📄 Project Documentation: Synthetic Healthcare Data Generation and SQL Database Preparation

1. Dataset Generation Script

File: `fake_records_generation.py`
Purpose: Automatically generate synthetic data in CSV format.

How it works:

- **Faker Library** was used to create realistic names, dates, and other details.
- Five related tables were generated:
    1. **Hospital Table**
        - Columns: `hospital_id`, `hospital_name`
        - Contains 5 hospitals.
    2. **Patient Table**
        - Columns: `patient_id`, `hospital_id`, `patient_name`, `dob`, `admission_datetime`, `discharge_datetime`
        - Contains 100,000 patients.
    3. **Diagnosis Table**
        - Each patient has between 2–4 diagnoses.
        - Columns: `diagnosis_id`, `patient_id`, `diagnosis_name`
    4. **Treatment Table**
        - Each patient has 5–8 medicines.

        - Columns: `treatment_id`, `patient_id`, `medicine_name`, `dose_time`, `duration`

5. **Billing Table**
        - One billing record per patient.
        - Columns: `bill_id`, `patient_id`, `bill_amount`, `payment_mode`

Efficiency Rationale:

- Using Python and Faker allowed **fast generation of 100,000+ records** with minimal manual effort.
- All data was exported in **CSV format**, which is portable and readable.

---

2. Database Conversion Script

File: `database conversion to sql.py`
Purpose: Convert the generated CSV files into an SQLite relational database (`mydatabase.db`).

How it works:

- **Pandas** reads each CSV file into a DataFrame.
- **SQLAlchemy** connects to SQLite and creates the database file.
- `.to_sql()` writes each DataFrame into a table inside `mydatabase.db`.

Efficiency Rationale:

- SQLite was selected because it:
    - Requires no server setup (single `.db` file).
    - Can handle large datasets (100k+ records) easily.
    - Works with any SQL tools (VS Code extensions, Python, or DB Browser).
- This method enabled **fast ingestion** of CSVs into structured tables, making them immediately queryable with SQL.

✅ 3. Database File

File: `mydatabase.db`
Purpose: Relational database containing all tables:

- `Hospital`
- `Patient`
- `Diagnosis`
- `Treatment`
- `Billing`

How it works:

- Tables are fully indexed by default on primary keys.
- All joins can be performed efficiently (e.g., linking Patients to Treatments).

Why This Approach Was the Most Efficient

1. Scalability

- Generating 100,000+ records manually or in Excel would be impractical.
- Faker automated the process in seconds.

2. Reproducibility

- The scripts can be re-run any time to regenerate fresh synthetic data.

3. Portability

- CSV files can be used in any platform (Excel, Pandas, SQL tools).
- SQLite `.db` file can be shared and opened anywhere.

4. Query Efficiency

- SQLite supports indexing, transactions, and optimized query plans.
- We can execute complex SQL queries (e.g., joins, aggregations) quickly.

5. Minimal Setup Overhead

- No need to install or configure MySQL or PostgreSQL servers.
- No admin credentials or server maintenance.
- Everything works locally in a self-contained database.

6. Compatibility

- Works seamlessly with:
  - VS Code extensions
  - Python (`sqlite3` or `SQLAlchemy`)
  - External GUI tools (DB Browser for SQLite)

How To Use the Database in VS Code

Step 1: Install a VS Code SQLite extension (e.g., **SQLite Viewer**).
Step 2: Open `mydatabase.db` in the extension.
Step 3: Use the query panel to run SQL queries.
Step 4: Explore results, export data, and analyze.

Summary of Benefits

Full compatibility with VS Code and Python

Easily reproducible and shareable

This method balances **efficiency**, **flexibility**, and **simplicity**, making it ideal for both development and demonstration purposes.