Hunt Gather Trade

Larry Kann

April 26, 2024

Relative Strength Index (RSI)

The Relative Strength Index (RSI) was originally introduced by J. Welles Wilder in 1978 and was designed to help remove the erratic behavior found in constructing momentum lines. [2] This is a popular tool/indicator for traders and analyst to use and is useful for helping to quantify trends. Put simply, the RSI evaluates the average size of recent increases in closing prices against decreases.

The original calculation for the RSI is: [2]

$$RSI = 100 - \frac{100}{1 + RS}$$

Where RS is define as: [2]

$$RS = \frac{\text{Average of } x \text{ days' up closes}}{\text{Average of } x \text{ days' down closes}}$$

Originally, the RSI used a 14 day (period) lookback for calculations [2] and a standard moving average (SMA) for smoothing [1]. Instead of the SMA, this example will use exponential smoothing. This means that when we do bar-by-bar computations, the upward cumulation will suffer an inclusion of zero when the bar moves downwards and vice versa.

We calculate the exponential smoothing with the following equation: [1]

$$Smoothed_t = \frac{(Lookback - 1) * Smoothed_{t-1} + CurrentValue}{Lookback}$$

This equation is derived from the expression:

$$S_t = \alpha X_t + (1 - \alpha) S_{t-1}$$

The RSI calculation, which is performed after each new bar is complete, looks like this:

$$RSI = 100 * \frac{UpSum}{UpSum + DnSum}$$

This computation reflects the proportion of total gains versus total losses over the specified period, normalized to a 0-100 scale, making it easy to interpret relative strength or weakness in the market context. As Masters suggests, it might be beneficial to subtract 50 from final result to center the indicator at 0, especially if you are dealing with predictive models. [1]

Define imports and get data for testing

Before we can build the RSI indicator, we need to do some housekeeping. We are going to use Python to make our RSI indicator. We want to utilize Numba's JIT compiler, which means we need to make sure that our code relies on NumPy Arrays and not Pandas data frames. We will need some data to test the indicator and a way to generate some visualizations for manual observation of the indicator's data.

This first code block will import the libraries we need for this project:

```python
import yfinance as yf
import pandas as pd
import numpy as np
from numba import jit
import talib
import matplotlib.pyplot as plt
```

This next block is a simple function that fetches daily OHLCV data on **SPY** for the last ten years. It drops the columns we aren't using and changes the column names into lowercase (a common schematic for frameworks). We store our historical data in the `history` variable.

```python
def fetch_spx_data():
    ticker_symbol = "SPY"
    ticker_data = yf.Ticker(ticker_symbol)
    data = ticker_data.history(
        period="10y",
        interval="1d"
    )
    columns_to_remove = [
        "Dividends",
        "Stock Splits",
        "Capital Gains"
    ]
    data = data.drop(columns=[col for col in columns_to_remove if col in data.
↪columns])
    data.columns = [col.lower() for col in data.columns]
    return data

history = fetch_spx_data()
```

Create the RSI Indicator

This function calculates the Relative Strength Index (RSI) of a given set of prices, using exponential smoothing to compute the average gains and losses. It offers an option to adjust the RSI scale from the traditional range of [0, 100] to a shifted range of [-50, 50]. This adjustment can be particularly useful in predictive modeling scenarios where a centered scale might provide clearer signals for some algorithms.

To ensure the function performs efficiently, even with the computationally intensive task of exponential smoothing, it employs Numba's JIT (Just-In-Time) compiler. This allows the function to run at speeds suitable for intensive backtesting and real-time financial market analysis, maintaining high performance (for Python) without sacrificing the precision needed for reliable trading signals.

Logic

1. Gains and Losses: For each period, calculate the difference between consecutive closing prices. Positive differences represent gains, while negative differences represent losses.
2. Exponential Smoothing: Apply exponential smoothing to gains and losses separately, using the 'lookback' period as the basis for smoothing. This method prioritizes recent price movements, providing a weighted average that more accurately reflects current market conditions.
3. RSI Calculation: Compute the RSI using the smoothed gains and losses, converting these values into an index that ranges from 0 to 100 (or -50 to 50 if adjusted).

Note: This function utilizes the equations and methods mentioned on the first page to calculate the RSI. Comments in the code explain the purpose of each section.

Docstring:

```
"""
Parameters:
- close (np.ndarray): The array of closing prices.
- period (int): The number of periods to calculate the average gains and losses.
- subtract_50 (bool): If True, shifts the RSI range from [0, 100] to [-50, 50] for use in predi

Returns:
- np.ndarray: An array of RSI values where each value corresponds to the RSI
      at a given point in time. The array has the same length as 'close'. RSI values
      for the initial 'period' points are set to NaN because there's not enough data
      to make the calculation.

Raises:
- ValueError: If 'period' is less than 1 or if 'close' array is empty.
"""
```

RSI Code

```
@jit(nopython=True)
def rsi(close, lookback, subtract_50=False):
    n = close.size
    output = np.full(n, np.nan)  # Use NaN to indicate non-ready periods

    # Arrays to store raw gains and losses
    gains = np.zeros(n)
    losses = np.zeros(n)

    # Calculate gains and losses
    for i in range(1, n):
        delta = close[i] - close[i - 1]
        if delta > 0:
            gains[i] = delta
        else:
            losses[i] = -delta

    # Arrays to store smoothed gains and losses
    smoothed_gain = np.zeros(n)
    smoothed_loss = np.zeros(n)

    # Initialize the first values based on the first non-zero gain/loss
    smoothed_gain[0] = gains[0]  # Assumes the first value is an initialization␣
↪step
    smoothed_loss[0] = losses[0]  # Same as above

    # Apply the smoothing formula to gains and losses
    for i in range(1, n):
        smoothed_gain[i] = ((lookback - 1) * smoothed_gain[i - 1] + gains[i]) /␣
↪lookback
        smoothed_loss[i] = ((lookback - 1) * smoothed_loss[i - 1] + losses[i]) /
↪ lookback

    # Calculate RSI
    for i in range(lookback, n):
        total = smoothed_gain[i] + smoothed_loss[i]
        if total != 0:
            rsi = 100 * (smoothed_gain[i] / total)
            output[i] = rsi - 50 if subtract_50 else rsi

    return output
```

Calculate RSI on Historical Data

In this section, we just create our RSI indicators and add the values to the data frame. I have also added the TA-Lib RSI indicator to compare it with the one I created above. This was done more out of curiosity than necessity.

```
lookback_period = 7
history['rsi'] = rsi(history['close'].values, lookback_period)
history['rsi_talib'] = talib.RSI(history['close'].values,␣
 ↪timeperiod=lookback_period)

print(history[['close', 'rsi']].tail())
print(history[['close', 'rsi_talib']].tail())
```

```
                                close        rsi
Date
2024-04-22 00:00:00-04:00  499.720001  33.432270
2024-04-23 00:00:00-04:00  505.649994  48.284286
2024-04-24 00:00:00-04:00  505.410004  47.780945
2024-04-25 00:00:00-04:00  503.489990  43.544091
2024-04-26 00:00:00-04:00  508.260010  55.087174
                                close  rsi_talib
Date
2024-04-22 00:00:00-04:00  499.720001  33.432270
2024-04-23 00:00:00-04:00  505.649994  48.284286
2024-04-24 00:00:00-04:00  505.410004  47.780945
2024-04-25 00:00:00-04:00  503.489990  43.544091
2024-04-26 00:00:00-04:00  508.260010  55.087174
```

Calculate daily returns and define RSI threshold values

In financial analysis, calculating returns is crucial for assessing investment performance, risk, and the overall dynamics of financial markets. Three common types of returns often computed in financial datasets include simple returns, log returns, and cumulative returns, each serving distinct purposes and providing different insights.

Simple returns, often referred to as relative changes, measure the percentage change from one period to another. In the provided code, `simple_returns` are calculated using the `pct_change()` method, which computes the percentage change between consecutive closing prices.

Log returns, calculated by taking the natural logarithm of the ratio of consecutive closing prices, are preferred for mathematical tractability in financial models.

Cumulative simple returns provide the total aggregated return of an investment over a period. In the code, `cumulative_simple_returns` are computed by cumulatively multiplying (1 plus each period's simple return) and subtracting one, effectively compounding the returns over time.

Each return type provides valuable insights depending on the context: simple returns for immediate period-over-period performance, log returns for continuous compounding and multi-period analysis, and cumulative returns for total investment performance over time. For this paper, I will use the `log_returns` for visualizations with the RSI indicator.

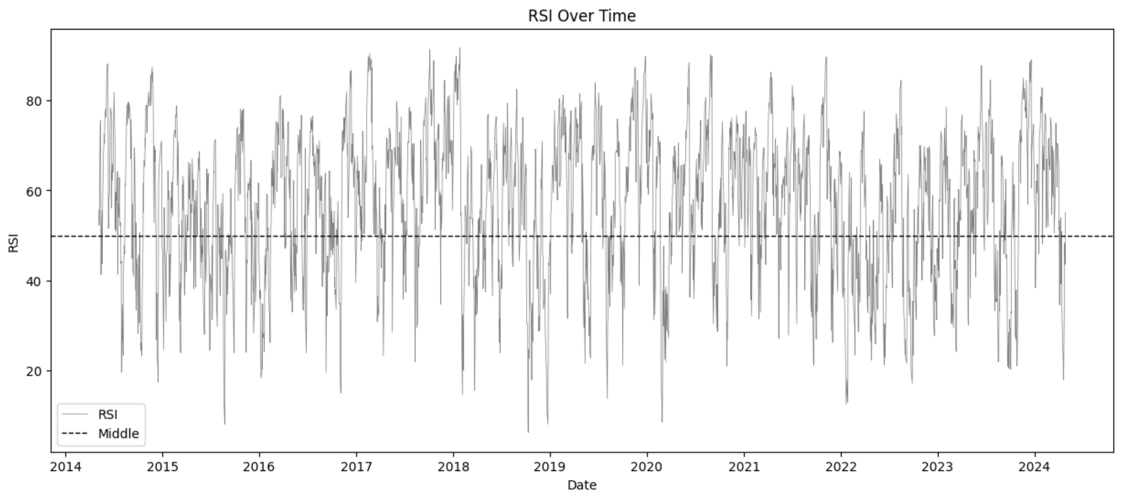
```
history['simple_returns'] = history['close'].pct_change()
history['log_returns'] = np.log(history['close'] / history['close'].shift(1))
history['cumulative_simple_returns'] = (1 + history['simple_returns']).
 ↪cumprod() - 1

rsi_top = 70
rsi_bottom = 30
```

Plot the indicator and returns

The following sections are a series of visualizations for examining the RSI and determining whether we can see any patterns or extract information manually. We will plot the RSI and returns separately and look for any apparent nonstationarity in our series. Then, we will create a series of scatter plots and get some initial impressions from our data.

RSI Plot

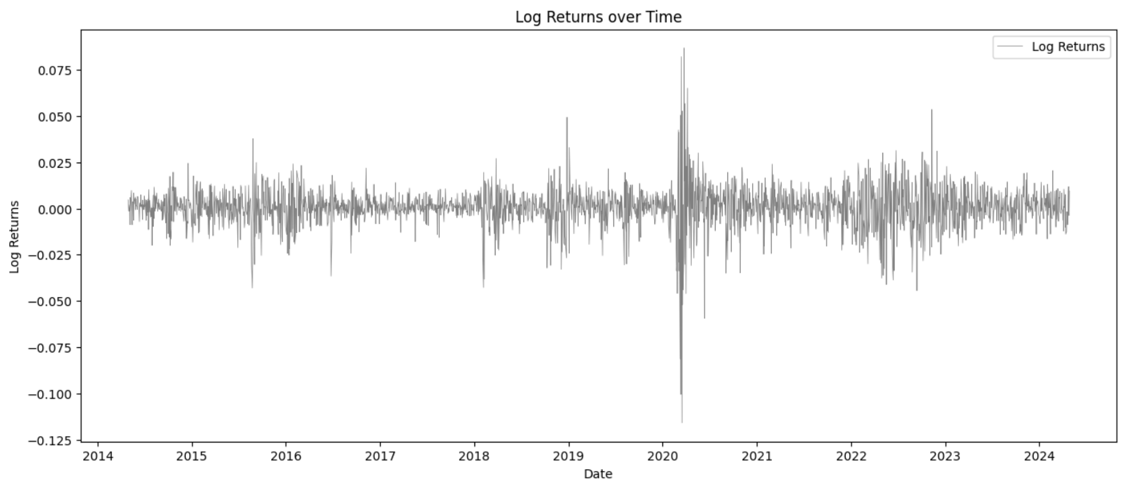
```
plt.figure(figsize=(15, 6))  # Wider than the default
plt.plot(history.index, history['rsi'], label='RSI', color='grey', linewidth=0.
 ↪5)
plt.title('RSI Over Time')
plt.xlabel('Date')
plt.ylabel('RSI')
plt.axhline(50, color='black', linestyle='--', label='Middle', linewidth=1)
plt.legend()
plt.show()
```



Impressions: Visual inspection of the RSI plot shows our indicator (at least in the 10 year window we are looking at) is mostly stationary. Values appear to never hit the min/max values.

Log Returns Plot

```
plt.figure(figsize=(15, 6))  # Wider than the default
plt.plot(history.index, history['log_returns'], label='Log Returns',␣
 ↪color='grey', linewidth=0.5)
plt.title('Log Returns over Time')
plt.xlabel('Date')
plt.ylabel('Log Returns')
plt.legend()
plt.show()
```

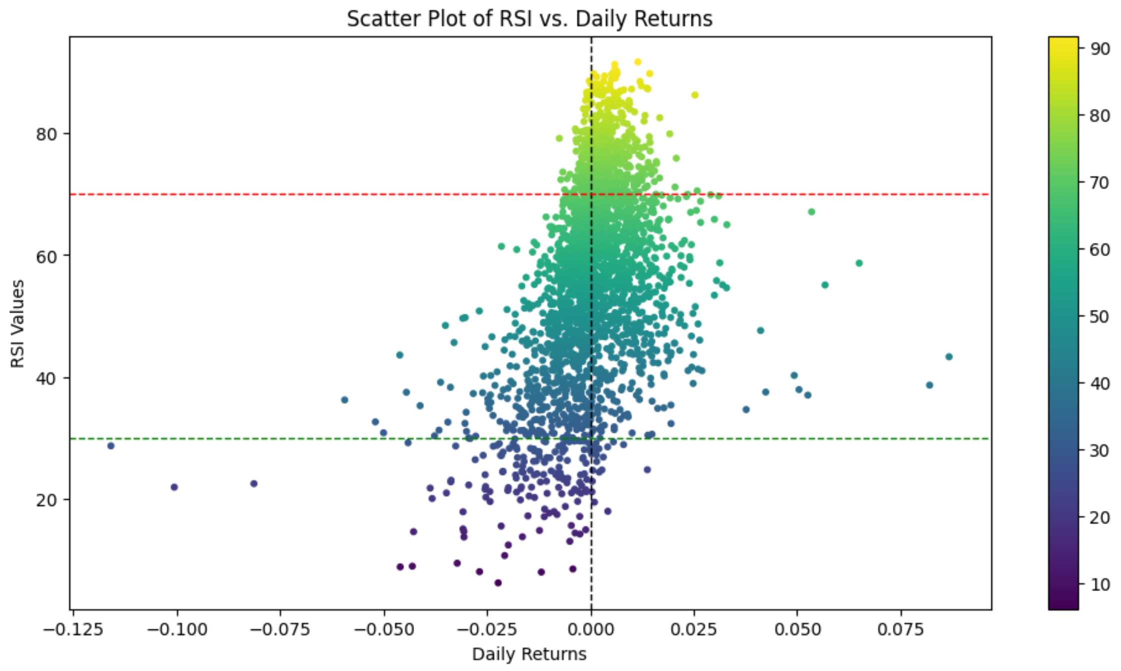


Impressions: Returns appear mostly stationary. There is a clear aberrancy in 2020 that might show a break in the mean in future testing.

Plot RSI vs. Log Returns

This plot will help us understand where the most valuable information in the RSI resides in comparison to the returns. A vertical line has been placed on the chart to help with visualization, along with two horizontal lines denoting where our threshold values are.

```
plt.figure(figsize=(12, 6))
plt.scatter(history['log_returns'], history['rsi'], c=history['rsi'], s=10,␣
 ↪cmap='viridis')

# Define center line and RSI tails/thresholds
plt.axvline(0, color='black', linestyle='--', linewidth=1)
plt.axhline(rsi_top, color='red', linestyle='--', linewidth=1,␣
 ↪label=f'Overbought ({rsi_top})')
plt.axhline(rsi_bottom, color='green', linestyle='--', linewidth=1,␣
 ↪label=f'Oversold ({rsi_bottom})')

# Colorbar to show the color scale in relation to the RSI values
plt.colorbar()

plt.title('Scatter Plot of RSI vs. Daily Returns')
plt.xlabel('Daily Returns')
plt.ylabel('RSI Values')
plt.show()
```

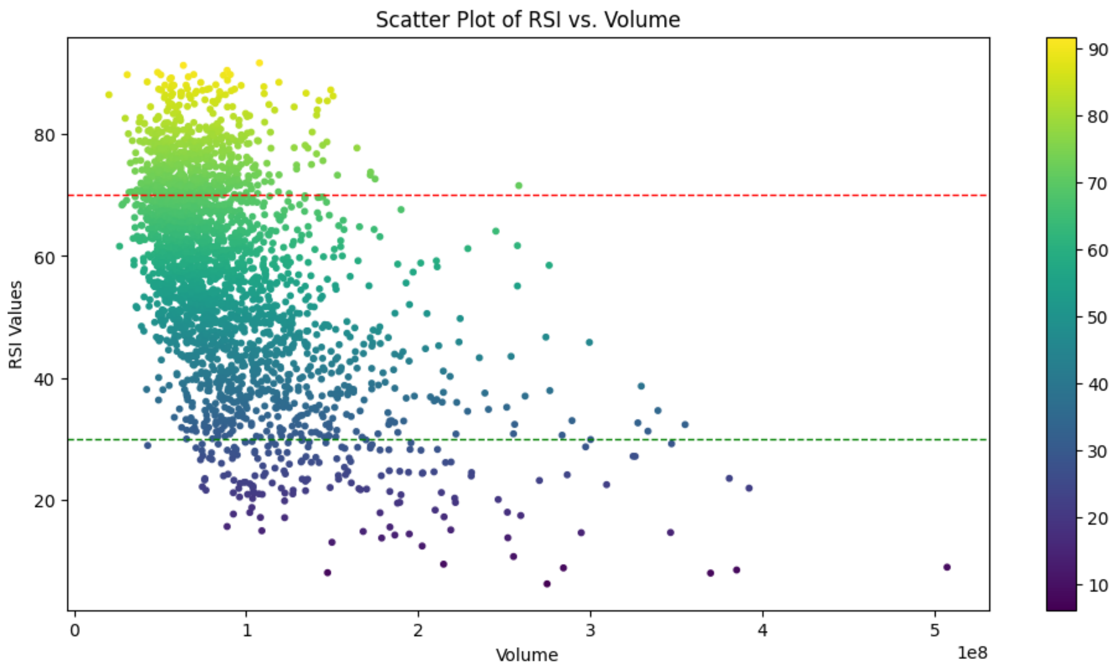


Impressions: When you look at the returns above/below our threshold lines, you can see that their is a stronger correlation with returns. For example, it appears that above the upper threshold (70) there are more positive returns than there are negative.

Plot RSI vs. Volume

These charts help us determine if there is any correlation between the volume and the RSI value. First, we plot all the data as a scatter plot to determine if there are any specific variations we want to examine. Then, we generate scatter plots for RSI values above and below any area of interest.

```
plt.figure(figsize=(12, 6))
plt.scatter(history['volume'], history['rsi'], c=history['rsi'], s=10,␣
 ↪cmap='viridis')
plt.colorbar()  # Adds a colorbar to show the color scale in relation to the␣
 ↪RSI values
plt.axhline(rsi_top, color='red', linestyle='--', linewidth=1,␣
 ↪label=f'Overbought ({rsi_top})')
plt.axhline(rsi_bottom, color='green', linestyle='--', linewidth=1,␣
 ↪label=f'Oversold ({rsi_bottom})')
plt.title('Scatter Plot of RSI vs. Volume')
plt.xlabel('Volume')
plt.ylabel('RSI Values')
plt.show()
```

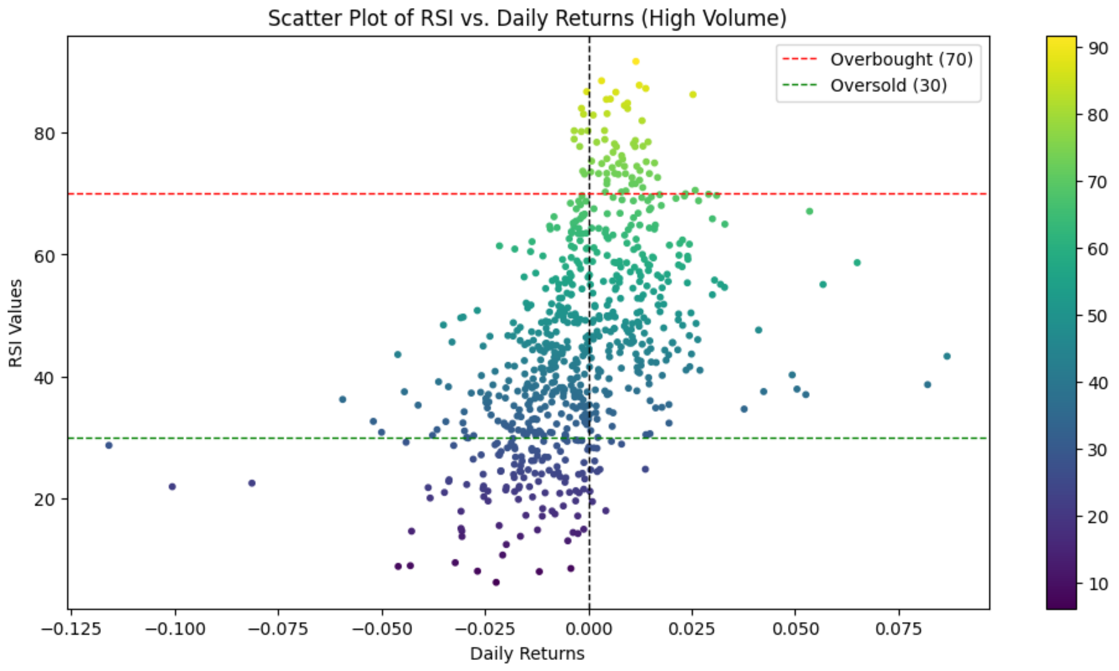


Impressions: The majority cluster is below 100,000,000 (1e8) volume. Above that value and the results are more disbursed and less tightly clustered. Further visualizations of data (RSI vs. returns) above and below this volume threshold would be ideal.

Plot RSI vs log returns above and below volume threshold

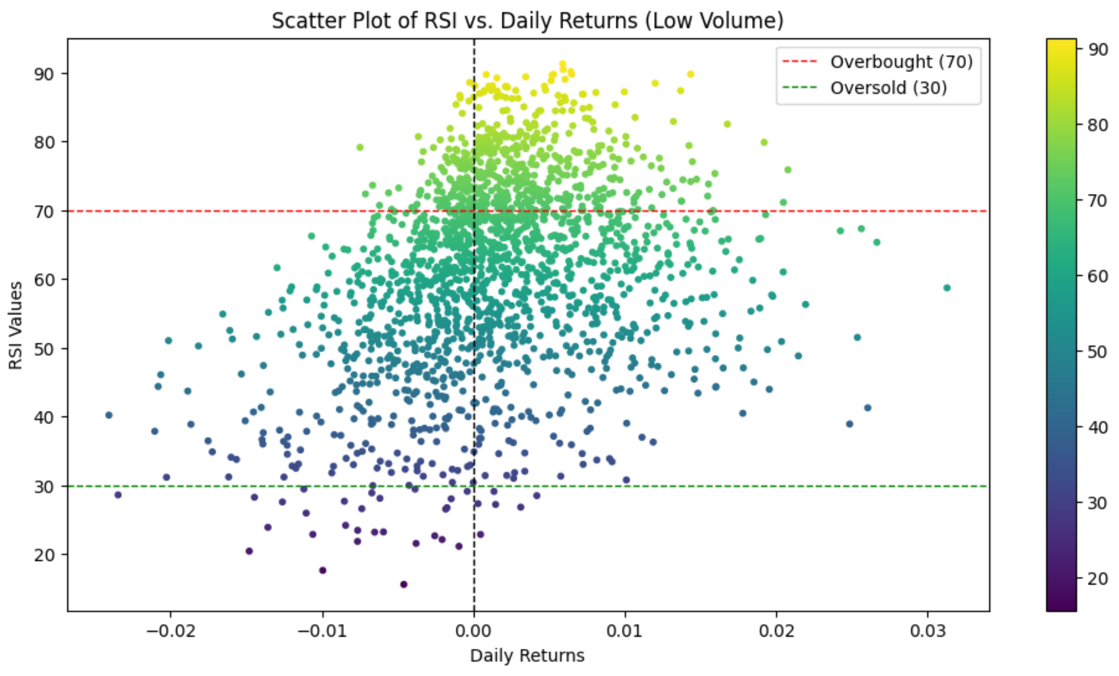
This comparison will separate our returns into two different plots. The first one will be high volume (defined as > 1e8 volume) followed by low volume (defined as < 1e8 volume). This should help us determine if the volume can help improve trade entry confidence.

```
# Define your volume threshold
volume_threshold_high = 100000000  # Example threshold (1e8)

# Filter the data where volume is above the threshold
high_volume_data = history[history['volume'] > volume_threshold_high]

plt.figure(figsize=(12, 6))
plt.scatter(high_volume_data['log_returns'], high_volume_data['rsi'],␣
 ↪c=high_volume_data['rsi'], s=10, cmap='viridis')

plt.axvline(0, color='black', linestyle='--', linewidth=1)
plt.axhline(rsi_top, color='red', linestyle='--', linewidth=1,␣
 ↪label=f'Overbought ({rsi_top})')
plt.axhline(rsi_bottom, color='green', linestyle='--', linewidth=1,␣
 ↪label=f'Oversold ({rsi_bottom})')

plt.colorbar()
plt.title('Scatter Plot of RSI vs. Daily Returns (High Volume)')
plt.xlabel('Daily Returns')
plt.ylabel('RSI Values')
plt.legend()
plt.show()
```



```
# Define your volume threshold for low volume
volume_threshold_low = 100000000  # Same or different threshold
```

```python
# Filter the data where volume is below the threshold
low_volume_data = history[history['volume'] < volume_threshold_low]

plt.figure(figsize=(12, 6))
plt.scatter(low_volume_data['log_returns'], low_volume_data['rsi'],␣
 ↪c=low_volume_data['rsi'], s=10, cmap='viridis')

plt.axvline(0, color='black', linestyle='--', linewidth=1)
plt.axhline(rsi_top, color='red', linestyle='--', linewidth=1,␣
 ↪label=f'Overbought ({rsi_top})')
plt.axhline(rsi_bottom, color='green', linestyle='--', linewidth=1,␣
 ↪label=f'Oversold ({rsi_bottom})')

plt.colorbar()
plt.title('Scatter Plot of RSI vs. Daily Returns (Low Volume)')
plt.xlabel('Daily Returns')
plt.ylabel('RSI Values')
plt.legend()
plt.show()
```



Impressions: Both plots show better returns (respectively) when we are above/below the RSI threshold values. When volume is above `1e8`, the associated returns show minimal returns in the opposite direction. There seems to be a stronger correlation with negative returns when below 30 than with positive returns when above 70. However, there appear to be more observations above the top threshold than below.

Summary

All interpretations, inclinations, or hypotheses drawn from this data are the author's interpretations. They may contain errors.

The RSI calculation functions as intended and appears to be stationary via eye test.

The scatter plot depicting the relationship between the Relative Strength Index (RSI) and returns reveals distinct patterns at the established thresholds. Notably, when the RSI exceeds the upper threshold, there is a predominant occurrence of positive returns, suggesting a strong correlation between high RSI values and upward market trends. Conversely, RSI values below the lower threshold are predominantly associated with negative returns, indicating a likely continuation of downward trends. *These observations support the hypothesis that RSI values beyond these thresholds can be predictive of the corresponding trend direction in returns*. This trend-predictive behavior of the RSI is particularly significant as it validates the use of RSI as a momentum indicator in technical analysis, providing insights into potential market movements based on historical price data.

The analysis of the RSI versus returns, segmented by trading volume (above and below $1 * 10^8$), corroborates the initial observations that RSI extremes predict return directions. Notably, in instances where the trading volume surpasses the $1 * 10^8$ threshold, there are significantly fewer occurrences of returns that contradict the expected trend direction. Specifically, above the upper threshold, negative returns are less frequent, enhancing the reliability of RSI as an indicator of positive market momentum under high-volume conditions.

It is important to highlight that, despite lower volumes, the pattern of RSI aligning with directional trends in returns remains evident. This persistence suggests that while volume amplifies the predictive clarity of the RSI, the indicator itself is robust across different volume levels. This observation opens avenues for adjusting the volume threshold in future analyses to refine our understanding of volume's impact on the predictive power of the RSI.

References

[1] Timothy Masters. *Statistically Sound Indicators for Financial Market Prediction*. Self-Published, 2 edition, 2020.

[2] John J. Murphy. *Technical Analysis of the Financial Markets*. New York Institute of Finance, 1999.