epsi

l'école d'ingénierie informatique

CERTIFICATION DÉVELOPPEUR EN INTELLIGENCE ARTIFICIELLE ET DATA SCIENCE RNCP 36581

BLOC DE COMPÉTENCES 1 : Créer un modèle de données d'une solution I.A en utilisant des méthodes de Data sciences

Développement et déploiement d'une application dans le respect du cahier des charges Client - Création d'un backend métier permettant le nettoyage et la visualisation des données.

Florent BACCARD | Tojoniaina RATSIMBAZAFY | Gaël SCHENCK | Alessia TORNABENE

SOMMAIRE

Introduction

Afin de répondre aux exigences de la Mise en Situation Professionnelle Reconstituée TPRE501 et de poser les bases d'une solution basée sur l'intelligence artificielle, nous avons choisi d'analyser les données liées à l'épidémie du VIH (Virus de l'Immunodéficience Humaine). Ce virus s'attaque au système immunitaire et réduit les défenses naturelles de l'organisme, rendant les individus plus vulnérables aux infections et maladies opportunistes.

L'objectif de ce projet est d'exploiter ces données afin de mieux comprendre leur structure, d'en extraire des informations pertinentes et de proposer une solution facilitant leur analyse et leur visualisation. Pour cela, nous avons structuré notre travail en plusieurs étapes clés :

➢ **Modélisation de la base de données (BDD) et conception UML :** Cette phase vise à définir l'architecture des données, en modélisant les relations entre les différentes entités afin d'assurer une organisation cohérente et optimisée pour les traitements à venir.

➢ **Mise en place d'un processus ETL (Extract, Transform, Load) :** Cette étape consiste à récupérer les jeux de données, les nettoyer et les transformer en formats exploitables, avant de les charger dans une base de données. Un travail de normalisation a été mené pour garantir la qualité des informations utilisées.

➢ **Développement d'une API pour l'obtention et la manipulation des données :** Afin de faciliter l'accès aux données et d'assurer leur exploitation par différents systèmes, nous avons mis en place une API permettant d'interroger la base de données de manière efficace. Une documentation détaillée accompagne cette API pour en garantir une utilisation optimale.

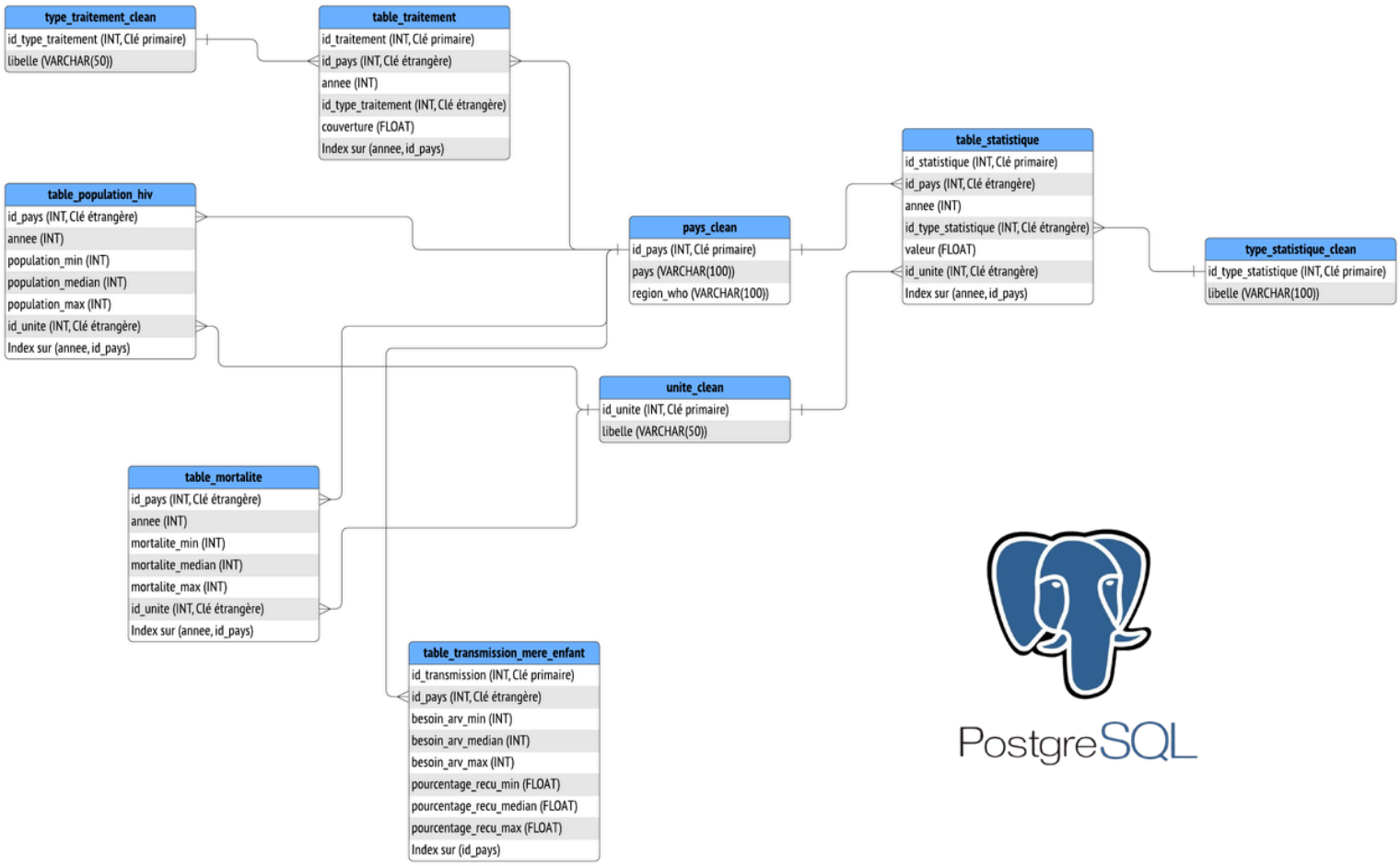
➢ **Visualisation des données avec Power BI :** Pour rendre les résultats plus accessibles et compréhensibles, nous avons utilisé Power BI afin de créer des dashboards interactifs. Un benchmark des différentes solutions de visualisation a été effectué pour justifier ce choix, et les visualisations mises en place permettent d'explorer les tendances clés et d'identifier des informations stratégiques issues des données.

Ce rapport détaille chacune de ces étapes en mettant en avant les choix méthodologiques, les outils utilisés et les résultats obtenus.

Modèle de BDD, UML et script

I. Modèle de la BDD

Après analyse et découpage préalable des jeux de données en notre possession, nous avons élaboré un modèle théorique de notre base de données. Dans un premier temps nous avons cherché à savoir quels seraient les éléments les plus importants et comment les organiser entre eux afin de garantir une cohérence de la donnée. Nous avons donc isolé dans une première strate les données concernant les pays, les statistiques et les traitements, qui sont ressortis à nos yeux comme les valeurs les plus importantes. Dans un second temps nous avons réunis dans une seconde strate les types de traitements, la population de personnes atteintes du VIH, les éléments liés à la mortalité, les données sur la transmission mère-enfant, le type de statistique et enfin les unités de ces mêmes statistiques.



II. UML (MCD, MLD, MPD)

Modèle logique de données (MLD)

pays_clean (id_pays PK, pays, region_who)
unite_clean (id_unite PK, libelle)
type_statistique_clean (id_type_statistique PK, libelle)
type_traitement_clean (id_type_traitement PK, libelle)

table_population_hiv (id_pays FK → pays_clean, annee, population_min, population_median, population_max, id_unite FK → unite_clean)

table_mortalite (id_pays FK → pays_clean, annee, mortalite_min, mortalite_median, mortalite_max, id_unite FK → unite_clean)

table_transmission_mere_enfant (id_transmission PK, id_pays FK → pays_clean, besoin_arv_min, besoin_arv_median, besoin_arv_max, pourcentage_recu_min, pourcentage_recu_median, pourcentage_recu_max)

table_traitement (id_traitement PK, id_pays FK → pays_clean, annee, id_type_traitement FK → type_traitement_clean, couverture)

table_statistique (id_statistique PK, id_pays FK → pays_clean, annee, id_type_statistique FK → type_statistique_clean, valeur, id_unite FK → unite_clean)

Modèle Physique de Données (MPD - SQL) *(complet, voir annexe)*

```
CREATE TABLE pays_clean (
   id_pays INT PRIMARY KEY AUTO_INCREMENT,
   pays VARCHAR(100),
   region_who VARCHAR(100)
);

CREATE TABLE unite_clean (
   id_unite INT PRIMARY KEY AUTO_INCREMENT,
   libelle VARCHAR(50)
);

CREATE TABLE type_statistique_clean (
   id_type_statistique INT PRIMARY KEY AUTO_INCREMENT,
   libelle VARCHAR(100)
);
```

Suite Annexe Page - 16 - 17

ETL : Gestion et traitement des jeux de données

I. Dataset d'entrée

Nous avons utilisé un set de données trouvé sur kaggle. Il comporte les informations relatives au virus du VIH, selon deux sources fiables, l'OMS (WHO en Anglais) et l'Unesco réparties en six fichiers CSV.

II. Benchmark

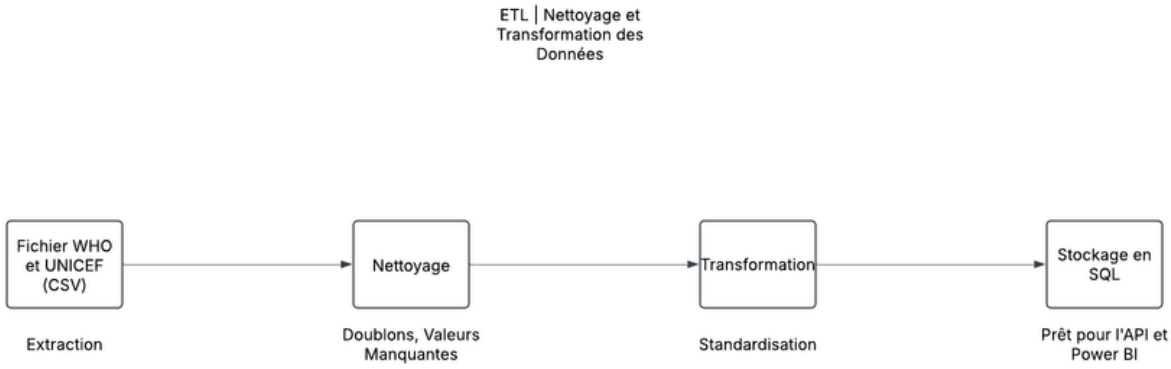
Afin de mettre en place un système de traitement nous avons évalué les diverses possibilités qui s'offrent à nous. Nous avons comparé Pandas, Talend, Apache HOP ainsi qu'Airflow, selon les 5 critères suivants : la performance de traitement des flux de données, la simplicité d'utilisation, la scalabilité (manière dont le système s'adapte aux modulations des volumes exploités), la compatibilité du système avec divers formats de documents, et enfin le coût que représente l'utilisation du logiciel. (voir annexe)
Nous avons jeté notre dévolu sur l'utilisation de Pandas, qui nous permet de travailler avec la même technologie que notre API, mais surtout qui correspond plus à nos besoins, afin de produire un résultat cohérent avec l'ensemble du projet. Le fait qu'il soit Open Source a été un facteur décisionnel étant donné que la communauté Python est très active et permet de trouver facilement des informations en cas de besoin.

Benchmark des solutions de collecte et transformation des données (ETL)

Critères	Pandas (choisi)	Talend	Apache Hop	Airflow
Performance	Excellente pour les données de taille moyenne, optimisable avec Dask	Bonne pour de gros volumes de données	Adapté aux projets ETL modulaires	Excellente pour orchestrer des workflows
Facilité d'utilisation	Simple pour les développeurs Python	Interface no-code, prise en main rapide	Interface graphique et scriptable	Plus complexe à configurer
Scalabilité	Nécessite Dask/Spark pour très gros volumes	Adapté aux grands volumes	Bonne scalabilité	Idéal pour le big data
Compatibilité	JSON, CSV, SQL, API	Nombreux connecteurs (BD, API, fichiers)	Connecteurs variés	Forte intégration avec le cloud
Coût	Gratuit et Open Source	Version gratuite + payante	Open Source	Open Source

III. Processus de nettoyage

Nous avons entamé le processus de nettoyage en prenant les différentes tables en compte et en les normalisant, pour cela nous avons renommé les colonnes comportant le pays dans chaque fichier, ainsi que supprimer les espaces. Chaque colonne a vu son nom être transformé en minuscules. Certains fichiers comportaient aussi une mention de la région géographique, qui a subi le même traitement.
Chacune de ces colonnes a aussi été scrutée par la fonction drop_duplicates() afin d'écarter les doublons. Nous avons organisé les pays en un seul fichier qui sert de référence à chaque autre fichiers inclus dans le set de base, ce qui nous permet par la suite de traiter chaque fichier afin qu'il corresponde à ce dernier et nous a permis d'éliminer les données non conformes, qui n'avaient pas d'informations suffisantes ou qui n'avaient pas les bons ID de pays par exemple.
Les données concernant les unités ont été transformées en table afin d'offrir un indice et une clé spéciale les concernant dans notre base de données. Un traitement similaire a été appliqué aux fichiers concernant les types de statistique (taux de HIV, taux de mortalité, et nombre de cas) ainsi qu'aux types de traitements (adultes ou enfants).



IV. Traitement et réorganisation

Nous avons commencé notre travail de traitement par la réunion des données en fonction de la table des pays qui a été mise en place grâce au nom de pays, son id, et la région de l'OMS à laquelle il appartient. Des fichiers comportent des données chiffrées associées à un indice, il nous a fallu les séparer afin de pouvoir conserver des valeurs exploitables. Toutes les colonnes ont aussi été typées afin que les données soient le plus cohérentes possible avec le format qu'elles étaient censées avoir.
Chaque colonne a bien entendu été renommée afin de garantir une unité dans les champs que nous explorerons par la suite au sein du projet. Chaque donnée non complète a été écartée, afin d'utiliser un jeu de données final complet.

```sql
-- ================================================================================
-- Script : create_tables.sql
-- Description :
-- Ce script crée toutes les tables nécessaires dans la base de données :
-- - Tables de référence (pays, unités, types)
-- - Tables de données (population_hiv, mortalité, transmission, traitements)
-- - Tables de statistiques
-- Il définit également toutes les contraintes et relations entre les tables.
-- ================================================================================

-- Création des tables de référence
CREATE TABLE IF NOT EXISTS pays (
    id_pays SERIAL PRIMARY KEY,
    nom_pays VARCHAR(100) NOT NULL,
    region VARCHAR(100),
    sous_region VARCHAR(100)
);

CREATE TABLE IF NOT EXISTS unite (
    id_unite SERIAL PRIMARY KEY,
    nom_unite VARCHAR(50) NOT NULL
);

CREATE TABLE IF NOT EXISTS type_statistique (
    id_type_statistique SERIAL PRIMARY KEY,
    nom_type_statistique VARCHAR(100) NOT NULL
);
```

Le fichier create_tables.sql est un script SQL essentiel pour l'initialisation de la base de données utilisée dans notre projet. Son rôle principal est de structurer la base de données en créant toutes les tables nécessaires et en définissant les relations entre elles.

Ce script automatise la création de la base de données et garantit une structure solide pour stocker, organiser et analyser les données du projet. Il permet :
➢D'assurer l'intégrité et la cohérence des informations.
➢D'optimiser la relation entre les données grâce aux clés étrangères.
➢De faciliter l'exploitation des données pour les analyses et les visualisations, notamment dans Power BI.

En exécutant ce script au démarrage du projet, on s'assure que la base de données est correctement initialisée et prête à recevoir les jeux de données nécessaires.

API : Obtention et manipulation des données

I. Mise en place de l'API

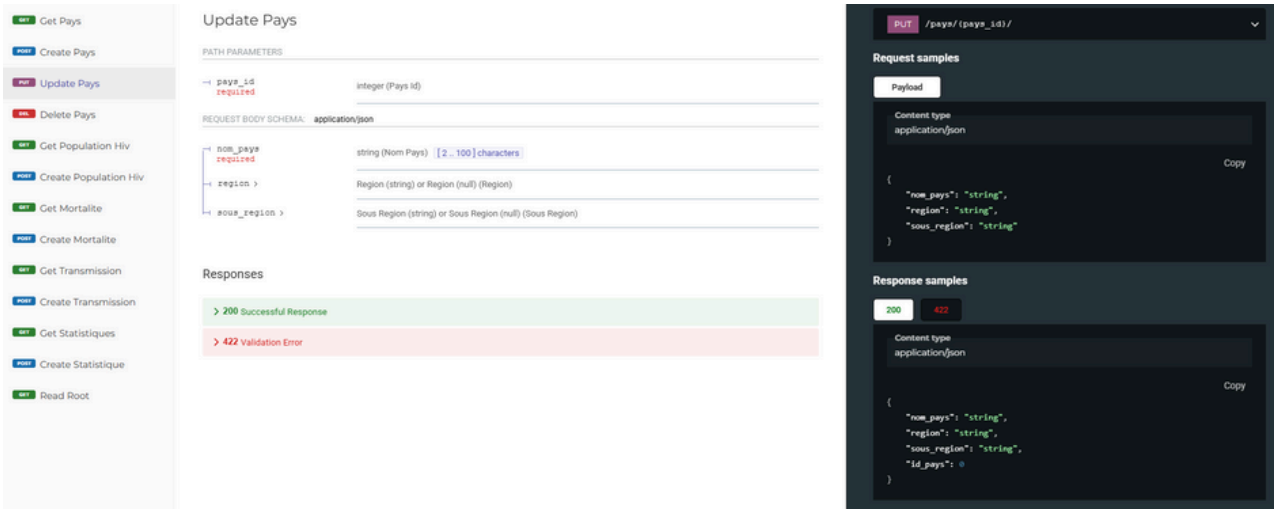
Nous avons choisi Python comme environnement afin de déployer notre API, étant l' un des langages de programmation les plus populaires et bénéficiant de nombreuses ressources en ligne grâce à sa communauté très développée. Le fait que ce langage soit flexible et adaptatif, doté d'une syntaxe claire et facile à mettre en œuvre, ainsi que le langage phare dans le domaine de l'IA et de l'analyse de données ont été des facteurs décisifs dans notre choix. (voir
annexe)
Dans un premier temps nous nous sommes appuyés sur nos schémas de base de données afin de concevoir l'architecture de notre API. Nous avons utilisé des dépendances simples d'utilisation, afin de garantir une maintenabilité relativement aisée du code (FastAPI, SQLAlchemy, Pydantic, PyMySQL, python-dotenv) et capable de corréler les informations que nous avons stocké en BDD
(MySQL).18
Nous avons défini les modèles (SQLAlchemy) pour représenter les tables de la BDD enconfigurant les clés primaires et auto-incrémentées afin d'éviter les erreurs de duplication. Nous y avons associé des schémas Pydantic afin de valider les données entrantes et sortantes pour s'assurer de la qualité des informations retournées.
Enfin, nous avons mis en place un ensemble de chemins d'accès (endpoints) qui respectent le fonctionnement d'un CRUD (Create, Read,
Update, Delete) afin que les utilisateurs puissent se servir de l'api pour obtenir et/ou manipuler
les informations que contiennent chaque table.

II. Documentation de l'API

L'API dispose d'une documentation interactive générée par FastAPI, que l'on peut consulter via Swagger ou ReDoc.
Vous trouverez plus de détails sur son fonctionnement dans le document joint appelé « Documentation API » ainsi que des exemples visuels détaillés fournis en annexe.

default

GET	`/pays/`	Get Pays
POST	`/pays/`	Create Pays
PUT	`/pays/{pays_id}/`	Update Pays
DELETE	`/pays/{pays_id}/`	Delete Pays
GET	`/population_hiv/`	Get Population Hiv
POST	`/population_hiv/`	Create Population Hiv
GET	`/mortalite/`	Get Mortalite
POST	`/mortalite/`	Create Mortalite
GET	`/transmission/`	Get Transmission
POST	`/transmission/`	Create Transmission
GET	`/statistiques/`	Get Statistiques
POST	`/statistiques/`	Create Statistique



PowerBI : Solution de visualisation

I. Benchmark

Afin de visualiser les informations retournées par nos ETL, nous avons soumis quatre logiciels à
notre banc d'essai. Nous avons comparé Power BI, Tableau, Grafana et Matplotlib selon les 20 critères de performance, leur facilité d'utilisation, l'interactivité du produit, sa compatibilité avec les différents formats de fichiers, et enfin leur coût. Nous avons orienté notre choix sur PowerBi qui répond totalement à nos attentes. Le fait qu'il soit compatible avec les données que nous utilisons associé à un coût d' utilisation faible et modulable selon les besoins et les usages, ainsi qu'une facilité d' utilisation grâce à son interface simple et pratique a achevé de nous convaincre.

Benchmark des solutions de visualisation des données

Critères	Power BI (choisi)	Tableau	Grafana	Matplotlib / Seaborn
Performance	Excellente pour des grands jeux de données	Très puissant, bien optimisé	Bon pour données en temps réel	Dépend des capacités Python
Facilité d'utilisation	Interface intuitive avec glisser-déposer	Interface puissante mais plus complexe	Simple pour monitoring	Requiert du code Python
Interactivité	Tableaux de bord interactifs avec filtres et drill-down	Très interactif et esthétique	Axé monitoring et alertes	Statique (peut être enrichi avec Dash)
Comptabilité	CSV, SQL, Excel, API	Connecte à de nombreuses sources	Bases de données, JSON	Intégré à Python, supporte CSV, JSON, SQL
Coût	Gratuit (avec version Pro payante)	Payant (cher pour entreprises)	Gratuit	Gratuit

II. Explications des visualisations

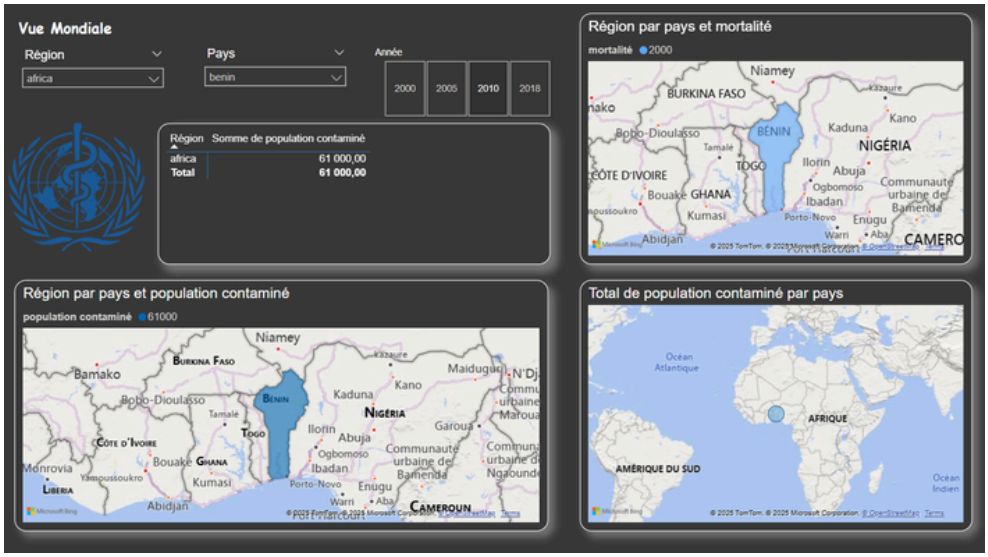
Les visualisations créées dans Power BI permettent de mettre en évidence les tendances et indicateurs clés liés au VIH et à la mortalité, en offrant une lecture claire et interactive des données. L'objectif est d'avoir une vue d'ensemble rapide et interactive, permettant d'analyser les tendances, les zones critiques et l'efficacité des traitements. Grâce à Power BI, ces visualisations sont dynamique et permettent d'approfondir l'analyse en filtrant les données par pays, année ou région.

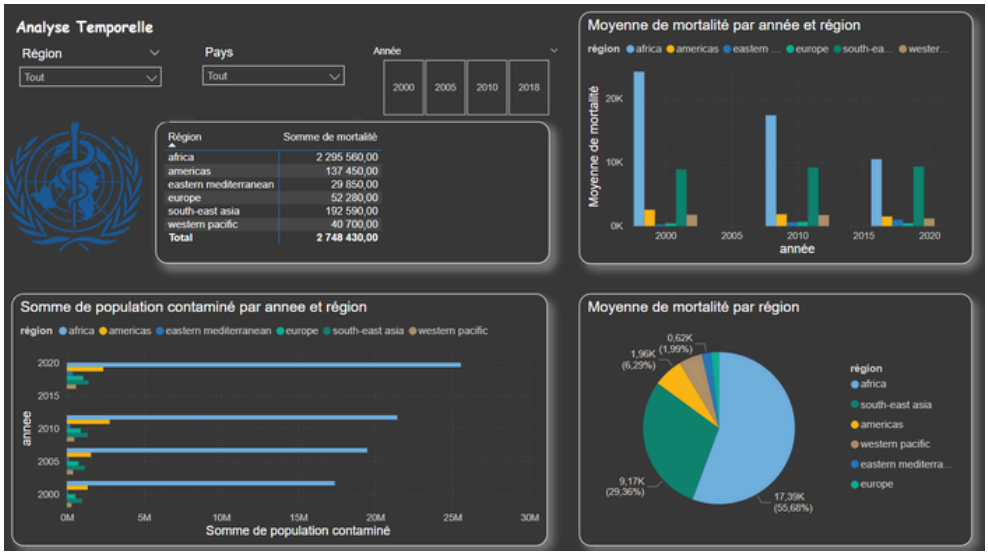
Conclusion

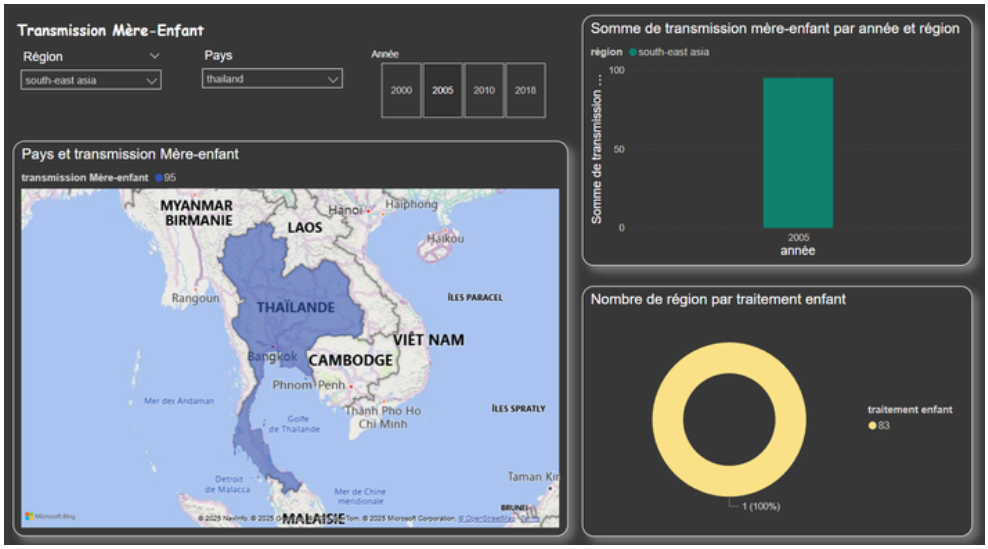
En résumé, ce projet permet d'ouvrir la voie vers une solution basée sur l'intelligence artificielle.
Les différentes étapes suivies, allant de la modélisation de la base de données à la création de visualisations interactives avec Power BI, ont démontré l'importance d'une architecture de données bien pensée, d'un processus ETL rigoureux et d'une API performante. Grâce à cette démarche méthodologique, nous avons pu extraire des informations pertinentes et stratégiques, facilitant ainsi la compréhension et l'analyse de l'épidémie du VIH.
Il sera bon d'envisager l'utilisation de set de données complémentaires, afin de renforcer les résultats de notre travail et d'offrir une meilleure vue d'ensemble du Virus, en ajoutant les données récoltées au cours du temps par exemple. Afin d'enrichir les perspectives potentielles qu'offre la visualisation de ces données.
Les résultats obtenus montrent que l'exploitation des données à travers des outils modernes et adaptés peut non seulement améliorer la qualité des analyses, mais également apporter des insights précieux pour la lutte contre des épidémies telles que le VIH.
En somme, ce projet met en lumière le potentiel des solutions basées sur l'intelligence artificielle pour répondre aux défis
sanitaires actuels et futurs.

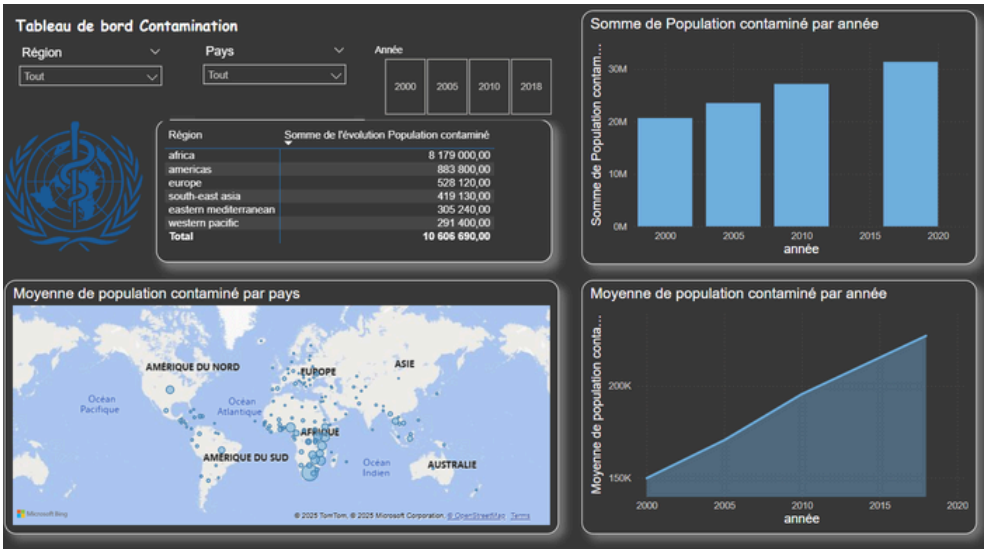
Annexe

Page Power BI









Extraction et validation des fichiers sources CSV

```python
# ---------------------- EXTRACTION (Extract) ----------------------
Tabnine | Edit | Test | Explain | Document
def extract_data():
    """
    EXTRACTION : Lecture des fichiers sources
    Returns:
        list: Liste des DataFrames extraits ou None en cas d'erreur
    """
    try:
        logging.info("🔄 Début de l'extraction...")

        files = [
            "../SourceData/art_coverage_by_country_clean.csv",
            "../SourceData/art_pediatric_coverage_by_country_clean.csv",
            "../SourceData/no_of_cases_adults_15_to_49_by_country_clean.csv",
            "../SourceData/no_of_deaths_by_country_clean.csv",
            "../SourceData/no_of_people_living_with_hiv_by_country_clean.csv",
            "../SourceData/prevention_of_mother_to_child_transmission_by_country_clean.csv"
        ]
```

```python
dataframes = []
for file_path in files:
    path = Path(file_path)
    if not path.exists():
        raise FileNotFoundError(f"Fichier introuvable : {file_path}")

    df = pd.read_csv(file_path)
    if df.empty:
        logging.warning(f"⚠️ Le fichier {file_path} est vide")
        continue

    # Vérification des colonnes nécessaires
    country_col = next((col for col in df.columns if "Country" in col), None)
    region_col = next((col for col in df.columns if "WHO Region" in col), None)
```

```python
    if country_col:
        df.rename(columns={country_col: "pays"}, inplace=True)
        if region_col:
            df.rename(columns={region_col: "region_who"}, inplace=True)
            df = df[["pays", "region_who"]].drop_duplicates()
        else:
            df = df[["pays"]].drop_duplicates()
        dataframes.append(df)
    else:
        logging.warning(f"⚠️ Aucune colonne 'Country' trouvée dans {file_path}")
        continue

if not dataframes:
    raise ValueError("Aucune donnée valide extraite des fichiers sources")
```

```python
        logging.info("✅ Extraction réussie")
        return dataframes
    .......

    except FileNotFoundError as e:
        logging.error(f"❌ Erreur d'accès fichier : {str(e)}")
        return None
    except ValueError as e:
        logging.error(f"❌ Erreur de données : {str(e)}")
        return None
    except Exception as e:
        logging.error(f"❌ Erreur inattendue lors de l'extraction : {str(e)}")
        return None
```

Importation et structuration des données dans PostgreSQL

```sql
SET session_replication_role = 'replica';

-- Import des données de référence
\COPY pays(id_pays, nom_pays, region) FROM 'DatasetClean/pays_clean.csv' WITH CSV HEADER;
\COPY unite(id_unite, nom_unite) FROM 'DatasetClean/unite_clean.csv' WITH CSV HEADER;
\COPY type_statistique(id_type_statistique, nom_type_statistique) FROM 'DatasetClean/type_statistique_clean.csv' WITH CSV HEADER;
\COPY type_traitement(id_type_traitement, nom_type_traitement) FROM 'DatasetClean/type_traitement_clean.csv' WITH CSV HEADER;

-- Modification des séquences pour les tables de référence
SELECT setval('pays_id_pays_seq', (SELECT MAX(id_pays) FROM pays));
SELECT setval('unite_id_unite_seq', (SELECT MAX(id_unite) FROM unite));
SELECT setval('type_statistique_id_type_statistique_seq', (SELECT MAX(id_type_statistique) FROM type_statistique));
SELECT setval('type_traitement_id_type_traitement_seq', (SELECT MAX(id_type_traitement) FROM type_traitement));

-- Import des données principales en utilisant la valeur médiane
CREATE TEMP TABLE temp_population_hiv (
    id_pays INTEGER,
    annee INTEGER,
    population_min DECIMAL,
    population_median DECIMAL,
    population_max DECIMAL,
    id_unite INTEGER
);
```

```
CREATE TABLE type_traitement_clean (
    id_type_traitement INT PRIMARY KEY AUTO_INCREMENT,
    libelle VARCHAR(100)
);

CREATE TABLE table_population_hiv (
    id_pays INT,
    annee INT,
    population_min INT,
    population_median INT,
    population_max INT,
    id_unite INT,
    PRIMARY KEY (id_pays, annee),
    FOREIGN KEY (id_pays) REFERENCES pays_clean(id_pays),
    FOREIGN KEY (id_unite) REFERENCES unite_clean(id_unite),
    INDEX idx_population_hiv (annee, id_pays)
);

CREATE TABLE table_mortalite (
    id_pays INT,
    annee INT,
    mortalite_min INT,
    mortalite_median INT,
    mortalite_max INT,
    id_unite INT,
    PRIMARY KEY (id_pays, annee),
    FOREIGN KEY (id_pays) REFERENCES pays_clean(id_pays),
    FOREIGN KEY (id_unite) REFERENCES unite_clean(id_unite),
    INDEX idx_mortalite (annee, id_pays)
);

CREATE TABLE table_transmission_mere_enfant (
    id_transmission INT PRIMARY KEY AUTO_INCREMENT,
    id_pays INT,
    besoin_arv_min INT,
    besoin_arv_median INT,
    besoin_arv_max INT,
    pourcentage_recu_min FLOAT,
    pourcentage_recu_median FLOAT,
    pourcentage_recu_max FLOAT,
    FOREIGN KEY (id_pays) REFERENCES pays_clean(id_pays),
    INDEX idx_transmission_mere_enfant (id_pays)
);
```

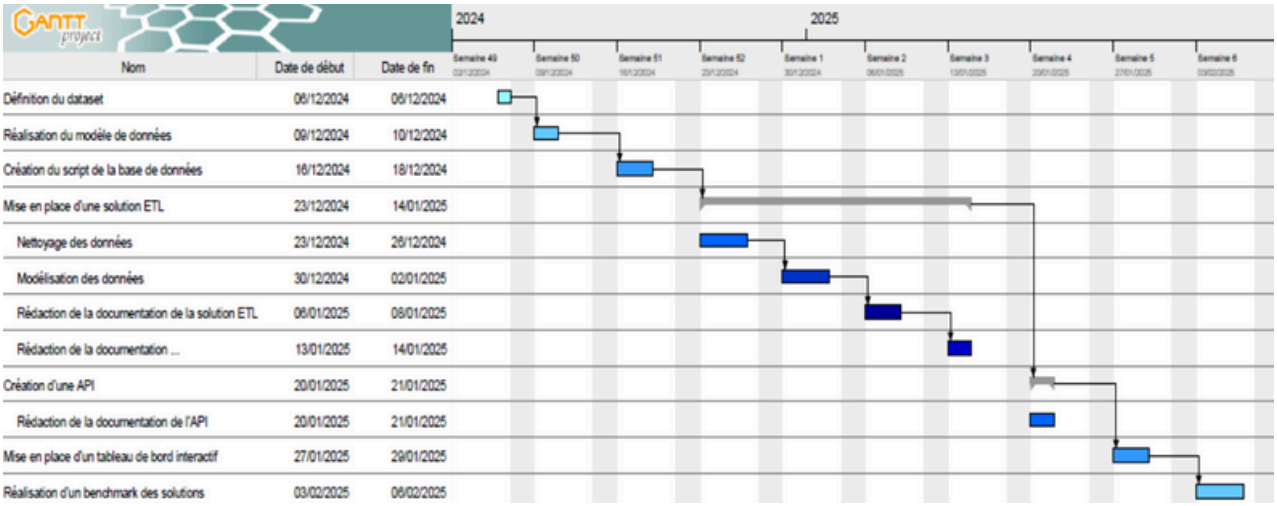
```
CREATE TABLE table_traitement (
    id_traitement INT PRIMARY KEY AUTO_INCREMENT,
    id_pays INT,
    annee INT,
    id_type_traitement INT,
    couverture FLOAT,
    FOREIGN KEY (id_pays) REFERENCES pays_clean(id_pays),
    FOREIGN KEY (id_type_traitement) REFERENCES
type_traitement_clean(id_type_traitement),
    INDEX idx_traitement (annee, id_pays)
);

CREATE TABLE table_statistique (
    id_statistique INT PRIMARY KEY AUTO_INCREMENT,
    id_pays INT,
    annee INT,
    id_type_statistique INT,
    valeur FLOAT,
    id_unite INT,
    FOREIGN KEY (id_pays) REFERENCES pays_clean(id_pays),
    FOREIGN KEY (id_type_statistique) REFERENCES
type_statistique_clean(id_type_statistique),
    FOREIGN KEY (id_unite) REFERENCES unite_clean(id_unite),
    INDEX idx_statistique (annee, id_pays)
);
```

```python
Tabnine | Edit | Test | Explain | Document
@app.get("/pays/", response_model=List[schemas.Pays])
async def get_pays(db: AsyncSession = Depends(get_db)):
    result = await db.execute(select(models.Pays))
    return result.scalars().all()


Tabnine | Edit | Test | Explain | Document
@app.post("/pays/", response_model=schemas.Pays)
async def create_pays(pays: schemas.PaysCreate, db: AsyncSession = Depends(get_db)):
    new_pays = models.Pays(**pays.dict())
    db.add(new_pays)
    await db.commit()
    await db.refresh(new_pays)
    return new_pays
```

```python
Tabnine | Edit | Test | Explain | Document
@app.put("/pays/{pays_id}/", response_model=schemas.Pays)
async def update_pays(
    pays_id: int, pays: schemas.PaysCreate, db: AsyncSession = Depends(get_db)
):
    result = await db.execute(select(models.Pays).where(models.Pays.id_pays == pays_id))
    db_pays = result.scalar_one_or_none()

    if not db_pays:
        raise HTTPException(status_code=404, detail="Pays non trouvé")

    await db.execute(
        update(models.Pays).where(models.Pays.id_pays == pays_id).values(**pays.dict())
    )
    await db.commit()
    return {**pays.dict(), "id_pays": pays_id}
```

```python
Tabnine | Edit | Test | Explain | Document
@app.delete("/pays/{pays_id}/")
async def delete_pays(pays_id: int, db: AsyncSession = Depends(get_db)):
    result = await db.execute(select(models.Pays).where(models.Pays.id_pays == pays_id))
    db_pays = result.scalar_one_or_none()

    if not db_pays:
        raise HTTPException(status_code=404, detail="Pays non trouvé")

    await db.execute(delete(models.Pays).where(models.Pays.id_pays == pays_id))
    await db.commit()
    return {"message": "Pays supprimé avec succès"}
```

Diagram de Gantt



Nom	Date de début	Date de fin
Définition du dataset	06/12/2024	06/12/2024
Réalisation du modèle de données	09/12/2024	10/12/2024
Création du script de la base de données	16/12/2024	18/12/2024
Mise en place d'une solution ETL	23/12/2024	14/01/2025
Nettoyage des données	23/12/2024	26/12/2024
Modélisation des données	30/12/2024	02/01/2025
Rédaction de la documentation de la solution ETL	06/01/2025	08/01/2025
Rédaction de la documentation ...	13/01/2025	14/01/2025
Création d'une API	20/01/2025	21/01/2025
Rédaction de la documentation de l'API	20/01/2025	21/01/2025
Mise en place d'un tableau de bord interactif	27/01/2025	29/01/2025
Réalisation d'un benchmark des solutions	03/02/2025	06/02/2025