Calvin Thomas

Summary: 1. The IV % mean-reversion off highest minute SPX volume (retail sales)
2. Expected value of capturing IV% spikes intraday is a strong long-vol opportunity in peak SPX volumes (options bubble? Peak >70M daily options volume on select Friday's, 10.38 billion options contracts cleared (OCC), >40% SPX volume (aka trillions) traded in 0dte).

We should put 500% as max goal on a bi-monthly or monthly basis: a tiny fraction of capital, <0.5% as an long vol hedge to HFT alpha, a saturated field which is undiversified across time (infinitesimally small holding periods). It is a long-vol hedge diversifier across time, as HFT quants conduct high-freq. intraday operations. Buying reverse iron butterfly 0dte. Our sell is at either when peak retail closes, or after algorithmic dump at 2 pm. It depends also if you bought at open or at 11 pm. The earliest is best. So flipping when optimal retail covering of their initial positions. The next part is incorporating QNN to sniff out and predict the highest minute SPX volume/strikes.

Implementing a specific trading strategy using options contracts to hedge against high-frequency trading (HFT) alpha and capture IV% mean-reversion and short-term market movements. We discussed in "0dte SPX + SPY strategies" the pressure on retailer delivery of physical underlying, or to close at predesignated 3X stop losses as time approaches expiry. One approach you could take is to use reverse iron butterfly options contracts with a 0dte to profit from intraday market volatility while limiting potential losses. We buy close-to-ATM strikes P/C, then sell OTM strikes P/C. We should use trading simulators and back-testing for validating results.

```
import numpy as np
import pandas as pd
import yfinance as yf

stock_symbol = 'SPY' # Replace with the desired stock symbol
start_date = '2022-01-01' # Replace with the desired start date
end_date = '2022-03-31' # Replace with the desired end date

stock_data = yf.download(stock_symbol, start=start_date, end=end_date)
```

```
# Calculate the strike prices for the reverse iron butterfly options contracts.
current_price = stock_data['Close'][-1]
upper_strike_price = current_price * 1.05
lower_strike_price = current_price * 0.95
```

 # Buy the put and call options at the lower strike price, and sell the put and call options at the upper strike price.

```
option_contract_size = 100 # Each option contract represents 100 shares

buy_put_option =
yf.Ticker(f'{stock_symbol}{end_date}P{int(lower_strike_price)}').option_chain('monthly').puts[0]
buy_call_option =
yf.Ticker(f'{stock_symbol}{end_date}C{int(lower_strike_price)}').option_chain('monthly').calls[0]
sell_put_option =
yf.Ticker(f'{stock_symbol}{end_date}P{int(upper_strike_price)}').option_chain('monthly').puts[0]
sell_call_option =
yf.Ticker(f'{stock_symbol}{end_date}C{int(upper_strike_price)}').option_chain('monthly').calls[0]

total_cost = (buy_put_option['ask'] + buy_call_option['ask']) * option_contract_size
total_credit = (sell_put_option['bid'] + sell_call_option['bid']) * option_contract_size
```

 # Set a maximum profit goal of 500% on a tiny fraction of your capital, less than 0.5%

```
max_profit_goal = 5.0 # 500% profit goal
max_loss_limit = total_cost * 0.005 # 0.5% of capital

if total_credit >= total_cost * (1 + max_profit_goal):
    sell_option = True
elif total_credit <= total_cost - max_loss_limit:
    sell_option = True
else:
    sell_option = False
```

 # Sell the option contracts at the optimal time, either when peak retail closes or after an algorithmic dump at 2 pm.

```
if sell_option:
    sell_date = end_date # Replace with the desired sell date and time
```

```
    sell_put_option =
yf.Ticker(f'{stock_symbol}{sell_date}P{int(lower_strike_price)}').option_chain('monthly').puts[0]
    sell_call_option =
yf.Ticker(f'{stock_symbol}{sell_date}C{int(lower_strike_price)}').option_chain('monthly').calls[0]
    total_profit = (sell_put_option['bid'] + sell_call_option['bid'] - buy_put_option['ask'] -
buy_call_option['ask']) * option_contract_size
else:
    total_profit = 0.0


# Print the results.
print(f'Total cost: ${total_cost:.2f}')
print(f'Total credit: ${total_credit:.2f}')
print(f'Total profit: ${total_profit:.2f}')

# To diversify your portfolio across time, you can repeat steps 2-6 for different expiration
dates and strike prices. You can also adjust the size of your position for each option contract
to limit your overall risk.


import numpy as np
import pandas as pd
import yfinance as yf

# Define parameters
stock_symbol = 'SPY'
start_date = '2022-01-01'
end_date_list = ['2022-04-15', '2022-05-20', '2022-06-17'] # Replace with desired expiration
dates
strike_price_ratio_list = [0.95, 0.975, 1.0, 1.025, 1.05] # Replace with desired strike price ratios
max_loss_limit_ratio = 0.005 # 0.5% of capital

# Calculate current price
stock_data = yf.download(stock_symbol, start=start_date, end=end_date_list[-1])
current_price = stock_data['Close'][-1]

# Loop through expiration dates and strike prices
total_profit = 0
for end_date in end_date_list:
    for ratio in strike_price_ratio_list:
        # Calculate strike prices
        upper_strike_price = current_price * ratio
        lower_strike_price = current_price * (2 - ratio)
```

```
        # Buy and sell option contracts
        buy_put_option =
yf.Ticker(f'{stock_symbol}{end_date}P{int(lower_strike_price)}').option_chain('monthly').puts[0]
        buy_call_option =
yf.Ticker(f'{stock_symbol}{end_date}C{int(lower_strike_price)}').option_chain('monthly').calls[0]
        sell_put_option =
yf.Ticker(f'{stock_symbol}{end_date}P{int(upper_strike_price)}').option_chain('monthly').puts[0]
        sell_call_option =
yf.Ticker(f'{stock_symbol}{end_date}C{int(upper_strike_price)}').option_chain('monthly').calls[0]

        option_contract_size = 100 # Each option contract represents 100 shares
        total_cost = (buy_put_option['ask'] + buy_call_option['ask']) * option_contract_size
        total_credit = (sell_put_option['bid'] + sell_call_option['bid']) * option_contract_size

        # Set maximum profit goal and limit for potential losses
        max_profit_goal = 5.0 # 500% profit goal
        max_loss_limit = total_cost * max_loss_limit_ratio

        # Sell option contract if it meets profit or loss criteria
        if total_credit >= total_cost * (1 + max_profit_goal) or total_credit <= total_cost -
max_loss_limit:
            sell_put_option =
yf.Ticker(f'{stock_symbol}{end_date}P{int(lower_strike_price)}').option_chain('monthly').puts[0]
            sell_call_option =
yf.Ticker(f'{stock_symbol}{end_date}C{int(lower_strike_price)}').option_chain('monthly').calls[0]
            total_profit += (sell_put_option['bid'] + sell_call_option['bid'] - buy_put_option['ask'] -
buy_call_option['ask']) * option_contract_size

# Print total profit
print(f'Total profit: ${total_profit:.2f}')
```

This code loops through each combination of expiration date and strike price ratio, buys and sells option contracts, and calculates the total profit based on the maximum profit goal and loss limit for each option contract. By diversifying your portfolio across time and strike prices, you can limit your overall risk and increase your chances of capturing short-term market movements.

 # To incorporate real-time market data, you can use a Python library like websocket-client to subscribe to market data feeds and update your trading strategy in real-time.

```
import websocket
```

```
import json
import numpy as np
import pandas as pd
import yfinance as yf

# Define parameters
stock_symbol = 'SPY'
expiration_date = '2022-06-17'
max_loss_limit_ratio = 0.005 # 0.5% of capital

# Define callback function to handle received messages
def on_message(ws, message):
    message_data = json.loads(message)

    # Extract current price from message
    current_price = message_data['data'][0]['p']

    # Calculate strike prices
    upper_strike_price = current_price * 1.05
    lower_strike_price = current_price * 0.95

    # Buy and sell option contracts
    buy_put_option =
yf.Ticker(f'{stock_symbol}{expiration_date}P{int(lower_strike_price)}').option_chain('monthly').p
uts[0]
    buy_call_option =
yf.Ticker(f'{stock_symbol}{expiration_date}C{int(lower_strike_price)}').option_chain('monthly').c
alls[0]
    sell_put_option =
yf.Ticker(f'{stock_symbol}{expiration_date}P{int(upper_strike_price)}').option_chain('monthly').
puts[0]
    sell_call_option =
yf.Ticker(f'{stock_symbol}{expiration_date}C{int(upper_strike_price)}').option_chain('monthly').
calls[0]

    option_contract_size = 100 # Each option contract represents 100 shares
    total_cost = (buy_put_option['ask'] + buy_call_option['ask']) * option_contract_size
    total_credit = (sell_put_option['bid'] + sell_call_option['bid']) * option_contract_size

    # Set maximum profit goal and limit for potential losses
    max_profit_goal = 5.0 # 500% profit goal
    max_loss_limit = total_cost * max_loss_limit_ratio

    # Sell option contract if it meets profit or loss criteria
```

```
    if total_credit >= total_cost * (1 + max_profit_goal) or total_credit <= total_cost -
max_loss_limit:
        sell_put_option =
yf.Ticker(f'{stock_symbol}{expiration_date}P{int(lower_strike_price)}').option_chain('monthly').p
uts[0]
        sell_call_option =
yf.Ticker(f'{stock_symbol}{expiration_date}C{int(lower_strike_price)}').option_chain('monthly').c
alls[0]
        total_profit = (sell_put_option['bid'] + sell_call_option['bid'] - buy_put_option['ask'] -
buy_call_option['ask']) * option_contract_size

        # Print total profit
        print(f'Total profit: ${total_profit:.2f}')

# Define callback function to handle errors
def on_error(ws, error):
    print(error)

# Define callback function to handle connection closed
def on_close(ws):
    print("Connection closed")

# Define callback function to handle connection opened
def on_open(ws):
    print("Connection opened")

# Connect to websocket and subscribe to market data feed
ws = websocket.WebSocketApp("wss://socket.polygon.io/stocks/crypto", on_open=on_open,
on_message=on_message, on_error=on_error, on_close=on_close)
ws.run_forever()
```

This code connects to a websocket and subscribes to a real-time market data feed. When a new message is received, it extracts the current price and uses it to buy and sell option contracts based on the same strategy as the previous code examples. If a profitable trade opportunity is detected, it sells the option contracts and prints the total profit.

#

To test your trading strategy and evaluate potential risks and rewards, you can use a trading simulator like Backtrader or Zipline. These simulators allow you to test your strategy using historical data and evaluate performance metrics like Sharpe ratio and maximum drawdown.

```
import backtrader as bt
import yfinance as yf

# Define parameters
stock_symbol = 'SPY'
start_date = '2022-01-01'
end_date = '2022-06-17'
capital = 10000
commission = 0.01 # 1% commission per trade
expiration_date = '2022-06-17'
strike_price_ratio = 1.0
max_loss_limit_ratio = 0.005 # 0.5% of capital

# Define trading strategy
class OptionsTradingStrategy(bt.Strategy):
    def __init__(self):
        self.stock_data = self.datas[0]
        self.total_profit = 0

    def next(self):
        current_price = self.stock_data.close[0]

        # Calculate strike prices
        upper_strike_price = current_price * 1.05
        lower_strike_price = current_price * 0.95

        # Buy and sell option contracts
        buy_put_option =
yf.Ticker(f'{stock_symbol}{expiration_date}P{int(lower_strike_price)}').option_chain('monthly').p
uts[0]
        buy_call_option =
yf.Ticker(f'{stock_symbol}{expiration_date}C{int(lower_strike_price)}').option_chain('monthly').c
alls[0]
        sell_put_option =
yf.Ticker(f'{stock_symbol}{expiration_date}P{int(upper_strike_price)}').option_chain('monthly').
puts[0]
        sell_call_option =
yf.Ticker(f'{stock_symbol}{expiration_date}C{int(upper_strike_price)}').option_chain('monthly').
calls[0]

        option_contract_size = 100 # Each option contract represents 100 shares
        total_cost = (buy_put_option['ask'] + buy_call_option['ask']) * option_contract_size
        total_credit = (sell_put_option['bid'] + sell_call_option['bid']) * option_contract_size
```

```
        # Set maximum profit goal and limit for potential losses
        max_profit_goal = 5.0 # 500% profit goal
        max_loss_limit = total_cost * max_loss_limit_ratio

        # Sell option contract if it meets profit or loss criteria
        if total_credit >= total_cost * (1 + max_profit_goal) or total_credit <= total_cost -
max_loss_limit:
            sell_put_option =
yf.Ticker(f'{stock_symbol}{expiration_date}P{int(lower_strike_price)}').option_chain('monthly').p
uts[0]
            sell_call_option =
yf.Ticker(f'{stock_symbol}{expiration_date}C{int(lower_strike_price)}').option_chain('monthly').c
alls[0]
            total_profit = (sell_put_option['bid'] + sell_call_option['bid'] - buy_put_option['ask'] -
buy_call_option['ask']) * option_contract_size
            self.total_profit += total_profit

            # Print total profit
            print(f'Total profit: ${self.total_profit:.2f}')

# Initialize backtest
cerebro = bt.Cerebro()

# Add data feed
stock_data = bt.feeds.YahooFinanceData(dataname=stock_symbol,
fromdate=pd.Timestamp(start_date), todate=pd.Timestamp(end_date))
cerebro.adddata(stock_data)

# Add trading strategy
cerebro.addstrategy(OptionsTradingStrategy)

# Set broker commission
cerebro.broker.setcommission(commission=commission)

# Set initial capital
cerebro.broker.setcash(capital)

# Run backtest
cerebro.run()

# Print performance metrics
print(f'Sharpe ratio: {cerebro.analyzers.sharpe.get_analysis()["sharperatio"]:.2f}')
```

```
print(f'Maximum drawdown:
{cerebro.analyzers.drawdown.get_analysis()["max"]["drawdown"]:.2%}')
```

#This code uses Backtrader to simulate trades based on the same trading strategy as the previous. It initializes the backtest with a data feed for historical stock prices, adds the trading strategy, and sets the broker commission and initial capital. It then runs the backtest and prints performance metrics like the Sharpe ratio and maximum drawdown.