Quantitative Analysis of Delta-Hedged Portfolio Risk: Monte Carlo Simulation Approach

Define the functions for calculation and simulation

In [ ]:
```
import numpy as np
from scipy.stats import norm, t
import math
import pandas as pd
import matplotlib.pyplot as plt
import os
from tqdm import tqdm
def simulate_brownian_motion_paths(S0, r, sigma, T, M, I):
    """
    Simulate stock price paths using geometric Brownian motion.
    :param S0: Initial stock price.
    :param r: Risk-free rate.
    :param sigma: Volatility of the underlying asset.
    :param T: Time to maturity.
    :param M: Number of time steps .
    :param I: Number of simulation paths.
    :return: Simulated paths array.
    """
    dt = T / M
    paths = np.zeros((M + 1, I))
    paths[0] = S0
    for t in range(1, M + 1):
        Z = np.random.normal(0, np.sqrt(dt), I)
        paths[t] = paths[t - 1] * np.exp((r - 0.5 * sigma ** 2) * dt + sigma * Z)
    return paths

def simulate_student_t_paths(S0, r, sigma, T, M, I, v):
    """
    Simulate stock price paths with increments distributed according to a Student-t distribution.
    :param S0: Initial stock price.
    :param r: Risk-free rate.
    :param sigma: Volatility of the underlying asset.
    :param T: Time to maturity.
    :param M: Number of time steps .
    :param I: Number of simulation paths.
    :param v: Degrees of freedom for the Student-t distribution.
    :return: Simulated paths array.
    """
    dt = T / M
    paths = np.zeros((M + 1, I))
    paths[0] = S0
    for i in range(1, M + 1):
        Z = t.rvs(df=v, size=I)
        paths[i] = paths[i - 1] * np.exp((r - 0.5 * sigma ** 2) * dt + sigma * np.sqrt(dt) * Z)
    return paths

def calculate_delta(S, K, r, sigma, T, t):
    """
    Calculate the delta of an option using the Black-Scholes formula.
    :param S: Current stock price.
    :param K: Strike price of the option.
```

```
    :param r: Risk-free rate.
    :param sigma: Volatility of the underlying asset.
    :param T: Time to maturity.
    :param t: Current time.
    :return: Delta value.
    """
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * (T - t)) / (sigma * np.sqrt(T - t))
    return norm.cdf(d1)

def black_scholes_price(S, K, r, sigma, T, t, option_type='call'):
    """
    Calculate the Black-Scholes option price.
    :param S: Current stock price.
    :param K: Strike price.
    :param r: Risk-free interest rate.
    :param sigma: Volatility of the stock.
    :param T: Time to maturity.
    :param t: Current time.
    :param option_type: Type of the option - 'call' or 'put'.
    :return: Price of the option.
    """
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * (T - t)) / (sigma * np.sqrt(T - t))
    d2 = d1 - sigma * np.sqrt(T - t)
    if option_type == 'call':
        return S * norm.cdf(d1) - K * np.exp(-r * (T - t)) * norm.cdf(d2)
    else:  # put option
        return K * np.exp(-r * (T - t)) * norm.cdf(-d2) - S * norm.cdf(-d1)



def calculate_var_es(losses, confidence_level=0.95):
    """
    Calculate VaR and ES for a series of losses.
    :param losses: Array of losses.
    :param confidence_level: Confidence level for VaR and ES calculation.
    :return: VaR and ES values.
    """

    sorted_losses = np.sort(losses)
    var_index = int((1 - confidence_level) * len(sorted_losses))
    VaR = sorted_losses[var_index]
    ES = sorted_losses[:var_index].mean()
    return VaR, ES



def calculate_portfolio_value(price_path, K, r, sigma, T):
    """
    Calculate the delta-hedged self-financing portfolio values

    """

    M = len(price_path) - 1  # Number of time steps
    dt = T / M  # Length of each time step

    # Calculate initial option price, delta, and theta
    C0 = black_scholes_price(price_path[0], K, r, sigma, T, 0)
    delta_0 = calculate_delta(price_path[0], K, r, sigma, T, 0)
    theta_call = -1  # The number of call options held
    theta_stock = 0-(delta_0*theta_call)  # To ensure delta-hedging
```

```
    portfolio_values = []

    # Initialize portfolio value
    portfolio_value = theta_stock * price_path[0] + theta_call * C0
    portfolio_shares={'call':1,
               'stock':theta_stock}
    portfolio_values.append(portfolio_value)

    # Iterate through the price path to update the portfolio
    for t in range(1,M):
        # stock price

        St = price_path[t]

        # Calculate option price at t+1 and delta at t+1
        Ct = black_scholes_price(St, K, r, sigma, T , dt * (t))
        delta_t = calculate_delta(St, K, r, sigma, T , dt * (t))
        portfolio_value = St * portfolio_shares['stock'] + portfolio_shares['call'] * Ct
        portfolio_values.append(portfolio_value)
        # # Update theta for call and stock based on delta-hedging and self-financing condition
        # theta_call = (portfolio_shares['stock']*St+portfolio_shares['call']*Ct)/(Ct-delta_t*St) # From the delta-hedging equation

        # theta_stock = -theta_call * delta_t  # From the self-financing condition

        # Set up the coefficient matrix A and constant vector b
        A = np.array([[St, Ct], [1,delta_t]])
        b = np.array([St * portfolio_shares['stock'] + portfolio_shares['call'] * Ct, 0])

        # Solve for the unknowns theta_{t+1}^S and theta_{t+1}^C
        # theta_t_plus_1 = np.linalg.solve(A, b)
        try:
            portfolio_shares['stock'],portfolio_shares['call'] = np.linalg.solve(A, b)
        except:
            continue



    return portfolio_values



def calculate_portfolio_VaR(portfolioprice,confidence_level):
    """
    This function is used to calculate the portfolio loss (e.g simple return, or log return)
    """


    portfolioprice = pd.Series(portfolioprice)

    # Calculate loss (considering them as 'losses')
    loss = portfolioprice - portfolioprice.shift(-1).dropna()

    # Sort the log returns in ascending order (since losses are negative, this actually sorts them by severity)

    sorted_loss = loss.sort_values(ascending=True)
    # Calculate VaR given confidence level

    var = sorted_loss.quantile(1 - confidence_level)

    return var
```

```
def calculate_portfolio_ES(portfolioprice,confidence_level):
    """
    This function is used to calculate the expected shortfall (e.g simple return, or log return)
    """

    portfolioprice = pd.Series(portfolioprice)

    # Calculate loss (considering them as 'losses')
    loss = portfolioprice - portfolioprice.shift(-1).dropna()

    # Sort the log returns in ascending order (since losses are negative, this actually sorts them by severity)

    sorted_loss = loss.sort_values(ascending=True)
    # Calculate VaR given confidence level

    var = sorted_loss.quantile(1 - confidence_level)

    # Calculate Expected Shortfall (ES)
    es = sorted_loss[sorted_loss >= var].mean()

    return es



def quantile_removal(values):
    #remove the top 10 and bot 10 percent of the data to aviod the abnomal data generated calculation errors by solve the
    # system of equations
    # Calculate the 10th and 90th percentiles
    values = np.array(values)
    p15 = np.percentile(values, 10)
    p85 = np.percentile(values, 90)

    # Keep only data between the 15th and 85th percentiles
    filtered_data = values[(values > p15) & (values < p85)]

    return filtered_data
```

Simulation step: Modify "I" below to increase the simulation times

In [ ]:
```
# Consider the different strike prices and different Vailty

results = {}

# for different strike price
for K in tqdm([100,140,180]):

    # for different variance
    for sigma in tqdm([0.2, 0.4]):

        key = f'K={K}_sigma={sigma}'
        results[key] = {
            'VaRs_Brownian': [],
            'VaRs_Student_t_1': [],
            'VaRs_Student_t_2': [],
            'VaRs_Student_t_3': [],
            'ES_Brownian': [],
```

```
        'ES_Student_t_1': [],
        'ES_Student_t_2': [],
        'ES_Student_t_3': []
    }

    # Parameters for simulation
    S0 = 100  # Initial stock price
    #K = 100  # Strike price
    T = 1.0  # Time to maturity (1 year)
    r = 0.05  # Risk-free rate
    #sigma = 0.2  # Volatility of the underlying asset
    M = 365  # Number of time steps
    # change the I here for increase the simulation path
    I = 2000  # Number of simulation paths
    v_1 = 4  # Degrees of freedom for the Student-t distribution
    v_2 = 6  # Degrees of freedom for the Student-t distribution
    v_3 = 8  # Degrees of freedom for the Student-t distribution
    dt = T / M  # Length of each time step in years

    # Simulate paths using both methods
    paths_brownian = simulate_brownian_motion_paths(S0, r, sigma, T, M, I)
    paths_student_t_1 = simulate_student_t_paths(S0, r, sigma, T, M, I, v_1)
    paths_student_t_2 = simulate_student_t_paths(S0, r, sigma, T, M, I, v_2)
    paths_student_t_3 = simulate_student_t_paths(S0, r, sigma, T, M, I, v_3)



    # VaRs_Brownian = []
    # VaRs_Student_t_1 = []
    # VaRs_Student_t_2 = []
    # VaRs_Student_t_3 = []
    # ES_Brownian=[]
    # ES_Student_t_1 = []
    # ES_Student_t_2 = []
    # ES_Student_t_3 = []

    # monte carlo simulation for all the price paths
    for i in range(I):


        # Brownian
        price_path=paths_brownian.T[i]
        portfolioprice = calculate_portfolio_value(price_path, K, r, sigma, T)
        var = calculate_portfolio_VaR(portfolioprice, confidence_level=0.95)
        ES = calculate_portfolio_ES(portfolioprice, confidence_level=0.95)
        results[key]['VaRs_Brownian'].append(var)
        results[key]['ES_Brownian'].append(ES)

        # VaRs_Brownian.append(var)
        # ES_Brownian.append(ES)

        # Student t , df = 4
        price_path=paths_student_t_1.T[i]
        portfolioprice = calculate_portfolio_value(price_path, K, r, sigma, T)
        var = calculate_portfolio_VaR(portfolioprice, confidence_level=0.95)
        ES = calculate_portfolio_ES(portfolioprice,confidence_level=0.95)
        results[key]['VaRs_Student_t_1'].append(var)
        results[key]['ES_Student_t_1'].append(ES)

        # VaRs_Student_t_1.append(var)
```

```
            # ES_Student_t_1.append(ES)

            # Student t , df = 6
            price_path=paths_student_t_2.T[i]
            portfolioprice = calculate_portfolio_value(price_path, K, r, sigma, T)
            var = calculate_portfolio_VaR(portfolioprice, confidence_level=0.95)
            ES = calculate_portfolio_ES(portfolioprice,confidence_level=0.95)
            results[key]['VaRs_Student_t_2'].append(var)
            results[key]['ES_Student_t_2'].append(ES)
            # VaRs_Student_t_2.append(var)
            # ES_Student_t_2.append(ES)


            # Student t , df = 8
            price_path=paths_student_t_3.T[i]
            portfolioprice = calculate_portfolio_value(price_path, K, r, sigma, T)
            var = calculate_portfolio_VaR(portfolioprice, confidence_level=0.95)
            ES = calculate_portfolio_ES(portfolioprice,confidence_level=0.95)
            results[key]['VaRs_Student_t_3'].append(var)
            results[key]['ES_Student_t_3'].append(ES)
            # VaRs_Student_t_3.append(var)
            # ES_Student_t_3.append(ES)
```

In [ ]:

```
# result data
ResultTable = pd.DataFrame.from_dict({(i, j): results[i][j]
                             for i in results.keys()
                             for j in results[i].keys()},
                            orient='index')
ResultTable
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	1990	1991	1992	1993	1994
(K=100_sigma=0.2, VaRs_Brownian)	-0.033592	-0.061254	-0.204885	-0.115617	-0.135150	-0.024037	-0.093868	-0.041426	-0.075178	-0.024620	...	-0.033339	-0.022048	-0.039765	-0.021967	-0.059596
(K=100_sigma=0.2, VaRs_Student_t_1)	-0.048053	-0.097157	-0.080911	-0.020875	-0.104531	-0.022526	-0.029385	-0.169044	-0.080344	-0.045329	...	-0.175078	-0.021132	-0.138837	-0.021956	-0.020347
(K=100_sigma=0.2, VaRs_Student_t_2)	-0.027025	-0.079674	-0.089333	-0.055933	-0.066058	-0.165361	-0.039547	-0.173320	-0.024367	-0.281438	...	-0.021104	-0.066665	-0.031755	-0.026873	-0.069476
(K=100_sigma=0.2, VaRs_Student_t_3)	-0.030196	-0.023698	-0.025704	-0.021416	-0.232304	-0.184760	-0.022065	-0.112939	-0.025869	-0.024579	...	-0.325046	-0.237069	-0.027283	-0.235291	-0.088197
(K=100_sigma=0.2, ES_Brownian)	-0.008132	-0.000954	0.012889	0.020242	-0.007034	-0.008940	-0.003994	-0.010187	-0.013586	-0.009301	...	-0.009676	-0.010740	-0.005534	-0.010203	-0.006536
(K=100_sigma=0.2, ES_Student_t_1)	0.019383	0.177665	0.146436	0.004568	0.128298	-0.000069	0.007030	0.046088	0.024453	0.018760	...	0.096126	-0.006711	0.229709	-0.004454	-0.007819
(K=100_sigma=0.2, ES_Student_t_2)	-0.008020	0.005843	0.128807	0.009245	0.014225	0.020312	-0.002410	0.050828	-0.002364	0.104717	...	-0.009260	0.031568	-0.004400	-0.006969	0.021661
(K=100_sigma=0.2, ES_Student_t_3)	-0.003970	-0.007559	-0.007522	-0.005916	0.013447	0.024059	-0.008158	0.032757	-0.005929	-0.008092	...	0.050182	0.103652	-0.007137	0.035520	0.013431
(K=100_sigma=0.4, VaRs_Brownian)	-0.294667	-0.047669	-0.390430	-0.307641	-0.606021	-0.546994	-0.096720	-0.090991	-0.308471	-0.095183	...	-0.242422	-0.073171	-0.058504	-0.059312	-0.292354
(K=100_sigma=0.4, VaRs_Student_t_1)	-0.162930	-0.219820	-0.192277	-0.344015	-0.485498	-0.037792	-0.404794	-0.082511	-0.195672	-0.200304	...	-0.266989	-0.042978	-0.148605	-0.293637	-0.070517
(K=100_sigma=0.4, VaRs_Student_t_2)	-0.295762	-0.243323	-0.065487	-0.041901	-0.175774	-0.029045	-0.064258	-0.066791	-0.033785	-0.440731	...	-0.048786	-0.046329	-0.036393	-0.065372	-0.235482
(K=100_sigma=0.4, VaRs_Student_t_3)	-0.092829	-0.071390	-0.106718	-0.128321	-0.090233	-0.039386	-0.036993	-0.042393	-0.378788	-0.056198	...	-0.077576	-0.044927	-0.125711	-0.557348	-0.043367
(K=100_sigma=0.4, ES_Brownian)	0.009007	-0.013096	0.024730	0.024440	0.051086	0.010766	-0.002285	-0.013797	0.040634	-0.014466	...	0.039478	-0.010574	-0.008179	-0.007199	-0.001716
(K=100_sigma=0.4, ES_Student_t_1)	0.078535	0.082886	0.075231	0.114025	0.131264	0.015822	0.104491	0.075822	0.109956	0.104326	...	0.163472	-0.000450	0.096124	0.222609	0.064281
(K=100_sigma=0.4, ES_Student_t_2)	1.165338	0.080220	0.022716	-0.001439	0.180597	-0.001657	0.030133	0.031539	-0.003890	0.097815	...	0.001665	0.001494	0.000724	-0.001128	0.174859
(K=100_sigma=0.4, ES_Student_t_3)	0.023637	0.007786	0.016013	0.030185	0.001695	0.001028	-0.005022	-0.004169	0.070237	0.000857	...	0.013847	-0.005102	0.026997	0.058181	-0.002636
(K=140_sigma=0.2, VaRs_Brownian)	-0.057038	-0.048475	-0.013695	-0.008238	-0.036814	-0.034914	-0.047795	-0.058009	-0.023464	-0.005798	...	-0.055762	-0.028106	-0.012566	-0.007459	-0.020822
(K=140_sigma=0.2, VaRs_Student_t_1)	-0.066125	-0.015740	-0.019127	-0.006742	-0.005080	-0.066958	-0.011610	-0.010428	-0.009225	-0.019728	...	-0.026549	-0.010845	-0.012023	-0.014114	-0.037255
(K=140_sigma=0.2, VaRs_Student_t_2)	-0.005703	-0.031725	-0.036197	-0.027188	-0.057575	-0.014774	-0.014690	-0.054475	-0.031375	-0.036455	...	-0.021984	-0.052238	-0.016397	-0.044109	-0.008621
(K=140_sigma=0.2, VaRs_Student_t_3)	-0.022739	-0.047114	-0.030722	-0.010191	-0.009576	-0.033803	-0.019889	-0.046101	-0.059336	-0.034924	...	-0.050850	-0.015272	-0.016899	-0.006138	-0.018754
(K=140_sigma=0.2, ES_Brownian)	0.006542	0.003056	-0.000291	-0.000619	0.000941	0.004145	0.005609	0.000193	0.000546	-0.001218	...	0.002638	0.000527	-0.000171	-0.000869	0.001686
(K=140_sigma=0.2, ES_Student_t_1)	0.006847	28044.420137	0.004074	0.008989	0.002081	0.173787	0.004556	0.044539	0.005026	0.010984	...	0.013437	0.009053	0.041076	0.027467	0.013115

	0	1	2	3	4	5	6	7	8	9	...	1990	1991	1992	1993	1994
(K=140_sigma=0.2, ES_Student_t_2)	0.001706	0.007015	0.010513	0.009800	0.005547	8.946885	0.003834	0.008740	0.009038	0.021295	...	0.031723	0.006118	0.004054	0.003619	0.002219
(K=140_sigma=0.2, ES_Student_t_3)	0.009956	0.075874	0.008879	0.004551	0.000532	0.004044	0.002517	0.002412	0.002690	0.008258	...	0.006476	0.003994	0.005064	-0.000496	0.003098
(K=140_sigma=0.4, VaRs_Brownian)	-0.254815	-0.144695	-0.321912	-0.374122	-0.687065	-0.165858	-0.095517	-0.102529	-0.082423	-0.057142	...	-0.221299	-0.197841	-0.208430	-0.336534	-0.261514
(K=140_sigma=0.4, VaRs_Student_t_1)	-0.157360	-0.098552	-0.118967	-0.138171	-0.051464	-0.181979	-0.233273	-0.129322	-0.059163	-0.092178	...	-0.036789	-0.047196	-0.180657	-0.128806	-0.070118
(K=140_sigma=0.4, VaRs_Student_t_2)	-0.039539	-0.131122	-0.224625	-0.254446	-0.032285	-0.238278	-0.065916	-0.124407	-0.203284	-0.064369	...	-0.112277	-0.070266	-0.170818	-0.088893	-0.100125
(K=140_sigma=0.4, VaRs_Student_t_3)	-0.130075	-0.302898	-0.114718	-0.056522	-0.058210	-0.033583	-0.116422	-0.201154	-0.105906	-0.052030	...	-0.158417	-0.294712	-0.046833	-0.227746	-0.079547
(K=140_sigma=0.4, ES_Brownian)	0.055450	0.033502	0.007887	0.020598	-0.028430	0.016223	0.061201	0.016420	-0.000365	0.005614	...	0.097551	0.033087	0.019831	0.018614	0.014501
(K=140_sigma=0.4, ES_Student_t_1)	0.058084	0.046134	0.081601	0.125863	0.038782	0.068427	0.142104	0.041392	0.109688	0.063121	...	0.008044	0.030206	0.172670	0.041511	0.029348
(K=140_sigma=0.4, ES_Student_t_2)	0.019759	0.242311	0.078988	0.062850	0.002911	0.070758	0.021268	0.066848	0.035371	0.048746	...	0.049348	0.009963	0.052914	0.012507	0.037528
(K=140_sigma=0.4, ES_Student_t_3)	0.031481	0.045615	0.035473	0.006519	0.010750	0.002571	0.041754	0.022056	0.038653	0.003995	...	0.047077	0.026695	0.004996	0.076185	0.013787
(K=180_sigma=0.2, VaRs_Brownian)	-0.002325	-0.002448	-0.002614	-0.002511	-0.002593	-0.002599	-0.003820	-0.003368	-0.002795	-0.002680	...	-0.001734	-0.002054	-0.002747	-0.002481	-0.002389
(K=180_sigma=0.2, VaRs_Student_t_1)	-0.002827	-0.001430	-0.001783	-0.001853	-0.001733	-0.001009	-0.001084	-0.001450	-0.001463	-0.001094	...	-0.000702	-0.006443	-0.001241	-0.001898	-0.001997
(K=180_sigma=0.2, VaRs_Student_t_2)	-0.002224	-0.001691	-0.001288	-0.001278	-0.001396	-0.001281	-0.000383	-0.002092	-0.001932	-0.001649	...	-0.001346	-0.001864	-0.000911	-0.003192	-0.000588
(K=180_sigma=0.2, VaRs_Student_t_3)	-0.001707	-0.002042	-0.001491	-0.002134	-0.002176	-0.003758	-0.002187	-0.002452	-0.002668	-0.001470	...	-0.002428	-0.002350	-0.002584	-0.002459	-0.002737
(K=180_sigma=0.2, ES_Brownian)	0.000092	0.000288	0.000199	0.001107	0.000344	0.000325	0.000444	0.000046	0.000397	0.000007	...	0.000302	0.000071	0.000024	-0.000023	0.000223
(K=180_sigma=0.2, ES_Student_t_1)	0.003090	0.001052	0.000907	0.000605	0.000734	0.001237	44.066957	0.002047	0.013878	0.002123	...	0.002284	0.003797	0.000865	0.000685	0.002613
(K=180_sigma=0.2, ES_Student_t_2)	0.000449	0.000721	0.001460	0.000250	0.000565	0.000595	0.000059	0.000667	0.000528	0.001593	...	0.003986	0.000910	0.000783	0.033625	0.000160
(K=180_sigma=0.2, ES_Student_t_3)	0.000582	0.000416	0.001826	0.000261	0.000835	0.000304	0.000224	0.000303	0.000450	0.000314	...	0.003576	0.002121	0.000579	0.000331	0.000363
(K=180_sigma=0.4, VaRs_Brownian)	-0.109017	-0.124563	-0.207351	-0.076926	-0.034914	-0.075195	-0.205205	-0.131486	-0.152973	-0.037203	...	-0.039689	-0.038729	-0.039098	-0.276168	-0.154277
(K=180_sigma=0.4, VaRs_Student_t_1)	-0.018915	-0.051258	-0.037170	-0.048818	-0.083621	-0.068893	-0.033734	-0.031889	-0.061032	-0.034458	...	-0.016410	-0.048384	-0.047914	-0.036215	-0.025814
(K=180_sigma=0.4, VaRs_Student_t_2)	-0.069966	-0.063453	-0.104241	-0.014299	-0.058486	-0.049046	-0.027803	-0.063395	-0.058319	-0.109605	...	-0.092039	-0.084854	-0.073990	-0.066522	-0.076863
(K=180_sigma=0.4, VaRs_Student_t_3)	-0.106598	-0.101444	-0.028850	-0.079388	-0.081125	-0.091418	-0.083483	-0.068151	-0.169627	-0.049163	...	-0.068602	-0.091495	-0.109478	-0.050041	-0.123160

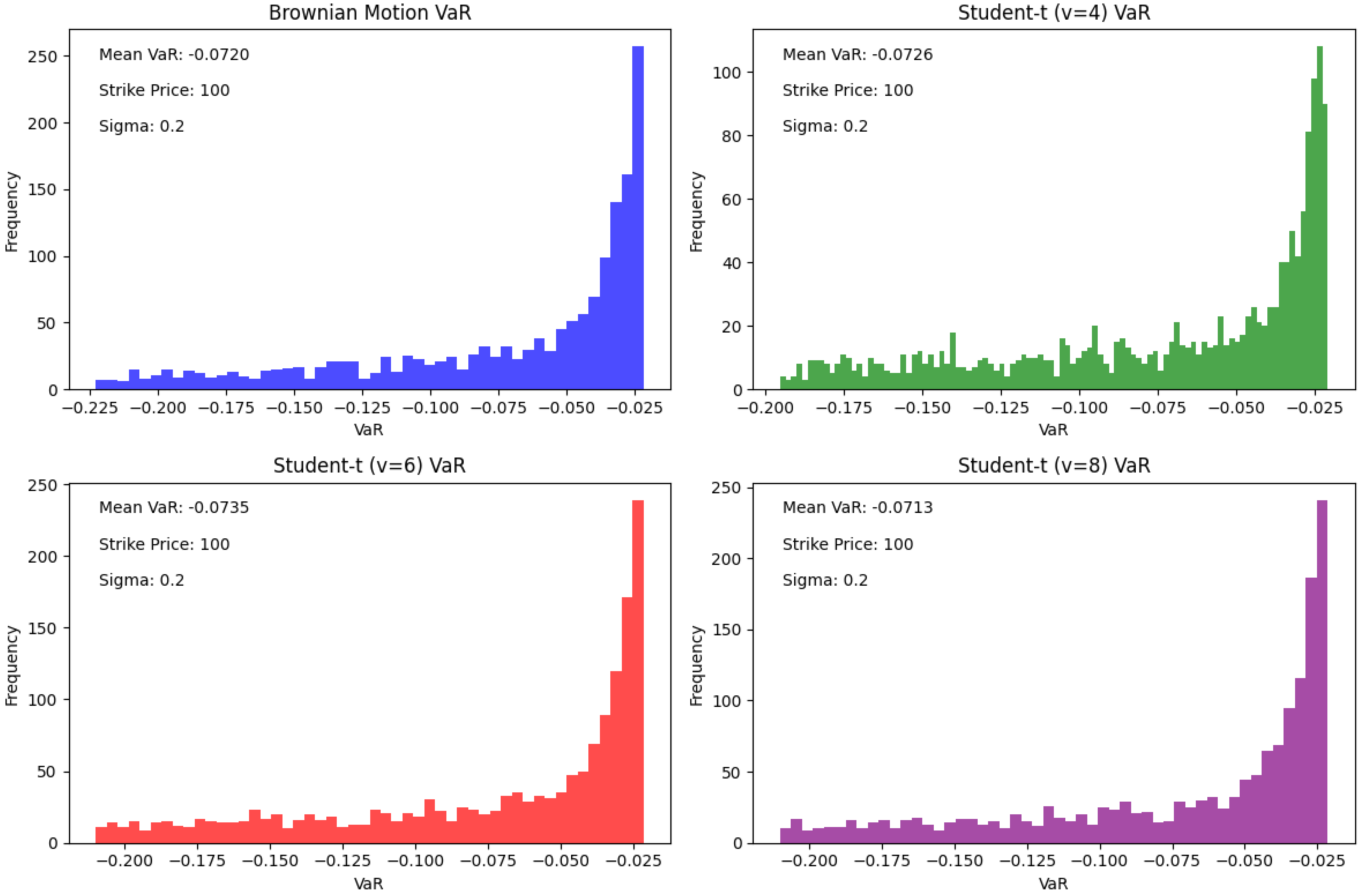
	0	1	2	3	4	5	6	7	8	9	...	1990	1991	1992	1993	1994
(K=180_sigma=0.4, ES_Brownian)	0.005044	0.017858	0.017001	0.058992	0.000113	0.009076	0.002709	0.013735	0.006468	0.000327	...	0.001310	0.003464	-0.001873	0.002535	0.006961
(K=180_sigma=0.4, ES_Student_t_1)	0.048151	0.044118	0.012770	0.029211	0.977060	0.799278	0.035505	0.044321	0.029109	0.072355	...	0.006245	0.030835	0.027454	0.032023	0.023082
(K=180_sigma=0.4, ES_Student_t_2)	0.020978	0.010042	0.012176	0.001686	0.032669	1.125316	0.037772	0.024677	0.035145	0.030982	...	0.033113	0.025192	0.014224	0.024778	0.025478
(K=180_sigma=0.4, ES_Student_t_3)	0.109782	0.019683	0.007394	0.010196	0.030001	0.009757	0.022313	0.017492	0.008384	0.007032	...	0.024162	0.020568	0.012581	0.014567	0.018409

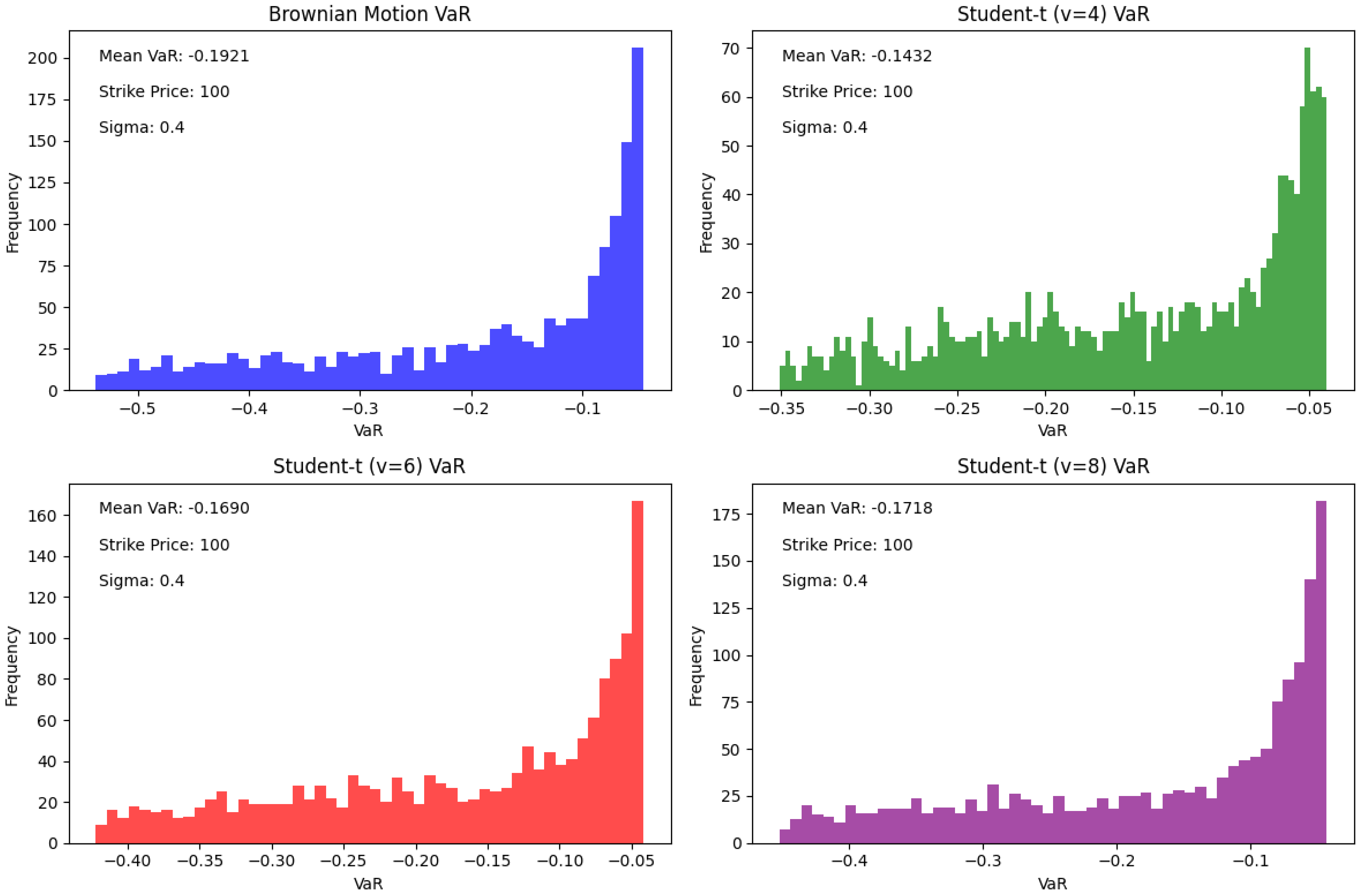
48 rows × 2000 columns

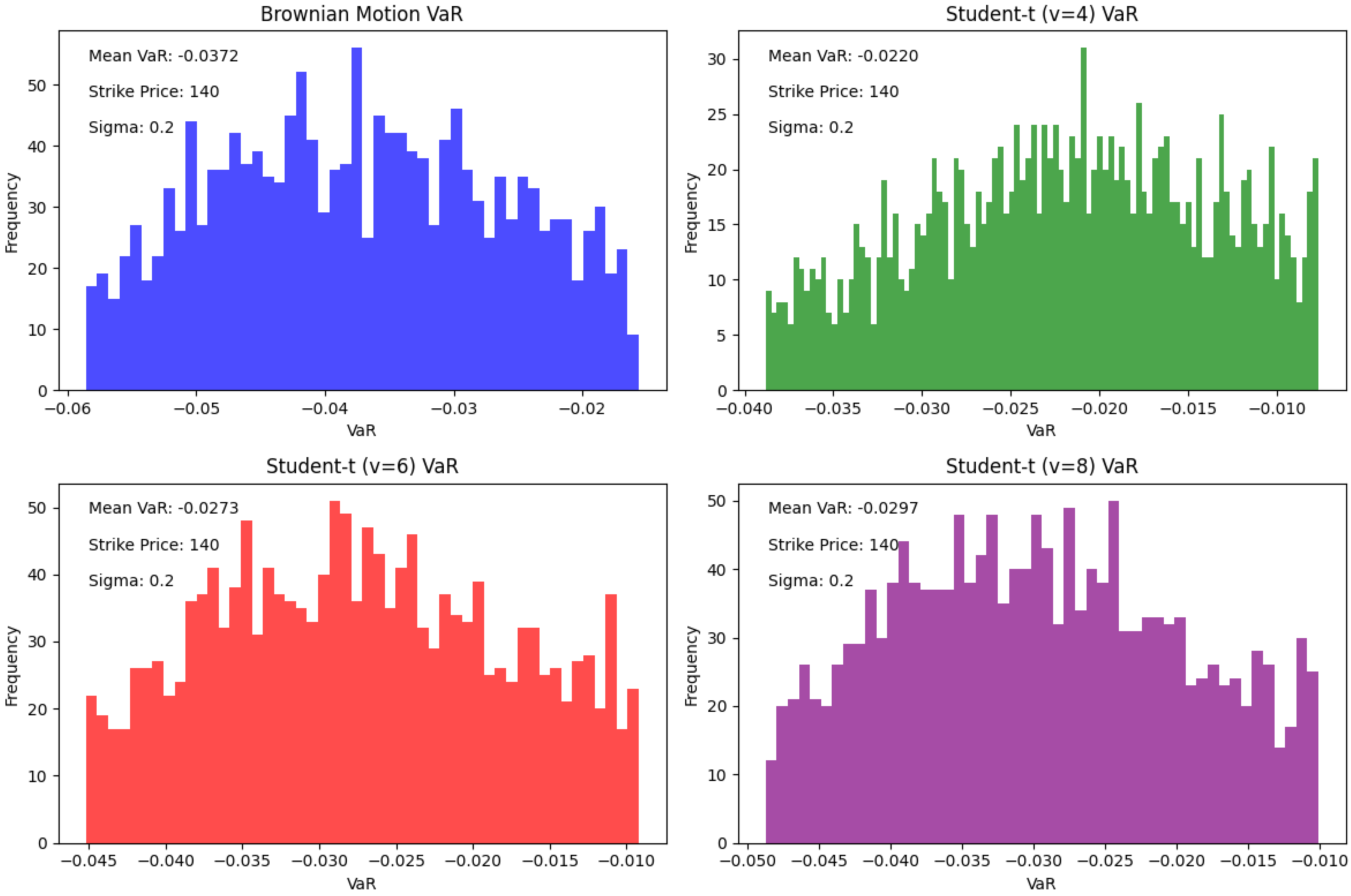
plots for VaR at different strike prices, Volatility and different price paths

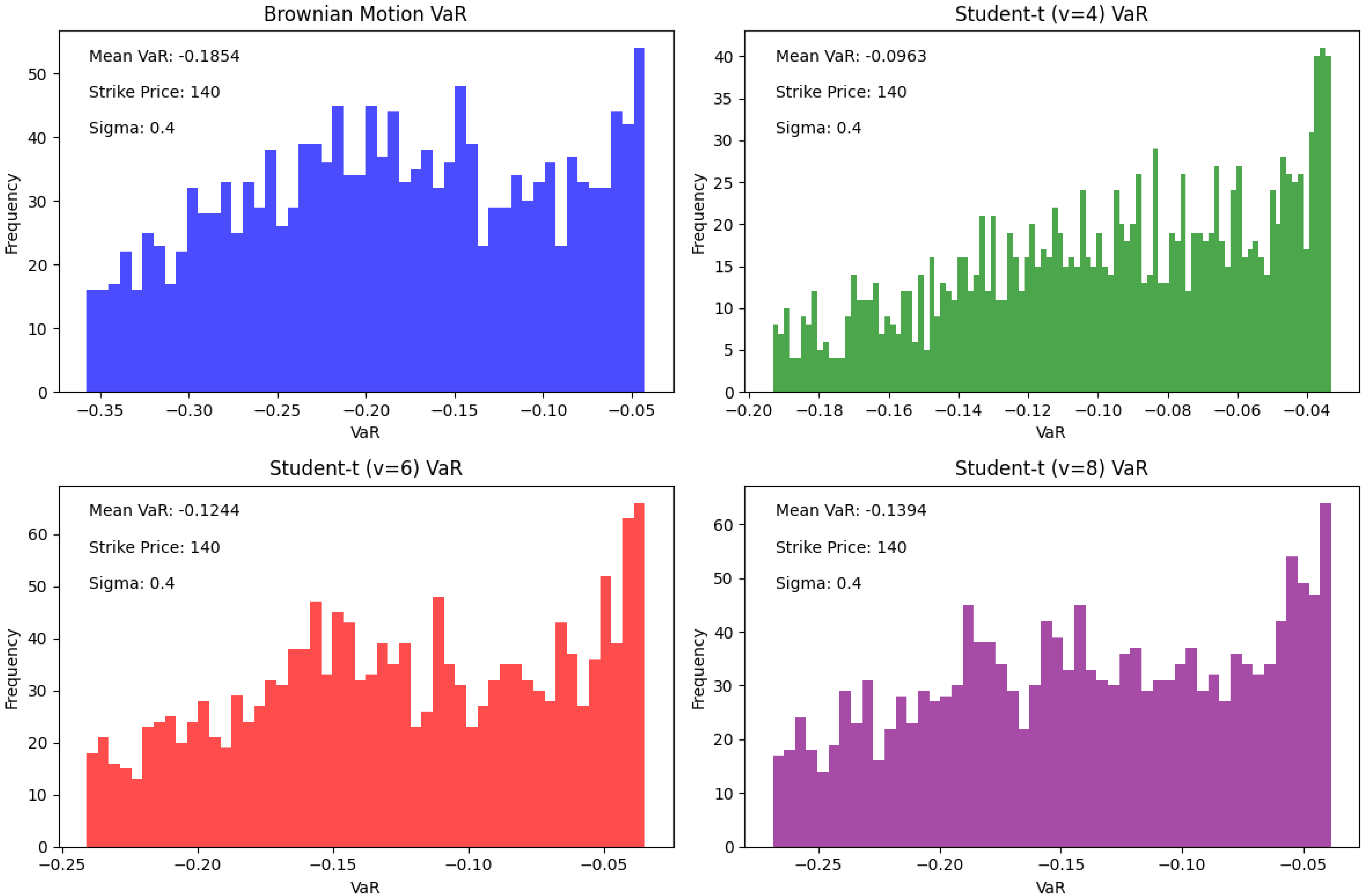
In [ ]:
```
os.makedirs('plots', exist_ok=True)
for K in [100,140,180]:
    #K = 100
    for sigma in [0.2,0.4]:
    #sigma = 0.2
        key = f'K={K}_sigma={sigma}'

        # Extracting the data for the specific K and sigma
        VaRs_Brownian = results[key]['VaRs_Brownian']
        VaRs_Student_t_1 = results[key]['VaRs_Student_t_1']
        VaRs_Student_t_2 = results[key]['VaRs_Student_t_2']
        VaRs_Student_t_3 = results[key]['VaRs_Student_t_3']

        # We need to clean out some data that will generate extreme VaR which does not make sense in our model. Those extreme data is not the extreme
        # loss but the computational error from solving the equations
        VaRs_Brownian = quantile_removal(VaRs_Brownian)
        VaRs_Student_t_1 = quantile_removal(VaRs_Student_t_1)
        VaRs_Student_t_2 = quantile_removal(VaRs_Student_t_2)
        VaRs_Student_t_3 = quantile_removal(VaRs_Student_t_3)
        # Function to add text annotations to the plots
        def add_annotations(ax, mean_var, K, sigma):
            ax.text(0.05, 0.95, f'Mean VaR: {mean_var:.4f}', transform=ax.transAxes, fontsize=10, verticalalignment='top')
            ax.text(0.05, 0.85, f'Strike Price: {K}', transform=ax.transAxes, fontsize=10, verticalalignment='top')
            ax.text(0.05, 0.75, f'Sigma: {sigma}', transform=ax.transAxes, fontsize=10, verticalalignment='top')

        # Plotting the histograms
        plt.figure(figsize=(12, 8))

        ax1 = plt.subplot(2, 2, 1)
        plt.hist(VaRs_Brownian, bins=50, color='blue', alpha=0.7)
        plt.xlabel('VaR')
        plt.ylabel('Frequency')
        plt.title('Brownian Motion VaR')
        add_annotations(ax1, np.mean(VaRs_Brownian), K, sigma)

        ax2 = plt.subplot(2, 2, 2)
        plt.hist(VaRs_Student_t_1, bins=100, color='green', alpha=0.7)
        plt.xlabel('VaR')
        plt.ylabel('Frequency')
        plt.title('Student-t (v=4) VaR')
```

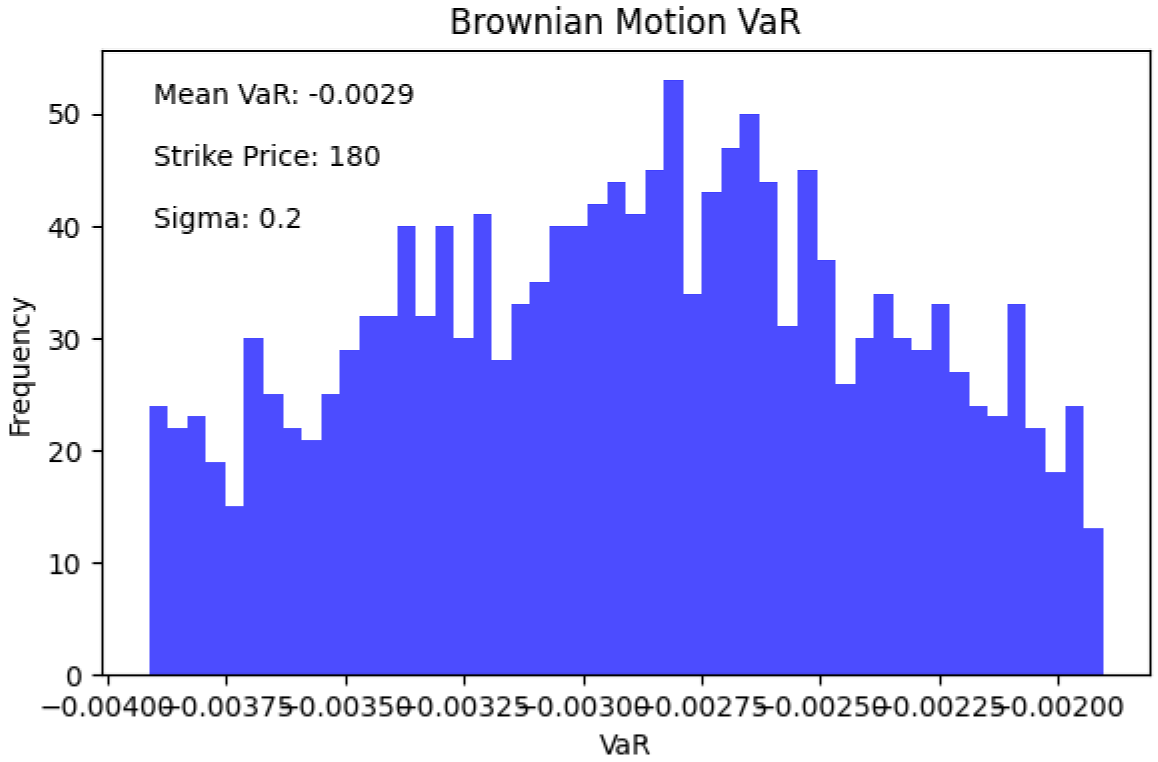
```
    add_annotations(ax2, np.mean(VaRs_Student_t_1), K, sigma)

    ax3 = plt.subplot(2, 2, 3)
    plt.hist(VaRs_Student_t_2, bins=50, color='red', alpha=0.7)
    plt.xlabel('VaR')
    plt.ylabel('Frequency')
    plt.title('Student-t (v=6) VaR')
    add_annotations(ax3, np.mean(VaRs_Student_t_2), K, sigma)

    ax4 = plt.subplot(2, 2, 4)
    plt.hist(VaRs_Student_t_3, bins=50, color='purple', alpha=0.7)
    plt.xlabel('VaR')
    plt.ylabel('Frequency')
    plt.title('Student-t (v=8) VaR')
    add_annotations(ax4, np.mean(VaRs_Student_t_3), K, sigma)

    plt.tight_layout()



    #plt.show()
    file_name = f'plots/Var_{K}_{sigma}.png'
    plt.savefig(file_name)
    plt.show()
    plt.close()
```

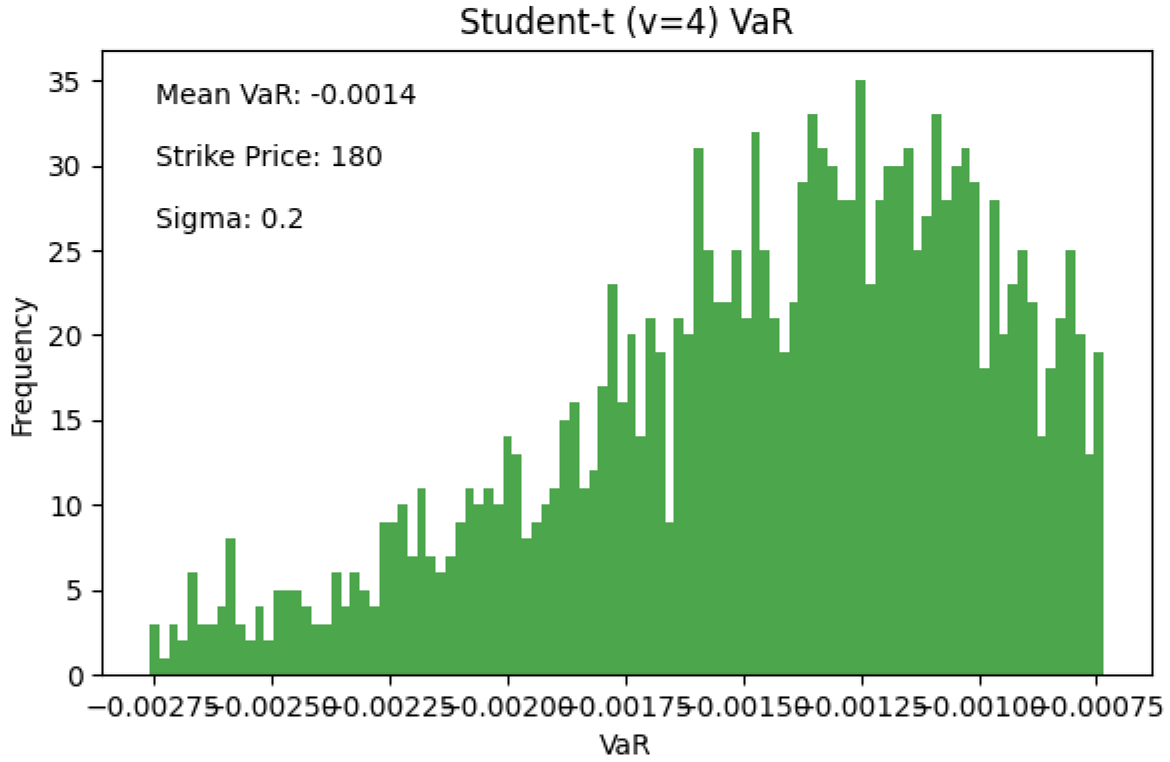

Brownian Motion VaR
Mean VaR: -0.0720
Strike Price: 100
Sigma: 0.2
Frequency
VaR
Student-t (v=4) VaR
Mean VaR: -0.0726
Strike Price: 100
Sigma: 0.2
Frequency
VaR
Student-t (v=6) VaR
Mean VaR: -0.0735
Strike Price: 100
Sigma: 0.2
Frequency
VaR
Student-t (v=8) VaR
Mean VaR: -0.0713
Strike Price: 100
Sigma: 0.2
Frequency
VaR

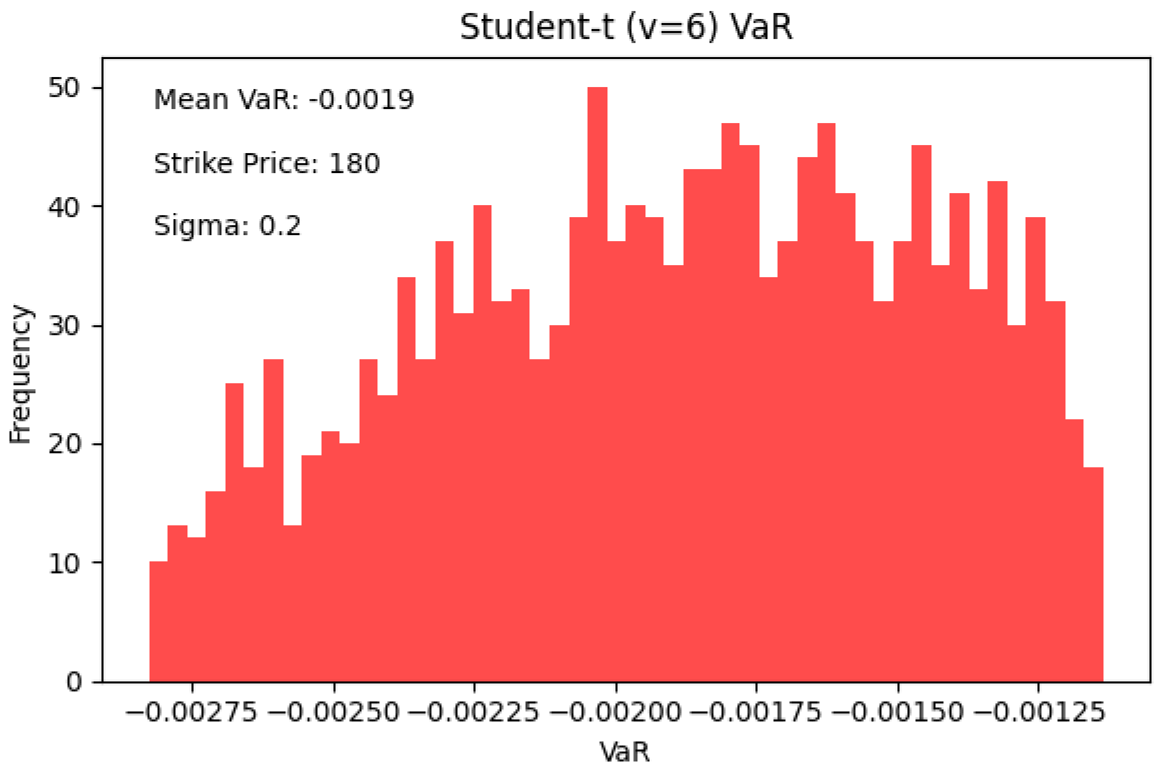

Brownian Motion VaR
Mean VaR: -0.1921
Strike Price: 100
Sigma: 0.4
Frequency
VaR
Student-t (v=4) VaR
Mean VaR: -0.1432
Strike Price: 100
Sigma: 0.4
Frequency
VaR
Student-t (v=6) VaR
Mean VaR: -0.1690
Strike Price: 100
Sigma: 0.4
Frequency
VaR
Student-t (v=8) VaR
Mean VaR: -0.1718
Strike Price: 100
Sigma: 0.4
Frequency
VaR

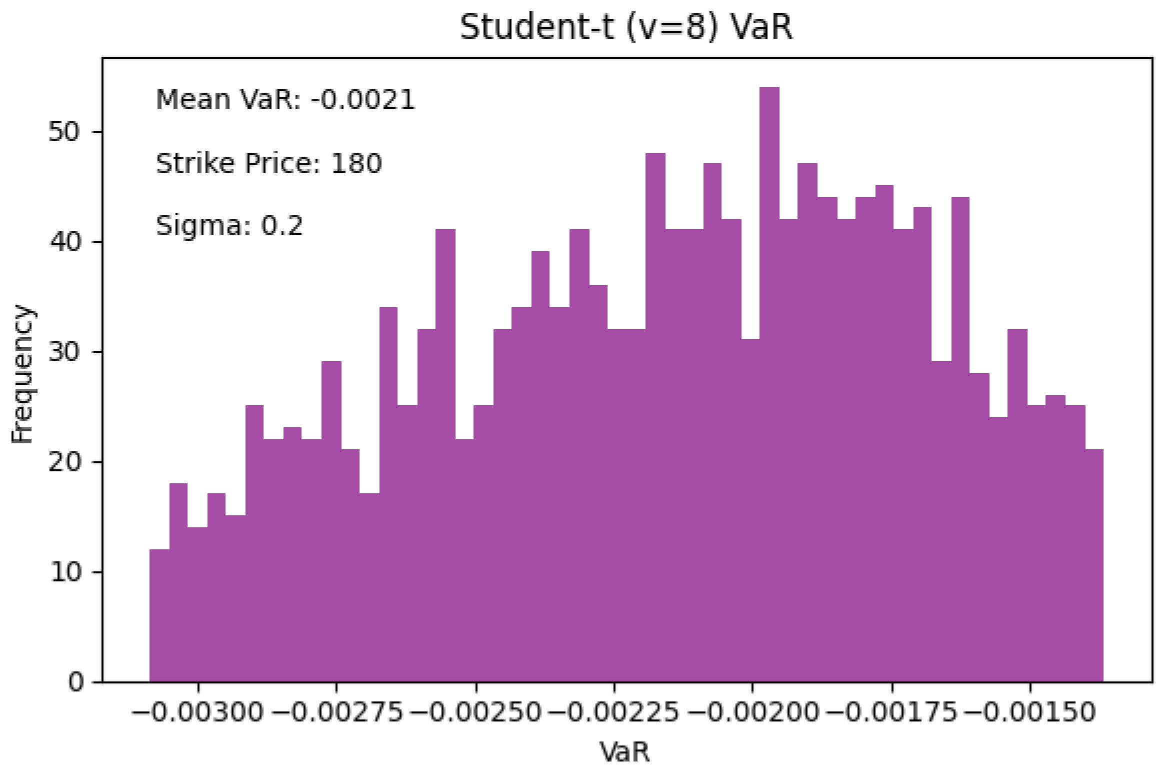

Brownian Motion VaR
Mean VaR: -0.0372
Strike Price: 140
Sigma: 0.2
Frequency
VaR
Student-t (v=4) VaR
Mean VaR: -0.0220
Strike Price: 140
Sigma: 0.2
Frequency
VaR
Student-t (v=6) VaR
Mean VaR: -0.0273
Strike Price: 140
Sigma: 0.2
Frequency
VaR
Student-t (v=8) VaR
Mean VaR: -0.0297
Strike Price: 140
Sigma: 0.2
Frequency
VaR

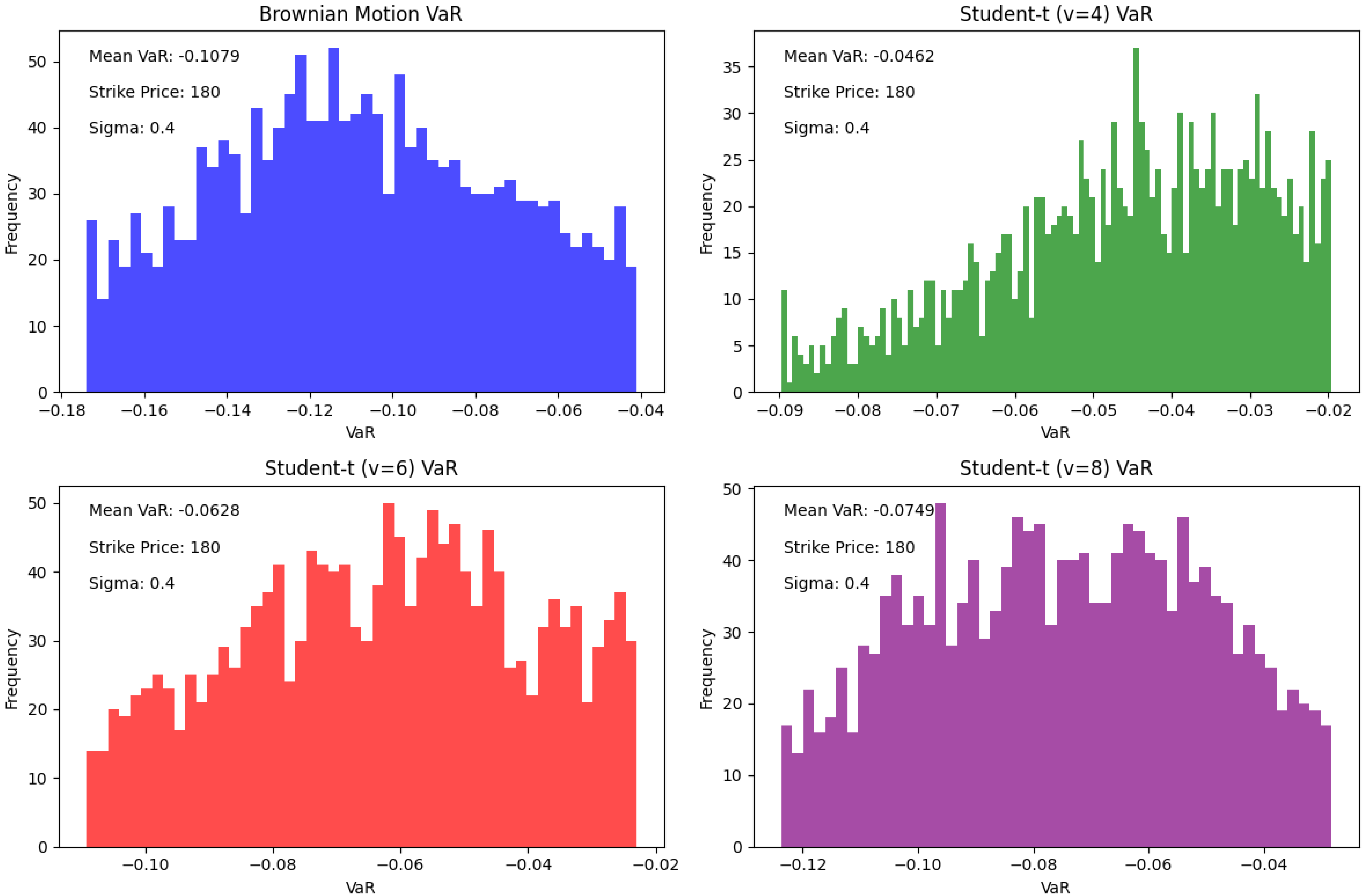

Brownian Motion VaR
Mean VaR: -0.1854
Strike Price: 140
Sigma: 0.4
Frequency
VaR
Student-t (v=4) VaR
Mean VaR: -0.0963
Strike Price: 140
Sigma: 0.4
Frequency
VaR
Student-t (v=6) VaR
Mean VaR: -0.1244
Strike Price: 140
Sigma: 0.4
Frequency
VaR
Student-t (v=8) VaR
Mean VaR: -0.1394
Strike Price: 140
Sigma: 0.4
Frequency
VaR


Brownian Motion VaR
Mean VaR: -0.0029
Strike Price: 180
Sigma: 0.2
Frequency
VaR


Student-t (v=4) VaR
Mean VaR: -0.0014
Strike Price: 180
Sigma: 0.2
Frequency
VaR


Student-t (v=6) VaR
Mean VaR: -0.0019
Strike Price: 180
Sigma: 0.2
Frequency
VaR


Student-t (v=8) VaR
Mean VaR: -0.0021
Strike Price: 180
Sigma: 0.2
Frequency
VaR

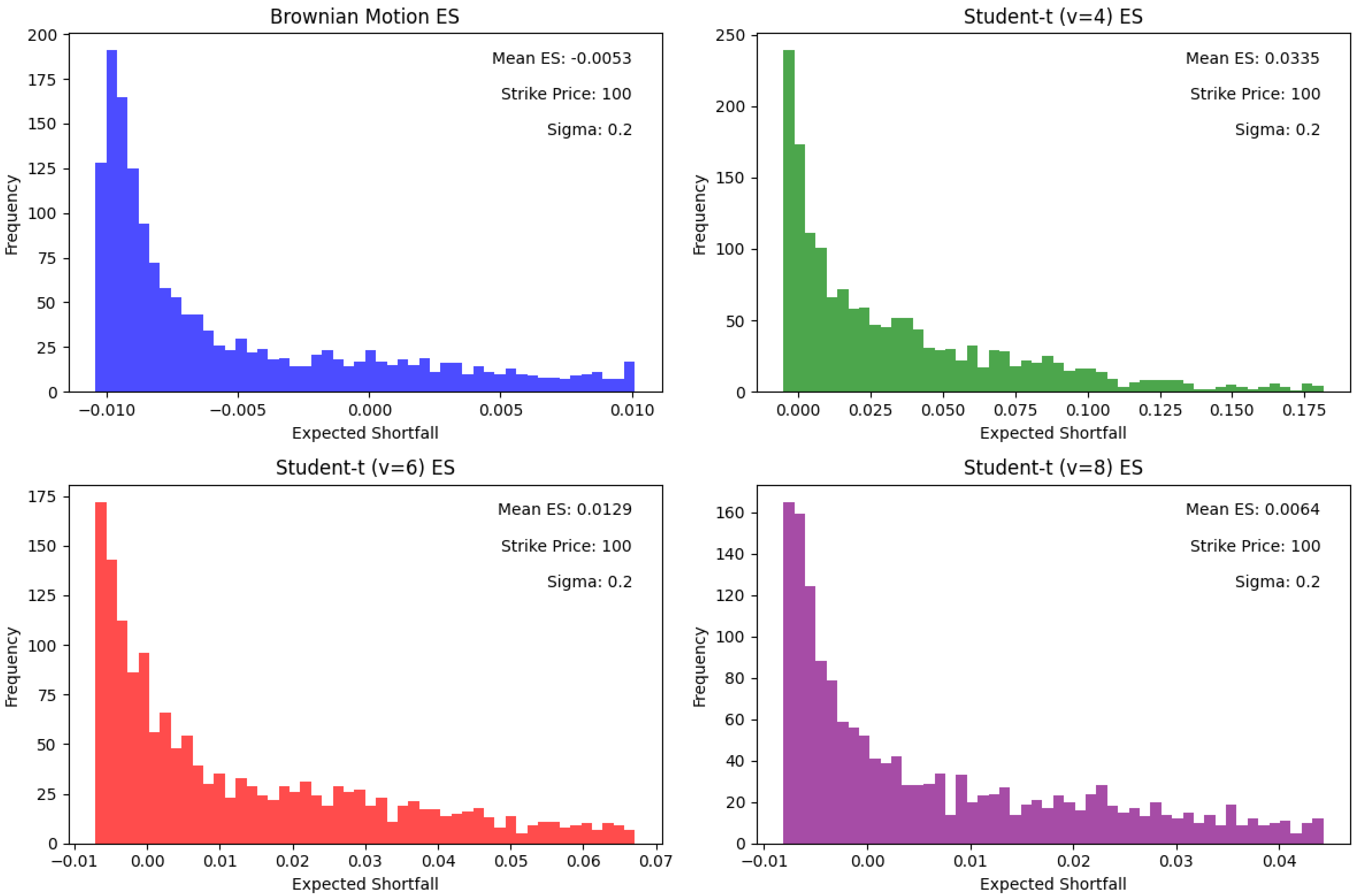


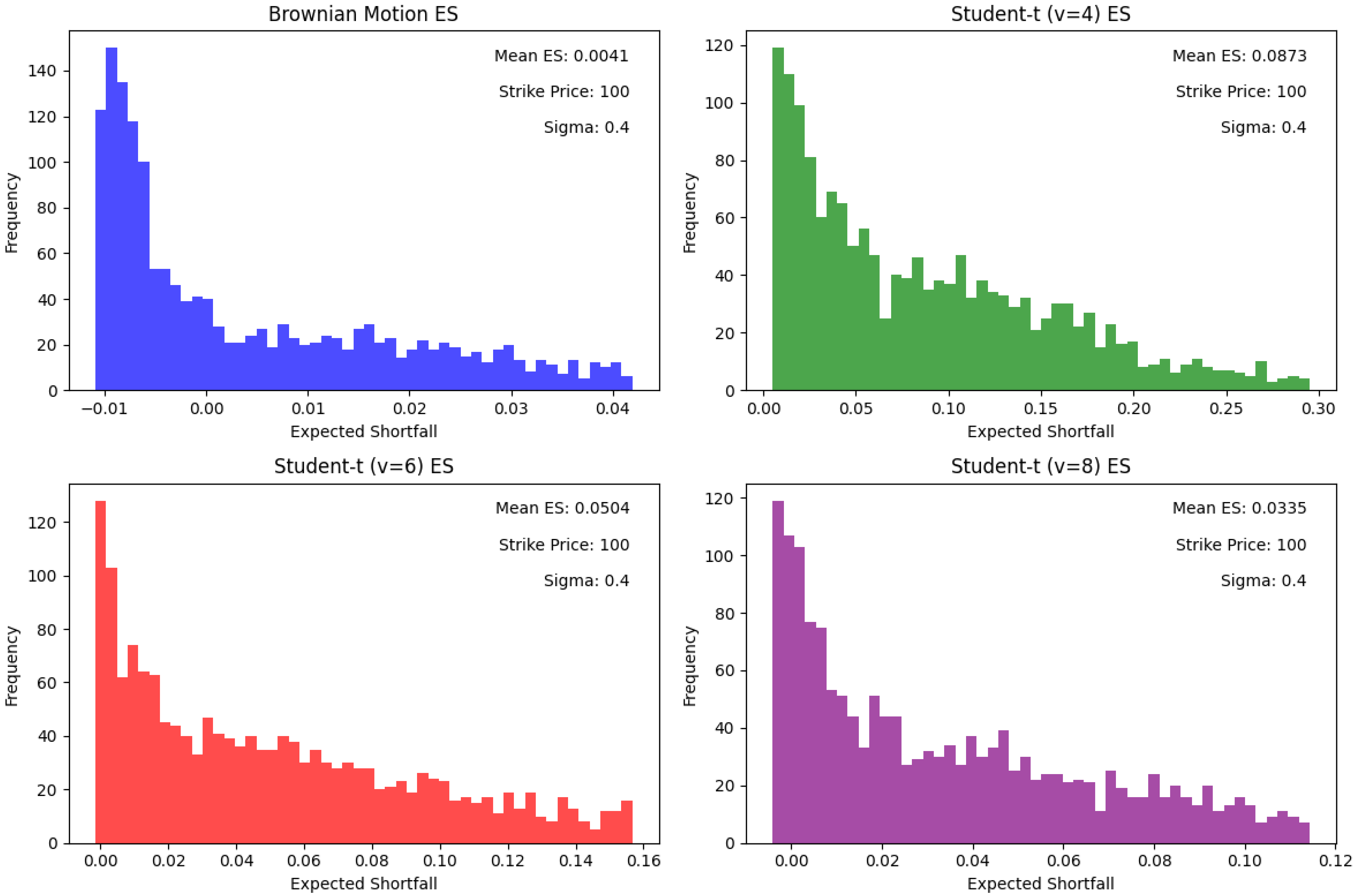
plots for ES at different strike prices, Volatility and different price paths

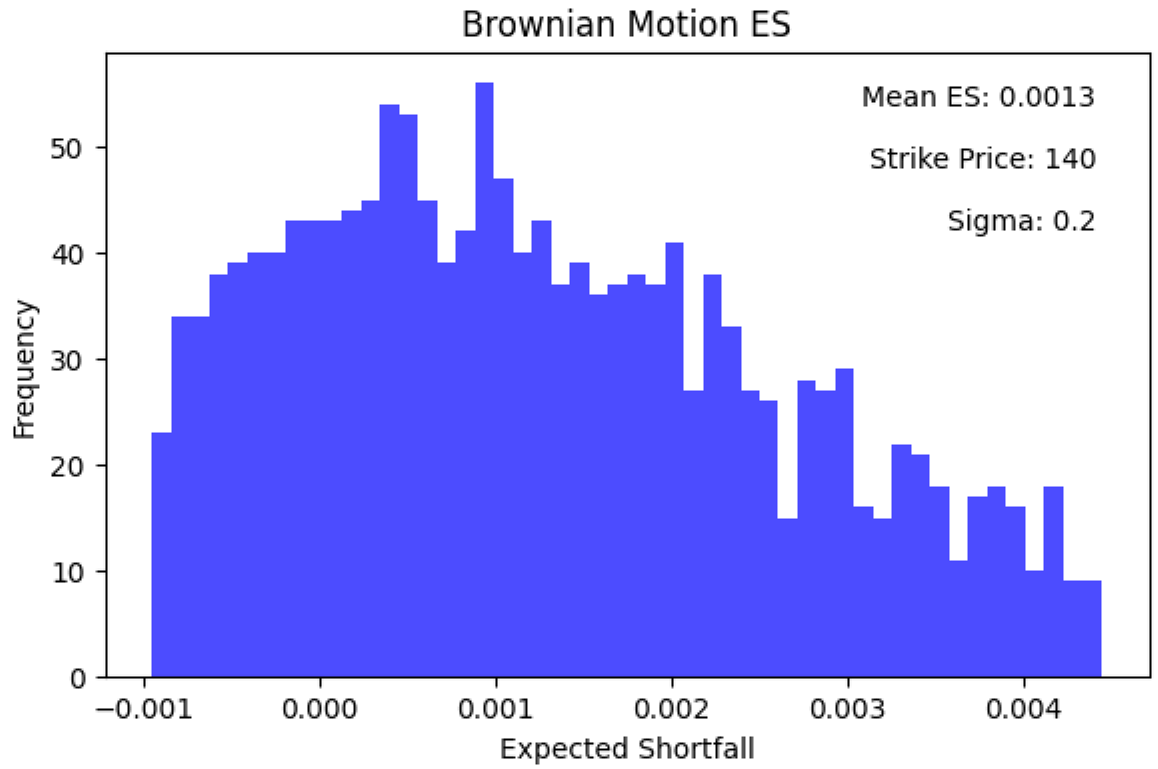
In [ ]:
```
#
for K in [100,140,180]:
    #K = 100
    for sigma in [0.2,0.4]:

        key = f'K={K}_sigma={sigma}'

        # Extracting the ES data for the specific K and sigma
```

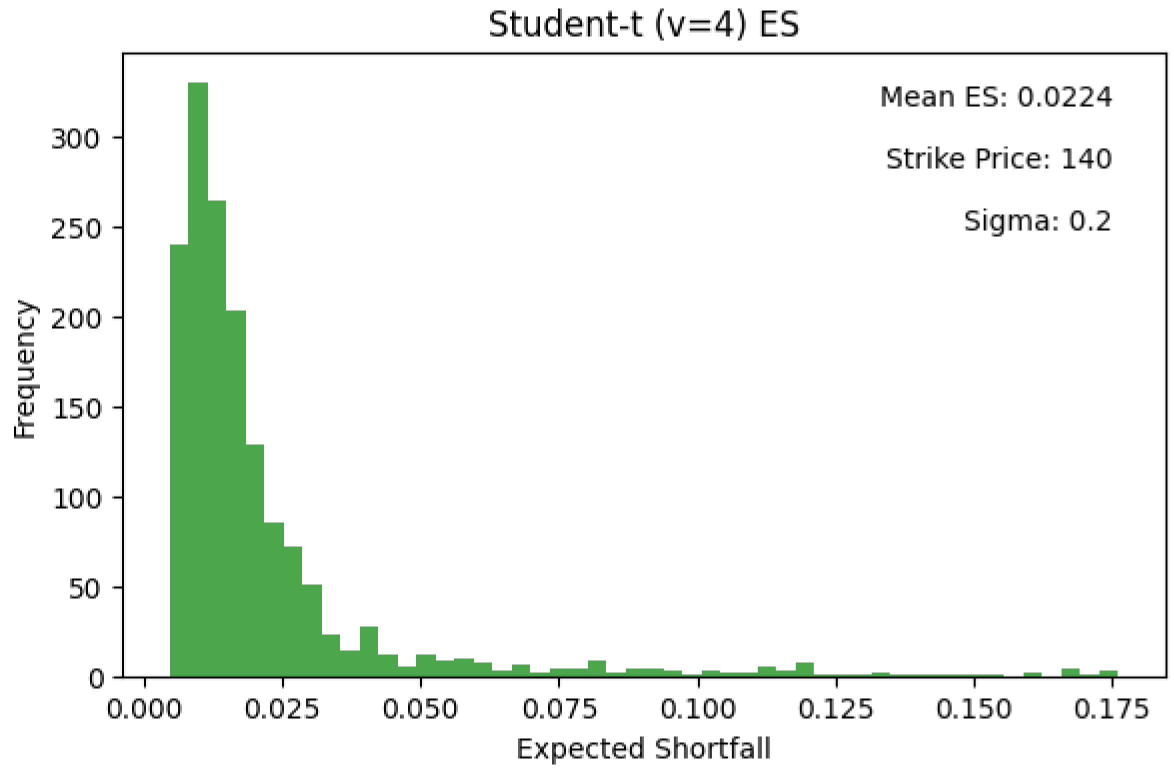
```
    ES_Brownian = results[key]['ES_Brownian']
    ES_Student_t_1 = results[key]['ES_Student_t_1']
    ES_Student_t_2 = results[key]['ES_Student_t_2']
    ES_Student_t_3 = results[key]['ES_Student_t_3']


    # We need to clean out some data that will generate extreme VaR which does not make sense in our model. Those extreme data is not the extreme
    # loss but the computational error from solving the equations
    ES_Brownian = quantile_removal(ES_Brownian)
    ES_Student_t_1 = quantile_removal(ES_Student_t_1)
    ES_Student_t_2 = quantile_removal(ES_Student_t_2)
    ES_Student_t_3 = quantile_removal(ES_Student_t_3)

    # Function to add text annotations to the plots
    def add_annotations(ax, mean_es, K, sigma):
        ax.text(0.95, 0.95, f'Mean ES: {mean_es:.4f}', transform=ax.transAxes, fontsize=10, verticalalignment='top', horizontalalignment='right')
        ax.text(0.95, 0.85, f'Strike Price: {K}', transform=ax.transAxes, fontsize=10, verticalalignment='top', horizontalalignment='right')
        ax.text(0.95, 0.75, f'Sigma: {sigma}', transform=ax.transAxes, fontsize=10, verticalalignment='top', horizontalalignment='right')

    # Plotting the histograms for Expected Shortfall
    plt.figure(figsize=(12, 8))

    # Brownian Motion ES
    ax1 = plt.subplot(2, 2, 1)
    plt.hist(ES_Brownian, bins=50, color='blue', alpha=0.7)
    plt.xlabel('Expected Shortfall')
    plt.ylabel('Frequency')
    plt.title('Brownian Motion ES')
    add_annotations(ax1, np.mean(ES_Brownian), K, sigma)

    # Student-t (v=4) ES
    ax2 = plt.subplot(2, 2, 2)
    plt.hist(ES_Student_t_1, bins=50, color='green', alpha=0.7)
    plt.xlabel('Expected Shortfall')
    plt.ylabel('Frequency')
    plt.title('Student-t (v=4) ES')
    add_annotations(ax2, np.mean(ES_Student_t_1), K, sigma)

    # Student-t (v=6) ES
    ax3 = plt.subplot(2, 2, 3)
    plt.hist(ES_Student_t_2, bins=50, color='red', alpha=0.7)
    plt.xlabel('Expected Shortfall')
    plt.ylabel('Frequency')
    plt.title('Student-t (v=6) ES')
    add_annotations(ax3, np.mean(ES_Student_t_2), K, sigma)

    # Student-t (v=8) ES
    ax4 = plt.subplot(2, 2, 4)
    plt.hist(ES_Student_t_3, bins=50, color='purple', alpha=0.7)
    plt.xlabel('Expected Shortfall')
    plt.ylabel('Frequency')
    plt.title('Student-t (v=8) ES')
    add_annotations(ax4, np.mean(ES_Student_t_3), K, sigma)

    plt.tight_layout()
    #plt.show()

    #plt.show()
    file_name = f'plots/ES_{K}_{sigma}.png'
    plt.savefig(file_name)
```

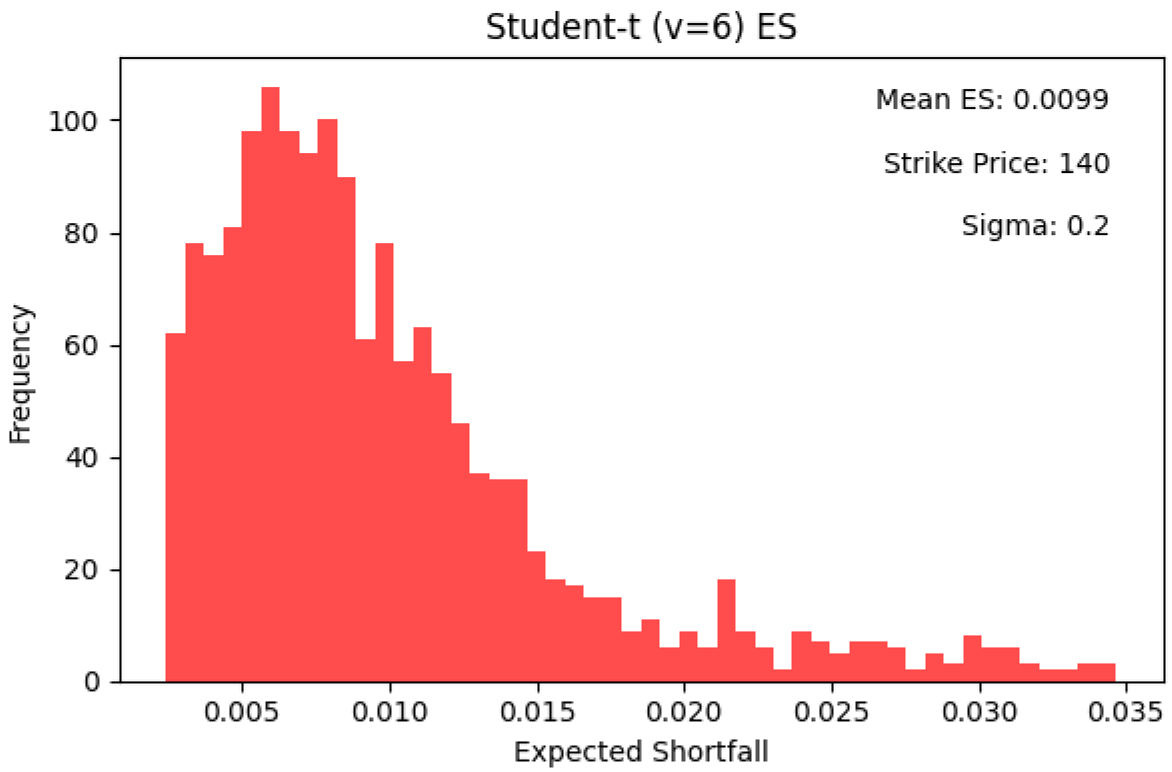
```
    plt.show()
    plt.close()
```

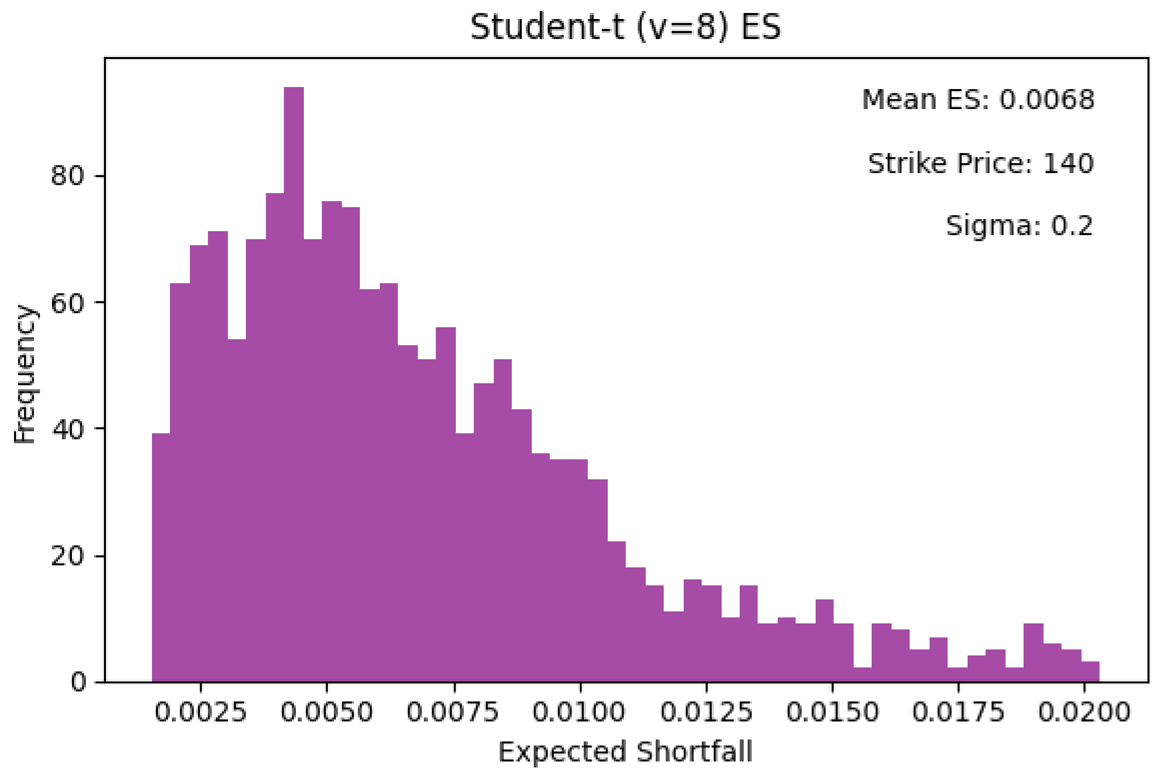


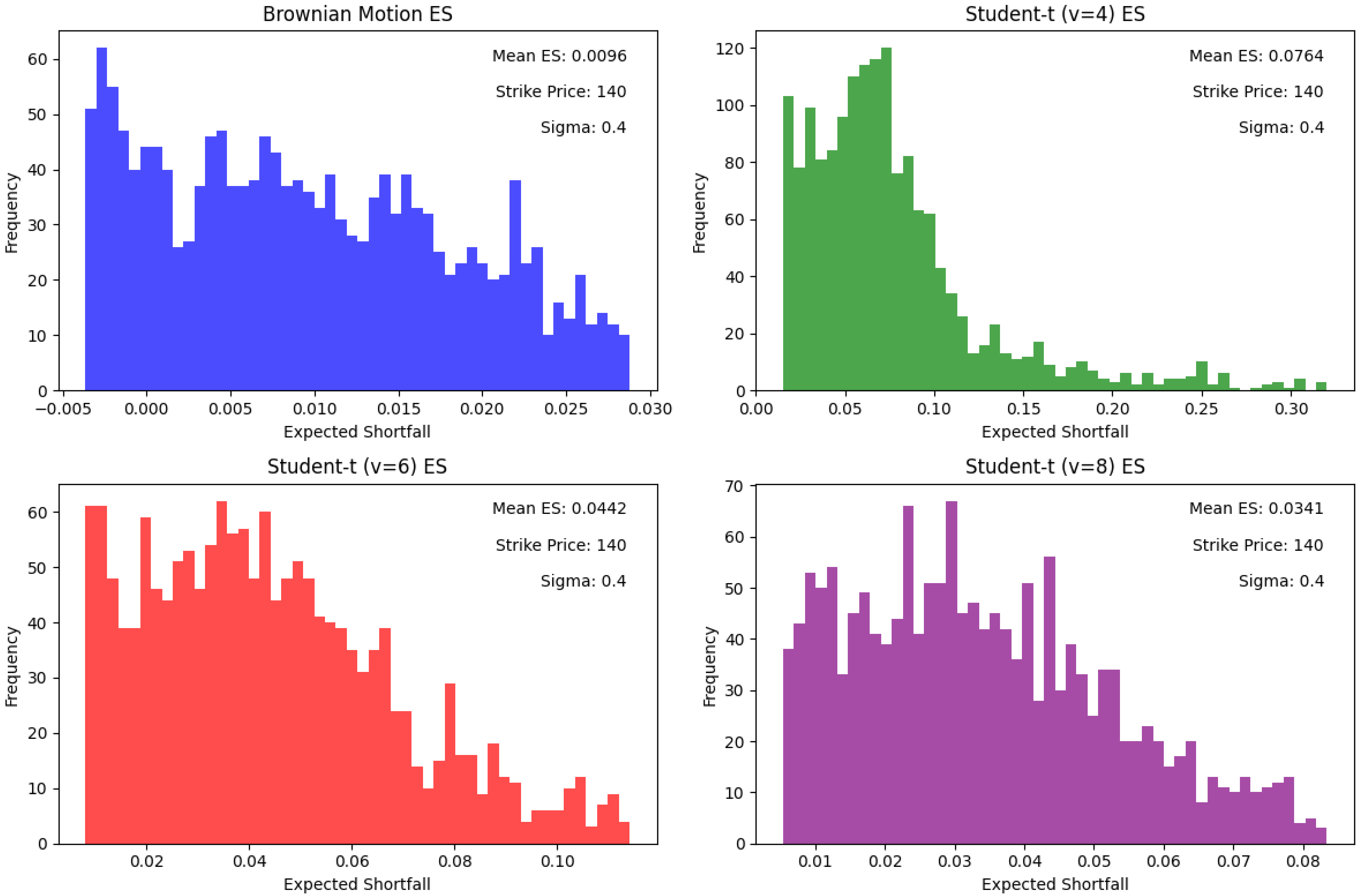

Brownian Motion ES
Mean ES: 0.0041
Strike Price: 100
Sigma: 0.4
Frequency
Expected Shortfall
Student-t (v=4) ES
Mean ES: 0.0873
Strike Price: 100
Sigma: 0.4
Frequency
Expected Shortfall
Student-t (v=6) ES
Mean ES: 0.0504
Strike Price: 100
Sigma: 0.4
Frequency
Expected Shortfall
Student-t (v=8) ES
Mean ES: 0.0335
Strike Price: 100
Sigma: 0.4
Frequency
Expected Shortfall

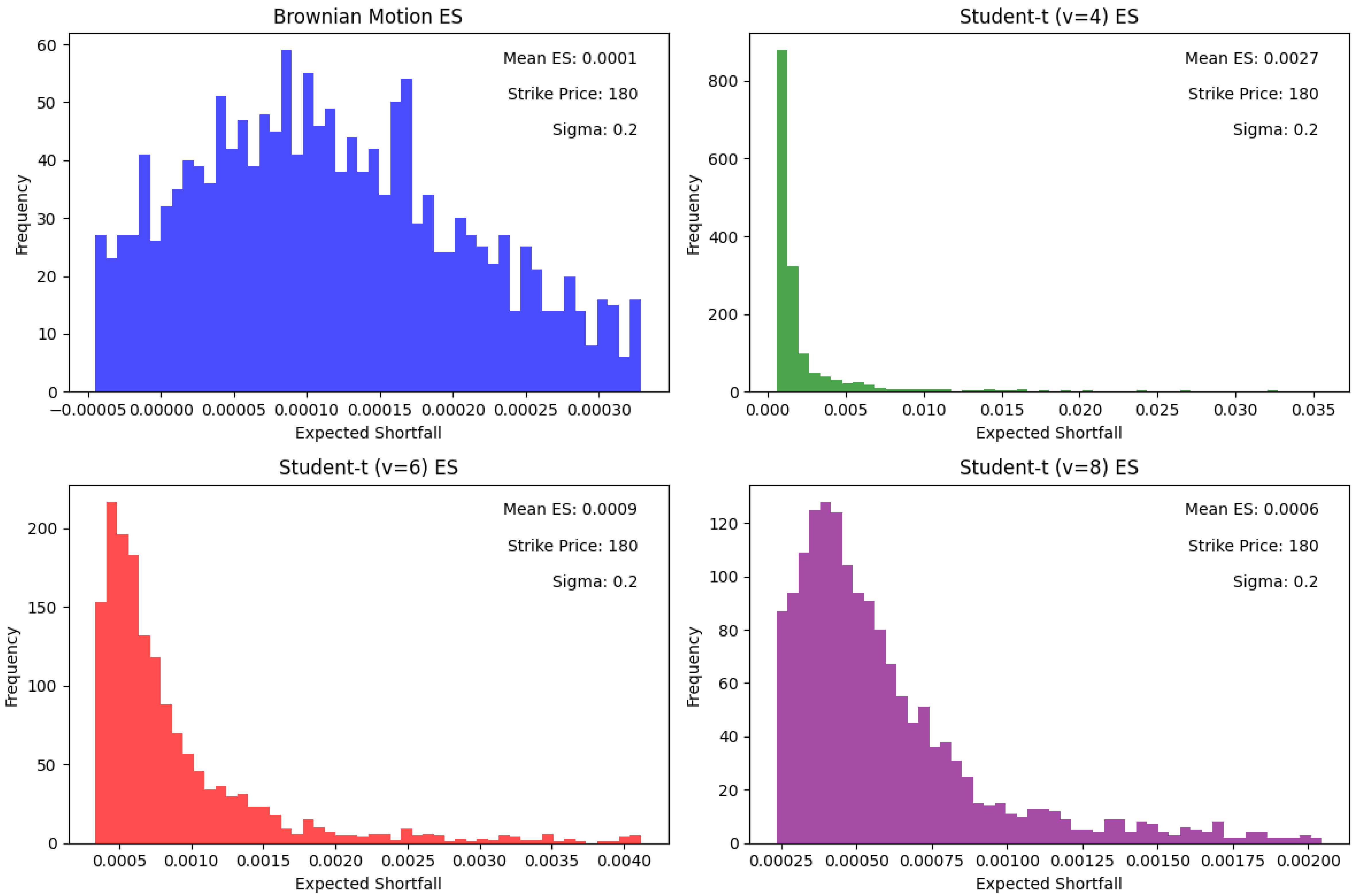

Brownian Motion ES
Mean ES: 0.0013
Strike Price: 140
Sigma: 0.2
Frequency
Expected Shortfall

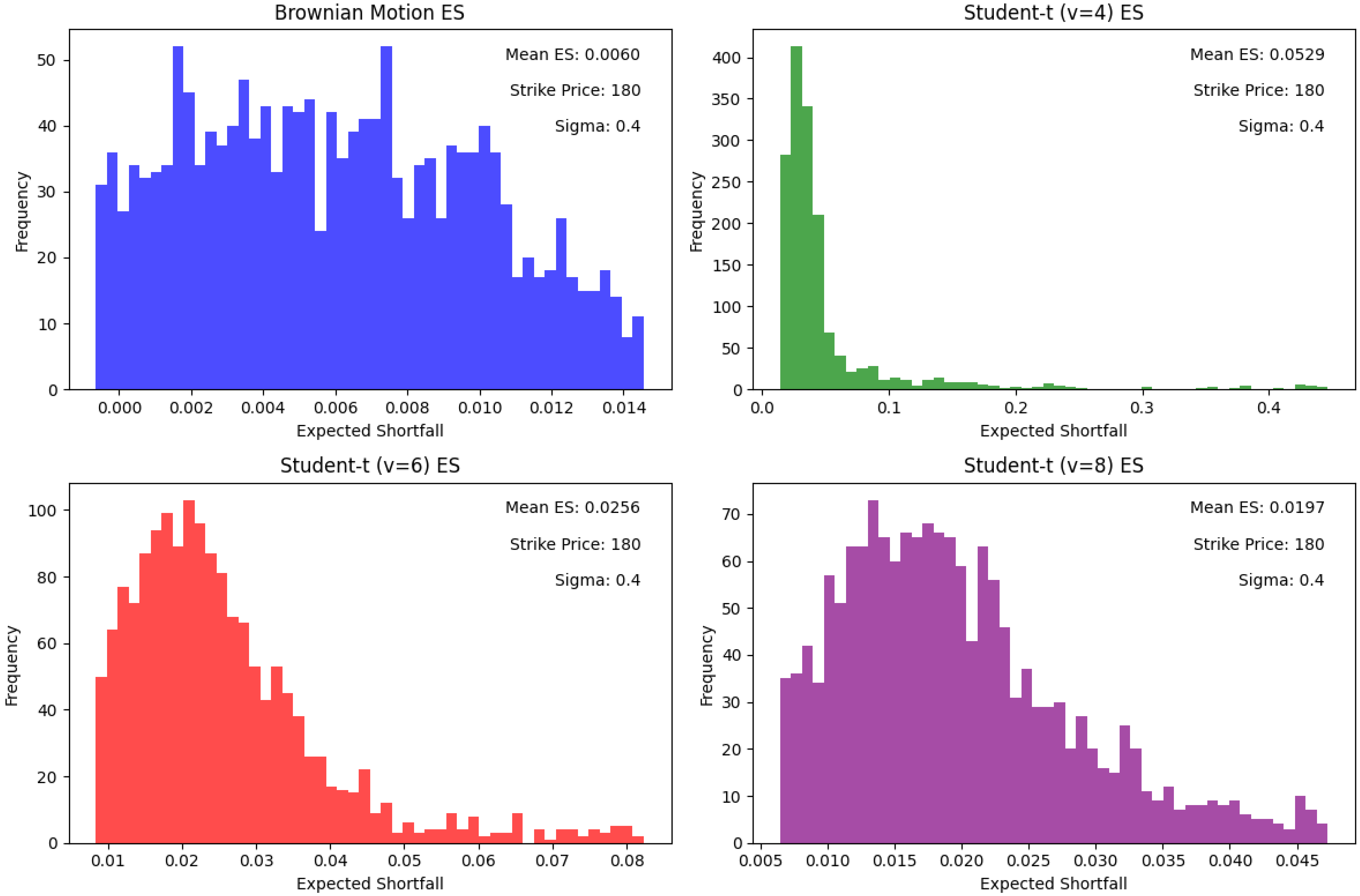

Student-t (v=4) ES
Mean ES: 0.0224
Strike Price: 140
Sigma: 0.2
Frequency
Expected Shortfall


Student-t (v=6) ES
Mean ES: 0.0099
Strike Price: 140
Sigma: 0.2
Frequency
Expected Shortfall


Student-t (v=8) ES
Mean ES: 0.0068
Strike Price: 140
Sigma: 0.2
Frequency
Expected Shortfall


Brownian Motion ES
Mean ES: 0.0096
Strike Price: 140
Sigma: 0.4
Frequency
Expected Shortfall
Student-t (v=4) ES
Mean ES: 0.0764
Strike Price: 140
Sigma: 0.4
Frequency
Expected Shortfall
Student-t (v=6) ES
Mean ES: 0.0442
Strike Price: 140
Sigma: 0.4
Frequency
Expected Shortfall
Student-t (v=8) ES
Mean ES: 0.0341
Strike Price: 140
Sigma: 0.4
Frequency
Expected Shortfall

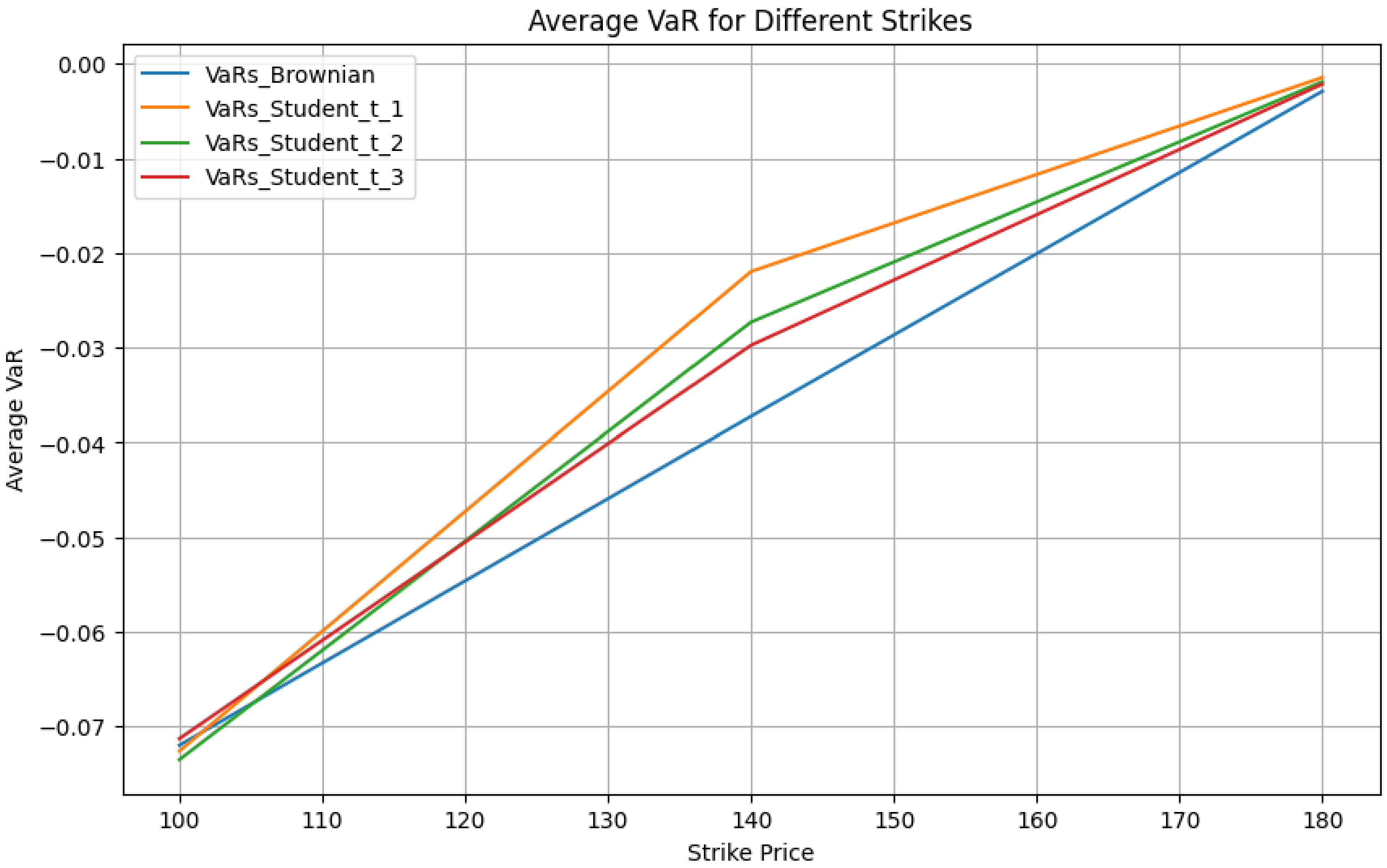

Brownian Motion ES
Mean ES: 0.0001
Strike Price: 180
Sigma: 0.2
Frequency
Expected Shortfall
Student-t (v=4) ES
Mean ES: 0.0027
Strike Price: 180
Sigma: 0.2
Frequency
Expected Shortfall
Student-t (v=6) ES
Mean ES: 0.0009
Strike Price: 180
Sigma: 0.2
Frequency
Expected Shortfall
Student-t (v=8) ES
Mean ES: 0.0006
Strike Price: 180
Sigma: 0.2
Frequency
Expected Shortfall

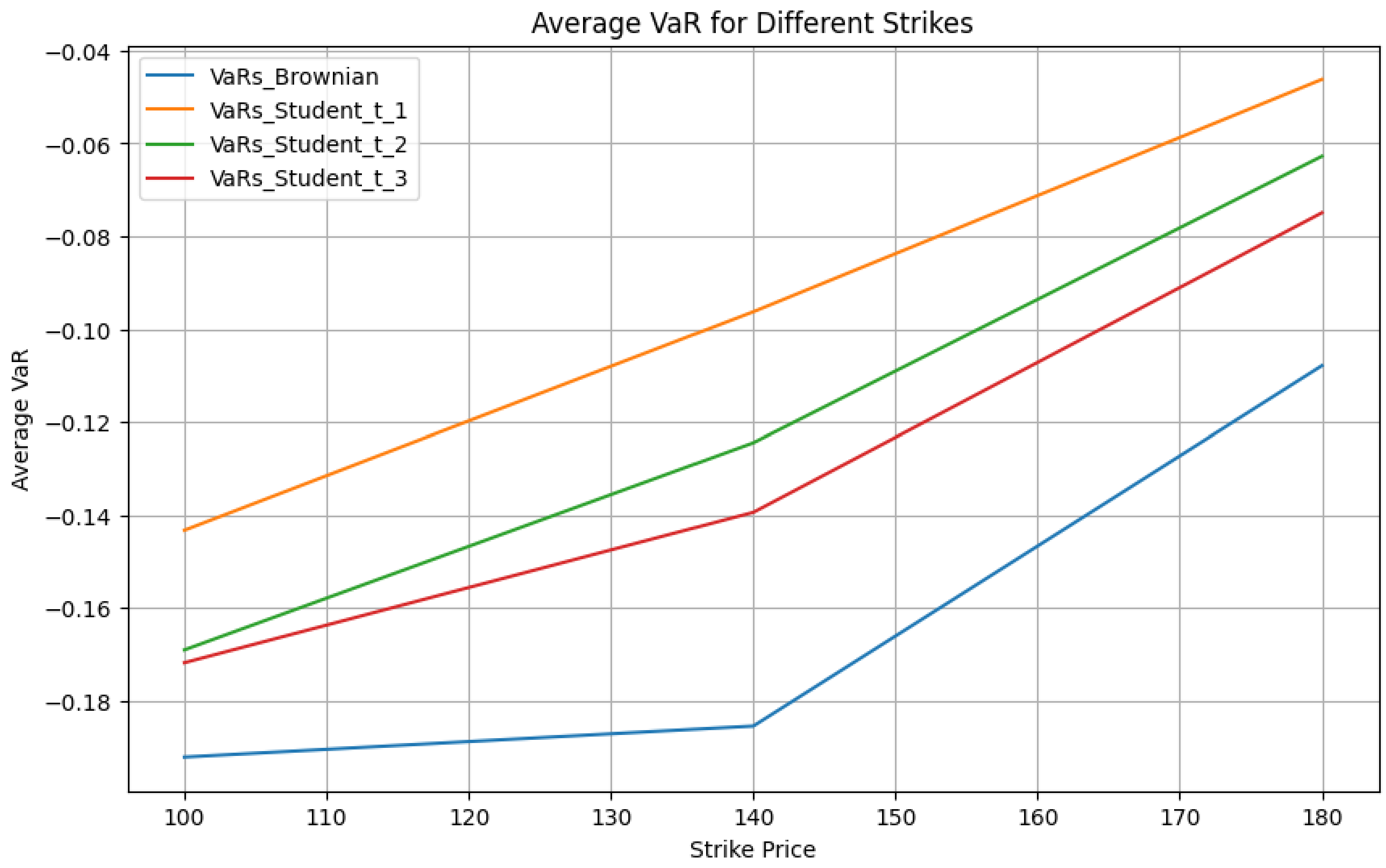


Mean VaR on different strike price

In [ ]:
```
# Mean VaR on different strike price

strike_prices = [100, 140,180]
for sigma in [0.2,0.4]:
    #sigma = 0.2  # Example volatility value
    distribution_types = ['VaRs_Brownian', 'VaRs_Student_t_1', 'VaRs_Student_t_2', 'VaRs_Student_t_3']

    plt.figure(figsize=(10, 6))
```

```python
for dist in distribution_types:
    avg_vars = []
    for K in strike_prices:
        key = f'K={K}_sigma={sigma}'

        avg_var = np.mean(quantile_removal(results[key][dist]))
        avg_vars.append(avg_var)
    plt.plot(strike_prices, avg_vars, label=dist)

plt.xlabel('Strike Price')
plt.ylabel('Average VaR')
plt.title('Average VaR for Different Strikes')
plt.legend()
plt.grid(True)
#plt.show()

#plt.show()
file_name = f'plots/AvgVaRonStrike_{K}_{sigma}.png'
plt.savefig(file_name)
plt.show()

plt.close()
```

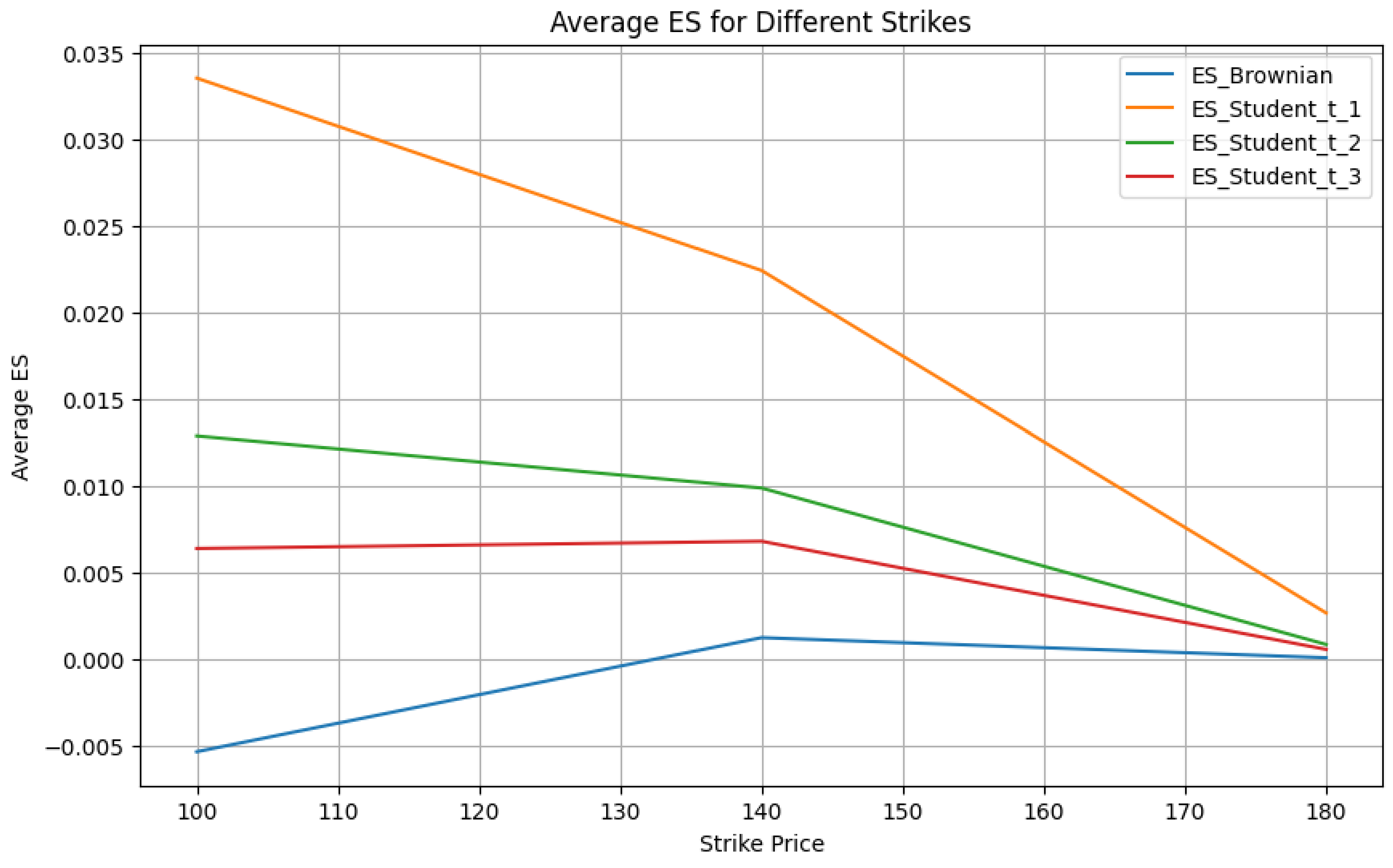


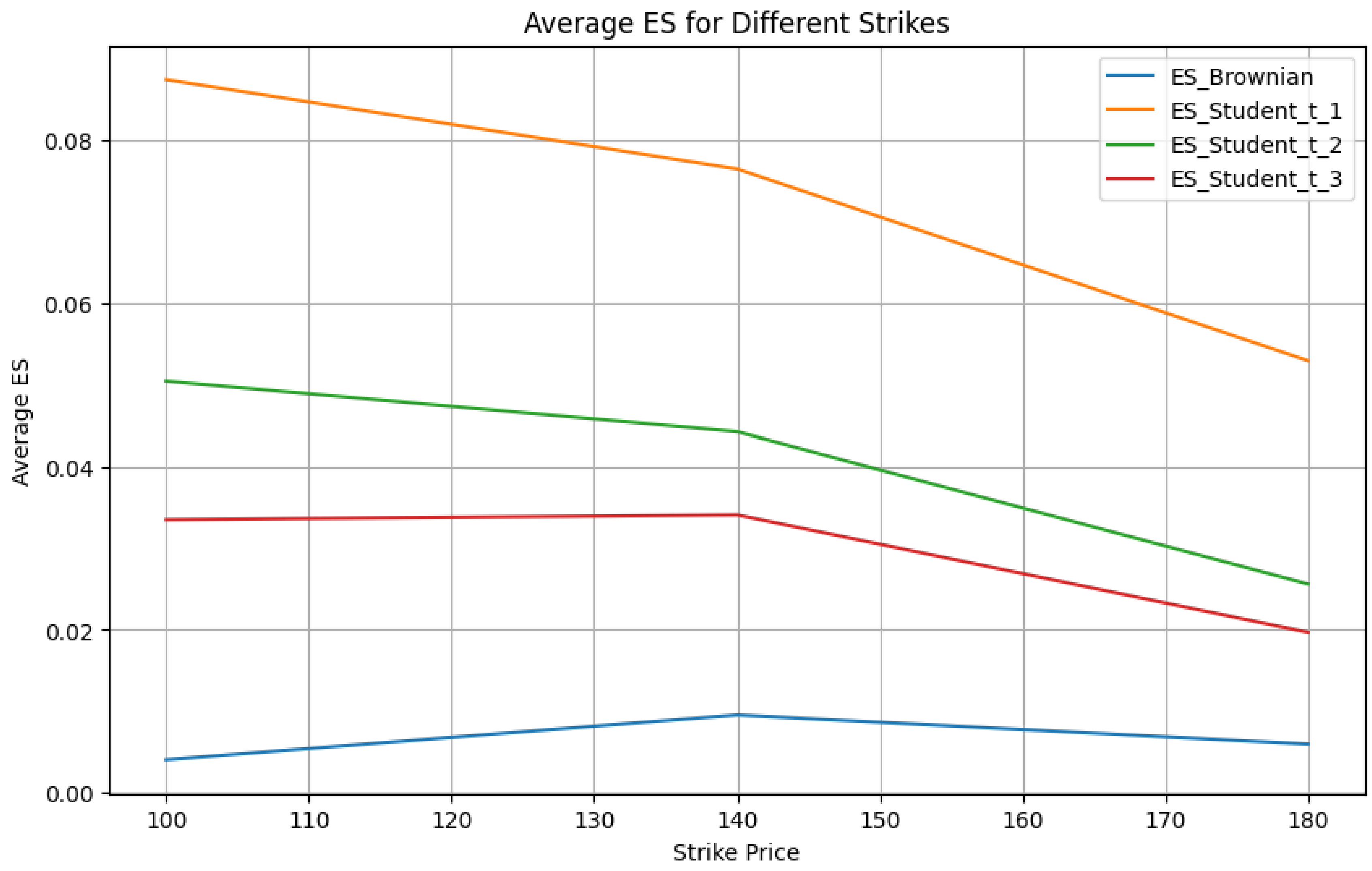


Mean ES on different strike price

In [ ]:
```
# Mean ES for different strike price
strike_prices = [100,  140, 180]
for sigma in [0.2,0.4]:

    plt.figure(figsize=(10, 6))

    distribution_types = ['ES_Brownian', 'ES_Student_t_1', 'ES_Student_t_2', 'ES_Student_t_3']

    for dist in distribution_types:
        avg_es = []
        for K in strike_prices:
            key = f'K={K}_sigma={sigma}'
            avg_es_value = np.mean(quantile_removal(results[key][dist]))
            avg_es.append(avg_es_value)
        plt.plot(strike_prices, avg_es, label=dist)

    plt.xlabel('Strike Price')
    plt.ylabel('Average ES')
    plt.title('Average ES for Different Strikes')
    plt.legend()
    plt.grid(True)
    #plt.show()
```

```
    file_name = f'plots/AvgESonStrike_{K}_{sigma}.png'
    plt.savefig(file_name)
    plt.show()
    plt.close()
```




Average ES for Different Strikes
ES_Brownian
ES_Student_t_1
ES_Student_t_2
ES_Student_t_3
Average ES
0.00
0.02
0.04
0.06
0.08
Strike Price
100
110
120
130
140
150
160
170
180

In [ ]:

In [ ]: