Response for Project Week 5

Course Code: Fintech545

Course Title: Quantitative Risk Management

Student Name: Wanglin (Steve) Cai

Student Net ID: WC191

Problem 1

Use the data in problem1.csv. Fit a Normal Distribution and a Generalized T distribution to this data. Calculate the VaR and ES for both fitted distributions.

Overlay the graphs the distribution PDFs, VaR, and ES values. What do you notice? Explain the differences.

1.1 Fitting the Normal Distribution and Generalized T Distribution

In this question, at first it asks us to fit a normal distribution and a generalized t distribution to the data.

First, we can check the dataset for problem 1:

	x
0	-0.002665
1	-0.045128
2	0.053635
3	0.010450
4	-0.016284

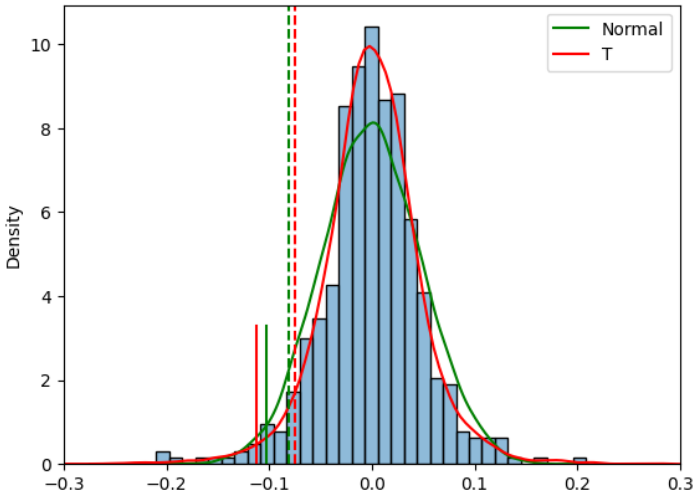
Then, we can fit the normal distribution and generalized t distribution to calculate the VaR and ES.

For the normal distribution, we can use the statistics of the dataset, i.e., the mean and the standard deviation to fit the distribution. After fitting the distribution, we found that the VaR for normal distribution is 0.082, and the Expected Shortfall for the fitted normal distribution is 0.102.

For the t distribution, we can also use the statistics of the dataset, i.e., the mean and the standard deviation as a starting point to do optimization to fit the distribution. However, we also need to fit the degree of freedom. In the optimization, we simply let the initial value of the degree of freedom to be 10. After the optimization, the fitted distribution has a degree of freedom equals to 4.25. For this fitted t distribution, we found that the VaR for it is 0.077, and the Expected Shortfall for the fitted distribution is 0.112.

1.2 Plotting the Graph

Then, we can plot the graph of the distribution PDFs, VaR and ES values. The graph is shown below:



From the graph, we can see that for the t distribution, it has fatter tail than the normal distribution, though the kurtosis of t distribution is larger. We can also see that the expected shortfall is indeed larger than VaR. An interesting finding is that for VaR, the VaR for the normal distribution is larger than the VaR for the T distribution. However, the ES for t distribution is larger than the ES for normal distribution. I think it can be well explained by the definition of ES and t distribution. Since t distribution has fatter tail at its both ends, the expected shortfall, which is the measurement of the average (not like VaR -- minimum) loss, is definitely larger than ES for normal distribution. For the VaR, I think it can be easily affected by parameters like kurtosis since it is the 'exact point' of the distribution. Therefore, the VaR for t distribution here is smaller than the VaR for normal distribution.

Problem 2

In your main repository, create a Library for risk management. Create modules, classes, packages, etc as you see fit. Include all the functionality we have discussed so far in class. Make sure it includes

1. Covariance estimation techniques.
2. Non PSD fixes for correlation matrices
3. Simulation Methods
4. VaR calculation methods (all discussed)
5. ES calculation

Create a test suite and show that each function performs as expected.

Testing Each Function in the Created Library

By building the library for risk management, we have created 5 different function groups mentioned above. For testing the functions in these groups, we have created a test suite. Then, we will show that our functions perform as expected.

There are two functions that are not belong to these 5 groups: demean() and return_calculate().

For the return_calculate() function, we use the file 'DailyPrices.csv' for testing.

```
DP_return
✓ 0.0s
```

	SPY	AAPL	MSFT	AMZN	TSLA	GOOGL	GOOG	META	NVDA	BRK-B
1	0.016127	0.023152	0.018542	0.008658	0.053291	0.007987	0.008319	0.015158	0.091812	0.006109
2	0.001121	-0.001389	-0.001167	0.010159	0.001041	0.008268	0.007784	-0.020181	0.000604	-0.001739
3	-0.021361	-0.021269	-0.029282	-0.021809	-0.050943	-0.037746	-0.037669	-0.040778	-0.075591	-0.006653
4	-0.006475	-0.009356	-0.009631	-0.013262	-0.022103	-0.016116	-0.013914	-0.007462	-0.035296	0.003987
5	-0.010732	-0.017812	-0.000729	-0.015753	-0.041366	-0.004521	-0.008163	-0.019790	-0.010659	-0.002033
...	...	...	...	...	...	...	...	...	...	...
244	-0.010629	0.024400	-0.023621	-0.084315	0.009083	-0.027474	-0.032904	-0.011866	-0.028053	-0.010742
245	-0.006111	-0.017929	-0.006116	-0.011703	0.025161	-0.017942	-0.016632	-0.002520	-0.000521	-0.000259
246	0.013079	0.019245	0.042022	-0.000685	0.010526	0.046064	0.044167	0.029883	0.051401	0.014720
247	-0.010935	-0.017653	-0.003102	-0.020174	0.022763	-0.076830	-0.074417	-0.042741	0.001443	-0.014346
248	-0.008669	-0.006912	-0.011660	-0.018091	0.029957	-0.043876	-0.045400	-0.030039	0.005945	-0.004117

After printing the result, we can see that the function matches our expectation.

2.1 Covariance Estimation Techniques

For the purpose of testing this module, we continued to utilize the file 'DailyPrices.csv'. Firstly, we drop the date variable in this file. Then, we can do the calculating.

```
DP = pd.read_csv("DailyPrices.csv")

# Calculate return
DP_return = RM.return_calculate(DP)
DP_return.drop('Date', axis=1, inplace=True)
```

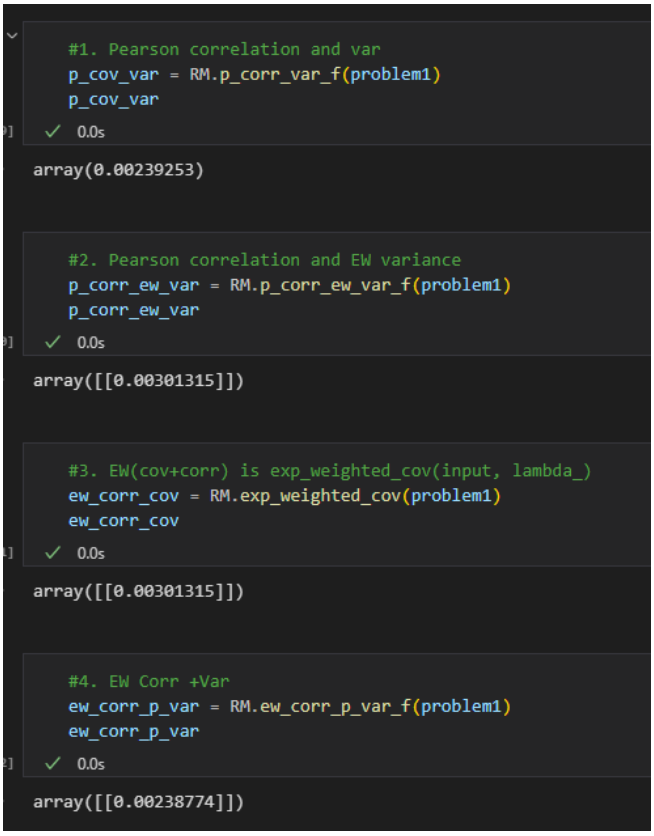
For covariance estimation, I build the function of calculating the exponentially weighted covariance matrix which is in the previous assignment. For this function, I add parameters like lambda with default value equals to 0.97. We can check that the output value matches our expectation.

```
ew_cov = RM.exp_weighted_cov(DP_return)
pd.DataFrame(ew_cov)
✓ 0.1s
```

	0	1	2	3	4	5	6	7	8	
0	0.000155	0.000210	0.000235	0.000285	0.000279	0.000279	0.000284	0.000331	0.000368	0.0001
1	0.000210	0.000421	0.000321	0.000329	0.000526	0.000417	0.000417	0.000563	0.000527	0.0001
2	0.000235	0.000321	0.000499	0.000505	0.000438	0.000521	0.000529	0.000664	0.000610	0.0001
3	0.000285	0.000329	0.000505	0.000939	0.000542	0.000657	0.000671	0.000923	0.000696	0.0002
4	0.000279	0.000526	0.000438	0.000542	0.002369	0.000434	0.000444	0.000636	0.001212	0.0001
...	...	...	...	...	...	...	...	...	...	
95	0.000034	0.000035	0.000032	-0.000016	0.000066	-0.000019	-0.000017	-0.000049	0.000013	0.0000
96	0.000171	0.000222	0.000224	0.000296	0.000252	0.000307	0.000307	0.000380	0.000468	0.0001
97	0.000209	0.000286	0.000297	0.000436	0.000666	0.000325	0.000330	0.000615	0.000536	0.0001

We also build covariance synthesis function (e.g., synthesize covariance matrix using pearson correlation and exponentially weighted variance). We use 'problem1.csv' as

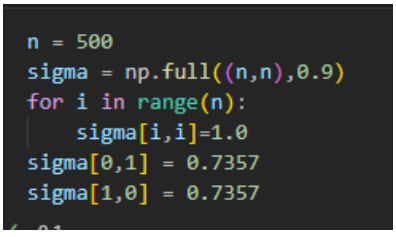
our testing file. The result is as follows:

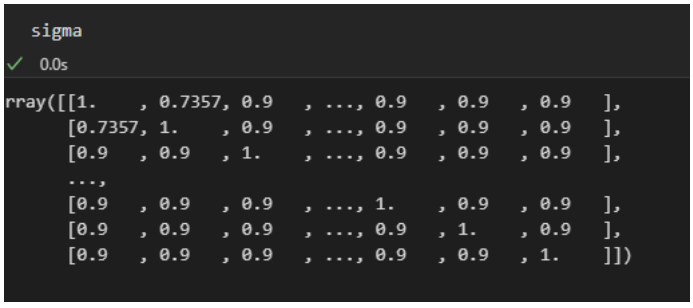


2.2 Non PSD Fixes for Correlation Matrices

For Non PSD Fixes for correlation matrices, I build functions according to the previous assignments. For example, I build the near_PSD(), higham() to convert a Non PSD correlation matrix to a PSD/PD correlation matrix.

First, we can generate a Non PSD by using the following code:





We also build a function to check if a matrix is PSD or not. For this matrix, we can

easily find that it is not a PSD matrix. However, if we use near_PSD() or the Higham() methods to convert the Non PSD matrix, the converted matrix will be a PSD matrix.

```
print("Sigma: ", RM.is_psd(sigma))
print("Sigma fixed with near_psd(): ", RM.is_psd(RM.near_PSD(sigma)))
print("Sigma fixed with Higham_psd(): ", RM.is_psd(RM.Higham(sigma)))
```

✓ 15.9s

```
Sigma:  False
Sigma fixed with near_psd():  True
Sigma fixed with Higham_psd():  True
```

```
RM.near_PSD(sigma)
```

✓ 0.5s

```
array([[1.        , 0.74381947, 0.88594237, ..., 0.88594237, 0.88594237,
        0.88594237],
       [0.74381947, 1.        , 0.88594237, ..., 0.88594237, 0.88594237,
        0.88594237],
       [0.88594237, 0.88594237, 1.        , ..., 0.90000005, 0.90000005,
        0.90000005],
       ...,
       [0.88594237, 0.88594237, 0.90000005, ..., 1.        , 0.90000005,
        0.90000005],
       [0.88594237, 0.88594237, 0.90000005, ..., 0.90000005, 1.        ,
        0.90000005],
       [0.88594237, 0.88594237, 0.90000005, ..., 0.90000005, 0.90000005,
        1.        ]])
```

We can also see that for the converted matrix, it is very close to the original matrix.

2.3 Simulation Methods

In the lecture notes and the codes provided. There are several simulation methods. We converted these simulation methods to functions in our risk management library. For example, there are direct simulation, pca simulation. (There is also Monte-Carlo simulation and Historic Simulatons, but they are related to VaR/ES calculation, so we put them in the VaR/ES Calculation section)

Recall the exponentially weighted covariance we simulated in the first function group.

```
ew_cov = RM.exp_weighted_cov(DP_return)
pd.DataFrame(ew_cov)
```

✓ 0.1s

	0	1	2	3	4	5	6	7	8	
0	0.000155	0.000210	0.000235	0.000285	0.000279	0.000279	0.000284	0.000331	0.000368	0.0001
1	0.000210	0.000421	0.000321	0.000329	0.000526	0.000417	0.000417	0.000563	0.000527	0.0001
2	0.000235	0.000321	0.000499	0.000505	0.000438	0.000521	0.000529	0.000664	0.000610	0.0001
3	0.000285	0.000329	0.000505	0.000939	0.000542	0.000657	0.000671	0.000923	0.000696	0.0002
4	0.000279	0.000526	0.000438	0.000542	0.002369	0.000434	0.000444	0.000636	0.001212	0.0001
...	...	...	...	...	...	...	...	...	...	...
95	0.000034	0.000035	0.000032	-0.000016	0.000066	-0.000019	-0.000017	-0.000049	0.000013	0.0000
96	0.000171	0.000222	0.000224	0.000296	0.000252	0.000307	0.000307	0.000380	0.000468	0.0001
97	0.000209	0.000286	0.000297	0.000436	0.000666	0.000325	0.000330	0.000615	0.000536	0.0001

We can use direct simulation to simulate the targeted covariance.

```
pd.DataFrame(np.cov(RM.direct_simulation(ew_cov).T))
```
✓ 0.2s

	0	1	2	3	4	5	6	7	8	9	...
0	0.000157	0.000212	0.000236	0.000289	0.000282	0.000283	0.000288	0.000338	0.000371	0.000126	... 0
1	0.000212	0.000423	0.000324	0.000335	0.000529	0.000424	0.000423	0.000569	0.000529	0.000153	... 0
2	0.000236	0.000324	0.000502	0.000511	0.000438	0.000530	0.000538	0.000674	0.000609	0.000168	... 0
3	0.000289	0.000335	0.000511	0.000948	0.000542	0.000669	0.000684	0.000944	0.000701	0.000204	... 0
4	0.000282	0.000529	0.000438	0.000542	0.002348	0.000443	0.000452	0.000635	0.001208	0.000119	... 0
...	...	...	...	...	...	...	...	...	...	...	...
95	0.000034	0.000036	0.000033	-0.000014	0.000069	-0.000018	-0.000015	-0.000048	0.000014	0.000046	... 0
96	0.000174	0.000226	0.000227	0.000304	0.000262	0.000313	0.000313	0.000386	0.000474	0.000108	... 0
97	0.000212	0.000289	0.000298	0.000439	0.000663	0.000328	0.000333	0.000616	0.000537	0.000168	... 0
98	0.000163	0.000205	0.000195	0.000272	0.000184	0.000284	0.000290	0.000369	0.000307	0.000144	... 0

We have checked the result, and it is very close to the original exponentially weighted covariance matrix.

We can also use PCA to simulate this matrix.

```
pd.DataFrame(np.cov(RM.simulate_pca(ew_cov, 25000)))
```
✓ 0.1s

```
Simulating with 100 PC Factors: 100.00% total variance explained
```

	0	1	2	3	4	5	6	7	8	9	...
0	0.000157	0.000212	0.000237	0.000289	0.000288	0.000283	0.000288	0.000336	0.000375	0.000126	...
1	0.000212	0.000422	0.000323	0.000332	0.000531	0.000423	0.000422	0.000571	0.000532	0.000154	...
2	0.000237	0.000323	0.000499	0.000507	0.000448	0.000527	0.000535	0.000671	0.000617	0.000167	...
3	0.000289	0.000332	0.000507	0.000945	0.000561	0.000668	0.000683	0.000933	0.000709	0.000203	...
4	0.000288	0.000531	0.000448	0.000561	0.002368	0.000453	0.000463	0.000653	0.001224	0.000126	...
...	...	...	...	...	...	...	...	...	...	...	...
95	0.000034	0.000035	0.000032	-0.000015	0.000067	-0.000020	-0.000017	-0.000051	0.000015	0.000047	...
96	0.000175	0.000226	0.000232	0.000306	0.000267	0.000315	0.000316	0.000390	0.000480	0.000107	...
97	0.000212	0.000288	0.000301	0.000442	0.000674	0.000332	0.000337	0.000616	0.000541	0.000166	...

We can see that the result for PCA with variance explained 100% is very close to the result of direct simulation. We can also tune the number of eigenvalues included in the PCA simulation. For example, we can only include 10 eigenvalues in the PCA simulation. In this case, the simulated matrix is:

```
pd.DataFrame(np.cov(RM.simulate_pca(ew_cov, 25000, nval = 10)))
```
✓ 0.1s

```
Simulating with 10 PC Factors: 77.18% total variance explained
```

	0	1	2	3	4	5	6	7	8	9
0	0.000156	0.000208	0.000232	0.000287	0.000277	0.000282	0.000287	0.000340	0.000371	0.000125
1	0.000208	0.000338	0.000322	0.000342	0.000539	0.000425	0.000428	0.000585	0.000549	0.000151
2	0.000232	0.000322	0.000421	0.000506	0.000459	0.000522	0.000530	0.000670	0.000587	0.000167
3	0.000287	0.000342	0.000506	0.000829	0.000516	0.000674	0.000687	0.000931	0.000714	0.000211
4	0.000277	0.000539	0.000459	0.000516	0.002327	0.000420	0.000426	0.000638	0.001266	0.000110
...	...	...	...	...	...	...	...	...	...	...
95	0.000033	0.000041	0.000031	-0.000015	0.000064	-0.000034	-0.000031	-0.000050	0.000029	0.000044
96	0.000173	0.000234	0.000240	0.000297	0.000241	0.000306	0.000308	0.000366	0.000494	0.000118

We can further decrease the number of eigenvalues included. In this case, we decrease the number to 2. The simulated matrix is as follows:

```
pd.DataFrame(np.cov(RM.simulate_pca(ew_cov, 25000, nval = 2)))
```

✓ 0.1s Python

```
Simulating with 2 PC Factors: 54.29% total variance explained
```

	0	1	2	3	4	5	6	7	8	9	...	90
0	0.000150	0.000197	0.000217	0.000275	0.000282	0.000261	0.000266	0.000312	0.000381	0.000119	...	0.000169
1	0.000197	0.000270	0.000305	0.000400	0.000420	0.000385	0.000392	0.000516	0.000524	0.000147	...	0.000213
2	0.000217	0.000305	0.000349	0.000467	0.000498	0.000455	0.000461	0.000646	0.000593	0.000155	...	0.000226
3	0.000275	0.000400	0.000467	0.000643	0.000696	0.000632	0.000640	0.000962	0.000781	0.000183	...	0.000273
4	0.000282	0.000420	0.000498	0.000696	0.000762	0.000689	0.000697	0.001091	0.000823	0.000179	...	0.000270
...	...	...	...	...	...	...	...	...	...	...	...	...
95	0.000024	0.000016	0.000005	-0.000016	-0.000032	-0.000025	-0.000024	-0.000121	0.000027	0.000035	...	0.000044

We can see that the precision is generally lower than others.

2.4 & 2.5 VaR and ES Calculation

For simplicity, we integrate the calculation process of VaR and ES since they can be calculated in one function. For example, a 5% VaR is just the 5% percentile of the P&L, if we want to get the ES, we just need to average the 5% in the tail of the P&L. Therefore, they can be calculated in one function.

For normal distribution, the VaR and ES is as follows:

```
RM.cal_VaR_ES_norm(problem1)
```

✓ 0.0s

```
(0.0801701797195414,
 0.10118919638373398,
```

For normal distribution with exponentially weighted covariance with lambda equals to 0.94 (default), the VaR and ES is as follows:

```
RM.cal_VaR_ES_ew_norm(problem1)
```

✓ 0.0s

```
(0.09096711648746494,
 0.11130460050658254,
```

For fitted t distribution, the VaR and ES is as follows:

```
RM.cal_VaR_ES_MLE_t(problem1)
```

✓ 0.0s

```
(0.07851107644018746,
 0.11583688492893608,
```

For fitted AR(1) model, the VaR and ES is as follows:

```
RM.cal_VaR_ES_AR1(problem1)
✓ 1.7s
(0.0808619927437214,
 0.10121299672724171,
```

For historical simulation method, the VaR and ES is as follows:

```
RM.cal_VaR_ES_hist(problem1.values)
✓ 0.1s
Output exceeds the size limit. Open the f
(array([0.07586151]),
 0.11677669788562187,
```

We have checked that the output is correct and match our expectations based on comparing the previous assignments.

Problem 3

Use your repository from #2.

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.

Fit a Generalized T model to each stock and calculate the VaR and ES of each portfolio as well as your total VaR and ES. Compare the results from this to your VaR form Problem 3 from Week 4.

The question asks us to fit a generalized t model to each stock and calculate the VaR and ES of each portfolio/total portfolio.

After fitting each stock with a t distribution, we can get the result. We can see that:

```
For Portfolio A,
Total Value is 299950.05907389
VaR is 8025.1991420117
ES is 10549.971150509953
For Portfolio B,
Total Value is 294385.59081765
VaR is 6788.831463675352
ES is 8884.435972928033
For Portfolio C,
Total Value is 270042.8305277
VaR is 5690.082249102618
ES is 7483.266302450677
For Portfolio all,
Total Value is 864378.48041924
VaR is 20343.18061139258
ES is 26630.368346637602
```

1. for portfolio A, the VaR is 8025.20, while the ES is 10549.97.
2. for portfolio A, the VaR is 6788.83, while the ES is 8884.44.
3. for portfolio A, the VaR is 5690.08, while the ES is 7483.27.
4. for the total portfolio, the VaR is 20343.18, while the ES is 26630.37.

For the simulated data, the result is different from the result in Problem 3 Week 4. We can see that the simulated result in this week is generally larger than the result last week. For example, the result using monte-carlo simulations with exponentially weighted covariance with lambda = 0.97 is smaller than this. However, the result using monte-carlo method with original covariance matrix is very similar to the result this week, though it is also smaller than the result this week. It could be explained easily since t distribution has fatter tail than multivariate normal, therefore, the VaR and ES should be larger.