

How Finance Teams can use ChatGPT to
write code and analyze financial data to
improve performance and efficiency

Application of ChatGPT to Build an FP&A Tool

Glenn Hopper

Table of Contents

ABSTRACT.....	2
APPROACH	2
TOOLS	2
SETUP	3
CODING ENVIRONMENT	3
SETUP TABLES	4
INCOME STATEMENT	4
BALANCE SHEET & STATEMENT OF CASH FLOWS	5
POPULATING THE TABLES.....	6
BALANCE SHEET AND STATEMENT OF CASH FLOWS.....	8
FINANCIAL STATEMENT ANALYSIS	10
CALCULATING RATIOS USING PYTHON	10
TRENDS.....	16
CORRELATION	18
FORECASTING.....	19
BENCHMARKING	22
ROBO ANALYST	23
CONCLUSION	29

Abstract

The field of Financial Planning and Analysis (FP&A) relies on a number of fundamental principles and practices, including financial modeling, budgeting, variance analysis, and strategic planning. With the rapid evolution of technology, financial analysts now have access to a range of tools that can assist them in their work. Among these is ChatGPT, an artificial intelligence tool that uses natural language processing to communicate and can perform a wide range of tasks, spanning from writing poetry to computer code.

This paper aims to demonstrate that even finance professionals with limited programming skills can leverage the power of AI to develop useful tools for FP&A tasks. By keeping the analysis simple, we aim to provide an introduction and proof of concept for the potential of ChatGPT and other AI tools in the field of FP&A. While this paper serves as a starting point, it is possible to extend the ideas presented here to create more advanced FP&A tools.

Approach

For this project, we are going to create database tables for three years of financial data. The tables will include data from the Income Statement, Balance Sheet, and Statement of Cash Flows. Once we've created the tables, we will populate them with sample data from a fictional company, and then write python scripts to extract and analyze the data to create financial ratios, identify variances, and make forecasts. All SQL statements and Python code will be written based on our prompts by ChatGPT.

Tools

- [ChatGPT](#)
- [Google Colab](#)
- [SQLite](#)
- [Python](#)

Setup

Coding Environment

This project was built in Google Colaboratory ("Colab"), which allows users to combine executable Python code and rich text with charts, images, and HTML in a single document.

First, the COLAB coding environment has to be set up to run the tools needed to make the FP&A application work. If the user is unsure how to set up the environment, ChatGPT can help:

Ask ChatGPT:

How do I set up an environment in google colab to use SQLite and python?

Response:

To set up an environment in Google Colab to use SQLite and Python, you don't need to do anything special. Google Colab already includes the sqlite3 module, which you can use to interact with SQLite databases in Python. To test that everything is set up correctly, you can try running the following code in a Colab notebook cell:

```
import sqlite3

# Connect to a database (or create one if it doesn't exist)
conn = sqlite3.connect(":memory:")

# Create a cursor
cursor = conn.cursor()

# Create a table
cursor.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT, age INTEGER)")

# Insert a row of data
cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ("Alice", 25))

# Save (commit) the changes
conn.commit()

# Query the database
cursor.execute("SELECT * FROM users")

# Fetch all rows
rows = cursor.fetchall()

# Print the rows
```

```
for row in rows:
    print(row)
```

(1, 'Alice', 25)

Setup Tables

With the environment in place, the next step is to set up the database tables that will house the required financial data. For this project, there are three tables -- one for each of the financial statements: Income Statement, Balance Sheet, and Statement of Cash Flows.

Income Statement

The Income Statement table contains rows that match the rows in the sample income statement, and columns that represent 36 months of financial information.

To stay within scope of the project, a very simple income statement is used.

ChatGPT was used to write the MySQL code to set up the table with the following prompt:

Ask ChatGPT:

Write a python script to create a SQLite table called Income Statement that includes the following rows: Sales, Cost of Goods Sold, Gross Profit, Selling, Administrative, Total Operating Expenses, Operating Income, Interest Revenue, Dividend Revenue, Interest on Loans, PreTax Income, Income Tax (25%), Net Income.

ChatGPT returned the following:

```
cursor.execute('''
    CREATE TABLE income_statement (
        Month TEXT,
        Sales REAL,
        Cost_of_Goods_Sold REAL,
        Gross_Profit REAL,
        Selling REAL,
        Administrative REAL,
        Total_Operating_Expenses REAL,
        Operating_Income REAL,
        Interest_Revenue REAL,
        Dividend_Revenue REAL,
        Interest_on_Loans REAL,
        PreTax_Income REAL,
        Income_Tax REAL,
        Net_Income REAL
    );
''')
```

<sqlite3.Cursor at 0x7fd648d220a0>

Balance Sheet & Statement of Cash Flows

Tables were also created for the Balance Sheet and Cashflow Statement.

(Note: To keep the accounting simple, we did not add adjustments to cashflow from accruals or depreciation and amortization to the Statement of Cashflows.)

Balance Sheet

```
cursor.execute('''
    CREATE TABLE balance_sheet (
        Month TEXT,
        Cash REAL,
        Accounts_Receivable REAL,
        Inventory REAL,
        Supplies REAL,
        Pre_Paid REAL,
        Total_Current_Assets REAL,
        Land REAL,
        Buildings REAL,
        Equipment REAL,
        Less_accum_Depreciation REAL,
        Net_PPE REAL,
        Total_Assets REAL,
        Short_Term_Notes_Payable REAL,
        Accounts_Payable REAL,
        Total_Current_Liabilities REAL,
        Long_Term_Notes_Payable REAL,
        Total_Liabilities REAL,
        Paid_in_Capital REAL,
        Distributions REAL,
        Retained_Earnings REAL,
        Total_Stockholders_Equity REAL,
        Total_Liabilities_Equity REAL
    );
''')
```

<sqlite3.Cursor at 0x7fd648d220a0>

Statement of Cash Flows

```
cursor.execute('''
    CREATE TABLE cashflows (
        Month TEXT,
        Cash_flows_from_operating_activities REAL,
        Cash_flows_from_investing_activities REAL,
        Cash_flows_from_financing_activities REAL,
        Net_Increase_in_cash REAL,
        Cash_at_beginning_of_period REAL,
    )
''')
```

```
Cash_at_end_of_period REAL
);
'''
```

Populating the Tables

Financial Statements aren't very useful without data, so the next step is to import data from our three financial statements into the appropriate table. We could use SQL to inject the data in each row, but that would be very time consuming. If we already have our financial data in a spreadsheet, we can use ChatGPT to help us write a Python script that lets us import the data into the appropriate table from a .CSV file.

For this project, we've stored our financial statements as CSVs on Google Drive. Before we can upload our files, we need to mount our Google Drive to Colab. We do this by running the following code:

```
from google.colab import drive
drive.mount('/content/drive')
```

These are the files we're going to import:

- [Income Statement](#)
- [Balance Sheet](#)
- [Statement of Cash Flows](#)

Ask ChatGPT:

Write a script in python that lets me import data from a csv file into the table income_statement

ChatGPT Response:

Here is a Python script that allows you to import data from a .csv file into the income_statement table:

```
import csv

# Open CSV file and read data into a list of rows
with open('/content/drive/My Drive/Colab
Notebooks/financial_statements/income_statement.csv', 'r') as f:
    reader = csv.reader(f)
    rows = [row for row in reader]

# Transpose rows and columns
rows = [list(row) for row in zip(*rows)]

# Insert rows into table
```

```

cursor.executemany('''
    INSERT INTO income_statement (Month, Sales, Cost_of_Goods_Sold,
    Gross_Profit, Selling, Administrative, Total_Operating_Expenses,
    Operating_Income, Interest_Revenue, Dividend_Revenue, Interest_on_Loans,
    PreTax_Income, Income_Tax, Net_Income)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
''', rows)

# Commit transaction
conn.commit()

```

This script assumes that the .csv file has the same structure as the financial_statements table in the database, with the values in each row corresponding to the columns in the table. It also assumes that you have already set up a MySQL connection and have the necessary credentials to access the database.

We can run our SELECT statement again to see if the table populated correctly:

```

# Query the database
cursor.execute("SELECT * FROM income_statement")

# Fetch all rows
rows = cursor.fetchall()

# Print the rows
for row in rows:
    print(row)

```

```

('Month', 'Sales', 'Cost of Goods Sold', 'Gross Profit', 'Selling', 'Administrative', 'Total Operating Expenses', 'Operating Income', 'Interest
Revenue', 'Dividend Revenue', 'Interest on Loans', 'PreTax Income', 'Income Tax (25%)', 'Net Income')
('Jan-19', 740119.0, 207233.0, 532886.0, 63946.0, 211023.0, 274969.0, 257917.0, 0.0, 0.0, 12000.0, 245917.0, 61479.0, 184438.0)
('Feb-19', 726024.0, 202989.0, 523035.0, 62764.0, 207122.0, 269886.0, 253149.0, 0.0, 0.0, 11927.0, 241222.0, 60306.0, 180917.0)
('Mar-19', 751399.0, 216432.0, 534967.0, 64196.0, 211847.0, 276043.0, 258924.0, 0.0, 0.0, 11853.0, 247071.0, 61768.0, 185303.0)
('Apr-19', 767888.0, 228884.0, 539004.0, 64680.0, 213446.0, 278126.0, 260878.0, 0.0, 0.0, 11779.0, 249099.0, 62275.0, 186824.0)
('May-19', 702106.0, 189032.0, 513074.0, 61569.0, 203177.0, 264746.0, 248328.0, 0.0, 0.0, 11705.0, 236623.0, 59156.0, 177467.0)
('Jun-19', 644183.0, 219066.0, 425117.0, 51014.0, 168346.0, 219360.0, 205757.0, 0.0, 0.0, 11630.0, 194126.0, 48532.0, 145595.0)
('Jul-19', 759086.0, 212544.0, 546542.0, 65585.0, 216431.0, 282016.0, 264526.0, 0.0, 0.0, 11555.0, 252971.0, 63243.0, 189728.0)
('Aug-19', 752333.0, 209876.0, 542457.0, 65095.0, 214813.0, 279908.0, 262549.0, 0.0, 0.0, 11480.0, 251070.0, 62767.0, 188302.0)
('Sep-19', 644319.0, 171409.0, 472910.0, 56749.0, 187272.0, 244022.0, 228888.0, 0.0, 0.0, 11404.0, 217485.0, 54371.0, 163113.0)
('Oct-19', 718325.0, 204321.0, 514004.0, 66821.0, 237470.0, 304290.0, 209714.0, 0.0, 0.0, 11328.0, 198386.0, 49596.0, 148789.0)
('Nov-19', 726048.0, 203293.0, 522755.0, 67958.0, 241513.0, 309471.0, 213284.0, 0.0, 0.0, 11251.0, 202033.0, 50508.0, 151525.0)
('Dec-19', 619022.0, 199038.0, 419984.0, 54598.0, 194033.0, 248631.0, 171353.0, 0.0, 0.0, 11174.0, 160179.0, 40045.0, 120135.0)
('Jan-20', 751686.0, 211022.0, 540664.0, 70286.0, 249787.0, 320073.0, 220591.0, 0.0, 0.0, 11097.0, 209494.0, 52374.0, 157121.0)
('Feb-20', 722377.0, 204876.0, 517501.0, 67275.0, 239085.0, 306361.0, 211140.0, 0.0, 0.0, 11019.0, 200121.0, 50030.0, 150091.0)
('Mar-20', 758888.0, 232089.0, 526799.0, 68484.0, 243381.0, 311865.0, 214934.0, 0.0, 0.0, 10941.0, 203993.0, 50998.0, 152995.0)
('Apr-20', 761325.0, 216172.0, 545153.0, 70870.0, 251861.0, 322731.0, 222422.0, 0.0, 0.0, 10862.0, 211560.0, 52890.0, 158670.0)
('May-20', 698831.0, 288840.0, 409991.0, 53299.0, 189416.0, 242715.0, 167276.0, 0.0, 0.0, 10783.0, 156493.0, 39123.0, 117370.0)
('Jun-20', 651048.0, 199656.0, 451392.0, 63195.0, 198612.0, 261807.0, 189585.0, 0.0, 0.0, 10704.0, 178881.0, 44720.0, 134160.0)
('Jul-20', 779321.0, 218210.0, 561111.0, 78556.0, 246889.0, 325444.0, 235667.0, 0.0, 0.0, 10624.0, 225042.0, 56261.0, 168782.0)
('Aug-20', 764072.0, 213940.0, 550132.0, 77018.0, 242058.0, 319076.0, 231055.0, 0.0, 0.0, 10544.0, 220511.0, 55128.0, 165383.0)
('Sep-20', 649441.0, 206741.0, 442700.0, 61978.0, 194788.0, 256766.0, 185934.0, 0.0, 0.0, 10464.0, 175470.0, 43868.0, 131603.0)

```



```
(
'Oct-20', 726438.0, 203403.0, 523035.0, 73225.0, 230136.0, 303361.0, 219675.0, 0.0, 0.0, 10383.0, 209292.0, 52323.0, 156969.0)
('Nov-20', 744544.0, 208472.0, 536072.0, 75050.0, 235872.0, 310922.0, 225150.0, 0.0, 0.0, 10302.0, 214849.0, 53712.0, 161136.0)
('Dec-20', 631651.0, 196862.0, 434789.0, 60870.0, 191307.0, 252177.0, 182611.0, 0.0, 0.0, 10220.0, 172391.0, 43098.0, 129294.0)
('Jan-21', 788394.0, 220750.0, 567644.0, 79470.0, 249763.0, 329233.0, 238410.0, 0.0, 0.0, 10138.0, 228273.0, 57068.0, 171204.0)
('Feb-21', 726486.0, 203416.0, 523070.0, 73230.0, 230151.0, 303381.0, 219689.0, 0.0, 0.0, 10055.0, 209634.0, 52409.0, 157226.0)
('Mar-21', 808991.0, 226517.0, 582474.0, 81546.0, 243474.0, 325020.0, 257453.0, 0.0, 0.0, 9972.0, 247481.0, 61870.0, 185611.0)
('Apr-21', 855319.0, 239489.0, 615830.0, 86216.0, 257417.0, 343633.0, 272197.0, 0.0, 0.0, 9889.0, 262308.0, 65577.0, 196731.0)
('May-21', 816022.0, 238486.0, 577536.0, 80855.0, 241410.0, 322265.0, 255271.0, 0.0, 0.0, 9805.0, 245466.0, 61366.0, 184099.0)
('Jun-21', 764333.0, 219013.0, 545320.0, 76345.0, 227944.0, 304288.0, 241031.0, 0.0, 0.0, 9721.0, 231310.0, 57828.0, 173483.0)
('Jul-21', 781302.0, 218765.0, 562537.0, 84381.0, 235141.0, 319521.0, 243016.0, 0.0, 0.0, 9636.0, 233380.0, 58345.0, 175035.0)
('Aug-21', 816499.0, 212022.0, 604477.0, 90672.0, 239373.0, 330044.0, 274433.0, 0.0, 0.0, 9551.0, 264881.0, 66220.0, 198661.0)
('Sep-21', 824032.0, 230729.0, 593303.0, 88995.0, 234948.0, 323943.0, 269360.0, 0.0, 0.0, 9466.0, 259894.0, 64973.0, 194920.0)
('Oct-21', 877991.0, 235837.0, 642154.0, 96323.0, 254293.0, 350616.0, 291538.0, 0.0, 0.0, 9380.0, 282158.0, 70539.0, 211618.0)
('Nov-21', 902636.0, 242738.0, 659898.0, 98985.0, 261320.0, 360304.0, 299594.0, 0.0, 0.0, 9294.0, 290300.0, 72575.0, 217725.0)
('Dec-21', 927097.0, 239587.0, 687510.0, 103126.0, 272254.0, 375380.0, 312130.0, 0.0, 0.0, 9207.0, 302923.0, 75731.0, 227192.0)
)
```

Balance Sheet and Statement of Cash Flows

Now we repeat the process to import that balance sheet and statement of cash flows from the saved .CSV files.

First, the Balance Sheet:

```
import csv

# Open CSV file and read data into a list of rows
with open('/content/drive/My Drive/Colab
Notebooks/financial_statements/balance_sheet.csv', 'r') as f:
    reader = csv.reader(f)
    rows = [row for row in reader]

# Transpose rows and columns
rows = [list(row) for row in zip(*rows)]

# Insert rows into table
cursor.executemany('''
    INSERT INTO balance_sheet (Month, Cash, Accounts_Receivable,
Inventory, Supplies, Pre_Paid, Total_Current_Assets, Land, Buildings,
Equipment, Less_accum_depreciation, Net_PPE, Total_Assets,
Short_Term_Notes_Payable, Accounts_Payable, Total_Current_Liabilities,
Long_Term_Notes_Payable, Total_Liabilities, Paid_in_Capital,
Distributions, Retained_Earnings, Total_Stockholders_Equity,
Total_Liabilities_Equity) VALUES
    (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
''', rows)

# Commit transaction
conn.commit()
```

And the same for our Statement of Cashflows:

```
# Open CSV file and read data into a list of rows
with open('/content/drive/My Drive/Colab
Notebooks/financial_statements/cashflow_statement.csv', 'r') as f:
    reader = csv.reader(f)
    rows = [row for row in reader]

# Transpose rows and columns
rows = [list(row) for row in zip(*rows)]

# Insert rows into table
cursor.executemany('''
    INSERT INTO cashflows (Month, Cash_flows_from_operating_activities,
Cash_flows_from_investing_activities,
Cash_flows_from_financing_activities, Net_Increase_in_cash,
Cash_at_beginning_of_period, Cash_at_end_of_period)
    VALUES (?, ?, ?, ?, ?, ?, ?)
''', rows)

# Commit transaction
conn.commit()
```

Financial Statement Analysis

Analyzing financial statements is at the heart of what every FP&A practitioner does every day. Along with financial modeling, budgeting, and strategic planning, financial statement analysis is the bread and butter of the profession.

Let's look at the basics of financial statement analysis, starting with some key financial ratios:

- **Profit Margin** :: Profit / Revenue
- **Acid Test / Quick Ratio** :: (Current Assets - Inventory) / Current Liabilities
- **Debt to Equity Ratio** :: Total Liabilities / Shareholders Equity
- **Working Capital Ratio** :: Current Assets / Current Liabilities
- **Return on Equity Ratio** :: (Earnings - Dividends) / Shareholders Equity
- **Return on Invested Capital (ROIC)** :: NOPAT / Invested Capital
- **Inventory Turnover Ratio** :: COGS / Average Inventory
- **Operating Leverage** :: % Change in EBIT / % Change in Sales
- **Financial Leverage** :: % change in Net Income / % Change in EBIT

Calculating Ratios Using Python

Let's create a table that contains these ratios. The months will be the same as in our financial statements, and the rows will be:

- Profit Margin
- Quick Ratio
- Debt to Equity Ratio
- Working Capital Ratio
- Return on Equity Ratio
- ROIC
- Inventory Turnover Ratio
- Operating Leverage
- Financial Leverage

This handy table will allow us to view and query on all of our key ratios at a glance. We can then quickly and easily compare our ratios to prior time periods, budgets, or industry standards.

To calculate these ratios, we'll use Python. Let's ask ChatGPT to help us write code to run these formulas on our financial statement tables.

ChatGPT Prompt:

I want to use Python to create a table that contains calculated values from three existing tables. Those tables are income_statement, balance_sheet, and cashflows. This table is called Ratios, and it contains the months row pulled from income_statement, and another row called Profit Margin. Profit Margin is calculated by pulling Gross_Profit from the income_statement table and dividing by Sales from the income_statement table.

The rest of the rows are created using the following formulas:

- Quick Ratio = (Current Assets - Inventory) / Current Liabilities
- Debt to equity ratio = Total Liabilities / Shareholders Equity
- Working Capital Ratio = Current Assets / Current Liabilities
- Return on Equity Ratio = (Earnings - Dividends) / Shareholders Equity
- Return on Invested Capital (ROIC) = Net Income / Invested Capital

- $\text{Inventory Turnover Ratio} = \text{COGS} / \text{Average Inventory}$
- $\text{Operating Leverage} = \% \text{ Change in EBIT} / \% \text{ Change in Sales}$
- $\text{Financial Leverage} = \% \text{ change in Net Income} / \% \text{ Change in EBIT}$

ChatGPT Response:

Here's a Python script you can use to create the Ratios table in SQLite:

```
cursor = conn.cursor()

cursor.execute("""
    CREATE TABLE Ratios (
        Month TEXT,
        Profit_Margin REAL,
        Quick_Ratio REAL,
        Debt_to_Equity REAL,
        Working_Capital REAL,
        Return_on_Equity REAL,
        ROIC REAL,
        Inventory_Turnover REAL,
        Operating_Leverage REAL,
        Financial_Leverage REAL
    );
""")

cursor.execute("""
    INSERT INTO Ratios (
        Month,
        Profit_Margin,
        Quick_Ratio,
        Debt_to_Equity,
        Working_Capital,
        Return_on_Equity,
        ROIC,
        Inventory_Turnover,
        Operating_Leverage,
        Financial_Leverage
    )
    SELECT
        income_statement.Month,
        (income_statement.Gross_Profit / income_statement.Sales) AS
Profit_Margin,
        ((balance_sheet.Total_Current_Assets - balance_sheet.Inventory)
/ balance_sheet.Total_Current_Liabilities) AS Quick_Ratio,
        (balance_sheet.Total_Liabilities /
balance_sheet.Total_Stockholders_Equity) AS Debt_to_Equity,
```

```

        (balance_sheet.Total_Current_Assets /
balance_sheet.Total_Current_Liabilities) AS Working_Capital,
        (
            (balance_sheet.Retained_Earnings -
balance_sheet.Distributions) /
            balance_sheet.Total_Stockholders_Equity
        ) AS Return_on_Equity,
        (income_statement.Net_Income / balance_sheet.Paid_in_Capital) AS
ROIC,
        (
            income_statement.Cost_of_Goods_Sold / (
                (balance_sheet.Inventory +
                (SELECT Inventory FROM balance_sheet WHERE Month =
(SELECT MAX(Month) FROM balance_sheet WHERE Month <
income_statement.Month))) / 2
            )
        ) AS Inventory_Turnover,
        (
            (income_statement.Operating_Income -
            (SELECT Operating_Income FROM income_statement WHERE Month =
(SELECT MAX(Month) FROM income_statement WHERE Month <
income_statement.Month))) /
            (SELECT Operating_Income FROM income_statement WHERE Month =
(SELECT MAX(Month) FROM income_statement WHERE Month <
income_statement.Month))
        ) * 100 AS Operating_Leverage,
        (
            (income_statement.Net_Income -
            (SELECT Net_Income FROM income_statement WHERE Month =
(SELECT MAX(Month) FROM income_statement WHERE Month <
income_statement.Month))) /
            (SELECT Net_Income FROM income_statement WHERE Month =
(SELECT MAX(Month) FROM income_statement WHERE Month <
income_statement.Month))
        ) * 100 AS Financial_Leverage
FROM
    income_statement
JOIN balance_sheet ON income_statement.Month = balance_sheet.Month;
""")

# Commit the changes
conn.commit()

```

Now let's take a look at that data in a formatted table to see if it imported correctly.

We can ask ChatGPT to help:

ChatGPT Prompt:

Please write a python script that outputs the entire contents of the ratios table in an easy-to-read table that contains column headers and text formatting. Profit_Margin, Return_on_Equity, and ROIC should be shown as a percentage with no decimal places, and the rest of the numbers should be displayed to two decimal places.

ChatGPT Response:

Here's a Python script that outputs the entire contents of the ratios table in a readable format:

```
from prettytable import PrettyTable

# Create a cursor
cursor = conn.cursor()

# Execute the query
cursor.execute("SELECT * FROM Ratios")

# Fetch all the rows
rows = cursor.fetchall()

# Get the column names
columns = [description[0] for description in cursor.description]

# Create a pretty table
table = PrettyTable(columns)

# Add each row of data to the table
for row in rows:
    month, profit_margin, quick_ratio, debt_to_equity, working_capital,
    return_on_equity, roic, inventory_turnover, operating_leverage,
    financial_leverage = row

    # Convert values to floats, or to 0 if None
    profit_margin = float(profit_margin) if profit_margin is not None
    else 0
    quick_ratio = float(quick_ratio) if quick_ratio is not None else 0
    debt_to_equity = float(debt_to_equity) if debt_to_equity is not None
    else 0
    working_capital = float(working_capital) if working_capital is not
    None else 0
```

```

    return_on_equity = float(return_on_equity) if return_on_equity is
not None else 0
    roic = float(roic) if roic is not None else 0
    inventory_turnover = float(inventory_turnover) if inventory_turnover
is not None else 0
    operating_leverage = float(operating_leverage) if operating_leverage
is not None else 0
    financial_leverage = float(financial_leverage) if financial_leverage
is not None else 0

    # Format Profit_Margin, Return_on_Equity, and ROIC as a percentage
with one decimal place
    profit_margin = "{:.1%}".format(profit_margin)
    return_on_equity = "{:.1%}".format(return_on_equity)
    roic = "{:.1%}".format(roic)

    # Format the other values as strings with 2 decimal places
    quick_ratio = "{:.2f}".format(quick_ratio)
    debt_to_equity = "{:.2f}".format(debt_to_equity)
    working_capital = "{:.2f}".format(working_capital)
    inventory_turnover = "{:.2f}".format(inventory_turnover)
    operating_leverage = "{:.2f}".format(operating_leverage)
    financial_leverage = "{:.2f}".format(financial_leverage)

    # Add the row to the table
    table.add_row([month, profit_margin, quick_ratio, debt_to_equity,
working_capital, return_on_equity, roic, inventory_turnover,
operating_leverage, financial_leverage])

# Commit the changes
conn.commit()

# Print the table
print(table)

```

Month	Profit Margin	Quick Ratio	Debt_to_Equity	Working_Capital	Return_on_Equity	ROIC	Inventory_Turnover	Operating_Leverage	Financial_Leverage
Month	0.0%	0.00	0.00	0.00	0.0%	0.0%	0.00	0.00	0.00
Jan-19	72.0%	16.30	0.16	17.89	10.0%	2.5%	1.14	0.00	0.00
Feb-19	72.0%	11.92	0.16	13.11	11.9%	2.4%	1.19	0.00	0.00
Mar-19	71.2%	13.25	0.15	14.70	13.8%	2.5%	1.15	0.00	0.00
Apr-19	70.2%	12.97	0.18	14.93	-0.7%	2.5%	0.00	0.00	0.00
May-19	73.1%	10.72	0.16	11.94	10.1%	2.4%	0.98	0.00	0.00
Jun-19	66.0%	8.96	0.16	10.02	11.7%	1.9%	1.24	0.00	0.00
Jul-19	72.0%	10.77	0.16	11.90	13.6%	2.5%	1.11	0.00	0.00
Aug-19	72.1%	12.14	0.18	13.74	-0.6%	2.5%	0.97	0.00	0.00
Sep-19	73.4%	8.62	0.17	9.54	9.9%	2.2%	0.99	0.00	0.00
Oct-19	71.6%	11.82	0.16	13.29	11.5%	2.0%	1.17	0.00	0.00
Nov-19	72.0%	9.36	0.20	10.49	13.0%	2.0%	1.99	0.00	0.00
Dec-19	67.8%	7.68	0.23	8.77	-1.4%	1.6%	1.09	0.00	0.00
Jan-20	71.9%	8.49	0.21	9.56	8.8%	2.1%	1.08	0.00	0.00
Feb-20	71.6%	8.85	0.20	9.89	10.5%	2.0%	1.05	0.00	0.00
Mar-20	69.4%	8.45	0.20	9.45	12.1%	2.0%	1.14	0.00	0.00
Apr-20	71.6%	8.51	0.23	9.48	-0.6%	2.1%	0.88	0.00	0.00
May-20	58.7%	6.76	0.21	7.70	8.4%	1.6%	1.41	0.00	0.00
Jun-20	69.3%	7.26	0.21	8.18	9.9%	1.8%	1.05	0.00	0.00
Jul-20	72.0%	8.41	0.20	9.30	11.7%	2.3%	1.11	0.00	0.00
Aug-20	72.0%	8.13	0.23	9.00	-0.4%	2.2%	0.95	0.00	0.00
Sep-20	68.2%	6.78	0.21	7.62	8.5%	1.8%	1.11	0.00	0.00
Oct-20	72.0%	7.55	0.21	8.36	10.2%	2.1%	1.08	0.00	0.00
Nov-20	72.0%	7.56	0.20	8.35	11.9%	2.1%	1.08	0.00	0.00
Dec-20	68.8%	6.35	0.23	7.12	-1.0%	1.7%	1.03	0.00	0.00
Jan-21	72.0%	7.96	0.21	8.71	8.6%	2.3%	1.17	0.00	0.00
Feb-21	72.0%	7.37	0.21	8.10	10.3%	2.1%	1.09	0.00	0.00
Mar-21	72.0%	8.17	0.20	8.88	12.3%	2.5%	1.23	0.00	0.00
Apr-21	72.0%	8.39	0.23	9.08	-0.2%	2.6%	1.31	0.00	0.00
May-21	70.8%	7.87	0.21	8.53	9.6%	2.5%	1.32	0.00	0.00
Jun-21	71.3%	7.41	0.21	8.06	11.5%	2.3%	1.23	0.00	0.00

Ratios are great, but so far we haven't done anything that you can't do in a spreadsheet. The real power comes when we can have our application spit out key ratios and variances that require our attention. We could do things like create scripts that look for variances (by month, year, etc.) in each category. We can look for variances month over month or anything that's outside of our three-month moving average. We can look at changes over prior year. We can look for these variances within the financial statements themselves, and within the ratios.

I asked ChatGPT for help with some of the things we could do with this data now that we have it all imported into our SQL tables.

Here's what ChatGPT came up with:

Here are some things you can look for and python scripts to help with analysis:

1. **Trends:** Plot the financial metrics over time and look for any trends, such as increasing or decreasing profit margins, ROE, etc. You can use the matplotlib library in python to create these plots.
2. **Correlation:** Check the correlation between different financial metrics, such as the relationship between profit margins and ROE. You can use the pandas library in python to calculate the correlation coefficients.
3. **Financial ratios:** Calculate commonly used financial ratios such as the debt-to-equity ratio and the current ratio. You can use the pandas library to calculate these ratios and add them to your Ratios table.*
4. **Forecasting:** Use machine learning techniques such as regression or time series forecasting to predict future financial metrics based on historical data. You can use the scikit-learn library in python to build and train your models.
5. **Benchmarking:** Compare your company's financial metrics to those of similar companies in your industry. You can use the pandas library to calculate the z-scores for each metric and determine how your company compares to its peers.

These are just a few ideas, but there are many other tools and techniques you can use for financial planning and analysis.

These are all great suggestions. Let's look at them one-by-one:

Trends

Let's look at the top and bottom line numbers for our fictional company. We can use pandas and matplotlib in Python to chart out the company's Sales and Net Income over the three years for which we have financial statements. We can also create dataframes that show Year Over Year (YOY) changes in each over the last 12 months.

ChatGPT Prompt:

Can you help me write a python script that looks at trends over time in a table called income_statement that contains the columns: Month, Sales, Cost_of_Goods_Sold, Gross_Profit, Selling, Administrative, Total_Operating_Expenses, Operating_Income, Interest_Revenue, Dividend_Revenue, Interest_on_Loans, PreTax_Income, Income_Tax, Net_Income. I'd like to identify trends and highlight any variations within Sales and Operating Income over time.

ChatGPT Response:

Certainly, here's a python script using the pandas library that plots the trends in Sales and Operating Income over time. It will also will give you the year-over-year growth rates for both 'Sales' and 'Operating_Income' as a percentage. You can use these growth rates to analyze trends and make informed decisions regarding your company's financial planning and analysis.

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the data into a pandas dataframe
df = pd.read_sql("SELECT * FROM income_statement", conn)

# Exclude the first row, which contains the headers
df = df[1:]

# Add a new column with a fixed string (e.g. "01-") concatenated with
the Month column
df['new_month'] = '01-' + df['Month']

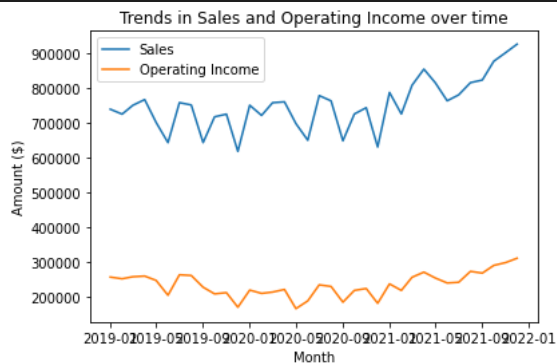
# convert the Month column to a datetime object
df['Month'] = pd.to_datetime(df['Month'], format='%b-%y')

# Plot the trends in Sales and Operating Income over time
plt.plot(df['Month'], df['Sales'], label='Sales')
plt.plot(df['Month'], df['Operating_Income'], label='Operating Income')
plt.xlabel('Month')
plt.ylabel('Amount ($)')
plt.title('Trends in Sales and Operating Income over time')
plt.legend()
```

```
plt.show()

# Calculate the year-over-year growth rate for 'Sales' and
'Operating_Income'
df['Sales_YoY'] = (df['Sales'].pct_change(12) * 100).round(2)
df['Operating_Income_YoY'] = (df['Operating_Income'].pct_change(12) *
100).round(2)

# Print the year-over-year growth rate for 'Sales' and
'Operating_Income'
print("Year-over-Year Growth Rate for Sales:")
print(df['Sales_YoY'].tail(12))
print("\nYear-over-Year Growth Rate for Operating Income:")
print(df['Operating_Income_YoY'].tail(12))
```



Year-over-Year Growth Rate for Sales:

```
25  4.88
26  0.57
27  6.60
28 12.35
29 16.77
30 17.40
31  0.25
32  6.86
33 26.88
34 20.86
35 21.23
36 46.77
```

Name: Sales_YoY, dtype: float64

Year-over-Year Growth Rate for Operating Income:

```
25  8.08
26  4.05
27 19.78
28 22.38
29 52.60
30 27.14
31  3.12
32 18.77
33 44.87
34 32.71
35 33.06
36 70.93
```

Name: Operating_Income_YoY, dtype: float64

Correlation

Check the correlation between different financial metrics, such as the relationship between profit margins and ROE. You can use the pandas library in python to calculate the correlation coefficients.

Let's look at a few income statement accounts that we would assume would have a very high correlation: Sales, Gross Profit, Total Operating Expenses, and Net Income.

We can get ChatGPT to help write code for that.

ChatGPT Prompt:

I'd like to write a Python script that pulls Sales, Gross_Profit, Total_Operating_Expenses, and Net_Income into a data frame and finds the correlation coefficient between each. Then it prints a table and a chart showing the correlation of each over time. When it pulls in the data for the data frame, it should exclude the first row, which contains header information.

ChatGPT Response:

Here's the code to accomplish what you've described:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Connect to the database and get the data
# conn = sqlite3.connect('database.db')
df = pd.read_sql_query("SELECT * FROM income_statement", conn)
df = df[1:] # exclude the first row, which contains header information

# Add a new column with a fixed string (e.g. "01-") concatenated with
the Month column
df['new_month'] = '01-' + df['Month']

# convert the Month column to a datetime object
df['Month'] = pd.to_datetime(df['Month'], format='%b-%y')

# Create a list of the columns to analyze
cols_to_analyze = ['Sales', 'Gross_Profit', 'Total_Operating_Expenses',
'Net_Income']

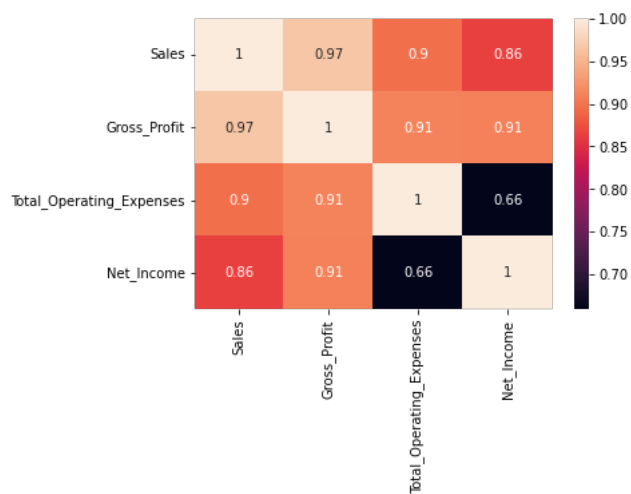
# Iterate over the columns and convert the values to numbers
for col in cols_to_analyze:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Drop NaN values from the data frame
df = df.dropna()
```

```
df_without_na = df[cols_to_analyze].dropna()

# Check that the data frame contains values
# print(df.columns)
# print(df.head())
# print(df.values)

# Calculate the correlation matrix and create a heatmap
corr = df[cols_to_analyze].dropna().corr()
# print(corr.shape)
sns.heatmap(corr, annot=True)
plt.show()
```



This heatmap is a great way to look at a glance and see the correlation between different financial metrics. We see in this example that Sales and Gross Profit are highly correlated. Interestingly, we see the least correlation between Total Operating Expenses and Net Income. Similarly, net income has a lower correlation coefficient to Sales than to Gross Profit. These observations may indicate additional reason to dig in to our expense structure.

A more detailed study that looked at individual income and expense accounts would likely net even greater results.

Forecasting

Forecasting: Use machine learning techniques such as regression or time series forecasting to predict future financial metrics based on historical data. You can use the scikit-learn library in python to build and train your models.

For the purpose of this project, we'll keep it simple and ask ChatGPT to write code that creates a 12-month moving average of our existing data and then rolls the results forward another 12 months.

Creating a table that contains 12-month moving averages is helpful also for trend analysis. We can look at actual results for each month and see how those compare.

ChatGPT Prompt:

I would like to pull the data from the income_statement table and use it to create a forecast for the next 12 months.

ChatGPT Response:

Here's a sample code that could be used to accomplish this in Google Colab:

```
import pandas as pd
from prettytable import PrettyTable
from datetime import timedelta
from prettytable import PrettyTable

# Query the income statement table
df = pd.read_sql_query("SELECT * from income_statement", conn)
df = df[1:] # exclude the first row, which contains header information

# Define a list of dates as text in the format "MMM-YY"
dates = ["Jan-19", "Feb-19", "Mar-19", "Apr-19", "May-19", "Jun-19",
         "Jul-19", "Aug-19", "Sep-19", "Oct-19", "Nov-19", "Dec-19",
         "Jan-20", "Feb-20", "Mar-20", "Apr-20", "May-20", "Jun-20",
         "Jul-20", "Aug-20", "Sep-20", "Oct-20", "Nov-20", "Dec-20",
         "Jan-21", "Feb-21", "Mar-21", "Apr-21", "May-21", "Jun-21",
         "Jul-21", "Aug-21", "Sep-21", "Oct-21", "Nov-21", "Dec-21"]

# Convert the list of dates to a Pandas Series and set it as the index
dates = pd.Series(dates, name="Month")
dates.index = pd.to_datetime(dates + "-01", format="%b-%y-%d")

# Set the Month column as the index
df = df.set_index('Month')

# Iterate over the columns and convert the values to numbers
for col in df.columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Drop NaN values from the data frame
df = df.dropna()

# Create a rolling mean of the last 12 months for each column
df_rolling = df.rolling(window=12).mean()
```

```

# Get the last month from the rolling mean data frame
last_month = df_rolling.index[-1]

# Convert the last_month string to a datetime object
last_month = pd.to_datetime(last_month, format="%b-%y")

# Convert the datetime object to a string in the desired format
next_month = last_month.strftime("%b-%y")

# Generate the values for the next 12 months
for i in range(12):
    last_month = pd.to_datetime(next_month + "-01", format="%b-%y-%d")
    next_month = last_month + pd.DateOffset(months=1)
    next_month = next_month.strftime("%b-%y")
    row_values = [next_month] + [None] * (len(df_rolling.columns) - 1)
    df_rolling.loc[len(df_rolling)] = row_values

# Drop the first 11 rows since they will have NaN values
df_rolling = df_rolling.dropna()

# Reset the index so the Month column is a regular column again
df_rolling = df_rolling.reset_index()

# Show the entire data frame
# display(df_rolling)

# Create a table object
table = PrettyTable()

# Set the header row
table.field_names = df_rolling.columns

# Add the rows from the data frame to the table
for row in df_rolling.to_dict(orient='records'):
    # Convert all the numerical values in the row to integers
    row = {k: int(v) if isinstance(v, float) else v for k, v in
row.items()}
    table.add_row(row.values())

# Print the table
print(table)

```

Month	Sales	Cost_of_Goods_Sold	Gross_Profit	Selling	Administrative	Total_Operating_Expenses	Operating_Income	Interest_Revenue	Dividend
Dec-19	712571	205343	507227	62081	208874	270955	236272	0	0
Jan-20	713534	205658	507876	62609	212104	274714	233161	0	0
Feb-20	713231	205816	507414	62985	214768	277753	229661	0	0
Mar-20	713855	207120	506734	63342	217396	280739	225995	0	0
Apr-20	713308	206061	507246	63858	220597	284456	222790	0	0
May-20	713035	214378	498656	63169	219450	282620	216036	0	0
Jun-20	713607	212761	500846	64184	221972	286157	214688	0	0
Jul-20	715293	213233	502060	65265	224511	289776	212283	0	0
Aug-20	716271	213572	502699	66259	226781	293040	209659	0	0
Sep-20	716698	216516	500182	66694	227407	294102	206079	0	0
Oct-20	717374	216440	500934	67228	226796	294025	206909	0	0
Nov-20	718916	216871	502044	67819	226326	294146	207898	0	0
Dec-20	719968	216690	503278	68342	226099	294441	208836	0	0
Jan-21	723027	217500	505526	69107	226097	295204	210321	0	0
Feb-21	723369	217379	505990	69603	225352	294956	211034	0	0
Mar-21	727545	216914	510630	70692	225360	296052	214577	0	0
Apr-21	735378	218858	516520	71971	225823	297794	218725	0	0
May-21	745143	214661	530482	74267	230156	304423	226058	0	0
Jun-21	754584	216274	538309	75363	232600	307963	230345	0	0
Jul-21	754749	216321	538428	75848	231621	307470	230957	0	0
Aug-21	759118	216161	542957	76986	231398	308384	234572	0	0
Sep-21	773667	218160	555507	79237	234744	313982	241524	0	0
Oct-21	786297	220863	565433	81162	236757	317920	247513	0	0
Nov-21	799471	223718	575752	83157	238878	322035	253716	0	0
Dec-21	824091	227279	596812	86678	245624	332302	264510	0	0

Benchmarking

The final recommendation ChatGPT had for deriving value from our data in the databases was to run benchmarking analyses.

Benchmarking: Compare your company's financial metrics to those of similar companies in your industry. You can use the pandas library to calculate the z-scores for each metric and determine how your company compares to its peers.

We could get ChatGPT to write this code if we had industry data for our fictional company.

From ChatGPT:

Benchmarking is a process of evaluating a company's financial performance by comparing it to similar companies in the same industry. This is done to gain insights into how well the company is performing in comparison to its competitors, and to identify areas where it may be underperforming or overperforming.

To carry out this process, one can use financial metrics such as revenue growth, profit margin, return on investment (ROI), and other key performance indicators (KPIs) that are relevant to the industry. By calculating the z-scores for each metric, a standardized score that indicates how far a particular value is from the average value of a group, it's possible to compare the company's performance against its peers.

Using the pandas library, one can easily perform this analysis on financial data sets. The resulting analysis can help a company identify areas where it is performing well and where it needs to improve. By benchmarking against similar companies in the industry, a company can gain valuable insights and develop strategies for improving its financial performance.

Robo Analyst

While there is nothing earth shattering in taking data from spreadsheets and moving it to a database, putting data in this format makes it far more accessible to other software and web-based computer tools. By stripping data from local files that can be manipulated by multiple actors in every instance (or copy) of the file, we can better protect the integrity of our data, while sharing it with a wider audience.

All of the actions performed in this project can be completed in tools such as Microsoft Excel or Google Sheets, but there are numerous advantages to moving your financial data to a database.

From ChatGPT:

There are several advantages to using databases rather than spreadsheets to track and report financial and performance data:

1. **Scalability:** Databases can handle a much larger volume of data than spreadsheets. As a company grows and its data volume increases, a database can continue to store and process the data efficiently, whereas a spreadsheet may become slow and unwieldy.
2. **Data integrity:** Databases enforce data integrity rules, ensuring that data is consistent and accurate. This reduces the risk of errors and helps maintain the quality of the data over time.
3. **Security:** Databases offer more advanced security features than spreadsheets. Access to sensitive data can be restricted to authorized users, and data can be encrypted to protect it from unauthorized access.
4. **Collaboration:** Databases can support multiple users working with the same data simultaneously. This enables collaboration and makes it easier for teams to work together and share information.
5. **Performance:** Databases are optimized for querying and analyzing large datasets, making it easier to generate reports and gain insights into company performance.
6. **Automation:** Databases can be integrated with other software applications to automate data processing and reporting tasks. This reduces manual effort and helps streamline business processes.

But perhaps most importantly in this new world of ever-increasing proliferation of AI/ML tools, moving financial data and reporting into a database system (be it through direct connections to software applications or use of a data lake or data warehouse) facilitates use of new and powerful tools that can help fuel data-driven decisions.

There is a wide array of publicly available machine learning tools that can be used to expand your FP&A Toolkit.

TensorFlow, for example, is a popular open-source software library for machine learning and artificial intelligence. It was developed by the Google Brain team and released in 2015. TensorFlow is used for a variety of tasks, including image recognition, natural language processing, and time series analysis. TensorFlow is designed to make it easier for developers to create and deploy machine learning models by providing a flexible and scalable platform for building and training neural networks. It includes a wide range of built-in functions and algorithms that can be used to implement various machine learning tasks.

Other examples of similar machine learning and artificial intelligence libraries and frameworks include:

- **PyTorch** - an open-source machine learning library developed by Facebook's AI research team. It is known for its ease of use and dynamic computation graph.
- **Keras** - a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It is designed to enable fast experimentation with deep neural networks.
- **Theano** - a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It can be used to build and train neural networks and deep learning models.
- **Scikit-learn** - a popular machine learning library for Python that provides simple and efficient tools for data mining and data analysis. It includes a wide range of algorithms for classification, regression, and clustering.
- **Caffe** - a deep learning framework developed by the Berkeley Vision and Learning Center (BVLC) and used for image and video recognition. It is known for its speed and efficiency in training convolutional neural networks.

There are also a number of low- and no-code tools out there that make machine learning more accessible to non-developers.

One such tool is called DataRobot (<https://www.datarobot.com>). DataRobot is a platform for automated machine learning (AutoML) that helps organizations build and deploy machine learning models at scale. The platform enables organizations to automate much of the work involved in building, deploying, and maintaining machine learning models, allowing them to accelerate their time to value and make better predictions.

Beyond machine learning applications, moving report data into a database allows integration with the new suite of tools available to let humans interact with the data in a more conversational manner. Imagine the power of CHatGPT applied to your company's financials. We could do this via an API to GPT-3 that would take written natural language questions from a user and output responses in the way a finance professional would.

Incorporating natural language into financial or performance queries is a key step in democratizing data. Companies around the world use dashboards to display pre-defined metrics that highlight current business performance. While these dashboards are incredibly useful, they are limited in that for the most part they only provide charts, graphs and at-a-glance information.

What if dashboard users could interact with the data in a new way? What if instead of just seeing canned reports, they could ask specific questions about the data.

These are questions that would normally require a Slack message or email to a business analyst. In this new world of natural language processing, they could be posed directly to the system.

Some example questions may include:

- What is the compound annual growth rate for XYZ company?
- How were sales this June compared to the same period last year?
- What is our profit margin for YTD June of 2021 vs. YTD June of 2020? 2019?
- Are there any negative trends in the company's financial performance that I need to look at more closely?

There are already tools on the market like Bink (<https://usebink.com>), which allows users to write complex database queries without the need for extensive knowledge of SQL or MongoDB. Bink also

supports NoSQL databases, enabling users to extract and analyze data from a wider range of sources, giving them access to valuable insights that would have been otherwise inaccessible. In this simplified example below, we will show how such a tool might work.

Starting with a simple 12-month income statement from our sample company, we can create a basic chatbot in Python that lets users ask questions about the information on the report.

Let's start by creating a new table to query. ChatGPT can help us with that.

ChatGPT Prompt:

Please write code for me that creates a new database called `income_statement_2021` with the same rows as the `income_statement` database. We will then populate the data from a file called `IncomeStatement_2021.CSV`

ChatGPT Response:

Use the following code to create the table `income_statement_2021`.

```
cursor.execute('''
    CREATE TABLE income_statement_2021 (
        Month TEXT,
        Sales REAL,
        Cost_of_Goods_Sold REAL,
        Gross_Profit REAL,
        Selling REAL,
        Administrative REAL,
        Total_Operating_Expenses REAL,
        Operating_Income REAL,
        Interest_Revenue REAL,
        Dividend_Revenue REAL,
        Interest_on_Loans REAL,
        PreTax_Income REAL,
        Income_Tax REAL,
        Net_Income REAL
    );
''')
```

You can use the following code to populate your data from the CSV file.

```
import csv

# Open CSV file and read data into a list of rows
with open('/content/drive/My Drive/Colab
Notebooks/financial_statements/IncomeStatement_2021.csv', 'r') as f:
    reader = csv.reader(f)
    rows = [row for row in reader]
```

```

# Transpose rows and columns
rows = [list(row) for row in zip(*rows)]

# Insert rows into table
cursor.executemany('''
    INSERT INTO income_statement_2021 (Month, Sales, Cost_of_Goods_Sold,
    Gross_Profit, Selling, Administrative, Total_Operating_Expenses,
    Operating_Income, Interest_Revenue, Dividend_Revenue, Interest_on_Loans,
    PreTax_Income, Income_Tax, Net_Income)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
''', rows)

# Commit transaction
conn.commit()

# Query the database
cursor.execute("SELECT * FROM income_statement_2021")

# Fetch all rows
rows = cursor.fetchall()

# Print the rows
for row in rows:
    print(row)

```

```

('Month', 'Sales', 'Cost of Goods Sold', 'Gross Profit', 'Selling', 'Administrative', 'Total Operating Expenses', 'Operating Income', 'Interest Revenue', 'Dividend Revenue', 'Interest on Loans', 'PreTax Income', 'Income Tax', 'Net Income')
('January', 788394.0, 220750.0, 567644.0, 79470.0, 249763.0, 329233.0, 238410.0, 0.0, 0.0, 10138.0, 228273.0, 57068.0, 171204.0)
('February', 726486.0, 203416.0, 523070.0, 73230.0, 230151.0, 303381.0, 219689.0, 0.0, 0.0, 10055.0, 209634.0, 52409.0, 157226.0)
('March', 808991.0, 226517.0, 582474.0, 81546.0, 243474.0, 325020.0, 257453.0, 0.0, 0.0, 9972.0, 247481.0, 61870.0, 185611.0)
('April', 855319.0, 239489.0, 615830.0, 86216.0, 257417.0, 343633.0, 272197.0, 0.0, 0.0, 9889.0, 262308.0, 65577.0, 196731.0)
('May', 816022.0, 238486.0, 577536.0, 80855.0, 241410.0, 322265.0, 255271.0, 0.0, 0.0, 9805.0, 245466.0, 61366.0, 184099.0)
('June', 764333.0, 219013.0, 545320.0, 76345.0, 227944.0, 304288.0, 241031.0, 0.0, 0.0, 9721.0, 231310.0, 57828.0, 173483.0)
('July', 781302.0, 218765.0, 562537.0, 84381.0, 235141.0, 319521.0, 243016.0, 0.0, 0.0, 9636.0, 233380.0, 58345.0, 175035.0)
('August', 816499.0, 212022.0, 604477.0, 90672.0, 239373.0, 330044.0, 274433.0, 0.0, 0.0, 9551.0, 264881.0, 66220.0, 198661.0)
('September', 824032.0, 230729.0, 593303.0, 88995.0, 234948.0, 323943.0, 269360.0, 0.0, 0.0, 9466.0, 259894.0, 64973.0, 194920.0)
('October', 877991.0, 235837.0, 642154.0, 96323.0, 254293.0, 350616.0, 291538.0, 0.0, 0.0, 9380.0, 282158.0, 70539.0, 211618.0)
('November', 902636.0, 242738.0, 659898.0, 98985.0, 261320.0, 360304.0, 299594.0, 0.0, 0.0, 9294.0, 290300.0, 72575.0, 217725.0)
('December', 927097.0, 239587.0, 687510.0, 103126.0, 272254.0, 375380.0, 312130.0, 0.0, 0.0, 9207.0, 302923.0, 75731.0, 227192.0)

```

Once we've created the database, we can get ChatGPT to write code that creates a simple chatbot to interact with the data.

ChatGPT Prompt:

Can you write a program to run in colab that lets users ask questions in natural language and then goes and queries the database to find the answer?

ChatGPT Response:

Yes, I can provide you with a sample Python program that uses natural language processing to allow users to ask questions about the "income_statement" table in your SQLite database, and then queries the database to find the answer. Here is a sample program that you can modify to suit your specific needs:

```
# Import necessary libraries
import sqlite3
import spacy
from spacy.matcher import PhraseMatcher

# Load the pre-trained English language model in Spacy
nlp = spacy.load("en_core_web_sm")

# Define a function to extract relevant entities from user input
def extract_entities(query):
    entities = []
    doc = nlp(query)
    matcher = PhraseMatcher(nlp.vocab)
    months = ["January", "February", "March", "April", "May", "June",
"July", "August", "September", "October", "November", "December"]
    fields = ["Sales", "Cost of Goods Sold", "Gross Profit", "Selling",
"Administrative", "Total Operating Expenses", "Operating Income",
"Interest Revenue", "Dividend Revenue", "Interest on Loans", "PreTax
Income", "Income Tax", "Net Income"]
    for month in months:
        matcher.add("MONTHS", None, nlp(month))
    for field in fields:
        matcher.add("FIELDS", None, nlp(field))
    matches = matcher(doc)
    for match_id, start, end in matches:
        entity = doc[start:end].text
        if entity in months or entity in fields:
            entities.append(entity)
    return entities

# Connect to the database
# conn = sqlite3.connect('/content/drive/My Drive/Colab
Notebooks/financial_statements/income_statement_2021.db')
c = conn.cursor()

# Define a function to execute a query based on user input
def execute_query(query):
    entities = extract_entities(query)
    month = None
```

```

    field = None
    for entity in entities:
        if entity in ["January", "February", "March", "April", "May",
"June", "July", "August", "September", "October", "November",
"December"]:
            month = entity
            elif field is None:
                # Handle multi-word field names by concatenating them with
underscores
                field = entity.lower().replace(" ", "_")
            else:
                field = None
        if month and field:
            c.execute("SELECT {} FROM income_statement_2021 WHERE Month =
?".format(field), (month,))
            result = c.fetchone()
            if result:
                return f"In {month}, {field.replace('_', ' ')} was
{result[0]}"
            else:
                return "No data found"
        else:
            return "Please specify a month and a field"

# Define a function to handle user input and respond with the result
def chat():
    print("Welcome! Ask me a question about the income statement by
specifying a month and a field (e.g. 'What was Sales in March?')\nType
'End' to terminate the chat.")
    while True:
        user_input = input("You: ")
        if user_input.lower() in ['exit', 'quit', 'bye', 'end']:
            print("Bot: Bye!")
            break
        else:
            response = execute_query(user_input)
            print("Bot:", response)

# Start the chat
chat()

```

Welcome! Ask me a question about the income statement by specifying a month and a field (e.g. 'What was Sales in March?')
Type 'End' to terminate the chat.

You: What were sales in March?

Bot: Please specify a month and a field

You: What was our Cost of Goods Sold in February?

Bot: In February, cost of goods sold was 203416.0

You: What was our Net Income in December?

Bot: In December, net income was 227192.0

You: Bye

Bot: Bye!

Conclusion

In this project, we took three years of financial statements for a fictional company and used ChatGPT to write code to store the financial data in a database. We then used ChatGPT to analyze the financial statements, find trends and correlations, and create a simple forecast. To further showcase the value of databases, we used ChatGPT to write code that let us query the data using natural language.

Databases offer more advanced analysis capabilities than spreadsheets, allowing financial analysts to create more sophisticated financial models and forecasts. Additionally, with a database, they can automate many of their analysis and reporting tasks, freeing up time for more strategic activities.

In the near future, querying financial data will become even easier with the help of natural language processing (NLP) and artificial intelligence (AI). Tools like ChatGPT can already be used to query financial data using natural language. In the future, software will be able to answer the kinds of questions that today have to be answered by a human financial analyst, saving time and improving accuracy.