**Detailed Breakdown of the "Autocorrelation Strength Analysis" Indicator Code**

This script is written in Pine Script (version 5) and is used in TradingView to create a technical indicator called "Autocorrelation Strength Analysis." This indicator analyzes the strength and reversals of trends using mathematical concepts like autocorrelation and trend strength, then displays relevant information on the chart using labels and colors. Here's a detailed explanation of the code, line by line, to help you understand how it works.

**1. Indicator Declaration**

pinescript

Copy code

//@version=5

//@ Yaseen Khalil

indicator("Autocorrelation Strength Analysis", max\_labels\_count = 500, overlay=true)

* **//@version=5**: Specifies that the code is written using version 5 of Pine Script, ensuring it uses the latest language features.
* **//@ Yaseen Khalil**: A comment identifying the author of the script.
* **indicator("Autocorrelation Strength Analysis", max\_labels\_count = 500, overlay=true)**: Defines the indicator with the name "Autocorrelation Strength Analysis." It sets the maximum number of labels that can be drawn on the chart to 500 and specifies that the indicator should overlay on the main price chart (overlay=true).

**2. Function Definitions**

These functions are designed to manage and manipulate arrays in different ways, such as adding or removing elements, keeping track of a maximum number of items, and clearing arrays.

**Adding and Removing Array Elements**

pinescript

Copy code

add\_to\_front(arr, val, maxItems)=>

array.insert(arr, 0, val)

if (array.size(arr) > maxItems)

array.pop(arr)

arr

* **add\_to\_front**: Adds a new value (val) to the front of an array (arr). If the array exceeds the maximum allowed items (maxItems), the last item is removed to maintain the limit.
* **Why This is Done**: This function keeps the array size manageable while maintaining the most recent data at the front, which is helpful for tracking the latest trends or values.

pinescript

Copy code

add\_to\_end(arr, val, maxItems)=>

array.push(arr, val)

if (array.size(arr) > maxItems)

array.remove(arr, 0)

arr

* **add\_to\_end**: Adds a value to the end of an array. If the array gets too big, the first item is removed to maintain the limit.
* **Why This is Done**: It ensures that the array always contains the most recent data, but keeps the oldest data within the specified limit.

**Removing Specific Elements**

pinescript

Copy code

remove\_element(simple line[] arr, int index)=>line.delete(array.remove(arr, index))

* **remove\_element**: Removes an element from an array at a specified index and deletes it from the chart if it's a graphical object (like a line or label).
* **Why This is Done**: This function is useful when you need to clean up or manage visual elements on the chart dynamically.

**Removing Last and First Elements**

pinescript

Copy code

remove\_last(simple line[] arr)=>line.delete(array.pop(arr))

remove\_first(simple line[] arr)=>line.delete(array.shift(arr))

* **remove\_last** and **remove\_first**: Remove the last or first element of an array, respectively, and delete the associated graphical object.
* **Why This is Done**: Helps manage chart objects efficiently by removing elements that are no longer needed.

**Adding Objects to Front or End**

pinescript

Copy code

add\_to\_front\_objects(arr, val, maxItems)=>

array.insert(arr, 0, val)

if (array.size(arr) > maxItems)

remove\_last(arr)

arr

* **add\_to\_front\_objects**: Adds objects (like lines or labels) to the front of an array and removes the last object if the array exceeds the maximum limit.
* **Purpose**: Maintains an organized list of the most recent graphical objects, keeping the chart clean and readable.

**Clearing Arrays**

pinescript

Copy code

clear\_all\_objects(arr)=>

len = array.size(arr)-1

for i = 0 to len >= 0 ? len : na

remove\_last(arr)

* **clear\_all\_objects**: Loops through the array and removes all elements, clearing the array completely.
* **Why This is Done**: To reset the indicator’s state, particularly when objects need to be fully refreshed or updated.

**3. Input Settings**

pinescript

Copy code

show\_turn\_points = input.bool(true, "Display Reversal Points")

limit\_label\_display = input.bool(false, "Restrict Label Count")

label\_limit = input.int(30, "Maximum Labels Displayed")

label\_scale = input.string("Compact", "Label Size", ["Compact", "Standard", "Extended"])

text\_color = input.string("Steel", "Font Color", ["Steel", "Royal"])

* **Purpose**: These inputs allow the user to customize the indicator’s behavior and appearance, such as toggling the display of reversal points, limiting the number of labels, and selecting label size and color.

**4. Autocorrelation and Trend Strength Calculations**

pinescript

Copy code

auto\_correlation = math.round(ta.correlation(close, close[1], 27) \* 100, 0) // Convert to percentage and round to whole number 14

peak\_correlation = ta.highest(auto\_correlation, 75)

trend\_strength = ta.correlation(time, close, 50)

avg\_range = ta.atr(10)

* **Calculations**:
  + **auto\_correlation**: Measures how the current price relates to the previous price over a 27-period lookback, then converts this value to a percentage. This helps identify how consistent the trend is over time.
  + **peak\_correlation**: Finds the highest autocorrelation value over the last 75 periods, helping identify strong trend points.
  + **trend\_strength**: Calculates the correlation between time and price over 50 periods, indicating how strong the trend is.
  + **avg\_range**: Uses the Average True Range (ATR) over 10 periods to measure market volatility, which helps position labels accurately.

**5. Defining Alert Conditions and Colors**

pinescript

Copy code

bool is\_peak\_reversal = auto\_correlation >= peak\_correlation

bool is\_generic\_alert = auto\_correlation >= 94

color chosen\_text\_color = na

if text\_color == "Steel"

chosen\_text\_color := color.gray

else if text\_color == "Royal"

chosen\_text\_color := color.rgb(65, 105, 225)

color transparent\_color = color.new(color.gray, 100)

color accent\_color = is\_peak\_reversal ? color.red : is\_generic\_alert ? color.green : chosen\_text\_color

* **Purpose**:
  + **Alert Conditions**: Sets up conditions for when the autocorrelation is high, signaling potential trend reversals or strong market alerts.
  + **Color Selection**: Chooses colors based on conditions and user input, ensuring that the indicator visually highlights key areas of interest, like peaks or strong signals.

**6. Creating and Managing Labels**

pinescript

Copy code

label\_array = array.new<label>()

reversal\_array = array.new<label>()

create\_label(index) =>

label.new(bar\_index[index], y = low[index] - avg\_range[index] \* 2, text = str.tostring(auto\_correlation[index]) + "%", color = transparent\_color, textcolor = is\_peak\_reversal[index] ? color.blue : auto\_correlation[index] >= 94 ? color.green : chosen\_text\_color, style = label.style\_label\_down, size = label\_scale == "Compact" ? size.small : label\_scale == "Standard" ? size.normal : label\_scale == "Extended" ? size.large : size.small)

* **Explanation**:
  + **create\_label**: Creates a label on the chart with the autocorrelation percentage, positioned below the price bar and color-coded based on the strength of the autocorrelation.
  + **Purpose**: Provides visual feedback directly on the chart, allowing traders to see critical levels and autocorrelation values easily.

**7. Display Logic for Labels**

pinescript

Copy code

if not limit\_label\_display

label.new(bar\_index, y = low - avg\_range \* 2, text = str.tostring(auto\_correlation) + "%", color = transparent\_color, textcolor = accent\_color, style = label.style\_label\_down, size = label\_scale == "Compact" ? size.small : label\_scale == "Standard" ? size.normal : label\_scale == "Extended" ? size.large : size.small)

* **Purpose**: Displays the autocorrelation label on every bar if label restrictions are not enabled. It provides constant visual data for traders to assess trend strength.

**8. Managing Alert States**

pinescript

Copy code

// Update previous state

previous\_upward\_alert := upward\_alert

previous\_downward\_alert := downward\_alert

* **Purpose**: Keeps track of previous alert states to ensure that repeated alerts do not overwhelm the chart, keeping it clean and readable.