The Art of Selling Covered LEAPS: Earn Premiums, Reduce Risk

LEAPS present a compelling investment strategy, offering income generation and risk control. This post will guide readers through developing a backtesting algorithm for covered calls, covering data acquisition, methodology, and results.

Selling Covered Leaps To Safely Invest

Covered Long-Term Equity Anticipation Securities (LEAPS) can be an attractive investment strategy, as the premiums earned offset potential losses from stock price fluctuations. While covered calls can help protect against downside risk, they also cap the underlying asset's gain at the strike price. Many new investors use calls to leverage their holdings, but shorting covered calls can help investors reduce portfolio volatility and risk.

Given the potential benefits of writing covered calls, this article will demonstrate how to properly source and utilize options data to backtest covered call trading strategies. Due to the Federal Reserve's plans to lower interest rates, high-yield cash accounts will continue to face declines (In the past several years Wealthfront has dropped their APY from 5% to 4%). As these cash accounts diminish in profit, having a safe investment strategy becomes increasingly important for low-risk investors.

What Are Covered Calls and LEAPS?

An American call option is a contract that grants its buyer the right to purchase 100 shares of a stock at a set price (the strike price) on or before the option’s expiration date. When you write a call, you collect a premium upfront. However, at any time between the sale and the option’s expiration, the purchaser has the right to buy the shares at the predetermined strike price.

In simple terms, a LEAP is a long-term option that typically expires in 1 to 3 years. This longer time horizon often results in a larger premium for the writer. If a LEAP position is closed at a loss, that loss can offset current-year capital gains or be carried forward to offset future gains, depending on the timing of the close.

Tax Rate Differences Between Long- and Short-Term Gains (Last Updated 2024)

A Necessary Assumption on All Calls

An option buyer technically has the right to purchase the shares at any point before expiration. Since options are priced with respect to the stock's movement, strike price, and time until expiration, exercising an option early would cause investors to lose out on additional profits. For this reason, it's reasonable to assume they won't exercise the option prematurely. Additionally, if the stock price falls below the strike price, exercising the option would force the buyer to purchase the shares above market value. For this reason, it is assumed that the buyer will not exercise the option.

Profit from writing a covered call can be calculated as:

$$\text{Profit} = \begin{cases} (S_T - S_0) \times 100 + C_0 \times 100, & \text{if } S_T \leq K \\ (K - S_0) \times 100 + C_0 \times 100, & \text{if } S_T > K \end{cases}$$

Where:

- S_0 = the initial stock price when purchased
- S_T = the stock price at option expiration
- K = the option's strike price
- C_0 = the premium received for selling the call
- The factor of 100 accounts for the standard option contract size (each option typically covers 100 shares).

A Quick Example

Imagine you own 100 shares of a stock priced at 49.50 each. You decide to write a LEAP call with a 50 strike price that expires in 18 months and receive a premium of 5 per share.

If the stock rises above 50, you sell at 50 while still retaining the premium. In this situation, the option can be purchased immediately prior to the expiration, leading to a "loss" which will be mostly offset by the price of the shares. If the shares are kept, this "loss" can be used as a tax write-off for other capital gains.

Let's assume that the stock price ends at 60 and that directly prior to expiration, the option can be bought back for 10.03 (estimated by $S_T - K$ plus a small factor due to the other Greeks). Profit can be calculated as the sum of the loss from purchasing the option back, the underlying stock gain, and the initial premium, with an added tax loss bonus:

$$\text{Profit} = -(10.03 \times 100) + ((60 - 50) \times 100) + (5 \times 100) = 497$$

$$\text{Tax Write-Off} = (10.03 \times 100) = 1003$$

By purchasing back the option and holding onto the underlying shares, the profit and additional gains can be written off. When the shares are eventually sold, taxes will have to be paid on gains, but this can be done during retirement when an investor's tax rate will be lower.

If the stock remains below 50 at expiration, you keep the shares and the premium. A necessary reminder is that any loss in the underlying stock will negatively affect the portfolio value.

Let's assume the stock price ends at 47:

$$\text{Profit} = ((47 - 50) \times 100) + (5 \times 100) = 200$$

While this profit is less than that of when the stock ends above the strike price, it serves as a reminder that money was made during an 18-month period of negative returns.

Setting Up A Covered Calls Trading Environment

To determine if selling covered calls can result in reasonable gains, a python backtesting environment can be created. For this, historical options data along with python will be used.

Acquiring Data

Prior to 2024, historical options data has never been available to the common investor for free. However, recently optionsDX (https://www.optionsdx.com/?post_type=product) offers free end of day options quotes for tickers including SPY, QQQ, TSLA, AAPL, NVDA, and others. Feel free to download as much data as possible. However, the results shown later will be based off the underlying equities of QQQ and SPY.

After downloading the options data, It can be cleaned and organized using the following Jupyter notebook setup

Follow this folder setup when storing the downloaded data

```
plaintext
options/
├── backtest/
│   └── covered_call_leap.ipynb
└── read_data/
    └── data/
        ├── clean/
        │   ├── QQQ.parquet
        │   └── SPY.parquet
        ├── raw/
            ├── QQQ
                └── downloaded QQQ options data
            └── SPY
                └── downloaded QQQ options data
        └── clean_data.ipynb
```

Lets take a look at the options/read_date/clean_data.ipynb

First import the Necessary packages

In [1]:
```
# Imports
import os
import pandas as pd
```

Then choose the ticker to clean

In [2]:
```
# Set Ticker... Let's imagine we want to clean the SPY options data
ticker = 'SPY'
```

Read and Concatenate all unclean SPY files

In [ ]:
```
# Initialize all_options_data to concatenate later
all_options_data_to_concat = []

# Ignore .DS_Store
folders = [file for file in sorted(os.listdir(f'data/raw/{ticker}')) if file !=

# Iterate through `data` folder
for folder in folders:

    # Print status
    print(f"Folder: {folder.split('_')[2].split('-')[0]}")

    # Get sub folder with each spy eod
    folder_path = os.path.join(f'data/raw/{ticker}', folder)

    # Iterate through files in each subfolder
    for file in os.listdir(folder_path):

        # Get file path
        file_path = os.path.join(folder_path, file)

        # Read file and append to array for later concatenation
        all_options_data_to_concat.append(pd.read_csv(file_path, low_memory=Fa

# Concatenate into one dataframe
all_options_data = pd.concat(all_options_data_to_concat, ignore_index=True)
```

Clean columns and assign proper feature types

In [ ]:
```
# Rename columns
all_options_data.rename(columns={
    '[QUOTE_UNIXTIME]': 'datetime', ' [QUOTE_READTIME]': 'quote_readtime', ' [(
    ' [EXPIRE_UNIX]': 'expire_unix', ' [DTE]': 'days_till_expiration', ' [C_DEL
    ' [C_IV]': 'c_iv', ' [C_VOLUME]': 'c_volume', ' [C_LAST]': 'c_last', ' [C_S
    ' [P_SIZE]': 'p_size', ' [P_LAST]': 'p_last', ' [P_DELTA]': 'p_delta', ' [P
    ' [P_VOLUME]': 'p_volume', ' [STRIKE_DISTANCE]': 'strike_distance_delete',
}, inplace=True)

# Cast to appropriate types
all_options_data['datetime'] = pd.to_datetime(all_options_data['datetime'], un
```

```
all_options_data['quote_readtime'] = pd.to_datetime(all_options_data['quote_rea
all_options_data['date'] = pd.to_datetime(all_options_data['date'], errors='coe
all_options_data['time'] = pd.to_numeric(all_options_data['time'], errors='coe
all_options_data['stock_price'] = pd.to_numeric(all_options_data['stock_price']
all_options_data['expiration_date'] = pd.to_datetime(all_options_data['expirat
all_options_data['expire_unix'] = pd.to_datetime(all_options_data['expire_unix
all_options_data['days_till_expiration'] = pd.to_numeric(all_options_data['days
all_options_data['c_delta'] = pd.to_numeric(all_options_data['c_delta'], errors
all_options_data['c_gamma'] = pd.to_numeric(all_options_data['c_gamma'], errors
all_options_data['c_vega'] = pd.to_numeric(all_options_data['c_vega'], errors=
all_options_data['c_theta'] = pd.to_numeric(all_options_data['c_theta'], errors
all_options_data['c_rho'] = pd.to_numeric(all_options_data['c_rho'], errors='co
all_options_data['c_iv'] = pd.to_numeric(all_options_data['c_iv'], errors='coe
all_options_data['c_volume'] = pd.to_numeric(all_options_data['c_volume'], erro
all_options_data['c_last'] = pd.to_numeric(all_options_data['c_last'], errors=
all_options_data['c_size'] = pd.to_numeric(all_options_data['c_size'], errors=
all_options_data['c_bid'] = pd.to_numeric(all_options_data['c_bid'], errors='co
all_options_data['c_ask'] = pd.to_numeric(all_options_data['c_ask'], errors='co
all_options_data['strike'] = pd.to_numeric(all_options_data['strike'], errors=
all_options_data['p_bid'] = pd.to_numeric(all_options_data['p_bid'], errors='co
all_options_data['p_ask'] = pd.to_numeric(all_options_data['p_ask'], errors='co
all_options_data['p_size'] = pd.to_numeric(all_options_data['p_size'], errors=
all_options_data['p_last'] = pd.to_numeric(all_options_data['p_last'], errors=
all_options_data['p_delta'] = pd.to_numeric(all_options_data['p_delta'], errors
all_options_data['p_gamma'] = pd.to_numeric(all_options_data['p_gamma'], errors
all_options_data['p_vega'] = pd.to_numeric(all_options_data['p_vega'], errors=
all_options_data['p_theta'] = pd.to_numeric(all_options_data['p_theta'], errors
all_options_data['p_rho'] = pd.to_numeric(all_options_data['p_rho'], errors='co
all_options_data['p_iv'] = pd.to_numeric(all_options_data['p_iv'], errors='coe
all_options_data['p_volume'] = pd.to_numeric(all_options_data['p_volume'], erro
all_options_data['strike_distance_delete'] = pd.to_numeric(all_options_data['st
all_options_data['strike_distance'] = pd.to_numeric(all_options_data['strike_di
```

Add variable for call price

In [ ]:
```
# Set `call_price` to midpoint between call bid and call ask
all_options_data['call_price'] = (all_options_data['c_bid'] + all_options_data
```

Drop unnecessary data

In [ ]:
```
# Drop columns
all_options_data = all_options_data.drop(columns=['datetime', 'quote_readtime'
all_options_data = all_options_data.dropna(subset=['call_price'])
```

Sort data

In [ ]:
```
# Sort SPY options data by date and expiration date
all_options_data = all_options_data.sort_values(by=['date', 'expiration_date']
```

Finally, convert to parquet for later use

In [ ]:
```
# Save for later
all_options_data.to_parquet(f'data/clean/{ticker}.parquet')
```

Now that clean options data has been acquired lets setup a trading environment where clear buy and sell signal can be established

Lets now use the options/covered_call_leap.ipynb file

Necessary imports

In [ ]:
```
# Imports
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import yfinance as yf

from dateutil.relativedelta import relativedelta

from tqdm import tqdm
tqdm.pandas()
```

Read in clean option files

In [ ]:
```
# Initialize a dictionary to hold dataframes for each ticker
portfolio_option_data = {}

# Iterate through tickers and set dictionary to hold proper data
for TICKER in TICKERS:

    # Read data
    portfolio_option_data[TICKER] = pd.read_parquet(f'../read_data/data/clean/
```

To create an "leap trading backtest environment", lets identify which leap option we would sell on a given day

This covered calls strategy looks for near-the-money calls

In [ ]:
```
# Initialize a dictionary to hold the options
portfolio_leap_options = {}

# Set a max days to expiration, such that in early years with less option data
MAX_DAYS_TILL_EXPIRATION = 400

# Find best leap contracts for each ticker
for TICKER in TICKERS:
    print(TICKER)

    # Group by date
    portfolio_leap_options[TICKER] = portfolio_option_data[TICKER].groupby(['da

        # Iterate through each group (for other code readers, think about the
        lambda group: group.loc[

            # Get first `expiration_date` that surpasses year from `date`... al
            (group['expiration_date'] == group.loc[
                group['expiration_date'] > (group.name + relativedelta(months=
                ['expiration_date'].min())

            ]
```

```
            # Of these `expiration_date`s, make sure that `days_till_expiratio
            .loc[group['days_till_expiration'] < MAX_DAYS_TILL_EXPIRATION]

            # Of these `expiration_date`s with aforementioned critetia, obtain
            .nsmallest(1, 'strike_distance')

            # Output these column
            [['expiration_date', 'strike', 'days_till_expiration', 'call_price

        # Reset index so date is column and drop added `level_1` index
        ).reset_index().drop(columns = ['level_1'])
```

The output of this code should look something like this | Index | date | expiration_date | strike | days_till_expiration | call_price | stock_price | |-------|----------|----------------|--------|----------------------|------------|------------| | 0 | 2010-06-03 | 2011-06-17 | 110.0 | 379 | 10.855 | 110.71 | | 1 | 2010-06-04 | 2011-06-17 | 105.0 | 378 | 11.965 | 107.00 | | 2 | 2010-06-07 | 2011-06-17 | 105.0 | 375 | 11.155 | 105.49 | | 3 | 2010-06-08 | 2011-06-17 | 105.0 | 374 | 11.685 | 106.57 | | 4 | 2010-06-09 | 2011-06-17 | 105.0 | 373 | 11.370 | 106.02 |

Now that for each given day it is known which LEAP would be sold, let's determine if rolling over into a new option prematurely is ever justified.

An example strategy could involve repurchasing an option when its value declines to 20% of its original premium and selling a new option the next day.

The ideal time to buy back an option is determined by the cost of the trade and the income potential of the new option.

The decision criterion can look something like is:

$$C_r > C_t + f$$

Where:

- C_r = Premium received from selling the new call (roll)
- C_t = Current market price to buy back the old call
- f = Brokerage fee for the trade

Should this strategy be used? It depends on whether you believe in the Efficient Market Hypothesis (EMH).

If you believe markets are efficient and future stock price changes have a 50% chance of increasing and a 50% chance of decreasing (i.e., following true geometric Brownian motion), this criterion should guide rollovers.

However, as demonstrated by historical data for SPY and QQQ, markets tend to rise over time. This upward trend means rolling over and locking in lower strike prices repeatedly may

cap the appreciation potential of the underlying shares. While this strategy maximizes covered call income, it limits the growth of the underlying portfolio value.

Here is the correspoding code to determine when to rollover for each option

Feel Free to try your own strategies

In [ ]:
```
# Function help determine when to sell option
def option_sell_date_helper(target_option, target_option_day_0, leaps, strategy
    # Get leap on day of target_option_data... aka the leap that would be rolle
    rollover_leap = leaps.loc[leaps['date'] == target_option['date']]

    if not rollover_leap.empty:

        # Benchmark leap strategy
        if strategy == 'benchmark':
            return False

        elif strategy == 'YOUR_STRATEGY':
            rollover_bool = 'YOUR_ROLLOVER_CRITERIA'
            return rollover_bool

# Function to determine when to sell an option
def option_sell_date(option, all_leap_options, all_option_data, strategy):

    # Gets useful `all_leap_options` to consider
    leaps = all_leap_options.loc[(all_leap_options['date'] > option['date']) &

    # Follows the `option` from `date` to `expiration date`
    target_option_data = all_option_data.loc[(all_option_data['expiration_date

    # Determine if option should roll over on each data
    target_option_data['sell_bool'] = target_option_data.apply(option_sell_date

    # If there exists a true `sell_bool`
    if target_option_data['sell_bool'].any() == True:

        # Set `sell_date` to minimum `target_option_date`
        sell_date = target_option_data.loc[target_option_data['sell_bool'] == 

        # Set `tax_rate` to short
        tax_rate = 'short'

    # If there does not exist a true `sell_bool`
    else:
        # Set `sell_date` to `expiration_date`
        sell_date = option['expiration_date']

        # Set `tax_rate` to short
        tax_rate = 'long'

    # Set `call_last_price` to the last price in the target_option_data
    call_last_price = target_option_data.loc[target_option_data['date'] == sel

    # If `call_last_price` is known set it
    if not call_last_price.empty:
        call_last_price = call_last_price.iloc[0]
```

```
    # Else set to None
    else:
        call_last_price = None

    return pd.Series(data = {'sell_date' : sell_date, 'call_last_price' : call
```

In [ ]:
```
# Initialize a dictionary for each benchmark trading strategy
portfolio_benchmark = {}

# Iterate through tickers
for TICKER in TICKERS:
    print(TICKER)

    # Copy full option data
    options_benchmark = portfolio_leap_options[TICKER].copy()

    # Get the leap options for each day
    options_benchmark[['sell_date', 'call_last_price', 'tax_rate']] = portfoli

    # Add to portfolio
    portfolio_benchmark[TICKER] = options_benchmark

    # Print count of leaps in each year
    print(portfolio_benchmark[TICKER].groupby(portfolio_benchmark[TICKER]['dat

all_strategies['benchmark'] = portfolio_benchmark
```

The output of this code for one strategy and ticker should look like | Index | date | expiration_date | strike | days_till_expiration | call_price | stock_price | sell_date | call_last_price | tax_rate | |-------|----------|----------------|--------|----------------------|------------|------------|------------|----------------|----------| | 0 | 2012-01-03 | 2013-01-18 | 57.0 | 381 | 5.400 | 56.90 | 2012-11-21 | 7.08 | short | | 1 | 2012-01-04 | 2013-01-18 | 57.0 | 380 | 5.450 | 57.14 | 2012-11-21 | 7.08 | short | | 2 | 2012-01-05 | 2013-01-18 | 58.0 | 379 | 5.105 | 57.61 | 2012-11-21 | 6.15 | short | | 3 | 2012-01-06 | 2013-01-18 | 58.0 | 378 | 5.120 | 57.78 | 2012-11-21 | 6.15 | short | | 4 | 2012-01-09 | 2013-01-18 | 58.0 | 375 | 4.985 | 57.63 | 2012-11-21 | 6.15 | short |

We now have a robust backtesting environment that provides clear, actionable decisions for each trading day based on the specified strategy. This environment supports structured analysis of multiple approaches and their outcomes over past market conditions.

Following this, code can be used to determine when options should be traded

Simply, chronologically iterate through this backtesting environment to choose which options will be traded when.

While you can simply buy an option immediately, there might be potential advantages to waiting for prices to revert. Feel Free to test out your own strategy!

In [ ]:
```
def leaps_to_trade(leap_options):
    # Set first leap to the first `leap_options['date']`
    leap_dates = [leap_options['date'].iloc[0]]
```

```
    # Set the first `sell_date` to the sell date of the first `leap_options['da
    sell_date = leap_options.loc[leap_options['date'] == leap_dates[-1]]['sell_

    # Determines if it was a quick or slow trade
    type = leap_options.loc[leap_options['date'] == leap_dates[-1]]['tax_rate']

    # Gets previous stock sell price
    previous_price = leap_options.loc[leap_options['date'] == leap_dates[-1]][

    # Iterate through the `leap_options` until the `sell_date` > the last `leap
    while sell_date < leap_options['date'].iloc[-1]:
        if strategy == 'benchmark':
            leap_dates.append(leap_options.loc[leap_options['date'] > sell_date

        if strategy == 'YOUR_STRATEGY':
            # Determine when you want to buy your next option
            leap_dates.append('YOUR_CRITERIA')

    return leap_dates

# Initiate a dictionary to find each trade performed
portfolio_trades = {}

# Iterate through each strategy
for strategy in strategies:
    # Initialize strategy
    portfolio_trades[strategy] = {}

    # Iterate through each ticker
    for TICKER in TICKERS:
        print(f'{strategy.upper()} : {TICKER}')

        leap_dates = leaps_to_trade(strategies[strategy][TICKER])

        # Get all the options from `leap_options` with the starting dates with
        portfolio_trades[strategy][TICKER] = strategies[strategy][TICKER].loc[
```

The output of this should look like

Date	Expiration Date	Strike	Days Till Expiration	Call Price	Stock Price	Sell Date	Call Last Price	Tax Rate
2010-06-03	2011-06-17	110.0	379	10.855	110.71	2010-12-01	14.540	short
2010-12-02	2011-12-16	123.0	379	9.575	122.57	2011-02-28	14.545	short
2011-03-01	2012-03-16	131.0	380	9.335	130.93	2011-08-23	2.425	short
2011-08-24	2012-09-21	118.0	394	11.280	118.10	2012-01-12	16.445	short
2012-01-13	2013-01-18	130.0	371	10.055	128.95	2012-08-21	13.915	short

Results

Having organized the trading options and timelines into distinct DataFrames for each strategy, the next step is to develop functions for backtesting and evaluating profitability and risk metrics for each approach. In my personal function, I chose to track a diverse set of metrics that include that include leap-to-leap returns (returns only on days that the strategy is actively buying or selling), daily returns including the cost to purchase back the covered call, daily returns excluding the cost to purchase back the covered call, tax losses from buying back covered calls, variance, downside deviation, value at risk (VaR), max drawdown, and others. These results will help determine how this strategy compares to investing in the underlying asset.

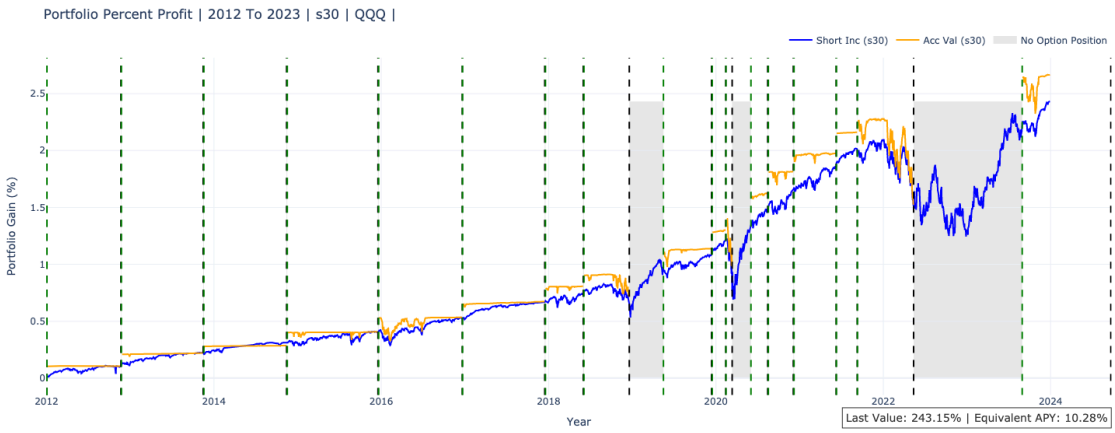
My personal strategy includes a completely self-sustainable portfolio where money is not transferred in or out. However, this would not be a completely accurate assessment. While in my strategy SPY shares are sold to repurchase the LEAPs, in a true portfolio, options are purchased by alternative income such that they can be used for tax write offs.

Using my personal strategy and an initial investment of $100,000 for each ticker, I achieved the following results:

The Short Inc blue line shows the true cost of the portfolio. This means what the investor would have at a current time if they decided to close their covered call by repurchasing.

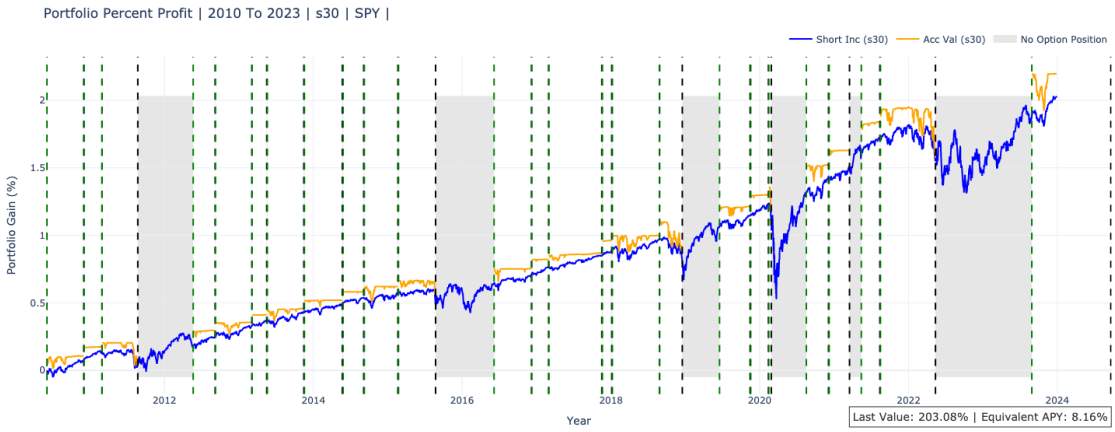
The Acc Val orange line shows the cost of letting the option expire and does include the cost of repurchasing the options. Since the downloaded options data ended on 2023-12-31, 2024 data cannot be calculated.

All metrics are calculated annually



Metric	Underlying Equity	s30
Variance	20.50%	15.74%
Max Drawdown	-35.63%	-27.59%
Value at Risk (VaR) at 5% confidence	-2.08%	-1.35%
Downside Deviation	0.0137	0.0105

With a cumalative ability to write off $449896 in taxes



Metric	Underlying Equity	s30
Variance	17.46%	14.80%
Max Drawdown	-34.32%	-31.82%
Value at Risk (VaR) at 5% confidence	-1.68%	-1.29%
Downside Deviation	0.0117	0.0100

With a cumalative ability to write off $318530 in taxes

Improved Analysis

The analysis demonstrates the use of a LEAPS on QQQ and SPY over an extended period. This method offers reduced volatility and downside risks compared to holding the underlying assets outright.

Key Observations:

1. Risk-Reduction Metrics:
    - Variance and Max Drawdown are consistently lower with the s30 strategy than the underlying assets:
        - For QQQ, the strategy reduces variance to 15.74% and drawdown to -27.59%, a significant improvement over the underlying equity's variance of 20.50% and drawdown of -35.63%.
        - SPY exhibits similar trends, with the variance at 14.80% (down from 17.46%) and drawdown at -31.82% (down from -34.32%).
    - These reductions indicate that covered calls smooth the returns and provide substantial downside protection during periods of market turmoil.
2. Profitability:
    - The strategy yields an annualized percentage yield (APY) of 10.28% for QQQ and 8.16% for SPY. While these returns are lower than those of the underlying equities

during bullish periods, they are generated in a much safer investment environment, appealing to conservative investors.
- The downside protection enables consistent portfolio growth while mitigating risks during volatile periods.

3. Value at Risk (VaR):

- The VaR, which estimates the potential loss in the portfolio at a 5% confidence level, is substantially reduced for the s30 strategy:
  - QQQ: -1.35% (s30) vs. -2.08% (Underlying Equity).
  - SPY: -1.29% (s30) vs. -1.68% (Underlying Equity).
- This improvement reflects the strategy's ability to reduce exposure to extreme downside scenarios.

4. Tax Advantages:

- The ability to write off taxes, with cumulative amounts of 449,896 for QQQ and 318,530 for SPY, adds another layer of value to the strategy. These tax efficiencies can significantly enhance after-tax returns for investors, especially those in high-tax brackets. Remember these "tax breaks" are not real losses although the options are repurchased for a higher price, the underlying asset is kept at a higher value. The unrealized underlying asset gain can be sold post retirment or during years when investors fall into lower tax brackets

Future Work To Be Done

Because of the structure of this code, it is possible to implement different expiration periods, rollover, and purchasing strategies. While I believe that I created a successful rollover and purchase strategy, implementing different strategies could potentially yield better results. In addition, profits can be made while completely ignoring the benefits of LEAPs. By changing the period till expiration to different values, numerous strategies can be imposed. While these LEAP strategies are commonly done in wealth management, I want to continue to research other options strategies such as covered ratio spreads for both defensive and offensive positions. By safely utilizing options, investors can better reach their investment goals.

Conclusion

The covered call strategy using LEAPS offers a compelling approach for investors seeking to balance risk and reward in their portfolios. By capping upside potential in exchange for steady income and downside protection, this strategy delivers consistent returns with lower volatility and drawdowns compared to holding the underlying assets outright. The analysis demonstrates that applying the strategy to QQQ and SPY results in an annualized return of 10.28% and 8.16%, while additional reducing the risk. These features make it particularly

suitable for conservative investors or those looking to stabilize portfolio performance during uncertain market conditions.

Moreover, the strategy provides notable tax advantages, with the ability to write off significant amounts. These efficiencies further enhance the net returns, especially for high-net-worth investors managing taxable accounts who might have tax rates near 50\% of their revenue.

While the strategy may underperform the underlying equities during strong bull markets, it shines in periods of volatility or downturns by safeguarding capital. This feature is especially evident in the 2020 market crash, where the strategy mitigated losses more effectively than a simple buy-and-hold approach.

Looking forward, this analysis lays the foundation for further exploration. Experimenting with alternative expiration periods, refining rollover criteria, and integrating other options-based strategies. Overall, the covered call strategy with LEAPS stands as a versatile and reliable method for achieving steady, risk-adjusted returns in diverse market environments.

Author: Samuel Friedman