Sophisticated Algorithmic Strategies Which You Need To Learn

Here, we will explore how prominent players in the algorithmic trading industry use advanced trading tactics.

ONEPAGECODE
SEP 18, 2023 · PAID

After exploring algorithmic strategies, we will explore more sophisticated methodologies, such as statistical arbitrage and pair correlation. As we explore the merits and drawbacks of these approaches, we will gain insights into how to design a trading strategy that accommodates the volatility of different instruments. In addition, we will learn how to create a trading strategy tailored for economic events and implement the fundamentals of statistical arbitrage trading.

Several key areas will be explored in this exploration. As we formulate a trading strategy, we'll focus on adapting it to the evolving nature of the instrument. It impacts not only the execution but also the potential success of trades as a whole. A trading strategy's success heavily depends on price variation, a measure of volatility. An agile and adaptable approach is essential for navigating the impact of such fluctuations. We will then examine the construction of a trading strategy focusing on economic news. Financial reports, interest rate adjustments, and shifts in unemployment rates can trigger market movements. Trading strategies that can capitalize on these shifts are essential for any trader. Finally, we will study and implement basic statistical arbitrage trading strategies. Statistical arbitrage is a complex yet rewarding strategy that allows traders to profit from price discrepancies identified through mathematical modelling. Understanding and implementing the essential concepts can significantly enhance a trader's performance and profitability.

We are analyzing the volatility of different assets to develop a technique for adapting it.

This is an walkthrough article. There is no external notebook to download, every code snippet and its result are there in this article. So enjoy

The level of investor confidence plays a significant role in understanding price volatility. As a result, it provides investors with an indication of their willingness to invest and how long they are prepared to retain their investments. The level of investor confidence declines when the volatility of prices rises, resulting in rapid and significant swings. On the other hand, when price volatility decreases, investors are tempted to hold longer-term positions with greater risk. Furthermore, volatility in specific asset categories tends to affect other areas, perpetuating volatility across industries, such as housing and consumer spending. For sophisticated strategies, adapting to volatility changes is imperative. As a result, they must take positions cautiously, plan holding periods, and promptly manage profit/loss expectations.

In many previous methods, volatility fluctuations of the trading instrument needed to be addressed or adequately compensated. We will examine how changes in volatility affect trading instruments and strategies to enhance profitability.

The adjustment of technical indicators to consider the fluctuating volatility of different trading instruments.

Previously, we examined how to generate trading signals based on predefined criteria in Understanding Market Trends through Technical Analysis. Our analysis was maintained by applying predetermined standards like using a 20-day moving average or selecting optimal periods and smoothing constants. Despite its merits, this straightforward approach has limitations. However, it is essential to note that these signals may become less reliable as market volatility increases. Our chosen strategy can fluctuate in effectiveness depending on external market factors, leading to significant disadvantages.

We then looked at Bollinger Bands and standard deviation as other indicators. A trading instrument's price can vary based on these indicators. Low volatility triggers more active signals for entering positions and less active signals for closing positions when price movements have a lower standard deviation. In volatile periods, however, the higher standard deviation in price movements dampens the signals that would suggest trading. Due to the increasing moving average volatility, the bands follow a wider spread. Therefore, these signals incorporate the impact of volatility on trading instruments.

Standard deviation can be integrated into any of the technical indicators previously discussed. As a result, it is possible to create a more advanced version of the basic technical indicator, with flexible time periods, smoothing factors, and volatility levels. Parameters are dynamically

adjusted when standard deviation is incorporated as a measure of volatility. In light of this, moving averages are able to adjust their historical data or time periods in response to market volatility. A shorter historical perspective captures more observations during periods of high volatility, while a longer historical perspective captures fewer observations during periods of low volatility. Similarly, smoothing factors can vary according to the degree of volatility. In this adjustment, newer observations receive a higher weight than older ones. Although we won't explore this topic in-depth, once the fundamental concept of utilizing volatility measures to create complex indicators is understood, applying this concept to technical indicators is straightforward.

Different trading instruments require different trading strategies to be adapted.

Adapting to market fluctuations can be employed in trading strategies. It is possible to incorporate shifting volatility into momentum or trend-following strategies to flexibly adjust the time periods applied to moving averages or to adjust the number of consecutive up/down days that represent entry signals. When detecting a trend or a reversal, we can use changing volatility to dynamically adjust our entry and exit thresholds. In this way, entry and exit points at each position are optimized.

In the world of mean reversion strategies, the process of applying volatility metrics is quite comparable. Moving averages with varying timeframes and thresholds for entering positions when overbuying or overselling is indicated would allow us to adopt a more flexible approach. In the same way, we can modify the exit thresholds if there are signs that prices will return to equilibrium in the near future. We'll explore a variety of ways to adjust for volatility in trading strategies in the upcoming sections, evaluating each method's influence on the overall strategy.

Based on market uncertainty, these strategies exploit price fluctuations and their tendency to revert to their average.

The purpose of this section is to develop a straightforward example of a mean reversion strategy. The following part of the presentation will illustrate how volatility adjustment can enhance and stabilize risk-adjusted performance of the strategy.

Financial markets can be taken advantage of using the trading signals generated by the absolute price oscillator to look for opportunities for mean reversion. We can identify profit opportunities when an asset's price diverges significantly from its average value, indicating a

potential profit opportunity. Trading in a dynamic trading market requires recognizing deviations in prices and executing trades accordingly, which can help us capitalize on the tendency of prices to return to their mean.

Our discussion in “Decoding Market Behavior with Technical Analysis” examined a practical method for taking advantage of market fluctuations using the Absolute Price Oscillator (APO). This strategy uses two fixed moving average values: a 10-day Fast Exponential Moving Average (EMA) and a 40-day Slow Exponential Moving Average (EMA). When the APO signal falls below -10, we initiate buying trades. Our sell trades will be executed when the APO signal value rises above +10. In addition, new trades will occur at prices different from the last trade in order to prevent excessive trading. Our positions will be closed when the APO signal goes negative and when it goes positive when the APO signal becomes negative. With this dynamic approach, we can respond to changing market conditions while balancing risk and opportunity.

Additionally, positions are terminated regardless of APO values, even if they generate significant profits. In addition to providing reliable profits through algorithmic means, this strategy also enables the initiation of new positions beyond simply relying on the value of the trading signal. We will examine the practical implementation of this approach in the next sections.

As before, we will collect information by following the same procedures. During the last four years, we have collected data on Goos. The ‘Datareader’ function in the pandas_datareader package will be used here. It retrieves price details on ‘goos’ from Yahoo Finance from January 2014 to January 2018. For future reference, an alternative file might be used if the ‘.pk1’ file, used to store the data on the disk, is not available. In spite of missing or corrupted files, this backup plan can still provide access to data.

```
import pandas as pd
from pandas_datareader import data



# Define the symbol and the date range for data fetching
SYMBOL = 'GOOG'
start_date = '2014-01-01'
end_date = '2018-01-01'
DATA_FILENAME = SYMBOL + '_data.pkl'
```

```
# Try to load the data from a file if it exists, otherwise fetch the data
from Yahoo Finance
try:
    stock_data = pd.read_pickle(DATA_FILENAME)
except FileNotFoundError:
    stock_data = data.DataReader(SYMBOL, 'yahoo', start_date, end_date)
    stock_data.to_pickle(DATA_FILENAME)
```

Several unchanging factors and adaptable elements will be discussed in this section when executing Fast and Slow EMA calculations.

```
NUM_PERIODS_FAST = 10
K_FAST = 2 / (NUM_PERIODS_FAST + 1)  # Smoothing factor for fast EMA
ema_fast = 0
ema_fast_values = []
```

```
NUM_PERIODS_SLOW = 40
K_SLOW = 2 / (NUM_PERIODS_SLOW + 1)  # Smoothing factor for slow EMA
ema_slow = 0
ema_slow_values = []
```

```
apo_values = []
```

Furthermore, it is essential to incorporate variables that establish and regulate trading strategies, along with mechanisms for managing profit and loss.

```
orders = []
positions = []
pnls = []
last_buy_price = 0
last_sell_price = 0
position = 0
buy_sum_price_qty = 0
buy_sum_qty = 0
```

```
sell_sum_price_qty = 0
sell_sum_qty = 0
open_pnl = 0
closed_pnl = 0
```

Finally, we establish the criteria for participation: the minimum price fluctuation since the previous transaction, the expected minimum profit per transaction, and the number of shares to exchange.

```
APO_VALUE_FOR_BUY_ENTRY = -10
APO_VALUE_FOR_SELL_ENTRY = 10
MIN_PRICE_MOVE_FROM_LAST_TRADE = 10
MIN_PROFIT_TO_CLOSE = 10
NUM_SHARES_PER_TRADE = 10
```

The fundamentals of trading will be discussed in this section. Each of the elements of the approach plays a crucial role in ensuring success. These elements include calculating and adjusting the Fast and Slow EMAs (Exponential Moving Averages), as well as the APO (Absolute Price Oscillator). Making informed decisions is made easier with these valuable tools, which help a company understand the market's trends and momentum. The approach also incorporates the concept of triggering long or short positions according to the signals generated by the market. It is crucial to establish a position in trading, regardless of whether it is long or short, because it determines not only the trade's direction but also its potential for profit or loss. To close long or short positions effectively, the approach outlines how to manage trading signals, open positions, open Profit and Loss (PnLs), and market prices. The closing of positions is as crucial as the opening of them. A trader's decision to exit a trade can determine whether they achieve profit or incur losses. Therefore, it is an essential element of any trading strategy.

```
closing_prices = data['Close']
for close_price in closing_prices:
    # If first observation, set initial EMA values
    if not ema_fast and not ema_slow:
        ema_fast = ema_slow = close_price
    else:
```

```
        ema_fast = (close_price - ema_fast) * K_FAST + ema_fast
        ema_slow = (close_price - ema_slow) * K_SLOW + ema_slow



    ema_fast_values.append(ema_fast)
    ema_slow_values.append(ema_slow)



    # Compute and append APO value
    apo_values.append(ema_fast - ema_slow)
```

It will analyze trading indicators in relation to trading limits or criteria, as well as positions, in order to perform trades. When we reach the closing price, we will begin a selling trade if we meet predetermined requirements. The Absolute Price Oscillator (APO) trading indicator must move above our Sell-Entry limit, a pre-determined value. Furthermore, the price difference between the previous trade and the current trade must be significant enough for the trade to be justified. Furthermore, if the asset is owned, there are two possible scenarios for selling it. In this case, the APO trading indicator will reach zero or exceed it. Additionally, if the current position has generated enough profit, it may be prudent to secure it. A selling trade will be executed when the program detects these conditions.

```
if ((apo > APO_VALUE_FOR_SELL_ENTRY and abs(close_price - last_sell_price) >
MIN_PRICE_MOVE_FROM_LAST_TRADE)
    or (position > 0 and (apo >= 0 or open_pnl > MIN_PROFIT_TO_CLOSE))):

    # Sell trade
    orders.append(-1)
    last_sell_price = close_price
    position -= NUM_SHARES_PER_TRADE  # Update position

    # Update sell sum quantities
    sell_sum_price_qty += (close_price * NUM_SHARES_PER_TRADE)
    sell_sum_qty += NUM_SHARES_PER_TRADE

    print(f"Sell {NUM_SHARES_PER_TRADE} @ {close_price}, Position:
{position}")
```

A purchase transaction at the closing price will only be executed if certain conditions are met. An APO trading indicator must first be below the Buy-Entry threshold. Additionally, the previous trade price must differ significantly from the present price. Our current position (short) requires either that the APO trading indicator is at or below zero, or that we have established a satisfactory level of profitability for profit protection.

```
if ((apo < APO_VALUE_FOR_BUY_ENTRY and abs(close_price - last_buy_price) >
MIN_PRICE_MOVE_FROM_LAST_TRADE)  # APO suggests buy and price has moved
enough
    or (position < 0 and (apo <= 0 or open_pnl > MIN_PROFIT_TO_CLOSE))):  #
We are short and APO turned negative or we have enough profit

    # Buy trade
    orders.append(+1)
    last_buy_price = close_price
    position += NUM_SHARES_PER_TRADE  # Update position

    # Update buy sum quantities
    buy_sum_price_qty += (close_price * NUM_SHARES_PER_TRADE)
    buy_sum_qty += NUM_SHARES_PER_TRADE

    print(f"Buy {NUM_SHARES_PER_TRADE} @ {close_price}, Position:
{position}")
else:
    # Neither buy nor sell conditions were met
    orders.append(0)



positions.append(position)
```

Profit and loss calculations are integrated into the strategy’s code. Market prices and trades that change positions require the code to be updated to reflect them and calculate both open and closed profits and losses.

```
# Updating Open and Closed PnL positions
open_pnl = 0
```

```
if position > 0:
    if sell_sum_qty > 0:  # Long position with some sales, compute PnL based
on sales
        open_pnl = abs(sell_sum_qty) * (sell_sum_price_qty/sell_sum_qty -
buy_sum_price_qty/buy_sum_qty)
    # Calculate remaining open position PnL
    open_pnl += abs(sell_sum_qty - position) * (close_price -
buy_sum_price_qty / buy_sum_qty)



elif position < 0:
    if buy_sum_qty > 0:  # Short position with some buys, compute PnL based
on buys
        open_pnl = abs(buy_sum_qty) * (sell_sum_price_qty/sell_sum_qty -
buy_sum_price_qty/buy_sum_qty)
    # Calculate remaining open position PnL
    open_pnl += abs(buy_sum_qty - position) *
(sell_sum_price_qty/sell_sum_qty - close_price)



else:
    # No open positions, update closed PnL and reset variables
    closed_pnl += (sell_sum_price_qty - buy_sum_price_qty)
    buy_sum_price_qty = sell_sum_price_qty = buy_sum_qty = sell_sum_qty =
last_buy_price = last_sell_price = 0



print(f"Open PnL: {open_pnl}, Closed PnL: {closed_pnl}")
pnls.append(closed_pnl + open_pnl)
```

This section aims to demonstrate how to collect trading strategy-related information with Python/Matplotlib code. Data include market prices, Fast and Slow EMA values, APO values, Buy and Sell trades, Positions, and Profits and Losses achieved throughout the life of the strategy. By visually presenting the data, we gain valuable insights into the strategy's performance.

```
# Prepare the dataframe and visualize trading strategy outcomes
data = data.assign(
    ClosePrice=pd.Series(close, index=data.index),
```

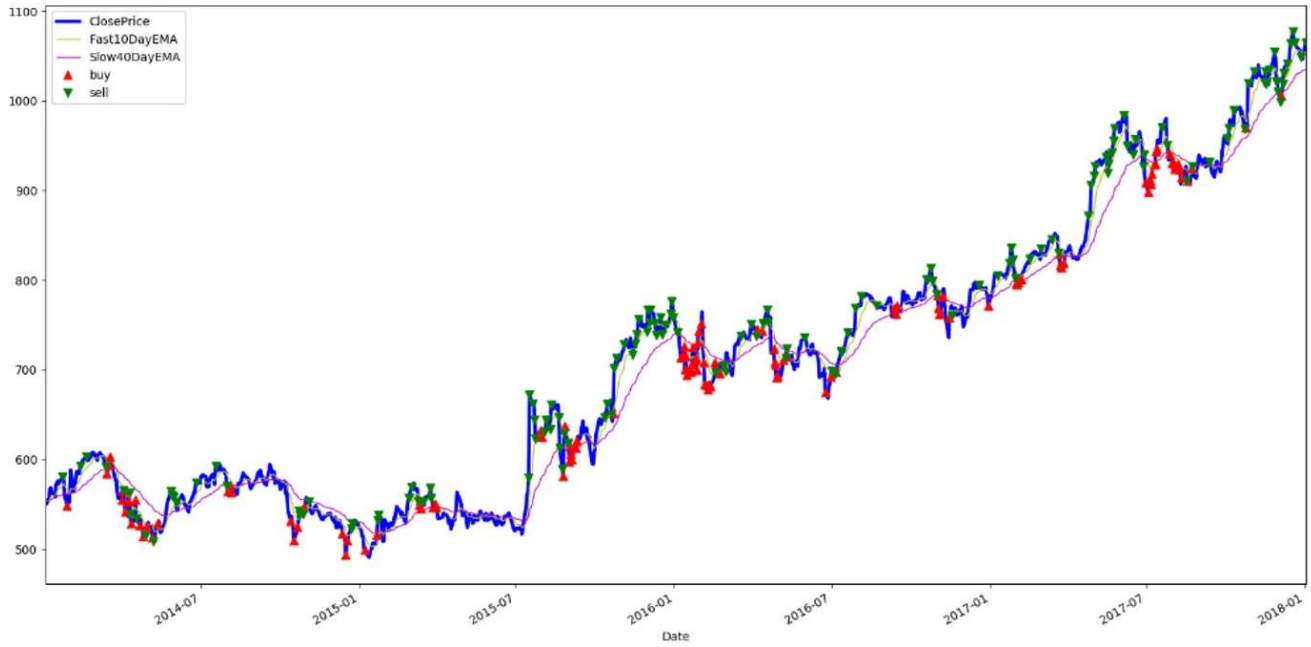
```
    Fast10DayEMA=pd.Series(ema_fast_values, index=data.index),
    Slow40DayEMA=pd.Series(ema_slow_values, index=data.index),
    APO=pd.Series(apo_values, index=data.index),
    Trades=pd.Series(orders, index=data.index),
    Position=pd.Series(positions, index=data.index),
    Pnl=pd.Series(pnls, index=data.index)
)
```

Previously, we calculated a range of series that we will now incorporate into our data frame. Besides the Market Price, the Fast and Slow EMA values will be included. Additionally, we will create a separate plot to illustrate the value of the APO trading signal. Visually identifying when a trade is initiated and closed will be easy with these plots.

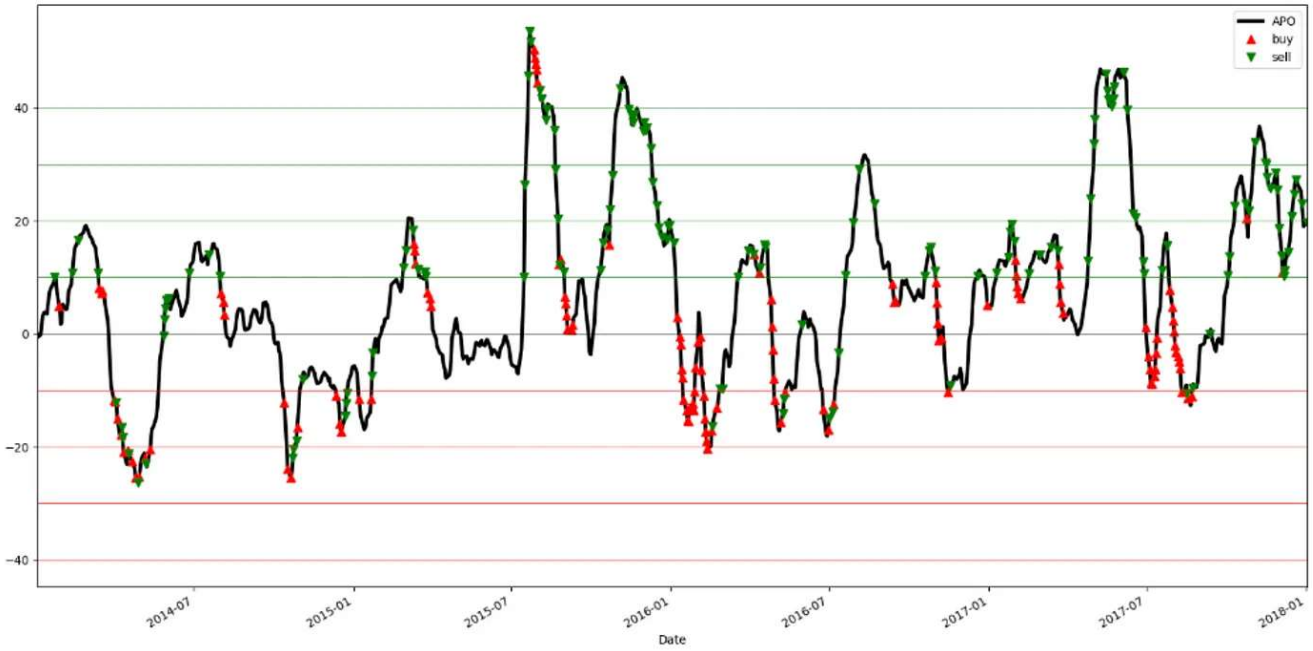
```
import matplotlib.pyplot as plt



# Define plotting function for reuse
def plot_data(data, indicator, marker_buy='^', marker_sell='v'):
    data[indicator].plot(color='k', lw=3., legend=True)
    plt.plot(data.loc[data.Trades == 1].index, data[indicator][data.Trades
== 1],
             color='r', lw=0, marker=marker_buy, markersize=7, label='buy')
    plt.plot(data.loc[data.Trades == -1].index, data[indicator][data.Trades
== -1],
             color='g', lw=0, marker=marker_sell, markersize=7,
label='sell')
    plt.legend()
    plt.show()



# Plot ClosePrice, Fast10DayEMA, Slow40DayEMA
data[['ClosePrice', 'Fast10DayEMA', 'Slow40DayEMA']].plot(color=['blue',
'y', 'm'], lw=[3, 1, 1], legend=True)
plot_data(data, 'ClosePrice')



# Plot APO
plot_data(data, 'APO')
```

```
# Add horizontal lines to APO plot
plt.axhline(y=0, lw=0.5, color='k')
for i in range(APO_VALUE_FOR_BUY_ENTRY, APO_VALUE_FOR_BUY_ENTRY*5,
APO_VALUE_FOR_BUY_ENTRY):
    plt.axhline(y=i, lw=0.5, color='r')
for i in range(APO_VALUE_FOR_SELL_ENTRY, APO_VALUE_FOR_SELL_ENTRY*5,
APO_VALUE_FOR_SELL_ENTRY):
    plt.axhline(y=i, lw=0.5, color='g')
```

Examining our trading patterns now, we can examine the EMA and APO values during trades. The underlying patterns are revealed by analyzing positions and PnL plots.



Our analysis of the storyline over the past four years allows us to observe exactly when buying and selling transactions took place, resulting from fluctuations in Google stock prices. Let's now examine the significance of the APO trading signal values for these buy and sell trades. According to the trading strategies, a positive APO indicates an ideal time to sell, and a negative APO indicates a good time to buy.



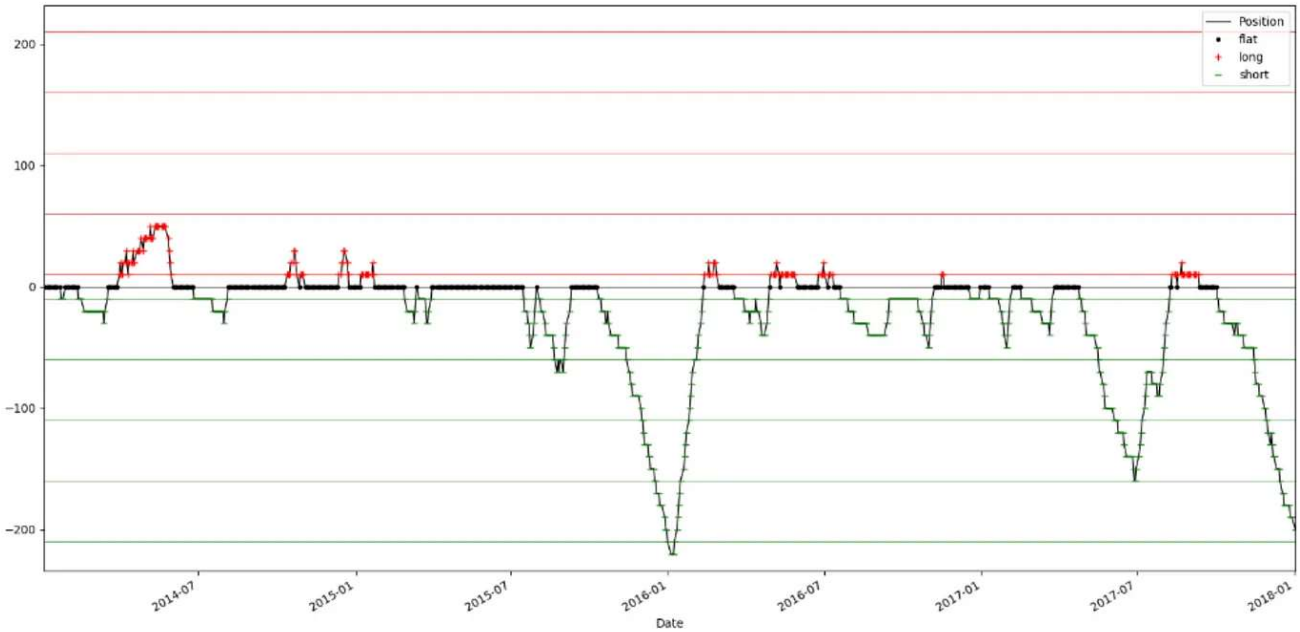
As we progress through the storyline, it becomes apparent that a significant number of selling transactions occur when APO signal values indicate positivity, while a significant number of buying transactions occur when APO signal values indicate negativity. Furthermore, we observe selling transactions taking place despite positive APO trading signal values, and buying transactions taking place despite negative APO trading signal values. In light of such events, we are left wondering at their rationale.

Our next section will focus on how to secure profits through trades. We will closely monitor how this strategy's position and profit margins change over time.

```
# Define a reusable function for plotting Positions and PnL
def plot_data_positions_pnl(data, indicator, marker_flat='.',
marker_long='+', marker_short='_', marker_pnl='.'):
    data[indicator].plot(color='k', lw=1., legend=True)
    if indicator == 'Position':
        plt.plot(data.loc[data.Position == 0].index,
data.Position[data.Position == 0], color='k', lw=0, marker=marker_flat,
label='flat')
        plt.plot(data.loc[data.Position > 0].index,
data.Position[data.Position > 0], color='r', lw=0, marker=marker_long,
label='long')
        plt.plot(data.loc[data.Position < 0].index,
data.Position[data.Position < 0], color='g', lw=0, marker=marker_short,
label='short')
        plt.axhline(y=0, lw=0.5, color='k')
```

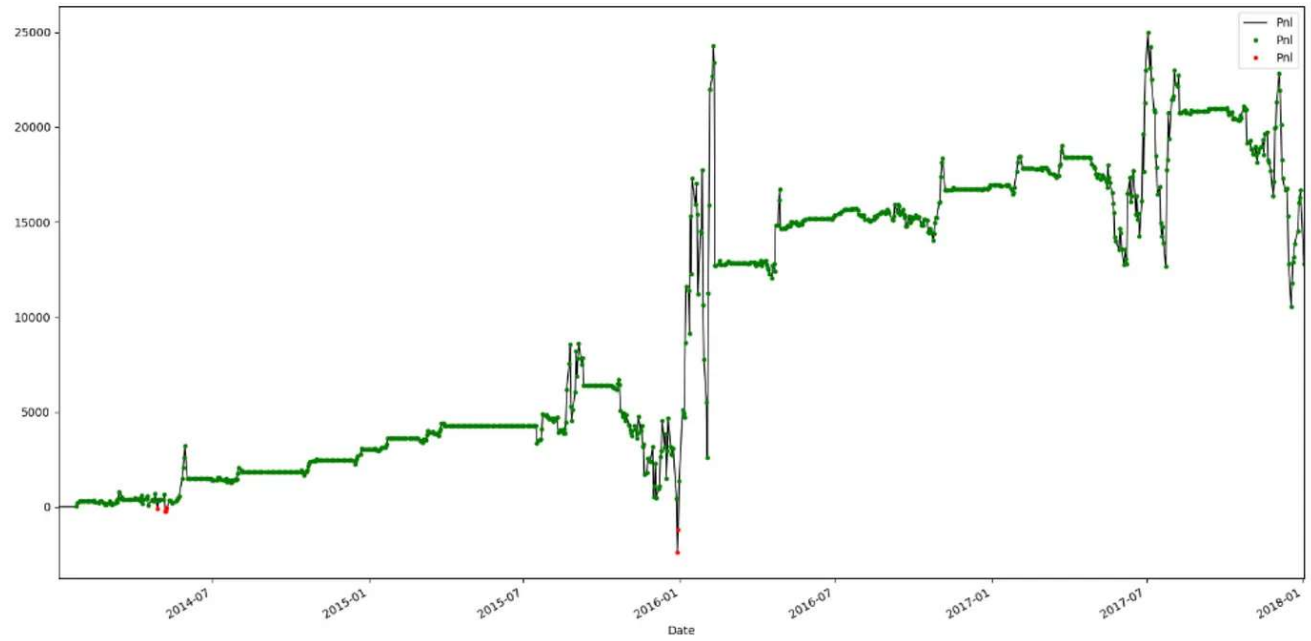
```
        for i in range(NUM_SHARES_PER_TRADE, NUM_SHARES_PER_TRADE*25,
NUM_SHARES_PER_TRADE*5):
            plt.axhline(y=i, lw=0.5, color='r')
        for i in range(-NUM_SHARES_PER_TRADE, -NUM_SHARES_PER_TRADE*25, -
NUM_SHARES_PER_TRADE*5):
            plt.axhline(y=i, lw=0.5, color='g')
    else: # indicator == 'Pnl'
        plt.plot(data.loc[data.Pnl > 0].index, data.Pnl[data.Pnl > 0],
color='g', lw=0, marker=marker_pnl)
        plt.plot(data.loc[data.Pnl < 0].index, data.Pnl[data.Pnl < 0],
color='r', lw=0, marker=marker_pnl)
    plt.legend()
    plt.show()



# Call function for Position and Pnl
plot_data_positions_pnl(data, 'Position')
plot_data_positions_pnl(data, 'Pnl')
```

When the code is executed, the desired result will be generated. Check out the fascinating visuals in these two charts.



In analyzing the plot displaying positions, we find that short positions are significant around January 2016, July 2017, and January 2018. Trading signals from the APO also show positive

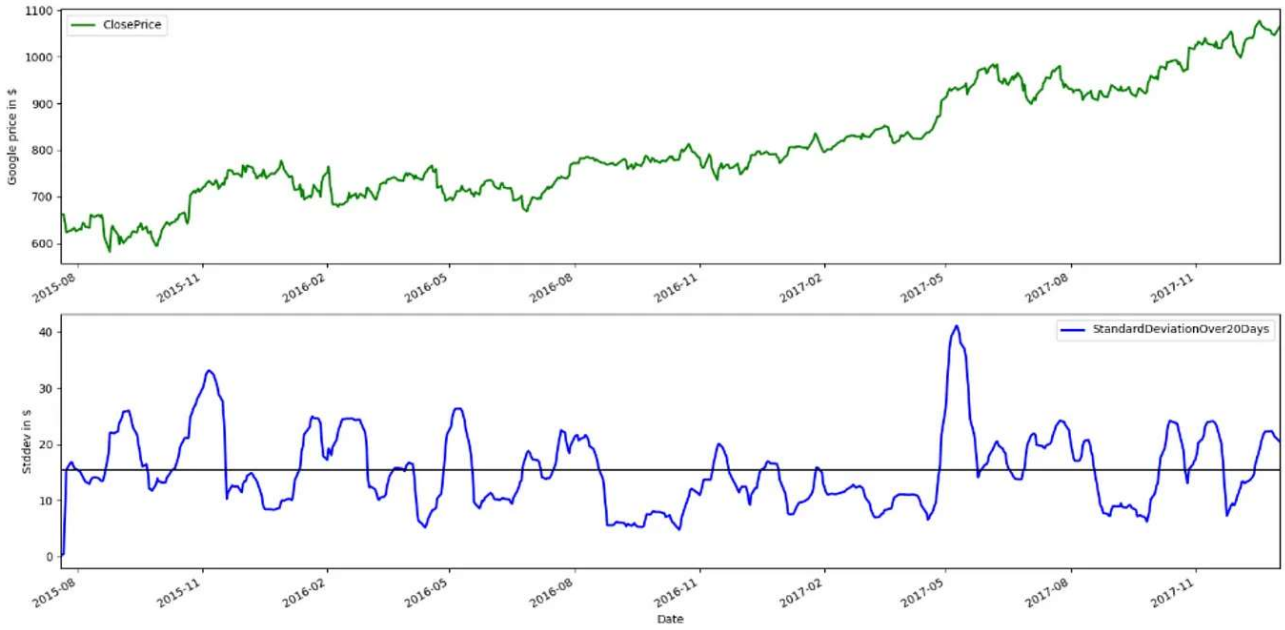
patches around these instances. We can now examine the evolution of profit and loss (PnL) for this trading strategy over the lifecycle of a stock.



Although the mean reversion strategy has consistently produced profitable returns over time, there was some fluctuation in returns from January 2016 to July 2017 since prominent positions were taken. Nevertheless, the strategy reaches around $15K, one of its highest profit and loss values.

A mean-reverting strategy that adapts to volatility fluctuations.

Our next step is to employ the previously discussed ideas to alter the duration of Fast and Slow EMA by utilizing a volatility gauge alongside a volatility-modified APO entry signal. Rather than using the volatility indicator, we will use the standard deviation (STDEV) indicator we examined earlier, ‘Deciphering the Markets with Technical Analysis’. We should refresh our memory about the Google dataset by taking a quick look at the results of this indicator:



Based on the data results, it is apparent that the measure of unpredictability within our investigation falls between $8 and $40 over a 20-day period. During this time period, the average measure hovers around $15. The degree of uncertainty will be encapsulated by an integration factor, ranging from 0 to 1. A factor representing standard deviation is constructed by applying the formula ‘stdev_factor = stdev/15’. Those values that are close to zero indicate minimal volatility. An increase over 1 signifies higher volatility than a value near 1, and a value below 1 signifies an increase below 1. In addition to STDEV (Standard Deviation), we’ll introduce specific modifications that will be elaborated in the discussion that follows.

Rather than using fixed κ_FAST EMA values, we’ll use volatility instead. Our method allows us to ensure that K__FAST and K__SLOW are able to adjust faster to recent data during instances of high volatility by multiplying with a stdev_factor. This approach aligns with our intuition and enhances the responsiveness of the indicators.

Our method of entering positions based on the primary trading signal (APO) is more flexible than relying on fixed criteria such as APO_VALUE_FOR_BUY_ENTRY and APO_VALUE_FOR_SELL_ENTRY. With the addition of dynamic thresholds, our equation will now take into account market volatility. The new thresholds will be calculated by multiplying APO_VALUE_FOR_BUY_ENTRY by stdev_factor, and APO_VALUE_FOR_SELL_ENTRY by stdev_factor. It represents the standard deviation factor ‘stdev_factor’. When volatility is high, this method allows us to be more cautious. As a result, we introduce an additional layer of risk management by adjusting the entry threshold according to the volatility factor. It emphasizes

the importance of minimizing potential losses during volatile periods, which is in alignment with the discussion of the preceding section. By doing so, our trading strategy becomes both more flexible and intuitive, and more able to deal with market fluctuations as they arise.

We will ultimately embrace inconsistency by implementing a flexible profit threshold. To close, we will consider the minimum profit divided by the standard deviation factor rather than relying on a minimum profit threshold. When faced with thrilling opportunities during periods of heightened volatility, this approach allows us to be more daring. Holding positions for extended periods can be riskier in times of above-average volatility.

Let's see how we can enhance the fundamental mean reversion strategy by making the necessary adjustments. The first step is creating a piece of code to monitor and revise the volatility gauge (STDEV).

```
import statistics as stats
import math as math



# Define constants and variables for standard deviation computation
SMA_NUM_PERIODS = 20  # Look-back period
price_history = []  # Container for past prices
```

This is essentially the primary strategy's fundamental loop, with a consistent position and PnL management.

```
# Fetch close prices from the data
close = data['Close']



# Iterate over the close prices
for close_price in close:
    price_history.append(close_price)
    if len(price_history) > SMA_NUM_PERIODS:  # Retain only the latest
'SMA_NUM_PERIODS' prices
        price_history.pop(0)
```

```
    sma = stats.mean(price_history)  # Calculate the simple moving average
(SMA)
    variance = sum((hist_price - sma) ** 2 for hist_price in price_history)
# Calculate variance



    stdev = math.sqrt(variance / len(price_history))  # Standard deviation
    stdev_factor = stdev / 15 if stdev != 0 else 1  # Standard deviation
factor



    # Update fast and slow EMA and compute APO trading signal
    if not ema_fast:  # Check if it's the first observation
        ema_fast = ema_slow = close_price
    else:
        ema_fast = (close_price - ema_fast) * K_FAST * stdev_factor +
ema_fast
        ema_slow = (close_price - ema_slow) * K_SLOW * stdev_factor +
ema_slow



    ema_fast_values.append(ema_fast)
    ema_slow_values.append(ema_slow)



    apo = ema_fast - ema_slow  # Calculate APO
    apo_values.append(apo)
```

Similar to what was discussed previously, the trading signal is used for position management as well. The logic of the selling strategy can be summarized as follows:

```
# Define conditions for sell trade
sell_condition_1 = apo > APO_VALUE_FOR_SELL_ENTRY * stdev_factor and
abs(close_price - last_sell_price) > MIN_PRICE_MOVE_FROM_LAST_TRADE *
stdev_factor
sell_condition_2 = position > 0 and (apo >= 0 or open_pnl >
MIN_PROFIT_TO_CLOSE / stdev_factor)
```

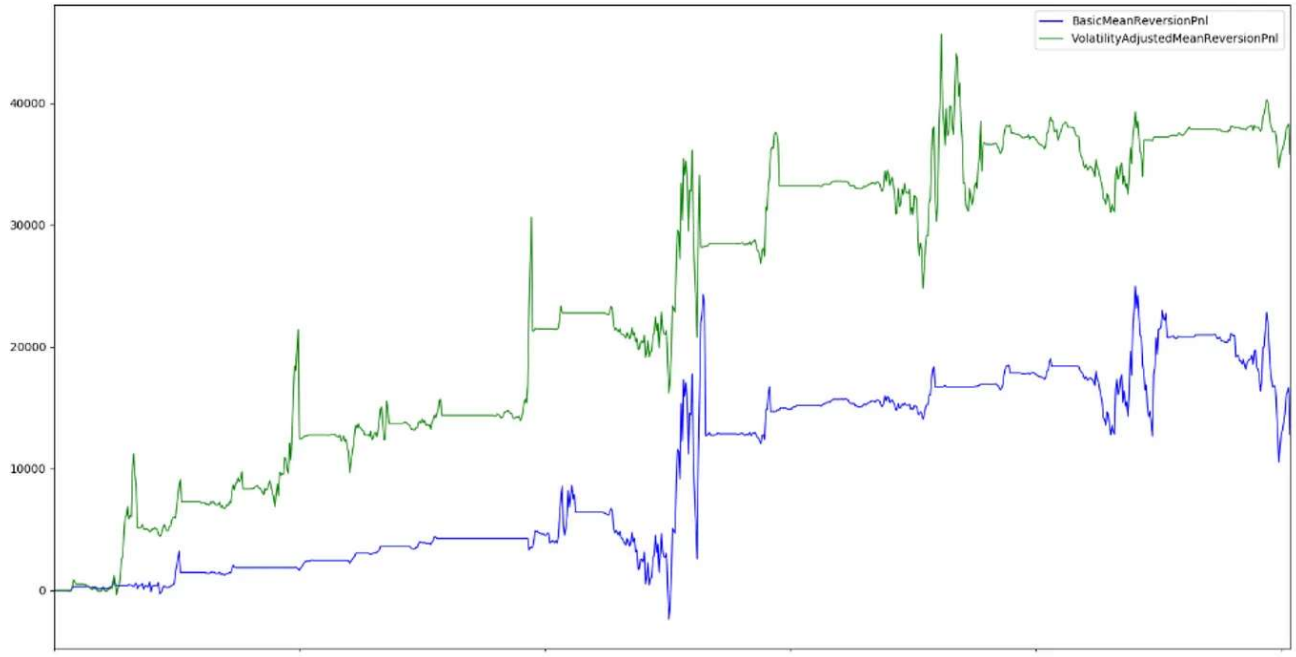
```
# Execute a sell trade if the conditions are met
if sell_condition_1 or sell_condition_2:
    orders.append(-1)  # Mark the sell trade
    last_sell_price = close_price
    position -= NUM_SHARES_PER_TRADE  # Update position
    sell_sum_price_qty += close_price * NUM_SHARES_PER_TRADE  # Update vwap
sell-price
    sell_sum_qty += NUM_SHARES_PER_TRADE
    print(f"Sell {NUM_SHARES_PER_TRADE} @ {close_price}, Position:
{position}")
```

We can now examine comparable reasoning for buying trades.

```
# Define conditions for buy trade
buy_condition_1 = apo < APO_VALUE_FOR_BUY_ENTRY * stdev_factor and
abs(close_price - last_buy_price) > MIN_PRICE_MOVE_FROM_LAST_TRADE *
stdev_factor
buy_condition_2 = position < 0 and (apo <= 0 or open_pnl >
MIN_PROFIT_TO_CLOSE / stdev_factor)
```

```
# Execute a buy trade if the conditions are met
if buy_condition_1 or buy_condition_2:
    orders.append(1)  # Mark the buy trade
    last_buy_price = close_price
    position += NUM_SHARES_PER_TRADE  # Update position
    buy_sum_price_qty += close_price * NUM_SHARES_PER_TRADE  # Update vwap
buy-price
    buy_sum_qty += NUM_SHARES_PER_TRADE
    print(f"Buy {NUM_SHARES_PER_TRADE} @ {close_price}, Position:
{position}")
else:
    # If neither sell nor buy conditions were met, do not trade
    orders.append(0)
```

The profits and losses of a strategy based on fixed mean reversion thresholds will be compared with the results of a strategy that compensates for volatility. The comparison will enable us to determine whether or not we have improved our performance.



This approach improves the performance of the strategy by an impressive 200% by adapting to market volatility.

Profitable trading using the absolute price oscillator's trading signal.

As an alternative to the mean reversion strategy we examined previously, we can develop a new approach. Our trend-tracking strategy can be built using the APO trading signal. A key difference lies in our entry points: when the APO exceeds a particular threshold, we go long, anticipating that price movements will continue. We go short if the APO falls below a certain threshold, anticipating a further decline in prices.

In general, this approach to trading differs very little from the previous strategy, though it does have slight differences in position management. Although initial expectations were negative, this trading method does not produce the opposite results. A trend-following and mean reversion strategy can both produce profits in similar market conditions.

In order to make any decision regarding whether to buy or sell, we must first determine the APO values. Our buying criteria specifically look for positive APO thresholds. Alternatively, when selling, we strive for a negative APO.

```
BUY_ENTRY_APO_THRESHOLD = 10 # The APO trading signal value needed to
initiate buy-orders or long positions
```

```
SELL_ENTRY_APO_THRESHOLD = -10 # The APO trading signal value required to
initiate sell-orders or short positions
```

Here are some tips on how to determine when to enter and exit a trade using the fundamental trading strategy.

Initially, examine the code responsible for handling signals and managing positions, ultimately resulting in sell trades.

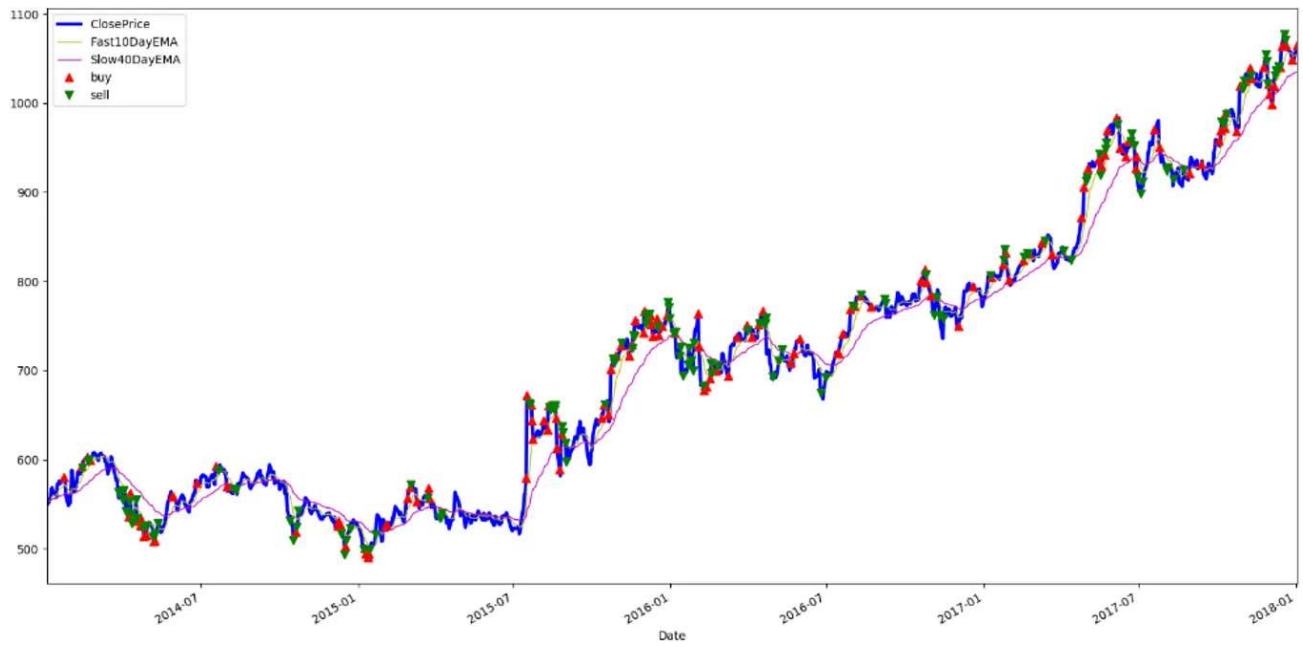
```
if ((apo < APO_VALUE_FOR_SELL_ENTRY and abs(close_price - last_sell_price) >
MIN_PRICE_MOVE_FROM_LAST_TRADE) # APO above sell entry threshold, we should
sell
    or
    (position > 0 and (apo <= 0 or open_pnl > MIN_PROFIT_TO_CLOSE))): # long
from positive APO and APO has gone negative or position is profitable, sell
to close position
    orders.append(-1) # mark the sell trade
    last_sell_price = close_price
    position -= NUM_SHARES_PER_TRADE # reduce position by the size of this
trade
    sell_sum_price_qty += (close_price*NUM_SHARES_PER_TRADE) # update vwap
sell-price
    sell_sum_qty += NUM_SHARES_PER_TRADE
    print( "Sell ", NUM_SHARES_PER_TRADE, " @ ", close_price, "Position: ",
position )
```

In this article, we'll examine the code responsible for managing signals and positions that ultimately result in buying trades being executed.

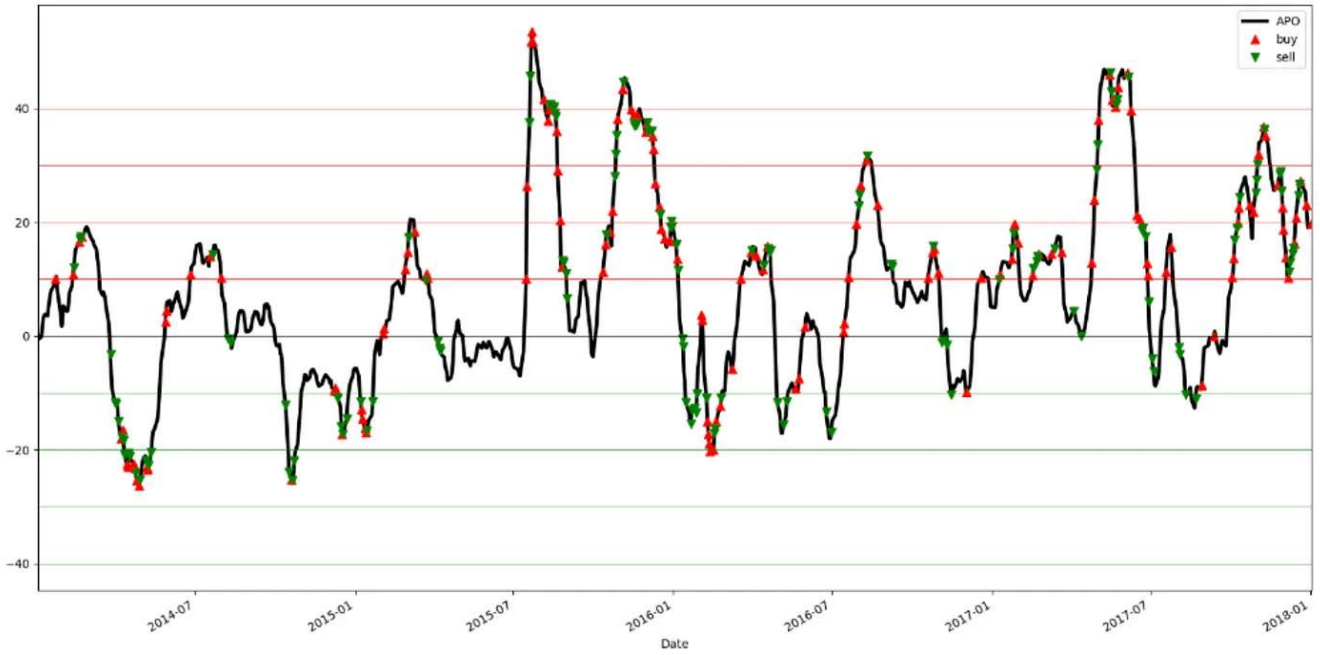
```
elif ((apo > APO_VALUE_FOR_BUY_ENTRY and abs(close_price - last_buy_price) >
MIN_PRICE_MOVE_FROM_LAST_TRADE)
    or
    (position < 0 and (apo >= 0 or open_pnl > MIN_PROFIT_TO_CLOSE))):



    orders.append(+1)
    last_buy_price = close_price
    position += NUM_SHARES_PER_TRADE
    buy_sum_price_qty += (close_price*NUM_SHARES_PER_TRADE)
```

```
        buy_sum_qty += NUM_SHARES_PER_TRADE
        print("Buy ", NUM_SHARES_PER_TRADE, " @ ", close_price, "Position: ",
    position)


    else:
        orders.append(0)
```

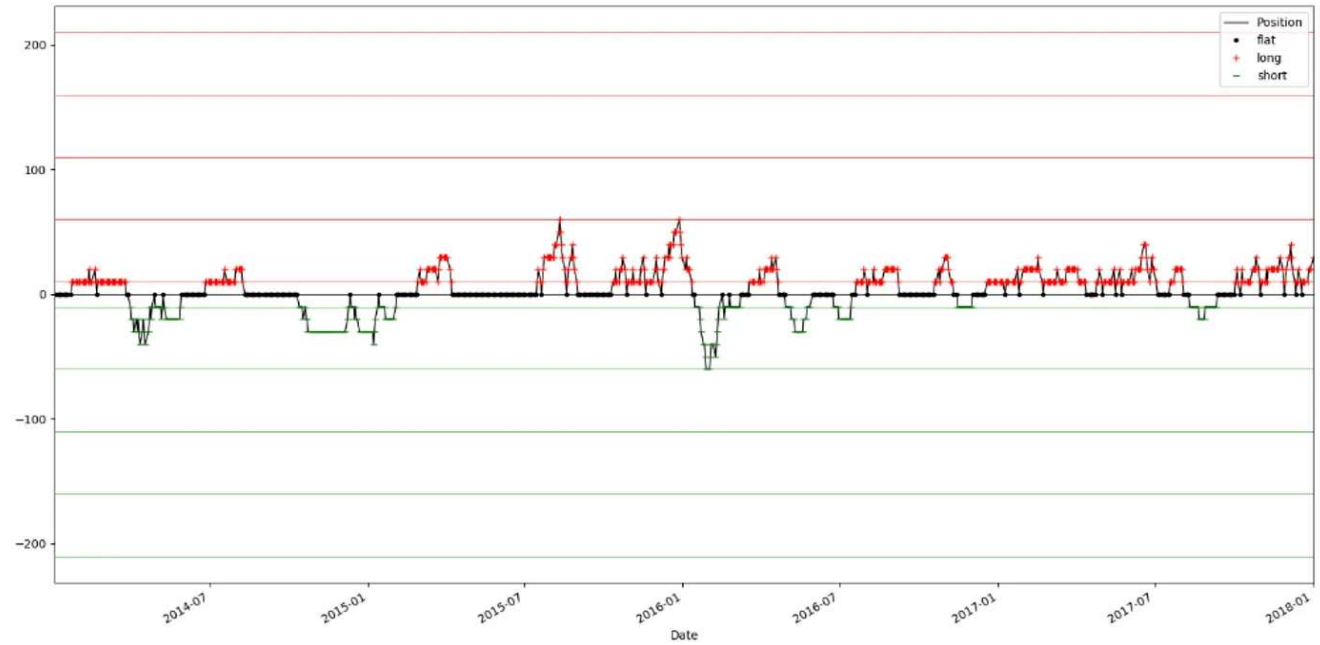
Let's look at the performance of trend-following trading strategies without going into the details of the plot generation code.



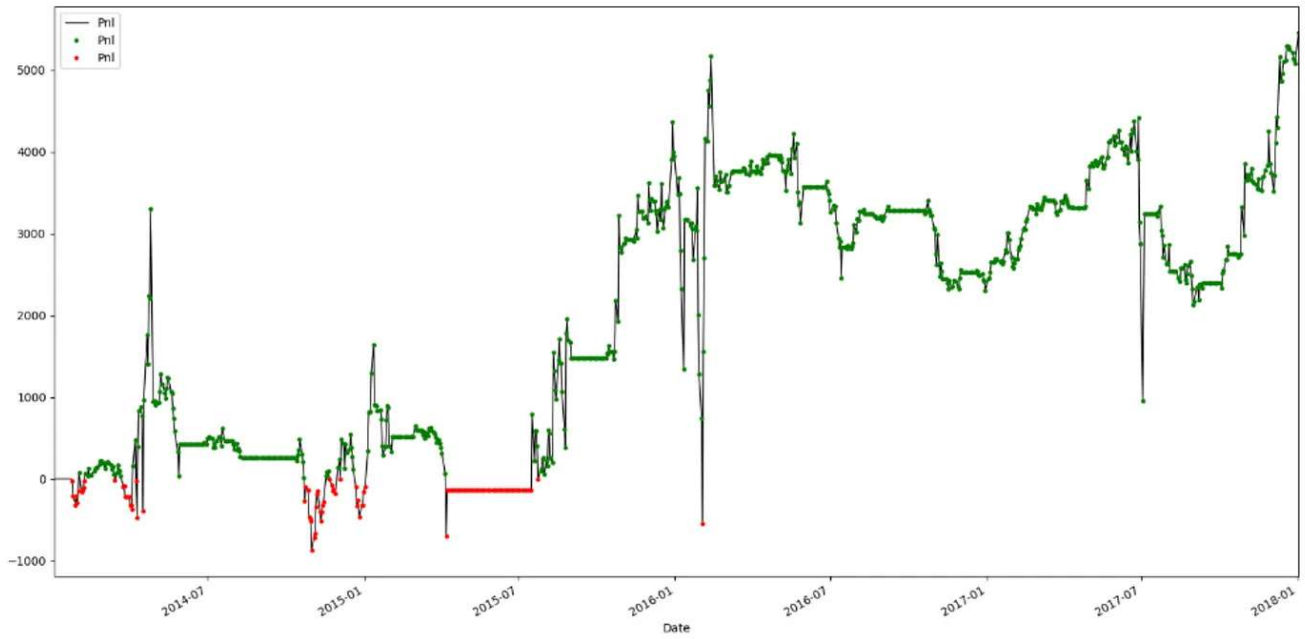
A focus on Google stock is presented in this narrative, illustrating how the price of trades changes throughout the lifetime of the trading strategy. Analysis of the APO signal values and the actual trade prices is crucial to fully understanding the trading strategy. Our following plot will delve deeper into this topic.



Trend-following strategies use APO trading signal values to determine whether favorable conditions exist for buying and selling trades are possible. Positive APO values indicate favorable trading conditions for buying, while negative APO values indicate possible selling opportunities. The interesting thing is that there are instances in which buying trades are made with negative APO values, and selling trades are made with positive APO values. In reality, this is a means of closing profitable positions, similar to mean reversion. Let's examine how positions develop throughout this trading strategy.



More often than not, this strategy involves buying. As a rule, the positions taken are small and closed rapidly, followed by opening new positions (often buying). As a result, this approach follows trends, especially when applied to assets such as Google stock. The upward trend of Google stocks tends to make most positions buy-oriented and yield profits before rapidly closing. As we move forward, let's examine how this trading strategy performed:



In this scenario, the trend-following approach generates approximately one-third of the profits the mean reversion strategy achieves. The trend-following strategy generates profits despite similar market conditions by effectively timing entry and exit points at varying prices.

In response to shifting volatility, this strategy adapts to and embraces trends.

Our volatility indicator, STDEV, can serve as a guide for estimating volatility. Market volatility can be seamlessly accounted for by tweaking our trend-tracking strategy. Adapting mean reversion trading strategies for volatility can be done similarly.

To summarize the given text without repeating terms, I've rewritten it as follows: The primary trading approach for the trend-tracking tactic, adapted to market fluctuations, is as follows: Here is the logic behind the selling process, which is governed by the trading approach.

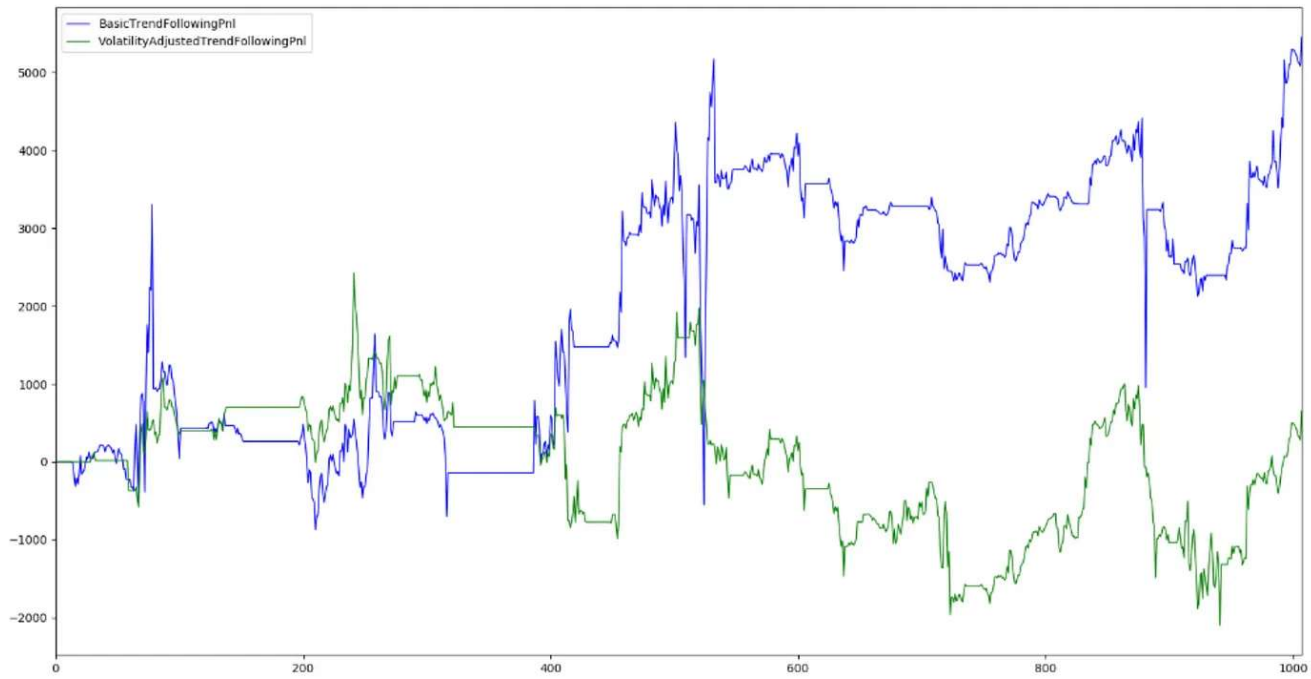
```
if ((apo < APO_VALUE_FOR_SELL_ENTRY/stdev_factor and abs(close_price -
last_sell_price) > MIN_PRICE_MOVE_FROM_LAST_TRADE*stdev_factor)
```

```
    or
    (position > 0 and (apo <= 0 or open_pnl >
MIN_PROFIT_TO_CLOSE/stdev_factor))):



    orders.append(-1)
    last_sell_price = close_price
    position -= NUM_SHARES_PER_TRADE
    sell_sum_price_qty += (close_price*NUM_SHARES_PER_TRADE)
    sell_sum_qty += NUM_SHARES_PER_TRADE
    print("Sell ", NUM_SHARES_PER_TRADE, " @ ", close_price, "Position: ",
position)
```

Within the trading logic, we will examine the code for executing buy trades.

```
elif ((apo > APO_VALUE_FOR_BUY_ENTRY/stdev_factor and abs(close_price -
last_buy_price) > MIN_PRICE_MOVE_FROM_LAST_TRADE*stdev_factor)
    or
    (position < 0 and (apo >= 0 or open_pnl >
MIN_PROFIT_TO_CLOSE/stdev_factor))):



    orders.append(+1)
    last_buy_price = close_price
    position += NUM_SHARES_PER_TRADE
    buy_sum_price_qty += (close_price*NUM_SHARES_PER_TRADE)
    buy_sum_qty += NUM_SHARES_PER_TRADE
    print("Buy ", NUM_SHARES_PER_TRADE, " @ ", close_price, "Position: ",
position)



else:
    orders.append(0)
```

The performance of a strategy that follows trends will now be assessed, as well as the impact of volatility fluctuations and their absence.



With flexible trading thresholds, trend-following strategies lose effectiveness. The volatility measure can, however, be modified in order to uncover alternative approaches as opposed to rigid trend-following strategies that may improve performance.

The art of navigating economic occurrences

A different category of trading methods is discussed in this segment. As an alternative to technical indicators, we can explore economic disclosures and assess the potential impact on trading instruments using a range of economic reports. These estimations can then be used to make informed trades. It is important to understand the nature of economic releases and how they impact instrument prices before we move forward.

Insights into the financial landscape are provided by the unveiling of economic data, which is always an exciting event. Reports like these provide insight into the economy's current state and assist in providing a picture of what's going on. These information releases play an important role in shaping our understanding of the financial climate, from inflation rates to employment figures. Economic fluctuations can be turbulent, but they act as a compass for businesses, investors, and government officials. We can see the economic tapestry around us through the fluctuation of consumer spending or the trajectory of manufacturing output. Decoding these intricate details gives us a better understanding of the economic forces at work, allowing us to adjust our strategies and make informed decisions. A successful economy depends on economic releases, the breadcrumbs that guide us towards prosperity.

A measure of economic activity measures the amount of economic activity in a specific country, region, or asset. The research and measures are provided by a variety of entities, including government agencies and private research firms. The indicators are accessible by regularly scheduled releases, outlined in the economic calendar. There are ample historical, projected, and actual data available. Economic activity is captured by various economic measures. In some cases, it affects housing prices, and in others, it reveals employment statistics. The impact of certain measures varies depending on the commodity—grain, corn, and wheat, for instance, while precious metals and energy resources are affected by others. Among widely recognized economic indicators is US labor department's Nonfarm Payrolls data (https://www.bls.gov/ces/). Non-agricultural jobs are the number of new jobs created. In a broad array of asset classes, this measure resonates. The Energy Information Administration also publishes a report on Crude Oil Stockpiles on a weekly basis. The index measures fluctuations in crude oil availability. Stocks and interest rates are only indirectly affected by this release, but the release has a significant effect on energy-related products such as oil and gas.

After learning how economic indicators are represented and understand their meaning, let's examine a compact list of noteworthy disclosures in the United States. It would be worthwhile for readers to explore in depth both the mentioned economic indicators and additional ones mentioned here, although we won't be going into specifics.

A few of the economic indicators and indices used in financial trading are listed below. ADP Employment, API Crude, and Balance of Trade data will be examined first. An economy can be analyzed by analyzing its import-export balance, employment trends, and crude oil production. Next, we will investigate Baker Hughes's Oil Rig Count, Business Optimism, and Business Inventory. A number of indexes are based on oil rig activity, optimism in business, and stocks of goods for future use. The Leading Index of CB as well as the Case-Shiller Index all show the importance of consumer confidence. Moreover, these statistics can be used to predict economic growth and to determine real estate trends. Challenger's job cuts will be discussed next week, along with Chicago PMI and spending on construction. These indices track various metrics, including construction spending, layoffs by US employers, and the manufacturing sector in Chicago. Afterward, we will examine consumer credit, inflation expectations, and durable goods orders. These indices also measure new orders for manufacturers' durable goods, consumer debt, and inflation expectations. EIA Crude Oil, EIA Natural Gas, and Empire State Manufacturing will be explored next. The New York Manufacturing Sector Statistics and Energy Inventory are available here. The Employment

Cost Index, Factory Orders, and Fed Beige Book will follow. As a result of such metrics, we can gain insight into labour costs, new orders received by manufacturers, and the New York Fed's 12 district economic conditions survey. Afterwards, a press conference and the manufacturing index will be released. This index indicates manufacturing activity, as well as hints from Fed press conferences, as well as providing insight into monetary policy. Aside from analyzing FOMC minutes and the Fed's national activity reports, our team will also analyze economic forecasts from the FOMC. You can find forecasts, minutes from the 2010 meeting, and detailed information on economic activity from these sources. The GDP, housing starts, and home sales should be monitored. As well as providing insight into the economy and home sales, they also provide information on residential construction. House price indexes, import prices, and industrial production will be analysed. Data on house prices, import costs, and mining, manufacturing, and utilities production are also available from these sources. Considering inflation, ISM Manufacturing, and ISM Non-Manufacturing will be covered next. By using these metrics, we can determine the intensity of inflation, the health of manufacturing, and the health of non-manufacturing. Then we'll discuss the ISM New York Index, Jobless Claims, and JOLTs.

Additionally, the economic status of New York, unemployment rates, and job openings can be calculated. Also worth considering is the Markit Composite PMI. There is also a Markit Manufacturing PMI and a Michigan Consumer Sentiment Index. Business data is provided on various topics, such as manufacturing data, consumer data, and data on the private sector. Payroll data will also be analyzed along with the NAHB Housing Market Index.

Additionally, these metrics contain information about farmworkers, mortgage loan applications, housing market conditions, and employment. The Nonfarm Productivity Index, the PCE, and the PPI follow. In this way, statisticians can gain insight into how nonfarm sectors are doing, how consumers are spending, and how domestic product prices are changing. Redbook sales, personal spending, and retail sales follow. Their measurements include retail sales, consumer purchases, and consumer expenditures. We also examine WASDE and wholesale inventories in addition to total vehicle sales. Based on these data, we can estimate domestic vehicle sales, supply and demand for agricultural commodities, and wholesale inventories.

Economic release format

Economic release calendars, both free and paid, can be used for historic release data or via an API. We will skip over the issues involved with accessing historical data, since our main focus

is on using economic release data for trading. It is typical for economic release calendars to look like this:

Calendar	CST	Economic indicator	Actual	Previous	Consensus	Forecast
2019-05-03	07:30 AM	Non Farm Payrolls Apr	263K	189K	185K	178K
2019-06-07	07:30 AM	Non Farm Payrolls May	75K	224K	185K	190K
2019-07-05	07:30 AM	Non Farm Payrolls Jun	224K	72K	160K	171K
2019-08-02	07:30 AM	Non Farm Payrolls Jul	164K	193K	164K	160K

Arrangements are made in advance for release dates and times. It is also common to find calendars that provide information about past releases, often from preceeding months or years. Various economists or firms predict the release value based on consensus estimations. Price fluctuations can occur when this expected value differs from reality. There are many calendar providers that offer a Forecast category, which represents the provider's forecast for the economic release.

Electronic economic release services

Having a sound understanding of how trading strategies and corresponding servers communicate economic releases and price movements is vital before diving into the analysis of economic releases and price movement. Rapid and direct connections are offered by several service providers for electronic delivery. Economic indicators provided by these providers are typically available in formats compatible with automated trading strategies. Within a few microseconds to milliseconds after their release, these releases are received by the trading servers. In this day and age, algorithmic traders often use economic release providers as supplementary data sources, which contribute significantly to improving their trading performance.

Economic releases in trading

Let's look at some potential trading strategies that can be influenced by economic indicators, their scheduling, and their electronic delivery to trading servers now that we have a solid understanding of them. Economic releases can be incorporated into algorithmic trading in a number of ways, but we'll explore the most popular and intuitive approach. Our analysis of past data on expected and actual economic values reveals a correlation between these disparities and subsequent price movements. In general, there are two strategies to consider. Taking advantage of discrepancies between expected and actual economic indicators, the first strategy takes advantage of price movements that fall short of expectations. Historical analysis suggests that there should have been a greater movement in prices in such cases, presenting a profitable opportunity. A trend-following strategy is somewhat similar to this approach.

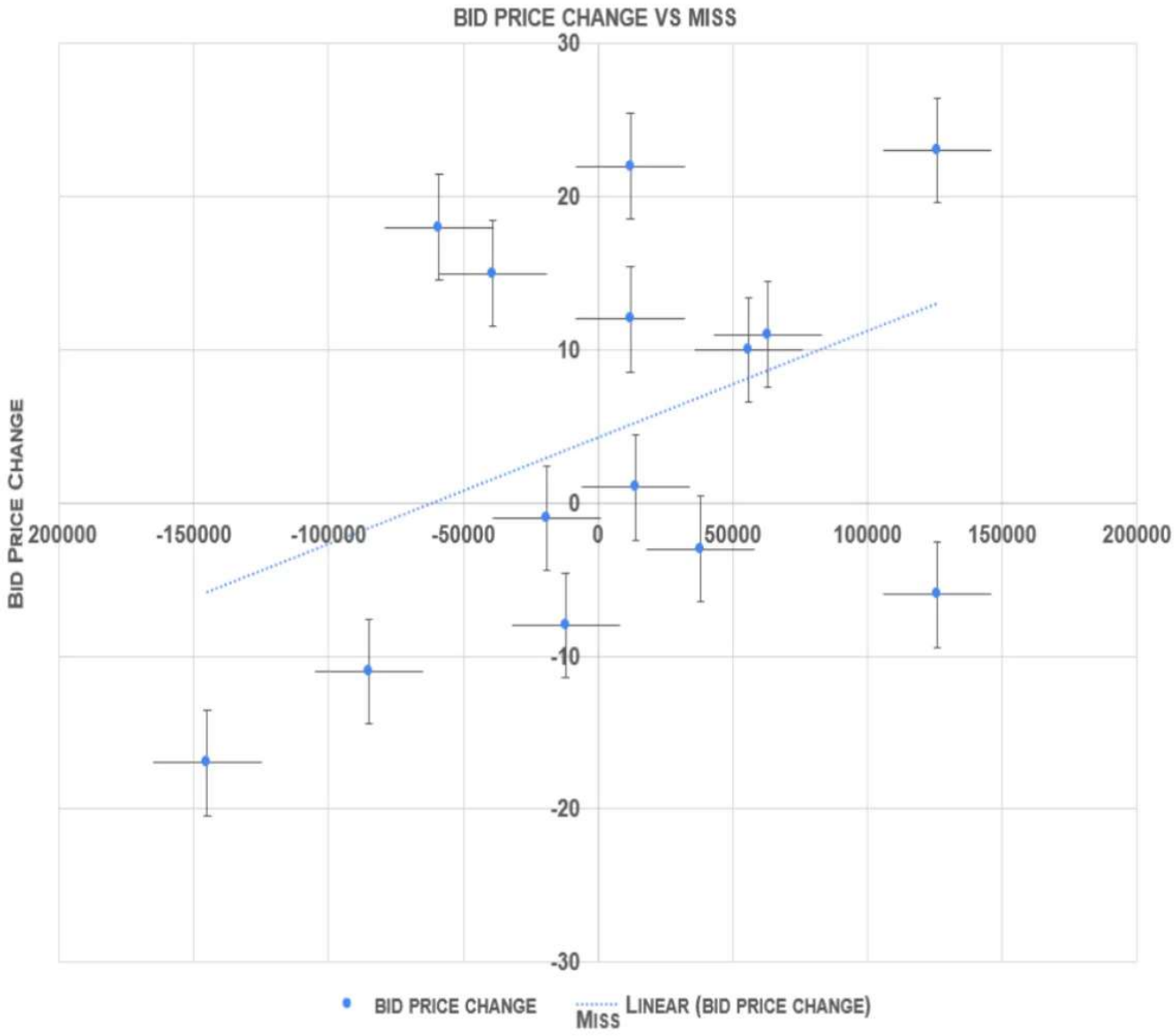
An alternative method identifies exaggerated movements in prices and places a counter-bet. A balancing strategy assumes that prices will return to their previous levels. As we discussed in the previous section, "Predicting the Markets with Basic Machine Learning," this approach is often enhanced by using classification techniques. In order to merge two or more economic indicators that occur at the same time, we need to employ classification methods. As a result, these methods achieve greater precision and specificity by providing more detailed thresholds and boundaries for each indicator. In this example, we will not cover the intricacies of applying classification methods to this particular trading strategy.

Review some recent Non Farm Payroll announcements and how they affected futures for the S&P 500. It would be inappropriate to include the actual analysis code because tick data is

not available for free. This analysis and its application to a variety of datasets should not, however, be difficult to understand.

	A	B	C	D	E	F	G
1	date	time CST	actual	consensus	miss	bid price change	ask price change
2	2019-03-08	7:30:00	25000	170000	-145000	-17	-16
3	2019-06-07	7:30:00	90000	175000	-85000	-11	-11
4	2018-10-05	7:30:00	121000	180000	-59000	18	18
5	2018-12-07	7:30:00	161000	200000	-39000	15	16
6	2018-08-03	7:30:00	170000	189000	-19000	-1	-1
7	2019-08-02	7:30:00	148000	160000	-12000	-8	-8
8	2019-04-05	7:30:00	182000	170000	12000	22	23
9	2018-07-06	7:30:00	202000	190000	12000	12	12
10	2018-09-07	7:30:00	204000	190000	14000	1	1
11	2019-07-05	7:30:00	191000	153000	38000	-3	-2
12	2019-05-03	7:30:00	236000	180000	56000	10	10
13	2018-11-02	7:30:00	246000	183000	63000	11	11
14	2019-01-04	7:30:00	301000	175000	126000	-6	-6
15	2019-02-01	7:30:00	296000	170000	126000	23	23

A scatter plot can provide an intuitive visual representation of price fluctuations and inaccuracies in economic indicator disclosures.



Price increases when actual indicator values surpass expected values, as can be seen in the chart above. A price decrease occurs when actual values fall short of expectations. S&P index value generally increases when NonFarm Payroll job additions are significant, thus indicating a strong economy and a healthy economy. In addition, prices are affected by larger deviations from expectations. In this way, we can anticipate two important factors: direction and magnitude of price movements induced by deviations. Now let's look at how this knowledge can be utilized.

Utilizing the absence of data and conducting research is one way to go about this. With the help of this research, we can adopt a trend-following approach, where we buy assets when there is a significant lack of data and sell assets when there is a significant lack of data. When our anticipated price movement materializes, we will close either a long or short position, as

we expect it to. If our research matches the observed price movement, this strategy effectively succeeds. It is also important to take into account the time delay between the release of information and the commencement of price fluctuations. We need to initiate positions quickly enough to take advantage of being able to access this information before other participants gain access to it and before the price movement subsides.

As an alternative, one can utilize research on prices and the absence of expected values to identify exaggerated changes and then take a contrary stance. In the absence of an upward deviation from our predictions, when the price declines instead, we interpret the movement as an error, urging us to assume prices will rise, as our research demonstrates. In another instance of overreaction, prices rise following a positive deviation that aligns with our research findings. However, the magnitude of the increase exceeds our expectations. Rather than taking a negative stance early, our strategy waits until prices have deviated significantly from our projections, expecting the overreaction to diminish and prices to readjust slightly, allowing us to make profits. Compared to trend-following strategies, the mean-reversion trading strategy has a reduced sensitivity to timing discrepancies between economic indicators and the specified time frame for initiating trading positions, which is an advantage over trend-following strategies.

Understanding and implementing basic statistical arbitrage trading strategies

It was in the 1980s that Statistical arbitrage trading strategies (StatArb) became extremely popular due to their impressive returns. In StatArb, related items are analyzed to identify connections between temporary price changes. This strategy focuses on statistically significant relationships found in earlier research in order to predict price movements in specific trading instruments, taking into account the behavior of a wide variety of related products.

StatArb takes positions in related products like pair trading. In spite of this, there are notable distinctions to be considered. A StatArb account often includes a variety of trading instruments that differ from pairs trading. Instruments like futures, equities, options, and even currencies can be used. With such broad diversification, StatArb strategies have a unique characteristic. A further advantage of StatArb is that it combines mean reversion with trend-following. As an example, suppose the price deviation of a single trading instrument is lower than the price deviations of the entire instrument portfolio. StatArb strategies mimic trend-following strategies in this case. By positioning themselves for price alignment, they can

"catch up" with the portfolio. In StatArb, these strategies interplay to produce a versatile and sophisticated trading strategy.

Another scenario is when the observed price variation exceeds the expected deviation based on the correlation between the portfolio price fluctuations and the traded instrument. StatArb strategies employ a similar approach to mean reversion strategies in anticipating the price of the traded instrument will return to its portfolio level. In a variety of situations, StatArb trading strategies frequently use mean reversion strategies. As long as positions last for more than milliseconds or a few seconds, StatArb strategies can be classified as high-frequency trading (HFT) and medium-frequency trading (MFT).

Market participants' reactions lag behind the trading instrument in this specific approach, since the portfolio takes the lead. When such expectations are not met, for instance, when the trading instrument we intend to trade drives price movements across the portfolio, this strategy fails to produce positive results. Rather than the trading instrument's price catching up to the portfolio's price, it is now the portfolio price that catches up to the trading instrument's. According to this principle, lead-lag in StatArb, profit is obtained by identifying predominantly lagging trading instruments and building a portfolio primarily composed of leading instruments.

Trading dynamics can change across market hours, as we often see in different trading instruments. A particular instrument may take the lead sometimes, while at other times the reverse may occur. During Asia market hours, it is reasonable to assume that trading instruments on Asian electronic exchanges like Singapore, India, Hong Kong, and Japan guide global asset price movements. Additionally, instruments traded in Germany, London, and other European countries tend to be the most active during European market hours. The price movement during American market hours is determined by instruments traded in the US. Accordingly, the optimal trading strategy comprises tailoring portfolios and establishing distinct relationships between leading and lagging instruments.

Your investment portfolio should be adapted to your investment mix and connections

As portfolio structures and connections among various trading instruments evolve, it is essential to establish methods for adapting StatArb strategies consistently to these changes. StatArb's trading strategy relies on short-term relationships between dozens of instruments to make decisions, which is challenging due to the difficulty of comprehending and adapting to

fluctuations in price across diverse instruments. In addition, these instruments' weights change over time. An approach to this is to use principal component analysis, a technique for reducing dimensions, which can be used to develop, adapt, and track the portfolio's significance and weight as time goes on.

Furthermore, it is important to consider the links established between the trading instrument and the key instruments, and between the trading instrument and the assortment of key instruments. There are times when localized volatility and country-specific economic events disrupt the essential correlation for profitable StatArb trading. Brazilian real currency movements can be detracted by political or economic conditions in Brazil. Similarly, events such as Brexit in Britain or trade wars with China in America can disrupt both portfolio relationships and lead-lag dynamics, ultimately resulting in a loss of profitability for StatArb customers. The ability to cope with such circumstances may require additional statistical advantages and a higher level of sophistication beyond the capabilities of StatArb.

Costs associated with StatArb strategies' foundational elements

Trading StatArb involves one crucial aspect to keep in mind. Connecting with numerous electronic trading exchanges is essential to thrive in this business. Market data from various countries, continents, and markets can be found through these connections. Co-locating in numerous trading exchanges can, however, be financially burdensome due to the substantial infrastructure costs involved. Additionally, software development requires significant investments in addition to connections. As well as receiving, interpreting, and storing market data, this investment is necessary. Since the market data feed and order gateway communication formats of different exchanges differ dramatically, adapting to these differences is a critical task.

Ultimately, StatArb approaches require access to market data from all trading platforms. The result is that each exchange now requires a tangible data connection with every other exchange, resulting in escalating costs. Additionally, utilizing pricier microwaves for data delivery to trading systems will only exacerbate the problem. It is important to note that StatArb trading strategies are considerably more expensive when it comes to infrastructure when compared to alternative trading methods.

StatArb trading strategy in Python

Following our understanding of StatArb trading strategies and our considerations of practical aspects of establishing and managing an algorithmic trading venture, let's examine an actual

execution of a trading strategy and figure out how it works. A low-level programming language such as C++ is often employed by high-frequency algorithmic trading businesses.

It is essential to obtain comprehensive data on multiple major currencies worldwide before implementing StatArb trading strategies. In this article, we will compare the Australian Dollar with the US Dollar (AUD/USD). A comparison between the Australian Dollar and the American Dollar is presented here. Our next section will compare the British Pound with the US Dollar (GBP/USD). Significant value fluctuations have been experienced by both currencies in the past. Comparing the Canadian and American Dollars gives us valuable insights into their interaction. CHF/USD provides an indication of the Swiss Franc's strength against the American Dollar. In order to determine the value of the Euro, the Euro can be compared to the American Dollar (EUR/USD). JPY/USD (Japanese Yen versus American Dollar) needs to be fully evaluated. After that, we'll compare the Kiwi with the US dollar. New Zealand Dollars tend to be cheaper than American Dollars.

As an example of the StatArb trading approach in practice, we will analyze the correlations between the currency pair CAD/USD and other currency pairs.

Our data frame for these currency pairs will span 4 years.

```
import pandas as pd
from pandas_datareader import data as pdr



TRADING_INSTRUMENT = 'CADUSD=X'
SYMBOLS = ['AUDUSD=X', 'GBPUSD=X', 'CADUSD=X', 'CHFUSD=X', 'EURUSD=X',
'JPYUSD=X', 'NZDUSD=X']
START_DATE = '2014-01-01'
END_DATE = '2018-01-01'



symbols_data = {}



# Fetch data for each symbol
for symbol in SYMBOLS:
    data_file_path = f"{symbol}_data.pkl"
```

```
    # Attempt to read local file. If not existent, fetch data from Yahoo
Finance
    try:
        symbol_data = pd.read_pickle(data_file_path)
    except FileNotFoundError:
        symbol_data = pdr.get_data_yahoo(symbol, start=START_DATE,
end=END_DATE)
        symbol_data.to_pickle(data_file_path)

    symbols_data[symbol] = symbol_data
```

Explore every currency pair within our dataset by visualizing their price movements. There are fascinating patterns to be observed. Our visualization of the JPY/USD pair is magnified by a factor of 100 for the purpose of aiding our understanding.

```
import matplotlib.pyplot as plt
import numpy as np



price_data = pd.DataFrame()



for symbol in SYMBOLS:
    multiplier = 100.0 if symbol == 'JPYUSD=X' else 1.0



    label = f'{symbol} ClosePrice'
    price_data[label] = symbols_data[symbol]['Close'] * multiplier



# Set the color cycle
colors = ['b', 'g', 'r', 'c', 'm', 'k', 'y']
plt.gca().set_prop_cycle('color', colors)



# Plot each column with its label
for column in price_data:
```

```
    plt.plot(price_data.index, price_data[column], lw=2., label=column)


plt.xlabel('Date', fontsize=18)
plt.ylabel('Scaled Price', fontsize=18)
plt.legend(prop={'size': 18})
plt.show()
```

Executing the code will lead to the intended result. We'll now examine its storyline.

Currency pairs display varying levels of similarity in their price movements, as predicted and evident. In terms of correlation between CAD/USD, AUD/USD, and NZD/USD, the correlation appears to be the strongest, while CHF/USD and JPY/USD demonstrate the lowest correlation. We will incorporate all currencies into the trading model with the understanding that these connections are unpredictable.

The next step is to establish and measure the factors necessary for calculating and monitoring correlations, as well as the variations from moving averages.

```
import statistics as stats
```

```
SMA_NUM_PERIODS = 20
price_history = {}



PRICE_DEV_NUM_PRICES = 200
price_deviation_from_sma = {}



num_days = len(symbols_data[TRADING_INSTRUMENT].index)
correlation_history = {}
delta_projected_actual_history = {}



final_delta_projected_history = []
```

Defining StatArb trading parameters

An effective StatArb trading approach must be based on certain ultimate variables and thresholds before diving into the core strategy loop.

```
import statistics as stats



# Define the necessary constants and containers
SMA_NUM_PERIODS = 20
price_history = {}
PRICE_DEV_NUM_PRICES = 200
price_deviation_from_sma = {}
num_days = len(symbols_data[TRADING_INSTRUMENT].index)
correlation_history = {}
delta_projected_actual_history = {}
final_delta_projected_history = []



# Containers for order and position tracking
orders = []
```

```
positions = []
pnls = []



# Trade management variables
last_buy_price = 0
last_sell_price = 0
position = 0
buy_sum_price_qty = 0
buy_sum_qty = 0
sell_sum_price_qty = 0
sell_sum_qty = 0
open_pnl = 0
closed_pnl = 0



# Strategy parameters
StatArb_VALUE_FOR_BUY_ENTRY = 0.01
StatArb_VALUE_FOR_SELL_ENTRY = -0.01
MIN_PRICE_MOVE_FROM_LAST_TRADE = 0.01
NUM_SHARES_PER_TRADE = 1000000
MIN_PROFIT_TO_CLOSE = 10
```

Quantifying and computing StatArb trading signals

During each day's assessment, we will consider the various options and determine which calculations are needed. Our initial analysis will focus on computing simple moving averages and comparing prices to them.

```
for i in range(num_days):
    close_prices = {}



    # Generate ClosePrice series, calculate SMA for each symbol, and compute
price-deviation from SMA for each symbol
    for symbol in SYMBOLS:
        close_prices[symbol] = symbols_data[symbol]['Close'].iloc[i]
```

```
        # Initialize if symbol not in keys
        if symbol not in price_history:
            price_history[symbol] = []
            price_deviation_from_sma[symbol] = []



        price_history[symbol].append(close_prices[symbol])



        # Maintain most recent prices up to SMA_NUM_PERIODS
        if len(price_history[symbol]) > SMA_NUM_PERIODS:
            price_history[symbol].pop(0)



        # Calculate Simple Moving Average and deviation from it
        sma = stats.mean(price_history[symbol])
        price_deviation_from_sma[symbol].append(close_prices[symbol] - sma)



        # Maintain recent price deviations up to PRICE_DEV_NUM_PRICES
        if len(price_deviation_from_sma[symbol]) > PRICE_DEV_NUM_PRICES:
            price_deviation_from_sma[symbol].pop(0)
```

As a next step, let's calculate the connections between the variations in CAD/USD exchange rates and other currency pairs. The price deviations from the Simple Moving Average (SMA) that we computed earlier will be examined using covariance and correlation. Moreover, we will evaluate how each leading currency pair anticipates the CAD/USD price deviation. It can be determined which deviation is more significant by comparing the projected and actual price deviations. Using these differences we can calculate a final delta value that will help us make trading decisions.

Now let's examine the code block that populates the correlation_history and delta_projected_actual_history dictionaries.

```
    projected_dev_from_sma_using = {}
```

```
for symbol in SYMBOLS:
    if symbol == TRADING_INSTRUMENT: # Skip the comparison with itself
        continue



    correlation_label = TRADING_INSTRUMENT + '<-' + symbol



    # Initialize if first entry for this pair in the history dictionary
    if correlation_label not in correlation_history:
        correlation_history[correlation_label] = []
        delta_projected_actual_history[correlation_label] = []



    # Need at least two observations to compute covariance/correlation
    if len(price_deviation_from_sma[symbol]) < 2:
        correlation_history[correlation_label].append(0)
        delta_projected_actual_history[correlation_label].append(0)
        continue
```

Let’s examine how the covariance and correlation of the different currency pairs are calculated.

```
corr = np.corrcoef(price_deviation_from_sma[TRADING_INSTRUMENT],
price_deviation_from_sma[symbol])
    cov = np.cov(price_deviation_from_sma[TRADING_INSTRUMENT],
price_deviation_from_sma[symbol])
    corr_trading_instrument_lead_instrument = corr[0, 1] # get the
correlation between the 2 series
    cov_trading_instrument_lead_instrument = cov[0, 0] / cov[0, 1] # get the
covariance between the 2 series



correlation_history[correlation_label].append(corr_trading_instrument_lead_i
nstrument)
```

In this section, we'll analyze the code chunk that determines the discrepancy between the estimated and real price change and records it in our delta_projected_actual_history assortment.

```
projected_dev_from_sma_using[symbol] = price_deviation_from_sma[symbol][-1]
* cov_trading_instrument_lead_instrument



delta_projected_actual = (projected_dev_from_sma_using[symbol] -
price_deviation_from_sma[TRADING_INSTRUMENT][-1])



delta_projected_actual_history[correlation_label].append(delta_projected_act
ual)
```

With CAD/USD, we can combine projections and actual price deviations into a single signal. Combining predictions from various currency pairs yields this signal. CAD/USD as forecasted by other pairs will be weighed against the correlation strength between CAD/USD and the other pairs. To standardize the result, we must sum each correlation weight in order to determine the delta value. As our trading strategy's foundation, we will use this normalized delta value as our final signal.

```
# Calculate the sum of weights, which is the sum of correlations for each
symbol with TRADING_INSTRUMENT
sum_weights = sum(abs(correlation_history[TRADING_INSTRUMENT + '<-' +
symbol][-1]) for symbol in SYMBOLS if symbol != TRADING_INSTRUMENT)



# Initialize final prediction of price deviation in TRADING_INSTRUMENT
final_delta_projected = 0
close_price = close_prices[TRADING_INSTRUMENT]



# Iterate over symbols to weight projections by correlation
for symbol in SYMBOLS:
```

```
    if symbol != TRADING_INSTRUMENT:
        correlation_label = TRADING_INSTRUMENT + '<-' + symbol
        final_delta_projected += abs(correlation_history[correlation_label]
[-1]) * delta_projected_actual_history[correlation_label][-1]



# Normalize final_delta_projected by sum of weights
final_delta_projected = final_delta_projected / sum_weights if sum_weights
!= 0 else 0



# Append the final delta projection to history
final_delta_projected_history.append(final_delta_projected)
```

StatArb execution logic

As a first step, let's implement the StatArb signal following these steps:

Now that we have calculated StatArb signals, we can construct a trend-following strategy similar to that we discussed previously. Firstly, let's examine the trading logic of sale transactions.

```
# Check if we should execute a sell trade
if (final_delta_projected < StatArb_VALUE_FOR_SELL_ENTRY and abs(close_price
- last_sell_price) > MIN_PRICE_MOVE_FROM_LAST_TRADE) or \
   (position > 0 and open_pnl > MIN_PROFIT_TO_CLOSE):
    orders.append(-1)  # Append -1 to orders list to indicate a sell order
    last_sell_price = close_price  # Update the last sell price
    position -= NUM_SHARES_PER_TRADE  # Update the position after selling
    sell_sum_price_qty += (close_price * NUM_SHARES_PER_TRADE)  # Update the
volume weighted average price for sell
    sell_sum_qty += NUM_SHARES_PER_TRADE  # Update the quantity sold
    print("Sell ", NUM_SHARES_PER_TRADE, " @ ", close_price, "Position: ",
position)
    print("OpenPnL: ", open_pnl, " ClosedPnL: ", closed_pnl, " TotalPnL: ",
(open_pnl + closed_pnl))
```

Let's examine the purchasing process, which is strikingly similar to selling:

```
elif (final_delta_projected > StatArb_VALUE_FOR_BUY_ENTRY and
abs(close_price - last_buy_price) > MIN_PRICE_MOVE_FROM_LAST_TRADE) or \
     (position < 0 and open_pnl > MIN_PROFIT_TO_CLOSE):
    orders.append(+1)
    last_buy_price = close_price
    position += NUM_SHARES_PER_TRADE
    buy_sum_price_qty += (close_price * NUM_SHARES_PER_TRADE)
    buy_sum_qty += NUM_SHARES_PER_TRADE
    print("Buy ", NUM_SHARES_PER_TRADE, " @ ", close_price, "Position: ",
position)
    print("OpenPnL: ", open_pnl, " ClosedPnL: ", closed_pnl, " TotalPnL: ",
(open_pnl + closed_pnl))



else:
    orders.append(0)
positions.append(position)
```

To see how we've managed positions and updated profit and loss (PnL) in the past, let's examine the logic behind it.

```
open_pnl = 0
  if position > 0:
    if sell_sum_qty > 0:
      open_pnl = abs(sell_sum_qty) * (sell_sum_price_qty / sell_sum_qty -
buy_sum_price_qty / buy_sum_qty)

    open_pnl += abs(sell_sum_qty - position) * (close_price -
buy_sum_price_qty / buy_sum_qty)
  elif position < 0:
    if buy_sum_qty > 0:
      open_pnl = abs(buy_sum_qty) * (sell_sum_price_qty / sell_sum_qty -
buy_sum_price_qty / buy_sum_qty)

    open_pnl += abs(buy_sum_qty - position) * (sell_sum_price_qty /
sell_sum_qty - close_price)
  else:

    closed_pnl += (sell_sum_price_qty - buy_sum_price_qty)
    buy_sum_price_qty = 0
    buy_sum_qty = 0
    sell_sum_price_qty = 0
```

```
    sell_sum_qty = 0
    last_buy_price = 0
    last_sell_price = 0



  pnls.append(closed_pnl + open_pnl)
```

StatArb signal and strategy performance analysis

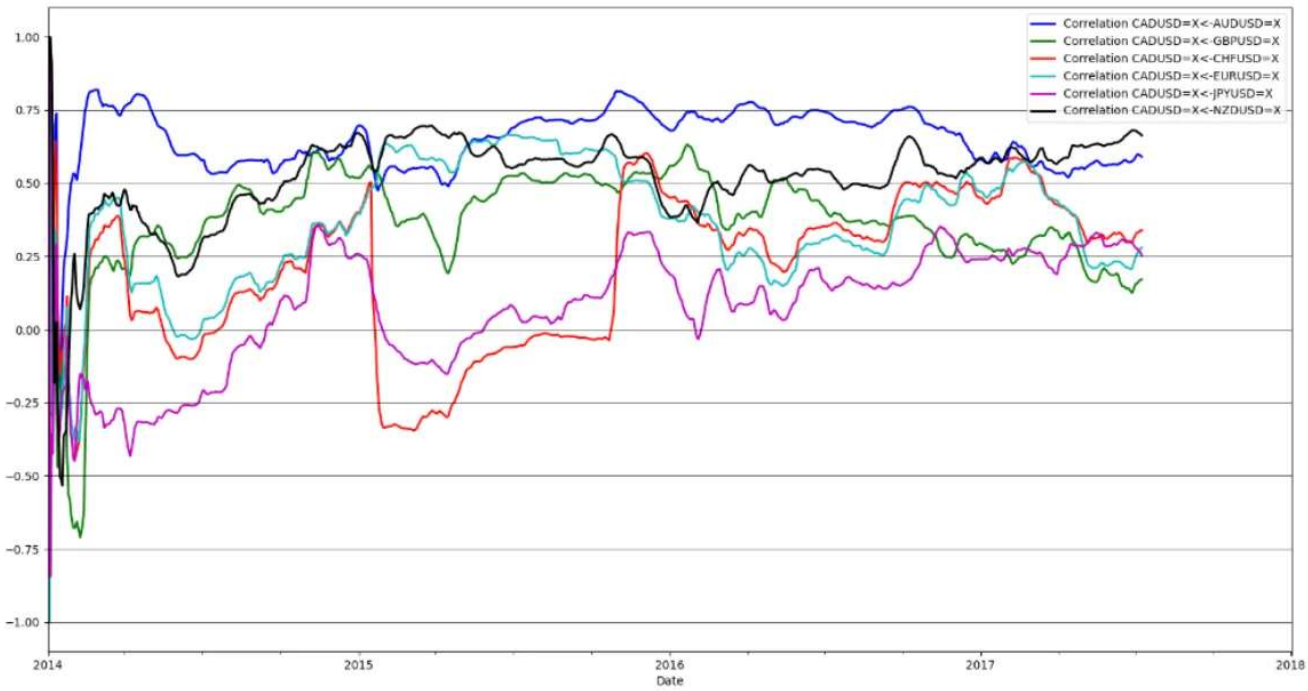
Following are the steps to examine the StatArb signal:

Starting with the connection between CAD/USD and the remaining currency pairs, let's imagine the broader picture of this trading approach.

```
correlation_data = pd.DataFrame()
for symbol in SYMBOLS:
  if symbol == TRADING_INSTRUMENT:
    continue



  correlation_label = TRADING_INSTRUMENT + '<-' + symbol
  correlation_data =
correlation_data.assign(label=pd.Series(correlation_history[correlation_labe
l], index=symbols_data[symbol].index))
  ax = correlation_data['label'].plot(color=next(cycol), lw=2.,
label='Correlation ' + correlation_label)



for i in np.arange(-1, 1, 0.25):
  plt.axhline(y=i, lw=0.5, color='k')
plt.legend()
plt.show()
```

An analysis of the evolving correlation between caduss and various currency pairs over the course of our trading strategy is shown in this plot. Strong correlations between pairs are indicated by correlations near -1 or +1, whereas stable correlations are indicative of pairs that are consistently linked. While currency pairs with fluctuating correlations, varying between

negative and positive values, suggest substantial instability or lack of correlation. Thus, these pairs cannot reliably predict the future. Since we cannot predict how correlations will evolve in the future, we must incorporate all available currency pairs into our StatArb trading strategy.



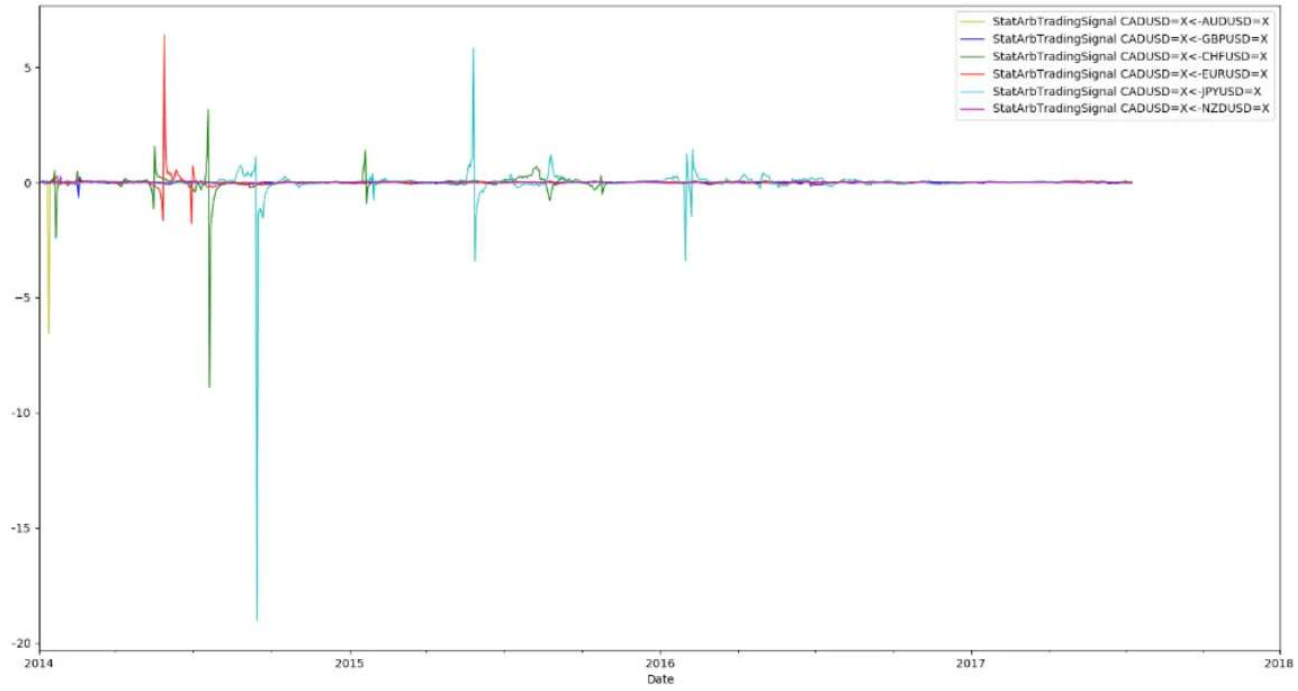
We anticipate that the CAD/USD price deviations will show a strong correlation with the AUD/USD and NZD/USD currency pairs. In contrast, JPY/USD price deviations are least correlated with CAD/USD price deviations.

We can now compare the expected and real price variations in CAD/USD as predicted by each distinct currency pair.

```
delta_projected_actual_data = pd.DataFrame()
for symbol in SYMBOLS:
  if symbol == TRADING_INSTRUMENT:
    continue



  projection_label = TRADING_INSTRUMENT + '<-' + symbol
  delta_projected_actual_data =
delta_projected_actual_data.assign(StatArbTradingSignal=pd.Series(delta_proj
ected_actual_history[projection_label],
index=symbols_data[TRADING_INSTRUMENT].index))
```

```
  ax =
delta_projected_actual_data['StatArbTradingSignal'].plot(color=next(cycol),
lw=1., label='StatArbTradingSignal ' + projection_label)
plt.legend()
plt.show()
```

The following is an example of StatArb's signal values when only using different currency pairs to predict price fluctuations in CAD/USD:

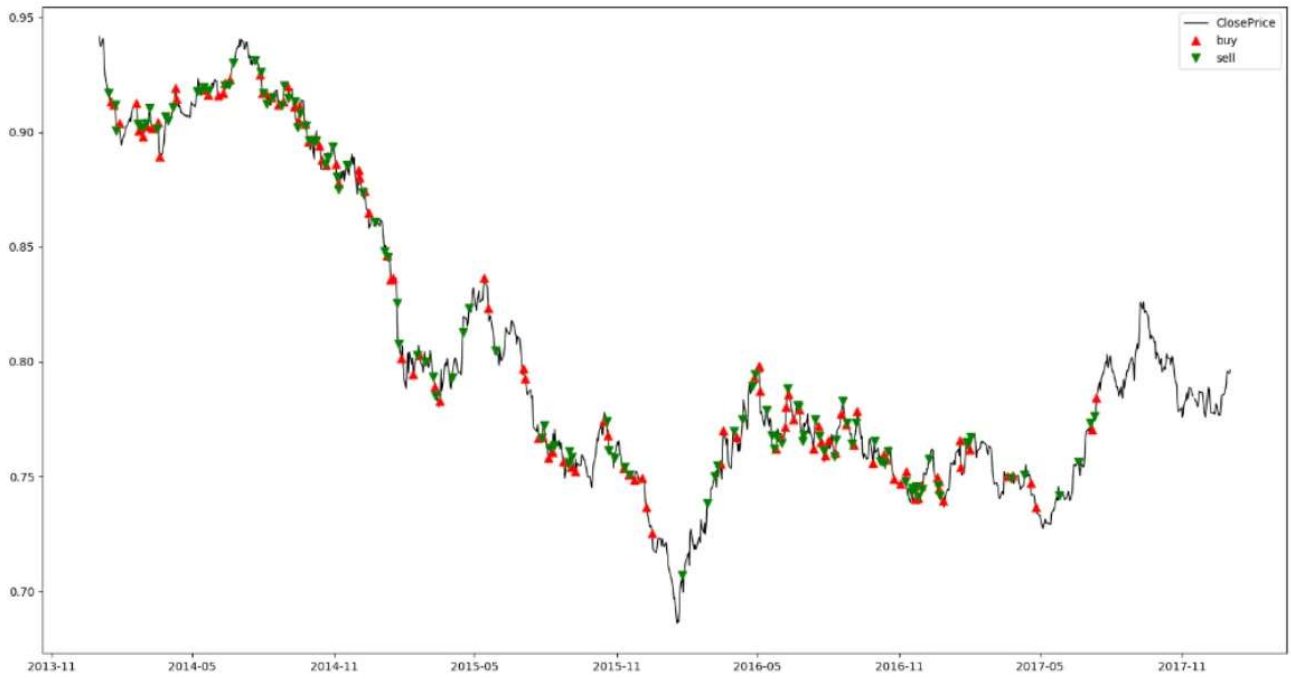


Several significant forecasts are implied in the storyline for JPY/USD and CHF/USD. In spite of this, the CAD/USD and CHF/USD pairs have a limited connection to one another, suggesting that some inaccuracies might result from weak predictive ties between CAD/USD—JPY/USD and CAD/USD—CHF/USD. This analysis highlights StatArb's advantage in using various prominent trading instruments. A weaker pair relationship can be compensated for by one characterized by a robust correlation, in instances where a pair relationship becomes weaker. Using multiple highly correlated pairs to minimize prediction errors was previously discussed.

Using the data frames we created earlier, let's graph the closing price, trade data, positions, and profit and loss results.

```
trading_data = symbols_data[TRADING_INSTRUMENT]
delta_proj_act_data = delta_projected_actual_data



# Assign new columns to the DataFrame
new_cols = {
    'ClosePrice': trading_data['Close'],
    'FinalStatArbTradingSignal': pd.Series(final_delta_projected_history,
index=trading_data.index),
    'Trades': pd.Series(orders, index=trading_data.index),
    'Position': pd.Series(positions, index=trading_data.index),
    'Pnl': pd.Series(pnls, index=trading_data.index),
}



for col_name, col_data in new_cols.items():
    delta_proj_act_data = delta_proj_act_data.assign(**{col_name: col_data})



# Set up markers for buy and sell
buy_markers = delta_proj_act_data.Trades == 1
sell_markers = delta_proj_act_data.Trades == -1



# Plotting the data
plt.plot(delta_proj_act_data.index, delta_proj_act_data.ClosePrice,
color='k', lw=1., label='ClosePrice')
plt.plot(delta_proj_act_data[buy_markers].index,
delta_proj_act_data.ClosePrice[buy_markers], 'r^', markersize=7,
label='buy')
plt.plot(delta_proj_act_data[sell_markers].index,
delta_proj_act_data.ClosePrice[sell_markers], 'gv', markersize=7,
label='sell')
plt.legend()
plt.show()
```

This narrative examines the prices at which caduss trades are bought and sold. A detailed examination of both the final trading signal and the plot is essential to understanding this StatArb signal and strategy. Its behavior is thus thoroughly understood as a result.



As a next step, we'll analyze the real code used to create a visual representation of the ultimate StatArb trading signal. It is possible to gain an understanding of the particular values that trigger buying and selling trades by displaying the buying and selling trades throughout the signal's lifecycle. In this analysis, we will see whether these actions align with the expectations we outlined earlier.

```
plt.plot(delta_projected_actual_data.index,
delta_projected_actual_data.FinalStatArbTradingSignal, color='k', lw=1.,
label='FinalStatArbTradingSignal')
plt.plot(delta_projected_actual_data.loc[delta_projected_actual_data.Trades
== 1].index,
delta_projected_actual_data.FinalStatArbTradingSignal[delta_projected_actual
_data.Trades == 1], color='r', lw=0, marker='^', markersize=7, label='buy')
plt.plot(delta_projected_actual_data.loc[delta_projected_actual_data.Trades
== -1].index,
delta_projected_actual_data.FinalStatArbTradingSignal[delta_projected_actual
_data.Trades == -1], color='g', lw=0, marker='v', markersize=7,
label='sell')
plt.axhline(y=0, lw=0.5, color='k')
for i in np.arange(StatArb_VALUE_FOR_BUY_ENTRY, StatArb_VALUE_FOR_BUY_ENTRY
* 10, StatArb_VALUE_FOR_BUY_ENTRY * 2):
  plt.axhline(y=i, lw=0.5, color='r')
for i in np.arange(StatArb_VALUE_FOR_SELL_ENTRY,
StatArb_VALUE_FOR_SELL_ENTRY * 10, StatArb_VALUE_FOR_SELL_ENTRY * 2):
```

```
    plt.axhline(y=i, lw=0.5, color='g')
plt.legend()
plt.show()
```

Our StatArb trading strategy uses the popular trend-tracking technique, so we anticipate making purchases when the indicator value is positive and making sales when the indicator value is negative. Observing the representation graphicalally, let's see if this holds true:

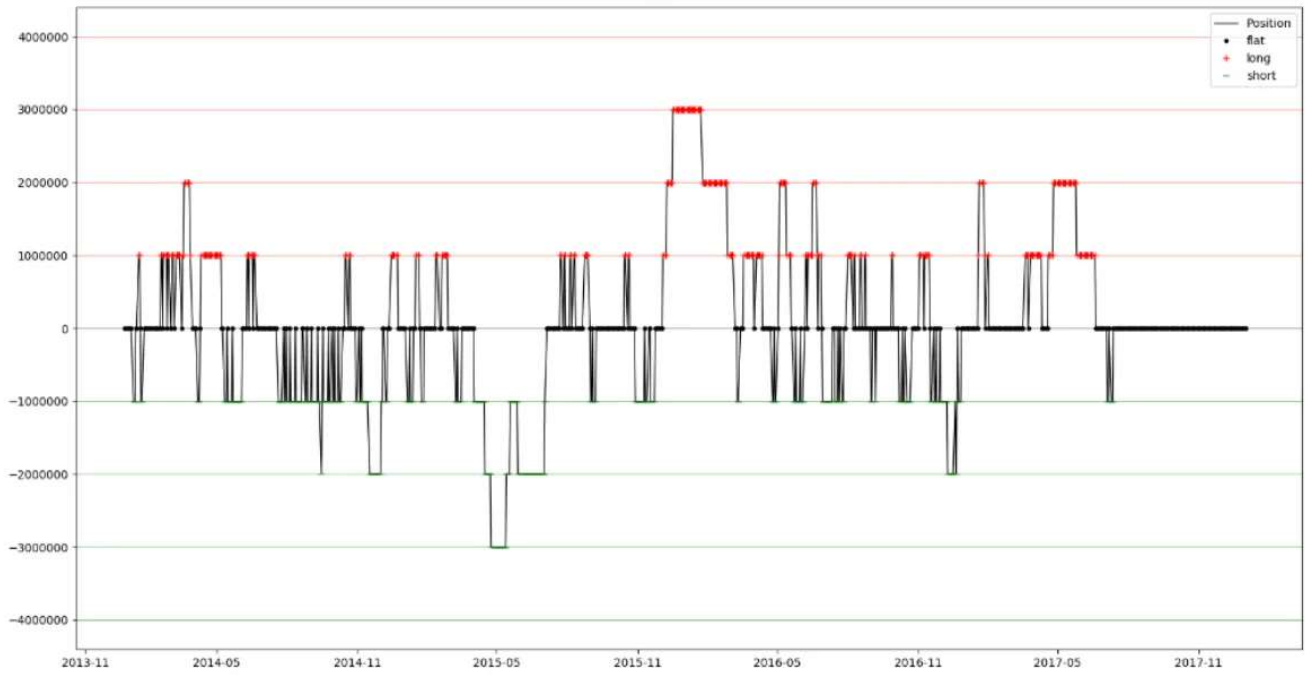
As we examine the storyline, the trend-tracking methods we understand, as well as the StatArb indication we developed, it becomes obvious that many purchases occur when the signal value is favorable. A selling transaction occurs when the signal values go from favorable to unfavorable. A lucrative position is closed when a trade is executed when the signal value is positive or when a trade is executed when the signal value is negative. Both mean reversion and trend-tracking trading techniques are aligned with this observation.

Our study of StatArb trading techniques comes to a close with an examination of the positions and profits/losses through the lens of a chart.

```
delta_proj_act_data = delta_projected_actual_data
positions = delta_proj_act_data.Position
```

```
index = delta_proj_act_data.index



# Set up markers for positions
flat_markers = positions == 0
long_markers = positions > 0
short_markers = positions < 0



# Plotting the data
plt.plot(index, positions, color='k', lw=1., label='Position')
plt.plot(index[flat_markers], positions[flat_markers], 'k.', label='flat')
plt.plot(index[long_markers], positions[long_markers], 'r+', label='long')
plt.plot(index[short_markers], positions[short_markers], 'g_',
label='short')



plt.axhline(y=0, lw=0.5, color='k')



# Plotting horizontal lines
for i in range(NUM_SHARES_PER_TRADE, NUM_SHARES_PER_TRADE * 5,
NUM_SHARES_PER_TRADE):
  plt.axhline(y=i, lw=0.5, color='r')
for i in range(-NUM_SHARES_PER_TRADE, -NUM_SHARES_PER_TRADE * 5, -
NUM_SHARES_PER_TRADE):
  plt.axhline(y=i, lw=0.5, color='g')



plt.legend()
plt.show()
```

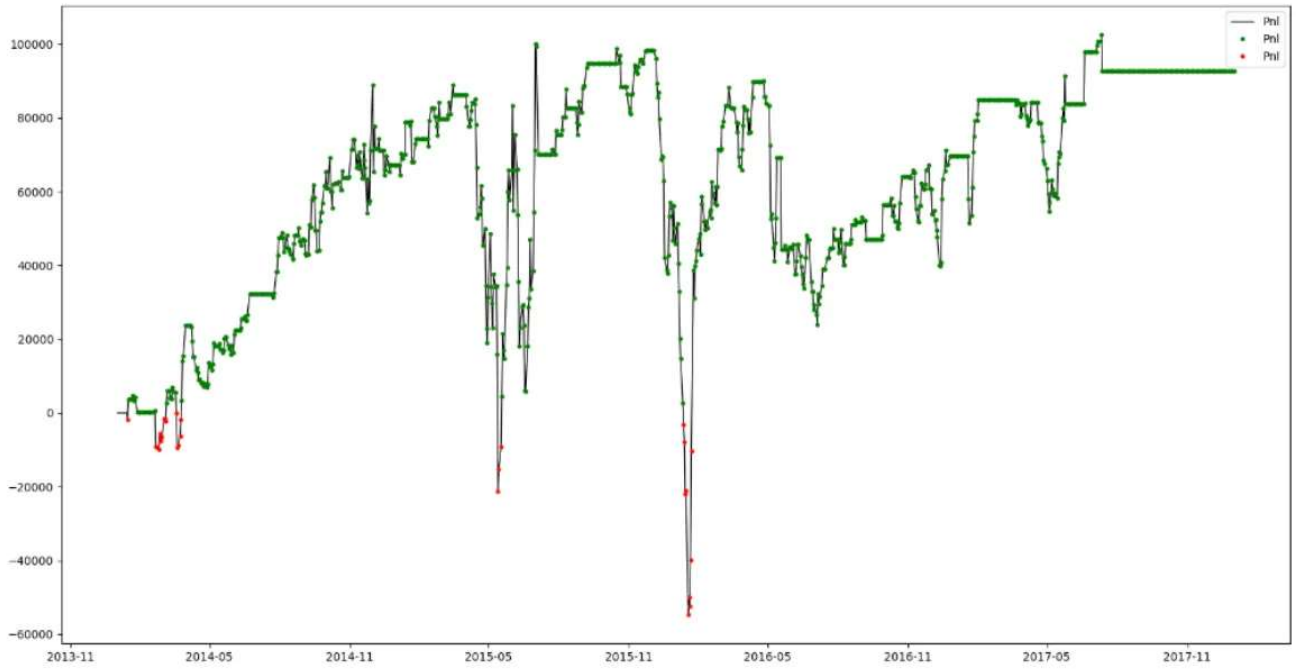
StatArb's position chart shows how its position has changed over time. In these positions, dollars are used instead of contracts to measure success. It is important to note that a position of 100K represents approximately one futures contract and not 100K contracts.



As we discussed earlier, the code for the PnL graph is the same.

```
delta_proj_act_data = delta_projected_actual_data
pnl = delta_proj_act_data.Pnl
index = delta_proj_act_data.index



# Set up markers for PnL
positive_pnl_markers = pnl > 0
negative_pnl_markers = pnl < 0



# Plotting the data
plt.plot(index, pnl, color='k', lw=1., label='Pnl')
plt.plot(index[positive_pnl_markers], pnl[positive_pnl_markers], 'g.',
label='Positive Pnl')
plt.plot(index[negative_pnl_markers], pnl[negative_pnl_markers], 'r.',
label='Negative Pnl')



plt.legend()
plt.show()
```

Due to its core connection between various currency pairs, we anticipate improved results with this strategy compared to our previous ones. A wide variety of currency pairs are used as primary trading instruments, providing greater flexibility in varying market conditions. Due to this, we expect improved performance in various market conditions.



We have used trading signals discussed in earlier sections to construct effective and resilient trading approaches that follow trends and revert to the mean. To increase their dynamism and adaptability to multiple market conditions, we added a volatility-based trading signal to these fundamental strategies. Additionally, we explored a completely new type of trading strategy that utilized economic releases as its basis. As an example, we examined the analysis required to execute these types of strategies. Further, we examined statistical arbitrage, one of our most complex and advanced trading strategies to date. StatArb trading signals and strategies were meticulously quantified and parameterized by combining CAD/USD and other major currencies as leading signals. As shown by our visualization of each step of the process, the trading strategy produced outstanding results. The next section delves into algorithmic strategy risk measurement and management. A market risk assessment, an operational risk assessment, and a potential software implementation glitches evaluation are included in this process.

Comments

Write a comment...