

Esnault Julien - Galaad - Sofiane

TP BIGDATA

M1 DEV FULL STACK - PAR02 - 2024

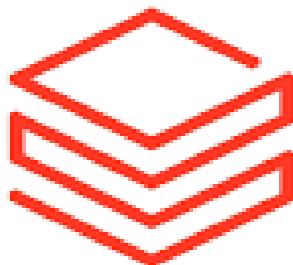
15/12/2024

EFREI

Databricks Notebook :

<https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3445251035974576/367390688357069/1714409181088165/latest.html>

Repo Github : <https://github.com/julienESN/databricks-vaccination-analysis>



**databricks**

## Présentation du projet, des données, et des objectifs

Présentation du projet :

Le projet consiste en la mise en place d'une architecture **Lakehouse** pour le traitement et l'analyse des données mondiales de vaccination. L'objectif principal est de créer un pipeline de données évolutif et robuste en utilisant les principes de stockage en **zones Bronze, Silver, et Gold**, qui sont des pratiques courantes dans l'ingénierie des données et permettent d'assurer la qualité, la traçabilité et la performance des traitements.

Le projet utilise **Databricks** et **Apache Spark** pour la transformation des données, et s'appuie sur le modèle **Lakehouse** qui combine les avantages des architectures **Data**

**Lake et Data Warehouse**, tout en permettant une gestion optimisée des données en temps réel et historiques.

### Les données :

Les données utilisées dans ce projet proviennent de diverses sources publiques qui répertorient les informations sur les vaccins COVID-19 administrés à travers le monde(<https://data.who.int/dashboards/covid19/data?n=c>).

Le dataset de base inclut des informations détaillées par pays et par région, telles que :

- **Le nombre total de vaccinations** réalisées dans chaque pays.
- **Le nombre de personnes vaccinées avec une dose** et celles ayant reçu la dose de rappel (booster).
- **Les données démographiques** et géographiques telles que la région OMS et l'ISO3 des pays.

Les données sont stockées sous forme de fichiers CSV, avec des colonnes comprenant des informations sur les **vaccinations**, les **personnes vaccinées** et les **dates de mise à jour** des informations. Ce jeu de données est chargé dans un système de stockage distribué comme **DBFS** (Databricks File System).

### Les objectifs du projet :

L'objectif principal de ce projet est de construire une **pipeline de données** qui permet d'agréger et de transformer les données de vaccination dans un format structuré, afin de produire des insights et visualisations utiles pour les analystes et décisionnaires.

Les étapes suivantes doivent être réalisées :

#### 1. Chargement des données brutes (Bronze) :

- Récupérer les données de vaccination sous forme brute dans la zone **Bronze**, sans transformation initiale.

#### 2. Nettoyage et transformation des données (Silver) :

- Filtrer les données, combler les valeurs manquantes, et transformer les données dans un format standardisé (par exemple, conversion de dates et types de données).
- Créer des dimensions et nettoyer les colonnes inutiles.

### 3. Modélisation des données (Gold) :

- Créer des tables analytiques dans la zone **Gold** en utilisant un modèle en **étoile**. Ce modèle permettra de réaliser des analyses sur les données de vaccination par pays et par région OMS.
- 

### 4. Enrichissement des données :

- Ajouter des enrichissements aux données, comme des calculs de ratios ou des agrégations, pour mieux comprendre la progression des vaccinations à travers le monde.

### 5. Visualisations et analyse :

- Générer des visualisations dans des outils comme **Power BI** ou **Apache Superset**, ou directement dans **Databricks Notebooks**, pour faciliter l'analyse des résultats des vaccins.

## Flux de données et architecture

### 1. Architecture globale du projet :

Le projet repose sur une architecture **Lakehouse**, qui combine les avantages des **Data Lakes** (stockage brut et massif des données) et des **Data Warehouses** (données structurées pour l'analyse). Cette architecture se compose de trois zones principales pour le stockage et le traitement des données : **Bronze**, **Silver** et **Gold**.

- **Zone Bronze** : Cette zone stocke les **données brutes** telles qu'elles sont ingérées depuis les fichiers sources (fichiers CSV). Aucune transformation n'est effectuée dans cette zone.
- **Zone Silver** : Dans cette zone, les données sont **nettoyées**, **transformées**, et enrichies. Les valeurs manquantes sont traitées, les types de données sont corrigées, et les colonnes inutiles sont supprimées.
- **Zone Gold** : Les données transformées et enrichies sont modélisées pour l'analyse. Cette zone suit un **modèle en étoile** (Star Schema), permettant des requêtes rapides pour des analyses avancées et des visualisations.

L'architecture de traitement des données peut être représentée par le schéma suivant:

Données Sources (CSV) --> Zone Bronze (stockage brut) --> Zone Silver (transformation) --> Zone Gold (modèle analytique)

## 2. Flux de données :

Le flux de données suit une séquence d'étapes clairement définies :

### 1. Ingestion des données (Zone Bronze) :

- Les données sources sont récupérées sous forme de fichiers CSV et stockées dans la zone **Bronze** via **Apache Spark**.
- **Objectif** : Stocker les données dans leur format brut pour permettre un audit ou un reprocessing en cas de besoin.

### 2. Exemple de code :



```
df_bronze = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load("/FileStore/tables/vaccination_data.csv")

df_bronze.write.format("delta").mode("overwrite").save("/mnt/datalake/bronze/vaccination_data")
```

snappify.com

## Nettoyage et transformation des données (Zone Silver) :

- Les données de la zone **Bronze** sont lues, puis des transformations sont appliquées :
  - Suppression des colonnes inutiles.
  - Traitement des valeurs manquantes.
  - Conversion des types de données (exemple : formats de date).
- Les données nettoyées sont stockées dans la zone **Silver**.

Exemple de code :

```
df_silver = spark.read.format("delta").load("/mnt/datalake/bronze/vaccination_data")

df_silver = df_silver.drop("VACCINES_USED", "NUMBER_VACCINES_TYPES_USED", "DATA_SOURCE") \
    .filter(df_silver["TOTAL_VACCINATIONS"].isNotNull()) \
    .fillna({"WHO_REGION": "Unknown"})

df_silver.write.format("delta").mode("overwrite").save("/mnt/datalake/silver/vaccination_data_cleaned")
```

snappify.com

### 3. Modélisation et enrichissement des données (Zone Gold) :

- Les données **nettoyées** de la zone Silver sont agrégées et enrichies pour générer les tables **Fait** et **Dimensions** selon un **modèle en étoile** :
  - **Table Fait** : Contient les mesures analytiques comme le nombre total de vaccinations, les personnes vaccinées avec une dose, etc.
  - **Dimensions** : Comptent des informations géographiques (pays, région) et temporelles (année, mois, jour).

```
# Création de la table Fait
fact_table = df_silver.select(
    "COUNTRY", "WHO_REGION", "DATE_UPDATED",
    "TOTAL_VACCINATIONS", "PERSONS_VACCINATED_1PLUS_DOSE"
)

fact_table.write.format("delta").mode("overwrite").save("/mnt/datalake/gold/fact_covid_vaccinations")
```

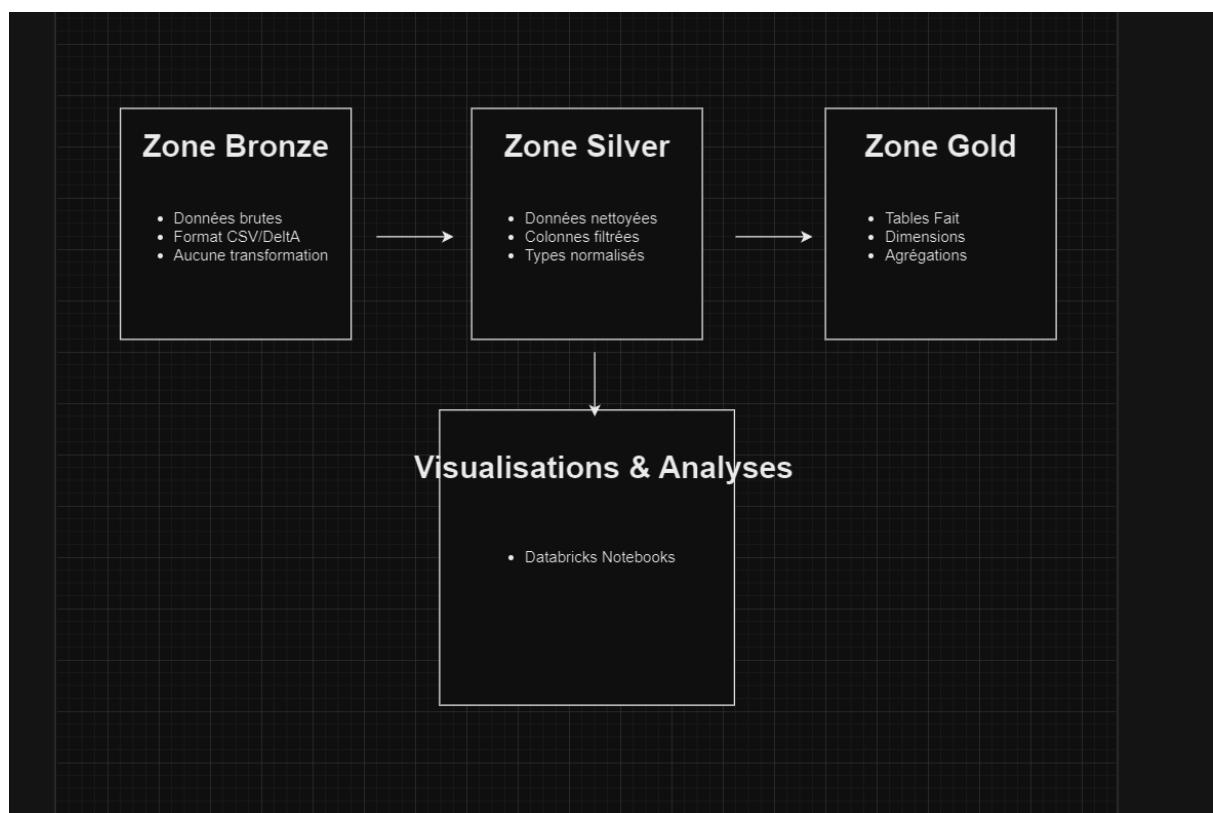
snappify.com

### 4. Visualisation et analyse :

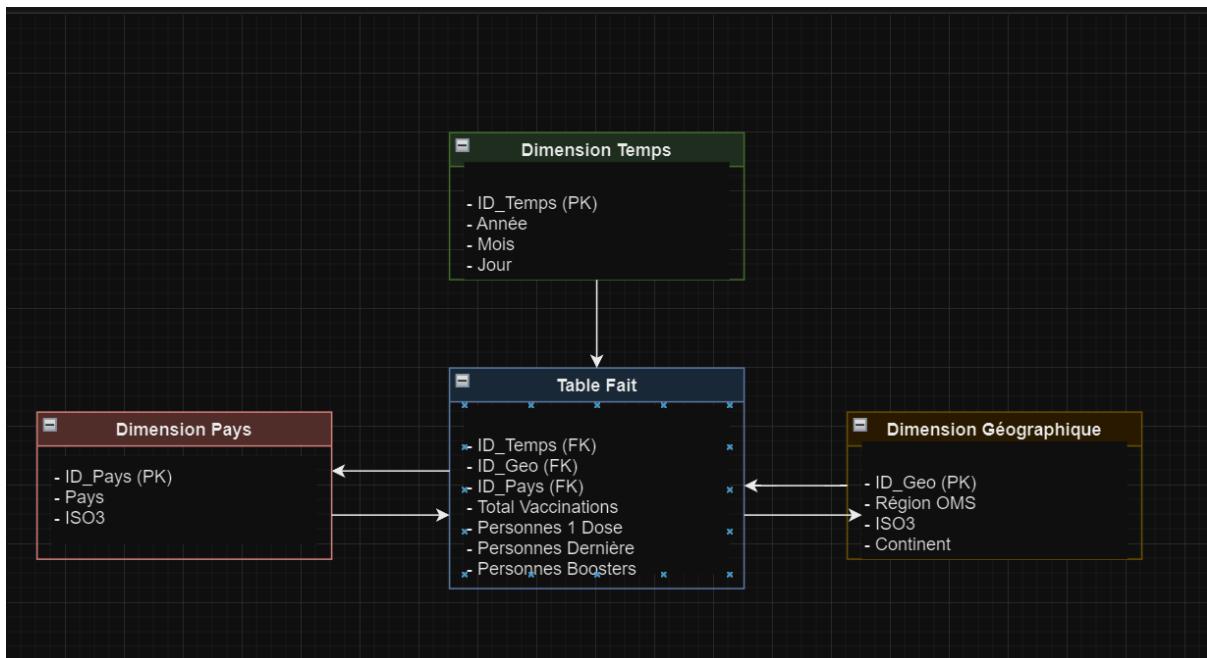
- Les tables **Gold** sont utilisées pour créer des visualisations dans **Power BI**, **Apache Superset**, ou directement dans **Databricks Notebooks**.
- Ces visualisations permettent d'analyser la progression des vaccinations à travers le monde par pays, région OMS, et périodes temporelles.

### 3.Schéma d'architecture du projet :

Le schéma suivant illustre l'architecture de l'ensemble du projet :



### 3. Modèle Conceptuel des Données (MCD)



Le Modèle Conceptuel des Données (MCD) suit un modèle en étoile pour faciliter l'analyse des données. Ce modèle est constitué d'une table Fait contenant les mesures analytiques et de plusieurs tables Dimensions qui décrivent les axes d'analyse.

---

#### Table Fait

La Table Fait centralise les mesures principales liées aux vaccinations. Elle inclut des clés étrangères permettant de relier les différentes dimensions :

- ID\_Temps (FK) : Référence à la dimension Temps.
  - ID\_Geo (FK) : Référence à la dimension Géographique.
  - ID\_Pays (FK) : Référence à la dimension Pays.
  - Mesures analytiques :
    - Total Vaccinations
    - Personnes 1 Dose
    - Personnes Dernière
    - Personnes Boosters
-

## Tables Dimensions

### 1. Dimension Temps

Cette table décrit les informations temporelles associées aux données de vaccination :

- ID\_Temps (PK) : Clé primaire.
- Année
- Mois
- Jour

### 2. Dimension Géographique

Cette table contient les informations géographiques :

- ID\_Geo (PK) : Clé primaire.
- Région OMS
- ISO3
- Continent

### 3. Dimension Pays

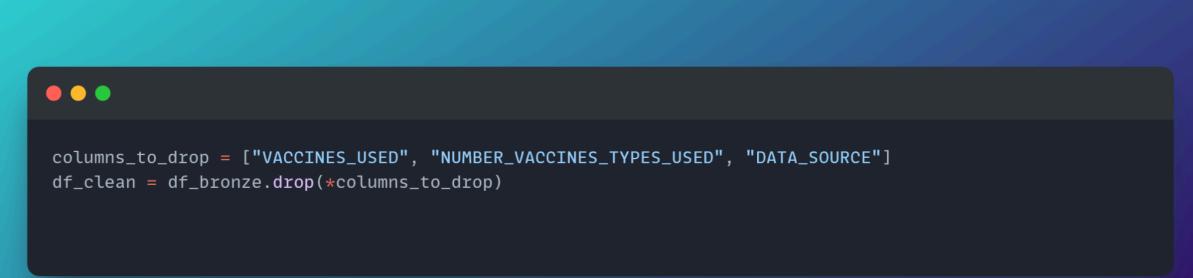
Cette table fournit les détails des pays :

- ID\_Pays (PK) : Clé primaire.
- Pays
- ISO3

## 4. Traitements effectués et transformations

### 1. Suppression des colonnes inutiles

Dans la **zone Silver**, certaines colonnes inutiles pour les analyses ont été supprimées afin de réduire la taille des données et améliorer les performances des traitements.



```
columns_to_drop = ["VACCINES_USED", "NUMBER_VACCINES_TYPES_USED", "DATA_SOURCE"]
df_clean = df_bronze.drop(columns_to_drop)
```

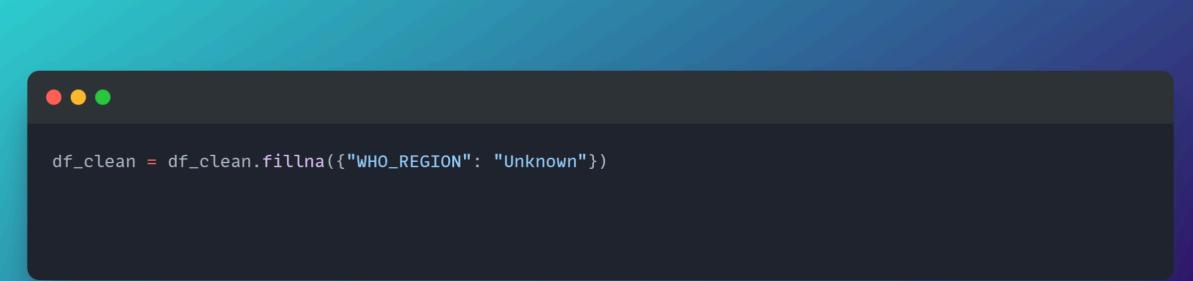
snappyf.com

### Explication :

- `drop()` permet de retirer les colonnes `VACCINES_USED`, `NUMBER_VACCINES_TYPES_USED` et `DATA_SOURCE` qui ne sont pas nécessaires pour l'analyse.

## 2. Gestion des valeurs manquantes

Les valeurs manquantes dans certaines colonnes critiques ont été remplacées pour assurer la continuité des données.



```
df_clean = df_clean.fillna({"WHO_REGION": "Unknown"})
```

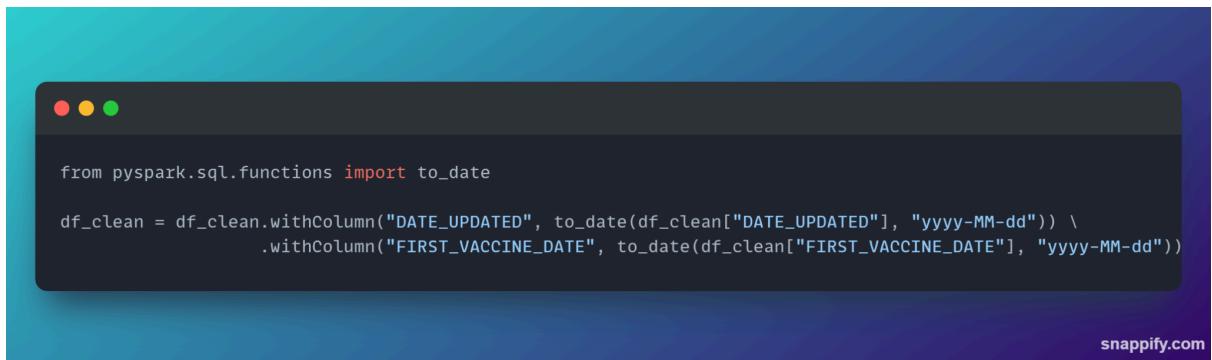
snappyf.com

### Explication :

- Les valeurs manquantes dans la colonne **WHO\_REGION** sont remplacées par "**Unknown**" pour éviter des erreurs lors des analyses ou des agrégations.

### 3. Conversion des types de données

Les colonnes de dates ont été converties au format standard **date** afin de permettre des opérations temporelles.



```
from pyspark.sql.functions import to_date

df_clean = df_clean.withColumn("DATE_UPDATED", to_date(df_clean["DATE_UPDATED"], "yyyy-MM-dd")) \
    .withColumn("FIRST_VACCINE_DATE", to_date(df_clean["FIRST_VACCINE_DATE"], "yyyy-MM-dd"))
```

The screenshot shows a Jupyter Notebook cell with the following Python code:

```
from pyspark.sql.functions import to_date

df_clean = df_clean.withColumn("DATE_UPDATED", to_date(df_clean["DATE_UPDATED"], "yyyy-MM-dd")) \
    .withColumn("FIRST_VACCINE_DATE", to_date(df_clean["FIRST_VACCINE_DATE"], "yyyy-MM-dd"))
```

The code uses the `to_date` function from the `pyspark.sql.functions` module to convert two columns, `DATE_UPDATED` and `FIRST_VACCINE_DATE`, from their original string formats to the `date` type. The new columns are created by specifying the column names and the desired date format `yyyy-MM-dd`.

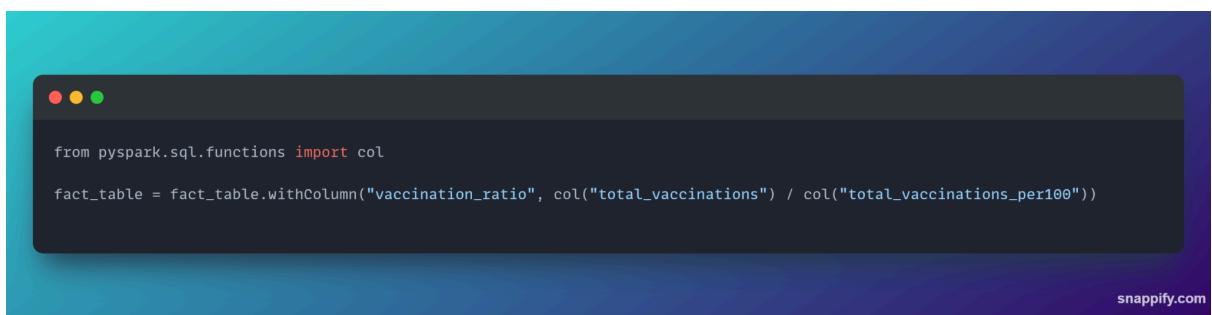
**Explication :**

- La fonction **to\_date()** est utilisée pour convertir les chaînes de caractères en dates au format **yyyy-MM-dd**.

### 4. Agrégations et enrichissements

#### a. Ajout du ratio de vaccination par population

Un ratio de vaccination a été calculé pour mieux analyser la couverture vaccinale.



```
from pyspark.sql.functions import col

fact_table = fact_table.withColumn("vaccination_ratio", col("total_vaccinations") / col("total_vaccinations_per100"))
```

The screenshot shows a Jupyter Notebook cell with the following Python code:

```
from pyspark.sql.functions import col

fact_table = fact_table.withColumn("vaccination_ratio", col("total_vaccinations") / col("total_vaccinations_per100"))
```

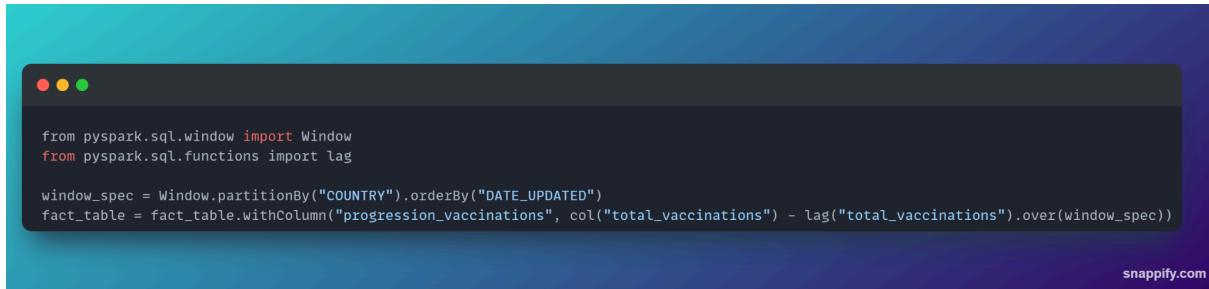
The code uses the `withColumn` method of a DataFrame to add a new column named `vaccination_ratio`. This column is calculated by dividing the `total_vaccinations` column by the `total_vaccinations_per100` column. The result is a float representing the vaccination ratio per 100 people.

**Explication :**

- `vaccination_ratio` est calculé en divisant le nombre total de vaccinations par `total_vaccinations_per100`.

### b. Progression des vaccinations par pays

La progression quotidienne des vaccinations a été calculée en utilisant une fenêtre de temps.



```

from pyspark.sql.window import Window
from pyspark.sql.functions import lag

window_spec = Window.partitionBy("COUNTRY").orderBy("DATE_UPDATED")
fact_table = fact_table.withColumn("progression_vaccinations", col("total_vaccinations") - lag("total_vaccinations").over(window_spec))

```

snappify.com

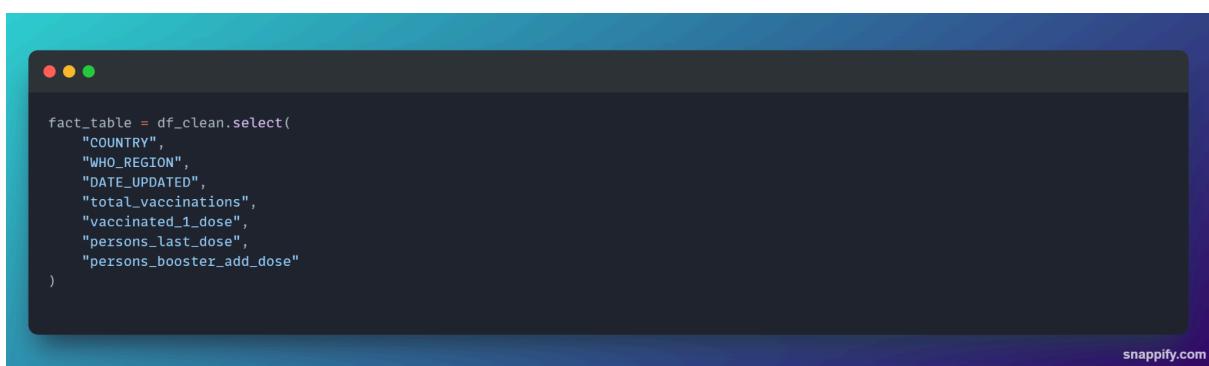
### Explication :

- La fonction `lag()` permet de récupérer la valeur précédente pour calculer la différence et ainsi obtenir la progression.

## 5. Création des tables Dimensions et Fait (Zone Gold)

### a. Création de la table Fait

La table **Fait** regroupe les mesures analytiques, comme le total des vaccinations et les ratios calculés.



```

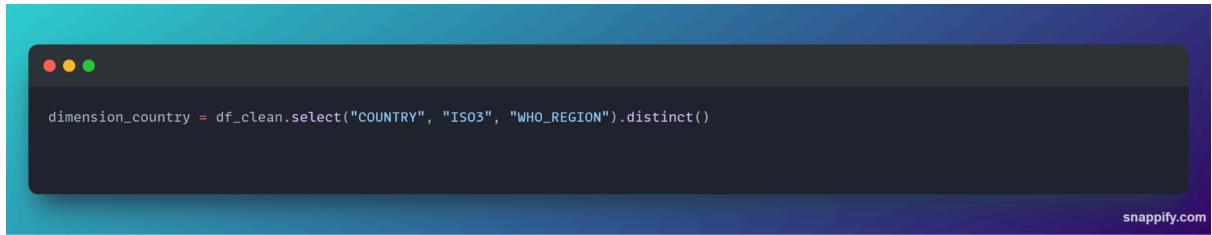
fact_table = df_clean.select(
    "COUNTRY",
    "WHO_REGION",
    "DATE_UPDATED",
    "total_vaccinations",
    "vaccinated_1_dose",
    "persons_last_dose",
    "persons_booster_add_dose"
)

```

snappify.com

### b. Création des tables Dimensions

**Dimension géographique :**

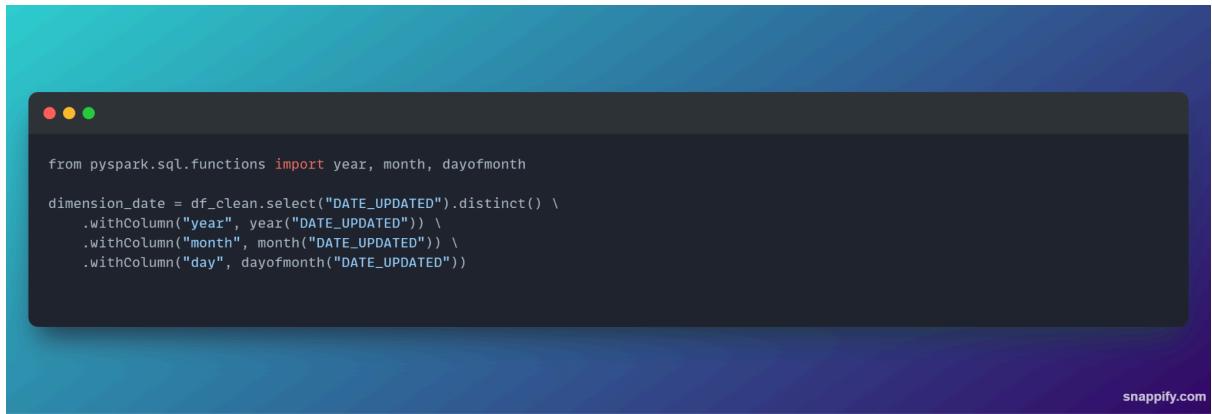


```
dimension_country = df_clean.select("COUNTRY", "ISO3", "WHO_REGION").distinct()
```

snappyf.com

A screenshot of a Jupyter Notebook cell. The cell contains Python code using PySpark's SQL API to select columns 'COUNTRY', 'ISO3', and 'WHO\_REGION' from a DataFrame 'df\_clean' and then call the 'distinct()' method to remove duplicates. The code is highlighted in purple. The cell has three colored window control buttons (red, yellow, green) at the top left. The background of the notebook interface is dark blue.

**Dimension temporelle :**



```
from pyspark.sql.functions import year, month, dayofmonth

dimension_date = df_clean.select("DATE_UPDATED").distinct() \
    .withColumn("year", year("DATE_UPDATED")) \
    .withColumn("month", month("DATE_UPDATED")) \
    .withColumn("day", dayofmonth("DATE_UPDATED"))
```

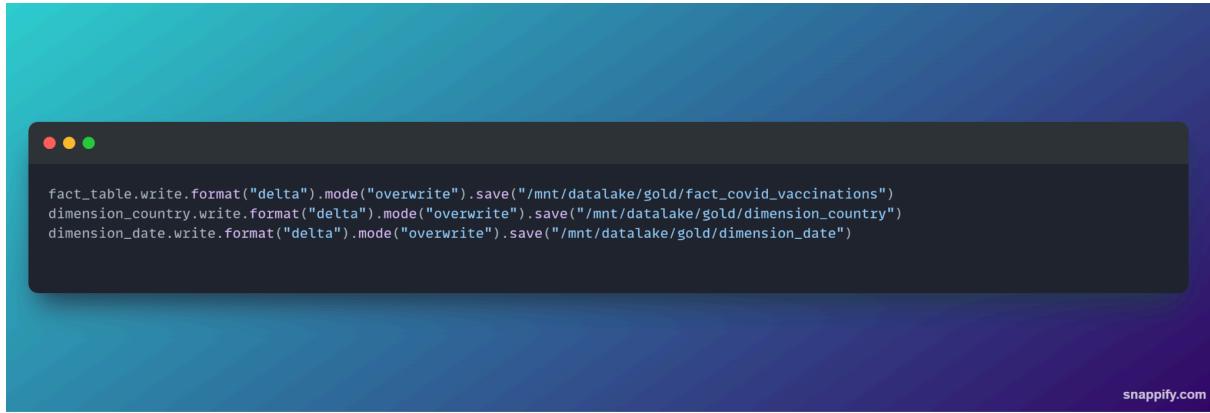
snappyf.com

A screenshot of a Jupyter Notebook cell. The cell contains Python code using PySpark's SQL API to select the 'DATE\_UPDATED' column from a DataFrame 'df\_clean', then call the 'distinct()' method, and finally add three new columns: 'year', 'month', and 'day' using the corresponding functions from the 'pyspark.sql.functions' module. The code is highlighted in purple. The cell has three colored window control buttons (red, yellow, green) at the top left. The background of the notebook interface is dark blue.

## 6. Sauvegarde des données

Les données transformées et enrichies ont été sauvegardées dans les zones correspondantes :

- **Zone Bronze** : données brutes.
- **Zone Silver** : données nettoyées.
- **Zone Gold** : tables Fait et Dimensions.



```

fact_table.write.format("delta").mode("overwrite").save("/mnt/datalake/gold/fact_covid_vaccinations")
dimension_country.write.format("delta").mode("overwrite").save("/mnt/datalake/gold/dimension_country")
dimension_date.write.format("delta").mode("overwrite").save("/mnt/datalake/gold/dimension_date")

```

snappify.com

## Résumé des transformations :

Zone	Transformations réalisées
Bronze	<b>Chargement des données brutes sans aucune transformation.</b>
Silver	<b>Suppression des colonnes, gestion des valeurs manquantes, conversion des types de données.</b>
Gold	<b>Création des tables Fait et Dimensions, enrichissements (ratios, progression), et agrégations.</b>



## Présentation des Visualisations

### 1. Progression des vaccinations par pays

Cette visualisation présente l'évolution cumulée des vaccinations pour les 5 premiers pays sélectionnés.

- **Objectif** : Comparer l'avancement des vaccinations dans différents pays au fil du temps.

- **Détails :**

- Les dates ont été normalisées pour assurer une comparaison cohérente entre les pays.
- Les courbes montrent la tendance de progression des vaccinations cumulées.

```
# Agrégation des données par pays et date
vaccination_progress = fact_table.groupby("COUNTRY", "DATE_UPDATED") \
    .agg({"total_vaccinations": "sum"}) \
    .withColumnRenamed("sum(total_vaccinations)", "total_vaccinations_sum")

# Conversion en Pandas DataFrame
vaccination_df = vaccination_progress.toPandas()

# Normalisation des dates pour chaque pays
all_dates = pd.date_range(start=vaccination_df["DATE_UPDATED"].min(), end=vaccination_df["DATE_UPDATED"].max())

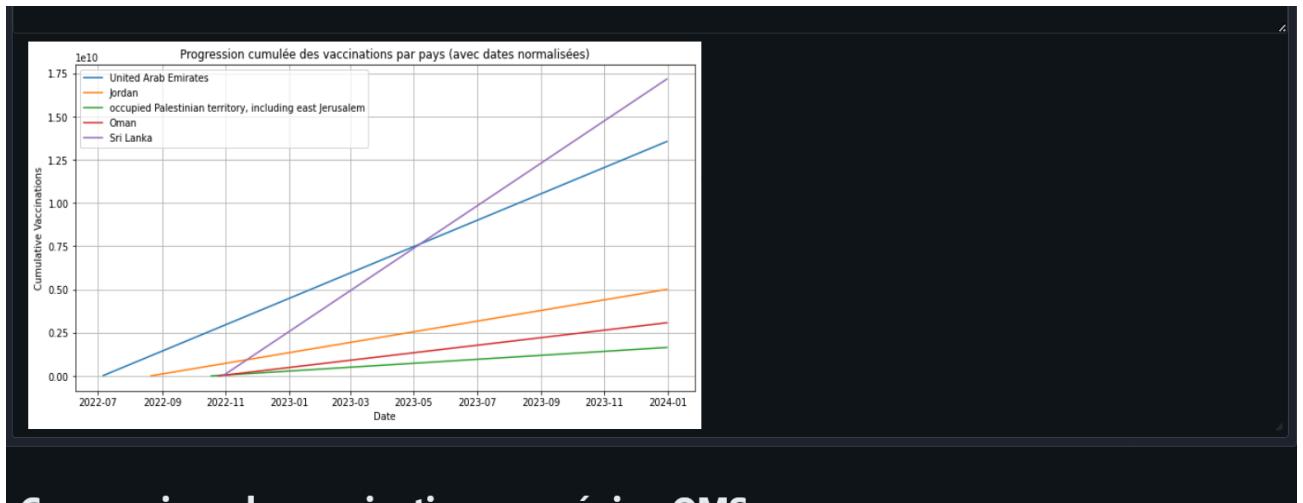
# Création d'une table normalisée
normalized_data = []
for country in vaccination_df["COUNTRY"].unique():
    country_data = vaccination_df[vaccination_df["COUNTRY"] == country]
    country_data = country_data.set_index("DATE_UPDATED").reindex(all_dates)
    country_data["total_vaccinations_sum"] = country_data["total_vaccinations_sum"].interpolate()
    country_data["COUNTRY"] = country
    normalized_data.append(country_data)

normalized_df = pd.concat(normalized_data).reset_index().rename(columns={"index": "DATE_UPDATED"})

# Visualisation
plt.figure(figsize=(12, 6))
for country in normalized_df["COUNTRY"].unique():
    plt.plot(normalized_df[normalized_df["COUNTRY"] == country]["DATE_UPDATED"],
             normalized_df[normalized_df["COUNTRY"] == country]["total_vaccinations_sum"], label=country)

plt.title("Progression cumulée des vaccinations par pays (avec dates normalisées)")
plt.xlabel("Date")
plt.ylabel("Cumulative Vaccinations")
plt.legend()
plt.grid(True)
plt.show()
```

snappyf.com



## Comparaison des vaccinations par région OMS

### 2. Comparaison des vaccinations par région OMS

Cette visualisation utilise un **graphique à barres** pour comparer le nombre total de vaccinations par **région OMS**.

- **Objectif :** Identifier les régions ayant enregistré le plus grand nombre de vaccinations.
- **Détails :**
  - Les barres sont triées par ordre décroissant pour faciliter l'analyse visuelle.
  - Chaque barre possède une annotation affichant la valeur exacte des vaccinations.

```

# Agrégation des vaccinations par région OMS
region_aggregation = fact_table.groupby("WHO_REGION") \
    .agg({"total_vaccinations": "sum"}) \
    .withColumnRenamed("sum(total_vaccinations)", "total_vaccinations_sum")

# Conversion en Pandas DataFrame et tri décroissant
region_df = region_aggregation.toPandas().sort_values(by="total_vaccinations_sum", ascending=False)

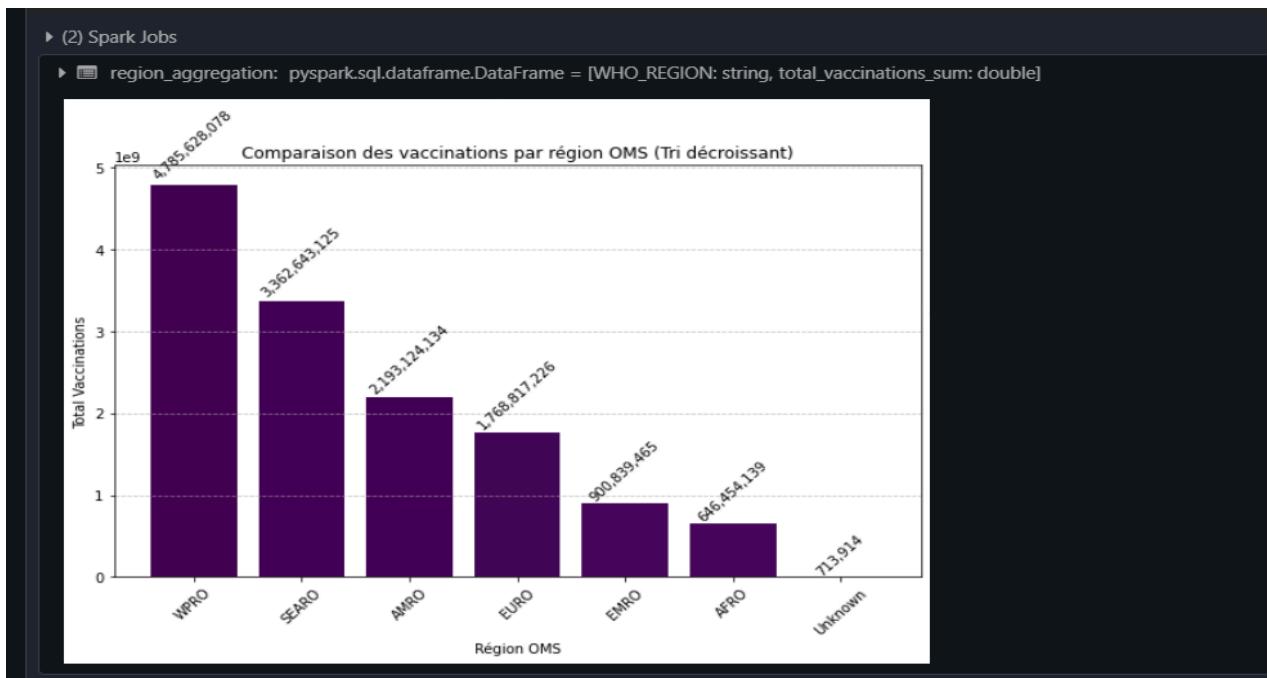
# Visualisation
plt.figure(figsize=(10, 6))
bars = plt.bar(region_df["WHO_REGION"], region_df["total_vaccinations_sum"], color=plt.cm.tab20.colors)

# Annotations des barres
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval), ha='center', va='bottom', fontsize=10)

plt.title("Comparaison des vaccinations par région OMS")
plt.ylabel("Total Vaccinations")
plt.xlabel("Région OMS")
plt.xticks(rotation=45)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

```

snappyf.com



### 3. Analyse temporelle des vaccinations par année

Cette visualisation illustre l'évolution des **vaccinations cumulées** par année à l'aide d'une courbe linéaire.

- **Objectif :** Observer la tendance globale des vaccinations à l'échelle mondiale.
- **Détails :**
  - Les points clés sont annotés pour mettre en évidence les valeurs exactes.

```

# Agrégation des données par année
yearly_vaccinations = fact_table.withColumn("year", year("DATE_UPDATED")) \
    .groupBy("year").agg({"total_vaccinations": "sum"}) \
    .withColumnRenamed("sum(total_vaccinations)", "total_vaccinations_sum")

# Conversion en Pandas DataFrame
yearly_df = yearly_vaccinations.toPandas()

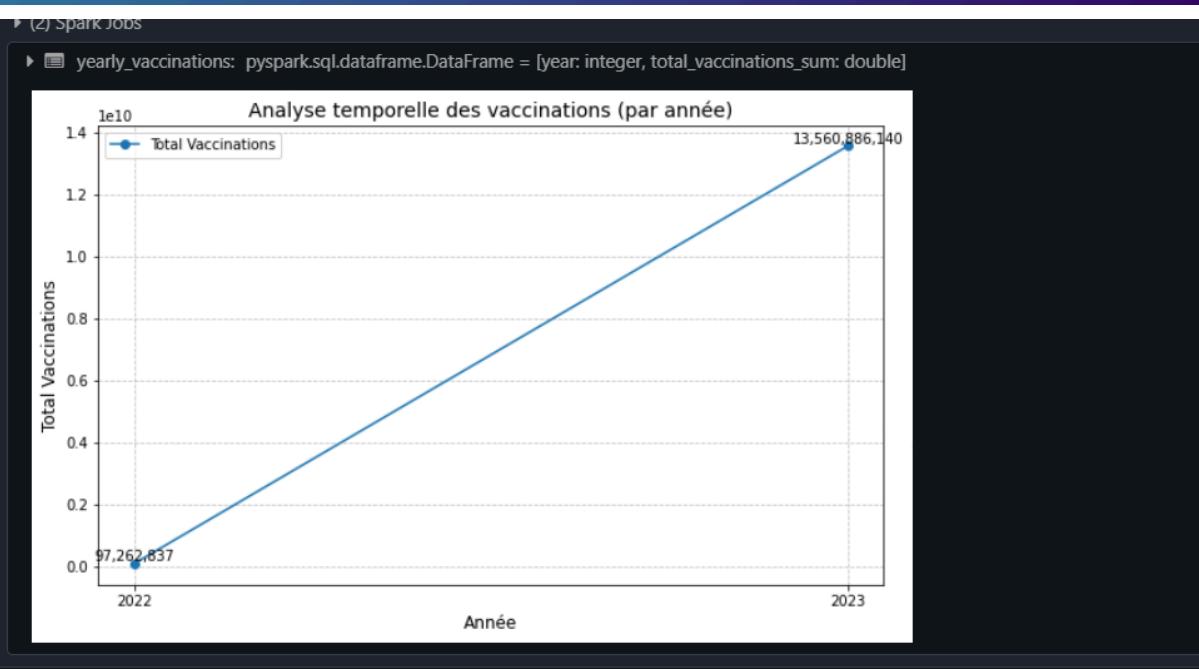
# Visualisation avec annotations
plt.figure(figsize=(10, 6))
plt.plot(yearly_df["year"], yearly_df["total_vaccinations_sum"], marker='o', linestyle='-', color='tab:blue', label='Total Vaccinations')

# Ajout des annotations sur les points
for x, y in zip(yearly_df["year"], yearly_df["total_vaccinations_sum"]):
    plt.text(x, y, f"{y:.0f}", ha='center', va='bottom', fontsize=10)

plt.title("Analyse temporelle des vaccinations (par année)", fontsize=14)
plt.xlabel("Année", fontsize=12)
plt.ylabel("Total Vaccinations", fontsize=12)
plt.grid(True, linestyle="--", alpha=0.7)
plt.legend()
plt.show()

```

snappyf.com



## Résumé des Visualisations

Visualisation	Objectif	Type de Graphique
Progression des vaccinations par pays	Comparer l'évolution par pays	Courbe linéaire
Comparaison des vaccinations par région OMS	Comparer les régions OMS	Graphique à barres
Analyse temporelle des	Visualiser la progression	Courbe linéaire avec points

vaccinations (par année)	annuelle globale	
--------------------------	------------------	--

## Conclusion

Ce projet de traitement et d'analyse des données de vaccination mondiale a permis de mettre en place une **architecture Lakehouse** robuste et performante, articulée autour des zones **Bronze**, **Silver** et **Gold**. Grâce à l'utilisation de **Databricks** et **Apache Spark**, nous avons pu assurer une gestion efficace des données, depuis leur ingestion jusqu'à leur modélisation et visualisation.

### Résultats atteints :

1. **Ingestion et stockage :**
  - Les données brutes ont été récupérées sous forme de fichiers CSV et stockées dans la **zone Bronze** pour assurer la traçabilité et la reproductibilité.
2. **Transformation et nettoyage :**
  - Les données ont été traitées dans la **zone Silver**, où des étapes telles que la suppression des colonnes inutiles, la gestion des valeurs manquantes et la conversion des types de données ont été appliquées.
3. **Modélisation et enrichissement :**
  - Dans la **zone Gold**, un **modèle en étoile** a été créé avec une table **Fait** et des tables **Dimensions** pour faciliter les analyses et les agrégations.
4. **Visualisations :**
  - Plusieurs visualisations ont été réalisées afin de mieux comprendre la progression des vaccinations :
    - **Progression cumulée des vaccinations par pays.**
    - **Comparaison des vaccinations par région OMS.**
    - **Analyse temporelle annuelle des vaccinations.**

### Enseignements tirés :

Ce projet a démontré l'importance d'une **architecture structurée** pour la gestion des données à grande échelle. L'approche Lakehouse permet de combiner les avantages des Data Lakes et des Data Warehouses, assurant ainsi une flexibilité pour le stockage des données brutes et une performance optimisée pour l'analyse.

Ce projet a permis d'acquérir des compétences pratiques dans la mise en œuvre d'une architecture de type Lakehouse et de réaliser des analyses exploitables pour mieux comprendre les tendances mondiales des vaccinations.