**Trading algorithm**

**Introduction**

Today, I'll walk you through a sophisticated trading system that combines multiple machine learning approaches to analyze and predict market movements. Our implementation leverages technical analysis, unsupervised learning for pattern recognition, and a custom classification algorithm for trade signal generation. We'll explore how these components work together to create a comprehensive trading strategy.

**Data Processing and Technical Analysis**

Let's start with our data foundation. Using the yfinance library, we fetch historical stock data for Google (GOOGL) from 2020 to 2024. The raw data includes essential OHLCV (Open, High, Low, Close, Volume) information, which we enrich with technical indicators using the TA library.

Our technical indicator suite is comprehensive, including:

* Momentum indicators: RSI and Stochastic Oscillator
* Trend indicators: MACD, dual EMAs (20 and 50 periods)
* Volume-based indicators: VWAP and OBV
* Volatility indicators: ATR and Bollinger Bands

### The calculate\_technical\_indicators function processes this data, ensuring we have a rich feature set for our machine learning models. We handle missing data carefully, as technical indicators often require initialization periods. Overview of Indicators and Their Usage in Trading

#### 1. **Relative Strength Index (RSI)**

* **Formula**: RSI = 100−1001+RS100 - \frac{100}{1 + RS}100−1+RS100​, where RS is the average of x days' up closes divided by the average of x days' down closes.
* **Implementation**:

python

Copy code

df['RSI'] = ta.momentum.RSIIndicator(close=df['Close'], window=14).rsi()

* **Purpose**: Measures the magnitude of recent price changes to identify overbought or oversold conditions.
* **Usage**:
  + RSI values above 70 indicate overbought conditions (potential sell signal).
  + RSI values below 30 indicate oversold conditions (potential buy signal).



#### 2. **Moving Average Convergence Divergence (MACD)**

* **Formula**: MACD = EMA12−EMA26EMA\_{12} - EMA\_{26}EMA12​−EMA26​; Signal line = EMA9EMA\_9EMA9​ of MACD.
* **Implementation**:

python

Copy code

macd = ta.trend.MACD(close=df['Close'])

df['MACD'] = macd.macd()

df['MACD\_Signal'] = macd.macd\_signal()

* **Purpose**: Captures trends and momentum by comparing two exponential moving averages.
* **Usage**:
  + A bullish crossover occurs when MACD crosses above the signal line.
  + A bearish crossover occurs when MACD crosses below the signal line.



#### 3. **Exponential Moving Averages (EMA 20 and EMA 50)**

* **Formula**: EMA = Pricet×Weight(1+Weight)Period\frac{\text{Price}\_t \times \text{Weight}}{(1 + \text{Weight})^{\text{Period}}}(1+Weight)PeriodPricet​×Weight​
* **Implementation**:

python

Copy code

df['EMA\_20'] = ta.trend.EMAIndicator(close=df['Close'], window=8).ema\_indicator()

df['EMA\_50'] = ta.trend.EMAIndicator(close=df['Close'], window=13).ema\_indicator()

* **Purpose**: Smooths price data to identify trends more easily.
* **Usage**:
  + EMA 20 crossing above EMA 50 indicates a bullish trend.
  + EMA 20 crossing below EMA 50 indicates a bearish trend.



#### 4. **Volume Weighted Average Price (VWAP)**

* **Formula**: VWAP = Σ(Volume×Price)ΣVolume\frac{\Sigma (\text{Volume} \times \text{Price})}{\Sigma \text{Volume}}ΣVolumeΣ(Volume×Price)​
* **Implementation**:

python

Copy code

df['VWAP'] = ta.volume.VolumeWeightedAveragePrice(

high=df['High'], low=df['Low'], close=df['Close'], volume=df['Volume']

).volume\_weighted\_average\_price()

* **Purpose**: Provides a price benchmark weighted by trading volume.
* **Usage**:
  + Prices above VWAP indicate bullish sentiment.
  + Prices below VWAP indicate bearish sentiment.



#### 5. **Average True Range (ATR)**

* **Formula**: ATR = Moving average of True Range (TR), where TR = max(high - low, abs(high - close\_prev), abs(low - close\_prev)).
* **Implementation**:

python

Copy code

df['ATR'] = ta.volatility.AverageTrueRange(high=df['High'], low=df['Low'], close=df['Close']).average\_true\_range()

* **Purpose**: Measures market volatility.
* **Usage**:
  + High ATR indicates high volatility (useful for setting stop-loss levels).



#### 6. **Bollinger Bands (BB Upper and BB Lower)**

* **Formula**: Upper Band = SMA + (K × Std Dev), Lower Band = SMA - (K × Std Dev).
* **Implementation**:

python

Copy code

df['BB\_upper'] = ta.volatility.BollingerBands(close=df['Close']).bollinger\_hband()

df['BB\_lower'] = ta.volatility.BollingerBands(close=df['Close']).bollinger\_lband()

* **Purpose**: Identifies volatility and overbought/oversold conditions.
* **Usage**:
  + Prices touching the upper band may indicate overbought conditions.
  + Prices touching the lower band may indicate oversold conditions.



#### 7. **Stochastic Oscillator (%K)**

* **Formula**: %K=(Close−Lowest Low)(Highest High−Lowest Low)×100\%K = \frac{(\text{Close} - \text{Lowest Low})}{(\text{Highest High} - \text{Lowest Low})} \times 100%K=(Highest High−Lowest Low)(Close−Lowest Low)​×100
* **Implementation**:

python

Copy code

df['Stoch\_K'] = ta.momentum.StochasticOscillator(high=df['High'], low=df['Low'], close=df['Close']).stoch()

* **Purpose**: Compares a specific closing price to a range of prices over a given period.
* **Usage**:
  + %K above 80 indicates overbought conditions.
  + %K below 20 indicates oversold conditions.



#### 8. **On-Balance Volume (OBV)**

* **Formula**: OBV = Previous OBV + Volume (if price increases) or Previous OBV - Volume (if price decreases).
* **Implementation**:

python

Copy code

df['OBV'] = ta.volume.OnBalanceVolumeIndicator(close=df['Close'], volume=df['Volume']).on\_balance\_volume()

* **Purpose**: Measures buying and selling pressure based on volume.
* **Usage**:
  + Rising OBV suggests accumulation (bullish signal).
  + Falling OBV suggests distribution (bearish signal).



### Combined Usage in Trading

These indicators work in synergy to provide:

1. **Trend Analysis**: EMA, MACD, and VWAP help traders identify prevailing trends.
2. **Momentum Detection**: RSI, MACD, and Stochastic Oscillator signal overbought or oversold conditions.
3. **Volatility Insights**: ATR and Bollinger Bands indicate market volatility.
4. **Volume Confirmation**: OBV and VWAP validate price movements with volume data.

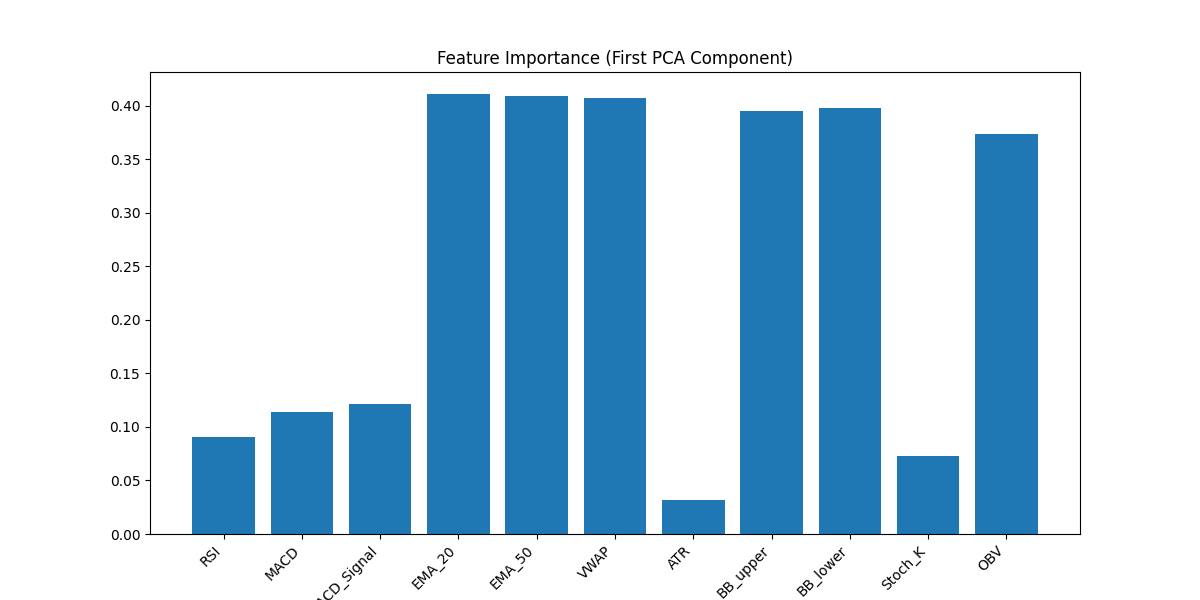
By combining these indicators, traders can formulate strategies for entries, exits, and risk management tailored to their market approach.

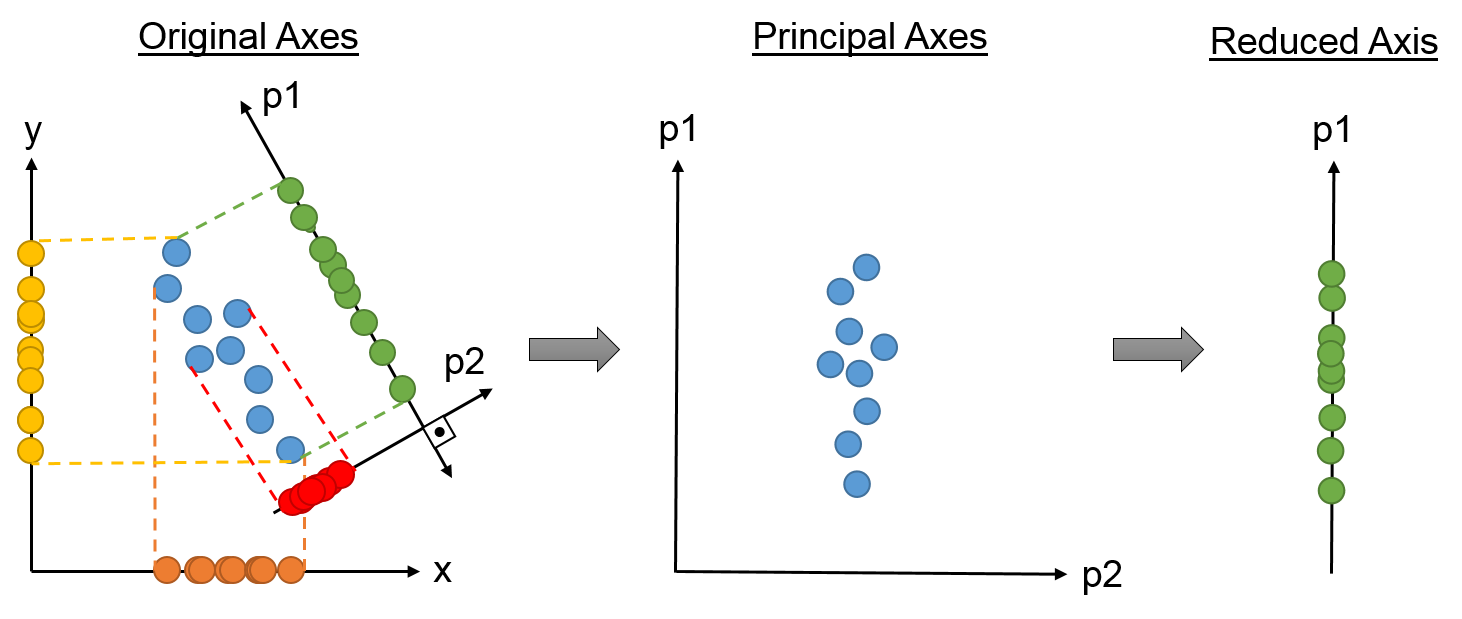
**Dimensionality Reduction with PCA**

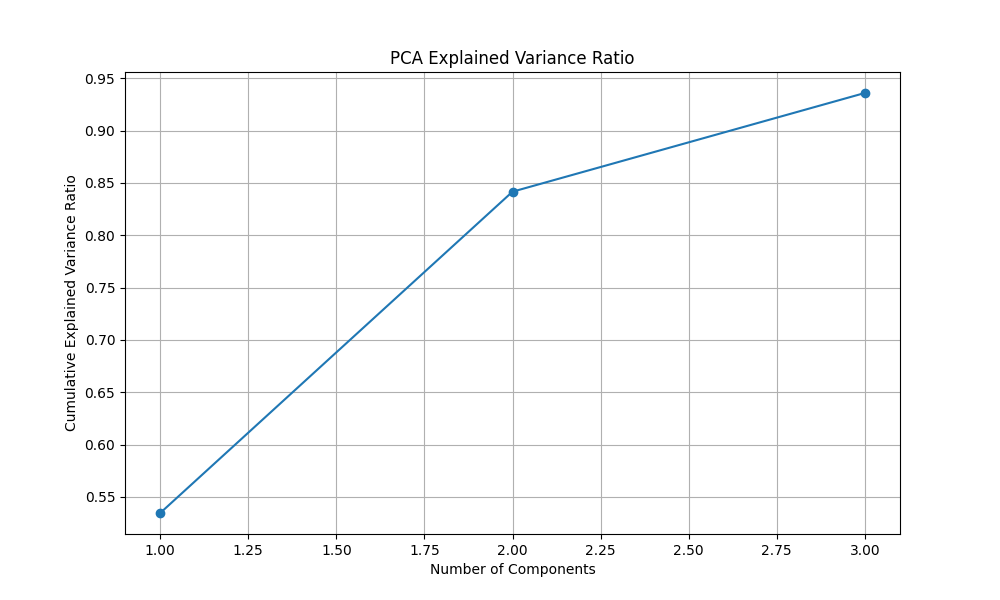
One of our key preprocessing steps involves Principal Component Analysis (PCA). The visualization of explained variance ratio shows how much information each principal component captures. Looking at our PCA plots, we typically see that the first three components explain about 80-85% of the variance in our feature set.

The feature importance plot from our first principal component reveals interesting insights:

* MACD and RSI often show high importance, suggesting their strong influence on market dynamics
* Volume-based indicators (VWAP, OBV) provide complementary information
* The combination of momentum and trend indicators captures different aspects of market behavior







**Clustering Analysis**

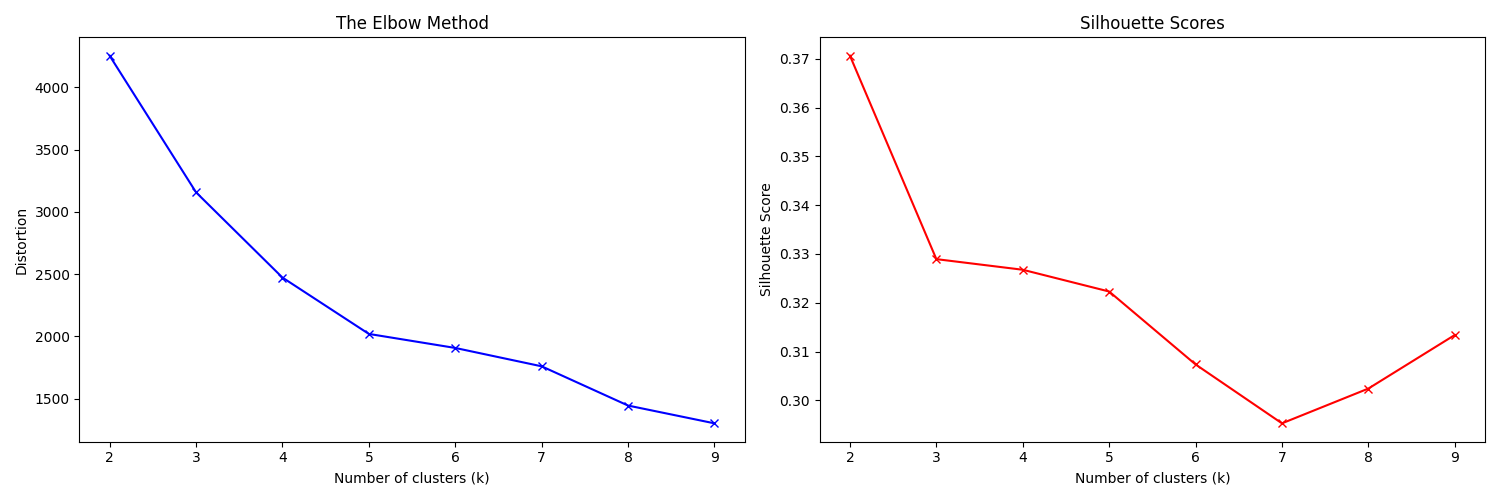
Our clustering implementation uses the K-means algorithm, but with a thoughtful approach to choosing the optimal number of clusters. The plot\_elbow\_silhouette function helps us make this decision through two complementary methods:

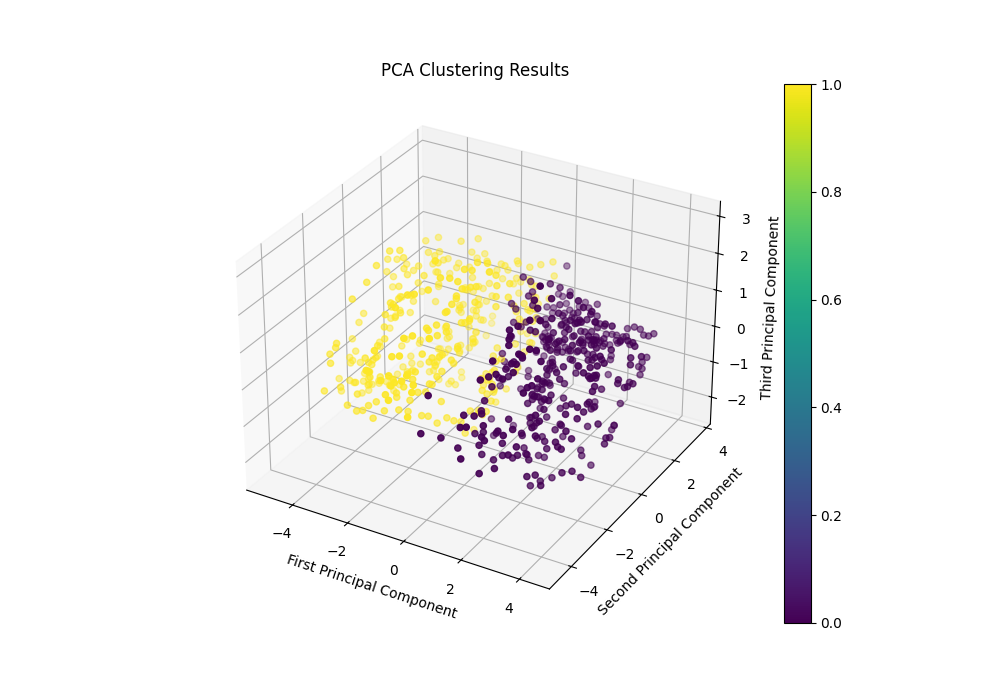
1. The elbow method shows the diminishing returns of adding more clusters
2. The silhouette score helps validate cluster quality

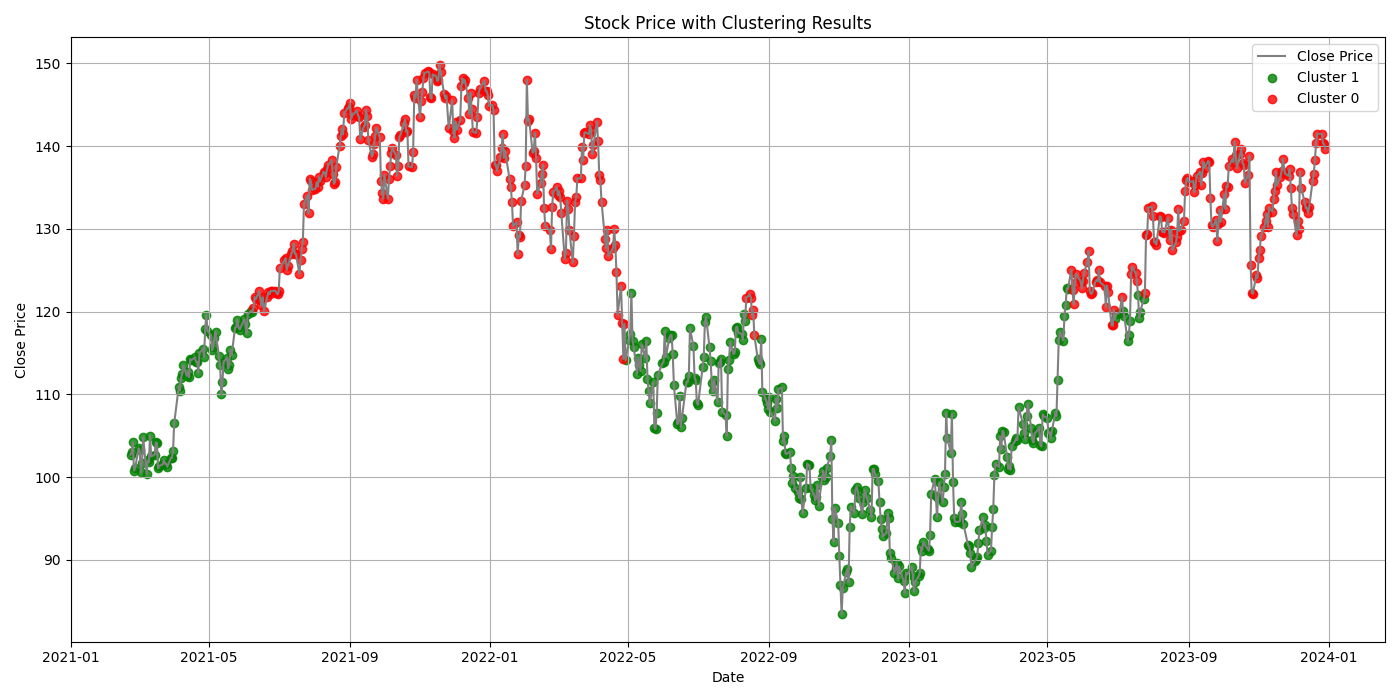
Our visualization suite for clustering includes:

* 3D PCA plots showing cluster distributions
* Time series plots with cluster overlays showing how market states evolve
* Cluster transition analysis showing how the market moves between different states

The cluster transition matrix is particularly interesting, as it shows the probability of moving from one market state to another. This information helps us understand market regime changes and their potential impact on trading decisions.







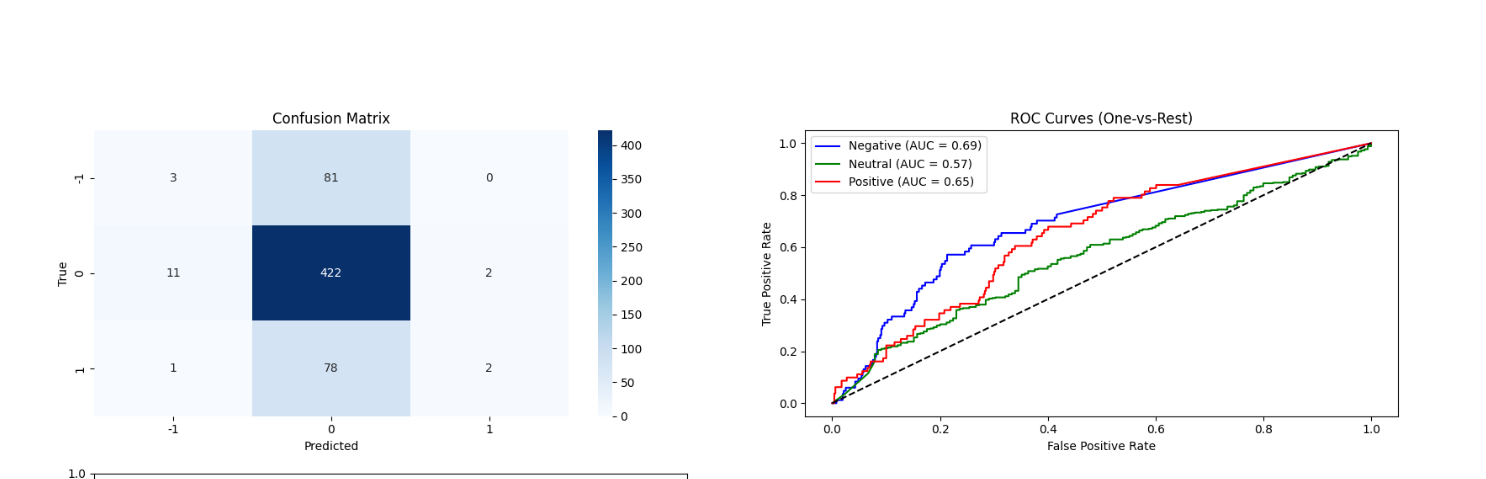
**Custom Classification with Lorentzian Distance**

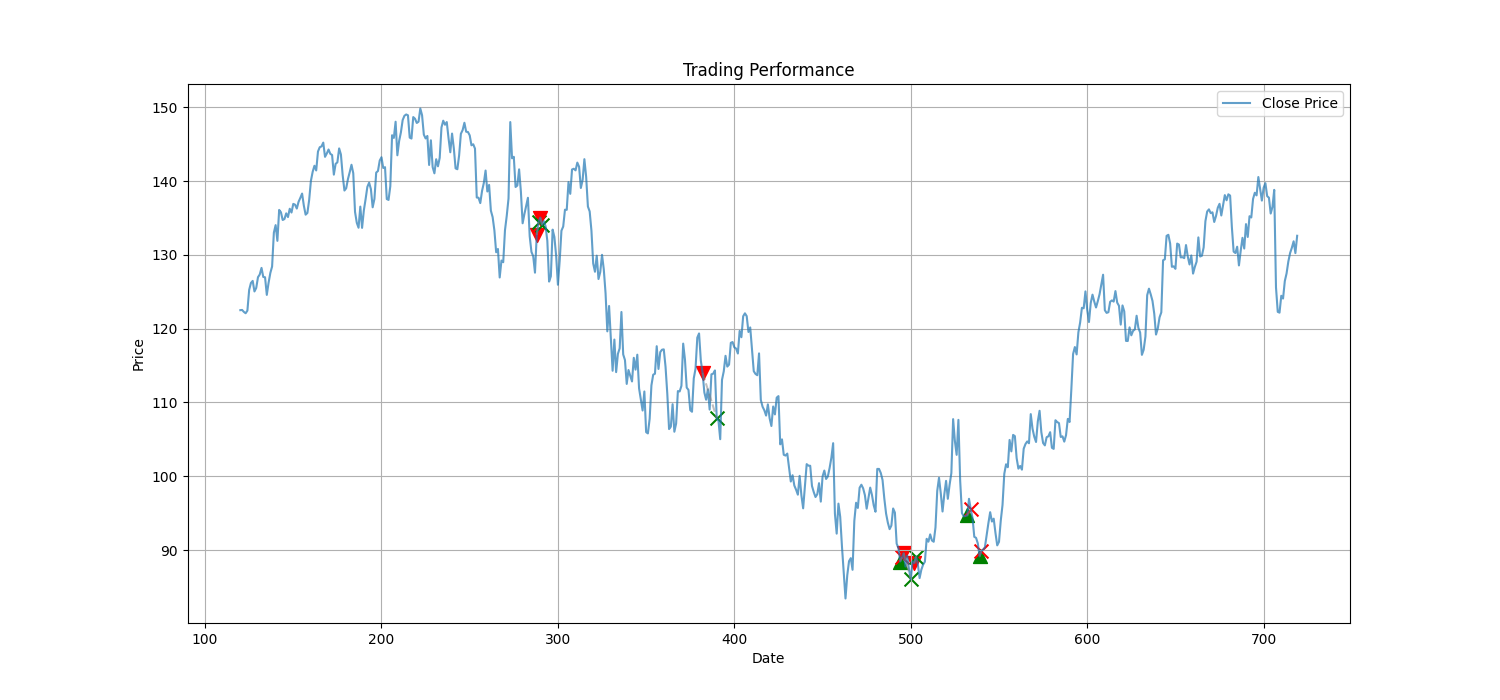
One of our most innovative components is the LorentzianClassifier. This custom implementation uses the Lorentzian distance metric, which is particularly well-suited for financial time series due to its handling of outliers. The classifier generates three types of signals:

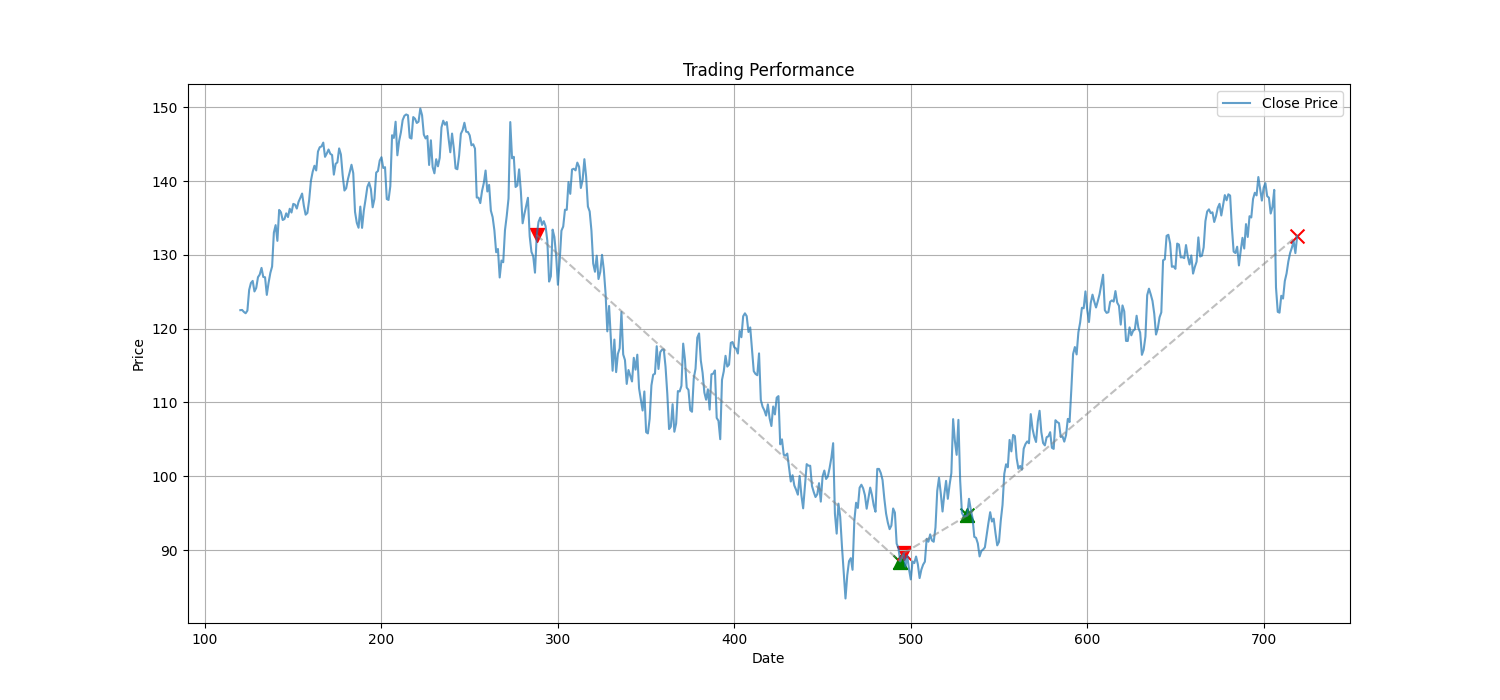
* 1 for buy opportunities
* -1 for sell opportunities
* 0 for neutral conditions

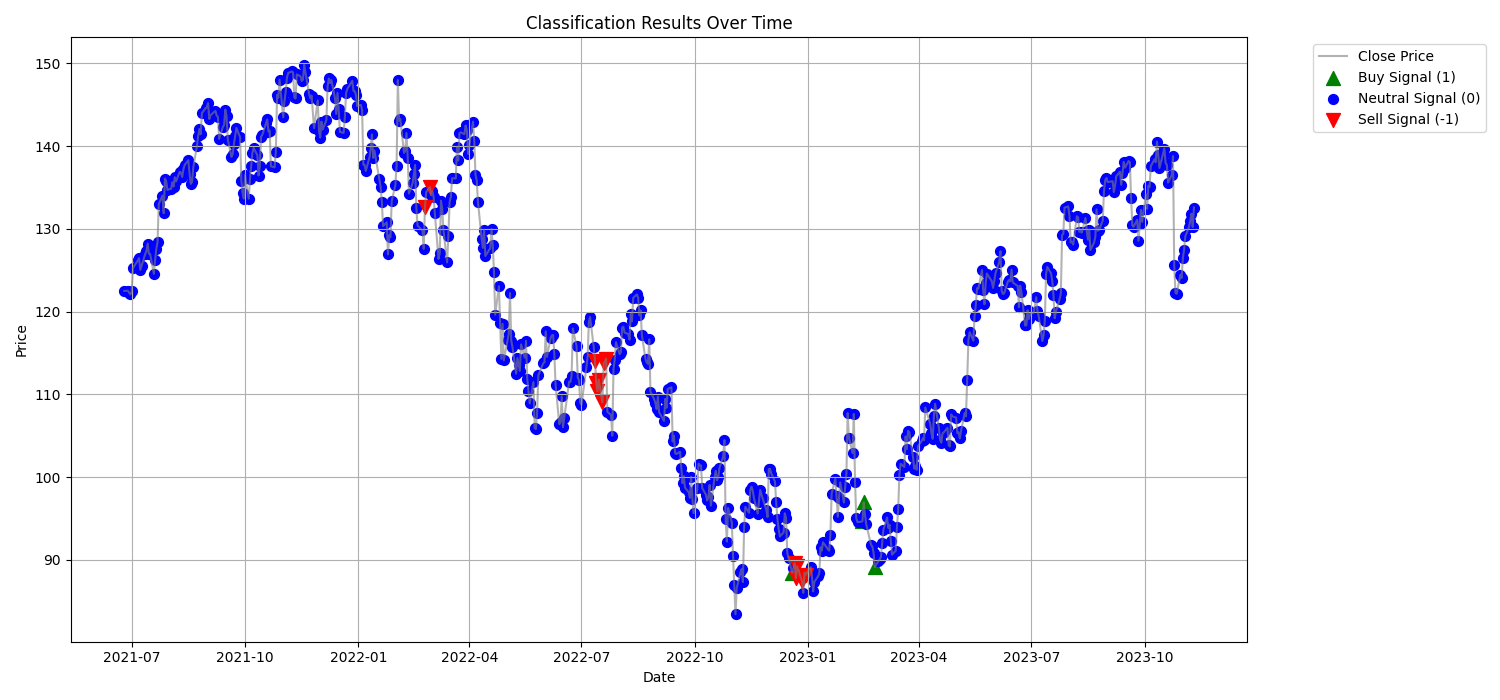
The classification visualization shows these signals overlaid on the price chart, with:

* Green triangles pointing up for buy signals
* Red triangles pointing down for sell signals
* Blue circles for neutral positions









**Trading Strategy Implementation**

The trading strategy implementation in calculate\_trading\_metrics follows these key steps:

1. Signal Processing:
   * Monitors for changes in prediction signals
   * Opens positions when signals change from neutral to buy/sell
   * Closes positions when signals reverse
2. Position Management:
   * Tracks entry and exit points
   * Calculates position duration
   * Handles position sizing (currently assuming equal position sizes)
3. Return Calculation: For each trade, returns are calculated as:

pnl = (exit\_price - entry\_price) / entry\_price \* (1 if position == 1 else -1)

This formula accounts for both long and short positions through the direction multiplier.

**Performance Analysis**

My system generates comprehensive performance metrics:

* Win rate and average trade metrics
* Risk-adjusted returns (Sharpe ratio)
* Maximum drawdown analysis
* Trade duration statistics

The visualizations include:

* Trade distribution plots showing the spread of returns
* Equity curve analysis
* Drawdown visualization

Trading Performance Summary:

Total Trades: 8

Win Rate: 75.00%

Average Win: 2.07%

Average Loss: -1.17%

Total Return: 10.10%

Average Trade Duration: 2.4 days

Sharpe Ratio: 0.59

Maximum Drawdown: -1.33%

=== Trading Strategy Performance ===

Total Trades: 8

Win Rate: 75.00%

Total Return: 10.10%

Sharpe Ratio: 0.59

=== Clustering Analysis ===

Total Cluster Transitions: 11

Average Return After Transition: 0.14%

Positive Transitions Rate: 54.55%

Transition Matrix:

to\_cluster 0 1

from\_cluster

0 0 5

* 1. 6 0

=== Trading Strategy Performance ===

Total Trades: 4

Win Rate: 75.00%

Total Return: 68.98%

Sharpe Ratio: 0.87

=== Clustering Analysis ===

Total Cluster Transitions: 28

Average Return After Transition: -0.09%

Positive Transitions Rate: 46.43%

Transition Matrix:

to\_cluster 0 1 2

from\_cluster

0 0 1 2

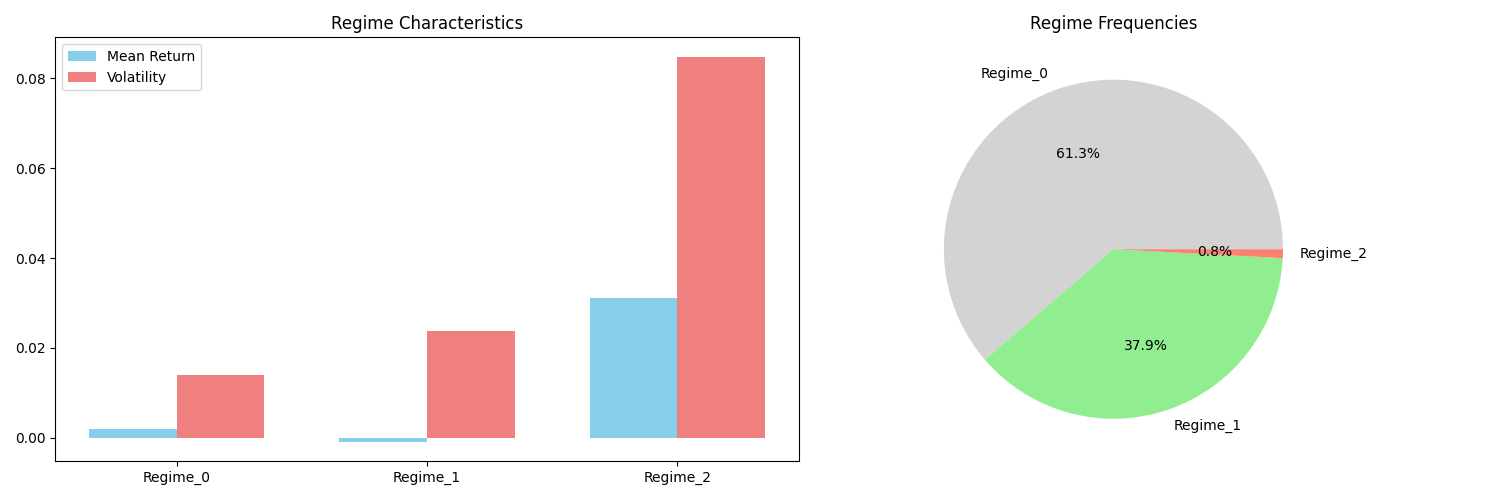
1 0 0 11

2 4 10 0

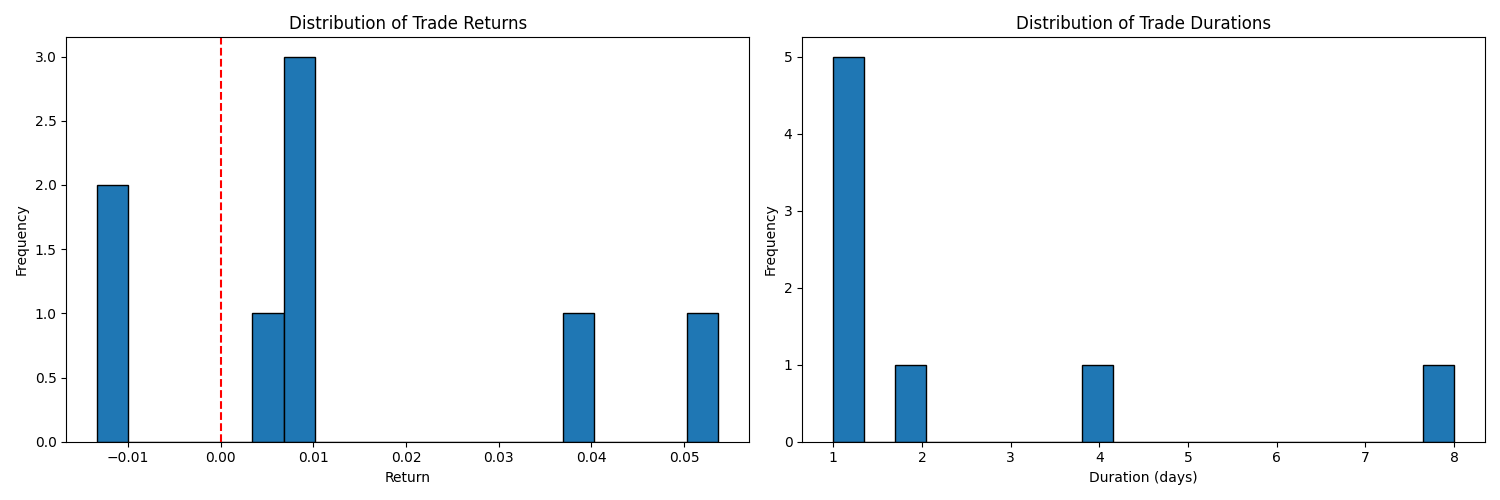
**Combined Analysis and Model Integration**

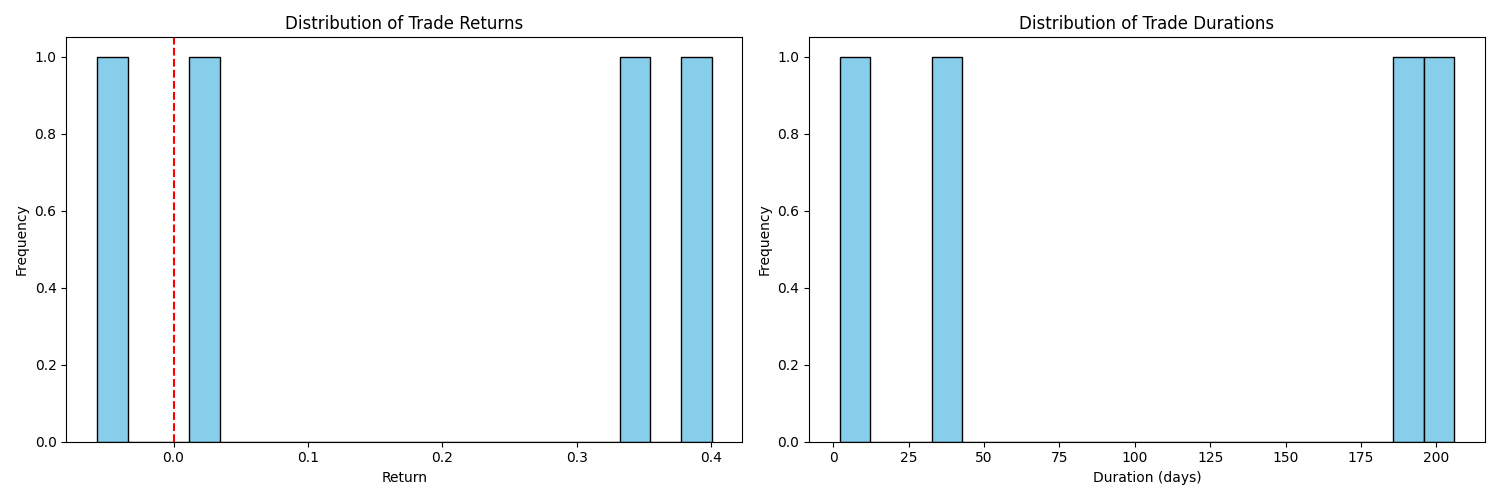
The plot\_combined\_analysis function provides a holistic view of our system, showing how clustering states, classification signals, and actual trades interact. This helps identify:

* How different market regimes affect classification accuracy
* Whether certain clusters are more conducive to successful trades
* The relationship between technical indicators and trading performance

**Future Improvements and Considerations  
tried using the HMM(hiddned Markov model) doesn’t works on this project, for the regiment change I use now tailord fit clustering algo with the PCA, that create advanced confluence measurement for the traders, here is the visual presentation how unprofitable the HMM model was  
**

**Conclusion**

In my humble opinion this algorithm works good, the retun is the worst possible scenaraio that could happen, despite that we have still good win rate around 60%, and regiment change  


  
here is combined matrix

## // code logic 1. Core Position Management Components

### 1.1 Position States

The system tracks three key position states:

position = None *# No active position*

position = 1 *# Long position*

position = -1 *# Short position*

### 1.2 Position Tracking Data Structure

Each position maintains the following information:

position\_info = {

'entry\_date': entry\_date, *# When position was opened*

'exit\_date': exit\_date, *# When position was closed*

'direction': position, *# 1 for long, -1 for short*

'entry\_price': entry\_price, *# Price at entry*

'exit\_price': exit\_price, *# Price at exit*

'pnl': pnl, *# Profit/loss for the trade*

}

## 2. Position Entry Logic

### 2.1 Entry Conditions

def calculate\_trading\_metrics(predictions, data):

trades = []

position = None

entry\_price = None

entry\_date = None

for i in range(1, len(predictions)):

current\_price = data[i]

signal = predictions[i]

*# Opening new position*

if position is None and signal != 0:

position = signal *# Set position direction (1 or -1)*

entry\_price = current\_price *# Record entry price*

entry\_date = i *# Record entry date*

This code:

1. Waits for a non-zero signal (1 or -1)
2. Only enters when there's no existing position (position is None)
3. Records all entry details including price and timestamp

Example scenario:

* Current position: None
* New signal: 1 (Buy)
* Price: $100
* Result: Opens long position at $100

## 3. Position Exit Logic

### 3.1 Exit Conditions

python

Copy

*# Close position if signal changes*

if position is not None and signal != position:

trades.append({

'entry\_date': entry\_date,

'exit\_date': i,

'direction': position,

'entry\_price': entry\_price,

'exit\_price': current\_price,

'pnl': (current\_price - entry\_price) / entry\_price \* (1 if position == 1 else -1)

})

position = None

entry\_price = None

entry\_date = None

This code:

1. Checks for signal change (exit trigger)
2. Calculates profit/loss
3. Records trade details
4. Resets position tracking variables

Example scenarios:

1. Long Position Exit:
   * Entry: $100 (long)
   * Exit: $110
   * PnL: ($110 - $100) / $100 = +10%
2. Short Position Exit:
   * Entry: $100 (short)
   * Exit: $90
   * PnL: ($90 - $100) / $100 \* (-1) = +10%

## 4. Return Calculation Details

### 4.1 PnL Formula Breakdown

python

Copy

pnl = (current\_price - entry\_price) / entry\_price \* (1 if position == 1 else -1)

Let's break this down:

1. For Long Positions (position = 1):

python

Copy

*# Example: Buy at $100, sell at $110*

pnl = ($110 - $100) / $100 \* 1 = +10%

*# Example: Buy at $100, sell at $90*

pnl = ($90 - $100) / $100 \* 1 = -10%

1. For Short Positions (position = -1):

python

Copy

*# Example: Short at $100, cover at $90*

pnl = ($90 - $100) / $100 \* (-1) = +10%

*# Example: Short at $100, cover at $110*

pnl = ($110 - $100) / $100 \* (-1) = -10%

### 4.2 Trade Metrics Collection

python

Copy

metrics = {

'total\_trades': len(trades),

'winning\_trades': len([t for t in trades if t['pnl'] > 0]),

'win\_rate': len([t for t in trades if t['pnl'] > 0]) / len(trades),

'avg\_win': np.mean([t['pnl'] for t in trades if t['pnl'] > 0]),

'avg\_loss': np.mean([t['pnl'] for t in trades if t['pnl'] <= 0]),

'total\_return': sum(t['pnl'] for t in trades),

'avg\_trade\_duration': np.mean([t['exit\_date'] - t['entry\_date'] for t in trades]),

'max\_drawdown': min(t['pnl'] for t in trades),

'sharpe\_ratio': np.mean([t['pnl'] for t in trades]) / np.std([t['pnl'] for t in trades])

}

## 5. Position Management Example Flow

Here's a complete example flow:

*# Initial state*

position = None

entry\_price = None

entry\_date = None

*# Day 1: Buy signal*

signal = 1

price = 100

*# Opens long position*

position = 1

entry\_price = 100

entry\_date = 1

*# Day 2: Price rises, maintain signal*

signal = 1

price = 105

*# Holds position*

*# Day 3: Sell signal*

signal = -1

price = 110

*# Closes position and records trade*

pnl = (110 - 100) / 100 \* 1 *# = +10%*

*# Resets position tracking*

position = None

entry\_price = None

entry\_date = None

Referrence: perplexity.ai – used for resource minining  
claude.ai – helped debugging  
  
  
  
# goude

## Core Data Functions

### fetch\_stock\_data(symbol, start\_date, end\_date)

This function serves as the data acquisition layer of our system. It uses the yfinance library to download historical stock data for a given symbol between specified dates. The function returns a pandas DataFrame containing essential price data like Open, High, Low, Close, and Volume, with the index reset to make the date a regular column.

### calculate\_technical\_indicators(df)

This function enriches the raw price data with various technical indicators that traders commonly use for analysis. Here's what each indicator tells us:

1. RSI (Relative Strength Index):

- Measures momentum by comparing the magnitude of recent gains to recent losses

- Range: 0-100, with values above 70 typically indicating overbought conditions and below 30 indicating oversold conditions

2. MACD (Moving Average Convergence Divergence):

- Shows the relationship between two moving averages of the price

- Generates: MACD line and Signal line

- Helps identify trend changes and momentum

3. EMAs (Exponential Moving Averages):

- EMA\_20: Short-term trend (8-period)

- EMA\_50: Medium-term trend (13-period)

- Gives more weight to recent prices than simple moving averages

4. VWAP (Volume Weighted Average Price):

- Shows the average price weighted by volume

- Important for identifying fair value and potential support/resistance levels

5. Additional Technical Indicators:

- ATR (Average True Range): Measures volatility

- Bollinger Bands: Shows potential support/resistance levels based on volatility

- Stochastic Oscillator: Momentum indicator comparing closing price to price range

- OBV (On Balance Volume): Cumulative volume indicator showing buying/selling pressure

## Analysis and Classification

### create\_labels(df, threshold=0.02)

This function creates trading signals based on price movements:

- Returns > threshold: Label = 1 (Buy signal)

- Returns < -threshold: Label = -1 (Sell signal)

- Otherwise: Label = 0 (Hold/Neutral)

### prepare\_data(df, features)

Orchestrates the data preparation process by:

1. Calculating technical indicators

2. Creating labels

3. Removing any rows with missing values

4. Extracting feature values and labels

### LorentzianClassifier

A custom classifier that uses the Lorentzian distance metric for prediction:

1. Key Methods:

- fit(): Stores training data and labels

- predict\_proba(): Calculates class probabilities using k-nearest neighbors

- predict(): Returns the most likely class for each input

2. Advantage of Lorentzian Distance:

- Better handles outliers than Euclidean distance

- More suitable for financial data which often has fat-tailed distributions

## Visualization Functions

### plot\_results(y\_true, y\_pred, probabilities, pca, feature\_names)

Creates a comprehensive visualization dashboard with four key plots:

1. Confusion Matrix:

- Shows prediction accuracy across all classes

- Helps identify where the model makes mistakes

2. ROC Curves:

- Displays model performance at different classification thresholds

- Shows trade-off between true positive and false positive rates

3. Feature Importance:

- Based on PCA first component

- Helps identify which indicators are most influential

4. Explained Variance:

- Shows how much variance is captured by each principal component

- Helps determine optimal dimensionality reduction

### visualize\_stock\_with\_classification(stock\_data, cluster\_labels)

Creates a visual representation of the stock price with cluster assignments:

- Grey line: Base price movement

- Green points: Cluster 1 data points

- Red points: Cluster 0 data points

This helps identify market regimes and potential pattern changes.

### plot\_trade\_distribution(trades)

Visualizes the distribution of trading results:

1. Returns Distribution:

- Shows the spread of profitable and unprofitable trades

- Helps assess risk/reward characteristics

2. Duration Distribution:

- Shows how long trades typically last

- Helps optimize holding periods

## Performance Analysis

### calculate\_trading\_metrics(predictions, data)

Calculates comprehensive trading performance statistics:

- Win rate and total return

- Average win/loss size

- Trade durations

- Sharpe ratio and maximum drawdown

- Detailed trade log

### analyze\_cluster\_transitions(data, cluster\_labels)

Studies how the market moves between different regimes:

- Tracks cluster changes over time

- Calculates return statistics for different transitions

- Creates a transition probability matrix

## Main Workflow

The main() function ties everything together in this sequence:

1. Data Preparation:

- Fetches historical data

- Calculates technical indicators

- Prepares features and labels

2. Model Building:

- Scales the data

- Performs PCA for dimensionality reduction

- Determines optimal number of clusters

- Trains the Lorentzian classifier

3. Analysis:

- Makes predictions using time series cross-validation

- Calculates trading metrics

- Analyzes cluster transitions

4. Visualization:

- Creates various plots and charts

- Prints performance summaries

This system combines multiple analytical approaches (technical analysis, clustering, and classification) to create a comprehensive trading analysis framework. The use of both clustering and classification helps identify market regimes while generating specific trading signals, potentially leading to more robust trading strategies.