Problem 1

A

```
arithmetic_return = mylib.arithmeticReturn(asset)
arithmetic_return = arithmetic_return - arithmetic_return.mean()
print(arithmetic_return.iloc[-5:, :])
print("Standard Deviation: ")
print(arithmetic_return.std())
```

```
Last 5 rows:
Date             SPY      AAPL      EQIX
2024-12-27 -0.011492 -0.014678 -0.006966
2024-12-30 -0.012377 -0.014699 -0.008064
2024-12-31 -0.004603 -0.008493  0.006512
2025-01-02 -0.003422 -0.027671  0.000497
2025-01-03  0.011538 -0.003445  0.015745

Standard Deviation:
SPY     0.008077
AAPL    0.013483
EQIX    0.015361
```

B

```
log_return = mylib.logReturn(asset)
log_return = log_return - log_return.mean()
print(log_return.iloc[-5:, :])
print("Standard Deviation: ")
print(log_return.std())
```

```
Last 5 rows:
Date             SPY      AAPL      EQIX
2024-12-27 -0.011515 -0.014675 -0.006867
2024-12-30 -0.012410 -0.014696 -0.007972
2024-12-31 -0.004577 -0.008427  0.006602
2025-01-02 -0.003392 -0.027930  0.000613
2025-01-03  0.011494 -0.003356  0.015725

Standard Deviation:
SPY     0.008078
AAPL    0.013446
EQIX    0.015270
```

Problem 2

A

```
n_stock = np.array([100, 200, 150])
price = asset.iloc[-1, :]
pf_value = n_stock @ price
print("Portfolio Value:", pf_value)
```

```
Portfolio Value: 251862.4969482422
```

B

a

```
delta = np.multiply(n_stock, price).values / pf_value
sigma = mylib.covEW(arithmetic_return, .97)
pf_sigma = np.sqrt(delta @ sigma @ delta)
var = - pf_value * stats.norm.ppf(.05) * pf_sigma
es = - pf_value * pf_sigma * (-stats.norm.pdf(stats.norm.ppf(.05))/.05)

var_stock = -n_stock * price * stats.norm.ppf(.05) * np.sqrt(np.diag(sigma))
es_stock = -n_stock * price * np.sqrt(np.diag(sigma)) * (-stats.norm.pdf(stats.norm.ppf(.05))/.05)

result_delta_normal = pd.DataFrame({
    'VaR': var_stock,
    'ES': es_stock
})

result_delta_normal.loc['Total'] = [var, es]

result_delta_normal
```

	VaR	ES
SPY	825.801984	1035.589004
AAPL	944.781091	1184.793604
EQIX	2931.344128	3676.023797
Total	3856.318301	4835.978728

b

```
pf = pd.DataFrame(columns=['stock', 'holding', 'price', 'dist'])
pf.loc[:, 'stock'] = asset.columns
pf.loc[:,'holding']=n_stock
pf.loc[:,'price'] = asset.iloc[-1, :].values
pf.loc[:,'dist'] = "T"
result_sim_copula = mylib.varesSimCopula(pf, arithmetic_return)
result_sim_copula = result_sim_copula[['VaR', 'ES']]
result_sim_copula
```

	VaR	ES
SPY	778.302425	1038.920756
AAPL	1035.600188	1464.892622
EQIX	3397.741427	4843.803932
Total	4388.772702	6100.974082

c

```
var_historical = arithmetic_return.quantile(.05)
es_historical = arithmetic_return[arithmetic_return <= var_historical].mean()

var_historical_stock = -n_stock * price * var_historical
es_historical_stock = -n_stock * price * es_historical

var_historical_total = var_historical_stock.sum()
es_historical_total = es_historical_stock.sum()

result_historical = pd.DataFrame({
    'VaR': var_historical_stock,
    'ES': es_historical_stock
})

result_historical.loc['Total'] = [var_historical_total, es_historical_total]

result_historical
```

	VaR	ES
SPY	872.403863	1080.104204
AAPL	1067.114956	1437.785272
EQIX	3635.077091	4714.893996
Total	5574.595909	7232.783472

C

According to the result, VaR and ES calculated by historical simulation is the greatest, followed by T distribution using a Gaussian Copula and delta normal. Exponentially weighted covariance is used in a, and more weight is given to recent data. In b, we assume the return following a T distribution, accounting for the fat tails of the return. A Gaussian Copula is used to interpret the relationship between these stocks. In c, we simply simulated using historical data, and got a rough calculation of VaR and ES.

Problem 3

A

```python
ttm = .25
P = 3
S = 31
K = 30
rf = .10

def bs_call(S, K, T, r, sigma):
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return S * stats.norm.cdf(d1) - K * np.exp(-r * T) * stats.norm.cdf(d2)

def obj(sigma):
    return bs_call(S, K, ttm, rf,sigma) - P

iv = brentq(obj, 1e-6, 5)
print("Implied volatility:", iv)
```

```
Implied volatility: 0.3350803924787904
```

B

```python
d1 = (math.log(S / K) + (rf + 0.5 * iv**2) * ttm) / (iv * math.sqrt(ttm))
d2 = d1 - iv * math.sqrt(ttm)
delta = stats.norm.cdf(d1)
print("Delta:", delta)
vega = S * stats.norm.pdf(d1) * np.sqrt(ttm)
print("Vega:", vega)

term1 = - (S * stats.norm.pdf(d1) * iv) / (2 * math.sqrt(ttm))
theta = term1 - rf * K * np.exp(-rf * ttm) * stats.norm.cdf(d2)
print("Theta:", theta)

iv_2 = iv + .01
P_2 = bs_call(S, K, ttm, rf, iv_2)
print("Price change:", P_2 - P)
```

```
Delta:  0.6659296527386921
Vega:   5.640705439230117
Theta:  -5.544561508358896
Price change: 0.05649842751734013
```

The price change can also be calculated using Vega.

$$5.64 \times 1\% = 0.0564$$

C

```python
def bs_put(S, K, T, r, sigma):
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return K * np.exp(-r * T) * stats.norm.cdf(-d2) - S * stats.norm.cdf(-d1)

price_put = bs_put(S, K, ttm, rf, iv)
left = P + K * math.exp(-rf * ttm)
right = price_put + S
print("Left:", left)
print("Right:", right)
```

```
Left of euqation:  32.25929736084998
Right of equation: 32.25929736084998
```

D

```python
# Assume IV = 25%
sigma = .25
trading_days = 255
holding_days = 20
sigma_daily = sigma * math.sqrt(holding_days) / math.sqrt(trading_days)
price_call = bs_call(S, K, ttm, rf, sigma)
price_put = bs_put(S, K, ttm, rf, sigma)
pf_value = S + price_call + price_put
def delta(S, K, ttm, rf, sigma, call=True):
    d1 = (math.log(S / K) + (rf + 0.5 * sigma**2) * ttm) / (sigma * math.sqrt(ttm))
    delta = stats.norm.cdf(d1)
    if not call:
        delta -= 1
    return delta

# Delta Normal
delta_call = delta(S, K, ttm, rf, sigma)
delta_put = delta(S, K, ttm, rf, sigma, False)
delta_stock = 1.0
delta_pf = delta_call + delta_put + delta_stock
vol_pf = np.sqrt((delta_call ** 2 + delta_put ** 2 + delta_stock ** 2) * sigma_daily**2)

var_dn = -delta_pf * vol_pf * stats.norm.ppf(.05)
es_dn = delta_pf * vol_pf * (stats.norm.pdf(stats.norm.ppf(.05)) / .05)

print("VaR Delta Normal:", var_dn)
print("ES Delta Normal:", es_dn)

# Monte Carlo
n_sim = 100000
stock_returns = np.random.normal(0, sigma_daily, n_sim)
stock_price_last = S * (1+stock_returns)

call_price_last = bs_call(stock_price_last, K, 0.25-20/trading_days, rf, sigma)
put_price_last = bs_put(stock_price_last, K, 0.25-20/trading_days, rf, sigma)

pf_value_last = stock_price_last + call_price_last + put_price_last

price_change = pf_value_last - pf_value
var_mc = -np.percentile(price_change, 5)
es_mc = -price_change[price_change < -var_mc].mean()
print("VaR MC:", var_mc)
print("ES MC:", es_mc)
```

```
VaR Delta Normal: 0.20270987353546513
ES Delta Normal: 0.25420635945872544
VaR MC: 3.9840583406808237
ES MC: 4.3145858777130845
```

E

VaR and ES calculated using delta normal is less than using Monte Carlo. Delta normal does not take time into account.