Problem 1

- Closed Form Greeks for GBSM

I define a function called gbsm_greeks that calculates the Greeks (sensitivity measures) of an option using the Black-Scholes-Merton (BSM) model. The function first calculates the time to expiration (tau) and the two d values, which are used in the calculations for the Greeks. Then, depending on the type of option, it calculates the delta, gamma, vega, theta, and rho values using different formulas based on the BSM model. Finally, the function returns the calculated delta, gamma, vega, theta, and rho values as a tuple.

- Finite Difference Derivative

I define a function called finite_difference that calculates the Greeks of an option using finite difference methods. The input parameters are the same as in the previous function, except for an additional parameter delta_shift which is set to a default value of 0.01.

The function first calculates the values of the underlying asset price, volatility, and risk-free interest rate shifted up and down by delta_shift using a simple formula. Then, it defines an inner function called option_price which calculates the option's price using the Black-Scholes-Merton formula based on the shifted input parameters. Using the option_price function, the function calculates the finite difference approximations for delta, gamma, vega, theta, and rho by computing the numerical derivatives using the shifted input parameters. Finally, the function returns the calculated delta, gamma, vega, theta, and rho values as a tuple.

I then print out the results as a chart for easier comparison:

	Call Closed-form	Call Finite Diff.	Put Closed-form	Put Finite Diff.
Delta	0.397279	0.533743	-0.602241	-0.465778
Gamma	0.040038	0.039949	0.040038	0.039949
Vega	19.710180	19.710095	19.710180	19.710095
Theta	-24.835212	-25.067507	-18.807900	-18.955581
Rho	7.583586	7.583554	-7.277011	-7.277045

- Binomial Tree

I initialize the option values at expiration for both the call and put options. The function then iterates backwards through the binomial tree, computing the expected value of the option at each time step and checking if early exercise is optimal for each option. If early exercise is optimal, the option value is updated with the exercise value, otherwise, it is updated with the expected value. After iterating through the entire binomial tree, the code computes the option values with and without the dividend by adjusting the final option value with the dividend amount and discounting it to the present value using the risk-free rate.

The results are:

Call with dividend: 3.78
Put with dividend: 4.15
Call without dividend: 4.66
Put without dividend: 3.27

I then write two functions to calculate the prices and Greeks of options using a binomial tree model.

The first function, binomial_tree, creates a binomial tree to calculate the price of an option with given parameters such as the stock price, strike price, time to expiration, risk-free rate, implied volatility, and dividend information. The function first calculates the up and down factors of the binomial tree based on the volatility and time step size. Then, it creates the stock price tree and adjusts the prices for dividends if specified. The option price tree is initialized with zeros and the function calculates the option's value at maturity by taking the maximum of the intrinsic value and zero for both call and put options. Finally, the function steps back through the tree to calculate the option's value at each node, considering the option's value if exercised early for American style options.

The second function, binomial_tree_greeks, uses the binomial_tree function to calculate the prices of the option with specified parameter shifts for each of the Greeks: delta, gamma, theta, vega, and rho. The function uses finite difference methods to calculate the Greeks, by calculating the option prices with shifted stock price, volatility, and risk-free rate values. The delta is the sensitivity of the option price to the stock price, gamma is the sensitivity of delta to the stock price, theta is the sensitivity of the option price to time, vega is the sensitivity of the option price to volatility, and rho is the sensitivity of the option price to the risk-free rate.

The Greeks are:

Call Option Greeks:		
	Call without div	Call with div
Delta	0.536898	0.500120
Gamma	0.038292	0.045113
Theta	-25.932238	-25.884008
Vega	19.657366	19.821456
Rho	7.550520	7.066599
Put Option Greeks:		
	Put without div	Put with div
Delta	-0.471247	-0.508301
Gamma	0.039587	0.046210
Theta	-19.577328	-19.518492
Vega	19.621387	19.702896
Rho	-5.868226	-6.173442

- Sensitivity

Lastly, I calculate the sensitivity of a European call and put option to a change in dividend amount. The delta of an option is the change in the option price divided by the change in the underlying asset price. In this case, the underlying asset is the dividend amount. Therefore, the delta of the call and put options are calculated by subtracting the option price without the dividend from the option price with the dividend and dividing by the dividend amount. These deltas represent the sensitivity of the option prices to a change in the dividend amount.

The result is:

Delta Call with Dividend: -0.5339781069654976
Delta Put with Dividend: 0.4654847770590109

Problem 2

For this problem, I firstly extract the daily closing prices of AAPL and computes the daily returns based on the closing prices. Then I fit a normal distribution to the AAPL returns using the mean of zero and the standard deviation of the returns.

Next, I simulate 10 days' worth of AAPL returns by drawing random samples from the previously fitted normal distribution. The number of simulations and days can be changed by modifying the n_simulations and n_days variables, respectively. The simulated returns are then used to calculate the simulated prices for 10 days ahead based on the initial price of 151.03.

Finally, I get the 10th day's simulated prices and stores them in a list called simulation_price. These simulated prices can be used to estimate the probability distribution of future stock prices and to make informed investment decisions.

I then use black_scholes, implied_volatility, portfolio_value functions that I defined before and iterate through each portfolio and calculate the Mean, VaR, and ES:

	Mean	**VaR**	**ES**
Straddle	2.982048	0.009344	0.018604
SynLong	0.710028	17.702116	21.874643
CallSpread	0.240339	3.525782	3.91628
PutSpread	0.258879	2.712724	2.857014
Stock	0.886034	17.001801	20.93669
Call	1.846038	5.590132	6.054338
Put	1.13601	4.426767	4.633052
CoveredCall	-0.924905	13.397191	17.13173
ProtectedPut	1.951894	7.31635	7.836915

Lastly, I calculate the Delta-Normal VaR (Value at Risk) and ES (Expected Shortfall) for each portfolio in a dataset. I first define a function called "delta" which calculates the delta value for a given stock or option based on its parameters. Then I define another function called "delta_normal_var_pnl" which takes in a portfolio, the initial stock price, risk-free rate, dividend yield, current date, and a matrix of simulated prices. This function calculates the portfolio delta based on the holdings of stocks and options in the portfolio, and then uses the delta values and simulated prices to calculate the P&L (profit and loss) of the portfolio under each simulation. I then calculate the Delta-Normal VaR as the 5th percentile of the P&L distribution, and the ES as the mean of the P&L values that fall below the VaR.

Finally, the code loops through each portfolio in the dataset, calls the "delta_normal_var_pnl" function, and storeS the results in a DataFrame called "results_dn_df". The VaR and ES values are stored in the rows "VaR" and "ES" respectively, and each portfolio's results are stored in the corresponding column of the DataFrame:

	VaR	**ES**
Straddle	2.112789	2.601771
SynLong	17.022811	20.962562
CallSpread	2.614241	3.21928
PutSpread	2.87778	3.718832
Stock	17.001801	20.93669
Call	9.5678	11.782167
Put	8.309372	10.737845
CoveredCall	8.687934	10.698666
ProtectedPut	10.956413	13.49216

This week’s result is a little higher than last week’s. The simulated VaR and ES are smaller in the normal distribution than in the AR(1) model probably because the normal distribution assumes that returns are independently and identically distributed, which may not be the case in real financial markets. In contrast, the AR(1) model incorporates a degree of serial correlation in returns, which reflects the fact that returns tend to be correlated over time.

Problem 3

This section is aiming to construct an optimized stock portfolio based on the Fama-French 4-factor model and Carhart's momentum factor. I first load historical stock prices and factor data, calculates the excess returns, and estimates the expected returns using the Fama-French 4-factor model and Carhart's momentum factor. The excess returns of the stocks are calculated by subtracting the risk-free rate from the stock returns. Next, a linear regression is performed for each stock using the factors as explanatory variables. The regression coefficients (betas) represent the sensitivity of the stock returns to each factor.

The expected returns of each stock are then calculated using these betas and the factor returns.

AAPL	-0.293824
META	-0.675949
UNH	0.141270
MA	-0.078367
MSFT	-0.225500
NVDA	-0.577209
HD	-0.093769
PFE	0.060310
AMZN	-0.699768
BRK-B	-0.019689
PG	-0.016499
XOM	0.412650
TSLA	-0.983864
JPM	-0.117920
V	-0.085881
DIS	-0.629154
GOOGL	-0.489423
JNJ	0.103228
BAC	-0.383005
CSCO	-0.096899

Then, the code calculates the covariance matrix of the stock returns, and using these expected returns and covariance matrix, it finds the optimal portfolio weights by minimizing the negative Sharpe ratio.

	Weight
AAPL	0.0000
META	0.0000
UNH	0.0000
MA	0.0000
MSFT	0.0000
NVDA	0.0000
HD	0.0000
PFE	0.0000
AMZN	0.0000
BRK-B	0.0000
PG	0.0000
XOM	0.6804
TSLA	0.0000
JPM	0.0000
V	0.0000
DIS	0.0000
GOOGL	0.0000
JNJ	0.3196
BAC	0.0000
CSCO	0.0000

The super-efficient portfolio consists of 68.04% XOM and 31.96% of JNJ, and the corresponding Sharpe-ratio is 1.07.