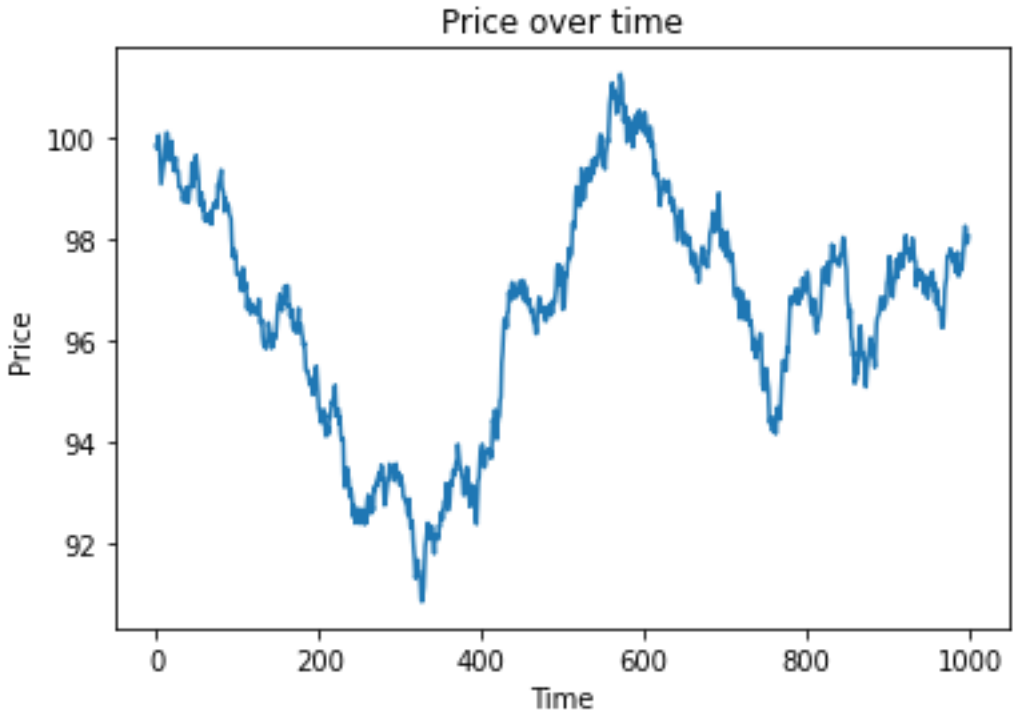
Problem 1

Classical Brownian Motion

- For this question, I set mean of return to be 0, standard deviation of return to be 0.2, number of periods to be 1000, and p0 to be 100. Then I calculate the prices at time t for 1000 times, where pt is p0 plus the last cumulative sum of r. Furthermore, I calculate the expected value and the standard deviation of those 1000 pts.
- Expected value of price at time T: 100.08883058129018
  Standard deviation of price at time T: 6.146024963940399



- For classical Brownian motion, because the mean of r is 0, so the expected value of pt is simply p0 = 100, and the expected standard deviation is sigma times square root of T, where T stands for the number of periods.

```
exp_sd = sigma * np.sqrt(T)
exp_sd
```

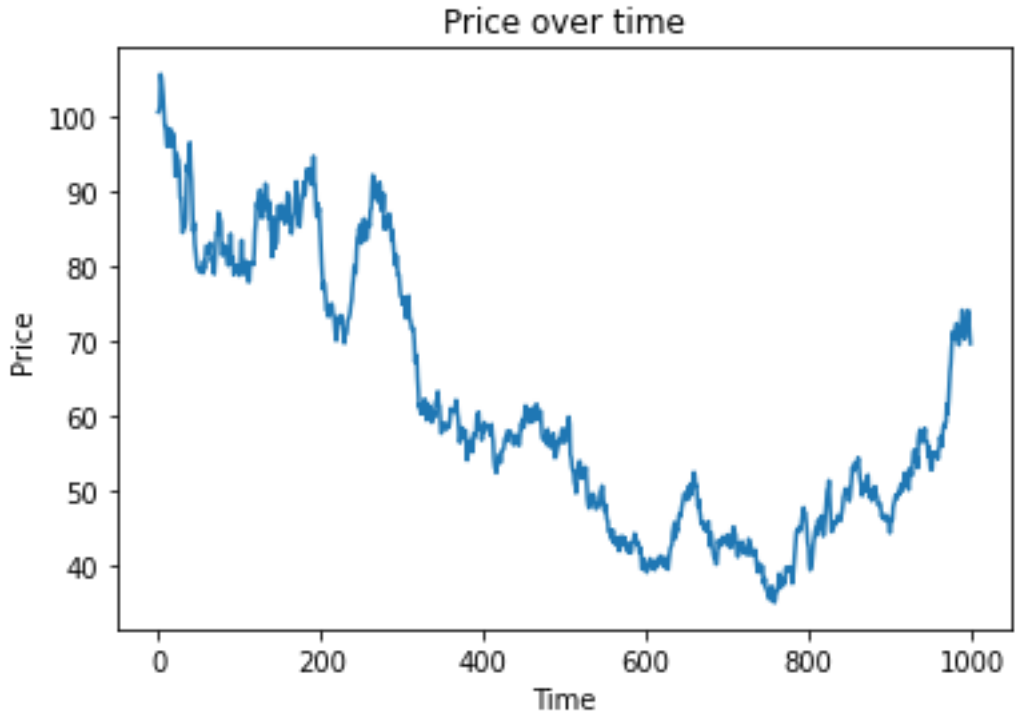
```
6.324555320336759
```

- Here, we can see that both of them match the expected values.

Arithmetic Return System

- For this question, I set mean of return to be 0, standard deviation of return to be 0.02, number of periods to be 1000, and p0 to be 100. Then I calculate the prices at time t for 1000 times, where pt is po times the last cumulative product of r. Furthermore, I calculate the expected value and the standard deviation of those 1000 pts.

- Mean of P: 97.01556522443481
  Standard deviation of P: 66.93666804930811



- For arithmetic return system, the expected value of pt is still po because the expected value of (1+r) is 1, and the expected standard deviation of pt is the mean of the standard deviation of (pt-1*1). Here, The standard deviation of pt is equal to pt-1 * σ, which means the standard deviation of the price at time t is equal to the previous period's price multiplied by the standard deviation of the return σ. However, all those 1000 pts should have a σ of 1.

```
exp_sd  =  np.mean(np.std(p_2))
exp_sd
```

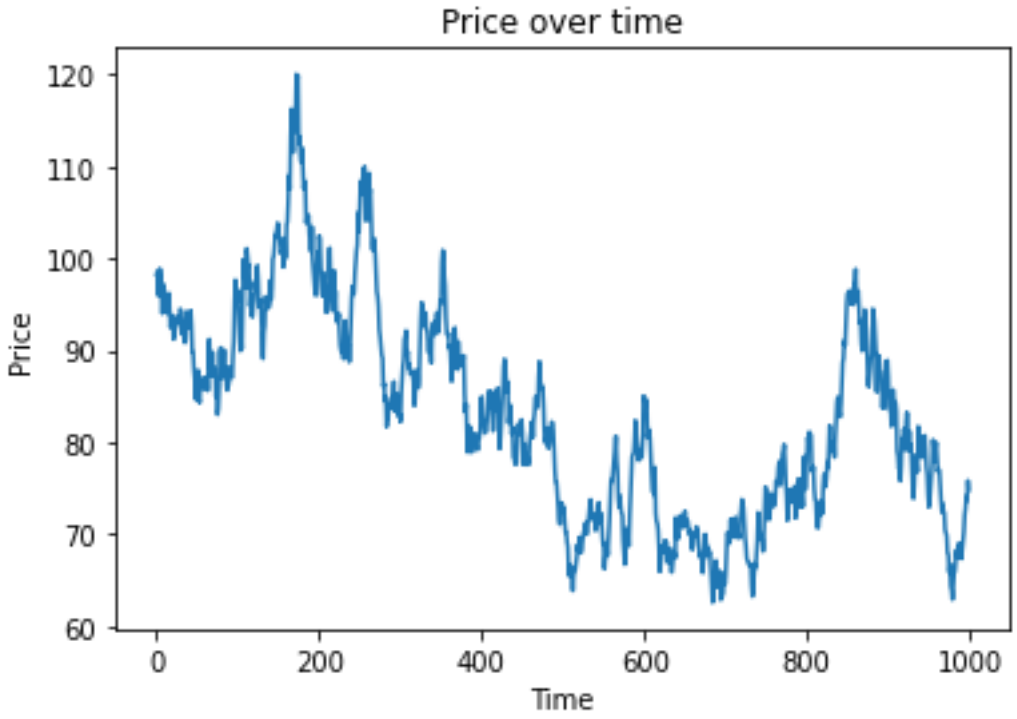
```
66.84226006813446
```

- We could see that both results match the expected values.

Geometric Brownian Motion

- For this question, I set standard deviation of return to be 0.02, number of periods to be 1000, and p0 to be 100. Then I calculate the prices at time t for 1000 times, where pt is po times the exponential of the last cumulative product of r. Furthermore, I calculate the expected value and the standard deviation of those 1000 pts.

- Mean of P: 121.12276053740375
  Standard deviation of P: 81.460525362564



- For geometric Brownian motion, the expected value of pt is p0 times the exponential of (mu*T), where mu is (sigma^2/2). The expected standard deviation is the product of po and the square root of a function related to mu and sigma.

```
mu = (sigma**2/2)
exp_mean = P_0 * np.exp(mu*T)
exp_sd = P_0 * np.sqrt((np.exp(sigma**2 * T) - 1) * np.exp(2 * mu * T))
exp_mean, exp_sd
```

```
(122.14027581601698, 85.65723733877934)
```

- Both results match the expected values.

Problem 2

- To implement calculate_return, I write a function which first retrieves the column names in the data, excluding the date column if it exists. It then checks if the number of variables in the data is the same as the number of variables retrieved. If they are not the same, it raises a ValueError. The function then converts the data to an array and computes the returns for each variable based on the chosen method. If method is set to 'DISCRETE', it subtracts 1 from the ratio of each price with the previous day's price. If method is set to 'LOG', it takes the logarithm of the ratio of each price with the previous day's price. If an invalid method is passed to the function, it raises a ValueError. Finally, the function creates a new dataframe with the daily returns for each variable, where each column represents a variable, and each row represents a date. The date column is included in the new data, and the function returns this new dataframe.
- Using the above function, I could get the arithmetic return of the daily prices:

	Date	SPY	AAPL	MSFT	AMZN	TSLA	GOOGL	GOOG	META	NVDA	...	PNC	MDLZ	MO	ADI	GILD	
1	2/15/2022 0:00	0.016127	0.023152	0.018542	0.008658	0.053291	0.007987	0.008319	0.015158	0.091812	...	0.012807	-0.004082	0.004592	0.052344	0.003600	-0.
2	2/16/2022 0:00	0.001121	-0.001389	-0.001167	0.010159	0.001041	0.008268	0.007784	-0.020181	0.000604	...	0.006757	-0.002429	0.005763	0.038879	0.009294	0.
3	2/17/2022 0:00	-0.021361	-0.021269	-0.029282	-0.021809	-0.050943	-0.037746	-0.037669	-0.040778	-0.075591	...	-0.034949	0.005326	0.015017	-0.046988	-0.009855	0.
4	2/18/2022 0:00	-0.006475	-0.009356	-0.009631	-0.013262	-0.022103	-0.016116	-0.013914	-0.007462	-0.035296	...	-0.000646	-0.000908	0.007203	-0.000436	-0.003916	-0.
5	2/22/2022 0:00	-0.010732	-0.017812	-0.000729	-0.015753	-0.041366	-0.004521	-0.008163	-0.019790	-0.010659	...	0.009494	0.007121	-0.008891	0.003243	-0.001147	-0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
244	2/3/2023 0:00	-0.010629	0.024400	-0.023621	-0.084315	0.009083	-0.027474	-0.032904	-0.011866	-0.028053	...	-0.004694	-0.011251	-0.001277	-0.002677	0.038211	0.
245	2/6/2023 0:00	-0.006111	-0.017929	-0.006116	-0.011703	0.025161	-0.017942	-0.016632	-0.002520	-0.000521	...	-0.014451	0.003945	0.001066	-0.007102	0.022012	0.
246	2/7/2023 0:00	0.013079	0.019245	0.042022	-0.000685	0.010526	0.046064	0.044167	0.029883	0.051401	...	-0.000368	-0.016473	-0.008518	0.019544	-0.003590	-0.
247	2/8/2023 0:00	-0.010935	-0.017653	-0.003102	-0.020174	0.022763	-0.076830	-0.074417	-0.042741	0.001443	...	-0.008469	-0.004456	-0.001289	-0.018009	-0.004416	0.
248	2/9/2023 0:00	-0.008669	-0.006912	-0.011660	-0.018091	0.029957	-0.043876	-0.045400	-0.030039	0.005945	...	-0.016588	-0.007717	-0.003656	0.004275	-0.001634	0.

248 rows × 101 columns

- After preprocessing the META column, I calculate the five VaRs:

```
VaR using normal distribution: -0.06560
VaR using normal distribution with EWM variance: -0.09019
VaR using MLE fitted T distribution: -0.05693
VaR using fitted AR(1) model: -0.06560
VaR using Historic Simulation: 0.05462
```

  - VaR using normal distribution: This method assumes that returns follow a normal distribution and calculates the VaR based on the standard deviation and a given confidence level.
  - VaR using normal distribution with EWM variance: This method is similar to the previous one, but instead of using the sample standard deviation, it uses an exponentially weighted moving variance to account for changes in volatility over time.
  - VaR using MLE fitted T distribution: This method fits a T distribution to the returns data using maximum likelihood estimation and then calculates the VaR based on the fitted distribution.
  - VaR using fitted AR(1) model: This method fits an autoregressive model of

order 1 (AR(1)) to the returns data and then uses the estimated parameters to simulate returns and calculate VaR.

- VaR using Historical Simulation: This method uses historical returns data to simulate possible future returns and then calculates VaR based on the simulated distribution.
- Based on the results, it seems like the VaR calculated using the normal distribution with EWM variance and the VaR using Historical Simulation are significantly different from the other methods. This may be due to the fact that they take into account changes in volatility over time, which the other methods do not. The normal distribution and T distribution assume that volatility is constant over time, and AR(1) only takes the first order.

Problem 3

- Exponentially Weighted Covariance Model
  - I first extract the list of stocks and holdings for each portfolio and the daily prices for each stock in each portfolio. Then, the daily returns for each portfolio are calculated using the log returns, and the covariance matrix for each portfolio is calculated using an exponentially weighted covariance with lambda = 0.94.
  - The total covariance matrix is also calculated using an exponentially weighted covariance with lambda = 0.94. The portfolio values for each portfolio and the total portfolio value are calculated.
  - The VaR for each portfolio and the total VaR are then calculated using the confidence level of 0.95 and the inverse cumulative distribution function of the standard normal distribution (z-score). The VaR is calculated as the product of the z-score and the square root of the product of the portfolio value and the product of the portfolio value and the covariance matrix for that portfolio. Finally, the VaR values are printed out for each portfolio and the total VaR.
- Result
  - Portfolio A VaR: $5691.545367913773
  - Portfolio B VaR: $4531.820720018284
  - Portfolio C VaR: $3837.7173331134054
  - Portfolio total VaR: $13704.717058155928
- AR(1) Model
  - This time the covariance matrix for each portfolio and for the total portfolio is calculated using the AR(1) model.
  - I want to use this model to take a look at the general time-series trend of the prices instead of focusing on the recent prices.
- Result
  - Portfolio A VaR: $259.31091393274534
  - Portfolio B VaR: $220.6713737467063
  - Portfolio C VaR: $172.94982943584452
  - Portfolio total VaR: $652.9321171152961
- Comparison
  - The difference in VaR estimates between the two models can arise due to the different ways in which they estimate the covariance matrix. The AR(1) model may capture certain patterns in the data that are not captured by the exponentially weighted covariance model. Additionally, the exponentially weighted covariance model gives more weight to recent returns, which can result in higher VaR estimates if recent returns have been more volatile.