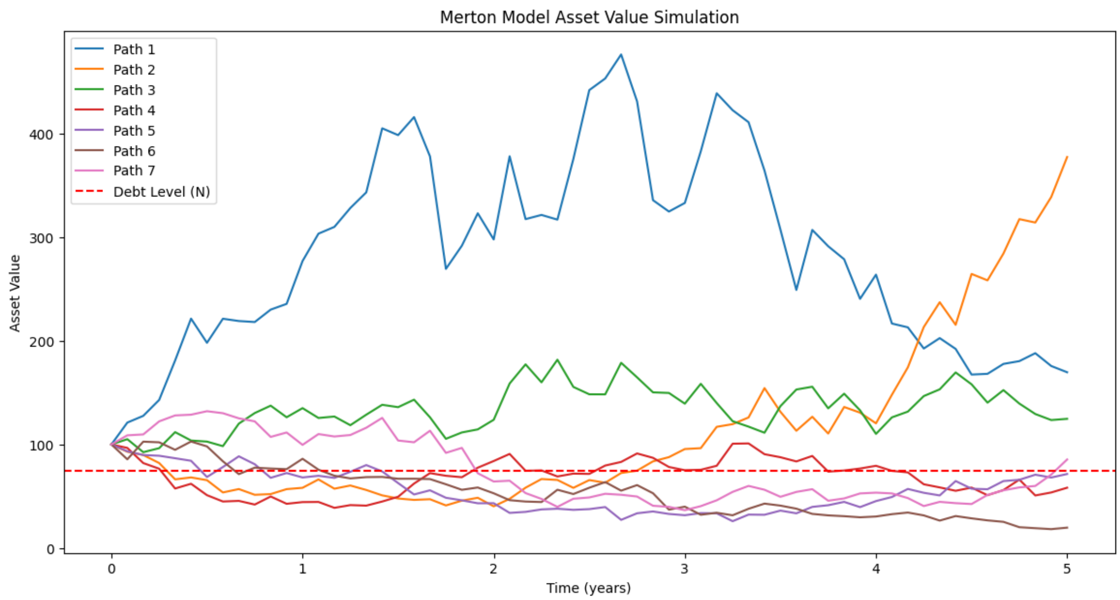
ps3

November 4, 2023

Jerry Tsai, Yi Pan, Ao Zhan, Yu-Chen Chen, Lei Huang, Jacob Yee

[9]:
```
#Question 1
#(a)
# Merton model parameters
V0 = 100  # Initial asset value
mu = 0.09  # Expected return
sigma_V = 0.40  # Asset volatility
T = 5  # Maturity in years
dt = 1/12  # Time interval in years
N = 75  # Nominal face value of debt

# Number of time steps
num_steps = int(T / dt)

# Simulate seven scenarios for the asset value process
np.random.seed(0)
paths = {}
for i in range(7):
    # Generate asset values
    asset_values = [V0]
    for _ in range(num_steps):
        dW = np.random.normal(0, np.sqrt(dt))
        dV = mu * asset_values[-1] * dt + sigma_V * asset_values[-1] * dW
        asset_values.append(asset_values[-1] + dV)
    paths[i] = asset_values

# Plot the scenarios
plt.figure(figsize=(14, 7))
for i in range(7):
    plt.plot(np.linspace(0, T, num_steps + 1), paths[i], label='Path {}'.
 ↪format(i+1))

plt.axhline(y=N, color='r', linestyle='--', label='Debt Level (N)')
plt.title('Merton Model Asset Value Simulation')
plt.xlabel('Time (years)')
plt.ylabel('Asset Value')
plt.legend()
```

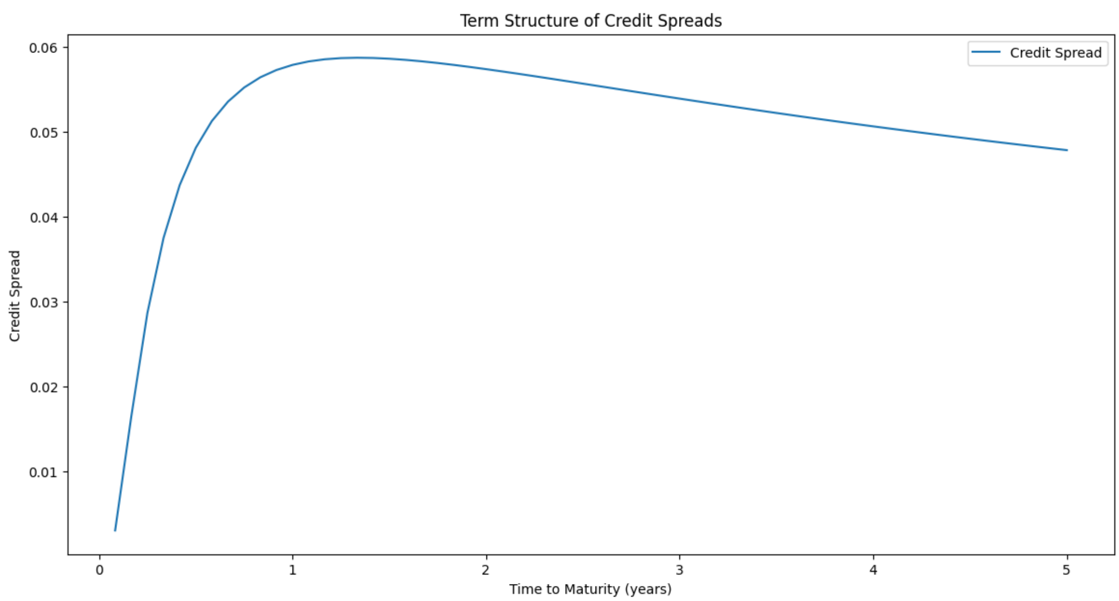
```
plt.show()

# Determine which paths result in default
defaults = []
for i in range(7):
    if paths[i][-1] < N:
        defaults.append(i+1)

print('Paths resulting in default: {}'.format(defaults))
```




```
Paths resulting in default: [4, 5, 6]
```

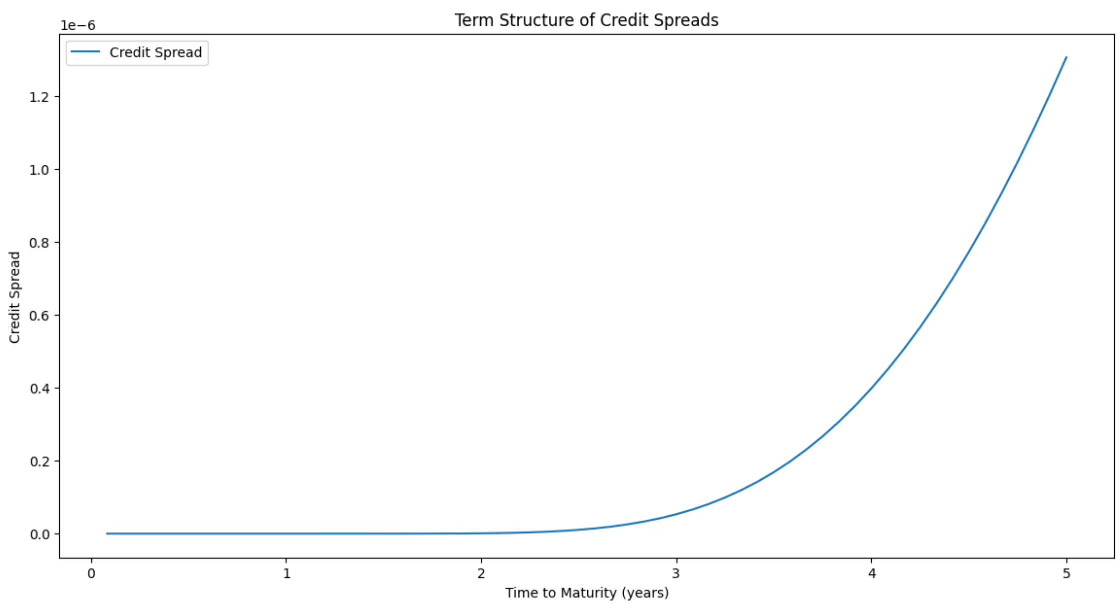
[10]:
```
#(b)
# Merton model parameters
V0 = 100  # Initial asset value
mu = 0.09  # Expected return
sigma_V = 0.40  # Asset volatility
T = 5  # Maturity in years
dt = 1/12  # Time interval in years
N = 75  # Nominal face value of debt
r = 0.03  # Risk-free interest rate

# Calculate the term structure of credit spreads
num_steps = int(T / dt)
times = np.linspace(dt, T, num_steps)
credit_spreads = []
```

```
for t in times:
    d1 = (np.log(V0 / N) + (r + 0.5 * sigma_V**2) * t) / (sigma_V * np.sqrt(t))
    d2 = d1 - sigma_V * np.sqrt(t)
    put_option_value = N * np.exp(-r * t) * norm.cdf(-d2) - V0 * norm.cdf(-d1)
    risky_bond_price = N * np.exp(-r * t) - put_option_value
    YTM = -np.log(risky_bond_price / N) / t
    credit_spread = YTM - r
    credit_spreads.append(credit_spread)

# Plot the term structure of credit spreads
plt.figure(figsize=(14, 7))
plt.plot(times, credit_spreads, label='Credit Spread')
plt.title('Term Structure of Credit Spreads')
plt.xlabel('Time to Maturity (years)')
plt.ylabel('Credit Spread')
plt.legend()
plt.show()
```



[11]:
```
#(c)
# Merton model parameters
V0 = 100  # Initial asset value
mu = 0.09  # Expected return
T = 5  # Maturity in years
dt = 1/12  # Time interval in years
r = 0.03  # Risk-free interest rate
N2 = 50
sigma_V2 = 0.1
```

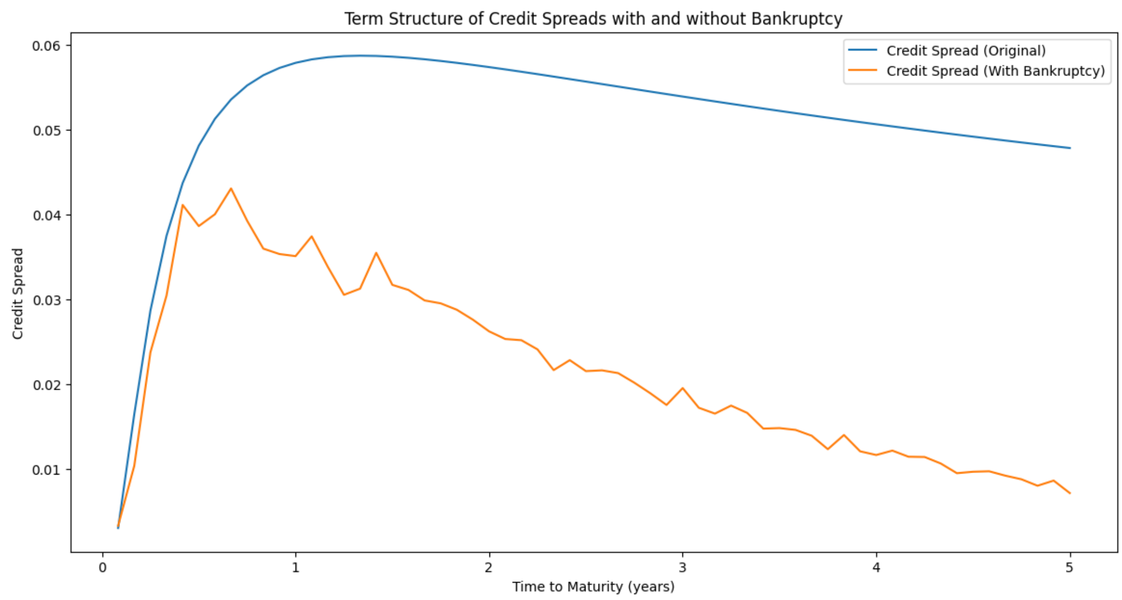
```
# Calculate the term structure of credit spreads
num_steps = int(T / dt)
times = np.linspace(dt, T, num_steps)
credit_spreads_2 = []

for t in times:
    d1 = (np.log(V0 / N2) + (r + 0.5 * sigma_V2**2) * t) / (sigma_V2 * np.
 ↪sqrt(t))
    d2 = d1 - sigma_V2 * np.sqrt(t)
    put_option_value = N2 * np.exp(-r * t) * norm.cdf(-d2) - V0 * norm.cdf(-d1)
    risky_bond_price = N2 * np.exp(-r * t) - put_option_value
    YTM = -np.log(risky_bond_price / N2) / t
    credit_spread_2 = YTM - r
    credit_spreads_2.append(credit_spread_2)

# Plot the term structure of credit spreads
plt.figure(figsize=(14, 7))
plt.plot(times, credit_spreads_2, label='Credit Spread')
plt.title('Term Structure of Credit Spreads')
plt.xlabel('Time to Maturity (years)')
plt.ylabel('Credit Spread')
plt.legend()
plt.show()
```



As a firm improves in credit quality (lower V and N), the term structure of credit spreads would flatten and shift downwards. This means that the additional yield investors require to hold the firm's debt over the risk-free rate would decrease, reflecting the firm's lower default risk. The term structure would also become less steep because the difference in credit risk between short-term and long-term debt would diminish.

[12]:
```
#(d)
# Merton model parameters
rho = 0.9  # Threshold for bankruptcy
num_paths = 1000  # Number of simulation paths

# Function to simulate one path of the asset value
def simulate_asset_path(V0, mu, sigma_V, T, dt, rho, N):
    num_steps = int(T / dt)
    Vt = V0
    for _ in range(num_steps):
        dW = np.random.normal(0, np.sqrt(dt))
        Vt = Vt * np.exp((mu - 0.5 * sigma_V**2) * dt + sigma_V * dW)
        if Vt < rho * N:
            return Vt, _ * dt  # Return the value at bankruptcy and the time of␣
↪bankruptcy
    return Vt, T  # Return the value at maturity and the maturity time if no␣
↪bankruptcy

# Function to calculate the bond price using Monte Carlo simulation
def monte_carlo_bond_price(V0, mu, sigma_V, T, N, r, dt, rho, num_paths):
    discounted_payoffs = []
    for _ in range(num_paths):
        Vt, tau = simulate_asset_path(V0, mu, sigma_V, T, dt, rho, N)
        if tau < T:
            # If bankruptcy occurs before maturity, bondholders receive the␣
↪asset value
            discounted_payoff = np.exp(-r * tau) * Vt
        else:
            # If no bankruptcy, bondholders receive the minimum of asset value␣
↪or face value of debt
            discounted_payoff = np.exp(-r * T) * min(Vt, N)
        discounted_payoffs.append(discounted_payoff)
    return np.mean(discounted_payoffs)

# Calculate the bond price with the possibility of bankruptcy
P0_with_bankruptcy = monte_carlo_bond_price(V0, mu, sigma_V, T, N, r, dt, rho,␣
↪num_paths)

# Calculate credit spreads with bankruptcy
credit_spreads_with_bankruptcy = []

for t in times:
```

```
    bond_price = monte_carlo_bond_price(V0, mu, sigma_V, t, N, r, dt, rho,␣
  ↪num_paths)
    YTM = np.log(N / bond_price) / t
    credit_spread = YTM - r
    credit_spreads_with_bankruptcy.append(credit_spread)

# Plot the term structure of credit spreads with and without bankruptcy
plt.figure(figsize=(14, 7))
plt.plot(times, credit_spreads, label='Credit Spread (Original)')
plt.plot(times, credit_spreads_with_bankruptcy, label='Credit Spread (With␣
  ↪Bankruptcy)')
plt.title('Term Structure of Credit Spreads with and without Bankruptcy')
plt.xlabel('Time to Maturity (years)')
plt.ylabel('Credit Spread')
plt.legend()
plt.show()
```



If the market believes that the firm has a high potential for recovery or if the bankruptcy threshold is set such that it is not too sensitive, the credit spread curve could indeed be less steep. However, the original spread is higher than with bankruptcy is counterintuitive because the introduction of bankruptcy risk should increase the credit spreads.

[1]:
```
#Question 2
#(a)
import pandas as pd
import numpy as np
from scipy.stats import norm
```

```
q2data=pd.read_csv('C:/Users/jakey/Downloads/ps3_q2_data.csv')
#first calcualte monthly returns
q2data['monthly returns']=q2data['price'].pct_change()
#monthly stock return volatility
monthlyreturnvol=q2data['monthly returns'].std()
print("The monthly stock return volatility: ", monthlyreturnvol)
```

```
The monthly stock return volatility:  0.22583331913135868
```

(b) from Vassalou and Xing, 2004: the value of the assets at any time t is

$$ln(V_{A,t+T}) = ln(V_{A,t}) + (\mu - \frac{\sigma_A^2}{2})T + \sigma_A\sqrt{T}\epsilon_{t+T}$$

[2]:
```
q2data['asset value']=0
q2data['asset volatility']=0
mu=.004
T=12
volupdate=[]
#setting initial asset volatility
q2data.iat[0,6]=monthlyreturnvol
q2data.iat[0,5]=np.exp(np.log(q2data.iat[0,1])+(mu-(q2data.iat[0,6]**2/
 ↪2))*T+q2data.iat[0,6]*np.sqrt(T))
for i in range(1,12,1):
    q2data.iat[i,5]=np.exp(np.log(q2data.iat[i-1,5])+(mu-(q2data.iat[0,6]**2/
 ↪2))*T+q2data.iat[0,6]*np.sqrt(T))
#update volatility by ln(V_j+1/V_j) for j=1...m-1
for i in range(1,12,1):
    q2data.iat[i,6]=np.log(q2data.iat[i,5]/q2data.iat[i-1,5])
sigma_A=q2data.iat[2,6]
V_A=q2data.iat[2,5]
print("The monthly asset value: ",V_A , "and the asset volatility: ", sigma_A)
```

```
The monthly asset value:  39.867078238947265 and the asset volatility:
0.5243054373755429
```

[3]:
```
#(c)
q2data['monthly asset return']=0
for i in range(1,12,1):
    q2data.iat[i,7]=((q2data.iat[i,5]/q2data.iat[i-1,5])-1)*100
A=sum(q2data['monthly asset return'])/11
print("Average monthly asset return: ", A)
```

```
Average monthly asset return:  68.9285126503683
```

[4]:
```
#N=STD+.5*LTD
N=16.334+.5*18.294
A0=A
DD=1/(sigma_A*np.sqrt(T))*((np.log(A0/N))+(mu-.5*sigma_A**2)*T)
DD
```

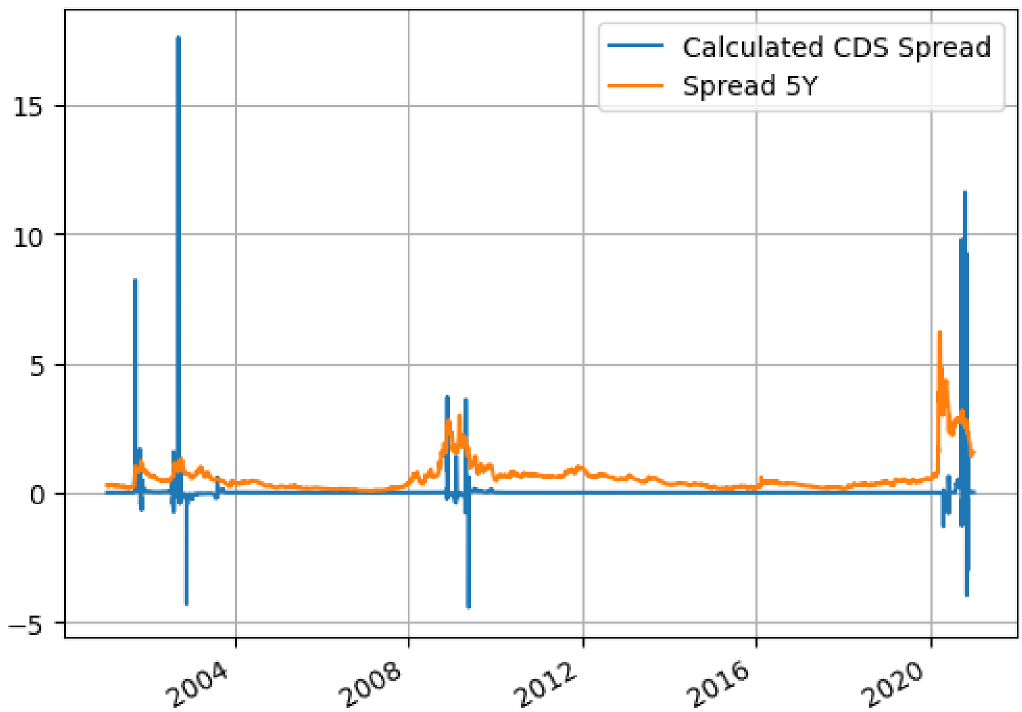
```
probdefault=1-norm.cdf(DD)
print("The distance to default: ", DD, "and the physical probability of default:
  ↪ ", probdefault)
```

The distance to default: -0.33378731147280066 and the physical probability of default: 0.6307299705857085

[6]:
```
#Question 3
#(a)
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm
import statsmodels.api as sm
import statistics
import matplotlib.dates as mdates
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
df = pd.read_csv('C:/Users/jakey/Downloads/boeing.csv')
df['date'] = pd.to_datetime(df['date'])

#Given values
lambda_ = 0.15 #percentage standard deviation of the default barrier
R = 0.5 #recover rate
T = 5 #yr
L_ = 0.6 # leverage ration
#credit default spread
def PS(S0,L_,D,lambda_,sigma,S,T):
    # (sigma*S/(S+L_*D)) in the lecture
    A = (sigma*S/(S+L_*D))**2*T+lambda_**2
    d = np.exp(lambda_**2)*(S0+L_*D)/(L_*D)
    PS = norm.cdf(-A/2+np.log(d)/A)-d*norm.cdf(-A/2-np.log(d)/A)
    return PS

# function G is given by Rubinstein and Reiner (1991):
def G (u,r,sigma,S0,L_,D,lambda_):
    z = np.sqrt(1/4+2*r/sigma**2)
    d = np.exp(lambda_**2)*(S0+L_*D)/(L_*D)

    term1 = d**(z + 1/2) * norm.cdf((-np.log(d)/(sigma * np.sqrt(u)) - z *␣
  ↪sigma * np.sqrt(u)))
    term2 = d**(-z + 1/2) * norm.cdf((-np.log(d)/(sigma * np.sqrt(u)) + z *␣
  ↪sigma * np.sqrt(u)))
    return term1+term2
def credit_default_spread(r, R, PS0, PST, T, lambda_, sigma):
    xi = lambda_**2/sigma**2
    g_diff = G(T + xi,r,sigma,S0,L_,D,lambda_) - G(xi, r,sigma,S0,L_,D,lambda_)
```

```python
    numerator = r * (1 - R) * (1 - PS0 + np.exp(r*xi) * g_diff)
    denominator = PS0 - PST * np.exp(-r*T) - np.exp(r*xi) * g_diff

    return numerator / denominator
def RPV01 (PS0,PST,r,T):
    xi = lambda_**2/sigma**2
    g_diff = G(T + xi,r,sigma,S0,L_,D,lambda_) - G(xi, r,sigma,S0,L_,D,lambda_)
    return (PS0-PST*np.exp(-r*T)-np.exp(r*xi)*g_diff)/r

# moving average as reference stock price and sigma
cds_spreads = []
df['moving_avg_S'] = df['S'].rolling(window=20).mean()
df['moving_avg_sigma'] = df['sigma_stock'].rolling(window=20).mean()
df = df.dropna()
# Iterate through rows of the dataframe
for index, row in df.iterrows():
    S0 = row["S"]  # stock price at time t
    L_ = 0.6  # Given leverage ratio
    D = row["D"]
    lambda_ = row["lambda"]
    sigma = row["sigma_stock"]
    r = row["r"]
    R = row["R"]
    T = 5  # maturity
    reference_S = row['moving_avg_S']
    reference_sigma = row['moving_avg_sigma']
    PS_t = PS(S0, L_, D, lambda_, reference_sigma, reference_S,T)
    PS_0 = PS(S0, L_, D, lambda_, reference_sigma, reference_S,0)
    cds = credit_default_spread(r, R, PS_0, PS_t, T, lambda_, sigma)
    cds_spreads.append(cds)

df["calculated_cds_spread"] = cds_spreads

locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
formatter = mdates.ConciseDateFormatter(locator)
fig, ax = plt.subplots()
ax.plot(df['date'], df['calculated_cds_spread'], label='Calculated CDS Spread')
ax.plot(df['date'], df['spread5y'], label='Spread 5Y')
ax.xaxis.set_major_locator(locator)
ax.xaxis.set_major_formatter(formatter)
fig.autofmt_xdate()
ax.grid(True)
ax.legend()

plt.show()
```

```python
cds_calculated = df['calculated_cds_spread']
cds_5y = df['spread5y']
threshold = np.std(cds_calculated - cds_5y)

divergences = cds_calculated - cds_5y
significant_divergences = np.abs(divergences) > threshold

undervalued = divergences > threshold
overvalued = divergences < -threshold

print("Undervalued Equity Points:")
print(df['S'][undervalued])

print("Overvalued Equity Points:")
print(df['S'][overvalued])
```




```
Undervalued Equity Points:
177      29.760000
206      32.599998
399      41.500000
428      35.520000
1986     40.180000
2090     38.849998
4961    156.800000
```

```
4980    167.110000
4989    144.390000
4990    148.600010
Name: S, dtype: float64
Overvalued Equity Points:
176      32.610001
189      36.619999
190      36.000000
191      35.759998
192      36.180000
           ...
5026    217.149990
5027    216.090000
5028    216.250000
5029    216.670000
5030    214.060000
Name: S, Length: 917, dtype: float64
```

[7]:

```python
#(b)
X = df['S']
y = df['calculated_cds_spread']
X = sm.add_constant(X)
model = sm.OLS(y,X).fit()
coefficients = model.params
dc_ds = coefficients[1]
RPV = []
# Iterate through rows of the dataframe
for index, row in df.iterrows():
    S0 = row["S"]  # stock price at time t
    L_ = 0.6  # Given leverage ratio
    D = row["D"]
    lambda_ = row["lambda"]
    sigma = row["sigma_stock"]
    r = row["r"]
    R = row["R"]
    T = 5  # maturity
    reference_S = row['moving_avg_S']
    reference_sigma = row['moving_avg_sigma']
    PS_t = PS(S0, L_, D, lambda_, reference_sigma, reference_S,T)
    PS_0 = PS(S0, L_, D, lambda_, reference_sigma, reference_S,0)
    RPV.append(RPV01 (PS_0,PS_t,r,T))
df['RPV01'] = RPV
df['hedge ratio of equity'] = df['RPV01']*dc_ds

print(df['hedge ratio of equity'])
```

```
19      -0.000096
20      -0.000096
```

```
21      -0.000095
22      -0.000096
23      -0.000096
           …
5026    -0.000435
5027    -0.000434
5028    -0.000436
5029    -0.000438
5030    -0.000433
Name: hedge ratio of equity, Length: 5012, dtype: float64
```

[8]:
```python
#(c)

historical_market_cds_spreads = np.random.normal(100, 20, 252)
historical_cg_implied_cds_spreads = np.random.normal(50, 10, 252)
historical_equity_prices = np.random.normal(50, 5, 252)

average_spread_difference = np.mean(historical_market_cds_spreads -
 ↪historical_cg_implied_cds_spreads)


threshold = 2 * np.mean(historical_cg_implied_cds_spreads)

if average_spread_difference > threshold:

    cds_position = 'buy'
elif average_spread_difference < -threshold:

    cds_position = 'sell'
else:
    cds_position = 'no_arbitrage'

notional_cds = 1000000
hedge_ratio = 1

future_market_cds_spreads = np.random.normal(100, 20, 63)
future_equity_prices = np.random.normal(50, 5, 63)
if cds_position == 'buy':
    cds_pnl = (historical_market_cds_spreads[0] -
 ↪future_market_cds_spreads[-1]) * notional_cds
elif cds_position == 'sell':
    cds_pnl = (future_market_cds_spreads[-1] -
 ↪historical_market_cds_spreads[0]) * notional_cds
else:
    cds_pnl = 0
```

```python
equity_pnl = -hedge_ratio * (future_equity_prices[-1] -
 ↪historical_equity_prices[0]) * (notional_cds / historical_equity_prices[0])

total_pnl = cds_pnl + equity_pnl

print(f"CDS Position: {cds_position}")
print(f"CDS PnL: {cds_pnl}")
print(f"Equity PnL: {equity_pnl}")
print(f"Total PnL: {total_pnl}")
```

```
CDS Position: no_arbitrage
CDS PnL: 0
Equity PnL: 125515.97600712982
Total PnL: 125515.97600712982
```