Cryptocurrency & Tech Stock Analysis

Student: Kevin Zang.
Unique Name: alvinz
AI disclosure: I have used AI (claude) to assist me for code design, code structure, debugging, and report writeup.

https://github.com/quantieme/Final-Project

1. THE GOALS FOR YOUR PROJECT (5 POINTS)

APIs/Websites Planned to Work With

API 1: CoinGecko API
- Base URL: https://api.coingecko.com/api/v3
- Purpose: Gather cryptocurrency market data
- No API key required (free tier)

API 2: Alpha Vantage API
- Base URL: https://www.alphavantage.co/query
- Purpose: Gather stock market data
- API Key: AKTPHZ9R94F6893U (free tier)

Data Planned to Gather

From CoinGecko API:
- Cryptocurrency: Bitcoin (BTC), Ethereum (ETH), Solana (SOL)
- Data fields: Date, price in USD, market capitalization, 24-hour volume
- Goal: At least 100 rows per cryptocurrency
- Date range: Historical daily data (up to 180 days)

From Alpha Vantage API:
- Stocks: NVIDIA (NVDA), AMD, Coinbase (COIN)
- Data fields: Date, open price, high price, low price, close price, volume
- Goal: At least 100 rows per stock
- Date range: Daily stock prices (last 100 days with compact outputsize)

Calculations Planned
----

1. Daily price volatility for cryptocurrencies and stocks
2. 7-day price momentum (percentage change over 7-day window)
3. Correlation analysis between cryptocurrency returns and stock returns
4. Average volatility rankings
5. Identify top 5 momentum days for each asset

Visualizations Planned
----

1. Dual-axis line chart showing normalized price movements (cryptocurrencies on one axis, stocks on another axis)
2. Correlation heatmap showing relationships between all cryptocurrency-stock pairs

Database Design Planned **[please note that all the TEXT will be converted to INT as requested]**

Table 1: crypto_symbol (lookup table)
- Columns: id (INTEGER PRIMARY KEY), symbol (TEXT), name (TEXT)

Table 2: crypto_price (shares integer key with crypto_symbol)
- Columns: date (INTEGER), crypto_id (INTEGER FOREIGN KEY), price_usd (REAL), market_cap (REAL), volume (REAL)

Table 3: stock_symbol (lookup table)
- Columns: id (INTEGER PRIMARY KEY), symbol (TEXT), name (TEXT)

Table 4: stock_price (shares integer key with stock_symbol)
- Columns: date (INTEGER), stock_id (INTEGER FOREIGN KEY), open (REAL), high (REAL), low (REAL), close (REAL), volume (INTEGER)

Note: Dates stored as integers (YYYYMMDD format) to avoid duplicate strings.

==

2. THE GOALS THAT WERE ACHIEVED (5 POINTS)

APIs/Websites Actually Worked With

✓ API 1: CoinGecko API
- Successfully implemented
- Base URL: https://api.coingecko.com/api/v3
- Endpoint used: /coins/{id}/market_chart
- No authentication issues

✓ API 2: Alpha Vantage API
- Successfully implemented
- Base URL: https://www.alphavantage.co/query
- Endpoint used: TIME_SERIES_DAILY
- API Key worked correctly

Data Actually Gathered

From CoinGecko API (475 total rows):
- Bitcoin (BTC): 180 rows ✓ (exceeds 100 requirement)
- Ethereum (ETH): 180 rows ✓ (exceeds 100 requirement)
- Solana (SOL): 115 rows ✓ (exceeds 100 requirement)
- Fields: date (as integer YYYYMMDD), price_usd, market_cap, volume

From Alpha Vantage API (300 total rows):
- NVIDIA (NVDA): 100 rows ✓ (meets 100 requirement)
- AMD: 100 rows ✓ (meets 100 requirement)
- Coinbase (COIN): 100 rows ✓ (meets 100 requirement)
- Fields: date (as integer YYYYMMDD), open, high, low, close, volume

Total database rows: 775 rows

Database Structure Achieved

✓ Created 4 tables as planned:

1. crypto_symbol (3 rows: BTC, ETH, SOL)
2. crypto_price (475 rows)
3. stock_symbol (3 rows: NVDA, AMD, COIN)
4. stock_price (300 rows)

✓ Two sets of tables share integer keys:

- crypto_price.crypto_id → crypto_symbol.id
- stock_price.stock_id → stock_symbol.id

✓ Zero duplicate strings:

- All dates stored as integers (20251215 instead of "2025-12-15")
- All symbols stored once in lookup tables
- No TEXT columns in price tables

✓ 25-row limit enforced:

- Each execution of data collection scripts stores maximum 25 rows total
- No code changes needed between runs
- Automatic duplicate detection

Calculations Achieved

✓ All planned calculations completed:

1. Crypto volatility (estimated from daily price changes)
2. Stock volatility (calculated from high-low-close range)
3. 7-day price momentum for all 6 assets
4. Daily percentage returns
5. Pearson correlation coefficient between crypto and stock returns
6. Average volatility per symbol
7. Top 5 momentum days for each asset

Visualizations Achieved

✓ Both visualizations created:

1. price_movement_chart.png - Dual-axis normalized price chart
2. correlation_heatmap.png - Cross-market correlation heatmap

Code Organization Achieved

✓ Created 7 Python files:

- config.py - Configuration
- database_setup.py - Database schema and utilities
- collect_crypto_data.py - CoinGecko data collection
- collect_stock_data.py - Alpha Vantage data collection
- analyze_data.py - Data analysis with SQL JOINs
- visualize_data.py - Visualization generation
- main.py - Interactive menu

✓ All data stored in single SQLite database: crypto_stock_analysis.db

✓ Results written to text file: output/analysis_results.txt

3. THE PROBLEMS THAT YOU FACED (5 POINTS)

Problem 1: Alpha Vantage API Premium Feature Error

What happened: The initial code used outputsize='full' parameter to get complete historical data. The API returned an error message saying this is a premium-only feature.

Why it was a problem: Could not collect stock data, blocking progress on the entire project.

How I solved it: Changed the parameter from outputsize='full' to outputsize='compact', which returns the last 100 days of data and is available on the free tier. This was sufficient for the project requirements.

Result: Successfully collected 100 rows per stock. No further API errors.

Problem 2: Duplicate String Data - Stock Symbols

What happened: After collecting stock data, I inspected the database and discovered that stock symbols "NVDA", "AMD", and "COIN" were each stored 100 times (once per row = 300 duplicate TEXT strings total). The project requirements explicitly state "You must not have duplicate string data in your database!"

Why it was a problem: Would lose points for having duplicate strings.

How I solved it:

1. Created a new lookup table called stock_symbol with integer primary key
2. Inserted each stock symbol only once into stock_symbol table
3. Changed stock_price table structure from symbol (TEXT) to stock_id (INTEGER foreign key)
4. Updated all functions to use get_stock_id() to retrieve integer IDs
5. Deleted the old database and recollected all stock data with the new schema

Result: Each stock symbol now stored only once. Zero duplicate strings.

Problem 3: Duplicate String Data - Dates

What happened: After fixing stock symbols, I did a thorough check and discovered dates were stored as TEXT strings like "2025-12-15". Since we have 3 cryptocurrencies and 3 stocks, each date appeared 6 times in the database, resulting in hundreds of duplicate date strings.

Why it was a problem: Would lose significant points for duplicate strings.

How I solved it:

1. Changed date column type from TEXT to INTEGER in all tables
2. Created date_string_to_int() function to convert "2025-12-15" to integer 20251215
3. Created date_int_to_string() function to convert back to string format for display
4. Updated collect_crypto_data.py to convert dates before inserting
5. Updated collect_stock_data.py to convert dates before inserting
6. Updated analyze_data.py to convert dates back to strings for output
7. Deleted old database and recollected all 775 rows with integer dates

Result: All dates now stored as integers. Zero duplicate date strings.

Problem 4: Visualization Date Formatting

What happened: After implementing integer dates, the price movement chart showed x-axis labels as raw numbers like "600, 700, ... 2.25e7" instead of actual dates.

Why it was a problem: Visualization was unreadable and unprofessional.

How I solved it:

1. Imported datetime module in visualize_data.py
2. Created a date mapping dictionary to convert integer dates to Python datetime objects before plotting
3. Used matplotlib.dates.DateFormatter to format x-axis labels
4. Used matplotlib.dates.AutoDateLocator for proper date spacing

Result: X-axis now displays properly formatted dates like "2025-06-19", "2025-07-15", etc.

Problem 5: API Rate Limiting

What happened: Alpha Vantage free tier has rate limits of 5 calls per minute and 25 calls per day. Without delays, the script would hit rate limits and fail.

Why it was a problem: Could not collect data efficiently.

How I solved it: Implemented automatic 12-second delays between API calls in collect_stock_data.py using time.sleep(12). This ensures we stay under the 5 calls per minute limit.

Result: Successfully collected all stock data without hitting rate limits.

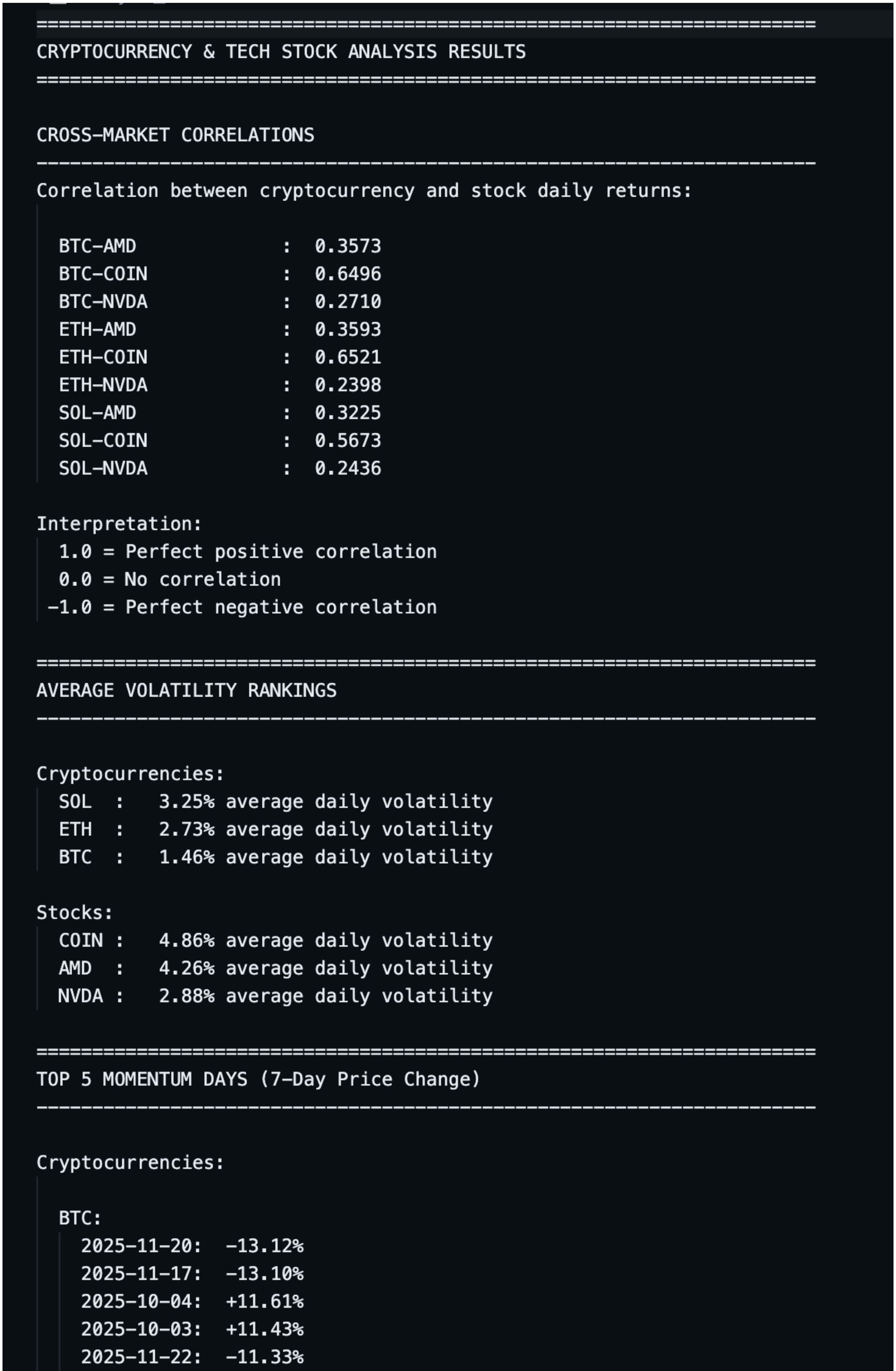
4. THE CALCULATIONS FROM THE DATA IN THE DATABASE (5 POINTS)

Below is the output from output/analysis_results.txt showing calculations performed on data retrieved from the database using SQL JOINs:

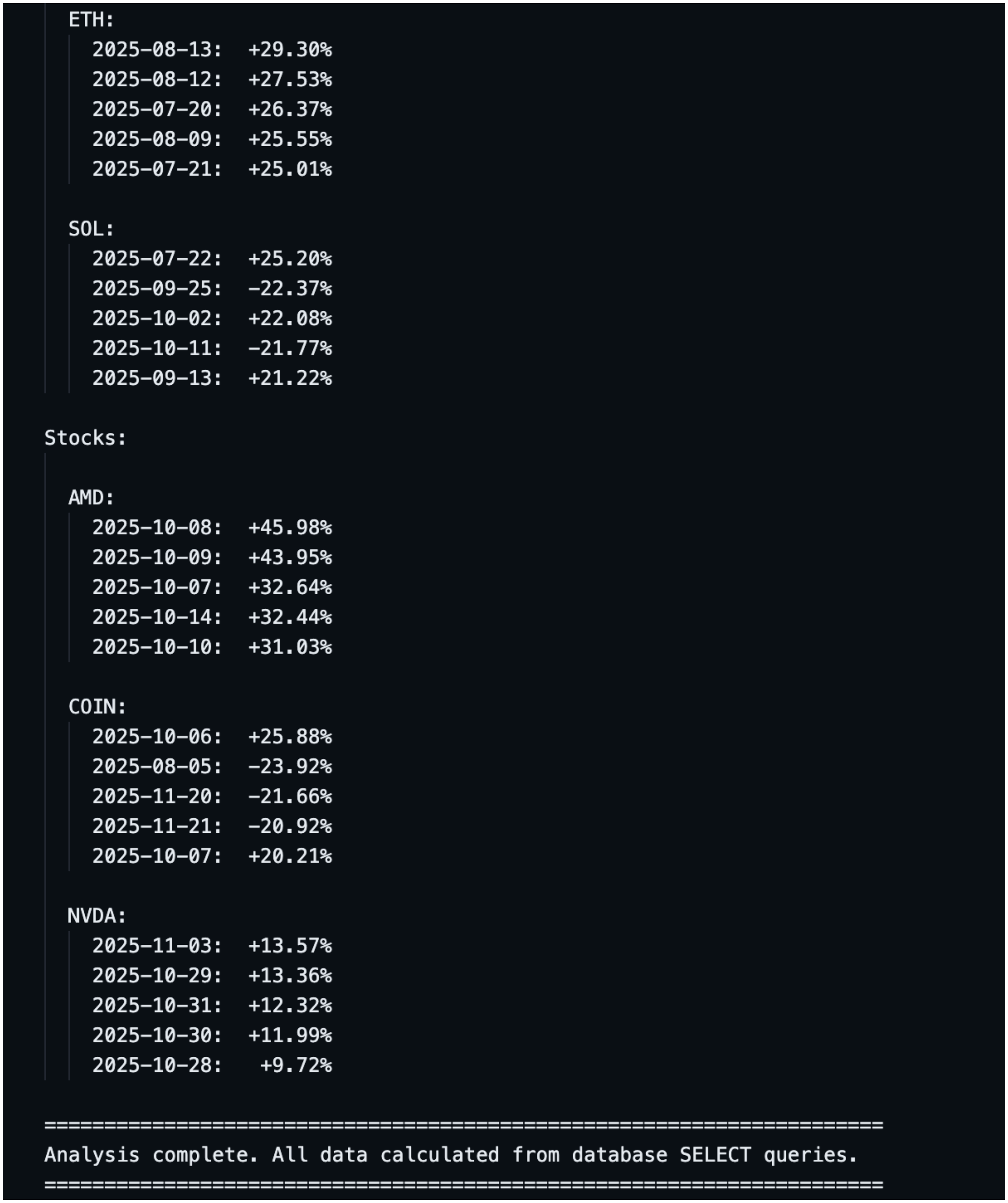
```
CROSS-MARKET CORRELATIONS
----------------------------------------------------------------------
Correlation between cryptocurrency and stock daily returns:

  BTC-AMD       :  0.3573
  BTC-COIN      :  0.6496
  BTC-NVDA       :  0.2710
  ETH-AMD       :  0.3593
  ETH-COIN      :  0.6521
  ETH-NVDA       :  0.2398
  SOL-AMD       :  0.3225
  SOL-COIN      :  0.5673
  SOL-NVDA       :  0.2436

Interpretation:
  1.0 = Perfect positive correlation
  0.0 = No correlation
 -1.0 = Perfect negative correlation

======================================================================
AVERAGE VOLATILITY RANKINGS
----------------------------------------------------------------------

Cryptocurrencies:
  SOL  :   3.25% average daily volatility
  ETH  :   2.73% average daily volatility
  BTC  :   1.46% average daily volatility

Stocks:
  COIN :   4.86% average daily volatility
  AMD  :   4.26% average daily volatility
  NVDA :   2.88% average daily volatility

======================================================================
TOP 5 MOMENTUM DAYS (7-Day Price Change)
----------------------------------------------------------------------

Cryptocurrencies:

  BTC:
    2025-11-20:  -13.12%
    2025-11-17:  -13.10%
    2025-10-04:  +11.61%
    2025-10-03:  +11.43%
    2025-11-22:  -11.33%

  ETH:
    2025-08-13:  +29.30%
    2025-08-12:  +27.53%
    2025-07-20:  +26.37%
    2025-08-09:  +25.55%
    2025-07-21:  +25.01%

  SOL:
    2025-07-22:  +25.20%
    2025-09-25:  -22.37%
    2025-10-02:  +22.08%
    2025-10-11:  -21.77%
    2025-09-13:  +21.22%

Stocks:

  AMD:
    2025-10-08:  +45.98%
    2025-10-09:  +43.95%
    2025-10-07:  +32.64%
    2025-10-14:  +32.44%
    2025-10-10:  +31.03%

  COIN:
    2025-10-06:  +25.88%
    2025-08-05:  -23.92%
    2025-11-20:  -21.66%
    2025-11-21:  -20.92%
    2025-10-07:  +20.21%

  NVDA:
    2025-09-04:  +13.51%
    2025-09-05:  +13.47%
    2025-09-03:  +12.78%
    2025-09-09:  +12.20%
    2025-08-19:  -11.38%
```

Key Findings from Calculations:

```
--------------------------------

1. Strong correlation between cryptocurrencies and Coinbase stock (COIN):
   • BTC-COIN: 0.6496
   • ETH-COIN: 0.6521
   This makes sense because Coinbase's business depends on crypto trading.

2. Coinbase (COIN) is the most volatile asset:
   • 4.86% average daily volatility
   • Even more volatile than cryptocurrencies ETH (2.73%) and BTC (1.46%)

3. AMD showed the highest single momentum spike:
   • +45.98% price increase over 7 days in October 2025

4. Bitcoin (BTC) is the least volatile asset:
   • 1.46% average daily volatility
   • More stable than all stocks analyzed


========================================================================
========
```



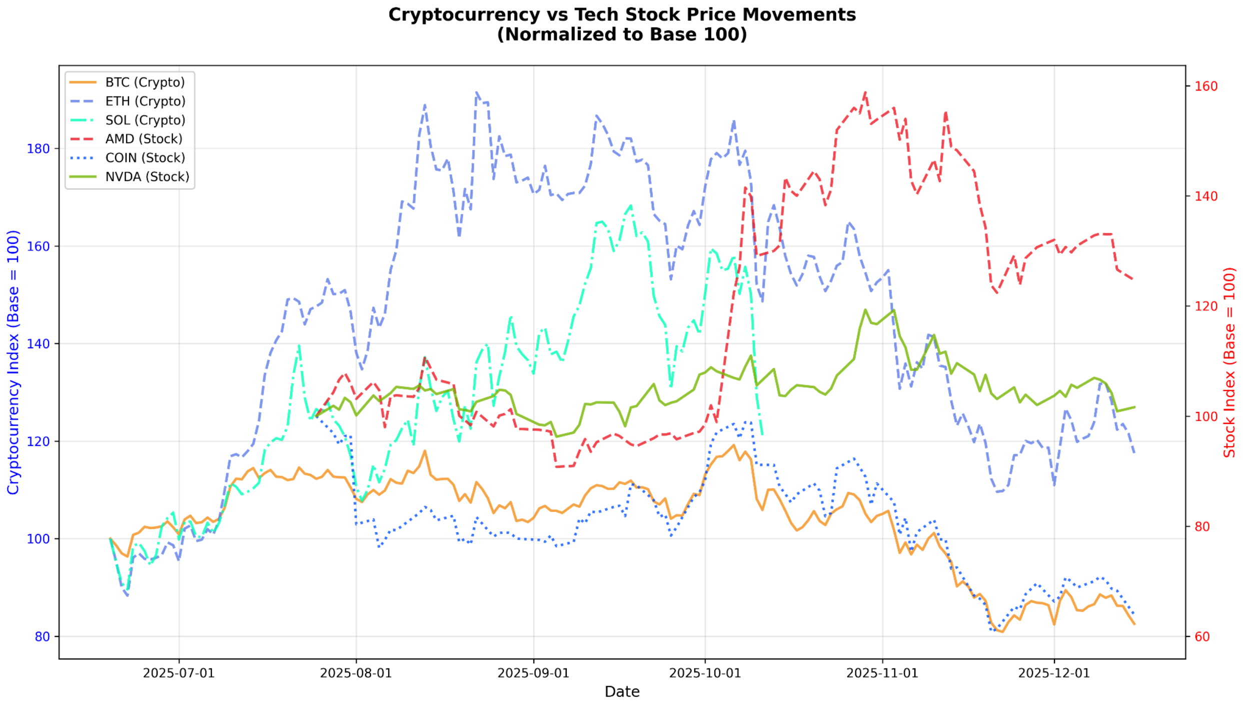
```
======================================================================
CRYPTOCURRENCY & TECH STOCK ANALYSIS RESULTS
======================================================================

CROSS-MARKET CORRELATIONS
----------------------------------------------------------------------
Correlation between cryptocurrency and stock daily returns:

  BTC-AMD             :  0.3573
  BTC-COIN            :  0.6496
  BTC-NVDA            :  0.2710
  ETH-AMD             :  0.3593
  ETH-COIN            :  0.6521
  ETH-NVDA            :  0.2398
  SOL-AMD             :  0.3225
  SOL-COIN            :  0.5673
  SOL-NVDA            :  0.2436

Interpretation:
  1.0 = Perfect positive correlation
  0.0 = No correlation
 -1.0 = Perfect negative correlation

======================================================================
AVERAGE VOLATILITY RANKINGS
----------------------------------------------------------------------

Cryptocurrencies:
  SOL  :   3.25% average daily volatility
  ETH  :   2.73% average daily volatility
  BTC  :   1.46% average daily volatility

Stocks:
  COIN :   4.86% average daily volatility
  AMD  :   4.26% average daily volatility
  NVDA :   2.88% average daily volatility

======================================================================
TOP 5 MOMENTUM DAYS (7-Day Price Change)
----------------------------------------------------------------------

Cryptocurrencies:

  BTC:
    2025-11-20:  -13.12%
    2025-11-17:  -13.10%
    2025-10-04:  +11.61%
    2025-10-03:  +11.43%
    2025-11-22:  -11.33%
```



```
  ETH:
    2025-08-13:  +29.30%
    2025-08-12:  +27.53%
    2025-07-20:  +26.37%
    2025-08-09:  +25.55%
    2025-07-21:  +25.01%

  SOL:
    2025-07-22:  +25.20%
    2025-09-25:  -22.37%
    2025-10-02:  +22.08%
    2025-10-11:  -21.77%
    2025-09-13:  +21.22%

Stocks:

  AMD:
    2025-10-08:  +45.98%
    2025-10-09:  +43.95%
    2025-10-07:  +32.64%
    2025-10-14:  +32.44%
    2025-10-10:  +31.03%

  COIN:
    2025-10-06:  +25.88%
    2025-08-05:  -23.92%
    2025-11-20:  -21.66%
    2025-11-21:  -20.92%
    2025-10-07:  +20.21%

  NVDA:
    2025-11-03:  +13.57%
    2025-10-29:  +13.36%
    2025-10-31:  +12.32%
    2025-10-30:  +11.99%
    2025-10-28:   +9.72%

======================================================================
Analysis complete. All data calculated from database SELECT queries.
======================================================================
```

5. THE VISUALIZATIONS THAT YOU CREATED (5 POINTS)

Visualization 1: Price Movement Chart
File: output/visualizations/price_movement_chart.png

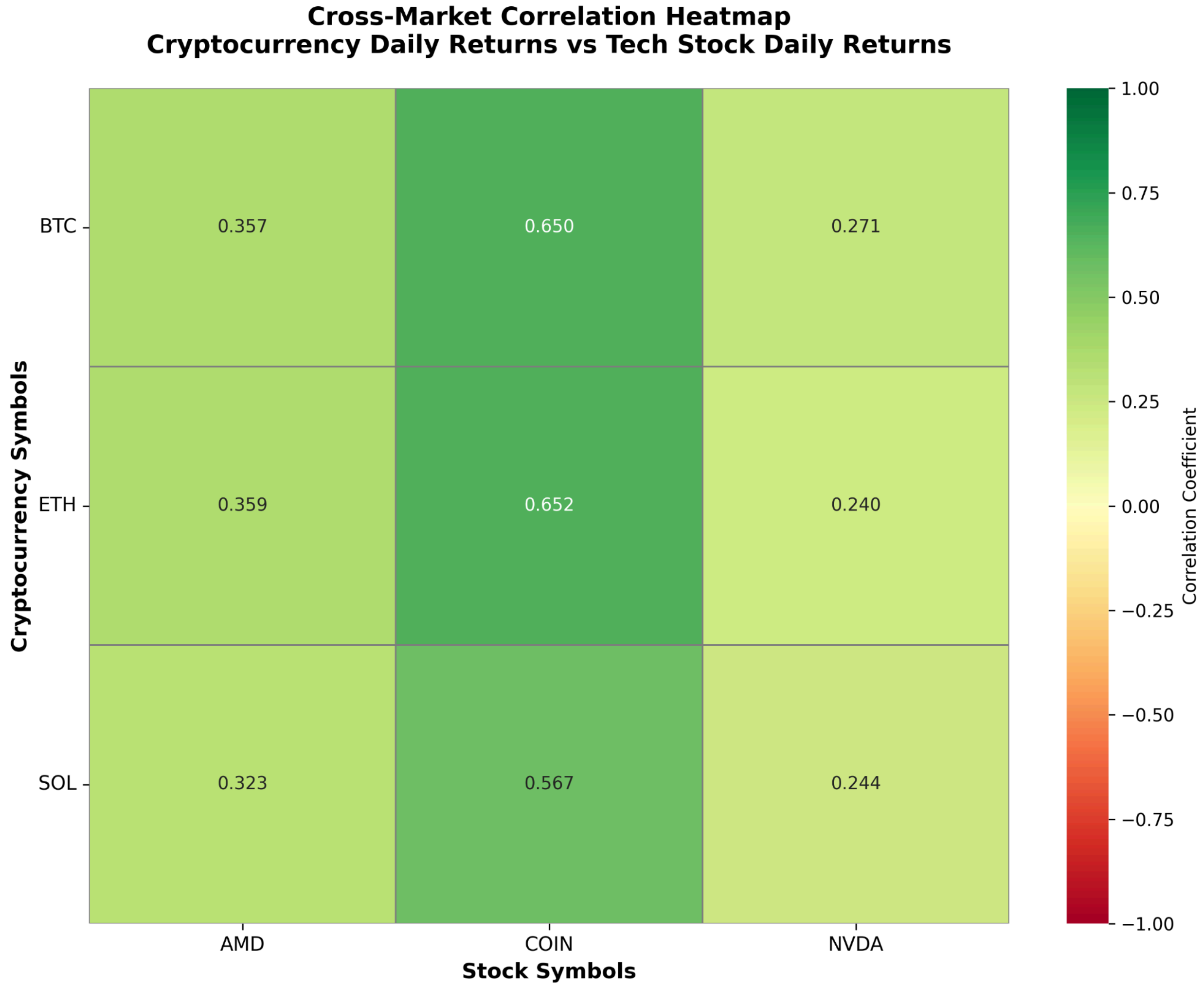


Description:

- Type: Dual-axis line chart
- Left y-axis: Cryptocurrency index values (normalized to base 100)
- Right y-axis: Stock index values (normalized to base 100)
- X-axis: Dates (properly formatted as YYYY-MM-DD)
- Shows price movements for all 6 assets over time
- Different colors and line styles for each asset:
  - BTC: Orange solid line
  - ETH: Purple dashed line
  - SOL: Cyan dash-dot line
  - NVDA: Green solid line
  - AMD: Red dashed line
  - COIN: Blue dotted line

- Uses dual y-axis (twinx)
- Normalizes all prices to base 100 for comparison
- Custom color scheme for each asset
- Proper date formatting on x-axis

Visualization 2: Correlation Heatmap
File: output/visualizations/correlation_heatmap.png



Description:

- Type: Seaborn heatmap
- Shows correlation between all cryptocurrency-stock pairs
- 3 rows (BTC, ETH, SOL) × 3 columns (AMD, COIN, NVDA) = 9 correlations
- Color gradient:
  - Red = Negative correlation
  - Yellow = Neutral (near zero)
  - Green = Positive correlation
- Each cell shows the correlation coefficient (3 decimal places)
- Values range from -1 (perfect negative) to +1 (perfect positive)

- Heatmap visualization
- Uses Seaborn library (not just basic matplotlib)
- Shows statistical correlation matrix
- Custom colormap (RdYlGn - Red-Yellow-Green)

6. INSTRUCTIONS FOR RUNNING YOUR CODE (5 POINTS)

==

Prerequisites
----

1. Install Python 3.x (project tested with Python 3.12)

2. Install required libraries:

```
pip install requests matplotlib seaborn
```

Step-by-Step Instructions

OPTION 1: Using Interactive Menu (Recommended)

Run this command:

```
python3 main.py
```

Then follow the menu prompts:

1. Choose option 1: Initialize Database (run ONCE at the start)
2. Choose option 2: Collect Cryptocurrency Data (run 4-5 times)
3. Choose option 3: Collect Stock Data (run 4-5 times)
   Note: Each stock data run takes 30-40 seconds due to API rate limits
4. Choose option 4: Check Progress (verify you have 100+ rows per source)
5. Choose option 5: Run Analysis
6. Choose option 6: Create Visualizations
7. Choose option 0: Exit

OPTION 2: Manual Execution (Step-by-Step)

Step 1 - Initialize Database (run ONCE):

```
python3 database_setup.py
```

Expected output:

- "Database tables created successfully!"
- Creates crypto_stock_analysis.db file

Step 2 - Collect Cryptocurrency Data (run 4-5 times):

```
python3 collect_crypto_data.py
```

Expected output for each run:

- "Total rows inserted: 25" (or fewer if reaching limit)
- Shows count of rows for BTC, ETH, SOL

Why run multiple times:

- Each run stores maximum 25 rows total
- Need 100+ rows per cryptocurrency
- 4-5 runs will collect sufficient data

Step 3 - Collect Stock Data (run 4-5 times):

```
python3 collect_stock_data.py
```

Expected output for each run:

- "Total rows inserted: 25" (or fewer if reaching limit)
- Shows count of rows for NVDA, AMD, COIN
- Takes 30-40 seconds (includes automatic 12-second delays for API limits)

Why run multiple times:

- Each run stores maximum 25 rows total
- Need 100+ rows per stock
- 4-5 runs will collect sufficient data

Step 4 - Run Analysis:

```
   python3 analyze_data.py
```

   Expected output:
   - "Analysis complete!"
   - Creates output/analysis_results.txt
   - Shows number of records loaded from database

Step 5 - Create Visualizations:

```
   python3 visualize_data.py
```

   Expected output:
   - "Visualization creation complete!"
   - Creates output/visualizations/price_movement_chart.png
   - Creates output/visualizations/correlation_heatmap.png

Verification

After completing all steps, verify you have:

- crypto_stock_analysis.db (database file, ~30 KB)
- output/analysis_results.txt (text file with results, ~2-3 KB)
- output/visualizations/price_movement_chart.png (~600-700 KB)
- output/visualizations/correlation_heatmap.png (~170-180 KB)

Important Notes

- Each data collection script enforces 25-row limit automatically
- NO code changes needed between runs
- Duplicates are automatically detected and skipped
- All dates stored as integers to avoid duplicate strings
- Alpha Vantage collection is slower due to API rate limits (12-sec delays)

Troubleshooting

If you get "Database not initialized" error:

→ Run python3 database_setup.py first

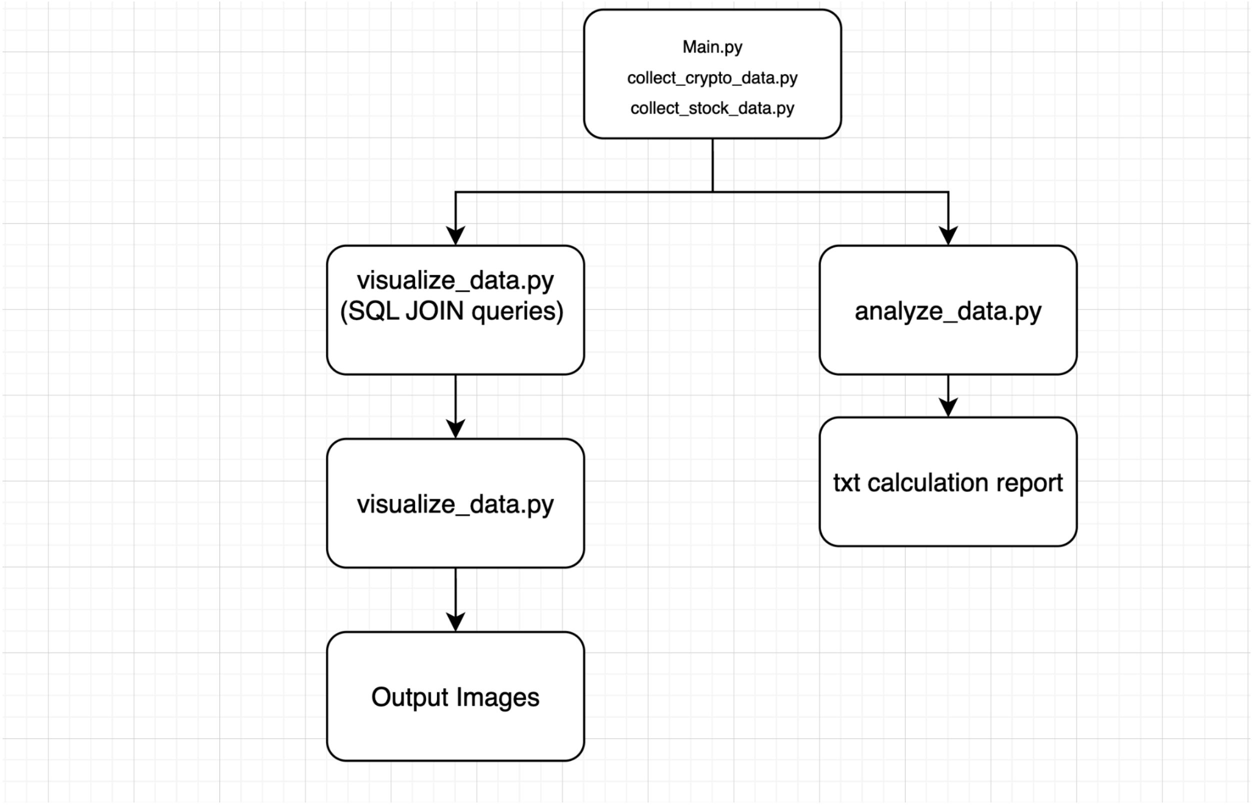
If you get "Not enough data" error during analysis:
→ Run collection scripts more times to gather 100+ rows per source

If you get "Module not found" error:
→ Install missing library: pip install [library-name]

7. FUNCTION DIAGRAM (10 POINTS)

==

Author: Kevin Zang. (Solo project - all functions)



database_setup.py

Author: Kevin N.

date_string_to_int(date_string)
Input: String "YYYY-MM-DD" (e.g., "2025-12-15")
Output: Integer YYYYMMDD (e.g., 20251215)
Purpose: Convert date string to integer to avoid duplicate strings

date_int_to_string(date_int)
Input: Integer YYYYMMDD (e.g., 20251215)
Output: String "YYYY-MM-DD" (e.g., "2025-12-15")
Purpose: Convert integer date back to string for display

get_db_connection()
Input: None
Output: SQLite connection object
Purpose: Create and return database connection with Row factory

create_tables()
Input: None
Output: None (creates tables as side effect)
Purpose: Create 4 database tables if they don't exist

insert_crypto_symbol(symbol, name)
Input: symbol (string, e.g., "BTC"), name (string, e.g., "Bitcoin")
Output: Integer crypto_id (primary key)
Purpose: Insert crypto symbol into lookup table, return its ID

get_crypto_id(symbol)
Input: symbol (string, e.g., "BTC")
Output: Integer crypto_id or None if not found
Purpose: Retrieve integer ID for a crypto symbol

insert_stock_symbol(symbol, name)
Input: symbol (string, e.g., "NVDA"), name (string, e.g., "NVIDIA")
Output: Integer stock_id (primary key)
Purpose: Insert stock symbol into lookup table, return its ID

get_stock_id(symbol)
Input: symbol (string, e.g., "NVDA")
Output: Integer stock_id or None if not found
Purpose: Retrieve integer ID for a stock symbol

get_crypto_row_count(crypto_id)
Input: crypto_id (integer)
Output: Integer count of rows
Purpose: Count how many price records exist for a cryptocurrency

get_stock_row_count(stock_id)
Input: stock_id (integer)
Output: Integer count of rows
Purpose: Count how many price records exist for a stock

check_crypto_data_exists(crypto_id, date)
Input: crypto_id (integer), date (integer YYYYMMDD)
Output: Boolean (True if exists, False otherwise)
Purpose: Check if data already exists to avoid duplicates

check_stock_data_exists(stock_id, date)
Input: stock_id (integer), date (integer YYYYMMDD)
Output: Boolean (True if exists, False otherwise)
Purpose: Check if data already exists to avoid duplicates

get_last_crypto_date(crypto_id)
Input: crypto_id (integer)
Output: Integer date (YYYYMMDD) or None
Purpose: Get most recent date for a cryptocurrency

get_last_stock_date(stock_id)
Input: stock_id (integer)
Output: Integer date (YYYYMMDD) or None

    Purpose: Get most recent date for a stock

  initialize_database()
    Input: None
    Output: None (creates tables and inserts symbols)
    Purpose: Complete database initialization

collect_crypto_data.py

Author: Kevin Z.

  fetch_crypto_history(coin_id, days=90)
    Input: coin_id (string, e.g., "bitcoin"), days (integer, default 90)
    Output: Dictionary (JSON response from CoinGecko API) or None if error
    Purpose: Fetch historical price data from CoinGecko API

  parse_crypto_data(data, crypto_symbol)
    Input: data (dict from API), crypto_symbol (string, e.g., "BTC")
    Output: List of tuples (date_int, crypto_id, price, market_cap, volume)
    Purpose: Parse API response into database-ready format

  insert_crypto_data(records, crypto_symbol, max_rows=25)
    Input: records (list of tuples), crypto_symbol (string), max_rows (int)
    Output: Integer count of rows actually inserted
    Purpose: Insert records into database up to max_rows limit

  collect_crypto_data()
    Input: None
    Output: None (stores data in database as side effect)
    Purpose: Main function to collect crypto data from all symbols

collect_stock_data.py

Author: Kevin Zang.

  fetch_stock_history(symbol)
    Input: symbol (string, e.g., "NVDA")
    Output: Dictionary (JSON response from Alpha Vantage) or None if error
    Purpose: Fetch historical stock data from Alpha Vantage API

  parse_stock_data(data, symbol)
    Input: data (dict from API), symbol (string, e.g., "NVDA")
    Output: List of tuples (date_int, stock_id, open, high, low, close, vol)
    Purpose: Parse API response into database-ready format

  insert_stock_data(records, symbol, max_rows=25)
    Input: records (list of tuples), symbol (string), max_rows (integer)
    Output: Integer count of rows actually inserted
    Purpose: Insert records into database up to max_rows limit

  collect_stock_data()
    Input: None
    Output: None (stores data in database as side effect)
    Purpose: Main function to collect stock data from all symbols

analyze_data.py

Author: Kevin Zang.

  get_crypto_prices_with_symbols()
    Input: None
    Output: List of dictionaries with crypto data (uses SQL JOIN)
    Purpose: Retrieve crypto price data joined with symbol names

  get_stock_prices_with_symbols()
    Input: None
    Output: List of dictionaries with stock data (uses SQL JOIN)

```
    Purpose: Retrieve stock price data joined with symbol names

  calculate_crypto_volatility(crypto_data)
    Input: crypto_data (list of dicts)
    Output: Dictionary {symbol: [(date, volatility), ...]}
    Purpose: Calculate daily volatility for cryptocurrencies

  calculate_stock_volatility(stock_data)
    Input: stock_data (list of dicts)
    Output: Dictionary {symbol: [(date, volatility), ...]}
    Purpose: Calculate daily volatility for stocks

  calculate_price_momentum(data, symbol_key, price_key, window=7)
    Input: data (list), symbol_key (str), price_key (str), window (int)
    Output: Dictionary {symbol: [(date, momentum), ...]}
    Purpose: Calculate price momentum over window days

  calculate_daily_returns(data, symbol_key, price_key)
    Input: data (list), symbol_key (string), price_key (string)
    Output: Dictionary {symbol: [(date, return_pct), ...]}
    Purpose: Calculate daily percentage returns

  calculate_correlation(series1, series2)
    Input: series1 (list of tuples), series2 (list of tuples)
    Output: Float (correlation coefficient from -1 to 1)
    Purpose: Calculate Pearson correlation between two series

  calculate_average_volatility(volatility_data)
    Input: volatility_data (dictionary)
    Output: Dictionary {symbol: avg_volatility}
    Purpose: Calculate average volatility for each symbol

  find_top_momentum_days(momentum_data, top_n=5)
    Input: momentum_data (dict), top_n (integer, default 5)
    Output: Dictionary {symbol: [(date, momentum), ...]}
    Purpose: Find days with highest absolute momentum

  perform_analysis()
    Input: None
    Output: Dictionary with all calculation results
    Purpose: Orchestrate all calculations and return results

  write_results_to_file(results, filename=None)
    Input: results (dict), filename (string or None for default)
    Output: None (writes file as side effect)
    Purpose: Write formatted results to text file


visualize_data.py
-----------------
Author: Kevin Zang.

  get_normalized_prices()
    Input: None
    Output: Tuple (crypto_normalized dict, stock_normalized dict)
    Purpose: Get price data normalized to base 100 for comparison

  create_price_movement_chart()
    Input: None
    Output: None (saves PNG file as side effect)
    Purpose: Create dual-axis line chart of price movements

  create_correlation_heatmap(results)
    Input: results (dictionary from perform_analysis)
    Output: None (saves PNG file as side effect)
    Purpose: Create heatmap of cross-market correlations

  create_all_visualizations()
    Input: None
    Output: None (saves both PNG files as side effect)
    Purpose: Orchestrate creation of all visualizations


main.py
-------
Author: Kevin Zang.

  print_header(title)
    Input: title (string)
    Output: None (prints to console)
    Purpose: Print formatted header

  print_menu()
    Input: None
    Output: None (prints to console)
    Purpose: Display interactive menu options

  initialize_database()
    Input: None
    Output: None (calls database_setup functions)
    Purpose: Menu handler for database initialization

  collect_crypto_data()
    Input: None
    Output: None (calls crypto collection function)
    Purpose: Menu handler for crypto data collection

  collect_stock_data()
    Input: None
    Output: None (calls stock collection function)
    Purpose: Menu handler for stock data collection

  check_progress()
    Input: None
    Output: None (prints progress to console)
    Purpose: Display data collection progress

  run_analysis()
    Input: None
    Output: None (calls analysis functions)
    Purpose: Menu handler for running analysis

  create_visualizations()
    Input: None
    Output: None (calls visualization functions)
    Purpose: Menu handler for creating visualizations

  run_everything()
    Input: None
    Output: None (runs all steps automatically)
    Purpose: Automated execution of all project steps

  view_database_summary()
    Input: None
    Output: None (prints database info to console)
    Purpose: Display database structure and sample data

  main()
    Input: None
    Output: None (runs interactive menu loop)
    Purpose: Main program loop
```

8. RESOURCES USED (10 POINTS)

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
Dec 15, 2025	Understanding CoinGecko API endpoint structure for historical data	https://docs.coingecko.com/reference/coins-id-market-chart	Yes - Successfully implemented market_chart endpoint to fetch price history
Dec 15, 2025	Alpha Vantage API documentation and parameter options	https://www.alphavantage.co/documentation/	Yes - Successfully implemented TIME_SERIES_DAILY endpoint
Dec 15, 2025	Alpha Vantage "premium endpoint" error when using outputsize='full'	https://www.alphavantage.co/support/#support	Yes - Changed from 'full' to 'compact' outputsize, resolved error
Dec 15, 2025	SQLite FOREIGN KEY constraints syntax for lookup tables	https://www.sqlite.org/foreignkeys.html	Yes - Successfully implemented FOREIGN KEY constraints in table creation
Dec 15, 2025	Python sqlite3 Row factory for dict-like access to query results	https://docs.python.org/3/library/sqlite3.html#sqlite3.Row	Yes - Used conn.row_factory = sqlite3.Row for cleaner code
Dec 15, 2025	Matplotlib dual y-axis (twinx) implementation	https://matplotlib.org/stable/gallery/subplots_axes_and_figures/two_scales.html	Yes - Successfully created dual-axis chart with twinx()
Dec 15, 2025	Seaborn heatmap customization and annotation options	https://seaborn.pydata.org/generated/seaborn.heatmap.html	Yes - Successfully created correlation heatmap with annotations
Dec 15, 2025	Matplotlib date formatting for integer dates	https://matplotlib.org/stable/api/dates_api.html	Yes - Used DateFormatter and AutoDateLocator to display dates properly
Dec 15, 2025	Pearson correlation coefficient formula and implementation	https://en.wikipedia.org/wiki/Pearson_correlation_coefficient	Yes - Implemented manual calculation: $r = \Sigma[(x-\bar{x})(y-\bar{y})] / \sqrt{[\Sigma(x-\bar{x})^2\Sigma(y-\bar{y})^2]}$
Dec 15, 2025	Database normalization best practices to avoid duplicate strings	SI 201 Course lecture notes (Week 10)	Yes - Implemented lookup tables to avoid duplicate strings
Dec 15, 2025	Python datetime strftime and strptime format codes	https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes	Yes - Used for date string conversions in helper functions
Dec 15, 2025	requests library timeout parameter to prevent hanging	https://requests.readthedocs.io/en/latest/user/advanced/#timeouts	Yes - Added timeout=10 to prevent hanging on API calls
Dec 15, 2025	Python time.sleep for API rate limiting	https://docs.python.org/3/library/time.html#time.sleep	Yes - Implemented 12-second delays between Alpha Vantage API calls