Chapter 1

No-code Generative AI for Basic Quantitative Finance

In this chapter we want to demonstrate how Gen AI can be used to do the basic tasks for which quantitative traders and investors used to hire a professional programmer. We shall find out to what extent we have succeeded. In all the following examples, we have used a ChatGPT GPT-4o subscription (at US$ 20/month as of this writing) since it is the most well-known Gen AI service. But you can try Microsoft Copilot (which has live internet access and often gives different answers than ChatGPT because of different "finetuning"), Google's Gemini Pro ($19.99/month), xAI's Grok, or DeepSeek to see if they can do better.

We shall see that while we can enter English instructions into ChatGPT, its best output is often code rather than numerical answers.

Ernie has basic Python programming skills but is much more proficient in Matlab. So, the following is written from the point of view of someone who is a novice Python programmer but an expert Matlab programmer. Let's see if ChatGPT can help such a person create useful Python code, either from English instructions, or translating from existing Matlab codes.

We will ask ChatGPT to do the following:

a. Retrieve *adjusted* historical prices of SPY and BIL.
b. Compute daily returns from those adjusted prices.
c. Compute the Sharpe ratio from those daily returns.

d. Analyze the data contained in an Excel spreadsheet.
e. Translate Matlab codes that use quadratic optimization to compute the efficient frontier of an ETF portfolio to Python code.
f. Use those Python codes to plot the efficient frontier.

In this and the next chapter, we will use the following boxes to denote prompts to ChatGPT, responses from ChatGPT, and our commentaries.

For user prompts, we will use the following format:

User Prompt

This is a user prompt.

For output from ChatGPT, including code, we will use the following:

ChatGPT Response

A response from ChatGPT

```
# Here is an amazing Python code example.
import my_genai_trading_system # :)
% Or a MATLAB code example.
load_system('my_genai_trading_system') % :)
```

If we embed our own commentary within its response, we will enclose it with [Author Note: ...], as follows:

ChatGPT Response

```
# Here is an amazing Python code example.
import spicy  # [Author Note: This should be 'scipy', not 'spicy'.]
```

Code, data, and relevant resources produced or required for this chapter can be found in the corresponding chapter folder within the book's repository: `https://github.com/genai-for-traders/genai-for-trading-and-am`. For this chapter, the files are located in the *chapter01* folder and includes scripts, Jupyter notebooks, input data files, and any necessary code for downloading or generating data.

1.1 Retrieving Historical Market Data

As readers of Ernie's first book (Chan, 2021) know, when we backtest a trading strategy on a stock, we must compute its daily returns. To properly compute its daily returns, we must use its closing prices adjusted for splits and dividends. (Imagine if a stock split 1 to 2 before the market opens on day t. Without adjusting for splits, the return from day $t-1$ to t would be about -50% or so.) So, we entered one of the most obvious prompts into ChatGPT:

User Prompt

Can you retrieve the daily closing prices of the ETF SPY for the most recent 100 days? Please make sure they are split and dividend adjusted.

ChatGPT Response

It seems that I currently don't have access to the *yfinance* library, which I typically use to retrieve financial data. Therefore, I'm unable to directly fetch the daily prices of the ETF SPY for the most recent 100 days.

However, you can easily obtain this information by using financial data services like Yahoo Finance, Google Finance, or Bloomberg. These platforms provide comprehensive historical data, including split and dividend adjustments. If you would like, I can guide you through the process of retrieving this data using these platforms.

Despite the seemingly unhelpful response as a no-code assistant, it did provide a decent Python script that may accomplish the task, which we included in Appendix A.1. We will discuss that later. Let's say we use that script to download the below prices on our desktop, and upload them as a csv file back into ChatGPT:

```
2023-08-25: $438.42
2023-08-28: $441.20
2023-08-29: $447.58
…
2023-12-11: $461.99
```

We asked this question on 2023-12-12, so the retrieved prices correctly ended on 2023-12-11. They are identical to what you would see on Yahoo Finance if you had done so on the same day. It is difficult to verify that on a later date (such as when you are reading this). Adjusted prices change depending on when you retrieve them, but they should produce the same daily returns no matter how the prices look to you. So, let's now compute the daily returns.

User Prompt

Please compute daily returns based on the above prices.

ChatGPT Response

…

To compute the daily returns, we use the formula:

$$\text{Daily Return} = \left(\frac{\text{Price}_{\text{today}} - \text{Price}_{\text{previous day}}}{\text{Price}_{\text{previous day}}}\right) \times 100\%$$

(Continued)

(*Continued*)

I'll calculate the daily returns for the SPY ETF based on the adjusted closing prices provided:

$$1.2023-08-28:((441.20-438.42)/438.42)\times 100\%$$
$$2.2023-08-29:((447.58-441.20)/441.20)\times 100\%$$

... and so on for each subsequent day.

Let's calculate these returns.

Here are the daily returns for the first five days:

$$1.2023-08-28:((441.20-438.42)/438.42)\times 100\% = 0.63\%$$
$$2.2023-08-29:((447.58-441.20)/441.20)\times 100\% = 1.45\%$$

...

These percentages represent the daily return for each respective day. Would you like me to continue calculating the daily returns for the remaining days?

Correct though these formula are, the output is difficult to download and verify independently. So we ask:

User Prompt

Can you formulate the above output as a csv file with two columns?

ChatGPT Response

Certainly! Here's the data formatted as a CSV file with two columns, "Date" and "Daily Return":

```
Date,Daily Return
2023-09-12,-0.0055
2023-09-13,0.0012
...
2023-12-11,0.0039
```

We can copy and paste this output in a plain text file called SPY Daily Returns.csv, and re-open it in Excel. Spot checking a few prices around the dividend ex-date of 2023-09-15, we verify that the returns were computed correctly, based on the adjusted closing prices.

Anticipating that we will need the risk-free rate in order to compute Sharpe ratio and other performance metrics for SPY (or other trading strategies), we ask:

User Prompt

Over the same period of time as above, please retrieve the daily risk free rate in % form.

ChatGPT Response

Retrieving the daily risk-free rate over a specific period, like the one for the SPY ETF adjusted closing prices, can be a bit more complex because there isn't a direct source that provides these rates in a daily format suitable for immediate download and use in financial calculations.

(By the time you are reading this, ChatGPT will probably have integrated different data sources that will enable it to download the yield on the 3-Month U.S.Treasury Yield.)

In other words, it can't do it unless data are available from Yahoo Finance. But there is a proxy that we can use for risk free returns: the SPDR Bloomberg 1–3 Month T-Bill ETF ("BIL"). We will retrieve this by a simple modification of the Python script described in Appendix A.1. We can use the codes in Appendix A.3 to compute the daily returns of BIL, which we can use as the (daily) risk free rate throughout the book.

As any data scientist knows, one of the best ways to check whether the results of a computation are correct is to plot them.

User Prompt

Plot the annualized returns chart with the same x-axis. We only need YYYY as x-axis labels.

You might think that the response to such a simple prompt should be straightforward, but it was not. On the first attempt, ChatGPT incorrectly annualized the returns by multiplying the daily returns by $\sqrt{252}$ instead of 252. At another attempt it couldn't parse the date/time format YYYY-MM-DD hh:mm:ss-05:00 correctly.

User Prompt

What does the date and time format YYYY-MM-DD hh:mm:ss-05:00 mean?

The response to this prompt is correct. The last hh:mm is the timezone offset from UTC. For example, in December, YYYY-MM-DD hh:mm:ss in New York will have −05:00 at the end.

After hours (!) of prompting, it finally produced the correct plot. We reproduced the Python code accompanying the plot in Appendix A.3. We recommend that you check out that code because you will see that the raw data are not quite usable. As a bonus, it gives us a bird's eye view of the US economy since 2007.

1.2 Computing Sharpe Ratio

With strenuous prompting (ChatGPT is like a mediocre and inexperienced intern), we finally downloaded data on our favorite ETFs SPY and BIL, computed the daily returns of both, and can now use these data to compute its Sharpe ratio. As we suggested previously, we want ChatGPT to display the Python code for the computation instead of generating numerical results directly.

> **User Prompt**
>
> Suppose a daily returns series A is stored in A.csv, and the daily risk free rate is stored in B.csv, and the dates of A and B are not identical though they overlap, how do we compute the Sharpe ratio of A? Please display the Python code.

We display the (largely correct) response obtained in Appendix A.4. The Sharpe ratio computed in Appendix A.4 is for the period 2023-08-25 to 2023-12-11, which was a very bullish period. You can of course adapt the same prompts and their resulting Python codes for other performance metrics like maximum drawdown, Calmar ratio, and the like.

1.3 Data Formatting and Analysis

Not every spreadsheet that comes our way will be nicely formatted like the one downloaded from Yahoo. One of the most thankless and time-consuming task for a data analyst … eh sorry … scientist, is to format the input data file correctly so we can analyze it. For example, a colleague sent us the Indices.xlsx file saved in `https://github.com/genai-for-traders/genai-for-trading-and-am/chapter01/`. We have anonymized the names of the indices to "A" "B" "C" but left everything else unchanged. The file contains lots of floating detritus such as the names of these indices, the start and end dates of the data, and #N/A for those dates that did not have data. Let's see if ChatGPT can make the task of analyzing this data easier.

> **User Prompt**
>
> This file contains four columns of pricing data. Can you tell me the column names for these data?
>
> [**Author Note:** we can attach a file to the prompt in ChatGPT.]

> **ChatGPT Response**
>
> Unfortunately, I encountered an error while trying to access and read the newly uploaded file. This could be due to a variety of reasons such as file format issues or compatibility problems.

That's not promising! When we tried this a couple of months back, it had no problem analyzing the same data. After several hours of trial and error, a trivial solution worked: we just logged out and back in again. This is often frustratingly the case with ChatGPT: the indeterministic and irreproducible nature of the results. Now the response is:

ChatGPT Response

The Excel file you provided contains the following column names:

1. Unnamed: 0
2. Unnamed: 1
3. [A column named with a date: 2000-01-03]
4. Unnamed: 3
5. Unnamed: 4
6. Unnamed: 5
7. Unnamed: 6
8. Unnamed: 7
9. Unnamed: 8
10. Unnamed: 9

...

Clearly, it needs a bit of human help. So

User Prompt

Use row 6, columns G to J as the column names for the pricing data.

ChatGPT Response

```
A Index
B Index
C Index
D Index
...
```

which is correct.

User Prompt

Compute the daily returns of these four indices.

ChatGPT Response

It appears there was an error in calculating the daily returns, likely due to the presence of non-numeric data in the columns.

The NaNs are tripping ChatGPT up. So

User Prompt

For those cells with NaN, you can output NaN as returns. But otherwise, display their numerical results.

After this prompt, ChatGPT attempted to compute the daily returns for several subsets of the data, and without the dates column. Finally, we prompted

User Prompt

I want the analysis to start from the first row to the last row, and output a .csv file for the results.

It computed the current daily returns for all the data saved in a .csv file. For example,

User Prompt

What are the returns on 11/6/2023?

ChatGPT Response

The daily returns for the indices on November 6, 2023, are as follows:

```
A Index: 0.001378
B Index: -0.004321
C Index: -0.003129
D Index: -0.000333
...
```

User Prompt

Using these daily returns, please compute the annualized Sharpe ratio of the A Index from 2019/1/7 to 2023/7/3, assuming risk free rate of 0.

ChatGPT Response

The annualized Sharpe ratio of the A Index for the period from January 7, 2019, to July 3, 2023, assuming a risk-free rate of 0, is approximately 0.732.

Once we spot-checked a few numbers like this, it is much more convenient and efficient to use ChatGPT to extract other statistical information on all or parts of the data instead of using Excel or Python. ChatGPT can be a great friend to data analysts … sorry … scientists.

1.4 Translating Matlab Codes to Python Codes

Readers of Ernie's previous three books know that they are written in Matlab, for reasons explained in his third book, *Machine Trading* (Chan, 2017). Even though he subsequently added Python codes to the online code repositories for the first two books, there aren't such codes for *Machine Trading*. We will see if we can use ChatGPT to translate such Matlab codes to Python.

The first code example in *Machine Trading* (ef.m) is about computing the efficient frontier of a portfolio of ETFs and finding the tangency portfolio (the one that maximizes Sharpe ratio historically). For our readers' convenience, we have reproduced that in Appendix A.5.

Now let's ask ChatGPT to translate this into Python. You can just upload ef.m and ask for translation, or you can copy and paste the entire file as text into the prompt.

User Prompt

Translate the following Matlab code to Python:

```
% ef.m
% Find efficient frontier of a set of ETFs
% Note in this program mean and sd of returns are row vectors,

clear;

load('inputDataOHLCDaily_ETF_20150417', 'stocks', 'tday', 'cl');

...
```

Author Note: see A.5

ChatGPT Response

To translate this MATLAB code to Python, we'll use libraries such as NumPy for numerical computations and SciPy for optimization (specifically, quadratic programming). The MATLAB code is used to find the efficient frontier for a set of ETFs, compute the tangency portfolio, and find the minimum variance portfolio. Here's the equivalent Python code:

```
import numpy as np
import pandas as pd
from scipy.optimize import minimize

# Assuming 'inputDataOHLCDaily_ETF_20150417.csv' is a CSV file with columns
'stocks', 'tday', and 'cl'
# Load data (you'll need to replace this with your actual data loading
mechanism)
data = pd.read_csv('inputDataOHLCDaily_ETF_20150417.csv')
stocks = data['stocks'].unique()
cl = data.pivot(index='tday', columns='stocks', values='cl')
```

(Continued)

(Continued)

```
# Remove 'EWZ' and 'FXI'
cl = cl.drop(columns=['EWZ', 'FXI'])

# Calculate returns
R = cl.pct_change().dropna()

# Mean and covariance of returns
mi = R.mean()
C = R.cov()

# Prepare different target mean portfolio returns for efficient frontier
m = np.linspace(mi.min(), mi.max(), 21)
v = np.zeros_like(m)

# Variance of portfolio to be minimized
H = 2 * C.values

# Short sale constraint
A = -np.eye(len(mi))
b = np.zeros(len(mi))

# No linear term in minimization
f = np.zeros(len(mi))

# Efficient frontier calculation
for i in range(len(m)):
    Aeq = np.vstack([mi, np.ones(len(mi))])
    beq = np.array([m[i], 1])

    def objective(F):
        return F @ H @ F

    constraints = ({'type': 'eq', 'fun': lambda F: np.dot(Aeq[0], F) -
beq[0]},
                   {'type': 'eq', 'fun': lambda F: np.dot(Aeq[1], F) -
beq[1]})
    bounds = [(0, None)] * len(mi)
    result = minimize(objective, np.ones(len(mi)) / len(mi), bounds=bounds,
constraints=constraints)
    v[i] = result.fun

sd = np.sqrt(v)
plt.scatter(sd, m)
plt.xlabel('Standard Deviation')
plt.ylabel('Mean Return')
plt.title('Efficient Frontier')
plt.show()

# Find tangency portfolio
sharpe_ratio = m / sd
idx = np.argmax(sharpe_ratio)

# Find minimum variance portfolio
idx_min = np.argmin(sd)

# For further details like the weights of the tangency and minimum variance
portfolios,
# use the 'idx' and 'idx_min' with the 'minimize' function as shown above.
```

ChatGPT goes on to respond:

ChatGPT Response

This code does the following:

Loads the ETF data and computes daily returns. Calculates the mean and covariance of the returns. Uses quadratic programming (via scipy.optimize.minimize) to find the efficient frontier, tangency portfolio, and minimum variance portfolio.

Note: The inputDataOHLCDaily_ETF_20150417.csv should be replaced with the actual file name and path you have. This code assumes the data is in a long format with "stocks," "tday" (date), and "cl" (close price) columns. Adjust the data loading and manipulation according to your actual data format. The plotting part (using matplotlib.pyplot) is minimal; you might want to enhance it based on your specific visualization requirements.

The first hurdle in comparing the output of the Python codes to the Matlab codes is that the Python code expects a .csv input file in the "long format," i.e., it assumes that price data corresponding to different stocks to be stacked into rows one day at a time. For example,

```
20050512    EWC 1412
20050512    EWG 1388
20050512    EWJ 894
…
20150417    EWY 6102
20150417    EWZ 3453
20150417    FXI 5003
```

Assuming that we are not expert Matlab programmers, we ask ChatGPT to create Matlab code that can save the information in the input .mat file into a .csv.

User Prompt

In the Matlab code, we assumed that the input file inputDataOHLCDaily_ETF_20150417.mat has a cell array stocks with length S, a double array tday with length T, and a double array cl with dimension TxS. What Matlab code can we write to output a csv file that conforms to the format assumed in the input .csv file of the Python code?

ChatGPT Response

```
% Load data from .mat file
load('inputDataOHLCDaily_ETF_20150417.mat', 'stocks', 'tday', 'cl');
```

(Continued)

(Continued)

```
% Initialize a table with 'tday' as the first column
outputData = array2table(tday, 'VariableNames', {'tday'});

% Add columns for each stock, with closing prices as values
for i = 1:length(stocks)
    outputData.(stocks{i}) = cl(:, i);
end

% Reshape the data to long format
outputDataLong = stack(outputData, stocks, 'NewDataVariableName', 'cl',
'IndexVariableName', 'stocks');

% Write to CSV
writetable(outputDataLong, 'outputData.csv');
```

The output data file looks reasonable. Renaming outputData.csv to "inputDataOHLCDaily_ETF_20150417.csv" will let us test the Python code.

From this point on, our voyage becomes treacherous. The Python codes ChatGPT translated from Matlab didn't work at first. We will display below the tedious prompt-vs-response between the human and the machine where we coached ChatGPT to produce codes that finally work, not because we are sadistic, but because it illustrates what it takes to squeeze the correct codes out of ChatGPT.

Running the Python code results in error "ModuleNotFoundError: No module named 'scipy'." Prompting ChatGPT for help generates:

ChatGPT Response

The error "ModuleNotFoundError: No module named 'scipy'" indicates that the SciPy library, which is essential for the optimization part of the code (specifically, the minimize function used for quadratic programming), is not installed in your Python environment.

To resolve this, you will need to install the SciPy library. If you are using a standard Python installation, you can install SciPy using pip, which is the Python package manager. Here's how you can do it:

Open your command line interface (CLI). Run the following command:

```
pip install scipy
```

which we did. Running the Python code again resulted in error:

```
NameError                                 Traceback (most recent call last)
Cell In[1], line 50
     47     v[i] = result.fun
     49 sd = np.sqrt(v)
---> 50 plt.scatter(sd, m)
     51 plt.xlabel('Standard Deviation')
     52 plt.ylabel('Mean Return')

NameError: name 'plt' is not defined
```

Confronting ChatGPT with this error yields:

ChatGPT Response

The error you're encountering, "NameError: name 'plt' is not defined," indicates that the matplotlib.pyplot module, commonly imported as plt, has not been imported in your Python script. This module is necessary for plotting the efficient frontier.

To resolve this, you need to import matplotlib.pyplot at the beginning of your script. Here's the corrected segment of the Python code with the necessary import statement:

```
...
import matplotlib.pyplot as plt  # Import matplotlib for plotting

# [Rest of your code]
```

Nice of ChatGPT to tell us that, but why didn't it add that in the beginning? Anyway, the code now generated the Figure 1.1. (If you want to save this to a .png file, remember to add `plt.tight_layout()` before `plt.savefig(plot_file_path).`)

Comparing this plot with the efficient frontier plotted by the Matlab code in Appendix A.5, we see clearly that there is something wrong with the Python code. The scales of the x and y axes are correct. The coordinates of the minimum variance portfolio (corresponding to the green dot on the Matlab chart) also seem right. But what happened to the portfolios with the lowest two mean returns? Drilling down into the details, the mean returns vectors mi from both programs match. The covariance matrices C from both programs also match. Following the suggestion from ChatGPT above, let's compute the weights for the tangency portfolio:

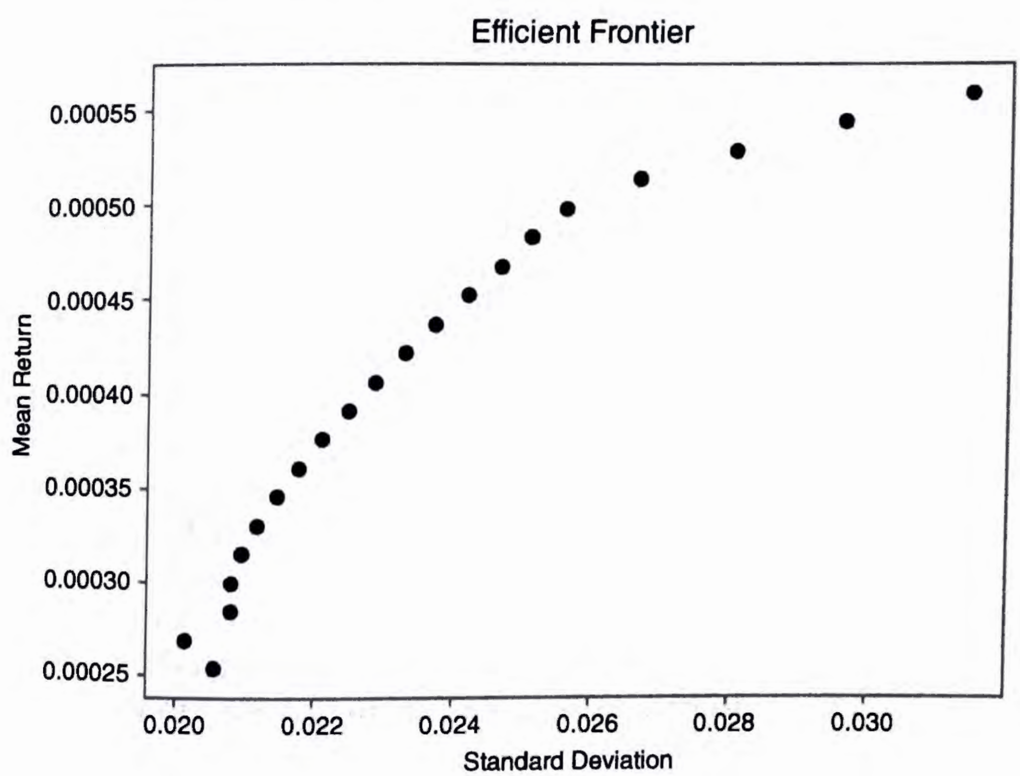

Figure 1.1 Efficient frontier based on Python code generated by ChatGPT.

```
Aeq   np.vstack([mi, np.ones(len(mi))])
beq   np.array([m[idx], 1])

def objective(F):
    return F   H   F

constraints   ({'type': 'eq', 'fun': lambda F: np.dot(Aeq[0], F)   beq[0]},
               {'type': 'eq', 'fun': lambda F: np.dot(Aeq[1], F)   beq[1]})
bounds   [(0, None)] * len(mi)
result   minimize(objective, np.ones(len(mi)) / len(mi), bounds bounds,
constraints constraints)
tangency_portfolio_weights - result.x
```

`tangency_portfolio_weights` are found to be `[0.19413052, 0.2978653, 0., 0.01713741, 0., 0.49086677]` which are very different from `F = [0.3815, 0, 0.6040, 0.0001, 0.0143, 0]` found by Matlab. Plugging these weights back into the array for the Sharpe ratios at different weights, we found that Matlab gave a Sharpe ratio of `0.0283` for the tangency portfolio vs Python's `0.0194`. Clearly, Matlab's optimization is better. This is, however, not really the fault of ChatGPT, as a line-by-line comparison of the codes revealed nothing erroneous. In fact, we are pleasantly surprised that it was able to find an optimization package ("minimize") in scipy that reasonably mirrors the quadratic optimization function in Matlab. The fault lies squarely with Python's subpar optimization package. As Ernie has maintained in *Machine Trading* (Chan, 2017), Python is a poor cousin of Matlab—you use it at your own peril. While Matlab's codes are developed and maintained commercially by a team of full-time professionals and PhDs, Python's codes are developed and maintained by essentially a group of part-time volunteers. (A bit of grapevine gossip: the Father of Deep Learning and Nobel prize laureate, Dr. Geoff Hinton himself, was known to prefer Matlab to Python in his own research.)

1.5 Conclusion

We have asked ChatGPT to retrieve historical data from the web, compute returns and Sharpe ratio from these returns, conduct exploratory data analysis, apply quadratic optimization to compute the efficient frontier, and in general translate Matlab to Python codes. With an annoyingly large amount of coaching (i.e., prompting) and vigilant error-checking, we conclude that ChatGPT is capable of these tasks. At the very least, it can generate a first draft from which human quants can use to build better codes. This is especially helpful when the human is not an expert programmer. For example, finding the exact syntax to generate the precise chart we want in Python is tedious and time consuming for a native Matlab programmer like Ernie—a task that ChatGPT can do efficiently using English instructions and with minimal room for serious error. The keyword here is "codes"—we found it much better to ask ChatGPT to suggest codes that we can audit (see Alshahwan et al., 2024) and modify than to provide the direct numerical answers to our questions.

In the next chapter, we will be more ambitious and see if Gen AI can help us backtest or even suggest quantitative trading strategies.