FinTech 545 Project Week 04: Leslie Dees

Problem 1:

Calculate and compare the expected value and standard deviation of price at time t(P_t), given each of the 3 types of price returns, assuming r_t ~ N(0, sigma^2). Simulate each return equation using r_t~N(0, σ^2) and show the mean and standard deviation match your expectations.

I calculate the expected values and standard deviations of the prices at time *t(P_t)* for analysis purposes for only a single stock. This was due to 2 factors. Firstly, the question was unclear on what stocks I was meant to perform my calculations on. Secondly, by specifying a single stock the visualizations and analysis of the expected value and standard deviations was much more simple.

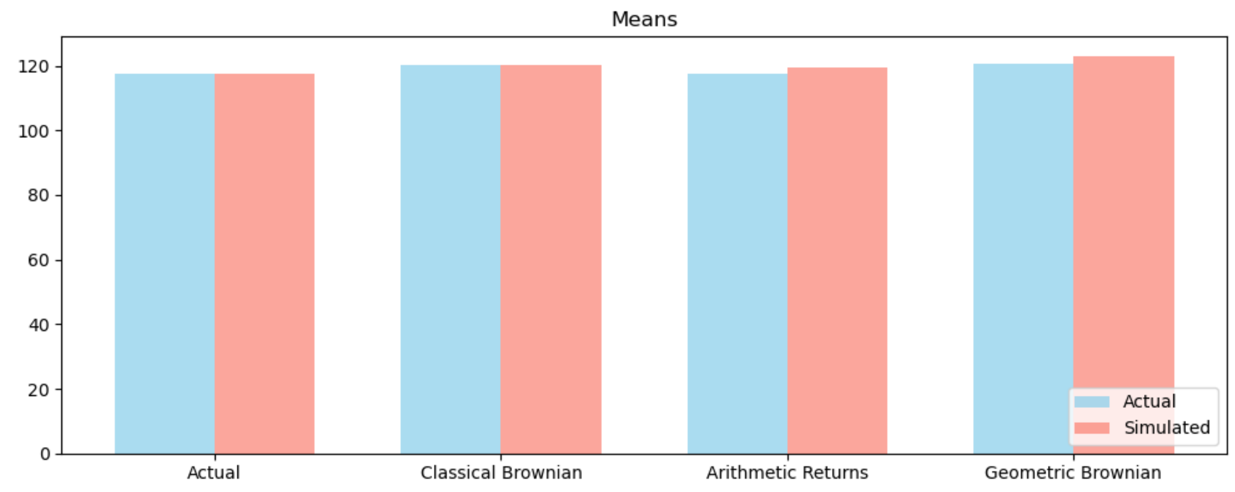
In order to select a stock to perform calculations on, I attempted to find the stock with an average of its returns as close to 0.0 as possible. This was to help satisfy the condition of r_t ~ N(0, sigma^2). From the calculations that I performed I selected PLD with a mean of 0.000031. Then, I found the initial price from the prices array and also the returns for the column of PLD.

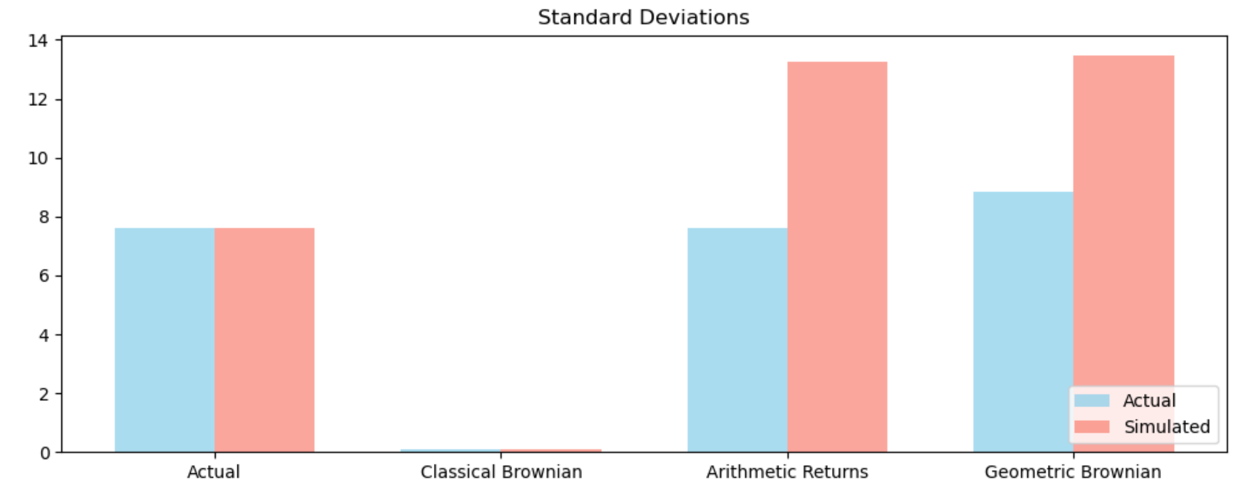
I built a function called calculate_prices which takes in the returns pandas Series, the initial price, and finally the method of price calculation. I obtained from these calculations my original set of price returns and standard deviations. As a comparison, the True Expected Value of PLD is $117.45 and the True Standard deviation was found to be $7.59. Below you can see the full table of obtained values.

Type of Formulation	Expected Value ($)	Standard Deviation ($)
True Value	117.449	7.595
Classical Brownian	120.091	0.076
Arithmetic Return	117.449	7.595
Geometric Brownian	120.693	8.824

From this we can see that in general, the standard deviation for the classical Brownian formulation is drastically smaller than the other methods. We can also see that the arithmetic return has the exact same values as the True values, which tells me that the returns and prices of the csv files were calculated using the arithmetic mean. Finally, the geometric mean has a slightly higher expected value and standard deviation than the others.

After performing this test, I then simulated 1000 trials of simulated return distributions. In order to do this I calculated the standard deviation of the returns for this stock and generated a random normal distribution with 0 mean and std based on the standard deviation that I provided. Then, I performed the exact same calculations of price results as in the true calculation test, however in this case I used the simulated returns. After 1000 trials I took the average of the expected values and standard deviations and plotted them below.





We can see that the simulated results follow the general trends of their actual counterparts, except they tend to have higher volatility. The geometric standard deviation and expected values are higher than all of the rest, while we did see that the arithmetic returns mean was similar to the actual value with a higher standard deviation.

From this test, I do generally verify my expectations. I see that the general trends are followed, however I cannot explain the drastic increases in simulated standard deviations. I expected that it is due to the randomness of WHEN the returns are applied resulting in skews away from the actual trends.

Problem 2:

Implement a function similar to the "return_calculate()" in this week's code. Allow the user to specify the method of return calculation. Use DailyPrices.csv. Calculate the arithmetic returns for all prices. Remove the mean from the series so that the mean(META)=0 Calculate VaR

I recreated the return_calculate() method exactly as it was in Julia. I verified the results by comparing them to the DailyReturn.csv provided to us. I then took out the META column to perform all calculations. While I could have used the entire prices for this, I was unsure of the wording of the question, so I decided to just perform calculations with the META column. I centered it by taking the mean and subtracting from each value, therefore centering the mean at 0.0.

1. Using a normal distribution.

For the normal distribution, I first found the mean and standard deviation of the META column. The mean was simple as it was 0, but for the standard deviation I had to calculate using numpy. I built a function called calc_var_normal that takes in the mean, standard deviation, and sets alpha to 0.05 to return the VaR. This method uses the scipy.stats.norm.ppf to find the value at the given quartile. For this distribution I obtained a VaR of -0.0543, or -5.43%.

2. Using a normal distribution with an Exponentially Weighted variance (λ = 0. 94)

For the normal distribution with an exponentially Weighted variance, I utilized my prebuilt calculate_ewma_covariance_matrix function with a lambda of 0.94. I took the square root of the diagonals of the covariance matrix to obtain the standard deviations. From there, I followed suit and used the calc_var_normal method to obtain a VaR of -0.0300, or -3.00%.

3. Using a MLE fitted T distribution.

For the MLE fitted T distribution I had to recode my prior mle_t_distribution method into one that only took in a y variable, which in this case was the Series of META centered returns. It provided an optimized mean, standard deviation, and degrees of freedom which I then provided into the t.ppf function. I used the t_mean as the loc and the t_std to scale the distribution. Then, I could input the alpha of 0.05 and obtain a VaR of -0.0421, or -4.21%.

4. Using a fitted AR(1) model.

Fitting an AR(1) model once again required the recoding of my prior AR process. I created a method called simulate_ar1_process which performed autoregression with a set of N=1. It iterated for 10000 times to simulate the data. I then obtained the last value from the AR(1) process, and used the np.percentile function to find the final value's quartile of 0.05 as my VaR. This gave me a VaR of -0.0636 or -6.36%.

5. Using a Historic Simulation.

Finally, I attempted to mimic the Julia code provided for my historic simulation. I obtained 86 shares from the portfolio holdings to obtain my portfolio value, and then simulated 10000 draws with replacement to simulate historical values. Then I calculated new prices from the simulated returns to get simulated portfolio values. Finally, I obtained the alpha percentile of the portfolio value for alpha = 0.05, and normalized it to the full portfolio value to get a percentile. This provided me with a VaR of -0.0041, or -0.41%.

Compare the 5 values.

Comparing these values, I can easily see that the historical simulation proved to show the lowest expected Value at Risk of -0.41%. Opposing to this, the AR(1) process gave the highest VaR of -6.36%. What this tells me is that there is a large room of variation in determining the actual Value at Risk, and just using a single function to determine what the true value at risk is inadequate. I would expect that a normal simulation is not ideal, since real data is almost never normal. I also believe that fitting any distribution to the data will never be optimally accurate due to discrepancies between time periods, so MLE fitting t distribution could also be off. I believe that historical simulation is the best way to actually determine the value due to its replication of past events.

Problem 3:

Using Portfolio.csv and DailyPrices.csv. Assume the expected return on all stocks is 0. This file contains the stock holdings of 3 portfolios. You own each of these portfolios. Using an exponentially weighted covariance with lambda = 0.94, calculate the VaR of each portfolio as well as your total VaR (VaR of the total holdings). Express VaR as a $. Discuss your methods and your results. Choose a different model for returns and calculate VaR again. Why did you choose that model? How did the model change affect the results?

I separated the portfolios out into A, B, and C for the following calculations that were done independently for each. I build a method called calculate_portfolio_var that takes in a portfolio, a list of prices, a list of returns, and a lambda value for the exponentially weighted covariance matrix with a set alpha of 0.05. This method performs the following calculations. I calculated the total portfolio value by multiplying the holding amount of each stock by the latest price of the stock, while also creating a list each individual stock delta. I normalized each delta by dividing the stock value by the portfolio value. The exponentially weighed covariance matrix was then obtained and turned into a standard deviation set of std_dev for each stock column.

Then, using the corresponding exponentially weighed standard deviations, I reduced the standard deviations to only contain the stocks present in the portfolio. I then calculated p_sig based on the following formula from the notes: $\sqrt{\nabla R^T \Sigma \nabla R}$. My VaR was finally calculated by multiplying the negative delta by the norm.ppf at my alpha value times the p_sig. The portfolio VaR are listed in the table below for exponentially weighted covariance matrix of lambda = 0.94.

Portfolio	Portfolio Value ($)	Value at Risk ($)
A	1,089,316.16	-47,798.41
B	574,542.41	-23,249.99
C	1,387,409	-78,936.99
All	3,051,268.07	-91,962.09

To follow this up, I decided to perform Value at Risk calculations using an exponentially weighted covariance matrix with a lambda value of 0.80. This was since I wanted to test how the values of lambda results in the expected Value at Risk. The results are listed below.

Portfolio	Portfolio Value ($)	Value at Risk ($)
A	1,089,316.16	-47,947.51
B	574,542.41	-23,174.65
C	1,387,409	-75,319.36
All	3,051,268.07	-90,127.72

What we can see first of all is that both of the total portfolio VaR is less than the sum of each portfolio value. This is describing what was mentioned in class about subadditivity. Secondly, it is notable that when the lambda value decreased, the expected Value at Risk is predicted to be significantly lower than when a higher lambda value was used. I wanted to describe through this experiment the premise that lower values of lambda in an exponentially weighted covariance matrix resulted in results that more heavily weight more recent historical values. While we see that the VaR has shifted down, this shows that recently in history there is a lower relative risk compared to prior in history. However, this doesn't represent the historical volatility of the stocks in the portfolio. Therefore, it perhaps could be better to use a larger lambda value that increases the importance of historical events on today's predicted Value at Risk.