Name: Jiaqi Fang
Netid: jf380

Problem1:

- Part1: Mean: 0.050198, Variance: 0.010332, Skewness: 0.120445, Kurtosis: 0.222927
- Part2:
    - Normal distribution is a distribution that involves with mean and standard deviation. It should be used when the dataset is symmetric, has moderate tails and no extreme values. For Student t distribution, it should be used when the dataset/sample size is small, the variance of the population is unknown and has excess kurtosis. In this case, we have a large data set with 1000 observations, however the variance is unknown. Choosing between normal distribution and a t-distribution will also involves with skewness check and kurtosis check. This dataset has a 0.12 for skewness which means normality and symmetric, and a 0.22 kurtosis means the tail are not weighted heavier. In this case, I think a normal distribution will be better.
- Part3:

```
AIC values:
Normal AIC: -1731.586729
t Distribution AIC: -1731.418369
```

    - With the output, the normal distribution has a smaller AIC model which indicates a better AIC model. With a lower AIC, it suggests that the model can explain the data better. AIC also penalizes excessive complexity, so this means the dataset is less overfitting. Overall the normal distribution has a better model than t distribution.

Problem2:

- Part1: I use cov() function get the pairwise covariance matrix

```
Pairwise Covariance Matrix:
          x1        x2        x3        x4        x5
x1  1.470484  1.454214  0.877269  1.903226  1.444361
x2  1.454214  1.252078  0.539548  1.621918  1.237877
x3  0.877269  0.539548  1.272425  1.171959  1.091912
x4  1.903226  1.621918  1.171959  1.814469  1.589729
x5  1.444361  1.237877  1.091912  1.589729  1.396186
```

-
- Part2: for this part I use **np.linalg.eigvalsh** to compute the eigenvalues of the matrix. I use **np.all** function to check if every elements in eigenvalues is greater than the -1e-8(minimize torlance value for numeric error). And got the result it is not a PSD

```
Matrix is PSD: False
Eigenvalues: [-0.31024286 -0.13323183  0.02797828  0.83443367  6.78670573]
```

- Part3: Higham's method
  - My algorithm follows the following picture

$$\Delta S_0 = 0,\ Y_0 = A,\ \gamma_0 = max\ Float$$
$$Loop\ k \in 1 \ldots max\ Iterations$$
$$R_k = Y_{k-1} - \Delta S_{k-1}$$
$$X_k = P_S(R_k)$$
$$\Delta S_k = X_k - R_k$$
$$Y_k = P_U(X_k)$$
$$\gamma_k = \gamma(Y_k)$$
$$if\ |\gamma_k - \gamma_{k-1}| < tol\ then\ break$$

The first line computes the difference between current matrix and the adjustment from the last iteration, then run through Ps method. Ps method will decomposes the input, set all negative eigenvalues to zero, reconstruct the matrix. After ps, will subtract the output from Rk. Then, will go through Pu method, which forces all diagonal item to 1. After covert yk again, I will test if it is smaller than total. The output is follow

```
Higham's Nearest PSD Matrix:
          x1        x2        x3        x4        x5
x1  1.470484  1.332361  0.884378  1.627602  1.399556
x2  1.332361  1.252078  0.619028  1.450604  1.214450
x3  0.884378  0.619028  1.272425  1.076847  1.059658
x4  1.627602  1.450604  1.076847  1.814469  1.577928
x5  1.399556  1.214450  1.059658  1.577928  1.396186

Matrix is PSD: True
```

- Part3: Rebonato and Jackel method
  - My algorithm follows the following picture

$$C\,S = \Lambda\,S \text{ with } \Lambda = diag(\lambda_i)$$

$$\Lambda' : \lambda_i' = max(\lambda_i, 0)$$

$$T : t_i = \left[\sum_{j=1}^{n} s_{i,j}^2 \lambda_j'\right]^{-1},\ T = diag(t_i)$$

$$B = \sqrt{T}\, S \sqrt{\Lambda'}$$

$$BB^T = \hat{C} \approx C$$

- The algorithm start, then Eigenvalue clipping, which make sure all the eigenvalue is greater than 0. Then I form the T through the formula, and constructing B with the T that I just get. Eventually I form the PSD matrix. The output is follow:

```
Rebonato and Jackel's Nearest PSD Matrix:
          x1        x2        x3        x4        x5
x1  1.470484  1.327009  0.842583  1.624464  1.364833
x2  1.327009  1.252078  0.555421  1.433109  1.165906
x3  0.842583  0.555421  1.272425  1.052789  1.060424
x4  1.624464  1.433109  1.052789  1.814469  1.544993
x5  1.364833  1.165906  1.060424  1.544993  1.396186

Matrix is PSD: True
```

- Part4:
  - Here is the result for overlap covariance matrix:

```
Overlap Covariance Matrix:
           x1        x2        x3        x4        x5
x1  0.418604  0.394054  0.424457  0.416382  0.434287
x2  0.394054  0.396786  0.409343  0.398401  0.422631
x3  0.424457  0.409343  0.441360  0.428441  0.448957
x4  0.416382  0.398401  0.428441  0.437274  0.440167
x5  0.434287  0.422631  0.448957  0.440167  0.466272

Matrix is PSD: True
```

- Part5:
- From the data we see that the overlap covariance is quite similar almost all the data are around 0.4. On the other side, the covariance we get from Higham's and Rebonato and Jackel method has data that are more varied. This is probably because the only overlap data is used in Part4 and this loss lots of data that is available for analyzing. The Higham's and Rebonato and Jackel method use non overlap data which maximizes the number of independent observations and can capture more pattern than part4 model do. While we use overlapping data, the effective sample size will be reduced, which mean the variance or covariance will be underestimation or flatten. However, with the Higham's and Rebonato and Jackel method, we can mathematical validity for the covariance (PSD, but may not perfectly match the original data's covariance structure because of eigenvalue adjustments.

Problem3:

- Part1:
  - In this part we are asked to fit a multivariate normal. So we need to get a mean vector and covariance matrix. For getting the mean vector, I take average of each variable (x1 and x2) across the dataset, and I got the result.

```
Estimated mean vector: [0.04600157 0.09991502]
```

  - For getting the covariance matrix, I use unbiased sample covariance to estimate covariance matrix. The covariance matrix reveals how x1 and x2 vary both individually and jointly. The value 0.00492354 explains there is a positive linear relationship between x1 and x2

```
Estimated covariance matrix:
 [[0.0101622  0.00492354]
 [0.00492354 0.02028441]]
```

- Part2:
  - In this part I use two methods:
    - Method 1: Conditional Distributions: I get mu1 and mu2 from the mean vector I had from part1, and get the sigma11, sigma12, and sigma22 from the covariance matrix as well. I go through the formula that showed below and got the mean and variance from conditional distribution method

Math formula:

$$\bar{\mu} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(a - \mu_2)$$

$$\bar{\Sigma} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{12}$$

Code formula:

```
cond_mean = mu_2 + (sigma_12 / sigma_11) * (x1_value - mu_1)
cond_var= sigma_22 - (sigma_12 * sigma_12 / sigma_11)
```

Output:

```
Method 1 (conditional formula) given X1 = 0.6:
cond_mean: 0.3683249958609774
cond_var: 0.01789896964508753
```

    - Method 2: OLS method: the best linear model will be the following

$$X_2 = \alpha + \beta X_1 + \text{error}$$

Then, I can get the beta and alpha from the sigma12, sigma11, mu2 and mu1, which I can get from last part:

$$\beta = \frac{\sigma_{12}}{\sigma_{11}}, \quad \alpha = \mu_2 - \beta\mu_1.$$

Then, I can compute the mean and variance from the following formula:

$$\text{cond\_mean\_ols} = \alpha + \beta \times 0.6.$$

$$\text{cond\_var\_ols} = \sigma_{22} - \frac{\sigma_{12}^2}{\sigma_{11}}.$$

That give me the result:

```
Method 2 (OLS) given X1 = 0.6:
OLS slope = 0.4844959102797748
OLS intercept = 0.07762744969311253
OLS_mean: 0.3683249958609774
OLS_var: 0.01789896964508753
```

- Part3:

From the formula that introduced in Cholesky algorithm on the class material

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} L_{11} & L_{21} & L_{31} \\ 0 & L_{22} & L_{32} \\ 0 & 0 & L_{33} \end{pmatrix}$$

$$= \begin{pmatrix} L_{11}^2 & & \text{(symmetric)} \\ L_{21}L_{11} & L_{21}^2 + L_{22}^2 & \\ L_{31}L_{11} & L_{31}L_{21} + L_{32}L_{22} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{pmatrix},$$

$$\mathbf{L} = \begin{pmatrix} \sqrt{A_{11}} & 0 & 0 \\ A_{21}/L_{11} & \sqrt{A_{22} - L_{21}^2} & 0 \\ A_{31}/L_{11} & \left(A_{32} - L_{31}L_{21}\right)/L_{22} & \sqrt{A_{33} - L_{31}^2 - L_{32}^2} \end{pmatrix}$$

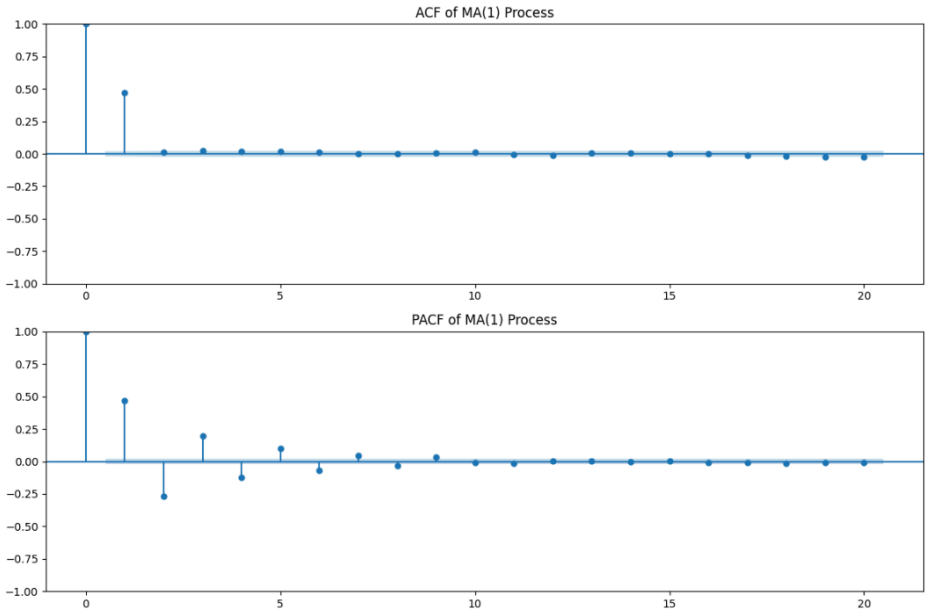
I use seed 21 and draw 10000 normal sample and simulate with this algorithm. I eventually got the result as following:

```
X2_simulation: [0.35344841 0.20018958 0.13940819 ... 0.28335429 0.44463506
0.48294723]
Simulated mean of X2 | X1=0.6: 0.37135179656862566
Simulated variance of X2 | X1=0.6: 0.017759021836734952
```

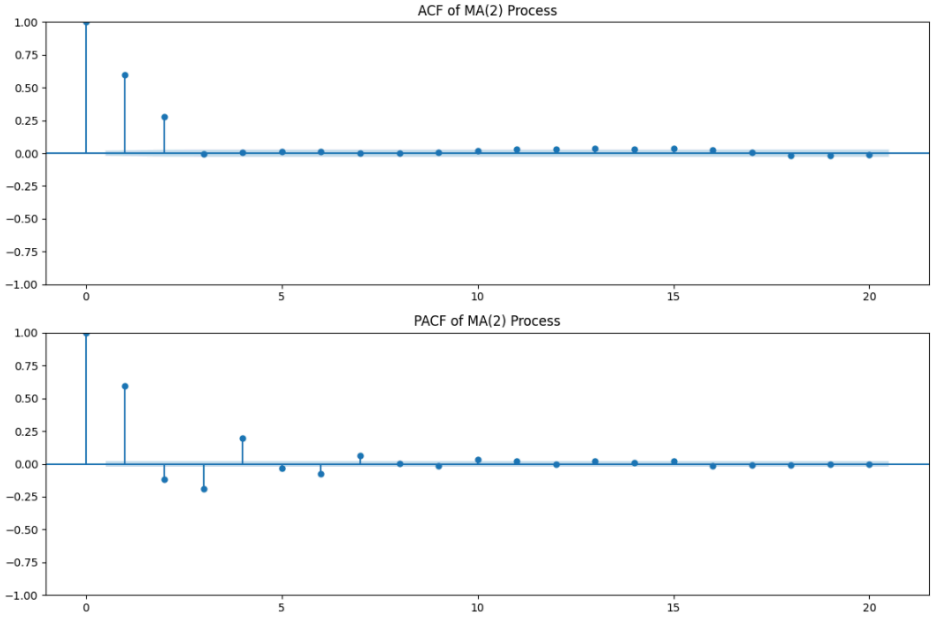
The output average of the mean and variance of cholskey root is very close to the mean and variance we get from two methods(conditional and OLS method).which means the mean and variance we got from Part 2 is consistent with the 10000 sample draw, it is robust and reliable. With the close match up between mean and variance, this means the two method I use is reliable and valid.

Problem4:

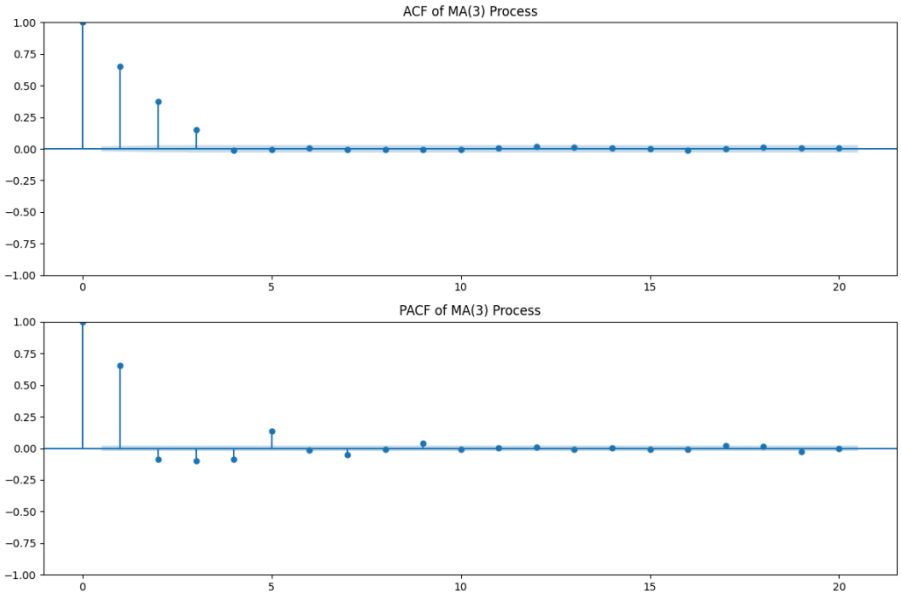
- Part1:



Input: ar=[1], ma=[1, 0.7]



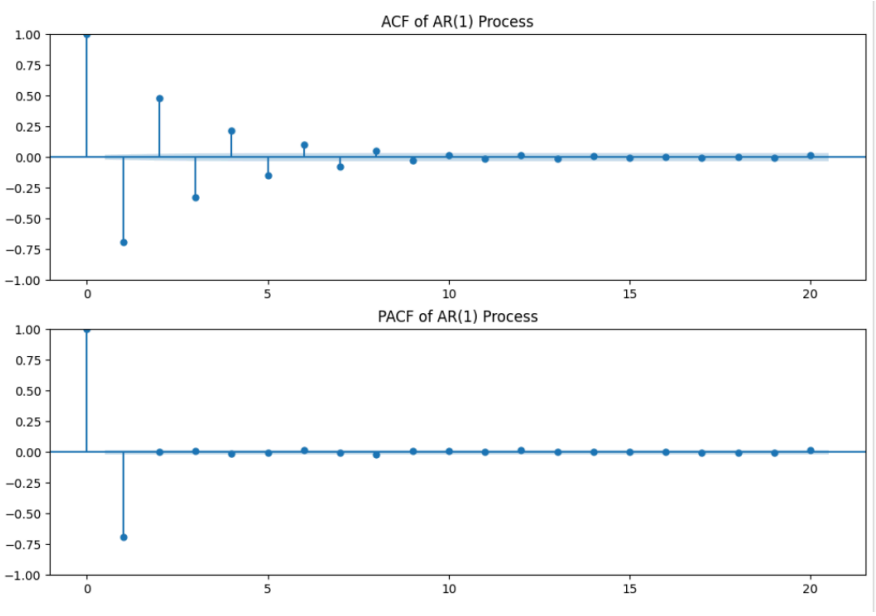
Input: ar=[1], ma=[1, 0.7, 0.5]



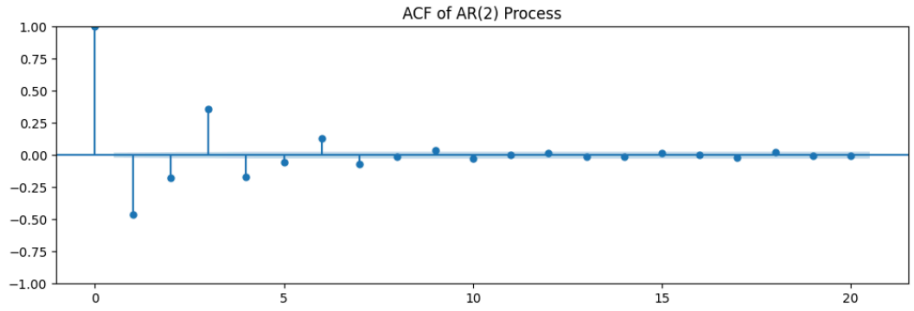
Input: ar=[1], ma=[1, 0.7, 0.5, 0.3]

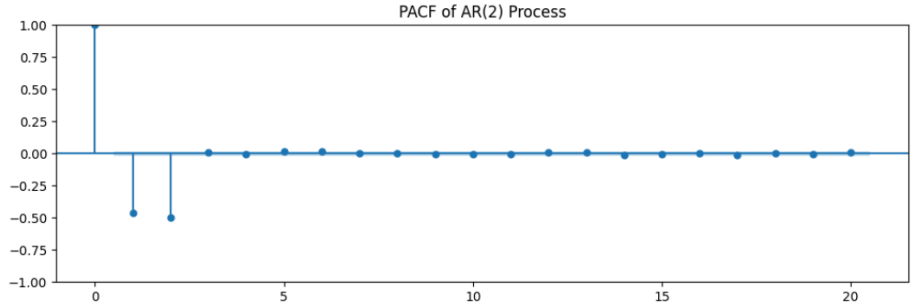
From the picture above we can find out that PACF gradually narrow down and ACF graph of MA(q) has a cutoff after lag(q).

- Part2:

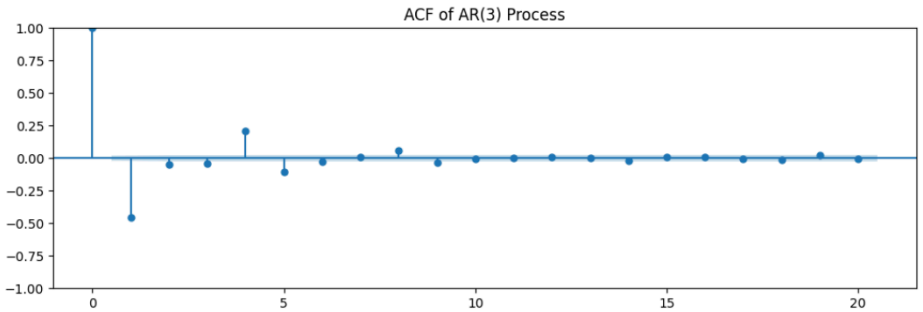


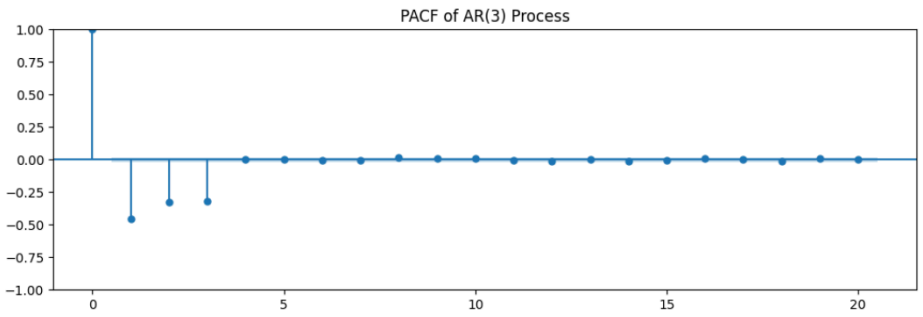
Input: ar=[1, 0.7], ma = [1]





Input: ar=[1, 0.7, 0.5], ma = [1]



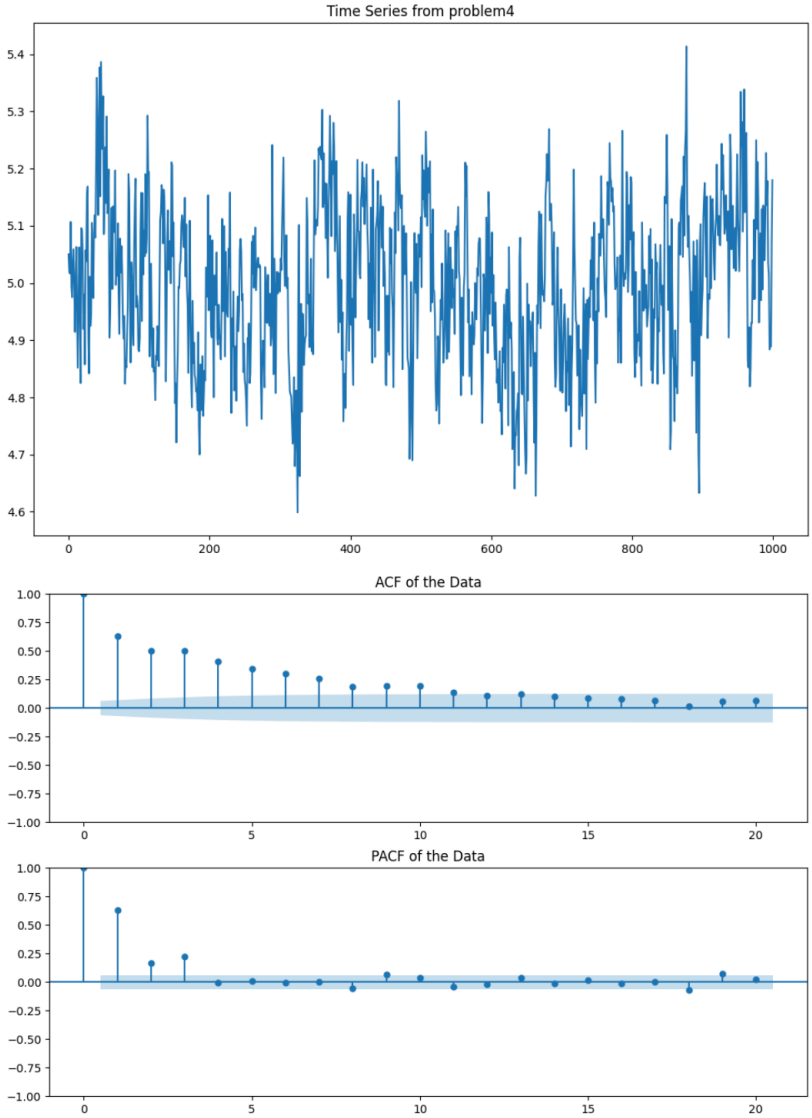


Input: ar=[1, 0.7, 0.5, 0.3], ma = [1]

From the graph above we can see it is opposite than the MA models, the ACF slowly narrow down, the PACF of AR(q) model has a cutoff after lag q.

- Part3:

- With fitting the data we get this data output:



- From this graph, we can find out the data fluctuates around a consistent mean without any prominent trends or seasonality visible. With the lack of trend or seasonal pattern, the graph may indicates it is stationary. From the two graph below, we can discover that the ACF gradually narrow down to zero. This ACF shows that the higher MA model the better the result. But there is a problem with High MA model which means the model will be too complex and that also means the model will overfit the data. With valuing data through AIC, AIC will punish excessive complexity, which will result a worse model. So we should look more on AR model. The PACF graph has a cut off after lag 3. This means the lag1 to 3 indicates and capture the most of the variance from the data. With only 3 lag this will also has a model that avoid overfitting and complexity. This indicates that the model fits well with the AR(3) model.

- Part4:
    - To check the best model I run through AR(1) to AR(4) and MA(1) to MA(4), I tested all the model buy calculating their AIC and turnout the AR(3) model with order(3,0,0) has the best result overall, since it has the smallest AIC. With a lower AIC, it suggests that the model can explain the data better. AIC also penalizes excessive complexity, so this means this AR(3) model is less overfitting and the best among all the other dataset. Output graph below:

```
Model Comparison Results:
   Model      Order     AICc
0  AR(3)  (3, 0, 0) -1746.22
1  AR(4)  (4, 0, 0) -1744.22
2  AR(2)  (2, 0, 0) -1696.05
3  MA(4)  (0, 0, 4) -1677.50
4  AR(1)  (1, 0, 0) -1669.07
5  MA(3)  (0, 0, 3) -1645.07
6  MA(2)  (0, 0, 2) -1559.21
7  MA(1)  (0, 0, 1) -1508.90
```

Problem 5:

- Part1:
  To calculate the exponentially weighted covariance matrix function: I follow the procedure in the class material. And with the λ = 0.97

$$w_{t-i} = (1 - \lambda)\,\lambda^{i-1}$$

$$\widehat{w_{t-i}} = \frac{w_{t-i}}{\sum_{j=1}^{n} w_{t-j}}$$

$$\widehat{\sigma_t^2} = \sum_{=1}^{n} w_{t-i}\left(x_{t-i} - \bar{x}\right)^2$$

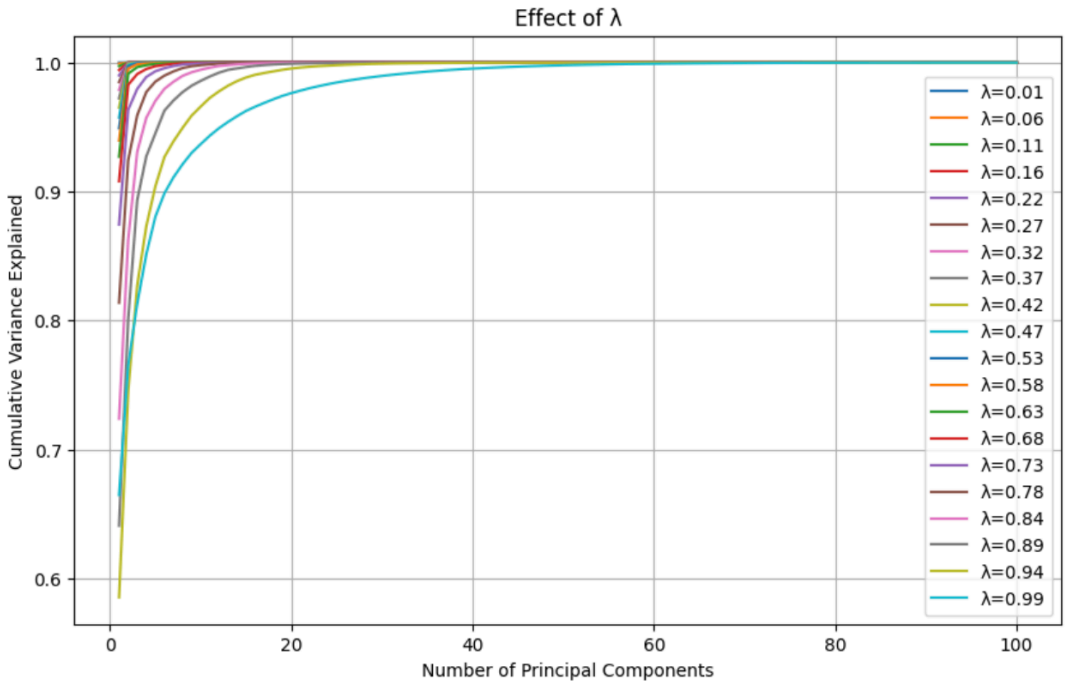
$$\widehat{cov(x,y)} = \sum_{i=1}^{n} w_{t-i}\left(x_{t-i} - \bar{x}\right)\left(y_{t-i} - \bar{y}\right)$$

With these formula I got the output:

```
Exponentially Weighted Covariance Matrix (λ = 0.97):
          SPY      AAPL      NVDA      MSFT      AMZN      META     GOOGL  `
SPY   0.000072  0.000054  0.000124  0.000080  0.000112  0.000081  0.000088
AAPL  0.000054  0.000139  0.000041  0.000084  0.000081  0.000056  0.000071
NVDA  0.000124  0.000041  0.000663  0.000133  0.000196  0.000186  0.000145
MSFT  0.000080  0.000084  0.000133  0.000161  0.000174  0.000126  0.000122
AMZN  0.000112  0.000081  0.000196  0.000174  0.000323  0.000188  0.000201
...        ...       ...       ...       ...       ...       ...       ...
KKR   0.000135  0.000041  0.000220  0.000105  0.000188  0.000119  0.000161
MU    0.000148  0.000055  0.000304  0.000151  0.000173  0.000165  0.000167
PLD   0.000059  0.000060  0.000014  0.000061  0.000057  0.000014  0.000025
LRCX  0.000127  0.000084  0.000322  0.000152  0.000185  0.000229  0.000158
EQIX  0.000053  0.000038  0.000050  0.000054  0.000071  0.000074  0.000051

          AVGO      TSLA      GOOG  ...      SBUX       MMC       MDT  \
SPY   0.000176  0.000234  0.000088  ...  0.000040  0.000031  0.000029
AAPL  0.000140  0.000173  0.000071  ...  0.000008  0.000014  0.000008
NVDA  0.000223  0.000235  0.000148  ...  0.000038  0.000012  0.000008
MSFT  0.000196  0.000279  0.000123  ...  0.000017  0.000025  0.000018
AMZN  0.000295  0.000373  0.000199  ...  0.000019  0.000021  0.000013
...        ...       ...       ...  ...       ...       ...       ...
KKR   0.000276  0.000481  0.000163  ...  0.000088  0.000069  0.000058
MU    0.000897  0.000479  0.000180  ...  0.000072 -0.000004  0.000026
PLD   0.000064  0.000121  0.000030  ...  0.000083  0.000044  0.000059
LRCX  0.000583  0.000443  0.000162  ...  0.000071  0.000006  0.000017
EQIX  0.000064  0.000154  0.000053  ...  0.000059  0.000045  0.000026

            CB       LMT       KKR       MU       PLD      LRCX      EQIX
SPY   0.000027  0.000026  0.000135  0.000148  0.000059  0.000127  0.000053
AAPL  0.000026  0.000003  0.000041  0.000055  0.000060  0.000084  0.000038
NVDA -0.000009  0.000027  0.000220  0.000304  0.000014  0.000322  0.000050
MSFT  0.000024 -0.000015  0.000105  0.000151  0.000061  0.000152  0.000054
AMZN  0.000015 -0.000008  0.000188  0.000173  0.000057  0.000185  0.000071
...        ...       ...       ...       ...       ...       ...       ...
KKR   0.000070  0.000064  0.000418  0.000253  0.000065  0.000193  0.000098
MU   -0.000006  0.000035  0.000253  0.001475  0.000156  0.000707  0.000085
PLD   0.000033  0.000029  0.000065  0.000156  0.000250  0.000084  0.000088
LRCX  0.000009 -0.000018  0.000193  0.000707  0.000084  0.000738  0.000073
EQIX  0.000030  0.000006  0.000098  0.000085  0.000088  0.000073  0.000153

[100 rows x 100 columns]
```

- Part2:

  To find out the influence by λ, I use the code line: lambda_values = np.linspace(0.01, 0.99, 20) to create an array of 20 values that are equally spaced between 0.01 and 0.99. I loop through each λ to compute the EWMA covariance, then perform PCA, and eventually calculate the cumulative variance. The plot below is the output:



- Part3: from the graph there are 20 line with different λ value. From this graph we can find out that 0.99 is on bottom and the 0.01 one is on the top. With the same number of principal components, the lower the λ value the higher cumulative variance explained. With increasing λ value, the first principal components capture more variance. This mean high λ value correspond to higher explained variance percentages overall. At the same time, lower λ values risk diminishing model sensitivity. When low λ values reduces weight on recent observation will lead to inadequately capture the variance.

Problem 6:

- Part1: I first read the csv file and try to run through Cholesky Root method, but I found out the covariance matrix that provided is not positive definite.

  ```
  Covariance matrix is Positive Definite (PD): False
  ```

  So I use Higham's method to find the nearest positive definite.(part of the screenshot

```
Higham's Nearest PSD Matrix:
                x1            x2            x3            x4            x5  \
x1    5.315251e-03  6.609109e-07 -3.975723e-06  1.537962e-06 -4.787720e-06
x2    6.609109e-07  1.781312e-03  2.793018e-07 -2.318668e-07 -1.767473e-06
x3   -3.975723e-06  2.793018e-07  4.924608e-03  1.131462e-06 -3.970215e-06
x4    1.537962e-06 -2.318668e-07  1.131462e-06  2.544771e-03 -1.555702e-06
x5   -4.787720e-06 -1.767473e-06 -3.970215e-06 -1.555702e-06  9.133101e-03
...            ...           ...           ...           ...           ...
x496 -9.972524e-07 -7.374728e-07 -4.905357e-06  1.436200e-06  1.055879e-05
x497  6.089889e-08  1.438608e-08 -7.346708e-07  3.373734e-07  1.040297e-07
x498  7.936150e-07  7.170738e-07 -1.080232e-06  3.438228e-07  8.826131e-07
x499  6.102625e-06  2.136593e-06  1.810569e-06  2.750017e-06 -1.000336e-05
x500 -1.673841e-07  1.416918e-07 -2.110012e-07  2.931466e-07  1.809168e-07

                x6            x7            x8            x9           x10  \
x1    7.149027e-07  6.207943e-06 -1.878452e-06  4.660371e-06  5.791606e-07
x2    1.305869e-06 -3.083304e-06  5.919198e-07  1.945374e-06  2.969406e-07
x3    1.734963e-06 -1.246778e-06  2.207709e-06  6.203153e-06 -1.243975e-06
x4   -3.174481e-07  7.590897e-07 -2.250509e-07 -7.256591e-07 -2.275846e-06
x5   -7.802437e-06 -1.881759e-05 -3.749109e-06 -1.602133e-06  2.144879e-06
...            ...           ...           ...           ...           ...
x496 -1.745172e-06 -3.405648e-06 -4.855977e-06  4.324896e-06  4.239300e-06
x497 -1.784277e-07 -6.584650e-07 -5.241199e-07 -1.186280e-06 -4.593006e-08
x498 -5.751298e-07  1.139639e-07  4.833031e-07  6.644169e-07  6.424195e-07
x499 -6.091127e-07 -3.083471e-06  1.142999e-06  1.282545e-06 -1.971312e-06
x500 -2.166929e-07 -5.455886e-07  5.193669e-07 -5.232449e-07 -1.448961e-07
```

  The calculation method is same with the one I use in question 2. After I find the nearest positive definite, I simulate the model through Cholesky factorization algorithm. I generate z ~N(0,1) of shape(10000 * 500), then multiply Z by Lt, and the resulting sample have covariance Σ. I follow the math formula:

$$\mathrm{Var}(\mathbf{Z}L^T) = L\,\mathrm{Var}(\mathbf{Z})\,L^T = LIL^T = LL^T = \Sigma.$$

  And use time() formula to time the time spent on this.

```
[[-0.00378849 -0.00469357  0.07311074 ... -0.04355983 -0.02278509
   0.01471134]
 [-0.01616741 -0.0235807  -0.07875646 ... -0.07531008 -0.04851867
   0.00088425]
 [ 0.11254952  0.00404193  0.04886215 ... -0.09072555  0.08792444
   0.00816659]
 ...
 [-0.05220491 -0.09819959 -0.00584543 ... -0.02244307 -0.0066773
   0.00730107]
 [-0.03300452 -0.01461411 -0.02469322 ... -0.02691835  0.14031834
   0.00537795]
 [ 0.04738729 -0.04181144 -0.09833275 ... -0.02623829 -0.09448303
   0.0267569 ]]
Time (Cholesky): 0.2046 seconds
```

- Part2:
- For PCA, I first sorted the eigenvalues in descending order ensures the largest principal components come first. Then we select 75% variance by finding the k follows

$$: \sum_{i=1}^{k} \lambda_i \geq 0.75 \times \sum_{i=1}^{n} \lambda_i.$$

Then we found out k = 45, which means the first 45 out of 500 captures at least 75% of the variance. Then we do 10000 draws with reduced dimension k = 45, follows the formula:

$$: \mathbf{Z}_k \sqrt{\Lambda_k} Q_k^T$$

The eventual output and time taken for PCA is :

```
k=45
[[ 0.08788078  0.01648881 -0.05526702 ... -0.06730822  0.1023446
   0.02012903]
 [ 0.10932989  0.01155781 -0.08378176 ...  0.01505862  0.00303375
   0.02898716]
 [-0.13493637 -0.0344283   0.16271951 ... -0.04937761  0.00726931
  -0.03856626]
 ...
 [ 0.08783624  0.01780638 -0.12283303 ...  0.06970027  0.05053552
   0.02286392]
 [ 0.00588567 -0.04270026 -0.02572529 ... -0.01948072 -0.11276811
   0.0307553 ]
 [-0.0730733  -0.03279773  0.07854064 ...  0.00827236 -0.00982329
  -0.0095053 ]]
Time (PCA): 0.0690 seconds
```

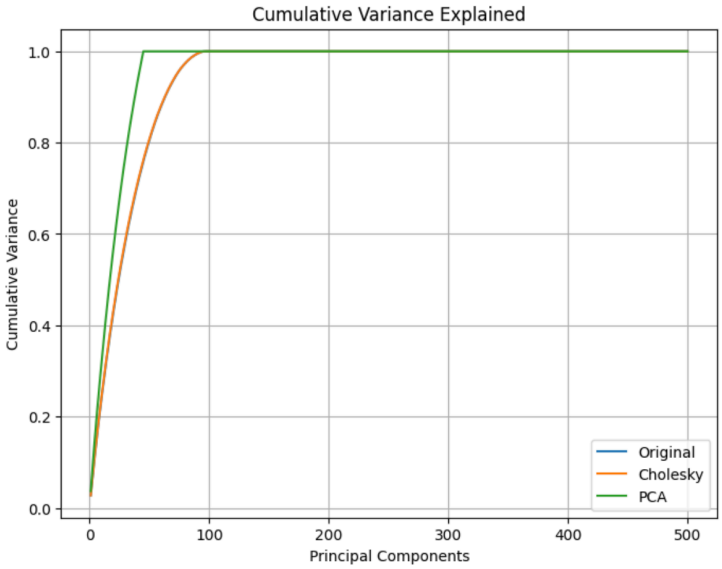
- Part3:

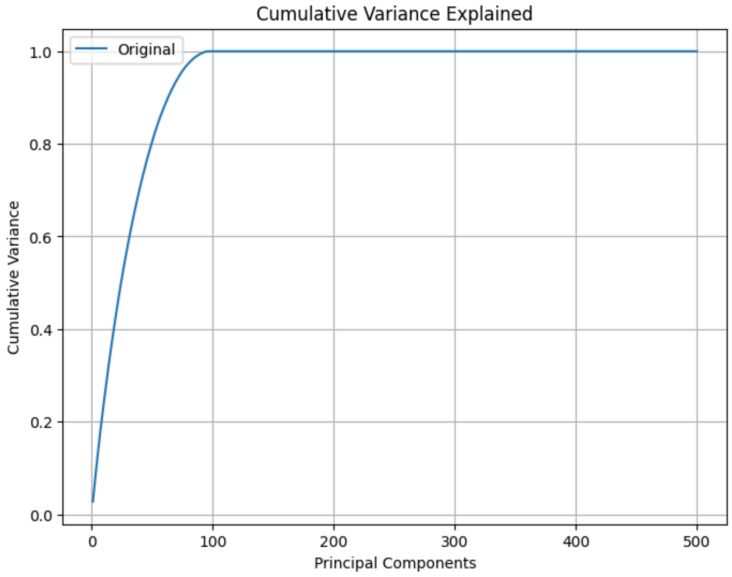
  With the code I find the output:

  ```
  Frobenius difference (Cholesky vs original): 0.020943948321592378
  Frobenius difference (PCA vs original): 0.08315006650779189
  ```

  The Frobenius norm difference measures how much corrected matrices deviate from the original matrix. In the output I found out that PCA has a higher frobenius than Cholesky. This means the Cholesky-based method make smaller modification, which means it retains more original matrix structure. Since PCA only use 45 components out of 500, it leads to a bigger approximation error and that's why it has a higher fronbenius difference score then Cholesky method. This means in the future if we looking for minimally adjust matrix we will use Cholesky method.

- Part4:





From the graph I find out that the Cholesky method has an identical graph as original graph. This is probably because the Cholesky method uses the entire matrix that is same as the original matrix. In that case, the Cholesky covariance matrix has the same eigenvalue distribution as the original. This is why the cumulative explained graph for Original and Cholesky is similar. For PCA because it find the principal components that can explain 75% of the variance. As a result, its eigenvalue distribution diverges from that of the original matrix, leading to a different cumulative variance profile. This means PCA will get more cumulative variance explained by increase principal components.

- Part5

```
Time taken (Cholesky): 0.2008 seconds
Time taken (PCA): 0.0628 seconds
```

From the output we find that the PCA take way less time then Cholesky method. This is probably because the way PCA method process. When PCA's decomposition is done it use the reduced matrix, 45 components out of 500. However for Cholesky method, it will run through all 500 components and this make the difference between time take on two different method.

- Part6

Overall, for Cholesky and PCA they are both good method but it may have a tradeoff. For Cholesky, it can provide entire covariance structure, it is more straight forward. However, it may take more time to process it. For a large N, it may take a long time to process it. For PCA, it is more flexible, you keep only top k principal components, it has better interpretability because it can identifies major directions, since PCA sorts eigenvalues from largest to smallest, showing which directions (principal components) have the most variance. At the same time PCA take less time comparing to Cholesky method since it only process k components at total.

Overall, if I want a model that is most accurate and can has the result exact same with the full covariance, I will take Cholesky method. If I want some method that is faster and has better interpretability. I will choose PCA method.