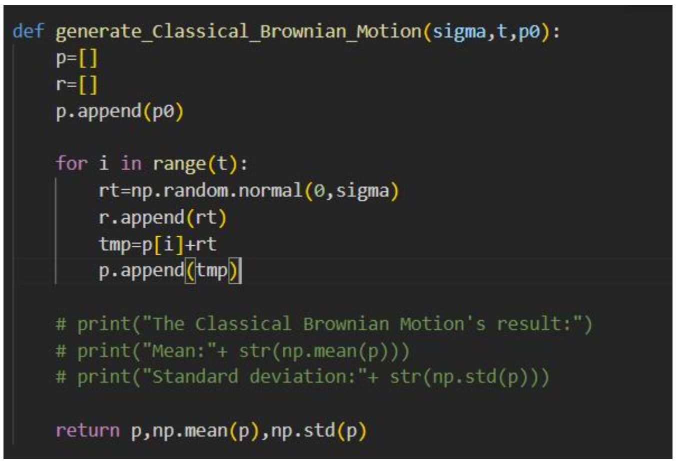
Problem 1

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 $r_t \sim N(0, \sigma^2)$. Simulate each return equation using $r_t \sim N(0, \sigma^2)$ and show the mean and standard deviation match your expectations.

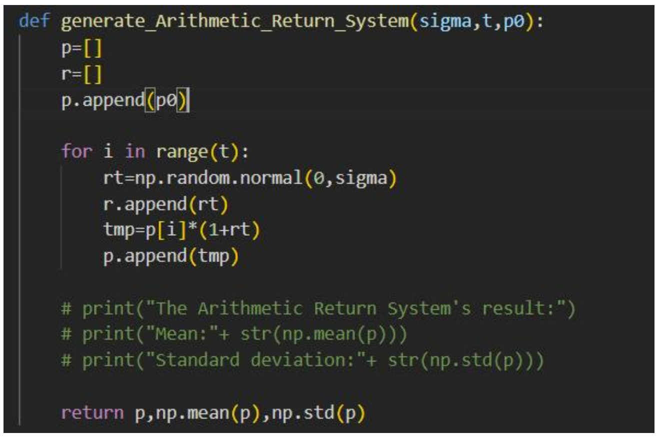
Answer:

①Classical_Brownian_Motion



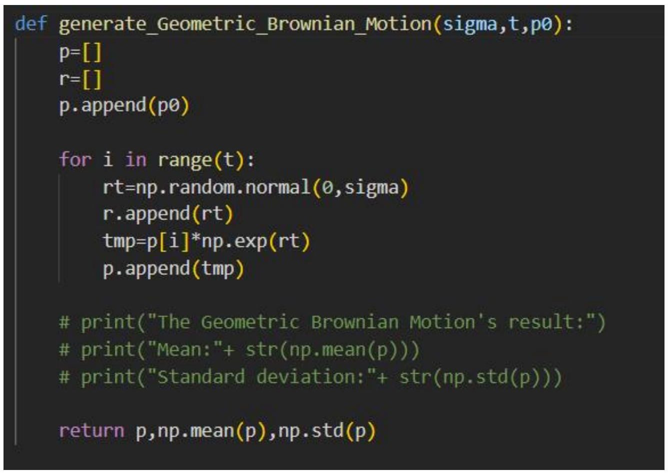
```python
def generate_Classical_Brownian_Motion(sigma,t,p0):
    p=[]
    r=[]
    p.append(p0)

    for i in range(t):
        rt=np.random.normal(0,sigma)
        r.append(rt)
        tmp=p[i]+rt
        p.append(tmp)

    # print("The Classical Brownian Motion's result:")
    # print("Mean:"+ str(np.mean(p)))
    # print("Standard deviation:"+ str(np.std(p)))

    return p,np.mean(p),np.std(p)
```

②Arithmetic_Return_System



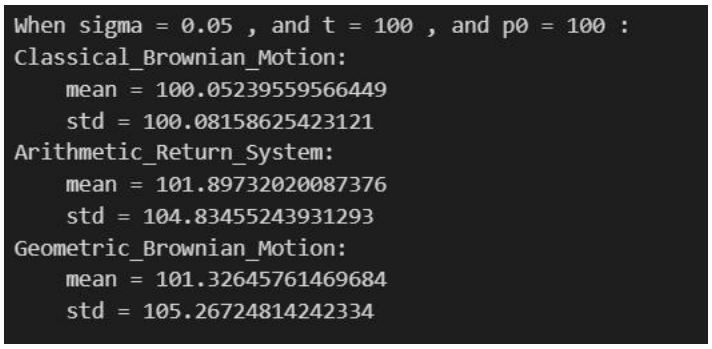
```python
def generate_Arithmetic_Return_System(sigma,t,p0):
    p=[]
    r=[]
    p.append(p0)

    for i in range(t):
        rt=np.random.normal(0,sigma)
        r.append(rt)
        tmp=p[i]*(1+rt)
        p.append(tmp)

    # print("The Arithmetic Return System's result:")
    # print("Mean:"+ str(np.mean(p)))
    # print("Standard deviation:"+ str(np.std(p)))

    return p,np.mean(p),np.std(p)
```

③Geometric_Brownian_Motion



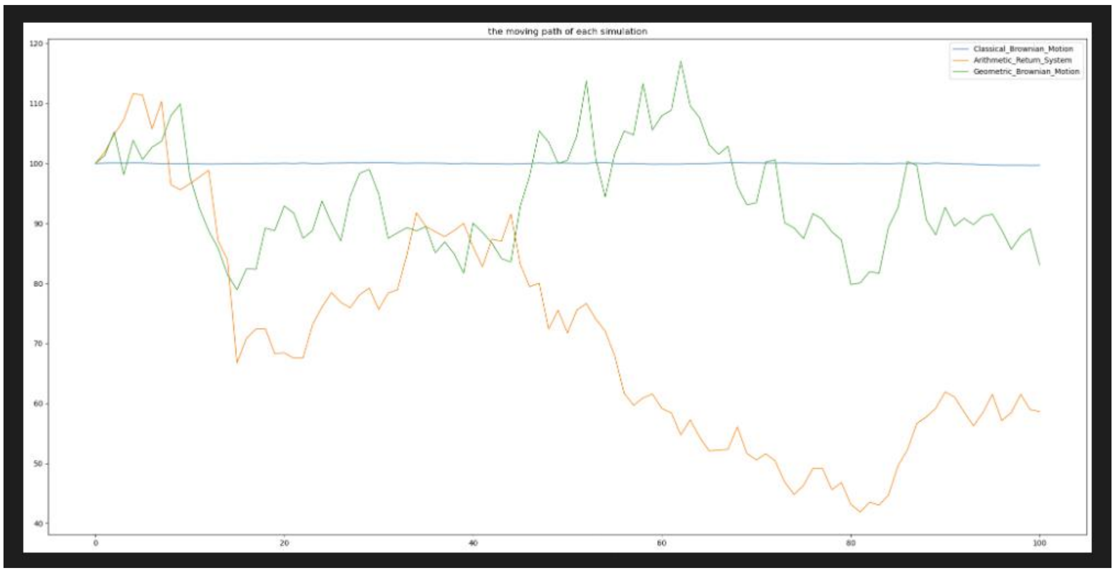
```python
def generate_Geometric_Brownian_Motion(sigma,t,p0):
    p=[]
    r=[]
    p.append(p0)

    for i in range(t):
        rt=np.random.normal(0,sigma)
        r.append(rt)
        tmp=p[i]*np.exp(rt)
        p.append(tmp)

    # print("The Geometric Brownian Motion's result:")
    # print("Mean:"+ str(np.mean(p)))
    # print("Standard deviation:"+ str(np.std(p)))

    return p,np.mean(p),np.std(p)
```

④ The result of running 1 time:

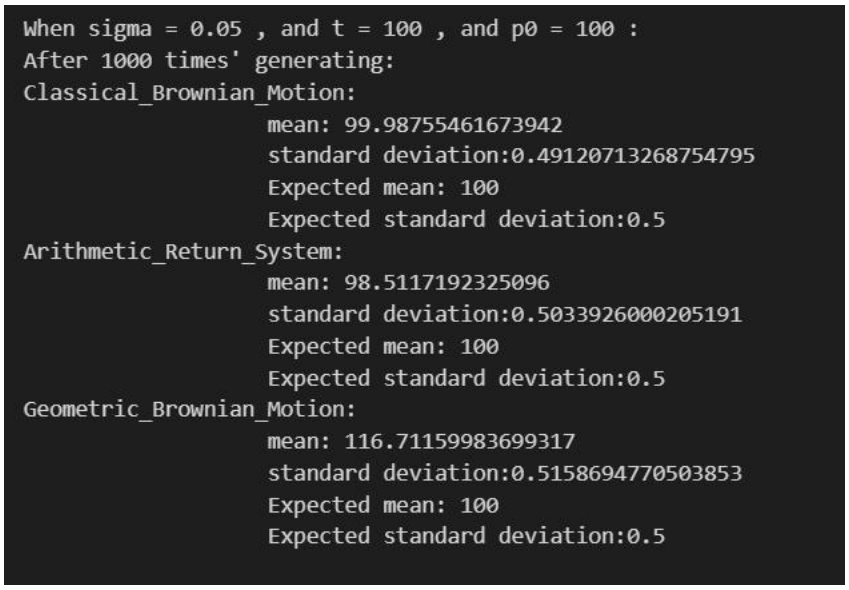


```
When sigma = 0.05 , and t = 100 , and p0 = 100 :
Classical_Brownian_Motion:
    mean = 100.05239559566449
    std = 100.08158625423121
Arithmetic_Return_System:
    mean = 101.89732020087376
    std = 104.83455243931293
Geometric_Brownian_Motion:
    mean = 101.32645761469684
    std = 105.26724814242334
```

the moving path:



⑤ The result of running 1000 times:



```
When sigma = 0.05 , and t = 100 , and p0 = 100 :
After 1000 times' generating:
Classical_Brownian_Motion:
                   mean: 99.98755461673942
                   standard deviation:0.49120713268754795
                   Expected mean: 100
                   Expected standard deviation:0.5
Arithmetic_Return_System:
                   mean: 98.5117192325096
                   standard deviation:0.5033926000205191
                   Expected mean: 100
                   Expected standard deviation:0.5
Geometric_Brownian_Motion:
                   mean: 116.71159983699317
                   standard deviation:0.5158694770503853
                   Expected mean: 100
                   Expected standard deviation:0.5
```

From the result of running 1000 times, we found out that:

*Classical_Brownian_Motion:

the mean of the last price $\approx$ P0

the standard deviation of the last price $\approx$ $\sqrt{t}$ *sigma

*Arithmetic_Return_System:

the mean of the last price ≈ P0

The standard deviation of the log (last price) ≈ $\sqrt{t}$ *sigma

*Geometric_Brownian_Motion:

the mean of the last price ≈ P0

The standard deviation of the log (last price) ≈ $\sqrt{t}$ *sigma

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

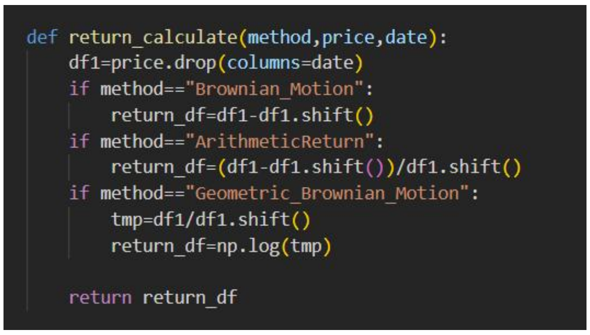
Calculate VaR

1. Using a normal distribution.
2. Using a normal distribution with an Exponentially Weighted variance ($\lambda = 0.94$)
3. Using a MLE fitted T distribution.
4. Using a fitted AR(1) model.
5. Using a Historic Simulation.

Compare the 5 values.

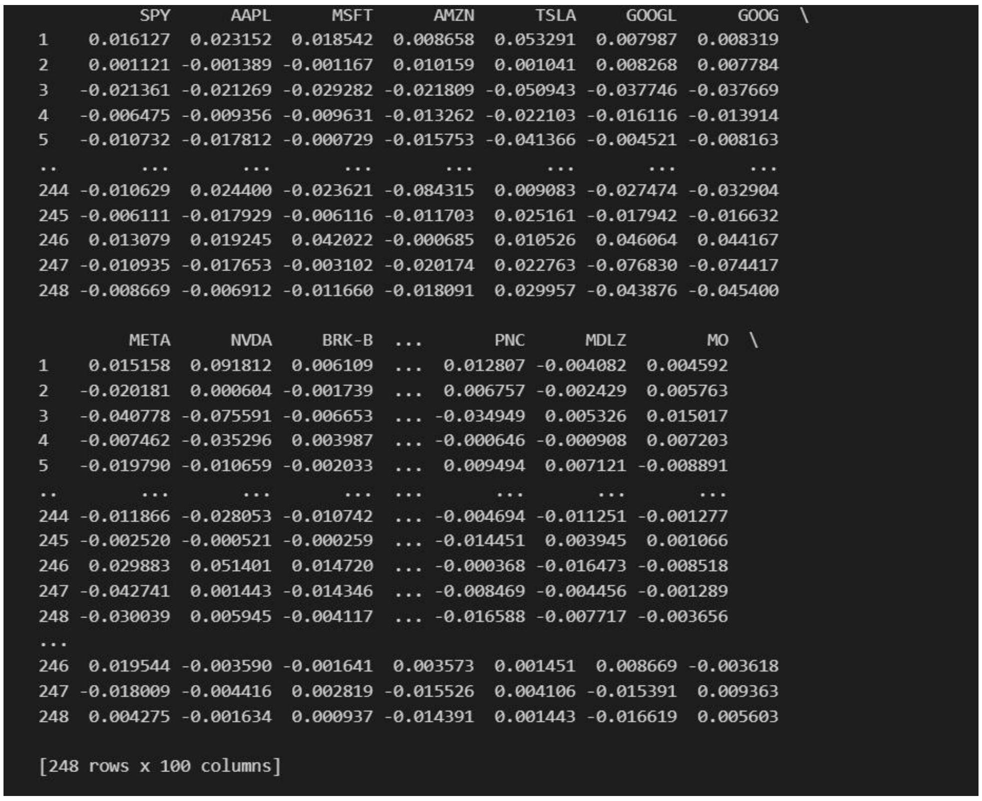
Answer:

①return_calculate():



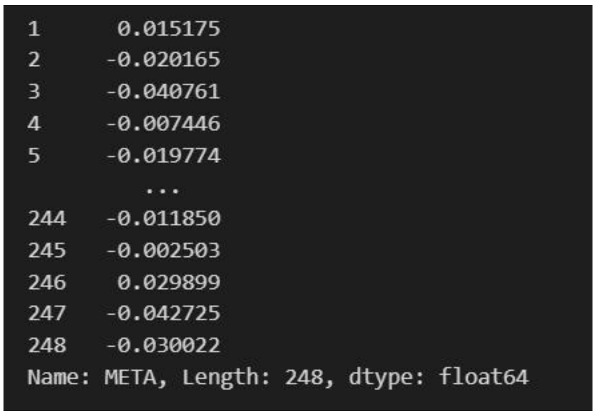
```python
def return_calculate(method,price,date):
    df1=price.drop(columns=date)
    if method=="Brownian_Motion":
        return_df=df1-df1.shift()
    if method=="ArithmeticReturn":
        return_df=(df1-df1.shift())/df1.shift()
    if method=="Geometric_Brownian_Motion":
        tmp=df1/df1.shift()
        return_df=np.log(tmp)

    return return_df
```

② arithmetic returns of all prices:



```
          SPY      AAPL      MSFT      AMZN      TSLA     GOOGL      GOOG  \
1    0.016127  0.023152  0.018542  0.008658  0.053291  0.007987  0.008319
2    0.001121 -0.001389 -0.001167  0.010159  0.001041  0.008268  0.007784
3   -0.021361 -0.021269 -0.029282 -0.021809 -0.050943 -0.037746 -0.037669
4   -0.006475 -0.009356 -0.009631 -0.013262 -0.022103 -0.016116 -0.013914
5   -0.010732 -0.017812 -0.000729 -0.015753 -0.041366 -0.004521 -0.008163
..        ...       ...       ...       ...       ...       ...       ...
244 -0.010629  0.024400 -0.023621 -0.084315  0.009083 -0.027474 -0.032904
245 -0.006111 -0.017929 -0.006116 -0.011703  0.025161 -0.017942 -0.016632
246  0.013079  0.019245  0.042022 -0.000685  0.010526  0.046064  0.044167
247 -0.010935 -0.017653 -0.003102 -0.020174  0.022763 -0.076830 -0.074417
248 -0.008669 -0.006912 -0.011660 -0.018091  0.029957 -0.043876 -0.045400

         META      NVDA     BRK-B  ...       PNC      MDLZ        MO  \
1    0.015158  0.091812  0.006109  ...  0.012807 -0.004082  0.004592
2   -0.020181  0.000604 -0.001739  ...  0.006757 -0.002429  0.005763
3   -0.040778 -0.075591 -0.006653  ... -0.034949  0.005326  0.015017
4   -0.007462 -0.035296  0.003987  ... -0.000646 -0.000908  0.007203
5   -0.019790 -0.010659 -0.002033  ...  0.009494  0.007121 -0.008891
..        ...       ...       ...  ...       ...       ...       ...
244 -0.011866 -0.028053 -0.010742  ... -0.004694 -0.011251 -0.001277
245 -0.002520 -0.000521 -0.000259  ... -0.014451  0.003945  0.001066
246  0.029883  0.051401  0.014720  ... -0.000368 -0.016473 -0.008518
247 -0.042741  0.001443 -0.014346  ... -0.008469 -0.004456 -0.001289
248 -0.030039  0.005945 -0.004117  ... -0.016588 -0.007717 -0.003656
...
246  0.019544 -0.003590 -0.001641  0.003573  0.001451  0.008669 -0.003618
247 -0.018009 -0.004416  0.002819 -0.015526  0.004106 -0.015391  0.009363
248  0.004275 -0.001634  0.000937 -0.014391  0.001443 -0.016619  0.005603

[248 rows x 100 columns]
```

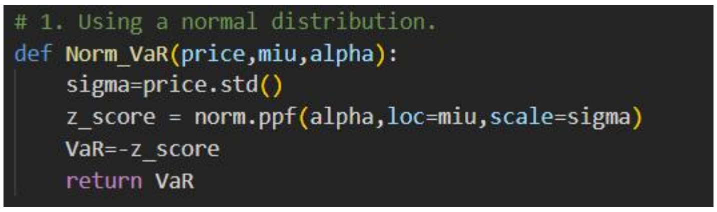
③ remove mean of “META”



```
1      0.015175
2     -0.020165
3     -0.040761
4     -0.007446
5     -0.019774
         ...
244   -0.011850
245   -0.002503
246    0.029899
247   -0.042725
248   -0.030022
Name: META, Length: 248, dtype: float64
```

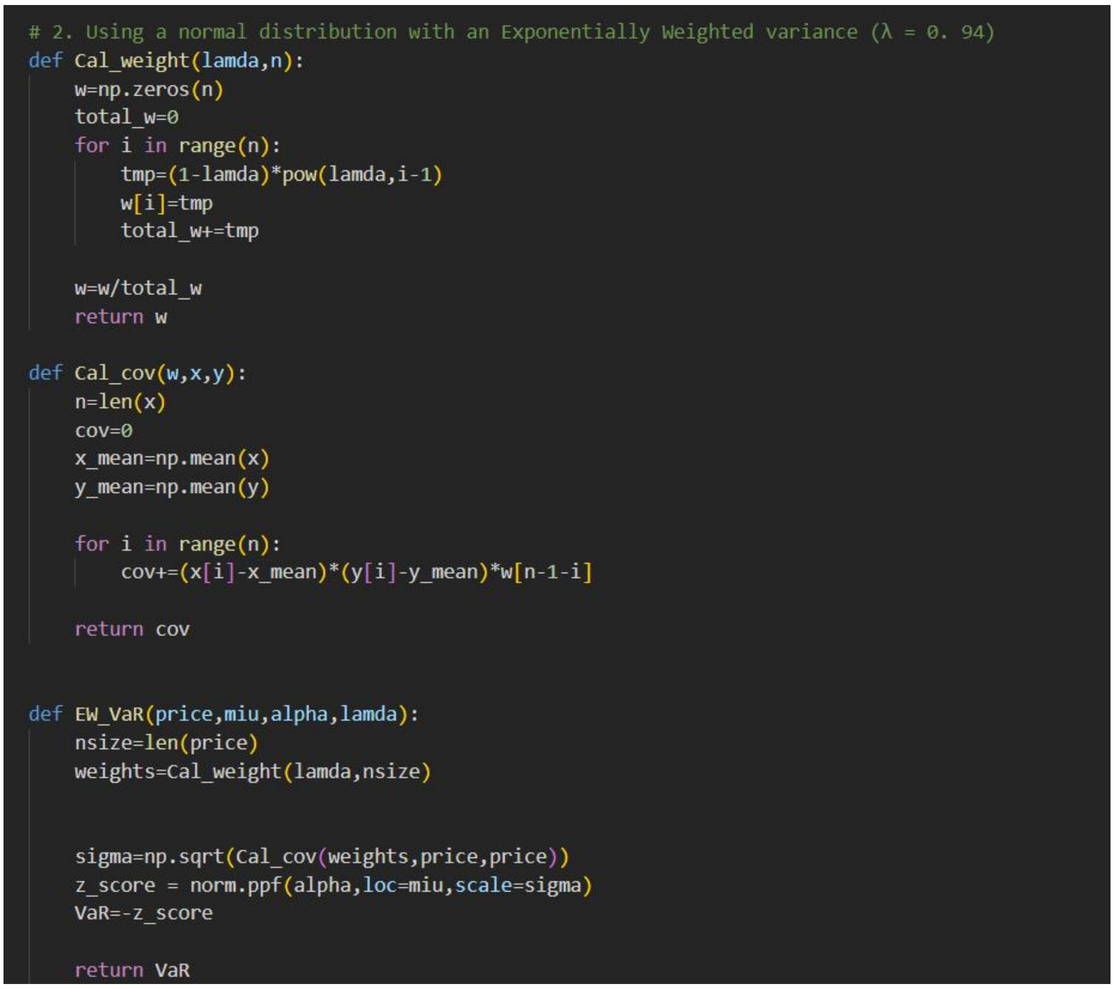
④ Calculate VaR:

* Using a normal distribution.



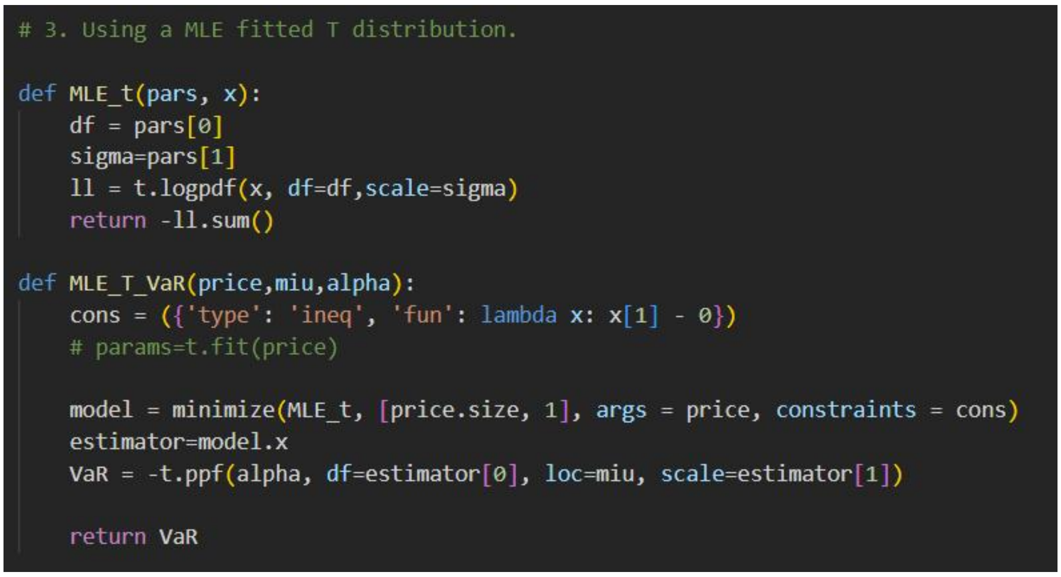
```python
# 1. Using a normal distribution.
def Norm_VaR(price,miu,alpha):
    sigma=price.std()
    z_score = norm.ppf(alpha,loc=miu,scale=sigma)
    VaR=-z_score
    return VaR
```

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



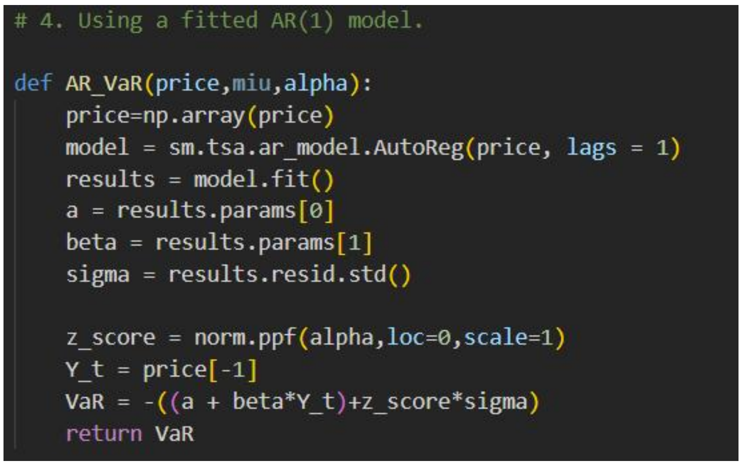
```python
# 2. Using a normal distribution with an Exponentially Weighted variance (λ = 0. 94)
def Cal_weight(lamda,n):
    w=np.zeros(n)
    total_w=0
    for i in range(n):
        tmp=(1-lamda)*pow(lamda,i-1)
        w[i]=tmp
        total_w+=tmp

    w=w/total_w
    return w

def Cal_cov(w,x,y):
    n=len(x)
    cov=0
    x_mean=np.mean(x)
    y_mean=np.mean(y)

    for i in range(n):
        cov+=(x[i]-x_mean)*(y[i]-y_mean)*w[n-1-i]

    return cov


def EW_VaR(price,miu,alpha,lamda):
    nsize=len(price)
    weights=Cal_weight(lamda,nsize)


    sigma=np.sqrt(Cal_cov(weights,price,price))
    z_score = norm.ppf(alpha,loc=miu,scale=sigma)
    VaR=-z_score

    return VaR
```

* Using a MLE fitted T distribution.



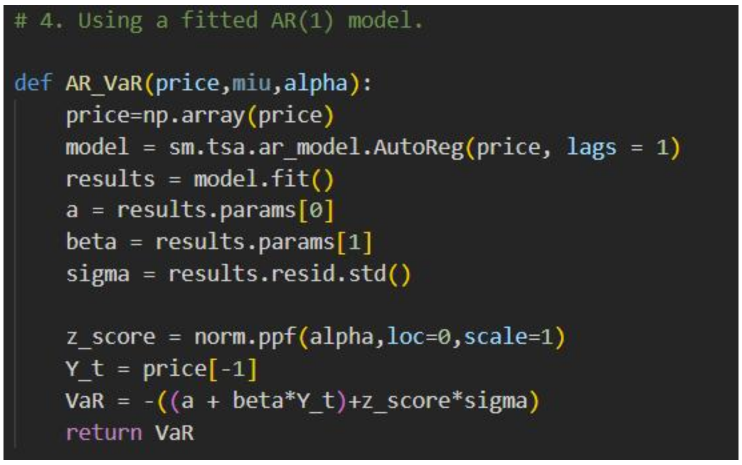
```python
# 3. Using a MLE fitted T distribution.

def MLE_t(pars, x):
    df = pars[0]
    sigma=pars[1]
    ll = t.logpdf(x, df=df,scale=sigma)
    return -ll.sum()

def MLE_T_VaR(price,miu,alpha):
    cons = ({'type': 'ineq', 'fun': lambda x: x[1] - 0})
    # params=t.fit(price)

    model = minimize(MLE_t, [price.size, 1], args = price, constraints = cons)
    estimator=model.x
    VaR = -t.ppf(alpha, df=estimator[0], loc=miu, scale=estimator[1])

    return VaR
```

* Using a fitted AR(1) model.



```python
# 4. Using a fitted AR(1) model.

def AR_VaR(price,miu,alpha):
    price=np.array(price)
    model = sm.tsa.ar_model.AutoReg(price, lags = 1)
    results = model.fit()
    a = results.params[0]
    beta = results.params[1]
    sigma = results.resid.std()

    z_score = norm.ppf(alpha,loc=0,scale=1)
    Y_t = price[-1]
    VaR = -((a + beta*Y_t)+z_score*sigma)
    return VaR
```

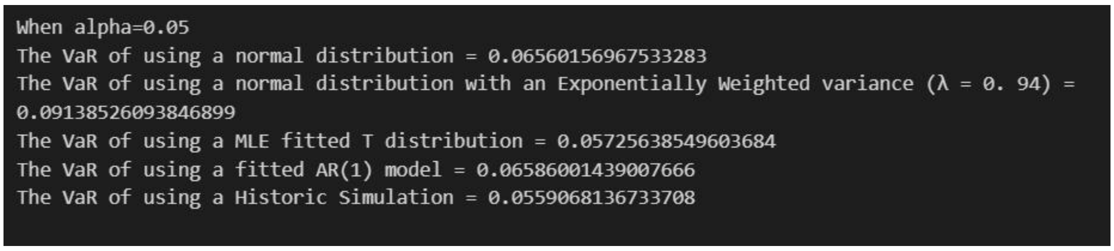
* Using a Historic Simulation.



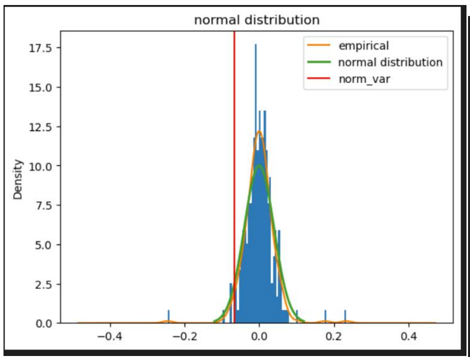
```python
# 4. Using a fitted AR(1) model.

def AR_VaR(price,miu,alpha):
    price=np.array(price)
    model = sm.tsa.ar_model.AutoReg(price, lags = 1)
    results = model.fit()
    a = results.params[0]
    beta = results.params[1]
    sigma = results.resid.std()

    z_score = norm.ppf(alpha,loc=0,scale=1)
    Y_t = price[-1]
    VaR = -((a + beta*Y_t)+z_score*sigma)
    return VaR
```

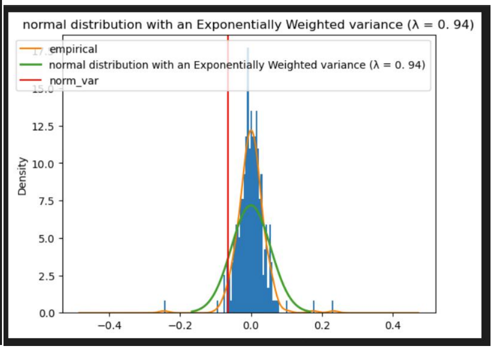
⑤Compare VaRs:

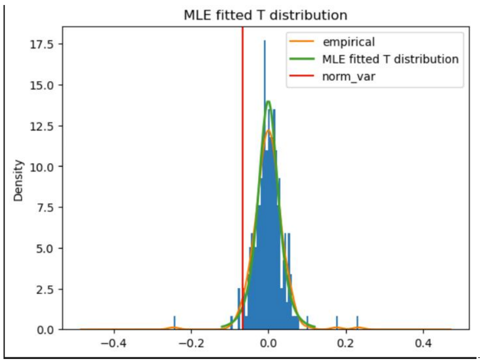
* when alpha=0.05

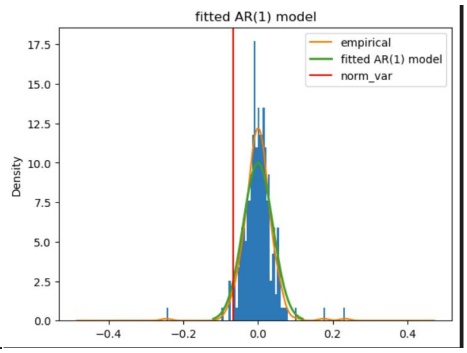


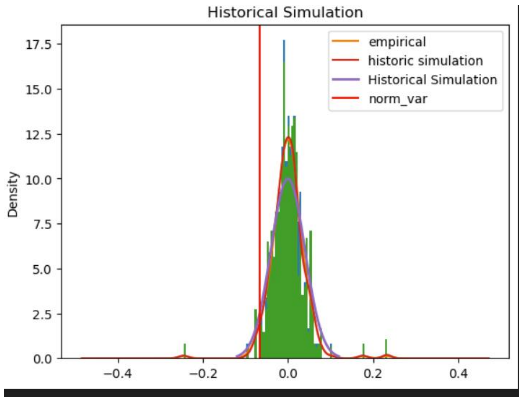
```
When alpha=0.05
The VaR of using a normal distribution = 0.06560156967533283
The VaR of using a normal distribution with an Exponentially Weighted variance (λ = 0. 94) =
0.09138526093846899
The VaR of using a MLE fitted T distribution = 0.05725638549603684
The VaR of using a fitted AR(1) model = 0.06586001439007666
The VaR of using a Historic Simulation = 0.0559068136733708
```











Except the VaR calculated by Exponentially Weighted variance method, all VaRs are around 0.06, while the VaR calculated by Exponentially Weighted variance method is much larger than other VaRs. According to the plots, the MLE fitted T distribution methods best fitted the original distribution.
The Exponentially Weighted variance method contributes to the largest VaR(much larger than others), the historical simulation methods contributes to the smallest VaR, others are all around 0.06 level.

Problem 3

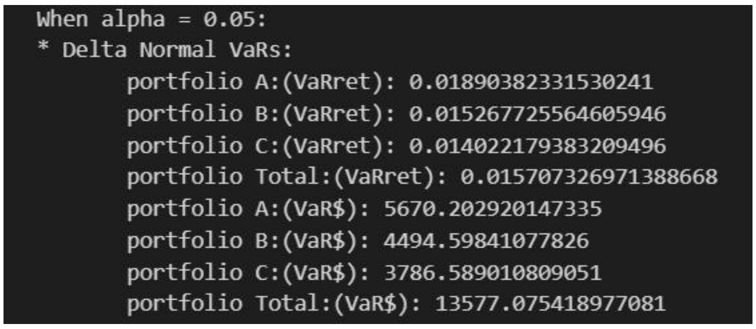
Using Portfolio.csv and DailyPrices.csv. Assume the expected return on all stocks is 0.

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?

①Delta Normal VaR:



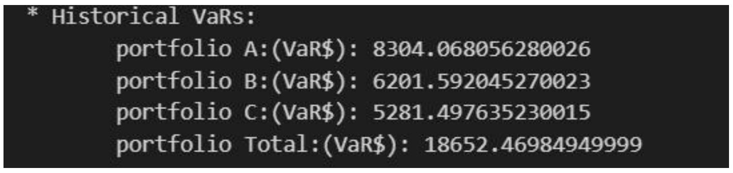
```
 When alpha = 0.05:
 * Delta Normal VaRs:
        portfolio A:(VaRret): 0.01890382331530241
        portfolio B:(VaRret): 0.015267725564605946
        portfolio C:(VaRret): 0.014022179383209496
        portfolio Total:(VaRret): 0.015707326971388668
        portfolio A:(VaR$): 5670.202920147335
        portfolio B:(VaR$): 4494.59841077826
        portfolio C:(VaR$): 3786.589010809051
        portfolio Total:(VaR$): 13577.075418977081
```

Process: according to the latest price to calculate each stock's weights -> calculate the return of each stock and then calculated the exponentially weighted covariance with lambda=0.94 -> calculated the std of weighted portfolio(sigma), then find the α % of the distribution with sigma.

.

The VaRret of each portfolio is around 0.016, much smaller than the VaR of "META", and the portfolio C has the lowest VaRret and VaR$, while the portfolio A has the highest VaRret and VaR$(except the Total).

②Historical VaR:



```
 * Historical VaRs:
        portfolio A:(VaR$): 8304.068056280026
        portfolio B:(VaR$): 6201.592045270023
        portfolio C:(VaR$): 5281.497635230015
        portfolio Total:(VaR$): 18652.46984949999
```

I choose historical VaR, as I think the financial data(like the return of stocks/portfolios) is not always normally distributed. The urgent events, which could contributed to the shock of the financial markets, emerged frequently. Therefore, the financial data always has fat-tail.

The results of Historical VaR has a huge increment comparing to the Delta Normal VaR. And still, the portfolio C has the lowest VaR$, while the portfolio A has the highest VaRret and VaR$(except the Total). And the differences between each two of 3 portfolios become even larger.