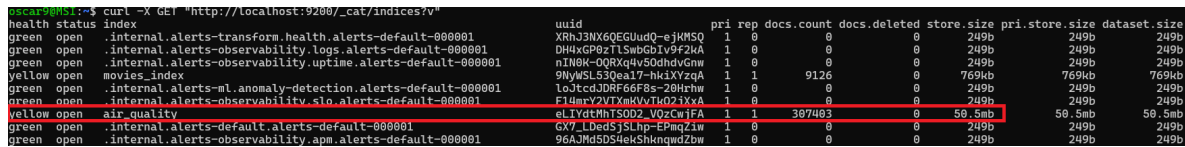
Pour améliorer l'expérience utilisateur, certains champs clés, comme les noms et les organismes, sont enrichis avec un analyseur n-gram. Cela facilite les recherches partielles et l'auto-complétion, rendant l'exploration des données plus intuitive. De plus, on a également ajouté un champ supplémentaire de type **keyword**, aux attributs les plus important, pour des raisons de performance.

Enfin, des indicateurs de qualité et de couverture des données sont intégrés pour évaluer la fiabilité des informations stockées. Ces métriques aident à identifier les éventuelles lacunes ou incohérences dans les données, assurant ainsi une base solide pour l'analyse. (Voir annexe 7.1)

**Vérification de l'indexation :** Pour vérifier que les données ont été correctement indexées, exécutez la commande suivante :

```
1 curl -X GET "http://localhost:9200/_cat/indices?v"
```

Cette commande affiche la liste des indices Elasticsearch, y compris **air\_quality**, si l'indexation a réussi :



health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size	dataset.size
green	open	.internal.alerts-transform.health.alerts-default-000001	XRh33NX6QEGUudQ-ejKMSQ	1	0	0	0	249b	249b	249b
green	open	.internal.alerts-observability.logs.alerts-default-000001	DH4xGP0zTlSmbGb1v9F2kA	1	0	0	0	249b	249b	249b
green	open	.internal.alerts-observability.uptime.alerts-default-000001	nIN8K-DQRXq4v50dhvGmw	1	0	0	0	249b	249b	249b
yellow	open	movies_index	9MyMSLS3Qea17z-mh3IXz9A	1	1	9126	0	769kb	769kb	769kb
green	open	.internal.alerts-ml.anomaly-detection.alerts-default-000001	lo3tcdJDRf66F8s-28Hrhw	1	0	0	0	249b	249b	249b
green	open	.internal.alerts-observability.slo.alerts-default-000001	F14mrY2VTXmkVvTh02iXxA	1	0	0	0	249b	249b	249b
yellow	open	air_quality	eLIVdtMhTSOD2_VQzCwJfA	1	1	307403	0	50.5mb	50.5mb	50.5mb
green	open	.internal.alerts-default.alerts-default-000001	Gx7_LDedsjSLhp-EPmqZ1w	1	0	0	0	249b	249b	249b
green	open	.internal.alerts-observability.apm.alerts-default-000001	96A3MdSD54ekShkmgqdZw	1	0	0	0	249b	249b	249b

FIGURE 3 – Vérification de la création de l'indice **air\_quality**

## 5 Partie 4 : Requêtes

Dans cette section, nous présentons plusieurs types de requêtes pour explorer les données de qualité de l'air indexées dans Elasticsearch. Ces requêtes permettent de rechercher des sites spécifiques, d'analyser les tendances de pollution, d'explorer les données avec des recherches approximatives, et d'obtenir des statistiques agrégées.

### 5.1 Requête textuelle

La requête effectue une recherche dans l'index **air\_quality**. Plus précisément, elle cherche les documents dans lesquels le champ **Organisme** contient le terme "Atmo". Nous avons également voulu voir combien (**\_count**) de nom de site vérifie le query (2).

```

1 {
2   "query": {
3     "match": {
4       "Organisme": "Atmo"
5     }
6   }
7 }

```

## Résultat

```

1 [ (1)
2   { [...] "Organisme": "ATMO NORMANDIE" [...] },
3   { [...] "Organisme": "ATMO SUD" [...] },
4   { [...] "Organisme": "ATMO GRAND EST" [...] },
5   { [...] "Organisme": "ATMO HAUTS DE FRANCE" [...] },
6   { [...] "Organisme": "ATMO AUVERGNE-RHÔNE-ALPES" [...] },
7   { [...] "Organisme": "ATMO OCCITANIE" [...] },
8   { [...] "Organisme": "ATMO NOUVELLE-AQUITAINE" [...] },
9   { [...] "Organisme": "ATMO BOURGOGNE-FRANCHE-COMTE" [...] },
10  { [...] "Organisme": "ATMO REUNION" [...] },
11  { [...] "Organisme": "ATMO GUYANE" [...] }
12 ]
13 { (2)
14   "count" : 220955,
15   [...]
16 }

```

Dans ce cas, 10 types d'organismes ayant ATMO en leur nom ont été obtenus, soit 220 955 documents sur un total.

## 5.2 Requêtes d'agrégation

Les différentes requêtes d'agrégation suivantes permettent, d'une part, de compter le nombre d'instance pour lequel on a un niveau de qualité **high**, **medium** ou **low**. D'autre part, on cherche la distribution (en pourcentage) des polluants sur l'ensemble des données. Une représentation graphique de ces résultats, réalisée avec Kibana, est fournie à la section 6.

```

1 {
2   "size": 0,
3   "aggs": {
4     "quality_counts": {
5       "terms": {
6         "field": "quality_level",
7         "size": 3
8       }
9     }
10  }
11 }

```

## Résultat

```

1 "aggregations" : {
2   "quality_counts" : {
3     "doc_count_error_upper_bound" : 0,
4     "sum_other_doc_count" : 0,
5     "buckets" : [
6       {
7         "key" : "high",
8         "doc_count" : 323349
9       },
10      {
11        "key" : "medium",
12        "doc_count" : 22114
13      },
14      {
15        "key" : "low",
16        "doc_count" : 9689
17      }
18    ]
19  }
20 }

```



```

1 {
2   "size": 0,
3   "aggs": {
4     "pollutant_distribution": {
5       "terms": {
6         "field": "Polluant",
7         "size": 10
8       },
9       "aggs": {
10        "percentage": {
11          "bucket_script": {
12            "buckets_path": {
13              "count": "_count"
14            },
15            "script": "params.count *
16                    100 / 307403"
17          }
18        }
19      }
20    }
21  }
22 }

```

## Résultat

```

1   "aggregations" : {
2     "pollutant_distribution" : {
3       "doc_count_error_upper_bound" : 0,
4       "sum_other_doc_count" : 0,
5       "buckets" : [
6         {
7           "key" : "NO2",
8           "doc_count" : 62526,
9           "percentage" : {
10            "value" : 20.34007475528866
11          }
12        },
13        {
14          "key" : "NO",
15          "doc_count" : 62172,
16          "percentage" : {
17            "value" : 20.224916477718175
18          }
19        }
20      ]
21    }
22  }

```

## 5.3 Requête N-gram

Cette requête permet une recherche partielle sur les noms de sites en utilisant l'analyseur N-gram considérant que dans notre cartographie nous définissons `min_gram` comme 3 et `max_gram` comme 4.

```

1 {
2   "query": {
3     "match": {
4       "nom site.ngram": "SAI"
5     }
6   }
7 }

```

## Résultat

```

1 [
2   { [...] "nom site": "SAINT EXUPERY" [...] },
3   { [...] "nom site": "SAINT LAURENT" [...] },
4   { [...] "nom site": "SAINT DENIS" [...] },
5   { [...] "nom site": "SAINT GERMAIN" [...] },
6   { [...] "nom site": "SAINT MICHEL" [...] }, [...]
7 ]

```

## 5.4 Requête floue (fuzzy)

Cette requête permet une recherche approximative sur les noms de sites, utile pour corriger les fautes de frappe ou les erreurs de saisie. Nous avons également voulu voir combien (`_count`) de nom de site vérifie le query (2).

```

1 {
2   "query": {
3     "fuzzy": {
4       "nom site": {
5         "value": "Arbes",
6         "fuzziness": 2
7       }
8     }
9   }
10 }

```

## Résultat

```

1 [ (1)
2   { [...] "nom site": "AMBES" [...] },
3   { [...] "nom site": "Arles" [...] },
4   { [...] "nom site": "Montpellier Pres d'Arenes" [...] },
5   { [...] "nom site": "Tarbes lycee Dupuy" [...] },
6   { [...] "nom site": "Vesoul Pres Caillet" [...] }
7 ]
8 { (2)
9   "count" : 4956,
10  [...]
11 }

```

La requête suivante effectue une recherche floue sur `nom site`, tolérant jusqu'à deux erreurs (`fuzziness: 2`) pour retrouver des sites malgré des fautes de frappe. Une agrégation (`aggs`) sur `nom site.keyword` extrait jusqu'à 1000 noms uniques, triés alphabétiquement (`"order": {"_key": "asc"}`). L'option

"size": 0 empêche le retour des documents individuels, affichant uniquement les résultats agrégés. Idéale pour la correction d'erreurs et la saisie semi-automatique. Ainsi, comme on peut le constater, 5 types de `nom site` sont pris en compte.

```

1  "size": 0,
2  "query": {
3    "fuzzy": {
4      "nom site": {
5        "value": "Arbes",
6        "fuzziness": 2
7      }
8    }
9  },
10 "aggs": {
11   "unique_nom_site": {
12     "terms": {
13       "field": "nom site.keyword",
14       "size": 1000,
15       "order": { "_key": "asc" }
16     }
17   }
18 }

```

## Résultat

```

1  [...]
2    "aggregations" : {
3      "unique_nom_site" : {
4        "doc_count_error_upper_bound" : 0,
5        "sum_other_doc_count" : 0,
6        "buckets" : [
7          {
8            "key" : "AMBES",
9            "doc_count" : 708
10         },
11         {
12           "key" : "Arles",
13           "doc_count" : 1062
14         },
15         {
16           "key" : "Montpellier Pres d'Arenes",
17           "doc_count" : 1062
18         },
19         {
20           "key" : "Tarbes lycee Dupuy",
21           "doc_count" : 1062
22         },
23         {
24           "key" : "Vesoul Pres Caillet",
25           "doc_count" : 1062
26         }
27       ]
28     }
29 }

```

## 5.5 Série temporelle

Cette requête permet d'obtenir une agrégation des valeurs journalières d'un polluant mesuré sur un site spécifique. Dans ce cas, les mesures moyennes par jour du polluant *NO* (Monoxyde d'Oxygène) à Corcorde entre le 1er et le 20 février ont été extraites.

```
1  {
2    "query": {
3      "bool": {
4        "must": [
5          {
6            "term": {
7              "nom site.keyword": "Concorde"
8            }
9          },
10         {
11           "term": {
12             "Polluant": "NO"
13           }
14         },
15         {
16           "range": {
17             "Date de debut": {
18               "gte": "2025-02-01T00:00:00",
19               "lt": "2025-02-20T00:00:00"
20             }
21           }
22         }
23       ]
24     }
25   }
26 }
```

### Résultats

```
1  [...]
2  "aggregations" : {
3    "daily_values" : {
4      "buckets" : [
5        {
6          "key_as_string" :
7            "2025/02/04 00:00:00",
8          "key" : 1738627200000,
9          "doc_count" : 24,
10         "average_valeur" : {
11           "value" : 15.65833322176089
12         }
13       },
14       {
15         "key_as_string" :
16           "2025/02/05 00:00:00",
17         "key" : 1738713600000,
18         "doc_count" : 24,
19         "average_valeur" : {
20           "value" : 21.22499997324
21         }
22       },
23       {
24         "key_as_string" :
25           "2025/02/06 00:00:00",
26         "key" : 1738800000000,
27         "doc_count" : 24,
28         "average_valeur" : {
29           "value" : 16.64583330353101
30         }
31       }
32     ]
33   }
34 }
```

Cette requête agrège les valeurs moyennes d'un polluant mesuré sur **Concorde** entre le 1er et le 20 février 2025. Chaque jour (**key\_as\_string**) contient en général 24 relevés horaires (**doc\_count**).

Les niveaux varient : 15.66 le 4 février, un pic à 21.22 le 5 février, puis 16.64 le 6 février. Aucun relevé les 8 et 9 février (**average\_valeur**: null), suggérant une panne ou une absence de mesures. Les résultats jusqu'au 20 février suivent cette tendance et seront mieux visualisés dans Kibana.

L'analyse graphique, abordée dans la section suivante, permettra d'interpréter ces variations plus efficacement.

## 6 Partie 5 : Analyse et Visualisation avec Kibana

Les représentations graphiques ci-dessus illustrent les requêtes élaborées en amont sur Elasticsearch concernant la qualité de l'air.

### 6.1 Application de la requête textuelle

Ce qui a été fait en premier c'est d'appliquer un filtre pour **Organisme** comme **is** dans ce cas "**ATMO**" :

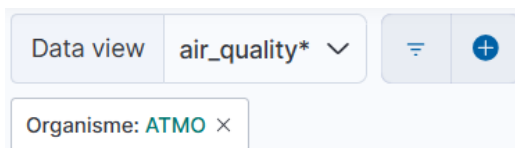


FIGURE 4 – Définition d'un filtre pour un **Organisme** comme **ATMO**.

Sur cette base, nous avons pu obtenir les graphiques suivants :

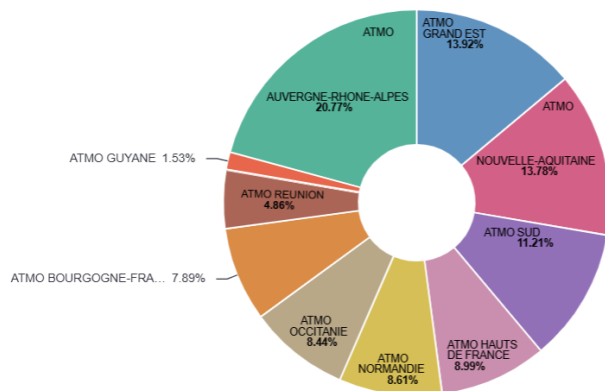


FIGURE 5 – Répartition en pourcentage des organismes détectés dans les documents.

Top 10 values of Organisme.keyword	Count of records
ATMO/AUVERGNE-RHONE-ALPES	45,854
ATMO GRAND EST	30,753
ATMO NOUVELLE-AQUITAINE	30,447
ATMO SUD	24,768
ATMO HAUTS DE FRANCE	19,866
ATMO NORMANDIE	19,029
ATMO OCCITANIE	18,645
ATMO BOURGOGNE-FRANCHE-COMTE	17,442
ATMO REUNION	10,739
ATMO GUYANE	3,372

FIGURE 6 – Tableau récapitulatif des organismes contenant **ATMO**, avec le nombre d'enregistrements associés à chaque organisme.

Dans ce cas, le premier 5 est un *pie chart* avec toutes les valeurs d'**Organisme** trouvées ainsi que leur pourcentage d'influence sur le total des documents détectés. Et le deuxième 6 est un tableau avec chaque valeur d'**Organisme** qui contient *ATMO* et le nombre d'enregistrements pour chacun. Enfin, nous affichons le nombre total d'enregistrements obtenus qui coïncide avec ce qui a été obtenu en elastic search.



FIGURE 7 – Nombre total d’enregistrements contenus dans *Organisme ATMO*.

### 6.2 Application des requêtes d’agrégation

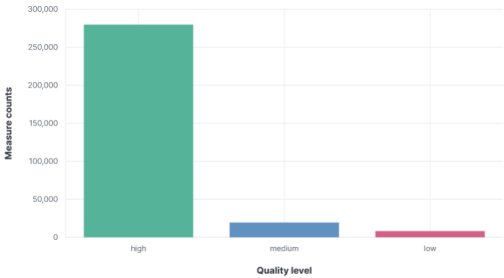


FIGURE 8 – Distribution des mesures selon le niveau de qualité (élevé, moyen, faible).

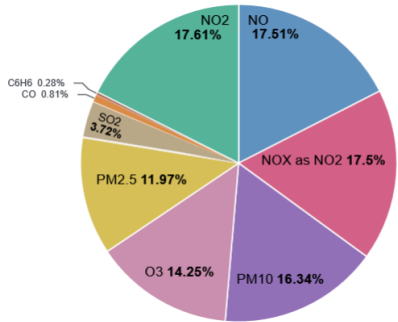


FIGURE 9 – Répartition des principaux polluants atmosphériques en pourcentage.

Le premier graphique, un diagramme à barres, illustre la distribution des mesures en fonction de la qualité de l’air. Il révèle une prédominance de mesures indiquant une qualité de l’air élevée, tandis que les niveaux moyen et faible sont beaucoup moins fréquents. Cela suggère que, globalement, la qualité de l’air est majoritairement bonne selon les données analysées.

Le deuxième graphique, un diagramme circulaire, présente la répartition des polluants atmosphériques. On y observe une concentration significative de **NO<sub>2</sub>** (17,61 %), **NO** (17,51 %) et **NO<sub>x</sub>** en tant que **NO<sub>2</sub>** (17,5 %), ainsi que des particules fines **PM<sub>10</sub>** (16,34 %) et **PM<sub>2,5</sub>** (11,97 %), qui sont des indicateurs clés de la pollution de l’air.

### 6.3 Application de la requête N-gram

On remarque que quelques-uns des noms de site ont bien été retrouvés dans le résultat des requêtes floues (fuzzy) et N-gram. Cela démontre l’efficacité de ces techniques pour capturer des variations et des erreurs potentielles dans les noms de sites.



FIGURE 10 – Tag cloud représentant l'importance des mesures par nom de site

Comme illustré dans la Figure 10, les noms de sites les plus fréquents ou les plus importants sont mis en avant dans le tag cloud. Cela permet de visualiser rapidement les sites où les mesures de pollution de l'air sont les plus importants.

#### 6.4 Application de la requête floue (fuzzy)

Ce qui a été fait en premier c'est d'appliquer un filtre pour `nom site` comme un *query DSL* à cause de limitations de Kibana pour ce cas :

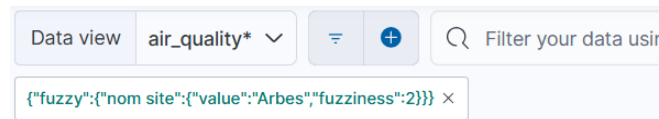


FIGURE 11 – Définition d'un filtre pour `nom site` comme un *query DSL*.

Sur cette base, nous avons pu obtenir les graphiques suivants :

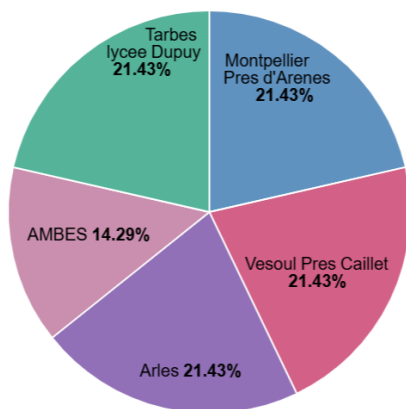


FIGURE 12 – Distribution des mesures selon le niveau de qualité (élevé, moyen, faible).



FIGURE 13 – Répartition des principaux polluants atmosphériques en pourcentage.

Dans ce cas, le premier 12 est un *pie chart* avec toutes les valeurs d'`nom site` trouvées ainsi que leur pourcentage d'influence sur le total des documents détectés. Et le deuxième 13 est un nuage de tags qui montre les `nom site` en fonction de leur influence.

Enfin, nous affichons le nombre total d'enregistrements obtenus qui coïncide avec ce qui a été obtenu en elastic search.



FIGURE 14 – Tag cloud représentant l'importance des mesures par nom de site

## 6.5 Application de la série temporelle

En référence aux séries temporelles, nous commençons par définir différents filtres, parmi lesquels la **Date de debut** (entre le 1er et le 20 février), le **nom site** comme *Concorde* et le **Polluant** comme *NO* (Monoxyde d'azote).

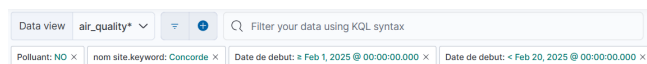


FIGURE 15 – Définition de plusieurs filtres pour la série temporelle.

Sur cette base, nous avons pu obtenir les graphiques suivants :

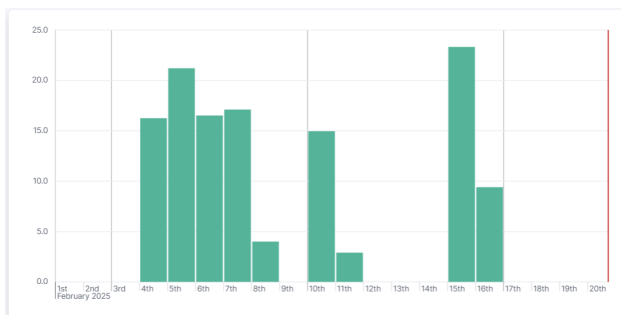


FIGURE 16 – Distribution des mesures selon le niveau de qualité (élevé, moyen, faible).



FIGURE 17 – Répartition des principaux polluants atmosphériques en pourcentage.

Dans ce cas, le premier 16 est un graphique à barres qui montre la moyenne de la **valeur** du **polluant** (*NO*) obtenue chaque jour entre le 1er et le 20 février. Et le deuxième 17 affiche la valeur moyenne obtenue sur toute cette plage de dates.

## 7 Partie 6 : Traitement des Données

### 7.1 Spark

L'objectif de cette section est de traiter les données issues d'Elasticsearch en utilisant **Apache Spark**. Spark permet de manipuler des volumes importants de données de manière efficace grâce à son moteur de calcul distribué.

Nous allons ici récupérer les données stockées dans Elasticsearch et les charger dans un **DataFrame** Spark. Ce processus est essentiel pour effectuer des transformations, des analyses avancées et des agrégations sur les données de la qualité de l'air.

#### Envoi des données sur Spark

Le code suivant initialise une session Spark, configure les paramètres pour se connecter à Elasticsearch et charge les données dans un **DataFrame** :

```
1  import org.apache.spark.sql.SparkSession
2
3  // Importation des implicites pour utiliser les fonctionnalités Spark SQL
4  import spark.implicits._
5
6  // Arrêt d'une session active si elle existe
7  SparkSession.getActiveSession.foreach(_.stop())
8
9  // Création d'une nouvelle session Spark avec connexion à Elasticsearch
10 val spark = SparkSession.builder()
11     .appName("Read from ES")
12     .config("spark.es.nodes", "localhost")
13     .config("spark.es.port", "9200")
14     .config("spark.es.mapping.date.rich", "false")
15     .config("spark.es.nodes.wan.only", "true")
16     .master("local[*]")
17     .getOrCreate()
18
19 // Chargement des données d'Elasticsearch dans un DataFrame Spark
20 val air_quality_poldf = spark.read
21     .format("org.elasticsearch.spark.sql")
22     .option("es.read.fields.as.array.include", "")
23     .option("es.mapping.date.rich", "false")
24     .load("air_quality")
25
26 // Affichage du schéma des données chargées
27 air_quality_poldf.printSchema()
```

Le code présenté effectue plusieurs étapes clés pour connecter Spark à Elasticsearch et analyser les données de qualité de l'air. Tout d'abord, il arrête toute session Spark active afin d'éviter les conflits potentiels. Ensuite, une nouvelle session Spark est initialisée avec une configuration spécifiquement adaptée à Elasticsearch. Les données de l'index `air_quality` sont ensuite chargées dans un **DataFrame** Spark, et le schéma des données est affiché pour validation. Cette connexion entre Spark et Elasticsearch permet d'exploiter pleinement la puissance de Spark SQL pour analyser et traiter les données de qualité de l'air. Grâce à cette intégration, il est possible d'effectuer des requêtes complexes, d'appliquer des transformations et de stocker les résultats pour des analyses ultérieures.

Une fois les données chargées, nous avons effectué plusieurs analyses distribuées :

#### Analyse par Type d'Implantation

La première analyse consiste à calculer les moyennes de pollution et le nombre de mesures pour chaque type d'implantation. Cette analyse permet d'identifier les zones les plus impactées par la pollution. Le code suivant illustre cette opération :



```

1 val avgByImplantation = air_quality_poldF
2   .groupBy("type d'implantation")
3   .agg(
4     avg("valeur").as("moyenne_pollution"),
5     count("*").as("nombre_mesures")
6   )
7   .orderBy(desc("moyenne_pollution"))

```

Les résultats de cette analyse sont présentés dans la section 7.1.

## Performances des Organismes

Une évaluation des performances des organismes a également été réalisée. Cette analyse calcule les niveaux moyens de pollution par zone administrative de surveillance (ZAS), ce qui permet d'identifier les zones les plus touchées par la pollution. Le code suivant montre comment ces statistiques sont calculées :

```

1 val avgPollutionByZAS = air_quality_poldF
2   .groupBy("Zas")
3   .agg(
4     avg("valeur").as("moyenne_pollution")
5   )
6   .orderBy(desc("moyenne_pollution"))

```

Les résultats de cette analyse sont présentés dans la section 7.1.

## Qualité des Données

La distribution des niveaux de qualité des mesures a été analysée pour évaluer la fiabilité globale du système de surveillance. Cette analyse permet de comprendre comment les données sont réparties en fonction de leur qualité. Le code suivant illustre cette opération :

```

1 val qualityDistribution = air_quality_poldF
2   .groupBy("quality_level")
3   .count()
4   .orderBy(desc("count"))

```

Les résultats de cette analyse sont présentés dans la section 7.1.

## Valeurs Extrêmes

Les pics de pollution les plus significatifs ont été identifiés et analysés. Cette analyse permet de comprendre les situations extrêmes et leurs caractéristiques. Le code suivant montre comment les polluants les plus fréquents sont identifiés :

```

1 val mostCommonPollutants = air_quality_poldF
2   .groupBy("Polluant")
3   .agg(
4     count("*").as("nombre_mesures")
5   )
6   .orderBy(desc("nombre_mesures"))

```

Les résultats de cette analyse sont présentés dans la section 7.1.

## Statistiques Globales

Enfin, des indicateurs statistiques fondamentaux ont été calculés sur l'ensemble du jeu de données. Ces indicateurs incluent la moyenne, l'écart-type, les valeurs minimales et maximales, ainsi que le nombre total de mesures. Le code suivant illustre cette opération :

```
1  val globalStats = air_quality_poiDF.agg(  
2    avg("valeur").as("moyenne_globale"),  
3    stddev("valeur").as("ecart_type"),  
4    min("valeur").as("min"),  
5    max("valeur").as("max"),  
6    count("*").as("total_mesures")  
7  )
```

Les résultats de cette analyse sont présentés dans la section 7.1.

Ces analyses permettent d'obtenir une vue d'ensemble des données de qualité de l'air, en mettant en évidence les tendances, les performances des organismes, la qualité des données, et les situations extrêmes. Les résultats de ces analyses peuvent être utilisés pour prendre des décisions éclairées et améliorer la surveillance de la qualité de l'air.

# Annexes

## Template mapping

```
1  {
2    "index_patterns": ["air_quality*"],
3    "settings": {
4      "analysis": {
5        "analyzer": {
6          "ngram_analyzer": {
7            "type": "custom",
8            "tokenizer": "ngram_tokenizer",
9            "filter": ["lowercase"]
10         }
11       },
12       "tokenizer": {
13         "ngram_tokenizer": {
14           "type": "ngram",
15           "min_gram": 3,
16           "max_gram": 4,
17           "token_chars": ["letter",
18                           "digit"]
19         }
20       }
21     },
22   },
23   "mappings": {
24     "properties": {
25       "@timestamp": {
26         "type": "date"
27       },
28       "Date de debut": {
29         "type": "date",
30         "format": "yyyy/MM/dd HH:mm:ss
31                  ||yyyy/MM/dd||
32                  epoch_millis||
33                  strict_date_optional_time"
34       },
35       "Date de fin": {
36         "type": "date",
37         "format": "yyyy/MM/dd HH:mm:ss
38                  ||yyyy/MM/dd||
39                  epoch_millis||
40                  strict_date_optional_time"
41       },
42       "Organisme": {
43         "type": "text",
44         "fields": {
45           "keyword": {
46             "type": "keyword",
47             "ignore_above": 256
48           },
49           "ngram": {
50             "type": "text",
51             "analyzer": "ngram_analyzer"
52           }
53         }
54       },
55       "Polluant": {
56         "type": "keyword"
57       },
58       "Reglementaire": {
59         "type": "keyword"
60       },
61       "Zas": {
62         "type": "text",
63         "fields": {
64           "keyword": {
65             "type": "keyword",
66             "ignore_above": 256
67           }
68         }
69       },
70       "code qualite": {
71         "type": "keyword"
72       },
73     },
74     "code site": {
75       "type": "keyword"
76     },
77     "code zas": {
78       "type": "keyword"
79     },
80     "couverture de donnees": {
81       "type": "float"
82     },
83     "couverture temporelle": {
84       "type": "float"
85     },
86     "discriminant": {
87       "type": "keyword"
88     },
89     "nom site": {
90       "type": "text",
91       "fields": {
92         "keyword": {
93           "type": "keyword",
94           "ignore_above": 256
95         },
96         "ngram": {
97           "type": "text",
98           "analyzer": "ngram_analyzer"
99         }
100      },
101     },
102     "procedure de mesure": {
103       "type": "keyword"
104     },
105     "taux de saisie": {
106       "type": "float"
107     },
108     "type d'evaluation": {
109       "type": "keyword"
110     },
111     "type d'implantation": {
112       "type": "keyword"
113     },
114     "type d'influence": {
115       "type": "keyword"
116     },
117     "type de valeur": {
118       "type": "keyword"
119     },
120     "unite de mesure": {
121       "type": "keyword"
122     },
123     "valeur": {
124       "type": "float"
125     },
126     "valeur brute": {
127       "type": "float"
128     },
129     "validite": {
130       "type": "integer"
131     },
132     "quality_level": {
133       "type": "keyword"
134     }
135   }
136 }
```

## Réponses SPARK queries

```
scala> air_quality_polDF.printSchema()
```

```
root
|-- @timestamp: string (nullable = true)
|-- @version: string (nullable = true)
|-- Date de debut: string (nullable = true)
|-- Date de fin: string (nullable = true)
|-- Organisme: string (nullable = true)
|-- Polluant: string (nullable = true)
|-- Reglementaire: string (nullable = true)
|-- Zas: string (nullable = true)
|-- code qualite: string (nullable = true)
|-- code site: string (nullable = true)
|-- code zas: string (nullable = true)
|-- couverture de donnees: float (nullable = true)
|-- couverture temporelle: float (nullable = true)
|-- discriminant: string (nullable = true)
|-- nom site: string (nullable = true)
|-- procedure de mesure: string (nullable = true)
|-- quality_level: string (nullable = true)
|-- taux de saisie: float (nullable = true)
|-- timestamp_debut: string (nullable = true)
|-- timestamp_fin: string (nullable = true)
|-- type d'evaluation: string (nullable = true)
|-- type d'implantation: string (nullable = true)
|-- type d'influence: string (nullable = true)
|-- type de valeur: string (nullable = true)
|-- unite de mesure: string (nullable = true)
|-- valeur: float (nullable = true)
|-- valeur brute: float (nullable = true)
|-- valideite: integer (nullable = true)
```

```
scala> avgByImplantation.show()
```

```
+-----+-----+-----+
| type d'implantation | moyenne_pollution | nombre_mesures |
+-----+-----+-----+
| Urbaine | 23.051646436587628 | 237865 |
| Periurbaine | 20.603242090905557 | 72634 |
| Rurale regionale | 19.582851316649325 | 18637 |
| Rurale nationale | 17.010153224562895 | 9555 |
| Rurale pres des v... | 14.137275439583679 | 16461 |
+-----+-----+-----+
```

```
scala> qualityDistribution.show()
```

```
+-----+-----+
| quality_level | count |
+-----+-----+
| high | 323349 |
| medium | 22114 |
| low | 9689 |
+-----+-----+
```

```
scala> globalStats.show()
+-----+-----+-----+-----+
| moyenne_globale| ecart_type| min| max|total_mesures|
+-----+-----+-----+-----+
| 21.793167631689613| 26.58965821228122| -14.2| 920.8| 355152|
+-----+-----+-----+-----+
```

```
scala> avgPollutionByZAS.show()
+-----+-----+
|           Zas| moyenne_pollution|
+-----+-----+
|   ZAG MONTPELLIER| 42.54687378953462|
|   ZAR PERPIGNAN| 39.10323573891542|
|   ZAG STRASBOURG| 38.88623252316286|
| ZAR VALLEE-DE-L-ARVE| 34.71745102356462|
|   ZAG TOULON| 32.81374763674375|
|   ZAG TOULOUSE| 31.370475671384284|
|   ZAG PARIS| 30.50804569789925|
|   ZAG SAINT-ETIENNE| 29.704442259339825|
| ZR PROVENCE-ALPES...| 29.52072482978068|
|   ZAR PAYS-DE-SAVOIE| 28.36575583828834|
|   ZAR BESANCON| 27.952372882814057|
|   ZR CORSE| 26.942561250201052|
|   ZAG BORDEAUX| 26.565197753683755|
|   ZAG AVIGNON| 26.418482614024164|
|   ZAG ROUEN| 26.2345692413355|
|   ZAG BAYONNE| 26.07019774540259|
|   ZAR BREST| 25.723352186287105|
|   ZAR LIMOGES| 25.274837030587587|
| ZAR BELFORT-MONTB...| 25.11558658203024|
| ZAG CLERMONT-FERRAND| 24.68830753072778|
+-----+-----+
```

only showing top 20 rows

```
scala> mostCommonPollutants.show()
+-----+-----+
| Polluant| nombre_mesures|
+-----+-----+
|   NO2| 62526|
|   NO| 62172|
| NOX as NO2| 62157|
|   PM10| 58015|
|   O3| 50658|
|   PM2.5| 42493|
|   SO2| 13239|
|   CO| 2884|
|   C6H6| 1008|
+-----+-----+
```

## Liste de polluants

Code du polluant	Nom court du polluant	Unité de concentration
01	SO <sub>2</sub>	µg/m <sup>3</sup>
03	NO <sub>2</sub>	µg/m <sup>3</sup>
04	CO	mg/m <sup>3</sup>
08	O <sub>3</sub>	µg/m <sup>3</sup>
12	NO <sub>x</sub>	µg/m <sup>3</sup>
19	Pb dans les PM <sub>10</sub>	µg/m <sup>3</sup>
24	PM <sub>10</sub>	µg/m <sup>3</sup>
39	PM <sub>2.5</sub>	µg/m <sup>3</sup>
80	As dans les PM <sub>10</sub>	ng/m <sup>3</sup>
82	Cd dans les PM <sub>10</sub>	ng/m <sup>3</sup>
87	Ni dans les PM <sub>10</sub>	ng/m <sup>3</sup>
P6	BaP dans les PM <sub>10</sub>	ng/m <sup>3</sup>
V4	C <sub>6</sub> H <sub>6</sub>	µg/m <sup>3</sup>

TABLE 1 – Liste des polluants disponibles