

```
=====
```

README - Générateur de données et import Snowflake

```
=====
```

1. DESCRIPTION

```
-----
```

Ce script Python génère des jeux de données factices (clients, inventaire, commandes) puis les importe automatiquement dans Snowflake.

Il est conçu pour :

- Créer des DataFrames simulant un environnement e-commerce
- Normaliser les colonnes (MAJUSCULES) pour compatibilité Snowflake
- Créer les tables nécessaires dans Snowflake
- Charger les DataFrames via la méthode optimisée `write_pandas`

Tables générées :

- CUSTOMERS
- INVENTORY
- ORDERS

2. PRÉREQUIS

```
-----
```

- Python 3.9+
- Packages Python :
 - pandas
 - faker
 - snowflake-connector-python
 - snowflake-connector-python[pandas]

Installer les dépendances :

```
pip install pandas faker snowflake-connector-python  
"snowflake-connector-python[pandas]"
```

3. CONFIGURATION

```
-----
```

Paramètres Snowflake à modifier dans le code principal :

```
user='Pollom'  
password='****'  
account='tjcpvnb-gw55905'  
warehouse='mon_entrepot'  
database='ma_base'  
schema='mon_schema'
```

⚠ Recommandation sécurité :

- Utiliser des variables d'environnement pour le mot de passe au lieu de l'écrire en dur.

4. UTILISATION

Exécuter simplement le script :

```
python generate_and_load.py
```

Étapes réalisées automatiquement :

1. Génération des DataFrames (CUSTOMERS, INVENTORY, ORDERS)
2. Connexion à Snowflake
3. Création des tables (ou remplacement si elles existent déjà)
4. Chargement des données dans Snowflake
5. Vérification avec SELECT COUNT(*)

5. VÉRIFICATION

Depuis Snowsight ou SnowSQL, exécuter :

```
USE WAREHOUSE mon_entrepot;  
USE DATABASE ma_base;  
USE SCHEMA mon_schema;
```

```
SELECT COUNT(*) FROM CUSTOMERS;  
SELECT COUNT(*) FROM INVENTORY;  
SELECT COUNT(*) FROM ORDERS;
```

```
SELECT * FROM CUSTOMERS LIMIT 10;  
SELECT * FROM INVENTORY LIMIT 10;  
SELECT * FROM ORDERS ORDER BY SOLD_AT DESC LIMIT 10;
```

6. NOTES DE SÉCURITÉ

-
- Ne pas laisser les identifiants en clair dans le code (utiliser variables d'environnement).
 - Vérifier que votre rôle Snowflake a les droits sur le warehouse/base/schema.
 - Les tables sont recréées à chaque exécution (CREATE OR REPLACE TABLE).

7. EXTENSIONS POSSIBLES

-
- Passer en mode "append" (overwrite=False dans write_pandas).
 - Ajouter un volume de données plus important.
 - Utiliser TIMESTAMP_TZ si la conservation du fuseau horaire est nécessaire.
 - Déployer dans un job automatisé (Airflow, Prefect, cron).

=====

"""

=====

Script : generate_and_load.py

Auteur : Paul

Description :

Génère des données factices (clients, inventaire, commandes)
et les importe dans Snowflake.

Fonctionnalités principales :

- Génération de DataFrames via Faker et distributions aléatoires
- Normalisation des colonnes en MAJUSCULE pour compatibilité Snowflake
- Création des tables Snowflake avec un schéma explicite
- Chargement des données via write_pandas

=====

"""

```
import datetime
```

```
import random
```

```
import pandas as pd
```

```
from faker import Faker
```

```
# --- Snowflake ---
```

```
import snowflake.connector
```

```
from snowflake.connector.pandas_tools import write_pandas
```

```
def gaussian_clamped(rng: random.Random, mu: float, sigma: float, a: float, b: float) -> float:
```

```
    """
```

Tire une valeur aléatoire suivant une loi normale tronquée entre a et b.

Args:

rng (random.Random): générateur aléatoire

mu (float): moyenne

sigma (float): écart-type

a (float): minimum

b (float): maximum

Returns:

float: valeur comprise entre a et b

```
    """
```

```
    val = rng.gauss(mu, sigma)
```

```
    return max(a, min(b, val))
```

```
def generate_orders(
```

```
    orders: int,
```

```
    seed: int = 42,
```

```
start: str = "2024-01-01",
end: str = "2025-09-01",
inventory: pd.DataFrame = None,
customers: pd.DataFrame = None,
) -> pd.DataFrame:
    """
```

Génère un DataFrame de commandes factices.

Args:

orders (int): nombre de commandes
seed (int): graine aléatoire
start (str): date de début (YYYY-MM-DD)
end (str): date de fin (YYYY-MM-DD)
inventory (pd.DataFrame): produits disponibles
customers (pd.DataFrame): clients disponibles

Returns:

pd.DataFrame: tableau des commandes
"""

```
rng = random.Random(seed)
```

```
# Plage temporelle pour SOLD_AT
```

```
start_dt = datetime.datetime.fromisoformat(start).replace(tzinfo=datetime.timezone.utc)
```

```
end_dt = datetime.datetime.fromisoformat(end).replace(tzinfo=datetime.timezone.utc)
```

```
delta_seconds = int((end_dt - start_dt).total_seconds())
```

```
# Produits et prix
```

```
product_ids = list(range(1000, 1250))
```

```
prices = {}
```

```
if inventory is not None and not inventory.empty:
```

```
    product_ids = inventory["PRODUCT_ID"].tolist()
```

```
    prices = dict(zip(inventory["PRODUCT_ID"], inventory["UNIT_PRICE"], strict=True))
```

```
# Clients
```

```
customer_ids = list(range(1, 1001))
```

```
if customers is not None and not customers.empty:
```

```
    customer_ids = customers["CUSTOMER_ID"].tolist()
```

```
rows = []
```

```
for i in range(1, orders + 1):
```

```
    pid = rng.choice(product_ids)
```

```
    qty = rng.choices([1, 2, 3, 4, 5], weights=[0.6, 0.2, 0.12, 0.06, 0.02])[0]
```

```
    sold_at = start_dt + datetime.timedelta(seconds=rng.randint(0, delta_seconds))
```

```
    cid = rng.choice(customer_ids)
```

```
    unit_price = prices.get(pid, round(rng.uniform(5, 500), 2))
```

```
    rows.append(
```

```
        {
```

```
            "ID": i,
```

```

        "PRODUCT_ID": pid,
        "CUSTOMER_ID": cid,
        "QUANTITY": qty,
        "UNIT_PRICE": unit_price,
        "SOLD_AT": sold_at,
    }
)

```

```

df = pd.DataFrame(rows)
df["SOLD_AT"] = pd.to_datetime(df["SOLD_AT"], utc=True)
return df

```

Catégories et adjectifs pour générer l'inventaire

```

CATEGORIES = [
    ("Apparel", ["T-Shirt", "Hoodie", "Jeans", "Sneakers", "Jacket"]),
    ("Electronics", ["Headphones", "Smartphone", "Tablet", "Smartwatch", "Charger"]),
    ("Home & Kitchen", ["Mug", "Kettle", "Blender", "Vacuum", "Toaster"]),
    ("Beauty", ["Shampoo", "Conditioner", "Face Cream", "Perfume", "Lipstick"]),
    ("Grocery", ["Coffee Beans", "Olive Oil", "Pasta", "Granola", "Tea"]),
]
ADJECTIVES = ["Classic", "Premium", "Eco", "Urban", "Sport", "Comfort", "Pro", "Lite",
"Max", "Essential"]

```

def generate_inventory_data(products: int, seed: int = 42) -> pd.DataFrame:

"""

Génère un DataFrame d'inventaire de produits factices.

Args:

products (int): nombre de produits
seed (int): graine aléatoire

Returns:

pd.DataFrame: tableau des produits

"""

```

rng = random.Random(seed)
rows = []
product_ids = list(range(1000, 1000 + products))
for pid in product_ids:
    cat, names = rng.choice(CATEGORIES)
    base = rng.choice(names)
    adj = rng.choice(ADJECTIVES)
    product_name = f'{adj} {base}'
    base_price = {"Apparel": 39, "Electronics": 299, "Home & Kitchen": 79, "Beauty": 25,
"Grocery": 12}[cat]
    price = round(gaussian_clamped(rng, base_price, base_price * 0.25, base_price * 0.4,
base_price * 1.8), 2)

```

```

stock_qty = int(gaussian_clamped(rng, 80, 60, 0, 400))
rows.append(
    {
        "PRODUCT_ID": pid,
        "PRODUCT_NAME": product_name,
        "CATEGORY": cat,
        "UNIT_PRICE": price,
        "STOCK_QUANTITY": stock_qty,
    }
)
return pd.DataFrame(rows)

```

def generate_customers(customers: int, seed: int = 42) -> pd.DataFrame:

"""

Génère un DataFrame de clients factices.

Args:

customers (int): nombre de clients

seed (int): graine aléatoire

Returns:

pd.DataFrame: tableau des clients

"""

```
rng = random.Random(seed)
```

```
fake = Faker()
```

```
Faker.seed(seed)
```

```
rows = []
```

```
channels = [("online", 0.65), ("store", 0.35)]
```

```
for cid in range(1, customers + 1):
```

```
    name = fake.name()
```

```
    email = fake.email()
```

```
    city = fake.city()
```

```
    channel = rng.choices([c for c, _ in channels], weights=[w for _, w in channels])[0]
```

```
    rows.append(
```

```
        {
```

```
            "CUSTOMER_ID": cid,
```

```
            "NAME": name,
```

```
            "EMAIL": email,
```

```
            "CITY": city,
```

```
            "CHANNEL": channel,
```

```
        }
    )

```

```
return pd.DataFrame(rows)
```

def create_tables_if_needed(conn):

```
"""
```

Crée (ou remplace) les tables CUSTOMERS, INVENTORY, ORDERS dans Snowflake.

Args:

conn: connexion active Snowflake

```
"""
```

```
ddl = [
```

```
    """
```

```
    CREATE OR REPLACE TABLE CUSTOMERS (  
        CUSTOMER_ID INTEGER,  
        NAME STRING,  
        EMAIL STRING,  
        CITY STRING,  
        CHANNEL STRING
```

```
    );
```

```
    """,
```

```
    """
```

```
    CREATE OR REPLACE TABLE INVENTORY (  
        PRODUCT_ID INTEGER,  
        PRODUCT_NAME STRING,  
        CATEGORY STRING,  
        UNIT_PRICE NUMBER(10,2),  
        STOCK_QUANTITY INTEGER
```

```
    );
```

```
    """,
```

```
    """
```

```
    CREATE OR REPLACE TABLE ORDERS (  
        ID INTEGER,  
        PRODUCT_ID INTEGER,  
        CUSTOMER_ID INTEGER,  
        QUANTITY INTEGER,  
        UNIT_PRICE NUMBER(10,2),  
        SOLD_AT TIMESTAMP_NTZ
```

```
    );
```

```
    """,
```

```
]
```

```
with conn.cursor() as cur:
```

```
    for stmt in ddl:
```

```
        cur.execute(stmt)
```

```
def load_df(conn, df: pd.DataFrame, table: str):
```

```
    """
```

Charge un DataFrame dans une table Snowflake.

Args:

conn: connexion active Snowflake

df (pd.DataFrame): DataFrame à charger


```

    table (str): nom de la table cible
    """
    if table.upper() == "ORDERS" and "SOLD_AT" in df.columns:
        df = df.copy()
        df["SOLD_AT"] = pd.to_datetime(df["SOLD_AT"], utc=True).dt.strftime('%Y-%m-%d
%H:%M:%S')

    success, nchunks, nrows, _ = write_pandas(
        conn,
        df,
        table_name=table,
        overwrite=True,
        auto_create_table=False,
        quote_identifiers=True,
    )
    print(f"✅ Table {table}: {nrows} lignes chargées ({'OK' if success else 'KO'})")

```

```

def show_counts(conn):
    """

```

Affiche le nombre de lignes dans chaque table cible.

Args:

conn: connexion active Snowflake

```

    """

```

```

    with conn.cursor() as cur:

```

```

        for t in ["CUSTOMERS", "INVENTORY", "ORDERS"]:

```

```

            cur.execute(f"SELECT COUNT(*) FROM {t}")

```

```

            print(f"📦 {t} = {cur.fetchone()[0]} lignes")

```

```

if __name__ == "__main__":

```

```

    # 1) Génération des données locales

```

```

    customers_df = generate_customers(customers=100, seed=42)

```

```

    inventory_df = generate_inventory_data(products=100, seed=42)

```

```

    orders_df = generate_orders(orders=100, seed=42, inventory=inventory_df,
customers=customers_df)

```

```

    # 2) Connexion Snowflake

```

```

    conn = snowflake.connector.connect(

```

```

        user='Pollom',

```

```

        password='dCZFA4nmEbRqiqz',

```

```

        account='tjkipvb-gw55905',

```

```

        warehouse='mon_entrepot',

```

```

        database='ma_base',

```

```

        schema='mon_schema'

```

```

    )


```

try:

```
# 3) Création / remplacement des tables  
create_tables_if_needed(conn)
```

```
# 4) Chargement des DataFrames  
load_df(conn, customers_df, "CUSTOMERS")  
load_df(conn, inventory_df, "INVENTORY")  
load_df(conn, orders_df, "ORDERS")
```

```
# 5) Vérification rapide  
show_counts(conn)
```

```
# Info version Snowflake  
with conn.cursor() as cur:  
    cur.execute("SELECT current_version()")  
    print(f" Snowflake version: {cur.fetchone()[0]}")
```

finally:

```
conn.close()
```