Robert Silaghi, Phu Le, and Sophia Danak

Github Link: https://github.com/phulvinh/si201-asset-classes-final-project.git

1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

The objective of our project was to analyze the performance of companies issuing convertible securities in relation to market conditions.

APIs / Websites Planned:

- SEC API - fetch filings for convertibles
- Stock Price API (Yahoo Finance) - historical stock prices
- FRED API - historical 10-year Treasury yields

Data Planned to Gather:

- Filing information: company, filing date, type
- Stock prices: daily open, close, high, low, volume
- Interest rates: daily 10-year Treasury yields

2. The goals that were achieved, including what APIs/websites you actually worked with and what data did you gather (10 points)

The objective of our project remained the same and was achieved, as we successfully analyzed the performance of companies issuing convertible bonds across various market conditions.

APIs / Websites Used:

- SEC API - fetch filings for convertibles
- Stock Price API - historical stock prices
- FRED API - historical 10-year Treasury yields

Data Gathered:

- 1. Filing Information (from SEC API)
    - Cik - Central Index Key for the company
    - Company_name
    - Ticker
    - Filing_date
    - Filing_url - URL link to the full filing HTML
    - Is_convertible - Boolean flag indicating a convertible debt filing
- 2. Stock Prices (from StockData API)
    - Dates
    - Close - Closing price of the stock on that date
    - High - Highest price of the stock on that date
    - Low - Lowest price of the stock on that date
    - Volume - Number of shares traded
- 3. Interest Rates (from FRED API)
    - Dates
    - Daily 10-year Treasury yield (%)

3. The problems that you faced (10 points)

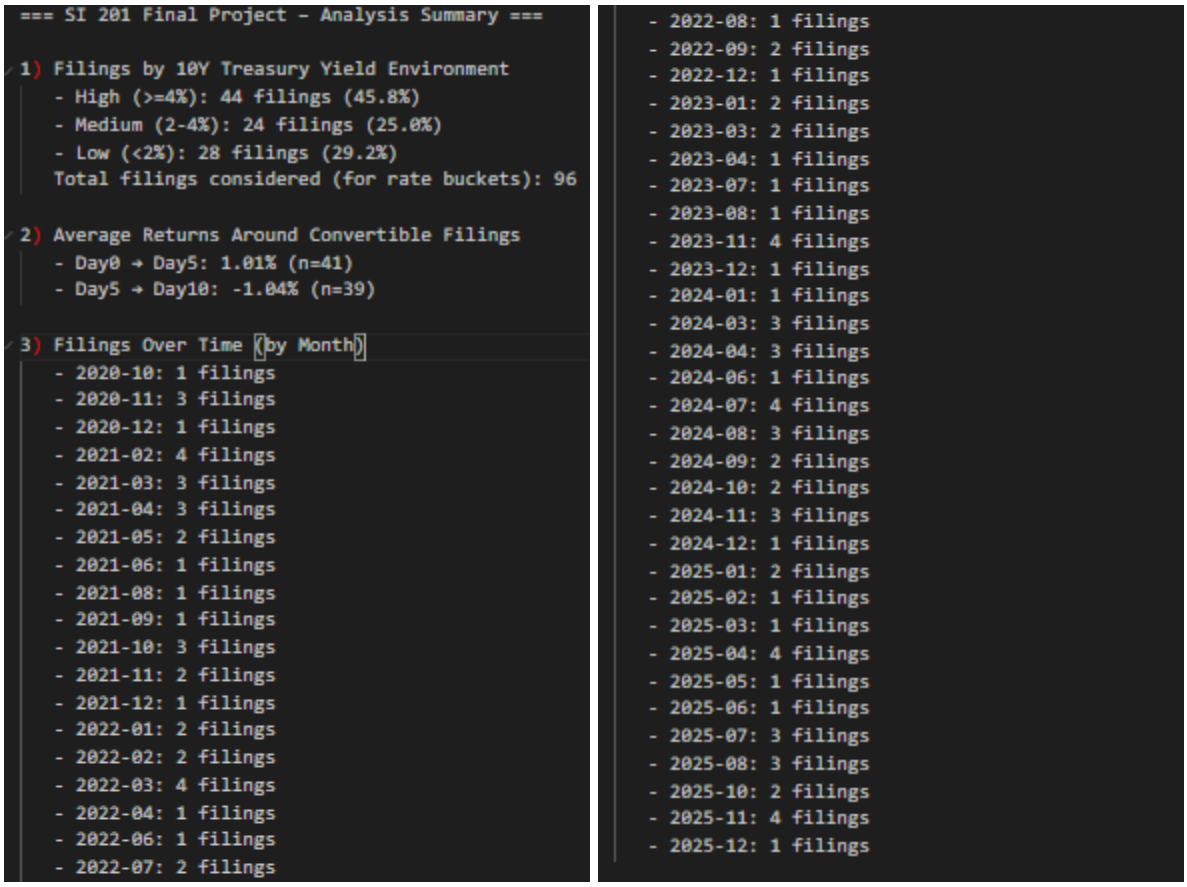
Data Alignment: Weekends and holidays caused discrepancies in 5-day and 10-day return windows. This had to be properly mitigated so as not to crash our code if our API couldn't return a certain value.

API Limitations and PyCache: There are very few free stock price APIs with usable request limits. The only one we found allowed just 100 requests per day and required querying by the company. Repeated function calls often returned the same data due to caching, which required us to implement PyCache to help us achieve the 100 data points per table without repetition.

Outliers: Missing data points and extreme return outliers had to be filtered to prevent distortion in calculations.

Stock-Price API Data: Despite being able to pull 100 8-K filings, companies, and interest rates, some of these were private. Therefore, we couldn't grab their stock price data, making us unable to fulfill the 100-row requirement for each API. However, this was essentially resolved through the addition of 0 as the null value, and then our request to ignore all zeros in our median calculations.

4. The calculations from the data in the database (i.e., a screenshot) (10 points)



```
=== SI 201 Final Project - Analysis Summary ===

1) Filings by 10Y Treasury Yield Environment
   - High (>=4%): 44 filings (45.8%)
   - Medium (2-4%): 24 filings (25.0%)
   - Low (<2%): 28 filings (29.2%)
   Total filings considered (for rate buckets): 96

2) Average Returns Around Convertible Filings
   - Day0 → Day5: 1.01% (n=41)
   - Day5 → Day10: -1.04% (n=39)

3) Filings Over Time (by Month)
   - 2020-10: 1 filings
   - 2020-11: 3 filings
   - 2020-12: 1 filings
   - 2021-02: 4 filings
   - 2021-03: 3 filings
   - 2021-04: 3 filings
   - 2021-05: 2 filings
   - 2021-06: 1 filings
   - 2021-08: 1 filings
   - 2021-09: 1 filings
   - 2021-10: 3 filings
   - 2021-11: 2 filings
   - 2021-12: 1 filings
   - 2022-01: 2 filings
   - 2022-02: 2 filings
   - 2022-03: 4 filings
   - 2022-04: 1 filings
   - 2022-06: 1 filings
   - 2022-07: 2 filings
   - 2022-08: 1 filings
   - 2022-09: 2 filings
   - 2022-12: 1 filings
   - 2023-01: 2 filings
   - 2023-03: 2 filings
   - 2023-04: 1 filings
   - 2023-07: 1 filings
   - 2023-08: 1 filings
   - 2023-11: 4 filings
   - 2023-12: 1 filings
   - 2024-01: 1 filings
   - 2024-03: 3 filings
   - 2024-04: 3 filings
   - 2024-06: 1 filings
   - 2024-07: 4 filings
   - 2024-08: 3 filings
   - 2024-09: 2 filings
   - 2024-10: 2 filings
   - 2024-11: 3 filings
   - 2024-12: 1 filings
   - 2025-01: 2 filings
   - 2025-02: 1 filings
   - 2025-03: 1 filings
   - 2025-04: 4 filings
   - 2025-05: 1 filings
   - 2025-06: 1 filings
   - 2025-07: 3 filings
   - 2025-08: 3 filings
   - 2025-10: 2 filings
   - 2025-11: 4 filings
   - 2025-12: 1 filings

4) Distribution of Short-Term Returns (Day0, Day5)
   - Number of observations: 44
   - Median return: -0.56%
   - Range: -37.34% to 147.66%

5) Relationship Between Interest Rates and Returns
   - Number of paired observations: 43
   - Median 10Y Treasury yield: 4.11%
   - Median Day0, Day5 return: -0.83%
   - Scatter plot shows wide dispersion, suggesting no strong linear relationship.
```

The calculation functions are on the next page.

```python
from db import get_connection
from datetime import datetime
from collections import defaultdict
import sqlite3
import matplotlib.pyplot as plt
from statistics import median

def load_interest_rates():
    conn = get_connection()
    cur = conn.cursor()

    cur.execute("""
        SELECT date, treasury_10y
        FROM interest_rates
        WHERE treasury_10y IS NOT NULL
        ORDER BY date
    """)

    rows = cur.fetchall()
    conn.close()

    rates = []
    for date_str, value in rows:

        try:
            d = datetime.fromisoformat(date_str).date()
            rates.append((d, float(value)))

        except Exception:
            continue

    return rates
```

```python
def get_latest_rate_on_or_before(target_date, rates):
    latest = None

    for d, r in rates:

        if d <= target_date:
            latest = r

        else:
            # Because rates are sorted ascending, once d > target_date we can stop
            break

    return latest
```

```python
def calculate_filings_by_rate_bucket():
    rates = load_interest_rates()

    if not rates:
        print("No interest-rate data found in DB.")
        return {}

    conn = get_connection()
    cur = conn.cursor()

    cur.execute("SELECT id, filing_date FROM filings")

    filings = cur.fetchall()

    conn.close()

    buckets = defaultdict(int)

    for filing_id, filing_date_str in filings:

        try:
            filing_date = datetime.fromisoformat(filing_date_str).date()

        except Exception:
            # Skip weird dates
            continue

        rate = get_latest_rate_on_or_before(filing_date, rates)

        if rate is None:
            # No rate available on or before this date
            continue

        if rate < 2.0:
            bucket = "Low (<2%)"
        elif rate < 4.0:
            bucket = "Medium (2-4%)"
        else:
            bucket = "High (>=4%)"

        buckets[bucket] += 1

    return dict(buckets)
```

```python
# FILINGS OVER TIME
def calculate_filings_per_month():
    conn = get_connection()
    cur = conn.cursor()

    cur.execute("""
        SELECT substr(filing_date, 1, 7) AS ym,
               COUNT(*)
        FROM filings
        GROUP BY ym
        ORDER BY ym
    """)

    rows = cur.fetchall()
    conn.close()

    # Filter out any None/empty ym
    return [(ym, count) for ym, count in rows if ym]
```

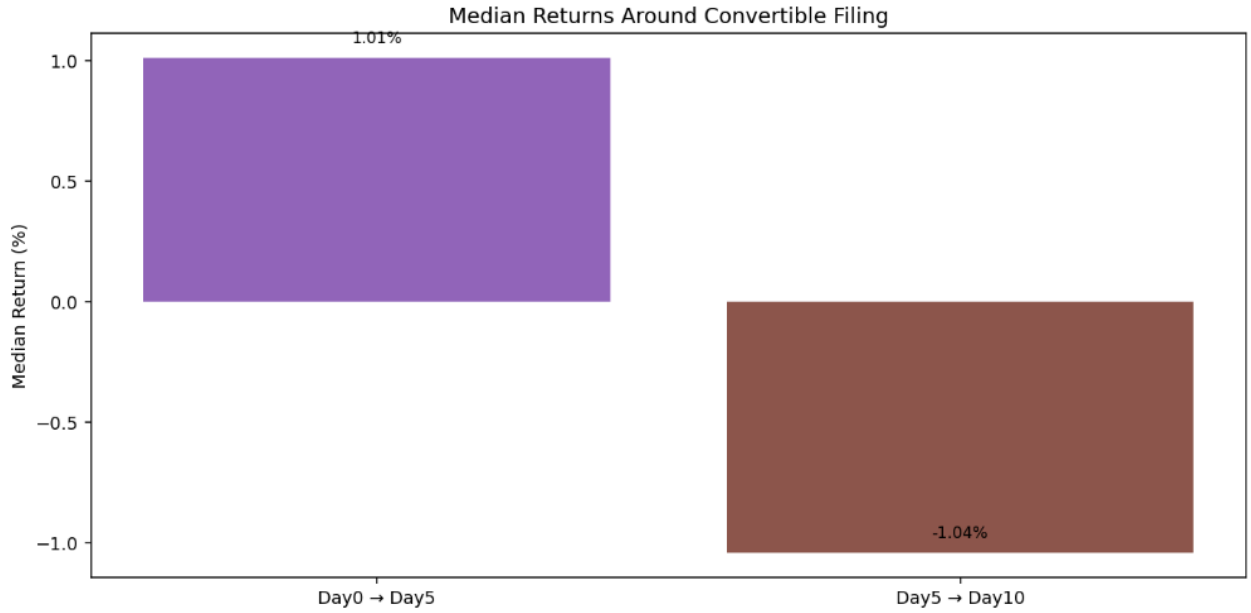
```python
# AVERAGE RETURNS FOR DAY0 to 5 and 5 to 10

def load_compact_stock_returns():
    conn = get_connection()
    cur = conn.cursor()

    cur.execute("""
        SELECT c.ticker, r.filing_date, r.return_day0_to_day5, r.return_day5_to_day10
        FROM stock_returns r
        JOIN companies c ON r.company_id = c.id
    """)

    rows = cur.fetchall()
    conn.close()

    results = []
    for ticker, filing_date, r0_5, r5_10 in rows:

        try:
            r0_5_val = float(r0_5) if r0_5 is not None else None

        except Exception:
            r0_5_val = None

        try:
            r5_10_val = float(r5_10) if r5_10 is not None else None

        except Exception:
            r5_10_val = None

        results.append((ticker, filing_date, r0_5_val, r5_10_val))

    return results
```

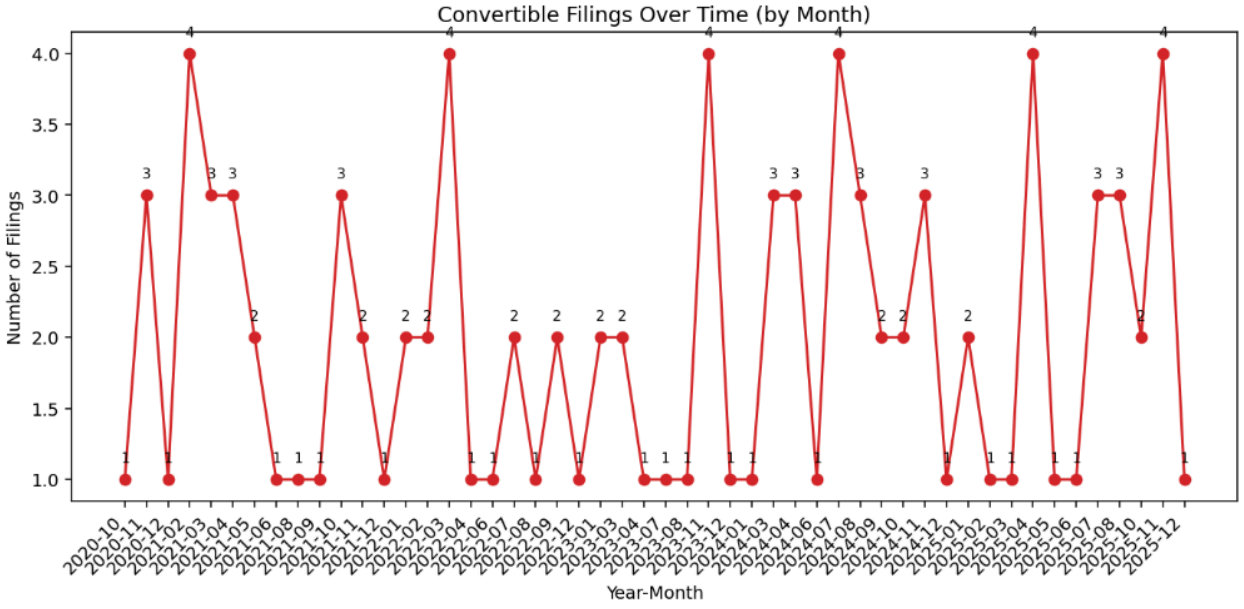
```python
def calculate_avg_returns():
    rows = load_compact_stock_returns()
    r0_5_list = []
    r5_10_list = []

    for _, _, r0_5, r5_10 in rows:

        if r0_5 not in (None, 0.0):
            r0_5_list.append(r0_5)

        if r5_10 not in (None, 0.0):
            r5_10_list.append(r5_10)


    avg0_5 = median(r0_5_list) if r0_5_list else None
    avg5_10 = median(r5_10_list) if r5_10_list else None

    return {
        "avg_day0_5": avg0_5,
        "avg_day5_10": avg5_10,
        "count_day0_5": len(r0_5_list),
        "count_day5_10": len(r5_10_list)
    }
```

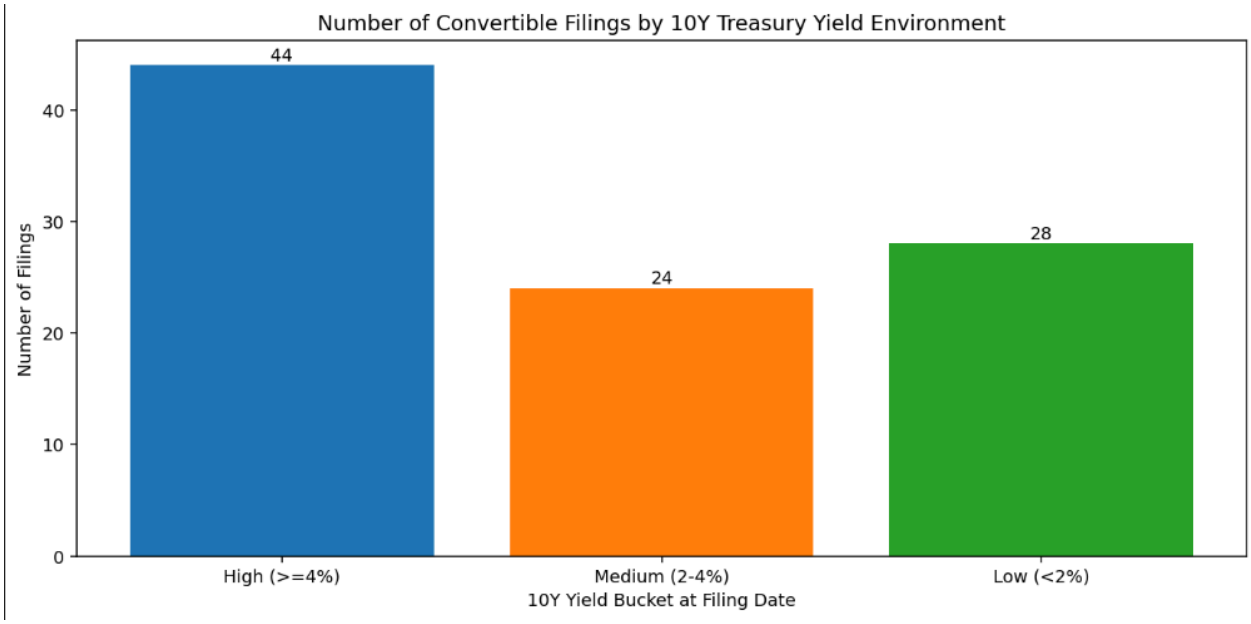
```python
def calculate_return_distribution():
    rows = load_compact_stock_returns()
    values = []

    for _, _, r0_5, _ in rows:
        if r0_5 not in (None, 0.0):
            values.append(r0_5)

    return values
```

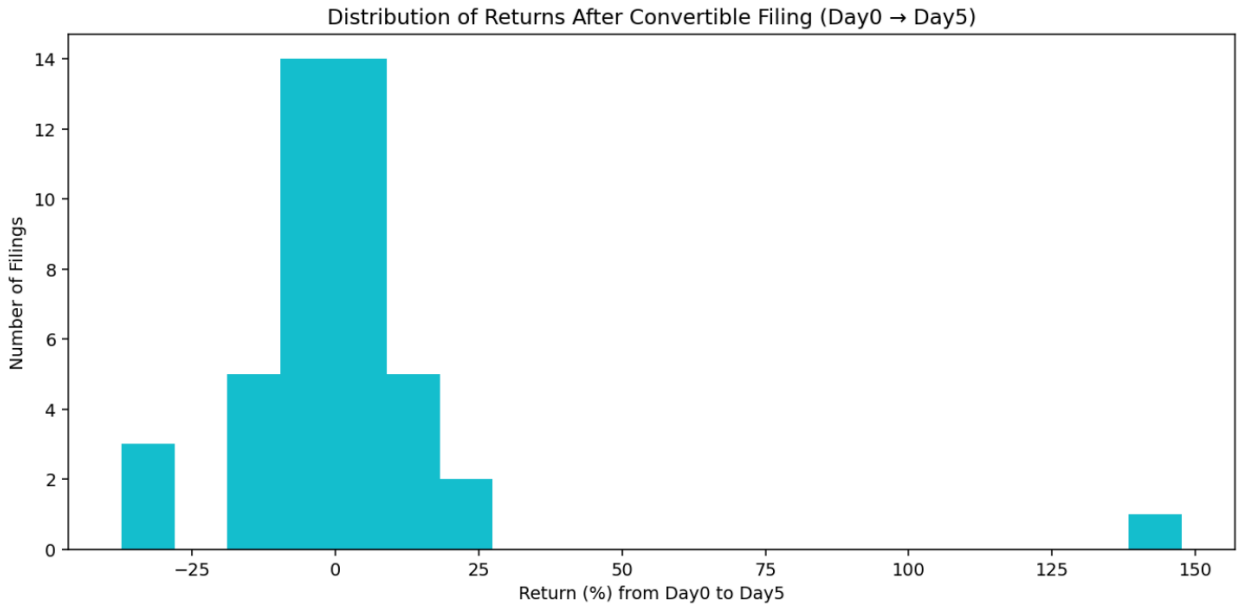
```python
def calculate_yield_vs_return():
    # Load historical 10Y Treasury yield data
    rates = load_interest_rates()

    if not rates:
        print("No interest-rate data found in DB.")
        return []

    # Load compact stock return data
    rows = load_compact_stock_returns()
    points = []

    for _, filing_date_str, r0_5, _ in rows:
        try:
            # Convert filing date string to date object
            filing_date = datetime.fromisoformat(filing_date_str).date()
        except Exception:
            continue

        rate = get_latest_rate_on_or_before(filing_date, rates)
        if rate is None:
            continue

        if r0_5 not in (None, 0.0):
            points.append((rate, r0_5))

    return points
```

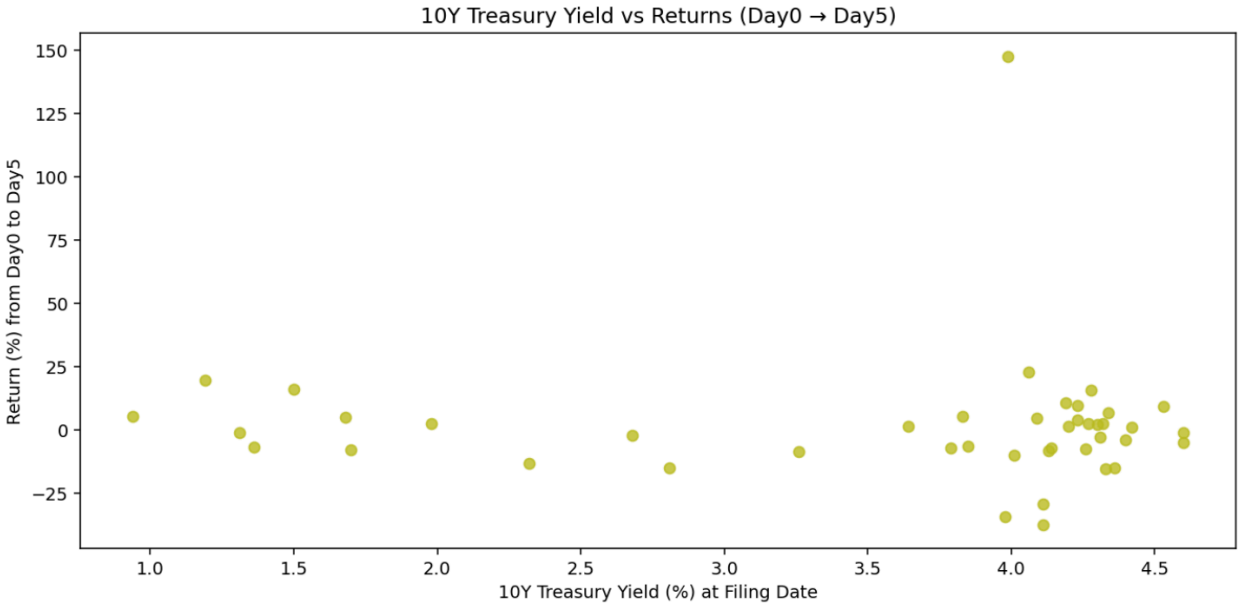
5. The visualization that you created (i.e. screen shot or image file) (10 points)






Number of Convertible Filings by 10Y Treasury Yield Environment
Number of Filings
0
10
20
30
40
44
24
28
High (>=4%)
Medium (2-4%)
Low (<2%)
10Y Yield Bucket at Filing Date


Distribution of Returns After Convertible Filing (Day0 → Day5)
Number of Filings
0
2
4
6
8
10
12
14
−25
0
25
50
75
100
125
150
Return (%) from Day0 to Day5

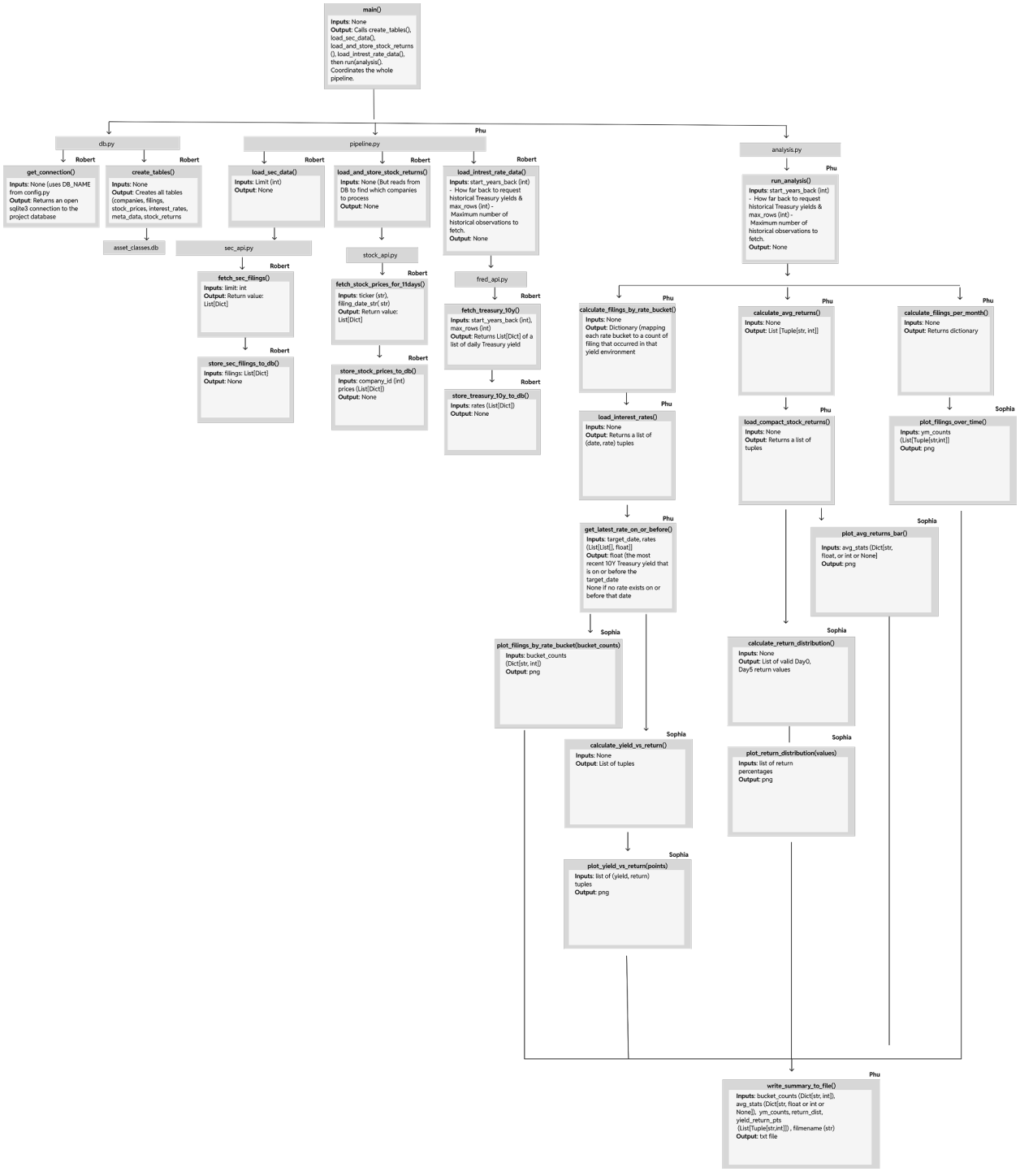

10Y Treasury Yield vs Returns (Day0 → Day5)
Return (%) from Day0 to Day5
150
125
100
75
50
25
0
−25
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
10Y Treasury Yield (%) at Filing Date

6. Instructions for running your code (10 points)

All you need to do to run our code is to run main.py.

Main.py starts by using the function create_tables() in the db.py, which creates the SQL framework. Afterwards, it runs load_sec_data() with a limit of 25. This function itself runs other functions, such as fetch_sec_filings() and store_sec_filings_to_db(). Afterwards, the code runs load_and_store_stock_returns(). This function retrieves each company's stock prices around its earliest filing date, calculates the percentage returns from Day 0 to Day 5 and Day 5 to Day 10, and stores or updates these returns in the stock_returns table. And then lastly, load_interest_rate_data() runs, fetching the interest rates and storing them in our database. Main then runs run_analysis(), which calls all of our functions in that file to gather the data from the database and generate the graphics and the summary text file.

7. An updated function diagram with the names of each function, the input, and output and who was responsible for that function (20 points)



8. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
Throughout project	Debugging Python code across multiple files (excluding config.py). We also used assistance for creating and debugging our database structure in db.py, understanding Python features such as pycache, adding debugging print statements, and writing SQL commands such as INSERT OR IGNORE and conflict-handling inserts.	ChatGPT (explanations & correcting code snippets)	Yes. Provided fixes and clarified logic.
11/23/25	During early development, we needed help understanding the flow of our program, adjusting API data usage, and generating initial graphs. We also needed help understanding Python packages such as datetime and timedelta.	ChatGPT (package explanations, logic clarification)	Yes. ChatGPT helped us understand how date-related packages work.
11/30/25	We encountered an issue where SEC filings were not	ChatGPT (debugging logic, function restructuring advice)	Yes. ChatGPT helped clarify how to modify our fetching logic.

	correctly mapped one-to-one with companies. We needed help understanding how to restructure our fetching logic so each filing corresponded to a single company.		
12/05/25	We experienced major issues with missing stock data and exceeding free API limits. We were unsure how to limit API calls while still retrieving enough data for our database, given request constraints.	ChatGPT (API rate-limiting strategies and logic suggestions)	Yes. ChatGPT helped us implement API call limiting logic.
12/07/25	Our stock_returns table was missing rows due to private companies lacking stock data. We needed help determining whether placeholder values were acceptable and how to insert those values without breaking the database.	ChatGPT (conditional logic, date methods)	Yes. ChatGPT helped us insert placeholder values (0.0) for private companies, helping us properly achieve 100 entries per database despite query limitations.
12/08/25	We discovered recurring strings in the filings table that were unnecessary and not used elsewhere in the project. We were unsure how to remove them safely.	Grading Session (Daniel, in-person assistance)	Yes. We were able to remove the unnecessary column and clean up the database.
12/15/25	We added two	ChatGPT	Yes. ChatGPT helped

	additional visualizations and needed help interpreting and summarizing the resulting summary statistics, which were more complex than previous graphs.	(visualization guidance and assistance writing analysis summaries)	us complete the new visualizations and successfully incorporate the new information into the analysis summary.