Data preparation

In [1]:

```
import numpy as np
# pip install pandas_datareader
import pandas_datareader as web
import pandas as pd
# import data
df = web.get_data_yahoo("SBUX", start = "2011-01-01", end = "2020-12-31", interv
df.reset_index(inplace = True) # convert index into a column
df['Date'] = pd.to_datetime(df['Date']) # convert the strings to dates

df['cc'] = np.log(df['Adj Close']/df['Adj Close'].shift(1))
df
```

Out[1]:

	Date	High	Low	Open	Close	Volume	Adj Close	c
0	2011-01-03	16.709999	16.230000	16.245001	16.625000	12764600.0	13.786985	Na
1	2011-01-04	16.645000	16.219999	16.625000	16.240000	13306000.0	13.467708	-0.02343
2	2011-01-05	16.420000	16.125000	16.129999	16.174999	11501800.0	13.413807	-0.00401
3	2011-01-06	16.250000	15.895000	16.184999	15.980000	13253400.0	13.252091	-0.01212
4	2011-01-07	16.430000	15.930000	16.020000	16.389999	19791400.0	13.592102	0.02533
...	...	...	...	...	...	...	...	.
2512	2020-12-24	102.360001	101.680000	102.300003	102.010002	1949200.0	99.844879	-0.00049
2513	2020-12-28	104.379997	102.309998	102.919998	104.339996	5055200.0	102.125427	0.02258
2514	2020-12-29	105.779999	104.470001	104.889999	105.629997	4780900.0	103.388046	0.01228
2515	2020-12-30	106.620003	105.779999	105.989998	105.970001	3654100.0	103.720833	0.00321
2516	2020-12-31	107.139999	105.620003	106.000000	106.980003	3566300.0	104.709404	0.00948

2517 rows × 8 columns

In [2]:

```
cc = df['cc'][1:]
T = len(cc)

print(cc)
```

```
1       -0.023430
2       -0.004010
3       -0.012129
```

```
4       0.025334
5      -0.000305
          ...
2512   -0.000490
2513    0.022584
2514    0.012288
2515    0.003214
2516    0.009486
Name: cc, Length: 2516, dtype: float64
```

Bootstrap SE and Moments Estimators

Bootstrap samples

Suppose the mean of cc returns $\{r_t\}_{t=1}^T \overset{i.i.d.}{\sim} N(\mu, \sigma^2)$. Both parameters are invariant to time.

A boostrap sample results from drawing sample data repeatedly **with replacement** from the original data.

In [3]:

```
np.random.choice(cc, replace = True, size = T) # it gives a bootstap sample with
```

Out[3]:

```
array([-0.01038018,  0.01722806,  0.01096674, ...,  0.01873363,
        0.03656849, -0.00058114])
```

We generate 50,000 bootstrap samples.

In [4]:

```
np.random.seed(123) # for reproducible randomness
B = 50000
boot_samples = np.zeros((T, B))
for i in range(B):
    boot_samples[:, i] = np.random.choice(cc, replace = True, size = T) # creat

boot_samples[:, 0:5] # first 5 bootstrap samples (first 5 columns of boot_sample
```

Out[4]:

```
array([[-0.02216055,  0.02124918, -0.00141245,  0.00014023,  0.04141245],
       [-0.00055629, -0.01474538,  0.05571364, -0.00036652,  0.00149592],
       [ 0.01075645, -0.00588539, -0.01355654,  0.0065586 ,  0.00370512],
       ...,
       [ 0.00668364,  0.00182617,  0.00062334,  0.00082184,  0.01598616],
       [ 0.00347644,  0.00322185,  0.00428967, -0.00734845, -0.02217051],
       [ 0.01020658,  0.06029376, -0.00398252,  0.00075013,  0.0019643 ]])
```

We can estimate μ and σ by the original sample mean and standard deviation:

$$\hat{\mu} = \frac{1}{T}\sum_{t=1}^{T} r_t$$

$$\hat{\sigma} = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(r_t - \hat{\mu})^2}$$

We can caculate the bootstrap means and sample standard deviations: $\hat{\mu}^{*,b}$ and $\hat{\sigma}^{*,b}, b = 1, \ldots, 50000$.

In [5]:

```
mu_hat, sig_hat = np.mean(cc), np.std(cc, ddof = 1)
print("The sample mean and standard deviation are {:.4f} and {:.4f}, resp.".form
# {} refers to a variable, and :.4f rounds the number off to 4 decimal places
print("=====================================================================")

boot_means = np.mean(boot_samples, axis = 0) # apply function np.mean() to boots
boot_stds = np.std(boot_samples, ddof = 1, axis = 0) # apply function np.std() w

print('First 5 bootstrap sample means: ', np.round(boot_means[:5], 4)) # round c
print('First 5 bootstrap sample standard deviations: ', np.round(boot_stds[:5],
```

```
The sample mean and standard deviation are 0.0008 and 0.0165, resp.
======================================================================
First 5 bootstrap sample means:  [0.0007 0.001  0.0008 0.0014 0.0006]
First 5 bootstrap sample standard deviations:  [0.0159 0.0162 0.0163 0.0163 0.01
6 ]
```

Use Bootstrap to evaluate the bias, variance and MSE of sample mean $\hat{\mu}$

The **bias** of $\hat{\mu}$ and bootstrap estimator of bias are:

$$\begin{aligned}
&\text{Bias}(\hat{\mu}) = E[\hat{\mu} - \mu]. \\
&\text{Bias}_{bt}(\hat{\mu}) = \overline{\hat{\mu}^*} - \hat{\mu}, \text{ where } \overline{\hat{\mu}^*} = B^{-1}\sum_{b=1}^{B}\hat{\mu}^{*,b}
\end{aligned}$$

The **MSE** of $\hat{\mu}$ and bootstrap estimator of MSE are:

$$\begin{aligned}
&\text{MSE}(\hat{\mu}) = E\left[(\hat{\mu} - \mu)^2\right]. \\
&\text{MSE}_{bt}(\hat{\mu}) = B^{-1}\sum_{b=1}^{B}\left(\hat{\mu}^{b,*} - \hat{\mu}\right)^2
\end{aligned}$$

The **variance** of $\hat{\mu}$ and SE are:

$$\begin{aligned}
Var(\hat{\mu}) &= Var(\frac{1}{T}\sum_{t=1}^{T} r_t) = \frac{1}{T^2}\sum_{t=1}^{T} Var(r_t) = \frac{\sigma^2}{T} \\
SD(\hat{\mu}) &= \sqrt{Var(\hat{\mu})} = \frac{\sigma}{\sqrt{T}} \\
SE(\hat{\mu}) &= \frac{\hat{\sigma}}{\sqrt{T}} \quad \text{(plug-in estimator)} \\
\text{SE}_{bt} &= \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}\left(\hat{\mu}^{*,b} - \overline{\hat{\mu}^*}\right)^2} \\
\text{SE}_{q-bt} &= \frac{\hat{\mu}^*_{[3/4]} - \hat{\mu}^*_{[1/4]}}{z_{3/4} - z_{1/4}}
\end{aligned}$$

.

where $\hat{\sigma}$ is the sample standard deviation, z_α is the α quantile of $N(0, 1)$, and $\hat{\mu}^*_{[\alpha]}$ is the α quantile of $\left\{\hat{\mu}^{*,b}\right\}_{b=1}^{B}$.

In [6]:

```
from scipy.stats import iqr, norm

# focus on mu_hat

B_Bias = np.mean(boot_means) - mu_hat

B_MSE = np.mean((boot_means - mu_hat) ** 2)

SE = sig_hat / np.sqrt(T)
# in what follows, work with the bootstrap sample means / estimates of mu
B_SE = np.std(boot_means, ddof = 1)

IQR_SE = iqr(boot_means)/(norm.ppf(0.75) - norm.ppf(0.25))

print('mu_hat: {:.4f}'.format(mu_hat))
print('B_Bias: {:.4f}'.format(B_Bias))
print('B_MSE:  {:.4f}'.format(B_MSE))
print('SE:     {:.4f}'.format(SE))
print('B_SE:   {:.4f}'.format(B_SE))
print('IQR_SE: {:.4f}'.format(IQR_SE))
```

```
mu_hat: 0.0008
B_Bias: -0.0000
B_MSE:  0.0000
SE:     0.0003
B_SE:   0.0003
IQR_SE: 0.0003
```

However, the SE of an estimator $\hat{\theta}$ is not always easy to derive. But boostrap SEs are easy to construct.

In [7]:

```
# focus on sigma_hat

B_Bias_sig = np.mean(boot_stds) - sig_hat

B_MSE_sig = np.mean((boot_stds - sig_hat) ** 2)

# in what follows, work with the bootstrap sample standard deviations / estimate
B_SE_sig = np.std(boot_stds, ddof = 1)

IQR_SE_sig = iqr(boot_stds)/(norm.ppf(0.75) - norm.ppf(0.25))

print('sig_hat: {:.4f}'.format(sig_hat))
print('B_Bias: {:.4f}'.format(B_Bias_sig))
print('B_MSE:  {:.4f}'.format(B_MSE_sig))
print('B_SE:   {:.4f}'.format(B_SE_sig))
print('IQR_SE: {:.4f}'.format(IQR_SE_sig))
```

```
sig_hat: 0.0165
B_Bias: -0.0000
B_MSE:  0.0000
B_SE:   0.0007
IQR_SE: 0.0007
```

Bootstrap Inference

Using bootstrap SE (and based on asymptotic normal approximation)

Recall that a $(1-\alpha)$ Confidence Interval for $\hat{\mu}$ based on asymptotic normal approximation is:

$$\left[\hat{\mu} - \mathrm{SE}(\hat{\mu}) z_{1-\alpha/2}, \quad \hat{\mu} + \mathrm{SE}(\hat{\mu}) z_{1-\alpha/2}\right]$$

A natural thought is to plug in bootstrap SE estimators to obtain

$$\left[\hat{\mu} - \mathrm{SE}_{bt}(\hat{\mu}) z_{1-\alpha/2}, \quad \hat{\mu} + \mathrm{SE}_{bt}(\hat{\mu}) z_{1-\alpha/2}\right].$$

A more reliable SE estimator using bootstrap is based on the quantiles of $\{\hat{\mu}^{*,b}\}_{b=1}^{B}$:

$$\left[\hat{\mu} - \mathrm{SE}_{q-bt}(\hat{\mu}) z_{1-\alpha/2}, \quad \hat{\mu} + \mathrm{SE}_{q-bt}(\hat{\mu}) z_{1-\alpha/2}\right]$$

In [8]:

```
alpha = 0.05
CI = [mu_hat - SE*norm.ppf(1-alpha/2), mu_hat + SE*norm.ppf(1-alpha/2)]
print("The 95% CI using the asymptotic SE is [{:.6f}, {:.6f}]" .format(CI[0], CI

CI_bt = [mu_hat - B_SE*norm.ppf(1-alpha/2), mu_hat + B_SE*norm.ppf(1-alpha/2)]
print("The 95% CI using the bootstrap SE is [{:.6f}, {:.6f}]" .format(CI_bt[0],

CI_qbt = [mu_hat - IQR_SE*norm.ppf(1-alpha/2), mu_hat + IQR_SE*norm.ppf(1-alpha/
print("The 95% CI using the IQR SE is [{:.6f}, {:.6f}]" .format(CI_qbt[0], CI_qb
```

```
The 95% CI using the asymptotic SE is [0.000162, 0.001450]
The 95% CI using the bootstrap SE is [0.000160, 0.001452]
The 95% CI using the IQR SE is [0.000164, 0.001447]
```

Bootstrap Percentile CIs

Actually, we don't even need to get SE estimators to construct CIs. We can construct CIs by directly employing the bootstrap quantiles.

Based on $P(|\hat{\mu} - \mu| \leq q_{T,1-\alpha}) = 1 - \alpha$, we can construct the **symmetric** bootstrap percentile CI:

$$[\hat{\mu} - q^*_{T,1-\alpha}, \quad \hat{\mu} + q^*_{T,1-\alpha}]$$

where $q^*_{T,1-\alpha}$ is the $(1-\alpha)$ quantile of $\left\{|\hat{\mu}^{*,b} - \hat{\mu}|\right\}_{b=1}^{B}$.

Based on $P(c_{T,\alpha/2} \leq \mu - \hat{\mu} \leq c_{T,1-\alpha/2}) = 1 - \alpha$, we can construct the **equal-tail** bootstrap percentile CI:

$$[\hat{\mu} + c^*_{T,\alpha/2}, \quad \hat{\mu} + c^*_{T,1-\alpha/2}]$$

where $c^*_{T,\alpha/2}$ is the $\alpha/2$ quantile of $\left\{\hat{\mu} - \hat{\mu}^{*,b}\right\}_{b=1}^{B}$

In [9]:

```
q_et_1 = np.quantile(mu_hat - boot_means, 0.025)
q_et_2 = np.quantile(mu_hat - boot_means, 0.975)
q_sym = np.quantile(np.abs(boot_means - mu_hat), 0.95)

CI_et = [mu_hat + q_et_1, mu_hat + q_et_2]
print("The 95% equal-tail bootstrap CI is [{:.6f}, {:.6f}]" .format(CI_et[0], CI

CI_sym = [mu_hat - q_sym, mu_hat + q_sym]
print("The 95% symmetric bootstrap CI is [{:.6f}, {:.6f}]" .format(CI_sym[0], CI
```

```
The 95% equal-tail bootstrap CI is [0.000162, 0.001460]
The 95% symmetric bootstrap CI is [0.000157, 0.001455]
```

Estimation of VaR

Let $L_1 = W_1 - W_0$ be the profit of the investment, where W_0 is the initial wealth. We have

$$L_1 = W_0(e^r - 1),$$

where r is the cc return. The VaR_α is the α-quantile of L_1 and

$$VaR_\alpha = W_0(e^{q_\alpha^r} - 1),$$

where q_α^r is the α-quantile of r.

Suppose $r \sim N(\mu, \sigma^2)$, where μ and σ^2 are unknown. We will use the SBUX data to estimate the unknown parameters and construct the parametric VaR estimator.

In [10]:

```
mu_hat, sig_hat = norm.fit(cc) # maximum likelihood estimates
print(np.round([mu_hat, sig_hat], 4))
```

```
[0.0008 0.0165]
```

The maximum likelihood estimate $\hat{\mu}_{MLE}$ is the same as sample mean, while the maximum likelihood estimate $\hat{\sigma}_{MLE} = \sqrt{\sum_{t=1}^T (r_t - \hat{\mu})^2 / T}$ is close to sample standard deviation $\hat{\sigma} = \sqrt{\sum_{t=1}^T (r_t - \hat{\mu})^2 / (T-1)}$. Besides, $\hat{\sigma}_{MLE}$ is biased but consistent.

The parametric VaR_α estimator is $W_0(e^{\hat{q}_\alpha^r} - 1)$, where $\hat{q}_\alpha^r$ is the α-quantile of $N(\hat{\mu}, \hat{\sigma}^2)$.

In [11]:

```
W0 = 10000
alpha = 0.1
L1 = W0 * (np.exp(cc) - 1)
VaR_Para_Est = W0 * (np.exp(norm.ppf(alpha, loc = mu_hat, scale = sig_hat)) - 1)
# note the alpha quantile of N(mu_hat, sig_hat^2)
print("The parametric estimate of VaR(0.1) is %.4f" % VaR_Para_Est)
```

```
The parametric estimate of VaR(0.1) is -201.0891
```

Next, we construct the nonparametric VaR estimator.

In [12]:

```
VaR_NonP_Est = W0 * (np.exp(np.quantile(cc, alpha)) - 1)
print("The nonparametric estimate of VaR(0.1) is %.4f" % VaR_NonP_Est)
```

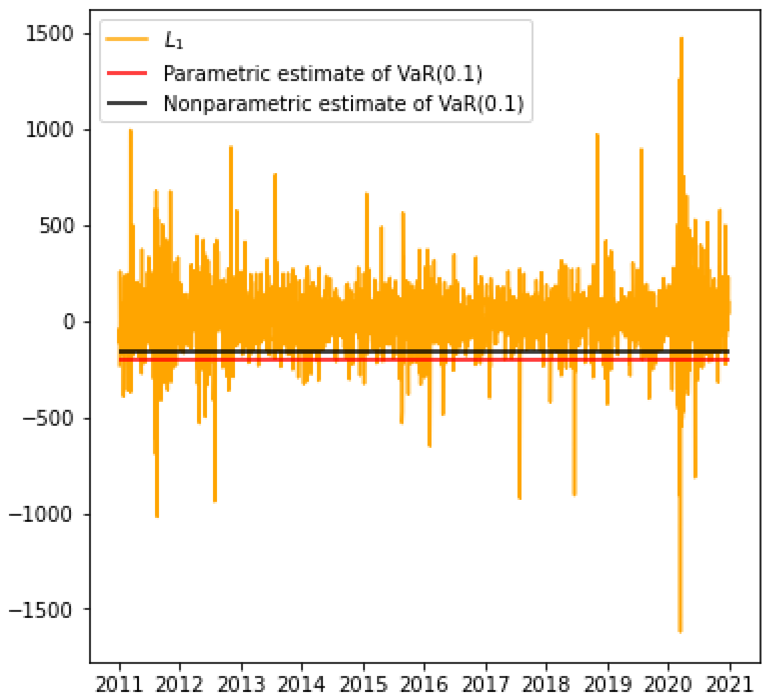
```
The nonparametric estimate of VaR(0.1) is -155.2387
```

In [13]:
```python
np.quantile(L1, alpha) # same result; sample qunatile of L1
```

Out[13]:
```
-155.2387207189759
```

In [14]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize = (6, 6))
plt.plot(df['Date'][1:], L1, color = 'orange', label = r'$L_1$')
plt.hlines(y = VaR_Para_Est, xmin = df['Date'].iloc[1], xmax = df['Date'].iloc[-
           label = 'Parametric estimate of VaR(0.1)')
plt.hlines(y = VaR_NonP_Est, xmin = df['Date'].iloc[1], xmax = df['Date'].iloc[-
          label = 'Nonparametric estimate of VaR(0.1)')
plt.legend()
plt.show()
```



Inference on VaR

We consider the bootstrap inference using the nonparametric VaR estimator.

In [15]:
```python
def VaR_np(y, p = alpha, W0 = W0):
    mu = np.mean(y)
    q = np.quantile(y, p)
    return W0 * (np.exp(q) - 1)

boot_VaR_Est = np.zeros(B)
for i in range(B):
    boot_VaR_Est[i] = VaR_np(boot_samples[:, i]) # calculate VaR(0.1) for each b

B_Bias = np.mean(boot_VaR_Est) - VaR_NonP_Est
B_MSE = np.mean((boot_VaR_Est - VaR_NonP_Est) ** 2)
```

```
# in what follows, work with the bootstrap sample VaR_est
B_SE = np.std(boot_VaR_Est, ddof = 1)

IQR_SE = iqr(boot_VaR_Est)/(norm.ppf(0.75) - norm.ppf(0.25))

print('VaR_hat: {:.4f}'.format(VaR_NonP_Est))
print('B_Bias:  {:.4f}'.format(B_Bias))
print('B_MSE:   {:.4f}'.format(B_MSE))
print('B_SE:    {:.4f}'.format(B_SE))
print('IQR_SE:  {:.4f}'.format(IQR_SE))
```

```
VaR_hat: -155.2387
B_Bias:  1.3375
B_MSE:   27.9215
B_SE:    5.1121
IQR_SE:  3.9852
```

The bootstrap CIs for $VaR_{0.1}$ are:

In [16]:

```
CI_bt = [VaR_NonP_Est - B_SE*norm.ppf(1-alpha/2), VaR_NonP_Est + B_SE*norm.ppf(1
print("The 95% CI using the bootstrap SE is  [{:.6f}, {:.6f}]" .format(CI_bt[0],

CI_qbt = [VaR_NonP_Est - IQR_SE*norm.ppf(1-alpha/2), VaR_NonP_Est + IQR_SE*norm.
print("The 95% CI using the IQR SE is        [{:.6f}, {:.6f}]" .format(CI_qbt[0]

q_et_1 = np.quantile(VaR_NonP_Est - boot_VaR_Est, 0.025)
q_et_2 = np.quantile(VaR_NonP_Est - boot_VaR_Est, 0.975)
q_sym = np.quantile(np.abs(boot_VaR_Est - VaR_NonP_Est), 0.95)

CI_et = [VaR_NonP_Est + q_et_1, VaR_NonP_Est + q_et_2]
print("The 95% equal-tail bootstrap CI is    [{:.6f}, {:.6f}]" .format(CI_et[0],

CI_sym = [VaR_NonP_Est - q_sym, VaR_NonP_Est + q_sym]
print("The 95% symmetric bootstrap CI is     [{:.6f}, {:.6f}]" .format(CI_sym[0]
```

```
The 95% CI using the bootstrap SE is  [-163.647322, -146.830120]
The 95% CI using the IQR SE is        [-161.793728, -148.683713]
The 95% equal-tail bootstrap CI is    [-168.414669, -147.888652]
The 95% symmetric bootstrap CI is     [-167.239938, -143.237504]
```

Estimation of ES

The expected shortfall is

$$ES_\alpha = E[L_1|L_1 \le VaR_\alpha] = \frac{E[L_1 1\{L_1 \le VaR_\alpha\}]}{E[1\{L_1 \le VaR_\alpha\}]} = \alpha^{-1}\int_0^\alpha VaR(u)du.$$

The parametric ES estimator is

$$\widehat{ES}_\alpha^{para} = E[L_1|L_1 \le \widehat{VaR}_\alpha^{para}] = \frac{E[L_1 1\{L_1 \le \widehat{VaR}_\alpha^{para}\}]}{E[1\{L_1 \le \widehat{VaR}_\alpha^{para}\}]}.$$

In [17]:

```
import scipy.integrate as integrate

def VaR(alpha, W0 = W0, mu = mu_hat, sigma = sig_hat):
```

```
    q_r = norm.ppf(alpha, mu, sigma)
    return W0 * (np.exp(q_r) - 1)

ES_Para_Est = integrate.quad(VaR, 0, alpha)[0] / alpha
print("The parametric estimate of ES(0.1) is %.4f" % ES_Para_Est)

ind = (L1 <= VaR_NonP_Est) * 1
ES_NonP_Est = np.mean(L1 * ind) / np.mean(ind)
print("The nonparametric estimate of ES(0.1) is %.4f" % ES_NonP_Est)
```

```
The parametric estimate of ES(0.1) is -277.0206
The nonparametric estimate of ES(0.1) is -276.6078
```

Inference on ES

We consider the bootstrap inference using the nonparametric ES estimator.

In [18]:

```
def ES_np(y, p=alpha, W0 = W0):
    mu = np.mean(y)
    q = np.quantile(y, p)
    VaR_np = W0 * (np.exp(q) - 1)
    L_1 = W0 * (np.exp(y) - 1)
    I_1 = (L_1 <= VaR_np) * 1
    return np.mean(L_1 * I_1)/np.mean(I_1)

boot_ES_Est = np.zeros(B)
for i in range(B):
    boot_ES_Est[i] = ES_np(boot_samples[:, i]) # calculate VaR(0.1) for each boo

B_Bias = np.mean(boot_ES_Est) - ES_NonP_Est
B_MSE = np.mean((boot_ES_Est - ES_NonP_Est) ** 2)
# in what follows, work with the bootstrap sample VaR_est
B_SE = np.std(boot_ES_Est, ddof = 1)

IQR_SE = iqr(boot_ES_Est)/(norm.ppf(0.75) - norm.ppf(0.25))

print('ES_hat: {:.4f}'.format(ES_NonP_Est))
print('B_Bias: {:.4f}'.format(B_Bias))
print('B_MSE:  {:.4f}'.format(B_MSE))
print('B_SE:   {:.4f}'.format(B_SE))
print('IQR_SE: {:.4f}'.format(IQR_SE))
```

```
ES_hat: -276.6078
B_Bias: 0.4367
B_MSE:  168.5644
B_SE:   12.9760
IQR_SE: 12.9530
```

The bootstrap CIs for $ES_{0.1}$ are:

In [19]:

```
CI_bt = [ES_NonP_Est - B_SE*norm.ppf(1-alpha/2), ES_NonP_Est + B_SE*norm.ppf(1-a
print("The 95% CI using the bootstrap SE is  [{:.6f}, {:.6f}]" .format(CI_bt[0],

CI_qbt = [ES_NonP_Est - IQR_SE*norm.ppf(1-alpha/2), ES_NonP_Est + IQR_SE*norm.pp
print("The 95% CI using the IQR SE is        [{:.6f}, {:.6f}]" .format(CI_qbt[0]

q_et_1 = np.quantile(ES_NonP_Est - boot_ES_Est, 0.025)
```

```python
q_et_2 = np.quantile(ES_NonP_Est - boot_ES_Est, 0.975)
q_sym = np.quantile(np.abs(boot_ES_Est - ES_NonP_Est), 0.95)

CI_et = [ES_NonP_Est + q_et_1, ES_NonP_Est + q_et_2]
print("The 95% equal-tail bootstrap CI is    [{:.6f}, {:.6f}]" .format(CI_et[0],

CI_sym = [ES_NonP_Est - q_sym, ES_NonP_Est + q_sym]
print("The 95% symmetric bootstrap CI is     [{:.6f}, {:.6f}]" .format(CI_sym[0]
```

```
The 95% CI using the bootstrap SE is  [-297.951407, -255.264097]
The 95% CI using the IQR SE is        [-297.913605, -255.301900]
The 95% equal-tail bootstrap CI is    [-301.286402, -250.358267]
The 95% symmetric bootstrap CI is     [-301.995358, -251.220146]
```