Health Data Project — Rapport Final

DATA ENGINEERING ON CLOUD

Sekari Inès - Nkuida Malaïka

Table des matières

1. Présentation du sujet

Contexte

Au cours des dernières années, la transformation numérique du secteur de la santé a engendré un volume croissant de données hétérogènes, issues aussi bien des systèmes hospitaliers que des dispositifs connectés ou des dossiers médicaux partagés (DMP). Cette explosion des données, associée à la diversification des pratiques médicales (hospitalisation à domicile, télémédecine, parcours ambulatoires), renforce le besoin d'analyses transversales capables de relier les informations cliniques, administratives et sociales

Les établissements de santé font aujourd'hui face à de multiples défis : garantir la qualité et la sécurité des soins, adapter leur organisation à l'évolution des besoins des patients, et répondre aux exigences de performance et de transparence. Dans ce contexte, la capacité à centraliser, croiser et historiser ces flux d'information devient un enjeu stratégique pour améliorer la qualité des soins, optimiser la prise en charge des patients et éclairer les politiques publiques.

Objectif du projet

Pour la société dans son ensemble, la valorisation des données multi-sources permet non seulement d'améliorer l'efficience du système de santé, mais aussi de produire des connaissances utiles à la recherche, de renforcer la prévention, et d'accompagner la prise de décision publique. L'intégration et l'analyse de jeux de données variés – jusqu'ici cloisonnés et difficilement exploitables – ouvrent ainsi la voie à une approche plus globale et personnalisée de la santé.

L'architecture Lakehouse s'impose ainsi comme une réponse innovante : elle permet de gérer à la fois la variété et le volume des données, d'en garantir la gouvernance, tout en facilitant leur valorisation par des analyses avancées et des visualisations pertinentes.

L'objectif principal de ce projet est de fournir une vision globale et approfondie des parcours de soins en santé, en s'appuyant sur la consolidation et l'analyse de données issues de multiples sources hospitalières et médicales. Ce travail vise à mieux comprendre l'évolution des prises en charge, à identifier les différentes typologies de patients ou de pathologies, et à évaluer leur impact tant sur le plan social que méthodologique.

Plus concrètement, cet objectif se décline en plusieurs axes :

- **Croiser et harmoniser** les jeux de données relatifs aux admissions, diagnostics, traitements et catégories de motifs afin de reconstituer des parcours individuels de soins.
- **Analyser l'évolution** des diagnostics et des modalités de prise en charge pour détecter les tendances émergentes et les disparités éventuelles selon les profils de patients ou les périodes.
- **Mettre en lumière les déterminants sociaux, médicaux ou organisationnels** qui influencent la santé et l'accès au soin.
- **Développer des indicateurs de suivi et de performance**, afin de guider l'amélioration continue des pratiques hospitalières et d'informer la décision publique.
- **Explorer la capacité d'un modèle Lakehouse** à structurer, historiser et valoriser ce type d'information à grande échelle.

Toutes ces ambitions reposent sur une exploitation rigoureuse, structurée et documentée des données multi-sources collectées dans le cadre du projet.

Les sources de données

Les analyses réalisées dans le cadre de ce projet s'appuient sur quatre jeux de données principaux, chacun apportant une brique complémentaire à la compréhension du parcours patient et à la caractérisation des prises en charge :

hospital-triage.csv

Ce fichier contient des données anonymisées sur les visites aux urgences, symptômes, diagnostics. Les données issues de ce fichier sont très riches et couvrent plusieurs dimensions du séjour patient, structurées autour de quatre grands ensembles :

Profil et contexte d'admission du patient

- **patient_id** : identifiants uniques du patient et de son dossier.
- **dep_name** : nom de l'hôpital d'accueil.
- **age, gender, ethnicity, race** : caractéristiques sociodémographiques.
- **lang (langue), religion, maritalstatus (situation matrimoniale), employstatus (emploi), insurance_status (statut d'assurance)** : variables de contexte social.
- **disposition** : destination à la sortie (ex : retour domicile, transfert).
- **arrivalmode** : mode d'arrivée aux urgences : ambulance, voiture, etc.
- **arrivalmonth, arrivalday, arrivalhour_bin** : date et heure (plages horaires) d'arrivée.
- **previousdispo** : disposition lors d'une visite précédente.
- **n_surgeries, n_edvisits, n_admissions** : nombre de chirurgies, de passages aux urgences et d'hospitalisations antérieures.

Présence de maladies et antécédents médicaux

- Plus de cent colonnes binaires pour chaque pathologie/antécédent possible (ex : asthma, heartfailure, diabetes, fracture, etc.).
- Valeur généralement : 1=présence, 0=absence de la maladie ou historique associé pour ce patient.

Motifs d'admission ou de recours aux soins

- Plus de 100 colonnes de type : cc_xxx où chaque variable représente un motif précis d'admission ou de plainte principale (par exemple : cc_chestpain, cc_fever, cc_backpain).
- Indique par patient les raisons exactes du recours à l'établissement, renseignant à la fois les urgences cliniques et plaintes fréquentes.

Classes de traitements et médicaments administrés

- Une quarantaine de colonnes, de type : meds_<classe> (ex : meds_antibiotics, meds_analgesics, meds_antivirals…).
- Permet de savoir pour chaque patient s'il a reçu une classe de médicament/traitement durant son passage.

Remarque :
Les mesures biologiques et vitales, examens complémentaires et scores de triage forment aussi un sous-ensemble exploitable, non détaillé ici pour la concision.

meds_cat.csv

Dictionnaire permettant de lier chaque type de médicament ingéré à sa classe thérapeutique.

- **medicament_code** : identifiant du médicament.
- **medicament_label** : nom usuel (ex : "Amoxicilline").
- **classe_therapeutique** : grande famille médicale (ex : "antibiotique", "antidiabétique").

icd_code.csv

Ce référentiel (dictionnaire) fait le lien entre un diagnostic, sa codification et ses propriétés médicales.

- **diagnosis :** libellé du diagnostic (ex : "Pneumopathie")
- **Code CIM-10 :** code international (ex : "J18.9")
- **Catégorie médicale :** regroupement médical principal (ex : "Infections respiratoires")
- **description :** description longue ou précisions médicales.

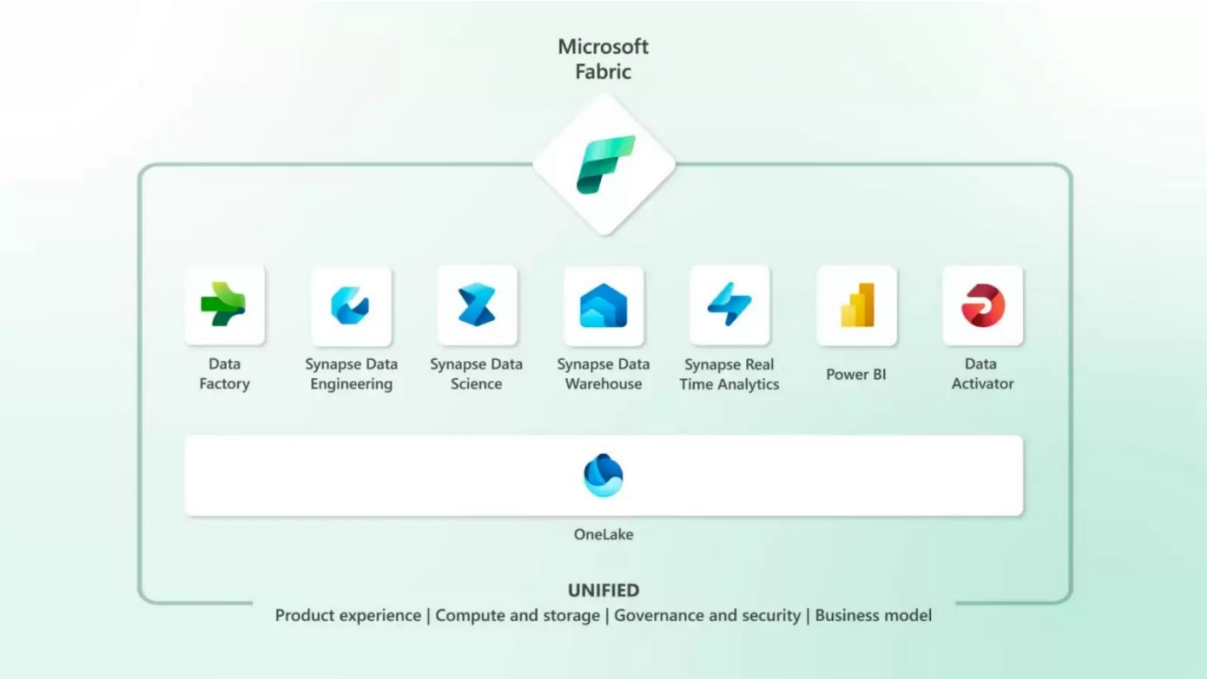
motifs_categorie.csv

Outil de regroupement qui classe chaque motif d'admission selon un grand axe clinique.

- **Motif** : libellé du motif précis (correspond à une des colonnes "cc_xxx" de hospital-triage).
- **Catégorie_motif** : regroupement large ("Urgence", "Suivi chronique", "Douleur", etc.).
- **Description_longue** : explication étendue du motif (contextualisation médicale).

Chaque source présente ses spécificités en termes de format, de granularité et de structuration, ce qui justifie une étape de préparation et d'intégration minutieuse en amont de l'analyse.

2. Outils et technologies utilisés



Pour répondre aux enjeux d'intégration, d'analyse et de valorisation des données de santé hétérogènes, notre projet s'appuie sur un écosystème moderne, performant et évolutif.
Voici les principaux outils et plateformes exploités :

Microsoft Fabric (Lakehouse)

Microsoft Fabric constitue le socle central de notre architecture data. Il s'agit d'une plateforme de données et d'analytique complète, dédiée à l'unification du stockage, du traitement et de la valorisation des données.

Parmi ses atouts majeurs :

- **Solution Tout-en-Un** : Fabric centralise l'ensemble des opérations (ingestion, traitement, stockage, restitution) en une seule plateforme, réduisant la complexité et évitant l'intégration manuelle de briques multiples.
- **Interface conviviale** : la prise en main est facilitée pour les data engineers/scientists comme pour les métiers, optimisant la gestion quotidienne de la donnée.
- **Modèle SaaS** : la plateforme est entièrement gérée par Microsoft. Les mises à jour, la scalabilité et la maintenance sont prises en charge, ce qui nous permet de nous concentrer sur la valeur métier.
- **Sécurité et conformité** : Fabric garantit un haut niveau de sécurité, de traçabilité et de conformité réglementaire, essentiels pour les données sensibles du secteur santé.

PySpark

Le traitement, le nettoyage et l'exploitation avancée des volumes importants de données sont réalisés via **PySpark**, la version Python de l'API Spark.

- Permet de manipuler efficacement de grands jeux de données (distribution, parallélisation).

- Adapté aux opérations complexes de croisement et d'enrichissement.

Power BI
Pour la restitution et la visualisation des analyses, **Power BI** s'impose comme l'outil privilégié :

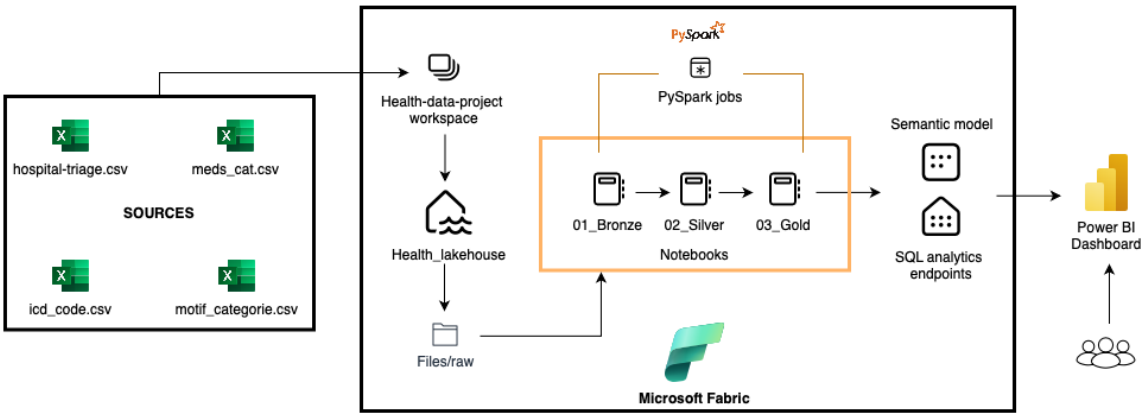
- Puissant, interactif, connecté en natif à Fabric.
- Permet la création de tableaux de bord dynamiques, favorisant l'exploration et la prise de décision pour les utilisateurs métiers.
- Assure la sécurité au niveau de la restitution des données sensibles.

GitHub
La gestion du code, des scripts de transformation et de l'ensemble des artefacts du projet s'effectue via **Git/GitHub** :

- Facilite la collaboration entre les membres de l'équipe.
- Permet l'historisation, la gestion des versions, et une traçabilité complète des évolutions techniques.
- Sécurise les développements et facilite le passage à l'échelle ou la reproductibilité du projet.

Ce socle technologique garantit flexibilité, robustesse, sécurité et efficacité pour traiter et valoriser les données de santé, du brut à l'insight décisionnel.

3. Architecture



Détails de l'architecture :

- **Stockage initial (Zone Lakehouse – Raw Data) :**
  Tous les fichiers de données sources (hospital-triage.csv, meds_cat.csv, icd_code.csv, motif_categorie.csv) sont déposés dans un espace de stockage du Lakehouse spécialement dédié.
  Cette zone permet de centraliser et d'archiver toutes les données brutes, sans transformation.

Objectif : disposer d'un référentiel de données original, assurant l'auditabilité et la traçabilité.

- **Bronze Layer (données brutes intégrées) :**
  À l'aide de notebooks PySpark/Spark, chaque fichier source est importé **tel quel** dans la couche dite *Bronze* du Lakehouse avec un léger pré-nettoyage.
  Les jeux de données sont chargés dans des tables distinctes dont la source principale éclatée en 4 tables :

    - Table patient provenant d'hospital-triage (caractéristiques patients)

- Table maladie provenant d'hospital-triage (pathologies antérieures et diagnostic)
- Table motif_admission provenant d'hospital-triage (motifs de prise en charge)
- Table médicament provenant d'hospital-triage (traitements consommés lors de la prise en charge)
- référentiels : icd_code, meds_code, motifs_code
  Aucune transformation ni nettoyage significatif n'est appliqué à cette étape : le focus est sur la conservation de l'historique et la fidélité des données d'origine.

- **Silver Layer – Zone de Données Nettoyées et Transformées**
  Les données bronzes sont nettoyées, standardisées, dédoublonnées et enrichies.

  - Correction des valeurs invalides ou manquantes
  - Standardisation des variables métier (âge, genre…)
  - Extraction de features analytiques (diagnostics, médicament, motifs d'admission, timestamps…)
  - Préparation des données pour la modélisation en étoile (tables de faits et dimensions).

- **Gold Layer - Zone Analytique**
  Résultat final, structuré pour l'exploitation métier ; création d'un modèle dimensionnel pour l'analyse (tables de faits/dimensions adaptées). Dans cette couche, les données sont nettoyées, normalisées, auditées ; elles sont croisées pour créer une **modélisation analytique** adaptée à la visualisation et à l'analyse.

  - Construction d'un **modèle en étoile** :
    - Table de faits : Consultation patient, centralisant les mesures d'intérêt
    - Tables de dimensions :
      - Patients (profil)
      - Hôpital (3 choix)
      - Maladies (enrichie avec icd_code)
      - Médicaments (enrichie avec meds_code)
      - Motifs d'admission (enrichie avec motifs_code)
      - Temps (timestamp/jour/horaire/année)
  - Modélisations facilitant l'exploration rapide et multi-angle des données (par patient, par motif, par médicament, par période…).
  - Agrégations, calculs statistiques, génération d'indicateurs clés (KPI).
  - Cette structuration optimise la vitesse et la cohérence des analyses transversales et temporelles.

- **Modèle sémantique :**
  Un **modèle sémantique** est construit dans Fabric pour synthétiser les indicateurs clés et faciliter la consommation des données par les utilisateurs finaux.
  Il centralise les mesures (par exemple : taux de passage pour motif respiratoire, score de gravité moyen), les dimensions d'analyse (âge, période, motif...), et les calculs métiers, assurant ainsi l'homogénéité des analyses.

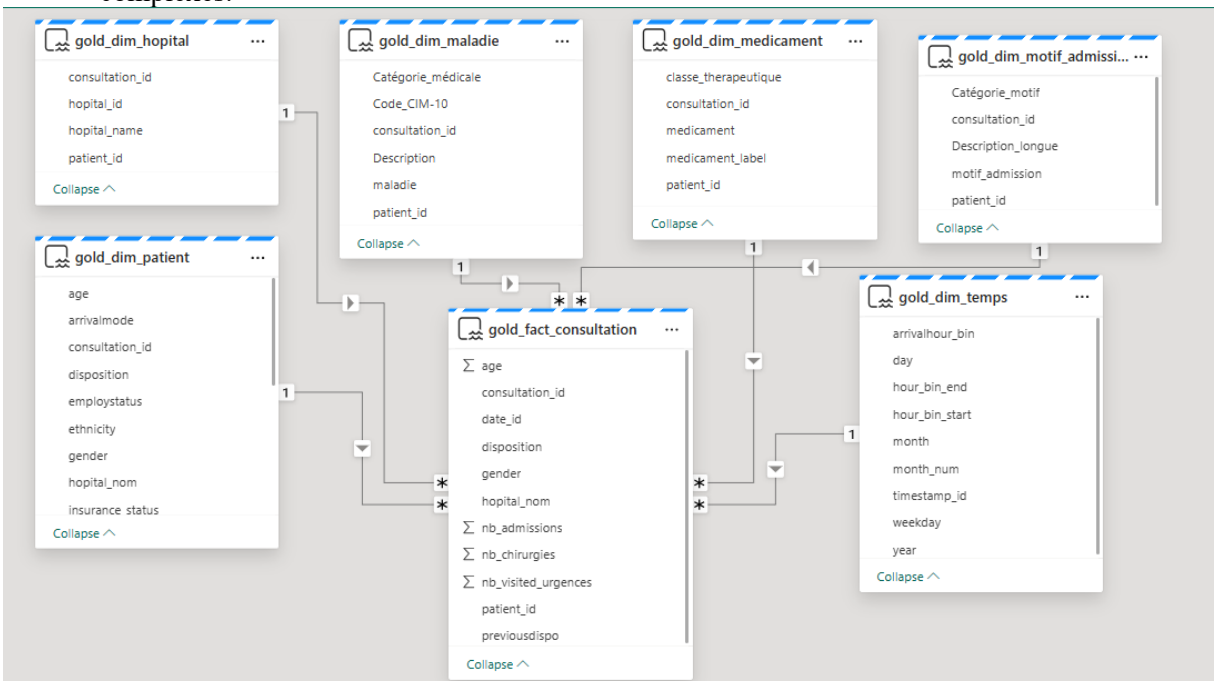
- **Restitution & DataViz – Power BI :**
  Le modèle sémantique est exposé à Power BI pour permettre une analyse visuelle interactive. Les utilisateurs finaux peuvent explorer dynamiquement les parcours patients, les tendances de pathologies/motifs ou encore l'usage des classes thérapeutiques.

Cette architecture modulaire et robuste est conçue pour évoluer facilement : elle garantit rapidité d'analyse, intégrité des données sources, traçabilité totale, et simplicité de restitution métier.

4. Modélisation dimensionnelle/MCD

Schéma en étoile
Pourquoi un schéma en étoile ?
Nous avons opté pour un modèle en étoile car :

- Il est facile à comprendre et facilite l'analyse.
- La structure centrée sur une table de faits entourée de tables de dimensions permet d'accéder rapidement aux attributs utiles sans jointures complexes.
- C'est le modèle de référence pour la BI sur des données transactionnelles/sanitaires complexes.



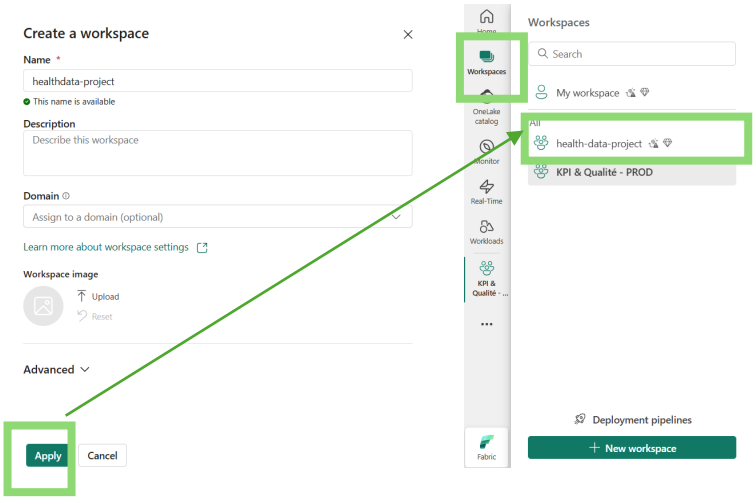
Dans ce schéma :
Fact_consultation est la table de faits centrale, représentant les consultations patients et centralisant les mesures d'intérêt.

Les tables de dimensions entourent la table de faits, fournissant des contextes supplémentaires pour l'analyse.

5. Implémentation technique

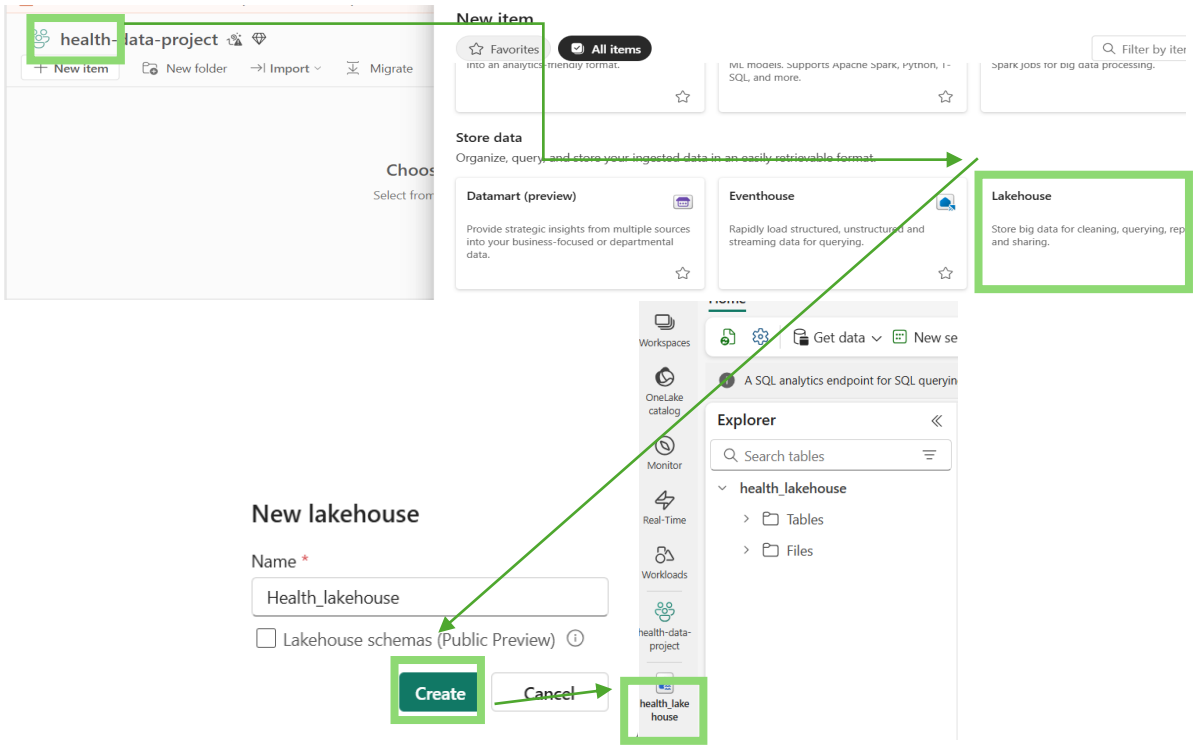
Traitements et transformations
Lien Github : https://github.com/inessekari/data-cloud-project
Dans notre démarche, nous utilisons deux services principaux de Microsoft Fabric : l'un pour la partie ETL (traitement et transformation des données), l'autre pour la visualisation et la restitution analytique. Tous les scripts de transformation, de nettoyage et de modélisation sont versionnés sur GitHub via des notebooks pour une meilleure traçabilité et collaboration.

ETL – Création du workspace
La phase ETL (Extract, Transform, Load) est réalisée à travers les notebooks fournis par Microsoft Fabric, ce qui nous permet d'automatiser chaque étape du processus, de l'import brut jusqu'au modèle prêt à l'analyse.
Nous avons tout d'abord créé un **Workspace** dédié afin de centraliser le projet, d'organiser le travail en binôme et de structurer proprement l'ensemble des ressources.



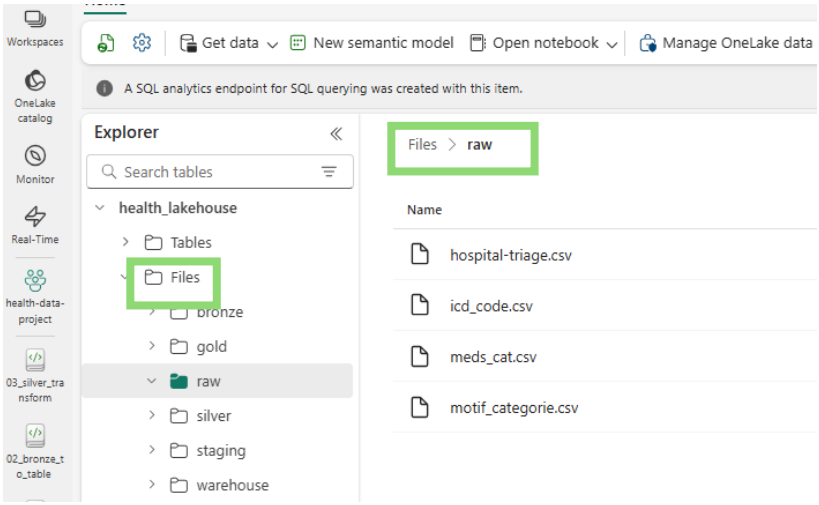
ETL – Création du lakehouse

Dans ce workspace, nous avons mis en place un **Lakehouse** qui nous sert de socle unique pour stocker, transformer et requêter l'ensemble des données du projet.

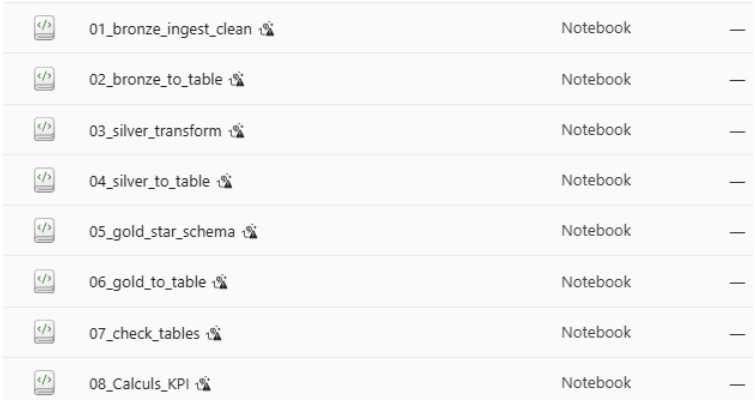


ETL – Chargement des données

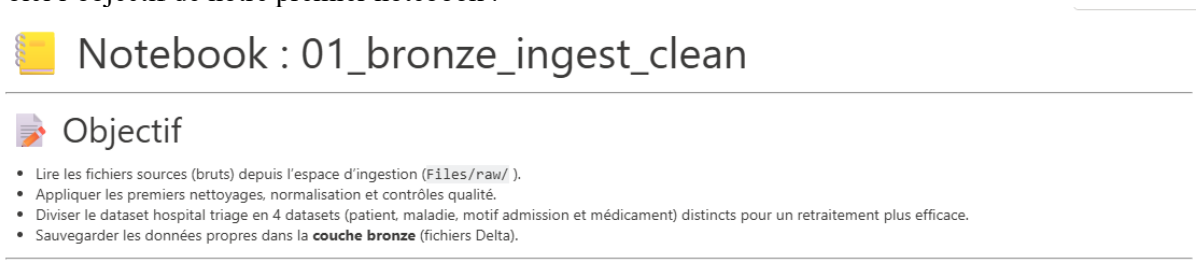
Pour organiser les sources de données, nous avons créé dans l'espace "Files" un sous-dossier « raw » dans lequel nous déposons les différents fichiers CSV nous servant de sources. Cela garantit une gestion claire des sources ainsi qu'une reprise aisée en cas de mise à jour des fichiers.



Nous avons utilisé des notebooks pour ingérer chaque source brute en tables **Bronze** (raw data). Ces notebooks documentent les opérations de typage, de renommage et de contrôle de qualité de la donnée lors de l'import, offrant visibilité et facilité de maintenance. Chaque notebook est renommé pour plus de précision et de clarté.



Comme précisé dans les détails de notre Architecture, nous avons quatre sources de données et dont une source principale et des référentiels. Notre premier objectif est de scinder la source principale en quatre sources : patient, maladie, médicament et motif d'admission.
Voici l'objectif de notre premier notebook :



Nous commençons par lire le dataset hospital-triage afin de déterminer quelles colonnes sont nécessaires à la création de nos 4 différents dataframes : On voit 972 colonnes, et nous savons qu'il y a énormément de type de maladie, diagnostic, médicament et motif d'admission tout simplement car la répartition est binaire

```python
# Lecture des données sources
df_hospital = spark.read.csv(
    'Files/raw/hospital-triage.csv',
    header=True,
    inferSchema=True,
    sep=',',
    quote='"',
    escape='"'
)

display(df_hospital.head(5))
```

16 sec - Command executed in 16 sec 41 ms by Inès SEKARI on 10:22:55 PM, 5/21/25 — PySpark (Python)

Spark jobs (6 of 6 succeeded) Resources Log

Table | New chart — 972 columns, 5 rows

Table view — Download — Search — Inspect

	dep_name	esi	age	gender	ethnicity	race	lang	religion	maritalstatus	employstatus	
1	B	4	40	Male	Hispanic or L...	White or C...	English	None	Single	Full Time	Oth
2	B	4	66	Male	Hispanic or L...	Native Haw...	English	Pentecostal	Married	Not Employed	Cor
3	B	2	66	Male	Hispanic or L...	Native Haw...	English	Pentecostal	Married	Not Employed	Cor
4	A	2	66	Male	Hispanic or L...	Native Haw...	English	Pentecostal	Married	Not Employed	Cor
5	A	3	84	Female	Hispanic or L...	Other	Other	Pentecostal	Widowed	Retired	Me

On ajoute une colonne profile_id qui concatène un grand nombre d'information sur le client, mais nous nous sommes rendu compte que cet id n'était pas unique. C'est pourquoi nous avons créé la colonne patient_id afin d'avoir un id unique par patient. Cet id nous servira à faire des jointures plus tard si besoin.

```python
from pyspark.sql import functions as F
from pyspark.sql.functions import monotonically_increasing_id

# Ajoute les ID en concaténant les infos profil
df_hospital = (
    df_hospital
        .withColumn(
            "profile_id",
            F.concat_ws("_", F.col("age"), F.col("gender"), F.col("ethnicity"), F.col("lang"), F.col("maritalstatus"))
        )
)


df_hospital = df_hospital.withColumn("patient_id", monotonically_increasing_id())

display(df_hospital.head(5))
```

2 sec - Command executed in 2 sec 271 ms by Inès SEKARI on 10:23:00 PM, 5/21/25 — PySpark (Python)

Spark jobs (4 of 4 succeeded) Resources Log

Table | New chart — 974 columns, 5 rows

Table view — Download — Search — Inspect

_withdrawal-alcohol	cc_woundcheck	cc_woundinfection	cc_woundre-evaluation	cc_wristinjury	cc_wristpain	profile_id	patient_id
	0	0	0	0	0	40_Male_Hispa...	0
	0	0	0	0	0	66_Male_Hispa...	1
	0	0	0	0	0	66_Male_Hispa...	2
	0	0	0	0	0	66_Male_Hispa...	3
	0	0	0	0	0	84_Female_His...	4

Nous identifions chaque spécificité des colonnes afin de déterminer leur catégorie. Par exemple, pour les médicaments, on remarque que les colonnes concernées commencent toujours par 'meds_'…, nous créons donc une fonction afin de catégoriser les colonnes.

```python
import re

def categorize_column(colname):
    col = colname.lower()

    # 1. Médicaments
    if col.startswith('meds_'):
        return 'traitement/médicament'

    # 2. Examens biologiques
    bio_patterns = [
        r'_last$', r'_min$', r'_max$', r'_median$'
    ]
    if any(re.search(pat, col) for pat in bio_patterns):
        return 'examen biologique/mesure'

    # 3. Examens urinaires/cultures
    if ('ua_' in col) or ('culture' in col):
        return 'examen urinaire/culture'

    # 4. Signe vital
    vital_keywords = [
        'triage_vital', 'pulse', 'resp', 'spo2', 'temp', 'sbp', 'dbp', 'o2_device'
    ]
    if any(k in col for k in vital_keywords):
        return 'signe vital'

    # 5. Examens d'imagerie ou actes
    img_keywords = [
        'cxr', 'ekg', 'echo', 'ct', 'xr', 'mri', 'us', 'img'
    ]
    if any(col.endswith('_count') and k in col for k in img_keywords):
        return "examen imagerie/acte médical"

    # 6. Mesures de parcours patient
    parcours_keywords = [
        'n_edvisits', 'n_admissions', 'n_surgeries'
    ]
    if col in parcours_keywords:
        return 'mesure parcours patient'

    # 7. Examens d'urines/cultures divers (nombre de positifs/total)
    if any(key in col for key in ['_npos', '_count']):
        # Si déjà classé par ailleurs, on n'y arrive pas ici
        if ('ua_' in col) or ('culture' in col):
            return 'examen urinaire/culture'
        else:
            return 'autre comptage'

    # Par défaut
    return 'autre'

# Ex d'utilisation :
cols = [
    'n_edvisits', 'triage_vital_hr', 'pulse_last', 'hemoglobin_max', 'cxr_count',
    'meds_antibiotics', 'bloodua_npos', 'urineculture,routine_last', 'sbp_median', 'poctroponini._median'
]

for col in cols:
    print(f"{col:35s} => {categorize_column(col)}")
```

<1 sec - Command executed in 237 ms by Inès SEKARI on 10:23:02 PM, 5/21/25

Ce qui nous donne :

```
n_edvisits                          => mesure parcours patient
triage_vital_hr                     => signe vital
pulse_last                          => examen biologique/mesure
hemoglobin_max                      => examen biologique/mesure
cxr_count                           => examen imagerie/acte médical
meds_antibiotics                    => traitement/médicament
bloodua_npos                        => examen urinaire/culture
urineculture,routine_last           => examen biologique/mesure
sbp_median                          => examen biologique/mesure
poctroponini._median                => examen biologique/mesure
```

Ensuite, nous créons nos 4 dataframes différents, comme expliqué précédemment.

```python
df_patient = df_hospital.select('patient_id', 'profile_id','dep_name', 'age', 'gender', 'ethnicity', 'race', 'lang', 'religion', 'marita
df_maladie = df_hospital.select('patient_id','2ndarymalig', 'abdomhernia', 'abdomnlpain', 'abortcompl', 'acqfootdef', 'acrenlfail', 'acu
df_motif_admission = df_hospital.select('patient_id','cc_abdominalcramping', 'cc_abdominaldistention', 'cc_abdominalpain', 'cc_abdominal
df_medicament = df_hospital.select('patient_id','meds_analgesicandantihistaminecombination', 'meds_analgesics', 'meds_anesthetics', 'med

```

<1 sec - Command executed in 769 ms by Inès SEKARI on 10:23:07 PM, 5/21/25 PySpark (Python)

On applique quelques nettoyages :

```python
from pyspark.sql.functions import col, upper, sum as spark_sum

# 1. Suppression des doublons
df_patient = df_patient.dropDuplicates()
df_maladie = df_maladie.dropDuplicates()
df_motif_admission = df_motif_admission.dropDuplicates()
df_medicament = df_medicament.dropDuplicates()

# 2. Suppression des lignes trop vides
thresh = int(0.2 * len(df_patient.columns))
thresh = int(0.2 * len(df_maladie.columns))
thresh = int(0.2 * len(df_motif_admission.columns))
thresh = int(0.2 * len(df_medicament.columns))

df_patient = df_patient.dropna(thresh=thresh)
df_maladie = df_maladie.dropna(thresh=thresh)
df_motif_admission = df_motif_admission.dropna(thresh=thresh)
df_medicament = df_medicament.dropna(thresh=thresh)

# 3. Suppression si colonne-clé manquante
df_patient = df_patient.dropna(subset=['age', 'gender'])

# 4. Imputation générique
df_patient = df_patient.na.fill({'gender': 'UNKNOWN', 'age': -1})

# 5. Mise en majuscule du genre
df_patient = df_patient.withColumn('gender', upper(col('gender')))

# 6. Casting des colonnes principales
df_patient = df_patient.withColumn("age", col("age").cast("int"))

```

Puis on affiche un aperçu :

Table view

	patient_id	profile_id	dep_name	age	gender	ethnicity	race	lang	religion	maritalstatus	employstatus	insurance_status	disposition
1	113	55_Male_Non-...	A	55	MALE	Non-Hispanic	White or C...	English	Christian	Married	Self Employed	Medicaid	Admit
2	151	56_Male_Hisp...	A	56	MALE	Hispanic or L...	Other	Other	Catholic	Married	Disabled	Medicare	Admit
3	228	39_Female_His...	A	39	FEMALE	Hispanic or L...	Other	English	Christian	Married	Not Employed	Medicaid	Discharge
4	406	27_Female_No...	C	27	FEMALE	Non-Hispanic	White or C...	English	Catholic	Single	Full Time	Commercial	Discharge
5	427	76_Female_No...	B	76	FEMALE	Non-Hispanic	White or C...	English	Catholic	Married	Retired	Commercial	Admit

Table view

	patient_id	2ndarymalig	abdomhernia	abdomnlpain	abortcompl	acqfootdef	acrenlfail	acutecvd	acutemi	acutphanm	adjustmentdisorders	adltrespfl
1	162	0	0	0	0	0	0	0	0	0	0	0
2	217	0	0	0	0	0	0	0	0	0	0	0
3	320	0	0	0	0	0	0	0	0	0	0	0
4	772	0	0	0	0	0	0	0	0	0	0	0
5	776	0	0	0	0	0	0	0	0	0	0	0

Table view

	patient_id	cc_abdominalcramping	cc_abdominaldistention	cc_abdominalpain	cc_abdominalpainpregnant	cc_abnormallab	cc_abscess	cc_addictionproblem	cc_agitation	cc_alcoh
1	198	0	0	0	0	0	0	0	0	0
2	566	0	0	0	0	0	0	0	0	0
3	809	0	0	0	0	0	0	0	0	0
4	948	0	0	0	0	0	0	0	0	0
5	1028	0	0	0	0	0	0	0	0	0

Table view

	patient_id	meds_analgesicandantihistamin...	meds_analgesics	meds_anesthetics	meds_anti_obesitydrugs	meds_antiallergy	meds_antiarthritics	meds_antiasthmatics	meds_antibiotics
1	663	0	0	0	0	0	0	0	0

ETL – Chargement de la table bronze

Notre couche bronze est terminée, nous enregistrons donc nos fichiers au format delta avant de passer à un prochain notebook pour créer nos tables, et ainsi de suite jusqu'au dernier notebook.

```python
# Écriture des données Bronze
df_patient.write.format("delta").mode("overwrite").option("overwriteSchema", "true").save("Files/bronze/hospital-triage/")
df_maladie.write.format("delta").mode("overwrite").option("overwriteSchema", "true").save("Files/bronze/hospital-triage/")
df_motif_admission.write.format("delta").mode("overwrite").option("overwriteSchema", "true").save("Files/bronze/hospital-triage/")
df_medicament.write.format("delta").mode("overwrite").option("overwriteSchema", "true").save("Files/bronze/hospital-triage/")
```

1 min 16 sec - Command executed in 1 min 16 sec 345 ms by Inès SEKARI on 10:36:47 PM, 5/21/25 PySpark (Python)

Spark jobs (20 of 20 succeeded) Resources Log

Notebook : 02_bronze_to_table

Objectif

- Migrer les fichiers de la **couche bronze** (`/files/bronze`) vers des tables Delta cataloguées.
- Garantir la cohérence et la traçabilité entre les fichiers déposés et la table officielle "bronze".
- Permettre la requête SQL directe sur les jeux de données bronze (`bronze.nom_table`).

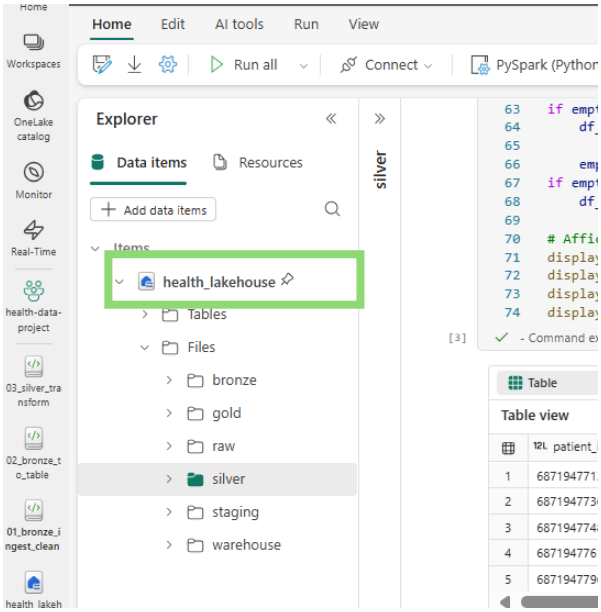
```python
# 0. ENVIRONNEMENT
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()


# 1. VARIABLES
datasets = [
    "maladie",
    "icd_code",
    "meds_code",
    "motifs_code",
    "medicament",
    "motif_admission",
    "patient"
]
file_format = "delta"


# 2. MIGRATION FILE → TABLE (écriture dans le Lakehouse)
for dataset in datasets:
    src_path = f"Files/bronze/{dataset}"
    table_name = f"bronze_{dataset.replace('-', '_')}"  # remplace '-' car interdit dans les noms de tables SQL
    # Lecture du fichier Delta
    df = spark.read.format(file_format).load(src_path)
    # Écriture dans le Lakehouse (table Delta)
    df.write.format("delta").mode("overwrite").saveAsTable(table_name)
    print(f"✅ Table créée/MAJ : {table_name}")


# 3. VÉRIFICATION : Afficher toutes les tables du Lakehouse
display(spark.sql("SHOW TABLES"))



1 min 48 sec - Session ready in 12 sec 353 ms. Command executed in 1 min 36 sec 115 ms by Lena DEMANOU on 5:02:22 PM, 5/24/25
```

```
✅ Table créée/MAJ : bronze_maladie
✅ Table créée/MAJ : bronze_icd_code
✅ Table créée/MAJ : bronze_meds_code
✅ Table créée/MAJ : bronze_motifs_code
✅ Table créée/MAJ : bronze_medicament
✅ Table créée/MAJ : bronze_motif_admission
✅ Table créée/MAJ : bronze_patient
```

ETL – Modèle dimensionnel

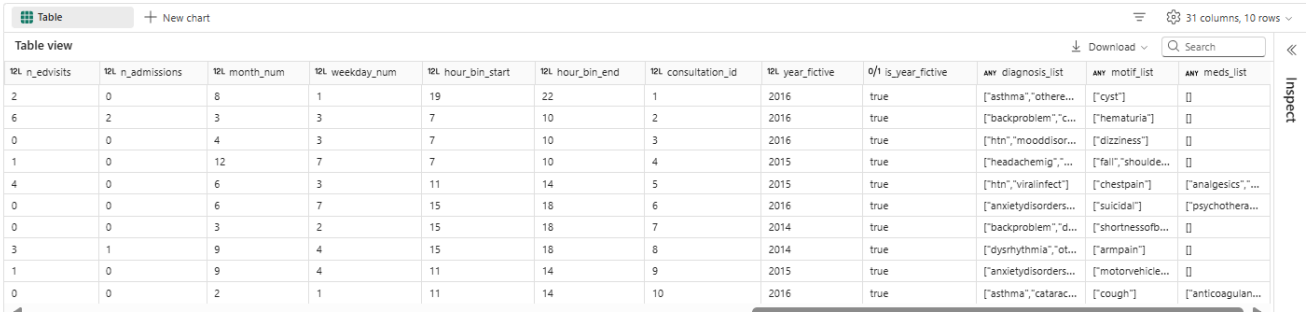
Nous avons ensuite procédé à la transformation des tables bronze en table silver puis en table gold pour alimenter le **modèle dimensionnel** (tables de faits et tables de dimensions selon le schéma en étoile), via des notebooks dédiés à chaque couche. Chaque notebook correspond à une couche (bronze, silver, gold), facilitant la maintenance et l'itération sur le modèle.

Pour ce faire, tout nos notebooks sont connectés à notre lakehouse crée, Health_lakehouse.



La couche Gold est l'endroit où le modèle dimensionnel se fera. Un notebook est présent pour la création des tables de dimension et de la table de fait > Notebook : 5_gold_star_schema. Les tables de dimension sont créées avant la table de fait car les clés étrangères de ces tables iront vers la table de fait.

Exemple de création de tables de dimension :
Prenons la dimension maladie, qui nécessite l'ajout du référentiel icd_code afin de l'enrichir.
On démarre la session spark et on lit nos différents fichiers : Sachant que silver_df_hopital est la concaténation de nos 4 tables patient, médicament, motif admission et maladie, voici un extrait : Nous avons transformer les colonnes binaires en une seule colonne avec liste des maladies, médicament et motif d'admission.



```python
from pyspark.sql import SparkSession
import pyspark.sql.functions as F

# 1. INIT SPARK & LOAD SILVER ----------------------------------------------
spark = SparkSession.builder.appName("StarSchemaHealthcare").getOrCreate()

# Lecture données silver
silver_df_hopital = spark.read.format("delta").load("Files/silver/hopital")
silver_df_icd_code = spark.read.format("delta").load("Files/silver/icd_code")
silver_df_meds_code = spark.read.format("delta").load("Files/silver/meds_code")
silver_df_motifs_code = spark.read.format("delta").load("Files/silver/motifs_code")

display(silver_df_hopital.head(10))
display(silver_df_icd_code.head(10))
display(silver_df_meds_code.head(10))
display(silver_df_motifs_code.head(10))
```

✓ 23 sec - Command executed in 23 sec 152 ms by Lena DEMANOU on 6:17:57 PM, 5/24/25

On remarque que dans silver_df_hopital, trois colonnes nous intéressent : consultation_id , pour l'identifiant unique de consultation, patient_id pour l'identification du patient et diagnosis_list qui est l'entièreté des colonnes diagnostic qui correspondait au patient. Cela signifie que le patient a ces antécédents médicaux. Nous voulons enrichir ces données avec notre dictionnaire icd_code.
Nous avons donc « explosé » la liste de diagnostic afin d'en créer des lignes, puis nous avons joins le dictionnaire icd_code à nos données, au passage, nous avons renommé diagnosis par maladie pour que cela soit plus parlant. On voit bien que les données ont été enrichi en ne gardant que les colonnes nécessaires (patient_id n'étant pas nécéssaire pour la dimension maladie).

```python
# DIMENSION MALADIE - enrichie via mapping ICD
from pyspark.sql.functions import explode
from delta.tables import DeltaTable

output_path = "Files/gold/dim_maladie/"

# 1. Construction de la dimension maladie
from pyspark.sql.functions import explode


# 1. "Exploser" les listes : une ligne par code maladie
maladie_exploded = silver_df_hopital\
    .withColumn("diagnosis", explode("diagnosis_list")) \

# 2. Dimension maladie : couple patient_id, diagnosis (optionnellement distinct)
dim_maladie = maladie_exploded.select("consultation_id","patient_id", "diagnosis").distinct()

# 3. Enrichir avec description, catégorie, etc.
dim_maladie = dim_maladie.join(silver_df_icd_code, on="diagnosis", how="left")

dim_maladie = dim_maladie.withColumnRenamed("diagnosis", "maladie")

consultation = silver_df_hopital.select("consultation_id").distinct()
dim_maladie = consultation.join(dim_maladie, on="consultation_id", how="left")
```

Table | New chart | 6 columns, 20 rows

Table view | Download | Search

	consultation_id	maladie	patient_id	Code_CIM-10	Catégorie_médicale	Description
1	148	diabmelnoc	8589964878	E14	Endocrinologie	Diabète sucré
2	148	otacqdefor	8589964878	0	domaine_clinique	Déformation ac...
3	148	viralinfect	8589964878	B34	Infectiologie	Infection virale
4	148	otitismedia	8589964878	H66	ORL	Otite moyenne
5	148	unclassified	8589964878	NULL	NULL	NULL
6	148	asthma	8589964878	J45	Pneumologie	Asthme
7	148	skininfectn	8589964878	L08	Dermatologie	Infection de la p...
8	148	thyroiddsor	8589964878	E07.9	domaine_clinique	Maladie de la th...
9	463	diabmelnoc	8589959342	E14	Endocrinologie	Diabète sucré
10	463	esophgealdx	8589959342	K22	Gastroentérologie	Maladies de l'œ...
11	463	diverticulos	8589959342	K57	Gastroentérologie	Diverticulose de...
12	463	neoplsmunsp	8589959342	C80	Oncologie	Tumeur
13	463	othergidx	8589959342	K92.9	domaine_clinique	Maladie gastro-i...
14	463	bladdercncr	8589959342	C67	Oncologie	Tumeur maligne...

Code_CIM-10
L08

Nous avons une table de fait fact_consultation et voici comment nous l'avons construite.
Screen notebook

Grâce à l'aperçu ci-dessus, nous voyons bien que la table de fait est utilisée pour enregistrer les différentes clés se référant aux dimensions.

ETL – SCD
La gestion des évolutions dans les tables de dimension (méthode **SCD – Slowly Changing Dimension**) permet de conserver l'historique des informations sensibles aux changements dans le temps.
Dans notre modèle, nous avons adapté la méthode d'historisation selon la nature des dimensions et les besoins analytiques :

Dimension	Colonnes historisées	Méthode SCD	Colonnes non historisées
Dim_Patient	Aucune	SCD 0	Toutes (profil patient inchangé)
Dim_Maladie	Toutes	SCD 1	(Ecrasement lors des corrections)

- **SCD 0** (Dim_Patient) :
  - Les valeurs restent stables dans le temps (aucune historisation).
  - Exemple : Si le statut marital ou la langue change, la valeur **n'est pas mise à jour** dans l'entrepôt (on garde toujours la valeur d'origine).
- **SCD 1** (Dim_Maladie) :
  - Écrasement à chaque modification, sans traçabilité de l'historique.
  - Exemple : Si la catégorie ou la description d'une maladie change dans le référentiel, la nouvelle valeur remplace immédiatement l'ancienne (on perd l'historique).

Ce choix permet une architecture simplifiée tout en restant pertinente pour l'analyse :
- Les **informations patients** sont considérées comme fixes dans le modèle.
- Les **référentiels médicaux** sont mis à jour à chaque correction ou évolution métier.

ETL – Workflow
À chaque étape du pipeline, nous avons mis en place un workflow clair, aussi bien dans Fabric que dans la structuration sur GitHub :
- Étape 1 : Ingestion des fichiers bruts & pré-nettoyage (Zone Bronze)
- Étape 2 : Nettoyage, contrôle qualité, transformation et enrichissement (Zone Silver)
- Étape 3 : Transformation vers les dimensions et la table de faits + SCD (Zone Gold)
- Étape 4 : Exploitation et visualisation des données avec Power BI

Ce workflow garantit une traçabilité des traitements : chaque opération est consignée dans un notebook, et toute évolution du pipeline peut être documentée et versionnée. Cela assure la fiabilité du processus, la reproductibilité des résultats et la simplicité en cas d'audit ou de correction.

6. Construction du modèle sémantique/Power BI

Le modèle sémantique permet aux utilisateurs finaux d'explorer et d'analyser les données de façon intuitive, sans s'encombrer de la complexité technique des sources brutes ou des étapes ETL. Il repose sur une organisation logique avec une table de faits, des dimensions, des mesures calculées, des colonnes calculées, ainsi que d'éventuelles hiérarchies
Dans notre projet, nous incluons toutes les tables transformées (uniquement Gold).
Ce modèle fournit ainsi une vue d'ensemble : chaque table de faits (par exemple, les séjours patients, les diagnostics) est reliée à ses tables de dimension (patients, maladies, hôpitaux, temps), garantissant la cohérence des analyses et facilitant le croisement de critères métier.
Lien entre les tables :
- La table de faits Fact_consultation est liée aux dimensions Dim_patient, Dim_hopital, Dim_maladie, Dim_motif_admission, Dim_medicament, Dim_temps, via leur clé respective.
- Ce schéma en étoile favorise la lisibilité et la performance des analyses, et permet la création rapide de rapports interactifs.

À l'issue de cette étape, le modèle sémantique est opérationnel et prêt à être exploité dans Power BI, pour des dashboards et des analyses avancées adaptées aux besoins métier.

- Mesures KPI’s

Nous avons effectué un certain nombre de mesure KPI’s d’utilité pour notre dashboard, en voici un exemple :

```
#a. Nombre de passages aux urgences par période (jour/mois/année)
kpi_passages_par_jour = fact_consultation.groupBy("date_id").count().withColumnRenamed("count", "nb_passages")

display(kpi_passages_par_jour.head(5))


fact_consultation = fact_consultation.join(
    kpi_passages_par_jour,
    on="date_id",
    how="left"
)

display(fact_consultation.head(5))

Lena DEMANOU

from pyspark.sql import functions as F
from pyspark.sql.window import Window
from pyspark.sql.functions import col, datediff, lag

#c. Répartition des arrivées par mode d’admission
kpi_arrival_mode = dim_patient.groupBy("arrivalmode").count().orderBy("count", ascending=False)

display(kpi_arrival_mode.head(5))

dim_patient = dim_patient.join(
    kpi_arrival_mode,
    on="arrivalmode",
    how="left"
)

display(dim_patient.head(5))
```

Table | New chart | 2 colun

Table view | Download | Search

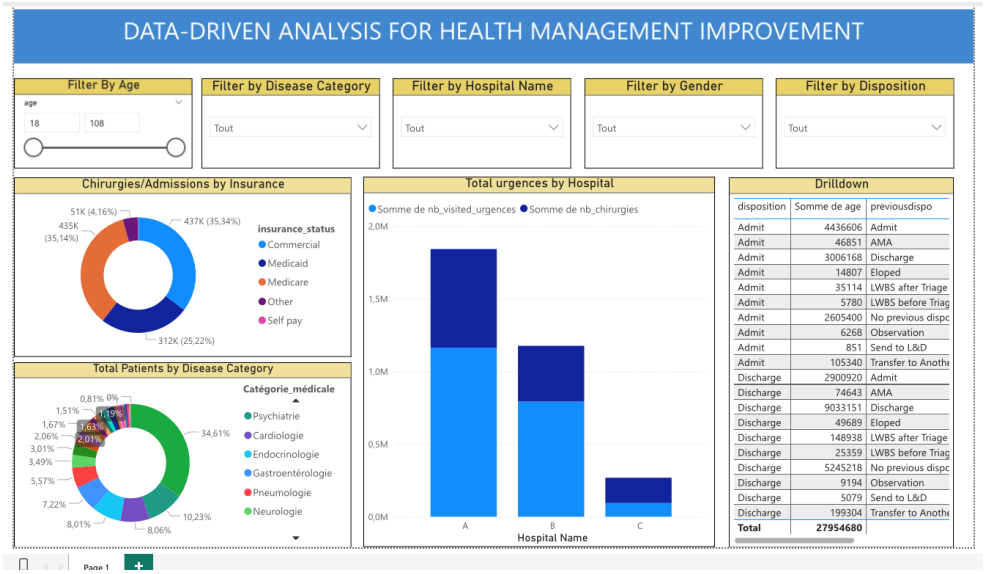
	date_id	nb_passages
1	2015_March...	598
2	2014_July_M...	416
3	2016_Dece...	323
4	2016_Febru...	284
5	2016_April_...	234

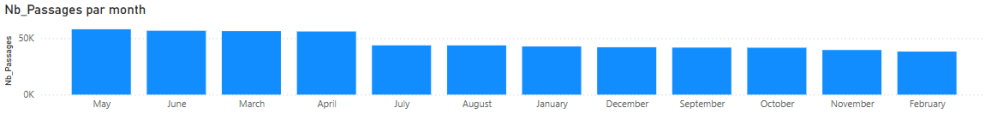
Table | New chart | 16 colun

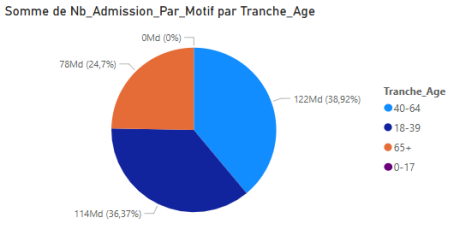
Table view | Download | Search

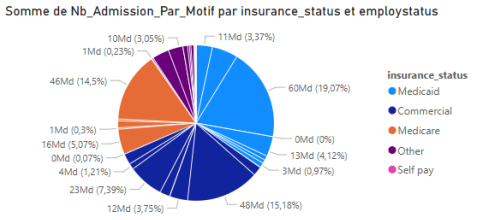
	date_id	consultation_id	patient_id	hopital_nom	age	gender	disposition	previousdispo	liste_maladie	liste_medicament	liste_motif_admission	nb_chi
1	2016_Nove...	9	68719478631	A	48	MALE	Discharge	Admit	[]	[]	["legpain"]	3
2	2015_Dece...	15	68719479565	C	40	FEMALE	Discharge	No previous dispo	["hyperlipidem","...	[]	["eyeredness","eyeproble...	2
3	2015_July_W...	17	68719479956	A	65	MALE	Admit	Discharge	[]	["unclassifieddrugpro...	["chestpain"]	0
4	2017_Septe...	35	68719483747	A	39	FEMALE	Discharge	Discharge	["attentiondeficitc...	[]	["suicidal","alcoholintoxic...	0
5	2014_Nove...	37	68719484194	C	26	FEMALE	Discharge	Discharge	["headachemig","...	[]	["armpain"]	4

7. Visualisation et analyses









8. Conclusion et perspectives

Dans le cadre de ce projet, nous avons exploité l'environnement **Microsoft Fabric** pour construire un **pipeline analytique de bout en bout** orienté santé. En partant de données brutes, nous avons mis en œuvre une architecture en couches **Bronze → Silver → Gold**, assurant une **nettoyage, transformation et structuration progressive** des données via des **tables Delta Lake**.

Des **notebooks Python et SQL** ont été mobilisés pour effectuer les étapes clés :

- **Nettoyage des données sensibles**
- **Jointures intelligentes entre les dimensions et la table de faits**

- **Ajout de KPIs métiers** (âge, tranche d'âge, durée de séjour…)

Enfin, une **modélisation sémantique** a été conçue dans Power BI (modèle gold), connectée au **Lakehouse**, puis utilisée pour créer un **rapport interactif** permettant :

- D'identifier les profils patients les plus fréquents aux urgences
- D'optimiser l'allocation des ressources médicales
- Et d'aider à la **prise de décision par les autorités sanitaires**

Ce projet démontre **l'efficacité du cloud analytique** pour transformer des données de santé complexes en **insights exploitables** pour améliorer les soins.

9. Sources

https://learn.microsoft.com/fr-fr/fabric/fundamentals/microsoft-fabric-overview