

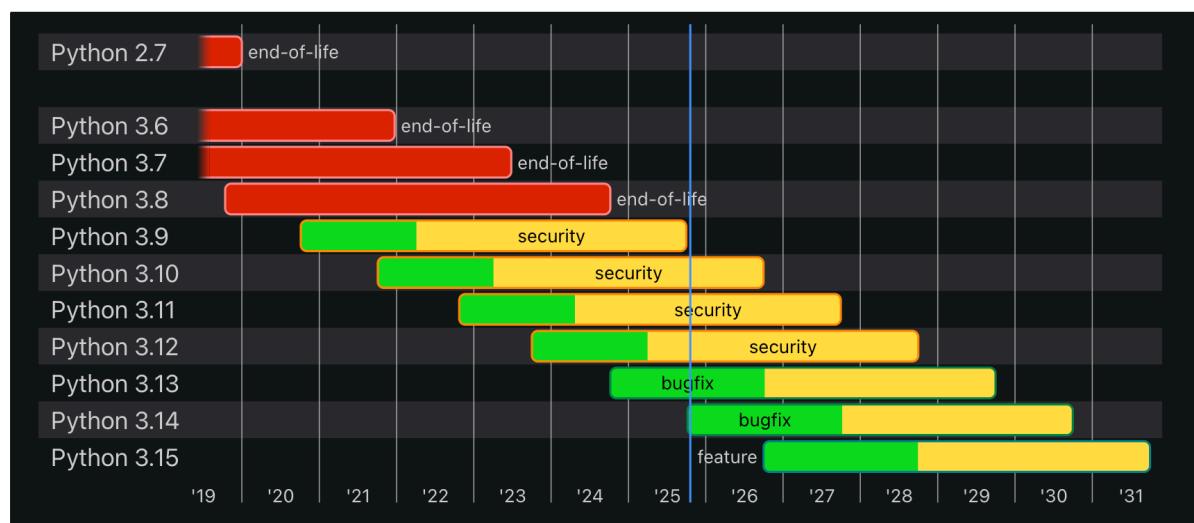
# Partage de connaissances - Master MIASHS

Partage de connaissances - Master MIASHS

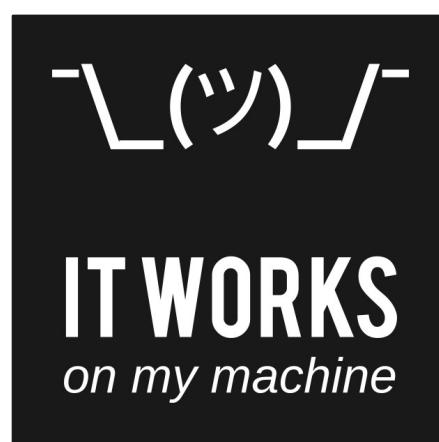
- Gérer les dépendances d'un projet Python
- Utiliser le terminal
- Partager et collaborer sur du code
- VS Code
- Positron: nouveau IDE pour R
- Rédaction scientifique et communication autour de projets data
- Librairie Python
- Business Intelligence
- Considération autour des modèles de prédictions
- Feature engineering
- Les pipelines sklearn
- Sauvegarder un modèle
- MLFlow: Surveiller, enregistrer et réutiliser des modèles
- L'écosystème d'outils data

## Gérer les dépendances d'un projet Python

Python et les librairies Python évoluent rapidement:



Un code écrit pour `pandas 1.0` ne fonctionnera probablement pas avec `pandas 2.0`.



On a besoin d'expliciter les dépendances utilisées par un projet afin de garantir la reproductibilité de l'environnement d'exécution.

**uv** est devenu le principal outil pour gérer les environnements (remplace progressivement **conda** ).

```
1 # Créer un environnement virtuel
2 uv init
3
4 # Ajouter des dépendances
5 uv add pandas
6
7 # Activer l'environnement dans VS Code ou dans le terminal
8 .venv\Scripts\activate      # Windows (PowerShell)
9 source .venv/bin/activate   # MacOS
10
11 # Synchronise l'environnement virtuel avec le `uv.lock`
12 uv sync
13
14 # uvx permet d'exécuter "à la volée" des commandes dépendant de librairies Python
15 uvx marimo edit
```

## Utiliser le terminal

```
1 # Se déplacer dans un dossier
2 cd ./dossier
3 # Et revenir au dossier parent
4 cd ..
5
6 # Flèche ⬆ / ⬇ pour se déplacer dans l'historique
7 # CTRL + C pour interrompre une commande en cours d'exécution
8
9 # Ouvrir le dossier dans VS Code (-r pour ouvrir dans la fenêtre actuel VS Code)
10 code . -r
11
12 # MacOS - définir des alias
13 alias c='code . -r'
14 alias cdmaster='cd "/Users/mathisderenne/Documents/02 - Scolaire/M2 MIASHS"'
15 alias cdgh='cd /Users/mathisderenne/GitHub/'
16
17 # Windows (PowerShell) - définir des alias
18 function c { code . -r }
19 function cdgh { cd "C:/Users/mathisderenne/GitHub" }
```

Améliorer son expérience dans le terminal en installant un shell: <https://starship.rs/>

## Partager et collaborer sur du code

**Git** : permet de versionner un dossier:

- `tutoriel interactif` pour apprendre à maîtriser Git

- interface de l'extension VS Code Git

`Git` ≠ `GitHub / GitLab` : plateforme d'hébergement de projets Git

#### Quelques conventions sur la structure des projets Git:

- `README.md` décrit le projet
- `.gitignore` précise les éléments ignorés par Git
- dossiers courant:
  - `./src` : code source du projet
  - `./assets` : images, logos
  - `./docs` , `./data` , `./test`

`GitHub Pages` permet d'héberger une page statique gratuitement → utile pour CV en ligne ([tutoriel en 5 minutes](#)) / projets démos.

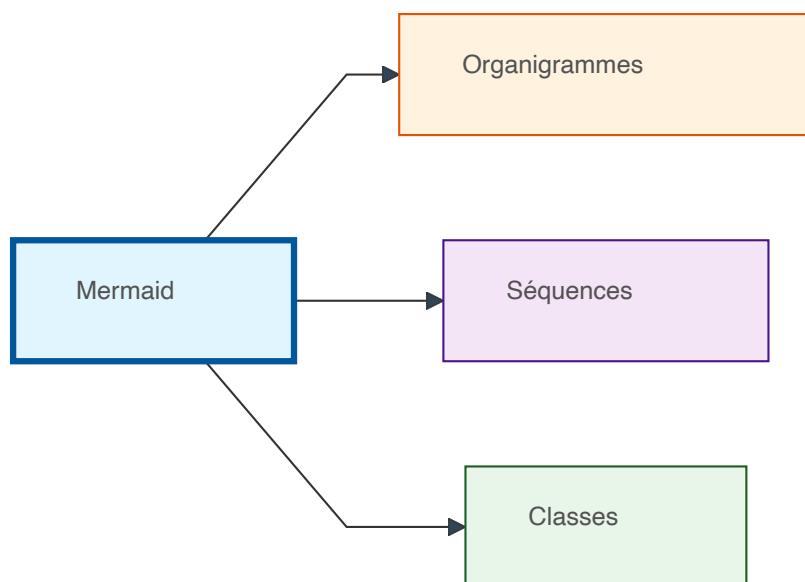
#### VS Code

- éditer les notebooks (Python et R)
- formater le code automatiquement ([extension Ruff](#))
- lire des fichiers PDF ([extension PDF Viewer](#))
- explorer des fichiers de données tabulaires ([extension Data Wrangler](#))
- GitHub Copilot ([gratuit pour les étudiants](#))
- mes `settings.json`

#### Positron: nouveau IDE pour R

#### Rédaction scientifique et communication autour de projets data

- rédiger rapport de stage/alternance: `Typst` ([extension Tinymist](#))
- présenter des projets data avec `MystMD`
- créer des diagrammes avec `Mermaid` ([Mermaid Live Editor](#))



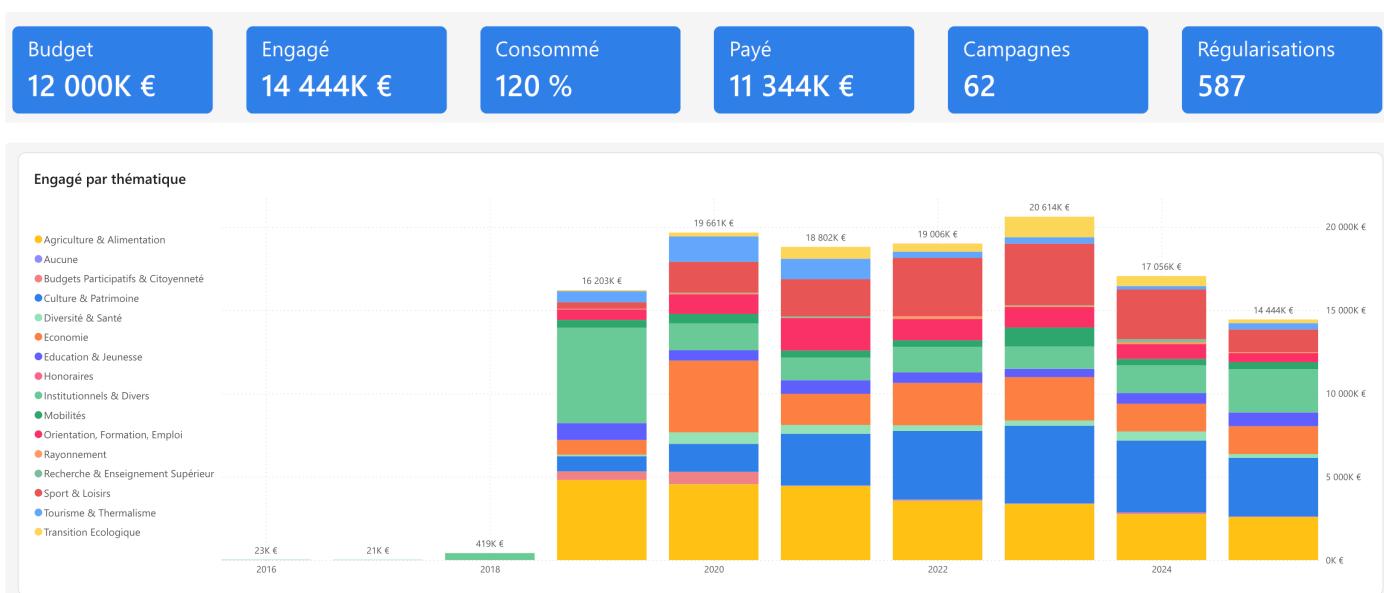
- création de web data app avec **Streamlit**
  - notebooks **Marimo** (démô [ici](#))

## Librairie Python

- manipuler des données tabulaires: [polars](#) ([user guide](#), [comparaison avec pandas](#))
  - création de visuels: [altair](#) ([exemple](#))

## Business Intelligence

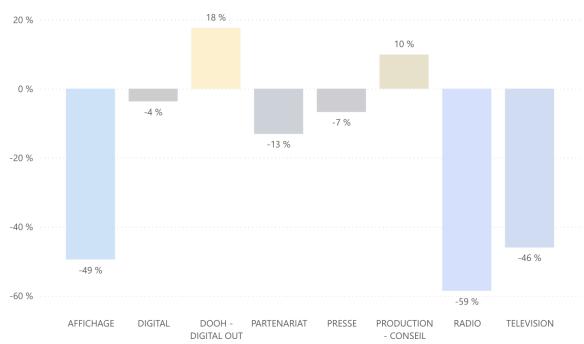
Rendre interprétable et actionnable les données métiers.



#### Evolution engagé AE+FT par média

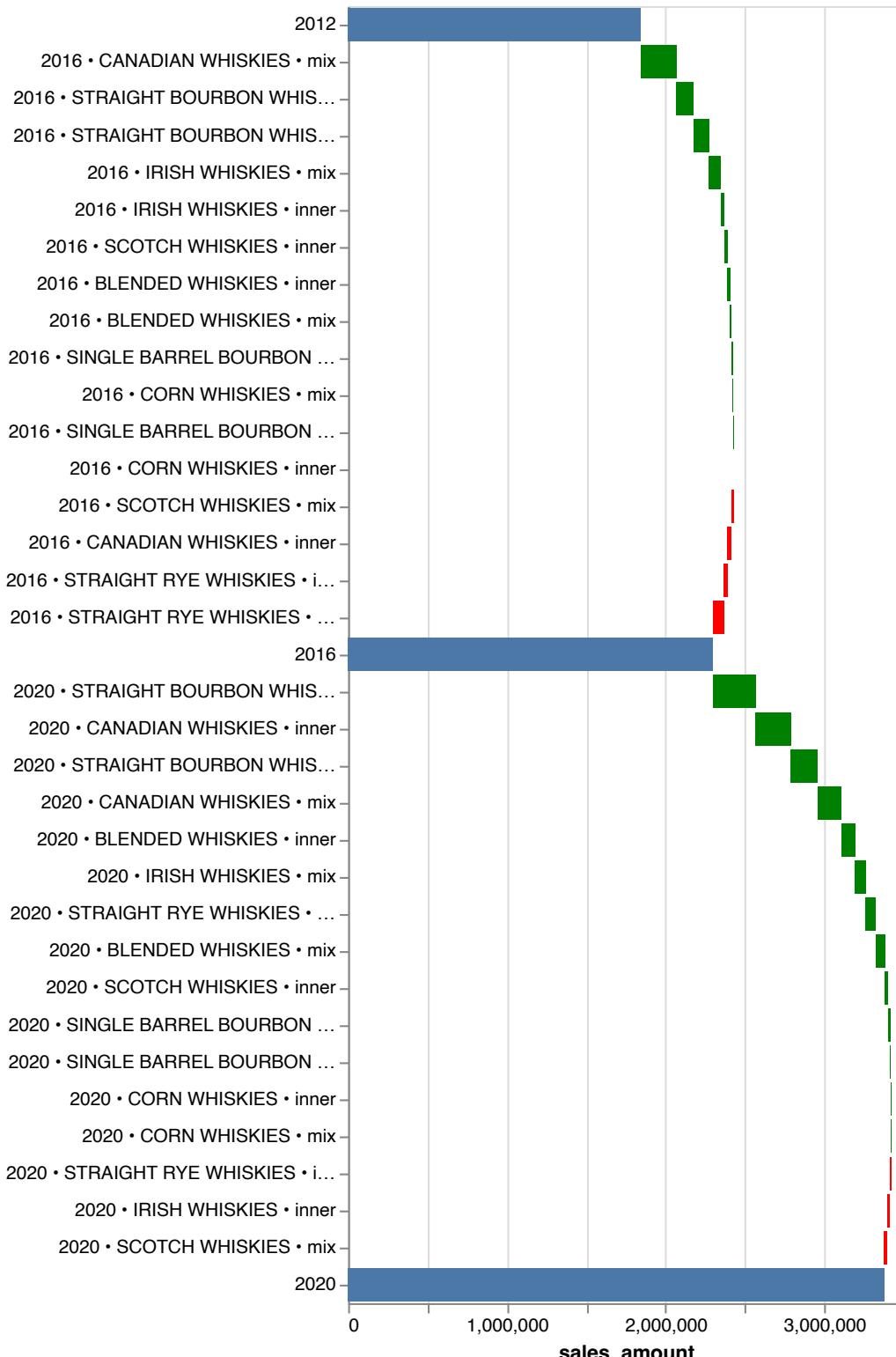
Année Media	2024			2025		
	Engagé annuel	Engagé à date	Evolution vs n-1	Engagé annuel	Engagé à date	Evolution vs n-1
AFFICHAGE	1 040K €	1 029K €	+93 %	525K €	520K €	-49 %
CINEMA				3K €	3K €	
DIGITAL	1 892K €	1 725K €	-18 %	1 692K €	1 662K €	-4 %
DOOH - DIGITAL OUT OF HOME	305K €	305K €	+42 %	358K €	358K €	+18 %
PARTENARIAT	6 803K €	6 208K €	-29 %	5 702K €	5 396K €	-13 %
PRESSE	6 148K €	5 501K €	-5 %	5 831K €	5 131K €	-7 %
PRODUCTION - CONSEIL	96K €	90K €	-11 %	98K €	98K €	+10 %
PRODUCTION - DIGITAL				1K €	1K €	
RADIO	414K €	399K €	-72 %	166K €	166K €	-59 %
TELEVISION	358K €	126K €	-31 %	68K €	68K €	-46 %

#### Evolution engagé à date 2025 vs 2024 par média



Décomposer un indicateur à partir de ses composantes →

- par catégorie (de produits, etc.)
- dans le temps
- calculer l'évolution en % YoY
- décomposer par quantité et panier moyen: [Answering "Why did the KPI change?" using decomposition](#)



- fonctionne aussi avec des métriques de funnel (voir): `revenue = impressions * click_rate * conversion_rate * spend`

Package Python `icanexplain`

## Considération autour des modèles de prédictions

- faut-il mieux prédire `prix total` ou bien `prix moyen` et `quantité` ?  
→ prédire directement le prix total: offre généralement des meilleures performances, modèle simple mais effet boîte noir

- prédire indépendamment les deux variables:
  - explicite différents facteurs d'influence du **prix total**
  - meilleure interprétabilité et actionnabilité
  - permet d'isoler les erreurs et d'analyser la performance individuelle des modèles

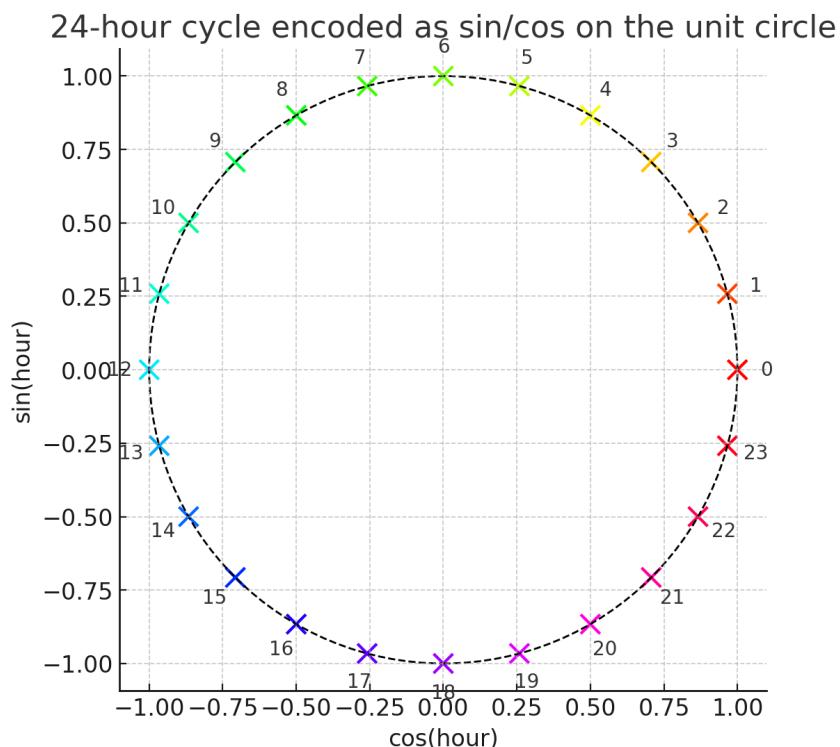
## Feature engineering

**Objectif:** améliorer la performance d'un modèle en lui fournissant les données explicatives clés

- apporter des données externes explicatives (ex: données de localisation)
- transformant les features pour réduire la complexité du problème

Exemples de features engineering:

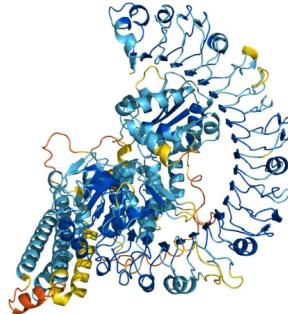
- faut-il mieux fournir: (hauteur et largeur) ou (surface et ratio hauteur/largeur)
- métrique dérivée: **ancienneté = aujourd'hui - date\_creation**
- encodages cycliques: **angle en degré, heure de la journée**



- encoder des coordonnées spatiales (x, y)
  - représentation polaire (angle + distance)
  - sans apport de données externes: K-means distance
  - avec des données externes: distances aux grandes villes
  - apport de données externes: démographique (population, niveau de revenu, etc.), géographique (surface, type de terrain), etc.
- données textuelles: TF-IDF (Term Frequency-Inverse Document Frequency)
- données temporelles: analyser les saisonnalités à différentes échelles (par semaine,

mois, jour de l'année) avec par exemple la transformée de Fourier ou vos connaissances métiers et ajouter des colonnes explicatives pertinente (`is_holiday`, `is_weekend`)

- cas de problèmes beaucoup plus complexe: comment encoder une position du jeu d'échec ? comment encoder une image ? comment encoder des données textuelles ? comment encoder le repliement de protéines ? ([Geometric Deep Learning](#))



→ nombreux types de données ne possèdent pas une modélisation évidente / pleinement efficace

→ souvent une combinaison: features engineering avancé; architecture spécifique; grande puissance de calcul.

→ l'absence de modélisation pleinement efficace participe aussi à l'absence de robustesse des modèles.

## Les pipelines sklearn

```
1 import polars as pl
2
3 # Load data
4 df = pl.read_csv("../data/data.csv")
5
6 # Preprocess data at row-level check ✅
7 df = df.with_columns(
8     [
9         pl.col("date").dt.weekday().alias("weekday"),
10        pl.col("date").dt.hour().alias("hour"),
11        pl.col("date").dt.month().alias("month"),
12    ]
13 )
14
15 # Preprocess data row using heuristics from dataset ❌
16 import polars.selectors as cs
17
18 df = df.with_columns(
19     (cs.numeric() - cs.numeric().mean()) / cs.numeric().std()
20 )
21
22 # Preprocess data using sklearn pipeline ✅
```

```

23 from sklearn.preprocessing import StandardScaler, OneHotEncoder
24 from sklearn.compose import ColumnTransformer
25
26 numeric_features = df.select(cs.numeric()).columns
27 categorical_features = df.select(cs.categorical()).columns
28
29 preprocessor = ColumnTransformer(
30     transformers=[
31         ("num", StandardScaler(), numeric_features),
32         ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_features),
33     ]
34 )
35
36 # Define model
37 from sklearn.linear_model import LinearRegression
38
39 model = LinearRegression()
40
41 # Define pipeline
42 from sklearn.pipeline import Pipeline
43
44 pipeline = Pipeline(steps=[
45     ("preprocess", preprocessor),
46     ("regressor", LinearRegression())
47 ])
48
49 # Used the same way as model
50 model.fit(X_train, y_train)
51 y_pred = model.predict(X_test)

```

Pourquoi utiliser les pipelines `sklearn` :

- Regrouper pré-traitement + modèle dans un seul objet `pipeline`
- Éviter les fuites de données entre les jeux d'entraînement et de validation (lors d'un `test_train_split` ou `cross-validation`)
- Faciliter la réutilisation et le déploiement
- Simplifier la validation croisée et la recherche d'hyperparamètres:
  - `StandardScaler` → `PCA(n_components=0.7)` → `RandomForest(n_estimators=100)`
- Modularité et extensibilité → on peut facilement combiner plusieurs pipelines, créer des branches parallèles (`ColumnTransformer`), et insérer facilement de nouvelles étapes personnalisées (`FunctionTransformer`)

## Sauvegarder un modèle

```
1 from joblib import load, dump
2
3 # Train the model
4 model = model.fit(X_train, y_train)
5
6 # Save the model
7 dump(model, "../model/model.pkl")
8
9 # Load the model
10 model = load("../model/model.pkl")
```

## MLFlow: Surveiller, enregistrer et réutiliser des modèles

- C'est quoi une API ?

## L'écosystème d'outils data

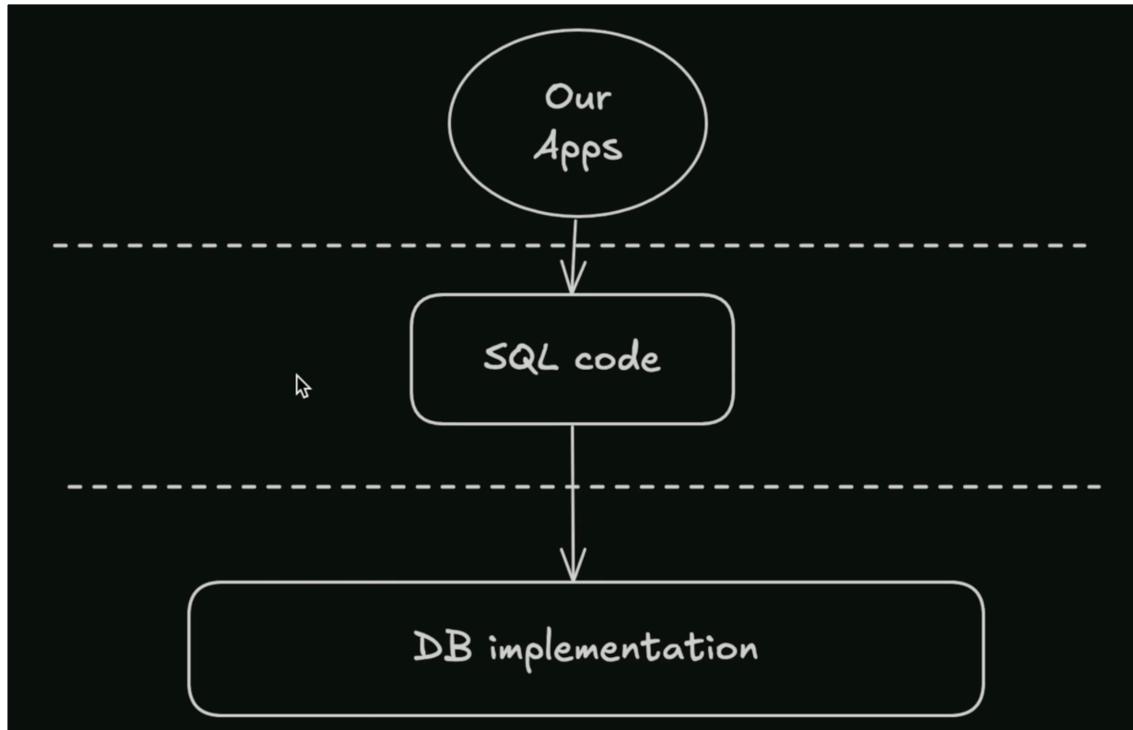
- une base de données (OLTP ou OLAP)
- un outil d'ingestion de données
- un outil de transformation de données
- un outil de présentation des données
- un outil d'orchestration



SQL needs to die (plus d'infos [ici](#) et [ici](#))

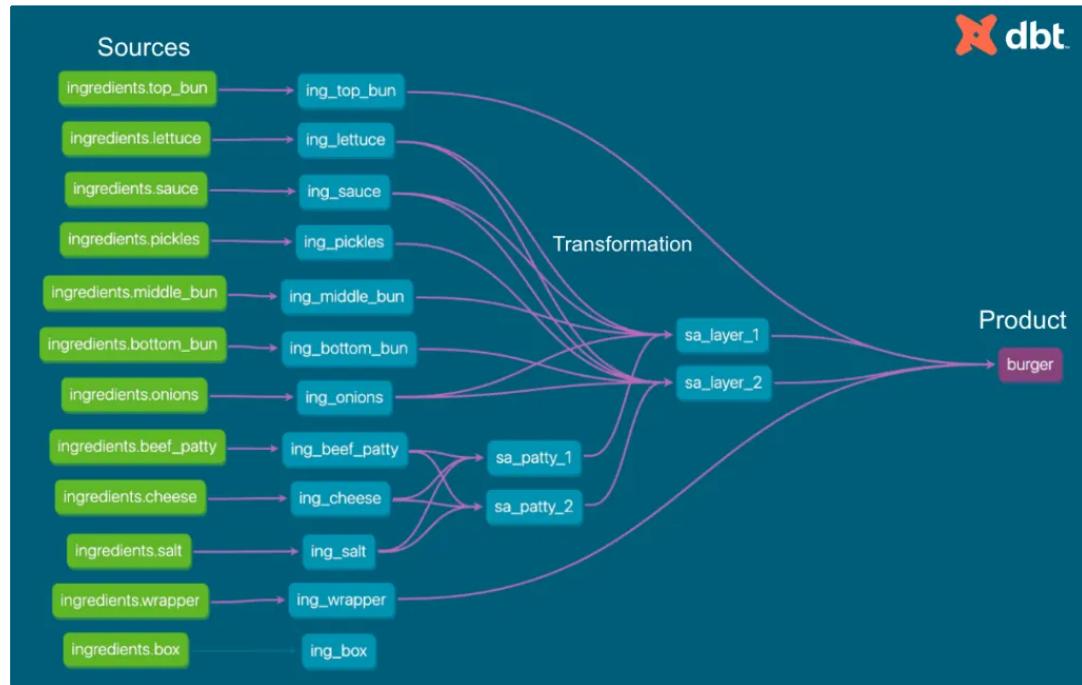
- SQL a très peu évolué depuis ses débuts
- SQL n'est pas standardisé (reste un langage relativement bas niveau des bases de données)
- SQL n'est pas aisément composable
- SQL n'a pas d'interface programmatique native
- SQL n'a pas connaissance des types de données

- duplication de la définition des données entre DB et applicatif

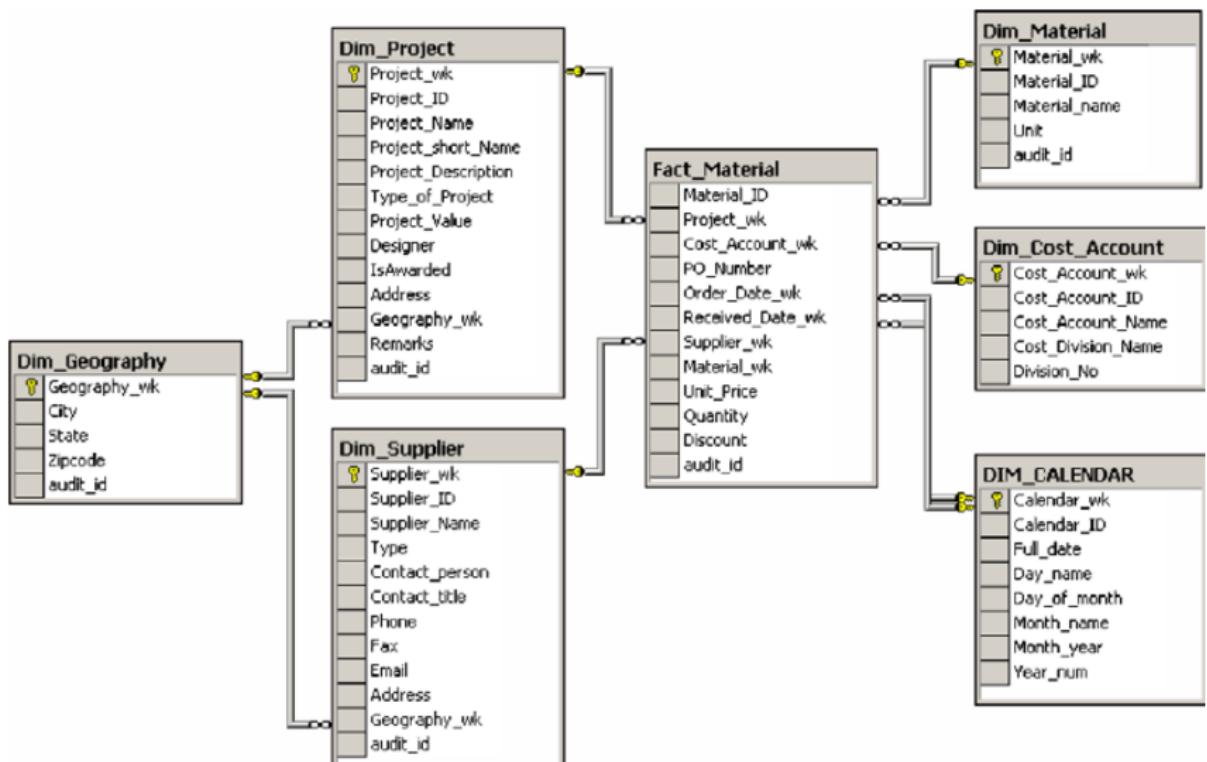


#### But SQL is still everywhere

- SQL reste toujours l'interface **finale** pour parler au moteur de la base de données
- La syntaxe SQL continue de s'améliorer ([DuckDB](#) - [FriendlySQL](#): FROM avant SELECT, GROUP BY ALL, etc.)
- Côté applicatif: les ORM (object relational mapping) ou [sqlc](#)
- Côté analytique:
  - outils de transformation basée sur du SQL (dbt ou SQLMesh):
    - Plusieurs fichiers SQL sont responsables de nettoyer et structurer progressivement la donnée.
    - Ces outils comprennent le code SQL et créent des graphs de dépendances acycliques



- La compréhension du code SQL permet d'avoir du table ou du column level lineage.
- Ces outils permettent de documenter directement les tables ou colonnes de la base de données
- notion de layer sémantique (Modèle sémantique Power BI, [LookML](#), [Malloy.dev](#) - [blog de post](#)) → malgré tout nécessaire car on ne peut pas tout pré-calculer et on peut aussi faire des jointures syntaxiquement valables mais qui n'ont pas de sens sémantiquement
- modélisation dans l'entrepôt de données (modèle en étoile - table de faits et de dimensions)



#### **Tendances globales:**

- nouveaux outils cloud avec des interfaces modernes (Snowflake, Databrick, AirByte, Superset) mais il existe toujours des outils legacy (Talend, QLick, Microsoft SSIS)
- de plus en plus de pratiques issues du développement: des outils déclaratifs, versioning, tests unitaires, type hint, etc.
- implémentation de référence: [GitLab Data](#)