ChatGPT

Swing Trading Strategy Using ADX and Supertrend

Introduction

Swing trading involves capturing short- to medium-term price moves, often holding positions for a few days to a few weeks. To enhance the probability of success, traders often combine indicators that gauge **trend direction** and **trend strength**. In this strategy, we use the **Supertrend** indicator for clear buy/sell signals and the **Average Directional Index (ADX)** to confirm that the trend is strong enough to warrant a trade. This Jupyter Notebook will demonstrate how to fetch stock price data, calculate the Supertrend and ADX indicators, and filter a list of stocks for potential swing trade signals. We will use daily timeframe data by default (a common choice for swing trading [1]), but the approach can be adapted to other timeframes (some traders even use 15-minute or 4-hour charts for swing strategies [2]). The notebook will output a list of stocks that are currently showing a **bullish** or **bearish swing signal** (e.g. indicating a potential **buy** or **sell** opportunity), based on the combined indicator criteria.

Clarifying the Strategy Components:
- **Time Frame:** We'll use **daily data** for illustration, which smooths out intraday noise and aligns with typical swing trading horizons [1]. (You can adjust this to lower timeframes like 4H if shorter swings are desired [2].)
- **Stock Universe:** The code can work with any list of stock tickers (e.g., from NSE, NYSE, NASDAQ, S&P 500, etc.). You can specify your own list; for demonstration, we might use a few well-known stocks.
- **Data Source:** We will fetch historical price data (Open-High-Low-Close, *OHLC*) using a Python API (such as `yfinance` for Yahoo Finance). This ensures the notebook uses up-to-date market data without requiring a separate file download.
- **Signal Output:** The notebook will generate a filtered list of stocks with their current signal. Specifically, it will identify stocks where the **Supertrend** has just flipped direction (uptrend ↔ downtrend or vice versa) and the **ADX** confirms a strong trend. These conditions indicate a potential entry point for a swing trade. We will label these signals as "Buy" (for bullish signals) or "Sell" (for bearish signals). Stocks that do not meet the criteria will be ignored in the output.
- **Backtesting:** This notebook focuses on *signal generation*. However, one could extend it to include backtesting—evaluating how past signals from this strategy would have performed. For simplicity, we will not do a full backtest here, but we will outline how to implement one as a next step.

By the end of the notebook, you will have a well-documented Python implementation of the ADX + Supertrend strategy, and a clear idea of which stocks (from your list) are showing a swing trading signal **today**.

Strategy Overview: ADX + Supertrend

Before diving into code, it's important to understand *why* these two indicators complement each other for swing trading:

- **Supertrend Indicator:** The Supertrend is a popular trend-following indicator that appears as a line plotted on price charts [3] [4]. It is derived from average true range (ATR) calculations and a multiplier factor. When the Supertrend line flips **below the price**, it turns typically green and signifies an **uptrend** (often interpreted as a *buy* signal). Conversely, when the line flips **above the price** (often shown in red), it signifies a **downtrend** (a *sell* signal) [4]. Because it adjusts to volatility (via ATR), Supertrend can react to trend changes and provide relatively clear entry/exit points. However, like any trend indicator, it can produce false signals in sideways markets (whipsaws) [5].

- **Average Directional Index (ADX):** ADX is a momentum indicator that measures the **strength of a trend**, regardless of its direction. The ADX value rises when the market is trending strongly (up or down) and falls during range-bound or choppy periods. A common practice is to use a threshold (such as **25**) on ADX: if ADX is above 25, the trend is considered strong; if below 20, the market is likely trendless or weakly trending [6]. ADX does **not** tell us the trend *direction* (it doesn't say "up" or "down"), but it works in tandem with its companion indicators +DI and –DI (Directional Movement Index) to gauge who's in control (buyers or sellers). In our strategy, we will use the ADX value as a **filter** to confirm that a Supertrend signal occurs in a strongly trending environment [7] [6]. This helps avoid taking trades when the market is flat, thereby reducing false signals.

- **Combining the Two:** Using Supertrend and ADX together means we only act on Supertrend's buy/sell indications when we have confirmation of trend strength. For example, a typical rule might be: *"Go long (buy) when Supertrend turns bullish and ADX is above 25 (indicating a strong uptrend). Go short (sell) when Supertrend turns bearish and ADX is above 25 (strong downtrend)."* This way, ADX serves as a gatekeeper, allowing trades only in robust trending conditions [8]. The combination of a trend indicator with a strength filter significantly **reduces false breakouts** and whipsaws [9], improving the quality of signals. (Optionally, one could also check the +DI and -DI values: for instance, require +DI > –DI for buy signals, ensuring price movement is actually upward. However, since the Supertrend already encapsulates direction, we can keep our rules simpler.)

Indicator Parameters: We will use commonly recommended parameters for our indicators, which you can tweak as needed:

- *Supertrend*: ATR lookback period of 10 and multiplier of 3.0 (these are standard defaults often used in many Supertrend implementations [10]). These settings strike a balance between sensitivity and reliability, as noted in literature. A lower ATR period or multiplier makes Supertrend more sensitive (quicker to flip, but possibly more false signals), while higher values make it more robust but slower to respond.
- *ADX*: Lookback period of 14 (the classic default for ADX) and a **signal threshold of 25**. ADX(14) $\geq$ 25 will denote a strong trend condition in our strategy [6]. You may adjust this threshold (20 is a bit more lenient, 30 more strict) depending on how selective you want to be.

With this understanding, let's move on to implementing the strategy step by step in a Jupyter Notebook format.

Data Collection and Preparation

First, we'll import necessary libraries and gather historical price data for the stocks we want to analyze. We use the `yfinance` library to fetch daily OHLC data. Ensure you have `yfinance` installed (`!pip install yfinance`) or install it in your environment before running the code.

For demonstration, let's assume a small list of stock tickers. You can replace these with any tickers of interest (for example, stocks from a particular exchange or index). We will fetch around 1-2 years of daily data for each stock to have enough history to calculate indicators and possibly to observe past signals.

```
# Step 1: Import libraries
import pandas as pd
import numpy as np

# If using in a local Jupyter environment, you might need to install
yfinance:
# !pip install yfinance
import yfinance as yf

# Define the list of stock tickers to analyze
stocks = ["AAPL", "MSFT", "GOOG", "TSLA", "AMZN"]  # example tickers (Apple,
Microsoft, Google, Tesla, Amazon)

# Define the timeframe for swing trading analysis
timeframe = "1d"
# daily data (you can use '1h', '4h', etc., if needed and if data is
available)
period = "1y"           # fetch 1 year of historical data (you can adjust,
e.g., "2y" for 2 years)

# Fetch historical data for all stocks
data_dict = {}  # to store data for each stock
for ticker in stocks:
    # Download daily historical data for the past 'period'
    df = yf.download(ticker, period=period, interval=timeframe)
    if not df.empty:
        data_dict[ticker] = df
    else:
        print(f"No data found for {ticker}")
```

In the code above, we download daily price data for each stock ticker in the list. The data includes columns like **Open, High, Low, Close, Adj Close, Volume**. We store each DataFrame in a dictionary `data_dict` for easy access. If any ticker returns no data (for example, due to an incorrect symbol or lack of data), we print a warning.

Indicator Calculation: Supertrend and ADX

Next, we will calculate the Supertrend and ADX indicators for each stock's DataFrame. To keep our code organized and reusable, we'll write helper functions: one for **Supertrend** and one for **ADX/DMI**.

Calculating Supertrend

The Supertrend calculation involves the following steps [4] :

1. Compute the Average True Range (**ATR**), which measures volatility. ATR can be calculated from True Range (max of high-low, high-prevClose, low-prevClose) over a lookback period (we'll use 10 days as per our parameter choice).
2. Compute the **basic upper band** and **basic lower band**:
3. Basic Upper Band = (High + Low) / 2 + (Multiplier × ATR)
4. Basic Lower Band = (High + Low) / 2 - (Multiplier × ATR)
5. Initialize **final upper band** and **final lower band** as copies of the basic bands, then adjust them to ensure they do not retreat once established:
6. Final Upper Band: At each time step, if the current basic upper band is lower than the previous final upper band *or* the previous closing price was higher than the previous final upper band, then update the final upper band to the current basic upper band; otherwise, keep the previous final upper band.
7. Final Lower Band: Analogous logic; if the current basic lower band is higher than the previous final lower band *or* the previous closing price was lower than the previous final lower band, then update the final lower band to the current basic lower band; otherwise, keep the previous final lower band.
8. Determine the **Supertrend line**:
9. If the prior Supertrend was equal to the final upper band and the current price closes *below* the final upper band, then the Supertrend stays with the final upper band (trend remains down).
10. If the prior Supertrend was equal to the final upper band and price closes *above* it, then switch the Supertrend to the final lower band (trend turns up).
11. If the prior Supertrend was equal to the final lower band and the current price closes *above* the final lower band, then stay with the final lower band (trend remains up).
12. If the prior Supertrend was equal to the final lower band and price closes *below* it, then switch the Supertrend to the final upper band (trend turns down).
13. Once the Supertrend line is determined, we can derive a **trend direction** signal: if price is above the Supertrend line, we're in an **uptrend**; if below, in a **downtrend** [11] .

Let's implement this in a function. We'll use pandas operations and some iterative logic for steps 3-4, since those steps depend on previous values.

```
def calculate_supertrend(df, atr_period=10, atr_multiplier=3.0):
    """
    Calculate Supertrend indicator for a given DataFrame of OHLC data.
    Adds Supertrend values and trend direction to the DataFrame.
    Returns the DataFrame with new columns: 'Supertrend', 'ST_Trend'.
    'ST_Trend' is True for bullish (uptrend) and False for bearish
(downtrend).
    """
    high = df['High']
    low = df['Low']
    close = df['Close']

    # Step 1: Calculate ATR (Average True Range)
    # True Range components
    price_diffs = pd.DataFrame({
        'high_low': high - low,
        'high_prevclose': (high - close.shift(1)).abs(),
        'low_prevclose': (low - close.shift(1)).abs()
    })
    true_range = price_diffs.max(axis=1)  # element-wise max of the three
differences
    # ATR calculation: using exponential moving average for smoother results
(Wilder's method)
    atr = true_range.ewm(span=atr_period, adjust=False).mean()

    # Step 2: Basic upper and lower bands
    hl2 = (high + low) / 2
    basic_upper_band = hl2 + atr_multiplier * atr
    basic_lower_band = hl2 - atr_multiplier * atr

    # Initialize final bands
    final_upper_band = pd.Series(0.0, index=df.index)
    final_lower_band = pd.Series(0.0, index=df.index)
    supertrend = pd.Series(0.0, index=df.index)

    # Step 3 & 4: Compute final bands and Supertrend
    for i in range(len(df)):
        if i == 0:
            # seed initial bands with basic bands
            final_upper_band.iat[i] = basic_upper_band.iat[i]
            final_lower_band.iat[i] = basic_lower_band.iat[i]
            supertrend.iat[i] = 0  # Supertrend value is not defined for the
first data point
        else:
            # Final Upper Band
            if (basic_upper_band.iat[i] < final_upper_band.iat[i-1]) or
(close.iat[i-1] > final_upper_band.iat[i-1]):
                final_upper_band.iat[i] = basic_upper_band.iat[i]
            else:
                final_upper_band.iat[i] = final_upper_band.iat[i-1]
            # Final Lower Band
            if (basic_lower_band.iat[i] > final_lower_band.iat[i-1]) or
(close.iat[i-1] < final_lower_band.iat[i-1]):
                final_lower_band.iat[i] = basic_lower_band.iat[i]
            else:
                final_lower_band.iat[i] = final_lower_band.iat[i-1]
            # Supertrend
            if supertrend.iat[i-1] == final_upper_band.iat[i-1] and
close.iat[i] <= final_upper_band.iat[i]:
                supertrend.iat[i] = final_upper_band.iat[i]
            elif supertrend.iat[i-1] == final_upper_band.iat[i-1] and
close.iat[i] > final_upper_band.iat[i]:
                supertrend.iat[i] = final_lower_band.iat[i]
            elif supertrend.iat[i-1] == final_lower_band.iat[i-1] and
close.iat[i] >= final_lower_band.iat[i]:
                supertrend.iat[i] = final_lower_band.iat[i]
            elif supertrend.iat[i-1] == final_lower_band.iat[i-1] and
```

```
close.iat[i] < final_lower_band.iat[i]:
                supertrend.iat[i] = final_upper_band.iat[i]

    # Determine trend direction: True for uptrend, False for downtrend
    st_trend = close > supertrend  # True if price is above Supertrend line
    # Append to DataFrame
    df['Supertrend'] = supertrend
    df['ST_Trend'] = st_trend.astype(int)  # 1 for uptrend, 0 for downtrend
(or use True/False)
    return df
```

A few notes on the implementation: we use an exponential moving average (`ewm`) to calculate ATR for a smoother result (Wilder's ATR uses a similar smoothing method). We then iteratively compute the final bands and supertrend because each new value depends on the previous value's state. After computing the Supertrend line, we create a boolean column `ST_Trend` which is `True/1` when price is above the Supertrend (bullish trend) and `False/0` when price is below it (bearish trend). This will make it easier to identify trend changes: a flip from 0 to 1 means a new uptrend signal, and 1 to 0 means a new downtrend signal.

Calculating ADX (Average Directional Index)

The ADX calculation involves a few more steps because it includes the directional movement components (+DI and -DI):

1. Calculate the directional movement (**+DM** and **-DM**):
2. +DM on day *i* = High[i] - High[i-1] if that is greater than Low[i-1] - Low[i] and also positive; otherwise 0.
3. -DM on day *i* = Low[i-1] - Low[i] if that is greater than High[i] - High[i-1] and positive; otherwise 0.
4. +DM and -DM measure the magnitude of upward and downward moves.
5. Calculate the True Range (TR) as we did for ATR (we can reuse the ATR from above or compute a new one with the chosen period, typically 14 for ADX).
6. Compute smoothed averages of TR, +DM, and -DM over the ADX period. Wilder's method uses a smoothing technique (exponential-like). We can use the `ewm` with `span=period` for simplicity, which is equivalent to Wilder's smoothing for large periods.
7. Calculate the **Directional Indicators**: +DI = 100 * (smoothed +DM / smoothed TR), and -DI = 100 * (smoothed -DM / smoothed TR).
8. Compute **DX** = 100 * (|+DI - -DI| / (+DI + -DI)) – this is the directional index for each period.
9. ADX is then the smoothed average of DX over the period. Typically, the first ADX value (after `period` days) is an average of the first `period` DX values, and subsequent ADX values use Wilder smoothing. We can approximate this with an exponential moving average on DX values for simplicity.

We'll implement ADX and also return +DI and -DI if needed for further analysis. The function will add an `ADX` column (and possibly `+DI`, `-DI` for completeness) to the DataFrame.

```python
def calculate_adx(df, adx_period=14):
    """
    Calculate ADX (Average Directional Index) for a given OHLC DataFrame.
    Adds ADX, +DI, -DI columns to the DataFrame.
    """
    high = df['High']
    low = df['Low']
    close = df['Close']

    # Initialize Plus and Minus DM and TR
    plus_dm = pd.Series(0.0, index=df.index)
    minus_dm = pd.Series(0.0, index=df.index)
    tr = pd.Series(0.0, index=df.index)

    for i in range(1, len(df)):
        # Calculate daily +DM and -DM
        up_move = high.iat[i] - high.iat[i-1]
        down_move = low.iat[i-1] - low.iat[i]
        if up_move > down_move and up_move > 0:
            plus_dm.iat[i] = up_move
        else:
            plus_dm.iat[i] = 0.0
        if down_move > up_move and down_move > 0:
            minus_dm.iat[i] = down_move
        else:
            minus_dm.iat[i] = 0.0
        # True Range
        tr.iat[i] = max(high.iat[i] - low.iat[i],
                        abs(high.iat[i] - close.iat[i-1]),
                        abs(low.iat[i] - close.iat[i-1]))

    # Smooth TR, +DM, -DM using exponential moving average (approximating
Wilder's smoothing)
    atr = tr.ewm(span=adx_period, adjust=False).mean()          # Average
True Range over period
    sm_plus_dm = plus_dm.ewm(span=adx_period, adjust=False).mean()
    sm_minus_dm = minus_dm.ewm(span=adx_period, adjust=False).mean()

    # Calculate directional indicators
    plus_di = 100 * (sm_plus_dm / atr)
    minus_di = 100 * (sm_minus_dm / atr)

    # Calculate DX and ADX
    dx = 100 * (abs(plus_di - minus_di) / (plus_di + minus_di))
    adx = dx.ewm(span=adx_period, adjust=False).mean()  # smoothed DX

    # Append to DataFrame
    df['ADX'] = adx
    df['DI+'] = plus_di
    df['DI-'] = minus_di
    return df
```

In the ADX code above, we iterate through the DataFrame to get +DM, -DM, and TR for each day. We then use exponential moving averages to smooth these series. We compute +DI and -DI in percentage terms, and finally DX and ADX. The result is added to the DataFrame with columns `ADX`, `DI+`, and `DI-`. If you only care about ADX and not the individual DI lines, you could omit storing `DI+` and

`DI-`, but they are useful for deeper analysis or for additional confirmation rules (for example, ensuring `DI+ > DI-` when taking a long trade signal, as an extra check [6] [8]).

Note: The smoothing method here (EMA with span=period) is an approximation of the original Wilder's smoothing. It should be very close to the true ADX calculation for our purposes. ADX values typically start being valid after the initial period; for instance, with a 14-day ADX, you might disregard the first 14 values as they are used for initial averaging.

Generating Swing Trade Signals

With `Supertrend` and `ADX` calculated, identifying swing trading signals becomes straightforward. We want to find points where the Supertrend **trend direction flips**, *and at the same time* ADX is above our threshold (indicating a strong trend). Specifically:

- A **Buy Signal** (bullish swing entry) occurs when the Supertrend flips from *bearish* to *bullish*. In terms of our DataFrame: yesterday the trend (`ST_Trend`) was 0 (downtrend) and today it is 1 (uptrend). We will check if at today's close, `ST_Trend` is 1 and the previous day was 0. Additionally, we require today's `ADX` $\geq$ 25 (our chosen threshold) to ensure it's a strong trend [6]. When this condition is met, we flag the stock as a **"Buy"** candidate.

- A **Sell Signal** (bearish swing entry or exit from longs) occurs when Supertrend flips from *bullish* to *bearish* (trend change to downtrend) and ADX $\geq$ 25. In DataFrame terms: yesterday `ST_Trend` was 1 and today it's 0, with ADX high. We flag this as a **"Sell"** signal.

We will iterate over each stock’s DataFrame (already containing the new columns) and check the last two rows for these conditions. We can store the signals in a dictionary for output.

Let's implement the signal extraction:

```python
# Apply the indicator calculations to each stock's data
signals = {}  # to store the signals for each stock
adx_threshold = 25  # define the ADX threshold for a "strong" trend

for ticker, df in data_dict.items():
    if df.shape[0] < 2:
        continue  # not enough data
    # Calculate Supertrend and ADX for the DataFrame
    df = calculate_supertrend(df, atr_period=10, atr_multiplier=3.0)
    df = calculate_adx(df, adx_period=14)

    # Check the last two data points for a trend flip
    last_idx = df.index[-1]
    prev_idx = df.index[-2]
    last_trend = df.at[last_idx, 'ST_Trend']
    prev_trend = df.at[prev_idx, 'ST_Trend']
    last_adx = df.at[last_idx, 'ADX']

    signal = None
    if prev_trend == 0 and last_trend == 1 and last_adx >= adx_threshold:
            signal = "Buy"    # Bullish flip with strong trend
        elif prev_trend == 1 and last_trend == 0 and last_adx >= adx_threshold:
            signal = "Sell"   # Bearish flip with strong trend

        if signal:
            signals[ticker] = signal

# Show the resulting signals
print("Swing Trading Signals (ADX + Supertrend):")
for ticker, sig in signals.items():
    print(f"{ticker}: {sig}")
```

The above code does the following for each stock: - Ensures we have at least 2 data points (to compare yesterday vs today). - Computes Supertrend and ADX indicators. - Looks at the most recent day (`last_idx`) and the day before that (`prev_idx`). - Determines the trend state on those days (`ST_Trend` being 0 or 1). - Retrieves the latest ADX value. - If a trend flip occurred (0→1 or 1→0) and ADX is above the threshold, assigns the corresponding **"Buy"** or **"Sell"** signal. - Stores the signal in a dictionary.

Finally, it prints out the signals for each stock that had one. Stocks with no current signal (no trend flip or weak trend) are not listed, effectively filtering them out.

Example Output and Interpretation

After running the above code, the output might look something like:

```
Swing Trading Signals (ADX + Supertrend):
AAPL: Buy
TSLA: Sell
```

This would mean, for example, that Apple's stock just generated a bullish swing trading signal (Supertrend turned bullish with strong ADX), and Tesla's stock generated a bearish signal (trend turned down, ADX high). These are hypothetical results for illustration; actual output will depend on the latest market data at the time you run the notebook.

How to interpret these signals? If a stock is flagged as "Buy", a swing trader might consider entering a long position on that stock, anticipating an upward swing. A "Sell" signal could prompt a trader to exit a long position or even consider a short position to profit from a downward swing. Importantly, these signals should be used with proper **risk management** (e.g., setting stop-loss orders near the Supertrend line which often acts as support/resistance [12], and position sizing appropriately).

Also note that a strong trend (ADX $\geq$ 25) doesn't guarantee the trend will continue – it simply indicates momentum is present. It's wise to confirm no major resistance/support is immediately ahead, and possibly to combine this strategy with other analyses (like chart patterns or volume analysis) if desired. Nonetheless, by requiring a strong ADX, we are aligning our trades with the existing momentum, which is generally a favorable practice for swing trading [9].

Next Steps: Enhancements and Backtesting

This notebook provides a **signal generation** tool. To further develop this into a complete strategy, you might consider: - **Backtesting:** Evaluate how this ADX+Supertrend strategy would have performed historically for various stocks. This involves simulating trades (buy on "Buy" signal, sell on "Sell" signal or maybe on the opposite signal) and calculating returns, win/loss ratio, drawdowns, etc. Backtesting can be done using libraries like `backtrader`, `zipline`, or even simple pandas logic. - **Parameter Optimization:** The chosen parameters (ATR period, multiplier for Supertrend, ADX period, ADX threshold) could be optimized. Different stocks or asset classes might perform better with slightly different settings. For instance, some traders use Supertrend(7,3) or (10,2) etc., and ADX threshold 20 or 30 depending on how many signals they want. **Optimize with caution** to avoid overfitting – the goal is robust performance, not just historical perfection [13] [14]. - **Multi-timeframe Confirmation:** Some strategies use a higher timeframe to confirm the trend (e.g., weekly trend is up) and a lower timeframe for entries. This can reduce false signals by ensuring you trade in the direction of the larger trend [15]. - **Additional Filters:** The current strategy can occasionally still give false signals (for example, right before a sudden reversal or during news-driven spikes). You could add filters like volume surge confirmation, a requirement that a longer-term moving average is sloping in favor of the trade, or check that **+DI > -DI** for buys (and vice versa for sells) to add further confidence that the move has directional strength [6] [8]. - **Exit Strategy:** Here, we assumed an exit on an opposite signal. In practice, you might decide on a profit target or a trailing stop-loss (possibly using the Supertrend line itself as a trailing stop) to exit trades. Supertrend flipping is a relatively late exit signal, so active traders might take profit earlier or use partial exits.

By incorporating the above, you can enhance the basic ADX+Supertrend strategy into a more complete trading system. Nonetheless, the combination of these two indicators already gives a solid foundation for swing trading – it ensures you trade in the direction of the trend and only when the trend is strong [9]. Always remember to test strategies thoroughly and be mindful of market conditions; no single strategy works all the time, but with the right risk controls and indicator tuning, ADX + Supertrend can be a powerful tool in a swing trader's arsenal.

Sources:

1. VikramDivekar, *Advanced Supertrend ADX Strategy with Highest Return – TradingView Idea* (2025) – (Describes using Supertrend signals with ADX > 20 filter and suitable timeframes for swing trading) [7] [2].
2. *Multi-Indicator Dynamic Trend Trading Strategy: SuperTrend, ADX...* – FMZ.com (2025) – (Explains using Supertrend (ATR 10, factor 3) for direction and ADX (threshold 25) for trend strength on daily swing trading) [10] [16].
3. FMZQuant, *ADX and Supertrend Persistent Entry Strategy* – Medium.com (Jul 2025) – (Highlights that ADX > 25 denotes a strong trend and combines ADX with Supertrend to improve signal quality; lists conditions for buy/sell signals) [6] [8] [9].
4. Cedric Thompson, *Supertrend Indicator: What It Is and How It Works* – Investopedia (2024) – (Overview of the Supertrend indicator, its ATR-based formula, and interpretation: Supertrend line below price = uptrend/buy signal, above price = downtrend/sell signal) [4] [11].

---

[1] [10] [16] Multi-Indicator Dynamic Trend Trading Strategy: SuperTrend, ADX and Liquidity Delta Comprehensive Analysis System
https://www.fmz.com/lang/en/strategy/494337

[2] [7] Advanced Supertrend ADX Strategy with Highest Return by VikramDivekar — TradingView
https://www.tradingview.com/script/RqlCW3XF-Advanced-Supertrend-ADX-Strategy-with-Highest-Return/

[3] [4] [5] [11] [12] Supertrend Indicator: What It Is and How It Works
https://www.investopedia.com/supertrend-indicator-7976167

[6] [8] [9] [13] [14] [15] ADX and Supertrend Persistent Entry Strategy | by FMZQuant | Jul, 2025 | Medium
https://medium.com/@FMZQuant/adx-and-supertrend-persistent-entry-strategy-fb8bfab04be5