Rapport complet – Cohortes & Rétention (SQL + Python)

Mohammed Amine KHAMLICHI

27 novembre 2025

Résumé exécutif

Ce projet démontre de bout en bout comment un data analyste peut :

— générer des données e-commerce synthétiques et reproductibles ;
— calculer la rétention par cohorte en Python et en SQL avec parité garantie ;
— produire des visuels (EDA, heatmap) et des insights prêts à être communiqués ;
— industrialiser la démarche via tests automatisés, linting et CI.

Jeu de référence : 600 utilisateurs, 942 commandes, seed 42 (janvier 2023 – décembre 2024).

KPI	Valeur
Rétention moyenne M+1	23.6 % (meilleure cohorte juillet 2023 : 45.8 %)
Rétention moyenne M+3	7.4 %
Taille médiane de cohorte	24 clients
AOV (Average Order Value)	≈ 51 €

Au-delà des chiffres, le rapport met en lumière la valeur ajoutée méthodologique : gouvernance des jeux de données synthétiques, articulation Python/SQL, automatisation des tests et capacité à transformer un flux analytique en récit business compréhensible pour un décideur.

1 Contexte professionnel

Objectif : fournir un artefact GitHub convaincant pour un recrutement en alternance Data Analyst :

— chaîne analytique complète (données synthétiques → SQL → Python → visualisations) ;
— livrables exploitables par un décideur (insights, recommandations documentées) ;
— preuves de maturité technique (tests, CI, qualité logicielle, reproductibilité).

Public cible et promesse de valeur

Recruteurs & managers. Cadrage métier, génération de données, modélisation, restitution. Livrables réutilisables.

Pair-programming & mentorat. Conventions de style, tests automatisés, structure modulaire et commentaires ciblés facilitent la revue et l'onboarding.

Enjeux métiers

— **Mesure fiable de la rétention** : comparer des cohortes dans le temps et identifier les campagnes performantes.
— **Capacité à simuler** : données synthétiques sans contrainte PII.
— **Capitalisation documentaire** : README, case study et présent rapport comme kit prêt à l'emploi.

2 Présentation générale du dépôt

— `src/` : scripts Python (génération, rétention, CLI).
— `sql/` : schéma et requête analytiques.
— `notebooks/` : EDA et heatmap (auto-import du module `src`).
— `tests/` : Pytest (unit + CLI + DuckDB).
— `docs/` : documentation technique, case study, rapport.
— `.github/workflows/` : CI flake8 + pytest.

Conventions :

— **Nommer par fonctionnalité** (génération, analytics).
— **Séparer code et artefacts** : `data/`, `outputs/` ignorés par Git, régénérés via Makefile.
— **Documenter au plus près** : README ciblés (ex : `data/README.md`).

3 Objectifs détaillés

3.1 Objectifs analytiques

1. Simuler un dataset cohérent pour tester des KPI de fidélisation sans contrainte RGPD.
2. Produire une matrice de rétention par cohorte (table, heatmap, insights).
3. Assurer la parité SQL/Python pour un déploiement warehouse.

Résultats attendus : reproductibilité (hash stable), écart $< 0.1\%$ SQL vs pandas, insights générés automatiquement.

3.2 Objectifs techniques

1. Versionner la pipeline avec instructions d'installation claires.
2. Inclure tests, flake8, CI GitHub Actions.
3. Offrir des notebooks reproductibles.

Critères de succès : `make test` OK sur runner propre ; notebooks rejouables après `pip install -r requirements.txt` ; scénarios métier documentés.

4 Jeu de données synthétique

4.1 Modélisation des utilisateurs

— `signup_date` uniforme entre janvier 2023 et décembre 2024.
— RNG : `numpy.random.default_rng(seed)` pour la reproductibilité.
— 600 lignes triées chronologiquement.

4.2 Modélisation des commandes

— Premier achat : offset aléatoire 0–19 jours après `signup_date`.
— 1 à 5 achats par utilisateur ; probabilité de churn à chaque étape ($p = 0{,}5$).
— Intervalles 15–60 jours ; montants $\sim N(50, 20)$ tronqués à [5 ;300], arrondis à 2 décimales.

4.3 Structure finale

Fichier	Colonnes	Détails
`users.csv`	`user_id`, `signup_date`	600 lignes, triées par date
`orders.csv`	`order_id`, `user_id`, `order_date`, `amount`	942 lignes, jointure simple
`outputs/retention.csv`	`cohort_month`, `cohort_size`, 0..7	Matrice % actifs

Contrôles automatiques : doublons `order_id`, dates valides ; vérifications exploratoires dans les notebooks.

5 Architecture logicielle

5.1 Génération (Python)

`src/generate_data.py` :

— `GenerationConfig` : `n_users`, `seed`, `start`, `end`.
— `generate_users`, `generate_orders`, CLI (`-n-users`, `-seed`, `-users-output`, `-orders-output`).
— Logging du nombre de lignes écrites et création des dossiers.

5.2 Calcul de rétention

`src/retention.py` :

— Validation stricte (colonnes obligatoires, dates convertibles, dataset non vide).
— `cohort_month`, `order_month`, `month_index` $= 12 \times$ année + mois.
— Pivot des actifs distincts, `cohort_size` en M0, pourcentages arrondis à 0.1.
— Insights : taille médiane, moyenne M+1 + meilleure cohorte, médiane horizon max, décroissance.
— CLI : `-input`, `-output`, `-insights` ; export CSV + impressions.

Observabilité : `run_cli` renvoie `RetentionSummary` pour instrumentation (Airflow, Dagster).

5.3 Notebooks

— `01_eda.ipynb` : volumétrie mensuelle.

— `02_retention.ipynb` : table et heatmap (M+0..M+6), ajout automatique du repo au `sys.path`.

— Captures réutilisables dans README/portfolio.

5.4 SQL

`sql/queries.sql` (CTE `first_purchase`, `cohort_index`, `cohort_size`, `retention`) ; `retention_pct` pour parité avec pandas. Compatible DuckDB/PostgreSQL.

6 Méthodologie analytique

6.1 Chaînage

1. Ingestion : lecture des CSV, typage dates, intégrité clés.
2. Feature engineering : mois de cohorte, mois de commande, index relatif.
3. Agrégation : actifs distincts par (cohorte, index), pivot.
4. Normalisation : division par `cohort_size`.
5. Restitution : CSV, insights, visualisations.

6.2 Garde-fous

— Erreur bloquante si dataset vide ou colonnes manquantes.

— Dates converties avec `errors='coerce'` pour détecter les anomalies.

— Cohorte sans M+0 ignorée pour éviter les divisions par zéro.

7 Parité SQL / Python

7.1 Stratégie

1. Exécuter `build_retention` côté pandas.
2. Charger la même table commandes dans DuckDB (`con.register`).
3. Lancer `sql/queries.sql` et récupérer `cohort_month`, `month_index`, `retention_pct`.
4. Comparer chaque cellule via `pytest.approx` (tolérance 1e-6).

7.2 Bénéfices

— Transférabilité warehouse (Snowflake, BigQuery, DuckDB).

— Confiance accrue pour reviewers SQL/Python.

— Préparation aux entretiens : argument de parité.

8 Workflow analytique

8.1 Vue d'ensemble

a) Génération : `python src/generate_data.py`.

b) Analyse Python : `python src/retention.py -insights 5`.

c) Visualisation : Jupyter Lab + notebooks.

d) Analyse SQL : `.read sql/queries.sql` dans DuckDB/Postgres.

e) Documentation : mise à jour `docs/case_study.md`, `README.md`.

f) Qualité : `python -m pytest -q`, flake8, CI.

8.2 Commandes principales

```
python src/generate_data.py --n-users 600 --seed 42
python src/retention.py --input data/orders.csv --output outputs/
   retention.csv --insights 5
python -m pytest -q
python -m jupyter lab
```

Makefile (option GNU) :

```
make install
make data
make retention
make test
make notebook
```

Compatible Windows (PowerShell + .venv), macOS, Linux.

9 Analyse des résultats

M+0..M+6 : M+0 = 100 %, M+1 ≈ 20−30 %, au-delà de M+4 la plupart des cohortes tombent à 0–5 %.

Points d'attention : pic M+1 à 45 % sur été 2023 ; T1 2024 autour de 15 %.

Heatmap : couleurs chaudes = forte activité, froides = faible présence ; filtrable (M+0..M+6).

Insights CLI (exemple) :

```
Key insights:
- Median cohort size: 24 customers.
- Average M+1 retention: 23.6% (best cohort 2023-07 at 45.8%).
- Median retention at M+6: 0.0% (max/min 4.8% / 0.0%).
- Retention decays from 23.6% (M+1) to 0.2% (M+6).
```

10 Recommandations marketing

1. Nurturing D+7 / D+30 (email/SMS/cross-sell) pour remonter M+1 vers 30 %.
2. Répliquer la cohorte forte juillet 2023 (canal/offre/UX) sur les campagnes futures.
3. Programme VIP (P90) : avantages exclusifs pour garder > 10 % actifs à M+3.
4. Réactivation M+4/M+5 avec offres de retour si décroissance rapide.

11 Dimensions techniques

— Linting flake8 (`max-line-length=100`, `extend-ignore=E203,W503`).
— Pytest : multi-mois, dataset vide, round-trip CLI, parité SQL/Python.
— CI GitHub Actions : flake8 + pytest sur Python 3.11.
— `.gitignore` : `.venv`, `data/*.csv`, `outputs`, `.pytest_cache`.

Packaging : `requirements.txt` (pandas 2.2, numpy 2.1, etc.). Roadmap : `pyproject.toml`, hooks `pre-commit` (bandit, detect-secrets).

12 Documentation et communication

— **README** : aperçu, architecture, quickstart, pipeline, qualité, roadmap.
— **docs/documentation.md** : dictionnaire de données, méthode, QA.
— **docs/case_study.md** : synthèse business pour portfolio.
— **docs/rapport.tex** : version détaillée.

Séquence entretien : URL GitHub → heatmap → case study → rapport PDF.

13 Plan de test

Test	Description
`test_retention_handles_multi_month_churn`	Cohorte avec réactivation à M+2.
`test_build_retention_empty_input`	Erreur explicite sur dataset vide.
`test_cli_round_trip`	Exécute `retention.py` en sous-processus, contrôle `cohort_size` + logs.
`test_sql_query_retention`	Compare DuckDB (`sql/queries.sql`) à pandas.

Tests manuels : varier `-n-users`, seed, OS ; futurs : nbval/papermill, benchmarks 10k+ utilisateurs.

14 Perspectives DataOps et scalabilité

— Orchestration (Airflow/Dagster) pour enchaîner génération, calcul, SQL, export.
— Observabilité : Great Expectations, `dbt tests`, jobs cron GitHub Actions.
— Scalabilité : DuckDB pour plusieurs millions de lignes ; conteneurisation possible.

15 Roadmap d'amélioration

1. Intégrer un orchestrateur (Airflow, Dagster).
2. Publier un dashboard Streamlit/Power BI connecté à `outputs/retention.csv`.
3. Ajouter des tests de performance (datasets x10).
4. Connecter Great Expectations pour surveiller les distributions.
5. Fournir un package Python installable (`pip install cohort-retention`).

16 Synthèse SWOT

Forces

— Stack analytique complète.
— Documentation riche (notebooks, README, case study, rapport).
— CI + tests.

Faiblesses

— Données purement synthétiques.
— Pipeline non orchestrée (exécution manuelle).

Opportunités

— Intégration warehouse / BI.
— Packaging open-source, dashboard public.

Menaces

— Besoin de cas métiers variés.
— Concurrence de projets similaires.

Lecture synthétique. Force : cohérence bout-en-bout et documentation. Frein : données synthétiques ; la roadmap (warehouse, dashboard) y répond.

Annexe A : structure du dépôt

```
sql_cohort_retention_v2/
|-- src/
|-- sql/
|-- notebooks/
|-- docs/
|-- tests/
|-- data/     (genere)
|-- outputs/  (genere)
|-- README.md
|-- requirements.txt
|-- Makefile
‘-- .github/workflows/ci.yml
```

Annexe B : jeu d’essai notebooks

```
from pathlib import Path
import sys
import pandas as pd
from src.retention import build_retention

ROOT = Path('..').resolve()
if str(ROOT) not in sys.path:
    sys.path.insert(0, str(ROOT))

orders = pd.read_csv(ROOT / 'data' / 'orders.csv', parse_dates=['
   order_date'])
ret = build_retention(orders)
ret
```

Contact : LinkedIn – Mohammed Amine KHAMLICHI.