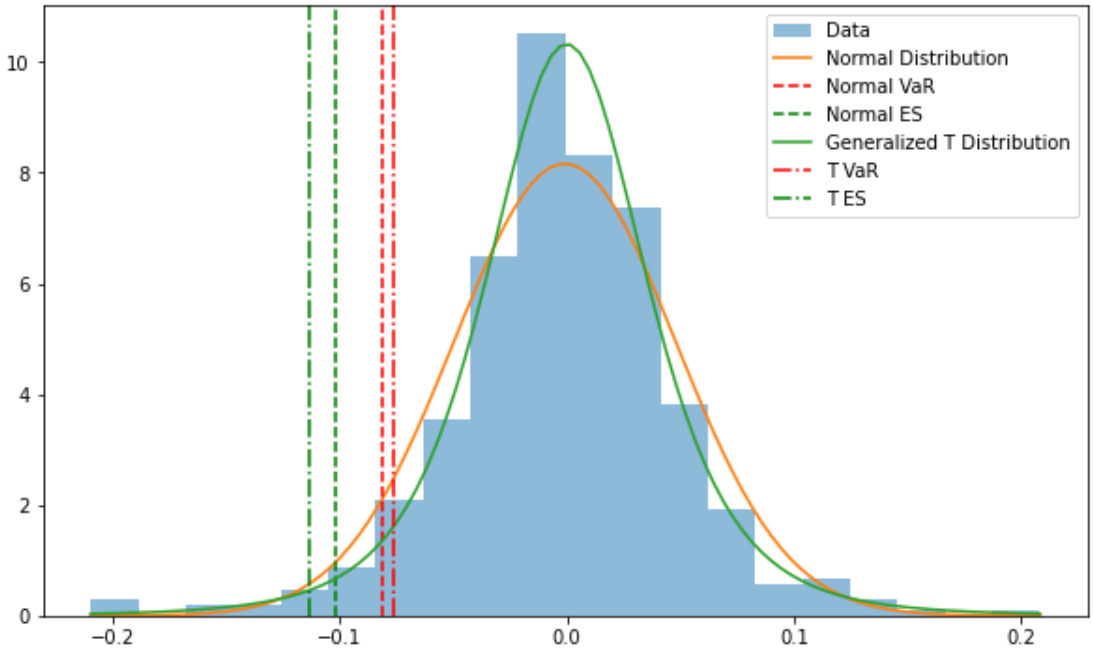
Problem 1

I use stats.norm and stats.t from scipy to fit the data into normal and generalized T distribution. Then I calculate the VaR and ES by definition (Expected Shortfall is the average of the left tail of the distribution, where the tail is defined as values whose CDF is less than or equal to Alpha):

- Normal Distribution:
  - VaR: 0.08
  - ES: 0.10
- Generalized T Distribution:
  - VaR: 0.08
  - ES: 0.11

I plot PDFs and VaR/ES values for both distributions:



Problem 2

In this problem, I test exp_weighted_cov, near_psd, chol.psd, and direct_simulation, and they work out as expected. The function of calculating VaR and ES will be tested in the next problem.

Problem 3

I first initialize an empty dictionary t_param and an empty list sim_data. It then loops through each column in the return_data dataframe and performs the following steps:

1. Subtract the mean of the column from each element of the column to center the distribution around zero.
2. Fit a Student's t-distribution to the centered column using the t.fit method from the scipy.stats module. This returns the degrees of freedom df, the location parameter loc, and the scale parameter scale for the fitted t-distribution.
3. Store the t-distribution parameters in the t_param dictionary.
4. Generate 10,000 random samples from the fitted t-distribution using the t.rvs method from the scipy.stats module with the df, loc, and scale parameters as input. Append the simulated data to the sim_data list.

After simulating the data for each stock, I convert the sim_data list to a numpy array and transposes it to create a dataframe sim_returns with 10,000 rows (for the 10,000 simulated scenarios) and 100 columns (for the 100 stocks).

I then initialize a dataframe current_prices with the most recent prices of each stock in the dailyPrice dataframe. For each portfolio in the portfolio dataframe, the code performs the following steps:

1. Sets the index of the port dataframe to be the stock symbols.
2. Joins the port dataframe with the current_prices dataframe to get the most recent prices for each stock in the portfolio.
3. Multiplies the simulated returns for each stock in the portfolio by the corresponding stock price to get the simulated price changes for each stock in each scenario.
4. Calculates the simulated portfolio values for each scenario by multiplying the simulated price changes by the portfolio holdings and summing across all stocks in the portfolio.
5. Calculates the VaR and ES for the simulated portfolio values using the var.calculate_var and var.calculate_es functions from the var.py module, respectively.
6. Prints the VaR and ES for the current portfolio.

This result is lower than my VaR from Problem 3 from Week 4.