Stock Price Retriever - Project Documentation

Submitted by: Shruti
Position: Data Analyst Intern
Company: ITUS Capital
Date: October 2025

Project Overview

This project creates a dynamic connection between a SQLite database (containing stock price data) and Google Sheets. Users can simply type a stock ticker symbol and date in Google Sheets, and the closing price automatically appears - just like Excel formulas, but connected to a real database.

Why I Chose This Approach (Mac + Python + Google Sheets)

Reason: I am using a Mac computer, which made the traditional Excel + VBA approach challenging because:

1. **Mac Excel has limited VBA support** - Many VBA features don't work properly on Mac
2. **SQLite ODBC drivers are difficult to install on Mac** - Compatibility issues
3. **Python works perfectly on Mac** - It's already installed and fully supported
4. **Google Sheets is platform-independent** - Works the same on Mac, Windows, or any device

Benefits of this approach:

- Works on any computer (Mac, Windows, Linux)
- Can access from anywhere (Google Sheets is cloud-based)

How It Works

User Experience:

1. User opens Google Sheets
2. Types ticker symbol in cell **A1** (example: **ITC.NS**)
3. Types date in cell **A2** (example: **2010-01-04**)
4. Cell **A4** automatically shows the closing price (example: **₹53.54**)
5. When user changes A1 or A2, A4 updates automatically!

Behind the Scenes:

1. Python script continuously monitors Google Sheets
2. When it detects a change in A1 or A2, it:
    - Reads the ticker and date

    - Queries the SQLite database
    - Fetches the closing price
    - Updates cell A4 with the result
3. Everything happens automatically in the background!

Step-by-Step Setup Process

Step 1: Create Google Cloud Project

What: Google Cloud Console is where we create API credentials to allow our Python script to access Google Sheets.

How I did it:

1. Went to: https://console.cloud.google.com/
2. Clicked "New Project"
3. Named it: "Stocksqlite"
4. Clicked "Create"

Why: We need a project to organize our API access.

Step 2: Enable Google Sheets API

What: This gives our script permission to read and write to Google Sheets.

How I did it:

1. In Google Cloud Console, clicked on "APIs & Services"
2. Clicked "Enable APIs and Services"
3. Searched for "Google Sheets API"
4. Clicked on it and pressed "Enable"
5. Went back and also enabled "Google Drive API"

Why: Without enabling these APIs, our script cannot connect to Google Sheets.

Step 3: Create Service Account

What: A service account is like a robot user that our Python script uses to access Google Sheets. It gets its own email address.

How I did it:

1. In Google Cloud Console, went to "APIs & Services" → "Credentials"
2. Clicked "Create Credentials" → "Service Account"
3. Filled in:
    - **Service account name:** stock-price-retriever
    - **Service account ID:** auto-filled

4. Clicked "Create and Continue"
5. Skipped the optional steps (clicked "Continue" then "Done")

Step 4: Download Credentials File

What: This JSON file contains the secret keys that prove our script is allowed to access Google Sheets.

How I did it:

1. On the Credentials page, I saw my service account listed
2. Clicked on the service account name
3. Went to the "Keys" tab
4. Clicked "Add Key" → "Create New Key"
5. Selected "JSON" format
6. Clicked "Create"

Why: This file is like a password - our Python script needs it to prove it's authorized to access Google Sheets.

Step 5: Create and Share Google Sheet

What: Creating the actual spreadsheet where users will input ticker symbols and dates.

How I did it:

1. Went to: https://sheets.google.com
2. Clicked "Blank" to create a new spreadsheet
3. Named it: "Stocksqlite"
4. Copied the Sheet ID from the URL:
5. **Most Important:** Clicked the "Share" button
6. Pasted the service account email
7. Gave it "Editor" access
8. Clicked "Share"

Step 6: Set Up Python Environment on Mac

What: Installing the necessary Python libraries.

How I did it:

1. Opened Terminal on my Mac
2. Created a project folder:
3. Installed required Python packages:

bash

```
pip3 install gspread google-auth
```

4. Placed these files in the folder:

   - **prices.db (the SQLite database from the email)**
   - **stocksqlite-320c0219c9a7.json (credentials file)**
   - **stock_price_retriever.ipynb (my Python script)**

Why: Python needs these libraries to communicate with Google Sheets.

Step 7: Configure the Python Script

What: Updating the script with my specific Google Sheet ID and file paths.

How I did it:

In the Python script, I updated these lines:

python
```
# My Google Sheet ID (from Step 5)
SHEET_ID = "1X9_Navxt6YO3q7ODPqTL1CK-kI7ln6oyd_rEKPDB6XM"

# Path to database
DB_PATH = "prices.db"

# Path to credentials

CREDENTIALS_FILE = "stocksqlite-320c0219c9a7.json"
```

Step 8: Run the Script

What: Starting the automatic monitoring system.

How I did it:

1. In Terminal, navigated to my project folder:
2. Ran the script:

bash
```
  python3 stock_price_retriever.ipynb
```

3. Chose option "1" (Auto-Update Mode)
4. The script started monitoring my Google Sheet!

Why: This keeps the script running in the background, constantly checking for changes in A1 and A2.

Technical Details

Database Structure

The SQLite database **(prices.db)** contains a table called **prices** with:

Column	Type	Description
date	DATETIME	Trading date (format: YYYY-MM-DD HH:MM:SS)
ticker_symbol	TEXT	Stock ticker (example: ITC.NS, RELIANCE.NS)
close	REAL	Closing price for that day

SQL Query Used

sql
SELECT close
FROM prices
WHERE ticker_symbol = ?

AND date(date) = ?

This finds the closing price for a specific ticker on a specific date.

Google Sheets API Integration

The script uses:

- **gspread library:** Python library to interact with Google Sheets
- **google-auth library:** Handles authentication with Google
- **Service Account authentication:** Secure way to access Google Sheets without manual login

How Auto-Update Works

1. Script reads cells A1 and A2 every 2 seconds
2. Compares current values with last known values
3. If changed:
 - Queries SQLite database
 - Formats the result (adds ₹ symbol)
 - Updates cell A4
 - Applies color formatting:
  • Green + Bold = Exact match found
  • Blue = Nearest date used (if exact date not available)
  • Red = Error or no data found

4. Repeats forever until stopped with Ctrl+C

Example Usage

Example 1: ITC Stock Price

User actions:

- Opens Google Sheet
- Types in A1: **ITC.NS**
- Types in A2: **2010-01-04**

Result in A4: ₹53.54 (green, bold)

Terminal shows:

```
[14:23:45] Change detected! Updating...
🔍 Fetching price...
   Ticker: ITC.NS
   Date: 2010-01-04

✅ Closing Price: ₹53.54
```

Example 2: Changing Date

User actions:

- Changes A2 to: 2010-01-05

Result in A4: ₹54.07 (updated automatically, green, bold)

Terminal shows:

```
[14:24:12] Change detected! Updating...
🔍 Fetching price...
   Ticker: ITC.NS
   Date: 2010-01-05

✅ Closing Price: ₹54.07
```

Example 3: Date Not Available (Uses Nearest)

User actions:

- Types in A1: ITC.NS
- Types in A2: 2010-01-09 (weekend, market closed)

Result in A4: ₹54.14 (nearest: 2010-01-08) (blue color)

Terminal shows:

```
ℹ️  ₹54.14
```

Example 4: Ticker Not Found

User actions:

- Types in A1: BAJFINANCE.NS
- Types in A2: 2010-01-04

Result in A4: No data found (red color)

Terminal shows:

❌ No data found

Advantages of This Solution

1. Cross-Platform Compatibility

- Works on Mac, Windows, Linux
- No operating system limitations

2. Cloud-Based

- Access from anywhere with internet
- No need to carry database file around
- Multiple people can use the same sheet

3. Real-Time Updates

- Automatic refresh when inputs change
- No manual button clicking needed
- Instant feedback

4. Easy to Extend

- Can add more features easily (charts, multiple stocks, etc.)
- Can query date ranges
- Can export to CSV or other formats

Visual Features

Color Coding

- **Green Text + Bold:** Exact price found for that date
- **Blue Text:** Showing nearest available date
- **Red Text:** Error or no data

Formatting

- Rupee symbol (₹) for Indian stocks
- Comma separators for thousands (₹2,847.50)
- Two decimal places for precision

Error Handling

The script handles various error cases:

1. **Empty cells:** Waits for user input
2. **Invalid date format:** Shows "Invalid date format"
3. **Ticker not found:** Shows "No data found"
4. **Database connection error:** Shows clear error message
5. **Google Sheets connection error:** Shows specific error with solution
6. **Network issues:** Retries automatically

How to Use

One-Time Setup:

1. Create Google Cloud project
2. Enable APIs
3. Create service account
4. Download credentials JSON
5. Share Google Sheet with service account email
6. Install Python packages
7. Update script configuration

Daily Use:

1. Run: python3 stock_price_retriever.py
2. Choose option 1 (Auto-Update Mode)
3. Open Google Sheet in browser
4. Type ticker in A1, date in A2
5. See price in A4 automatically!

What I Learned

Technical Skills:

- Google Cloud Console and API management
- Service account authentication
- Python programming with external APIs
- SQLite database queries
- Real-time data monitoring

Problem-Solving:

- Adapted the assignment requirements to work on Mac
- Found alternative solution when traditional Excel approach didn't work
- Implemented continuous monitoring for automatic updates

Project Files

Files Included in Submission:

1. **stock_price_retriever.py** - Main Python script
2. **Stock Price Retriever-Documentation.pdf** - This document
3. **requirements.txt** - Python dependencies list
4. **README.txt** - Quick start guide

File NOT Included (but required):

- stocksqlite-320c0219c9a7.json - Credentials (private)

Acknowledgments

- **ITUS Capital:** For providing the assignment and database
- **Google Cloud Platform:** For free API access
- **Python Community:** For excellent libraries (gspread, google-auth)

Required URL

- **Google Sheet URL:** https://docs.google.com/spreadsheets/d/1X9_Navxt6YO3q7ODPqTL1CK-kI7ln6oyd_rEKPDB6XM/edit

---

Conclusion

This project successfully demonstrates:

- Dynamic database querying
- Real-time updates in spreadsheet
- Professional API integration
- Cross-platform compatibility
- Modern development practices

The solution provides the exact functionality requested in the assignment: users can input a ticker symbol and date, and the closing price appears automatically. The Mac + Python + Google Sheets approach offers additional benefits like cloud access and cross-platform support, making it a robust and professional solution.

Thank you for this opportunity to demonstrate my technical and problem-solving skills.

Submitted by: Shruti Mall
Date: 9 October 2025
For: ITUS Capital - Data Analyst Intern Position