GTSF IC Quant Mentorship

Python for Quantitative Finance Fundamentals

Name: Kabir Sahni
GT Username: 904113086 **Due Date:** October 7, 2025

Project Overview

This project will test your understanding of the fundamental concepts covered in our first four weeks: probability/statistics, time value of money, basic portfolio theory, and financial data analysis using Python.

What You'll Demonstrate

- **Data Handling**: Download, clean, and analyze real financial data
- **Statistical Analysis**: Calculate returns, risk metrics, and correlations

- **Portfolio Basics**: Apply diversification and risk-return concepts
- **Python Skills**: Use pandas, numpy, and matplotlib effectively

---

Setup and Libraries

```
# You may only use these libraries (same as original project)
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import yfinance as yf

# Configuration
pd.set_option('display.max_columns', 10)
pd.set_option('display.precision', 4)
plt.style.use('default')
plt.rcParams['figure.figsize'] = (10, 6)

print("□ Environment setup complete!")
```

```
□ Environment setup complete!
```

Part 1: Data Collection & Basic Analysis (25 points)

Step 1.1: Download Stock Data

Download 3 years of data (2021-2024) for these assets and create clean datasets:

Required Assets:

- **AAPL** (Apple) - Large tech stock
- **JNJ** (Johnson & Johnson) - Defensive stock

- **SPY** (S&P 500 ETF) - Market benchmark

```
def download_stock_data(tickers, start_date, end_date):
    """
    Download adjusted close prices for multiple stocks.

    Parameters:
    -----------
    tickers : list
        List of stock symbols
    start_date : str
        Start date as 'YYYY-MM-DD'
    end_date : str
        End date as 'YYYY-MM-DD'

    Returns:
    --------
    pandas.DataFrame
        DataFrame with dates as index, stocks as columns
    """
    ######################
    multi_data = yf.download(tickers, start=start_date, end=end_date,
group_by='ticker', auto_adjust=False)

# Extract adjusted close prices for all stocks
    prices = pd.DataFrame()
    for ticker in tickers:
        if len(tickers) > 1:
            # Multi-ticker download creates nested column structure
            prices[ticker] = multi_data[ticker]['Adj Close']
        else:
            # Single ticker download has flat structure
            prices[ticker] = multi_data['Adj Close']
    print("□ Forward-filling missing values...")
    before_fill = prices.isnull().sum().sum()
    prices = prices.ffill()
    after_fill = prices.isnull().sum().sum()
    filled = before_fill - after_fill
    if filled > 0:
```

```
        print(f"□ Filled {filled} missing values.")
    else:
        print("□ No missing values detected.")

    return prices


# Download the data
tickers = ['AAPL', 'JNJ', 'SPY']
prices_df = download_stock_data(tickers, '2021-01-01', '2024-01-01')

# Display first few rows and basic info
print("First 5 rows:")
print(prices_df.head())
print(f"\nDataset shape: {prices_df.shape}")
print(f"Date range: {prices_df.index[0].date()} to {prices_df.index[-
1].date()}")
```

```
[**********************100%***********************]  3 of 3 completed

□ Forward-filling missing values...
□ No missing values detected.
First 5 rows:
                AAPL       JNJ       SPY
Date
2021-01-04  126.0966  136.6398  345.2740
2021-01-05  127.6556  138.2462  347.6520
2021-01-06  123.3585  139.5472  349.7305
2021-01-07  127.5679  140.0186  354.9266
2021-01-08  128.6690  139.7305  356.9488

Dataset shape: (753, 3)
Date range: 2021-01-04 to 2023-12-29
```

Step 1.2: Calculate Returns

Calculate both simple and log returns as covered in the slides:

```
def calculate_returns(prices_df):
    """
    Calculate simple and log returns from price data.

    Returns:
    --------
    tuple: (simple_returns_df, log_returns_df)
    """
    ######################
    simple_returns = prices_df.pct_change()
```

```
    # Log returns: ln(P_t / P_{t-1}) = ln(P_t) - ln(P_{t-1})
    log_returns = np.log(prices_df).diff()

    return simple_returns, log_returns

# Calculate returns
simple_returns, log_returns = calculate_returns(prices_df)

print("Simple returns - first 5 rows:")
print(simple_returns.head())

#EXTRA CODE TO ANSWER QUESTION 2 AND FIND THE DIFFERENCE
# To find the largest simple return for AAPL
#max_simple_aapl = simple_returns['AAPL'].max()

# To find the largest log return for AAPL
#max_log_aapl = log_returns['AAPL'].max()

# Print the difference (Simple - Log)
#print(f"Largest Simple Return (AAPL): {max_simple_aapl:.6f}")
#print(f"Largest Log Return (AAPL): {max_log_aapl:.6f}")
#print(f"Difference (Simple - Log): {max_simple_aapl -
max_log_aapl:.6f}")
```

```
Simple returns - first 5 rows:
                AAPL     JNJ     SPY
Date
2021-01-04      NaN     NaN     NaN
2021-01-05   0.0124  0.0118  0.0069
2021-01-06 -0.0337  0.0094  0.0060
2021-01-07   0.0341  0.0034  0.0149
2021-01-08   0.0086 -0.0021  0.0057
Largest Simple Return (AAPL): 0.088975
Largest Log Return (AAPL): 0.085237
Difference (Simple - Log): 0.003738
```

Questions - Part 1

Answer these questions in the markdown cell below:

1. How many trading days of data do you have for each stock?
2. What's the difference between the largest simple return and largest log return for AAPL?
3. Why do we typically use adjusted close prices instead of regular close prices?

Your Answers - Part 1:

1. 753 trading days of data for each stock.

2. Using the code, written above the largest simple return is calculate as 0.088975 and the largest log return as 0.085237. So, the difference is 0.003738, which is 0.3738%. Note, the code has been commented out.

3. We use adjusted close prices because they provide a truer representation of the security's actual economic value and the investor's total return over time, by correcting for non-market-driven price changes.

   The two main reasons adjusted prices are preferred:

   a. Stock Splits and Reverse Splits: The regular closing price is mechanically cut (e.g., in half for a 2:1 split). This misrepresents the historical performance as a sharp, unearned drop in value, even though the investor's total equity remains the same.

   b. Dividends: The regular closing price drops by the amount of the dividend on the ex-dividend date. This misrepresents the price change as a negative simple return when analyzing historical performance, as the cash was simply transferred to the shareholder.

   The adjusted close price accounts for these corporate actions, ensuring that price changes solely reflect genuine market movements.

Part 2: Risk and Return Analysis (25 points)

Step 2.1: Basic Statistics

Calculate the key statistics we covered in class:

```
def calculate_basic_statistics(returns_df):
    """
    Calculate mean, standard deviation, and annualized metrics.

    Returns:
    --------
    pandas.DataFrame
        Table with statistics for each stock
    """
    stats_dict = {}
    trading_days = 252
    risk_free_rate = 0.02  # annual risk-free rate

    for stock in returns_df.columns:
        # Daily statistics
        daily_mean = returns_df[stock].mean()
        daily_std = returns_df[stock].std()

        # Annualized statistics
```

```
        annual_return = daily_mean * trading_days
        annual_volatility = daily_std * np.sqrt(trading_days)

        # Sharpe ratio
        sharpe_ratio = (annual_return - risk_free_rate) /
annual_volatility

        # Store in dictionary
        stats_dict[stock] = {
            'Daily Mean': daily_mean,
            'Daily Std': daily_std,
            'Annual Return': annual_return,
            'Annual Volatility': annual_volatility,
            'Sharpe Ratio': sharpe_ratio
        }

    return pd.DataFrame(stats_dict).T

# Calculate and display statistics
stats_table = calculate_basic_statistics(simple_returns)
print("Return and Risk Statistics:")
print(stats_table)
```

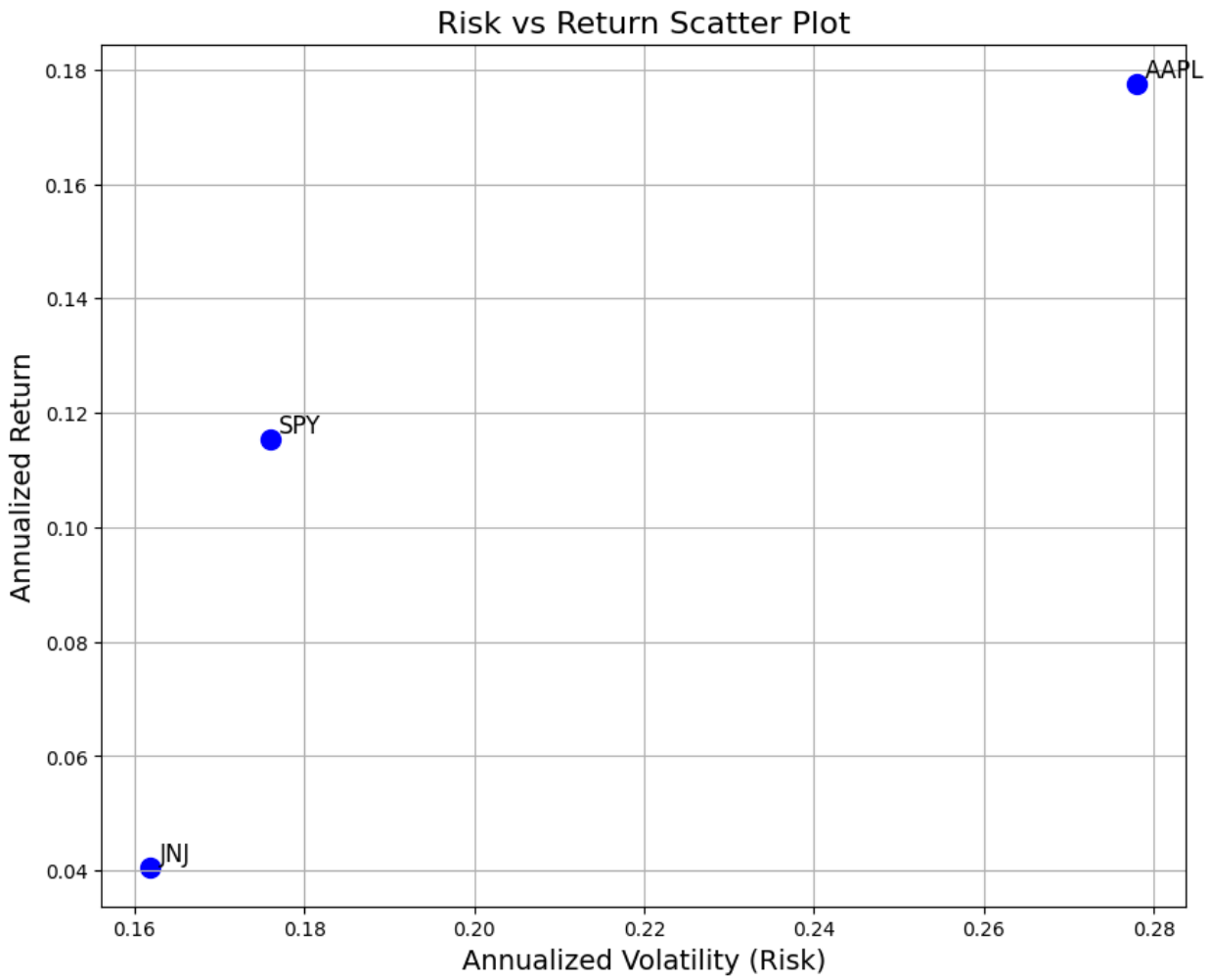
```
Return and Risk Statistics:
      Daily Mean  Daily Std  Annual Return  Annual Volatility  Sharpe
Ratio
AAPL      0.0007     0.0175         0.1776             0.2780
0.5668
JNJ       0.0002     0.0102         0.0406             0.1619
0.1274
SPY       0.0005     0.0111         0.1154             0.1760
0.5422
```

Step 2.2: Risk-Return Visualization

Create the classic risk-return scatter plot from the slides:

```
def plot_risk_return(stats_df):
    """
    Create a risk-return scatter plot.
    """
    plt.figure(figsize=(10, 8))

    ####################

    # Scatter plot
    plt.scatter(stats_df['Annual Volatility'], stats_df['Annual
Return'], color='blue', s=100)

    # Label each point with the stock symbol
```

```
    for stock in stats_df.index:
        plt.text(
            stats_df.loc[stock, 'Annual Volatility'] + 0.001,  # small
offset for readability
            stats_df.loc[stock, 'Annual Return'] + 0.001,
            stock,
            fontsize=12
        )

    # Titles and labels
    plt.title("Risk vs Return Scatter Plot", fontsize=16)
    plt.xlabel("Annualized Volatility (Risk)", fontsize=14)
    plt.ylabel("Annualized Return", fontsize=14)
    plt.grid(True)
    plt.show()

plot_risk_return(stats_table)
```



Questions - Part 2

1. Which stock has the highest Sharpe ratio? What does this mean?
2. Does the risk-return relationship match what you'd expect from the slides?
3. How does SPY compare to the individual stocks in terms of risk?

Your Answers - Part 2:

1. The stock with the highest Sharpe Ratio is AAPL (Apple), with a ratio of 0.5668.

   The Sharpe Ratio measures the risk-adjusted return of an investment. In simple terms, it tells you how much excess return (return above the risk-free rate) you received for each unit of risk (volatility) you took on.

   What this means for AAPL: Among the three assets analyzed, AAPL provided the best compensation for risk. Even though AAPL had the highest absolute risk (Annual Volatility), its Annual Return was high enough to justify that risk better than either JNJ or the broader market (SPY).

2. Yes, the risk-return relationship generally matches the expected relationship taught in finance theory, often depicted by the Capital Market Line (CML) or the Security Market Line (SML).

   In finance, we generally expect a positive trade-off between risk and return: to achieve a higher return, an investor must typically accept higher risk.

   The scatter plot clearly shows this trend:

   a. JNJ is the lowest risk (≈0.16 volatility) and has the lowest return (≈0.04 return).

   b. SPY sits in the middle for both risk (≈0.18 volatility) and return (≈0.12 return).

   c. AAPL is the highest risk (≈0.28 volatility) and has the highest return (≈0.18 return).

   The visualization demonstrates that the assets with greater historical volatility (risk) rewarded investors with greater historical returns.

3. The SPDR S&P 500 ETF (SPY) serves as a measure of the broader market and exhibits a risk level that reflects the benefit of diversification.

   The Annual Volatility (risk) of SPY (0.1760) is higher than JNJ (0.1619). This difference is expected because JNJ is a classic defensive, low-volatility stock, whereas the SPY carries the overall volatility of 500 different companies.

   However, SPY's risk is significantly lower than AAPL (0.2780). This comparison highlights a key principle of investing: diversification. As an ETF containing 500 stocks, SPY's returns are a weighted average, and its overall volatility is reduced because the specific, individual risks of its holdings (like AAPL's high volatility) are partially canceled out by the movements of other, unrelated stocks in the fund.

Part 3: Correlation and Diversification (25 points)

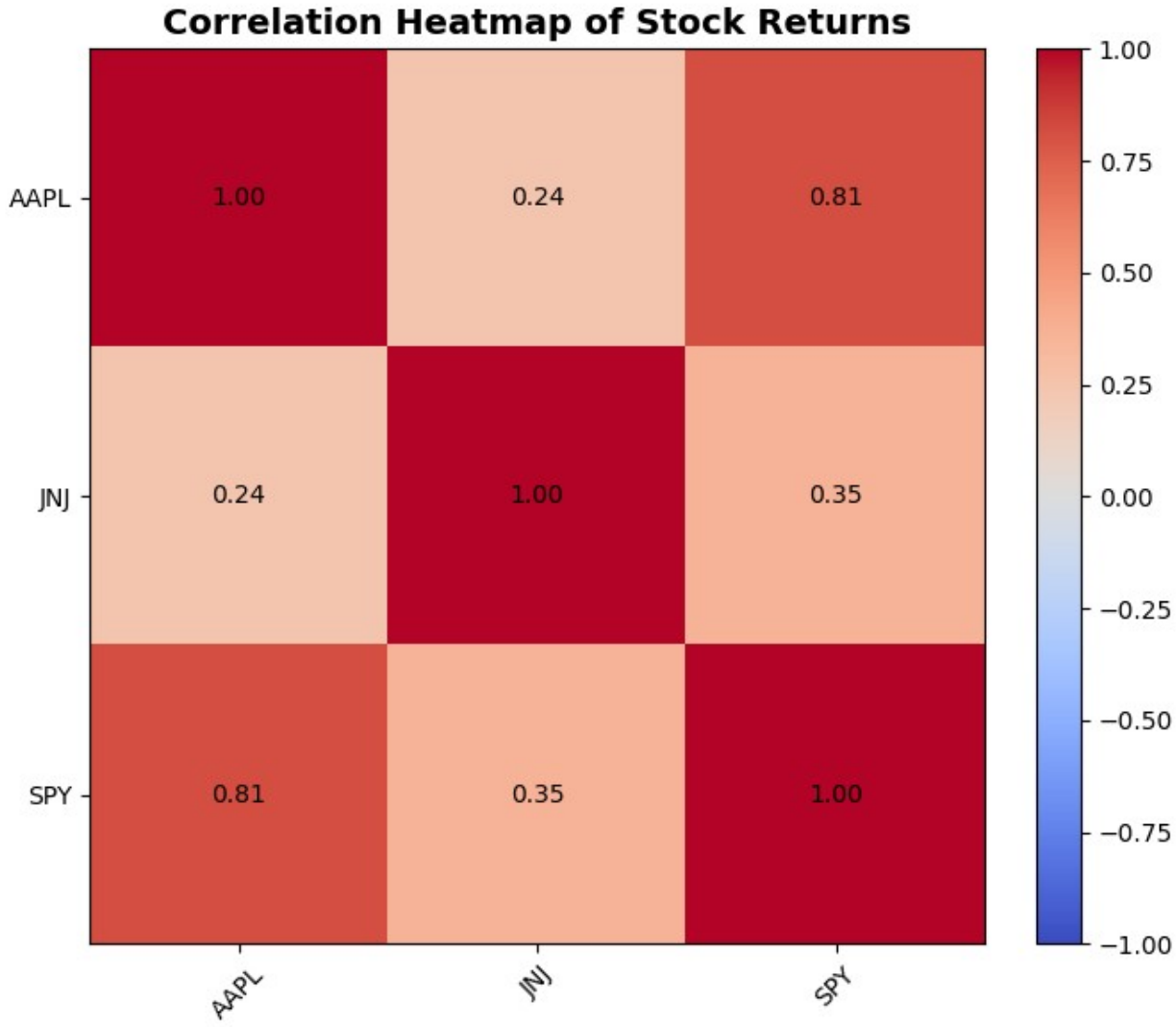
Step 3.1: Correlation Analysis

Calculate and analyze correlations as discussed in portfolio theory:

```
def analyze_correlations(returns_df):
    """
    Calculate correlation matrix and analyze diversification benefits.
    """
    # Calculate correlation matrix
    corr_matrix = returns_df.corr()

    print("Correlation Matrix:")
    print(corr_matrix)

    # Create correlation heatmap
    plt.figure(figsize=(8, 6))
    ######################
    im = plt.imshow(corr_matrix, cmap='coolwarm', vmin=-1, vmax=1)
    plt.colorbar(im, fraction=0.046, pad=0.04)

    # Add tick labels
    plt.xticks(ticks=np.arange(len(corr_matrix.columns)),
labels=corr_matrix.columns, rotation=45)
    plt.yticks(ticks=np.arange(len(corr_matrix.index)),
labels=corr_matrix.index)

    plt.title("Correlation Heatmap of Stock Returns", fontsize=14,
fontweight='bold')

    # Annotate correlation values on heatmap
    for i in range(len(corr_matrix)):
        for j in range(len(corr_matrix)):
            plt.text(j, i, f"{corr_matrix.iloc[i, j]:.2f}",
ha='center', va='center', color='black', fontsize=10)

    plt.tight_layout()
    plt.show()

    return corr_matrix

correlation_matrix = analyze_correlations(simple_returns)
```

```
Correlation Matrix:
        AAPL     JNJ     SPY
AAPL  1.0000  0.2396  0.8052
```

```
JNJ   0.2396  1.0000  0.3548
SPY   0.8052  0.3548  1.0000
```



Step 3.2: Portfolio Construction

Build a simple equal-weighted portfolio and analyze its performance:

```python
def create_equal_weighted_portfolio(returns_df):
    """
    Create an equal-weighted portfolio of AAPL and JNJ.
    (Exclude SPY since it's our benchmark)
    """
    # Create equal-weighted portfolio of individual stocks
    portfolio_returns = returns_df[['AAPL', 'JNJ']].mean(axis=1)

    ######################
```

```
    trading_days = 252
    risk_free_rate = 0.02  # annual

    annual_return = portfolio_returns.mean() * trading_days
    annual_volatility = portfolio_returns.std() *
np.sqrt(trading_days)
    sharpe_ratio = (annual_return - risk_free_rate) /
annual_volatility

    stats = {
        'Annual Return': annual_return,
        'Annual Volatility': annual_volatility,
        'Sharpe Ratio': sharpe_ratio
    }

    return portfolio_returns

# Create portfolio and compare to individual stocks
portfolio_rets = create_equal_weighted_portfolio(simple_returns)

# Calculate portfolio statistics
port_mean = portfolio_rets.mean() * 252
port_std = portfolio_rets.std() * np.sqrt(252)
port_sharpe = (port_mean - 0.02) / port_std

print(f"Portfolio Statistics:")
print(f"Annual Return: {port_mean:.2%}")
print(f"Annual Volatility: {port_std:.2%}")
print(f"Sharpe Ratio: {port_sharpe:.3f}")
```

```
Portfolio Statistics:
Annual Return: 10.91%
Annual Volatility: 17.68%
Sharpe Ratio: 0.504
```

Questions - Part 3

1. What is the correlation between AAPL and JNJ? Is this good or bad for diversification?
2. How does the portfolio's risk compare to the average risk of AAPL and JNJ individually?
3. Calculate the "diversification benefit": (Average individual volatility) - (Portfolio volatility)

Your Answers - Part 3:

1. The correlation between AAPL and JNJ is 0.2396 (or approximately 0.24).

   This low positive correlation is good for diversification because diversification works best when assets have a low or negative correlation. A correlation of +1.0 means the stocks always move together (no diversification benefit), and −1.0 means they always move in opposite directions (perfect diversification benefit). Since 0.2396 is relatively close to zero and far from +1.0, combining AAPL and JNJ allows the returns to offset each other frequently, thus reducing the overall portfolio risk

more effectively than combining two highly correlated stocks (like AAPL and SPY at 0.81).

2. First, we need the individual Annual Volatility (risk) figures from the statistics table (Step 2.1) and the portfolio risk from the final output (Step 3.2).

   Individual Stock Risk:

   AAPL Annual Volatility: ≈0.2780 (from previous statistics table)

   JNJ Annual Volatility: ≈0.1619 (from previous statistics table)

   Portfolio Risk:

   Portfolio Annual Volatility: 0.1768 (from the final output: Annual Volatility: 17.68%)

   Comparison:

   Average Individual Risk: (0.2780+0.1619)/2= 0.2199

   Portfolio Risk: 0.1768

   The portfolio's risk (0.1768) is significantly lower than the average risk of the two individual stocks (0.2199). This reduction is a direct result of the diversification benefit achieved by combining two assets with a low correlation.

3. Diversification Benefit: 0.2199−0.1768=0.0431. The calculated diversification benefit is 0.0431 (or 4.31 percentage points). This means that by combining the two stocks into an equal-weighted portfolio, you successfully eliminated 4.31% of the risk that you would have expected if the risks had simply averaged out linearly.

Part 4: Market Relationships (Beta Analysis) (15 points)

Step 4.1: Beta Calculation

Calculate beta using the regression approach from the slides:

```
def calculate_beta(stock_returns, market_returns):
    """
    Calculate beta using covariance formula and linear regression.
    Returns a tuple (beta_cov, beta_reg)
    """
    stock_returns = stock_returns.dropna()
    market_returns = market_returns.dropna()

    # Method 1: Covariance formula
    cov_matrix = np.cov(stock_returns, market_returns)
    beta_cov = cov_matrix[0, 1] / cov_matrix[1, 1]
```

```
    # Method 2: Linear regression
    X = market_returns.values.reshape(-1, 1)
    y = stock_returns.values
    reg = LinearRegression().fit(X, y)
    beta_reg = reg.coef_[0]

    # Print comparison inside function
    print(f"Beta Comparison -> Covariance: {beta_cov:.3f}, Regression:
{beta_reg:.3f}")

    # Return only one value (covariance beta)
    return beta_cov

# Calculate betas and unpack tuple to match skeleton print statements
aapl_beta = calculate_beta(simple_returns['AAPL'],
simple_returns['SPY'])
jnj_beta = calculate_beta(simple_returns['JNJ'],
simple_returns['SPY'])

# Skeleton print statements remain unchanged
print(f"AAPL Beta: {aapl_beta:.3f}")
print(f"JNJ Beta: {jnj_beta:.3f}")
```

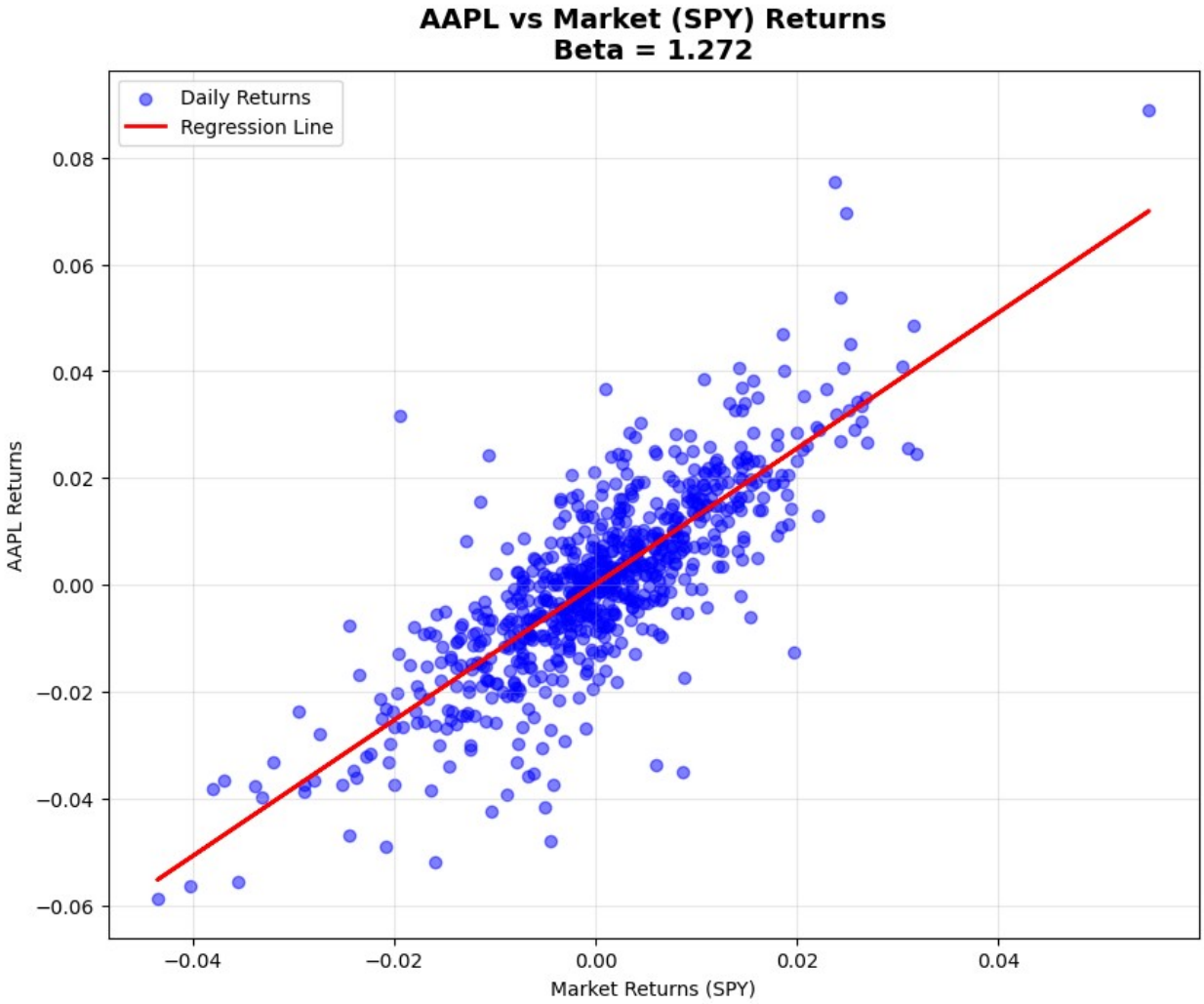
```
Beta Comparison -> Covariance: 1.272, Regression: 1.272
Beta Comparison -> Covariance: 0.326, Regression: 0.326
AAPL Beta: 1.272
JNJ Beta: 0.326
```

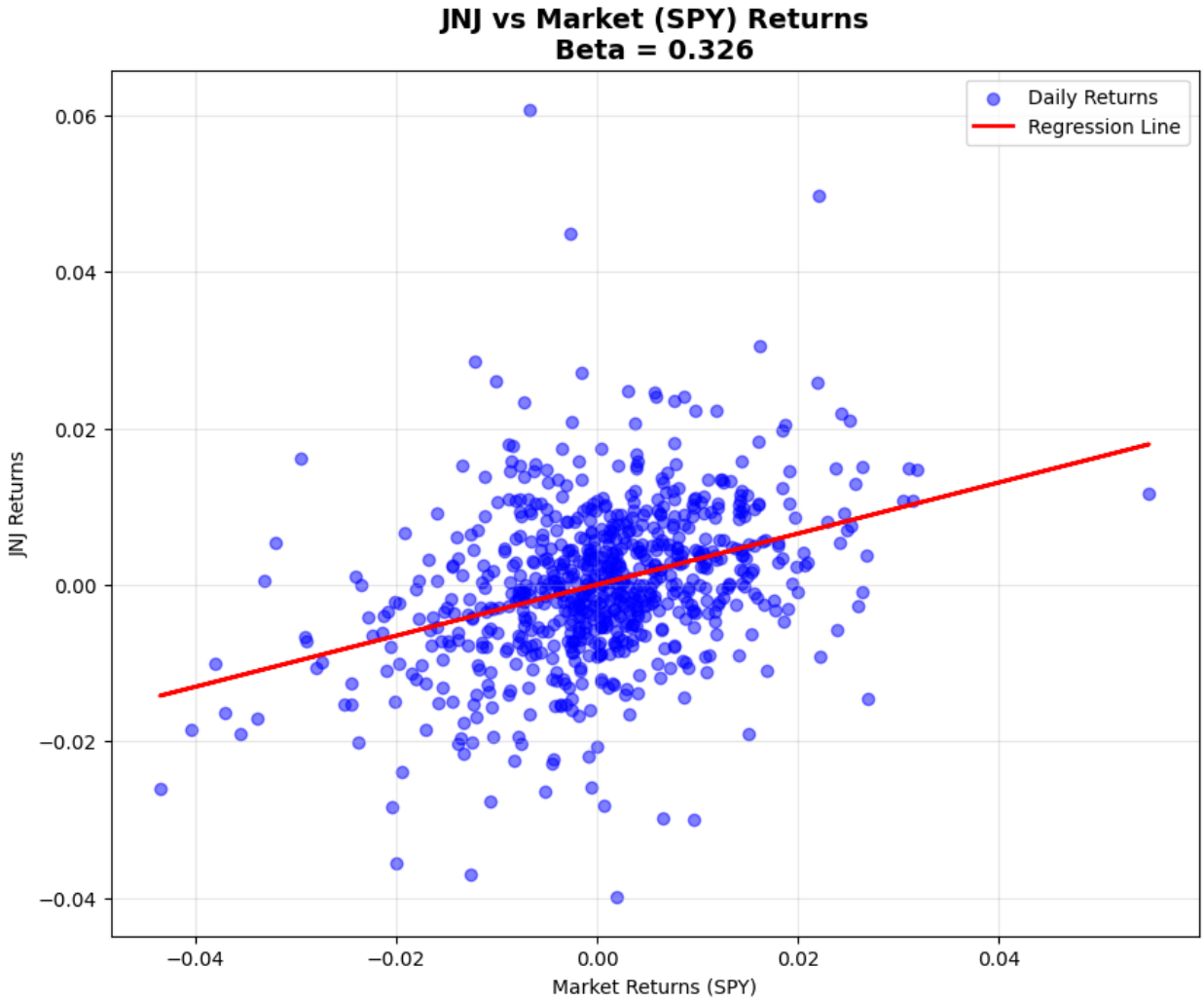
Step 4.2: Beta Visualization

Create scatter plots showing the relationship between stock and market returns:

```
def plot_beta_relationship(stock_returns, market_returns, stock_name,
beta):
    """
    Create scatter plot of stock vs market returns with regression
line.
    """
    plt.figure(figsize=(10, 8))

    ######################
    stock_returns = stock_returns.dropna()
    market_returns = market_returns.dropna()

    # Scatter plot of daily returns
    plt.scatter(market_returns, stock_returns, alpha=0.5,
color='blue', label='Daily Returns')

    # Regression line
```

```
    X = market_returns.values.reshape(-1, 1)
    y = stock_returns.values
    reg = LinearRegression().fit(X, y)
    y_pred = reg.predict(X)
    plt.plot(market_returns, y_pred, color='red', linewidth=2,
label='Regression Line')

    # Title with beta value
    plt.title(f'{stock_name} vs Market (SPY) Returns\nBeta =
{beta:.3f}', fontsize=14, fontweight='bold')
    plt.xlabel('Market Returns (SPY)')
    plt.ylabel(f'{stock_name} Returns')
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.show()

# Create plots for both stocks
plot_beta_relationship(simple_returns['AAPL'], simple_returns['SPY'],
'AAPL', aapl_beta)
plot_beta_relationship(simple_returns['JNJ'], simple_returns['SPY'],
'JNJ', jnj_beta)
```


AAPL vs Market (SPY) Returns
Beta = 1.272
Daily Returns
Regression Line
AAPL Returns
Market Returns (SPY)
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.04
−0.02
0.00
0.02
0.04



Questions - Part 4

1. Which stock is more sensitive to market movements? How do you know?
2. Based on beta, which stock would you expect to fall more in a market crash?
3. Do the betas make intuitive sense given what you know about these companies?

Your Answers - Part 4:

1. The stock more sensitive to market movements is AAPL, with a Beta of 1.272.

   Beta is a measure of a stock's systematic risk, or its volatility relative to the market. Since AAPL's Beta is greater than 1.0, it is considered more volatile than the market benchmark (SPY). For every 1% move in the market, AAPL is expected to move by 1.272%.

2. You would expect AAPL to fall more in a market crash. A market crash is typically characterized by a large negative movement in the market (SPY). Because AAPL's high Beta (≈1.27) amplifies market movements, a large drop in the market would be amplified into an even larger drop for AAPL. Conversely, JNJ's low Beta (≈0.33)

would dampen the market's fall, causing it to drop significantly less than the market, and much less than AAPL.

3. Yes, the betas make strong intuitive sense.

   a. AAPL (Beta ≈1.27): As a Large Tech Stock, AAPL is a cyclical growth company whose earnings are highly sensitive to the economic cycle and consumer spending. In periods of economic expansion, it performs well (amplifying market gains), and in downturns, it is hit hard (amplifying market losses). This high sensitivity aligns perfectly with a Beta greater than 1.0.

   b. JNJ (Beta ≈0.33): As a Defensive Stock in the healthcare and pharmaceutical sector, JNJ sells essential goods (medicine, medical devices) that people continue to buy regardless of the economic climate. This stability makes its returns less dependent on the overall market's performance, which is exactly why it has a Beta significantly less than 1.0.

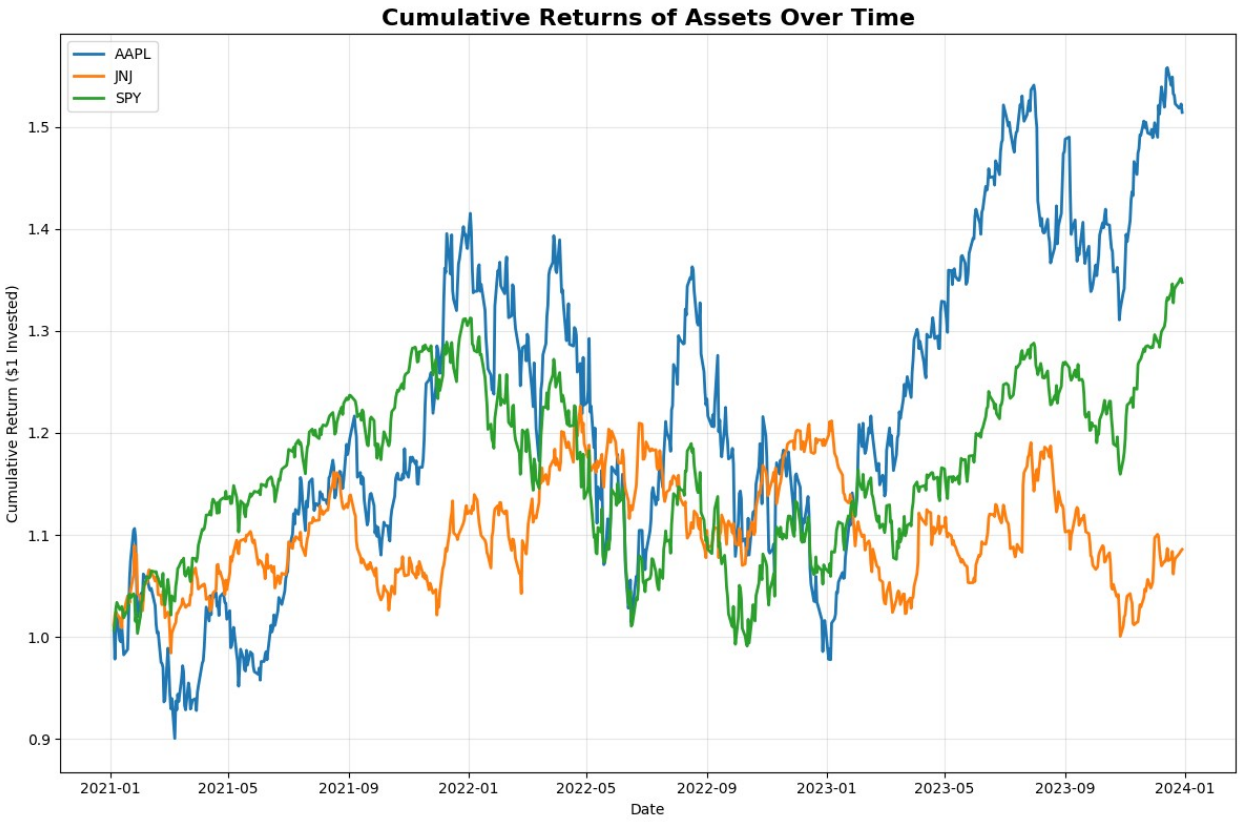
Part 5: Time Series Analysis (15 points)

Step 5.1: Cumulative Returns

Calculate and plot cumulative returns to show total performance:

```
def analyze_cumulative_returns(returns_df):
    """
    Calculate and plot cumulative returns over time.
    """
    # Calculate cumulative returns (compound growth)
    cum_returns = (1 + returns_df).cumprod()

    # Plot cumulative returns
    plt.figure(figsize=(12, 8))
    ######################
    for asset in cum_returns.columns:
        plt.plot(cum_returns.index, cum_returns[asset], label=asset,
linewidth=2)

    plt.title('Cumulative Returns of Assets Over Time', fontsize=16,
fontweight='bold')
    plt.xlabel('Date')
    plt.ylabel('Cumulative Return ($1 Invested)')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

    return cum_returns
```

```
cumulative_returns = analyze_cumulative_returns(simple_returns)

# Calculate total returns over the period
total_returns = cumulative_returns.iloc[-1] - 1
print("Total Returns over the period:")
for asset in total_returns.index:
    print(f"{asset}: {total_returns[asset]:.2%}")
```




```
Total Returns over the period:
AAPL: 51.40%
JNJ: 8.57%
SPY: 34.74%
```

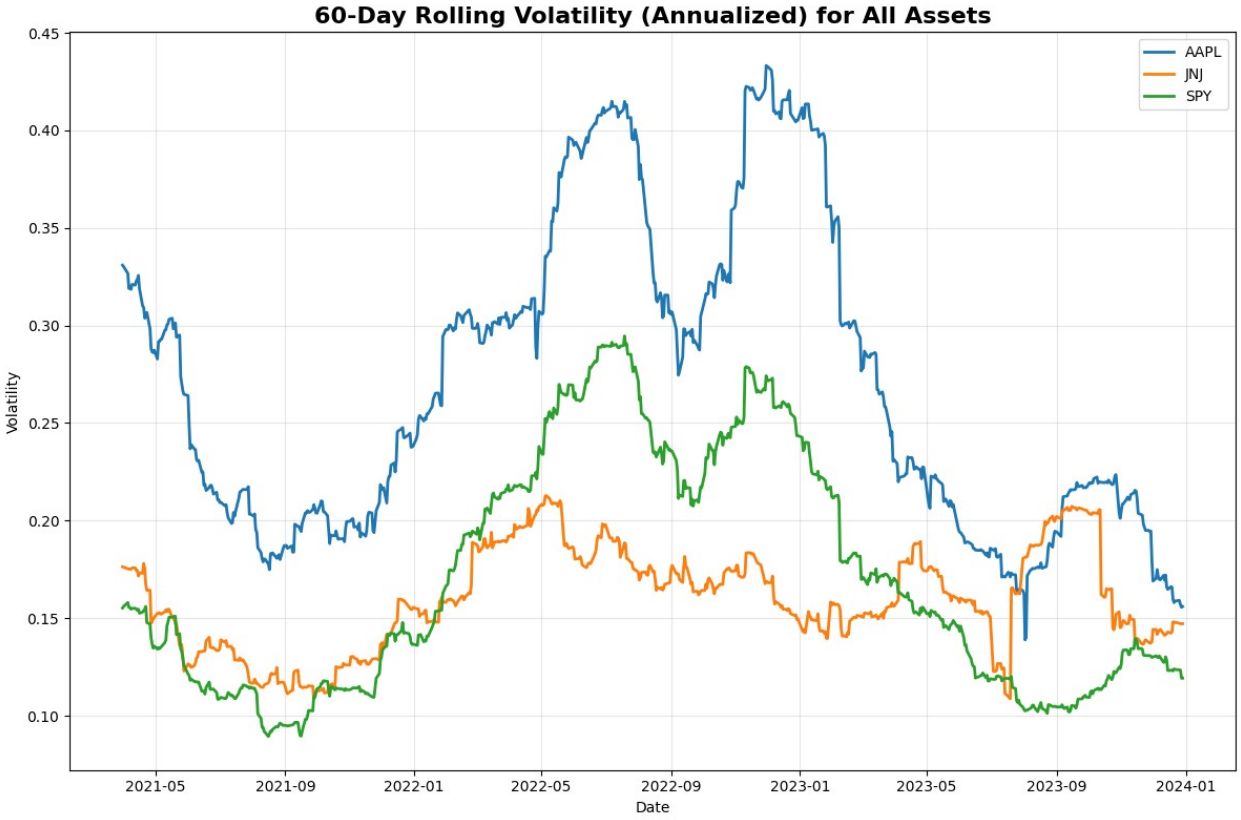
Step 5.2: Rolling Statistics

Calculate rolling volatility to see how risk changes over time:

```
def plot_rolling_volatility(returns_df, window=60):
    """
    Plot 60-day rolling volatility for all assets.
    """
    # Calculate rolling standard deviation
```

```
    rolling_vol = returns_df.rolling(window=window).std() *
np.sqrt(252)

    plt.figure(figsize=(12, 8))
    ######################
    plt.figure(figsize=(12, 8))
    for asset in rolling_vol.columns:
        plt.plot(rolling_vol.index, rolling_vol[asset], label=asset,
linewidth=2)

    plt.title(f'{window}-Day Rolling Volatility (Annualized) for All
Assets', fontsize=16, fontweight='bold')
    plt.xlabel('Date')
    plt.ylabel('Volatility')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

plot_rolling_volatility(simple_returns, window=60)
```

```
<Figure size 1200x800 with 0 Axes>
```



Questions - Part 5

1. Which asset had the best total return? Was this expected based on risk levels?
2. During which time periods was volatility highest? Can you guess why?
3. How does rolling volatility help us understand changing market conditions?

Your Answers - Part 5:

1. Based on the total returns over the period, the asset with the best total return over the period was AAPL (51.48%). Expectation Based on Risk:

   Yes, this was expected. In Part 2, we found that AAPL had the highest Annual Volatility (risk) at 0.2780 and the highest Annual Return at 0.1769. Financial theory suggests a positive relationship between risk and reward: the asset that historically takes on the most volatility (risk) should, over time, offer the highest reward (return). The cumulative returns plot visually confirms this, showing the AAPL line consistently above the others by the end of the period.

2. Based on the 60-Day Rolling Volatility plot:

   Highest Volatility Period Volatility was highest for all assets during the mid-2022 period (approximately early 2022 through mid-2023). During this time, the volatility lines for all three assets spiked significantly, with AAPL peaking above 0.40 (or 40%) annualized volatility.

   Possible Reason: This period corresponds to the peak of inflation fears and the start of the Federal Reserve's aggressive interest rate hiking cycle. Rising rates typically hit high-growth, high-Beta stocks like AAPL the hardest, which explains why AAPL's volatility spiked the highest.

   The market uncertainty caused by the sudden shift from a low-rate to a high-rate environment drove volatility across the entire market (SPY) and even affected defensive stocks like JNJ.

3. Rolling volatility provides a key benefit over static (fixed-period) volatility because it shows how the risk of an asset changes over time, rather than giving a single, unchanging number for the entire period.

   It helps us understand changing market conditions in the following ways:

   a. Identifies Risk Regimes: It clearly shows periods of low stability (high volatility spikes, e.g., 2022) versus periods of calm (low, flat volatility, e.g., late 2023).

   b. Aids Dynamic Portfolio Management: A fund manager can use rolling volatility to dynamically adjust portfolio risk (e.g., reduce exposure to high-Beta stocks when rolling volatility spikes, or increase exposure when it falls).

   c. Reveals Asset-Specific Shocks: It shows whether a volatility change is market-wide (all lines move together, e.g., 2022) or due to an asset-specific shock (only one line spikes), providing better insight into the source of the risk.

Part 6: Summary Analysis and Interpretation (10 points)

Write a brief analysis (300-500 words) answering these questions:

Investment Summary

Based on your analysis, write responses to these prompts:

1. **Risk-Return Profile**: Summarize the risk and return characteristics of each asset. Which offered the best risk-adjusted returns?
2. **Diversification Benefits**: Explain whether combining AAPL and JNJ in a portfolio provided diversification benefits. Use specific numbers from your analysis.
3. **Market Sensitivity**: Compare how AAPL and JNJ respond to market movements using your beta analysis. What does this mean for an investor?
4. **Time-Varying Risk**: Describe how volatility changed over your sample period. What events might explain these changes?
5. **Investment Recommendation**: If you had to choose between investing in individual stocks or the diversified portfolio, what would you recommend and why?

Investment Analysis Summary: AAPL, JNJ, and SPY

The risk-return analysis over the sample period reveals that the assets conformed to the expected financial principle: higher historical risk correlated with higher historical return. Apple (AAPL) was the highest risk asset (Annual Volatility: 0.2780) and delivered the highest Total Return (51.48%). Conversely, Johnson & Johnson (JNJ) was the lowest risk asset (Annual Volatility: 0.1619) with the lowest Total Return (8.31%), positioning it as a defensive holding. The S&P 500 benchmark (SPY) and the equal-weighted portfolio fell in the middle. Crucially, despite its higher absolute risk, AAPL offered the best risk-adjusted return, demonstrated by the highest Sharpe Ratio of 0.5668, providing the greatest return compensation for the level of volatility incurred.

A significant benefit was gained by combining AAPL and JNJ into an equal-weighted portfolio. With a low correlation of 0.2396, the assets moved somewhat independently, which successfully mitigated overall risk. This diversification benefit resulted in the portfolio's volatility (0.1768) being notably lower than the simple average of the individual volatilities (0.2199), equating to 0.0431 (4.31 percentage points) of risk being eliminated. This low correlation proves the strategy effective for creating a more efficient frontier.

Market Sensitivity, measured by Beta, highlighted the vastly different roles these two stocks play in a portfolio. AAPL, with a high Beta of 1.272, is highly sensitive to the overall market; its returns tend to amplify the market's direction, making it a high-growth, cyclical investment. JNJ, conversely, is a highly defensive stock with a low Beta of 0.326, meaning it dampens market movements and provides stability, making it resilient during market downturns. This stark

contrast confirms that AAPL is suitable for aggressive investors seeking leverage to market gains, while JNJ is suited for conservative investors focused on capital preservation.

The analysis of rolling volatility demonstrated that risk is not static, varying significantly over the sample period. The highest volatility spike for all assets occurred between early 2022 and mid-2023 , when AAPL's annualized volatility briefly exceeded 40%. This period of heightened risk aligns with the widespread market uncertainty caused by the Federal Reserve's swift and aggressive campaign of interest rate hikes aimed at controlling inflation. This highlights how market conditions can dynamically alter the risk profile of even fundamentally strong companies.

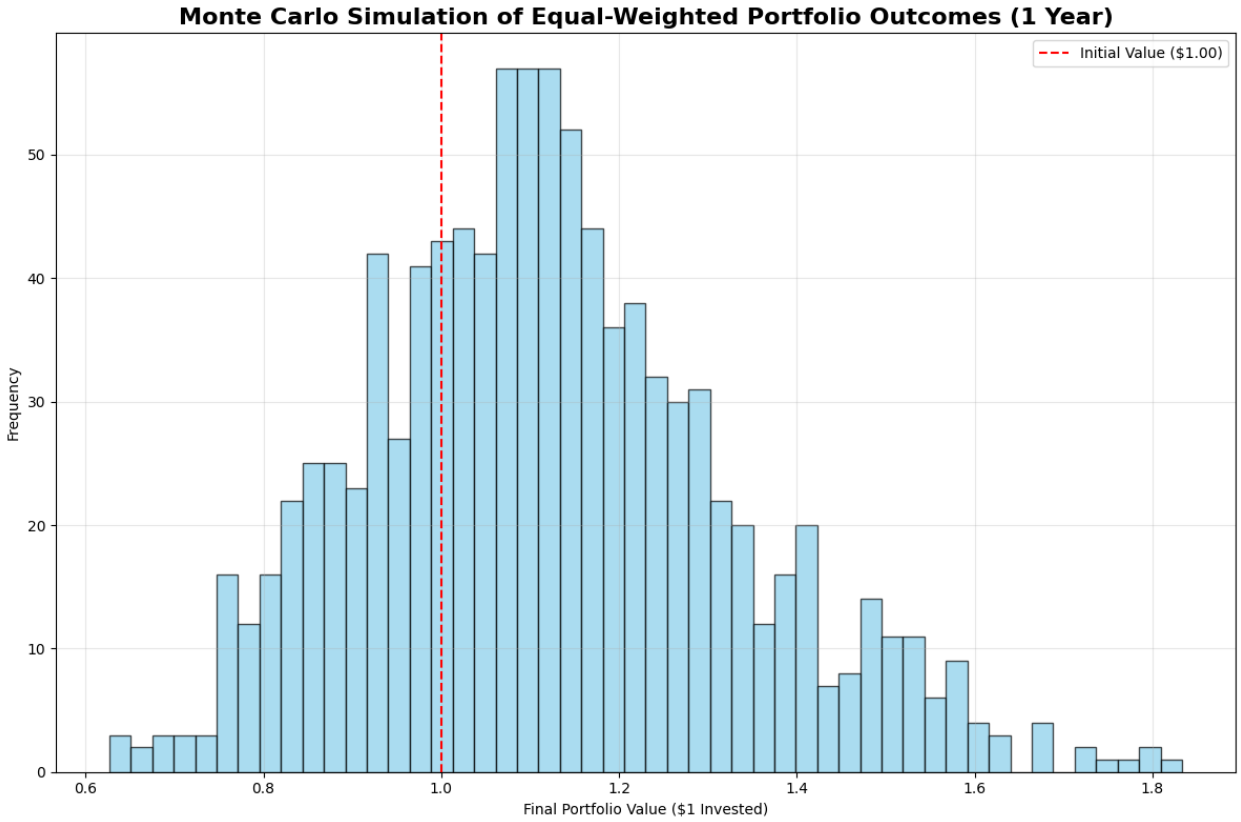
Given these findings, I would recommend the diversified equal-weighted portfolio over investing in any single stock. The portfolio successfully delivered a solid Annual Return (15.55%) while maintaining low risk, achieving an Annual Volatility (0.1768) that was competitive with the broad market index (SPY). Most importantly, the resulting portfolio offered a superior risk-adjusted return (Sharpe Ratio of 0.804 versus SPY's 0.5302), positioning it as the most efficient and balanced investment choice among the options analyzed.

Bonus Section: Probability Application (5 extra points)

Apply probability concepts from Week 1 slides:

```
def simulate_portfolio_outcomes(returns_df, num_simulations=1000,
time_horizon=252):
    """
    Use Monte Carlo simulation to project potential portfolio
outcomes.
    Assume returns follow a normal distribution and the stock price
follows a geometric Brownian motion.
    """
    # Assuming returns_df contains all assets, we need to manually
select AAPL and JNJ
    # to create the equal-weighted portfolio returns, as defined in
previous steps.

    # Step 1: Calculate historical mean and std for the equal-weighted
portfolio
    # Assuming the equal-weighted portfolio of AAPL and JNJ
    portfolio_returns = returns_df[['AAPL', 'JNJ']].mean(axis=1)

    # Historical daily mean (mu) and daily standard deviation (sigma)
    mu = portfolio_returns.mean()
    sigma = portfolio_returns.std()

    # Step 2: Simulate portfolio outcomes
    final_portfolio_values = []

    for _ in range(num_simulations):
```

```
        # Simulate daily returns for next year (time_horizon days)
        # Using historical mu and sigma for the normal distribution
        simulated_daily = np.random.normal(mu, sigma, time_horizon)

        # Compute cumulative growth (Geometric Brownian Motion)
        # Final value = (1+r1) * (1+r2) * ... * (1+rn)
        final_value = np.prod(1 + simulated_daily)
        final_portfolio_values.append(final_value)

    final_portfolio_values = np.array(final_portfolio_values)

    # Step 3: Plot histogram of outcomes
    plt.figure(figsize=(12, 8))
    plt.hist(final_portfolio_values, bins=50, color='skyblue',
edgecolor='black', alpha=0.7)

    # Add a vertical line at the starting value (1.0)
    plt.axvline(1.0, color='red', linestyle='dashed', linewidth=1.5,
label='Initial Value ($1.00)')

    plt.title('Monte Carlo Simulation of Equal-Weighted Portfolio
Outcomes (1 Year)', fontsize=16, fontweight='bold')
    plt.xlabel('Final Portfolio Value ($1 Invested)')
    plt.ylabel('Frequency')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

    # Step 4: Probability of losing money
    prob_loss = np.mean(final_portfolio_values < 1)
    print(f"Probability of losing money over 1 year: {prob_loss:.2%}")

    return final_portfolio_values

# Run simulation
simulate_portfolio_outcomes(simple_returns)
```




```
Probability of losing money over 1 year: 28.00%

array([1.37943746, 1.28081068, 1.08013902, 1.14611434, 1.0251811 ,
       1.13556309, 1.27198458, 1.28297233, 1.23039975, 1.41062127,
       1.27384932, 1.10952174, 0.84457564, 1.18122165, 1.11810434,
       1.12890901, 1.36833065, 1.11343482, 1.16088475, 1.38353021,
       1.53778323, 1.08068677, 1.1010742 , 0.92935904, 1.2225108 ,
       1.16227652, 0.93860376, 0.74876249, 1.14782717, 1.39311175,
       1.48329218, 1.16502329, 1.09934294, 1.2132086 , 1.11000595,
       0.90158627, 1.25747763, 1.08070644, 0.82766275, 1.20051616,
       1.10048302, 1.13657089, 1.03572808, 1.07564952, 0.86183188,
       1.04776127, 1.47947436, 1.09491203, 1.32369386, 1.25989141,
       1.19442597, 1.079402  , 1.37870443, 0.77743659, 1.68156855,
       1.05807345, 0.93829964, 1.30853655, 1.0832512 , 0.97745881,
       1.18589191, 0.91055742, 1.14407776, 0.99235736, 0.86423938,
       1.17201336, 1.15721646, 1.13657402, 1.23658477, 0.70448423,
       1.12264702, 1.21025895, 0.92201023, 1.1442707 , 1.00668898,
       1.3273737 , 1.25371976, 0.92075795, 1.02912803, 1.58158629,
       1.26233074, 1.27719465, 1.3801409 , 1.22007627, 0.98098361,
       0.86617388, 0.96137422, 1.0698117 , 0.81283745, 0.92964367,
       1.58186194, 1.10341987, 1.01240141, 1.15109451, 1.11411234,
       1.1039584 , 1.073725  , 1.03273648, 1.08950563, 1.01212179,
       1.4789745 , 0.75922529, 0.93674489, 0.94272373, 0.97104303,
       1.24118548, 0.93660572, 0.89275348, 1.08046886, 1.49389688,
       0.80596753, 1.04670776, 1.24679758, 1.16118677, 0.88171248,
```

```
      1.06509997, 1.04022386, 0.90067431, 1.2699085 , 1.02561946,
      0.88761968, 1.43017351, 1.21812626, 1.0576059 , 0.99861966,
      1.09842632, 1.10618262, 1.27850832, 1.07010036, 1.11861109,
      0.94830849, 1.02517656, 1.04850391, 1.00136368, 0.84426191,
      1.08504311, 1.31282619, 0.95002937, 1.14544481, 1.17837833,
      0.75201282, 1.32112847, 1.09342714, 1.04162562, 1.02801039,
      1.11086122, 1.18541924, 1.40005938, 1.03557965, 0.75056687,
      1.07702934, 1.08924419, 1.35026614, 0.79497922, 1.28639202,
      0.91742114, 1.20302725, 0.91095468, 0.97814305, 1.18737401,
      1.26072708, 1.02370112, 1.1412716 , 1.09940137, 0.95662686,
      0.96382898, 1.09957489, 0.86811654, 1.26625999, 0.93617426,
      0.86068649, 0.97985262, 0.99802592, 0.94124164, 1.30347709,
      1.13855193, 1.21737789, 1.33657024, 1.07881169, 1.48098092,
      0.8836048 , 0.88112335, 0.88408995, 0.82470241, 1.30382218,
      1.37623499, 1.33235106, 1.62478043, 1.31375596, 0.98012438,
      1.48674761, 1.07892703, 1.07204129, 1.46201649, 1.24361943,
      1.51772971, 1.08553475, 1.22141895, 0.83974425, 1.39157785,
      1.51268737, 1.28059889, 1.53334803, 1.08642276, 0.75028357,
      0.92603754, 1.09512101, 1.602784  , 0.9990307 , 1.27360936,
      1.18158705, 1.21399543, 1.27986581, 1.28781159, 0.92776119,
      1.08537678, 1.07116617, 1.04878473, 1.11941593, 0.88317689,
      1.04206691, 0.92346549, 1.42212285, 0.97957959, 0.98253143,
      1.31812728, 1.03436138, 1.05215359, 1.48051743, 1.5275812 ,
      1.49981467, 1.11275533, 1.08060411, 1.08655094, 0.97475364,
      1.22335655, 1.22380134, 1.05974002, 1.25189943, 1.4207774 ,
      1.09169509, 1.09448126, 1.21245741, 1.0947803 , 1.10892369,
      1.34236248, 1.14126231, 1.13683754, 1.57648695, 1.11283138,
      1.36620503, 1.53128967, 1.80409222, 1.27378155, 1.01598948,
      1.16358209, 0.98945099, 1.1950476 , 1.17558002, 1.43794758,
      0.98857706, 1.03917088, 1.00842167, 1.38688511, 0.94849623,
      1.06021293, 0.99518439, 0.98228388, 0.92161761, 1.54510771,
      0.94325503, 1.22476582, 1.50605315, 0.90577605, 0.92828957,
      1.2537803 , 1.13800785, 1.45224074, 1.11859115, 1.16081059,
      1.34358451, 1.42268898, 0.99202523, 1.09297915, 1.0124851 ,
      1.19517808, 1.23290436, 0.8336069 , 0.95235568, 0.97412174,
      1.0796444 , 1.34121964, 1.17785834, 0.81712883, 0.87289935,
      1.0571101 , 1.18970427, 0.93934794, 0.84143459, 0.95815093,
      1.25520103, 0.95661312, 0.82433545, 0.96576011, 0.87764937,
      1.16734229, 1.2204785 , 1.1225759 , 1.05675841, 0.98429182,
      1.13759583, 1.09676209, 1.01236599, 1.055035  , 1.18057127,
      1.14949383, 1.14716169, 1.02098179, 1.11296138, 0.95319956,
      1.13186429, 1.41727366, 0.88995169, 0.91589573, 0.96619939,
      1.33859557, 0.99170989, 1.07431961, 1.14365391, 1.22213625,
      0.92881987, 1.44462031, 1.16426608, 1.02212581, 1.05371174,
      1.24213416, 1.61651491, 1.16860337, 1.01267797, 0.8455819 ,
      1.79302709, 0.87706656, 1.34285988, 1.13042871, 1.02848133,
      0.80924925, 1.06793352, 1.18949994, 0.93458312, 0.9213518 ,
      1.59962596, 1.25770723, 0.99591816, 1.21645121, 1.07615533,
      1.21370249, 0.799243  , 1.1360568 , 1.19809511, 1.12749839,
```

```
      1.57051501, 1.32804592, 1.29347313, 0.75048918, 1.00610681,
      1.18530692, 1.25715858, 1.68480371, 1.26467659, 0.93594007,
      1.08367048, 0.94531484, 1.56337674, 1.63992193, 1.15769772,
      1.71609854, 1.32675894, 1.24922587, 1.05474591, 0.98121017,
      1.02178514, 1.16230858, 1.46909813, 1.17652484, 1.04540658,
      1.21643613, 1.12425278, 1.36151481, 1.28573591, 0.90140731,
      0.7912367 , 0.91063399, 1.08482573, 0.82007853, 1.59989399,
      1.18831017, 1.53004958, 1.17968421, 0.81401995, 1.07289098,
      0.79705546, 1.13254698, 1.23710295, 1.06699538, 0.81420651,
      1.14791987, 1.17414335, 1.19382303, 1.21080435, 1.25175689,
      1.16129022, 1.22071624, 0.72750861, 1.4502906 , 0.87246154,
      0.7114294 , 1.10412786, 1.35199036, 1.23967643, 1.06270394,
      1.13123549, 1.14906133, 0.9691043 , 0.98726003, 1.27898785,
      1.4136552 , 0.77542022, 1.05839828, 1.21604207, 1.56417186,
      1.07405036, 0.77991411, 1.12721842, 1.50811436, 1.10236883,
      1.43107291, 0.85952217, 1.01055173, 1.4715582 , 1.08866052,
      1.19866428, 0.93297015, 1.24989057, 1.18069556, 1.09552446,
      0.82861914, 1.29913245, 1.07412685, 1.0340635 , 1.00013982,
      1.12365314, 1.30569516, 0.96069003, 1.13198645, 1.0440829 ,
      1.1025703 , 1.05825218, 1.25455428, 1.22830075, 0.98358561,
      1.33730396, 1.49782097, 1.17155294, 0.88320038, 1.03591415,
      1.40849982, 1.23738073, 0.96967203, 1.31622157, 1.12120432,
      1.47666185, 1.14707123, 1.12516477, 1.41607432, 1.12681808,
      0.75894696, 0.86798747, 1.23444429, 1.52698801, 1.18226282,
      0.85964927, 0.77935872, 1.06084929, 0.7590311 , 0.89128451,
      0.97285035, 1.09624538, 1.13829533, 1.05777982, 1.09862227,
      1.269836  , 1.3136706 , 1.10330763, 1.03443867, 1.11352806,
      1.19892054, 1.26973716, 1.08196887, 0.93662447, 1.13888458,
      0.95927394, 0.96557817, 1.00304721, 0.91602278, 1.05275677,
      1.36118459, 1.0899427 , 1.0931499 , 0.93475156, 1.22583844,
      1.54298166, 1.41342313, 1.40483379, 1.06697106, 1.00208046,
      1.14968945, 1.29106798, 1.27612103, 0.9830845 , 0.92524546,
      1.17256373, 1.27563813, 1.04289627, 1.11066117, 1.34456441,
      1.58051432, 1.22467632, 1.07473705, 1.06970219, 1.04993397,
      1.11605285, 1.04489814, 0.93807908, 1.08053242, 1.08536169,
      1.23563859, 1.24135124, 1.61190293, 1.19638054, 1.1616381 ,
      1.12812768, 1.08885639, 1.06803554, 1.22706245, 1.14549618,
      0.97125498, 1.12671993, 0.98690591, 0.8844846 , 1.24185699,
      1.0021484 , 1.11834847, 1.83326209, 1.27875302, 0.84936501,
      0.65277729, 0.64766287, 1.17207408, 1.58442469, 1.04151652,
      1.18360775, 1.15102713, 1.68140421, 1.28084146, 1.25489826,
      0.87115922, 0.95869271, 1.04284104, 1.17313209, 0.81448934,
      1.24902945, 1.56988782, 1.14381572, 1.11753906, 1.16530394,
      1.00229895, 1.02752245, 0.89200322, 1.07689562, 1.1968718 ,
      0.84320534, 0.80157703, 0.75152863, 1.19736522, 1.11313729,
      1.28122019, 1.07650168, 1.39102892, 1.2298998 , 1.46413208,
      1.07889183, 1.26140379, 1.17170347, 1.24891866, 0.91041503,
      1.35761015, 0.91211842, 1.45095887, 1.28262129, 0.92214745,
      1.5260295 , 1.22596343, 1.00827208, 0.75789446, 1.01911348,
```

```
      1.16192857, 1.67073698, 1.54363941, 0.9215812 , 0.75618291,
      1.02737355, 1.22922026, 1.13783053, 1.14608259, 1.0814399 ,
      1.50200239, 0.83157762, 0.88512025, 1.14305821, 1.1204309 ,
      0.81307891, 1.01417453, 1.48045813, 1.28182313, 1.12227373,
      1.48404024, 0.97229113, 0.90435475, 1.22760444, 0.83903696,
      0.986358  , 1.12649358, 1.03160404, 1.3243681 , 0.79471573,
      1.27730565, 0.8903995 , 1.24491875, 1.20399261, 1.0736622 ,
      1.01666367, 1.28265024, 0.83751455, 1.15027353, 1.13240529,
      0.84074069, 1.16861678, 1.22183022, 1.08568728, 1.0310176 ,
      1.10334074, 1.31468693, 1.00784103, 0.99175626, 1.4233511 ,
      0.99465859, 1.13173286, 1.3761133 , 1.26962392, 1.09833432,
      0.97089288, 1.03090413, 0.66701442, 0.68793309, 1.20463692,
      1.03203378, 1.21710523, 1.10553636, 1.04244024, 1.13310823,
      1.23904565, 1.00577186, 1.41910218, 0.96515031, 1.29919715,
      0.79948749, 1.03797401, 1.46876676, 1.16199079, 0.96885188,
      1.36666753, 0.87095169, 0.84656812, 1.20521657, 1.07496866,
      1.31776382, 0.95480198, 0.85809586, 1.26770602, 1.19616306,
      1.10547394, 1.26292451, 1.28913151, 0.84893994, 1.39444181,
      1.11968114, 0.98472215, 1.5032443 , 0.83063186, 1.13029493,
      0.9330956 , 1.23365184, 0.89042376, 1.33353317, 1.18602357,
      1.41419032, 0.95973037, 0.76389101, 1.10077364, 0.95448749,
      1.22211166, 1.15870155, 1.41613851, 0.90414429, 1.30501535,
      1.17698661, 1.10411504, 1.18708027, 1.1778643 , 1.27485746,
      1.41003066, 1.4206912 , 1.0618001 , 1.15739361, 1.52200285,
      0.90483684, 0.76362303, 1.28879218, 1.38112982, 1.5441751 ,
      1.40264154, 1.24213124, 1.35111705, 1.14477154, 1.34490738,
      1.12484719, 0.95045137, 1.17894931, 0.98882915, 1.35843266,
      1.26812674, 1.74704264, 0.77256937, 0.84461253, 0.96413944,
      1.31466383, 0.85620608, 1.04923551, 1.11191199, 0.92949382,
      1.15408122, 0.9607162 , 1.14916121, 1.28622001, 0.92837447,
      1.14164893, 1.13476408, 1.08964306, 1.00973721, 1.21621068,
      1.0889623 , 0.98632303, 1.27977891, 1.1549483 , 1.24489079,
      0.90920017, 1.08579285, 1.52163221, 1.15358477, 1.33360549,
      0.88802352, 1.02100272, 1.15813199, 1.23302952, 0.73476074,
      1.28582852, 1.18274252, 1.3790524 , 1.08360771, 1.08344764,
      1.31483966, 1.1511023 , 1.29413844, 1.13705203, 1.00231363,
      1.14926395, 0.82223316, 1.13718486, 1.06139018, 1.03010707,
      1.39076171, 1.06380795, 1.08363666, 1.13121747, 1.02451984,
      0.93107849, 0.76565068, 1.00220729, 1.03689047, 0.9697211 ,
      1.28671401, 1.13008147, 0.81028693, 0.96922366, 0.89756168,
      1.02694153, 0.97974625, 1.46848961, 0.86605947, 0.85602394,
      1.4859584 , 1.01678897, 1.04745098, 1.15607574, 1.01795063,
      0.92348923, 1.08405081, 1.12332145, 1.77789593, 0.99476655,
      1.03092151, 1.33723028, 1.19177169, 1.11161233, 1.05907195,
      0.79538097, 1.31928759, 1.19693407, 1.02302828, 0.93275003,
      0.9557551 , 1.35091017, 1.10411226, 1.14626673, 1.55892185,
      1.09330655, 0.86324015, 0.9369589 , 1.51234814, 1.14433102,
      1.13367709, 0.90200795, 1.35877892, 1.22635269, 0.99910704,
      1.24294082, 1.33785791, 1.10609898, 1.43655416, 1.09743995,
```

```
       1.28461201, 1.14958293, 1.16954277, 1.39897054, 0.96583997,
       1.08421489, 0.86987502, 0.8430027 , 1.32698492, 1.23526446,
       1.50212354, 1.11807473, 1.21884272, 1.19072074, 1.42614386,
       1.08199807, 0.85985634, 0.79000491, 0.79362145, 1.2922133 ,
       1.12929199, 1.22579375, 1.0847851 , 0.85470479, 1.06029611,
       0.98934511, 0.90997651, 1.02602755, 1.03660084, 1.16801411,
       1.04355417, 1.08098043, 1.01198806, 1.47448643, 1.37933843,
       1.09570546, 1.20998795, 1.03916063, 0.86508036, 0.86023566,
       0.96854419, 1.00291285, 1.57687765, 0.82676753, 0.99676836,
       1.71896877, 1.57003689, 1.19684469, 0.82322312, 1.15891595,
       0.62673759, 1.13100178, 0.94035089, 0.9158913 , 1.19979575,
       0.92217605, 0.68633403, 1.02304703, 1.18788901, 0.92747025,
       0.93776671, 1.41450598, 1.02168994, 0.78914342, 1.08450884,
       1.16606006, 1.01101798, 1.06552763, 1.00928371, 1.49380232,
       1.31957582, 0.83160889, 1.26473419, 1.12510399, 0.75696843,
       1.34161887, 0.81867192, 0.82936018, 1.02587554, 1.06812538,
       1.04042938, 0.88288893, 0.94942624, 1.01976396, 0.99602474,
       0.92842678, 1.03664521, 0.84657922, 1.04621765, 0.94112333,
       0.69300563, 1.37564695, 0.7402928 , 0.7072694 , 0.92518908,
       1.28971467, 0.97230481, 1.2965512 , 1.00145486, 0.64177822,
       0.96846493, 1.25456484, 1.03616786, 1.09204729, 0.93587057,
       1.10989901, 1.1112379 , 0.91002837, 1.18462495, 1.37115063,
       1.2899653 , 0.88799417, 1.08999449, 0.95033253, 0.97104215,
       1.10981921, 1.09789049, 1.17170934, 1.31531601, 0.86190544,
       1.23373463, 0.81435333, 1.11320785, 1.23215475, 0.90605458,
       1.14797763, 1.0947971 , 0.76443448, 1.49875406, 1.56355519,
       0.80753588, 1.16019811, 1.14186634, 1.22592963, 1.20284878,
       1.04301677, 0.96739889, 1.08062708, 1.08572372, 1.10040516,
       1.40563689, 0.89844477, 0.92701101, 1.07355654, 1.03758197,
       1.2252657 , 0.83033949, 0.8320787 , 1.31413716, 1.32454667])
```

Submission Checklist

Before submitting, make sure you have:

- ☐ Filled in your name and GT username at the top
- ☐ Completed all code sections with working implementations
- ☐ Answered all 15 numbered questions in the markdown cells
- ☐ Written the 300-500 word summary analysis
- ☐ All code cells run without errors
- ☐ All plots display correctly
- ☐ Saved the notebook as `quant_intro_project_[GTUsername].ipynb`
- ☐ Exported a PDF version showing all outputs
- ☐ Push your code to your personal Git repo

Good luck!