

Rapport – Projet Big Data

Anas OUJJA, Sofian YADIR, Chaima MESSADI, Nour MANAI

Table des matières

1	Introduction	3
2	Configuration et architecture de la plateforme Big Data	4
3	Transformation	6
4	Analyse et Visualisation des Données d'Émissions de CO2 avec PySpark	7
4.1	Initialisation de la session Spark	7
4.2	Chargement des données	7
4.3	Préparation des données	7
4.4	Visualisation des données	7
4.5	Commandes et étapes nécessaires pour la visualisation	8
4.6	Conclusion	8
5	Analyse et Visualisation de la Relation entre la Consommation d'Énergie Renouvelable et la Superficie Forestière avec PySpark	9
5.1	Initialisation de la session Spark	9
5.2	Chargement des données	10
5.3	Préparation des données	10
5.4	Visualisation des données	10
5.5	Commandes et étapes nécessaires pour la visualisation	10
5.6	Conclusion	10
6	Optimisation des Calculs de Valeurs Nulles avec PySpark	11
6.1	Initialisation de la session Spark	11
6.2	Chargement des données	11
6.3	Préparation des données	11
6.4	Affichage des résultats	12
6.5	Écriture des résultats dans HDFS	12
6.6	Concaténation des fichiers CSV	12
6.7	Ajout de l'en-tête au fichier final	13
6.8	Conclusion	13
7	Analyse et Visualisation des Données d'Émissions de CO2 avec OpenSearch Dashboards	14
7.1	Chargement de données sur OpenSearch	14
7.1.1	Creation de fichiers Configs et Templates	14
7.1.2	Chargement de données	14
7.2	Visualisation par visBuilder	14

- 7.3 Visualisation Line Chart 19
- 7.4 Visualisation en VEGA 21
 - 7.4.1 Code 21
- 7.5 Dashboard 23
- 7.6 Controls 23
- 7.7 Conclusion 24

1 Introduction

Dans le cadre de ce projet de Big Data, nous avons entrepris une analyse approfondie de la dataset **World Development Indicators (WDI)** de la Banque Mondiale. Notre objectif principal est d'extraire des connaissances significatives de cette vaste source de données afin de mieux comprendre les tendances et les dynamiques du développement à l'échelle mondiale.

La dataset WDI de la Banque Mondiale est une ressource précieuse qui recueille une multitude d'indicateurs sur le développement économique, social et environnemental des pays du monde entier au fil du temps. Pour exploiter pleinement le potentiel de cette dataset, nous avons mis en œuvre une approche exhaustive, couvrant la collecte, la transformation, l'analyse et la visualisation des données.

Afin de répondre à notre objectif ambitieux, nous avons fait appel à un ensemble d'outils puissants et polyvalents du domaine du Big Data. Plus précisément, nous avons utilisé Hadoop, Spark et OpenSearch pour traiter, analyser et présenter les informations contenues dans la dataset WDI.

Hadoop, en tant que framework de traitement distribué, nous a permis de gérer efficacement le stockage et le traitement des données à grande échelle, en fournissant une infrastructure robuste pour la manipulation des données distribuées. Spark, de son côté, a été essentiel pour ses capacités de traitement rapide et ses bibliothèques riches, nous permettant d'exécuter des analyses complexes sur la dataset WDI avec une efficacité remarquable. Enfin, OpenSearch nous a offert des fonctionnalités avancées de recherche et de visualisation, facilitant la compréhension et l'interprétation des résultats de nos analyses.

Dans ce rapport, nous présenterons en détail notre approche méthodologique, les outils et techniques que nous avons utilisés, ainsi que les résultats obtenus de notre analyse de la dataset WDI. Nous mettrons en lumière les connaissances clés que nous avons découverts, les défis que nous avons rencontrés en cours de route, et les implications potentielles de nos résultats pour la prise de décision dans divers domaines du développement mondial.

En résumé, ce projet représente une exploration passionnante du potentiel du Big Data pour éclairer les enjeux complexes du développement mondial. En utilisant des outils avancés et une approche méthodologique rigoureuse, nous avons cherché à apporter une contribution significative à la compréhension et à la promotion du progrès mondial à travers l'analyse des données de la Banque Mondiale.

2 Configuration et architecture de la plateforme Big Data

Avant de plonger dans le traitement et l'analyse des données sur notre plateforme Big Data, il est essentiel de comprendre sa configuration et son architecture. Notre plateforme est conçue de manière modulaire et évolutive, composée de quatre composants principaux qui interagissent de manière harmonieuse pour permettre un traitement efficace des données. Voici un aperçu détaillé de chaque composant :

```
[anas@r3-16-de1 anas comps]$ ls
files  hadoop  opensearch  python
```

FIGURE 1 – Les composants

Stockage des fichiers : Nous utilisons simplement un répertoire sur le système de fichiers de notre machine pour stocker les données. Ce répertoire est organisé de manière à faciliter l'accès et la gestion des fichiers de données.

```
[anas@r3-16-de1 anas files]$ ls
figures  generated_files  indicators_economy.txt  indicators_environment.txt  WDI_CSV.csv  WDI_economy.csv  WDI_environment.csv  WDI_transformed.csv
```

FIGURE 2 – Contenu du répertoire de stockage des fichiers

Gestion des scripts Python : Nous avons un répertoire dédié où nous stockons tous nos scripts Python. Ces scripts sont utilisés pour diverses tâches, telles que la transformation des données, l'analyse et la génération de visualisations.

```
[anas@r3-16-de1 anas comps]$ cd python/
[anas@r3-16-de1 anas python]$ ls
ajout_header.py  chauma_version.py  create_conf_environment.py  create_merged_file.py  create_template_environment.py  transform.py
analyse_env.py  create_conf_economy.py  create_csvs_from_indicators.py  create_template_economy.py  deux_analyse_env.py  trois_analyse_env.py
```

FIGURE 3 – Contenu du répertoire de gestion des scripts Python

Cluster Hadoop : Notre plateforme intègre également un cluster Hadoop, utilisé pour le stockage et le traitement distribué des données volumineuses. Ce cluster est configuré à l'aide de Docker Compose et est composé de plusieurs conteneurs exécutant différentes composantes de Hadoop, telles que HDFS, YARN et MapReduce. Les composants du cluster Hadoop sont interconnectés pour faciliter les échanges de données entre eux.

```
[anas@r3-16-de1 anas ~]$ cd projet/comps/hadoop/
[anas@r3-16-de1 anas hadoop]$ ls
compute_nulls.sh  get_figures.sh  hadoop-8goRAM.env  overrides  start-hadoop.sh
docker-compose.yml  hadoop-16oRAM.env  hadoop.env  purge-hadoop.sh  stop-hadoop.sh
```

FIGURE 4 – Contenu du répertoire Hadoop

Le fichier **docker-compose.yml** configure notre cluster Hadoop. Voici une explication détaillée de chaque service :

- **Namenode** : Gère le namespace HDFS et régule l'accès des clients aux fichiers. Il utilise des volumes pour persister les métadonnées et inclut des configurations pour le réseau, les ports et l'environnement.
- **Datanode1 et Datanode2** : Stockent les données réelles. Chaque Datanode est configuré avec des volumes pour persister les données et inclut une condition de service pour s'assurer que le Namenode est disponible avant de démarrer.
- **Resourcemanager** : Gère les ressources et le scheduling des tâches. Il inclut des préconditions de service pour s'assurer que le Namenode et les Datanodes sont disponibles avant de démarrer.

- **Nodemanager1 et Nodemanager2** : Gèrent les ressources sur les nœuds et exécutent les conteneurs d'application. Ils incluent des préconditions de service pour s'assurer que les autres composants Hadoop nécessaires sont disponibles avant de démarrer.
- **Historyserver** : Fournit une vue des applications terminées. Il utilise des volumes pour persister les données de l'historique des tâches et inclut des préconditions de service.
- **Spark-Master et Spark-Workers** : Gèrent l'exécution des jobs Spark. Le Spark-Master coordonne les ressources et les Workers exécutent les tâches. Les conteneurs Spark sont configurés pour interagir avec Hadoop et sont dépendants du Spark-Master.

OpenSearch pour la visualisation : Nous avons également intégré OpenSearch dans notre plateforme pour la visualisation des données. Tout comme pour Hadoop, nous avons configuré un cluster OpenSearch en utilisant Docker Compose. Ce cluster est composé de plusieurs nœuds OpenSearch, ainsi que de Kibana pour la visualisation. Les nœuds OpenSearch et Kibana sont configurés pour travailler ensemble afin de permettre une exploration et une analyse efficaces des données.

```
[anas@r3-16-de1 anas hadoop]$ cd ../opensearch/
[anas@r3-16-de1 anas opensearch]$ ls
bug.json          docker-compose.yml  load_all.sh         opensearch.yml      wdi_economy_template.json
certs             files               logstash.conf       premier-lancement-opensearch.sh  wdi_environment_conf.conf
create_confs_templates.sh  generate-certs.sh  nettoie-opensearch.sh  transform.sh         wdi_environment_template.json
docker-compose-sanscert.yml  lance-opensearch.sh  opensearch-dashboards.yml  wdi_economy_conf.conf
```

FIGURE 5 – Contenu du répertoire OpenSearch

Explication des services OpenSearch définis dans docker-compose.yml

- **os01, os02, os03** : Ce sont les nœuds du cluster OpenSearch. Ils sont configurés pour découvrir et se connecter entre eux, formant ainsi un cluster résilient et évolutif. Chaque nœud a des paramètres pour la mémoire Java, la sécurité SSL et la configuration réseau.
- **Kibana** : Fournit une interface de visualisation pour les données indexées dans OpenSearch. Il se connecte aux nœuds OpenSearch et utilise des certificats pour sécuriser les communications.
- **Logstash** : Collecte et transforme les données avant de les envoyer à OpenSearch pour l'indexation. Il est configuré pour utiliser des certificats SSL et se connecte aux nœuds OpenSearch.

En plus de configurer les clusters, il est crucial de préparer notre environnement avant de commencer à travailler sur notre plateforme. Pour cela, nous utilisons des scripts shell qui automatisent cette tâche. Le script principal, `/projet/set_my_env.sh`, installe Python 3 et Pandas, des outils essentiels pour exécuter des scripts Python depuis la VM et transformer le dataset original.

```
#!/bin/bash

sudo yum -y update
sudo yum -y install python3
sudo yum -y install python3-pip
pip3 install pandas

exit 0
```

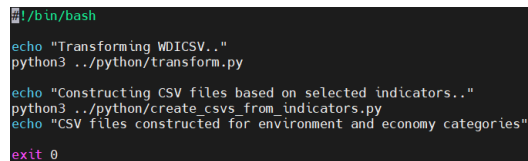
FIGURE 6 – Le script set_my_env.sh

3 Transformation

Dans cette section, nous présentons le processus de traitement du dataset original. Ce traitement s'est avéré nécessaire suite au téléchargement du dataset depuis le site de la Banque Mondiale et à son transfert vers notre VM. Nous avons rapidement constaté que la structure du dataset n'était pas adaptée pour être manipulée efficacement avec Spark ou pour être visualisée avec OpenSearch.

Pour obtenir un dataset exploitable, nous avons entrepris de le remodeler de manière à ce que chaque indicateur soit présenté dans une colonne distincte. Étant donné le nombre important d'indicateurs dans le dataset, nous avons fait le choix de nous concentrer sur une sélection d'indicateurs spécifiques. Pour ce faire, nous avons créé deux fichiers texte : **/projet/comps/files/indicators_economy.txt** et **/projet/comps/files/indicators_environment.txt**. Ces fichiers contiennent une liste des indicateurs pertinents pour l'environnement et l'économie, soigneusement sélectionnés avec l'aide de ChatGPT. Cette approche nous a permis de réduire la complexité du jeu de données tout en préservant les informations essentielles pour notre analyse.

Pour exécuter ces opérations de transformation, nous avons créé un script shell (**/projet/comps/opensearch/transform.sh**). Ce script orchestre l'exécution de deux scripts Python dédiés aux différentes tâches de transformation du dataset.



```
#!/bin/bash
echo "Transforming WDI CSV.."
python3 ../python/transform.py

echo "Constructing CSV files based on selected indicators.."
python3 ../python/create_csvs_from_indicators.py
echo "CSV files constructed for environment and economy categories"

exit 0
```

FIGURE 7 – Le script transform.sh

4 Analyse et Visualisation des Données d'Émissions de CO2 avec PySpark

Cette partie du rapport décrit les étapes suivies pour analyser et visualiser les données relatives aux émissions de CO2 à l'aide de PySpark et Matplotlib. Le but est d'exploiter la puissance de PySpark pour le traitement de grandes quantités de données et d'utiliser Matplotlib pour visualiser ces données de manière efficace.

```
from pyspark.sql import SparkSession
import pandas as pd
import matplotlib.pyplot as plt

# Initialiser une session Spark
spark = SparkSession.builder \
    .appName("Analyse Environment") \
    .getOrCreate()

# Lire le fichier CSV
df = spark.read.csv("/data/spark/files/WDI_environment.csv", header=True, inferSchema=True)

# Convertir le DataFrame Spark en DataFrame Pandas
pandas_df = df.toPandas()

# Exemple de visualisation avec Matplotlib
plt.figure(figsize=(10, 6))
plt.hist(pandas_df["CO2 emissions (kt)"].dropna(), bins=30, edgecolor='k') # Utiliser dropna() pour enlever les valeurs nulles
plt.title("Distribution de CO2 emissions (kt)")
plt.xlabel("Valeurs de CO2 emissions (kt)")
plt.ylabel("Fréquence")
plt.grid(True) # Ajouter une grille pour mieux visualiser
plt.savefig("/data/spark/files/figures/histogram.png")
```

4.1 Initialisation de la session Spark

- Importation de la classe `SparkSession` depuis le module `pyspark.sql`.
- Création d'une session Spark avec un nom explicite ("*Analyse Environment*"), ce qui facilite l'identification du processus dans un environnement multi-utilisateur ou lors de diagnostics.

4.2 Chargement des données

- Utilisation de la méthode `read.csv` de l'objet Spark pour charger les données depuis un fichier CSV. Les options `header=True` et `inferSchema=True` permettent respectivement de reconnaître la première ligne comme en-têtes de colonnes et d'inférer automatiquement le type de chaque colonne.

4.3 Préparation des données

- Conversion du DataFrame Spark en DataFrame Pandas pour utiliser les outils de visualisation de Matplotlib, qui ne sont pas directement compatibles avec les DataFrames Spark.

4.4 Visualisation des données

- Préparation de la visualisation en utilisant la bibliothèque Matplotlib pour créer un histogramme des émissions de CO2. Utilisation de la méthode `dropna()` pour exclure les valeurs nulles qui pourraient fausser l'analyse.
- Définition des paramètres de l'histogramme comme le nombre de `bins` et la couleur des bords des barres.

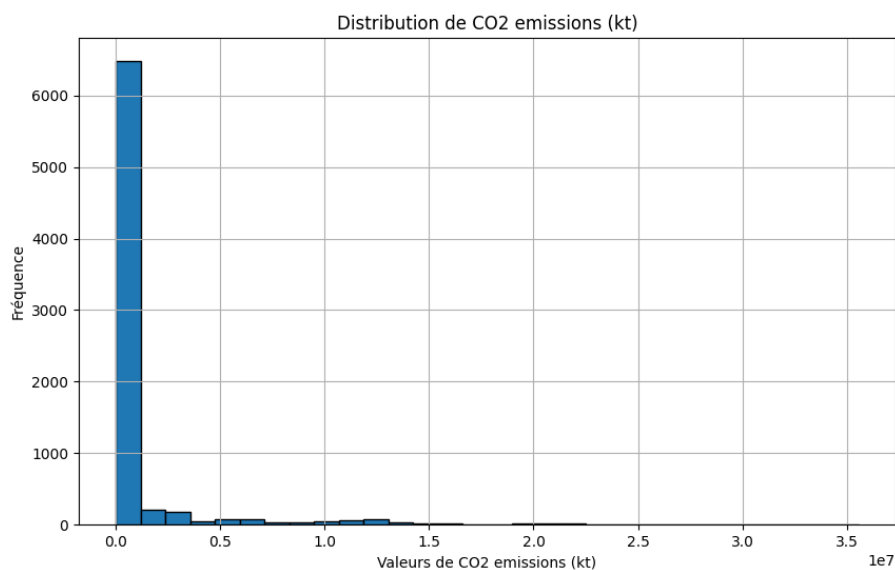
4.5 Commandes et étapes nécessaires pour la visualisation

- Exécution d'une session interactive de bash à l'intérieur d'un conteneur Docker : **docker exec -it spark-master bash**
- **cd /data/spark/python**
- **export PATH=\$PATH:/spark/bin/**
- **./analyse_env.py**
- Après l'exécution de ce script, une image nommée "histogram.png" est créée dans le dossier /projet/comps/files/figures.
- Pour visualiser cette image localement, on peut utiliser la commande **scp** pour la transférer depuis le serveur : **scp -i ./ensie.pem anas@51.75.95.72 :/home/anas /projet/-comps/files/figures/histogram.png chemin local**

Avant de pouvoir exécuter le script, certaines installations étaient nécessaires :

- Installation des bibliothèques **pandas** et **numpy** pour la manipulation des données :
apt-get update
apt-get install python3
apt-get install python3-pip
sudo apt-get install -y build-essential libatlas-base-dev
sudo python3 -m pip install --upgrade pip setuptools wheel
apk add --no-cache python3 py3-pip python3-dev build-base musl-dev libc-dev openblas-dev
pip3 install numpy
pip3 install pandas
- Installation de la bibliothèque **matplotlib** pour la visualisation des données :
apk add --no-cache zlib-dev jpeg-dev freetype-dev lcms2-dev openjpeg-dev tiff-dev tk-dev tcl-dev harfbuzz-dev fribidi-dev libimagequant-dev libwebp-dev
sudo python3 pip install matplotlib

4.6 Conclusion



La visualisation générée montre la distribution des émissions de CO2 parmi les différents enregistrements du dataset. L'histogramme met en évidence la concentration des valeurs vers des émissions plus faibles, tandis que des valeurs extrêmement élevées sont nettement moins fréquentes. Cette analyse et cette visualisation aident à comprendre rapidement la répartition des émissions de CO2 et peuvent servir de base pour des analyses plus détaillées sur les facteurs influençant ces émissions.

5 Analyse et Visualisation de la Relation entre la Consommation d'Énergie Renouvelable et la Superficie Forestière avec PySpark

Cette section décrit les étapes suivies pour analyser et visualiser la relation entre la consommation d'énergie renouvelable et la superficie forestière à l'aide de PySpark et Matplotlib. L'objectif est d'exploiter PySpark pour le traitement de grandes quantités de données et Matplotlib pour visualiser ces relations de manière efficace.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, avg
import pandas as pd
import matplotlib.pyplot as plt

# Initialiser une session Spark
spark = SparkSession.builder \
    .appName("Visualisation WDI avec PySpark") \
    .getOrCreate()

# Lire le fichier CSV
df = spark.read.csv("hdfs://tmp/data/WDI_environment.csv", header=True, inferSchema=True)

# Filtrer les données selon les critères spécifiés
df_filtered = df.filter(
    (col("renewable energy consumption (% of total final energy consumption)").between(30, 100)) &
    (col("forest area (% of land area)").between(40, 100))
)

# Agréger les données par pays et calculer la moyenne de la consommation d'énergie renouvelable et de la superficie forestière
df_grouped = df_filtered.groupBy("Country Name").agg(
    avg("renewable energy consumption (% of total final energy consumption)").alias("avg_renewable_energy"),
    avg("forest area (% of land area)").alias("avg_forest_area")
)

# Convertir le DataFrame Spark en DataFrame Pandas pour la visualisation
pandas_df = df_grouped.toPandas()

# Limiter 10 pays. Vous pouvez choisir vos propres pays ou les 10 premiers dans le DataFrame
selected_countries = pandas_df["Country Name"].unique()[:10]
pandas_df = pandas_df[pandas_df["Country Name"].isin(selected_countries)]

# Créer la visualisation avec Matplotlib
plt.figure(figsize=(12, 8))
for country in pandas_df["Country Name"].unique():
    country_data = pandas_df[pandas_df["Country Name"] == country]
    plt.plot(country_data["avg_forest_area"], country_data["avg_renewable_energy"], marker='o', linestyle='-', label=country)

plt.title("Visualisation de la consommation d'énergie renouvelable et de la superficie forestière")
plt.xlabel("Superficie forestière moyenne (% de la superficie totale des terres)")
plt.ylabel("Consommation d'énergie renouvelable moyenne (% de la consommation énergétique totale finale)")
plt.legend(title="Pays", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()

# Enregistrer la figure
output_dir = "/data/spark/files/figures/"
output_path = f"{output_dir}/renewable_energy_forest_area_visualization_limited.png"
plt.savefig(output_path)
plt.show()

# Arrêter la session Spark
spark.stop()
```

5.1 Initialisation de la session Spark

- Importation des classes `SparkSession` et fonctions depuis le module `pyspark.sql`.
- Création d'une session Spark avec un nom explicite ("*Visualisation WDI avec PySpark*"), ce qui facilite l'identification du processus dans un environnement multi-utilisateur ou lors de diagnostics.

5.2 Chargement des données

- Utilisation de la méthode `read.csv` de l'objet Spark pour charger les données depuis un fichier CSV. Les options `header=True` et `inferSchema=True` permettent respectivement de reconnaître la première ligne comme en-têtes de colonnes et d'inférer automatiquement le type de chaque colonne.

5.3 Préparation des données

- Filtrage des données selon les critères spécifiés (consommation d'énergie renouvelable et superficie forestière).
- Agrégation des données par pays pour calculer la moyenne de la consommation d'énergie renouvelable et de la superficie forestière.
- Conversion du DataFrame Spark en DataFrame Pandas pour utiliser les outils de visualisation de Matplotlib.

5.4 Visualisation des données

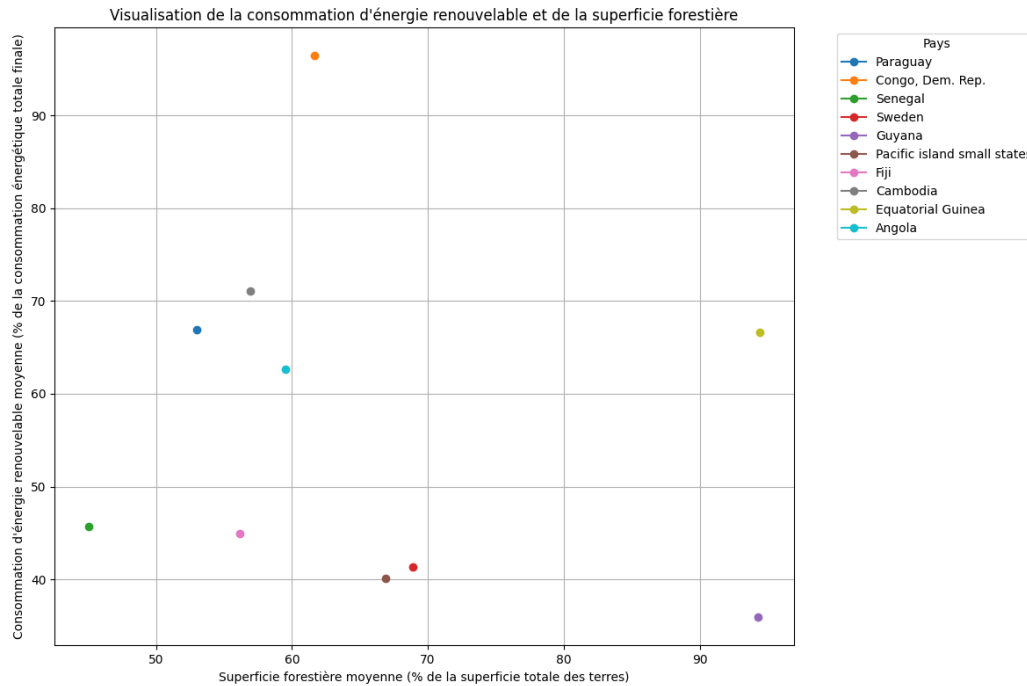
- Préparation de la visualisation en utilisant la bibliothèque Matplotlib pour créer un scatter plot représentant la consommation d'énergie renouvelable et la superficie forestière par pays.
- Limitation de la visualisation aux 10 premiers pays du DataFrame pour une meilleure lisibilité.

5.5 Commandes et étapes nécessaires pour la visualisation

- Exécution d'une session interactive de bash à l'intérieur d'un conteneur Docker : **`docker exec -it spark-master bash`**
- **`cd /data/spark/python`**
- **`export PATH=$PATH:/spark/bin/`**
- **`./deux_analyse_env.py`**
- Après l'exécution de ce script, une image nommée "histogram.png" est créée dans le dossier `/projet/comps/files/figures`.
- Exécution du script Python transformant et visualisant les données : **`python3 /projet/-comps/opensearch/transform.sh`**
- Après l'exécution de ce script, une image nommée "renewable_energy_forest_area_visualization_limited.png" est créée dans le dossier `/projet/comps/files/figures`.
- Pour visualiser cette image localement, on peut utiliser la commande `scp` pour la transférer depuis le serveur : **`scp -i ./ensie.pem anas@51.75.95.72 :/home/anas /projet/-comps/files/figures/renewable_energy_forest_area_visualization_limited.png chemin local`**

5.6 Conclusion

La visualisation générée montre la relation entre la consommation d'énergie renouvelable et la superficie forestière parmi les différents pays du dataset. Le scatter plot met en évidence les pays avec des valeurs particulièrement élevées ou faibles pour ces deux indicateurs. Cette analyse et cette visualisation aident à comprendre rapidement la relation entre la consommation d'énergie renouvelable et la superficie forestière, et peuvent servir de base pour des analyses plus détaillées sur les facteurs influençant ces indicateurs.



6 Optimisation des Calculs de Valeurs Nulles avec PySpark

Cette partie du rapport décrit l'utilisation de PySpark pour effectuer des calculs parallèles sur le dataset, en particulier pour compter les valeurs nulles pour chaque pays et pour chaque indicateur. PySpark permet de traiter efficacement de grandes quantités de données en utilisant le parallélisme et la puissance de calcul distribuée.

6.1 Initialisation de la session Spark

- Importation des classes `SparkSession` et fonctions depuis le module `pyspark.sql`, ainsi que des types de données nécessaires depuis `pyspark.sql.types`.
- Création d'une session Spark avec un nom explicite ("*Calcul des valeurs nulles dans le dataset WDI*"), ce qui facilite l'identification du processus dans un environnement multi-utilisateur ou lors de diagnostics.

6.2 Chargement des données

- Utilisation de la méthode `read.csv` de l'objet Spark pour charger les données depuis un fichier CSV stocké dans HDFS. Les options `header=True` et `inferSchema=True` permettent respectivement de reconnaître la première ligne comme en-têtes de colonnes et d'inférer automatiquement le type de chaque colonne.

6.3 Préparation des données

- Sélection des colonnes nécessaires pour l'analyse.
- Calcul du nombre de valeurs nulles pour chaque pays et pour chaque indicateur en utilisant la méthode `groupBy` et les fonctions d'agrégation de PySpark.

6.4 Affichage des résultats

- Affichage des résultats directement dans la console pour une vérification rapide.
- Conversion du DataFrame Spark en DataFrame Pandas pour un affichage plus convivial (optionnel).

6.5 Écriture des résultats dans HDFS

- Écriture des résultats dans HDFS en partitions, ce qui génère plusieurs fichiers CSV. Chaque fichier correspond à une partition des données, facilitant ainsi la gestion de grandes quantités de données.
- Extraction des en-têtes des colonnes originales et sauvegarde dans un fichier séparé.
- Création d'un DataFrame Spark vide avec les colonnes d'en-tête et écriture dans HDFS pour maintenir la structure des données.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, sum as spark_sum
from pyspark.sql.types import StructType, StructField, StringType
import os

# Initialize a Spark session
spark = SparkSession.builder \
    .appName("Calcul des valeurs nulles dans le dataset WDI") \
    .getOrCreate()

# Read the CSV file from HDFS
df = spark.read.csv("hdfs://namenode:9000/tmp/data/WDI_environment.csv", header=True, inferSchema=True)

# Display the schema of the DataFrame to understand the data structure
df.printSchema()

# Select the necessary columns
columns = df.columns

# Calculate the number of null values for each country and for each indicator
null_counts = df.groupBy("Country Name").agg(
    *[spark_sum(col(c).isNull().cast("int")).alias(f"{c}_null_count") for c in columns if c != "Country Name"]
)

# Display the results
null_counts.show()

# Convert the result to a Pandas DataFrame for a more friendly display (optional)
pandas_null_counts = null_counts.toPandas()
print(pandas_null_counts)

# Write the results in partitions to HDFS
output_dir_hdfs = "hdfs://namenode:9000/tmp/data/null_counts_by_country"
null_counts.write.mode("overwrite").option("header", "false").csv(output_dir_hdfs)

# Extract the header from the original DataFrame and save it to a separate file
header_columns = ["Country Name"] + [f"{c}_null_count" for c in columns if c != "Country Name"]

# Create an empty Spark DataFrame with the header columns
schema = StructType([StructField(col_name, StringType(), True) for col_name in header_columns])
header_df = spark.createDataFrame([], schema)

# Define the path for the header CSV file
header_csv_path = "hdfs://namenode:9000/tmp/data/null_counts_by_country"

# Write the header DataFrame to HDFS
header_df.write.mode("append").option("header", "true").csv(header_csv_path)

#header_df.write.mode("overwrite").option("header", "true").csv(header_csv_path)

# Stop the Spark session
spark.stop()
```

6.6 Concaténation des fichiers CSV

Après avoir généré plusieurs fichiers CSV sans en-tête, il était nécessaire de les concaténer pour obtenir un fichier unique contenant toutes les données agrégées. Pour ce faire, nous avons utilisé les commandes Docker et HDFS suivantes :

- Tout d'abord, nous avons accédé au conteneur Docker exécutant le namenode HDFS en utilisant la commande suivante :
`docker exec -it namenode bash`

- Ensuite, nous avons utilisé la commande `hdfs dfs -getmerge` pour concaténer tous les fichiers CSV dans un seul fichier, sans inclure les en-têtes de chaque fichier :
`hdfs dfs -getmerge /tmp/data/null_counts_by_country/ /tmp/data/generated_files/fichier_without_header.csv`
- Cette opération a permis de créer un fichier résultant nommé `fichier_without_header.csv` dans le répertoire `/tmp/data/generated_files`, ne contenant pas d'en-tête.

6.7 Ajout de l'en-tête au fichier final

Pour ajouter l'en-tête au fichier résultant, nous avons utilisé le script Python suivant :

```
import pandas as pd

# Chemin vers le fichier source pour l'en-tête
source_header_file = "~/projet/comps/files/WDI_environment.csv"

# Chemin vers le fichier cible
target_file = "/home/anas/projet/comps/files/generated_files/fichierChaima2.csv"

# Lire le fichier source pour obtenir l'en-tête
header_df = pd.read_csv(source_header_file, nrows=0)

# Lire le fichier cible sans en-tête
target_df = pd.read_csv(target_file, header=None)

# Ajouter les en-têtes au DataFrame cible
target_df.columns = header_df.columns

# ✦ Écrire le DataFrame cible avec les nouveaux en-têtes dans un nouveau fichier
output_file = "/home/anas/projet/comps/files/generated_files/versionChaima2_with_header.csv"
target_df.to_csv(output_file, index=False)

print(f"Le fichier avec l'en-tête ajouté a été enregistré sous {output_file}")
```

Pour exécuter ce script, nous avons suivi les étapes suivantes :

- Installation de la bibliothèque `fsspec` avec la commande :
`sudo pip3 install fsspec`
- Navigation vers le répertoire contenant le script Python :
`cd ~/projet/comps/python`
- Exécution du script avec la commande :
`python3 ajout_header.py`

6.8 Conclusion

L'utilisation de PySpark a permis d'effectuer des calculs parallèles sur le dataset, améliorant ainsi l'efficacité du traitement des données. La génération de fichiers CSV en partitions et leur concaténation ultérieure permettent de maintenir une cohérence dans la structure des résultats, facilitant ainsi les analyses ultérieures. L'ajout de l'en-tête au fichier final a été réalisé avec succès en utilisant un script Python.

7 Analyse et Visualisation des Données d'Émissions de CO2 avec OpenSearch Dashboards

7.1 Chargement de données sur OpenSearch

7.1.1 Creation de fichiers Configs et Templates

Execution de fichier `/projet/comps/opensearch/create_confs_templates.sh` qui va lancer des scripts Python qui vont créer les fichiers `Template.json` et `Config.conf` pour les fichiers `wdienvironment.csv` et `wdieconomy.csv`

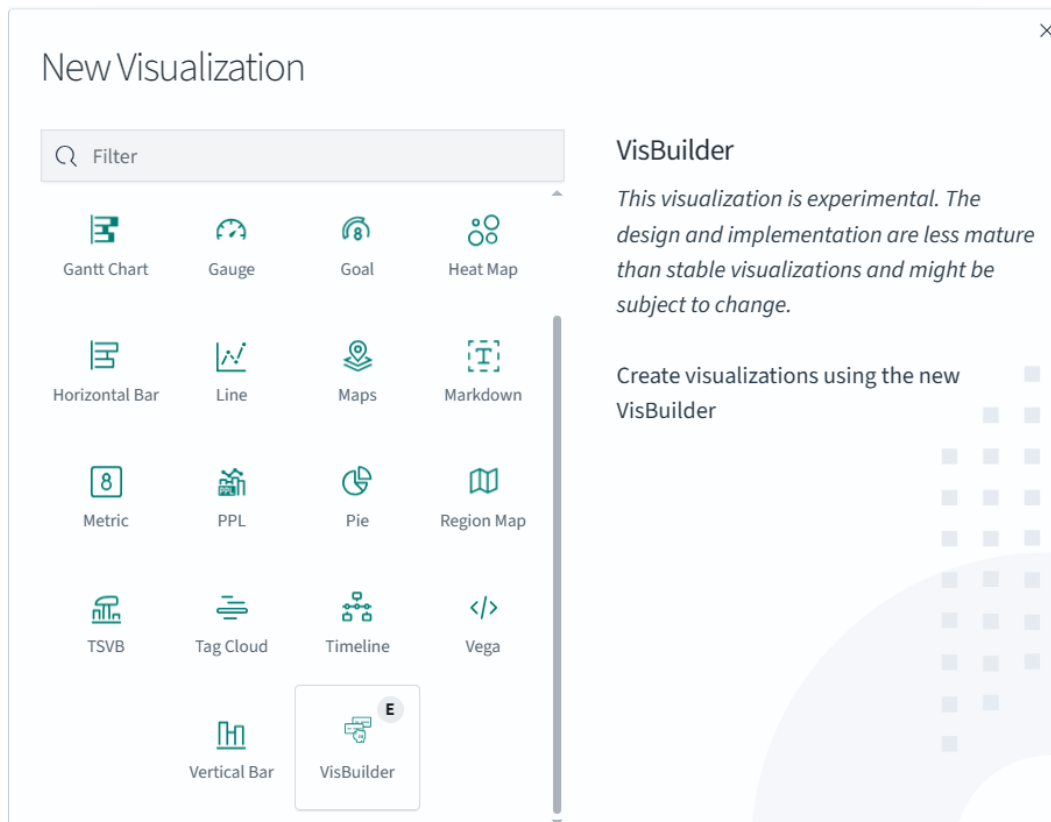
```
[anas@r3-16-de1 anas ~]$ cd projet/comps/opensearch
[anas@r3-16-de1 anas opensearch]$ ls
bug.json          docker-compose.yml  load_all.sh         opensearch.yml      wdi_economy_template.json
certs             files               logstash.conf       premier-lancement-opensearch.sh  wdi_environment_conf.conf
create_confs_templates.sh  generate-certs.sh  nettoie-opensearch.sh  transform.sh         wdi_environment_template.json
docker-compose-sanscert.yml  lance-opensearch.sh  opensearch-dashboards.yml  wdi_economy_conf.conf
```

7.1.2 Chargement de données

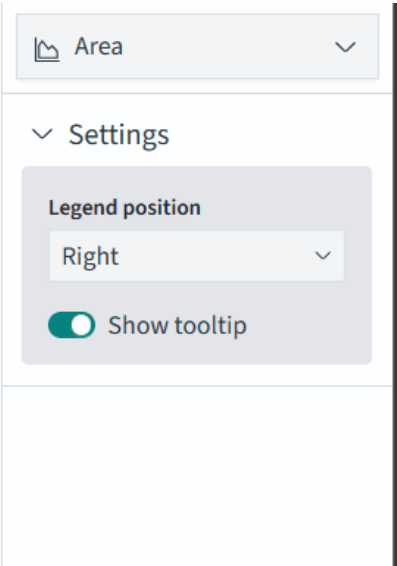
Exécution du fichier `/projet/comps/opensearch/load_all.sh` qui effectuera à son tour 2 requêtes curl pour charger les données des deux fichiers `.csv` dans OpenSearch.

7.2 Visualisation par visBuilder

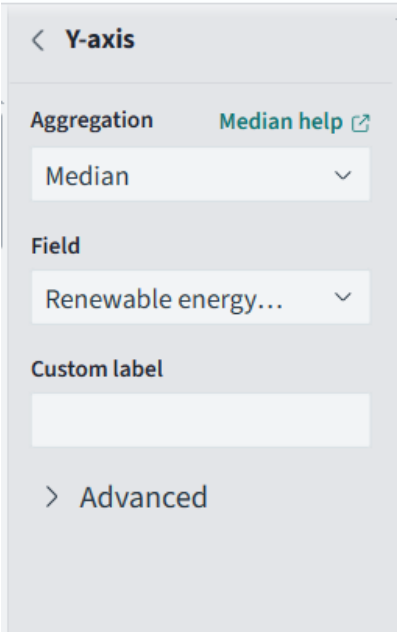
Dans "Create Visualization", nous avons sélectionné "VisBuilder" dans la liste des types de visualisation.



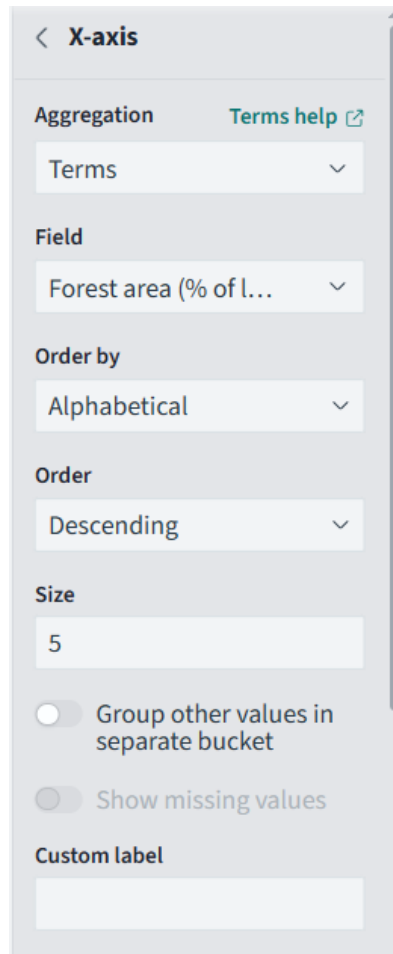
Dans VisBuilder, nous avons choisi "Area" pour le type de graphique.



Pour configurer l’Axe Y dans VisBuilder, nous avons sélectionné ”Median” comme type de métrique, ce qui nous permettra d’afficher la valeur médiane des données. Puis, nous avons choisi le champ renewable energy consumption (pourcentage of total final energy consumption) pour cette métrique, afin de visualiser les données de consommation d’énergie renouvelable en pourcentage de la consommation énergétique totale finale.



Pour configurer l'Axe X, nous avons choisi "Terms" comme type d'agrégation et avons sélectionné le champ 'forest area (pourcentage of land area)'. Cela nous a permis de visualiser les données de la superficie forestière en pourcentage de la superficie totale des terres sur l'axe X.




The image shows a configuration panel for the X-axis of a chart. The panel is titled "X-axis" with a back arrow. It contains several settings:

- Aggregation:** A dropdown menu set to "Terms". A "Terms help" link with an external icon is next to it.
- Field:** A dropdown menu set to "Forest area (% of l...".
- Order by:** A dropdown menu set to "Alphabetical".
- Order:** A dropdown menu set to "Descending".
- Size:** A text input field containing the number "5".
- Group other values in separate bucket:** A toggle switch that is currently turned off.
- Show missing values:** A toggle switch that is currently turned off.
- Custom label:** An empty text input field.

Pour configurer le Split Area, nous avons choisi "Terms" comme type d'agrégation et avons sélectionné le champ 'Country Name'. Pour l'ordre, nous avons opté pour "Descending". Cela nous a permis de diviser la visualisation en fonction des noms des pays, affichés par ordre décroissant.

< Split series

Aggregation

Terms help 

Terms

▼

Field

Country Name

▼

Order by

Alphabetical

▼

Order

Descending

▼

Size

5

☐

Group other values in separate bucket

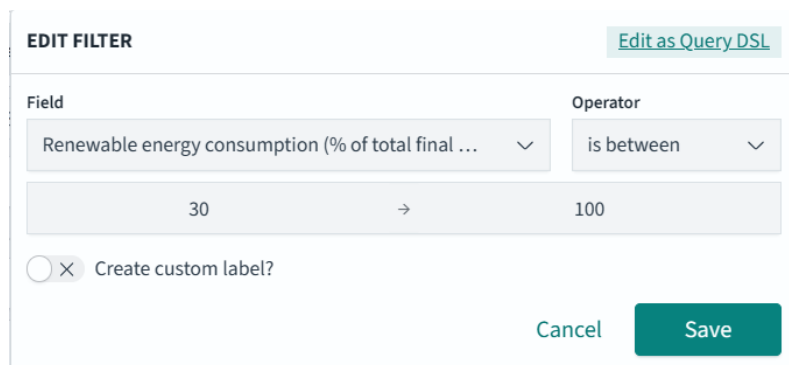
☐

Show missing values

Custom label

> Advanced

Pour appliquer les filtres, nous avons ajouté un filtre pour la consommation d'énergie renouvelable. Nous avons cliqué sur "Add Filter". Nous avons sélectionné le champ 'renewable energy consumption (pourcentage of total final energy consumption)', puis nous avons choisi "is between" comme opérateur. Nous avons entré les valeurs "30" et "100" pour définir le filtre, ce qui nous a permis de restreindre notre visualisation aux données de consommation d'énergie renouvelable comprises entre 30 pourcent et 100 pourcent.



EDIT FILTER [Edit as Query DSL](#)

Field: Renewable energy consumption (% of total final ...)

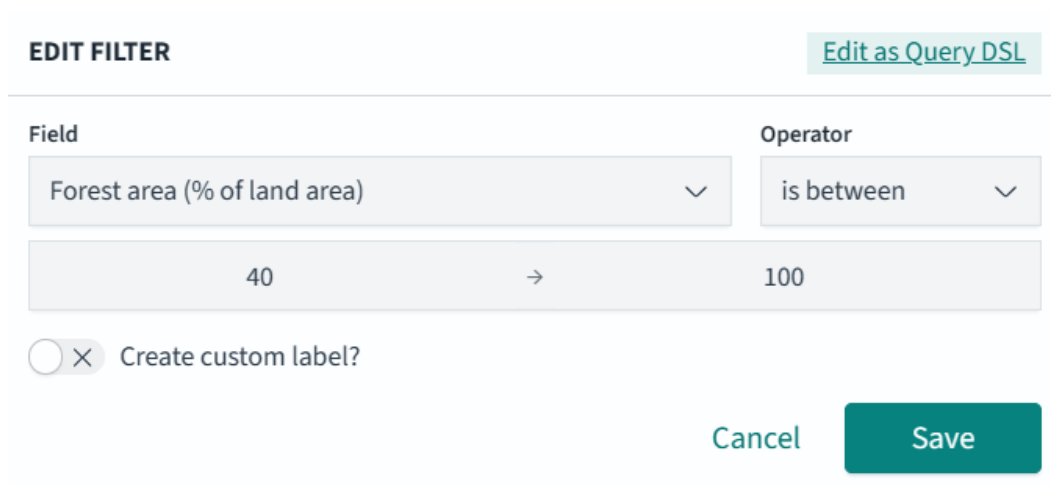
Operator: is between

30 → 100

☐ Create custom label?

Cancel Save

Pour ajouter un filtre pour la superficie forestière, nous avons cliqué de nouveau sur "Add Filter". Nous avons sélectionné le champ 'forest area (pourcentage of land area)' et avons choisi "is between" comme opérateur. Ensuite, nous avons entré les valeurs "40" et "100" pour définir le filtre, ce qui nous a permis de restreindre notre visualisation aux données de superficie forestière comprises entre 40 pourcent et 100 pourcent.



EDIT FILTER [Edit as Query DSL](#)

Field: Forest area (% of land area)

Operator: is between

40 → 100

☐ Create custom label?

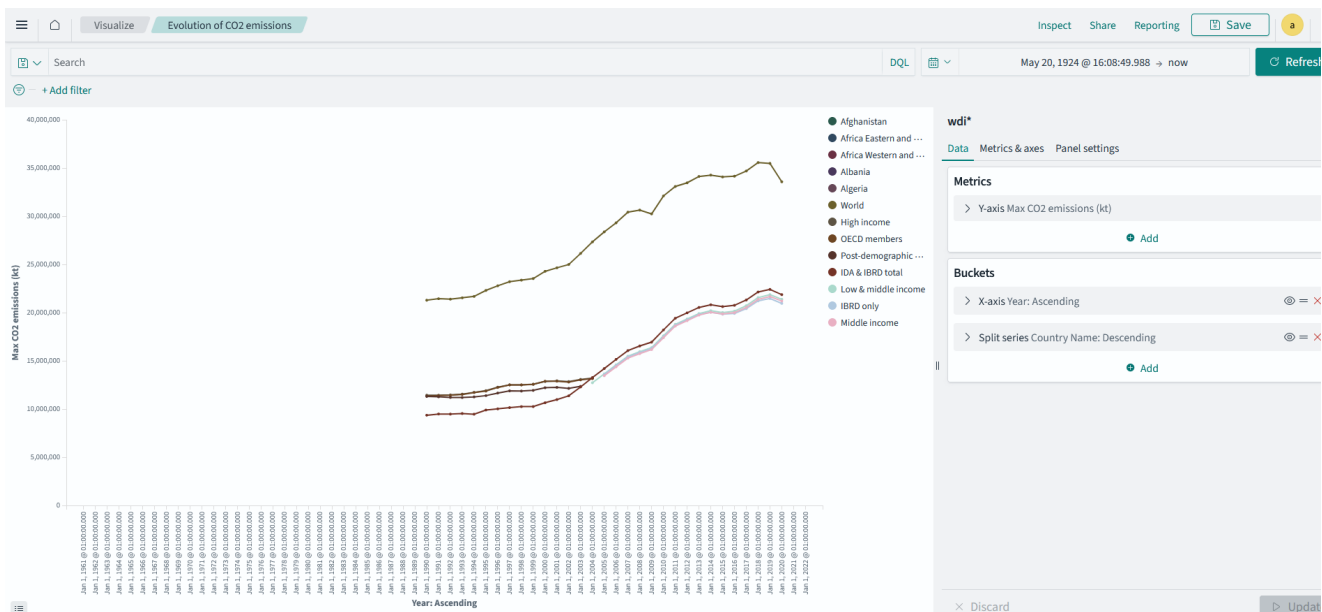
Cancel Save

Après avoir configuré les axes, le split area, et les filtres, nous avons cliqué sur "Update" pour générer la visualisation.



7.3 Visualisation Line Chart

Nous avons réalisé une visualisation sous forme de Line chart pour représenter la quantité d'émissions de CO2 en kt pour chaque pays du monde au fil des années.



— Y-axis : émissions de CO2 en kt

Metrics

Y-axis

Aggregation

Max

Max help

Field

CO2 emissions (kt)

Custom label

Advanced

+

 Add

— X axis : les afficher pour chaque année

X-axis

Terms help

Aggregation

Terms

Field

Year

Order by

Alphabetical

Order

Ascending

Size

64

☐ Group other values in separate bucket

☐ Show missing values

Custom label

— Split series : les trier par émissions CO2 et afficher que le 5 valeur plus élevée

Split series

Sub aggregation

Terms

Field

Country Name

Order by

Metric: Max CO2 emissions (kt)

Order

Descending

Size

5

☐ Group other values in separate bucket

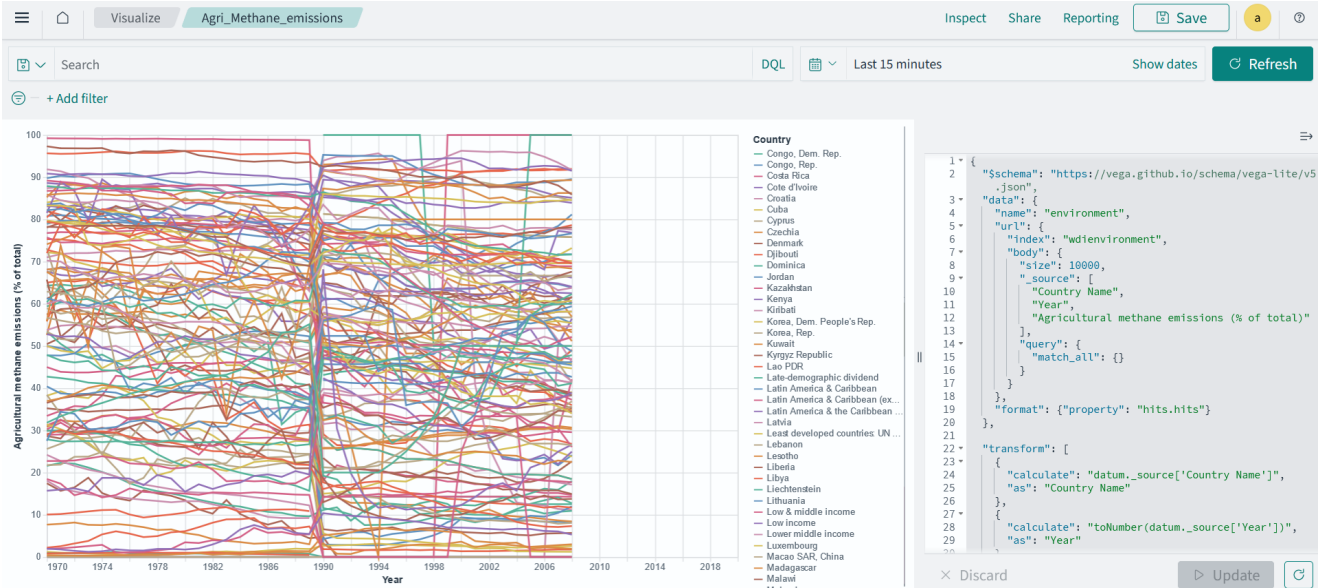
☐ Show missing values

Custom label

> Advanced

7.4 Visualisation en VEGA

Nous avons réalisé une visualisation VEGA pour représenter le pourcentage d'émissions de méthane agricole par rapport aux émissions totales pour chaque pays dans le monde au fil des années.



7.4.1 Code

— Accés au données

```
"data": {
  "name": "environment",
  "url": {
    "index": "wdienvironment",
    "body": {
      "size": 10000,
      "_source": [
        "Country Name",
        "Year",
        "Agricultural methane emissions (% of total)"
      ],
      "query": {
        "match_all": {}
      }
    }
  },
  "format": {"property": "hits.hits"}
},
```

— Transformation de données

```
"transform": [
  {
    "calculate": "datum._source['Country Name']",
    "as": "Country Name"
  },
  {
    "calculate": "toNumber(datum._source['Year'])",
    "as": "Year"
  },
  {
    "calculate": "toNumber(datum._source['Agricultural methane emissions (% of total)'])",
    "as": "Agricultural methane emissions (% of total)"
  },
  {
    "filter": "isFinite(datum['Year']) && isFinite(datum['Agricultural methane emissions (% of total)'])"
  },
  /* {
    "filter": "Array.isArray(dashboardFilters) && dashboardFilters.includes(datum['CountryName'])"
  }
  */
], /*
```

— Tentative échouée de lier la visualisation VEGA aux Controls du Dashboard.

```
"params": [
  {
    "name": "dashboardFilters",
    "value": [],
    "bind": {
      "input": "select",
      "options": [{"signal": "getFilterValues('Country Name')"}],
      "name": "Select Country: "
    }
  }
]
```

— Pour faire la visualisation

```
"mark": "line",
"encoding": {
  "x": {
    "field": "Year",
    "type": "quantitative",
    "title": "Year",
    "axis": {"format": "d"},
    "scale": {"domain": [1970, 2020]}
  },
  "y": {
    "field": "Agricultural methane emissions (% of total)",
    "type": "quantitative",
    "title": "Agricultural methane emissions (% of total)",
    "scale": {"domain": [0, 100]}
  },
  "color": {
    "field": "Country Name",
    "type": "nominal",
    "title": "Country",
    "legend": {"symbolLimit": 200}
  }
}
```

7.5 Dashboard

Nous créons un nouveau tableau de bord et y ajoutons les visualisations que nous avons réalisées.

7.6 Controls

Nous avons créé un panneau de contrôles pour filtrer les données visualisées dans les graphiques par pays.

[GitHub](#) 