Background

In this project, we will create functions that allow us to simulate future prices of a stock based on properties of that stock. We will use these simulations to answer questions about the probability that the stock's price will be in a certain rate at a certain time. Before jumping into the code, we will provide a brief bit of mathematical background for the models that we will be using.

A **yield rate** for a stock is (roughly speaking) the percentage increase or decrease in the value of the stock over a certain period. Suppose that the price of a stock today is 100 and the stock has a yield rate of 12% over the next year. Then the price of the stock at the end of the year will be $100 \cdot e^{0.12} = 112.75$.

Let S_0 denote the current price of a stock. Suppose that we randomly generate simulated daily yield rates for the stock over the next n days. Denote these simulated rates as $R_1, R_2, \ldots, R_n$. Let S_t denote the price of the stock at the end of day t. Then simulate value of S_t is given by $S_t = S_0 e^{R_1} e^{R_2} \cdot \ldots \cdot e^{R_t}$, or $S_t = S_0 e^{R_1 + R_2 + \cdots + R_t}$. For convenience, define the cumulative yield rate on day t to be $CR_t = R_1 + R_2 + \cdots + R_t$. Then we can write the formula for the simulated price on day t as $S_t = S_0 e^{CR_t}$.

Let's explain these concepts with an example:

- Assume that the current price of the stock is 120.
- Suppose that the simulated daily yields over the next 5 days (R_1 through R_5) are given by:

```
daily_yields = [0.02, 0.01, -0.04, 0.03, 0.05]
```

- Then the cumulative daily yields (CR_1 through CR_5) are given by:

```
cumulative_yields = [0.02, 0.03, -0.01, 0.02, 0.07]
```

- The simulated daily prices would be given by:

```
simulated_prices = [120e0.02, 120e0.03, 120e-0.01, 120e0.02, 120e0.07]
```

Part A: Stock Simulation Function

In this section, you will create and test a function to generate sequences of simulated daily stock prices, or runs.

CODE CELL 1

Write a function named **simulate_stock**. This function will randomly generate a simulated sequence of daily stock prices (which we will call a run) based on several parameters.

The parameters for **simulate_stock** are **start, rate, vol,** and **days**:

- **start** will represent the current price of the stock. This will be the starting price for the run.
- **rate** will be the expected annual yield rate for the stock.
- **vol** will be the annual volatility of the stock. This is a measure of how much uncertainty there is in the future price of the stock. Stock with a higher volatility will have prices that are a lot "swingier". In statistical terms, the volatility is the standard deviation of the stock's annual yield.
- **days** will be the number of days into the future that we would like to simulate prices.

The function should return an array that contains **days + 1** elements. The first element of the returned array should be the starting price, **start**, while the later elements should be simulated stock prices. For example, if **days = 4**, then the function should return an array with 5 values: the starting price and 4 simulated prices.

To calculate the array of simulated prices, we will assume that the daily yield rates follow a normal distribution with a mean determined by the parameter **rate**, and a standard deviation determined by the parameter **vol**.

1. Use **np.random.normal** to create an array of randomly generated daily yields. Since the parameters **rate** and **vol** relate to the annual yield, we will have to scale them down to work with daily yields. The stock market has around 252 trading days in a given year, so to scale down to daily yields, we will need to divide **rate** by 252252 and divide **vol** by 252252 (i.e. `vol/(252**0.5)`) (these facts would be covered in a probability course). Use these modified values in the parameters **loc** (mean) and **scale** (standard deviation). The number of elements in this array should be equal to **days**. I suggest naming the array **daily_yields**.
2. To calculate the simulated stock price at the end of each day, we need to know the cumulative yields. Use **np.cumsum** to calculate this. I suggest naming the results **cumulative_yields**.
3. Create an array called **daily_multipliers** by exponentiating each of the cumulative yields. You can accomplish this using **np.exp**.
4. Multiply the daily multipliers by the starting price to get the simulated daily prices for each day. Round these simulated prices to 2 decimal places.
5. The number of elements in the array that you have created should be equal to days. We wish to add the starting price to the beginning of this array. You can accomplish this using **np.concatenate.**
6. Return the array of **days + 1** elements created in Step 5.

Note: This function SHOULD NOT use any loops. Use numpy arrays instead.

EXPECTED OUTPUT: None

CODE CELL 2

Test your code by performing the instructions below:

- Simulate 60 daily prices for a stock with a current price of 500, an expected annual return of 8%, and an annual volatility of 0.3. Store the result in a variable.
- Display the prices using a line plot.
- Run this cell a few times to get a sense as to how the simulated results might vary.
- At the top of this cell, use numpy to set a random seed of 1. Run the cell again. If your function was written correctly, your simulated run should have a final price of 514.44.

EXPECTED OUTPUT: Plot

Part B: Annual Yield Function

In this section, we will create and test a function that takes a simulated run as its input, and calculates the annual yield during for that particular run.

CODE CELL 1

Write a function named **find_yield**. This function should accept a single parameter called **run**, which is expected to be an array of simulated daily prices for a stock. The function should return the annual yield for the stock over the simulated period, rounded to four decimal places.

The formula for calculating the annual yield is as follows: $\text{annual yield} = \ln\left(\frac{\text{Final Price}}{\text{Initial Price}}\right) * \frac{252}{\text{Days in Run}}$. You can use the function **math.log** in the math package to calculate the natural logarithm.

EXPECTED OUTPUT: None

Recall that the number of days in a run is equal to one less than the length of the run since the run contains the starting price.

CODE CELL 2

Call the **find_yield** function on the run you created in Part A, printing the result. If everything is correct, you should get 0.1196 as the result. Note that this is quite a bit different from the expected 8% annual yield that we used to simulate this run. This is a result of the randomness involved in our simulation.

EXPECTED OUTPUT:

```
0.1196
```

CODE CELL 3

In a new cell, use a loop to create 25 simulated runs for a stock with a current price of 100, an expected annual yield of 6%, and an annual volatility of 0.4 over a period of 200 days. Plot all 25 runs in the same line plot. For each run, use **find_yield** to calculate the annual yield for the run, and store the value in a list. You should see a wide range of simulated results.

Hint: Each iteration of your loop should do three things: Simulate a run, calculate the yield for that run and add the result to a list, and add a line plot of the run to a figure using **plt.plot**.

EXPECTED OUTPUT: Plot

CODE CELL 4

Print the list of annual yields created in the previous code cell. Again, you will likely see a wide range of results.

EXPECTED OUTPUT:

```
Annual yield of run # 0: x.xxxx
Annual yield of run # 1: x.xxxx
Annual yield of run # 2: x.xxxx
...
Annual yield of run # 24: x.xxxx
```

Part C: Finding Seeds that Generate Specific Outcomes

The purpose of this section is to give you some experience working with seeds. Through trial-and-error, you will try to find seeds that result in specific outcomes.

CODE CELL 1

Create a new cell. Use numpy to set the random seed to an integer of your choice. Simulate daily runs for three stocks (Stock A, Stock B, and Stock C) over a period of 100 days. The parameters for these stocks are as follows:

- Stock A has a current price of 78, an expected annual return of 4%, and a volatility of 1.2.
- Stock B has a current price of 75, an expected annual return of 8%, and a volatility of 0.8.
- Stock C has a current price of 72, an expected annual return of 16%, and a volatility of 0.6.

Create line plots for all three runs on the same figure, including a legend indicating which line goes with which stock. Change the seed until you get a result in which Stock A has highest final price after 100 days.

EXPECTED OUTPUT: Plot

CODE CELL 2

Repeat the steps in the previous code cell, but select a seed that results in Stock B having the highest final price after 100 days.

EXPECTED OUTPUT: Plot

CODE CELL 3

Repeat the steps in the previous code cell, but select a seed that results in Stock C having the highest final price after 100 days.

EXPECTED OUTPUT: Plot

Part D: Monte Carlo Simulation

As we have seen, any two runs generated from the same set of parameters might vary considerably from one-another. You might be asking how simulation is useful if I get very different results every time we run a simulation. The answer is that we do not expect simulation to be able to tell us exactly what will occur, we use it to get an idea of the range of possible outcomes that might occur. In order to perform that sort of analysis, we need to perform many simulations, and then look at the range of outcomes occurring in this simulations. The process of performing several simulations to estimate probabilities relating to the outcome of a certain event is called **Monte Carlo Simulation**.

CODE CELL 1

Write a function named **monte_carlo**. The function should accept five parameters: **start, rate, vol, days,** and **num_runs**. The function should use a loop to generate a number of simulated stock runs equal to **num_runs**. The characteristics of the runs are provided by the parameters **start, rate, vol,** and **days**.

Each time you loop executes, the following steps should be performed:

- Simulate a run using the supplied parameters. Store the resulting array in a variable.
- Determine the final simulated price of the stock and append it into a list called **final_prices**.
- Determine the annual yield for the simulated run and append it into a list called **annual_yields**.

When the loop is done executing, convert the two lists you have constructed to numpy arrays and return both of them.

EXPECTED OUTPUT: None

CODE CELL 2

Set a seed of 1, and run a Monte Carlo simulation consisting of 10,000 simulated runs for a stock with a current price of 200, an expected annual return of 10%, and a volatility of 0.4. Each run should be over a period of 500 days. Create a histogram of the final prices. Use `bins=np.arange(0, 1600, 50)`, and set the **edgecolor** to black. Set the size of the figure to be [10,5].

If your code is correct, your histogram should have a peak around 200 and should have a long tail trailing off to the right. This shows that the majority of the simulated final prices are near 200, but there are some very large outliers.

CODE CELL 3

Use np.percentile to calculate the 10th, 25th, 50th, 75th, and 90th percentiles for the final prices in the simulated runs generated for the stock. Display the results by creating five lines of output, with each line using the format below. Round the display percentiles to 2 decimal places.

If done correctly, you should get a 10th percentile of 118.05 and a 90th percentile of 505.91.

EXPECTED OUTPUT:

```
10th percentile: xx.xx
25th percentile: xx.xx
...
90th percentile: xx.xx
```

Part E: Effects of Volatility

In this part, we will explore the effect of volatility on simulated stock prices. We will do this by performing two Monte Carlo simulations. The two simulations will use different volatilities, but will otherwise use the same parameters.

CODE CELL 1

Set a seed of 1 and then run Monte Carlo simulations for two stocks (Stock A and Stock B), each with 10,000 runs lasting over a period of 150 days. Both stocks being simulated have a current price of 100, and an expected annual yield of 12%. However, the Stock A has a volatility of 0.3, and Stock B has a volatility of 0.7.

Calculate the average of the simulated annual yields for each stock, rounded to four decimal places. Print the results in the format below.

EXPECTED OUTPUT:

```
Average Annual Yield for A over 10000 runs: x.xxxx
Average Annual Yield for B over 10000 runs: x.xxxx
```

CODE CELL 2

Use **plt.hist** to create a figure with two histograms on the same axes. Each histogram should display the distribution of final prices for each stock over the 10,000 simulated runs in one of the two Monte Carlo simulations. Set a figure size of [10,5]. Set an alpha level of 0.6 and use **np.arange(0,600, 10)** for the bins in each plot. Set the **edgecolor** to black. Display a legend indicating which histogram is for which stock. Finally, set the title of the figure to be **"Histogram of Final Prices over 10,000 Runs"**.

You should see that the histogram for Stock A is tightly clustered with a peak near 120. The histogram for Stock B should have a shorter peak near 80, and a large tail trailing off to the right. This shows that the stock with the higher volatility (Stock B) has a much wider range of likely outcomes.

EXPECTEDD OUTPUT: Plot

Part F: Comparing Two Stocks

In this section, we will use Monte Carlo simulation to estimate probabilities relating to the performance of two stocks with different parameters.

CODE CELL 1

Set a seed of 1, and then run Monte Carlo simulations for two stocks (Stock A and Stock B), each with 10,000 runs lasting over a period of 252 days. Both stocks being simulated have a current price of 120. Stock A has an expected annual yield of 8% and a volatility of 0.2. Stock B has an expected annual yield of 5% and a volatility of 0.5.

Calculate the following:

- The proportion of the simulated runs in which Stock A has a higher final price than Stock B.
- The proportion of the simulated runs in which Stock A has a final price greater than 150.
- The proportion of the simulated runs in which Stock B has a final price greater than 150.
- The proportion of the simulated runs in which Stock A has a final price less than 100.
- The proportion of the simulated runs in which Stock B has a final price less than 100.

Round all values to four decimal places, and display your results in the format below.

EXPECTED OUTPUT:

```
Proportions of runs in which...
```

```
-------------------------------
A ends above B:   xxxx

A ends above 150: xxxx
B ends above 150: xxxx

A ends below 100: xxxx
B ends below 100: xxxx
```

Part G: Expected Call Payoff

We will conclude this project by exploring an application of Monte Carlo simulation. A **call option** is a particular type of investment whose final value (or payoff) is based on the price of a stock. When you purchase a call based on a stock, it will have a specified expiration date, as well as a strike price that we will denote by K. Let S denote the price of the stock when the call expires. If $S>K$, then you will receive a payoff of $S-K$ dollars from your call. If $S\leq K$, then your payoff would be $0.

For example, assume that you pay $10 for a call on a stock. Suppose that the call expires in one year and has a strike price of $120. If the price of the stock one year later is $150, then you will receive a payoff $30 from the call. If, on the other hand, the price of the stock after one year was $110, then you would not get any money back from the call (your payoff would be $0).

In a sense, you can think of a call option as a "bet" that the price of the stock will be greater than the strike price when the call expires.

We will use Monte Carlo to estimate the expected payoff of a call on a particular stock. The stock will have a current price of 200, an expected annual yield of 11%, and a volatility of 0.4. The call will have a strike price of 225 and will expire in 150 days.

CODE CELL 1

Consider a call with a strike price of 225. Calculate the payoff of this call for each of the 10,000 simulated runs of the stock. You can do this by first subtracting 225 from each of the final stock prices, and then setting any negative values to zero. To set the negative values to zero, you can use Boolean masking or **np.where**.

Print the average call payoff over the 10,000 runs. Set a seed of 1 at the beginning of this cell.

This section should not contain any loops.

EXPECTED OUTPUT:

```
xxxx
```