Génération du Fichier sales_data.csv

On va générer un jeu de données aléatoire contenant 1 million de lignes avec les colonnes suivantes :

transaction_id : Identifiant unique de la transaction.

customer_id : Identifiant unique du client.

product_id : Identifiant unique du produit.

quantity : Quantité de produits achetés.

price : Prix unitaire du produit en DH.

transaction_date : Date de la transaction.

region : Région où la transaction a eu lieu.

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import time
import sqlite3

# Paramètres pour la génération de données
num_rows = 1_000_000  # 1 million de lignes
regions = ['North America', 'Europe', 'Asia', 'South America', 'Africa', 'Australia']
start_date = datetime(2020, 1, 1)  # Date de début pour les transactions
end_date = datetime(2023, 12, 31)  # Date de fin pour les transactions

# Génération des données
np.random.seed(42)  # Pour la reproductibilité

# transaction_id : Identifiant unique
transaction_ids = np.arange(1, num_rows + 1)

# customer_id : Identifiant client aléatoire entre 1 et 100 000
customer_ids = np.random.randint(1, 100_001, num_rows)

# product_id : Identifiant produit aléatoire entre 1 et 10 000
product_ids = np.random.randint(1, 10_001, num_rows)

# quantity : Quantité aléatoire entre 1 et 20 (au lieu de 1 à 10)
quantities = np.random.randint(1, 21, num_rows)

# price : Prix aléatoire entre 10 et 500 (avec 2 décimales)
prices = np.round(np.random.uniform(10, 500, num_rows), 2)

# transaction_date : Date aléatoire entre start_date et end_date
date_range = (end_date - start_date).days
transaction_dates = [start_date + timedelta(days=np.random.randint(0, date_range)) for _ in range(num_rows)]

# region : Région aléatoire parmi la liste définie
regions_data = np.random.choice(regions, num_rows)

# Création du DataFrame
df = pd.DataFrame({
    'transaction_id': transaction_ids,
    'customer_id': customer_ids,
    'product_id': product_ids,
    'quantity': quantities,
    'price': prices,
    'transaction_date': transaction_dates,
    'region': regions_data
})

# Sauvegarde du DataFrame dans un fichier CSV
df.to_csv('sales_data.csv', index=False)
print("Fichier 'sales_data.csv' sauvegardé avec succès.")
```

```
Fichier 'sales_data.csv' sauvegardé avec succès.
```

1. Échantillonnage et Sous-ensemble de Données

Tâche :

- Charger un échantillon aléatoire de 1 % des lignes du fichier sales_data.csv

- Sélectionner uniquement les colonnes customer_id, product_id, quantity, et price
- Spécifier les types de données appropriés pour réduire la consommation de mémoire

```
df_sample = pd.read_csv('sales_data.csv', usecols=['customer_id', 'product_id', 'quantity', 'price'],
                        dtype={'customer_id': 'int32', 'product_id': 'int32', 'quantity': 'int8', 'price': 'float32'},
                        skiprows=lambda i: i > 0 and np.random.rand() > 0.01)
```

```
df_sample.head()
```

	customer_id	product_id	quantity	price
0	22663	9540	13	332.709991
1	18048	4180	1	442.820007
2	92788	3149	6	459.769989
3	38624	2138	13	115.480003
4	93385	6532	5	315.529999

Next steps: Generate code with df_sample | View recommended plots | New interactive sheet

2. Conversion en Formats de Fichiers Efficaces

Tâche :

- Convertir l'échantillon de données en formats Feather et Parquet
- Comparer la taille des fichiers et mesurer le temps de chargement.

```
# Conversion en feather
start_time = time.time()
df_sample.to_feather('sales_sample.feather')
feather_time = time.time() - start_time
```

```
# Conversion en Parquet
start_time = time.time()
df_sample.to_parquet('sales_sample.parquet')
parquet_time = time.time() - start_time
```

```
# Comparaison des temps de sauvegarde
print(f"Temps de sauvegarde Feather : {feather_time} secondes")
print(f"Temps de sauvegarde Parquet : {parquet_time} secondes")
```

```
Temps de sauvegarde Feather : 0.020730972290039062 secondes
Temps de sauvegarde Parquet : 0.014330625534057617 secondes
```

Feather est un format rapide adapté aux petites et moyennes données, par contre que parquet est optimisé pour les grandes données grâce à une compression efficace

3. Utilisation de HDF5

Tâche :

- Créer un fichier HDF5 (sales_data.h5) et stocker l'échantillon de données dans une table appelée sales_sample
- Ajouter une deuxième table contenant les transactions dont le prix est supérieur à 100 DH
- Lire les données de la table sales_sample et afficher les 5 premières lignes

```
with pd.HDFStore('sales_data.h5') as store:
    store.put('sales_sample', df_sample)
    store.put('high_price_transactions', df_sample[df_sample['price'] > 100])
```

```
with pd.HDFStore('sales_data.h5') as store:
    df_sample_hdf5 = store['sales_sample']
```

```
    df_sample_hdf5.head()
```

	customer_id	product_id	quantity	price
0	22663	9540	13	332.709991
1	18048	4180	1	442.820007
2	92788	3149	6	459.769989
3	38624	2138	13	115.480003
4	93385	6532	5	315.529999

Next steps: Generate code with df_sample_hdf5 | View recommended plots | New interactive sheet

4. Lecture par Morceaux

Tâche :

- Lire le fichier sales_data.csv par morceaux de 100 000 lignes
- Pour chaque morceau, filtrer les transactions ayant une quantité supérieure à 10
- Combiner les résultats filtrés en un seul DataFrame et calculer le total des ventes (quantity * price)

```
chunksize = 100000
results = []
```

```
for chunk in pd.read_csv('sales_data.csv', chunksize=chunksize):
    filtered_chunk = chunk[chunk['quantity'] > 10]  # Filtrer les transactions ayant une quantité > 10
    results.append(filtered_chunk)
```

```
df_filtered = pd.concat(results)
```

```
total_sales = (df_filtered['quantity'] * df_filtered['price']).sum()
print(f"Total des ventes pour les transactions avec quantité > 10 : {total_sales:.2f} DH")
```

```
Total des ventes pour les transactions avec quantité > 10 : 1973843584.72 DH
```

5. Chargement dans une Base de Données

Tâche :

- Créer une base de données SQLite (sales.db) et charger l'intégralité du fichier sales_data.csv dans une table appelée sales
- Exécuter une requête SQL pour extraire les transactions ayant eu lieu dans la région "Europe" et dont le prix est supérieur à 50 DH
- Calculer le total des ventes pour ces transactions

```
# Créer une base de données SQLite
conn = sqlite3.connect('sales.db')

# Charger les données dans la table 'sales'
df = pd.read_csv('sales_data.csv')
df.to_sql('sales', conn, if_exists='replace', index=False)
```

```
1000000
```

```
query = """
SELECT SUM(quantity * price) AS total_sales
FROM sales
WHERE region = 'Europe' AND price > 50
"""
```

```
result = pd.read_sql_query(query, conn)
print(f"Total des ventes en Europe avec prix > 50 DH : {result['total_sales'].values[0]} DH")
conn.close()
```

```
Total des ventes en Europe avec prix > 50 DH : 440375271.4700049 DH
```

```
Start coding or generate with AI.
```