Part 1 - HW

Strategy

In [119…

```python
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
pd.reset_option('all')
from math import floor
import requests
from io import StringIO
import requests
import json
import pandas_datareader as web
import datetime
import scipy.optimize as sco
import scipy.interpolate as sci
```

In [120…

```python
def MACD(asset, slow_ema, fast_ema, smooth):

    df = yf.download(tickers = asset, period = "1y", interval = "1d")
    df = df[["Close"]].copy()

    df["Slow EMA"] = df["Close"].ewm(slow_ema).mean()
    df["Fast EMA"] = df["Close"].ewm(fast_ema).mean()

    df["MACD"] = df["Fast EMA"] - df["Slow EMA"]
    df["Signal"] = df["MACD"].ewm(smooth).mean()
    df["Histogram"] = df["MACD"] - df["Signal"]

    plt.figure(figsize = (12.5, 4))

    plt.plot(df.index, df["MACD"], label = "MACD Line")
    plt.plot(df.index, df["Signal"], label = "Signal Line")

    for i in range(0, len(df)):
        if df["Histogram"][i] > 0:
            plt.bar(df.index[i], df["Histogram"][i], color = "green")
        else:
            plt.bar(df.index[i], df["Histogram"][i], color = "red")

    plt.title("Histogram for: " + asset)
    plt.legend()
    plt.show()

    macd_entry_exit(df, asset)
```

In [121…

```python
def macd_entry_exit(df, asset):

    buy = []
    sell = []
    indicator = 0

    for i in range(0, len(df)):
        if df["MACD"][i] > df["Signal"][i]:
            sell.append(np.nan)
            df.loc[df.index[i], "Indicator"] = 1
            if indicator != 1:
                buy.append(df["Close"][i])
                indicator = 1
            else:
                buy.append(np.nan)
        elif df["MACD"][i] < df["Signal"][i]:
            buy.append(np.nan)
            df.loc[df.index[i], "Indicator"] = -1
            if indicator != -1:
                sell.append(df["Close"][i])
                indicator = -1
            else:
                sell.append(np.nan)
        else:
            buy.append(np.nan)
            sell.append(np.nan)

    df["Buy Signal"] = buy
    df["Sell Signal"] = sell

    plt.figure(figsize = (12.5,4) )
    plt.plot(df.index, df["Close"], label = asset)
    plt.scatter(df.index, df["Buy Signal"], color = "green", label = "Buy Signal", marker = "^", alpha = 1)
    plt.scatter(df.index, df["Sell Signal"], color = "red", label = "Sell Signal", marker = "v")
    plt.title("Buy and Sell Strategy: " +  asset)
    plt.legend()
    plt.show()

    strategy_returns(df, asset)
```

In [ ]:

```python
def strategy_returns(df, asset):

    df["% Return"] = df["Close"].pct_change()
    df["Buy And Hold"] = np.cumprod(df["% Return"]+1).replace(np.nan, 1)


    df["Indicator"] = df["Indicator"].shift(1)
    df["Strategy Returns"] = np.cumprod(df["Indicator"] * df["% Return"] + 1).replace(np.nan, 1)


    plt.figure(figsize = (12.5, 4))
    plt.plot(df["Strategy Returns"], label = "Strategy Returns")
    plt.plot(df["Buy And Hold"], label = "Buy And Hold")

    plt.legend()
    plt.title("Profit Curve: " + asset)
    plt.show()

    total_return_strategy = df["Strategy Returns"].iloc[-1] - 1
    total_return_buy_and_hold = df["Buy And Hold"].iloc[-1] - 1

    print(f"Total Return (Strategy): {total_return_strategy:.2%}")
    print(f"Total Return (Buy and Hold): {total_return_buy_and_hold:.2%}")

    investment_value = 10000
    number_of_stocks = floor(investment_value/df["Close"][0])

    strategy_returns = number_of_stocks * (df['Strategy Returns'].iloc[-1] - 1)

    print(f"Total Profit with a ${investment_value} Investment: ${strategy_returns:.2f}")

    #based off 10 year US Treasury Bond Yield
    risk_free_rate = 0.0425

    sharpe_ratio = annual_sharpe_ratio(df, risk_free_rate)

    print(f"Annual Sharpe Ratio: {sharpe_ratio:.2f}")
```

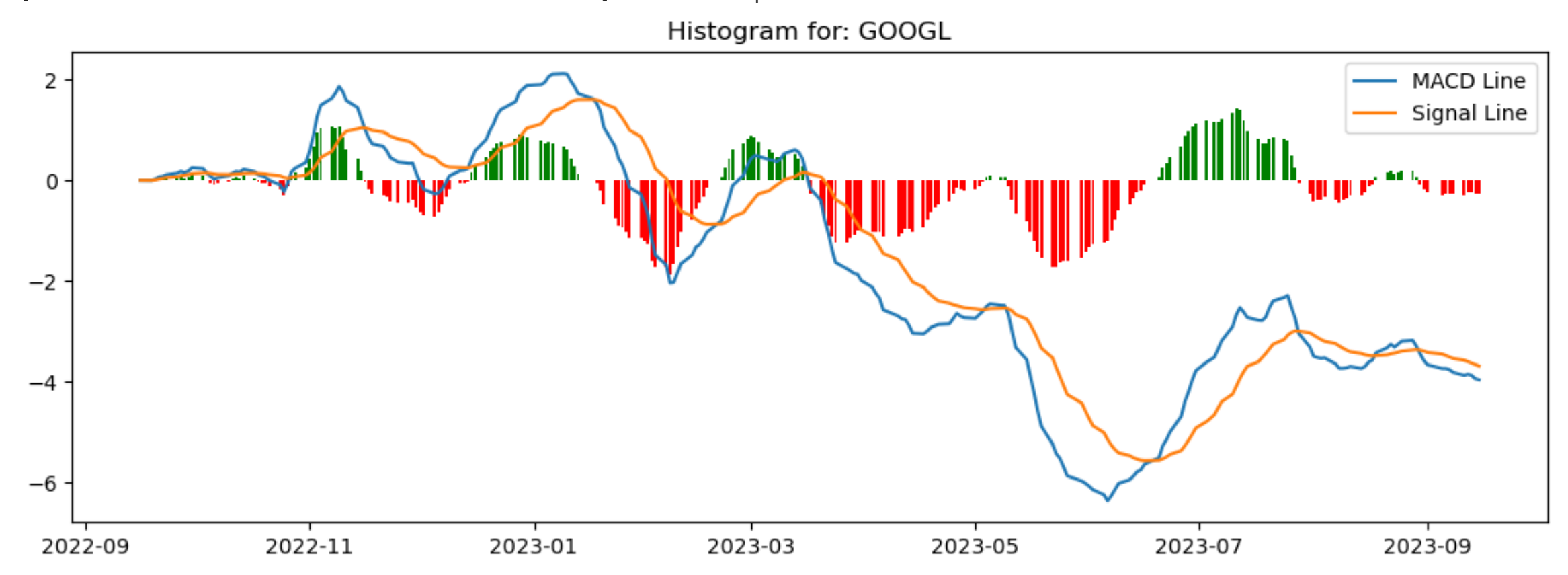
In [ ]:

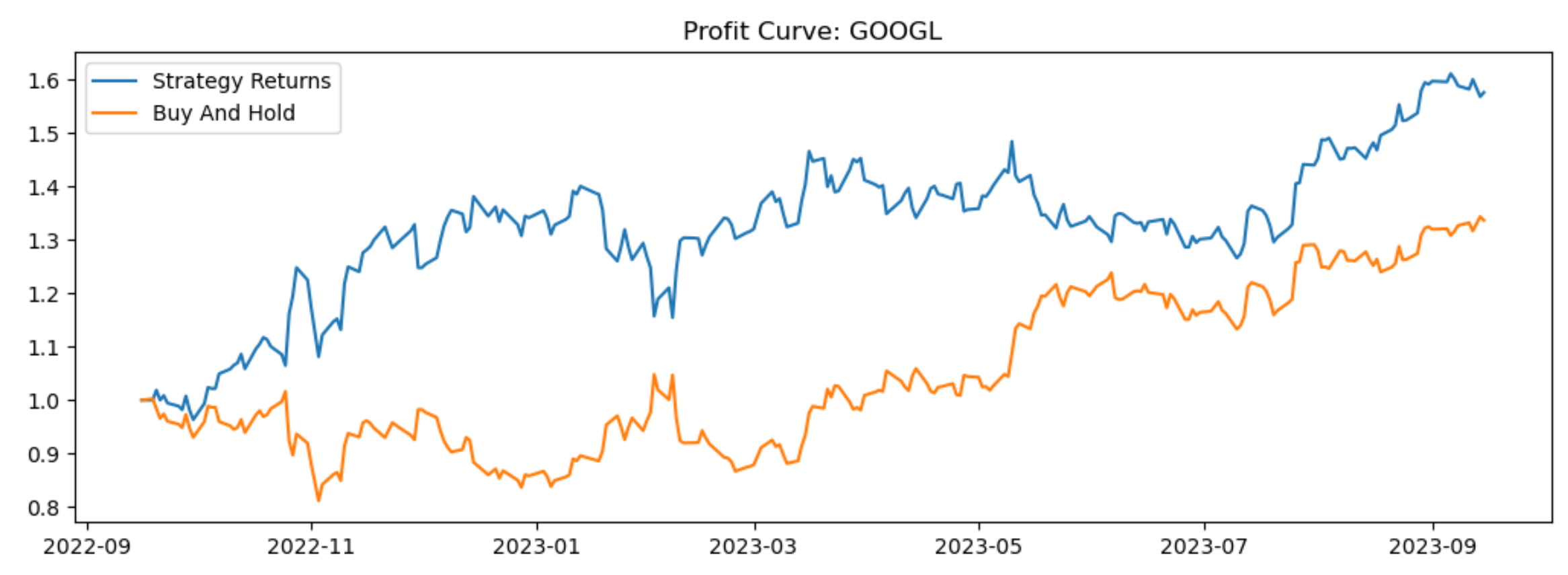
```python
def annual_sharpe_ratio(df, risk_free_rate):

    average_annual_return = df['% Return'].mean() * 252  # Assuming 252 trading days in a year

    excess_return = average_annual_return - risk_free_rate

    annual_std_dev = df['% Return'].std() * np.sqrt(252)

    sharpe_ratio = excess_return / annual_std_dev

    return sharpe_ratio
```

In [124…

```python
slow_ema = 12
fast_ema = 26
smooth = 9

MACD("GOOGL", slow_ema, fast_ema, smooth)
```

```
[*********************100%%**********************]  1 of 1 completed
```






```
Total Return (Strategy): 57.57%
Total Return (Buy and Hold): 33.66%
Total Profit with a $10000 Investment: $55.84
Annual Sharpe Ratio: 0.90
```

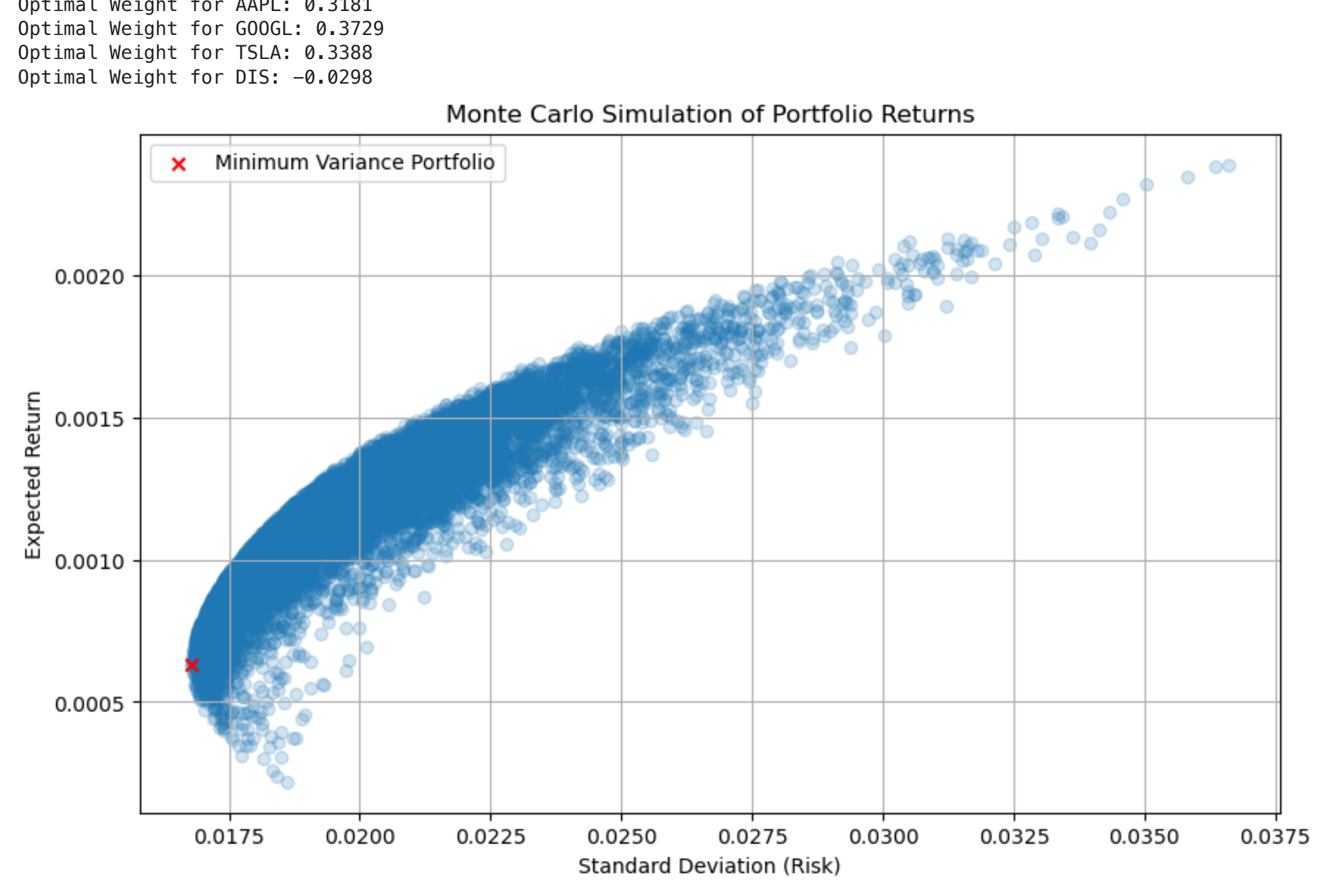
Part 2 - HW

Minimum Variance Portfolio

In [125…

```python
symbols = ["AAPL", "GOOGL", "TSLA", "DIS"]


def get_df_stocks(tickers):
    data = yf.download(
        tickers,
        start = "2018-01-01",
        interval="1d")["Adj Close"]
    return data


df = get_df_stocks(symbols)


def portfolio_variance(weights, returns, cov_matrix):
    portfolio_return = np.sum(weights * returns)
    portfolio_stddev = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    return portfolio_stddev


returns = df.pct_change().mean()
cov_matrix = df.pct_change().cov()


constraints = ({'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1})

initial_weights = [1/len(symbols)] * len(symbols)


result = sco.minimize(portfolio_variance, initial_weights, args=(returns, cov_matrix),
                      method='SLSQP', constraints=constraints)

optimal_weights = result.x
for i, symbol in enumerate(symbols):
    print(f"Optimal Weight for {symbol}: {optimal_weights[i]:.4f}")


num_simulations = 10000


portfolio_returns = np.zeros(num_simulations)
portfolio_stddevs = np.zeros(num_simulations)


for i in range(num_simulations):

    random_weights = np.random.rand(len(symbols))
    random_weights /= random_weights.sum()


    random_portfolio_return = np.sum(random_weights * returns)
    random_portfolio_stddev = np.sqrt(np.dot(random_weights.T, np.dot(cov_matrix, random_weights)))


    portfolio_returns[i] = random_portfolio_return
    portfolio_stddevs[i] = random_portfolio_stddev


plt.figure(figsize=(10, 6))
plt.scatter(portfolio_stddevs, portfolio_returns, alpha=0.2)
plt.xlabel('Standard Deviation (Risk)')
plt.ylabel('Expected Return')
plt.title('Monte Carlo Simulation of Portfolio Returns')


min_variance_idx = portfolio_stddevs.argmin()
min_variance_return = portfolio_returns[min_variance_idx]
min_variance_stddev = portfolio_stddevs[min_variance_idx]
plt.scatter(min_variance_stddev, min_variance_return, color='red', marker='x', label='Minimum Variance Portfolio')


plt.legend()
plt.grid(True)
plt.show()


print(f"Minimum Variance Portfolio Return: {min_variance_return:.4f}")
print(f"Minimum Variance Portfolio Standard Deviation: {min_variance_stddev:.4f}")
```

```
[*********************100%%**********************]  4 of 4 completed
Optimal Weight for AAPL: 0.3181
Optimal Weight for GOOGL: 0.3729
Optimal Weight for TSLA: 0.3388
Optimal Weight for DIS: -0.0298
```




```
Minimum Variance Portfolio Return: 0.0006
Minimum Variance Portfolio Standard Deviation: 0.0168
```

Purchase assets on Alpaca

In [126…

```python
API_KEY = "PKPK4VA4MV6QDQGP5PLB"
SECRET_KEY = "b7BTjQduW1usldizLO4mnAnaZfenhQMRi3j27p8H"
BASE_URL = "https://paper-api.alpaca.markets"
ACCOUNT_URL = BASE_URL + "/v2/account"
ORDERS_URL =  BASE_URL + "/v2/orders"

def create_order(symbol, qty, side, type_, time_in_force):
    data = {
            "symbol": symbol,
            "qty": qty,
            "side": side,
            "type": type_,
            "time_in_force": time_in_force
            }
    r = requests.post(ORDERS_URL, json=data, headers = {'APCA-API-KEY-ID': API_KEY,
                                                        'APCA-API-SECRET-KEY':  SECRET_KEY})

    return json.loads(r.content)
```

In [127…

```python
def purchase_minimum_variance_portfolio_assets(symbols, optimal_weights):

    df_stocks = get_df_stocks(symbols)

    # Step 2: Get account balance
    r = requests.get(ACCOUNT_URL, headers={'APCA-API-KEY-ID': API_KEY,
                                           'APCA-API-SECRET-KEY': SECRET_KEY})
    info = json.loads(r.content)
    account_val = float(info["cash"])

    # Step 3: Calculate the number of shares to purchase
    shares = []
    for i, symbol in enumerate(symbols):
        weight = optimal_weights[i]
        price = df_stocks[symbol][-1]
        qty = (weight * account_val) / price
        qty = qty // 1
        shares.append(qty)

    # Step 4: Purchase assets
    for i, symbol in enumerate(symbols):
        qty = shares[i]
        if qty > 0:
            create_order(symbol, qty, "buy", "market", "gtc")

# Purchase assets for the minimum variance portfolio
purchase_minimum_variance_portfolio_assets(symbols, optimal_weights)
```

```
[*********************100%%**********************]  4 of 4 completed
```

In [ ]:

In [ ]:

In [ ]: