In [ ]:
```
#Shanglin Li
#Hashed Name:  8afd6878a54824b8adaba8caf6c1f8bc7414708d
```

In [ ]:
```
import sys
sys.path.append('/Users/ansel_li/Fintech545/public/')
```

In [ ]:
```
import numpy as np
import pandas as pd
from risk_mgmt import calc_return, PSD_fix, gbsm, VaR, risk
```

Problem1

In [ ]:
```
# Log return
p1_data = pd.read_csv('problem1.csv')
log_return = calc_return.return_calculate(p1_data,'LOG')
log_return = log_return[['Price1','Price2','Price3']]
log_return.head()
```

Out[ ]:

	Price1	Price2	Price3
1	-0.035169	-0.000252	0.018486
2	-0.002239	-0.001236	0.000946
3	NaN	NaN	-0.026082
4	NaN	NaN	0.007248
5	0.049850	-0.003192	0.009794

In [ ]:
```
# Pairwise covariance
covar = log_return.cov()
covar
```

Out[ ]:

	Price1	Price2	Price3
Price1	0.000972	0.000029	0.000085
Price2	0.000029	0.000065	0.000018
Price3	0.000085	0.000018	0.000138

In [ ]:
```
# PSD fix
print(PSD_fix.is_psd(covar))
```

```
True
```

d. Discuss when you might see data like this in the real world

For a data with missing values, we can

1. Omit it and calculate the pairwise data directly. But it may result in some imprecision when there are too much of them missing
2. Fill the missing data with randomly generated numbers with certain algorithm like mean value between the missing values, or using random normal generate with data's mean and standard deviation.

Problem 2

In [ ]:
```
# read the data
p2_data = pd.read_csv('problem2.csv')
p2_data
```

Out[ ]:

	Underlying	Strike	IV	TTM	RF	DivRate
0	107.669823	102.31725	0.2	142	0.045	0.052899

In [ ]:
```
# Read price info
S = p2_data['Underlying'].values[0]
K = p2_data['Strike'].values[0]
sigma = p2_data['IV'].values[0]
T = p2_data['TTM'].values[0]/255
rf = p2_data['RF'].values[0]
q = p2_data['DivRate'].values[0]
```

In [ ]:
```
# call function to find prices and greeks
option_data = gbsm.gbsm_greeks(S,K,T,rf,q,sigma)
Call_price = option_data['P']
Delta = option_data['delta']
Gamma = option_data['gamma']
Vega = option_data['vega']
Rho = option_data['rho']
print("Call option price:", Call_price)
print("Delta:", Delta)
print('Gamma:', Gamma)
print('Vega:', Vega)
print('Rho:', Rho)
```

```
Call option price: 8.750112481033398
Delta: 0.6316672417005217
Gamma: 0.02236809662810111
Vega: 28.879866718413588
Rho: 33.000458844581445
```

In [ ]:
```
# monte carlo Normal simulation
returns = np.random.normal(0, sigma, size=100)
normVaR, normES = VaR.normal_var(returns, mean=0, alpha=0.05, nsamples=10000
print("VaR:", normVaR)
print("ES:", normES)
```

```
VaR: 0.3117809274622665
ES: 0.39234749488272513
```

h. This portfolio’s payoff structure most closely resembles what?

A: It is like a covered call

Problem 3

In [ ]:
```
p3_covar = pd.read_csv('problem3_cov.csv')
ER_data = pd.read_csv('problem3_ER.csv')
rf = ER_data['RF'].values[0]
er = ER_data[['Expected_Value_1','Expected_Value_2','Expected_Value_3']].val
er
```

Out[ ]:
```
array([0.10966427, 0.12420111, 0.10710191])
```

In [ ]:
```
# Calculate the max sharpe ratio and weights
w_max_sharpe, sharpe = risk.max_sharpe_ratio_weights(p3_covar, er, rf, 'True
print("Max sharpe ratio:", sharpe)
print("Max Sharpe portfolio weight:", w_max_sharpe)
```

```
Max sharpe ratio: 0.5218568887570968
Max Sharpe portfolio weight: [0.3507 0.2856 0.3638]
```

In [ ]:
```
risk_parity_portfolio = risk.risk_parity_weights(p3_covar)
print('Risk parity portfolio:',risk_parity_portfolio)
```

```
Risk parity portfolio: [0.3499 0.2857 0.3644]
```

In [ ]:
```
risk_contribution_ms = risk.risk_contribution(w_max_sharpe, p3_covar)
risk_contribution_risk_p = risk.risk_contribution(risk_parity_portfolio, p3_
```

In [ ]:
```
risk_contribution_risk_p - risk_contribution_ms
```

Out[ ]:
```
0   -0.000151
1    0.000026
2    0.000112
dtype: float64
```

Problem 4

In [ ]:
```
stocks=['Asset1', 'Asset2', 'Asset3']
portfolio_weights = pd.read_csv('problem4_startWeight.csv').values[0]
p4_returns = pd.read_csv('problem4_returns.csv')
print('The initial weight:', portfolio_weights)
# Calculate portfolio return and updated weights for each day
n = p4_returns.shape[0]
m = len(stocks)

pReturn = np.empty(n)
weights = np.empty((n, len(portfolio_weights)))
lastW = portfolio_weights.copy()
matReturns = p4_returns[stocks].values
```

```
print("\nEach weights:")
for i in range(n):
    # Save Current Weights in Matrix
    weights[i, :] = lastW

    # Update Weights by return
    lastW = lastW * (1.0 + matReturns[i, :])

    # Portfolio return is the sum of the updated weights
    pR = lastW.sum()

    # Normalize the weights back so sum = 1
    lastW = lastW / pR

    # Store the return
    pReturn[i] = pR - 1
    print(lastW)

# Set the portfolio return in the Update Return DataFrame
p4_returns["Portfolio"] = pReturn
# Calculate the total return
totalRet = np.exp(np.sum(np.log(pReturn + 1))) - 1
# Calculate the Carino K
k = np.log(totalRet + 1) / totalRet

# Carino k_t is the ratio scaled by 1/K
carinoK = np.log(1.0 + pReturn) / pReturn / k

# Calculate the return attribution
attrib = pd.DataFrame(matReturns * weights * carinoK[:, np.newaxis], columns

# Set up a DataFrame for output
Attribution = pd.DataFrame({"Value": ["TotalReturn", "Return Attribution"]})

# Loop over the stocks
for s in stocks + ["Portfolio"]:
    # Total Stock return over the period
    tr = np.exp(np.sum(np.log(p4_returns[s] + 1))) - 1

    # Attribution Return (total portfolio return if we are updating the port
    atr = tr if s == "Portfolio" else attrib[s].sum()

    # Set the values
    Attribution[s] = [tr, atr]

# Realized Volatility Attribution

# Y is our stock returns scaled by their weight at each time
Y = matReturns * weights

# Set up X with the Portfolio Return
X = np.column_stack((np.ones(n), pReturn))

# Calculate the Beta and discard the intercept
B = np.linalg.inv(X.T @ X) @ X.T @ Y
```

```
B = B[1, :]

# Component SD is Beta times the standard Deviation of the portfolio
cSD = B * np.std(pReturn)

# Add the Vol attribution to the output
vol_attrib = pd.DataFrame({"Value": ["Vol Attribution"], **{stocks[i]: [cSD[

Attribution = pd.concat([Attribution, vol_attrib], ignore_index=True)
print('')
print(Attribution)
```

```
The initial weight: [0.37782635 0.36330546 0.25886818]

Each weights:
[0.37322451 0.36249127 0.26428422]
[0.37146542 0.35617318 0.2723614 ]
[0.36805836 0.35487591 0.27706573]
[0.37158255 0.34868393 0.27973351]
[0.36486543 0.36154515 0.27358942]
[0.36314882 0.36273412 0.27411706]
[0.35964966 0.37607531 0.26427503]
[0.32247843 0.40632223 0.27119934]
[0.30039854 0.43209018 0.26751128]
[0.34501985 0.41201558 0.24296458]
[0.32642169 0.42464402 0.24893429]
[0.2962571  0.41730728 0.28643562]
[0.27838056 0.4334545  0.28816495]
[0.28523851 0.44108855 0.27367294]
[0.28239936 0.44396344 0.2736372 ]
[0.25841339 0.45866093 0.28292569]
[0.25227395 0.45940523 0.28832083]
[0.25122589 0.4725095  0.27626461]
[0.25576158 0.46172779 0.28251063]
[0.25290569 0.47663443 0.27045987]

               Value    Asset1    Asset2    Asset3  Portfolio
0        TotalReturn -0.203172  0.561752  0.243720   0.190415
1  Return Attribution -0.078657  0.202951  0.066121   0.190415
2     Vol Attribution  0.009023  0.010447  0.005675   0.025145
```

Problem 5

In [ ]:

```
# read data
p5_prices = pd.read_csv('problem5.csv')
returns = calc_return.return_calculate(p5_prices)
returns.head()
returns['P1+2'] = returns['Price1'] + returns['Price2']
returns['P3+4'] = returns['Price3'] + returns['Price4']
returns['ALL'] = returns['P1+2'] + returns['P3+4']
returns.drop('Date',inplace=True, axis=1)
returns.head()
```

Out[ ]:

	Price1	Price2	Price3	Price4	P1+2	P3+4	ALL
1	-0.000205	-0.000210	0.000383	-0.000130	-0.000415	0.000253	-0.000162
2	0.000092	0.000534	-0.000759	-0.000360	0.000626	-0.001119	-0.000494
3	0.000017	-0.000501	0.000232	0.000183	-0.000484	0.000415	-0.000069
4	0.000082	0.000352	0.000135	0.000254	0.000434	0.000389	0.000823
5	-0.000196	-0.000383	0.000154	-0.000008	-0.000580	0.000145	-0.000434

In [ ]:

```
list_p = returns.columns.values
alpha = 0.05
nsample = 10000
mean = 0
for i in list_p:
    print(f'The VaR(5%) of {i} is :{VaR.MLE_t_var(returns[i], mean, alpha, r
```

```
The VaR(5%) of Price1 is :0.000462471106808633
The VaR(5%) of Price2 is :0.0005891449598025266
The VaR(5%) of Price3 is :0.0006789608171660899
The VaR(5%) of Price4 is :0.0006466363741413171
The VaR(5%) of P1+2 is :0.0008186813742070512
The VaR(5%) of P3+4 is :0.001061544853925538
The VaR(5%) of ALL is :0.0015634334980447393
```

In [ ]: