Options Strategy Visualizer

https://www.markaliaga.com/projects/option-strategy-visualizer

Code explanation:

BLOCK 1

-First we have to import the libraries in Python that will allow us to work with arrays and calculus. In this case, the module used is **Numpy**.
For the graphs we have **Matplotlib** and for combinations we use **itertools.**

Once we have all the extra modules that we will use, we can start implementing the constant values that will be be used for all the program, in this case and for this model of the algorithm, we will put the next values as a constants:

Strike_list —-> List of the available strike price of our group of options

fixed_premium —> To simplify the operations and the example we implement a fix premium for calls and puts.

underlaying_prices = np.linspace() —> This function allows us to create an array to calculate the payoffs of the underlying asset, so we are able to graph when the options start making money and when the option is not worth it.

To understand better, watch this table:

say that: **np.linspace(90, 110, 5)**

Price is going from 90 to 110 in 5 steps, what will be the payoff value?

With:
Strike = 100
Premium = 5

Underlying price	Payoff	Payoff value
90	(underlaying_price - strike, 0) - premium	-5
...	...	...
...	...	...
105	(105 - 100) - 5	0
110	(110 - 100) - 5	5

BLOCK 2

Once we have all the parameters ready, we can start making our first function, as we know, we want all the available combinations that we can do with n numbers of options in different strike prices and premium (but for this example we make the premium constant).

Knowing that we can imagine that for now we need something to combine all, well thats exactly what we are going to do.

def generate_basic_options(): —> creating a function

options = [] —> list to save all the combinations with the parameters.

#this is the core of our function:

for opt_type, position, strike in product(['Call', 'Put'], ['Long', 'Short'], strike_list):

#with this function we take one element of every list where: a= operation type, b = investor position and s= strike price.
With all these values we create x combinations, with one strike price, we can make 4 different options (Call Long, Call Short, Put Long, Put Short).

options.append({   —--> here we create a dictionary to save all the single combinations with their info.

“type”: otp_type,
“position”: position,
“strike”: strike,
“premium”: fixed_premium
})

return options

#in this first block is where in a future steps we will be available to add diferents parameter for every combination.

BLOCK 3

def compute_option_payoff(option, prices):   —--> this function will calculate the different payoffs depending on the intrinsic behavior of the contract.

```
    strike = option['strike']
    premium = option['premium']

    if option['type'] == 'Call':
        intrinsic_value = np.maximum(prices - strike, 0)
    else:  # Put
        intrinsic_value = np.maximum(strike - prices, 0)
```

CALL	S_0 - K = C (payoff)
	
	S = 100
	K= 50
	
	C= 50
		In this case, the payoff of the call will be the price of the asset - the strike price. So K<S.

PUT	K - S = C (payoff)
	S = 100
	K =50
	C = -50
		In this case the payoff of the put will have value if K>S.

#then if we have all the payoffs and we supposed that we just bought that we have to reduce the premium cost to know the real value.

```
if option['position'] == 'Long':
        return intrinsic_value - premium
```

#and in the same case that before but we put a - sign of th intrinsic value

so lets say:

Put option:

FInal price = 105
K= 100
Premium = 5

this put does not have value, so who sells that put get the premium,
-0 + 5

```
    else:  # Short
        return -intrinsic_value + premium
```

BLOCK 5

#function ot make all the possible strategies given the numbers of options, possible combinations and the possible prices.

get two inputs for the function: Combination dictionary and possible prices.

```
def compute_strategy_payoff(combination, prices):
    total_payoff = np.zeros_like(prices) #accumulator array .
```

# with this loop we calculate the individual option for all the prices, giving a payoffs array.

```
    for option in combination:
        total_payoff += compute_option_payoff(option, prices)
```

#the total payoff is the net result of that specific combination, that will be very useful to graph the strategy.

```
    return total_payoff
```

BLOCK 5

#thats a simple function to make readable the description of the combination X so we will have something like:

"Long Call @100 + Short Put @95" each one will be the combination X.

```
def describe_combination(combination):
    return " + ".join([f"{opt['position']} {opt['type']} @{opt['strike']}" for opt in combination])
```

BLOCK 6

#thats a filter function to eliminate that values that eliminates by them self,
So let's say you have a Long position at 100 and for the same K = 100 you have a Short… that will be 0.

1 + (-1) = 0

So we have to eliminate this combinations.

```
def is_valid_combination(combination):
    seen = set()  —> empty set that will store the repetitive values

 for opt in combination:   —-> loop to iterates in every option contract in the combinations looking for two values: type and strike
        key = (opt['type'], opt['strike'])
        if key in seen:
            return False
```

SO:
If the key is found, it means you already added an option with the same type and strike (imagine you already had a Call @ 100 and now you are adding another Call @ 100). The function immediately returns False.

```
seen.add(key)
    return True
```

BLOCK 7. Main Menu

def strategy_viewer_menu(n_options):   —> function to start all the menu and the combinations, just have to provide the number of options you have.

```
    base_options = generate_basic_options()
    all_combinations = list(combinations(base_options, n_options))

    valid_combinations = [c for c in all_combinations if is_valid_combination(c)]
```

#We iterate over every combination possible and we check if is valid, once we check we take the len(amount of valid combinations) and we show it to the user with the output.

```
    print(f"\nTotal valid combinations: {len(valid_combinations)}\n")
```

#this part is just to show the information in a clear way with the table form and the function that we defined earlier .

```
    print(f"{'Index':<6} | Strategy Description")
    print("-" * 50)

    for i, combo in enumerate(valid_combinations):
        desc = describe_combination(combo)
        print(f"{i:<6} | {desc}")
```

#main loop, waiting for the input of the user, where “exit” is 0 and some index is 1.

```
    while True:
        try:
            selection = input("\nEnter the strategy index to view (or type 'exit'): ")
            if selection.lower() == 'exit':
                print("Exiting.")
                break
```

#if the user inserts a real index 0 or bigger than 0, then we check in the list for the combination in that position and we obtain the payoff and the description.

```
        idx = int(selection)
        if 0 <= idx < len(valid_combinations):
            combo = valid_combinations[idx]
            payoff = compute_strategy_payoff(combo, underlying_prices)
            desc = describe_combination(combo)
```

—------------------------- Graphs—-------------------------------

```
            plt.figure(figsize=(10, 5))
            plt.plot(underlying_prices, payoff, label=desc)
            plt.axhline(0, color='black', linestyle='--')
            plt.title(f"Strategy #{idx}: {desc}")
            plt.xlabel("Underlying Price at Expiration")
            plt.ylabel("Profit / Loss")
            plt.legend()
            plt.grid(True)
            plt.show()


        else:
            print("Index out of range.")
    except ValueError:
        print("Invalid input. Please enter a number or 'exit'.")
```

WORKFLOW:

*To view it in a better resolution, please check this url:
https://www.markaliaga.com/projects/option-strategy-visualizer