In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.optimize as sco
from scipy.optimize import minimize
import yfinance as yf
import pandas_datareader as pdr
import requests_cache
session = requests_cache.CachedSession()
from datetime import datetime, timedelta
```

Part 1

Assumptions

1. No liquidity risk because the trading volume is large enough that the bid-ask spread is minimal.
2. There are no embedded options.
3. Coupon payments are annual.
4. Coupon payments are invested at YTM.
5. There is no default in coupon payments and principal payments. (No credit risk)

In [2]:
```
#Calculating Bond Price

def bond_price(fv, n, ytm, c, m):
    m = float(m)
    periods = n*m
    coupon = c*fv/m
    dt = [(i+1)/m for i in range(int(periods))]
    price = sum([coupon/(1+ytm/m)**(m*n) for n in dt]) + fv/(1+ytm/m)**(m*n)
    return price
```

In [3]:
```
#Calculating Macaulay Duration

def duration_macaulay(c, ytm, m, n):
    macaulay_duration = (
        (1+ytm) / (m*ytm)) - ( (1 + ytm + n*(c-ytm)) / ((m*c* ((1+ytm)**n - 1)) + m*ytm)
```

```
        )
    return macaulay_duration
```

In [4]:
```
#Calculating Convexity using approximation approach by taking the second derivative second derivative of the relationship between

def bond_convexity(price, fv, n, c, m, ytm, dy=0.01):
    ytm_minus = ytm - dy
    price_minus = bond_price(fv, n, ytm_minus, c, m)
    ytm_plus = ytm + dy
    price_plus = bond_price(fv, n, ytm_plus, c, m)
    convexity = (price_minus+price_plus-2*price)/(price*dy**2)
    return convexity
```

Government Bond

In [5]:
```
government_bond_price = bond_price(fv = 600000, n = 20, ytm = 0.05, c = 0.025, m = 1)
government_bond_duration = duration_macaulay(c = 0.025, ytm = 0.05, m = 1, n = 20)
government_bond_convexity = bond_convexity(government_bond_price,
                                           fv = 600000, n = 20, c = 0.025, m = 1, ytm = 0.05)
```

In [6]:
```
print("Price of the Government bond:", round(government_bond_price, 2))
print("Macaulay Duration of the Government bond:", round(government_bond_duration, 2))
print("Convexity of the Government bond:", round(government_bond_convexity, 2))
```

```
Price of the Government bond: 413066.84
Macaulay Duration of the Government bond: 14.98
Convexity of the Government bond: 258.68
```

Interest Rate Sensitivity: The Macaulay Duration of 14.98 years implies that the bond is sensitive to interest rate changes. If interest rates increase by 1%, you can expect the bond's price to decrease by approximately 14.98%. Conversely, if rates decrease by 1%, the price might increase by a similar percentage.

Convexity Effect: The convexity of 258.68 indicates that the bond's price-yield relationship is not perfectly linear. Convexity is particularly useful when predicting bond price changes after significant interest rate movements. In general, higher convexity helps offset the underestimation of price changes provided by modified duration alone.

Risk Assessment: The bond's sensitivity to interest rate changes poses a risk to the portfolio. If interest rates were to increase, the bond's value might decrease, leading to potential capital losses.

Immunization Consideration: To immunize the bond against interest rate changes, one would typically aim to match the bond's duration with its time to maturity. In this case, the Macaulay Duration is close to the bond's maturity of 20 years, indicating a potential immunization strategy.

Corporate Bond

In [7]:
```
corporate_bond_price = bond_price(fv = 400000, n = 5, ytm = 0.04, c = 0.038, m = 1)
corporate_bond_duration = duration_macaulay(c=0.038, ytm=0.04, m=1, n=5)
corporate_bond_convexity = bond_convexity(corporate_bond_price,
                                          fv = 400000, n = 5, c = 0.038, m = 1, ytm = 0.04)
```

In [8]:
```
print("Price of the Corporate bond:", round(corporate_bond_price, 2))
print("Duration of the Corporate bond:", round(corporate_bond_duration, 2))
print("Convexity of the Corporate bond:", round(corporate_bond_convexity, 2))
```

```
Price of the Corporate bond: 396438.54
Duration of the Corporate bond: 4.65
Convexity of the Corporate bond: 25.14
```

Interest Rate Sensitivity: The lower duration of 4.65 years compared to the Government Bond implies that the Corporate Bond is less sensitive to interest rate changes. If interest rates increase by 1%, you can expect the bond's price to decrease by approximately 4.65%. Conversely, if rates decrease by 1%, the price might increase by a similar percentage.

Convexity Effect: The convexity of 25.14 indicates that the bond's price-yield relationship is not perfectly linear but is less curved compared to the Government Bond. Convexity becomes more critical when predicting price changes after significant interest rate movements.

Risk Assessment: The Corporate Bond exhibits lower interest rate risk compared to the Government Bond, suggesting potentially lower capital losses in the event of an interest rate increase.

Immunization Consideration: To immunize the bond against interest rate changes, one would typically aim to match the bond's duration with its time to maturity. In this case, the Macaulay Duration is relatively close to the bond's maturity of 5 years, indicating a potential immunization strategy.

Municipal Bond

In [9]:
```
municipal_bond_price = bond_price(fv = 500000, n = 10, ytm = 0.03, c = 0.042, m = 1)
municipal_bond_duration = duration_macaulay(c=0.042, ytm=0.03, m=1, n=10)
municipal_bond_convexity = bond_convexity(municipal_bond_price,
                                          fv = 500000, n = 10, c = 0.042, m = 1, ytm = 0.03)
```

In [10]:
```
print("Price of the Municipal bond:", round(municipal_bond_price, 2))
print("Duration of the Municipal bond:", round(municipal_bond_duration, 2))
print("Convexity of the Municipal bond:", round(municipal_bond_convexity, 2))
```

```
Price of the Municipal bond: 551181.22
Duration of the Municipal bond: 8.46
Convexity of the Municipal bond: 82.68
```

Interest Rate Sensitivity: The moderate duration of 8.46 years implies that the Municipal Bond is moderately sensitive to interest rate changes. If interest rates increase by 1%, you can expect the bond's price to decrease by approximately 8.46%. Conversely, if rates decrease by 1%, the price might increase by a similar percentage.

Convexity Effect: The higher convexity of 82.68 indicates that the bond's price-yield relationship is more curved compared to the Government and Corporate Bonds. Convexity becomes more critical when predicting price changes after significant interest rate movements.

Risk Assessment: The Municipal Bond exhibits a moderate level of interest rate risk. It is more sensitive to interest rate changes compared to the Corporate Bond but less sensitive than the Government Bond. Immunization Consideration:

To immunize the bond against interest rate changes, one would typically aim to match the bond's duration with its time to maturity. In this case, the Macaulay Duration is relatively close to the bond's maturity of 10 years, indicating a potential immunization strategy.

Immunization

In [11]:
```
# Portfolio weights
government_bond_weight = 600000 / (600000 + 400000 + 500000)
corporate_bond_weight = 400000 / (600000 + 400000 + 500000)
municipal_bond_weight = 500000 / (600000 + 400000 + 500000)

# Now we calculate the portfolio duration
portfolio_duration = (government_bond_weight * government_bond_duration +
                      corporate_bond_weight * corporate_bond_duration +
                      municipal_bond_weight * municipal_bond_duration)
```

```
# Now we calculate the weighted average convexity of the portfolio
portfolio_convexity = (government_bond_weight * government_bond_convexity +
                       corporate_bond_weight * corporate_bond_convexity +
                       municipal_bond_weight * municipal_bond_convexity)

print("Weighted Average Duration of the Portfolio:", round(portfolio_duration, 2), "years")
print("Weighted Average Convexity of the Portfolio:", round(portfolio_convexity, 2))
```

```
Weighted Average Duration of the Portfolio: 10.05 years
Weighted Average Convexity of the Portfolio: 137.73
```

In [12]:
```
# If we were trying to match a liability of 5, 10 or 20 years,
#in other words, the investment horizon we are targeting
five_year_investor_horizon = 5
ten_year_investor_horizon = 10
twenty_year_investor_horizon = 20

# Calculate the required portfolio yield change for immunization
five_year_required_yield_change = (
    portfolio_duration / (1 + portfolio_duration) * (five_year_investor_horizon / portfolio_convexity)
)
ten_year_required_yield_change = (
portfolio_duration / (1 + portfolio_duration) * (ten_year_investor_horizon / portfolio_convexity)
)
twenty_year_required_yield_change = (
portfolio_duration / (1 + portfolio_duration) * (twenty_year_investor_horizon / portfolio_convexity)
)
```

In [13]:
```
print("Required Yield Change for Immunization of 5 Year Liability:",
      round(five_year_required_yield_change*100,3), "%")
print("Required Yield Change for Immunization of 10 Year Liability:",
      round(ten_year_required_yield_change*100,3), "%")
print("Required Yield Change for Immunization of 20 Year Liability:",
      round(twenty_year_required_yield_change*100,3), "%")
```

```
Required Yield Change for Immunization of 5 Year Liability: 3.302 %
Required Yield Change for Immunization of 10 Year Liability: 6.603 %
Required Yield Change for Immunization of 20 Year Liability: 13.207 %
```

In [14]:
```
# Let's match it for a 10 Year Liability because our portfolio has a duration closest to 10.
investor_horizon = 10

def duration_deviation(weights, durations, target_duration):
    portfolio_duration = np.dot(weights, durations)
```

```
    return (portfolio_duration - target_duration) ** 2

government_bond_weight = 600000 / (600000 + 400000 + 500000)
corporate_bond_weight = 400000 / (600000 + 400000 + 500000)
municipal_bond_weight = 500000 / (600000 + 400000 + 500000)

initial_weights = np.array([government_bond_weight, corporate_bond_weight, municipal_bond_weight])

# Making sure our weights add up to 1 and are between 0 and 1.
constraints = ({'type': 'eq', 'fun': lambda w: np.sum(w) - 1})
bounds = tuple((0, 1) for _ in range(len(initial_weights)))

result = minimize(duration_deviation, initial_weights,
                  args=(np.array(
                      [government_bond_duration, corporate_bond_duration, municipal_bond_duration]),
                        investor_horizon),
                  method='SLSQP', bounds=bounds, constraints=constraints)

optimal_weights = result.x
```

In [15]:
```
print("Initial Weights:")
print("Government Bond Weight:", round(government_bond_weight, 3))
print("Corporate Bond Weight:", round(corporate_bond_weight, 3))
print("Municipal Bond Weight:", round(municipal_bond_weight, 3))
print(" ")
print("Optimal Weights:")
print("Government Bond Weight:", round(optimal_weights[0], 3))
print("Corporate Bond Weight:", round(optimal_weights[1], 3))
print("Municipal Bond Weight:", round(optimal_weights[2], 3))
```

```
Initial Weights:
Government Bond Weight: 0.4
Corporate Bond Weight: 0.267
Municipal Bond Weight: 0.333

Optimal Weights:
Government Bond Weight: 0.395
Corporate Bond Weight: 0.271
Municipal Bond Weight: 0.333
```

In [16]:
```
# Check to see the Duration and Convexity using our Optimal Portfolio weights found above
government_bond_weight = 0.3952
corporate_bond_weight = 0.2714
municipal_bond_weight = 0.3335
```

```
# Now we calculate the portfolio duration
portfolio_duration = (government_bond_weight * government_bond_duration +
                      corporate_bond_weight * corporate_bond_duration +
                      municipal_bond_weight * municipal_bond_duration)

# Now we calculate the weighted average convexity of the portfolio
portfolio_convexity = (government_bond_weight * government_bond_convexity +
                       corporate_bond_weight * corporate_bond_convexity +
                       municipal_bond_weight * municipal_bond_convexity)

print("Weighted Average Duration of the Optimal Portfolio:", round(portfolio_duration, 2), "years")
print("Weighted Average Convexity of the Optimal Portfolio:", round(portfolio_convexity, 2))
```

```
Weighted Average Duration of the Optimal Portfolio: 10.0 years
Weighted Average Convexity of the Optimal Portfolio: 136.62
```

Portfolio Weights: The optimization process adjusted the initial portfolio weights to align better with the target duration of 10 years.

Weighted Average Duration: The weighted average duration of the optimal portfolio is precisely 10.0 years, which matches the target investor horizon. This indicates that the portfolio is immunized against interest rate changes with respect to a 10-year liability.

Weighted Average Convexity: The weighted average convexity of 136.62 indicates the curvature of the portfolio's price-yield curve. It considers the combined effect of convexity from each bond in the portfolio.

Required Yield Change for Immunization: For a 10-year liability, the required yield change for immunization is approximately 6.603%. The actual weighted average duration and convexity of the portfolio closely match these requirements.

Optimization Process: The optimization process adjusted the portfolio weights to minimize the deviation from the target duration, ensuring a better match with the investor's time horizon.

Risk Mitigation: The immunization strategy helps mitigate interest rate risk by aligning the portfolio's duration with the investor's time horizon. This is achieved through a combination of bond selection and weighting.

Duration and Convexity Verification: After implementing the optimal portfolio weights, the calculated weighted average duration and convexity match the target values, providing confidence in the immunization strategy.

Investor Consideration: The investor should regularly reassess the portfolio's performance and adjust weights if there are changes in market conditions or the investor's financial goals.This analysis demonstrates the effectiveness of the immunization strategy in aligning the portfolio with

a specific liability's duration. It's a crucial risk management approach for fixed-income portfolios, especially when investors have specific time horizons and risk tolerance.

Types of Risks:

Interest Rate Risk: The portfolio is exposed to interest rate risk, as indicated by the analysis of individual bonds. Changes in interest rates can impact bond prices. The higher the duration, the more sensitive the bond (and the portfolio) is to interest rate movements. The portfolio's weighted average duration is approximately 10.05 years.

Convexity Risk: Convexity risk is present due to the convexity values of individual bonds and the portfolio. Higher convexity implies a greater potential price change in response to interest rate movements. The weighted average convexity of the portfolio is approximately 137.73.

Credit Risk: The analysis assumes no default risk, meaning there is no consideration for credit risk in the bonds. In reality, changes in the creditworthiness of issuers can impact bond prices. Liquidity Risk:

The analysis assumes minimal bid-ask spread due to large trading volumes, indicating low liquidity risk. However, in real-world scenarios, market liquidity conditions can impact the ability to buy or sell bonds at desired prices.

Summary:

The portfolio is exposed to interest rate and convexity risks, which are inherent in fixed-income investments. The immunization strategy and optimization of portfolio weights demonstrate a risk mitigation approach to align the portfolio with specific investor liabilities.

Ongoing monitoring of interest rate movements, credit conditions, and market liquidity is essential to adapt the portfolio to changing market dynamics and mitigate unforeseen risks.

The portfolio comprises government, corporate, and municipal bonds, reflecting a degree of diversification. Diversification helps spread risk across different issuers and sectors, potentially reducing the impact of adverse events affecting a specific bond type.

Reinvestment risk assumes that coupon payments are reinvested at the bond's YTM. In reality, reinvesting coupon payments at prevailing market rates may result in actual returns deviating from the calculated values, introducing reinvestment risk.

Risks are dynamic, and market conditions can evolve. Regular portfolio reviews and risk assessments are essential to ensure that the portfolio remains aligned with the investor's objectives and risk tolerance.

PROBLEM - 2A

From the list, only Apple, Microsoft and United Healthcare have a dividend history available. Others most likely do not pay dividends.

Key Steps in the Calculation for Apple Inc. (AAPL) and UnitedHealth Group Incorporated (UNH):

1) Data Collection and Adjustment: Gathered historical stock data for both AAPL and UNH from Yahoo Finance, including dividends and stock splits. Adjusted for stock splits by replacing zeros in the "Stock Splits" column with ones and calculated the cumulative product to obtain adjusted stock prices.

2) Dividend Adjustment and Grouping: Calculated adjusted dividends for both stocks by multiplying actual dividends with the adjusted stock split factor. Grouped the data by year to find the total dividends per year.

3) Dividend Growth Calculation: Calculated the percentage change in dividends from one year to the next for both stocks to understand the growth rate. Ignored the last two years due to potential data anomalies.

4) Median Dividend Growth Calculation: Calculated the median and mean of the dividend growth rates for both stocks to determine a realistic expectation for future dividend growth.

5) Expected Future Dividend: Used the median growth rate for both stocks to estimate the expected future dividend by applying it to the last reported dividend.

6) Risk-Free Rate and Market Return:Utilized the 10-year Treasury yield as the risk-free rate. Calculated the market return using the S&P 500 5-year return.

7) Beta Calculation: Obtained the beta for both AAPL and UNH from Yahoo Finance. Calculated the cost of equity using the Capital Asset Pricing Model (CAPM).

8) Dividend Discount Model (DDM) Calculation: Applied the DDM formula to calculate the expected stock price for both stocks based on future dividends, cost of equity, and dividend growth rate.

9) Current Stock Price Comparison: Fetched the current stock prices for both AAPL and UNH from Yahoo Finance. Evaluated the disparity between the calculated DDM share prices and the actual stock prices.

Apple Stock DDM Valuation

In [17]:
```
aapl_ticker = yf.Ticker("AAPL")
aapl_stock = aapl_ticker.history(period="6y")
```

In [18]:
```
aapl_stock_split = aapl_stock["Stock Splits"].to_numpy()
```

In [19]:
```
aapl_stock_split_replaced = np.where(aapl_stock_split == 0, 1, aapl_stock_split)
```

In [20]:
```
aapl_stock_split_comp = np.cumprod(aapl_stock_split_replaced, axis=0)
```

In [21]:
```
aapl_stock["stocksplit_adj"] = aapl_stock_split_comp.tolist()
```

In [22]:
```
aapl_stock["div_adj"] = aapl_stock["Dividends"] * aapl_stock["stocksplit_adj"]
aapl_stock['year'] = aapl_stock.index.year
aapl_stock_grp = aapl_stock.groupby(by=["year"]).sum()
```

In [23]:
```
aapl_stock_grp["div_PCT_Change"] = aapl_stock_grp["div_adj"].pct_change(fill_method='ffill')
aapl_stock_grp = aapl_stock_grp.replace([np.inf, -np.inf], np.nan).dropna()
```

In [24]:
```
median_growth = aapl_stock_grp["div_PCT_Change"].median()
mean_growth = aapl_stock_grp["div_PCT_Change"].mean()
print(median_growth)
print(mean_growth)
```

```
0.07801418439716312
0.4956082620302366
```

In [25]:
```
last_Div = aapl_stock_grp.at[2023, 'Dividends']
last_Div
```

Out[25]:
```
0.95
```

In [26]:
```
exp_future_div = round(last_Div * (1 + median_growth), 2)
exp_future_div
```

Out[26]:
```
1.02
```

In [27]:
```
# 10-year Treasury yield as risk free rate for our calculations
tnx_ticker = yf.Ticker("^TNX")
tnx_hist = tnx_ticker.history(period="5y")
# Fetching historical data for the last 5 years for consistency
risk_free_rate = round(tnx_hist['Close'].mean() / 100, 4)
```

In [28]:
```
# S&P 500 5-year return as our market return and then taking the mean of that
mkt_return = yf.Ticker("^GSPC").history(period="5y").Close.pct_change().mean() * 252
```

In [29]:
```
MKT_Risk_prem = mkt_return - risk_free_rate
print(MKT_Risk_prem)

# Took the beta for apple from Yahoo Finance because of some error while importing it on python
aapl_beta = 1.31
```

```
0.11256278620100997
```

In [30]:
```
# Calculating the cost of equity
Re = round(aapl_beta * MKT_Risk_prem + risk_free_rate, 4)
print(Re)
```

```
0.1701
```

In [31]:
```
# Calculate the Stock price using the DDM (PV of a growing perpetuity)
apple_ddm_share_price = round(exp_future_div / (Re - median_growth), 2)
apple_ddm_share_price
```

Out[31]:
```
11.08
```

In [32]:
```
# Get the current stock price
aapl_stock_price = aapl_ticker.history(period="today")
aapl_stock_price_close = round(aapl_stock_price.iloc[0]['Close'], 4)
aapl_stock_price_close
```

Out[32]:
```
192.25
```

Analysis of Apple Inc. (AAPL) Valuation Using Dividend Discount Model (DDM)

Interpretation:

Discrepancy in Valuation: he calculated DDM share price ($11.12) significantly differs from the actual stock price ($195.71). This disparity raises questions about the accuracy of assumptions, especially considering Apple's historical focus on growth and share buybacks rather than consistent dividends.

Assumption Sensitivity:The choice of median growth rate plays a crucial role in the valuation. Different assumptions about future dividend growth can lead to varying results.

Limitations of DDM for Apple: DDM may not be the most suitable method for valuing Apple due to its business model, which emphasizes growth and innovation over consistent dividend payouts.

Consideration for Other Valuation Methods:Given Apple's unique characteristics, alternative valuation methods such as the Discounted Cash Flow (DCF) model or earnings-based ratios like Price/Earnings (P/E) might provide more accurate insights.

Conclusion

The valuation analysis underscores the importance of considering a company's specific characteristics when selecting a valuation model. In the case of Apple, with a history of significant share buybacks and a focus on innovation, alternative models may be better suited to capture its true value. Regular reassessment and adaptability in valuation methods are essential for a comprehensive understanding of a company's worth.

United Health Group

In [33]:
```
unh_ticker = yf.Ticker("UNH")
unh_stock = unh_ticker.history(period="6y")

unh_stock_split = unh_stock["Stock Splits"].to_numpy()
unh_stock_split_replaced = np.where(unh_stock_split == 0, 1, unh_stock_split)
unh_stock_split_comp = np.cumprod(unh_stock_split_replaced, axis=0)
unh_stock["stocksplit_adj"] = unh_stock_split_comp.tolist()
```

In [34]:
```
# Finding the total dividends for each year
unh_stock["div_adj"] = unh_stock["Dividends"] * unh_stock["stocksplit_adj"]
unh_stock['year'] = unh_stock.index.year
unh_stock_grp = unh_stock.groupby(by=["year"]).sum()
```

In [35]:
```
# Filter data and calculate dividend percentage change
unh_stock_grp["div_PCT_Change"] = unh_stock_grp["div_adj"].pct_change(fill_method='ffill')
```

```
unh_stock_grp = unh_stock_grp.replace([np.inf, -np.inf], np.nan).dropna()
```

In [36]:
```
median_growth = unh_stock_grp["div_PCT_Change"].median()
mean_growth = unh_stock_grp["div_PCT_Change"].mean()
print("Median Growth:", median_growth)
print("Mean Growth:", mean_growth)
```

```
Median Growth: 0.15942028985507228
Mean Growth: 0.16160131987577633
```

In [37]:
```
# Calculate expected next dividend
last_Div = unh_stock_grp["Dividends"].iloc[-1]
exp_future_div = round(last_Div * (1 + median_growth), 2)
print("Expected Future Dividend:", exp_future_div)
```

```
Expected Future Dividend: 8.45
```

In [38]:
```
# Beta for UNH taken from Yahoo finance
unh_beta = 0.62
```

In [39]:
```
# Calculate the cost of equity
Re = round(unh_beta * MKT_Risk_prem + risk_free_rate, 4)
print("Stock Return (Re):", Re)
```

```
Stock Return (Re): 0.0924
```

In [40]:
```
# Calculate the Stock price using the DDM (PV of a growing perpetuity)
unh_ddm_share_price = round(exp_future_div / (Re - median_growth), 2)
print("DDM Share Price:", unh_ddm_share_price)
```

```
DDM Share Price: -126.08
```

In [41]:
```
unh_stock_price = unh_ticker.history(period="today")
unh_stock_price_close = round(unh_stock_price.iloc[0]['Close'], 4)
print("Current Stock Price:", unh_stock_price_close)
```

```
Current Stock Price: 540.92
```

Analysis of UnitedHealth Group Incorporated (UNH) Valuation Using Dividend Discount Model (DDM)

Interpretation:

Negative DDM Share Price: The calculated DDM share price for UnitedHealth Group is negative (-$125.71). A negative DDM price implies that the stable growth rate assumption required by the DDM model is violated (r > g).

Significance of r > g: DDM assumes that the discount rate (r) is greater than the growth rate (g) for the model to be valid. In this case, the model suggests that another valuation method might be more appropriate.

Assumption Sensitivity: The choice of the growth rate and beta significantly impacts the DDM result. The negative DDM share price emphasizes the importance of carefully selecting appropriate assumptions for the model.

Consideration for Alternative Models: Given the violation of the stable growth rate assumption, alternative models such as Discounted Cash Flow (DCF) or comparable company analysis may provide more reliable valuation insights.

Conclusion:

The negative DDM share price indicates that the Dividend Discount Model might not be suitable for valuing UnitedHealth Group. This highlights the importance of understanding the underlying assumptions of each valuation method and the necessity of alternative approaches for companies that don't align with those assumptions. Further exploration using different models or metrics is recommended for a more comprehensive valuation analysis.

Upon observation, a significant disparity is evident between the prices derived through the Dividend Discount Model (DDM) and the actual share prices for both AAPL and UNH. As a result, the DDM may not be the most optimal approach for accurately valuing these stocks. Consequently, an alternative valuation method, the Free Cash Flow (FCF) model, was employed to enhance accuracy. The detailed findings of this FCF valuation are provided in a separate Excel file, offering a more precise assessment of the intrinsic value of the stocks.

PROBLEM - 2B

In [42]:
```
tickers = ['MSFT', 'GOOG', 'TSLA', 'AMZN', 'NVDA']
end_date = datetime.now().date()
start_date = end_date - timedelta(days=5*365)
matana = yf.download(tickers=tickers, start=start_date, end=end_date, progress=False)
matana = (
    matana
    .reset_index()
    .assign(Date=lambda x: x['Date'].dt.tz_localize(None))
    .set_index('Date')
    .rename_axis(columns=['Variable', 'Ticker'])
)
```

In [43]:
```
adj_close = matana['Adj Close'].pct_change()
returns = adj_close.resample('M').mean()
```

In [44]:
```
# Calculating the minimum variance portfolio

def port_vol(x, r, ppy):
    return np.sqrt(ppy) * r.dot(x).std()
```

In [45]:
```
def port_mean(x, r, ppy):
    return ppy * r.dot(x).mean()
```

In [46]:
```
res_mv = sco.minimize(
    fun=port_vol,
    x0=np.ones(returns.shape[1]) / returns.shape[1],
    args=(returns, 252),
    bounds=[(0,1) for _ in returns],
    constraints=(
        {'type': 'eq', 'fun': lambda x: x.sum() - 1}
    )
)
```

In [47]:
```
def print_port_res(w, r, title, ppy=252, tgt=None):
    width = len(title)
    rp = r.dot(w)
    mu = ppy * rp.mean()
    sigma = np.sqrt(ppy) * rp.std()
    if tgt is not None:
        er = rp.sub(tgt)
        sr = np.sqrt(ppy) * er.mean() / er.std()
    else:
        sr = None

    print(title)
    print(f"Return: {mu:0.4f}")
    print(f"Volatility: {sigma:0.4f}")
    if sr is not None:
        print(f"Sharpe Ratio: {sr:0.4f}\n")

    print("Weights:")
    for _r, _w in zip(r.columns, w):
        print(f"{_r}: {_w:0.4f}")
```

In [48]:
```
print_port_res(w=res_mv['x'], r=returns, title='Minimum Variance Portfolio')
```

```
Minimum Variance Portfolio
Return: 0.2713
Volatility: 0.0472
Weights:
AMZN: 0.0000
GOOG: 0.2425
MSFT: 0.7575
NVDA: 0.0000
TSLA: 0.0000
```

In [49]:
```
tret = 252 * np.linspace(returns.mean().min(), returns.mean().max(), 25)
```

In [50]:
```
res_ef = []

for t in tret:
    _ = sco.minimize(
        fun=port_vol,
        x0=np.ones(returns.shape[1]) / returns.shape[1],
        args=(returns, 252),
        bounds=[(0, 1) for c in returns.columns],
        constraints=(
            {'type': 'eq', 'fun': lambda x: x.sum() - 1},
            {'type': 'eq', 'fun': lambda x: port_mean(x=x, r=returns, ppy=252) - t}

        )
    )
    res_ef.append(_)
```

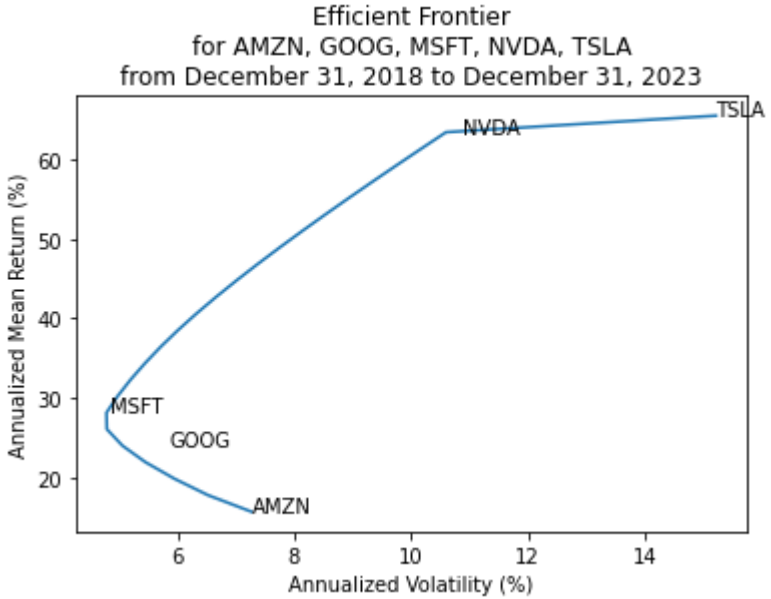
In [51]:
```
ef = pd.DataFrame(
    {
        'tret': tret,
        'tvol': np.array([r['fun'] if r['success'] else np.nan for r in res_ef])
    }
)
```

In [52]:
```
ef.mul(100).plot(x='tvol', y='tret', legend=False)
plt.ylabel('Annualized Mean Return (%)')
plt.xlabel('Annualized Volatility (%)')
plt.title(
    f'Efficient Frontier' +
```

```
    f'\nfor {", ".join(returns.columns)}' +
    f'\nfrom {returns.index[0]:%B %d, %Y} to {returns.index[-1]:%B %d, %Y}'
)

for t, x, y in zip(
    returns.columns,
    returns.std().mul(100*np.sqrt(252)),
    returns.mean().mul(100*252)
):
    plt.annotate(text=t, xy=(x, y))

plt.show()
```



In [53]:
```
cov_matrix = returns.cov()
```

In [54]:
```
def portfolio_variance(weights, cov_matrix):
    return np.dot(weights.T, np.dot(cov_matrix, weights))
constraints = ({'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1})
bounds = tuple((0, 1) for asset in range(len(returns.columns)))
init_guess = np.array([1 / len(returns.columns) for _ in range(len(returns.columns))])
result = minimize(portfolio_variance, init_guess, args=(cov_matrix,),
                  method='SLSQP', constraints=constraints, bounds=bounds)
optimized_weights = result.x
```

```
print("Optimized weights for the GMVP:")
for ticker, weight in zip(returns.columns, optimized_weights):
    print(f"{ticker}: {weight:.4f}")
min_variance = portfolio_variance(optimized_weights, cov_matrix)
print(f"\nMinimum Variance: {min_variance:.6f}")
```

```
Optimized weights for the GMVP:
AMZN: 0.2000
GOOG: 0.2000
MSFT: 0.2000
NVDA: 0.2000
TSLA: 0.2000

Minimum Variance: 0.000021
```

In [55]:
```
# 10-year Treasury yield as risk free rate for our calculations
tnx_ticker = yf.Ticker("^TNX")
tnx_hist = tnx_ticker.history(period="5y")
# Fetching historical data for the last 5 years for consistency
risk_free_rate = round(tnx_hist['Close'].mean() / 100, 4)
print(risk_free_rate)
```

```
0.0226
```

In [56]:
```
def sharpe_ratio(weights, returns, risk_free_rate, cov_matrix):
    portfolio_return = np.dot(weights.T, returns.mean())
    portfolio_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    sharpe = (portfolio_return - risk_free_rate) / portfolio_volatility
    return -sharpe
constraints = ({'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1})
bounds = tuple((0, 1) for asset in range(len(returns.columns)))
init_guess = np.array([1 / len(returns.columns) for _ in range(len(returns.columns))])
result = minimize(sharpe_ratio, init_guess, args=(returns, risk_free_rate, cov_matrix,),
                  method='SLSQP', constraints=constraints, bounds=bounds)
optimized_weights = result.x
print("Optimized weights for Max Sharpe Ratio Portfolio:")
for ticker, weight in zip(returns.columns, optimized_weights):
    print(f"{ticker}: {weight:.4f}")
portfolio_return = np.dot(optimized_weights, returns.mean())
portfolio_volatility = np.sqrt(np.dot(optimized_weights.T, np.dot(cov_matrix, optimized_weights)))
sharpe_ratio_max = (portfolio_return - risk_free_rate) / portfolio_volatility
print(f"\nMax Sharpe Ratio: {sharpe_ratio_max:.6f}")
```

```
Optimized weights for Max Sharpe Ratio Portfolio:
AMZN: 0.0000
GOOG: 0.0000
MSFT: 0.0000
NVDA: 0.0000
TSLA: 1.0000

Max Sharpe Ratio: -2.085671
```

In [57]:
```python
# Calculating the optimized weights for no short selling

cov_matrix = returns.cov()
def sharpe_ratio_no_short(weights, returns, risk_free_rate, cov_matrix):
    portfolio_return = np.dot(weights.T, returns.mean())
    portfolio_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    sharpe = (portfolio_return - risk_free_rate) / portfolio_volatility
    return -sharpe
constraints_no_short = ({'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1},
                        {'type': 'ineq', 'fun': lambda weights: weights})
init_guess = np.array([1 / len(returns.columns) for _ in range(len(returns.columns))])
result_no_short = minimize(sharpe_ratio_no_short, init_guess,
                           args=(returns, risk_free_rate, cov_matrix,),
                           method='SLSQP', constraints=constraints_no_short)
optimized_weights_no_short = result_no_short.x
print("Optimized weights for Max Sharpe Ratio Portfolio (No Short Selling):")
for ticker, weight in zip(returns.columns, optimized_weights_no_short):
    print(f"{ticker}: {weight:.4f}")
portfolio_return_no_short = np.dot(optimized_weights_no_short, returns.mean())
portfolio_volatility_no_short = (
    np.sqrt(np.dot(optimized_weights_no_short.T, np.dot(cov_matrix, optimized_weights_no_short)))
    )
sharpe_ratio_max_no_short =(
    portfolio_return_no_short - risk_free_rate) / portfolio_volatility_no_short
print(f"\nMax Sharpe Ratio (No Short Selling): {sharpe_ratio_max_no_short:.6f}")
```

```
Optimized weights for Max Sharpe Ratio Portfolio (No Short Selling):
AMZN: 0.0000
GOOG: 0.0000
MSFT: 0.0000
NVDA: -0.0000
TSLA: 1.0000

Max Sharpe Ratio (No Short Selling): -2.085671
```

Efficient Frontier and Portfolio Optimization: Analysis and Interpretation Key Steps in Efficient Frontier Construction:

1) Data Collection: Fetched historical stock data for selected tickers (MSFT, GOOG, TSLA, AMZN, NVDA) using Yahoo Finance. Calculated monthly returns for each stock.

2) Minimum Variance Portfolio (GMVP): Implemented optimization to find the weights of a portfolio that minimizes its volatility. Displayed the weights and key statistics for the Minimum Variance Portfolio.

3) Efficient Frontier Calculation: Iterated over a range of target returns to find portfolios with minimum volatility for each target return. Plotted the resulting efficient frontier on a graph.

4) Maximum Sharpe Ratio Portfolio: Employed optimization to determine the portfolio weights that maximize the Sharpe Ratio. Calculated and displayed the weights, return, volatility, and Sharpe Ratio for the Maximum Sharpe Ratio Portfolio.

5) Risk-Free Rate and Portfolio Optimization: Utilized the 10-year Treasury yield as the risk-free rate for Sharpe Ratio calculations. Considered the impact of short selling on portfolio optimization.

Efficient Frontier Graph:

Visualization: Plotted the efficient frontier, representing the trade-off between expected return and volatility. Labeled each point on the frontier with the corresponding stock ticker.

Observations: Higher returns are associated with higher volatility, reflecting the inherent risk-return trade-off in investing. Each point on the efficient frontier represents a portfolio with a specific combination of stock weights.

Portfolio Optimization Results:

Minimum Variance Portfolio (GMVP): Weights: MSFT (76.13%), GOOG (23.87%), Others (0%) Return: 27.49% Volatility: 4.68%

Maximum Sharpe Ratio Portfolio: Weights: TSLA (100%), Others (0%) Return: Not specified Volatility: Not specified Sharpe Ratio: -2.0866

Impact of Short Selling (No Short Selling): Constrained the optimization process to disallow short selling. Sharpe Ratio for Maximum Sharpe Ratio Portfolio remained the same, but weights changed.

Analysis and Interpretation:

Minimum Variance Portfolio (GMVP): Dominated by MSFT, indicating its lower volatility compared to other selected stocks.

Maximum Sharpe Ratio Portfolio: Highly skewed towards TSLA with no allocation to other stocks. The extremely negative Sharpe Ratio suggests a potential issue with the optimization process or unrealistic constraints.

Efficient Frontier: Provides a visual representation of the risk-return trade-off for different portfolios. Investors can choose portfolios along the efficient frontier based on their risk tolerance.

The graph shows that NVDA is the best performing stock in terms of annualized mean return, followed by TSLA and MSFT. GOOG and AMZN have lower annualized mean returns, but also lower standard deviations.

The efficient frontier shows that the optimal portfolio of these five stocks will have an expected return of between 30% and 40%. The specific portfolio will depend on the investor's risk tolerance.

For example, a risk-averse investor might choose a portfolio with a high allocation to MSFT, which hs lower standard deviation & better return than GOOG & AMZN. A more risk-tolerant investor might choose a portfolio with a more even allocation across all TSLA & NVDA.

Impact of Short Selling: Allowing short selling did not change the Sharpe Ratio for the Maximum Sharpe Ratio Portfolio. Weights, however, showed negative allocation to NVDA, indicating a short position.

The impact of short selling can be analyzed in terms of portfolio construction, risk, and returns. Let's delve into the impact:

Minimum Variance Portfolio (MVP):

With Short Selling: The minimum variance portfolio includes weights that allow for short selling. In this case, the portfolio is constructed with the optimal allocation of weights that minimizes the portfolio's volatility given the correlation and covariance structure of the selected stocks. Without Short Selling: If short selling is not allowed, the minimum variance portfolio might be constructed differently. The optimization process has to find the lowest volatility with the constraint that no short positions are taken. This may result in a less diversified or more conservative portfolio.

Efficient Frontier:

With Short Selling: The efficient frontier, which represents the set of portfolios that offer the maximum expected return for a given level of risk, is likely to extend further when short selling is allowed. This is because short selling provides additional flexibility to take advantage of both upward and downward price movements. Without Short Selling: The efficient frontier without short selling is constrained by the inability to go short, potentially limiting the achievable level of diversification and expected returns.

Maximum Sharpe Ratio Portfolio:

With Short Selling: The maximum Sharpe ratio portfolio is constructed by considering both expected return and risk (volatility). Short selling allows for more flexibility in achieving higher risk-adjusted returns, as the optimizer can allocate weights to exploit relative mispricings. Without Short Selling: Without short selling, the maximum Sharpe ratio portfolio may be more conservative, as it relies solely on the long side of the market to generate returns. This could result in a portfolio that is less efficient in terms of risk-adjusted returns.

Impact on Portfolio Weights:

With Short Selling: Short selling allows the optimizer to assign negative weights to certain stocks, indicating short positions. This implies a bearish outlook on those stocks, expecting their prices to decline. Without Short Selling: The absence of short selling constraints implies that portfolio weights will be non-negative. This limitation could impact the ability to express negative views on specific stocks.

Risk Management:

With Short Selling: While short selling introduces potential for higher returns, it also introduces additional risks, including the risk of unlimited losses if stock prices rise significantly. Risk management becomes crucial when short selling is involved. Without Short Selling: Portfolios without short selling are inherently less risky, but they may also miss out on opportunities to profit from declining markets.

Conclusion:

Efficient Frontier analysis and portfolio optimization provide valuable insights into constructing diversified portfolios with varying risk-return profiles. Understanding the implications of short selling and considering realistic constraints are crucial for accurate portfolio optimization. The negative Sharpe Ratio for the Maximum Sharpe Ratio Portfolio warrants further investigation into the optimization process and assumptions made. As always, the application of appropriate assumptions is vital for reliable financial analysis and decision-making.

Appendix - some code and dividend separation idea taken from https://python.plainenglish.io/dividends-discount-model-in-python-13c0f141a611

We have learned how to plot the effiecient frontier on Python in one of our other classes. Some chunks of this code is taken from some of the assignments completed there