Mini Financial Data Lake: Local ETL & Analytics Pipeline

Formet Iliane

2025

1 Project Overview

This project is about building a small-scale "data lake". Its main goal is to show how you can automatically pull stock price data and key financial metrics for US companies, store them in an organized way, and explore them in an interactive dashboard. Compared to manual copying and pasting from websites, this pipeline makes the process repeatable, reliable, and fast.

2 Learnings from the Project

The learnings of this project are:

- Master the basics of **data extraction**: using APIs (Yahoo Finance and SEC EDGAR) to fetch real financial data.
- Understand how to **transform and clean** data using a lightweight SQL engine (DuckDB) and store results in columnar Parquet files.
- Learn the concepts of a **data lake** with separate zones (raw, staged, analytics) to keep data reliable.
- Practice **caching** external data (fundamentals) to avoid hammering an API repeatedly.
- Build a user-friendly **dashboard** with Streamlit and Plotly to visualize prices, volumes, moving averages, and key ratios.
- Gain hands-on experience with **Python packaging** (virtual environments, requirements), and data-oriented file formats (Parquet).

3 Key Concepts

3.1 Finance Fundamentals

- **Stocks and Tickers:** A stock represents a share of a company. Each stock has a short code (ticker), for example `AAPL` for Apple.
- **OHLC Prices:**
  - *Open*: price at market open.
  - *High/Low*: highest and lowest price during the day.
  - *Close*: price at market close.
  - *Volume*: number of shares traded.

- **Returns:**
  - *Simple return*: (today's price / yesterday's price) minus 1.
  - *Log return*: natural log of today's price minus log of yesterday's price.
- **Valuation Ratios:**
  - *P/E ratio (Price/Earnings)*: share price divided by earnings per share.
  - *PEG ratio*: P/E divided by earnings growth rate.
  - *Price/Book, Price/Sales*: price relative to book value or revenue.
- **Profitability Metrics:**
  - *EPS (Earnings per Share)*: net profit per share.
  - *ROE, ROA*: return on equity/assets, measures how well the company uses its capital.
- **Financial Health Metrics:**
  - *Total Assets/Liabilities*: what the company owns vs owes.
  - *Debt/Equity*: debt divided by shareholders' equity.
  - *Current Ratio, Quick Ratio*: shortterm liquidity measures.
- **Cash Flow and Growth:**
  - *Operating Cash Flow, Free Cash Flow*: real cash generated by operations.
  - *Revenue/Earnings Growth*: yearoveryear increase in sales or profit.
- **Risk Metrics:**
  - *Volatility*: standard deviation of returns.
  - *Drawdown*: drop from peak to trough.
  - *Beta*: sensitivity of a stock to market movements.

3.2 Data Engineering Fundamentals

- **ETL vs ELT:**
  - *ETL*: Extract, Transform, then Load.
  - *ELT*: load raw data first, then transform inside a database.
- **Data Lake Zones:**

  ***Raw*:** data exactly as downloaded (no changes).
  ***Staged*:** cleaned and typed data, ready for analysis.
  ***Analytics*:** final fact and dimension tables for querying.
- **Parquet vs CSV:**
  - Parquet is a binary columnar format with schema and compression.
  - CSV is a simple text format, but slower and lacks type safety.
- **Dimensional Modeling:**
  - *Fact table*: holds measurements (e.g., prices).

  - *Dimension tables*: hold context (e.g., dates, tickers).

- **DuckDB:** a lightweight SQL database that can query Parquet files in place, without loading everything into memory.
- **Caching & Idempotence:** store fetched fundamentals with a timestamp so you don't re-download every run.
- **Python Packaging:** use a virtual environment and a `requirements.txt` file to freeze dependencies.
- **API Parsing:** understand JSON/XBRL structures when working with SEC EDGAR, and map ticker symbols to CIK identifiers.

4 Pipeline Architecture

All work happens "on demand" when clicking "Load Data" in the dashboard:

1. **Extraction:** if no raw data for the ticker, call `yfinance` to download historical prices and save to `data/raw`.
2. **Transformation:** use DuckDB to read the raw Parquet, filter by date, and write:
   - A *staged* Parquet file under `data/staged`.
   - Two *analytics* Parquet files: `fact_price` and `dim_date` under `data/analytics`.
3. **Fundamentals:** check if cached fundamentals are less than 24 hours old; if not, fetch from SEC EDGAR (JSON/XBRL) and yfinance, then cache under `data/fundamentals`.
4. **Dashboard Load:** the Streamlit app reads the staged price data and fundamentals, and renders charts and metrics on the spot.

5 Project Structure

```
DataPond/
 .streamlit/             # Streamlit theme configuration
 data/
    raw/                # Raw price data (Parquet)
    staged/             # Filtered price data (Parquet)
    analytics/          # fact_price, dim_date (Parquet)
    fundamentals/       # Financial fundamentals (Parquet)
    cache/              # Auxiliary caches (ticker→CIK mapping)
 scripts/
    extract.py          # Download prices + Symbol column
    transform.py        # Filter & write Parquet via DuckDB
    fundamentals.py     # Fetch & cache SEC & yfinance data
 dashboard/
    dashboard.py        # Streamlit + Plotly UI
 main.py                # Entry point: launches the dashboard
 clean_data.sh          # Cleans all data & caches
 requirements.txt       # Python dependencies
```

6 Technical Stack

- **Python 3.12**
- **Data Handling:** Pandas, PyArrow, Parquet
- **SQL Engine:** DuckDB
- **APIs:** yfinance, SEC EDGAR via `requests`
- **Dashboard:** Streamlit, Plotly
- **Packaging:** virtual environment, `requirements.txt`
- **Version Control & Cleaning:** Git, `clean_data.sh`

7 Step-by-Step Implementation

1. **Extract Historical Prices.** Created a script that uses `yfinance` to download daily OHLCV data for a given ticker. The result is saved unmodified as a Parquet file in `data/raw`, ensuring a reproducible record of data retrieved from the API.
2. **Stage and Filter Data.** Created a script that reads the raw Parquet file in DuckDB, applies a filter for the specified date range, and writes the filtered dataset to `data/staged`. Keeping raw and staged data separate allows the filter process to be rerun without re-downloading.
3. **Build Analytics Tables.** In the same script, DuckDB SQL is used to:
   - Create a *fact_price* table with columns `date`, `ticker`, `open`, `high`, `low`, `close`, and `volume`.
   - Create a *dim_date* table with one row per date and columns for `year`, `month`, and `day`.

   These tables are then written as Parquet files in `data/analytics`.
4. **Fetch Financial Fundamentals.** A script is developed to:
   - Map each ticker to its SEC CIK code, with the mapping cached locally.
   - Call the SEC EDGAR XBRL JSON API to retrieve revenue, net income, assets, and liabilities for the latest annual report.
   - Fall back to `yfinance.Ticker().info` when SEC data is unavailable.
   - Save the combined results with a timestamp to `data/fundamentals`.

   This caching mechanism prevents repeated API calls.
5. **Create the Dashboard.** In Streamlit:
   - A ticker selector and date range inputs are provided, with optional quick buttons for common intervals (1M, 3M, etc.).
   - On activation, extraction and transformation steps are executed if required, and the staged and fundamentals Parquets are loaded.
   - A Plotly candlestick chart is displayed, including 20-day and 50-day moving averages (when possible).
   - Key metrics (first/last price, percent change, high/low) are presented.
   - Tabs are included for fundamentals (valuation, profitability, financial health).
6. **Add a Clean-up Script.** A shell script (`clean_data.sh`) is provided to delete all data directories and caches, enabling a clean restart of the pipeline.

8 Possible Enhancements

- Integrate data quality checks (e.g., Great Expectations) to validate schemas and data ranges.
- Introduce Prefect for scheduled or more complex workflows.
- Containerize the application with Docker and deploy the dashboard to a server.
- Extend to intraday data via a streaming platform (Kafka, Faust).
- Incorporate forecasting models (ARIMA, LightGBM) for price predictions.
- Expand to non-US markets and handle currency conversion.