Oblig STK-MAT3700

Simen Meen Haugeng

October 2024

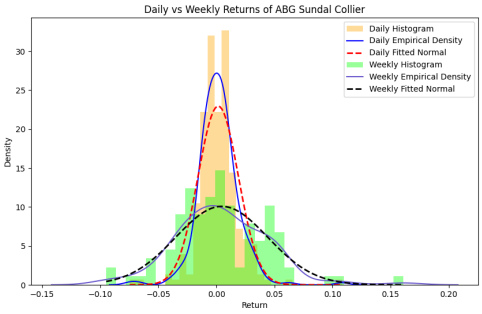
1

a

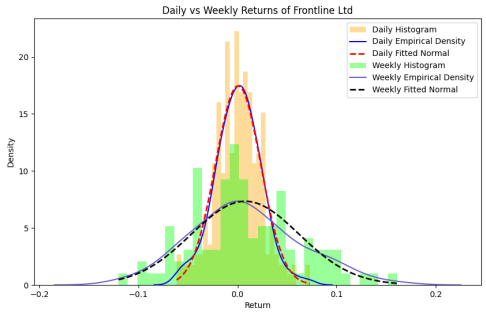
```
    import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import seaborn as sns
from scipy.optimize import minimize

# Download stock data
stocks = ['GJF.OL', 'ORK.OL', 'FRO.OL', 'WWI.OL', 'ABG.OL']
stock_names = {
    'GJF.OL': 'Gjensidige Forsikring ASA',
    'ORK.OL': 'Orkla ASA',
    'FRO.OL': 'Frontline Ltd',
    'WWI.OL': 'Wallenius Wilhelmsen ASA',
    'ABG.OL': 'ABG Sundal Collier'
}
#a)
daily_data = yf.download(stocks, period='1y', interval='1d')['Adj Close']
weekly_data = yf.download(stocks, period='2y', interval='1wk')['Adj Close']
# Calculate returns
daily_returns = daily_data.pct_change().dropna()
weekly_returns = weekly_data.pct_change().dropna()
# Calculate expected returns and volatility
expected_daily_returns = daily_returns.mean() * 252
daily_volatility = daily_returns.std() * np.sqrt(252)
expected_weekly_returns = weekly_returns.mean() * 52
weekly_volatility = weekly_returns.std() * np.sqrt(52)

print("\nExpected Returns:")
print(expected_daily_returns)
```

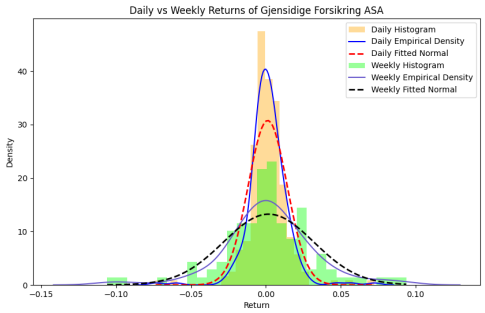
```
print("\nVolatility:")
print(daily_volatility)
print("\nWeekly Expected Returns:")
print(expected_weekly_returns)
print("\nWeekly Volatility:")
print(weekly_volatility)

# Loop through each ticker
for ticker in stocks:
    plt.figure(figsize=(10, 6))

    # Plot histogram and density for daily returns
    plt.hist(daily_returns[ticker], bins=30, density=True, alpha=0.4, color='orange', label=
    sns.kdeplot(daily_returns[ticker], label='Daily Empirical Density', color='blue')

    mu_daily, std_daily = norm.fit(daily_returns[ticker])
    x_daily = np.linspace(daily_returns[ticker].min(), daily_returns[ticker].max(), 100)
    p_daily = norm.pdf(x_daily, mu_daily, std_daily)
    plt.plot(x_daily, p_daily, 'r--', linewidth=2, label='Daily Fitted Normal')

    # Plot histogram and density for weekly returns
    plt.hist(weekly_returns[ticker], bins=30, density=True, alpha=0.4, color='lime', label=
    sns.kdeplot(weekly_returns[ticker], label='Weekly Empirical Density', color='slateblue')

    mu_weekly, std_weekly = norm.fit(weekly_returns[ticker])
    x_weekly = np.linspace(weekly_returns[ticker].min(), weekly_returns[ticker].max(), 100)
    p_weekly = norm.pdf(x_weekly, mu_weekly, std_weekly)
    plt.plot(x_weekly, p_weekly, 'k--', linewidth=2, label='Weekly Fitted Normal')


    plt.title(f"Daily vs Weekly Returns of {stock_names[ticker]}")
    plt.xlabel('Return')
    plt.ylabel('Density')
    plt.legend()

    plt.show()
```

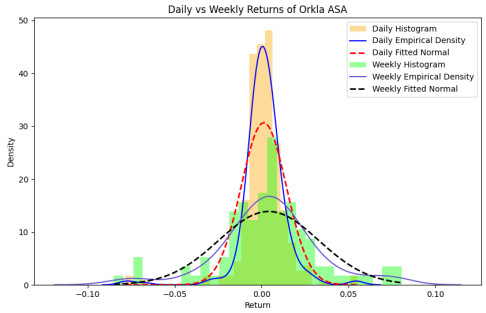
In the code above I have retrieved both daily and weekly data from each of the stock and calculating volatility and returns for both of them using built in functions in python and then annualized it by the of trading days and weeks in a year. I then made a loop that goes through each ticker and makes a histogram for both daily and weekly data in the same plot with belonging empirical and normal

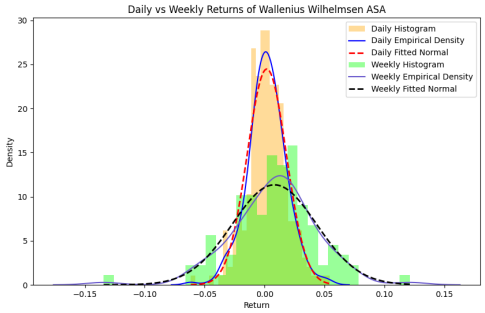


density which resulted in this output:










```
Daily expected Returns:
Ticker
ABG.OL    0.413223
FRO.OL    0.132497
GJF.OL    0.317580
ORK.OL    0.316952
WWI.OL    0.469668
dtype: float64

Daily volatility:
Ticker
ABG.OL    0.275946
```

```
FRO.OL    0.362910
GJF.OL    0.205958
ORK.OL    0.206584
WWI.OL    0.258772
dtype: float64

Weekly Expected Returns:
Ticker
ABG.OL    0.244274
FRO.OL    0.401645
GJF.OL    0.100463
ORK.OL    0.212985
WWI.OL    0.428592
dtype: float64

Weekly Volatility:
Ticker
ABG.OL    0.287093
FRO.OL    0.394079
GJF.OL    0.218419
ORK.OL    0.208392
WWI.OL    0.254881
dtype: float64
```

As we can see are the expected returns quite high for some of the stocks, especially ABG Sundal Collier and and Wallenius Wilhemsen both for daily and weekly data. Whats interesting is that Frontline seems to have a much higher expected return when using weekly data than daily data, which might be due to that the weekly data captures more data during the week which results in a an abnormal difference in expected return for the daily and weekly data. Whats also noticeable is that Frontline and Wallenius‘ fitted normal and empirical density are almost the same which means that their distributions are normal. We can also see that several of the weekly fitted normal and empirical densities seems to have quite ”fat tails” which could indicate that there is a quite high probability of rare events. If you look at the math behind it, its also plausible that it is due to that the weekly data have a higher SD which results in a wider and flatter pdf.

b

```
    #b)
# Calculate correlation and covariance matrices
corr_matrix_daily = daily_returns.corr()
cov_matrix_daily = daily_returns.cov() * 252
corr_matrix_weekly = weekly_returns.corr()
cov_matrix_weekly = weekly_returns.cov() * 52
```

```
print("\nDaily Correlation Matrix:")
print(corr_matrix_daily)
print("\nDaily Covariance Matrix:")
print(cov_matrix_daily)
print("\nWeekly Correlation Matrix:")
print(corr_matrix_weekly)
print("\nWeekly Covariance Matrix:")
print(cov_matrix_weekly)
```

```
Daily Correlation Matrix:
Ticker    ABG.OL    FRO.OL    GJF.OL    ORK.OL    WWI.OL
Ticker
ABG.OL  1.000000 -0.067768  0.130290  0.129922  0.127318
FRO.OL -0.067768  1.000000  0.001100  0.080908  0.264104
GJF.OL  0.130290  0.001100  1.000000 -0.031770  0.126293
ORK.OL  0.129922  0.080908 -0.031770  1.000000  0.013120
WWI.OL  0.127318  0.264104  0.126293  0.013120  1.000000

Daily Covariance Matrix:
Ticker    ABG.OL    FRO.OL    GJF.OL    ORK.OL    WWI.OL
Ticker
ABG.OL  0.075315 -0.006751  0.007365  0.007447  0.009037
FRO.OL -0.006751  0.131777  0.000082  0.006134  0.024796
GJF.OL  0.007365  0.000082  0.042425 -0.001367  0.006728
ORK.OL  0.007447  0.006134 -0.001367  0.043623  0.000709
WWI.OL  0.009037  0.024796  0.006728  0.000709  0.066891

Weekly Correlation Matrix:
Ticker    ABG.OL    FRO.OL    GJF.OL    ORK.OL    WWI.OL
Ticker
ABG.OL  1.000000 -0.014923  0.158720  0.144380  0.064248
FRO.OL -0.014923  1.000000 -0.016236 -0.132907  0.305938
GJF.OL  0.158720 -0.016236  1.000000 -0.049953  0.202012
ORK.OL  0.144380 -0.132907 -0.049953  1.000000 -0.146081
WWI.OL  0.064248  0.305938  0.202012 -0.146081  1.000000

Weekly Covariance Matrix:
Ticker    ABG.OL    FRO.OL    GJF.OL    ORK.OL    WWI.OL
Ticker
ABG.OL  0.080574 -0.001650  0.009873  0.008541  0.004655
FRO.OL -0.001650  0.151702 -0.001386 -0.010788  0.030415
GJF.OL  0.009873 -0.001386  0.048023 -0.002281  0.011300
ORK.OL  0.008541 -0.010788 -0.002281  0.043432 -0.007771
WWI.OL  0.004655  0.030415  0.011300 -0.007771  0.06515
```

Wee see that there isnt that high correlation between the stocks, which is because many of the stocks are in different sectors and therefore dont affect each other that much and/or is not affected the same by different happenings in the world. However, we can see that Wallenius and Frontline have a higher correlation which isnt that weird because they are both in the same kind of sector (industry/shipping). In terms of the covariance matrix we can see that the off-diagonal elements are quite small which isnt that weird because many of them dont have that high correlation either. Again the highest number is the covariance between WWI and FRO which again may be due to the slightly higher correlation between them.

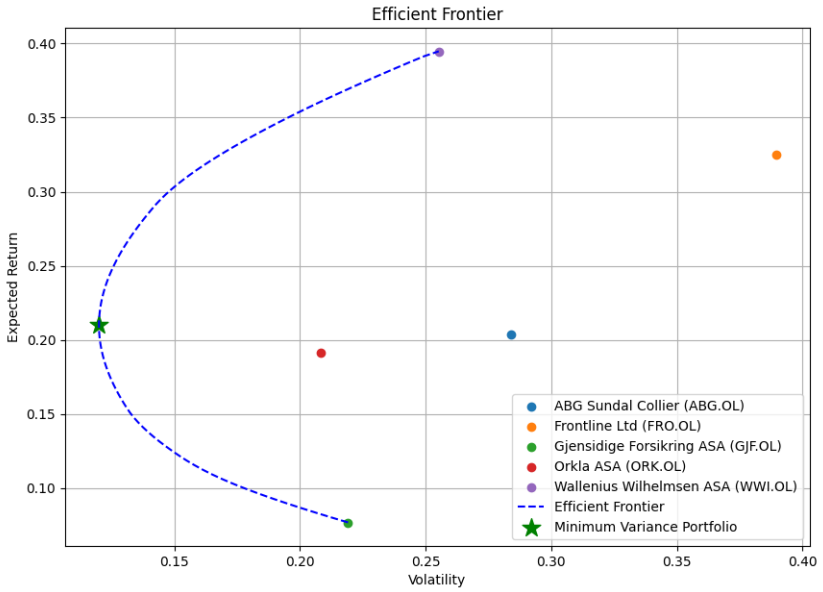
c

I have decided to use the weekly data to compute the efficient frontier because of the longer term perspective and that the weekly data captures more of the underlying data behind the stock.

```
#c)
# Function to calculate portfolio performance
def portfolio_return_volatility(weights, expected_weekly_returns, cov_matrix):
    portfolio_return = np.dot(weights, expected_weekly_returns)
    portfolio_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    return portfolio_return, portfolio_volatility

# Function to calculate portfolio variance
def portfolio_variance(weights, cov_matrix):
    return np.dot(weights.T, np.dot(cov_matrix, weights))

num_assets = len(stocks)

#Weights for stocks
initial_weights = num_assets * [1. / num_assets,]
#Make sure that the weights sum to 1
constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
# Bounds for weights: weights between 0 and 1 (no short-selling)
bounds = tuple((0, 1) for asset in range(num_assets))

# Minimize variance in terms of the minimum-variance portfolio
opt_results = minimize(portfolio_variance, initial_weights, args=(cov_matrix_weekly.values,)
# Optimal weights for minimum-variance portfolio
min_var_weights = opt_results.x
# Expected return and volatility of the minimum-variance portfolio
min_var_return, min_var_volatility = portfolio_return_volatility(min_var_weights, expected_w

print("\nMinimum Variance Portfolio:")
```

```
print("Weights:")
for i, stock in enumerate(stocks):
    print(f"{stock_names[stock]}: {min_var_weights[i]:.4f}")
print(f"Expected Return: {min_var_return:.4f}")
print(f"Volatility: {min_var_volatility:.4f}")

#Compute the efficient frontier
def efficient_frontier(expected_weekly_returns, cov_matrix_weekly, returns_range):
    efficient_portfolios = []
    num_assets = len(expected_weekly_returns)
    bounds = tuple((0, 1) for asset in range(num_assets))
    for target_return in returns_range:
        constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1},
                       {'type': 'eq', 'fun': lambda x: np.dot(x, expected_weekly_returns) -
        result = minimize(portfolio_variance, num_assets * [1. / num_assets], args=(cov_matr
        if result.success:
            portfolio_volatility = np.sqrt(result.fun)
            efficient_portfolios.append({'Return': target_return, 'Volatility': portfolio_vo
        else:
            print("Optimization failed for target return:", target_return)
    return efficient_portfolios

# Define range of target returns
target_returns = np.linspace(expected_weekly_returns.min(), expected_weekly_returns.max(), 1
# Compute efficient frontier
efficient_portfolios = efficient_frontier(expected_weekly_returns.values, cov_matrix_weekly.

# Extract returns and volatilities
ef_returns = [p['Return'] for p in efficient_portfolios]
ef_volatilities = [p['Volatility'] for p in efficient_portfolios]
# Plot efficient frontier, I tried doing it with a for loop but it didn't work so I did it m
data = {
    'Ticker': ['ABG.OL', 'FRO.OL', 'GJF.OL', 'ORK.OL', 'WWI.OL'],
    'Company': ['ABG Sundal Collier', 'Frontline Ltd', 'Gjensidige Forsikring ASA', 'Orkla A
    'Expected Return': [0.203643, 0.324811, 0.076677, 0.191109, 0.394796],
    'Volatility': [0.283856, 0.389489, 0.219141, 0.208403, 0.255246]
}

df = pd.DataFrame(data)
df.set_index('Ticker', inplace=True)
df.sort_values('Company', inplace=True)

#Plotting
plt.figure(figsize=(10, 7))
for index, row in df.iterrows():
    plt.scatter(row['Volatility'], row['Expected Return'], label=f"{row['Company']} ({index]
```

```
plt.plot(ef_volatilities, ef_returns, 'b--', label='Efficient Frontier')
plt.scatter(min_var_volatility, min_var_return, c='green', marker='*', s=200, label='Minimum

plt.title('Efficient Frontier')
plt.xlabel('Volatility')
plt.ylabel('Expected Return')
plt.legend()
plt.grid(True)
plt.show()

Minimum Variance Portfolio:
Weights:
Gjensidige Forsikring ASA: 0.0971
Orkla ASA: 0.0889
Frontline Ltd: 0.2608
Wallenius Wilhelmsen ASA: 0.3777
ABG Sundal Collier: 0.1754
Expected Return: 0.2101
Volatility: 0.1200
```



We can see that frontline is very much in the right corner because of its high return and volatility, while gjensidige is quite opposite with low return and low volatility. The three stocks with the highest returns are given the highest weights and I am actually a little surprised that Orkla has a much lower weight than ABG in terms of quite similar returns and that Orkla also have quite a

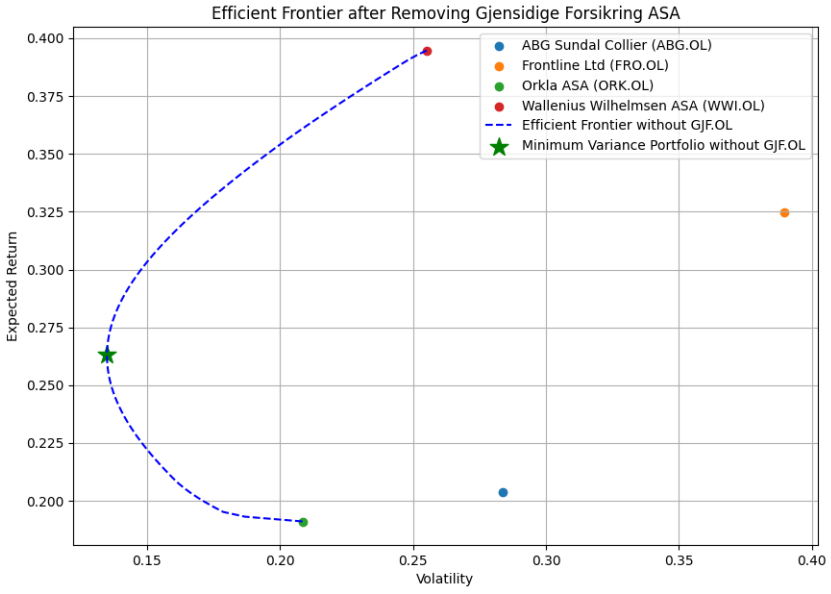
bit lower volatility which could help in terms of a lower risk portfolio. Orkla actually have a lower weight than Gjensidige, which is kind of weird to me(?) in terms of that Orkla have a higher return and lower volatility than Gjensidige. We see that WWI have almost 37 percent of the portfolio, which isnt that weird in terms of its high return and not that high volatility. Overall, the minimum variance protfolio results in an expected return of 21 percent and volatility of 12 percent. This is quite good, but I think we may need to pay attention to that very much of the portfolio is allocated Frontline and Wallenius, which also are in the same sector, so we are hevaliy reliant on the indutry/shipping sector. If that sector suddenly experiences some abnormalities, the portfolio may be doing a bit worse quite quick.

d

```
    # d)
# Removing Gjensidige Forsikring ASA from the portfolio
stocks_reduced = [stock for stock in stocks if stock != 'GJF.OL']
stock_names_reduced = {stock: stock_names[stock] for stock in stocks_reduced}

#Calculate cov_matrix again with reduced stocks
expected_weekly_returns_reduced = expected_weekly_returns[stocks_reduced]
cov_matrix_weekly_reduced = cov_matrix_weekly.loc[stocks_reduced, stocks_reduced]

num_assets = len(stocks_reduced)
initial_weights = num_assets * [1. / num_assets,]
constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
bounds = tuple((0, 1) for asset in range(num_assets))

# Minimize variance to find the minimum-variance portfolio
opt_results = minimize(portfolio_variance, initial_weights, args=(cov_matrix_weekly_reduced.
min_var_weights = opt_results.x

min_var_return, min_var_volatility = portfolio_return_volatility(
    min_var_weights,
    expected_weekly_returns_reduced.values,
    cov_matrix_weekly_reduced.values
)

print("\nMinimum Variance Portfolio without Gjensidige Forsikring ASA:")
print("Weights:")
for i, stock in enumerate(stocks_reduced):
    print(f"{stock_names_reduced[stock]}: {min_var_weights[i]:.4f}")
print(f"Expected Return: {min_var_return:.4f}")
print(f"Volatility: {min_var_volatility:.4f}")

#Compute the efficient frontier for the reduced portfolio
```

```
def efficient_frontier(expected_returns, cov_matrix, returns_range):
    efficient_portfolios = []
    num_assets = len(expected_returns)
    bounds = tuple((0, 1) for asset in range(num_assets))
    for target_return in returns_range:
        constraints = (
            {'type': 'eq', 'fun': lambda x: np.sum(x) - 1},
            {'type': 'eq', 'fun': lambda x: np.dot(x, expected_returns) - target_return}
        )
        result = minimize(portfolio_variance, num_assets * [1. / num_assets], args=(cov_matr
        )
        if result.success:
            portfolio_volatility = np.sqrt(result.fun)
            efficient_portfolios.append({
                'Return': target_return,
                'Volatility': portfolio_volatility,
                'Weights': result.x
            })
        else:
            print("Optimization failed for target return:", target_return)
    return efficient_portfolios

target_returns_reduced = np.linspace(expected_weekly_returns_reduced.min(), expected_weekly_

efficient_portfolios_reduced = efficient_frontier(
    expected_weekly_returns_reduced.values,
    cov_matrix_weekly_reduced.values,
    target_returns_reduced
)

ef_returns_reduced = [p['Return'] for p in efficient_portfolios_reduced]
ef_volatilities_reduced = [p['Volatility'] for p in efficient_portfolios_reduced]

data_reduced = {
    'Ticker': stocks_reduced,
    'Company': [stock_names_reduced[ticker] for ticker in stocks_reduced],
    'Expected Return': expected_weekly_returns_reduced.values,
    'Volatility': weekly_volatility[stocks_reduced].values
}

df_reduced = pd.DataFrame(data_reduced)
df_reduced.set_index('Ticker', inplace=True)
df_reduced.sort_values('Company', inplace=True)

plt.figure(figsize=(10, 7))
```

```
for index, row in df_reduced.iterrows():
    plt.scatter(row['Volatility'], row['Expected Return'], label=f"{row['Company']} ({index]

# Plot the efficient frontier
plt.plot(ef_volatilities_reduced, ef_returns_reduced, 'b--', label='Efficient Frontier witho

plt.scatter(min_var_volatility, min_var_return, c='green', marker='*', s=200, label='Minimum

plt.title('Efficient Frontier after Removing Gjensidige Forsikring ASA')
plt.xlabel('Volatility')
plt.ylabel('Expected Return')
plt.legend()
plt.grid(True)
plt.show()

Minimum Variance Portfolio without Gjensidige Forsikring ASA:
Weights:
Orkla ASA: 0.4600
Frontline Ltd: 0.0976
Wallenius Wilhelmsen ASA: 0.2796
ABG Sundal Collier: 0.1628
Expected Return: 0.2632
Volatility: 0.1349
```



Since Gjensidige had the lowest return, I wanted to see what happens if I made

a portfolio without that stock.
We can see that, after removing Gjensidige from the portfolio, that Frontline gets a much lower weight and Orkla get a very heigh 46 percent of the portfolio. This is most likely so the portfolio dont have to much volatility even though we removed Gjensidige. Wallenius also gets a lower weight than in problem c. This results in a portfolio with an expected return of roughly 26 percent and a volatility of 13,5 percent. The return are higher, but the volatility have also increased with 1,5 percent. The risk didnt go that much higher, mostly because Orkla receieved a bigger position in the portfolio which helps the volatility not sky rocketing. If we held the same position in Frontline and just evenly placed the former position in Gjensidige over the remaining stock, the return would possibly be higher, but the portfolio would also be a very risky one.

2

a

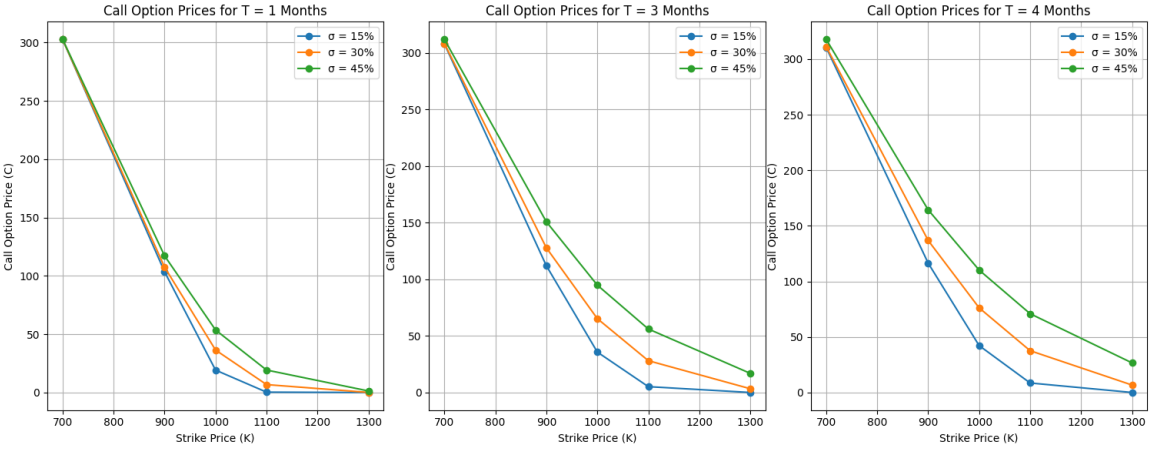
I see that the yield on a 10-year Norweagian government bond is 3,5 percent from this website: $https://www.worldgovernmentbonds.com/?utm_content = cmp - true$. The policy rate from Norges bank is per now 4,5 percent and since we have short term call options for 1, 3 and 6 months here I would use 4,5 percent as the risk free interest rate. I have also set S0 = 1000.

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
from scipy.optimize import brentq
import yfinance as yf
import pandas as pd


r = 0.045
S0 = 1000

sigma_n = [0.15, 0.30, 0.45]
Time = [1/12, 3/12, 4/12]

#Make an array of K_1, K_2, K_3
option_price = [0.7, 0.9, 1, 1.1, 1.3]
K_n = [S0 * n for n in option_price]

#a)
#Function for calculating call price using Black-Scholes
def black_scholes(S0, K, T, r, sigma):
    d1 = (np.log(S0 / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
```

```
    call_price = S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    return call_price

call_prices_list = []

#plot
fig, axs = plt.subplots(1, len(Time), figsize=(18, 6))
for index, T in enumerate(Time):
    ax = axs[index]
    for sigma in sigma_n:
        call_prices = []
        for K in K_n:
            call_price = black_scholes(S0, K, T, r, sigma)
            call_prices.append(call_price)
            call_prices_list.append({
                'Time': T * 12,
                'Sigma': sigma * 100,
                'Strike Price': K,
                'Call Option Price': call_price
            })
        ax.plot(K_n, call_prices, marker='o', label=f' = {sigma*100:.0f}%')
    ax.set_title(f'Call Option Prices for T = {T*12:.0f} Months')
    ax.set_xlabel('Strike Price (K)')
    ax.set_ylabel('Call Option Price (C)')
    ax.legend()
    ax.grid(True)

plt.tight_layout()
plt.show()

# Create a DataFrame from the call_prices_list
call_prices_df = pd.DataFrame(call_prices_list)
#Organize data
df = call_prices_df.pivot_table(
    index=['Time', 'Strike Price'],
    columns='Sigma',
    values='Call Option Price'
)

pd.options.display.float_format = '{:.2f}'.format
print(df)
```

```
Sigma              15.00  30.00  45.00
Time Strike Price
1.00 700.00       302.62 302.62 302.71
     900.00       103.45 107.43 117.13
```

```
     1000.00       19.18  36.38  53.58
     1100.00        0.28   6.76  19.25
     1300.00        0.00   0.04   1.28
3.00 700.00       307.83 308.12 311.81
     900.00       111.90 127.74 150.74
     1000.00       35.67  65.21  94.78
     1100.00        5.15  28.09  56.07
     1300.00        0.01   3.30  16.89
4.00 700.00       310.42 311.24 317.77
     900.00       116.46 136.85 164.40
     1000.00       42.24  76.20 110.20
     1100.00        8.60  37.66  70.94
     1300.00        0.06   6.74  26.67
```



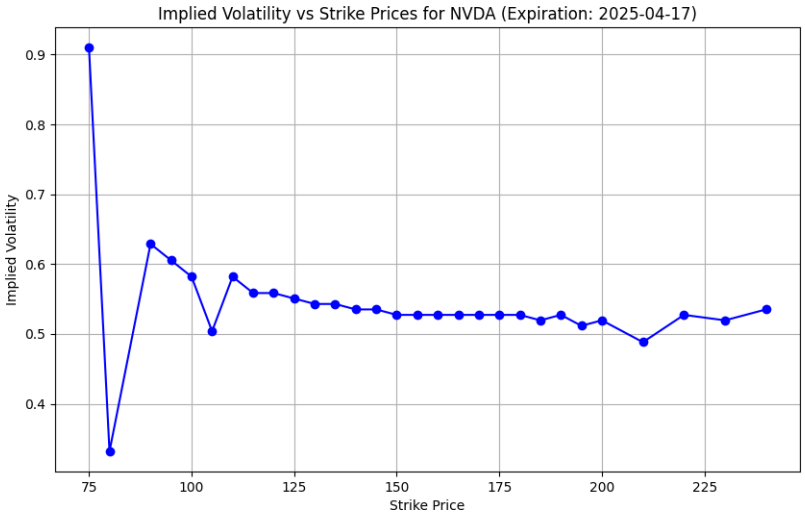
We see that the call option prices decreases as strike prices increases the price of the call option decreases. This is because the probability of profit is going down.

b

```
    #b)
#Function for calculating implied volatility using the hint
def find_implied_volatility(S0, K, T, r, market_price, tol=0.005, max_iter=100):
    # Objective function for implied volatility
    def objective(sigma):
        return black_scholes(S0, K, T, r, sigma) - market_price
    sigma_lower = 1e-6
    sigma_upper = 2.0

    f_lower = objective(sigma_lower)
    f_upper = objective(sigma_upper)
```

```
    if f_lower * f_upper > 0:
        # No sign change: adjust bounds or return NaN
        return np.nan

    for _ in range(max_iter):
        sigma_mid = (sigma_lower + sigma_upper) / 2
        f_mid = objective(sigma_mid)

        if (sigma_upper - sigma_lower) / 2 < tol:
            return sigma_mid

        if f_mid * f_lower < 0:
            sigma_upper = sigma_mid
            f_upper = f_mid
        else:
            sigma_lower = sigma_mid
            f_lower = f_mid

    return (sigma_lower + sigma_upper) / 2

#Using NVDA
ticker = yf.Ticker('NVDA')
current_price = ticker.history(period="1d")['Close'].iloc[-1]
S0 = current_price  #Underlying asset price
#Expiration dates for NVDA options
expirations = ticker.options
#Colect option with expiration 2025-04-17
expiration = expirations[11]
option_chain = ticker.option_chain(expiration)

#Extract call options
calls = option_chain.calls

#Time to expiration
today = pd.Timestamp('today').normalize()
expiration_date = pd.to_datetime(expiration)
T = (expiration_date - today).days / 365.0

strike_prices = []
implied_volatilities = []

for idx, row in calls.iterrows():
    strike = row['strike']
    market_price = row['lastPrice']

    if np.isnan(market_price) or market_price == 0:
```

```
            continue

        iv = find_implied_volatility(S0, strike, T, r, market_price)
        if np.isnan(iv):
            continue
        strike_prices.append(strike)
        implied_volatilities.append(iv)

# Plot Implied Volatility vs Strike Prices
plt.figure(figsize=(10, 6))
plt.plot(strike_prices, implied_volatilities, marker='o', linestyle='-', color='blue')
plt.title(f'Implied Volatility vs Strike Prices for NVDA (Expiration: {expiration})')
plt.xlabel('Strike Price')
plt.ylabel('Implied Volatility')
plt.grid(True)
plt.show()
print(expirations)
```



We see that the implied volatility is quite stable after around 125 USD, which do make sense since the underlying stock price is now around 135 USD. Its therefore unlikely, or less risky to believe that the stock will fall below these values greater than this in the enxt 6 months, and therefore stable and quite low implied volatility. However, for strike prices around 80 USD we see a big drop in implied volatility. The market sees it very unlikely that this is possible during the option exercise time. We also have a very high Implied volatility for 75 USD, we then have very high IV at 75 USD and very low IV at 80 USD which I find quite hard to understand fundamentally.

c

Since one of the assumptions of the black-scholes formula is that the risk free rate and the volatility of the underlying asset are known and constant, which are not the case in the real world, the implied volatilities for a call option would be the same for all strike prices. SO using the plot above as an example and NVDAs volatility is constant at 0.5, the plot would just be straight line from 0.5 for all the different strike prices.