Analysis of Options Hedging Strategies

Marius Boda

December 4, 2025

Contents

1 Introduction

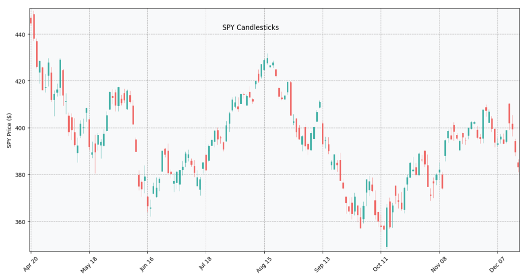
This project implements and analyzes several options hedging strategies, focusing on delta and delta-vega hedging of call options on the SPDR S&P 500 ETF (SPY). The analysis is based on real market data and aims to provide quantitative insights into the effectiveness of these risk management techniques. The core of the project involves simulating hedging strategies under various rehedging frequencies and market conditions, and evaluating their performance based on metrics like hedging error, profit and loss (PnL), and transaction costs.

For delta hedging, the analysis was conducted using five distinct datasets running a total of 157 simulations over 45-day intervals with daily rehedging. This simulation allows for a robust statistical analysis of the hedging outcomes. The project examines how different hedging strategies and parameters affect risk management effectiveness in options portfolios. The primary tool for this analysis is Python, with libraries such as Pandas, NumPy, and Matplotlib for data manipulation, computation, and visualization.

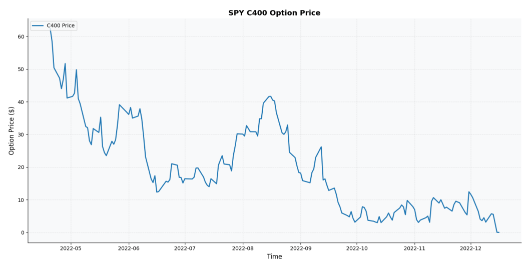
2 Data

The data used for this project was downloaded from the Refinitiv Workspace. It consists of daily price data for SPY and several of its call options with different strikes and maturities. The data spans from April 2022 to December 2022. Each dataset contains Open, High, Low, and Close prices for the underlying SPY ETF, along with the prices of the corresponding call options. The call options data was not always 100% clean, sometimes some days did not have any data entries, for this reason some of the simulations were more difficult as skipping a day of price change will affect the rehedging.

As an example of one of the data-frames, one of the datasets contains 168 data points, starting from 2022-04-20 and ending on 2022-12-16. This provides a good long time series data to conduct meaningful hedging simulations over lots of 45-day rolling windows. The data was loaded and processed using a custom Python function, ensuring that the dates are correctly parsed and the data is ready for analysis.



(a) SPY Underlying Price



(b) C400 Option Price

Figure 1: SPY Underlying Price and C400 Option Price.

Figure 1 shows an example of the underlying SPY price movement and the corresponding price of a call option.

3 Single Option Hedging

3.1 Delta Hedging a Single Option

Delta hedging is a strategy that aims to reduce the directional risk of an option position by taking an offsetting (or inverse) position in the underlying asset. The delta of an option is a measure of the rate of change of the option's price with respect to a change in the underlying asset's price. For a long call option, the delta is positive and so the respective hedging strategy takes a short in the underlying asset.

A simple delta hedging simulation was performed on a SPY Call C400 option with a maturity of 16/12/2022, using the last 45 days as the hedging period. The portfolio is rebalanced daily to maintain a delta-neutral position. Figure 2 shows the portfolio positions for the delta hedge, and Figure 3 shows the delta positions.

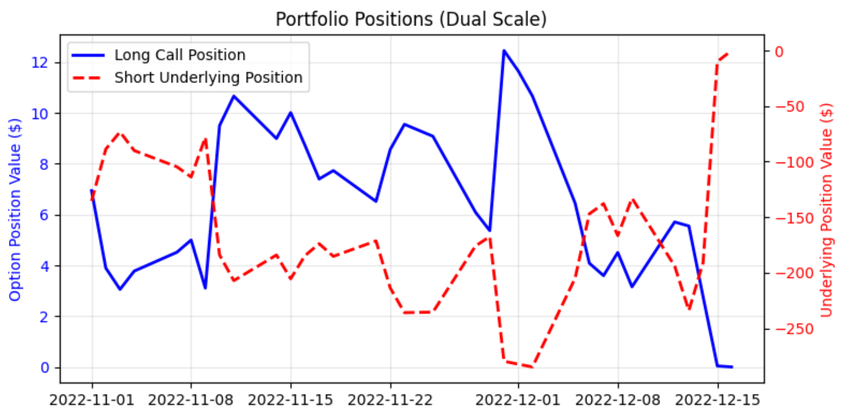


Figure 2: Portfolio positions for a delta hedge, showing the long call position and the short position in the underlying. A dual scale is used.

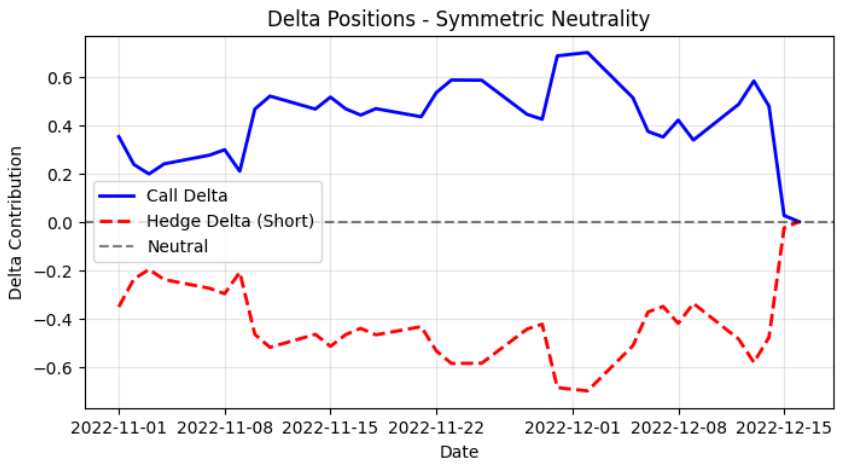


Figure 3: Delta positions for the C400 option hedge.

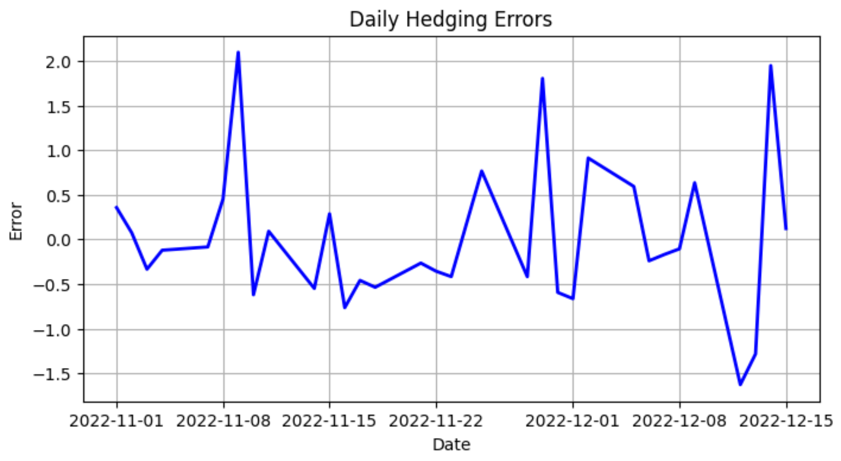


Figure 4: Daily Hedging Errors for a simple delta hedge.

Figure 4 shows the daily hedging errors for this simulation, which was performed using the `simple_delta_hedging` function from the `hedging.py` module (see appendix). The function's logic is: it iterates daily through the hedging period, first doing a calculation for the implied volatility from the option's market price and then using it to find the Black-Scholes delta. The daily hedging error is then calculated as the difference between the change in the option's price and the change in the delta-hedged position in the underlying asset. Finally, the function computes the Mean Squared Hedging Error (MSE) for the period. For this simple simulation it was 0.6786. A lower MSE indicates a more accurate hedge. The error is the difference between the change in the option's value and the change in the value of the replicating portfolio. For simple delta hedging it was common to see hedging errors of under 1.0 indicating decent performance. In the later sections we explore Delta-Vega hedging and see that it is a bit more difficult to get a lower MSE.

3.2 Delta-Vega Hedging a Single Option

While delta hedging neutralizes the portfolio's sensitivity to small changes in the underlying asset's price, it does not protect against changes in other market parameters, such as implied volatility. Vega measures the sensitivity of an option's price to changes in volatility. A delta-vega hedge aims to create a portfolio that is neutral to both delta and vega risk.

However, to hedge vega, another instrument whose value also depends on volatility is required. Typically, this is another option. Therefore, a true delta-vega hedge requires a portfolio of at least two options, which will be discussed in detail in Section 4.

3.3 Hedging Accuracy Compared

The accuracy of a hedging strategy is measured by its hedging error. A more accurate hedge will have a lower error. In the context of a single option, we can compare a simple delta hedge (without costs) to a more realistic one that includes transaction costs. The introduction of costs does not change the theoretical hedging error calculation but impacts the final Profit and Loss (PnL) of the strategy. The simple delta hedge simulation resulted in an MSE of 0.6786. When comparing different strategies, such as delta vs. delta-vega, we expect the latter to be more accurate in volatile markets, as it neutralizes an additional risk factor. This comparison is made in Section 4.

3.4 Effect of Adding Transaction Costs

In a realistic setting, every transaction incurs costs. The simulation was extended to include transaction costs to provide a more realistic measure of the profitability of the hedging strategy. The costs were modeled as a combination of a fixed cost per share and a percentage of the trade value (specifically, 0.01 per share and 0.05% of the trade value).

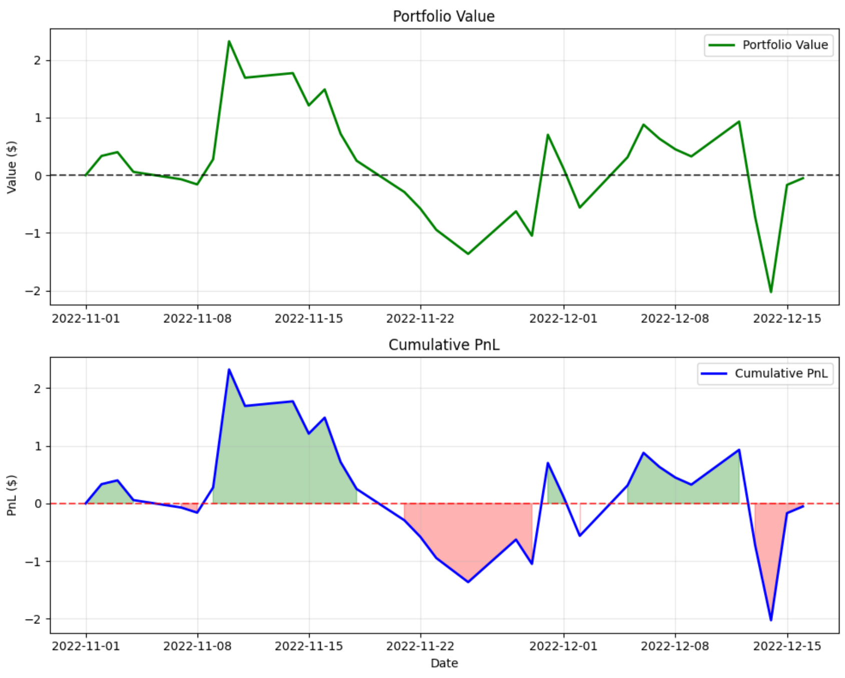


Figure 5: Portfolio Value and Cumulative PnL with Transaction Costs.

Figure 5 shows the portfolio value and cumulative PnL for the delta hedging simulation with transaction costs. This was achieved by calling the more comprehensive `delta_hedging` function, which includes parameters for transaction costs. See ‘hedging.py‘ in the appendix for details.

The PnL fluctuates around zero, indicating that the hedge is effective at reducing risk, but the cumulative costs cause a clear drag on the final performance, resulting in a negative final PnL.

4 Portfolio of Options

4.1 Delta Hedging a Portfolio of Options

To understand the performance of delta hedging across a wider range of conditions, 157 simulations on different options and time periods were simulated. This was done to get statistical averages and see the metrics. This large-scale analysis was performed using the `run_hedging_intervals` function, which repeatedly calls the `delta_hedging` function over different time windows. The full code is in the appendix (‘hedging.py‘).

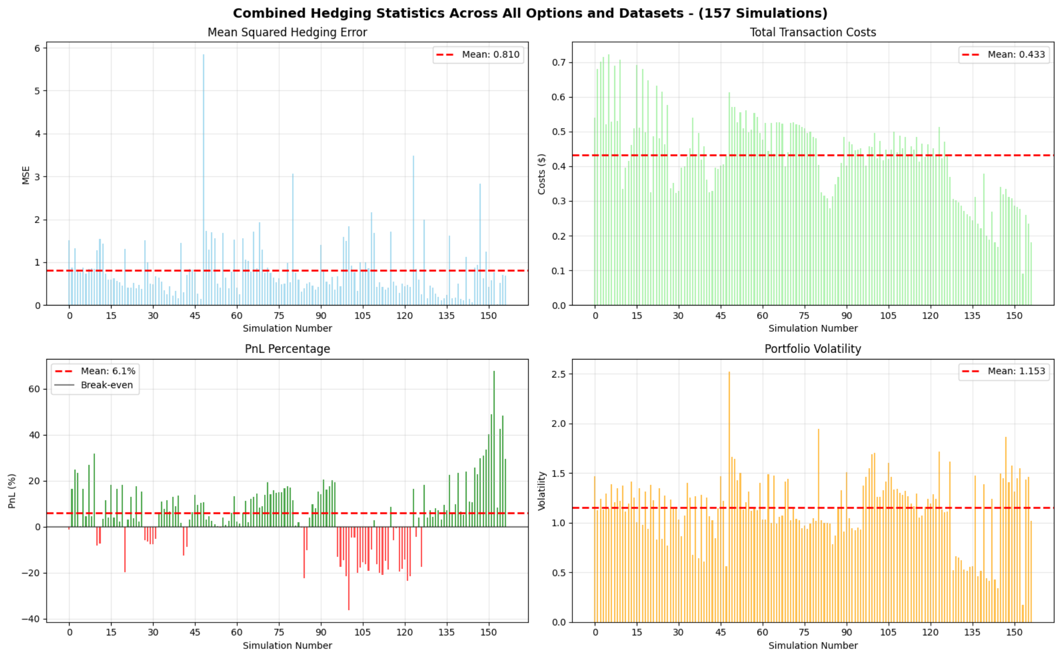


Figure 6: Combined Hedging Statistics Across All Options and Datasets (157 Simulations).

Figure 6 summarizes the key metrics from these simulations. The mean MSE across all simulations was 0.810. The mean PnL was 6.1%. The results show considerable variation in performance across different simulations, which is expected given the different market conditions and option characteristics.

The error in the replicating portfolio is a key measure of hedging effectiveness. Across the 157 delta hedging simulations, the distribution of these errors provides insight into the robustness of the strategy.

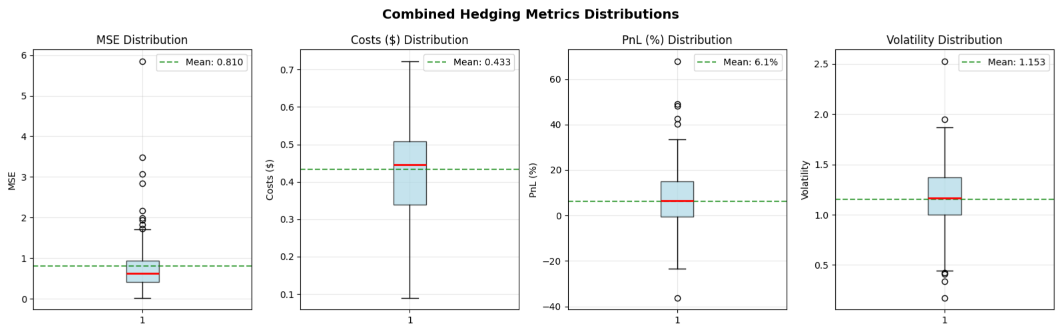


Figure 7: Combined Hedging Metrics Distributions.

Figure 7 shows the distributions of the hedging metrics. These plots were generated from the summary statistics DataFrame using the `plot_hedging_summary_distributions` function.

The distribution of MSE is right-skewed, with a median lower than the mean, indicating that most hedges performed well, but a few had large errors. The PnL distribution is centered slightly below zero, which is expected due to the consistent drag of transaction costs. The portfolio volatility distribution shows the variability in the value of the hedged portfolio, which is another measure of risk.

4.2 Delta-Vega Hedging a Portfolio of Options

A delta-vega hedging strategy was implemented using a portfolio of two options to neutralize both delta and vega risk. The strategy involves holding a position in a primary option, a position in a second option for the vega hedge, and a position in the underlying asset to neutralize the net delta of the two options.

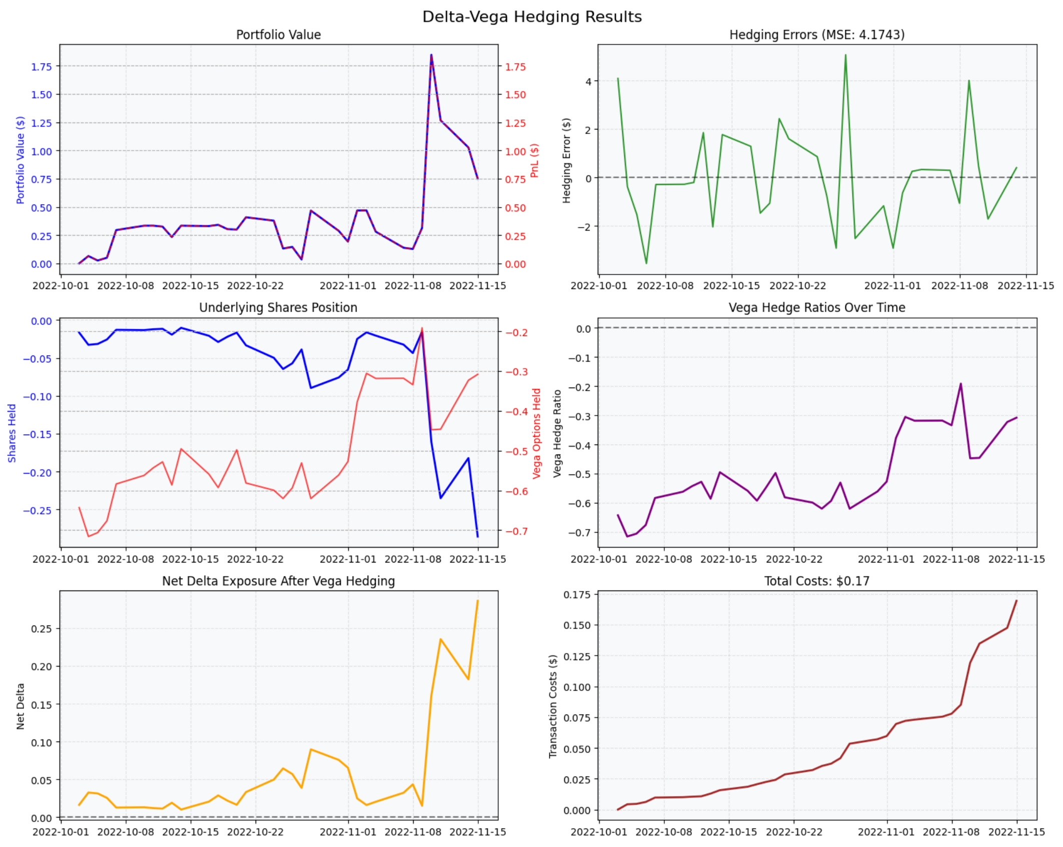


Figure 8: Delta-Vega Hedging Results for a Two-Option Portfolio.

The results of this simulation are shown in Figure 8. The simulation was conducted using the `delta_vega_hedging` function, which requires two options to create a vega-neutral portfolio.

The MSE for this strategy was 4.1743. This is notably higher than the MSE from the single-option delta hedge. This is likely due to the different market conditions and options used in this specific simulation (options with different maturities over a different 45-day period), rather than an indication that delta-vega hedging is inherently less accurate. A more direct comparison would require running both strategies on the same set of options and time periods. The final PnL was 0.75, with total transaction costs of 0.17.

5 References

References

[1] Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81(3), 637-654.

[2] Hull, J. C. (2018). *Options, Futures, and Other Derivatives*. Pearson.

A Python Codes Used

A.1 bs.py

```python
import numpy as np
import pandas as pd
from scipy.stats import norm
from datetime import datetime

def black_scholes_call(S, K, T, r, sigma):
    """Calculate the Black-Scholes price of a European call option."""
    if T <= 0:
        return max(S - K, 0)
    elif sigma <= 0:
        return max(S - K, 0)
    else:
        d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
        d2 = d1 - sigma * np.sqrt(T)
        call_price = S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
        return call_price

def black_scholes_delta(S, K, T, r, sigma):
    """Calculate the Black-Scholes delta of a European call option."""
    if T <= 0:
        return 1.0 if S > K else 0.0
    elif sigma <= 0:
        return 1.0 if S > K else 0.0
    else:
        d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
        delta = norm.cdf(d1)
        return delta

def black_scholes_vega(S, K, T, r, sigma):
    """Calculate the Black-Scholes vega of a European call option."""
    if T <= 0 or sigma <= 0:
        return 0.0
    else:
        d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
        vega = S * norm.pdf(d1) * np.sqrt(T)
        return vega

def black_scholes_gamma(S, K, T, r, sigma):
    """Calculate the Black-Scholes gamma of a European call option."""
    if T <= 0 or sigma <= 0:
        return 0.0
    else:
        d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
        gamma = norm.pdf(d1) / (S * sigma * np.sqrt(T))
        return gamma

def implied_volatility(C_market, S, K, T, r, tol=1e-6, max_iter=100):
    """Calculate the implied volatility using bisection method."""
    if C_market <= 0 or T <= 0:
        return 0.0

    low = 0.001
    high = 2.0

    for _ in range(max_iter):
        mid = (low + high) / 2
        C_model = black_scholes_call(S, K, T, r, mid)
        if abs(C_model - C_market) < tol:
            return mid
        elif C_model > C_market:
            high = mid
        else:
            low = mid

    return (low + high) / 2
```

A.2 data.py

```python
import pandas as pd

def data_load(file_path):
    df = pd.read_feather(file_path)
    df["Date"] = pd.to_datetime(df["Date"])
    df = df.set_index("Date").sort_index()
    df.index.name = None

    for col in df.columns:
        df[col] = pd.to_numeric(df[col], errors="coerce")

    return df
```

A.3 hedging.py

```python
import numpy as np
import pandas as pd
from options_lib.bs import black_scholes_delta, implied_
   volatility, black_scholes_vega, black_scholes_gamma

def simple_delta_hedging(df, start_date, end_date, option_col,
   K, r, maturity, freq=1):
    start_idx = df.index.get_loc(start_date)
    end_idx = df.index.get_loc(end_date) + 1

    df_hedge = df.iloc[start_idx:end_idx]
    OP = df[option_col].values[start_idx:end_idx]
    RE = df['Close'].values[start_idx:end_idx]
    n = len(df_hedge)

    deltas = np.zeros(n)
    A_errors = np.zeros(n - 1)
    iv_values = np.zeros(n)

    for i in range(n):
        T = (maturity - df_hedge.index[i]).days / 365
        iv = implied_volatility(OP[i], RE[i], K, T, r)
        iv_values[i] = iv
        deltas[i] = black_scholes_delta(RE[i], K, T, r, iv)

    for i in range(n-1):
        delta_idx = (i // freq) * freq
        current_delta = deltas[delta_idx]
        dC = OP[i+1] - OP[i]
        dR = RE[i+1] - RE[i]
        A_errors[i] = dC - current_delta * dR

    E = np.mean(A_errors**2)
    print(f"Mean Squared Hedging Error: {E:.4f}")

    return df_hedge, deltas, OP, RE, iv_values, A_errors

def delta_hedging(df, start_date, end_date, option_col, K, r, maturity, freq=1,
                  transaction_cost_per_share=0.0, transaction_cost_percentage=0.0):

    start_idx = df.index.get_loc(start_date)
    end_idx = df.index.get_loc(end_date) + 1

    df_hedge = df.iloc[start_idx:end_idx]
    OP = df[option_col].values[start_idx:end_idx]
    RE = df['Close'].values[start_idx:end_idx]
    n = len(df_hedge)

    deltas = np.zeros(n)
    iv_values = np.zeros(n)
    shares_held = np.zeros(n)
    cash_position = np.zeros(n)
    portfolio_values = np.zeros(n)
    cumulative_costs = np.zeros(n)
    pnl = np.zeros(n)

    for i in range(n):
        T = (maturity - df_hedge.index[i]).days / 365
        iv = implied_volatility(OP[i], RE[i], K, T, r)
        iv_values[i] = iv
        deltas[i] = black_scholes_delta(RE[i], K, T, r, iv)

    shares_held[0] = -deltas[0]
    cash_position[0] = deltas[0] * RE[0] - OP[0]
    portfolio_values[0] = OP[0] + shares_held[0] * RE[0] + cash_position[0]
    pnl[0] = 0.0

    for i in range(1, n):
        if i % freq == 0 or i == n-1:
            target_shares = -deltas[i]
            shares_to_trade = target_shares - shares_held[i-1]

            trade_value = abs(shares_to_trade) * RE[i]
            cost = (abs(shares_to_trade) * transaction_cost_per
_share +
                    trade_value * transaction_cost_percentage)

            cash_position[i] = cash_position[i-1] - cost -
shares_to_trade * RE[i]
            shares_held[i] = target_shares
            cumulative_costs[i] = cumulative_costs[i-1] + cost
        else:
            shares_held[i] = shares_held[i-1]
            cash_position[i] = cash_position[i-1]
            cumulative_costs[i] = cumulative_costs[i-1]

        portfolio_values[i] = OP[i] + shares_held[i] * RE[i] +
cash_position[i]
        pnl[i] = portfolio_values[i] - portfolio_values[0]

    A_errors = np.zeros(n - 1)
    for i in range(n-1):
        delta_idx = (i // freq) * freq
        current_delta = deltas[delta_idx]
        dC = OP[i+1] - OP[i]
        dR = RE[i+1] - RE[i]
        A_errors[i] = dC - current_delta * dR

    E = np.mean(A_errors**2)

    return (df_hedge, deltas, OP, RE, iv_values, A_errors,
            shares_held, cash_position, portfolio_values,
cumulative_costs, pnl)

def run_hedging_intervals(df, maturity, interval_length=45,
step_size=5, num_intervals=10,
                          option_col="C400", K=400, r=0.05, freq
=1,
                          transaction_cost_per_share=0.01,
transaction_cost_percentage=0.0005):

    results = []

    for i in range(num_intervals):
        start_idx = i * step_size
        end_idx = start_idx + interval_length

        if end_idx > len(df):
            break

        interval_data = df.iloc[start_idx:end_idx]
        if interval_data[[option_col, 'Close']].isna().any().
    any():
            continue

        start_date = df.index[start_idx]
        end_date = df.index[end_idx - 1]

        calendar_days = (end_date - start_date).days

        result = delta_hedging(df, start_date, end_date, option
    _col, K, r, maturity, freq,
                                transaction_cost_per_share,
    transaction_cost_percentage)

        stats = {
            'interval': len(results),
            'start_date': start_date,
            'end_date': end_date,
            'data_points': interval_length,
            'calendar_days': calendar_days,
            'mean_squared_error': np.mean(result[5]**2),
            'total_costs': result[9][-1],
            'final_pnl': result[10][-1],
            'portfolio_volatility': np.std(result[8]),
            'max_portfolio_value': np.max(result[8]),
            'min_portfolio_value': np.min(result[8]),
            'pnl_percentage': (result[10][-1] / result[2][0] *
    100) if result[2][0] != 0 else 0
        }
        results.append(stats)

    return pd.DataFrame(results)

def delta_vega_hedging(df1, df2, start_date, end_date, option_
```

```python
primary, option_vega,
                    K_primary, K_vega, r=0.05, maturity1=None, maturity2=None, freq=1,
                    transaction_cost_per_share=0.0, transaction_cost_percentage=0.0):

    df_hedge = df1.loc[start_date:end_date]
    OP_primary = df1.loc[start_date:end_date, option_primary].values
    OP_vega = df2.loc[start_date:end_date, option_vega].values
    RE = df1.loc[start_date:end_date, 'Close'].values
    n = len(df_hedge)

    deltas_primary = np.zeros(n)
    deltas_vega = np.zeros(n)
    vegas_primary = np.zeros(n)
    vegas_vega = np.zeros(n)
    iv_primary = np.zeros(n)
    iv_vega = np.zeros(n)
    alphas = np.zeros(n)
    net_deltas = np.zeros(n)

    shares_held = np.zeros(n)
    vega_option_held = np.zeros(n)
    cash_position = np.zeros(n)
    portfolio_values = np.zeros(n)
    cumulative_costs = np.zeros(n)
    pnl = np.zeros(n)

    for i in range(n):
        T1 = (maturity1 - df_hedge.index[i]).days / 365
        T2 = (maturity2 - df_hedge.index[i]).days / 365

        iv_primary[i] = implied_volatility(OP_primary[i], RE[i], K_primary, T1, r)
        deltas_primary[i] = black_scholes_delta(RE[i], K_primary, T1, r, iv_primary[i])
        vegas_primary[i] = black_scholes_vega(RE[i], K_primary, T1, r, iv_primary[i])

        iv_vega[i] = implied_volatility(OP_vega[i], RE[i], K_vega, T2, r)
        deltas_vega[i] = black_scholes_delta(RE[i], K_vega, T2, r, iv_vega[i])
        vegas_vega[i] = black_scholes_vega(RE[i], K_vega, T2, r
, iv_vega[i])

        if abs(vegas_vega[i]) > 1e-6:
            raw_alpha = -vegas_primary[i] / vegas_vega[i]
            alphas[i] = np.clip(raw_alpha, -5.0, 5.0)
        else:
            alphas[i] = 0.0

        net_deltas[i] = deltas_primary[i] + alphas[i] * deltas_vega[i]

    shares_held[0] = -net_deltas[0]
    vega_option_held[0] = alphas[0]
    cash_position[0] = (net_deltas[0] * RE[0] - alphas[0] * OP_vega[0]) - OP_primary[0]
    portfolio_values[0] = OP_primary[0] + shares_held[0] * RE[0] + vega_option_held[0] * OP_vega[0] + cash_position[0]
    pnl[0] = 0.0

    for i in range(1, n):
        if i % freq == 0 or i == n-1:
            target_shares = -net_deltas[i]
            target_vega_option = alphas[i]
            shares_to_trade = target_shares - shares_held[i-1]
            vega_option_to_trade = target_vega_option - vega_option_held[i-1]

            trade_value_shares = abs(shares_to_trade) * RE[i]
            trade_value_vega = abs(vega_option_to_trade) * OP_vega[i]
            cost = (abs(shares_to_trade) * transaction_cost_per_share +
                    trade_value_shares * transaction_cost_percentage +
                    abs(vega_option_to_trade) * transaction_cost_per_share +
                    trade_value_vega * transaction_cost_percentage)

            cash_position[i] = cash_position[i-1] - cost - shares_to_trade * RE[i] - vega_option_to_trade * OP_vega[i]
            shares_held[i] = target_shares
            vega_option_held[i] = target_vega_option
            cumulative_costs[i] = cumulative_costs[i-1] + cost
        else:
```

```python
            shares_held[i] = shares_held[i-1]
            vega_option_held[i] = vega_option_held[i-1]
            cash_position[i] = cash_position[i-1]
            cumulative_costs[i] = cumulative_costs[i-1]

        portfolio_values[i] = OP_primary[i] + shares_held[i] * RE[i] + vega_option_held[i] * OP_vega[i] + cash_position[i]
        pnl[i] = portfolio_values[i] - portfolio_values[0]

    A_errors = np.zeros(n - 1)
    for i in range(n-1):
        idx = (i // freq) * freq
        current_net_delta = net_deltas[idx]
        current_alpha = alphas[idx]
        dC_primary = OP_primary[i+1] - OP_primary[i]
        dC_vega = OP_vega[i+1] - OP_vega[i]
        dR = RE[i+1] - RE[i]
        A_errors[i] = dC_primary - current_net_delta * dR - current_alpha * dC_vega

    E = np.mean(A_errors**2)

    return (df_hedge, net_deltas, alphas, OP_primary, OP_vega, RE, iv_primary, iv_vega, A_errors,
            shares_held, vega_option_held, cash_position, portfolio_values, cumulative_costs, pnl)
```

A.4 plots.py

```python
import matplotlib.pyplot as plt
import mplfinance as mpf
import pandas as pd
import matplotlib.dates as mdates
import numpy as np

def plot_spy_and_options(df, option_cols):

    spy_df = df[["Open", "High", "Low", "Close"]]
    mc = mpf.make_marketcolors(up="#26a69a", down="#ef5350", edge="i", wick="i", volume="in")
    s = mpf.make_mpf_style(marketcolors=mc, gridstyle="--", facecolor="#f8f9fa")

    mpf.plot(
        spy_df,
        type="candle",
        style=s,
        title="SPY Candlesticks",
        ylabel="SPY Price ($",
        figsize=(14, 7),
        tight_layout=True,
        figratio=(16, 9),
    )

    for option_col in option_cols:
        if option_col in df.columns:
            opt_df = df[[option_col]]
            fig, ax = plt.subplots(figsize=(14, 7))
            ax.plot(opt_df.index, opt_df[option_col], color="#1f77b4", lw=2, label=f"{option_col} Price")
            ax.set_title(f"SPY {option_col} Option Price", fontsize=14, weight="bold")
            ax.set_xlabel("Time", fontsize=12)
            ax.set_ylabel("Option Price ($", fontsize=12)
            ax.grid(True, alpha=0.25)
            ax.spines["top"].set_visible(False)
            ax.spines["right"].set_visible(False)
            ax.legend(loc="upper left")
            plt.tight_layout()
            plt.show()
        else:
            print(f"Option column '{option_col}' not found in the data.")

    print(f"Data starts: {df.index.min()}")
    print(f"Data ends: {df.index.max()}")
    print(f"Number of days: {(df.index.max() - df.index.min()).days}")
    print(f"Number of data points: {len(df)}")

def plot_hedging_errors(df_hedge, A_errors):
    plt.figure(figsize=(8, 4))
    plt.plot(df_hedge.index[:-1], A_errors, 'b-', linewidth=2)
    plt.title('Daily Hedging Errors')
    plt.xlabel('Date')
    plt.ylabel('Error')
    plt.grid(True)
    plt.show()

def plot_positions(df_hedge, OP, RE, deltas):
    fig, ax1 = plt.subplots(figsize=(8, 4))

    ax1.plot(df_hedge.index, OP, 'b-', linewidth=2, label='Long Call Position')
    ax1.set_ylabel('Option Position Value ($)', color='b')
    ax1.tick_params(axis='y', labelcolor='b')
    ax1.grid(True, alpha=0.3)

    ax2 = ax1.twinx()
    ax2.plot(df_hedge.index, -deltas * RE, 'r--', linewidth=2, label='Short Underlying Position')
    ax2.set_ylabel('Underlying Position Value ($)', color='r')
    ax2.tick_params(axis='y', labelcolor='r')

    plt.title('Portfolio Positions (Dual Scale)')
    plt.xlabel('Date')

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    ax1.legend(lines1 + lines2, labels1 + labels2, loc='upper left')

    plt.tight_layout()
    plt.show()

def plot_delta_positions(df_hedge, deltas):
    fig, ax = plt.subplots(figsize=(8, 4))

    ax.plot(df_hedge.index, deltas, 'b-', linewidth=2, label='Call Delta')

    ax.plot(df_hedge.index, -deltas, 'r--', linewidth=2, label='Hedge Delta (Short)')

    ax.axhline(y=0, color='black', linestyle='--', alpha=0.5, label='Neutral')

    ax.set_ylabel('Delta Contribution')
    ax.set_xlabel('Date')
    ax.set_title('Delta Positions - Symmetric Neutrality')
    ax.legend()
    ax.grid(True, alpha=0.3)

    plt.show()

```

```
def plot_portfolio_and_pnl(dates, portfolio_values, pnl, title="Portfolio Value and PnL"):
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

    ax1.plot(dates, portfolio_values, 'g-', linewidth=2, label='Portfolio Value')
    ax1.axhline(y=portfolio_values[0], color='black', linestyle='--', alpha=0.7)
    ax1.set_title('Portfolio Value')
    ax1.set_ylabel('Value ($)')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    ax2.plot(dates, pnl, 'b-', linewidth=2, label='Cumulative PnL')
    ax2.axhline(y=0, color='red', linestyle='--', alpha=0.7)
    ax2.fill_between(dates, pnl, 0, where=(pnl >= 0), color='green', alpha=0.3)
    ax2.fill_between(dates, pnl, 0, where=(pnl < 0), color='red', alpha=0.3)
    ax2.set_title('Cumulative PnL')
    ax2.set_xlabel('Date')
    ax2.set_ylabel('PnL ($)')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

def plot_hedging_simulation_stats(stats_df, title="Hedging Simulation Statistics"):
    if len(stats_df) == 0:
        print("No data to plot")
        return

    x_values = range(len(stats_df))

    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16, 10))
    fig.suptitle(title, fontsize=14, fontweight='bold')

    mse_data = stats_df['mean_squared_error']
    bar_width = max(0.5, 8.0 / len(x_values))
    bars1 = ax1.bar(x_values, mse_data, color='skyblue', alpha=0.7, width=bar_width)
    mse_mean = mse_data.mean()
    ax1.axhline(y=mse_mean, color='red', linestyle='--', linewidth=2, label=f'Mean: {mse_mean:.3f}')
    ax1.set_title('Mean Squared Hedging Error')
    ax1.set_xlabel('Simulation Number')
    ax1.set_ylabel('MSE')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    if len(x_values) > 20:
        tick_spacing = max(1, len(x_values) // 10)
        ax1.set_xticks(x_values[::tick_spacing])
        ax1.set_xticklabels(x_values[::tick_spacing])

    cost_data = stats_df['total_costs']
    bars2 = ax2.bar(x_values, cost_data, color='lightgreen', alpha=0.7, width=bar_width)
    cost_mean = cost_data.mean()
    ax2.axhline(y=cost_mean, color='red', linestyle='--', linewidth=2, label=f'Mean: {cost_mean:.3f}')
    ax2.set_title('Total Transaction Costs')
    ax2.set_xlabel('Simulation Number')
    ax2.set_ylabel('Costs ($)')
    ax2.legend()
    ax2.grid(True, alpha=0.3)
    if len(x_values) > 20:
        ax2.set_xticks(x_values[::tick_spacing])
        ax2.set_xticklabels(x_values[::tick_spacing])

    pnl_data = stats_df['pnl_percentage']
    colors = ['red' if x < 0 else 'green' for x in pnl_data]
    bars3 = ax3.bar(x_values, pnl_data, color=colors, alpha=0.7, width=bar_width)
    pnl_mean = pnl_data.mean()
    ax3.axhline(y=pnl_mean, color='red', linestyle='--', linewidth=2, label=f'Mean: {pnl_mean:.1f}%')
    ax3.axhline(y=0, color='black', linestyle='-', alpha=0.5, label='Break-even')
    ax3.set_title('PnL Percentage')
    ax3.set_xlabel('Simulation Number')
    ax3.set_ylabel('PnL (%)')
    ax3.legend()
    ax3.grid(True, alpha=0.3)
    if len(x_values) > 20:
        ax3.set_xticks(x_values[::tick_spacing])
        ax3.set_xticklabels(x_values[::tick_spacing])

    vol_data = stats_df['portfolio_volatility']
    bars4 = ax4.bar(x_values, vol_data, color='orange', alpha=0.7, width=bar_width)
    vol_mean = vol_data.mean()
    ax4.axhline(y=vol_mean, color='red', linestyle='--', linewidth=2, label=f'Mean: {vol_mean:.3f}')
    ax4.set_title('Portfolio Volatility')
    ax4.set_xlabel('Simulation Number')
    ax4.set_ylabel('Volatility')
    ax4.legend()
    ax4.grid(True, alpha=0.3)
    if len(x_values) > 20:
        ax4.set_xticks(x_values[::tick_spacing])
        ax4.set_xticklabels(x_values[::tick_spacing])

    plt.tight_layout()
    plt.show()

def plot_hedging_summary_distributions(stats_df, title="Hedging Metrics Distributions"):
    if len(stats_df) == 0:
        print("No data to plot")
        return

    metrics = ['mean_squared_error', 'total_costs', 'pnl_percentage', 'portfolio_volatility']
    labels = ['MSE', 'Costs ($)', 'PnL (%)', 'Volatility']

    fig, axes = plt.subplots(1, 4, figsize=(16, 5))
    fig.suptitle(title, fontsize=14, fontweight='bold')

    for i, (metric, label) in enumerate(zip(metrics, labels)):
        data = stats_df[metric]
        axes[i].boxplot(data, patch_artist=True,
                        boxprops=dict(facecolor='lightblue', alpha=0.7),
                        medianprops=dict(color='red', linewidth=2))
        axes[i].set_title(f'{label} Distribution')
        axes[i].set_ylabel(label)
        axes[i].grid(True, alpha=0.3)

        mean_val = data.mean()
```

```python
        if '%' in label:
            axes[i].axhline(y=mean_val, color='green', linestyle='--', alpha=0.7, label=f'Mean: {mean_val:.1f}%')
        else:
            axes[i].axhline(y=mean_val, color='green', linestyle='--', alpha=0.7, label=f'Mean: {mean_val:.3f}')
        axes[i].legend()

    plt.tight_layout()
    plt.show()

def plot_delta_vega_hedging(df_hedge, net_deltas, alphas, OP_primary, OP_vega, RE,
                           A_errors, shares_held, vega_option_held, portfolio_values,
                           cumulative_costs, pnl, title="Delta-Vega Hedging Results"):

    fig, axes = plt.subplots(3, 2, figsize=(15, 12))
    fig.suptitle(title, fontsize=16)

    ax1 = axes[0, 0]
    ax1.plot(df_hedge.index, portfolio_values, 'b-', linewidth=2, label='Portfolio Value')
    ax1.set_ylabel('Portfolio Value ($)', color='b')
    ax1.tick_params(axis='y', labelcolor='b')
    ax1.grid(True, alpha=0.3)
    ax1.set_title('Portfolio Value')

    ax1_twin = ax1.twinx()
    ax1_twin.plot(df_hedge.index, pnl, 'r--', alpha=0.7, label='PnL')
    ax1_twin.set_ylabel('PnL ($)', color='r')
    ax1_twin.tick_params(axis='y', labelcolor='r')

    ax2 = axes[0, 1]
    ax2.plot(df_hedge.index[:-1], A_errors, 'g-', alpha=0.8, label='Hedging Errors')
    ax2.axhline(y=0, color='black', linestyle='--', alpha=0.5)
    ax2.set_ylabel('Hedging Error ($)')
    ax2.grid(True, alpha=0.3)
    ax2.set_title(f'Hedging Errors (MSE: {np.mean(A_errors**2):.4f})')

    ax3 = axes[1, 0]
    ax3.plot(df_hedge.index, shares_held, 'b-', linewidth=2, label='Underlying Shares')
    ax3.set_ylabel('Shares Held', color='b')
    ax3.tick_params(axis='y', labelcolor='b')
    ax3.grid(True, alpha=0.3)
    ax3.set_title('Underlying Shares Position')

    ax3_twin = ax3.twinx()
    ax3_twin.plot(df_hedge.index, vega_option_held, 'r-', alpha=0.7, label='Vega Options')
    ax3_twin.set_ylabel('Vega Options Held', color='r')
    ax3_twin.tick_params(axis='y', labelcolor='r')

    ax4 = axes[1, 1]
    ax4.plot(df_hedge.index, alphas, 'purple', linewidth=2, label='Vega Hedge Ratio')
    ax4.axhline(y=0, color='black', linestyle='--', alpha=0.5)
    ax4.set_ylabel('Vega Hedge Ratio')
    ax4.grid(True, alpha=0.3)
    ax4.set_title('Vega Hedge Ratios Over Time')

    ax5 = axes[2, 0]
    ax5.plot(df_hedge.index, net_deltas, 'orange', linewidth=2, label='Net Delta')
    ax5.axhline(y=0, color='black', linestyle='--', alpha=0.5)
    ax5.set_ylabel('Net Delta')
    ax5.grid(True, alpha=0.3)
    ax5.set_title('Net Delta Exposure After Vega Hedging')

    ax6 = axes[2, 1]
    ax6.plot(df_hedge.index, cumulative_costs, 'brown', linewidth=2, label='Cumulative Costs')
    ax6.set_ylabel('Transaction Costs ($)')
    ax6.grid(True, alpha=0.3)
    ax6.set_title(f'Total Costs: ${cumulative_costs[-1]:.2f}')

    plt.tight_layout()
    plt.show()
```