Fintech545 Assignment3

October 1, 2022

```
[illegible]
```

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.

1.1 Calculate the mean and variance theoretically

1. Classical Brownian Motion

$P_t = P_{t-1} + r_t$

$P_t = P_0 + \sum r_t$

$E(P_t) = E(P_0 + \sum r_t)$

$E(P_t) = P_0$

$var(P_t) = var(P_0 + r_t)$

$var(P_t) = t\sigma^2$

2. Arithmetic Return System

$P_t = P_{t-1}(1 + r_t)$

$E(P_t) = E(P_{t-1} + P_{t-1}r_t)$

$E(P_t) = E(P_{t-1})$

$E(P_t) = P_{t-1}$

$var(P_t) = var(P_{t-1} + P_{t-1}r_t)$

$var(P_t) = P_{t-1}^2\sigma^2$

3. Geometric Brownian Motion

$\ln(P_t) = \ln(P_{t-1}) + r_t$

$E(\ln(P_t)) = E(\ln(P_{t-1}) + r_t)$

$E(\ln(P_t)) = E(\ln(P_{t-1}))$

$E(\ln(P_t)) = \ln(P_0)$

$var(\ln(P_t)) = var(\ln(P_0) + r_t)$

$var(\ln(P_t)) = \sigma^2$

$var(\ln(P_t)) = t\sigma^2$

1.2 Do the simulation

Implement the simulation functions for 3 types of price returns. Return the expected value and variance of simulations.

```
[illegible]
```

Compare the simulation results and theoretically results

```
[illegible]
```

The results of the simulation and theoretical calculations matched.

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 INTC.csv. Calculate the arithmetic returns for INTC. Remove the mean from the series so that the mean(INTC)=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 Historic Simulation.

Compare the 4 values. Look at the empirical distribution of returns, in sample.

Download from Yahoo! Finance the prices since the end of the data in the CSV file (about 3 months). Look at the empirical distribution of returns, out of sample.

Discuss the ability of these models to describe the risk in this stock.

Implement "return_calculate()"

```
[illegible]
```

Read INTC data

```
[illegible]
```

[illegible]

Calculate the arithmetic returns for INTC and remove the mean

```
[illegible]
```

1. Calculate VaR using a normal distribution

```
[illegible]
```

2. Calculate VaR using a normal distribution with an Exponentially Weighted variance (λ=0.94)

```
[illegible]
    weight = weight/sum(weight)
    norm_data = data - data.mean()
    return norm_data.T @ (np.diag(weight) @ norm_data)
var = expo_weighted_cov(returns,0.94)
sim2 = np.random.normal(0,np.sqrt(var),size=10000)
-np.percentile(sim2,5)
```

Out[66]: 0.03931295126922911

3. Calculate VaR using a MLE fitted T distribution

In [67]:
```
from scipy import stats
from scipy.optimize import minimize
def MLE_T(p):
   return -1*np.sum(stats.t.logpdf(returns, df=p[0], loc = p[1],scale=p[2]))
constraints=({"type":"ineq", "fun":lambda x: x[0]-1},
                 {"type":"ineq", "fun":lambda x: x[2]})
df, loc, scale = minimize(MLE_T, x0 = (10,np.mean(returns),np.std(returns)),constraints=constraints).x

sim3= stats.t(df=df, scale=scale).rvs(10000)
-np.percentile(sim3,5)
```

Out[67]: 0.03318426205106343

4. Calculate VaR using a Historic Simulation

In [68]:
```
-np.percentile(returns,5)
```

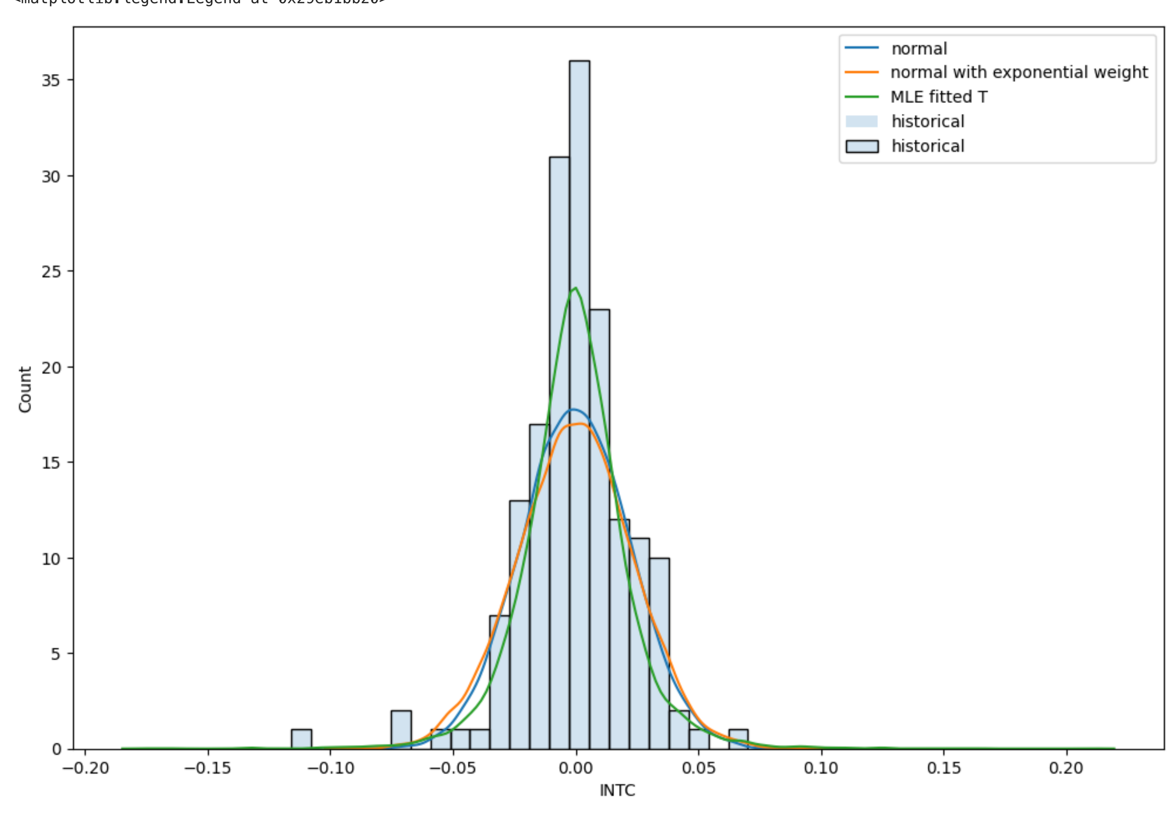
Out[68]: 0.029574903865632305

Draw historical distributions

In [69]:
```
f, ax = plt.subplots(1, 1, figsize =(12,8))
sns.histplot(returns, ax = ax, label='historical', alpha = 0.2)
sns.distplot(sim1, ax = ax, hist=False, label='normal')
sns.distplot(sim2, ax = ax, hist=False, label='normal with exponential weight')
sns.distplot(sim3, ax = ax, hist=False, label='MLE fitted T')
ax.legend()
```

```
/var/folders/1n/6zcygvs12455myprvytb14680000gn/T/ipykernel_12623/1985991635.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sim1, ax = ax, hist=False, label='normal')
/var/folders/1n/6zcygvs12455myprvytb14680000gn/T/ipykernel_12623/1985991635.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sim2, ax = ax, hist=False, label='normal with exponential weight')
/var/folders/1n/6zcygvs12455myprvytb14680000gn/T/ipykernel_12623/1985991635.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sim3, ax = ax, hist=False, label='MLE fitted T')
```

Out[69]: <matplotlib.legend.Legend at 0x29eb1bb20>



None of them fits very well with the historical data. The best fit distribution is MLE fitted T.

Download data from Yahoo Finance

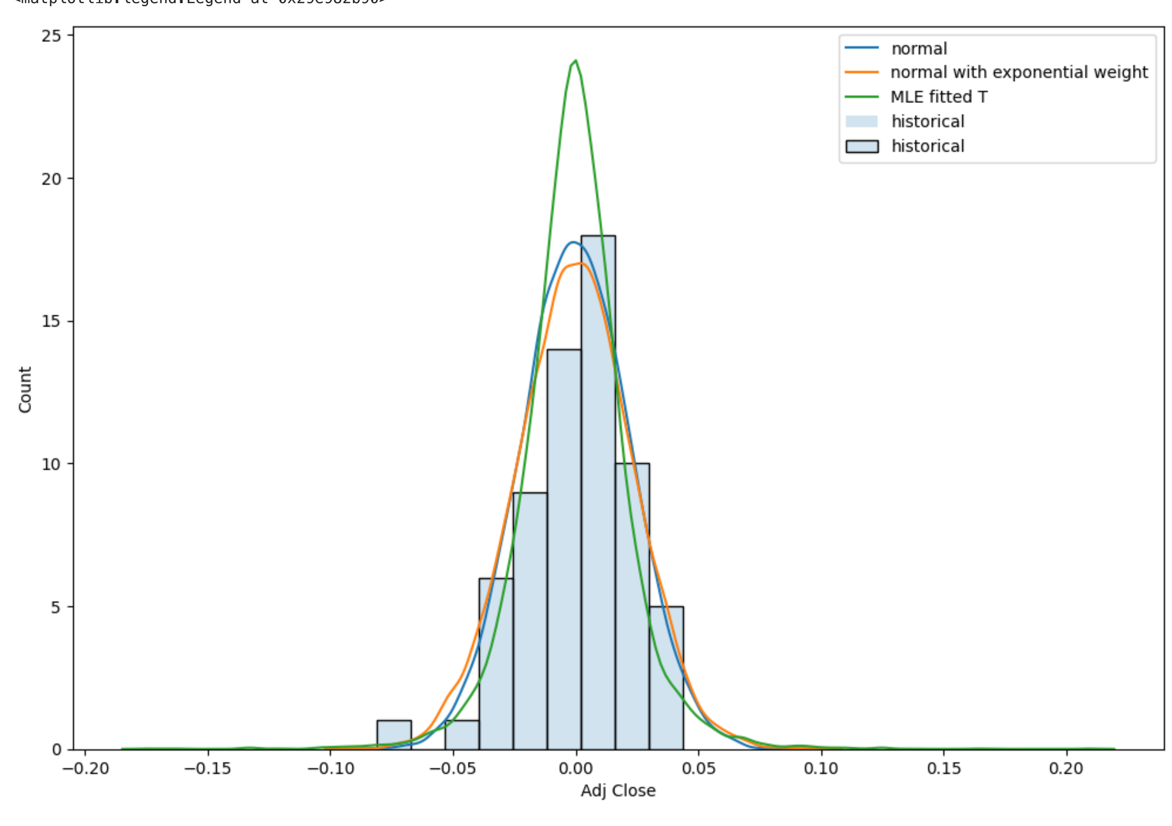
In [70]:
```
import yfinance as yf
intc_df = yf.download('INTC', start='2022-05-31', end='2022-09-01', progress=False)
f_returns = return_calculate(intc_df["Adj Close"])
f_returns -= f_returns.mean()
```

Draw future distributions

In [71]:
```
f, ax = plt.subplots(1, 1, figsize = (12,8))
sns.histplot(f_returns, ax = ax, label='historical', alpha = 0.2)
sns.distplot(sim1, ax = ax, hist=False, label='normal')
sns.distplot(sim2, ax = ax, hist=False, label='normal with exponential weight')
sns.distplot(sim3, ax = ax, hist=False, label='MLE fitted T')
ax.legend()
```

```
/var/folders/1n/6zcygvs12455myprvytb14680000gn/T/ipykernel_12623/1805923369.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sim1, ax = ax, hist=False, label='normal')
/var/folders/1n/6zcygvs12455myprvytb14680000gn/T/ipykernel_12623/1805923369.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sim2, ax = ax, hist=False, label='normal with exponential weight')
/var/folders/1n/6zcygvs12455myprvytb14680000gn/T/ipykernel_12623/1805923369.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sim3, ax = ax, hist=False, label='MLE fitted T')
```

Out[71]: <matplotlib.legend.Legend at 0x29e982b90>



The normal and the normal with exponential weight distributions both fit the future distribution.

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. Calculate the VaR of each portfolio as well as your total VaR(VaR of the total holdings).

Discuss your methods, why you chose those methods, and your results.

Since both Delta Normal VaR and Normal Monte Carlo VaR require the data being normality, we should perform the normality test first.

In [72]:
```
p = pd.read_csv("Portfolio.csv")
dp = pd.read_csv("DailyPrices.csv")
s = 0
for i in p[p["Portfolio"]=="A"]['Stock']:
    if shapiro(return_calculate(dp.loc[:,i]))[1]>0.05:
        s+=1
print("the percentage of portfolio A that is normaly distributed is:", s/len(p[p["Portfolio"]=="A"]))
s=0
for i in p[p["Portfolio"]=="B"]['Stock']:
    if shapiro(return_calculate(dp.loc[:,i]))[1]>0.05:
        s+=1
print("the percentage of portfolio B that is normaly distributed is:", s/len(p[p["Portfolio"]=="B"]))
s=0
for i in p[p["Portfolio"]=="C"]['Stock']:
    if shapiro(return_calculate(dp.loc[:,i]))[1]>0.05:
        s+=1
print("the percentage of portfolio C that is normaly distributed is:", s/len(p[p["Portfolio"]=="C"]))
```

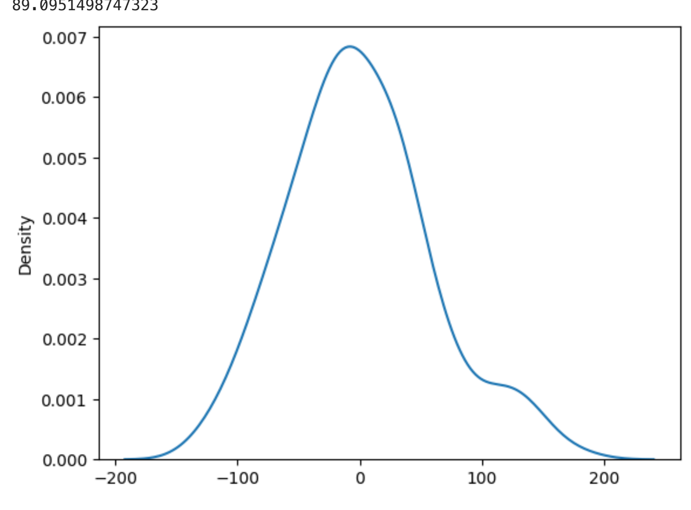
```
the percentage of portfolio A that is normaly distributed is: 0.5277777777777778
the percentage of portfolio B that is normaly distributed is: 0.46875
the percentage of portfolio C that is normaly distributed is: 0.59375
```

Since we only have half of the total stocks that are normality distributed, the historical VaR should be used.

Portfolio A historical VaR

In [73]:
```
portA_sim = []
for _ in range(200):
    value = 0
    for i in p[p["Portfolio"]=="A"]['Stock']:
        cp = dp.loc[:,i].iloc[-1]
        returns = return_calculate(dp.loc[:,i])
        value+=cp*(1+random.choice(list(returns)))
    portA_sim.append(value)
portA_sim -= np.mean(portA_sim)
sns.kdeplot(portA_sim)
print(-np.percentile(portA_sim,5))
```

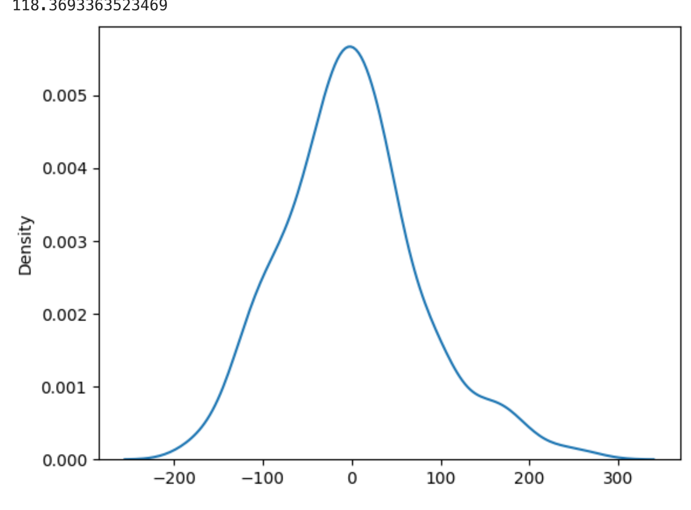
```
89.0951498747323
```



Portfolio B historical VaR

In [74]:
```
portB_sim = []
for _ in range(200):
    value = 0
    for i in p[p["Portfolio"]=="B"]['Stock']:
        cp = dp.loc[:,i].iloc[-1]
        returns = return_calculate(dp.loc[:,i])
        value+=cp*(1+random.choice(list(returns)))
    portB_sim.append(value)
portB_sim -= np.mean(portB_sim)
sns.kdeplot(portB_sim)
print(-np.percentile(portB_sim,5))
```

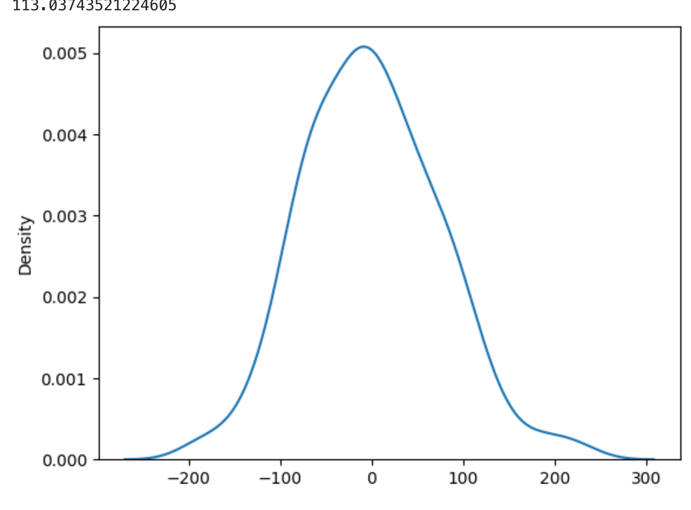
```
118.3693363523469
```



Portfolio C historical VaR

In [75]:
```
portC_sim = []
for _ in range(200):
    value = 0
    for i in p[p["Portfolio"]=="C"]['Stock']:
        cp = dp.loc[:,i].iloc[-1]
        returns = return_calculate(dp.loc[:,i])
        value+=cp*(1+random.choice(list(returns)))
    portC_sim.append(value)
portC_sim -= np.mean(portC_sim)
sns.kdeplot(portC_sim)
print(-np.percentile(portC_sim,5))
```

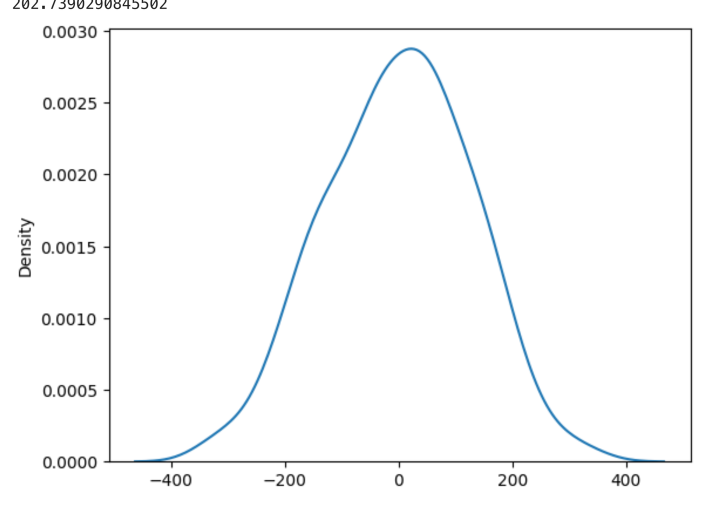
```
113.03743521224605
```



Portfolio total historical VaR

In [76]:
```
port_sim = []
for _ in range(200):
    value = 0
    for i in p['Stock']:
        cp = dp.loc[:,i].iloc[-1]
        returns = return_calculate(dp.loc[:,i])
        value+=cp*(1+random.choice(list(returns)))
    port_sim.append(value)
port_sim -= np.mean(port_sim)
sns.kdeplot(port_sim)
print(-np.percentile(port_sim,5))
```

```
202.7390290845502
```



In [ ]: