

Chapter 2

No-code Generative AI for Trading Strategies Development

In the previous chapter, we found that with sufficient patience and prompting, we can get ChatGPT to perform basic financial data science tasks such as retrieving, exploring, and plotting data and computing performance statistics from them. Though the instructions are all in English, the output are frequently codes (especially Python codes). We still consider that no-code usage of Gen AI.

In this chapter, we will see if no-code Gen AI can handle more challenging tasks such as turning a trading strategy specification in English into backtest codes, and even more ambitious, suggest a trading strategy based on a vague human insight. As a reminder, Ernie has basic Python programming skills, is much more proficient in MATLAB, and is a hedge fund founder. This means he has extensive experience and domain knowledge in finance. In the following examples, the author approaches the development of the examples as someone who is an expert in the finance field, but not a professional programmer or a specialist in prompt engineering (like many professional asset managers). He needs help coding in a new language. Let's see if ChatGPT can help him. Spoiler alert: ChatGPT's responses will likely improve in later releases.

As in Chapter 1, we use ChatGPT GPT-4o for our tests. You can also try Microsoft's Copilot, Google's Gemini Pro, xAI's Grok, or DeepSeek to see if they can do better. We will ask ChatGPT to do the following:

- a. Create Python codes to backtest a long-short equity investment strategy that Ernic specified, which uses Fama-French factors to predict individual stocks' returns.
- b. Create Python codes to backtest a hedged VIX futures carry strategy from a published paper.
- c. Conduct a literature search to find a portfolio optimization technique that uses deep reinforcement learning and that can adapt to regime changes.
- d. Suggest and conduct exploratory analysis of a SPX options calendar spread trading strategy.

Once again, 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:

ChatGPT Response

A response from ChatGPT

```
# This is a user prompt.
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: ...], for example:

ChatGPT Response

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

Once again, 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 *chapter02* folder and include scripts, Jupyter notebooks, input data files, and any necessary code for downloading or generating data.

2.1 Creating Codes from a Strategy Specification

In *Machine Trading* (Chan, 2017), Ernie created a toy strategy that used the Fama-French three-factor model to predict stocks' returns and backtested the performance of a trading strategy based on those predictions. Let's ask ChatGPT to create all the Python codes necessary for the backtest by just specifying the strategy in English. (Recall that Chan, 2017 only has Matlab codes.)

User Prompt

In the following dialogue, pretend you are a quantitative finance expert.

ChatGPT Response

... [Author Note: The response is a lot of unimportant babble. Some research suggests it results in better responses if we first prime ChatGPT on a particular application area. See Kiciman et al. (2023).]

User Prompt

Fama-French factors are often used to explain the returns of a portfolio of stocks. What are the Fama-French factors?

ChatGPT Response

The Fama-French factors are a set of three factors (and later expanded to five) introduced by Eugene F. Fama and Kenneth R. French. They are used in finance to explain the returns of a stock portfolio, going beyond the traditional Capital Asset Pricing Model (CAPM), which uses only market risk (beta) to explain returns. Here's a breakdown of these factors:

Original Three Factors

Market Risk ($R_m - R_f$):

This is the same as in CAPM, representing the excess return of a portfolio over the risk-free rate, capturing the market risk. R_m stands for the return of the market portfolio, and R_f is the risk-free rate.

Size Factor (SMB - Small Minus Big):

SMB stands for "Small [market capitalization] Minus Big". It captures the historical excess returns of small-cap stocks over big-cap stocks. The factor is based on the observation that, over long periods, small-cap stocks tend to outperform large-cap stocks.

(Continued)

(Continued)

Value Factor (HML - High Minus Low):

HML stands for "High [book-to-market ratio] Minus Low". This factor captures the excess returns of value stocks (with high book-to-market ratios) over growth stocks (with low book-to-market ratios). Value stocks are those that are typically undervalued by the market but have strong fundamentals.

...

So far so good—at least ChatGPT regurgitates the correct information that you can find in any quantitative finance textbook (e.g., Ruppert and Matteson, 2015) or even Wikipedia. Now let's get it to produce codes.

User Prompt

Let's focus only on the original Fama-French three factors. Instead of using them to explain contemporaneous returns of stocks in a portfolio, let's use them to predict the next day's returns of these stocks. Can you display the mathematical setup for this prediction?

ChatGPT Response

Certainly! To use the original Fama-French three factors for predicting the next day's returns of stocks in a portfolio, we set up a predictive linear regression model. In this model, the future stock returns are the dependent variable, and the current values of the Fama-French factors are the independent variables. The mathematical setup for this predictive model is as follows:

Let's denote: [Author Note: We did some light editing of the LaTeX code to make it more readable.]

$R_{i,t+1}$: The return of stock i in the next day (day $(t + 1)$).

$R_{m,t} - R_{f,t}$: The excess return of the market over the risk free rate in the current day (t).

SMB_t : The small minus big factor in the current day.

HML_t : The high minus low factor in the current day.

$(\beta_{i0}, \beta_{i1}, \beta_{i2}, \beta_{i3})$: Coefficients to be estimated for stock i .

$\epsilon_{i,t+1}$: The error term for stock i in month [Author Note: This should be day, not month] $t + 1$.

The linear regression model for each stock i can be represented as:

$$R_{i,t+1} = \beta_{i0} + \beta_{i1}(R_{m,t} - R_{f,t}) + \beta_{i2}SMB_t + \beta_{i3}HML_t + \epsilon_{i,t+1}$$

In this model:

β_{i0} , is the intercept term for stock i .

$\beta_{i1}, \beta_{i2}, \beta_{i3}$ are the coefficients measuring the sensitivities of stock i 's returns to the market excess return, SMB, and HML factors, respectively.

$\epsilon_{i,t+1}$ captures the residual effects not explained by the model.

This setup allows for the prediction of next day's returns for each stock in the portfolio based on the current values of the Fama-French factors. It's important to note that while this model uses historical relationships to make predictions, the accuracy of these predictions can vary due to market volatility, economic changes, and other factors not captured by the model.

User Prompt

Correct. Can you turn this mathematical model into Python code?

To which ChatGPT responded with Python codes that represent the linear regression of a single stock's returns against the Fama-French factors. Obviously, that isn't complete. Also, as a prerequisite, we will need data. Assume that the prices are contained in the file "CRSP_mid.csv," which contains data formatted as follows:

```
tday      syms   mid
20070103  ABT    8.867780
20070103  HON    37.689365
20070103  AA     25.932974
20070103  BEAM   45.437196
20070103  AEP    31.218061
... ... ...
20131231  GGP    20.055000
20131231  MHK    148.904995
20131231  ADS    262.945005
20131231  FB     54.655000
20131231  TSCO   NaN
[1309166 rows x 3 columns]
```

While we cannot share this data due to licensing restrictions, you can construct a similar file from any daily stock survivorship-bias-free data, such as that from Sharadar. (See a list of other data vendors in Chan [2017].)

Exercise: Ask ChatGPT to help you convert whatever data you purchased into the "long" format shown in Section 1.4.

We asked ChatGPT for the code for computing the future 1-day return, and it responded with something that looked like the code in Appendix 2A.1 at the end of this chapter. The original code wasn't perfect, but if you know a bit of Python, it is easy to fix it up.

Next, we need to upload the Fama-French data file, which is available from https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html as

"F-F_Research_Data_Factors_daily.CSV" We again prompted ChatGPT for the codes, and the response is passable. After we edited it lightly, the code is displayed in Appendix 2A.2. We also prompted for the codes for merging the labels (future 1-day return) and the features (Fama-French factors) into one dataframe for use as our training data. Those are shown in Appendix 2A.3.

Now we are ready to train a regression model, or more precisely, train regression models for all the stocks. We started with the following:

User Prompt

The linear regression model you proposed above is correct, but it is for one stock only. Can you display the codes for all the stocks in the data?

ChatGPT Response

...

```
# Assuming 'Stock_ID' and '1_Day_Returns' are columns in your next month
# returns data
# Train a model for each stock
for stock_id in combined_data['Stock_ID'].unique():
    stock_data = combined_data[combined_data['Stock_ID'] == stock_id]
    X = stock_data[['Mkt-RF', 'SMB', 'HML']] # Replace with actual
    Fama-French column names
    y = stock_data['1_Day_Returns']
    model = LinearRegression().fit(X, y)
    # Save or use the model for each stock
```

The code looks correct. But unfortunately, it can't handle any NaNs in the data, and financial data (or indeed any data) is full of such missing values. So, we try again.

User Prompt

Correct, but what if the X and y contains NaN?

ChatGPT Response

as most machine learning algorithms, including linear regression in scikit-learn, cannot handle NaN values directly. To address this, you have a few options: ...

The statement that "most machine learning algorithms cannot handle NaN" is *wrong*. Certainly most if not all algorithms in Matlab's Statistics and Machine Learning Toolbox can handle NaN. We want to catch ChatGPT in its lie:

User Prompt

Can LightGBM handle NaNs in the input?

ChatGPT Response

Yes, LightGBM, a gradient boosting framework that uses tree-based learning algorithms, can handle NaN (Not a Number) values in its input data. . . .

Which directly contradicts ChatGPT's own statement above! Self-consistency is not a forte of Gen AI.

User Prompt

Is there any Python package for linear regression that can handle NaN in the input data?

ChatGPT Response

... Statsmodels, another statistical modeling library in Python, can handle missing data using the missing='drop' parameter in its model-fitting functions. This tells the model to automatically drop observations with NaNs.

```
import statsmodels.api as sm
model = sm.OLS(y, X, missing='drop').fit()
```

Note that this approach drops any row with NaNs, similar to using `dropna()` in pandas, but it's done internally within the model.

So, we tried replacing the line

```
model = LinearRegression().fit(X, y)
```

with the `sm.OLS` line (after running `pip install statsmodel`). Well, that didn't work either, because `sm.OLS` is interpreting the `missing='drop'` option as weights. ChatGPT has no idea why, and suggested we remove all rows with NaN before running the `LinearRegression` in scikit-learn.

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Assuming 'combined_data' is your DataFrame
some_minimum_threshold=100 # Define a threshold as per your analysis

for stock_id in combined_data['Stock_ID'].unique():
    stock_data = combined_data[combined_data['Stock_ID'] == stock_id]

    # Remove rows where any of the columns have NaN values
    stock_data = stock_data.dropna(subset=['1_Day_Returns', 'Mkt-RF', 'SMB', 'HML'])

X = stock_data[['Mkt-RF', 'SMB', 'HML']]
y = stock_data['1_Day_Returns']
```

(Continued)

(Continued)

```
# Ensure that X and y have enough data points
if len(X) > some_minimum_threshold: # Define a threshold as per
your analysis
    model = LinearRegression().fit(X, y)

    print({
        'Stock_ID': stock_id,
        'Intercept': model.intercept_,
        'Beta_Market': model.coef_[0],
        'Beta_SMB': model.coef_[1],
        'Beta_HML': model.coef_[2]
    })
}
```

That seems to work. But we don't want to fit a regression model using all our data—we need to leave some data for out-of-sample testing of our trading strategy. So let's use the first half for training:

User Prompt

The program you created above using Linear Regression to predict 1-day return is correct, but can you modify it so that it uses only the first half of the data for fitting?

ChatGPT Response

...

```
# Split the data into two halves
halfway_point = len(combined_data) // 2
training_data = combined_data.iloc[:halfway_point]
#...
for stock_id in training_data['Stock_ID'].unique():
    # Filter training data for the current stock
    stock_data = training_data[training_data['Stock_ID'] == stock_id]
#...
```

The changes look correct and are easy enough to incorporate in our original program. (See the final code fragment in the %%Train model cell in ff_1day.py, available for download from our GitHub.)

We will need to make predictions using the previous trained (fitted) model. Let's ask ChatGPT how to do that.

User Prompt

With the fitted regression model above, how do you make predictions using input "features"?

ChatGPT responded with some very cumbersome codes that explicitly multiplies the regression coefficients such as "Beta_Market" with the input features contained in X. So, we asked it to make them more elegant.

User Prompt

Instead of explicitly using the names of the model coefficients to make these predictions, can you call some predict function generic to a regression model object?

ChatGPT Response

Certainly! To use a generic predict function that is common to regression model objects, you would typically save each fitted model object for each stock during the training process, and then use the predict method of these model objects to make predictions. This approach is more flexible and aligns with standard practices in model deployment.

Here's how you can adjust the code to include saving the fitted model objects and then using their predict methods:

Step 1: Save Each Fitted Model

While training the models, save each fitted model object into a dictionary where the keys are the stock IDs:

```
from sklearn.linear_model import LinearRegression

# Dictionary to store the model for each stock
fitted_models = {}

for stock_id in training_data['Stock_ID'].unique():
    # ... [Data preparation steps]

    # Fit the model
    model = LinearRegression().fit(X, y)

    # Store the fitted model
    fitted_models[stock_id] = model
```

Step 2: Predict Using the Saved Models

Use the predict method of each saved model to make predictions on new data:

```
def predict_returns(fitted_models, features):
    """
    Make predictions for each stock using the fitted models.

    :param fitted_models: Dictionary of fitted model objects for each stock
    :param features: DataFrame with new input features for prediction
    :return: A DataFrame with predicted returns for each stock
    """

    predictions = []

    for stock_id, model in fitted_models.items():
        # Select the features for the current stock
        # stock_features = features.loc[features['Stock_ID'] == stock_id,
        ['Market_Excess_Return', 'SMB', 'HML']]
        # Modify this to
        stock_features = features.loc[features['Stock_ID'] == stock_id,
        ['Mkt-RF', 'SMB', 'HML']]
```

(Continued)

(Continued)

```

# Check if there are features to predict on
if not stock_features.empty:
    # Make prediction
    predicted_return = model.predict(stock_features)

    # Store the predictions
    for i, prediction in enumerate(predicted_return):
        predictions.append({'Stock_ID': stock_id,
                            'Predicted_Return': prediction,
                            'Date': stock_features.index[i]})

return pd.DataFrame(predictions)

# Example usage
# features = pd.DataFrame(...) # Replace with your new input features
DataFrame
predicted_returns = predict_returns(fitted_models, training_data)
print(predicted_returns)

```

In this updated code:

- The `predict_returns` function iterates through each fitted model in `fitted_models`.
- It uses the `predict` method on the subset of features corresponding to each stock.
- The function compiles the predictions into a `DataFrame`.

Ensure that the features `DataFrame` is structured correctly with the Fama-French factors and stock IDs. This approach allows for more flexibility and is more aligned with typical model deployment practices.

Granted, we would only know to ask this question if we have prior knowledge that fitted models from *scikit-learn* or *statsmodels* come with a `predict` function, so clearly this won't work for someone completely unfamiliar with Python and its data science libraries. Furthermore, though this code looks useful because it helps us save the fitted models somewhere as well, it isn't quite correct.

A fundamental problem with the `predict_returns` function is that it is expecting the input `stock_features` to have an index, "stock_id." This may be appropriate for "cross-sectional" features (see Nautiyal and Chan [2021]), but certainly not for "time-series" features. When we told ChatGPT that these are time-series features that do not have `stock_id` as index, it thinks we must be predicting the returns of the portfolio as a whole, not the returns of the individual stocks in that portfolio. So, more prompting is needed to clarify.

User Prompt

Maybe it is common to use time-series factors to predict portfolio returns, but here, we have trained our models to predict returns for each stock. Modify your function accordingly.

ChatGPT Response

```
...
import pandas as pd
import numpy as np

def predict_returns(fitted_models, features_today):
    """
    predict returns for each stock for a single day using their respective
    fitted models.

    :param fitted_models: Dictionary of fitted model objects for each stock
    :param features_today: DataFrame or Series with input features for a
    single day
    :return: A Series with predicted returns for each stock
    """
    predictions = {}

    # Ensure features_today is in the correct format (2D array)
    if isinstance(features_today, pd.Series):
        features_today = features_today.values.reshape(1, -1)
    elif isinstance(features_today, pd.DataFrame):
        features_today = features_today.values

    # Iterate over each model and predict the return for each stock
    for stock_id, model in fitted_models.items():
        # Predict the return for the current stock using the common features
        predicted_return = model.predict(features_today)

        # If the model returns a list/array, take the first element as the
        # prediction
        if isinstance(predicted_return, (list, np.ndarray)):
            predictions[stock_id] = predicted_return[0]
        else:
            predictions[stock_id] = predicted_return

    return pd.Series(predictions)
```

Ultimately, none of the codes produced by ChatGPT quite worked. Ernie had to use his limited Python skills to mix-and-match different codes to produce the final function.

```
# Predicting returns

import pandas as pd
import numpy as np

def predict_returns(fitted_models, features_today):
    """
    Predict returns for each stock for a single day using their respective
    fitted models.

    :param fitted_models: Dictionary of fitted model objects for each stock
    :param features_today: DataFrame or Series with input features for a
    single day
    :return: A Series with predicted returns for each stock
    """
```

```

predictions = {}

# Ensure features_today is in the correct format (2D array)
features_today = features_today[['Mkt-RF', 'SMB', 'HML']]
features_today = features_today.values.reshape(1, -1)

# Iterate over each model and predict the return for each stock
for stock_id, model in fitted_models.items():
    # Predict the return for the current stock using the common features
    predicted_return = model.predict(features_today)

    # If the model returns a list/array, take the first element as the prediction
    predictions[stock_id] = predicted_return[0]

return pd.Series(predictions)

```

If you run this code as

```
predicted_returns = predict_returns(fitted_models, features_today)
```

where features today is a pd.Series that looks like

Mkt-RF	-0.04
SMB	0.08
HML	-0.11
RF	0.022

and fitted_models is an output of the training codes above, the output will look like this:

```

0033A    0.044187
3511B    0.057673
3FMCC   -0.139659
3FNMA   -0.112233
5933B    0.045648
...
XTO      0.016649
YHOO     -0.012752
YUM      0.015054
ZION     -0.039987
ZMH      -0.007693
Length: 553, dtype: float64

```

Now we need to incorporate these predictions in a backtest program for the trading strategy. But let's test if ChatGPT understands what a trading strategy is first.

User Prompt

Imagine a trading strategy that buys the top 50 stocks with the highest predicted returns and short the bottom 50, on a daily basis, and hold those positions for the next 30 days. What is the maximum number of positions we can hold for a particular stock at any one time?

ChatGPT Response

Since you are trading daily and holding each position for 30 days, it means that on any given day, you could have up to 30 overlapping positions for the same stock, each initiated on a different day over the past 30 days. This scenario assumes that the stock consistently appears either in the top 50 or bottom 50 each day for 30 consecutive days.

Impressive! ChatGPT is finally showing some conceptual intelligence that we felt wasn't quite apparent until now. However, when we asked it to convert this trading strategy to backtest code (with 1-day holding period for simplicity and to match the existing Matlab code), it produced versions that are wrong in one aspect or another. In one version it is using all the columns in the input data as features instead of just selecting the three Fama-French factors. In another version, it forgot that the input data has multiple rows of the same date that correspond to different *stock_Id's*. So, we had to ask the following:

User Prompt

This doesn't fix the issue. The problem is that the for loop goes over every row in the data, assuming that each row refers to data of a different date. But that's not true. Currently it will repeatedly calculate portfolio_returns for the same date many times as it iterates through the rows of the data.

To which it regrettably responded with more erroneous codes.

In a third version, ChatGPT thinks it is smart and looks at the data 1-day ahead of the current date to look for the value 31_Day_Returns. But that is unnecessary because we created 1_Day_Returns as the return in the next day already. This is another case where unless you understand Python, you won't be able to catch the error.

After many days (!) of prompting and manual adjustments, we finally arrived at the following working version.

```
# Backtest long-short trading strategy

import pandas as pd
import warnings
warnings.filterwarnings('ignore') # Without suppression of warnings, the
program will freeze in Jupyter Notebook due to too much output

# Function to execute the trading strategy
def execute_strategy(data, fitted_models, holding_period=1):
    portfolio_returns = []

    # Get unique dates from the data
    unique_dates = data['Date'].unique()

    for day in range(len(unique_dates) - holding_period):
        # Get the current date
        current_date = unique_dates[day]
```

```

# Filter data for the current date
data_today = data[data['Date'] == current_date]

# Get features for this day (assuming the features are the same for
# all stocks on a given day)
features_today = data_today[['Mkt-RF', 'SMB', 'HML', 'RF']].iloc[0]

# Predict returns using the predict_returns function
predicted_returns = predict_returns(fitted_models, features_today)

# Implement your strategy: select top 50 and bottom 50 stocks
top_50 = predicted_returns.nlargest(50)
bottom_50 = predicted_returns.nsmallest(50)

# Calculate returns for the selected stocks on a current date
current_date = unique_dates[day]
current_data = data[data['Date'] == current_date]

#print(current_date)

long_returns = current_data[current_data['Stock_ID'].isin(top_50.
index)]['1_Day_Returns'].mean()
short_returns = -current_data[current_data['Stock_ID'].isin(bottom_50.index)]['1_Day_Returns'].mean()

# Daily portfolio return
daily_portfolio_return = (long_returns + short_returns) / 2
portfolio_returns.append(daily_portfolio_return)

return portfolio_returns

```

To execute this function, ChatGPT helpfully suggested (correctly)

```
portfolio_returns=execute_strategy(training_data, fitted_models)
```

It produces a list, not a pd.Series, which isn't ideal because it does not include timestamps. We shan't be picky, so we said:

User Prompt

Correct! Please compute the Sharpe ratio.

ChatGPT Response

```

import numpy as np
# Calculate Sharpe Ratio
def calculate_sharpe_ratio(portfolio_returns, risk_free_rate=0.02):
    excess_returns = np.array(portfolio_returns) - risk_free_rate / 252
    sharpe_ratio = np.mean(excess_returns) / np.std(excess_returns) *
    np.sqrt(252)
    return sharpe_ratio

```

This brings up an important point that Ernie discussed in *Quantitative Trading* (Chan, 2021). Let's see if ChatGPT understands this nuance.

User Prompt

In the calculate_sharpe_ratio function defined above, the risk-free rate is defaulted to 0.02. For a long-short dollar-neutral portfolio, should we deduct risk-free rate from the returns in a Sharpe ratio calculation? In other words, is excess returns the raw returns minus the risk-free rate?

ChatGPT Response

In the context of a long-short dollar-neutral portfolio, the consideration of the risk-free rate in the calculation of the Sharpe ratio can be a bit nuanced. Here's how to think about it:

Excess Returns: Typically, the Sharpe ratio is calculated using excess returns, which are the portfolio's returns minus the risk-free rate. This is to assess how much additional return the investment is providing over a risk-free alternative (like Treasury bills).

Dollar-Neutral Portfolio: In a dollar-neutral portfolio, you're equally long and short in dollar terms. Theoretically, such a portfolio might be seen as self-financing, implying that its performance could be assessed without considering the risk-free rate. The rationale is that any interest you would earn on the long side (from investing money) might be offset by the interest you would pay on the short side (from borrowing money).

Practical Considerations: Practically, whether or not to include the risk-free rate in the Sharpe ratio calculation for a dollar-neutral portfolio depends on several factors:

Strategy and Leverage: If your strategy involves leverage or financing costs, these should be factored into the return calculation. **Benchmarking:** Consider what is standard practice in your benchmark group or among your peers. **Investor Expectations:** Consider what investors in your fund would expect. They might compare your fund's performance to other investments, which typically are assessed using excess returns over the risk-free rate. **Risk-Free Rate in Sharpe Ratio:**

Include Risk-Free Rate: If you want to maintain consistency with broader market practices and make your Sharpe ratio comparable to others, you might include the risk-free rate. **Exclude Risk-Free Rate:** If you believe that your portfolio's characteristics (like being dollar-neutral) make the risk-free rate less relevant, you could exclude it.

In summary, whether to use raw returns or excess returns (returns minus the risk-free rate) in the Sharpe ratio calculation for a long-short dollar-neutral portfolio depends on the specific characteristics of your portfolio and your performance measurement goals. Both approaches have rationales, but it's important to be consistent and clear about the method you choose, especially when communicating with stakeholders or investors.

Impressive! It got most of the nuance right. For a long-short portfolio such as ours, we do not typically subtract risk-free rate (R_f) from the raw returns in Sharpe ratio calculations for the self-financing reason ChatGPT described. In other words, investors with this portfolio can still earn margin interest on the cash deposited in their accounts, and as long as you don't include such margin interest in your P&L, you also don't need to subtract R_f from your returns. But if you are running a hedge fund and this portfolio is its sole holding, the fund's returns (ignoring management and performance fees) will already have included this margin interest, and so in reporting its Sharpe ratio, you must subtract R_f . During the post-COVID era of high interest rates that reached more than 5%, this is an important consideration. Since we are not including R_f as credit interest in our portfolio, we can also set R_f to 0 in our following Sharpe ratio calculation.

```
# Calculate Sharpe Ratio
def calculate_sharpe_ratio(portfolio_returns, risk_free_rate=0.0):
    excess_returns = np.array(portfolio_returns) - risk_free_rate / 252
    sharpe_ratio = np.mean(excess_returns) / np.std(excess_returns) *
    np.sqrt(252)
    return sharpe_ratio

# Backtest the strategy
sharpe_ratio = calculate_sharpe_ratio(portfolio_returns)
print("Sharpe Ratio:", sharpe_ratio)
```

which produces 6.5, much higher than the 3.7 that we obtained from Matlab, on the training set. Which is right?

The reason the Python program seemingly outperformed the Matlab program is that we have skipped all stocks with insufficient data in the Python program, whereas in the Matlab program, we have made a prediction of 0 returns for those stocks—which wasn't correct.

Let's see how it performs on the test set.

User Prompt

Let's say I have dataframe combined_data. I want to split it into 2 two halves: training_data and test_data using halfway_point = len(combined_data) // 2 training_data = combined_data.iloc[:halfway_point]. How should I compute test_data?

ChatGPT Response

```
test_data = combined_data.iloc[halfway_point:]
```

which is correct. Running the backtest on the test data resulted in a Sharpe ratio of 0.55, which is much better than the negative Sharpe from the Matlab program. The complete process can be downloaded from ff_1day.py on our GitHub. Now let's display the cumulative return ("equity" or "Growth of \$1") curve shown in Figure 2.1.

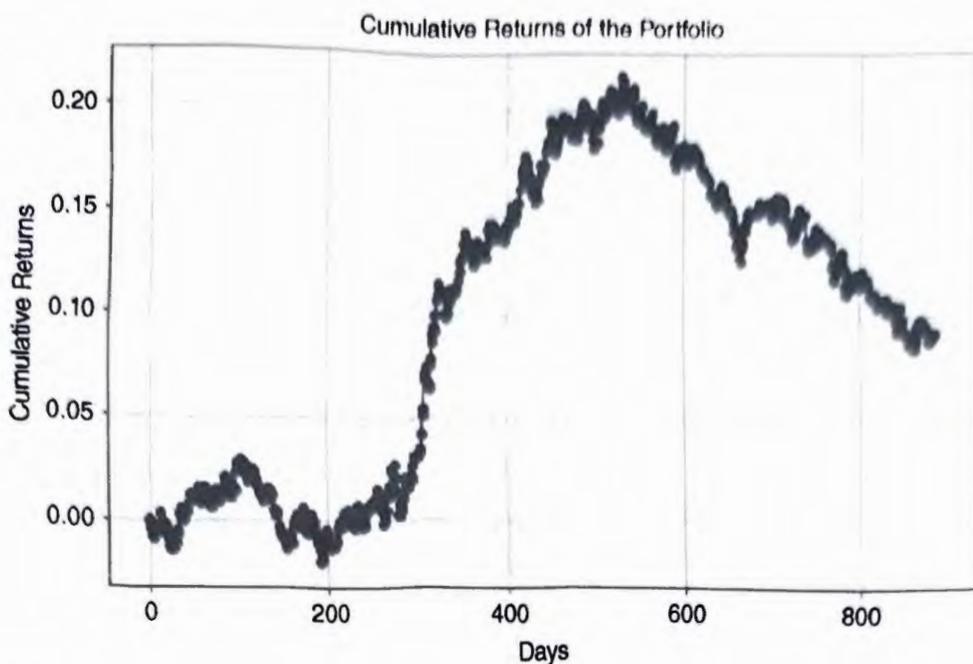


Figure 2.1 Cumulative returns of Fama-French three-factor strategy.

User Prompt

Suppose the daily returns of a portfolio is contained in a list `portfolio_returns`. Please display the cumulative returns curve.

ChatGPT Response

```
import matplotlib.pyplot as plt
import numpy as np

# Example portfolio_returns list (replace this with your actual returns)
portfolio_returns = [0.01, -0.005, 0.007, -0.003, 0.002, 0.01, -0.004]

# Calculate cumulative returns
cumulative_returns = np.cumprod(1 + np.array(portfolio_returns)) - 1

# Plotting the cumulative returns curve
plt.figure(figsize=(10, 6))
plt.plot(cumulative_returns, marker='o', linestyle='--')
plt.title('Cumulative Returns of the Portfolio')
plt.xlabel('Days')
plt.ylabel('Cumulative Returns')
plt.grid(True)
plt.show()
```

Remember, this is all on the test set. If you are wondering, the drawdown started around 2012-09, and it had been relentless until the end of the data 2023-12. This doesn't mean Fama-French factors can't explain returns any more. It just means factor loadings are no longer persistent from one day to the next. As a side note, we did not try to make the plots generated by ChatGPT look pretty since the goal was just to show the

raw outputs it produced. Sure, we could have asked it to create better-looking figures, but that wasn't the point of this exercise. By the way, if you ask ChatGPT to improve the style of plots in Python using Matplotlib or Seaborn, it is actually really good at it.

2.2 Summarizing a Trading Strategy Paper and Creating Backtest Codes from It

Ernie's second book, *Algorithmic Trading* (Chan, 2013, p.143), describes an interday momentum trading strategy involving the VIX futures. The strategy is based on a paper by Simon and Campasano (2014). Let's see if ChatGPT can read and understand that paper and produce backtest codes for it.

User Prompt

Pretend you are a quantitative finance expert. Please summarize the trading strategies described in this paper.

[Author Note: Remember we can upload a paper to ChatGPT-4o.]

ChatGPT Response

...

the paper explores relative value spread trades, designed to profit from deviations in the EH [Author Note: Expectations Hypothesis]

...

The paper's results suggest that the VIX futures market presents opportunities for strategic trades based on its term structure anomalies.

That summary is too high level. So, we asked it to be more specific.

User Prompt

More specifically, generate pseudocodes for the relative value spread trades, or any other trades based on term structure anomalies.

ChatGPT Response

The paper describes a beta-neutral trading strategy (BN) that involves maintaining a long position in constant-maturity VIX futures with a long-maturity tenor (L) and a simultaneous short position in constant-maturity VIX futures with a short-maturity tenor (S). The goal of this strategy is to ensure that the net position is VIX-neutral. Here's a simplified pseudocode for the beta-neutral trading strategy:

```

# Define the beta-neutral (BN) trading strategy
Define BN_Trading_Strategy(long_maturity_tenor, short_maturity_tenor,
constant_maturity_betas):
    # Initialize weights for long and short positions
    weight_long = (1 / beta_long) / ((1 / beta_long) + (1 / beta_short))
    weight_short = 1 - weight_long

    # Calculate daily payoff
    daily_payoff = weight_long * return_long - weight_short * return_short

    return daily_payoff

# Example usage
long_maturity_tenor = L # Specify the long-maturity tenor
short_maturity_tenor = S # Specify the short-maturity tenor
constant_maturity_betas = Table_4_data # Retrieve constant-maturity betas
from table 4

# Calculate the daily payoff for the beta-neutral trading strategy
daily_payoff = BN_Trading_Strategy(long_maturity_tenor, short_maturity_tenor,
constant_maturity_betas)

```

In this pseudocode:

`long_maturity_tenor` and `short_maturity_tenor` are the tenors for the long and short VIX futures positions, respectively. `constant_maturity_betas` represents the betas for each maturity, which can be retrieved from a specified table (Table 4 in the paper). The weights for the long (`weight_long`) and short (`weight_short`) positions are calculated based on these betas. The `daily_payoff` is computed as the weighted difference between the returns of the long and short positions.

This actually *isn't* the trading strategy the paper proposes. The strategy proposed isn't a calendar spread strategy. It is a strategy to buy or short VIX futures based on whether it is in backwardation or contango and then hedge its position by the ES future with the same sign. Instead, we want ChatGPT to focus on the following paragraph in the paper:

The simulations examine the profitability of shorting or buying VIX futures contracts when the VIX futures basis is in contango or backwardation, respectively, and the efficiency of hedging market risk with mini-S&P futures positions. The simulations then examine the impact of modifying entry rules and then introducing exit rules based on risk management strategies. All trading simulations examine the profitability of shorting or buying the front VIX futures contract that has at least ten business days until settlement when the magnitude of the daily roll is greater in magnitude than .10 VIX futures point (\$100) per day.¹³ The daily roll is defined as the difference between the front VIX futures price and the VIX, divided by the number of business days until the VIX futures contract settles, and measures potential profits assuming that the basis declines linearly until settlement.”

Let's see how we can prompt it to do so.

User Prompt

In your pseudocodes above, you missed an important trading rule. See especially the paragraph starting with "The simulations examine the profitability of shorting or buying VIX futures contracts when the VIX futures basis is in contango or backwardation,..."

[Author Note: We continued with the rest of the paragraph just quoted.]"

ChatGPT Response

...

```
# Define the beta-neutral trading strategy with contango/backwardation
# consideration
Define BN_Trading_Strategy_With_Contango_Backwardation(long_maturity_tenor,
short_maturity_tenor, constant_maturity_betas, VIX_term_structure_state):
    # Determine trade direction based on the state of the VIX futures term
    # structure
    if VIX_term_structure_state == "contango":
        # In contango, buy long-maturity and sell short-maturity VIX futures
        trade_direction = "buy_long_sell_short"
    elif VIX_term_structure_state == "backwardation":
        # In backwardation, sell long-maturity and buy short-maturity VIX
        # futures
        trade_direction = "sell_long_buy_short"
    else:
        return "No trade"

    # Calculate weights for long and short positions
    weight_long = (1 / beta_long) / ((1 / beta_long) + (1 / beta_short))
    weight_short = 1 - weight_long

    # Calculate daily payoff based on trade direction
    if trade_direction == "buy_long_sell_short":
        daily_payoff = weight_long * return_long - weight_short *
    return_short
    elif trade_direction == "sell_long_buy_short":
        daily_payoff = weight_short * return_short - weight_long *
    return_long

    return daily_payoff

# Example usage
VIX_term_structure_state = Determine_VIX_Term_Structure_State() # Function to
# determine if VIX is in contango or backwardation
daily_payoff = BN_Trading_Strategy_With_Contango_Backwardation(long_
maturity_tenor, short_maturity_tenor, constant_maturity_betas,
VIX_term_structure_state)
```

In this revised pseudocode, the function `Determine_VIX_Term_Structure_State()` is assumed to analyze the VIX term structure and return either "contango" or "backwardation". The trading strategy then adjusts its positions accordingly, either buying long and selling short in a contango state or the reverse in a backwardation state.

It still isn't the strategy described in the paragraph, but at least it mentioned the term structure criterion. Does ChatGPT know how the authors of the paper define backwardation or contango of the VIX futures? We shall find out.

User Prompt

Based on the paper, provide definition of the function Determine_VIX_Term_Structure_State(). That is, define backwardation or contango based on the paper's definition.

ChatGPT Response

Based on the information in the paper, the Determine_VIX_Term_Structure_State() function can be defined as follows:

Contango: The VIX futures curve is in a state of contango when it is upward sloping. This means that the futures prices are higher than the spot price of the VIX index, and they increase as the time to maturity increases. In a contango market, the long end of the futures curve is less volatile, and the price ranges (difference between high and low readings) and the volatility of prices decrease with increasing time to maturity.

Backwardation: The VIX futures curve is in a state of backwardation when it is downward sloping. This occurs when the futures prices are lower than the spot price of the VIX index, and they decrease as the time to maturity increases. The long end of the curve in backwardation would be more responsive to changes in the spot price.

Here's how the Determine_VIX_Term_Structure_State() function could be defined in pseudocode:

```
# Define a function to determine the state of the VIX term structure
Define Determine_VIX_Term_Structure_State(VIX_spot_price,
VIX_futures_prices):
    # Compare spot price with futures prices for different maturities
    is_upward_sloping = True
    for each maturity in VIX_futures_prices:
        if VIX_futures_prices[maturity] < VIX_spot_price:
            is_upward_sloping = False
            break

    # Determine the term structure state based on the slope
    if is_upward_sloping:
        return "contango"
    else:
        return "backwardation"

# Example usage
VIX_spot_price = Current_VIX_Spot_Price()    # Function to get the current VIX
spot price
VIX_futures_prices = Get_VIX_Futures_Prices()  # Function to get VIX futures
prices for different maturities

term_structure_state = Determine_VIX_Term_Structure_State(VIX_spot_price,
VIX_futures_prices)
```

(Continued)

(Continued)

In this pseudocode:

The `VIX_spot_price` is the current spot price of the VIX index. `VIX_futures_prices` is a dictionary (or similar data structure) that holds futures prices for various maturities. The function compares the spot price with futures prices for each maturity to check if the curve is upward or downward sloping, thus determining if the term structure is in contango or backwardation.

It got the definition of backwardation and contango right. But still, we are missing trading rules that incorporate them.

User Prompt

That is correct definition of backwardation and contango. But note the paragraph starting with "The simulations examine...." It has a version of the strategy embedded there. Please summarize that version and provide pseudocodes.

ChatGPT Response

Thank you for providing the detailed paragraph. Based on this information, the revised trading strategy involves:

Determining the State of the VIX Futures Basis: The strategy involves shorting VIX futures contracts when the VIX futures basis is in contango and buying them when in backwardation.

Entry Rules: Trades are entered when the front VIX futures contract, which has at least ten business days until settlement, shows a daily roll greater in magnitude than 0.10 VIX futures points (\$100) per day.

Daily Roll Calculation: The daily roll is calculated as the difference between the front VIX futures price and the VIX, divided by the number of business days until the VIX futures contract settles. This measures potential profits assuming that the basis declines linearly until settlement.

Exit Rules: Based on risk management strategies, the strategy may also involve specific exit rules, though these are not detailed in the paragraph you provided.

Alternative Thresholds: Footnote 13 indicates that simulations were also run with ± 0.15 daily roll thresholds, resulting in fewer but more profitable trades.

Here's a pseudocode representation of this strategy:

```
# Define the trading strategy based on VIX futures basis and daily roll
Define Trading_Strategy(VIX_spot_price, front_VIX_futures_price, days_until_settlement, daily_roll_threshold = 0.10):
    # Calculate the daily roll
    daily_roll = (front_VIX_futures_price - VIX_spot_price) / days_until_settlement

    # Determine the state of the VIX futures basis
    if daily_roll > daily_roll_threshold:
        basis_state = "contango"
```

```

    elif daily_roll < -daily_roll_threshold:
        basis_state = "backwardation"
    else:
        return "No trade"

    # Determine the trade action based on the basis state
    if basis_state == "contango":
        trade_action = "short VIX futures"
    elif basis_state == "backwardation":
        trade_action = "buy VIX futures"

    return trade_action

# Example usage
VIX_spot_price = Current_VIX_Spot_Price() # Function to get the current VIX
spot price
front_VIX_futures_price = Get_Front_VIX_Futures_Price() # Function to get
the price of the front VIX futures contract
days_until_settlement = Get_Days_Until_Settlement(front_VIX_futures_contract)
# Function to get the number of business days until settlement of the front
VIX futures contract

trade_action = Trading_Strategy(VIX_spot_price, front_VIX_futures_price,
days_until_settlement)

```

Getting closer. However, it is still strange that it completely ignored the explicit requirement (much discussed in the paper) to hedge the VIX future with E-mini future.

User Prompt

Apparently you have forgotten the specification “hedging market risk with mini-S&P futures positions”. How would that enter your strategy specification and pseudocode?

ChatGPT Response

Incorporating the hedging of market risk with mini-S&P futures into the trading strategy requires adding a component to the strategy that adjusts the position in mini-S&P futures based on the market exposure of the VIX futures position. The idea here is to hedge the market risk associated with the VIX futures trade by taking an offsetting position in mini-S&P futures.

Here's how this could be integrated into the strategy:

Assess Market Exposure: After determining the trade action for VIX futures (either shorting or buying), assess the market exposure this position creates. For instance, a long position in VIX futures could be considered as having negative market exposure (since VIX often increases when the market falls), and a short position in VIX futures as positive market exposure.

Hedge with Mini-S&P Futures: Based on the assessed market exposure from the VIX futures position, take an offsetting position in mini-S&P futures. If the VIX futures position has negative market exposure, a long position in mini-S&P futures would hedge this risk, and vice versa.

(Continued)

(Continued)

Here is the revised pseudocode with market risk hedging:

```

# Define the trading strategy with market risk hedging using mini-S&P futures
Define Trading_Strategy_With_Hedging(VIX_spot_price, front_VIX_futures_price,
days_until_settlement, S_and_P_position_size, daily_roll_threshold = 0.10):
    # Calculate the daily roll
    daily_roll = (front_VIX_futures_price - VIX_spot_price) /
days_until_settlement

    # Determine the state of the VIX futures basis
    if daily_roll > daily_roll_threshold:
        basis_state = "contango"
    elif daily_roll < -daily_roll_threshold:
        basis_state = "backwardation"
    else:
        return "No trade"

    # Determine the trade action based on the basis state
    if basis_state == "contango":
        VIX_trade_action = "short VIX futures"
        S_and_P_trade_action = "long mini-S&P futures" # Hedge against
positive market exposure
    elif basis_state == "backwardation":
        VIX_trade_action = "buy VIX futures"
        S_and_P_trade_action = "short mini-S&P futures" # Hedge against
negative market exposure

    # Adjust mini-S&P futures position size to hedge market risk
    Adjust_Mini_SP_Futures_Position(S_and_P_position_size,
S_and_P_trade_action)

    return VIX_trade_action, S_and_P_trade_action

# Example usage
VIX_spot_price = Current_VIX_Spot_Price()
front_VIX_futures_price = Get_Front_VIX_Futures_Price()
days_until_settlement = Get_Days_Until_Settlement(front_VIX_futures_contract)
S_and_P_position_size = Determine_Position_Size() # Function to determine
the size of the mini-S&P futures position

VIX_trade_action, S_and_P_trade_action = Trading_Strategy_With_
Hedging(VIX_spot_price, front_VIX_futures_price, days_until_settlement,
S_and_P_position_size)

```

In this pseudocode, `Adjust_Mini_SP_Futures_Position` is a function that adjusts the position in mini-S&P futures based on the calculated trade action and position size. This function would take into account the specifics of the hedge, such as the degree of leverage or the precise sizing of the mini-S&P position relative to the VIX position, to effectively balance the market risk.

After basically spoon-feeding information to ChatGPT (treating it as an inexperienced and mediocre intern), it finally added the hedge. But wait, whenever there is a hedge, we need to specify a hedge ratio. Doesn't ChatGPT know basic finance?

User Prompt

This is correct. Does the paper specify how we should determine the hedge ratio between the VIX and ES futures?

ChatGPT Response

I have thoroughly reviewed the paper "VIX Futures Term Structure and the Expectations Hypothesis" to find specific information regarding the determination of the hedge ratio between VIX and mini-S&P (ES) futures. However, the sections of the paper I was able to access did not provide explicit details or formulas for calculating this hedge ratio.

This mediocre intern is becoming even more annoying. But as mentors, we shall not lose patience.

User Prompt

See this paragraph starting with "The hedge ratio—the number of mini-S&P futures contracts to buy or sell per VIX futures position—is constructed using the out of sample parameter estimates from equation 3." Are you able to summarize the method of calculating the hedge ratio and display the pseudocodes?

ChatGPT Response

..., the hedge ratio (HR) is calculated using the following formula:

$$HR_t = \frac{\beta_1 \times 1000 + \beta_2 \times TTS_{t-1} \times 1000}{0.01 \times ES_{t-1} \times 50}$$

where:

- HR_t is the hedge ratio at time t .
- β_1 and β_2 are parameters estimated from equation 3 (not provided in your quote).
- TTS_{t-1} is the lagged value of some variable (not specified in your quote) at time $(t - 1)$.
- ES_{t-1} is the lagged value of the mini-S&P futures contract at time $(t - 1)$.
- The value of a one-point VIX futures price change is \$1,000.
- The gain or loss for a one percent mini-S&P futures price change is calculated as 0.01 times the lagged value of the mini-S&P futures contract times \$50, which is the value of one mini-S&P futures point.

The average hedge ratio is roughly one mini-S&P futures contract per VIX futures contract, with a range from 1/2 to 2 contracts, influenced mainly by fluctuations in the number of business days to settlement of the relevant VIX futures contracts and the level of S&P futures prices.

(Continued)

(Continued)

Here is the pseudocode for this hedge ratio calculation:

```

# Define a function to calculate the hedge ratio
Define Calculate_Hedge_Ratio(beta1, beta2, TTS_lagged, ES_lagged):
    # Calculate the value of a one-point VIX futures price change
    VIX_price_change_value = 1000

    # Calculate the gain or loss for a one percent mini-S&P futures price
    # change
    mini_SP_price_change_gain_loss = 0.01 * ES_lagged * 50

    # Calculate the hedge ratio
    HR = (beta1 * VIX_price_change_value + beta2 * TTS_lagged * VIX_price_
change_value) / mini_SP_price_change_gain_loss

    return HR

# Example usage
beta1 = Parameter_Estimate1 # Estimated from equation 3
beta2 = Parameter_Estimate2 # Estimated from equation 3
TTS_lagged = Lagged_Value_TTS # Lagged value of the TTS variable
ES_lagged = Lagged_Value_ES # Lagged value of the mini-S&P futures contract

hedge_ratio = Calculate_Hedge_Ratio(beta1, beta2, TTS_lagged, ES_lagged)

```

In this pseudocode, `Parameter_Estimate1` and `Parameter_Estimate2` represent the out of sample parameter estimates β_1 and β_2 from equation 3. `Lagged_Value_TTS` and `Lagged_Value_ES` are the lagged values of the TTS variable and the mini-S&P futures contract, respectively. The function `Calculate_Hedge_Ratio` computes the hedge ratio based on these inputs and the formula provided.

Apparently, ChatGPT couldn't remember the entire paper, hence it doesn't know what β_1 and β_2 are, nor does it know what TTS_{t-1} is. To avoid prolonging the agony, let's just quote what the paper said.

"The hedge ratios are constructed from regressions of VIX futures price changes on a constant and on contemporaneous percentage changes of the front mini-S&P 500 futures contract both alone and multiplied by the number of days that the VIX futures contract is from settlement, as shown below.

$$\Delta VIX_t^F = \beta_0 + \beta_1 * SPRET_t + \beta_2 * SPRET_t * TTS_t + \mu, \quad (3)"$$

That's the mysterious equation 3 that ChatGPT can't remember. Let's leave that aside and move on to a bigger issue: what about the exit rule? The paper specified that

"These simulations also assume that trades are exited at the end of 5 business days...."
Let's see if ChatGPT spotted that!

User Prompt

In the trading strategy specification Define_Trading_Strategy_With_Hedging(VIX_spot_price, front_VIX_futures_price, days_until_settlement, S_and_P_position_size,

daily_roll_threshold = 0.10), it appears that you only specified when we should enter a trade. What is the exit rule?

ChatGPT Response

The exit rule for the trading strategy involving VIX futures and hedging with mini-S&P futures, as described in your earlier quote from the paper, isn't explicitly detailed in the information provided...

User Prompt

That isn't true! See the paragraph that contains the sentence, "These simulations also assume that trades are exited at the end of"

ChatGPT Response

...

```
# Define the trading strategy with market risk hedging and a fixed duration
# exit rule
Define Trading_Strategy_With_Hedging_And_Fixed_Duration_Exit(VIX_spot_price,
front_VIX_futures_price, days_until_settlement, S_and_P_position_size, daily_
roll_threshold = 0.10, holding_period = 5):
    # Entry rules as before
    ...

    # Exit rule based on a fixed holding period of 5 business days
    Define Fixed_Duration_Exit_Rule(current_holding_days, holding_period):
        if current_holding_days >= holding_period:
            return "exit"
        else:
            return "hold"

    # Monitor the holding period and decide whether to exit
    current_holding_days = Count_Holding_Days() # Function to count the
    number of business days since the trade was entered
    exit_decision = Fixed_Duration_Exit_Rule(current_holding_days,
holding_period)

    return exit_decision

# Example usage
...
exit_decision = Trading_Strategy_With_Hedging_And_Fixed_Duration_Exit(...)
```

In this pseudocode:

holding_period is set to 5 business days, as specified in your sentence. The function Fixed_Duration_Exit_Rule checks whether the current holding period has reached or exceeded the specified duration of 5 business days. Count_Holding_Days is a hypothetical function that counts the number of business days since the trade was initiated.

ChatGPT finally found the relevant sentence in question! Now let's convert to Python codes.

User Prompt

Good. Now convert the pseudocodes to a complete Python program to backtest the complete trading strategy.

ChatGPT Response

```
import pandas as pd
import numpy as np
from datetime import timedelta

# Function to simulate getting historical data (Replace with actual data source)
def get_historical_data():
    # Dummy data for illustration
    data = pd.DataFrame({
        'date': pd.date_range(start='2020-01-01', periods=100, freq='B'),
        'VIX_spot': np.random.uniform(20, 30, 100),
        'VIX_futures': np.random.uniform(20, 30, 100),
        'ES_futures': np.random.uniform(3000, 4000, 100)
    })
    return data

# Function to calculate the daily roll
def calculate_daily_roll(VIX_futures, VIX_spot, days_until_settlement):
    return (VIX_futures - VIX_spot) / days_until_settlement

# Trading strategy implementation
def trading_strategy(data, daily_roll_threshold=0.10, holding_period=5):
    positions = []
    entry_days_remaining = None

    for index, row in data.iterrows():
        if entry_days_remaining is None:
            # Entry logic
            daily_roll = calculate_daily_roll(row['VIX_futures'], row['VIX_spot'], days_until_settlement=10)
            if abs(daily_roll) > daily_roll_threshold:
                position = 'short' if daily_roll > 0 else 'long'
                entry_days_remaining = holding_period
        else:
            # Holding the position
            entry_days_remaining -= 1
            if entry_days_remaining <= 0:
                # Exit logic after holding period
                positions.append(position)
                position = None
                entry_days_remaining = None

    return positions

# Main function to run the backtest
def backtest_strategy():
    data = get_historical_data()
```

```
positions = trading_strategy(data)
print("Trade positions:", positions)

# Run the backtest
backtest_strategy()
```

This script includes:

A dummy function `get_historical_data` to simulate historical data. You should replace this with actual data retrieval from a reliable source. Functions `calculate_daily_roll` and `trading_strategy` implement the entry and exit logic based on the daily roll and the fixed holding period. The `backtest_strategy` function runs the backtest and prints the trade positions. Please note:

This script is highly simplified and doesn't include real data sources, transaction costs, slippage, risk management features, or performance metrics.

The hedge ratio and hedging with mini-S&P futures are not included in this script. You would need to integrate this based on the specific hedge ratio formula and market data. For a realistic backtesting environment, consider using a backtesting framework like `backtrader` or `zipline`. This script is a starting point, and you would need to expand and refine it for more comprehensive and realistic backtesting.

Strangely, ChatGPT didn't translate the complete pseudocodes it generated earlier into the corresponding Python. This is quite inexcusable.

User Prompt

Why don't you include the hedging positions with E-mini futures, since you already have the pseudocodes for Define Calculate_Hedge_Ratio? Add that into the Python codes.

It responded by adding the `calculate_hedge_ratio` function definition and so forth. The final code is correct, and we saved it in `VX_ES.py` on our GitHub.

Overall, we find that ChatGPT is an exceptionally poor reader and content summarizer when technical details are involved. It cannot remember many details from a typical finance paper of about 40 pages. Even when presented with specific quotes from the paper, it took a great deal of human intelligence, expertise, and prompting to get it to generate pseudo backtest codes from a paper. This is far from the AGI paradise (or dystopia) that we were promised. Count us as unimpressed.

2.3 Searching for a Portfolio Optimization Algorithm Based on Machine Learning

Enough about spoon-feeding ChatGPT specific strategies that we want to backtest. How about if we use it to find these algorithms? One of us has developed a machine-learning-based portfolio optimization method called CPO (Chan et al., 2023). The CPO method uses a large number of input features spanning different markets, including fundamental,

macro-economic factors, technical, and even microstructure data. A machine learning algorithm is then trained on these data in addition to many historical hypothetical capital allocations, in order to determine which capital allocation will give the best Sharpe ratio for some forward period at any time.

However, this isn't the only way machine learning can be applied to portfolio optimization. A more "conventional" approach is to apply deep reinforcement learning (DRL), with the current portfolio allocation as the current "state," future returns as the "reward," reallocation decisions as "actions" that lead to the next state. The typical input to DRL are just returns or prices—we haven't seen any examples of using fundamental and other big data input, especially one using time-series features. So, we ask Copilot for such references. (We tried ChatGPT first, but it did not provide satisfactory responses.)

Again, Kiciman et al. (2023) said that we should warm up and orient LLMs to a specific domain to get better results.

User Prompt

Pretend you are a quantitative finance and deep learning expert in the following.

Copilot Response

[Author Note: some babble].

User Prompt

We intend to apply deep reinforcement learning to the portfolio optimization problem. What types of input have researchers used for this problem?

Copilot Response

Applying deep reinforcement learning (DRL) to portfolio optimization is a cutting-edge approach that combines the complexity of financial markets with advanced AI techniques. In this context, researchers have used various types of inputs to inform the DRL models. Here are some common types of inputs:

- 1. Price-based Data:** This is the most common input type and includes historical prices, returns, and other derived statistics like moving averages, volatility (measured as standard deviation of returns), and momentum indicators. This data can be at different frequencies (daily, hourly, minute-level, etc.).
- 2. Volume Data:** Trading volume and its derivatives can provide insights into market sentiment and liquidity, which are important factors in portfolio optimization.
- 3. Fundamental Data:** For equities, this might include company financials like earnings, revenue, debt levels, P/E ratios, etc. For other asset classes, relevant

fundamental data would be used (e.g., economic indicators for forex or commodities).

4. **Alternative Data:** This can range from satellite imagery for predicting crop yields in commodity markets to social media sentiment analysis for gauging market sentiment.
5. **Market Indicators:** Various market indices, inter-market spreads, and macroeconomic indicators (like interest rates, inflation rates, GDP growth rates) can be important for understanding the broader market context.
6. **Technical Indicators:** Indicators like the Relative Strength Index (RSI), Bollinger Bands, MACD (Moving Average Convergence Divergence), and others are often used by traders and can be useful inputs for a DRL model.
7. **Risk Metrics:** Measures like Value at Risk (VaR), Conditional Value at Risk (CVaR), or Sharpe Ratio can also be inputs to guide the DRL algorithm toward not just return maximization but also risk-adjusted return optimization.
8. **News and Event Data:** Real-time news feeds, earnings reports, and other event data can have a significant impact on market dynamics and, hence, on portfolio performance.
9. **Order Book Data:** In high-frequency trading scenarios, order book data (like bid-ask spreads, order depth, etc.) can be crucial.
10. **Macro Signals:** Signals from larger economic trends or geopolitical events can also be factored into more sophisticated models.

...

That looks encouraging—it mentions a large variety of input, exactly what we have in mind. So, let's drill down.

User Prompt

Can you give me references to research that used fundamental data as input for a DRL solution to portfolio optimization?

Copilot Response

...

One study, "Deep Reinforcement Learning for Portfolio Optimization Using Latent Feature State Space (LFSS) Module," explores the use of reinforcement learning in a financial context...

...

Another relevant study is titled "A New Deep Reinforcement Learning Model for Dynamic Portfolio Optimization."...

...

We actually know of one specific paper (Cong et al., 2021) that uses DRL with fundamental features, although it is more of a stock-selection algorithm than a general portfolio optimization one. It is still disappointing that ChatGPT didn't mention it. Let's see what these other papers are about.

User Prompt

In the Deep Reinforcement Learning for Portfolio Optimization Using Latent Feature State Space (LFSS) Module you cited, what kind of input data was used?

Copilot Response

...
The primary focus was on price and return data, which is typically derived from Open, High, Low, Close (OHLC) data.
...

That's not what we wanted!

User Prompt

In the new deep reinforcement learning model for dynamic portfolio optimization paper you cited, what kind of input data was used?

Copilot Response

The paper "A New Deep Reinforcement Learning Model for Dynamic Portfolio Optimization" did not provide specific details in the sections I accessed about the exact types of input data used ...

Well, that's quite disappointing. What's the use of AI if we have to read the original paper to find that information?

User Prompt

So, neither of the two papers you cited actually used fundamental data as input, yet you claimed that fundamental data is one type of data used for DRL. What evidence do you have on which to base that claim?

Copilot Response

You are correct in noting that the two papers I cited earlier did not specifically mention the use of fundamental data as input in their deep reinforcement learning models for portfolio optimization. However, the claim that fundamental data is used

in such models is based on broader trends and practices in the field of quantitative finance and deep learning.

...

In summary, while the specific papers I cited did not use fundamental data, it remains an important type of input for DRL-based portfolio optimization. I appreciate your diligence and thank you for pointing out the oversight!

For further exploration, you can refer to research papers that explicitly focus on fundamental data integration in DRL models for portfolio management.

User Prompt

Yes, so which research papers should I explore that cover that exact topic?

Copilot Response

Certainly! Here are some research papers that specifically explore the integration of fundamental data into deep reinforcement learning (DRL) models for portfolio optimization:

"Deep Reinforcement Learning for Portfolio Optimization Using Latent Feature State Space (LFSS) Module" Abstract: This paper presents a DRL framework for autonomous portfolio optimization. It includes a Latent Feature State Space (LFSS) Module for filtering and feature extraction from financial data. The study confirms robust results and performance advantages over several benchmarks.

"Deep Reinforcement Learning in Portfolio Management" Abstract: This paper implements two state-of-the-art continuous reinforcement learning algorithms (Deep Deterministic Policy Gradient and Proximal Policy Optimization) in portfolio management. While it doesn't explicitly focus on fundamental data, it provides insights into DRL techniques for portfolio optimization.

"A New Deep Reinforcement Learning Model for Dynamic Portfolio Optimization" Abstract: This model integrates data processing methods, deep learning, and reinforcement learning to enhance investor decision-making. While it doesn't directly use fundamental data, it explores dynamic portfolio optimization.

"Deep Reinforcement Learning for Optimizing Finance Portfolio Management" Abstract: This study examines how DRL can be adopted to maximize portfolio management in the financial sector. While it covers various aspects, including technical indicators, it provides insights into DRL applications.

...

It is great that Copilot finally cited some papers relevant to our query, but when we followed up by asking what specific features these papers used, it wasn't able to answer. In fact, you can verify that these papers did not use any fundamental data or time-series features at all. Once again, count us as unimpressed.

2.4 Explore Options Term Structure Arbitrage Strategies

In the ideal world of the famous Black-Scholes options pricing model that every quant finance student is forced to learn, volatility is constant, and prices can be described by Gaussian Brownian Motion (i.e., a random walk of log prices). None of these assumptions is true in the real world. That is a good thing for traders—if these assumptions were correct, implied volatilities would have been constant for any maturity, and so would these options' annualized time values. In other words, the term structure of options prices would be flat, and term structure arbitrage would have been impossible. By term structure arbitrage, we meant the possibility of earning options premium by buying and shorting options of different maturities without taking directional risk in the underlying. The simplest term structure arbitrage strategy is calendar spreads trading where all the options involved have the same strike price.

Let's see if ChatGPT can help us construct a term structure arbitrage strategy. Note that this task is more challenging than any of the other tasks we have asked ChatGPT to perform so far. Here we haven't specified a trading strategy and are asking ChatGPT to construct one based only on conceptual hints.

User Prompt

Pretend you are a quantitative finance and especially an options trading expert in the following.

User Prompt

Please summarize different trading strategies involving term structure arbitrage in options

ChatGPT Response

Term structure arbitrage in options trading involves exploiting inefficiencies in the pricing of options across different expiration dates. This type of arbitrage takes advantage of discrepancies in the implied volatility curve or the term structure of interest rates, which can affect option prices differently depending on their time to expiration. Here are several strategies that traders might use to exploit these inefficiencies:

1. Calendar Spreads (Horizontal Spreads)

- **Description:** Involves buying and selling two options of the same type (either calls or puts) and strike, but with different expiration dates. The trader might sell a short-term option and buy a long-term option, betting on the volatility difference over time.
- **Goal:** Profit from the difference in time decay or changes in the volatility term structure.

2. Diagonal Spreads

- **Description:** Similar to calendar spreads but involves options with different strikes and expiration dates. It's a combination of a vertical and a calendar spread.
- **Goal:** To take advantage of differences in volatility and time decay between two options.

3. Butterfly Spreads in Time

- **Description:** This strategy involves setting up a butterfly spread (buying one in-the-money option, selling two at-the-money options, and buying one out-of-the-money option) across different expiration dates.
- **Goal:** To profit from the convergence of the term structure toward a normal state if it is currently in an abnormal state.

4. Variance Swaps

- **Description:** Entering into a variance swap contract allows traders to speculate or hedge against the future variability of an asset's price, which can be related to the differences in the implied volatility term structure over time.
- **Goal:** To profit from the difference between the realized variance of an asset's returns and the implied variance from the term structure.

5. Interest Rate Options Arbitrage

- **Description:** Utilizing options on interest rate futures or directly on bonds, traders can exploit inefficiencies in the term structure of interest rates.
- **Goal:** To profit from the mispricing of options based on expected changes in interest rates affecting different maturities differently.

6. Volatility Surface Arbitrage

- **Description:** This involves analyzing the entire volatility surface (across both strike prices and expiration dates) to identify mispricings.
- **Goal:** To exploit discrepancies in implied volatility for options with different strikes and maturities.

7. Dynamic Hedging Strategies

- **Description:** Involves frequently adjusting the composition of a portfolio of options with different expirations to maintain a delta-neutral or gamma-neutral position.
- **Goal:** To profit from changes in the term structure of volatility or interest rates while managing risk.

Implementation Considerations

- **Market Analysis:** Requires deep understanding of the factors influencing option prices, including volatility, interest rates, and their term structures.
- **Risk Management:** Strategies involve significant risks, including the potential for loss if the term structure moves against the position or if the anticipated arbitrage opportunity does not materialize.
- **Execution:** Efficient execution and low transaction costs are crucial due to the potentially small margins of arbitrage opportunities.

(Continued)

(Continued)

These strategies are sophisticated and carry a high level of risk, requiring a solid understanding of options pricing models, market dynamics, and rigorous risk management practices. Traders employ these strategies in markets where they have identified specific inefficiencies or expect certain shifts in the term structure of volatility or interest rates.

It is good that ChatGPT describes a plethora of possible strategies, the simplest of them uses just calendar spreads. So, let's learn more about it.

User Prompt

Elaborate on the calendar spread strategy.

ChatGPT Response

The calendar spread strategy, also known as a time spread or horizontal spread, is a versatile options trading strategy that involves simultaneously entering into two options positions of the same type (either two calls or two puts), same strike price, but with different expiration dates. The trader typically sells a short-term option (the front month) and buys a long-term option (the back month) on the same underlying asset. This strategy is designed to take advantage of differences in time decay (theta) and potentially implied volatility between the two options.

How It Works

- **Short-Term Option (Sell):** By selling a short-term option, the trader collects premium due to the option's higher theta, or time decay, as options lose value faster as they approach expiration.
- **Long-Term Option (Buy):** By buying a long-term option, the trader pays for the option which has a slower rate of time decay. This position benefits from any increase in implied volatility or favorable movements in the underlying asset's price over time.

Objective

The primary goal of a calendar spread is to profit from the differential in time decay between the short-dated and longer-dated options. The ideal scenario is for the short-term option to expire worthless or nearly so, allowing the trader to keep the premium collected, while still holding the longer-term option that retains significant time value.

...

This doesn't sound like a viable strategy at all, since a long-dated option will typically have a higher time value than a short-dated one. If we don't expect any directional movement of the underlying, then after the short-dated option expires, we will just be

losing (time) value of the long-dated option all the way to its expiration. Maybe if we keep rolling forward the short-dated option to the next short-term expiration period this will work, but ChatGPT didn't suggest that.

How about the opposite strategy of shorting the long-dated option and buying the short-dated one? In this case, we have to keep rolling over (buying) the short-dated long position forward as a hedge to the long-dated short position.

In either strategy, we need to compare the annualized premium (price) of the short-dated options with the annualized premium of shorting the long-dated option, assuming there is no change in underlying price nor volatility.

User Prompt

You suggested that we should short a short-term option and buy a long-term option. Let's assume this is a put on the SPX. Can you plot the annualized time values of these two puts as a function of their time-to-expiration?

ChatGPT Response

... [Author Note: See plot in Figure 2.2.]

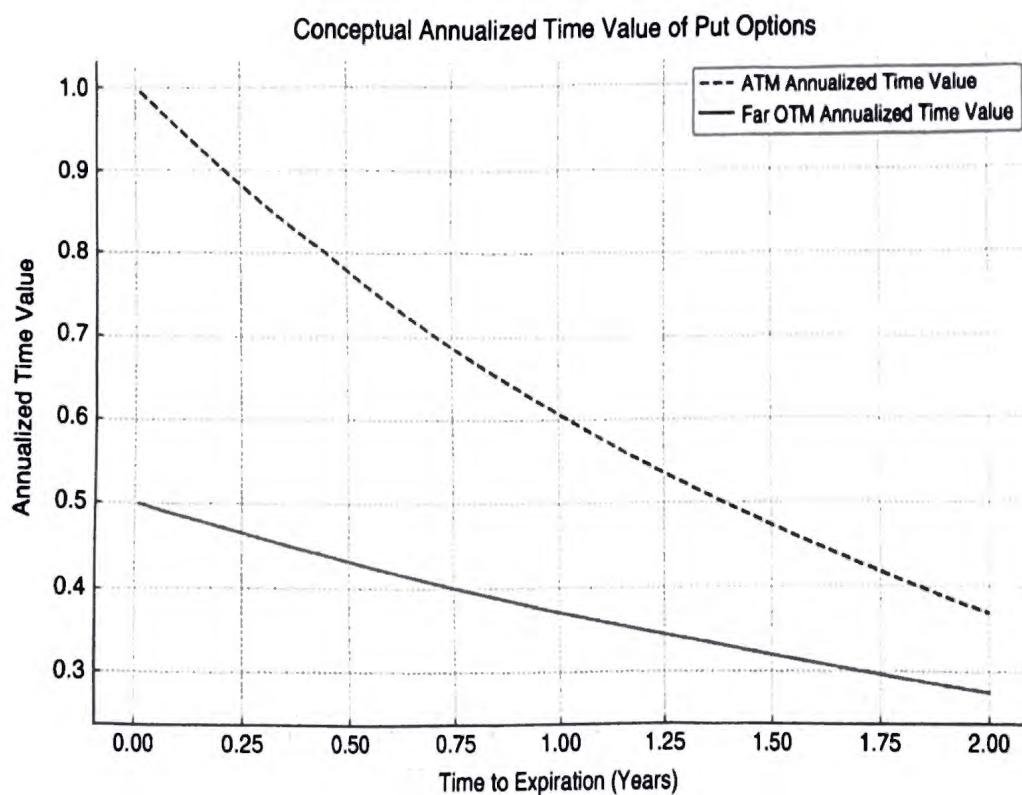


Figure 2.2 Incorrect plot of annualized time value of put options as function of time to expiration.

User Prompt

Can you plot the same for a far out-of-the-money put?

The plot in Figure 2.2 *isn't* correct—we compared these curves with actual option prices we observed from live market data, and they are wildly off the mark for OTM options. Let's explore further.

User Prompt

Please plot the same for implied volatilities this time.

ChatGPT Response

...

Figure 2.3 shows that implied volatility behaves opposite to options prices! That's quite counterintuitive—is it hallucinating? As we have learned, it is better to ask ChatGPT for codes than the output of some unknown models it uses internally. To produce code in this case, we need an options pricing model, but Black-Scholes model won't do.

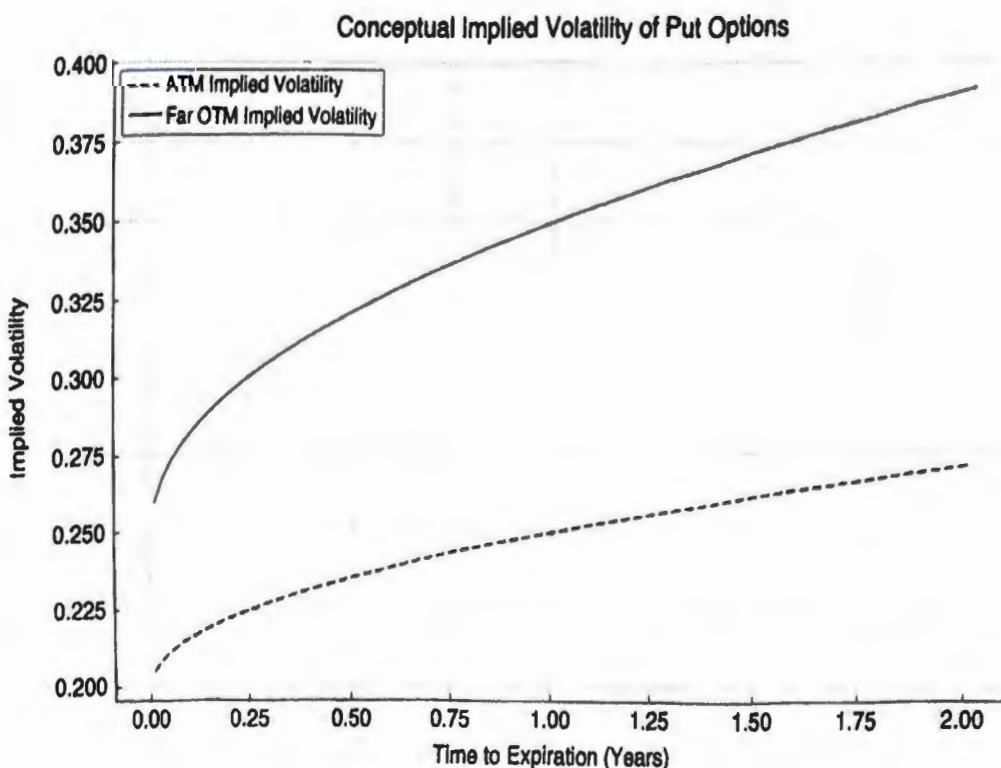


Figure 2.3 Incorrect plot of implied volatility of put options as function of time to expiration.

User Prompt

Do you know of a theoretical option pricing model that exhibits non-trivial volatility surface with such characteristics as the “volatility smile” and the fact that short-dated options have different implied volatility than long-dated options of the same strike, while short-dated options have more negative theta than long-dated ones?

ChatGPT Response

Yes, the phenomena you're describing, including the volatility smile and the observed patterns in implied volatility and theta across different expiration dates, are addressed by several advanced option pricing models that go beyond the Black-Scholes framework. These models incorporate features to account for market imperfections and observed behaviors that the Black-Scholes model, with its simplifying assumptions (e.g., constant volatility, log-normal distribution of stock prices), cannot fully capture. Here are a few theoretical models known for their ability to reflect a more complex and realistic volatility surface:

...

2. Stochastic Volatility Models Examples: Heston model, SABR model. Key Features: In stochastic volatility models, the volatility of the underlying asset is itself a random process. These models can capture the volatility smile because they allow for changes in volatility over time, independent of the underlying asset's price movements. Characteristics: These models can explain various empirical observations, including the volatility smile and changes in implied volatilities across different expiration dates. They also allow for correlation between the asset price and its volatility, which can affect the pricing of short-dated versus long-dated options.

...

ChatGPT listed four types of options pricing model: local volatility models that use options prices to calibrate the volatility surface at different expirations and strikes; stochastic volatility models (such as Heston) that use a few parameters to produce the general shape of the volatility surface; jump-diffusion model that allows for jumps in the price of the underlying in addition to Gaussian Brownian Motion; and the Levy Process Models that model the movement of the underlying asset as a Levy process. The Heston model seems the simplest and most appropriate if we just want to get a conceptual understanding of the term structure without getting an exact match with real data.

User Prompt

Let's pick the Heston model. Can you elaborate on that?

ChatGPT Response

The Heston model is a popular stochastic volatility model for pricing options, introduced by Steven Heston in 1993. It addresses some of the limitations of the Black-Scholes model, particularly its assumption of constant volatility. The Heston model allows the volatility of the underlying asset to be stochastic, meaning it can change over time according to a specific stochastic process. This feature enables the model to more accurately capture market phenomena such as the volatility smile.

Key Features of the Heston Model

- 1. Stochastic Volatility:** The model describes the evolution of the underlying asset's price and its variance (volatility squared) using two stochastic differential equations (SDEs). The first SDE models the asset price dynamics similarly to Black-Scholes but incorporates a stochastic volatility term. The second SDE models the dynamics of the variance itself.
- 2. Mean Reversion:** The variance process in the Heston model is mean-reverting, meaning it tends to return to a long-term average level over time. This is consistent with observed market behavior, where volatility spikes are usually temporary.
- 3. Correlation Between Asset Returns and Volatility:** The model allows for a correlation between the asset returns and volatility changes. This correlation is crucial for accurately capturing the leverage effect, where asset prices and their volatility tend to be inversely related.

Mathematical Formulation

The dynamics of the underlying asset price (S) and its variance (ν) in the Heston model are described by the following SDEs:

$$\begin{aligned} dS_t &= \mu S_t dt + \sqrt{\nu_t} S_t dW_t^S \\ d\nu_t &= \kappa(\theta - \nu_t)dt + \sigma\sqrt{\nu_t} dW_t^\nu \end{aligned}$$

where:

- S_t is the asset price at time t .
- ν_t is the variance of the asset price at time t .
- μ is the risk-free rate of return. [Author Note: This isn't correct! It should be the deterministic (i.e., non-stochastic) instantaneous drift of asset price returns! See Gatheral (2006).]
- κ is the rate of mean reversion of the variance.
- θ is the long-term variance mean level.
- σ is the volatility of the volatility (vol of vol), which determines the volatility of the variance process.
- dW_t^S and dW_t^ν are two Wiener processes for the asset price and variance, respectively, with a correlation coefficient ρ .

Pricing Options with the Heston Model

The Heston model allows for the derivation of a semi-analytical formula for the price of European-style options. The option pricing formula involves complex integrations due to the model's stochastic nature but can be efficiently computed using numerical methods such as the fast Fourier transform (FFT).

Applications and Limitations

- **Applications:** The Heston model is widely used in financial markets for pricing European options, capturing the volatility surface more accurately than the Black-Scholes model, and is also extended to price American options and exotic options under certain conditions.
- **Limitations:** Despite its advanced features, the Heston model can be challenging to calibrate to market data due to the complexity of its dynamics and the sensitivity of its parameters. Moreover, like all models, it makes simplifications and assumptions that may not hold under all market conditions.

The Heston model's ability to incorporate stochastic volatility and mean reversion makes it a powerful tool for understanding and pricing options in a more realistic manner, reflecting the dynamic nature of financial markets.

User Prompt

Please cite any publications that show the term structure of implied volatility of an option as function of time-to-expiration.

ChatGPT Response

...

Books and Papers

1. **Option Volatility and Pricing by Sheldon Natenberg** - This book is a widely regarded resource in the field of options trading. Natenberg provides detailed explanations of the behavior of implied volatility and its impact on options pricing, including discussions on the term structure of volatility.
2. **Dynamic Hedging by Nassim Nicholas Taleb** - Taleb's work focuses on the practical aspects of managing option positions, including insights into the term structure of volatility and how it can be exploited through dynamic hedging strategies.
3. **The Volatility Surface: A Practitioner's Guide by Jim Gatheral** - Gatheral provides an in-depth analysis of the volatility surface, which includes discussions on the term structure of implied volatility and how it varies with time to expiration and strike price.

...

We pick Jim Gatheral's book (Gatheral, 2006) as our reference as it seems to cover the relevant topic with lots of charts. Unfortunately, none of which is exactly what we want: a comparison of the term structure of ATM and OTM option prices based on the Heston model. We need to instantiate a Heston model to do that. But first, we need to know what inputs are reasonable.

User Prompt

What are reasonable estimates for μ when the asset in question is the SPX?

ChatGPT Response

The parameter μ in financial models typically represents the expected return or drift rate of the underlying asset over time. For the S&P 500 Index (SPX), reasonable estimates for μ should reflect the long-term average return of the index, accounting for dividends, inflation, and the overall growth of the economy.

Historical Perspective on μ for the SPX

The S&P 500 has historically returned about 7% to 10% annually after adjusting for inflation. This range accounts for dividends reinvested and represents a long-term perspective over multiple decades. The exact figure can vary depending on the specific time period considered, how dividends are treated, and the inflation adjustment method.

...

User Prompt

What are reasonable estimates for the rate of mean reversion of the variance when the asset in question is the SPX?

ChatGPT Response

The rate of mean reversion κ in the Heston model is a key parameter that describes how quickly the variance of the asset returns reverts to its long-term mean θ . For the S&P 500 Index (SPX), as with other financial assets, the specific value of κ can vary based on market conditions, the time period analyzed, and the calibration methodology used. However, empirical studies and market practice provide some guidance on reasonable estimates for this parameter.

Reasonable Estimates for κ

- **Typical Range:** For equity indices like the SPX, κ values are often found in the range of 1 to 3 per year. This range suggests that the variance tends to revert to its long-term mean within a few months to a year.

...

User Prompt

What are reasonable estimates for long-term variance mean level for SPX?

ChatGPT Response

The long-term variance mean level θ in the Heston model represents the level to which the variance of the asset returns is expected to revert over time. For the S&P 500 Index (SPX), as with other assets, this parameter is crucial for modeling the long-term behavior of market volatility. Estimating θ involves analyzing historical volatility data and calibrating the model to current market prices of options.

Reasonable Estimates for θ

Given that θ is the long-term mean of the variance, not volatility, it's important to distinguish between the two when discussing estimates. Variance ν_t is the square of volatility σ [Author Note: This is wrong. σ is the volatility of volatility here, not the volatility], so when practitioners refer to volatility levels, they're often discussing σ [Author Note: same error as above], whereas θ pertains to ν_t .

- **Converting Volatility to Variance:** To think about θ in terms of familiar volatility levels, you first need to square the annualized volatility. For example, if the long-term volatility expectation for the SPX is around 20% (or 0.20 in decimal form), the corresponding variance θ would be ($0.20^2 = 0.04$).
- **Typical Range for SPX:** Historically, the annualized volatility of the SPX has fluctuated, but a range of 15% to 30% (0.15 to 0.30 in decimal form) is common for estimating long-term volatility, depending on the time period and market conditions. Squaring these values gives a variance range of approximately 0.0225 to 0.09 for θ .

User Prompt

What are reasonable estimates for σ for SPX?

ChatGPT Response

In the context of the Heston model, σ represents the volatility of volatility (vol of vol), which quantifies the variability in the implied volatility of the underlying asset. This parameter is crucial for capturing the dynamics of the volatility surface, including the curvature and behavior of the implied volatility skew. For the S&P 500 Index (SPX), reasonable estimates for σ can vary based on historical market conditions, the calibration period, and the specific model calibration approach. [Author Note: In this description, it got right the parameters σ , which as described stands for the volatility of volatility.]

(Continued)

(Continued)

Reasonable Estimates for σ

- **Typical Range:** For the SPX, reasonable estimates for σ (volatility of volatility) often fall within the range of 0.1 to 0.4 per annum. This range reflects the observed variability in the market's implied volatility but can vary depending on the specific market environment and historical period analyzed.

User Prompt

What are reasonable estimates for correlation coefficient ρ for SPX?

ChatGPT Response

The correlation coefficient ρ in the context of the Heston model represents the correlation between the asset's returns and its volatility. For the S&P 500 Index (SPX), as well as other assets, this correlation tends to be negative, reflecting the common market observation that asset prices often move inversely to their volatility. This phenomenon is sometimes referred to as the "leverage effect," where market downturns are typically associated with increases in volatility.

Reasonable Estimates for ρ - Typical Range: For the SPX, reasonable estimates for the correlation coefficient ρ often fall within the range of (-0.5) to (-0.8). This range is consistent with empirical studies that have examined the relationship between stock returns and volatility.

We will generally pick the middle value of a given range of just provided estimates, except that we use the low end of the estimate of θ (long-term variance of SPX) as volatility level has dropped over the years, but a high-end estimate of σ (volatility of volatility of SPX) as that has increased over the years (see Chan (2013), Figure 5.10). We also pick the high-end estimate of μ (long-term return of SPX) because it has trended higher in recent years.

User Prompt

Let's assume $\kappa = 2, \theta = 0.15^2, \sigma = 0.4, \rho = -0.65, \mu = 0.1$, initial variance $v_0 = 0.15^2$. Display the Matlab code for computing the ATM option price using its Financial Toolbox.

ChatGPT Response

To compute the price of an At-The-Money (ATM) option on the SPX using the Heston model in MATLAB, you would typically use functions provided by MATLAB's Financial Toolbox. The Heston model parameters you've specified are:

- Rate of mean reversion, $\kappa = 2$

- Long-term variance mean level, $\theta = 0.15^2$
- Volatility of volatility, $\sigma = 0.4$
- Correlation coefficient, $\rho = -0.65$
- Expected return or drift rate, $\mu = 0.1$
- Initial variance, $v_0 = 0.15^2$

Here's a conceptual MATLAB script to calculate the price of an ATM option using the Heston model. Note that MATLAB's Financial Toolbox may require specific functions for option pricing, such as `optByHestonNI` (using numerical integration) to compute European option prices.

```
% Heston model parameters
kappa = 2;
theta = 0.15^2;
sigma = 0.4;
rho = -0.65;
v0 = 0.15^2;
r = 0.1; % Risk-free rate, assuming it's the same as mu for simplicity
% Author Note: this is wrong! r should be 0.03.

% Define the asset path
S0 = 100; % Example initial SPX level, adjust as needed [Author Note: should
be about 5,000 as of February, 2024.]
K = S0; % ATM option
T = 1; % Time to maturity in years

% Define option type
OptType = {'call', 'put'}; % Choose as needed

% Pricing the option using Heston model
for i = 1:length(OptType)
    price = optByHestonNI(kappa, theta, sigma, rho, v0, r, T, S0, K,
    'OptSpec', OptType{i});
    fprintf('%s option price using Heston model: %f\n', OptType{i}, price);
end
```

This script uses the `optByHestonNI` function to calculate option prices based on the Heston model parameters. The `OptSpec` option allows you to specify the type of option ('call' or 'put'). Adjust `S0`, `K`, and `T` as needed to reflect the current price of the SPX, the strike price for the ATM option, and the desired time to maturity, respectively.

Please ensure you have the Financial Toolbox installed and consult the MATLAB documentation for the latest functions and syntax, as function names and parameters can change between releases. This example assumes a simplified scenario for illustrative purposes.

We don't completely trust the codes generated, but we are glad that ChatGPT pointed to the Matlab Financial Instruments Toolbox (though not the Financial Toolbox that it suggested) function `optByHestonNI`. We will reference the official documentation at https://www.mathworks.com/help/fininst/optbyhestonni.html?s_tid=doc_ta#d126e204312 to clean up this script. Our final version is saved in GitHub as `HestonModelPutPlot.m`. The plot is displayed in Figure 2.4. The one for calls is saved in `HestonModelCallPlot.m` and displayed in Figure 2.5.

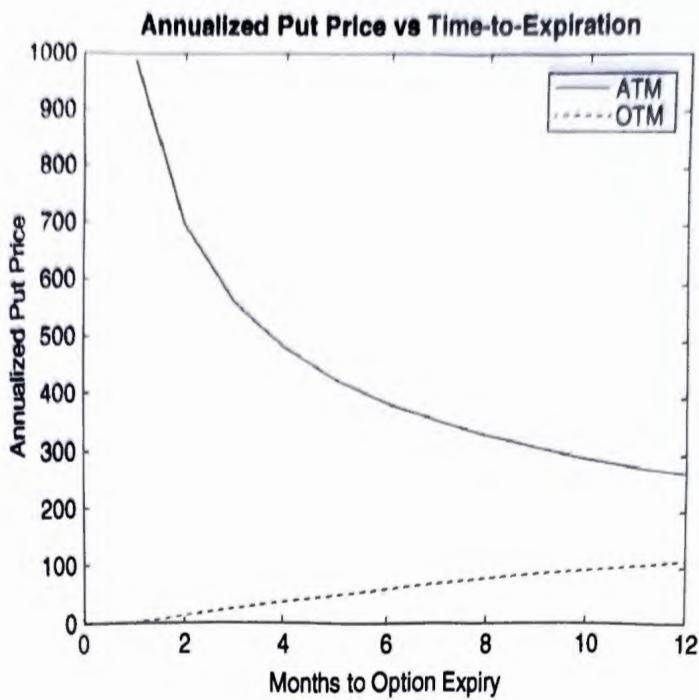


Figure 2.4 Annualized put option prices as function of time to expiration based on the Heston model.

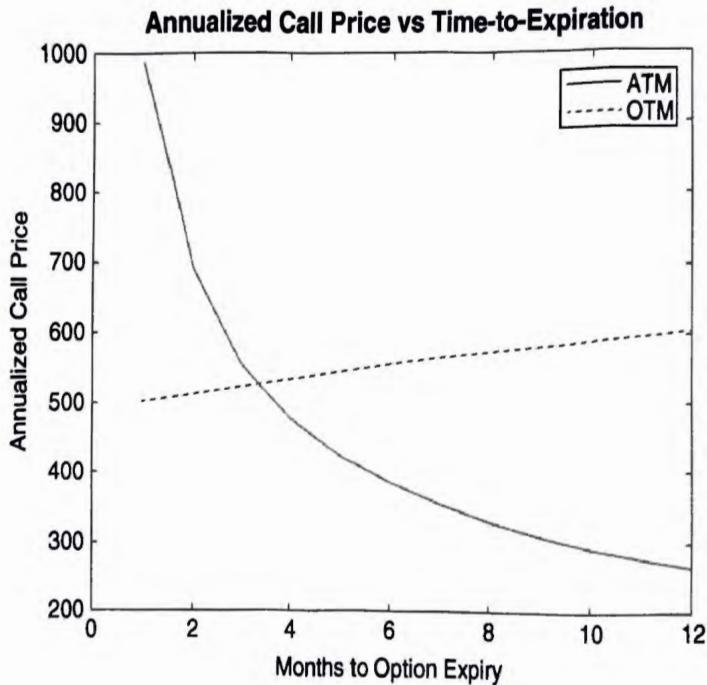


Figure 2.5 Annualized call option prices as function of time to expiration based on the Heston model.

★ ★ ★

Exercise: Ask ChatGPT for an equivalent Python version.

★ ★ ★

Based on these curves, it seems that we can short a long-dated OTM option and hedge that by buying a short-dated OTM option that is continuously being rolled forward before its expiration, unless it becomes an ATM option at some point due to SPX level changes. At that point, we should liquidate the positions because the cost of hedging the long-dated option will outweigh the annualized time value of that option. The opposite strategy, buying a long-dated ATM option and shorting a short-dated option won't work because when the short-dated option expires, the long-dated option will most likely not be ATM anymore and the noted term structure won't apply (unless we expect a catalyst in the market that might produce significant move in the underlying after the first option expires, such that the long-dated option becomes at the money). There is a wide range of SPX levels that make the options OTM, but a narrow range that make them ATM.

What about the implied volatility term structure? Are they the same or opposite to the annualized option value as ChatGPT suggested earlier? But wait, recall that in the Heston model, volatility is stochastic and changes constantly. What does the implied volatility term structure (also known as the volatility surface) even mean?

User Prompt

Given that Heston model uses a stochastic volatility model as the basis, does it even make sense to ask for the implied volatility of an option since it is stochastic over time?

ChatGPT Response

Your question touches on a nuanced aspect of option pricing theory and the interpretation of implied volatility within the context of stochastic volatility models like the Heston model.

Implied Volatility in the Black-Scholes Framework In the Black-Scholes model, implied volatility is a constant value that, when input into the model, produces an option price matching the market price. It represents a single, average measure of expected volatility over the life of the option. This model assumes volatility is known and constant, which simplifies the calculation but does not capture the complexity of real market conditions.

Stochastic Volatility Models Stochastic volatility models, including the Heston model, recognize that volatility is not constant but varies over time in a random manner. These models provide a more realistic framework for option pricing by incorporating a stochastic process for volatility.

Implied Volatility within Stochastic Volatility Models Even though the Heston model and other stochastic volatility models use a dynamic view of volatility, practitioners still compute implied volatilities from these models for several reasons:

- Market Benchmark:** Implied volatility serves as a standardized benchmark, allowing traders and analysts to compare option prices across different strikes and maturities in a consistent way, even within the context of models that recognize volatility's dynamic nature.

(Continued)

(Continued)

2. **Model Calibration:** Implied volatility is used to calibrate stochastic volatility models to market data. By adjusting model parameters to match the observed market implied volatilities across a range of strikes and maturities, practitioners ensure that the model reflects current market conditions as closely as possible.
3. **Volatility Surface:** The concept of implied volatility is central to constructing the volatility surface, which plots implied volatilities across different strikes and maturities. Even in the context of stochastic volatility models, the volatility surface provides valuable insights into market expectations and sentiment.
4. **Communication Tool:** Despite its limitations, implied volatility remains a useful tool for communicating market conditions and risk perceptions among traders, analysts, and other market participants.

Conclusion While it might seem counterintuitive to discuss implied volatility in the context of a model that inherently acknowledges volatility's stochastic nature, doing so allows for the practical application of these models in real-world trading and risk management. Implied volatility, derived from market prices of options, bridges the gap between theoretical models and market practice, serving as a common language for market participants and a tool for model calibration.

Unlike what academic theoreticians often do, we will not bother with plotting the volatility surface. It isn't empirically observable, and we can't make money from it directly. As traders, we prefer things that we can actually trade.

2.5 Conclusion

We asked ChatGPT to create the backtest codes for a long-short equity trading strategy using the Fama-French factors, and a hedged VIX futures carry strategy. We also asked ChatGPT to suggest an options calendar spread strategy and explore the term structure of options prices. Less demandingly, we asked Copilot to find out if someone has published a deep reinforcement learning technique for portfolio optimization. From these experiments, we conclude the following:

1. We can't really use ChatGPT to automate the entire process. What is needed is a combination of human intelligence, knowledge, experience and the LLM's suggestions. The situation is quite analogous to self-driving cars. Fully autonomous vehicles are rarely found except in a few experimental urban areas, but assisted driving technology has been embedded in almost all new cars for years already (Chan, 2022). We are pretty far from AGI (artificial general intelligence).
2. The responses from ChatGPT are often not reproducible: the same prompt can result in different responses. This is difficult for most engineering projects that require strict reproducibility.
3. It may be more efficient to use ChatGPT as a coach to speed up your own coding tasks than to ask it to write good codes every time you need one.

These conclusions are not dissimilar to those of Simon Willison (Django and Datasette creator) as reported in Claburn (2024). We need a “human in the loop” for the most serious applications of AI.

That is not to say that there aren’t occasional flashes of brilliance or evidence of conceptual understanding of quantitative finance, such as when ChatGPT understands that a trading strategy that holds a position for N days may have up to N positions on any single day, or whether subtracting the risk-free rate from the gross returns is always necessary when computing the Sharpe ratio. These “sparks of AGI” are well-documented in the paper by Bubeck et al. (2023). But such brilliance vanished when we asked ChatGPT to analyze the VIX paper and produce pseudocodes for the carefully described trading strategy contained therein. Even after pointing ChatGPT to the specific paragraph with the relevant information, it failed to capture the entire trading strategy. Perhaps it is just a limitation of how long a paper GPT-4o can handle at this time, but we can only say that the state-of-the-art available to a professional trader is simply not adequate as of this writing. As our effort to locate a paper on applying deep reinforcement learning to portfolio optimization shows, Copilot isn’t very good at locating information that meet our specific professional needs. We can perhaps improve that ability by fine-tuning it on a quantitative finance database, or with particularly expert prompt engineering. For example, it has been known that “chain-of-thought prompting,” i.e., “asking language models to generate step-by-step solutions enabled the models to solve problems that had previously seemed beyond their reach” (Brukbaker, 2024). But if such highly technical human skills are required of the user, it can hardly be called AGI. Of course, with each new generation of GPT (we are using GPT-4o as of this writing), its capability will improve. What ChatGPT and other Gen AI tools are good at is providing a first draft of something that may be partially correct. As such it is useful as a junior collaborator, motivating the senior human collaborator to try new things. Often, the first draft is the most difficult (a.k.a., writer’s block). Revisions and improvements are easier for humans. So, if you ask us: “After writing these two chapters, would you still use Gen AI to help with investment strategies discovery and backtest?” The answer is yes.

You may wonder, given the state of Gen AI, why we should dive deeper into Gen AI later on. The reason is this: most of the tasks we have given Gen AI so far are those that have a clear answer, and human experts can perform them equally well if not better. Of course, not all of us are experts in every subject, and it saves us time to get AI to do what we know how to do if it can do it well. That was the reason we explored no-code AI. However, there are tasks where there are no clear answers, and even human experts may disagree on what the best answers are. In medical diagnostics, we may ask, “Given the symptoms, what is the likely disease affecting the patient?” In asset management, “Will NVDA increase its earnings next quarter?”, “Will oil price go down next month?”, or “Is what the Fed chair just said in the press conference positive or negative for the S&P 500?” (The Gen AI answer to the last question is actually an example later in this book.) These are all questions with no clear answers, except in hindsight. This is the realm of discretionary or fundamental trading. Because machines and humans are on more equal

footing for these questions, we may find that Gen AI can add more value. We want to find out if building a “George Soros on a chip” is possible.

The following chapter will set the stage for this ambitious project. This will inevitably involve usage of Generative AI APIs.

★ ★ *

2.6 Exercises

1. Ask ChatGPT to create the backtest code for the paper Zhang et al. (2019) on Deep Reinforcement Learning for Trading. Does ChatGPT use Tensorflow? Can you create unit tests to see if it produces the codes correctly?
2. What is Retrieval Augmented Generation (RAG), and how would it correct some of the hallucinations the LLMs produce?
3. Ask the latest version of ChatGPT to see if it can now summarize the VIX strategy from the cited paper correctly.
4. Ask ChatGPT to create the code to compute the expected profit of an iron condor for SPX options with a holding period of a month, assuming normal distribution of returns and reasonable volatility estimates. Watch how it does numerical integration with aplomb! Bonus question: ask it to generate the code to compute its Sharpe ratio too.

Appendix 2A.1 Computing Next-day's Return

Here is the Python program that computes the next-day (or in general, next-N-day) returns. These returns will be used as “labels” (or the Y variable) for the Fama-French regression model.

```

import pandas as pd

def calculate_N_day_ahead_returns(file_path, output_file_path, N=1):
    try:
        # Load the data
        stock_prices = pd.read_csv(file_path)

        # Rename column names
        stock_prices = stock_prices.rename(columns={'tday': 'Date',
        'syms': 'Stock_ID', 'mid': 'Price'})

        # Convert the 'Date' column from integer YYYYMMDD format to
        # datetime format
        stock_prices['Date'] = pd.to_datetime(stock_prices['Date'],
        format='%Y%m%d')

        # Sort the data by 'Stock_ID' and 'Date'
        stock_prices.sort_values(by=['Stock_ID', 'Date'], inplace=True)

        # Shift the prices by -N days to align with the price N days ahead
        stock_prices['Price_in_N_days'] = stock_prices.groupby('Stock_ID')[['Price']].shift(-N)

        # Calculate the N-day returns
        stock_prices[str(N) + '_Day_Returns'] = (stock_prices['Price_in_N_
        days'] - stock_prices['Price']) / stock_prices['Price']

        # Dropping the column used for calculation
        stock_prices.drop(columns=['Price_in_N_days'], inplace=True)

        # Save the results to a new CSV file
        stock_prices.to_csv(output_file_path, index=False)

    return "Successfully computed 30-day returns and saved to " +
output_file_path
    except Exception as e:
        return f"An error occurred: {e}"

# Example usage
file_path = 'CRSP_mid.csv' # Replace with the actual file path
output_file_path = 'CRSP_nextDayReturns.csv' # Replace with the desired
output file path
result = calculate_N_day_returns(file_path, output_file_path)
print(result)

```

Appendix 2A.2 Uploading the Fama-French Factors

The Fama-French factors file was downloaded from https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html. We just need to create a Python Pandas dataframe to hold the data with proper dates.

```
import pandas as pd

# Path to the file (replace with your file path)
file_path = 'F-F_Research_Data_Factors_daily.CSV'

# First attempt: Skipping initial rows that might contain descriptions
# or headers
ff_data_1 = pd.read_csv(file_path, skiprows=3)

# Assuming successful read, filtering out non-date entries
ff_data_filtered = ff_data_1[ff_data_1['Unnamed: 0'].apply(lambda x: str(x).isdigit())]

# Rename first column to 'Date'
ff_data_filtered.rename(columns={ff_data_filtered.columns[0]: 'Date'}, inplace=True)

# Change first column to datetime type
ff_data_filtered ['Date'] = pd.to_datetime(ff_data_filtered['Date'])

# Example query: Finding the SMB factor on a specific date
smb_factor = ff_data_filtered[ff_data_filtered['Date'] == '20151106']['SMB'].iloc[0]
# This returns the correct numerical value of 0.87
```

Including a “unit-test” like the last line of this code is important in any program, but especially for those generated by ChatGPT.

Now, we need to combine the daily returns data and the factor data correctly, using the former as the “label” or “Y” variable in our regression fit and the latter as “features” or “X” variable. Prompting ChatGPT for that, the response more or less looks like Appendix 2.3, which follows, with some human editing.

Appendix 2A.3 Combining Fama-French Factors with Next-day's Returns

We have created the dataframe that contained the Fama-French factors as features for our training data. Now we need to combine them with the labels, which are the next-day's returns, in order to form the complete dataframe for training and testing a linear regression model.

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Assuming ff_data_filtered and next_day_returns_data are your loaded
DataFrames
```

```

## Load and parse daily returns of stocks
next_day_returns_data = pd.read_csv('CRSP_nextDayReturns.csv')
next_day_returns_data['Date'] = pd.to_datetime(next_day_returns_data['Date'],
format='%Y-%m-%d')
next_day_returns_data['Date'] = pd.to_datetime(next_day_returns_data['Date'],
format='%Y%m%d')

## Load and parse Fama-French data
file_path = 'F-F_Research_Data_Factors_daily.CSV'
# First attempt: Skipping initial rows that might contain descriptions
or headers
ff_data_1 = pd.read_csv(file_path, skiprows=3)

# Assuming successful read, filtering out non-date entries
ff_data_filtered = ff_data_1[ff_data_1['Unnamed: 0'].apply(lambda x: str(x).
isdigit())]

# Rename first column to 'Date'
ff_data_filtered.rename(columns={ff_data_filtered.columns[0]: 'Date'},
inplace=True)

# Change first column to datetime type
ff_data_filtered ['Date'] = pd.to_datetime(ff_data_filtered['Date'])

# Example query: Finding the SMB factor on a specific date
smb_factor = ff_data_filtered[ff_data_filtered['Date'] ==
'20151106']['SMB'].iloc[0]
# This returns the correct numerical value of 0.87

# Merge the datasets on the 'Date' column
combined_data = pd.merge(ff_data_filtered, next_day_returns_data, on='Date')

# Unit test
# Function to perform the unit test
def test_factors_on_date(data, date, expected_smb, expected_hml):
    # Extracting the row for the given date
    data_on_date = data[data['Date'] == date]

    # Checking if the date exists in the data
    if data_on_date.empty:
        return "Date not found in the data."

    # Extracting the SMB and HML values
    smb_value = data_on_date['SMB'].iloc[0]
    hml_value = data_on_date['HML'].iloc[0]

    # Testing the conditions
    smb_test = abs(smb_value - expected_smb) < 0.01 # Approximate check
    hml_test = abs(hml_value - expected_hml) < 0.01 # Approximate check

    return {
        "SMB Test Passed": smb_test,
        "HML Test Passed": hml_test,
        "Actual SMB Value": smb_value,
        "Actual HML Value": hml_value
    }

# Performing the test for 20151106

```

(Continued)

(Continued)

```

test_results = test_factors_on_date(combined_data, '20100423', 0.37, 0.62)
test_results

# Function to perform the unit test for MSFT 1 day return
def test_msft_return_on_date(data, date, expected_return):
    # Extracting the row for the given date
    data_on_date = data[(data['Date'] == date) & (data['Stock_ID'] == 'MSFT')]
    print(data_on_date)

    # Checking if the date exists in the data
    if data_on_date.empty:
        return "Date not found in the data."

    # Extracting the MSFT 1 day return value
    msft_return_value = data_on_date['1_Day_Returns'].iloc[0]

    # Testing the condition
    return_test = abs(msft_return_value - expected_return) < 0.01  # Approximate check

    return {
        "MSFT Return Test Passed": return_test,
        "Actual MSFT 1 Day Return Value": msft_return_value
    }

test_msft_return_on_date(combined_data, '20100423', 0.00508690122)
{'MSFT Return Test Passed': True,
 'Actual MSFT 1 Day Return Value': 0.0035501048894635}

test_msft_return_on_date(combined_data, '20081021', -0.078)
{'MSFT Return Test Passed': True,
 'Actual MSFT 1 Day Return Value': -0.0777468408652821}

```

It passed all the unit tests.