

Rapport projet

Membres du groupe

- Lucas TREILLE
- Asâad MEHIDI
- Kévin PARRY
- Mathis BOULAIS

Le projet

Ce projet vise à analyser les relations entre trois variables principales collectées par pays sur plusieurs années :

1. **PIB par habitant** : une mesure de la richesse économique moyenne dans chaque pays.
2. **Taux de suicide** : exprimé par un indicateur annuel, souvent calculé en nombre de suicides pour 100 000 habitants.
3. **Température moyenne annuelle** : un indicateur climatique, reflétant la moyenne des températures mesurées dans chaque pays.

Les données

PIB par habitant par pays

- **Source potentielle** : Banque Mondiale, FMI, OCDE.
- **Variables attendues** :
 - country : Nom du pays.
 - year : Année.
 - gdp_per_capita : PIB par habitant (en USD ou ajusté en parité de pouvoir d'achat).
- **Fréquence** : Année.
- **Format attendu** : CSV

Taux de suicide par pays

- **Source potentielle** : OMS (Organisation Mondiale de la Santé).
- **Variables attendues** :
 - country : Nom du pays.
 - year : Année.
 - suicide_rate : Nombre de suicides pour 100 000 habitants.
- **Fréquence** : Année.

Température moyenne annuelle par pays

- **Source potentielle** : Données climatiques (NOAA, World Bank Climate Data, Our World in Data).
- **Variables attendues** :
 - country : Nom du pays.
 - year : Année.
 - average_temperature : Température moyenne annuelle (en °C).
- **Fréquence** : Année.

Informations avant projet :

- Comme nous en avons discuté, je n'ai pas la possibilité d'exporter le fichier .dbc en raison d'un problème de droits. J'ai tenté à nouveau avant le rendu, mais cela n'a pas fonctionné. J'ai donc exporté toutes les données au format HTML et veillé à ce que tous les résultats soient visibles, afin que vous puissiez consulter les données et les résultats sans difficulté.
-
- Certaines fonctionnalités n'ont pas pu être implémentées (Celles en plus de la gestion et de l'analyse de données) en raison des limitations de la version Community, rendant leur intégration complexe, voire impossible, malgré plusieurs heures de recherche. Nous espérons que cela n'aura qu'un impact limité sur l'évaluation globale du projet.

L'objectif

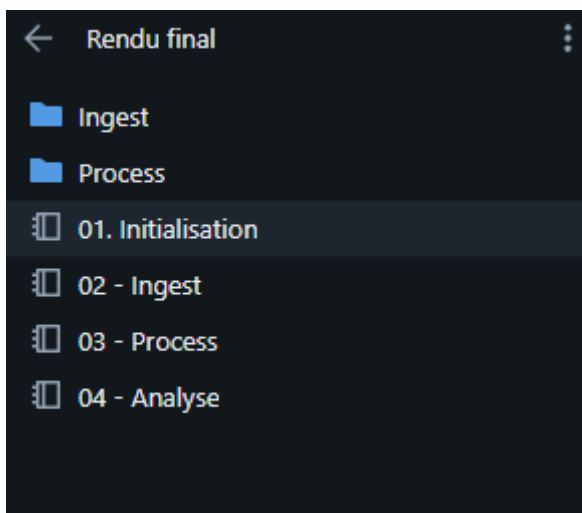
L'objectif principal est d'analyser si et comment le **PIB par habitant** et la **température moyenne annuelle** influencent le **taux de suicide** dans différents pays et sur plusieurs années.

Les traitements effectués

Tous les traitements ne sont volontairement pas affichés car cela serait trop volumineux, mais le reste des traitements nécessaire se trouve dans le .dbc

Structure du projet :

- Dossier 'Ingest' qui stocke nos données brutes
- Dossier 'Process' qui contient nos données en cours de transformation
- Initialisation : Configuration de l'environnement de travail.
- Ingest : Import des données sources.
- Process : Transformation des données.
- Analyse : Production des rapports et indicateurs.



01. Initialisation



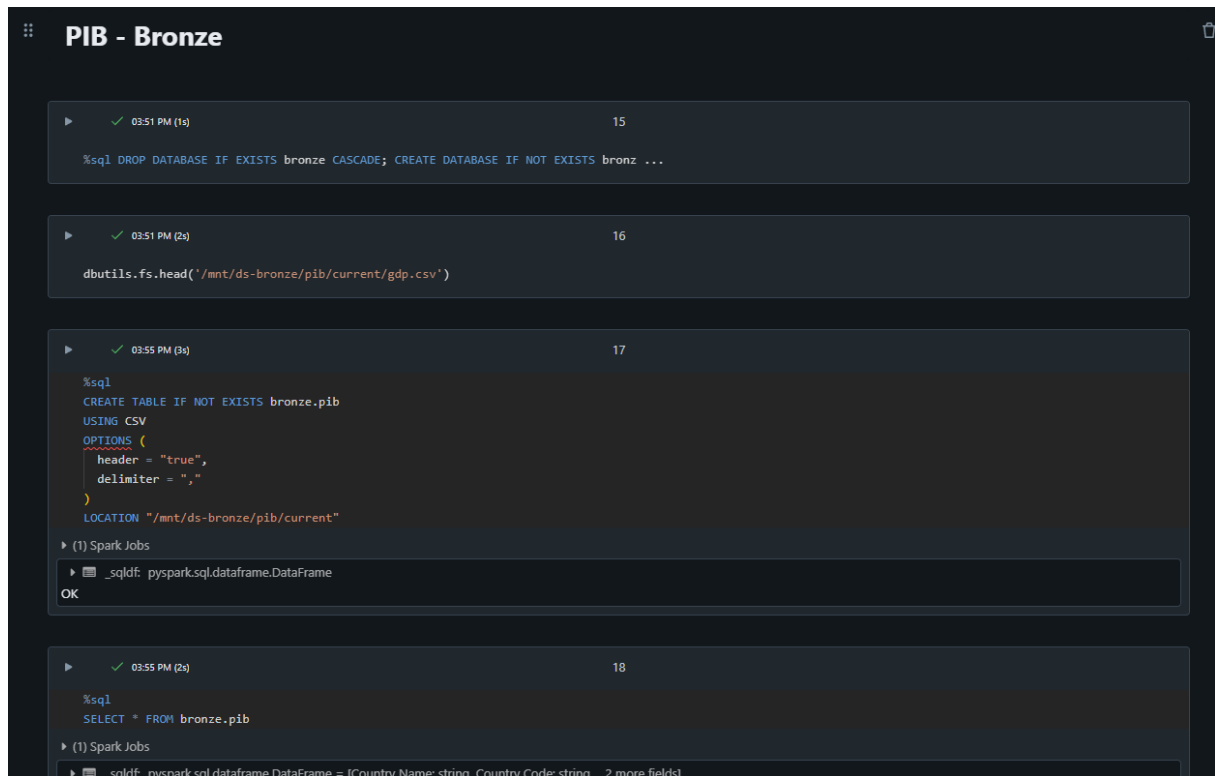
```
Bronze

▶ ✓ 03:51 PM (21s) 4 Python
try:    dbutils.fs.unmount("/mnt/ds-bronze") except Exception as e:    print(f ...

▶ ✓ 03:51 PM (21s) 5
dbutils.fs.mount(    source = "wasbs://ds-bronze@dlkefreixb56885.blob.core.window ...

▶ ✓ 03:51 PM (<1s) 6
dbutils.fs.ls('/mnt/ds-bronze/')
```

Ce code configure l'accès à un stockage Azure Blob pour la couche "Bronze" des données, on effectue cela pour Bronze/Silver/Gold.

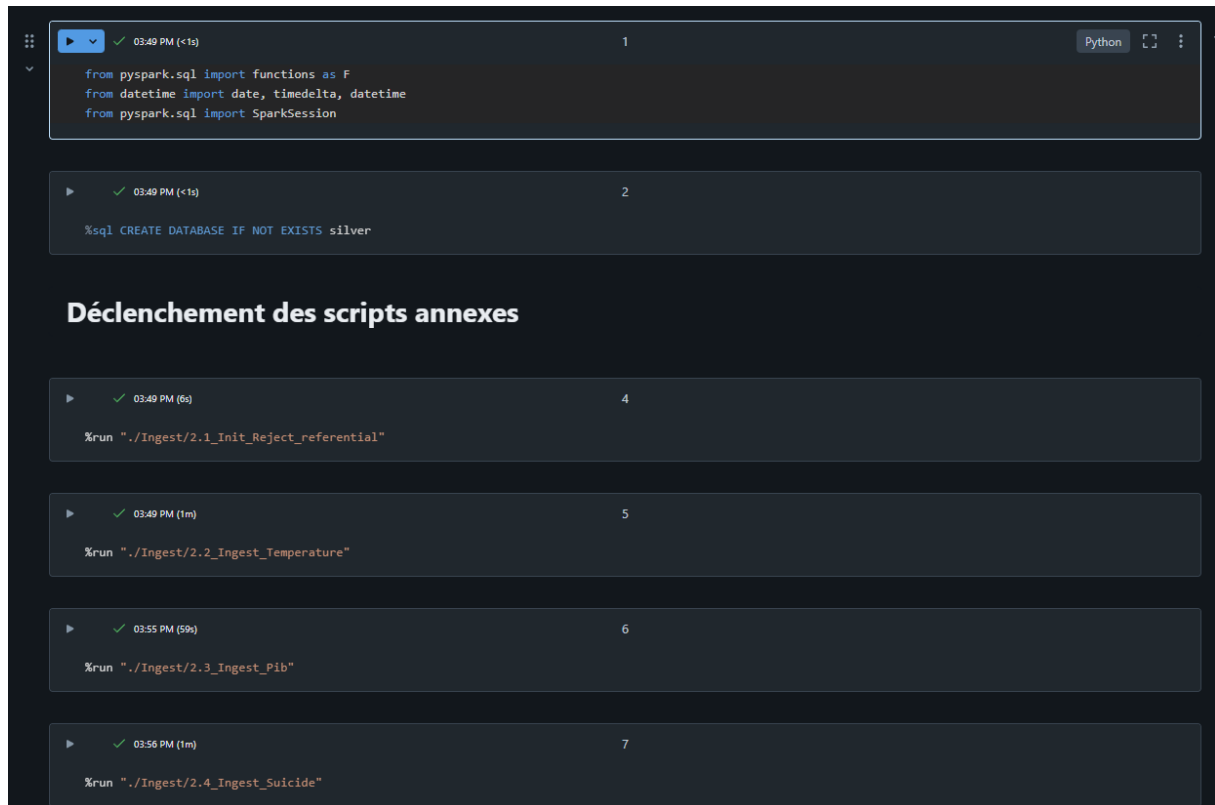


Dans cette séquence, on prépare la base de données pour les données PIB (Bronze) :

1. On crée une nouvelle base de données "bronze" en supprimant l'ancienne si elle existe
2. On vérifie le contenu du fichier CSV des données PIB
3. On crée une table `bronze.pib` en définissant :
 - La structure depuis le CSV
 - Un header (en-tête)
 - La virgule comme séparateur
4. On vérifie que les données sont bien chargées avec un `SELECT`

C'est la phase d'initialisation classique d'une couche Bronze dans un environnement Databricks. On fait pareil pour Temperature et Suicide.

02. Ingest



The screenshot displays a Jupyter Notebook with a dark theme. It contains several code cells, each with a status bar at the top indicating execution time and a success icon. The cells are numbered 1 through 7. Cell 1 is a Python cell importing Spark and datetime libraries. Cell 2 is a SQL cell creating a database. A section titled 'Déclenchement des scripts annexes' follows. Cells 4 through 7 are shell cells running scripts to initialize the reject reference, ingest temperature data, ingest PIB data, and ingest suicide data, respectively.

```
from pyspark.sql import functions as F
from datetime import date, timedelta, datetime
from pyspark.sql import SparkSession
```

```
%sql CREATE DATABASE IF NOT EXISTS silver
```

Déclenchement des scripts annexes

```
%run "../Ingest/2.1_Init_Reject_referential"
```

```
%run "../Ingest/2.2_Ingest_Temperature"
```

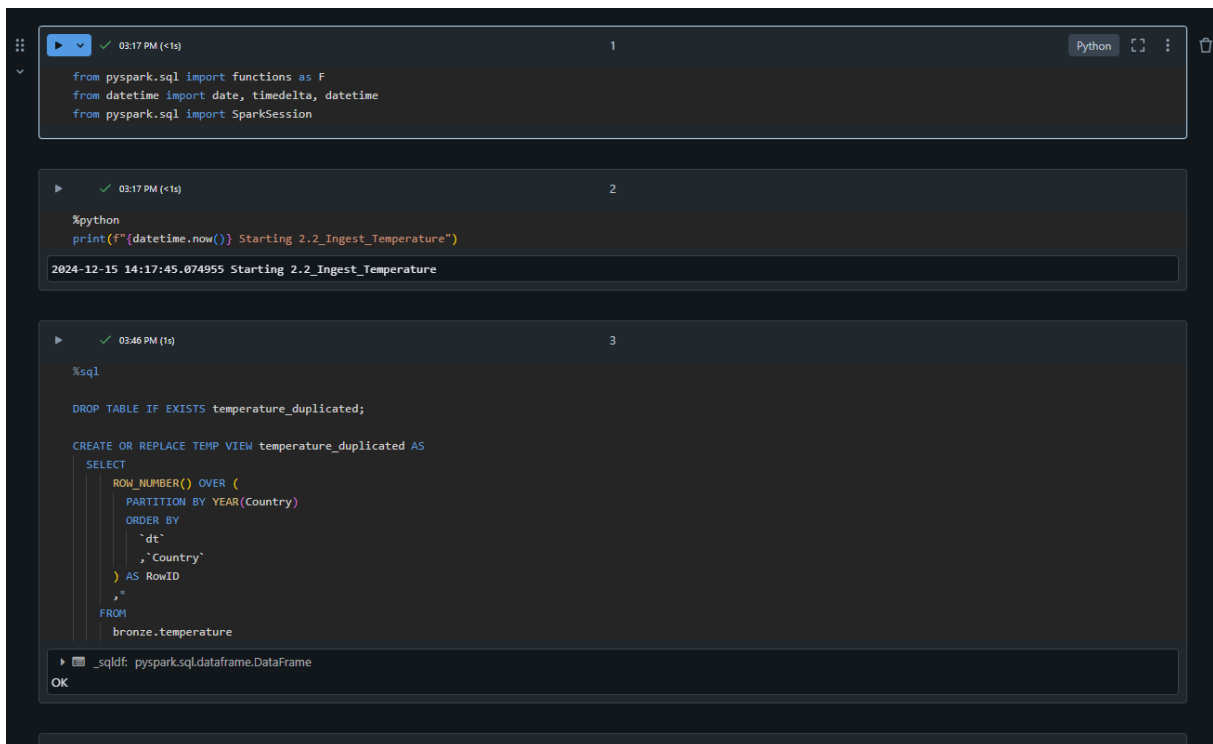
```
%run "../Ingest/2.3_Ingest_Pib"
```

```
%run "../Ingest/2.4_Ingest_Suicide"
```

Ce code prépare l'environnement de travail pour le traitement des données en :

1. Important les bibliothèques nécessaires pour Spark et la gestion des dates
2. Créant la base de données "silver" pour les données traitées
3. Exécutant une série de scripts d'ingestion dans l'ordre :
 - Initialisation du référentiel de rejet
 - Chargement des données de température
 - Chargement des données PIB
 - Chargement des données de suicide

02.2 Description Ingest Temperature (Dossier INGEST)



```
from pyspark.sql import functions as F
from datetime import date, timedelta, datetime
from pyspark.sql import SparkSession

%python
print(f"{datetime.now()} Starting 2.2_Ingest_Temperature")

2024-12-15 14:17:45.074955 Starting 2.2_Ingest_Temperature

%sql

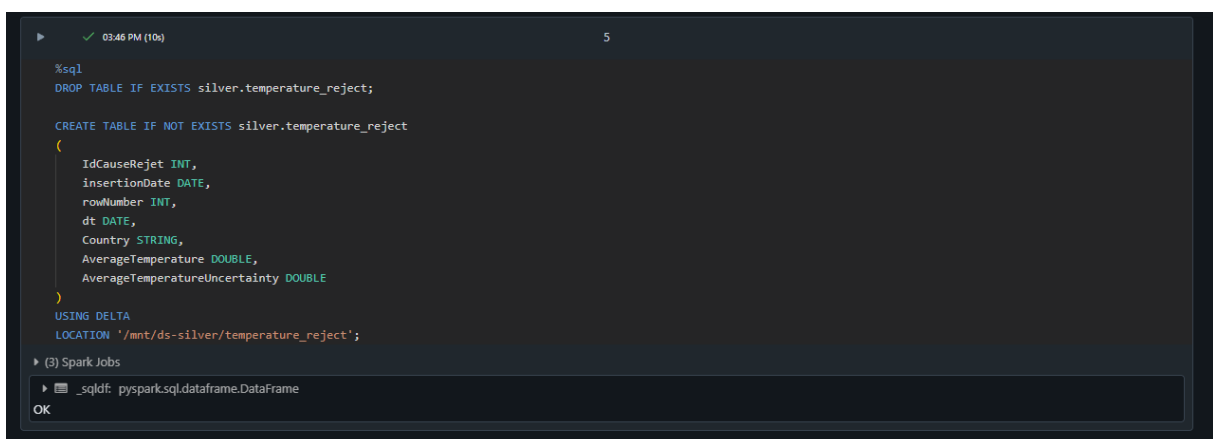
DROP TABLE IF EXISTS temperature_duplicated;

CREATE OR REPLACE TEMP VIEW temperature_duplicated AS
SELECT
  ROW_NUMBER() OVER (
    PARTITION BY YEAR(Country)
    ORDER BY
      `dt`
      , `Country`
  ) AS RowID
  , *
FROM
  bronze.temperature
```

OK

Ce code initialise le traitement des données de température en :

1. Important les bibliothèques nécessaires
2. Enregistrant l'heure de début du traitement
3. Créant une vue temporaire pour détecter les doublons dans les données de température par pays et année



```
%sql
DROP TABLE IF EXISTS silver.temperature_reject;

CREATE TABLE IF NOT EXISTS silver.temperature_reject
(
  IdCauseRejet INT,
  insertionDate DATE,
  rowNumber INT,
  dt DATE,
  Country STRING,
  AverageTemperature DOUBLE,
  AverageTemperatureUncertainty DOUBLE
)
USING DELTA
LOCATION '/mnt/ds-silver/temperature_reject';
```

OK

Ce code met en place la table qui stockera les données de température rejetées avec leur cause et date de rejet, en utilisant le format Delta Lake pour garantir la traçabilité. (Voir Ingest 2.1)

```
▶ 03:46 PM (23s)

%sql

TRUNCATE TABLE silver.temperature_reject;

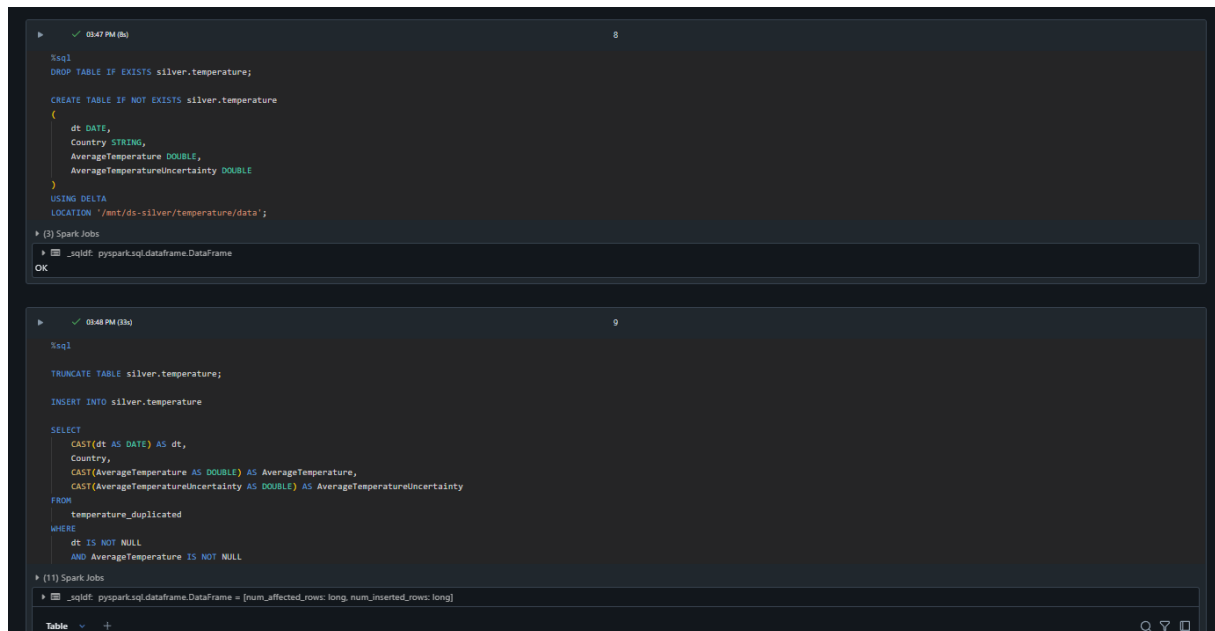
INSERT INTO silver.temperature_reject
WITH
/*
  IdReject : 1002
  Date manquante
*/
IdReject_1002 AS (
  SELECT
    1002 AS IdCauseRejet,
    CURRENT_DATE() AS insertionDate,
    RowID AS rowNumber,
    dt,
    Country,
    AverageTemperature,
    AverageTemperatureUncertainty
  FROM
    temperature_duplicated
  WHERE
    dt IS NULL
),

/*
  IdReject : 1001
  Temperature manquante
*/
IdReject_1001 AS (
  SELECT
    1001 AS IdCauseRejet,
    CURRENT_DATE() AS insertionDate,
    RowID AS rowNumber,
    dt,
    Country,
    AverageTemperature,
    AverageTemperatureUncertainty
  FROM
    temperature_duplicated
  WHERE
    AverageTemperature IS NULL
)

SELECT * FROM IdReject_1002
UNION ALL
SELECT * FROM IdReject_1001

▶ (8) Spark Jobs
```

Ce code identifie et stocke les données de température problématiques, en séparant les rejets selon leur cause : dates manquantes (1002) ou températures manquantes (1001).



The image shows two screenshots of a Jupyter Notebook interface. The top screenshot, labeled '8', shows a SQL cell with the following code:

```
%sql
DROP TABLE IF EXISTS silver.temperature;

CREATE TABLE IF NOT EXISTS silver.temperature
(
  dt DATE,
  Country STRING,
  AverageTemperature DOUBLE,
  AverageTemperatureUncertainty DOUBLE
)
USING DELTA
LOCATION '/mnt/ds-silver/temperature/data';
```

 Below the code, it says '(3) Spark Jobs' and shows a status bar with 'OK'. The bottom screenshot, labeled '9', shows a SQL cell with the following code:

```
%sql

TRUNCATE TABLE silver.temperature;

INSERT INTO silver.temperature

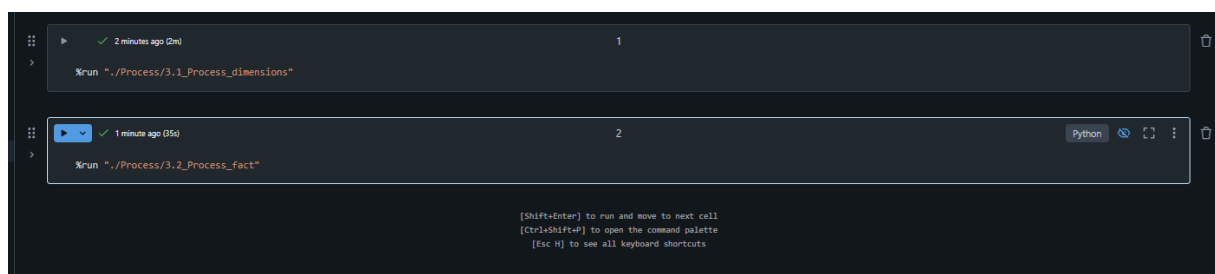
SELECT
  CAST(dt AS DATE) AS dt,
  Country,
  CAST(AverageTemperature AS DOUBLE) AS AverageTemperature,
  CAST(AverageTemperatureUncertainty AS DOUBLE) AS AverageTemperatureUncertainty
FROM
  temperature_duplicated
WHERE
  dt IS NOT NULL
  AND AverageTemperature IS NOT NULL
```

 Below the code, it says '(11) Spark Jobs' and shows a status bar with 'OK'. The status bar also displays 'Table' and a plus sign.

Ce code crée puis remplit la table finale des températures en format Delta, en ne gardant que les données valides et en s'assurant que les types de données sont corrects (dates et nombres).

On a le même processus pour le 2.3 -> PIB et le 2.4 -> Suicide

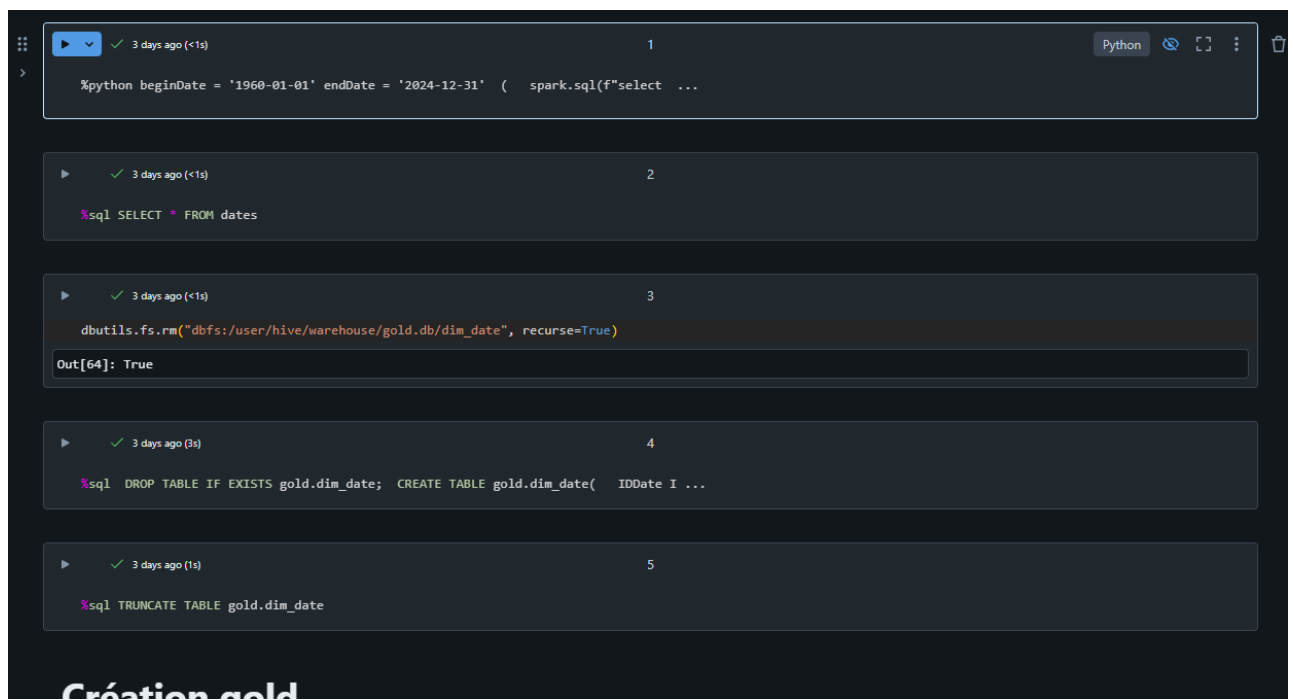
03. Initialisation



The image shows a Jupyter Notebook interface with two cells. The first cell, labeled '1', contains the command `%run ../Process/3.1_Process_dimensions`. The second cell, labeled '2', contains the command `%run ../Process/3.2_Process_fact`. The notebook interface includes a toolbar with icons for running, saving, and other actions. At the bottom, there are instructions: `[Shift+Enter]` to run and move to next cell, `[Ctrl+Shift+P]` to open the command palette, and `[Esc H]` to see all keyboard shortcuts.

Ce code initialise le NoteBook 03 et déclenche le traitement

3.1 Process dimensions



```
%python beginDate = '1960-01-01' endDate = '2024-12-31' ( spark.sql(f"select ...

%sql SELECT * FROM dates

dbutils.fs.rm("dbfs:/user/hive/warehouse/gold.db/dim_date", recurse=True)
Out[64]: True

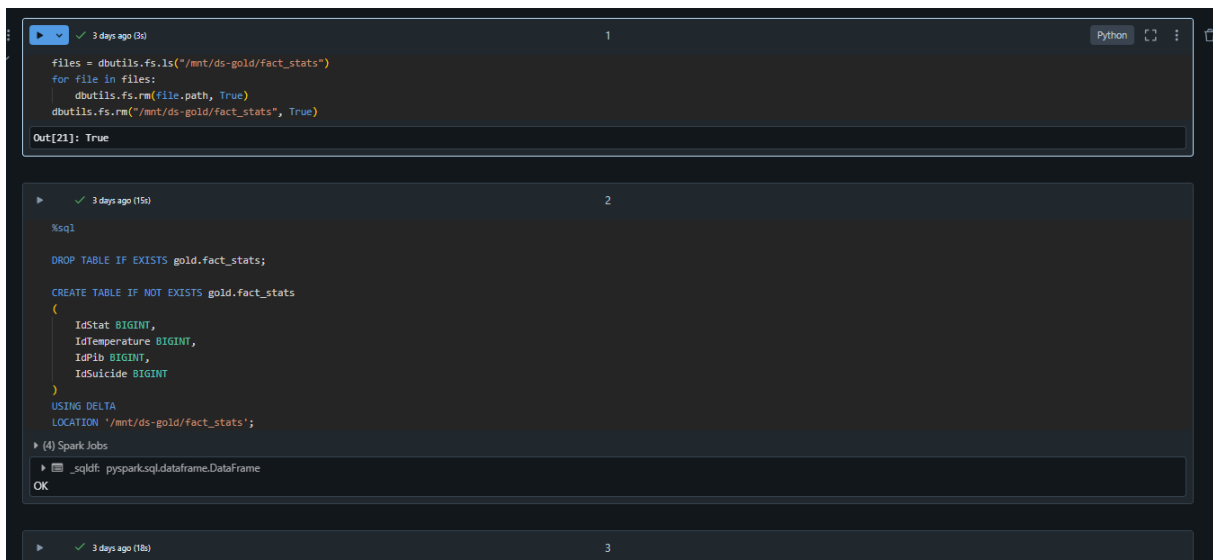
%sql DROP TABLE IF EXISTS gold.dim_date; CREATE TABLE gold.dim_date( IDDate I ...

%sql TRUNCATE TABLE gold.dim_date
```

Création gold

Ce code prépare la couche Gold en initialisant la table de dimensions temporelles : il définit la période d'étude, nettoie les anciennes données et crée la structure pour les nouvelles.

3.2 process fact



```
files = dbutils.fs.ls("/mnt/ds-gold/fact_stats")
for file in files:
    dbutils.fs.rm(file.path, True)
dbutils.fs.rm("/mnt/ds-gold/fact_stats", True)

Out[21]: True
```

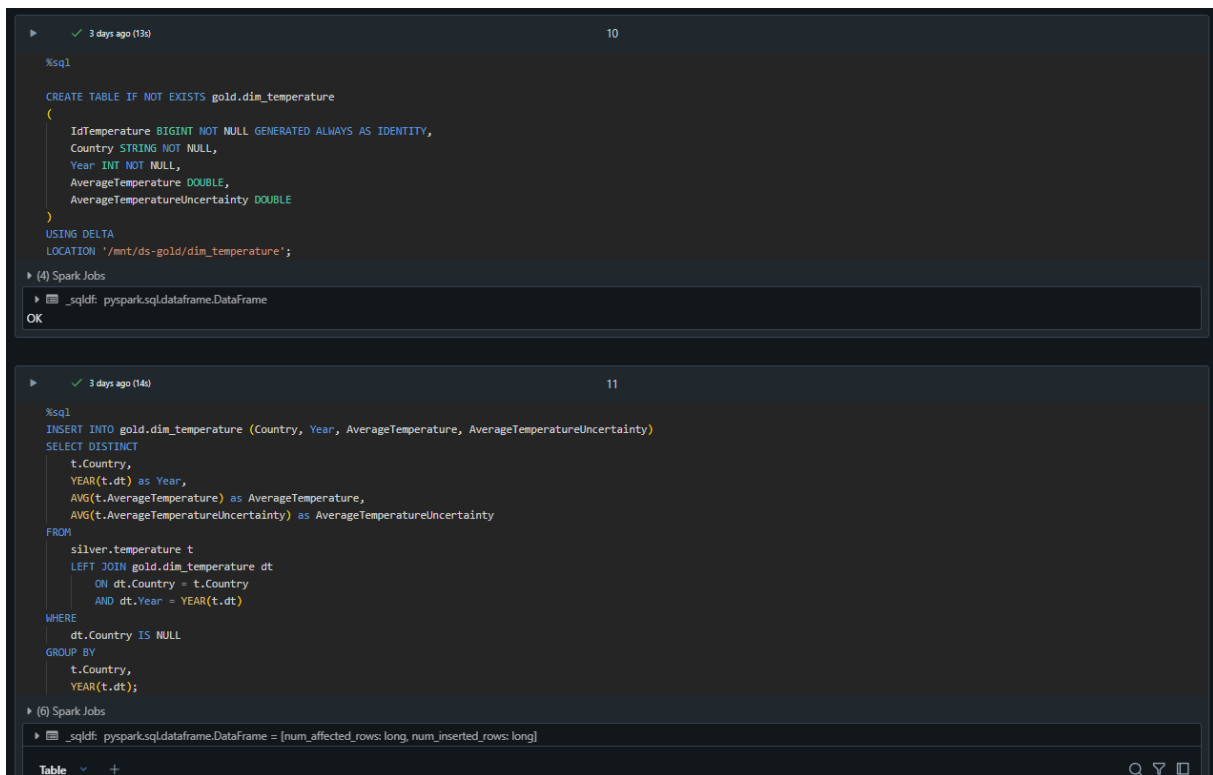
```
%sql

DROP TABLE IF EXISTS gold.fact_stats;

CREATE TABLE IF NOT EXISTS gold.fact_stats
(
    IdStat BIGINT,
    IdTemperature BIGINT,
    IdPib BIGINT,
    IdSuicide BIGINT
)
USING DELTA
LOCATION '/mnt/ds-gold/fact_stats';

(4) Spark Jobs
  _sqldf: pyspark.sql.dataframe.DataFrame
OK
```

On crée une table "gold.fact_stats" qui contient des mesures statistiques telles que la température minimale, maximale et moyenne, ainsi que le lever et le coucher du soleil, stockée dans un emplacement spécifique.



```
%sql

CREATE TABLE IF NOT EXISTS gold.dim_temperature
(
    IdTemperature BIGINT NOT NULL GENERATED ALWAYS AS IDENTITY,
    Country STRING NOT NULL,
    Year INT NOT NULL,
    AverageTemperature DOUBLE,
    AverageTemperatureUncertainty DOUBLE
)
USING DELTA
LOCATION '/mnt/ds-gold/dim_temperature';

(4) Spark Jobs
  _sqldf: pyspark.sql.dataframe.DataFrame
OK
```

```
%sql

INSERT INTO gold.dim_temperature (Country, Year, AverageTemperature, AverageTemperatureUncertainty)
SELECT DISTINCT
    t.Country,
    YEAR(t.dt) as Year,
    AVG(t.AverageTemperature) as AverageTemperature,
    AVG(t.AverageTemperatureUncertainty) as AverageTemperatureUncertainty
FROM
    silver.temperature t
LEFT JOIN gold.dim_temperature dt
    ON dt.Country = t.Country
    AND dt.Year = YEAR(t.dt)
WHERE
    dt.Country IS NULL
GROUP BY
    t.Country,
    YEAR(t.dt);

(6) Spark Jobs
  _sqldf: pyspark.sql.dataframe.DataFrame = [num_affected_rows: long, num_inserted_rows: long]

Table +
```

On crée une table "gold.dim_temperature" qui contient des données de température moyenne et leur incertitude par pays et par année, selon une structure de données raffinées.

On applique le même traitement pour les tables PUib et Suicide

```
3
Ksql
DELETE FROM gold.fact_stats;

INSERT INTO gold.fact_stats (
  IdStat,
  IdTemperature,
  IdPib,
  IdSuicide
)
WITH suicide_agg AS (
  SELECT
    country,
    year,
    SUM(SuicidesNo) AS TotalSuicides,
    SUM(Population) AS TotalPopulation
  FROM gold.dim_suicide
  GROUP BY country, year
)
SELECT
  ROW_NUMBER() OVER (ORDER BY t.Country, t.Year) AS IdStat,
  t.IdTemperature,
  p.IdPib,
  CAST(s.TotalSuicides AS BIGINT) AS IdSuicide
FROM
  gold.dim_temperature t
  INNER JOIN gold.dim_pib p
    ON t.Country = p.CountryName
    AND t.Year = p.Year
  INNER JOIN suicide_agg s
    ON t.Country = s.country
    AND t.Year = s.year
WHERE
  t.Year BETWEEN 1985 AND 2013
ORDER BY
  t.Country,
  t.Year;
```

▶ (10) Spark Jobs

▶ _sqlid: pyspark.sql.dataframe.DataFrame = [num_affected_rows: long, num_inserted_rows: long]

Cette partie du projet utilise une modélisation dimensionnelle des données avec deux types de tables : les tables de faits et les tables de dimensions.

La table "gold.fact_stats" est une table de faits qui contient les mesures clés, comme la température ou les heures de lever/coucher du soleil. Ces mesures sont généralement numériques et peuvent être agrégées.

Les autres tables, comme "gold.dim_suicide" ou "gold.dim_temperature", sont des tables de dimensions. Elles apportent du contexte aux mesures des faits, comme les informations de pays et d'année.

Le script SQL effectue des jointures entre la table de faits et les tables de dimensions pour pouvoir analyser les mesures selon différents angles, offrant ainsi une vue détaillée et flexible des données du projet.

Cette approche de modélisation facilite l'analyse des données en permettant d'explorer les mesures clés dans leur contexte, aidant ainsi à mieux comprendre les informations du projet.

4. Analyse

```
Graphiques détaillés - température - suicide - pib

df_global = spark.sql("""
SELECT
    Year,
    AVG(AverageTemperature) as GlobalTemp,
    AVG(GDPValue) as GlobalGDP,
    AVG(SuicideRate) as GlobalSuicideRate,
    SUM(TotalSuicides) as TotalSuicides,
    SUM(TotalPopulation) as TotalPopulation
FROM gold.fact_country_stats
GROUP BY Year
ORDER BY Year
""")
pdf_global = df_global.toPandas()

fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(12, 15))

ax1.plot(pdf_global['Year'], pdf_global['GlobalTemp'], color='red')
ax1.set_title('Température moyenne mondiale')
ax1.set_ylabel('Température (°C)')
ax1.grid(True)

ax2.plot(pdf_global['Year'], pdf_global['GlobalGDP'] / 1e12, color='blue')
ax2.set_title('PIB moyen mondial')
ax2.set_ylabel('PIB (en trillions)')
ax2.grid(True)

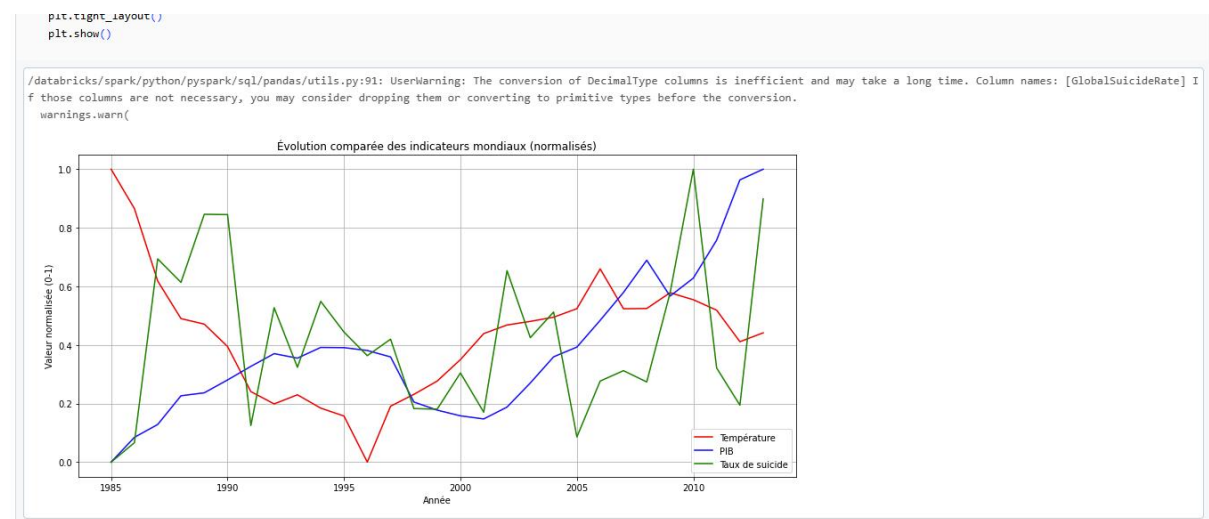
ax3.plot(pdf_global['Year'], pdf_global['GlobalSuicideRate'], color='green')
ax3.set_title('Taux de suicide moyen mondial')
ax3.set_ylabel('Pour 100,000 habitants')
ax3.grid(True)

plt.tight_layout()
plt.show()

(5) Spark Jobs
df_global: pyspark.sql.dataframe.DataFrame = [Year: integer, GlobalTemp: double ... 4 more fields]
```

Ce script Python crée des graphiques à partir de données sur la température, le suicide et le PIB, en utilisant PySpark pour récupérer les données et pour la visualisation. Il affiche l'évolution de ces mesures au fil des années dans un graphique à plusieurs sous-graphes.

Exemple de graphique avec combinaison de courbe :



On retrouve dans le point 4, 5 analyse sur notre sujet principale, celle-ci peuvent être consulté dans le projet directement depuis le **04 – Analyse**

MDC

