Part 1

In [ ]:
```
from datetime import datetime, date, timedelta
import pandas as pd
import yfinance as yf


def fetch_option_snapshot(expiry_list, valuation_date=date.today()):

    symbols = ['TSLA', 'SPY']


    irx = yf.Ticker("^IRX")
    rf_hist = irx.history(
        start=valuation_date,
        end=valuation_date + timedelta(days=1)
    )
    rf = rf_hist['Close'].iloc[-1] / 100.0
    print(f"[{valuation_date}] r = {rf:.4f}")


    vix = yf.Ticker("^VIX")
    vix_hist = vix.history(
        start=valuation_date,
        end=valuation_date + timedelta(days=1)
    )
    vix_price = vix_hist['Close'].iloc[-1]
    print(f"[{valuation_date}] VIX = {vix_price:.2f}")

    collected_frames = []

    for ticker_symbol in symbols:

        ticker_obj = yf.Ticker(ticker_symbol)

        price_hist = ticker_obj.history(
            start=valuation_date,
            end=valuation_date + timedelta(days=1)
        )

        spot = price_hist['Close'].iloc[-1]
        print(f"{ticker_symbol} spot = {spot:.2f}")

        available_expiries = [
            e for e in ticker_obj.options
            if e in expiry_list
        ]

        for expiry_str in available_expiries:

            try:
                chain_data = ticker_obj.option_chain(expiry_str)
```

```python
                maturity_date = datetime.strptime(
                    expiry_str, "%Y-%m-%d"
                ).date()

                tau = (maturity_date - valuation_date).days / 365.0

                for contract_side, df_raw in {
                    'call': chain_data.calls,
                    'put': chain_data.puts
                }.items():

                    df = df_raw.copy()

                    df['Symbol'] = ticker_symbol
                    df['Type'] = contract_side
                    df['S'] = spot
                    df['T'] = tau
                    df['r'] = rf
                    df['Expiry'] = expiry_str
                    df['vix'] = vix_price

                    mid = (df['bid'] + df['ask']) / 2
                    df['Price'] = mid.where(
                        (df['bid'] > 0) & (df['ask'] > 0),
                        df['lastPrice']
                    )

                    df.rename(columns={'strike': 'Strike'}, inplace=True)

                    selected_cols = [
                        'Symbol', 'Type', 'Expiry', 'Strike',
                        'Price', 'S', 'T', 'r',
                        'impliedVolatility', 'bid', 'ask', 'vix'
                    ]

                    collected_frames.append(df[selected_cols])

            except Exception as err:
                print(f"{ticker_symbol} | {expiry_str} failed")

    if collected_frames:
        return pd.concat(collected_frames, ignore_index=True)

    return pd.DataFrame()
```

In [201…

```python
expiries = ['2026-02-20', '2026-03-20', '2026-04-17']

DATA1 = fetch_option_snapshot(expiries, valuation_date=date(2026, 2, 12))
DATA2 = fetch_option_snapshot(expiries, valuation_date=date(2026, 2, 13))
```

```
[2026-02-12] r = 0.0360
[2026-02-12] VIX = 20.82
TSLA spot = 417.07
SPY spot = 681.27
[2026-02-13] r = 0.0359
[2026-02-13] VIX = 20.60
TSLA spot = 417.44
SPY spot = 681.75
```

Symbols Downloaded

The following symbols are being downloaded:

- **TSLA**
  - Represents Tesla, Inc. common stock.
  - Options on TSLA aalows retail investors like the normal person to speculate on the fate of the company.
- **SPY**
  - The SPDR S&P 500 ETF Trust.
  - An **Exchange-Traded Fund (ETF)** that tracks the performance of the S&P 500 Index.
  - This ETF allows the retail investor to speculate on the fate of the industry in general.
- **^VIX**
  - The CBOE Volatility Index.
  - Measures the market's expected 30-day forward-looking volatility.
  - Derived from S&P 500 index options.

Option Structure

- Each option contract is defined by:
  - Underlying symbol (TSLA or SPY)
  - Expiration date
  - Strike price
  - Contract type (call or put)
- The selected expirations:
  - February 20, 2026
  - March 20, 2026
  - April 17, 2026
- These are standard monthly options expiring on the third Friday of each month.

Part 2

Black Scholes Implementation

In [ ]:
```
import numpy as np
from scipy.stats import norm

S = 45
K = 40
T = 2
r = 0.1
vol = 0.1

d1 = (np.log(S/K) + (r + 0.5 * vol**2)*T ) / (vol * np.sqrt(T))

d2 = d1 - (vol * np.sqrt(T))

C = S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
P = K * np.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)


print('The value of d1 is: ', round(d1, 4))
print('The value of d2 is: ', round(d2, 4))
print('The price of the call option is: $', round(C, 2))
print('The price of the put option is: $', round(P, 2))
```

```
The value of d1 is:  2.3178
The value of d2 is:  2.1764
The price of the call option is: $ 12.27
The price of the put option is: $ 0.02
```

In [203…
```
def black_scholes(S, K, T, r, sigma, option_type='call'):
    d1 = (np.log(S/K) + (r + 0.5 * sigma**2)*T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    if option_type == 'call':
        return S * norm.cdf(d1) - K * np.exp(-r*T) * norm.cdf(d2)
    else:
        return K * np.exp(-r*T) * norm.cdf(-d2) - S * norm.cdf(-d1)
```

Bisection method

In [204…
```
import numpy as np
from scipy.stats import norm
import time

def black_scholes(S, K, T, r, sigma, option_type='call'):
    d1 = (np.log(S/K) + (r + 0.5*sigma**2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)

    if option_type == 'call':
```

```python
        return S*norm.cdf(d1) - K*np.exp(-r*T)*norm.cdf(d2)
    else:
        return K*np.exp(-r*T)*norm.cdf(-d2) - S*norm.cdf(-d1)


def bisection(func, a, b, tol=1e-6, max_iter=100):

    if func(a)*func(b) > 0:
        return None

    for _ in range(max_iter):
        mid = (a + b) / 2
        f_mid = func(mid)

        if abs(f_mid) < tol or (b - a)/2 < tol:
            return mid

        if func(a)*f_mid < 0:
            b = mid
        else:
            a = mid

    return (a + b)/2


def implied_vol_bisect(price, S, K, T, r, option_type):

    def objective(sigma):
        return black_scholes(S, K, T, r, sigma, option_type) - price

    return bisection(objective, 0.0001, 5.0, tol=1e-6)


df = DATA1.copy()
start_time = time.time()
iv_values = []

for _, row in df.iterrows():
    iv = implied_vol_bisect(
        row['Price'],
        row['S'],
        row['Strike'],
        row['T'],
        row['r'],
        row['Type']
    )
    iv_values.append(iv)
bisect_time = time.time() - start_time
df['IV_Bisect'] = iv_values

df = df.dropna(subset=['IV_Bisect'])


print("\n--- Strict ATM IV (Closest Strike) ---")

for symbol in ['TSLA', 'SPY']:
```

```
    symbol_df = df[df['Symbol'] == symbol].copy()

    for T_val in sorted(symbol_df['T'].unique()):

        temp = symbol_df[symbol_df['T'] == T_val].copy()

        # Closest strike to spot
        temp['Distance'] = abs(temp['Strike'] - temp['S'])
        atm_row = temp.loc[temp['Distance'].idxmin()]

        days = int(round(T_val * 365))

        print(f"{symbol} | {days} days | ATM IV = {atm_row['IV_Bisect']:.4f}")


print("\n--- ATM Band Average IV (0.95–1.05) ---")

band = df[(df['S']/df['Strike'] >= 0.95) &
          (df['S']/df['Strike'] <= 1.05)]

grouped = band.groupby(['Symbol','T'])['IV_Bisect'].mean()

for (symbol, T_val), iv in grouped.items():
    days = int(round(T_val * 365))
    print(f"{symbol} | {days} days | Avg IV = {iv:.4f}")
```

```
--- Strict ATM IV (Closest Strike) ---
TSLA | 8 days | ATM IV = 0.3674
TSLA | 36 days | ATM IV = 0.4310
TSLA | 64 days | ATM IV = 0.4453
SPY | 8 days | ATM IV = 0.1667
SPY | 36 days | ATM IV = 0.1696
SPY | 64 days | ATM IV = 0.1631

--- ATM Band Average IV (0.95–1.05) ---
SPY | 8 days | Avg IV = 0.1596
SPY | 36 days | Avg IV = 0.1703
SPY | 64 days | Avg IV = 0.1679
TSLA | 8 days | Avg IV = 0.3645
TSLA | 36 days | Avg IV = 0.4264
TSLA | 64 days | Avg IV = 0.4412
```

Implement Newton/Secant/Muller

In [205…

```
import time
from scipy.stats import norm

def vega(S, K, T, r, sigma):

    if sigma <= 0 or T <= 0:
        return 1e-8

    d1 = (np.log(S/K) + (r + 0.5*sigma**2)*T) / (sigma*np.sqrt(T))
    return S * np.sqrt(T) * norm.pdf(d1)
```

```python
def newton_method(func, derivative, x0, tol=1e-6, max_iter=100):

    x = x0

    for _ in range(max_iter):

        f_val = func(x)
        d_val = derivative(x)

        if abs(d_val) < 1e-10:
            return None

        x_new = x - f_val/d_val

        if abs(x_new - x) < tol:
            return x_new

        x = x_new

    return x


def implied_vol_newton(price, S, K, T, r, option_type):

    def objective(sigma):
        return black_scholes(S, K, T, r, sigma, option_type) - price

    def derivative(sigma):
        return vega(S, K, T, r, sigma)

    return newton_method(objective, derivative, 0.3, tol=1e-6)
```

In [206…

```python
df_newton = DATA1.copy()

start_time = time.time()

iv_newton = []

for _, row in df_newton.iterrows():

    iv = implied_vol_newton(
        row['Price'],
        row['S'],
        row['Strike'],
        row['T'],
        row['r'],
        row['Type']
    )

    iv_newton.append(iv)

newton_time = time.time() - start_time

df_newton['IV_Newton'] = iv_newton
```

```
df_newton = df_newton.dropna(subset=['IV_Newton'])

print(f"Newton time: {newton_time:.4f} seconds")
```

```
Newton time: 3.0181 seconds
```

In [207…

```
print(f"Bisection time: {bisect_time:.4f} seconds")
print(f"Newton time: {newton_time:.4f} seconds")
```

```
Bisection time: 15.9533 seconds
Newton time: 3.0181 seconds
```

The Bisection method is really slow because it just repeatedly slices the index in half till it converges. However, Newton's method is faster as it uses Vega to approximate and converge quicker.

Implied Vol Report

In [ ]:

```
iv_table = df_newton.groupby(
    ['Symbol', 'Type', 'T']
)['IV_Newton'].mean().reset_index()

iv_table['Days'] = (iv_table['T'] * 365).round().astype(int)

iv_table = iv_table.sort_values(['Symbol', 'Type', 'Days'])

print("\nImplied Volatility Table (Newton):")
print(iv_table[['Symbol', 'Type', 'Days', 'IV_Newton']])
```

```
Implied Volatility Table (Newton):
   Symbol  Type  Days  IV_Newton
0     SPY  call     8   0.229326
1     SPY  call    36   0.214164
2     SPY  call    64   0.167649
3     SPY   put     8   0.235923
4     SPY   put    36   0.240659
5     SPY   put    64   0.236938
6    TSLA  call     8   0.400497
7    TSLA  call    36   0.459484
8    TSLA  call    64   0.466506
9    TSLA   put     8   0.386563
10   TSLA   put    36   0.463062
11   TSLA   put    64   0.479070
```

In [209…

```
print("\nAverage ATM IV (Closest Strike):")

for symbol in ['TSLA', 'SPY']:

    symbol_df = df_newton[df_newton['Symbol'] == symbol]

    for T_val in sorted(symbol_df['T'].unique()):

        temp = symbol_df[symbol_df['T'] == T_val].copy()
        temp['Distance'] = abs(temp['Strike'] - temp['S'])
```

```
        atm_row = temp.loc[temp['Distance'].idxmin()]

        days = int(round(T_val * 365))

        print(f"{symbol} | {days} days | ATM IV = {atm_row['IV_Newton']:.4f}")
```

```
Average ATM IV (Closest Strike):
TSLA | 8 days | ATM IV = 0.3674
TSLA | 36 days | ATM IV = 0.4310
TSLA | 64 days | ATM IV = 0.4453
SPY | 8 days | ATM IV = 0.1667
SPY | 36 days | ATM IV = 0.1696
SPY | 64 days | ATM IV = 0.1631
```

Put Call Parity

In [ ]:

```
import numpy as np
import pandas as pd

def put_call_parity_analysis(df):
    calls = df[df['Type'] == 'call'].copy()
    puts  = df[df['Type'] == 'put'].copy()

    merged = pd.merge(
        calls,
        puts,
        on=['Symbol', 'Strike', 'Expiry', 'T', 'r', 'S'],
        suffixes=('_c', '_p')
    )
    merged['mid_c'] = (merged['bid_c'] + merged['ask_c']) / 2
    merged['mid_p'] = (merged['bid_p'] + merged['ask_p']) / 2

    merged['PV_K'] = merged['Strike'] * np.exp(-merged['r'] * merged['T'])

    merged['Theory_Call'] = merged['mid_p'] + merged['S'] - merged['PV_K']
    merged['Theory_Put']  = merged['mid_c'] - merged['S'] + merged['PV_K']

    merged['Call_Valid'] = merged['Theory_Call'].between(
        merged['bid_c'], merged['ask_c']
    )

    merged['Put_Valid'] = merged['Theory_Put'].between(
        merged['bid_p'], merged['ask_p']
    )

    return merged


parity_df = put_call_parity_analysis(DATA1)

print("\nFraction satisfying parity (Call side):")
print(parity_df.groupby('Symbol')['Call_Valid'].mean())

print("\nFraction satisfying parity (Put side):")
print(parity_df.groupby('Symbol')['Put_Valid'].mean())
```

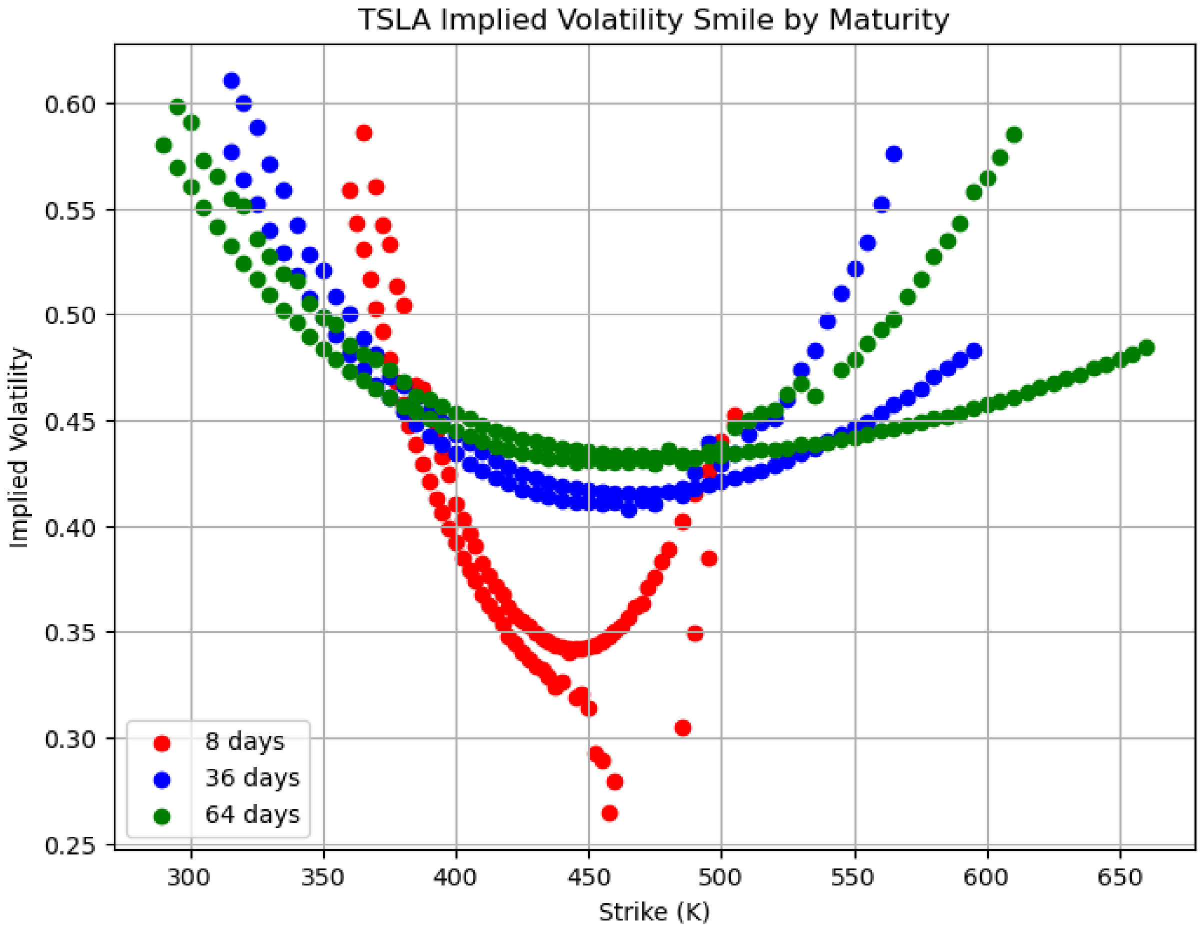
```
Fraction satisfying parity (Call side):
Symbol
SPY     0.411523
TSLA    0.333333
Name: Call_Valid, dtype: float64

Fraction satisfying parity (Put side):
Symbol
SPY     0.154321
TSLA    0.193900
Name: Put_Valid, dtype: float64
```

2D Plot Implied Volatility VS Strike K for closest to Maturity options

TSLA Smile

In [211…

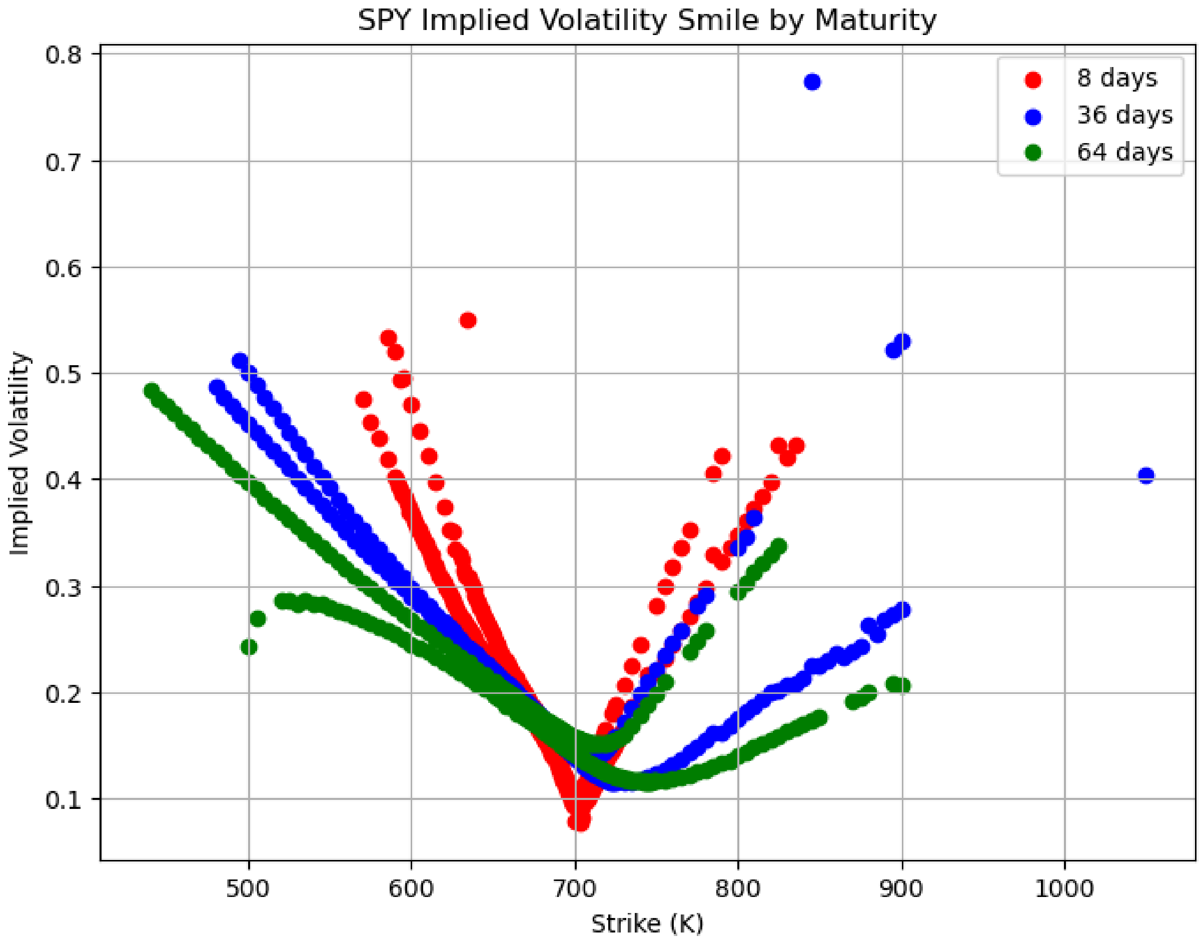
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))

maturities = sorted(df_newton['T'].unique())
colors = ['red', 'blue', 'green']

for i, T_val in enumerate(maturities):
    temp = df_newton[(df_newton['Symbol'] == 'TSLA') & (df_newton['T'] == T_val)]
    plt.scatter(temp['Strike'], temp['IV_Newton'],
                color=colors[i],
                label=f"{int(T_val*365)} days")

plt.xlabel("Strike (K)")
plt.ylabel("Implied Volatility")
plt.title("TSLA Implied Volatility Smile by Maturity")
plt.legend()
plt.grid(True)
plt.show()
```



SPY Smile

In [212…

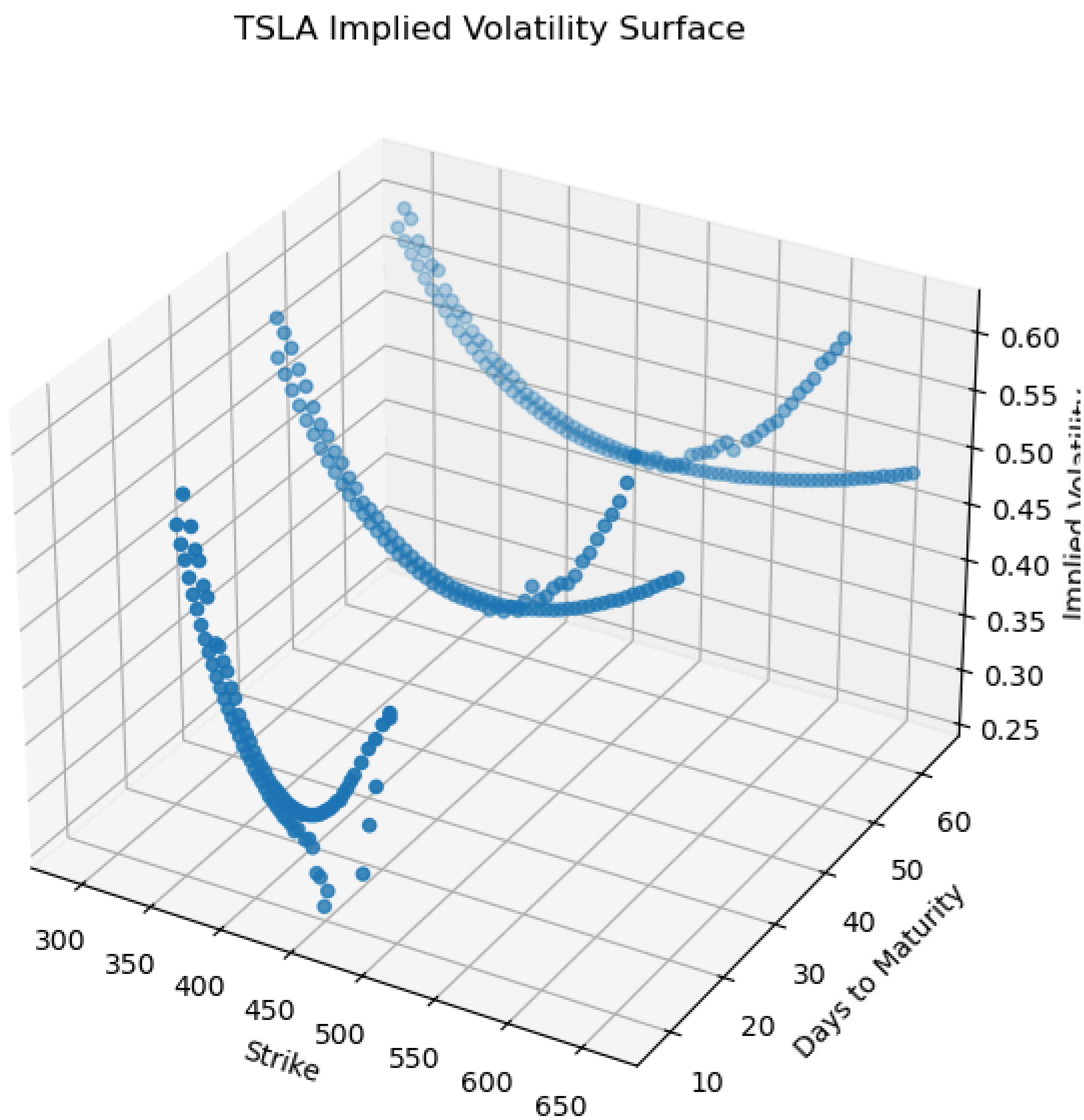
```
plt.figure(figsize=(8,6))

for i, T_val in enumerate(maturities):
    temp = df_newton[(df_newton['Symbol'] == 'SPY') & (df_newton['T'] == T_val)]
    plt.scatter(temp['Strike'], temp['IV_Newton'],
                color=colors[i],
                label=f"{int(T_val*365)} days")

plt.xlabel("Strike (K)")
plt.ylabel("Implied Volatility")
plt.title("SPY Implied Volatility Smile by Maturity")
plt.legend()
plt.grid(True)
plt.show()
```



Bonus: 3D Volatility Surface

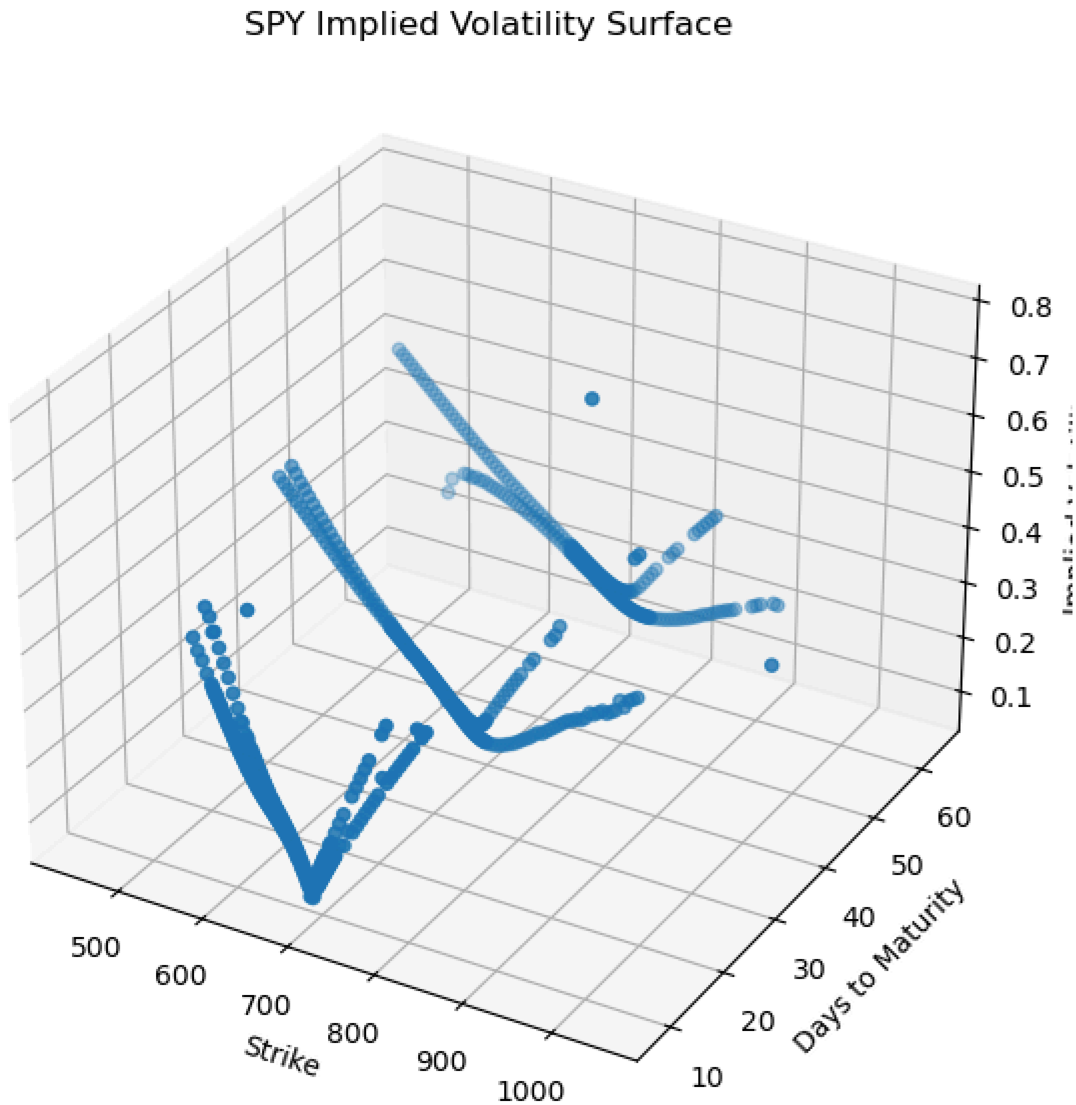
In [213…

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(9,7))
ax = fig.add_subplot(111, projection='3d')

tsla_df = df_newton[df_newton['Symbol'] == 'TSLA']

ax.scatter(
    tsla_df['Strike'],
    tsla_df['T'] * 365,
    tsla_df['IV_Newton']
)

ax.set_xlabel('Strike')
ax.set_ylabel('Days to Maturity')
ax.set_zlabel('Implied Volatility')

ax.set_title("TSLA Implied Volatility Surface")
plt.show()
```



In [214…

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(9,7))
ax = fig.add_subplot(111, projection='3d')

tsla_df = df_newton[df_newton['Symbol'] == 'SPY']

ax.scatter(
    tsla_df['Strike'],
    tsla_df['T'] * 365,
    tsla_df['IV_Newton']
)

ax.set_xlabel('Strike')
ax.set_ylabel('Days to Maturity')
ax.set_zlabel('Implied Volatility')

ax.set_title("SPY Implied Volatility Surface")
plt.show()
```



Greeks

In [215…

```
import numpy as np
from scipy.stats import norm

def bs_greeks_call(S, K, T, r, sigma):

    d1 = (np.log(S/K) + (r + 0.5*sigma**2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)

    delta = norm.cdf(d1)
    gamma = norm.pdf(d1) / (S * sigma * np.sqrt(T))
    vega  = S * np.sqrt(T) * norm.pdf(d1)

    return delta, gamma, vega
```

In [216…

```
def numerical_greeks(S, K, T, r, sigma, h=1e-4):

    # Delta
```

```python
    C_plus  = black_scholes(S+h, K, T, r, sigma, 'call')
    C_minus = black_scholes(S-h, K, T, r, sigma, 'call')
    delta_num = (C_plus - C_minus) / (2*h)

    # Gamma
    C_mid = black_scholes(S, K, T, r, sigma, 'call')
    gamma_num = (C_plus - 2*C_mid + C_minus) / (h**2)

    # Vega
    C_plus_vol  = black_scholes(S, K, T, r, sigma+h, 'call')
    C_minus_vol = black_scholes(S, K, T, r, sigma-h, 'call')
    vega_num = (C_plus_vol - C_minus_vol) / (2*h)

    return delta_num, gamma_num, vega_num
```

In [ ]:
```python
row = df_newton.iloc[0]

S = row['S']
K = row['Strike']
T = row['T']
r = row['r']
sigma = row['IV_Newton']

delta_bs, gamma_bs, vega_bs = bs_greeks_call(S, K, T, r, sigma)
delta_num, gamma_num, vega_num = numerical_greeks(S, K, T, r, sigma)
```

In [218…
```python
import pandas as pd

greeks_table = pd.DataFrame({
    'Method': ['Analytical', 'Numerical'],
    'Delta': [delta_bs, delta_num],
    'Gamma': [gamma_bs, gamma_num],
    'Vega':  [vega_bs, vega_num]
})

print(greeks_table)
```

```
       Method    Delta     Gamma     Vega
0  Analytical  0.94399  0.003118  6.96814
1   Numerical  0.94399  0.003121  6.96814
```

In [ ]:
```python
results = []

for _, row in df_newton.iterrows():

    if row['Type'] != 'call':
        continue

    S = row['S']
    K = row['Strike']
    T = row['T']
    r = row['r']
    sigma = row['IV_Newton']

    delta_a, gamma_a, vega_a = bs_greeks_call(S, K, T, r, sigma)
```

```
    delta_n, gamma_n, vega_n = numerical_greeks(S, K, T, r, sigma)

    results.append({
        'Symbol': row['Symbol'],
        'Strike': K,
        'T': T,
        'Delta_Analytical': delta_a,
        'Delta_Numerical': delta_n,
        'Delta_Diff': abs(delta_a - delta_n),
        'Gamma_Analytical': gamma_a,
        'Gamma_Numerical': gamma_n,
        'Gamma_Diff': abs(gamma_a - gamma_n),
        'Vega_Analytical': vega_a,
        'Vega_Numerical': vega_n,
        'Vega_Diff': abs(vega_a - vega_n)
    })

greeks_full_table = pd.DataFrame(results)
```

In [220…

```
print(greeks_full_table.head(20))
print(greeks_full_table)
print("\nAverage Absolute Differences:")
print(greeks_full_table[['Delta_Diff','Gamma_Diff','Vega_Diff']].mean())
```

```
   Symbol  Strike         T  Delta_Analytical  Delta_Numerical    Delta_Diff  \
0    TSLA   365.0  0.021918          0.943990         0.943990  2.179756e-11
1    TSLA   370.0  0.021918          0.932384         0.932384  2.683120e-11
2    TSLA   372.5  0.021918          0.927693         0.927693  8.063217e-11
3    TSLA   375.0  0.021918          0.918591         0.918591  4.576863e-10
4    TSLA   377.5  0.021918          0.913141         0.913141  3.743283e-10
5    TSLA   380.0  0.021918          0.902240         0.902240  8.994272e-11
6    TSLA   385.0  0.021918          0.885717         0.885717  4.838272e-10
7    TSLA   387.5  0.021918          0.867402         0.867402  2.418328e-10
8    TSLA   390.0  0.021918          0.852258         0.852258  1.793212e-10
9    TSLA   392.5  0.021918          0.832907         0.832907  4.911183e-11
10   TSLA   395.0  0.021918          0.814320         0.814320  4.078085e-10
11   TSLA   397.5  0.021918          0.790860         0.790860  1.736034e-11
12   TSLA   400.0  0.021918          0.767635         0.767635  1.242051e-10
13   TSLA   402.5  0.021918          0.738522         0.738522  1.372829e-10
14   TSLA   405.0  0.021918          0.706597         0.706597  1.004686e-10
15   TSLA   407.5  0.021918          0.671612         0.671612  1.895016e-10
16   TSLA   410.0  0.021918          0.634584         0.634584  3.771359e-10
17   TSLA   412.5  0.021918          0.594691         0.594691  2.022700e-10
18   TSLA   415.0  0.021918          0.552619         0.552619  3.128662e-10
19   TSLA   417.5  0.021918          0.509075         0.509075  1.535101e-10

    Gamma_Analytical  Gamma_Numerical    Gamma_Diff  Vega_Analytical  \
0           0.003118         0.003121  2.720156e-06         6.968140
1           0.003776         0.003797  2.071043e-05         8.071898
2           0.004114         0.004104  1.032171e-05         8.499527
3           0.004573         0.004582  8.746858e-06         9.301386
4           0.004991         0.004979  1.122156e-05         9.765009
5           0.005542         0.005548  5.540000e-06        10.658148
6           0.006706         0.006685  2.159123e-05        11.931813
7           0.007470         0.007481  1.066235e-05        13.241766
8           0.008254         0.008248  5.808854e-06        14.251376
9           0.009100         0.009095  4.867816e-06        15.452705
10          0.010024         0.010027  2.965017e-06        16.519345
11          0.010980         0.010965  1.441748e-05        17.752301
12          0.012050         0.012062  1.225158e-05        18.856336
13          0.013068         0.013063  5.090398e-06        20.086680
14          0.014070         0.014069  8.619134e-07        21.251064
15          0.015001         0.015007  5.185420e-06        22.317163
16          0.015913         0.015913  9.985163e-08        23.217655
17          0.016670         0.016666  3.570253e-06        23.935831
18          0.017235         0.017229  5.973085e-06        24.418437
19          0.017581         0.017579  2.080779e-06        24.626642

    Vega_Numerical     Vega_Diff
0         6.968140  4.914221e-08
1         8.071898  8.070518e-08
2         8.499527  9.632775e-08
3         9.301385  1.167718e-07
4         9.765009  1.353764e-07
5        10.658148  1.565051e-07
6        11.931813  2.039605e-07
7        13.241766  2.190548e-07
8        14.251376  2.350708e-07
9        15.452704  2.413361e-07
10       16.519344  2.474826e-07
```

```
11       17.752301  2.388640e-07
12       18.856336  2.303983e-07
13       20.086680  2.008979e-07
14       21.251064  1.638271e-07
15       22.317163  1.198856e-07
16       23.217655  7.653462e-08
17       23.935831  3.719780e-08
18       24.418437  1.000625e-08
19       24.626642  4.756480e-10
    Symbol  Strike         T  Delta_Analytical  Delta_Numerical    Delta_Diff  \
0     TSLA   365.0  0.021918          0.943990         0.943990  2.179756e-11
1     TSLA   370.0  0.021918          0.932384         0.932384  2.683120e-11
2     TSLA   372.5  0.021918          0.927693         0.927693  8.063217e-11
3     TSLA   375.0  0.021918          0.918591         0.918591  4.576863e-10
4     TSLA   377.5  0.021918          0.913141         0.913141  3.743283e-10
..     ...     ...       ...               ...              ...           ...
611    SPY   870.0  0.175342          0.001633         0.001633  1.210152e-12
612    SPY   875.0  0.175342          0.001602         0.001602  2.249422e-12
613    SPY   880.0  0.175342          0.001572         0.001572  2.610757e-11
614    SPY   895.0  0.175342          0.001231         0.001231  9.330980e-13
615    SPY   900.0  0.175342          0.000945         0.000945  5.102255e-12

     Gamma_Analytical  Gamma_Numerical    Gamma_Diff  Vega_Analytical  \
0            0.003118         0.003121  2.720156e-06         6.968140
1            0.003776         0.003797  2.071043e-05         8.071898
2            0.004114         0.004104  1.032171e-05         8.499527
3            0.004573         0.004582  8.746858e-06         9.301386
4            0.004991         0.004979  1.122156e-05         9.765009
..                ...              ...           ...              ...
611          0.000097         0.000097  3.454303e-07         1.504265
612          0.000093         0.000093  5.709469e-07         1.477763
613          0.000090         0.000090  4.778932e-08         1.452463
614          0.000069         0.000069  1.114716e-07         1.161955
615          0.000054         0.000054  5.736243e-08         0.912062

     Vega_Numerical     Vega_Diff
0          6.968140  4.914221e-08
1          8.071898  8.070518e-08
2          8.499527  9.632775e-08
3          9.301385  1.167718e-07
4          9.765009  1.353764e-07
..              ...           ...
611        1.504269  3.602386e-06
612        1.477766  3.429737e-06
613        1.452466  3.269839e-06
614        1.161958  2.734455e-06
615        0.912065  2.452499e-06

[616 rows x 12 columns]

Average Absolute Differences:
Delta_Diff    1.797183e-10
Gamma_Diff    5.485584e-06
Vega_Diff     2.494566e-06
dtype: float64
```

Problem 12

In [221…

```
merged = pd.merge(
    df_newton[['Symbol','Type','Strike','Expiry','T','IV_Newton']],
    DATA2[['Symbol','Type','Strike','Expiry','S','r','T','Price']],
    on=['Symbol','Type','Strike','Expiry'],
    suffixes=('_D1','_D2')
)


def reprice_option(row):

    S2 = row['S']
    K  = row['Strike']
    T2 = row['T_D2']
    r2 = row['r']
    sigma1 = row['IV_Newton']

    return black_scholes(S2, K, T2, r2, sigma1, row['Type'])


merged['BS_Reprice'] = merged.apply(reprice_option, axis=1)
```

In [222…

```
merged['Pricing_Error'] = merged['BS_Reprice'] - merged['Price']

print(merged[['Symbol','Type','Strike','BS_Reprice','Price','Pricing_Error']].head(
```

```
   Symbol  Type  Strike  BS_Reprice   Price  Pricing_Error
0    TSLA  call   365.0   53.325830  53.250       0.075830
1    TSLA  call   370.0   48.468122  48.425       0.043122
2    TSLA  call   372.5   46.010773  45.975       0.035773
3    TSLA  call   375.0   43.634492  43.625       0.009492
4    TSLA  call   377.5   41.178690  41.175       0.003690
5    TSLA  call   380.0   38.826053  38.850      -0.023947
6    TSLA  call   385.0   33.981208  34.025      -0.043792
7    TSLA  call   387.5   31.761260  31.850      -0.088740
8    TSLA  call   390.0   29.460793  29.575      -0.114207
9    TSLA  call   392.5   27.250276  27.400      -0.149724
10   TSLA  call   395.0   25.000859  25.175      -0.174141
11   TSLA  call   397.5   22.866182  23.075      -0.208818
12   TSLA  call   400.0   20.692930  20.925      -0.232070
13   TSLA  call   402.5   18.682968  18.950      -0.267032
14   TSLA  call   405.0   16.749470  17.050      -0.300530
15   TSLA  call   407.5   14.916989  15.250      -0.333011
16   TSLA  call   410.0   13.141213  13.500      -0.358787
17   TSLA  call   412.5   11.492730  11.875      -0.382270
18   TSLA  call   415.0    9.973351  10.375      -0.401649
19   TSLA  call   417.5    8.584416   9.000      -0.415584
```

Part 3

Part a

In [223…

```
import numpy as np

def swap_amounts_part_a(S_next, x_t, y_t, gamma):

    k = x_t * y_t
    P_t = y_t / x_t

    upper = P_t / (1 - gamma)
    lower = P_t * (1 - gamma)

    if S_next > upper:

        x_new = np.sqrt(k / (S_next * (1 - gamma)))
        y_new = k / x_new

        Delta_x = x_t - x_new
        Delta_y = (y_new - y_t) / (1 - gamma)

        return Delta_x, Delta_y

    elif S_next < lower:

        x_new = np.sqrt(k * (1 - gamma) / S_next)
        y_new = k / x_new

        Delta_x = (x_new - x_t) / (1 - gamma)
        Delta_y = y_t - y_new

        return Delta_x, Delta_y
    else:
        return 0.0, 0.0
```

In [224…

```
x0 = 1000
y0 = 1000
gamma = 0.003

print(swap_amounts_part_a(1.05, x0, y0, gamma))
print(swap_amounts_part_a(0.95, x0, y0, gamma))
print(swap_amounts_part_a(1.0, x0, y0, gamma))
y0 = 1000
gamma = 0.003

print(swap_amounts_part_a(1.1, x0, y0, gamma))
print(swap_amounts_part_a(0.9, x0, y0, gamma))
print(swap_amounts_part_a(1.0, x0, y0, gamma))
```

```
(22.632775023467843, 23.226559144115228)
(24.511763888454166, 23.855248578787155)
(0.0, 0.0)
(45.10399086828511, 47.37658296364267)
(52.668231617983444, 49.89046717069539)
(0.0, 0.0)
```

part b

In [225…

```python
import numpy as np

def lognormal_density(s, S0, sigma, dt):

    mu = np.log(S0) - 0.5 * sigma**2 * dt
    var = sigma**2 * dt

    return (1 / (s * np.sqrt(2*np.pi*var))) * \
           np.exp(-(np.log(s) - mu)**2 / (2*var))
```

In [226…

```python
def fee_revenue(s, x_t, y_t, gamma):

    Delta_x, Delta_y = swap_amounts_part_a(s, x_t, y_t, gamma)

    P_t = y_t / x_t
    upper = P_t / (1 - gamma)
    lower = P_t * (1 - gamma)

    # Case 1
    if s > upper:
        return gamma * Delta_y

    # Case 2
    elif s < lower:
        return gamma * Delta_x * s

    else:
        return 0.0
```

In [ ]:

```python
def expected_revenue(sigma, gamma):

    x_t = 1000
    y_t = 1000
    S0 = 1
    dt = 1/365

    s_min = 0.001
    s_max = 3.0
    N = 5000

    s_grid = np.linspace(s_min, s_max, N)

    integrand = []

    for s in s_grid:
        revenue = fee_revenue(s, x_t, y_t, gamma)
        density = lognormal_density(s, S0, sigma, dt)
        integrand.append(revenue * density)

    integrand = np.array(integrand)
```

```
    return np.trapz(integrand, s_grid)
```

In [228…

```
print(expected_revenue(0.6, 0.003))
```

```
0.03298335887507768
```

Part c

In [ ]:

```
import numpy as np
from scipy.stats import lognorm

def expected_fee_revenue(sigma, gamma, x0=1000, y0=1000):

    S0 = 1
    dt = 1/365
    k = x0 * y0
    P_t = y0 / x0

    # Integration grid for S
    s_grid = np.linspace(0.2, 3.0, 2000)

    # Lognormal density
    mu = np.log(S0) + (-0.5 * sigma**2) * dt
    vol = sigma * np.sqrt(dt)
    density = lognorm.pdf(s_grid, s=vol, scale=np.exp(mu))

    revenue = np.zeros_like(s_grid)

    for i, S_next in enumerate(s_grid):
        Delta_x, Delta_y = swap_amounts_part_a(S_next, x0, y0, gamma)

        upper = P_t / (1 - gamma)
        lower = P_t * (1 - gamma)

        if S_next > upper:
            revenue[i] = gamma * Delta_y
        elif S_next < lower:
            revenue[i] = gamma * Delta_x * S_next
        else:
            revenue[i] = 0

    integrand = revenue * density

    # Trapezoidal rule
    return np.trapz(integrand, s_grid)

sigmas_discrete = [0.2, 0.6, 1.0]
gammas_discrete = [0.001, 0.003, 0.01]

print("\nExpected Fee Revenue Table:")

for sigma in sigmas_discrete:
    print(f"\nσ = {sigma}")
    best_ER = -1
```

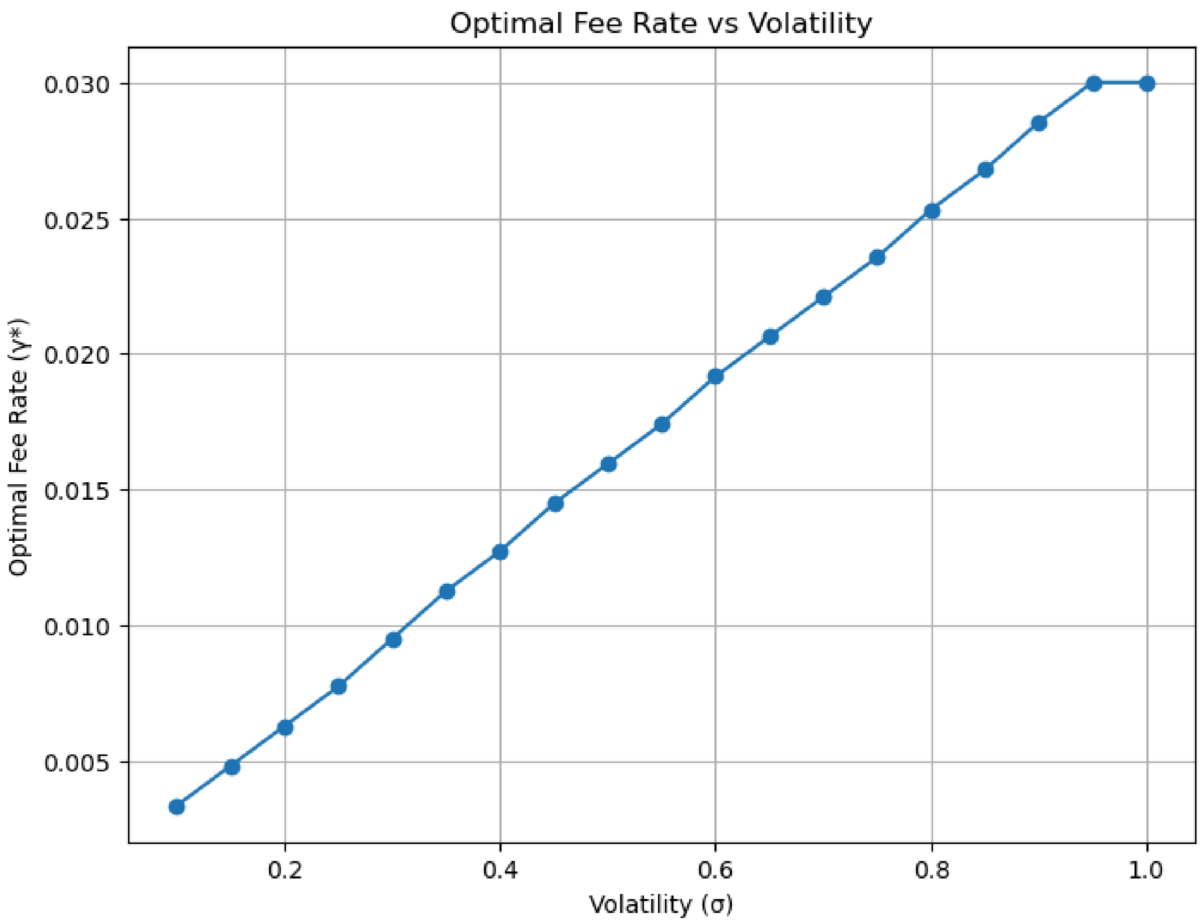
```
    best_gamma = None

    for gamma in gammas_discrete:
        ER = expected_fee_revenue(sigma, gamma)
        print(f"  γ = {gamma}:  E[R] = {ER:.6f}")

        if ER > best_ER:
            best_ER = ER
            best_gamma = gamma

    print(f"  → Optimal γ* = {best_gamma}")

sigma_grid = np.arange(0.1, 1.01, 0.01)
gamma_grid = np.linspace(0.001, 0.03, 80)

gamma_star_list = []

for sigma in sigma_grid:

    best_ER = -1
    best_gamma = None

    for gamma in gamma_grid:
        ER = expected_fee_revenue(sigma, gamma)

        if ER > best_ER:
            best_ER = ER
            best_gamma = gamma

    gamma_star_list.append(best_gamma)

gamma_star_array = np.array(gamma_star_list)
print("\nFirst 10 σ and corresponding γ*(σ):")
for i in range(10):
    print(f"σ = {sigma_grid[i]:.2f}, γ* = {gamma_star_array[i]:.5f}")
```

```
Expected Fee Revenue Table:

σ = 0.2
  γ = 0.001:  E[R] = 0.003686
  γ = 0.003:  E[R] = 0.008515
  γ = 0.01:  E[R] = 0.009418
  → Optimal γ* = 0.01

σ = 0.6
  γ = 0.001:  E[R] = 0.011924
  γ = 0.003:  E[R] = 0.032981
  γ = 0.01:  E[R] = 0.081076
  → Optimal γ* = 0.01

σ = 1.0
  γ = 0.001:  E[R] = 0.020061
  γ = 0.003:  E[R] = 0.057382
  γ = 0.01:  E[R] = 0.160686
  → Optimal γ* = 0.01

First 10 σ and corresponding γ*(σ):
σ = 0.10, γ* = 0.00320
σ = 0.11, γ* = 0.00357
σ = 0.12, γ* = 0.00357
σ = 0.13, γ* = 0.00430
σ = 0.14, γ* = 0.00467
σ = 0.15, γ* = 0.00467
σ = 0.16, γ* = 0.00504
σ = 0.17, γ* = 0.00541
σ = 0.18, γ* = 0.00577
σ = 0.19, γ* = 0.00614
```

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

sigma_grid = np.linspace(0.1, 1.0, 19)

gamma_grid = np.linspace(0.001, 0.03, 100)

gamma_star_list = []

for sigma in sigma_grid:

    revenues = []

    for gamma in gamma_grid:
        ER = expected_fee_revenue(sigma, gamma)
        revenues.append(ER)

    revenues = np.array(revenues)

    gamma_star = gamma_grid[np.argmax(revenues)]

    gamma_star_list.append(gamma_star)
```

```
gamma_star_array = np.array(gamma_star_list)

plt.figure(figsize=(8,6))
plt.plot(sigma_grid, gamma_star_array, marker='o')
plt.xlabel("Volatility (σ)")
plt.ylabel("Optimal Fee Rate (γ*)")
plt.title("Optimal Fee Rate vs Volatility")
plt.grid(True)
plt.show()
```




```
In [ ]:
```