Implied Volatility BoA

September 11, 2024

University of California Los Angeles
Master of Quantitative Economics -MQE-
ECON 451 - Financial Institutions and Monetary Policy

Nikolaos Papadatos

Homework 2 - Flow of Funds

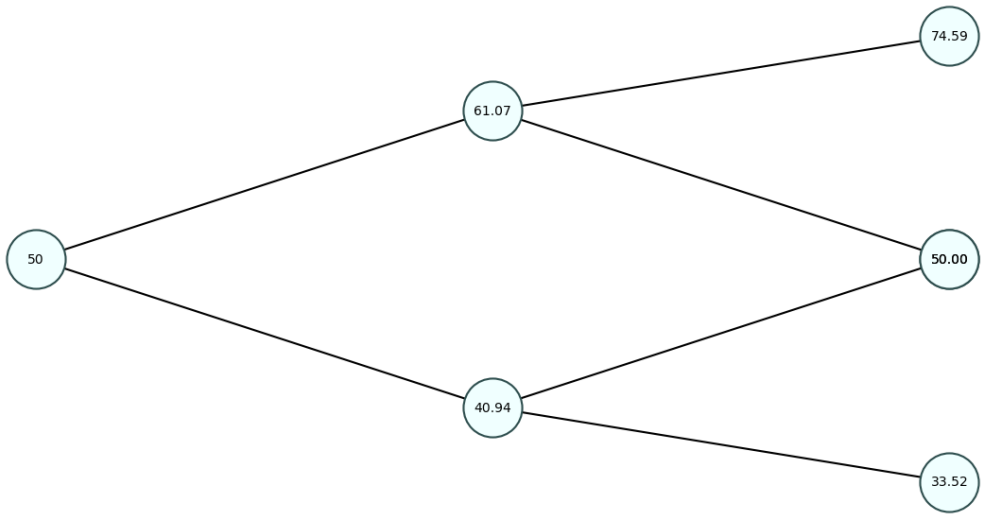
[ ]:
```
# import excel file
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf
import scipy.stats as stats
from scipy.stats import norm
from scipy.optimize import newton
from datetime import datetime
import seaborn as sns
import networkx as nx
from statistics import NormalDist
```

0.1 Theoretical Framework

The below binomial tree represents the asset price of an underlying 2-year European style put option. We compute the value of the put option using the replicating portfolio method with the following parameters.

[ ]:
```
# Create an empty graph
plt.figure(figsize=(12, 6))
G = nx.Graph()

# Add nodes with labels
G.add_node(0, label='50')
G.add_node(1, label='61.07')
G.add_node(2, label='40.94')
G.add_node(3, label='74.59')
G.add_node(4, label='50.00')
G.add_node(5, label='50.00')
G.add_node(6, label='33.52')
```

```
# Add edges
G.add_edge(0, 1)
G.add_edge(0, 2)
G.add_edge(1, 3)
G.add_edge(1, 4)
G.add_edge(2, 5)
G.add_edge(2, 6)

# Set positions for the nodes
pos = {0: (0, 0), 1: (1, 1), 2: (1, -1), 3: (2, 1.5), 4: (2, 0), 5: (2, 0), 6:␣
 ↪(2, -1.5)}

# Draw the graph with labels
nx.draw_networkx_nodes(G, pos, node_size=2000, node_color='azure',␣
 ↪edgecolors='darkslategrey', linewidths=1.5)
nx.draw_networkx_edges(G, pos, width=1.5)
labels = nx.get_node_attributes(G, 'label')
nx.draw_networkx_labels(G, pos, labels, font_size=10)

# Remove axis

plt.axis('off')

# Show the plot
plt.tight_layout()
plt.title('Binary Tree for underlying stock price at time 0', fontsize = 18,␣
 ↪y=1.05)
plt.show()
```



The value for the put option at $t = 2$ is - European asset price: 74.59; put value: 0 (do not exercise) - European asset price: 50.00; put value: $55 - 50 = 5$ - European asset price: 50.00; put value: $55 - 50 = 5$ - European asset price: 33.52; put value: $55 - 33.52 = 21.48$

The value for the put option at $t = 1$ is - European asset price: 61.07; put value: 2.027 - European asset price: 40.94; put value: 11.44

$$5 = \Delta(50) + B(1.05); 0 = \Delta(74.59) + B(1.05)$$
$$P_{t=1} = -0.203(61.07) + 14.44$$
$$P_{t=1} = 2.03$$

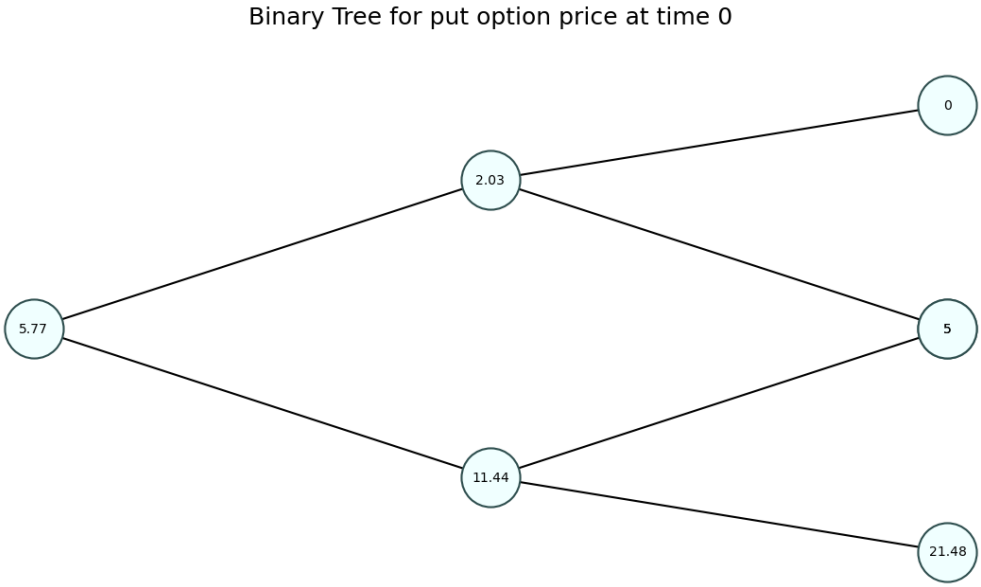
$$21.48 = \Delta(33.52) + B(1.05); 5 = \Delta(50) + B(1.05)$$
$$P_{t=1} = -1(40.94) + 52.38$$
$$P_{t=1} = 11.44$$

The value for the put option at $t = 0$ is - European asset price: 61.07; put value: 5.77 -

$$11.44 = \Delta(40.94) + B(1.05); 2.03 = \Delta(61.07) + B(1.05)$$
$$P_{t=0} = -0.467(50) + 29.13$$
$$P_{t=0} = 5.77$$

[ ]:
```
# Create an empty graph
plt.figure(figsize=(12, 6))
G = nx.Graph()

# Add nodes with labels
G.add_node(0, label='5.77')
G.add_node(1, label='2.03')
G.add_node(2, label='11.44')
G.add_node(3, label='0')
G.add_node(4, label='5')
G.add_node(5, label='5')
G.add_node(6, label='21.48')

# Add edges
G.add_edge(0, 1)
G.add_edge(0, 2)
G.add_edge(1, 3)
G.add_edge(1, 4)
G.add_edge(2, 5)
G.add_edge(2, 6)
```

```
# Set positions for the nodes
pos = {0: (0, 0), 1: (1, 1), 2: (1, -1), 3: (2, 1.5), 4: (2, 0), 5: (2, 0), 6:␣
 ↪(2, -1.5)}

# Draw the graph with labels
nx.draw_networkx_nodes(G, pos, node_size=2000, node_color='azure',␣
 ↪edgecolors='darkslategrey', linewidths=1.5)
nx.draw_networkx_edges(G, pos, width=1.5)
labels = nx.get_node_attributes(G, 'label')
nx.draw_networkx_labels(G, pos, labels, font_size=10)

# Remove axis

plt.axis('off')

# Show the plot
plt.tight_layout()
plt.title('Binary Tree for put option price at time 0', fontsize = 18, y=1.05)
plt.show()
```



1 Part 1 - Black & Scholes

For this analysis, we have selected Bank of America. We downloaded quarterly call reports starting from Q1 2021 through Q4 2023.

1.1 Total asset and (face) debt positions for the last three years

[ ]:
```
data = pd.read_csv ("BofA.csv")
data
```

[ ]:

	Quarter	Total Assets - item 12	Total Liabilities - item 21
0	Q1 2021	2,316,773,000	2,096,341,000
1	Q2 2021	2,350,294,000	2,124,998,000
2	Q3 2021	2,400,819,000	2,172,437,000
3	Q4 2021	2,519,525,000	2,283,098,000
4	Q1 2022	2,513,619,000	2,284,227,000
5	Q2 2022	2,440,022,000	2,209,100,000
6	Q3 2022	2,407,902,000	2,178,965,000
7	Q4 2022	2,418,508,000	2,193,059,000
8	Q1 2023	2,518,290,000	2,285,819,000
9	Q2 2023	2,449,804,000	2,219,795,000
10	Q3 2023	2,465,234,000	2,236,225,000
11	Q4 2023	2,540,116,000	2,303,910,000

In the quarterly call report, there is a Balance Sheet (Form Type - 031) on page 19, reported in thousands of dollars. For total assets, we refer to Item 12, which is **"Total assets (sum of items 1 through 11)."** This includes:

1. Cash and balances due from depository institutions
2. Securities
3. Federal funds sold and securities purchased under agreements to resell
4. Loans and lease financing receivables
5. Trading assets
6. Premises and fixed assets
7. Other real estate owned
8. Investments in unconsolidated subsidiaries and associated companies
9. Direct and indirect investments in real estate ventures
10. Intangible assets
11. Other assets

For the **debt components**, we looked added Item 21 , **Total Liabilities (sum of the items 13 through 20)**

13. Deposits:
    a. In domestic offices - Noninterest-bearing and Interest-bearing
    b. In foreign offices - Noninterest-bearing and Interest-bearing
14. Federal funds purchased and securities sold under agreements to repurchase:
    a. Federal funds purchased in domestic b.Securities sold under agreements to repurchase10
15. Trading liabilities
16. Other borrowed money (includes mortgage indebtedness and obligations under capitalized leases)
17. Subordinated notes and debentures1
18. Other liabilities

1.2 Variance in the firm's asset value

[ ]:
```
data_q4 = data.loc[[3, 7, 11]]
data_q4
```

[ ]:

	Quarter	Total Assets - item 12	Total Liabilities - item 21
3	Q4 2021	2,519,525,000	2,283,098,000
7	Q4 2022	2,418,508,000	2,193,059,000
11	Q4 2023	2,540,116,000	2,303,910,000

[ ]:
```
# need to remove commas from the numbers and convert to integers
data_q4['Total Assets  - item 12'] = data_q4['Total Assets  - item 12'].
 ↪replace(',', '', regex=True).astype('Int64')
# calculating the variance of total assets
variance_q4 = data_q4['Total Assets  - item 12'].var()
print(f'The variance of total assets is {variance_q4}')
```

```
The variance of total assets is 4236154872333333.5
```

[ ]:
```
deviation = np.sqrt(4236154872333333.5)
print(f'The standard deviation of total assets is {deviation}')
```

```
The standard deviation of total assets is 65085750.14804188
```

1.3 Average 10 year Treasury bond rate over the last three years

[ ]:
```
bond_rate = pd.read_excel("DGS10.xlsx")
bond_rate.head()
```

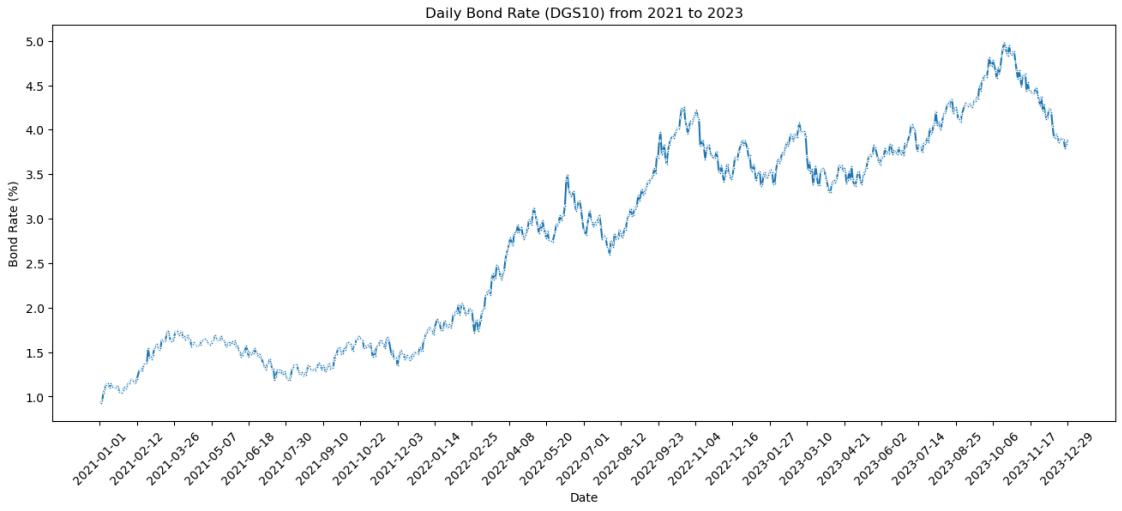
[ ]:

	DATE	DGS10
0	2019-05-02	2.55
1	2019-05-03	2.54
2	2019-05-06	2.51
3	2019-05-07	2.45
4	2019-05-08	2.49

[ ]:
```
# Filter the data for the 2021-2023 period
data_filtered = bond_rate[(bond_rate['DATE'] >= '2021-01-01') &␣
 ↪(bond_rate['DATE'] <= '2023-12-31')]
data_filtered['DATE'] = data_filtered['DATE'].astype(str)
data_filtered['DGS10'] = pd.to_numeric(data_filtered['DGS10'], errors='coerce')

# Create the plot
plt.figure(figsize=(16, 6))
sns.lineplot(x='DATE', y='DGS10', data=data_filtered, marker='o', markersize=2)

# Set plot title and labels
plt.title('Daily Bond Rate (DGS10) from 2021 to 2023')
plt.xlabel('Date')
```

```
plt.ylabel('Bond Rate (%)')

# Customize x-axis tick marks
n = 30  # Show ticks for every 30th date
plt.xticks(data_filtered['DATE'][::n], rotation=45)

# Show the plot
plt.show()
```

```
C:\Users\Alejandro\AppData\Local\Temp\ipykernel_2520\1322218207.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_filtered['DATE'] = data_filtered['DATE'].astype(str)
C:\Users\Alejandro\AppData\Local\Temp\ipykernel_2520\1322218207.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_filtered['DGS10'] = pd.to_numeric(data_filtered['DGS10'],
errors='coerce')
```



[ ]:
```
average_rate = data_filtered['DGS10'].mean()
print(f'The average bond rate from 2021 to 2023 is {average_rate.round(4)}')
```

```
The average bond rate from 2021 to 2023 is 2.7835
```

We use the values computed above as inputs into the Black-Scholes model to value equity as a call option. For this study, we assume that the debt is a zero-coupon bond with horizon 10 years.

[ ]:
```
# Need to remove commas from the numbers and convert to integers
data_q4['Total Liabilities - item 21'] = data_q4['Total Liabilities - item 21'].
 ↪replace(',', '', regex=True).astype(np.int64)

data_q4['Total Liabilities - item 21']
```

[ ]:
```
3     2283098000
7     2193059000
11    2303910000
Name: Total Liabilities - item 21, dtype: int64
```

[ ]:
```
# Total Assets for Q4 2023
print('Total Assets for Q4 2023:', data_q4['Total Assets  - item 12'][11])

# Total Debts for Q4 2023
print('Total Debts for Q4 2023:', data_q4['Total Liabilities - item 21'][11])

# Total Equity for Q4 2023
print('Total Equity for Q4 2023:', data_q4['Total Assets  - item 12'][11]-␣
 ↪data_q4['Total Liabilities - item 21'][11])
```

```
Total Assets for Q4 2023: 2540116000
Total Debts for Q4 2023: 2303910000
Total Equity for Q4 2023: 236206000
```

[ ]:
```
# Standard Deviation
print('Standard Deviation is:', round(np.sqrt(variance_q4), 6))
print('Standard Deviation as a share of Total Assets:', round(np.
 ↪sqrt(variance_q4), 3) / data_q4['Total Assets  - item 12'][11])
```

```
Standard Deviation is: 65085750.148042
Standard Deviation as a share of Total Assets: 0.025623140891203395
```

[ ]:
```
# Double Check Variance Calculation

# Mean for variance calc
mv = np.mean(data_q4['Total Assets  - item 12'])
#mv

# Manual check
a = abs((data_q4['Total Assets  - item 12'][3]) - mv)**2
b = abs((data_q4['Total Assets  - item 12'][7]) - mv)**2
c = abs((data_q4['Total Assets  - item 12'][11]) - mv)**2

round(np.sqrt((a+b+c)/(3-1)), 3)
```

[ ]:
```
65085750.148
```

1.4 Black-Scholes Model

[ ]:
```
# Define the Black-Scholes Model

mu = np.mean(data_q4['Total Assets  - item 12'])
sigs = round(np.sqrt(variance_q4), 6) # Adjust for variance

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

    if option == 'call':
        return (S * stats.norm.cdf(d1, scale = sigs )) - (K * np.exp(-r * T) *␣
 ↪stats.norm.cdf(d2, scale = sigs ))
    if option == 'put':
        return (K * np.exp(-r * T) * stats.norm.cdf(-d2, scale = sigs)) - (S *␣
 ↪stats.norm.cdf(-d1, scale = sigs))

# Alternative way to calculate d2
# d2 = (np.log(S / K) + ((r - (0.5 * (sigma ** 2))) * T)) / (sigma * np.sqrt(T))
```

[ ]:
```
# Set parameters
S = (data_q4['Total Assets  - item 12'][11])
K = (data_q4['Total Liabilities - item 21'][11])
T = 10
r = 0.0278        # 10 year T-bond average interest rate
sigma = round((np.sqrt(variance_q4)), 6)

# C = call option
# S = Current stock
# K = Strike price
# r = Risk-free interest rate
# t = time to maturity
# N = normal distribution
```

To calculate Black-Scholes, we must make a number of assumptions. Firstly, we must make a series of assumption standard for any calculation using the Black-Scholes model:

- No dividends are paid out during the life of the option
- Markets are random
- No transaction costs
- The risk-free rate and volatility are known and constant
- Returns are normally distributed.
- Can only be exercised at expiration

Secondly, we must make a series of assumptions which are unique to own asset and its valuation:

- The current stock is priced the same as the value of the Total Assets - The strike price is priced the same as the value of the Total Liabilities - the risk-free interest rate is equivalent to 10 year T-bonds over the last few years

[ ]:
```
# Call and Put Price for Black Scholes

call_price = black_scholes(S, K, T, r, sigma, option='call')
put_price = black_scholes(S, K, T, r, sigma, option='put')

print(f"Call price: {call_price:.2f}")
print(f"Put price: {put_price:.2f}")
```

```
Call price: 2296208291.70
Put price: 1500836011.75
```

We can also look at Black-Scholes for a current value of BofA stock option.

[ ]:
```
# Define the Black-Scholes Model

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

    if option == 'call':
        return (S * stats.norm.cdf(d1)) - (K * np.exp(-r * T) * stats.norm.
 ↪cdf(d2))
    if option == 'put':
        return (K * np.exp(-r * T) * stats.norm.cdf(-d2)) - (S * stats.norm.
 ↪cdf(-d1))

# Alternative way to calculate d2
# d2 = (np.log(S / K) + ((r - (0.5 * (sigma ** 2))) * T)) / (sigma * np.sqrt(T))
```

[ ]:
```
# As May 5th, a June 20th, 2025 call option for BofA from Yahoo finance␣
 ↪(highest volume option)

S = 37.25
K = 50
T = 1.125              # 1 year and half a month maturity
r = 0.05               # Assume similar to 1-year treasure
sigma = 0.2564


call_priceY = black_scholesY(S, K, T, r, sigma, option='call')
put_priceY = black_scholesY(S, K, T, r, sigma, option='put')

print(f"Call price: {call_priceY:.2f}")
```

```
print(f"Put price: {put_priceY:.2f}")
```

```
Call price: 1.19
Put price: 11.21
```

As of May 5th, the last listed price was 0.79 (assuming 1 year treasury rate) indicating that the asset is valued less than Black-Scholes would estimate.

The average 10-year Treasury bond rate over the last three years from 2021 to 2023 is approximately 2.78% according to FRED (https://fred.stlouisfed.org/series/RIFLGFCY10NA)

2 Part 2: Implied Volatility

2.1 Yahoo Finance

From Yahoo Finance we select the prices of call options corresponding to expiration dates May 31, July 19, September 20 and December 20, all of 2024. The data was downloaded March 30 of 2024

[ ]:
```
# Import excel file
df = pd.read_excel('data_BAC_call.xlsx')
df.head(3)
```

[ ]:

	Contract	Expiration_date	Strike	Last Price	Bid	Ask	\
0	BAC240531C00026000	2024-05-31	26	10.8	10.55	12.75	
1	BAC240531C00029000	2024-05-31	29	8.1	8.55	8.85	
2	BAC240531C00030000	2024-05-31	30	8.1	5.70	7.90	

	Change	% Change	Volume	Open Interest	Implied Volatility
0	0.0	0.0	20	8	0.7012
1	0.0	0.0	8	6	0.5684
2	0.0	0.0	1	2	0.6309

2.2 Implied Volatility

In order to compare the implied volatility calculated by Yahoo Finance with that calculated using the Black-Scholes formula, we follow a procedure that takes into account the number of days to maturity.*

[ ]:
```
# Current price of the underlying asset
current_price = yf.download('BAC', start='2024-04-29')["Adj Close"][0]
# Get the risk free rate using 3 month treasury bills
risk_free_rate = yf.download('^IRX', start='2024-04-29')["Adj Close"][0] / 100
```

```
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
```

We estimate the time to expiration as a fraction of a year. We then execute a function to estimate the implied volatility considering the Black-Scholes formula.*

[ ]:
```
# Add additional columns
df['Current Price'] = current_price
df['Risk Free Rate'] = risk_free_rate

# Convert to datetime
today = pd.to_datetime('today').normalize()
df['Expiration_date'] = pd.to_datetime(df['Expiration_date'])

# Calculate time to expiration
df['Time to Expiration'] = (df['Expiration_date'] - today) / pd.
 ↪Timedelta(days=365)
```

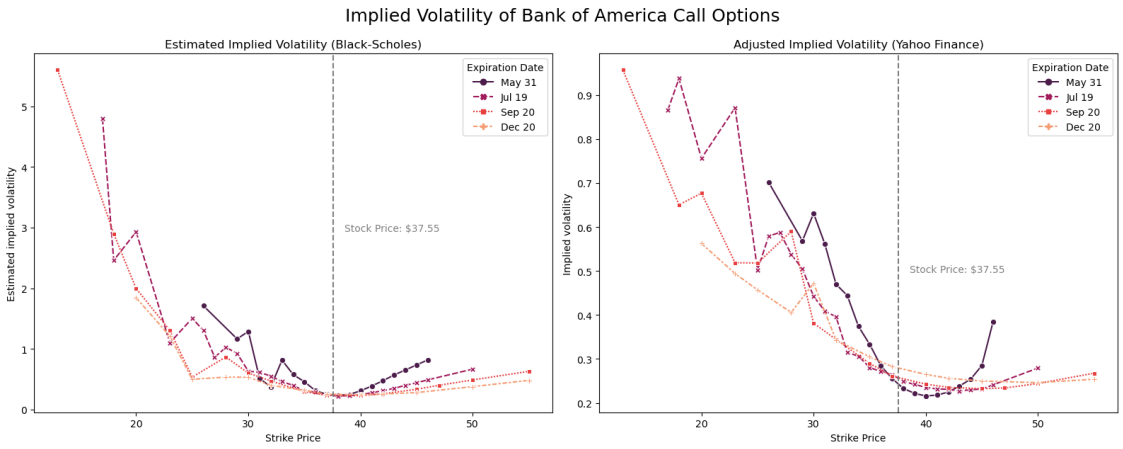
[ ]:
```
# Function to calculate implied volatility using the Black-Scholes formula and␣
 ↪the Newton-Raphson method

def implied_volatility(C, S, K, r, t):
    # Initialize implied volatility (you can adjust this)
    sigma = 0.2

    # Maximum number of iterations for the Newton-Raphson method
    max_iterations = 100
    tolerance = 1e-6

    for _ in range(max_iterations):
        d1 = (1 / (S * np.sqrt(t))) * (np.log(S / K) + (r + 0.5 * sigma**2) * t)
        d2 = d1 - sigma * np.sqrt(t)

        # Calculate the call option price using the Black-Scholes formula
        call_price = S * stats.norm.cdf(d1) - K * np.exp(-r * t) * stats.norm.
 ↪cdf(d2)

        # Calculate the Vega (sensitivity to volatility)
        vega = S * np.sqrt(t) * stats.norm.pdf(d1)

        # Update implied volatility using the Newton-Raphson method
        sigma -= (call_price - C) / vega

        # Check for convergence
        if abs(call_price - C) < tolerance:
            break

    return sigma
```

[ ]:
```
# Estimate imply volatility
df['Estimated Implied Volatility'] = df.apply(lambda row: implied_volatility(
    row['Last Price'],
    row['Current Price'],
```

```
    row['Strike'],
    row['Risk Free Rate'],
    row['Time to Expiration']
), axis=1)
```

[ ]:
```
# Function to generate plot
def generate_plot(data, y_variable, title):
    sns.lineplot(x='Strike', y=y_variable,
                 hue='Expiration_date', data=data, palette=custom_palette,
                 markers=True, style='Expiration_date',
                 legend='full')  # Show full legend
    # Add title and labels
    plt.title(title)
    plt.xlabel('Strike Price')
    plt.ylabel(f'{y_variable.capitalize()}')
    # Add vertical line at specific value of X (representing stock price)
    plt.axvline(x=current_price, color='grey', linestyle='--')  # Vertical line
    plt.text(current_price + 1, plt.gca().get_ylim()[1] * 0.5, f'Stock Price:␣
 ↪${current_price.round(2)}', color='grey')  # Text annotation

# Plot
# Specify color palette
custom_palette = sns.color_palette("rocket", 4)  # You can choose any palette,␣
 ↪"husl" is just an example

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# First subplot
plt.subplot(1, 2, 1)
generate_plot(df, 'Estimated Implied Volatility', 'Estimated Implied Volatility␣
 ↪(Black-Scholes)')

# Customizing legend labels
handles, labels = plt.gca().get_legend_handles_labels()
labels = ['May 31', 'Jul 19', 'Sep 20', 'Dec 20']  # Custom legend labels
plt.legend(handles, labels, title='Expiration Date')

# Second subplot
plt.subplot(1, 2, 2)
generate_plot(df, 'Implied Volatility', 'Adjusted Implied Volatility (Yahoo␣
 ↪Finance)')

# Adjust legend position
#plt.legend(loc='upper left')
# Customizing legend labels
handles, labels = plt.gca().get_legend_handles_labels()
```

```
labels = ['May 31', 'Jul 19', 'Sep 20', 'Dec 20']  # Custom legend labels
plt.legend(handles, labels, title='Expiration Date')
# Show plot
plt.tight_layout()
plt.suptitle('Implied Volatility of Bank of America Call Options', fontsize =␣
 ↪18, y=1.05)
plt.show()
```



An observable trend arises in the implied volatility of call options for Bank of America. Both estimates, whether calculated using the Black-Scholes formula or adjusted by Yahoo Finance, show increased volatility when the strike price is far from the current price of the asset. As the strike price gets closer to the current price, the volatility decreases. Moreover, the implied volatility tends to be elevated when the strike price is substantially lower than the current price. Additionally, implied volatility decreases as the time to expiration extends. In both graphs, the curve for May 31 is consistently higher than the curve for December 20.*