

Scheduled_ETL_pipeline_Covid19_Data Report:

In this project, I designed and implemented a scheduled data pipeline to automate the extraction, transformation, and loading (ETL) of data from an external API into Google Drive, and I produced various analytical dashboards using Tableau. Also i created a Gradient Boosting model to add predictions on top of the analysis. The model will be deployed using Flask and will be available to users via API.

Key Technologies and Tools Used:

Apache Airflow: I leveraged Apache Airflow, an open-source orchestration tool, to create a flexible and scalable workflow for managing the data pipeline. Airflow allowed for task scheduling, error handling, and easy monitoring of data transfer processes.

Python 3: I utilized Python 3 as the primary programming language to develop custom scripts and data processing logic. Python's extensive libraries and ecosystem enabled efficient data manipulation and transformation.

Ubuntu: The project ran on an Ubuntu-based environment, providing a stable and secure platform for executing data pipeline tasks.

Google Drive API: I integrated the Google Drive API into the project to programmatically interact with Google Drive, facilitating the storage and organization of extracted data files.

Flask: Flask is a lightweight Python web framework used for building web applications and APIs with simplicity and flexibility.

Tableau: Tableau is a powerful data visualization and business intelligence tool that creates interactive, insightful visualizations from various data sources.

Project Workflow:

Data Extraction: I configured the pipeline to retrieve data from the external API at scheduled intervals. This ensured that the most up-to-date information was available for analysis.

Data Processing and Transformation: After data retrieval, I applied data cleaning and transformation operations using Python. This step included data validation, normalization, and enrichment to prepare the data for downstream consumption.

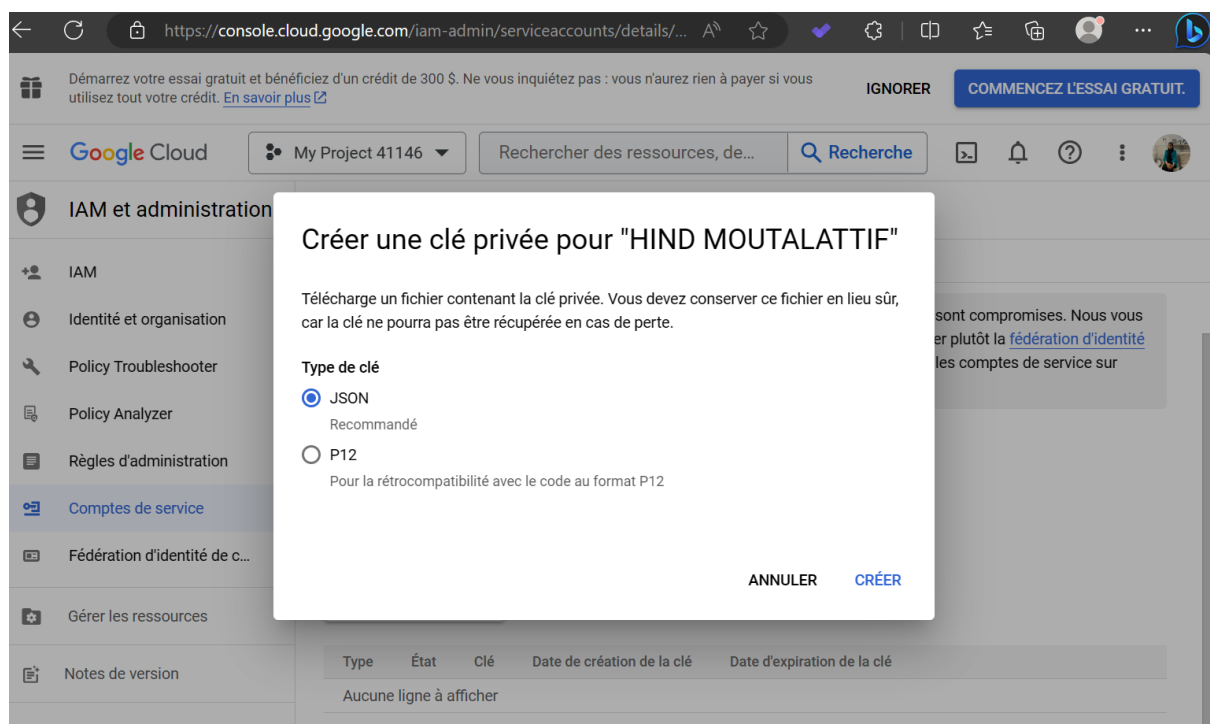
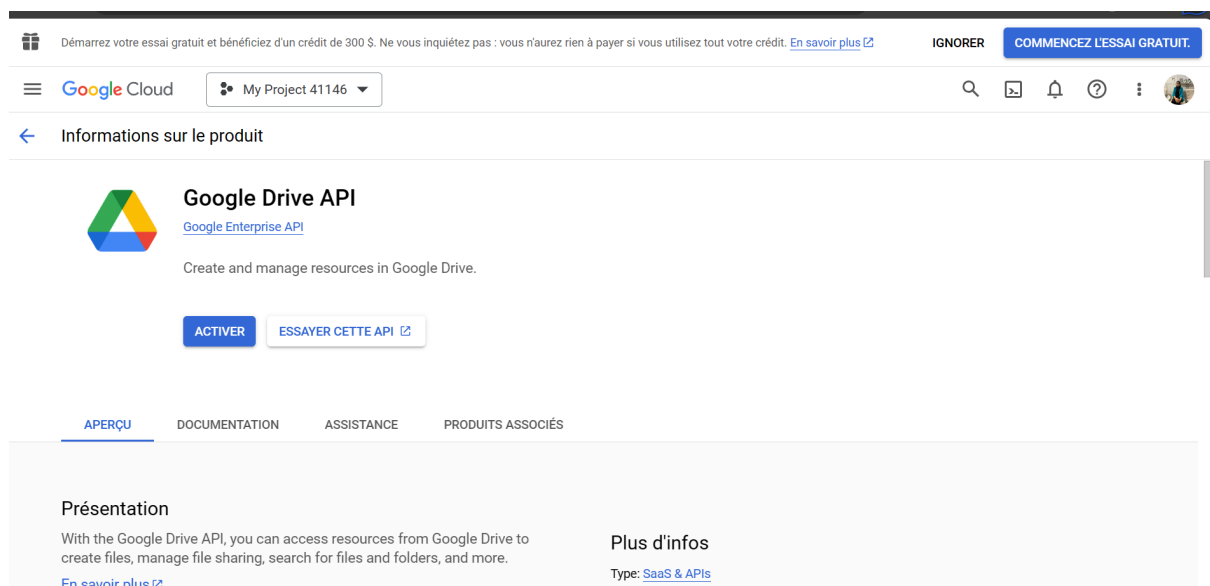
File Upload to Google Drive: I employed the Google Drive API to seamlessly upload the processed data files to designated folders in Google Drive. Each file was given a meaningful name and stored in an organized structure.

Tableau Dashboards : I tried to analyse cleaned data using Tableau by creating different dashboards.

Gradient Boosting Model and a simple Flask API : I created a Machine Learning model that can predict the daily increase in COVID-related death cases based on the number of people currently hospitalized and I created a simple API using Flask that will allow users to receive the output of this model.

1. Data pipeline:

Creation of Service_account_key_file in using google drive api to upload data to google drive:



Next , we try to install all the libraries to use upload our files to google drive:

```

root@DESKTOP-LB0766E:~$ python3 -m pip install --upgrade google-auth google-auth-httplib2 google-auth-oauthlib google-api-python-client
Defaulting to user installation because normal site-packages is not writeable
Collecting google-auth
  Downloading google_auth-2.23.2-py2.py3-none-any.whl (181 kB)
    _____ 182.0/182.0 KB 2.7 MB/s eta 0:00:00
Collecting google-auth-httplib2
  Downloading google_auth_httplib2-0.1.1-py2.py3-none-any.whl (9.3 kB)
Collecting google-auth-oauthlib
  Downloading google_auth_oauthlib-1.1.0-py2.py3-none-any.whl (19 kB)
Collecting google-api-python-client
  Downloading google_api_python_client-2.101.0-py2.py3-none-any.whl (12.3 MB)
    _____ 12.3/12.3 MB 17.6 MB/s eta 0:00:00
Collecting rsa<5,>=3.1.4
  Downloading rsa-4.9-py3-none-any.whl (34 kB)
Collecting cachetools<6.0,>=2.0.0

```

Next , we execute the covid_data.py file and run the airflow scheduler and webserver to see the web UI of airflow:

```

root@DESKTOP-LB0766E:~$ cd airflow
root@DESKTOP-LB0766E:~/airflow$ airflow scheduler

[2023-10-02T00:01:08.791+0100] {executor_loader.py:117} INFO - Loaded executor: SequentialExecutor
[2023-10-02 00:01:08 +0100] [239] [INFO] Starting gunicorn 21.2.0
[2023-10-02 00:01:08 +0100] [239] [INFO] Listening at: http://[::]:8793 (239)
[2023-10-02 00:01:08 +0100] [239] [INFO] Using worker: sync
[2023-10-02T00:01:08.955+0100] {scheduler_job_runner.py:798} INFO - Starting the scheduler
[2023-10-02T00:01:08.983+0100] {scheduler_job_runner.py:805} INFO - Processing each file at most -1 times
[2023-10-02T00:01:09.023+0100] {manager.py:166} INFO - Launched DagFileProcessorManager with pid: 241
[2023-10-02 00:01:09 +0100] [240] [INFO] Booting worker with pid: 240
[2023-10-02T00:01:09.033+0100] {scheduler_job_runner.py:1586} INFO - Adopting or re-running orphaned tasks for active dag runs
[2023-10-02T00:01:09.052+0100] {settings.py:60} INFO - Configured default timezone ezone('UTC')
[2023-10-02T00:01:09.097+0100] {scheduler_job_runner.py:1609} INFO - Marked 1 SchedulerJob instances as failed
[2023-10-02 00:01:09 +0100] [242] [INFO] Booting worker with pid: 242
[2023-10-02T00:01:09.190+0100] {manager.py:410} WARNING - Because we cannot use more than 1 thread (parsing processes = 2) when using sqlite. So we set parallelism to 1

```

Our dag is load_covid_data:

Do not use the **SequentialExecutor** in production. [Click here](#) for more information.

DAGs

All **3** Active **0** Paused **3** Running **0** Failed **1** Filter DAGs by tag

Search DAGs Auto-refresh

| DAG | Owner | Runs | Schedule | Last Run | Next Run |
|--|---------|--|----------|----------------------|----------------------|
| <input type="checkbox"/> first_airflow_dag | airflow | <div><div></div><div></div><div></div><div></div><div></div></div> | ***** | 2023-10-01, 00:33:00 | 2023-10-01, 23:00:00 |
| <input type="checkbox"/> load_covid_data | airflow | <div><div></div><div></div><div></div><div></div><div></div></div> | @daily | | 2023-09-30, 00:00:00 |
| <input type="checkbox"/> second_dag | | <div><div></div><div></div><div></div><div></div><div></div></div> | ***** | | 2023-10-01, 23:00:00 |

Showing 1-3 of 3 DAGs

And in this stage we could see that our three tasks are successfully executed:

DAG: load_covid_data Schedule: @daily Next Run: 2023-10-05, 00:00:00

Grid Graph Calendar Task Duration Task Tries Landing Times Gantt Details

<> Code Audit Log

06/10/2023 14:12:29 25 All Run Types All Run States Clear Filters Auto-refresh

Press **shift** + **/** for Shortcuts

deferred failed queued removed restarting running scheduled shutdown skipped success up_for_reschedule up_for_retry upstream_failed no_status

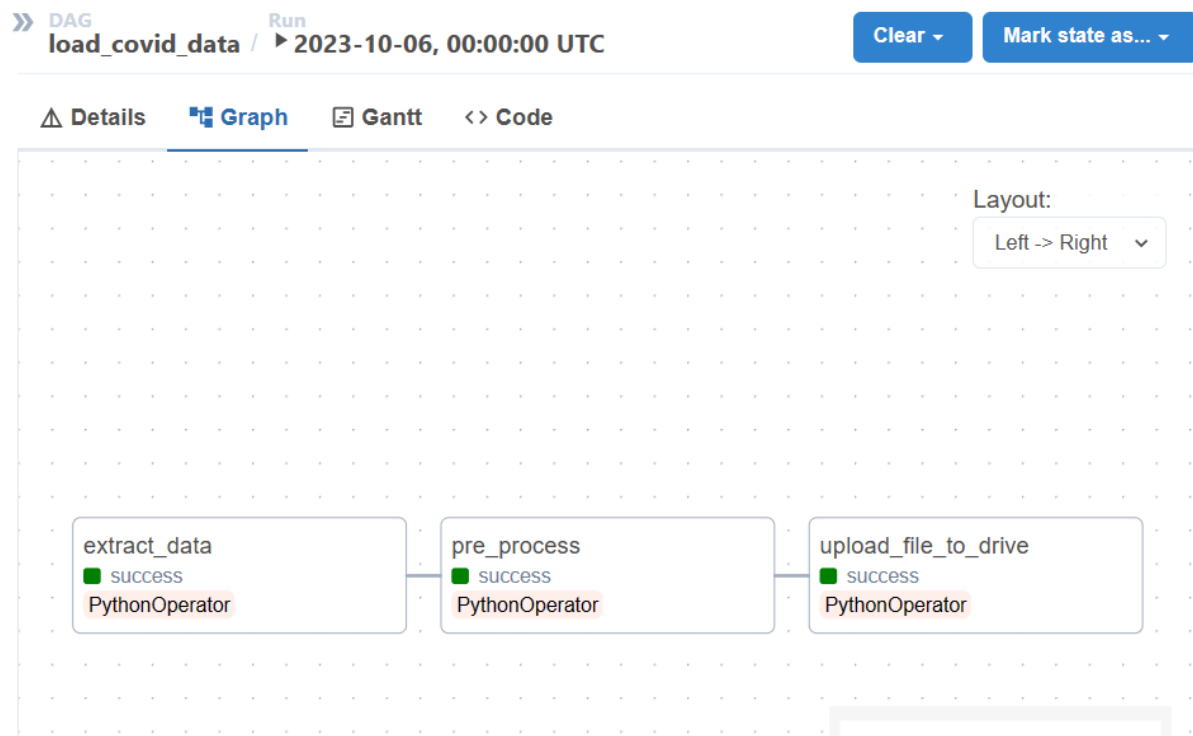
DAG load_covid_data

Details Graph Gantt <> Code

DAG Runs Summary

| | |
|----------------------|--------------------------|
| Total Runs Displayed | 19 |
| Total success | 4 |
| Total failed | 15 |
| First Run Start | 2023-10-01, 23:04:53 UTC |
| Last Run Start | 2023-10-06 14:14:20 UTC |

This is the Workflow of our scheduled ETL:



2. Tableau Dashboards

First we connect Tableau to Google Drive :

Tableau souhaite accéder à votre compte Google

 hindmoutalatif@gmail.com

Tableau pourra ainsi :

 Afficher et télécharger tous vos fichiers dans Google Drive 

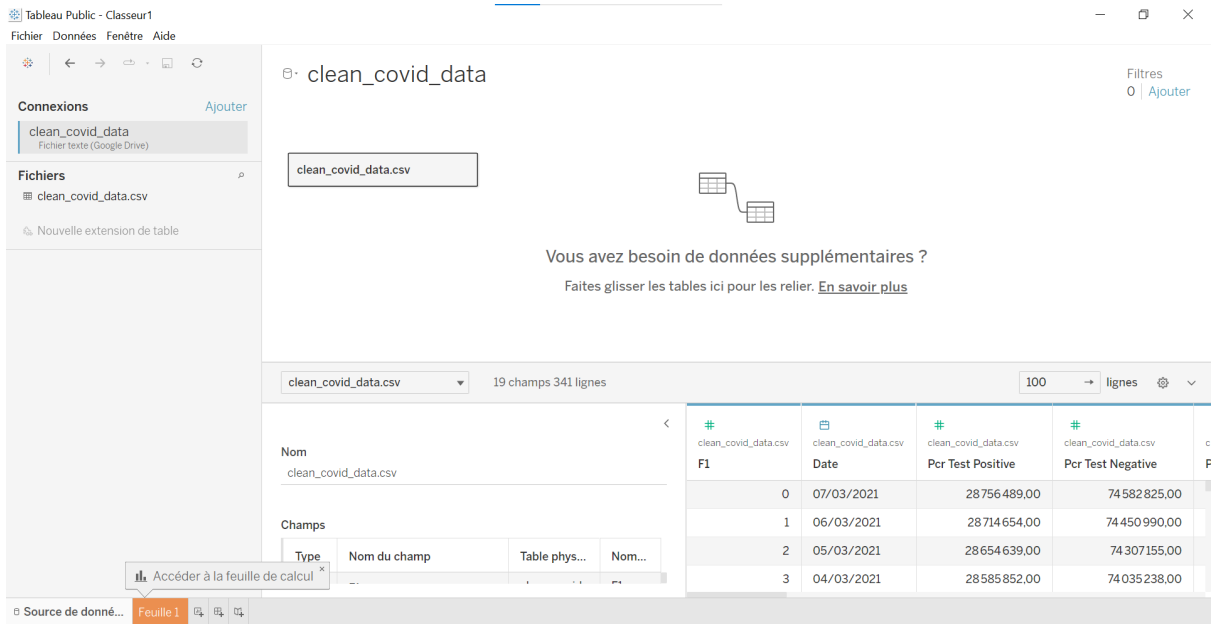
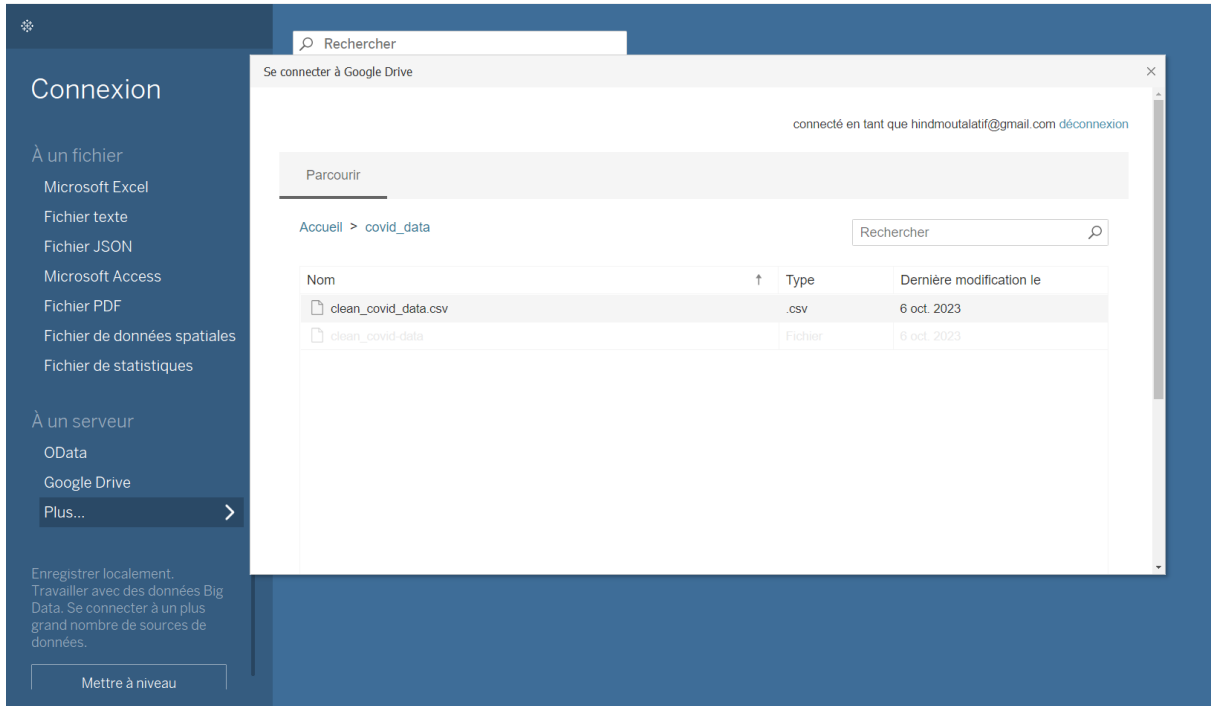
Vérifiez que l'application Tableau est digne de confiance

Il se peut que vous partagiez des informations sensibles avec ce site ou cette appli. Vous pouvez à tout moment consulter ou supprimer les autorisations d'accès dans votre [compte Google](#).

Découvrez comment Google vous aide à [partager vos données de manière sécurisée](#).

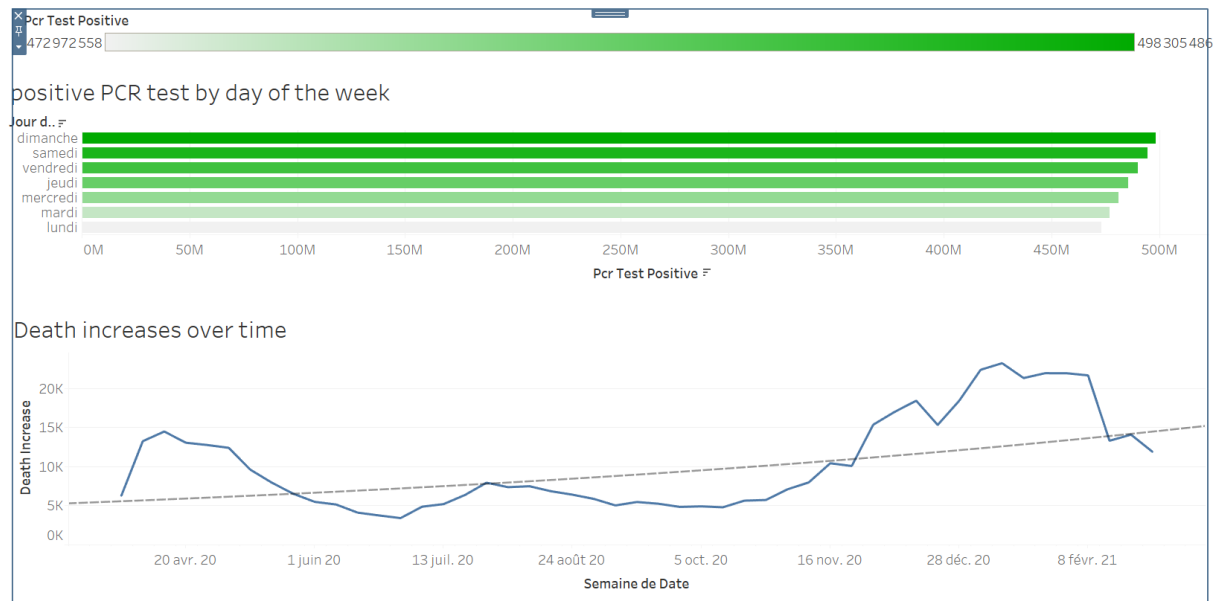
Consultez les [Règles de confidentialité](#) et les [Conditions d'utilisation](#) de Tableau.

[Annuler](#) [Autoriser](#)

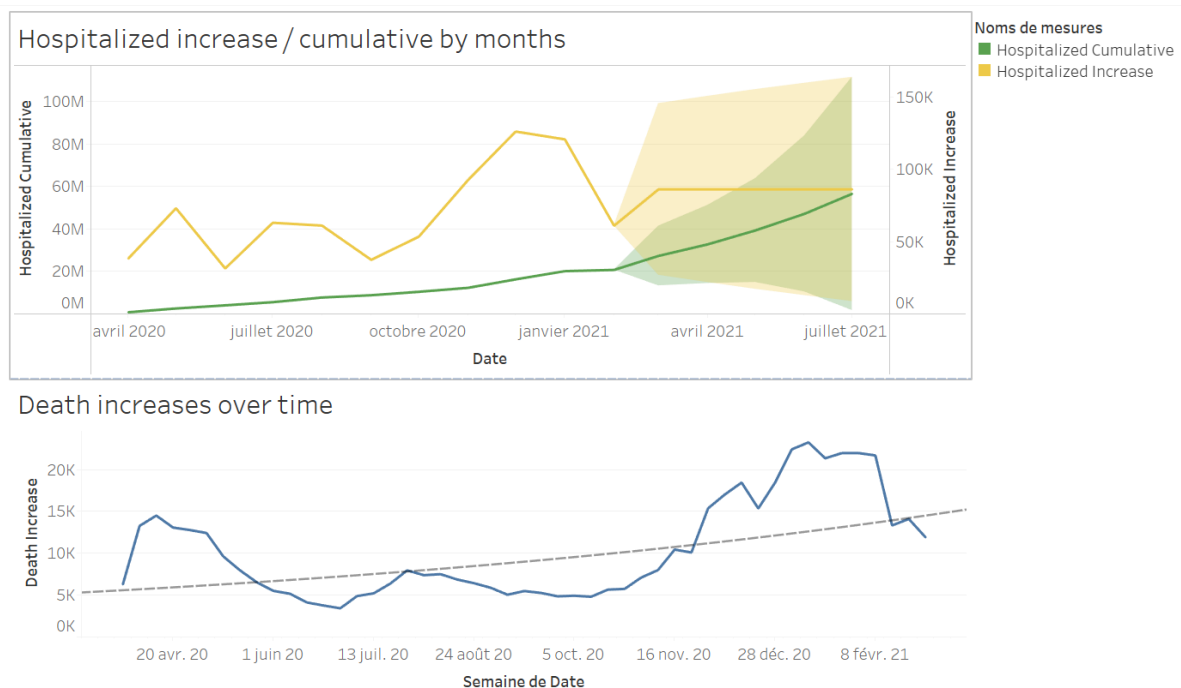


Now that we got our data into Tableau, here are the dashboards that we have created:

Most infectious days of the week :

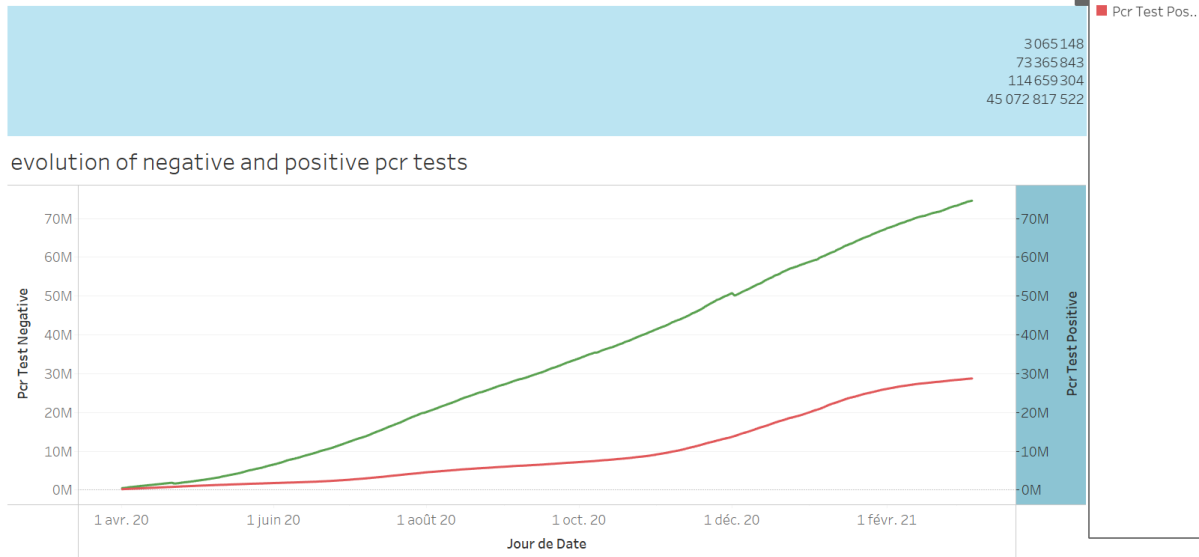


Death and hospitalized increases over time :

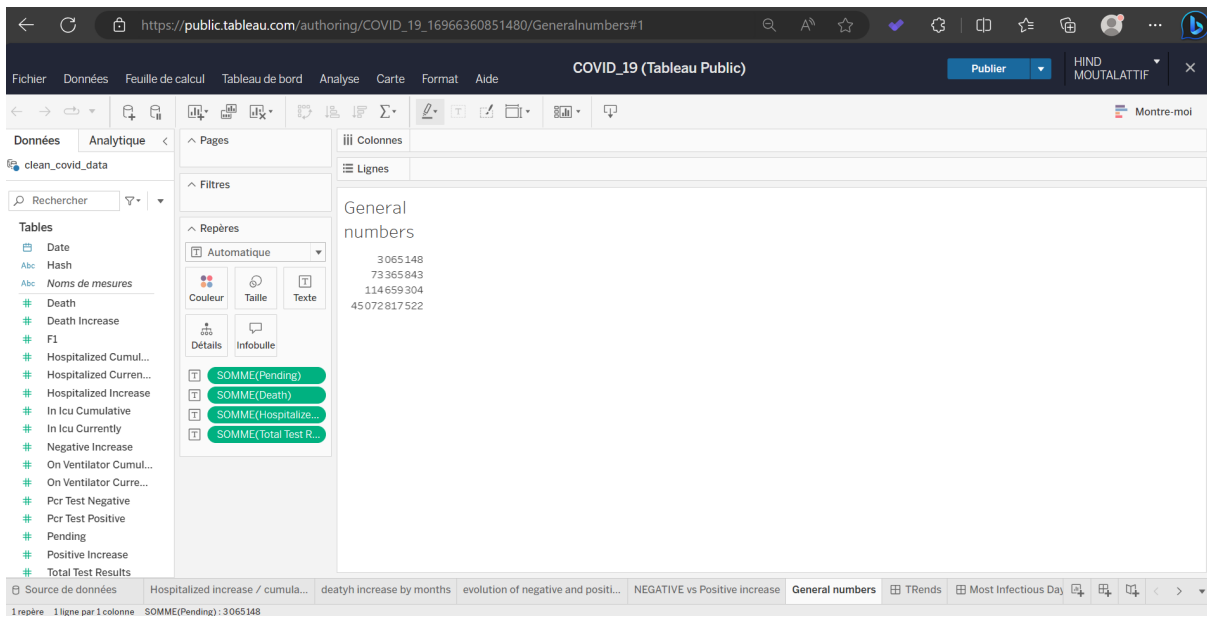


Positive tests vs negative tests evolution :

General numbers



Ensuite on enregistre et on publie notre classeur Covid_19 sur le serveur de tableau public



3. Gradient Boosting Model and Simple Flask API:

this api will give us the ability to see death increase based on currently hospitalized people using a Gradient boosting regressor ML model :

Code for creating our Gradient Boosting model that predicts death increase based on hospitalized people :

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from warnings import simplefilter

from flask import Flask, request, jsonify

# Ignore user warning
simplefilter(action='ignore', category=UserWarning)

def gradient_boosting(hospitalized_now: int):

    df = pd.read_csv("clean_covid_data.csv", index_col=0)
    df = df.dropna()

    x = df[["hospitalizedCurrently"]]
    y = df["deathIncrease"]

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42, shuffle=True)
    model = GradientBoostingRegressor(n_estimators=250, learning_rate=0.01, min_samples_leaf=20)

    model.fit(x_train, y_train)

    return round(model.predict([[hospitalized_now]])[0], 2)
```

Here is the code that helps creating the simple flask api :

```
Entrée [36]: app = Flask(__name__)

Entrée [37]: @app.route('/predict/<hospitalized_now>', methods=['GET'])
def predict(hospitalized_now):

    if request.method == 'GET':

        hospitalized_now = int(hospitalized_now)
        death_increase = gradient_boosting(hospitalized_now)

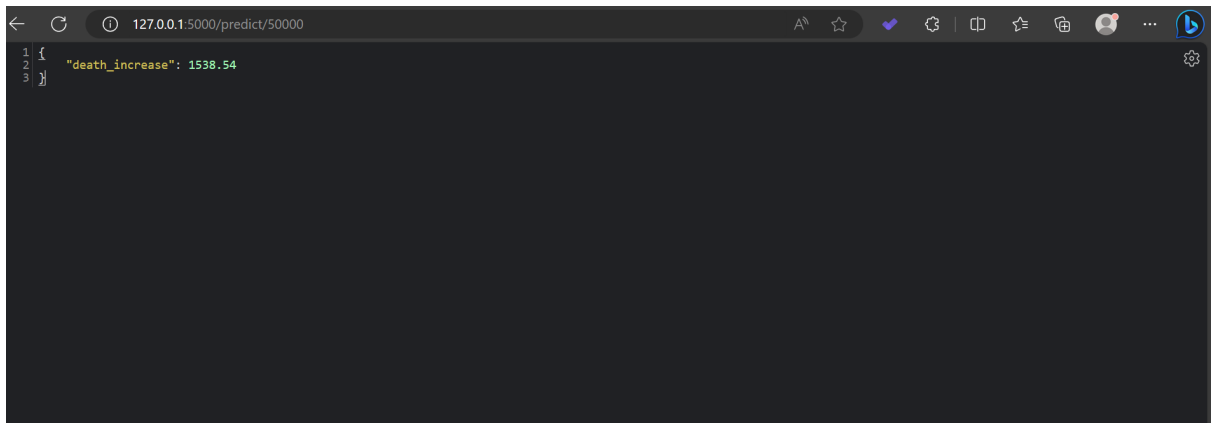
        return jsonify({
            "death_increase": death_increase
        })

Entrée [*]: app.run(port=5000)

* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off

* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [07/Oct/2023 22:08:25] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [07/Oct/2023 22:08:34] "GET /predict/50000 HTTP/1.1" 200 -
```

Our simple API looks like this :



A screenshot of a web browser window with a dark theme. The address bar shows the URL `127.0.0.1:5000/predict/50000`. The main content area displays a JSON response: `{ "death_increase": 1538.54 }`. The browser's developer tools are open, showing the response in the console. The JSON is formatted with line numbers 1, 2, and 3 on the left. The value `1538.54` is highlighted in green.

```
1 {  
2   "death_increase": 1538.54  
3 }
```

Done by HIND MOUTALATTIF