Submission Number: 2

Group Number: 40 (earlier Group 4)

(I was initially a part of group 4 but requested to move a separate group. Since I was informed of my new group's number by Student Support only at the last minute before submission deadline (ref. attached e-mail communication), I have done my analysis using 4 as the input. I kindly request you to accept my work.)

Group Members:

Full Legal Name	Location (Country)	E-Mail Address	Non-Contributing Member (X)
Ishaan Narula	India	ishaan.narula@outlook.com	

Statement of integrity: By typing the names of all group members in the text box below, you confirm that the assignment submitted is original work produced by the group (*excluding any non-contributing members identified with an "X" above*).

Ishaan Narula

Use the box below to explain any attempts to reach out to a non-contributing member. Type (N/A) if all members contributed.

I decided to attempt this submission individually, due to differences with group members. There was some contribution when we were one group. So, I did not add their names to non-contributing members. But, not much effort went into those contributions.

** Note, you may be required to provide proof of your outreach to non-contributing members upon request.*

Review Note: This document only includes selected code snippets. For this reason, please review all answers in conjunction with WQU_MScFE 620_Project2_Group4_Code_vF.ipynb file.

Inputs/ Assumptions

The following inputs/ assumptions have been used in the analysis.

```
#Inputs

x0_val = 95 #Current Stock Price
x0_val_ans4 = 105 #Assumed Stock Price for Answer 4
k_val = 105 #Strike Price
t_val1 = 3 #Steps in the Pricing Process (No. of Steps per Year x Years to Maturity) (Ans 1 Part a)
t_val2 = 2 #Steps in the Pricing Process (No. of Steps per Year x Years to Maturity) (Ans 2 Part a)
r_val = 0 #Risk-free Rate per Price Step (Calculated as Risk-free Rate per year/No. of Steps per year)
r_val_ans4 = 0.01 #Assumed Risk-free Rate per Price Step for Answer 4
u_val = 1.14 #Up Movement per Step in Binomial Model
```

Answer 1

Part (a)

To price the European call option, a binomial tree is first constructed taking $X_0 = 95,\ N = 3$ and $u = 1.14$. u is assumed to be the up movement *per step* in the model.

Function:

```
def Asset_Px_Binomial_Tree(t, u, d, x0):
    tree_holder = np.zeros([t+1, t+1])
    freq = np.zeros(t+1)
    d = 1/u

    for i in range(t+1):
        for j in range(i+1):
            tree_holder[j,i]=x0*(d**j)*(u**(i-j))
        freq[i]= Decimal(nCr(t, i))

    tree_holder = pd.DataFrame(tree_holder.round(2))
    return (tree_holder, freq)
```

Execution Code:

```
stockpx_binomial_tree1, freq_stockpx1 = Asset_Px_Binomial_Tree(t_val1, u_val, (1/u_val), x0_val)
stockpx_binomial_tree1
```

Output:

0	1	2	3
95.0	108.30	123.46	140.75
	83.33	95.00	108.30
		73.10	83.33
			64.12

As a next step, a general function to value a European option is written, which is then used to price the European call option in question.

Function:

```
def European_Option_Price(x0, k, u, t, r, option_type):

    d = 1/u
    p = (1 - d)/(u - d)

    if option_type == "call":
        price = reduce(lambda a, y: a + max(x0*(u**y)*(d**(t-y)) - k, 0) * nCr(t, y)*(p**y)*((1-p)**(t-y))
                       * (1/((1+r)**t)), [0]+list(range(t+1)))
        return price

    elif option_type == "put":
        price = reduce(lambda a, y: a + max(k - x0*(u**y)*(d**(t-y)), 0) * nCr(t, y)*(p**y)*((1-p)**(t-y))
                       * (1/((1+r)**t)), [0]+list(range(t+1)))
        return price

    else:
        price = print("Input format error")
        return price
```

Execution Code:

```
call_price = European_Option_Price(x0_val, k_val, u_val, t_val1, r_val, "call")

print("The price of the European call option with the given parameters is", "{:.3f}".format(call_price))
```

Output:

```
The price of the European call option with the given parameters is 4.799
```

Part (b)

Based on the function to price European options and a function to extract terminal prices, we derive the call payoff $H(\omega)$ for each price path and the call option value $V_t^H(\omega)$ for each node.

Functions:

Output:

Call Payoff $H(\omega)$

```
def Terminal_Px_Extractor(binomial_tree):
    terminal_px = pd.DataFrame(np.vstack([binomial_tree.iloc[:,-1].values]).T,
                               columns = ['X(w)'])
    return terminal_px
```

```
def European_Optionval_Tree(stockpx_binomial_tree, k, u, t, r, option_type):

    european_optionval_tree = pd.DataFrame()

    if option_type == "call":
        for i in range(stockpx_binomial_tree.shape[0]):
            for j in range(stockpx_binomial_tree.shape[1]):
                if i <= j:
                    european_optionval_tree.at[i, j] = European_Option_Price(stockpx_binomial_tree.at[i, j],
                                                                             k, u, (t-j), r, option_type)
                else:
                    european_optionval_tree.at[i, j] = 0
        return european_optionval_tree.round(3)

    elif option_type == "put":
        for i in range(stockpx_binomial_tree.shape[0]):
            for j in range(stockpx_binomial_tree.shape[1]):
                if i <= j:
                    european_optionval_tree.at[i, j] = European_Option_Price(stockpx_binomial_tree.at[i, j],
                                                                             k, u, (t-j), r, option_type)
                else:
                    european_optionval_tree.at[i, j] = 0
        return european_optionval_tree.round(3)

    else:
        european_optionval_tree = print("Input format error")
        return european_optionval_tree.round(3)
```

Execution Code:

```
call_payoffs = Terminal_Px_Extractor(stockpx_binomial_tree1)
call_payoffs['H(w)'] = [max(i - k_val, 0) for i in call_payoffs['X(w)']]

print('H(w) (call) = Max(Terminal Price - Strike, 0)')
call_payoffs
```

```
call_optionval_tree = European_Optionval_Tree(stockpx_binomial_tree1, k_val, u_val, t_val1, r_val, "call")
call_optionval_tree
```

	X(w)	H(w)
0	140.75	35.75
1	108.30	3.30
2	83.33	0.00
3	64.12	0.00

```
H(w) (call) = Max(Terminal Price - Strike, 0)
```

Call Option Value $V_t^H(\omega)$ for Each Node

0	1	2	3
4.799	9.449	18.460	35.75
	0.720	1.542	3.30
		0.000	0.00
			0.00

Answer 2

Part (a)

To price the European put option, a binomial tree is now constructed taking $N = 2$. As required, the rest of the parameters remain unchanged.

Function: *Refer to the function(s) in Answer1 Part (a)*

Execution Code:

```
stockpx_binomial_tree2, freq_stockpx2 = Asset_Px_Binomial_Tree(t_val2, u_val, (1/u_val), x0_val)
stockpx_binomial_tree2
```

Output:

0	1	2
95.0	108.30	123.46
	83.33	95.00
		73.10

Subsequently, the `European_Option_Price` function is used to value the European put option with the given parameters.

Function: *Refer to the function(s) in Answer1 Part (a)*

Execution Code:

```
put_price = European_Option_Price(x0_val, k_val, u_val, t_val2, r_val, "put")

print("The price of the European put option with the given parameters is", "{:.3f}".format(put_price))
```

Output:

```
The price of the European put option with the given parameters is 14.031
```

Part (b)

We first derive the put payoff $H(\omega)$ for each price path and the option value $V_t^H(\omega)$ for each node, before creating tables.

Function: *Refer to the function(s) in Answer 1 Part (b)*

Execution Code:

```
put_payoffs = Terminal_Px_Extractor(stockpx_binomial_tree2)
put_payoffs['H(w)'] = [max(k_val - i, 0) for i in put_payoffs['X(w)']]

print('H(w) (put) = Max(Strike - Terminal Price, 0)')
put_payoffs
```

Output:

(Please turn over)

```
put_optionval_tree = European_Optionval_Tree(stockpx_binomial_tree2, k_val, u_val, t_val2, r_val, "put")
put_optionval_tree
```

Payoff $H(\omega)$

	X(w)	H(w)
0	123.46	0.0
1	95.00	10.0
2	73.10	31.9

Value $V_t^H(\omega)$

	0	1	2
	14.031	5.327	0.0
		21.670	10.0
			31.9

H(w) (put) = Max(Strike - Terminal Price, 0)

The above output is now used to populate 3 tables: one showing stock price evolution, $X_t(\omega)$ and the put option's payoff $H(\omega)$, another showing the values of the option, $V_t^H(\omega)$, and the final one showing the hedging strategy, φ_t^H.

Code:

```
stockpx_table = pd.DataFrame(columns = ['w', 'X0(w)', 'X1(w)', 'X2(w)', 'H(w) = Max(K - X2(w), 0)'])

stockpx_table.at[0, 'w'] = '(u,u)'
stockpx_table.at[1, 'w'] = '(u,d)'
stockpx_table.at[2, 'w'] = '(d,u)'
stockpx_table.at[3, 'w'] = '(d,d)'

#Stock Price at t = 0
for i in range(4):
    stockpx_table.at[i, 'X0(w)'] = stockpx_binomial_tree2.at[0,0]

#Stock Price at t = 1
for i in range(2):
    stockpx_table.at[i, 'X1(w)'] = stockpx_binomial_tree2.at[0,1]

for i in range(2,4):
    stockpx_table.at[i, 'X1(w)'] = stockpx_binomial_tree2.at[1,1]

#Stock Price at t = 2
stockpx_table.at[0, 'X2(w)'] = stockpx_binomial_tree2.at[0,2]
stockpx_table.at[1, 'X2(w)'] = stockpx_binomial_tree2.at[1,2]
stockpx_table.at[2, 'X2(w)'] = stockpx_binomial_tree2.at[1,2]
stockpx_table.at[3, 'X2(w)'] = stockpx_binomial_tree2.at[2,2]

#Put Option Pay-off H(w)
for i in range(4):
    stockpx_table.at[i, 'H(w) = Max(K - X2(w), 0)'] = max(k_val - stockpx_table.at[i, 'X2(w)'], 0)

#stockpx_table.index += 1
stockpx_table
```

Output:

Stock Prices $X_t(\omega)$ and Payoffs $H(\omega)$

	w	X0(w)	X1(w)	X2(w)	H(w) = Max(K - X2(w), 0)
0	(u,u)	95	108.3	123.46	0
1	(u,d)	95	108.3	95	10
2	(d,u)	95	83.33	95	10
3	(d,d)	95	83.33	73.1	31.9

```
putval_table = pd.DataFrame(columns = ['w', 'V0(w)', 'V1(w)', 'V2(w)'])

putval_table.at[0, 'w'] = '(u,u)'
putval_table.at[1, 'w'] = '(u,d)'
putval_table.at[2, 'w'] = '(d,u)'
putval_table.at[3, 'w'] = '(d,d)'

#Put Option Price at t = 0
for i in range(4):
    putval_table.at[i, 'V0(w)'] = put_optionval_tree.at[0,0].round(3)

#Put Option Price at t = 1
for i in range(2):
    putval_table.at[i, 'V1(w)'] = put_optionval_tree.at[0,1].round(3)

for i in range(2,4):
    putval_table.at[i, 'V1(w)'] = put_optionval_tree.at[1,1].round(3)

#Put Option Price at t = 2
putval_table.at[0, 'V2(w)'] = put_optionval_tree.at[0,2].round(3)
putval_table.at[1, 'V2(w)'] = put_optionval_tree.at[1,2].round(3)
putval_table.at[2, 'V2(w)'] = put_optionval_tree.at[1,2].round(3)
putval_table.at[3, 'V2(w)'] = put_optionval_tree.at[2,2].round(3)

putval_table
```

Put Option Values $V_t^H(\omega)$

	w	V0(w)	V1(w)	V2(w)
0	(u,u)	14.031	5.327	0
1	(u,d)	14.031	5.327	10
2	(d,u)	14.031	21.67	10
3	(d,d)	14.031	21.67	31.9

```
hedging_strat_table = pd.DataFrame(columns = ['w', 'Q1(w)', 'Q2(w)'])

hedging_strat_table.at[0, 'w'] = '(u,u)'
hedging_strat_table.at[1, 'w'] = '(u,d)'
hedging_strat_table.at[2, 'w'] = '(d,u)'
hedging_strat_table.at[3, 'w'] = '(d,d)'

for j in range(1, hedging_strat_table.shape[1]):
    for i in range(hedging_strat_table.shape[0]):
        hedging_strat_table.iloc[i, j] = ((putval_table.iloc[i, j+1] - putval_table.iloc[i, j])/
                                          (stockpx_table.iloc[i, j+1] - stockpx_table.iloc[i, j])).round(3

hedging_strat_table
```

Hedging Strategy φ_t^H

	w	Q1(w)	Q2(w)
0	(u,u)	-0.654	-0.351
1	(u,d)	-0.654	-0.351
2	(d,u)	-0.655	-1
3	(d,d)	-0.655	-1

See in conjunction with Answer 3 (f)

Answer 3

Parts (a)-(c)

To construct matrices A and b, we use the node $X_1(\{(d,u),\ (d,d)\})$ (circled below) for stock price values. We also take a bond with a face value of $B = 100$ and assume it does not change across states.

Code:

```
stockpx_binomial_tree2, freq_stockpx2 = Asset_Px_Binomial_Tree(t_val2, u_val, (1/u_val), x0_val)
stockpx_binomial_tree2
```

```
A = pd.DataFrame(columns = ['State','Xt(w)', 'B'])
A['State'] = ['u', 'd']
A.set_index('State', inplace = True)
A = A.fillna('-')
A
```

```
for i in range(A.shape[0]):
    A['Xt(w)'].iloc[i] = stockpx_binomial_tree2.at[i+1, 2]

for i in range(A.shape[0]):
    A['B'] = 100

A
```

```
b = pd.DataFrame(columns = ['State', 'Vt(w)'])
b['State'] = ['u', 'd']
b.set_index('State', inplace = True)

for i in range(b.shape[0]):
    b.iloc[i] = put_optionval_tree.at[i+1, 2]

b
```

Output:

Stock Price Binomial Tree

	0	1	2
	95.0	108.30	123.46
		83.33	95.00
			73.10

Matrix A

State	Xt(w)	B
u	95	100
d	73.1	100

(2x2 matrix)

Matrix b

State	Vt(w)
u	10
d	31.9

(2x1 matrix)

Parts (d)-(f)

The no-arbitrage equation can be solved as: $Ax = b \quad \Longrightarrow A^{-1}.Ax = A^{-1}.b \quad \Longrightarrow (A^{-1}A).x = A^{-1}.b. \quad \Longrightarrow x = A^{-1}.b$

Code:

```
from numpy import linalg
linalg.solve(A.to_numpy(dtype = 'float'), b.to_numpy(dtype = 'float'))
```

Output:

```
array([[-1. ],
       [ 1.05]])
Stock qty. = -1,Bond qty. = +1.05
```

The no. of units of stock yielded by the matrix equations (-1) clearly matches with the hedging strategy (circled above).

Answer 4

Part (a)

To illustrate the value of the option portfolio and the stock-bond portfolio under the 3 states, taking X_0 equal to strike price. $N = 2$ is also assumed for this analysis. The option portfolio is constructed by subtracting the put option values at each node (short) from the call option values at those nodes (long). The stock-bond portfolio is constructed by subtracting the PV of 105 FV bond at risk-free rate of 1% per price step (borrow/ short) from the stock's price at each node (long).

Code:

```
stockpx_binomial_tree3, freq_stockpx3 = Asset_Px_Binomial_Tree(t_val2, u_val, 1/u_val, x0_val_ans4)
stockpx_binomial_tree3
```

```
call_optionval_tree2 = European_Optionval_Tree(stockpx_binomial_tree3, k_val, u_val,
                                               t_val2, r_val_ans4, "call")
call_optionval_tree2
```

```
put_optionval_tree2 = European_Optionval_Tree(stockpx_binomial_tree3, k_val, u_val,
                                              t_val2, r_val_ans4, "put")
put_optionval_tree2
```

```
option_portfolio_val = call_optionval_tree2 - put_optionval_tree2
option_portfolio_val
```

```
stock_bond_portfolio_val = pd.DataFrame()

for j in range(stockpx_binomial_tree3.shape[1]):
    for i in range(stockpx_binomial_tree3.shape[0]):

        if i <=j:
            stock_bond_portfolio_val.at[i,j] = (stockpx_binomial_tree3.at[i,j]
            - k_val/((1+r_val_ans4)**(t_val2-j))).round(3)

        else:
            stock_bond_portfolio_val.at[i,j] = 0

stock_bond_portfolio_val
```

Output:

Stock Price Binomial Tree

	0	1	2
	105.0	119.70	136.46
		92.11	105.00
			80.79

Call Values

	0	1	2
	6.734	14.554	31.46
		0.002	0.00
			0.00

Put Values

	0	1	2
	6.734	0.000	0.00
		12.765	0.00
			24.21

```
portfolio_val_comparison_table = pd.DataFrame(columns = ['w', 'X2(w)', 'K' , '(C-P)2(w)', '(S-B)2(w)'])

portfolio_val_comparison_table.at[0, 'w'] = '(u,u)'
portfolio_val_comparison_table.at[1, 'w'] = '(u,d)'
portfolio_val_comparison_table.at[2, 'w'] = '(d,u)'
portfolio_val_comparison_table.at[3, 'w'] = '(d,d)'

#Terminal Stock Price
portfolio_val_comparison_table.at[0, 'X2(w)'] = stockpx_binomial_tree3.at[0,2].round(3)
portfolio_val_comparison_table.at[1, 'X2(w)'] = stockpx_binomial_tree3.at[1,2].round(3)
portfolio_val_comparison_table.at[2, 'X2(w)'] = stockpx_binomial_tree3.at[1,2].round(3)
portfolio_val_comparison_table.at[3, 'X2(w)'] = stockpx_binomial_tree3.at[2,2].round(3)

#Strike Price
for i in range(4):
    portfolio_val_comparison_table.at[i, 'K'] = k_val

#Option Portfolio Value at t = 2
portfolio_val_comparison_table.at[0, '(C-P)2(w)'] = option_portfolio_val.at[0,2].round(3)
portfolio_val_comparison_table.at[1, '(C-P)2(w)'] = option_portfolio_val.at[1,2].round(3)
portfolio_val_comparison_table.at[2, '(C-P)2(w)'] = option_portfolio_val.at[1,2].round(3)
portfolio_val_comparison_table.at[3, '(C-P)2(w)'] = option_portfolio_val.at[2,2].round(3)

#Option Portfolio Value at t = 2
portfolio_val_comparison_table.at[0, '(S-B)2(w)'] = stock_bond_portfolio_val.at[0,2].round(3)
portfolio_val_comparison_table.at[1, '(S-B)2(w)'] = stock_bond_portfolio_val.at[1,2].round(3)
portfolio_val_comparison_table.at[2, '(S-B)2(w)'] = stock_bond_portfolio_val.at[1,2].round(3)
portfolio_val_comparison_table.at[3, '(S-B)2(w)'] = stock_bond_portfolio_val.at[2,2].round(3)

portfolio_val_comparison_table
```

Option-Based Portfolio Values

0	1	2
0.0	14.554	31.46
	12.763	0.00
		-24.21

Stock-Bond Portfolio Values

0	1	2
2.069	15.74	31.46
	-11.85	0.00
		-24.21

Value of the Two Portfolios for Each State

	w	X2(w)	K	(C-P)2(w)	(S-B)2(w)
0	(u,u)	136.46	105	31.46	31.46
1	(u,d)	105	105	0	0
2	(d,u)	105	105	0	0
3	(d,d)	80.79	105	-24.21	-24.21

Stock Price > Strike

Stock Price = Strike

Stock Price < Strike

Part (b)

For this part, the same assumptions as the ones for Answers 1-3 have been used. I also assume 2 steps in the pricing process and a risk-free rate of 1% per price step. Looking at the difference between the Long Call-Short Put portfolio and the Long Stock-Short Bond portfolio, the put-call parity is only approximately satisfied. This is because we are in discrete time. Increasing the number of steps and approaching continuity at the limit will progressively make this relationship more accurate.

Code:

```
stockpx_binomial_tree2
```

```
call_optionval_tree3 = European_Optionval_Tree(stockpx_binomial_tree2, k_val, u_val, t_val2,
                                               r_val_ans4, "call")
call_optionval_tree3
```

```
put_optionval_tree = European_Optionval_Tree(stockpx_binomial_tree2, k_val, u_val, t_val2,
                                             r_val_ans4, "put")
put_optionval_tree
```

```
option_portfolio_val2 = call_optionval_tree3 - put_optionval_tree
option_portfolio_val2
```

```
stock_bond_portfolio_val2 = pd.DataFrame()

for j in range(stockpx_binomial_tree2.shape[1]):
    for i in range(stockpx_binomial_tree2.shape[0]):

        if i <=j:
            stock_bond_portfolio_val2.at[i,j] = stockpx_binomial_tree2.at[i,j]
            - k_val/((1+r_val_ans4)**(t_val2-j))

        else:
            stock_bond_portfolio_val2.at[i,j] = 0

stock_bond_portfolio_val2
```

```
diff = option_portfolio_val2 - stock_bond_portfolio_val2
diff.round(3)
```

Output:

Stock Price Binomial Tree

0	1	2
95.0	108.30	123.46
	83.33	95.00
		73.10

Call Values

0	1	2
3.952	8.542	18.46
	0.000	0.00
		0.00

Put Values

0	1	2
13.755	5.274	0.0
	21.455	10.0
		31.9

Option-Based Portfolio Values

0	1	2
-9.803	3.268	18.46
	-21.455	-10.00
		-31.90

Stock-Bond Portfolio Values

0	1	2
-7.931	4.34	18.46
	-20.63	-10.00
		-31.90

Difference Between Above 2 Portfolio Values

0	1	2
-1.872	-1.072	0.0
	-0.825	0.0
		0.0

Reference

Shreve, S. (2005). Stochastic Calculus for Finance I: The Binomial Asset Pricing Model. New York: Springer