

*Sujet:*

**DATA WAREHOUSE PROJECT**

Réalisé par : Sabrine El Hssini  
Encadré par : Monsieur BENELALLAM Imade



# Table des matières

1	Extraction des données . . . . .	2
1.1	Identification des banques . . . . .	2
1.2	Table <code>banquée</code> sur PostgreSql . . . . .	5
1.3	Chargement des agences bancaires . . . . .	6
1.4	Géocodage des agences . . . . .	7
2	Automatisation de l'extraction des données . . . . .	12
2.1	Détection de la langue . . . . .	18
2.2	Analyse de sentiment . . . . .	18
2.3	Attribution thématique via LDA . . . . .	19
2.4	Chargement dans PostgreSQL . . . . .	19
2.5	Automatisation avec un DAG Airflow . . . . .	20
3	Phase 3 : Modélisation des données (Schéma en étoile dans PostgreSQL) . . . . .	24
	Phase 3 : Modélisation des données (Schéma en étoile dans PostgreSQL) . . .	24
3.1	Objectifs de la modélisation . . . . .	24
3.2	Architecture du schéma en étoile . . . . .	25
3.3	Implémentation des modèles . . . . .	26
4	Phase 5 : Visualisation des données — Dashboards . . . . .	31
4.1	Connexion au Data Warehouse . . . . .	31
4.2	Objectifs de la visualisation . . . . .	33
4.3	Types de visualisations réalisées . . . . .	33
4.4	Exemples concrets de graphiques . . . . .	34
4.5	AXE D'ANALYSE PERTINENT : carte de repartition des agences avec la note des avis . . . . .	37
4.6	AXE D'ANALYSE PERTINENT : Moyenne de note par <code>branch_name</code> et <code>topic</code> . . . . .	38
4.7	Intérêt de l'approche . . . . .	40
4.8	Autres graphiques pertinents . . . . .	41

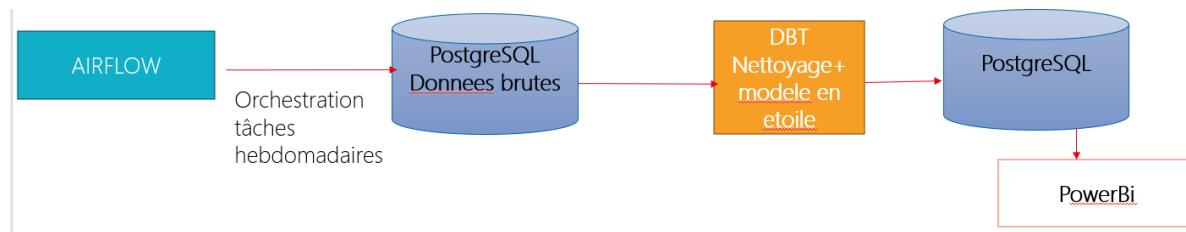
# Introduction

Avec la montée en puissance de l'expérience client comme levier stratégique, les banques au Maroc cherchent à mieux comprendre les avis de leurs clients, notamment ceux laissés en ligne sur des plateformes comme Google Maps. Ces avis contiennent des informations précieuses sur la satisfaction, les attentes, et les problèmes rencontrés dans les agences physiques.

Dans ce projet, nous avons mis en place un entrepôt de données (Data Warehouse) permettant d'analyser les avis clients laissés sur Google Maps concernant différentes banques marocaines. Le pipeline de données mis en œuvre couvre l'ensemble des étapes nécessaires : de l'extraction automatisée des données jusqu'à leur visualisation.

L'objectif principal est de structurer les données issues des avis en un modèle analytique robuste basé sur un schéma en étoile, puis d'offrir une vue synthétique et interactive de ces avis via un tableau de bord. Pour cela, nous avons mobilisé plusieurs outils modernes du data engineering, tels que PostgreSQL pour le stockage, DBT pour les transformations, Airflow pour l'automatisation et PowerBI pour la visualisation.

Ce document détaille chaque étape du projet, les choix techniques effectués, les transformations réalisées sur les données ainsi que les résultats obtenus à travers l'exploration des avis collectés.



**Figure 0.1 – Architecture du pipeline**

## 1 Extraction des données

La première étape de notre projet a consisté à extraire les avis clients relatifs aux banques marocaines depuis la plateforme Google Maps. Pour ce faire, nous avons utilisé l'API Google Places, qui permet d'accéder aux informations et aux avis publics disponibles pour chaque établissement référencé.

### 1.1 Identification des banques

Une liste de banques marocaines a été préalablement définie, incluant notamment Attijariwafa Bank, BCP, BMCE, CIH, entre autres. Pour chacune de ces banques, nous

## TABLE DES MATIÈRES

---

avons utilisé l'API Google Maps afin de récupérer leur *place\_id*, un identifiant unique permettant d'interroger l'API de manière précise. Le code suivant a été concu dans le but d'extraction efficace des données .

**Listing 1** – extraction des banques et des avis clients

```
import requests
import googlemaps
import pandas as pd
import time
API_KEY = "AIzaSyACo0eRv5pDIiaJ7pe1_W7qkVg8A4I_yTM"

gmaps = googlemaps.Client(key=API_KEY)

# Liste des banques      scraper
banques = [
    "Cr dit Agricole Maroc",
    "BMCE Bank",
    "Attijariwafa Bank",
    "Banque Populaire Maroc",
    "Soci t G n rale Maroc",
    "CIH Bank",
    "Al Barid Bank",
    "CFG Bank"
]

# Liste pour stocker toutes les agences bancaires
all_branches = []
# tape 1 : R cup rer les agences bancaires et leurs
# place_id
for banque in banques:
    print(f" Recherche des agences pour {banque}...")
    places_result = gmaps.places(query=banque + " Maroc")

    for place in places_result.get("results", []):
        all_branches.append({
            "Bank Name": banque,
            "Branch Name": place.get("name", "N/A"),
            "Place ID": place.get("place_id", "N/A"),
            "Address": place.get("formatted_address", "N/A")
        })

    time.sleep(2) # Pause pour viter d'tre bloqu par
```

## TABLE DES MATIÈRES

---

```
Google

# Convertir en DataFrame et sauvegarder
df_branches = pd.DataFrame(all_branches)
df_branches.to_csv("agences_bancaires_maroc.csv", index=False)
print(f"{len(df_branches)} agences r cup r es et enregistr es
dans agences_bancaires_maroc.csv !")

# Liste pour stocker les avis
all_reviews = []

# tape 2 : R cup rer les avis pour chaque agence
for _, row in df_branches.iterrows():
    place_id = row["Place ID"]
    banque = row["Bank Name"]
    agence = row["Branch Name"]
    print(f" R cup ration des avis pour {agence} ({banque})...")
    )

    # R cup rer les d tails de l agence , y compris les avis
    place_details = gmaps.place(place_id=place_id, fields=["reviews"])

    reviews = place_details.get("result", {}).get("reviews", [])
    for review in reviews:
        all_reviews.append({
            "Bank Name": banque,
            "Branch Name": agence,
            "Location": row["Address"],
            "Review Text": review.get("text", "N/A"),
            "Rating": review.get("rating", "N/A"),
            "Review Date": review.get("relative_time_description",
                "N/A")
        })

    time.sleep(2)  # Pause pour viter d' tre bloqu par
    Google

# Convertir en DataFrame et sauvegarder
df_reviews = pd.DataFrame(all_reviews)
df_reviews.to_csv("avis_banques_maroc.csv", index=False)
```

## TABLE DES MATIÈRES

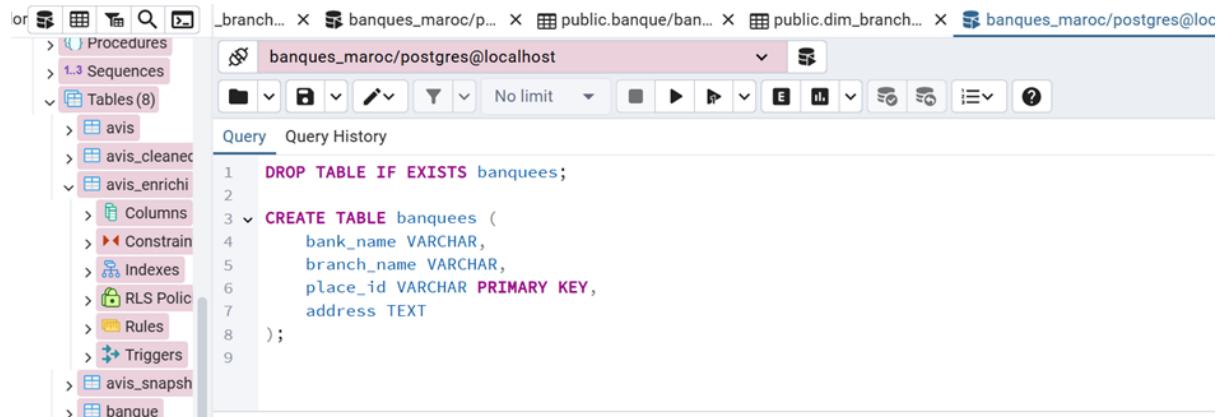
```
print(f"{len(df_reviews)} avis collect s et enregistr s dans  
avis_banques_maroc.csv !")
```

- ❖ Recherche des agences pour Crédit Agricole Maroc...
- ❖ Recherche des agences pour BMCE Bank...
- ❖ Recherche des agences pour Attijariwafa Bank...
- ❖ Recherche des agences pour Banque Populaire Maroc...
- ❖ Recherche des agences pour Société Générale Maroc...
- ❖ Recherche des agences pour CIH Bank...
- ❖ Recherche des agences pour Al Barid Bank...
- ❖ Recherche des agences pour CFG Bank...
- ✓ 143 agences récupérées et enregistrées dans agences\_bancaires\_maroc.csv !
- ❖ Récupération des avis pour Agricultural loan to Morocco (Crédit Agricole Maroc)...
- ❖ Récupération des avis pour Agricultural loan to Morocco (Crédit Agricole Maroc)...
- ❖ Récupération des avis pour Crédit Agricole (Crédit Agricole Maroc)...
- ❖ Récupération des avis pour Crédit du Maroc (Crédit Agricole Maroc)...
- ❖ Récupération des avis pour Credit Agricole Du Maroc (Agence Madagascar) (Crédit Agricole Maroc).
- ❖ Récupération des avis pour Siège régional Crédit du Maroc (Crédit Agricole Maroc)...
- ❖ Récupération des avis pour Crédit agricole (Crédit Agricole Maroc)...
- ❖ Récupération des avis pour Groupe Crédit Agricole du Maroc (Crédit Agricole Maroc)...
- ❖ Récupération des avis pour Crédit Agricole du Maroc (Crédit Agricole Maroc)...

Figure 1.1 – l'affichage de notre code python sur jupyter NOTEBOOK

## 1.2 Table banquée sur PostgreSQL

En parallèle, les métadonnées des banques sont stockées dans la table `banque`, qui contient pour chaque banque son `place_id`, son nom, et son adresse. Ce processus permet également d'éviter les doublons en ne réinsérant une banque que si son `place_id` n'existe pas encore dans la base.



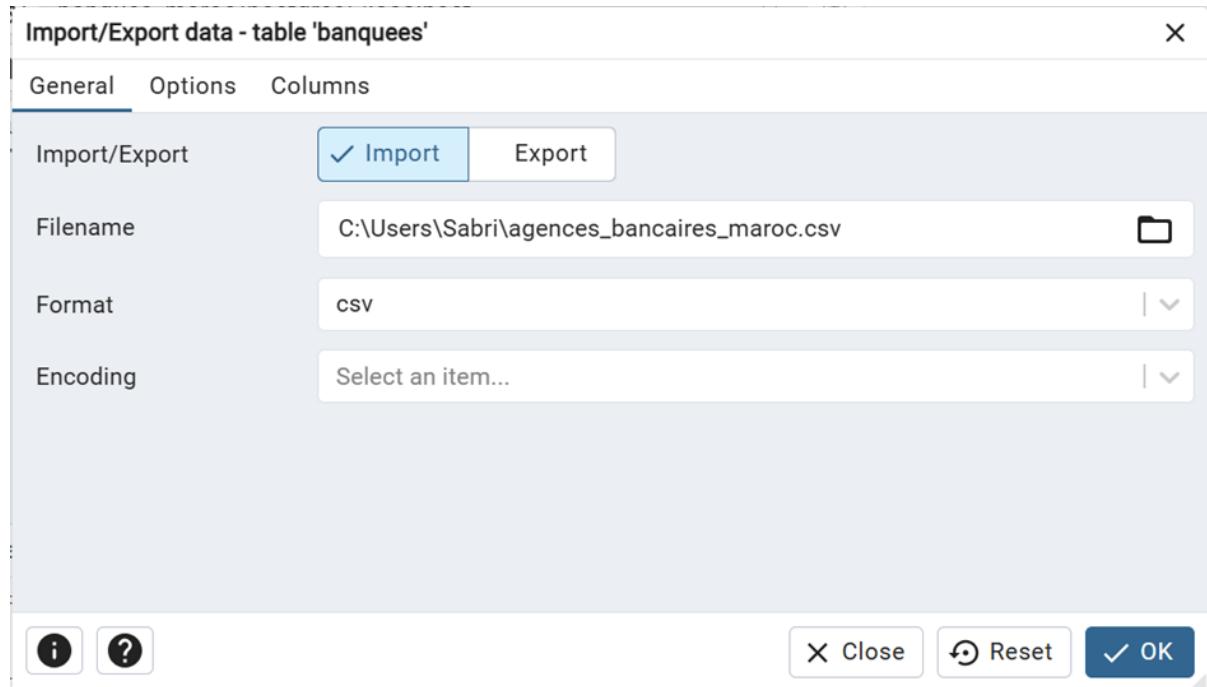
The screenshot shows the PgAdmin 4 interface connected to a database named 'banques\_maroc/postgres@localhost'. The left sidebar displays the schema structure with tables like 'avis', 'avis\_cleaned', 'avis\_enrichi', and 'banque'. The main window shows a query editor with the following SQL code:

```
1  DROP TABLE IF EXISTS banques;
2
3  CREATE TABLE banques (
4      bank_name VARCHAR,
5      branch_name VARCHAR,
6      place_id VARCHAR PRIMARY KEY,
7      address TEXT
8  );
9
```

Figure 1.2 – création de la table banques sur Pgadmin après connexion à notre base de données PostgreSQL avec localhost =5432

### 1.3 Chargement des agences bancaires

Pour compléter notre base, nous avons également importé fichier contenant la liste des agences bancaires au Maroc dans pgAdmin après connexion à notre base de données PostgreSql. Ce fichier a été chargé dans une table `banquees`, servant de source pour la suite du traitement.



**Figure 1.3** – Importation du fichier csv des banques dans la table banque via PGadmin après connexion a la base PostgreSql

Name	Modified	File Size
venv	3 days ago	
Geocodage.ipynb	yesterday	17.9 KB
LDA.ipynb	4 days ago	55.9 KB
Untitled.ipynb	3 days ago	26.3 KB
avis_banques_maroc.csv	2 months ago	40.4 KB
avis_banques_maroc.json	2 months ago	73.8 KB
avis_enrichi.csv	3 days ago	280.3 KB
avis_extraits.json	2 months ago	178.3 KB
avis.csv	3 days ago	254.1 KB
avis.json	2 months ago	188.5 KB
banques_maroc.csv	2 months ago	5.4 KB
banques_maroc.json	2 months ago	12.9 KB

**Figure 1.4** – les fichiers Json et CSV recuperes

Cette étape d'extraction constitue la base du pipeline et garantit une collecte fiable, automatisée et structurée des données issues de sources externes.

## 1.4 Géocodage des agences

The screenshot shows a MySQL Workbench interface. In the top navigation bar, 'Query History' is selected. Below it, a code editor displays the following SQL command:

```
1 ▾ ALTER TABLE banquees
2   ADD COLUMN latitude DOUBLE PRECISION,
3   ADD COLUMN longitude DOUBLE PRECISION;
4
```

Below the code editor, there are tabs for 'Data Output', 'Messages' (which is selected), and 'Notifications'. The 'Messages' tab displays the output of the query:

ALTER TABLE

Query returned successfully in 61 msec.

**Figure 1.5** – création des colonnes longitude et latitue dans la table banquees

Afin d'enrichir les informations sur les agences bancaires collectées (nom, adresse), nous avons procédé à une phase de **géocodage** visant à obtenir leurs coordonnées géographiques (`latitude`, `longitude`) ainsi que leurs `place_id` pour une identification unique.

### 4.i Méthodes utilisées

Deux approches ont été explorées pour réaliser le géocodage :

- **Nominatim (OpenStreetMap)** : dans un premier temps, nous avons utilisé l'API *Nominatim*, gratuite et open-source. Cette solution a permis d'obtenir des coordonnées pour un certain nombre d'adresses, mais s'est révélée insuffisante en raison de :
  - La non-reconnaissance de nombreuses adresses au format ambigu (ex. codes plus, abréviations),
  - Une précision parfois faible pour les points géographiques retournés.
- **Google Maps Geocoding API** : pour surmonter ces limitations, nous avons opté pour l'API Google Maps. Celle-ci offre :

- Une meilleure couverture géographique et qualité de reconnaissance d'adresse,
- L'accès aux `place_id`, utiles pour identifier les agences de manière stable,
- Une robustesse dans la récupération des coordonnées même avec des adresses partielles.

## Intégration technique

Le géocodage a été réalisé en Python via des scripts connectés à notre base PostgreSQL :

- Les adresses ont été extraites dynamiquement depuis la table `banquees` (contenant les agences sans coordonnées),
- Chaque adresse a été soumise à l'API (Nominatim puis Google Maps),
- En cas de réponse positive, la latitude, la longitude et le `place_id` ont été enregistrés en base,
- Des délais (`sleep`) ont été utilisés pour respecter les limites d'appel aux APIs.

**Listing 2** – Géocodage avec Nominatim

```
import psycopg2
from geopy.geocoders import Nominatim
import time
import re

# Fonction de nettoyage d'adresse
def nettoyer_adresse(adresse):
    if not adresse or not isinstance(adresse, str):
        return None

    adresse = re.sub(r'\b[\w]{4,7}\+\w{2,6}\b', '', adresse) # Supprime les codes Plus
    adresse = re.sub(r',\s*,+', ', ', adresse) # Supprime les virgules multiples
    adresse = re.sub(r'\s{2,}', ' ', adresse) # Supprime les espaces multiples
    adresse = adresse.strip().strip(' ,') # Supprime les virgules et espaces en trop

    if "morocco" not in adresse.lower():
        adresse += ", Morocco"

    return adresse.strip()
```

## TABLE DES MATIÈRES

---

```
#Connexion      la base PostgreSQL
conn = psycopg2.connect(
    dbname="banques_maroc",
    user="postgres",
    password="to",
    host="localhost",
    port="5432"
)
cur = conn.cursor()

#   Initialisation du geocodage
geolocator = Nominatim(user_agent="geoapi")

# Selection des adresses sans coordonnées
cur.execute("SELECT place_id, address FROM banquees WHERE
    latitude IS NULL OR longitude IS NULL")
rows = cur.fetchall()

# Geocodage
for place_id, raw_address in rows:
    adresse = nettoyer_adresse(raw_address)
    try:
        location = geolocator.geocode(adresse)
        if location:
            lat = location.latitude
            lon = location.longitude
            cur.execute(
                "UPDATE banquees SET latitude = %s, longitude = %
                s WHERE place_id = %s",
                (lat, lon, place_id)
            )
            conn.commit()
            print(f" Geocod : {adresse}")
        else:
            print(f"Non trouv : {adresse}")
            time.sleep(1)
    except Exception as e:
        print(f"Erreur pour {adresse} : {e}")

# Fermeture
```

## TABLE DES MATIÈRES

---

```
cur.close()  
conn.close()
```

Nous avons ensuite eu recours à l'API Google Maps, permettant d'obtenir de manière fiable les champs `latitude`, `longitude` et `place_id`.

**Listing 3** – Géocodage avec API Google Maps plus performant

```
import psycopg2  
import googlemaps  
import time  
  
# 1. Configuration  
API_KEY = "AIzaSyCVYwAfWZ9-FplgetVSNN6djcyEeM44uhM" #, ma cle  
gmaps = googlemaps.Client(key=API_KEY)  
  
  
conn = psycopg2.connect(  
    dbname="banques_maroc",  
    user="postgres",  
    password="to",  
    host="localhost",  
    port="5432"  
)  
cur = conn.cursor()  
  
# Requête pour adresses sans coordonnées  
cur.execute("SELECT place_id, address FROM banquees WHERE  
    latitude IS NULL OR longitude IS NULL")  
rows = cur.fetchall()  
  
for place_id, address in rows:  
    try:  
        geocode_result = gmaps.geocode(address)  
        if geocode_result:  
            lat = geocode_result[0]['geometry']['location']['lat'  
                ]  
            lon = geocode_result[0]['geometry']['location']['lng'  
                ]  
            new_place_id = geocode_result[0].get('place_id',  
                place_id)  
  
            # Vérifie si le place_id est déjà utilis
```

## TABLE DES MATIÈRES

---

```
        cur.execute("SELECT 1 FROM banquees WHERE place_id = %s", (new_place_id,))
        exists = cur.fetchone()

        if exists:
            # Mise jour uniquement des coordonnées
            cur.execute("""
                UPDATE banquees
                SET latitude = %s, longitude = %s
                WHERE place_id = %s
            """, (lat, lon, place_id))
        else:
            # Mise jour complète
            cur.execute("""
                UPDATE banquees
                SET latitude = %s, longitude = %s, place_id = %s
                WHERE place_id = %s
            """, (lat, lon, new_place_id, place_id))

        conn.commit()
        print(f"Good : {address}")
    else:
        print(f"Not found : {address}")
    time.sleep(1)
except Exception as e:
    print(f"Error for {address}: {e}")
    conn.rollback()

cur.close()
conn.close()
```

## Visualisation des données enrichies

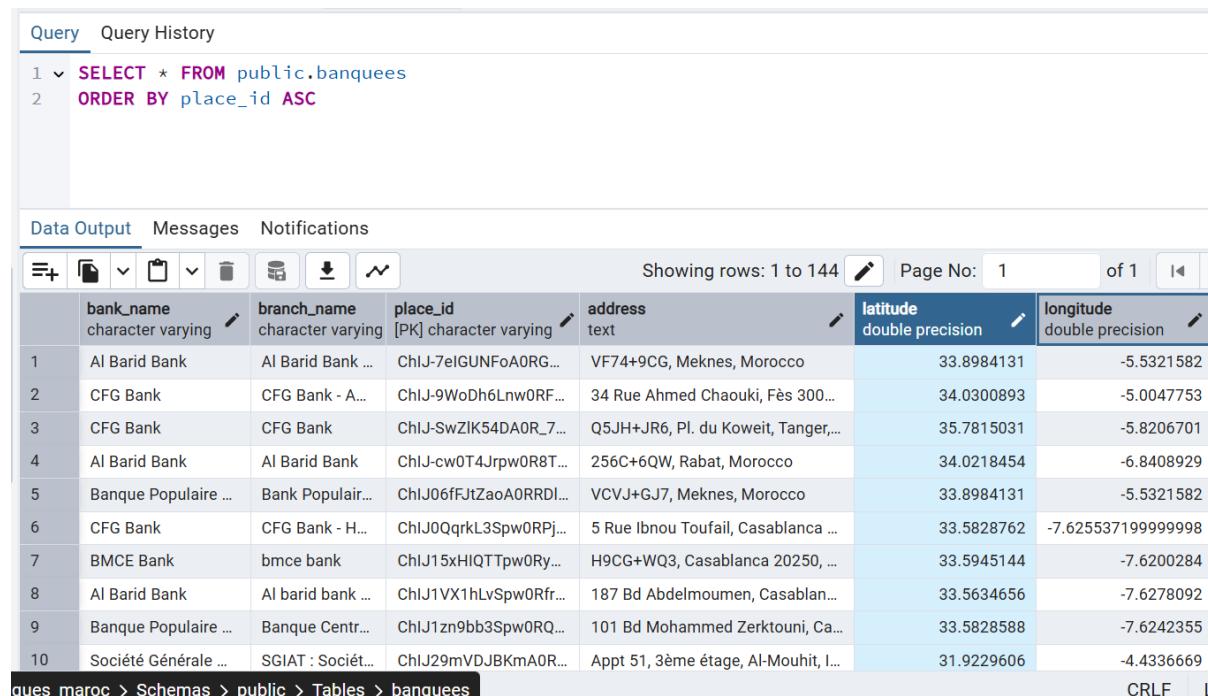
Une fois les coordonnées intégrées, la table des agences affiche pour chaque ligne :

- Le nom de la banque,
- L'adresse exacte,
- Les colonnes `latitude`, `longitude`,
- Et le `place_id` associé, garantissant l'unicité.

Ces données sont désormais prêtes à être utilisées dans la modélisation du Data Warehouse.

## TABLE DES MATIÈRES

---



The screenshot shows a database query results interface. At the top, there is a query history section with the following SQL code:

```
1 ✓ SELECT * FROM public.banques
2 ORDER BY place_id ASC
```

Below this is a navigation bar with tabs for "Data Output", "Messages", and "Notifications". The "Data Output" tab is selected. The main area displays a table of 144 rows, showing columns for bank name, branch name, place ID, address, latitude, and longitude. The table includes edit icons for each row. The interface also shows pagination information: "Showing rows: 1 to 144" and "Page No: 1 of 1".

	bank_name character varying	branch_name character varying	place_id [PK] character varying	address text	latitude double precision	longitude double precision
1	Al Barid Bank	Al Barid Bank ...	ChIJ7eIGUNFoA0RG...	VF74+9CG, Meknes, Morocco	33.8984131	-5.5321582
2	CFG Bank	CFG Bank - A...	ChIJ-9WoDh6Lnw0RF...	34 Rue Ahmed Chaouki, Fès 300...	34.0300893	-5.0047753
3	CFG Bank	CFG Bank	ChIJ-SwZIK54DA0R_7...	Q5JH+JR6, Pl. du Koweit, Tanger,...	35.7815031	-5.8206701
4	Al Barid Bank	Al Barid Bank	ChIJ-cw0T4Jrpw0R8T...	256C+6QW, Rabat, Morocco	34.0218454	-6.8408929
5	Banque Populaire ...	Bank Populair...	ChIJ06fFJtZaoA0RRDl...	VCVJ+GJ7, Meknes, Morocco	33.8984131	-5.5321582
6	CFG Bank	CFG Bank - H...	ChIJ0QqrkL3Spw0RPj...	5 Rue Ibnou Toufail, Casablanca ...	33.5828762	-7.625537199999998
7	BMCE Bank	bmce bank	ChIJ15xHIQTTpw0Ry...	H9CG+WQ3, Casablanca 20250, ...	33.5945144	-7.6200284
8	Al Barid Bank	Al barid bank ...	ChIJ1VX1hLvSpw0Rfr...	187 Bd Abdelmoumen, Casablan...	33.5634656	-7.6278092
9	Banque Populaire ...	Banque Centr...	ChIJ1zn9bb3Spw0RQ...	101 Bd Mohammed Zerkouni, Ca...	33.5828588	-7.6242355
10	Société Générale ...	SGIAT : Sociét...	ChIJ29mVDJBKmA0R...	Appt 51, 3ème étage, Al-Mouhit, I...	31.9229606	-4.4336669

Below the table, the breadcrumb navigation path is shown: "banques maroc > Schemas > public > Tables > banques".

Figure 1.6 – Résultat du géocodage

## 2 Automatisation de l'extraction des données

Afin de garantir une collecte régulière et automatique des avis clients, nous avons mis en place un processus d'extraction hebdomadaire à l'aide de la plateforme **Apache Airflow**. Cette automatisation permet de maintenir notre entrepôt de données à jour sans intervention manuelle.

Listing 4 – Extrait important du dag

```
from airflow import DAG
from airflow.operators.python import PythonOperator
import pendulum
import googlemaps
import psycopg2
import json
import os
default_args = {
    "owner": "airflow",
}
start_date = pendulum.today("UTC").add(days=-1)

with DAG(
```

## TABLE DES MATIÈRES

---

```
dag_id="dag_google_reviews_v2",
schedule="@weekly",
start_date=start_date,
catchup=False,
default_args=default_args,
tags=["banques", "avis", "googlemaps"]
) as dag:
    extract_task = PythonOperator(
        task_id="extract_reviews",
        python_callable=extract_reviews,
        provide_context=True
    )

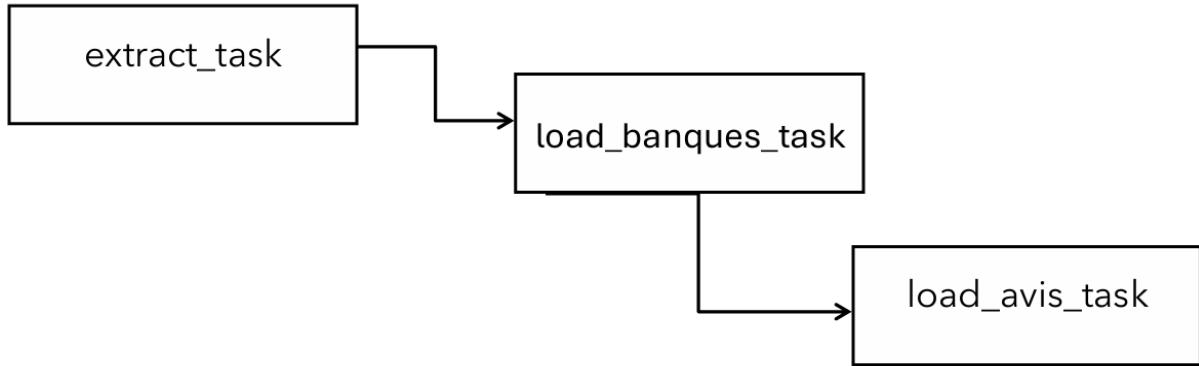
    load_banques_task = PythonOperator(
        task_id="load_banques_to_postgres",
        python_callable=load_banques_to_postgres,
        provide_context=True
    )

    load_avis_task = PythonOperator(
        task_id="load_to_postgres",
        python_callable=load_to_postgres,
        provide_context=True
    )

    extract_task >> load_banques_task >> load_avis_task
```

## Structure du DAG Airflow

Le DAG (*Directed Acyclic Graph*) défini pour ce projet est exécuté une fois par semaine. Il se compose des étapes suivantes :



**Figure 2.1 – Modelisation du dag**

1. **Récupération des avis Google Maps** : à partir de l'API Google Places, les avis sont collectés pour chaque banque listée dans notre fichier de référence.
2. **Vérification et insertion des banques** : pour chaque avis, on vérifie si la banque correspondante existe déjà dans la table `banque`. Si ce n'est pas le cas, elle est ajoutée en utilisant son `place_id` comme identifiant unique.
3. **Enregistrement des avis** : tous les nouveaux avis sont insérés dans la table `avis`, même s'il peut y avoir des doublons (ceux-ci seront gérés ultérieurement dans les étapes de nettoyage).

## Avantages de l'automatisation

Cette automatisation présente plusieurs avantages :

- **Gains de temps** : plus besoin d'exécuter manuellement les scripts d'extraction.
- **Actualisation régulière** : les nouvelles données sont intégrées chaque semaine.
- **Traçabilité** : grâce aux logs générés par Airflow, chaque exécution peut être suivie et auditée facilement.

Ce processus constitue la première étape du pipeline ETL, assurant la disponibilité des données brutes nécessaires aux phases de transformation et d'analyse ultérieures.

## Phase 2 : Nettoyage et transformation des données

Dans cette phase, nous avons appliqué un processus rigoureux de nettoyage et de transformation des avis clients à l'aide de **SQL** et de **DBT**. L'objectif est d'assurer la

## TABLE DES MATIÈRES

---

qualité, la cohérence et la pertinence des données avant leur exploitation analytique.

### 1. Suppression des doublons

Pour éviter les biais induits par les avis répétés, nous avons utilisé la fonction `ROW_NUMBER()` partitionnée par auteur, commentaire, identifiant de la banque et date de l'avis. Seul le premier avis (rang = 1) a été conservé dans chaque groupe.

### 2. Normalisation des textes

Les commentaires ont été normalisés pour permettre leur analyse linguistique (analyse de sentiments, détection de topics) :

- Passage en minuscules (`LOWER`).
- Suppression de la ponctuation avec `REGEXP_REPLACE`.
- Nettoyage des caractères spéciaux.
- Suppression des mots vides (*stopwords*) en **français** et **anglais** à l'aide d'expressions régulières SQL.

### 3. Traitement des valeurs manquantes

Les avis sans valeur pertinente ont été écartés :

- Suppression des commentaires vides ou `NULL`.
- Suppression des avis ne contenant pas d'identifiant de banque (`banque_id`).

### 4. Vue SQL nettoyée

Toutes les étapes précédentes ont été regroupées dans une vue SQL nommée `avis_clean`, intégrée dans DBT.

## TABLE DES MATIÈRES

---



The screenshot shows a SQL code editor window titled "avis\_clean". The tabs at the top are "General", "Definition", "Code", "Security", and "SQL". The "Code" tab is selected. The code itself is a complex SQL query using Common Table Expressions (CTEs) to clean up data from a table named "avis". It starts with a CTE named "avis\_source" that selects various columns from the "avis" table. This is followed by another CTE named "ranked" which adds a ranking column ("rang") using the ROW\_NUMBER() function partitioned by author, comment, and date. Finally, a CTE named "deduplicated" removes duplicates based on the ranked ID. The code ends with a SELECT statement that retrieves the cleaned data.

```
1  WITH avis_source AS (
2      SELECT avis.id,
3          avis.banque_id,
4          avis.auteur,
5          avis.commentaire,
6          avis.note,
7          avis.date_review
8      FROM avis
9  ), ranked AS (
10     SELECT avis_source.id,
11         avis_source.banque_id,
12         avis_source.auteur,
13         avis_source.commentaire,
14         avis_source.note,
15         avis_source.date_review,
16         row_number() OVER (PARTITION BY avis_source.auteur, avis_source.commentaire, avis_source.banque_id, avis_source.date_review)
17         FROM avis_source
18  ), deduplicated AS (
19     SELECT ranked.id,
20         ranked.banque_id,
21         ranked.auteur,
22         ranked.commentaire,
23         ranked.note,
24         ranked.date_review,
25         ranked.rang
26     FROM ranked
27 )
```

Figure 2.2 – extrait du code SQL pour nettoyer les avis

Listing 5 – Code complet SQL pour nettoyer les avis

```
WITH avis_source AS (
    SELECT avis.id,
        avis.banque_id,
        avis.auteur,
        avis.commentaire,
        avis.note,
        avis.date_review
    FROM avis
), ranked AS (
    SELECT avis_source.id,
        avis_source.banque_id,
        avis_source.auteur,
        avis_source.commentaire,
        avis_source.note,
        avis_source.date_review,
        row_number() OVER (PARTITION BY avis_source.auteur,
            avis_source.commentaire, avis_source.banque_id,
            avis_source.date_review ORDER BY avis_source.id)
        AS rang
    FROM avis_source
), deduplicated AS (
    SELECT ranked.id,
        ranked.banque_id,
        ranked.auteur,
        ranked.commentaire,
```

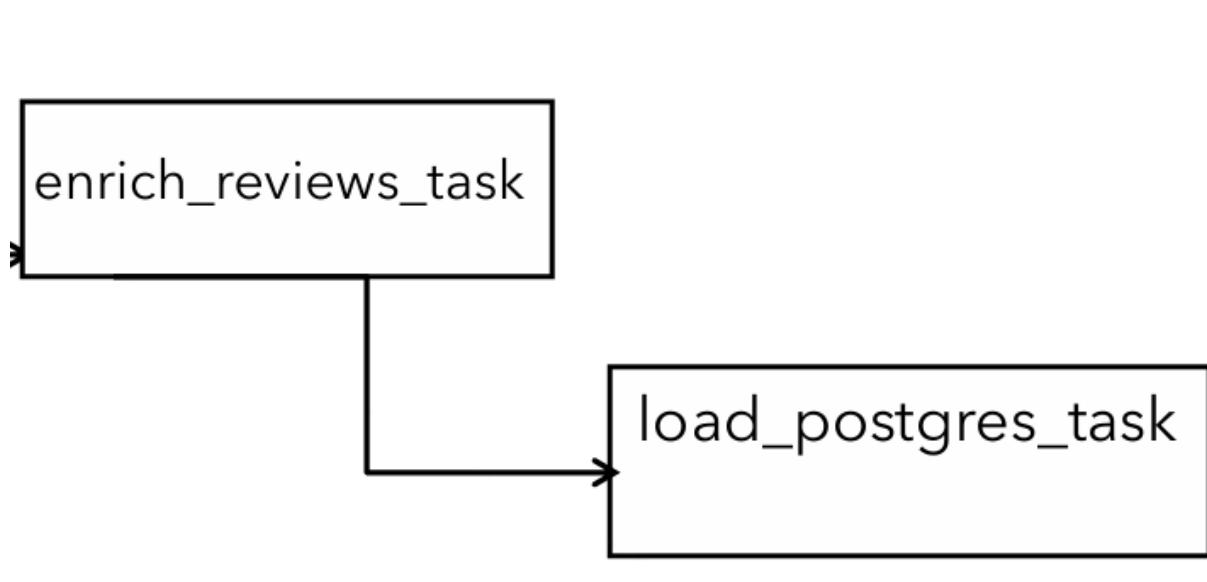
## TABLE DES MATIÈRES

---

```
ranked.note,
ranked.date_review,
ranked.rang
FROM ranked
WHERE ranked.rang = 1
), cleaned AS (
SELECT deduplicated.id,
deduplicated.banque_id,
lower(regexp_replace(deduplicated.auteur::text, '[^\w\s]::text, ''::text, 'g)::text) AS auteur,
lower(regexp_replace(deduplicated.commentaire, '[^\w\s]::text, ''::text, 'g)::text) AS
commentaire_nettoye,
deduplicated.note,
deduplicated.date_review
FROM deduplicated
WHERE deduplicated.commentaire IS NOT NULL AND TRIM(
    BOTH FROM deduplicated.commentaire) <> ''::text AND
deduplicated.banque_id IS NOT NULL
), stopwords_removed AS (
SELECT cleaned.id,
cleaned.banque_id,
cleaned.auteur,
regexp_replace(cleaned.commentaire_nettoye, '\m(le|la
|les|un|une|de|du|des|et|en| |a|pour|ce|cette|il|
elle|on|je|tu|nous|vous|ils|elles|avec|dans|au|aux
|ne|pas|que|qui|the|and|to|in|for|with|on|of|is|
are|was|were|you|your|i|my|me|he|she|it|they|them|
we|this|that|an|as|at|be|by|but|if|or|so|from|
because|what|which|how|can|could|should|would|will
|just|do|does|did|has|have|had|been|been|its|the|
very|their|who|there|dont|when|who|even|no|not|one
|still|only|more)\M)::text, ''::text, 'gi)::text)
AS commentaire,
cleaned.note,
cleaned.date_review
FROM cleaned
)
SELECT stopwords_removed.id,
stopwords_removed.banque_id,
stopwords_removed.auteur,
```

```
stopwords_removed.commentaire ,  
stopwords_removed.note ,  
stopwords_removed.date_review  
FROM stopwords_removed;
```

## Phase 2 : Enrichissement des avis clients



**Figure 2.3** – Modélisation du dag d'enrichissement des avis

L’objectif de cette étape est d’enrichir les avis collectés à partir des plateformes Google Maps afin de leur ajouter des informations linguistiques, sémantiques et analytiques utiles pour la modélisation et l’analyse.

### 2.1 Détection de la langue

Nous avons utilisé le modèle préentraîné `lid.176.bin` de **FastText** pour identifier automatiquement la langue de chaque commentaire. Ce modèle multilingue permet de détecter 176 langues. L’identification automatique nous a permis de filtrer les langues dominantes (français et anglais) et de mieux adapter le traitement du texte.

### 2.2 Analyse de sentiment

L’analyse de sentiment a été réalisée à l’aide du module **VADER** (Valence Aware Dictionary and sEntiment Reasoner) de la bibliothèque **NLTK**. VADER attribue un score de polarité à chaque commentaire, que nous avons converti en trois catégories :

- **Positive** : score > 0.1
- **Negative** : score < -0.1
- **Neutral** : sinon

## 2.3 Attribution thématique via LDA

Une modélisation thématique non supervisée a été appliquée avec **Latent Dirichlet Allocation (LDA)** pour détecter les sujets dominants des commentaires. Cette technique s'appuie sur le nettoyage des textes (passage en minuscules, suppression de la ponctuation et des stopwords) et la vectorisation du corpus via le modèle **Gensim**.

```

Topic 0:
0.027*"service" + 0.018*"bank" + 0.012*"bad" + 0.012*"agency" + 0.008*"customer" + 0.007*"like" + 0.007*"time" + 0.006*"worst" + 0.006*"phone" + 0.006*"account"

Topic 1:
0.031*"bank" + 0.025*"service" + 0.019*"good" + 0.011*"agency" + 0.008*"account" + 0.007*"staff" + 0.007*"customers" + 0.007*"banking" + 0.006*"bad" + 0.006*"poor"

Topic 2:
0.020*"service" + 0.019*"bank" + 0.012*"agency" + 0.010*"account" + 0.009*"services" + 0.006*"banking" + 0.006*"bad" + 0.006*"us" + 0.006*"money" + 0.005*"staff"

Topic 3:
0.016*"agency" + 0.015*"bank" + 0.011*"service" + 0.009*"customer" + 0.006*"account" + 0.006*"cash" + 0.006*"atm" + 0.005*"customers" + 0.005*"avoid" +
0.005*"contact"

Topic 4:
0.016*"bank" + 0.007*"service" + 0.007*"phone" + 0.007*"customers" + 0.007*"answer" + 0.006*"get" + 0.005*"time" + 0.005*"agency" + 0.005*"transfer" + 0.004*"never"

```

**Figure 2.4** – output des mots relatifs aux topics

Nous avons identifié **5 topics dominants**, que nous avons interprétés manuellement à partir des mots les plus représentatifs :

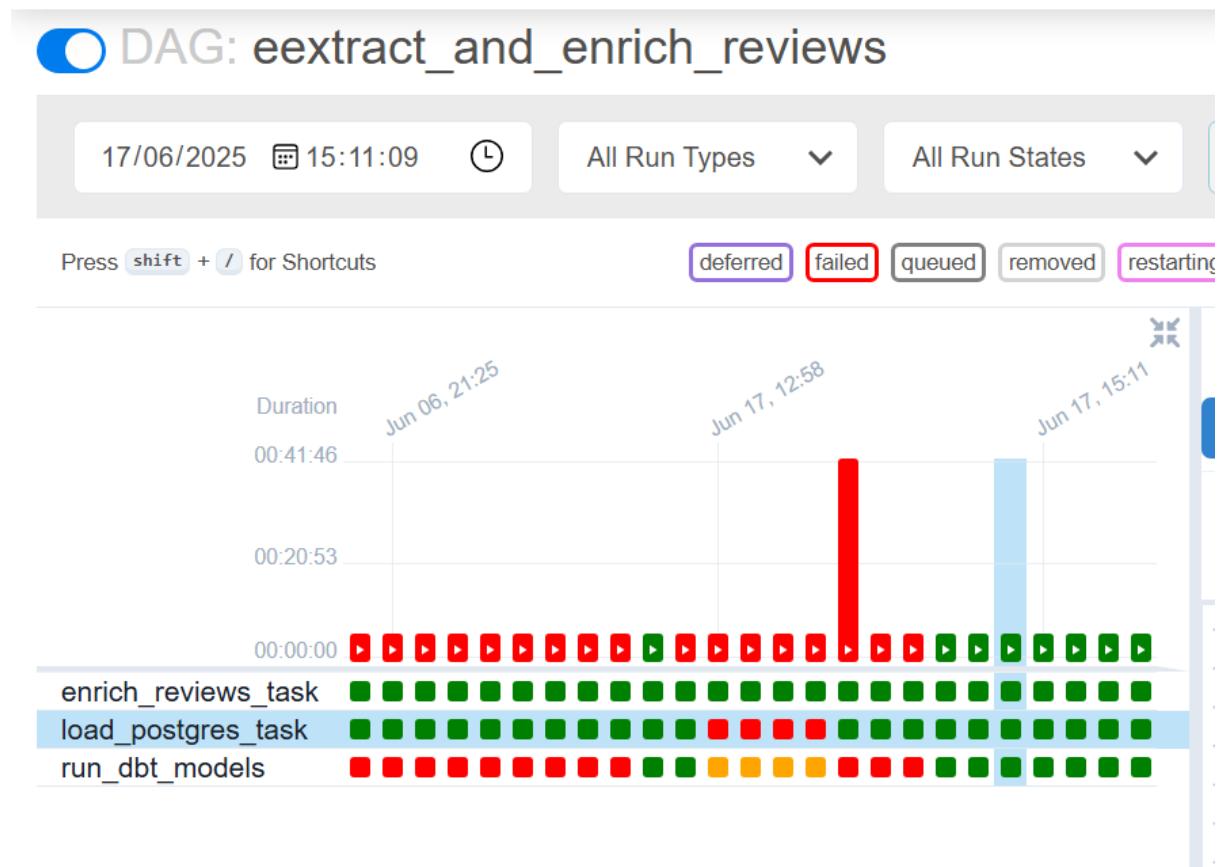
- **Topic 0** : Service client et temps d'attente
- **Topic 1** : Avis positifs sur l'agence
- **Topic 2** : Expérience bancaire générale
- **Topic 3** : ATM et retrait d'argent
- **Topic 4** : Problèmes de contact

Chaque commentaire est ainsi enrichi d'un topic et d'une étiquette de sentiment, en plus de la langue détectée.

## 2.4 Chargement dans PostgreSQL

Les données enrichies sont insérées dans la table `avis_enrichi` de la base PostgreSQL. Cette table sert de point central pour les modèles DBT ultérieurs. Un DAG Airflow automatisé a été mis en place pour exécuter cette étape de manière récurrente.

## 2.5 Automatisation avec un DAG Airflow



**Figure 2.5 – Historique d'exécution du DAG**

La figure ci-dessous présente l'historique d'exécution du DAG `eextract_and_enrich_reviews`, composé de trois tâches principales : `enrich_reviews_task`, responsable de l'enrichissement des avis via la détection de langue, l'analyse de sentiment et l'attribution d'un topic ; `load_postgres_task`, qui insère les données enrichies dans la base PostgreSQL ; et `run_dbt_models`, qui exécute les modèles DBT afin de produire les vues analytiques finales.

Chaque carré représente une exécution d'une tâche, et sa couleur indique son état : vert pour une exécution réussie, rouge pour un échec, orange pour une tâche replanifiée, et bleu pour une tâche en cours. L'historique met en évidence les itérations nécessaires pour corriger certaines erreurs (connexion, dépendances ou chemins d'accès) avant d'obtenir une exécution complète et fonctionnelle de bout en bout.

## TABLE DES MATIÈRES

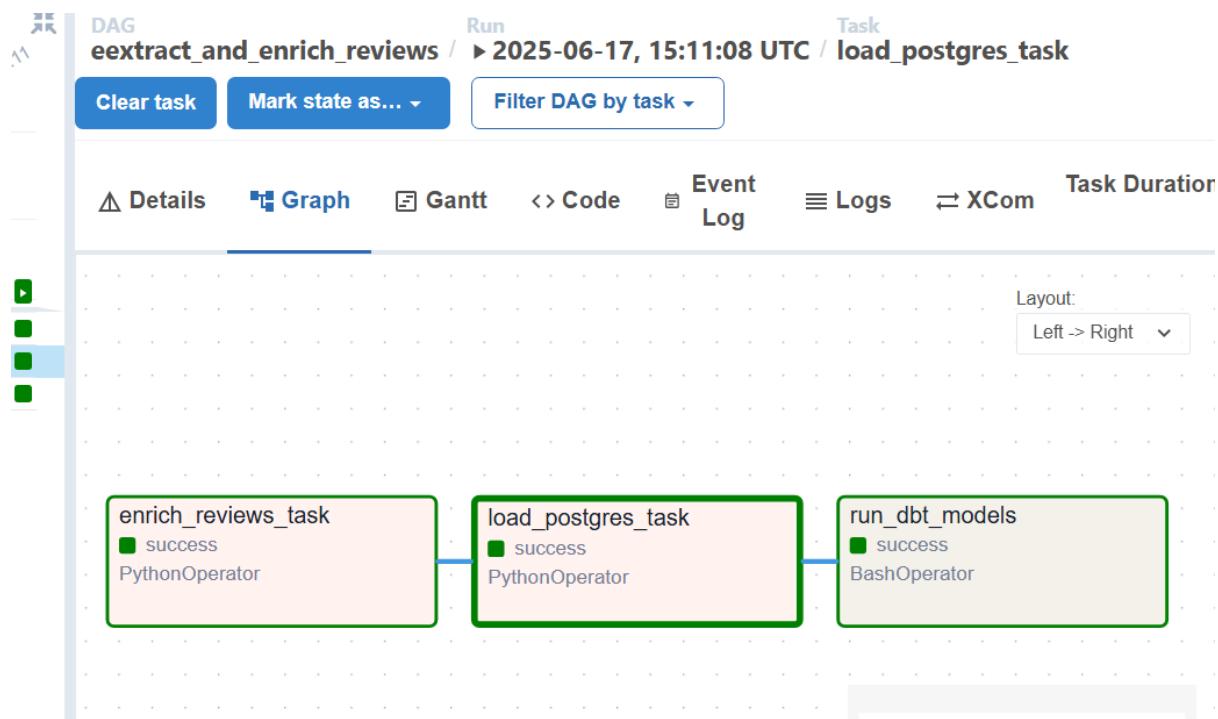


Figure 2.6 – Graphe du dag sur le localhost de Airflow

Listing 6 – Dag complet pour enrichir les avis

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.bash import BashOperator
from datetime import datetime, timedelta
import pandas as pd
import fasttext
from gensim import corpora, models
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import nltk
import sqlalchemy
from sqlalchemy import create_engine

nltk.download("vader_lexicon")

# Param tres de la base PostgreSQL
engine = create_engine("postgresql+psycopg2://postgres:
motdepasse@localhost:5432/banques_maroc")

default_args = {
    "owner": "airflow",
    "retries": 1,
    "retry_delay": timedelta(minutes=2)
```

## TABLE DES MATIÈRES

---

```
}

with DAG(
    dag_id="eextract_and_enrich_reviews",
    default_args=default_args,
    start_date=datetime(2025, 6, 1),
    schedule=None,
    catchup=False
) as dag:

    def extract_reviews():
        df = pd.read_sql("SELECT * FROM avis", engine)
        df.to_csv("/home/sabrine123/avis.csv", index=False)
        return

    def enrich_reviews():
        df = pd.read_csv("/home/sabrine123/avis.csv")
        df["commentaire"] = df["commentaire"].astype(str)

        # Détection de langue avec fastText
        ft_model = fasttext.load_model("/home/sabrine123/lid.176.bin")
        def detect_lang(text):
            try:
                return ft_model.predict(text)[0][0].replace(
                    "__label__", "")
            except:
                return "unknown"
        df["langue"] = df["commentaire"].apply(detect_lang)

        # Sentiment avec VADER
        analyzer = SentimentIntensityAnalyzer()
        def vader_sentiment(text):
            score = analyzer.polarity_scores(text)["compound"]
            if score > 0.1:
                return "Positive"
            elif score < -0.1:
                return "Negative"
            else:
                return "Neutral"
        df["sentiment"] = df["commentaire"].apply(vader_sentiment)
```

## TABLE DES MATIÈRES

---

```
)\n\n# Topics (LDA)\ntexts = df[\"commentaire\"].str.lower().str.replace(r\"[^\\w\\s]\", \"\", regex=True).str.split()\ndictionary = corpora.Dictionary(texts)\ncorpus = [dictionary.doc2bow(text) for text in texts]\nlda_model = models.LdaModel(corpus=corpus, id2word=\ndictionary, num_topics=4, passes=10, random_state=42)\ndef get_topic_name(bow):\n    topics = lda_model.get_document_topics(bow)\n    if topics:\n        return f\"Topic {max(topics, key=lambda x: x[1])\n[0]}\"\n    else:\n        return \"Inconnu\"\ndf[\"topic\"] = [get_topic_name(dictionary.doc2bow(text))\n    for text in texts]\n\ndf.to_csv(\"/home/sabrine123/avis_enrichi.csv\", index=\n    False)\n\ndef load_to_postgres():\n    df = pd.read_csv(\"/home/sabrine123/avis_enrichi.csv\")\n    with engine.begin() as conn:\n        conn.execute(\"DELETE FROM avis_enrichi\") # Ne pas\n            DROP pour viter erreurs\n    df.to_sql(\"avis_enrichi\", engine, if_exists=\"append\", \n        index=False)\n\nextract_reviews_task = PythonOperator(\n    task_id=\"extract_reviews_task\", \n    python_callable=extract_reviews\n)\n\nenrich_reviews_task = PythonOperator(\n    task_id=\"enrich_reviews_task\", \n    python_callable=enrich_reviews\n)\n\nload_postgres_task = PythonOperator(
```

```
task_id="load_postgres_task",
python_callable=load_to_postgres
)

run_dbt = BashOperator(
task_id='run_dbt_models',
bash_command="""
source ~/dbt-env/bin/activate && \
cd /home/Sabrine123/avis_reviews_dbt && \
dbt run
""",
dag=dag,
)

extract_reviews_task >> enrich_reviews_task >>
load_postgres_task >> run_dbt
```

### 3 Phase 3 : Modélisation des données (Schéma en étoile dans PostgreSQL)

Dans cette phase, nous avons structuré les données nettoyées dans un **modèle en étoile**, en suivant les bonnes pratiques de l'architecture des entrepôts de données. Ce modèle facilite les analyses ultérieures, notamment la création de rapports dans PowerBi.

Le fait étudié dans ce modèle est l'**avis client**, analysé à travers plusieurs axes : le **topic** (thème principal du contenu), la **note attribuée** (score de 1 à 5) et la **polarité du sentiment** (positif, neutre ou négatif). Ces informations permettent une analyse fine des opinions exprimées sur les agences bancaires.

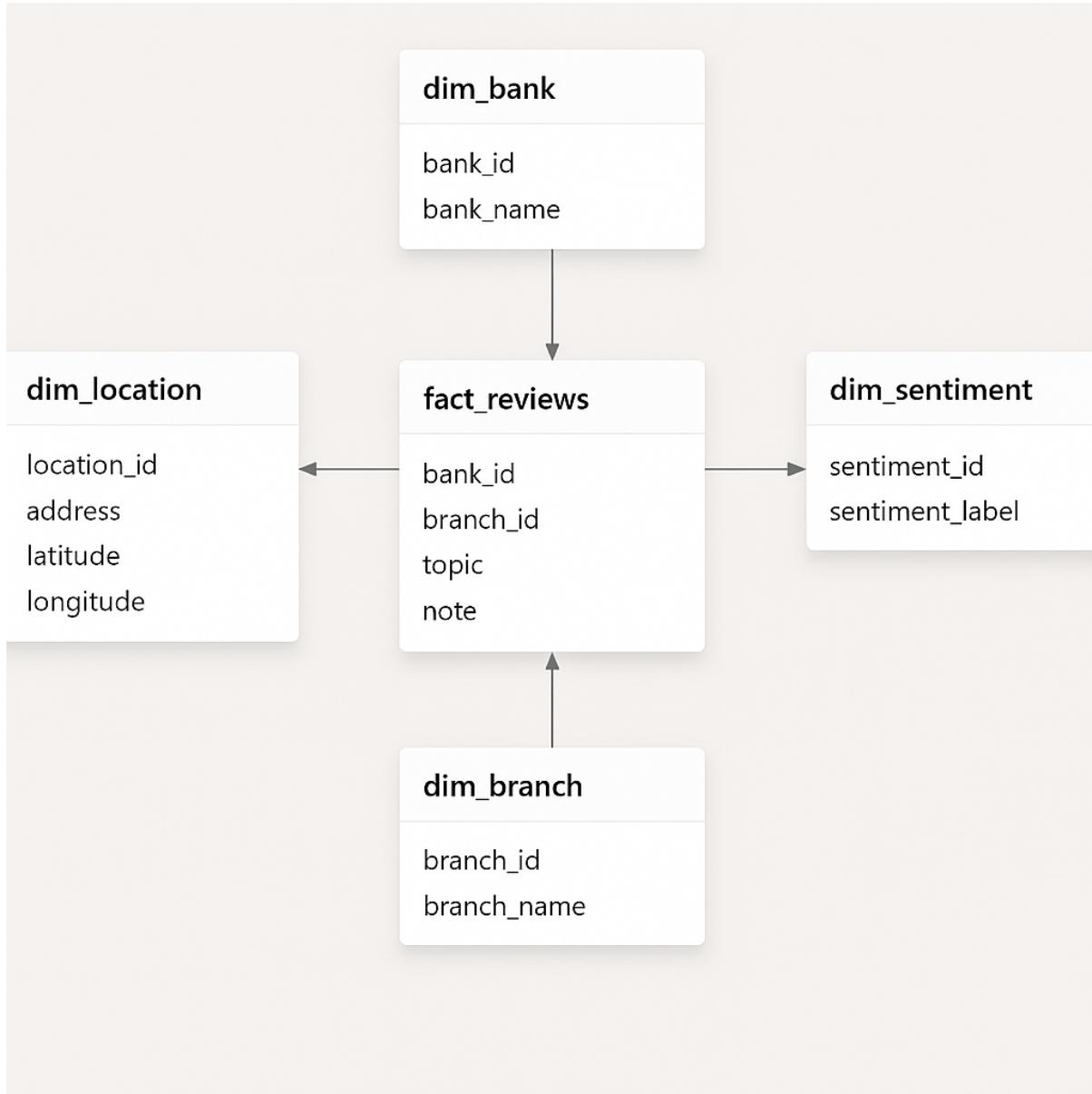
#### 3.1 Objectifs de la modélisation

L'objectif est de :

- Concevoir une **table de faits fact\_reviews\_clean** contenant les mesures principales (la polarité du sentiment, la note de l'avis et le topic du commentaire).
- Créer des **tables de dimensions** décrivant les entités de référence (banque, agence, localisation, sentiment).

- Faciliter les jointures efficaces pour les agrégations et la visualisation.

## 3.2 Architecture du schéma en étoile



La transformation des données se fait en deux étapes :

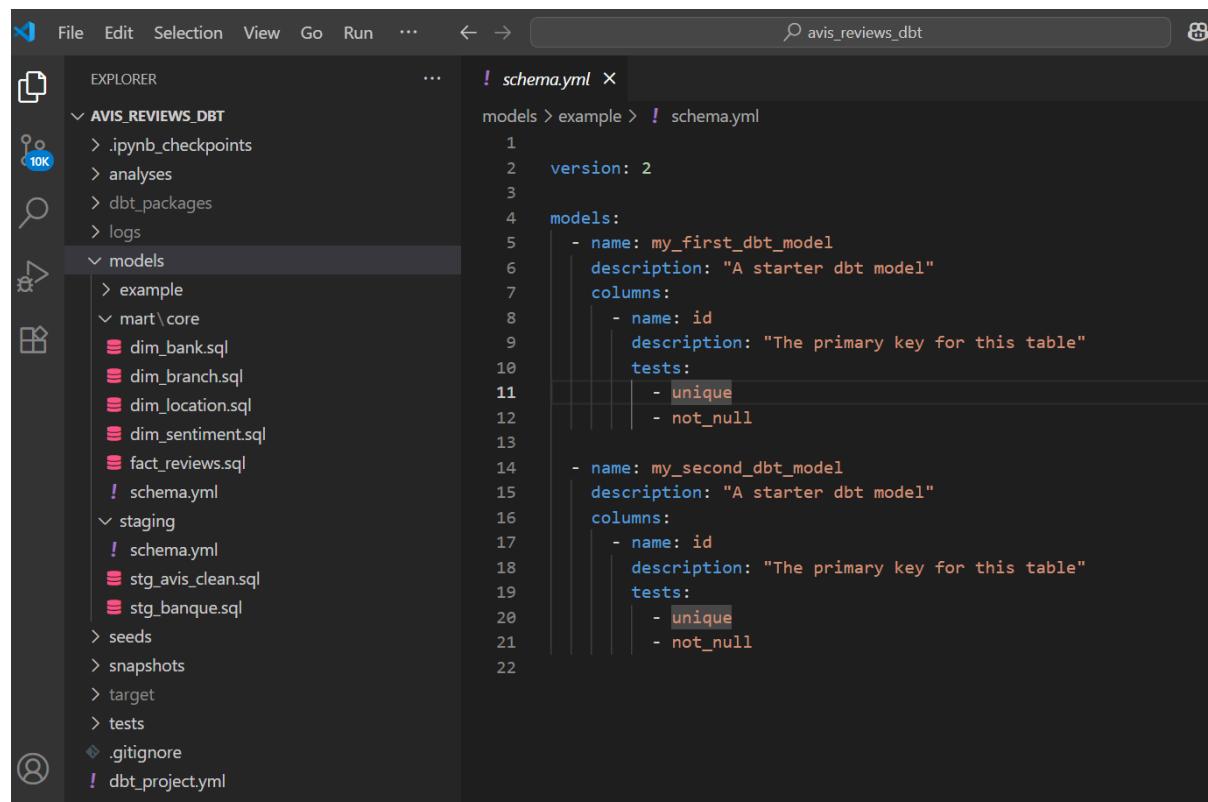
- **Étape 1 – Staging** : Création des vues `stg_avis_enrichi` et `stg_bank`, à partir des sources brutes et de la vue `avis_clean`. Ces vues préparent les données pour la modélisation.
- **Étape 2 – Modélisation** : Construction des modèles dans le dossier `marts/core`, comprenant :
  - `dim_bank` : informations uniques sur les banques (à partir de `stg_bank`) ;
  - `dim_branch` : nom et identifiants des agences ;

- `dim_location` : ville et région extraites des données géographiques ;
- `dim_sentiment` : catégorie de sentiment (positif, neutre, négatif),
- `fact_reviews` :table centrale contenant les identifiants des dimensions et les champs analytiques, liée à toutes les dimensions.

### 3.3 Implémentation des modèles

Les scripts SQL ont permis de créer ces tables à partir de la table `avis_enrichi`. Chaque dimension est alimentée via des requêtes de type `SELECT DISTINCT`, puis la table de faits est construite avec les clés étrangères vers chaque dimension.

L'intégration est ensuite assurée via DBT, garantissant une gestion versionnée, reproductible et automatisable du modèle.



The screenshot shows a code editor interface with a dark theme. On the left is an 'EXPLORER' sidebar showing the project structure:

```

AVIS_REVIEWS_DBT
  > .ipynb_checkpoints
  > analyses
  > dbt_packages
  > logs
  > models
    > example
    > mart\core
      > dim_bank.sql
      > dim_branch.sql
      > dim_location.sql
      > dim_sentiment.sql
      > fact_reviews.sql
      ! schema.yml
    > staging
      ! schema.yml
      > stg_avis_clean.sql
      > stg_banque.sql
  > seeds
  > snapshots
  > target
  > tests
  & .gitignore
  ! dbt_project.yml
  .
  .
  .

```

The main pane displays the contents of the `schema.yml` file:

```

version: 2

models:
  - name: my_first_dbt_model
    description: "A starter dbt model"
    columns:
      - name: id
        description: "The primary key for this table"
        tests:
          - unique
          - not_null

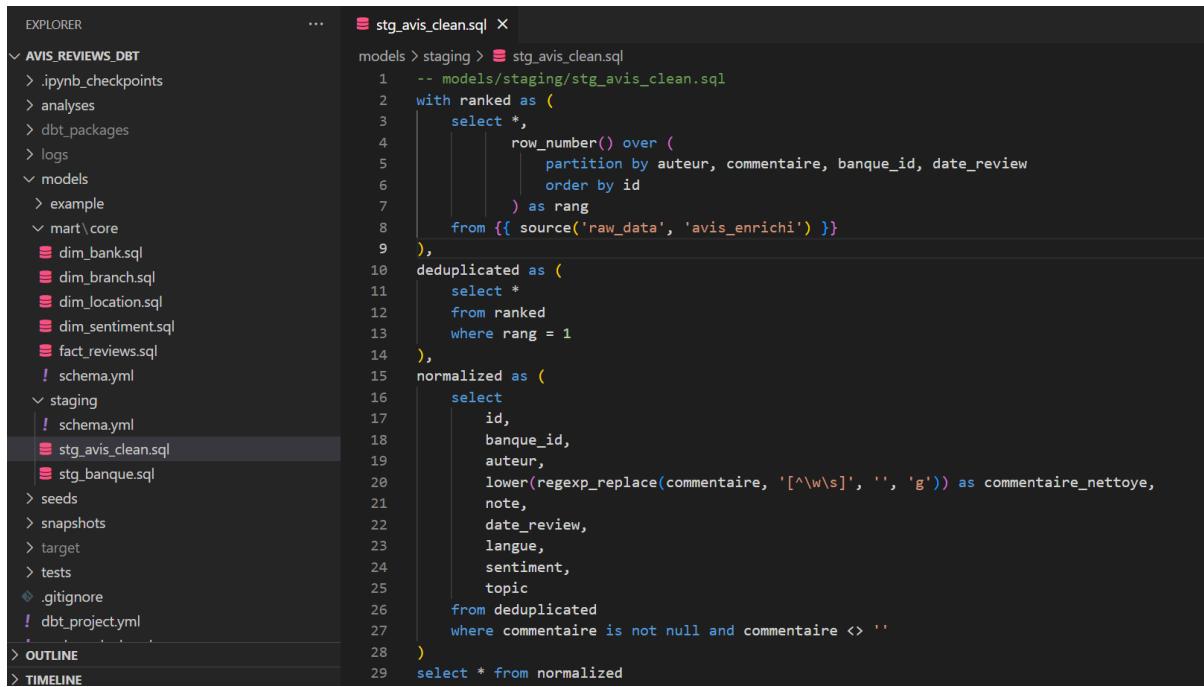
  - name: my_second_dbt_model
    description: "A starter dbt model"
    columns:
      - name: id
        description: "The primary key for this table"
        tests:
          - unique
          - not_null

```

**Figure 3.1** – Structure du projet dbt et fichier du projet schema

## TABLE DES MATIÈRES

---



```

EXPLORER
...
AVIS_REVIEWS_DBT
> ipynb_checkpoints
> analyses
> dbt_packages
> logs
models
> example
  < mrt\ core
    dim_bank.sql
    dim_branch.sql
    dim_location.sql
    dim_sentiment.sql
    fact_reviews.sql
  ! schema.yml
staging
  ! schema.yml
  stg_avis_clean.sql
  stg_banque.sql
seeds
snapshots
target
tests
.gitignore
! dbt_project.yml
> OUTLINE
> TIMELINE

```

models > staging > stg\_avis\_clean.sql

```

1 -- models/staging/stg_avis_clean.sql
2 with ranked as (
3   select *,
4     row_number() over (
5       partition by auteur, commentaire, banque_id, date_review
6         order by id
7     ) as rang
8   from {{ source('raw_data', 'avis_enrichi') }}
9 ),
10 deduplicated as (
11   select *
12   from ranked
13   where rang = 1
14 ),
15 normalized as (
16   select
17     id,
18     banque_id,
19     auteur,
20     lower(regexp_replace(commentaire, '[^\w\s]', '', 'g')) as commentaire_nettoye,
21     note,
22     date_review,
23     langue,
24     sentiment,
25     topic
26   from deduplicated
27   where commentaire is not null and commentaire <> ''
28 )
29 select * from normalized

```

**Figure 3.2** – une source staging

```

(dbt-env) sabrine123@DELL:~/avis_reviews_dbt$ dbt run
19:50:25 Running with dbt=1.9.0-b2
19:50:26 Registered adapter: postgres=1.8.2
19:50:27 Found 9 models, 10 data tests, 2 sources, 537 macros
19:50:27
19:50:27 Concurrency: 1 threads (target='dev')
19:50:27
19:50:27 1 of 9 START sql table model public.my_first_dbt_model ..... [RUN]
19:50:27 1 of 9 OK created sql table model public.my_first_dbt_model ..... [SELECT 2 in 0.12s]
19:50:27 2 of 9 START sql view model public.stg_avis_clean ..... [RUN]
19:50:27 2 of 9 OK created sql view model public.stg_avis_clean ..... [CREATE VIEW in 0.07s]
19:50:27 3 of 9 START sql view model public.stg_banque ..... [RUN]
19:50:27 3 of 9 OK created sql view model public.stg_banque ..... [CREATE VIEW in 0.06s]
19:50:27 4 of 9 START sql view model public.my_second_dbt_model ..... [RUN]
19:50:27 4 of 9 OK created sql view model public.my_second_dbt_model ..... [CREATE VIEW in 0.06s]
19:50:27 5 of 9 START sql view model public.dim_sentiment ..... [RUN]
19:50:27 5 of 9 OK created sql view model public.dim_sentiment ..... [CREATE VIEW in 0.06s]
19:50:27 6 of 9 START sql view model public.dim_bank ..... [RUN]
19:50:27 6 of 9 OK created sql view model public.dim_bank ..... [CREATE VIEW in 0.09s]
19:50:27 7 of 9 START sql view model public.dim_branch ..... [RUN]
19:50:27 7 of 9 OK created sql view model public.dim_branch ..... [CREATE VIEW in 0.04s]
19:50:27 8 of 9 START sql view model public.dim_location ..... [RUN]
19:50:27 8 of 9 OK created sql view model public.dim_location ..... [CREATE VIEW in 0.05s]
19:50:27 9 of 9 START sql view model public.fact_reviews ..... [RUN]
19:50:27 9 of 9 OK created sql view model public.fact_reviews ..... [CREATE VIEW in 0.07s]
19:50:27
19:50:27 Finished running 1 table model, 8 view models in 0 hours 0 minutes and 0.90 seconds (0.90s).
19:50:27
19:50:27 Completed successfully
19:50:27
19:50:27 Done. PASS=9 WARN=0 ERROR=0 SKIP=0 TOTAL=9

```

**Figure 3.3** – création des views réussie sur le terminal

## TABLE DES MATIÈRES

---

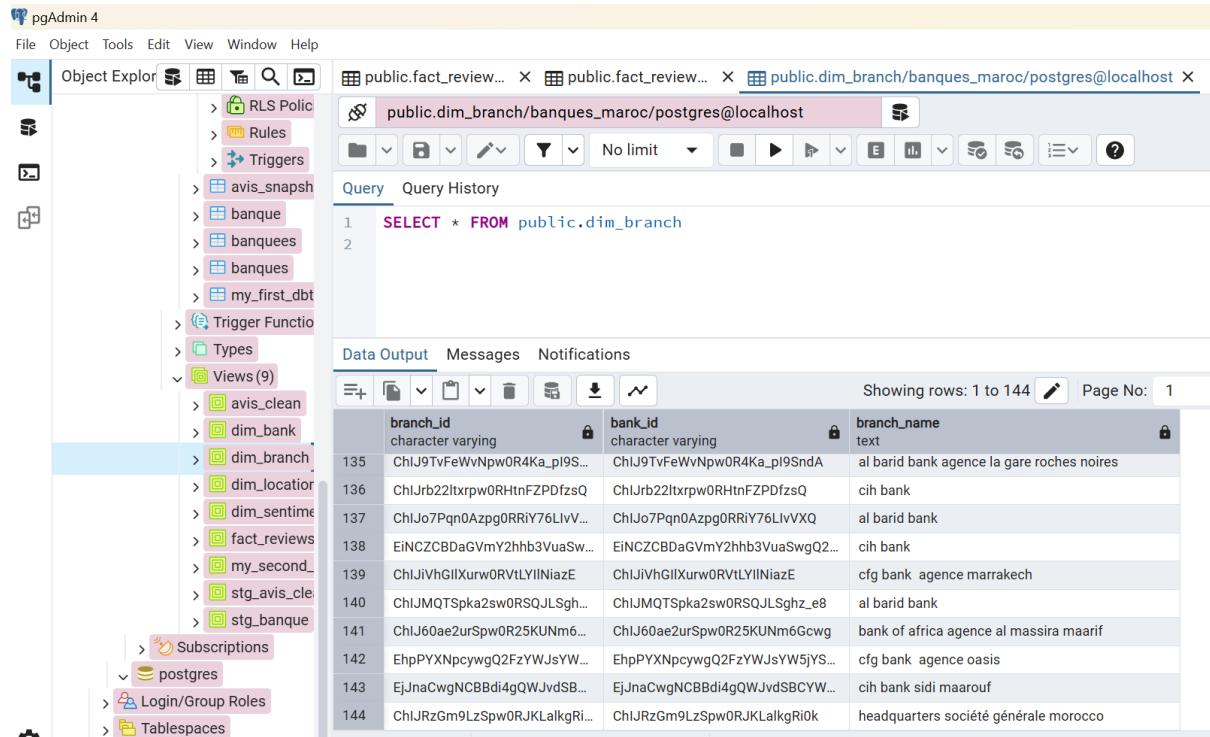


Figure 3.5 – Visualisation des views créées par dbt run sur Pgadmin

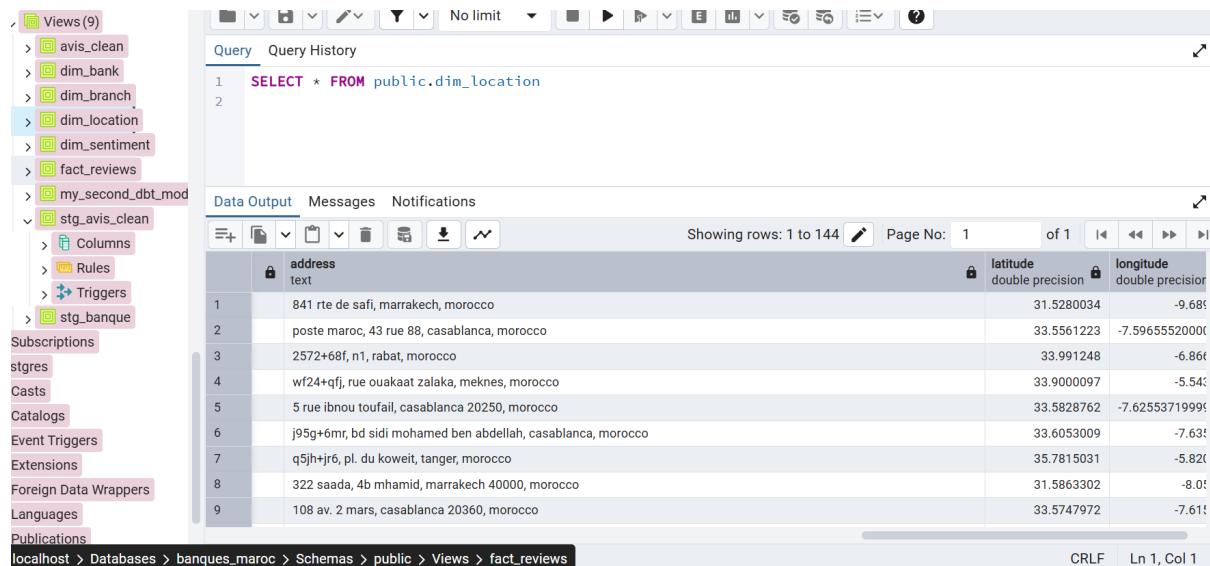


Figure 3.4 – Visualisation des views créées par dbt run sur Pgadmin

## Phase 4 : Modélisation des données (Schéma en étoile dans PostgreSQL)

---

**Listing 7** – Dag complet pour automatiser dbt run pour la modelisation en etoile

---

## TABLE DES MATIÈRES

---

dit View Window Help

public.dim\_location/banques\_maroc/postgres@localhost

Query History

```
1 SELECT * FROM public.dim_location
2
```

Data Output Messages Notifications

Showing rows: 1 to 144 Page No: 1 of 1

location_id	place_id	address	latitude	longitude
135	9bae050687953c271d21c6da28f82...	ChIJ9TvFeWvNpw0R4Ka_p19SndA	place prince sidi mohamed, ...	33.5909142 -7.5923356
136	9ddbeb3e9e5ae8498cf15163418602...	ChIJrb22ltxrpw0RHtnFZPDFzsQ	av. mohamed v, salé, morocco	34.0733067 -6.7668571
137	1cd0c8016df23908becbb57f978703de	ChIJo7Pqn0Azpg0RRriY76LlvVXQ...	hc6f+j9q, casablanca, moro...	33.5945144 -7.6200284
138	a0563d2358a37c5b34a821c6174d94...	EiNCZCBDaGVmYzhb3VuaSwgQ...	alles de la gare, bd chefchao...	33.6222242 -7.5090775
139	4cdbe25471fd6fda70a0c38961b53	ChIJiVhGlXurw0RVtLYIIINiaeE	40 bd mohamed vi, marrake...	31.6068736 -8.0006032
140	7786a7f7e676059fe6a07a0c38961b53	ChIJMQTSpka2sw0RSQJLSghz_e8	cccg+rxr, agadir 80000, mor...	30.4205162 -9.5838532
141	78d1eb0f555ee9fb8a840165cbe0030	ChIJ60ae2urSpw0R25KUNm6Gcwg	rue abou abdelah nafii, casa...	33.5842103 -7.6322071
142	84ae64333ace01c4033cace8e9a6d7...	EhpPYXNpcywqQ2fZYWjsYWSjY...	angle bd abderrahim bouabi...	33.5565296 -7.633413600000001
143	77e5673ed4958335536beeb807ac69...	EjJnaCwgNCBBdi4gQWJvdSBCY...	g8gx+4p3. av. abou bakr el k...	33.5279655 -7.6494997
144	4967e74ab7e505d5f6d3d8ae4e4cedb8	ChIJRzGm9LzSpw0RJKLalkgRi0k	55 bd abdelmoumen, casabl...	33.580191 -7.6248782

Total rows: 144 Query complete 00:00:00.283 CRLF Ln 1, 0

Figure 3.6 – Visualisation des views créées par dbt run sur Pgadmin

dit view window Help

public.dim\_bank/banques\_maroc/postgres@localhost

Query History

```
1 SELECT * FROM public.dim_bank
2
```

Data Output Messages Notifications

Showing rows: 1 to 144 Page No: 1 of 1

bank_id	bank_name	branch_name
1	bmce bank	bmce bank of africa
2	al barid bank	al barid bank bd panoramique
3	al barid bank	agency al barid bank
4	bmce bank	bmce bank yacoub el mansour
5	cfg bank	cfg bank headquarters palm agency
6	bmce bank	bmce bank marina
7	cfg bank	cfg bank
8	al barid bank	al barid bank agence jamaa fena
9	cfg bank	cfg bank agence casa 2 mars
10	banque populaire mar...	banque populaire agence

Figure 3.7 – Visualisation des views créées par dbt run sur Pgadmin

## TABLE DES MATIÈRES

---

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime

default_args = {
    'owner': 'airflow',
    'start_date': datetime(2024, 1, 1),
    'retries': 1
}

with DAG(
    dag_id='dbt_run_pipeline',
    default_args=default_args,
    schedule_interval='@weekly',
    catchup=False,
    description='DAG Airflow pour lancer DBT chaque semaine'
) as dag:

    run_dbt = BashOperator(
        task_id='run_dbt_command',
        bash_command="""
source ~/dbt-env/bin/activate && \
cd /home/Sabrine123/avis_reviews_dbt && \
dbt run
""",
        dag=dag,
    )
```

Dans cette phase, nous avons structuré notre entrepôt de données selon un **schéma en étoile**, afin d'optimiser l'analyse des avis clients bancaires via des outils de visualisation comme *PowerBi*.

L'objectif de cette modélisation est de permettre des analyses multidimensionnelles sans nécessiter de requêtes SQL complexes. Par exemple, il est désormais possible de visualiser directement dans PowerBI :

- les **topics dominants** par banque ou par agence,
- les **scores de polarité** moyens par banque ou par agence,
- le **nombre d'avis négatifs** selon la localisation géographique,

Grâce à ce modèle, l'import des données dans PowerBi permet une grande souplesse dans la création de tableaux de bord interactifs, facilitant l'exploration des retours clients par les parties prenantes.

## 4 Phase 5 : Visualisation des données — Dashboards

Dans cette phase finale du projet, nous avons conçu une série de visualisations interactives à l'aide de Power BI afin de valoriser les données enrichies issues du pipeline Airflow et du modèle en étoile construit dans PostgreSQL.

### 4.1 Connexion au Data Warehouse

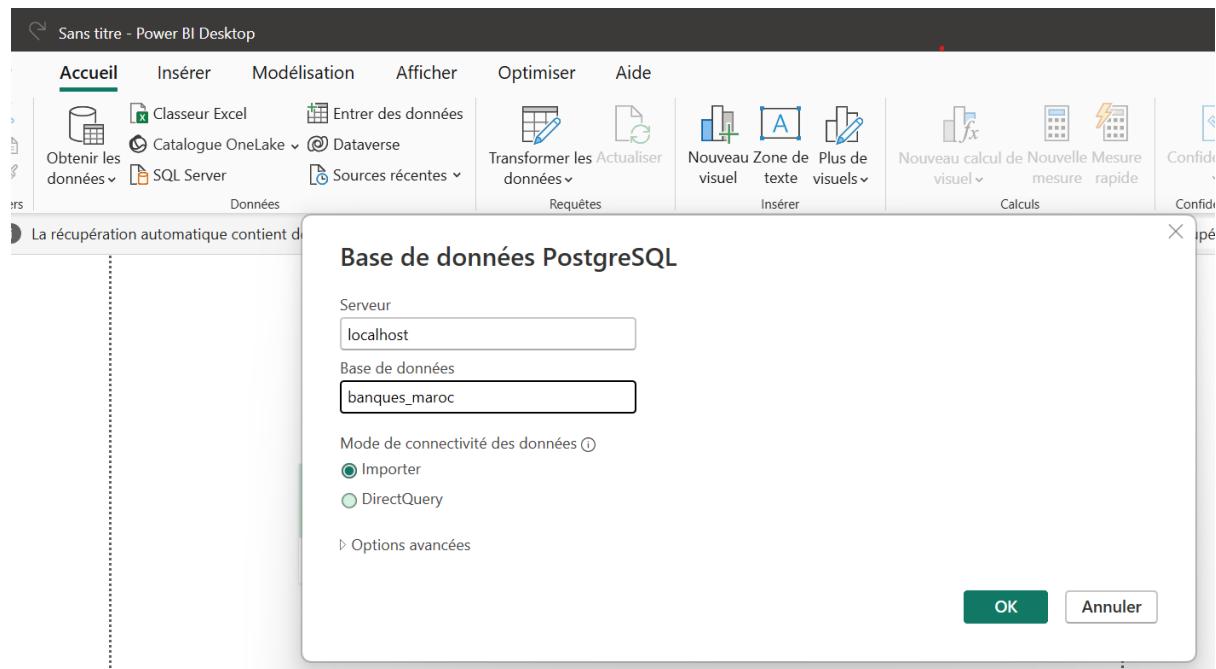
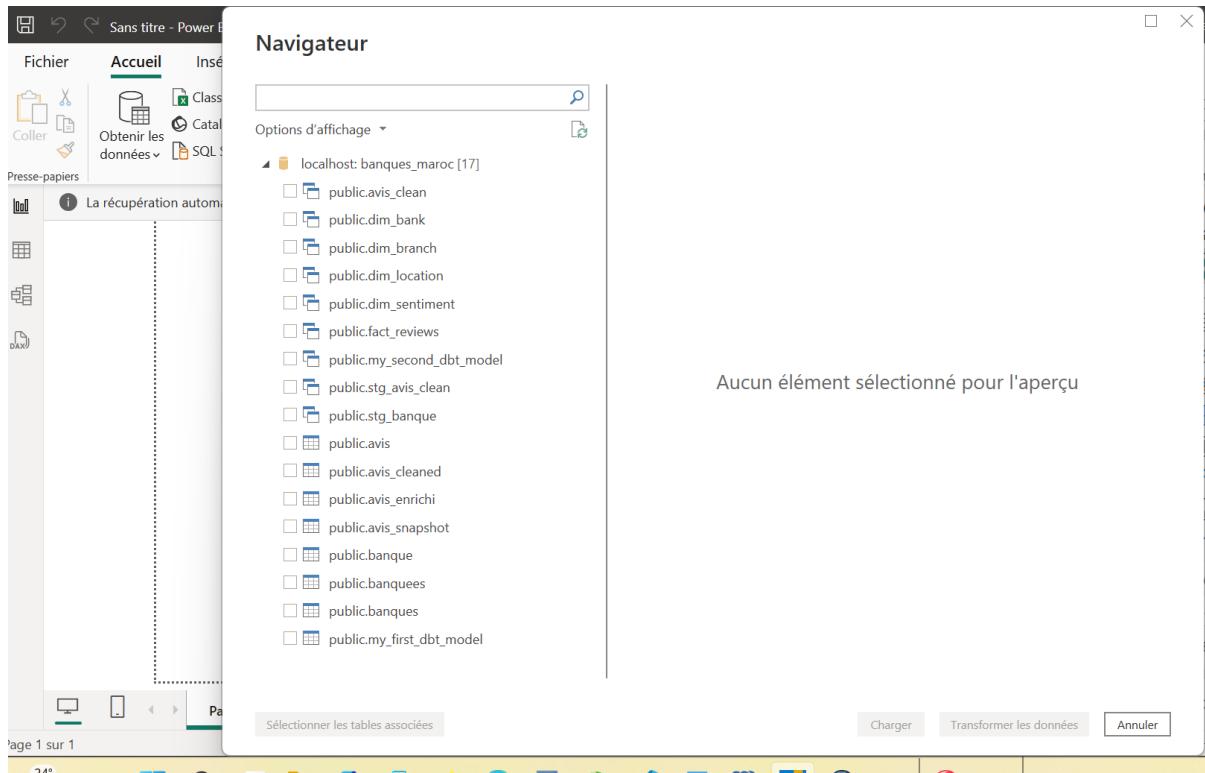


Figure 4.1 – Connexion a la base PostgreSql

## TABLE DES MATIÈRES

---



**Figure 4.2 – Connexion a la base PostgreSql**

La base de données PostgreSQL `banques_maroc` a été connectée à Power BI via une connexion directe à `localhost`, permettant l'importation des tables de dimension ainsi que de la table de faits `avis_enrichi`. Les relations entre les entités (`bank`, `branch`, `sentiment`, `topic`) ont été reconstituées dans Power BI via des jointures de type (1 :N).

## TABLE DES MATIÈRES

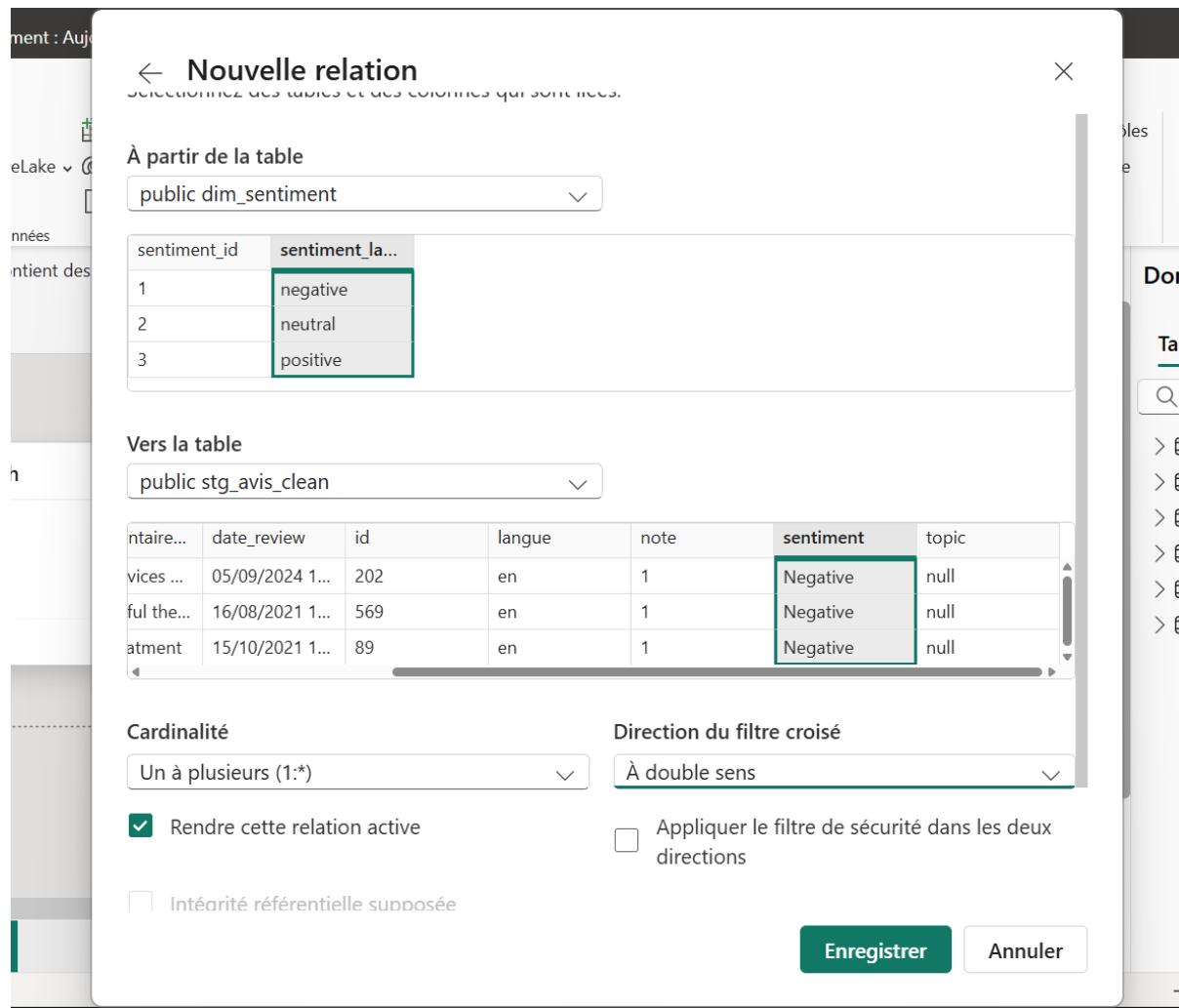


Figure 4.3 – Jointure des tables

## 4.2 Objectifs de la visualisation

Les tableaux de bord ont été conçus avec plusieurs objectifs :

- Visualiser la répartition des sentiments exprimés par les clients.
- Identifier les *topics* (thèmes dominants) abordés dans les commentaires.
- Localiser les agences associées à des avis négatifs.
- Analyser les notes des utilisateurs selon différentes dimensions (banque, région, thème, etc.).

## 4.3 Types de visualisations réalisées

Plusieurs éléments visuels ont été mis en place :

- **Diagrammes à barres** : nombre d'avis par sentiment, par banque, par topic.
- **Graphiques circulaires (camemberts)** : répartition des sentiments ou des thèmes.

## TABLE DES MATIÈRES

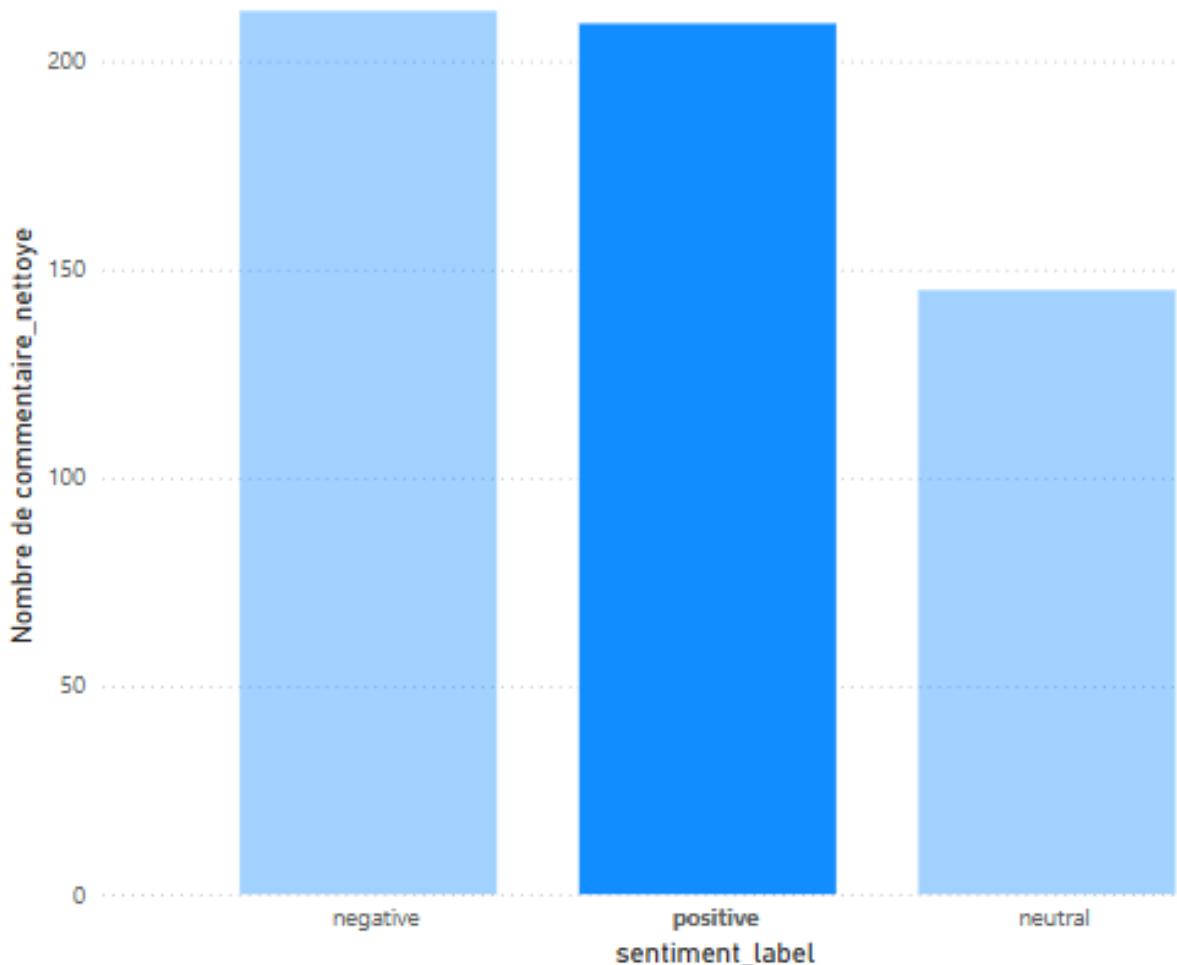
---

- **Cartes géographiques** : position des agences selon les coordonnées GPS.
- **KPI Cards** : nombre total d'avis, pourcentage de sentiments négatifs, note moyenne, etc.

## 4.4 Exemples concrets de graphiques

- Un histogramme représentant le nombre de commentaires positifs, neutres et négatifs.

**Nombre de commentaire\_nettoye par sentiment\_label**

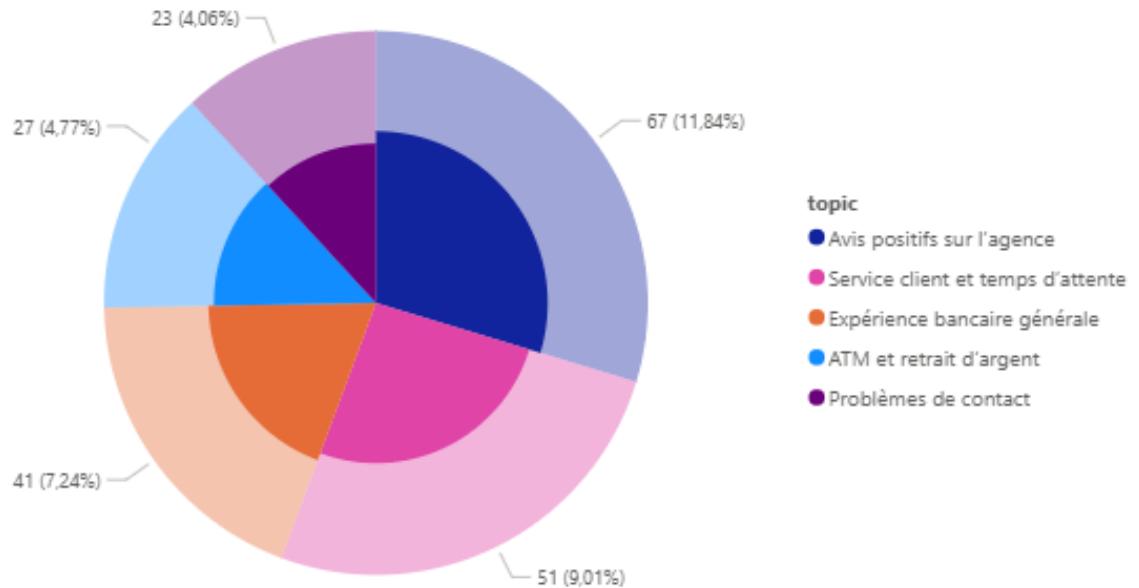


- Un graphique en camembert illustrant la répartition des topics évoqués.

## TABLE DES MATIÈRES

---

### Nombre de commentaire\_nettoye par topic



**Figure 4.4 – Répartition des sujets abordés dans les avis**

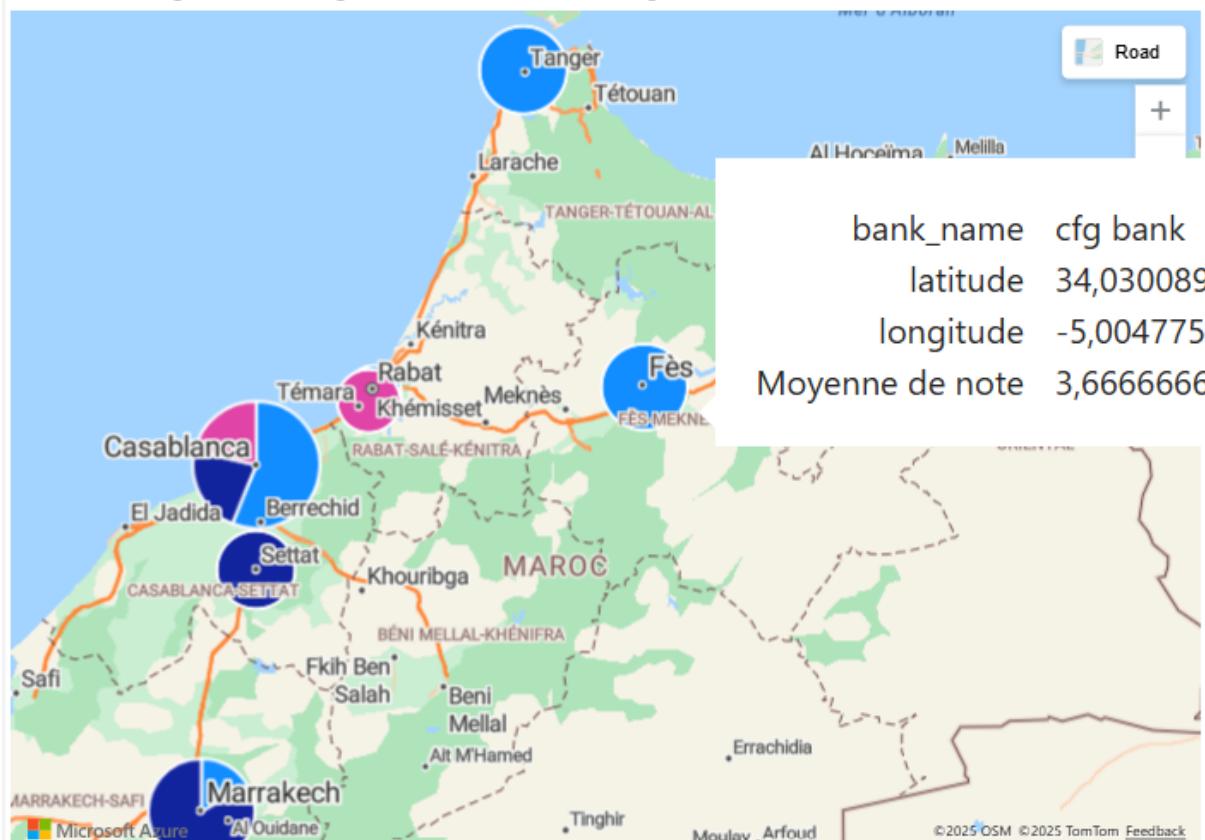
- Une carte géographique montrant les agences les plus critiquées ou les mieux notées.

## TABLE DES MATIÈRES

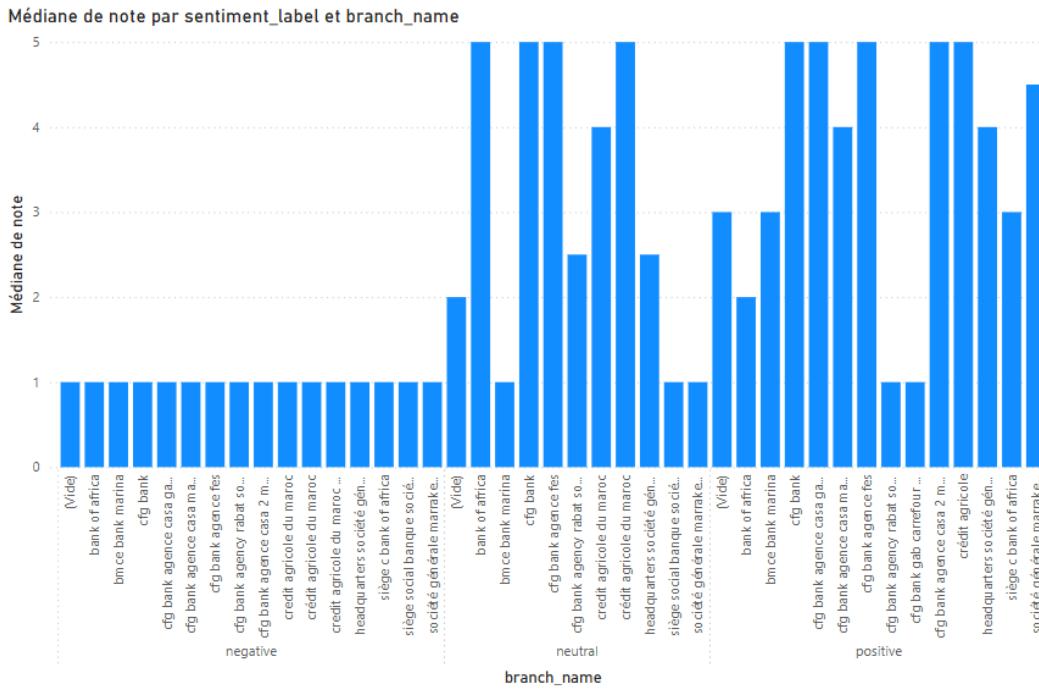
---

### Moyenne de note par latitude, longitude et bank\_name

bank\_name ● cfg bank ● crédit agricole maroc ● (Vide) ● société générale maroc ● bmce bank



## 4.5 AXE D'ANALYSE PERTINENT : carte de répartition des agences avec la note des avis



La carte ci-dessous représente la répartition géographique des agences bancaires au Maroc, avec une indication visuelle de leur performance maximale (note maximale) par agence. Il s'agit d'une carte interactive de type *bubble map*, générée via Power BI, où chaque bulle représente une agence localisée à ses coordonnées géographiques (latitude et longitude).

### 5..1 Observations principales :

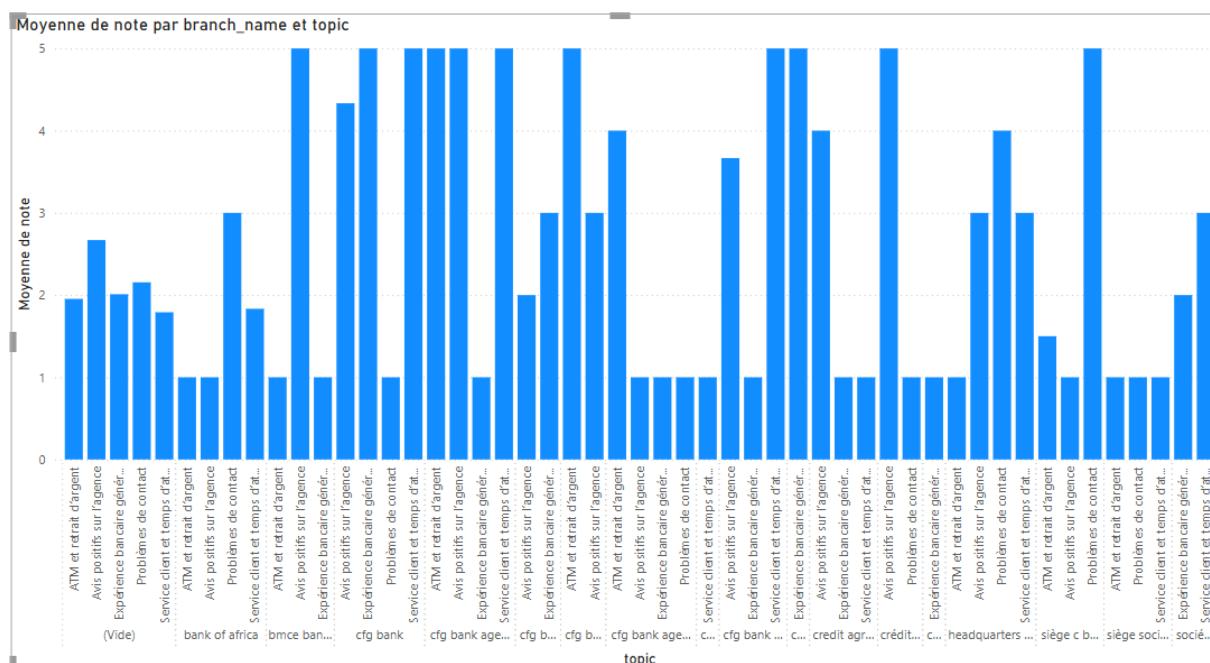
- **Casablanca** présente une concentration particulièrement élevée d'agences bancaires, avec plusieurs banques concurrentes représentées. Les performances (tailles des bulles) sont généralement fortes.
- **La région Rabat – Kénitra – Témara** montre également une forte densité d'agences bien notées.
- **Tanger, Fès et Marrakech** disposent chacune de plusieurs agences avec de bonnes performances visibles.

## TABLE DES MATIÈRES

## 5..2 Conclusion :

Cette visualisation permet de constater une forte implantation des principales banques marocaines dans les grands pôles urbains et économiques. Elle met aussi en évidence une concurrence importante dans certaines zones (Casablanca, Rabat), ainsi que des écarts de performance entre agences.

4.6 AXE D'ANALYSE PERTINENT :  
Moyenne de note par branch\_name et topic



Le graphique présente la **moyenne des notes** par agence bancaire (`branch_name`) et par sujet de retour client (`topic`). Il s'agit d'un histogramme en barres verticales.

## Structure du graphique

- **Axe des abscisses ( $x$ )** : combinaison de `branch_name` et `topic`, par exemple : “*cfg bank - ATM et retrait d’argent*”.
  - **Axe des ordonnées ( $y$ )** : moyenne des notes attribuées, allant de 1 à 5.
  - **Type de graphique** : Histogramme vertical.

## Observations générales

- **Hétérogénéité importante** : Certaines combinaisons agence + topic atteignent la note maximale de 5 (par exemple : “*cfg bank - Avis positifs sur l'agence*”), tandis que d’autres tombent proches de 1. Cela met en évidence des écarts de satisfaction significatifs entre agences et thématiques.
- **Thématiques récurrentes** : Les sujets les plus fréquents sont :
  - ATM et retrait d’argent
  - Avis positifs sur l’agence
  - Expérience bancaire générale
  - Problème de contact
  - Service client et temps d’attente
- **Problèmes fréquents** :
  - Le problème de contact est souvent noté autour de 1 ou 2.
  - Le service client et le temps d’attente suscitent également de nombreuses notes faibles dans plusieurs agences.

## Analyse par agence

- **cfg bank** : Présente des notes excellentes sur plusieurs topics, souvent proches de 5. Performance globalement très satisfaisante.
- **bmce bank** : Notes plutôt faibles ou moyennes, indiquant une marge d’amélioration notable.
- **crédit agricole** : Des notes très variées, allant de 1 à 5. Performance inégale selon les sujets.
- **Agences anonymes ((Vide))** : Moyenne autour de 2 à 3, ce qui reste modeste. Cela peut refléter un manque d’information ou une qualité de service non homogène.
- **headquarters / siège central** : Notes généralement basses. Ce point mérite une attention particulière car il concerne une entité stratégique.

## Points forts identifiables

Certaines notes maximales sur les topics comme **Avis positifs sur l’agence** ou **Expérience bancaire générale** révèlent des cas de **très forte satisfaction client**. Ces retours positifs doivent être valorisés et généralisés si possible.

## Axes d'amélioration

- **Standardisation du service client** : Les notes très basses sur les thématiques liées au **temps d'attente** et au **contact client** sont des signaux clairs. Il est recommandé de renforcer les procédures et la formation des équipes.

Ce graphique permet d'**identifier les points d'excellence** (notamment dans certaines agences comme **cfg bank**) mais surtout de **repérer des faiblesses récurrentes** sur des sujets critiques comme le service client. Il constitue un outil pertinent pour un **diagnostic opérationnel** et l'orientation d'actions correctives ciblées.

## 4.7 Intérêt de l'approche

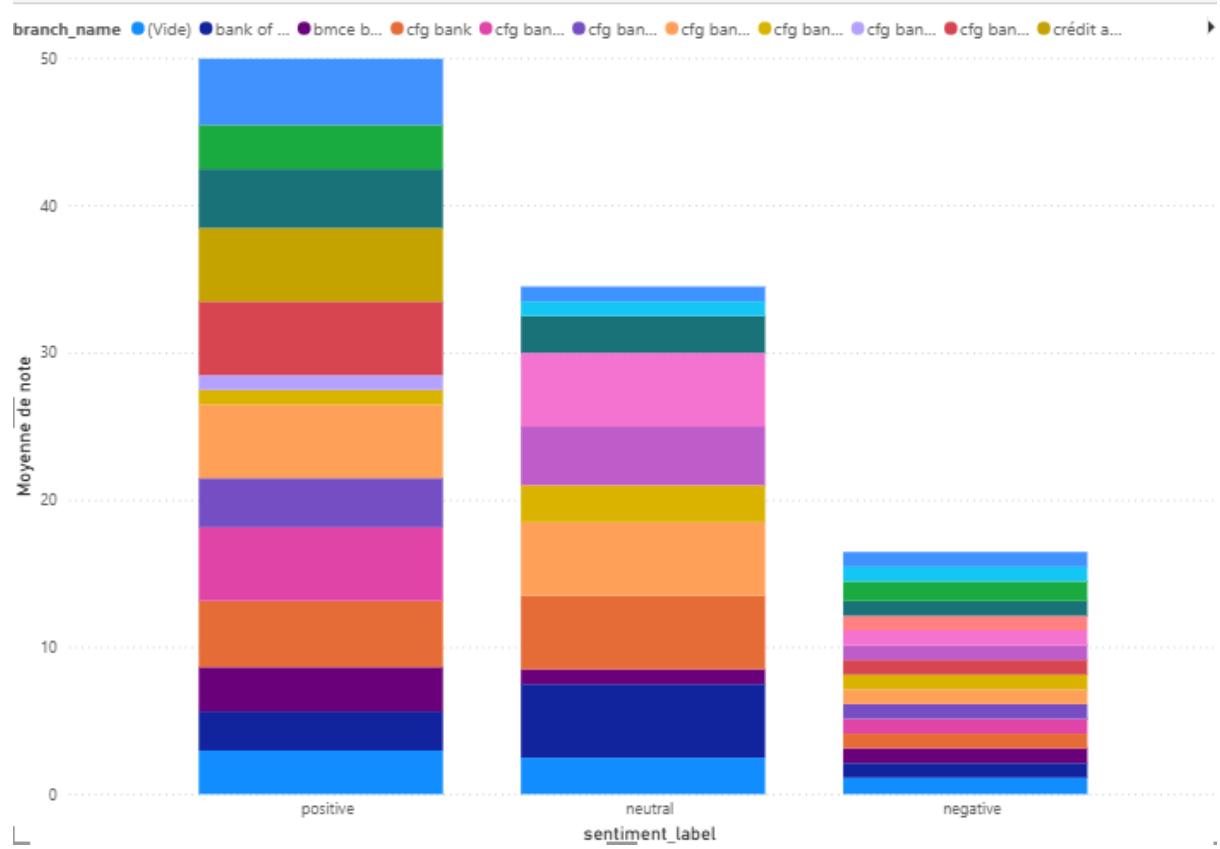
L'intégration de Power BI à notre pipeline de données offre une lecture intuitive et dynamique des avis clients. Grâce au modèle en étoile, les analyses croisées (par banque, par région, par sentiment, etc.) sont facilement réalisables, ouvrant la voie à un pilotage plus fin de la satisfaction client dans le secteur bancaire marocain.

## 4.8 Autres graphiques pertinents



## TABLE DES MATIÈRES

---



## TABLE DES MATIÈRES

---

### |Max de note par branch\_name

