Custom Strategies & TP/SL functions:

Changes will need to be made to **Bot_Class.py**, **TradingStrats.py** and **app.py** in order to add a custom strategy or new TP/SL function.

TradingStrats.py:

Create a new function to house your strategy, it should have the following definition at the bare minimum:

def <Strategy Name>(Trade_Direction, Close, High, Low, SL, TP, TP_choice, SL_choice, index, …(Any indicators you need for the strategy)):

So for example if I wanted to make a simple ema crossover strategy with two EMA's, I would have this definition:

```
def ema_crossover(Trade_Direction, Close, High, Low, SL, TP, TP_choice, SL_choice, current_index, ema_short, ema_long):
```

Now the Trading logic:

If we wanted to go short when the ema_short crosses below the ema_long, see 1st crossover above.
We would look for a candle where the ema_short is below the ema_long and on the previous candle the ema_short was above the ema_long. We would enter a short here, by setting Trade_Direction to 0 (indicating a short).

Code:

```
def ema_crossover(Trade_Direction, Close, High, Low, SL, TP, TP_choice, SL_choice, current_index, ema_short, ema_long):

    if ema_short[current_index] < ema_long[current_index] and ema_short[current_index - 1] > ema_long[current_index - 1]:
        Trade_Direction = 0
```

Similarly for the long we would look for a candle where the ema_short is above the ema_long and on the previous candle the ema_short was below the ema_long.

So together this would give the Code:

```
def ema_crossover(Trade_Direction, Close, High, Low, SL, TP, TP_choice, SL_choice, current_index, ema_short, ema_long):

    if ema_short[current_index] < ema_long[current_index] and ema_short[current_index - 1] > ema_long[current_index - 1]:
        Trade_Direction = 0

    elif ema_short[current_index] > ema_long[current_index] and ema_short[current_index - 1] < ema_long[current_index - 1]:
        Trade_Direction = 1
```

Now setting up the TP and SL, simply add the following line of code to work with the TP and SL functions that come out of the box:

```
stop_loss_val, take_profit_val = SetSLTP(-99, -99, Close, High, Low, Trade_Direction,
                                         SL, TP, TP_choice, SL_choice, current_index)
```

The final product, we need to return the Trade_Direction, stop_loss_val and take_profit_val:

```
def ema_crossover(Trade_Direction, Close, High, Low, SL, TP, TP_choice, SL_choice, current_index, ema_short, ema_long):

    if ema_short[current_index] < ema_long[current_index] and ema_short[current_index - 1] > ema_long[current_index - 1]:
        Trade_Direction = 0

    elif ema_short[current_index] > ema_long[current_index] and ema_short[current_index - 1] < ema_long[current_index - 1]:
        Trade_Direction = 1

    stop_loss_val, take_profit_val = SetSLTP(-99, -99, Close, High, Low, Trade_Direction,
                                             SL, TP, TP_choice, SL_choice, current_index)

    return Trade_Direction, stop_loss_val, take_profit_val
```

Next add this to Bot_Class.py:

Firstly in the __init__(….) function we need to define our emas.

```
64          self.EMA_short = None
65          self.EMA_long = None
```

The default bot already has two variables for ema_short and ema_long so we can just reuse them.

Next in update_indicators() we need to add an elif clause that generates the emas for us when self.strategy == ‘ema_crossover’:

```
121         elif self.strategy == 'heikin_ashi_ema':
122             self.use_close_pos = True
123             self.fastd = np.array(stochrsi_d(pd.Series(self.Close)))
124             self.fastk = np.array(stochrsi_k(pd.Series(self.Close)))
125             self.EMA200 = np.array(ema_indicator(pd.Series(self.Close), window=200))
126         elif self.strategy == 'ema_crossover':
127             self.EMA_short = np.array(ema_indicator(pd.Series(self.Close), window=10))
128             self.EMA_long = np.array(ema_indicator(pd.Series(self.Close), window=20))
129
```

update_indicators() gets called when a new candle comes in for the live bot and once at the start of a backtest to calculate the indicators.

Similarly we add an elif clause in Make_Decision() now feeding the relevant variables to our strategy:

```
elif self.strategy == 'ema_crossover':
    Trade_Direction, stop_loss_val, take_profit_val = ema_crossover(self.Trade_Direction, self.Close, self.High, self.Low,
                                                                    self.SL, self.TP, self.TP_choice, self.SL_choice,
                                                                    self.current_index, self.EMA_short, self.EMA_long)
```

Finally, in app.py we need to add ema_crossover to the strategy_options array:

```
strategy_options = ["StochRSIMACD", "tripleEMAStochasticRSIATR", "tripleEMA", "breakout", "stochBB", "goldenCross",
                    "candle_wick", "fibMACD", "EMA_cross", "heikin_ashi_ema2", "heikin_ashi_ema", "ema_crossover"]
```

And that’s all we need to do to add a new strategy.

Custom TP and SL:

Similar to the strategy array there is a TP_ TP_sub_options array and a SL_ SL_sub_options array inside app.py, these are linked to the if elif guards in SetSLTP() in TradingStrats.py

So If we want to add new SL or TP options we just need to add a new function in these guards with a new name, for both TP_choice and SL_choice.

```
def SetSLTP(stop_loss_val, take_profit_val, Close, High, Low, Trade_Direction, SL, TP, TP_choice, SL_choice, current_index):
    if TP_choice == '%':
        take_profit_val = (TP / 100) * Close[current_index]
    if SL_choice == '%':
        stop_loss_val = (SL / 100) * Close[current_index]

    if TP_choice == 'x (ATR)':
        ATR = np.array(average_true_range(pd.Series(High), pd.Series(Low), pd.Series(Close)))
        take_profit_val = TP * abs(ATR[current_index])

    if SL_choice == 'x (ATR)':
        ATR = np.array(average_true_range(pd.Series(High), pd.Series(Low), pd.Series(Close)))
        stop_loss_val = SL * abs(ATR[current_index])

    if TP_choice == 'x (Swing High/Low) level 1':
        high_swing = High[current_index]
        Low_swing = Low[current_index]
        high_flag = 0
        low_flag = 0
        j = current_index - 1
        while j > -1:
            if High[j] > high_swing and high_flag == 0:
                high_swing = High[j]
                if High[j - 1] < High[j] > High[j + 1]:
```

Then in app.py we just need to add the corresponding SL name and TP name that we have used in our guards and they will then be linked up in the GUI:

```
SL_sub_options = ['%', 'x (ATR)', 'x (Swing High/Low) level 1', 'x (Swing Close) level 1',
                  'x (Swing High/Low) level 2', 'x (Swing Close) level 2', 'x (Swing High/Low) level 3',
                  'x (Swing Close) level 3']
SL_sub.set("%")
```

```
TP_sub_options = ['%', 'x (ATR)', 'x (Swing High/Low) level 1', 'x (Swing Close) level 1',
                  'x (Swing High/Low) level 2', 'x (Swing Close) level 2', 'x (Swing High/Low) level 3',
                  'x (Swing Close) level 3']
TP_sub = StringVar()
```