# **Price Momentum Strategy Documentation**

This document provides a detailed explanation of the Python code designed to implement a price momentum strategy. The core hypothesis of this strategy is that stocks which have performed well historically are likely to continue doing so in the future. The code explores both long-only and long-short variations of this strategy, using different metrics to identify top and bottom performing stocks, and employs Hierarchical Risk Parity (HRP) for weighting in the long-short approach.

Disclaimer: This is an AI generated documentation, created and formatted by me for the sole purpose of increasing understandability and as a part of good coding practice.

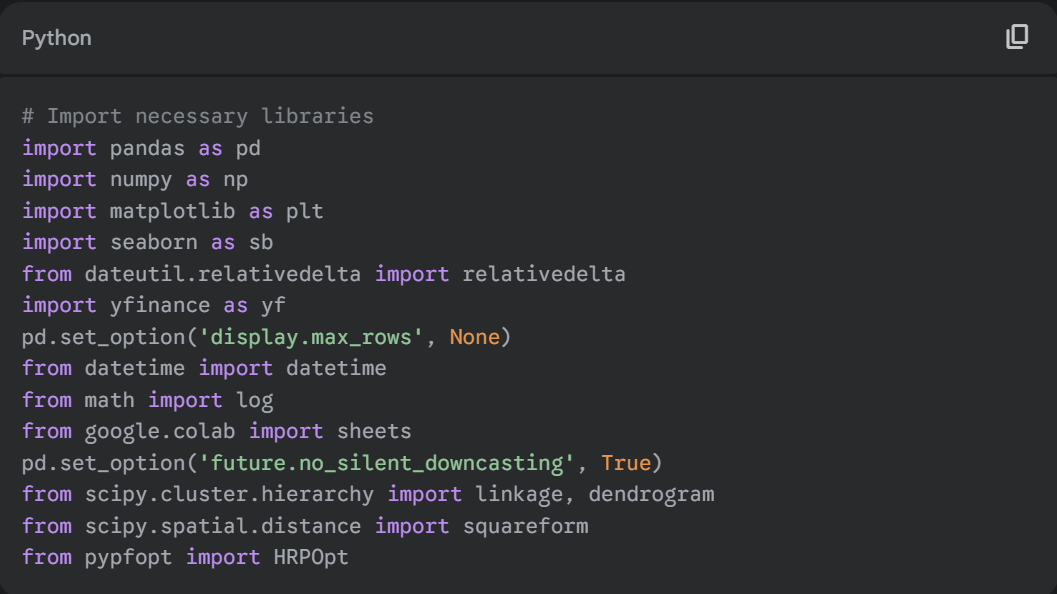
## **1. Project Overview**

The project aims to:

* Identify historically well-performing stocks based on cumulative returns, mean returns, risk-adjusted returns (Sharpe Ratio), and Volatility Trend.
* Implement a long-only momentum strategy where top-performing stocks are bought.
* Implement a long-short momentum strategy where top-performing stocks are bought (longed) and bottom-performing stocks are sold (shorted).
* Calculate and compare the performance of these strategies.
* Generate detailed trade sheets and portfolio performance summaries.

## **2. Setup and Libraries**

The code begins by importing necessary libraries for data manipulation, financial data download, and portfolio optimization.



**Imported Libraries:**

* **pandas (pd):** For data manipulation and analysis, especially DataFrames.
* **numpy (np):** For numerical operations, particularly array manipulation.
* **matplotlib.pyplot (plt):** For creating static, interactive, and animated visualizations (though not extensively used for plotting in the provided snippets).
* **seaborn (sb):** For statistical data visualization (though not extensively used in the provided snippets).
* **dateutil.relativedelta:** For performing date calculations involving relative differences.
* **yfinance (yf):** A popular library to download historical market data from Yahoo! Finance.
* **datetime:** Module from the datetime library for working with dates and times.
* **math.log:** For calculating the natural logarithm.
* **google.colab.sheets:** Used to create interactive Google Sheets directly from Colab (though most sheet generation calls are commented out in the provided code).
* **scipy.cluster.hierarchy.linkage:** For hierarchical clustering.
* **scipy.spatial.distance.squareform:** Converts a condensed distance matrix to a redundant square form.
* **pypfopt.HRPOpt:** Potentially for Hierarchical Risk Parity optimization (though a custom HRP implementation is present).

### **Global Variables and Settings:**

* **pd.set\_option('display.max\_rows', None):** Configures pandas to display all rows of a DataFrame without truncation.
* **pd.set\_option('future.no\_silent\_downcasting', True):** Sets a pandas option for future compatibility, preventing silent downcasting.
* **tickers:** A list of Indian stock tickers (NSE) used for the strategy.



* **holding\_period:** An integer representing the duration (in months) for which selected stocks are held. Initialized to 1.
* **skip\_period:** An integer representing the duration (in months) to skip between the formation period and the holding period. This is often used to avoid microstructure effects. Initialized to 1.

## **3. Core Functions for Return Calculation**

The code defines several functions to calculate different types of returns and risk metrics for individual tickers.

### **3.1. cum\_returns(ticker, form\_period, skip\_period, rolling\_date, holding\_period)**

Calculates the cumulative percentage return for a given ticker over a specified historical period.

* **ticker** (str): The stock ticker symbol.
* **form\_period** (int): The lookback period (in months) to consider for forming the portfolio.
* **skip\_period** (int): The period (in months) to skip after the formation period and before the holding period.
* **rolling\_date** (int): An offset in months to shift the entire calculation window backward in time. Used for backtesting over multiple periods.
* **holding\_period** (int): The period (in months) over which the cumulative return is calculated.

**Returns:**

* **log\_returns** (float): The cumulative percentage return of the stock over the specified period.

### **3.2. mean\_returns(ticker, form\_period, skip\_period, rolling\_date, holding\_period)**

Calculates the mean monthly log returns for a given ticker over a specified historical period.

* **ticker** (str): The stock ticker symbol.
* **form\_period** (int): The lookback period (in months) to consider for forming the portfolio.
* **skip\_period** (int): The period (in months) to skip after the formation period and before the holding period.
* **rolling\_date** (int): An offset in months to shift the entire calculation window backward in time.
* **holding\_period** (int): The period (in months) over which the mean monthly returns are calculated.

**Returns:**

* **float**: The mean of monthly log returns for the stock.

### **3.3. risk\_adj\_returns(ticker, form\_period, skip\_period, rolling\_date, holding\_period)**

Calculates the Sharpe Ratio (risk-adjusted return) for a given ticker over a specified historical period. It uses monthly log returns to compute the ratio of mean returns to the standard deviation of returns.

* **ticker** (str): The stock ticker symbol.
* **form\_period** (int): The lookback period (in months) to consider for forming the portfolio.
* **skip\_period** (int): The period (in months) to skip after the formation period and before the holding period.
* **rolling\_date** (int): An offset in months to shift the entire calculation window backward in time.
* **holding\_period** (int): The period (in months) over which the Sharpe Ratio is calculated.

**Returns:**

* **float**: The Sharpe Ratio of the stock.

### **3.4. volatility\_trend(ticker, form\_period, skip\_period, rolling\_date, holding\_period)**

Calculates the percentage change in volatility between the first and second halves of a specified period for a given ticker.

* **ticker** (str): The stock ticker symbol.
* **form\_period** (int): The lookback period (in months) to consider for forming the portfolio.
* **skip\_period** (int): The period (in months) to skip after the formation period and before the holding period.
* **rolling\_date** (int): An offset in months to shift the entire calculation window backward in time.
* **holding\_period** (int): The period (in months) over which the volatility trend is calculated.

**Returns:**

* **vol\_val** (float): The percentage change in volatility. Returns 0 if there isn't enough data or if the first half volatility is zero.

## **4. Return Parameters Dataframe Population**

The return\_params\_df function populates a DataFrame (return\_params) with the calculated cumulative returns, mean returns, risk-adjusted returns, and volatility trend for all specified tickers across various rolling dates.

### **4.1. return\_params**

An empty pandas DataFrame initialized to store return parameters.

* **Columns**: ['Date', 'Ticker', 'Cumulative Returns', 'Mean Returns', 'Risk Adjusted Returns', 'Volatility Trend'].

### **4.2. return\_params\_df(tickers, form\_period, skip\_period, rolling\_date, holding\_period)**

Iterates through the list of tickers and a range of rolling\_date values to calculate and append return metrics to the return\_params DataFrame.

* **tickers** (list of str): List of stock ticker symbols.
* **form\_period** (int): Lookback period for return calculations.
* **skip\_period** (int): Skip period for return calculations.
* **rolling\_date** (int): Current rolling date offset.
* **holding\_period** (int): Holding period for return calculations.

**Returns:**

* **return\_params** (pandas.DataFrame): The DataFrame containing calculated return parameters for all tickers across different dates.

The code then iteratively calls return\_params\_df for 12 rolling dates to populate return\_params.

## **5. Identifying Top and Bottom Performing Stocks**

After populating return\_params, the code identifies the top 5 and bottom 5 performing stocks for each rolling\_date based on the four different return metrics.

* **top\_by\_cumret**: Top 5 tickers by Cumulative Returns.
* **top\_by\_meanret**: Top 5 tickers by Mean Returns.
* **top\_by\_radret**: Top 5 tickers by Risk Adjusted Returns.
* **top\_by\_voltrend**: Top 5 tickers by Volatility Trend.
* **bottom\_by\_cumret**: Bottom 5 tickers by Cumulative Returns.
* **bottom\_by\_meanret**: Bottom 5 tickers by Mean Returns.
* **bottom\_by\_radret**: Bottom 5 tickers by Risk Adjusted Returns.
* **bottom\_by\_voltrend**: Bottom 5 tickers by Volatility Trend.

Note: In the provided code, only the results from the *last* iteration of the loop for rolling\_date will be printed for these variables.

## **6. Long-Only Strategy Implementation**

This section focuses on the long-only momentum strategy, where only top-performing stocks are bought.

### **6.1. initial\_cap**

* **initial\_cap** (int): Represents the initial capital for the strategy, set to 10000. (Note: This variable is defined but not actively used in the provided code snippets for calculations).

### **6.2. calculate\_weights(tickers, form\_period, skip\_period, rolling\_date, holding\_period)**

Calculates portfolio weights based on the inverse of monthly return volatility (standard deviation) for a given set of tickers. This implies a volatility-weighted allocation.

* **tickers** (list of str): List of stock ticker symbols for which to calculate weights.
* **form\_period** (int): Lookback period for volatility calculation.
* **skip\_period** (int): Skip period.
* **rolling\_date** (int): Rolling date offset.
* **holding\_period** (int): Holding period.

**Returns:**

* **weight\_n** (numpy.ndarray): An array of normalized weights for the input tickers.

### **6.3. backtest\_cum\_returns(tickers, form\_period, skip\_period, rolling\_date, holding\_period) (Long-Only Version)**

Calculates the weighted cumulative return for a portfolio of tickers over the holding\_period. This function is overloaded later for the long-short strategy.

* **tickers** (list of str): List of stock ticker symbols in the portfolio.
* **form\_period** (int): Lookback period.
* **skip\_period** (int): Skip period.
* **rolling\_date** (int): Rolling date offset.
* **holding\_period** (int): Holding period for calculating portfolio returns.

**Returns:**

* **float**: The weighted cumulative return of the portfolio.

### **6.4. Long-Only Trade Sheet Creation (trades\_sheet\_df, trades\_sheet\_df\_m, trades\_sheet\_df\_r, trades\_sheet\_df\_v)**

This section generates trade sheets for the long-only strategy based on each of the four performance metrics (Cumulative Returns, Mean Returns, Risk Adjusted Returns, and Volatility Trend).

* **trades\_sheet\_df**: DataFrame to store trade details for the strategy based on **Cumulative Returns**.
* **trades\_sheet\_df\_m**: DataFrame to store trade details for the strategy based on **Mean Returns**.
* **trades\_sheet\_df\_r**: DataFrame to store trade details for the strategy based on **Risk Adjusted Returns**.
* **trades\_sheet\_df\_v**: DataFrame to store trade details for the strategy based on **Volatility Trend**.

**Columns in Trade Sheets:**

* **Year** (str): The year of the trade.
* **Month** (str): The month of the trade.
* **stock** (str): The ticker symbol of the stock.
* **buy** (float): The price at which the stock was theoretically bought.
* **sell** (float): The price at which the stock was theoretically sold.
* **drawdown** (float): The maximum drawdown during the holding period.
* **upside** (float): The maximum upside (percentage change) during the holding period.
* **Monthly\_Return** (float): The log return over the holding period (treated as monthly).
* **Cumulative\_return** (float): The cumulative return of the stock in its formation period.
* **Mean\_Monthly\_Return** (float): The mean monthly return of the stock in its formation period.
* **Monthly\_Volatility** (float): Calculated as (1/Risk\_Adjusted\_Mean\_Return) \* Mean\_Monthly\_Return. *Note: This calculation is not a standard measure of monthly volatility; it appears to be a derivation.*
* **Risk\_Adjusted\_Mean\_Return** (float): The Sharpe Ratio of the stock in its formation period.
* **Weight\_norm** (float): The normalized weight assigned to the stock in the portfolio.

For each rolling\_date (simulating monthly rebalancing):

1. Top 5 stocks are selected based on the respective metric.
2. Weights are calculated using calculate\_weights.
3. Historical price data for the holding\_period is downloaded.
4. Trade details (buy/sell price, drawdown, upside, various returns, and normalized weight) are recorded.
5. A position is determined ('Buy' if weight.all() > 0, 'Sell' otherwise). *Note: The 'Sell' logic here seems incorrect for a long-only strategy as all weights would typically be positive. It might be a leftover from a long-short consideration.*

## **7. Performance Evaluation (Long-Only)**

### **7.1. returns\_performace(trades\_sheet\_df)**

Calculates the weighted average of 'Monthly\_Return' for a given trade sheet DataFrame, representing the overall performance of the strategy.

* **trades\_sheet\_df** (pandas.DataFrame): One of the long-only trade sheet DataFrames (trades\_sheet\_df, trades\_sheet\_df\_m, trades\_sheet\_df\_r, or trades\_sheet\_df\_v).

**Returns:**

* **weighted\_ret** (float): The weighted average of monthly returns.

### **7.2. performance\_df**

An empty DataFrame initialized to store the overall performance of each long-only strategy.

* **Columns**: ['Cumulative Returns', 'Mean Returns', 'Risk Adjusted Returns', 'Volatility Trend'].

The code then populates performance\_df by calling returns\_performace on each of the long-only trade sheets and prints which metric yielded the maximum returns and the corresponding value.

## **8. Long-Short Strategy Implementation**

This section introduces a long-short strategy, where top-performing stocks are longed and bottom-performing stocks are shorted. It uses Hierarchical Risk Parity (HRP) for weight allocation.

### **8.1. calculate\_weights\_LongShort(long\_tickers, short\_tickers, form\_period, skip\_period, rolling\_date, holding\_period=1)**

*Note: This function calculates weights based on inverse volatility for long and short positions and appears to be an older or alternative weighting method. The HRP-based calculate\_weights\_LongShort\_HRP is ultimately used for the long-short strategy.*

Calculates weights for long and short positions based on the inverse of their respective monthly volatilities. Short positions have negative inverse volatility.

* **long\_tickers** (list of str): Tickers for long positions.
* **short\_tickers** (list of str): Tickers for short positions.
* **form\_period** (int): Lookback period.
* **skip\_period** (int): Skip period.
* **rolling\_date** (int): Rolling date offset.
* **holding\_period** (int): Holding period.

**Returns:**

* **weight\_n** (numpy.ndarray): An array of normalized weights, where long positions have positive weights and short positions have negative weights.

### **8.2. calculate\_weights\_LongShort\_HRP(long\_tickers, short\_tickers, form\_period, skip\_period, rolling\_date, holding\_period=1)**

Calculates weights for long and short positions using the Hierarchical Risk Parity (HRP) algorithm. HRP aims to build a diversified portfolio by clustering assets based on their correlations and then allocating weights recursively.

* **long\_tickers** (list of str): Tickers for long positions.
* **short\_tickers** (list of str): Tickers for short positions.
* **form\_period** (int): Lookback period for HRP calculation.
* **skip\_period** (int): Skip period.
* **rolling\_date** (int): Rolling date offset.
* **holding\_period** (int): Holding period.

**Returns:**

* **final\_weights.values** (numpy.ndarray): An array of HRP-allocated weights, normalized separately for long and short positions. Long positions have positive weights, short positions have negative weights.

**Internal Steps of HRP:**

1. **Data Collection**: Downloads monthly 'Close' prices for all long\_tickers and short\_tickers.
2. **Return Calculation**: Computes monthly percentage returns.
3. **Covariance and Correlation**: Calculates the covariance and correlation matrices of the returns.
4. **Distance Matrix**: Converts the correlation matrix into a distance matrix using D\_i,j=sqrt0.5times(1−rho\_i,j).
5. **Hierarchical Clustering**: Performs hierarchical clustering on the distance matrix (using single linkage).
6. **Quasi-Diagonalization**: Orders the assets based on the hierarchical clusters to minimize the correlation within clusters.
7. **Recursive Bisection**: Recursively bisects the clusters and allocates weights based on the inverse variance of the sub-clusters.
8. **Normalization**: Normalizes long weights (sum to 1) and short weights (sum to -1) separately, then combines them.

### **8.3. backtest\_cum\_returns(long\_tickers, short\_tickers, form\_period, skip\_period, rolling\_date, holding\_period = 1) (Long-Short Version)**

Calculates the weighted cumulative return for a long-short portfolio over the holding\_period. This function is overloaded from the long-only version. Long positions contribute positively, and short positions contribute negatively to returns.

* **long\_tickers** (list of str): Tickers for long positions.
* **short\_tickers** (list of str): Tickers for short positions.
* **form\_period** (int): Lookback period.
* **skip\_period** (int): Skip period.
* **rolling\_date** (int): Rolling date offset.
* **holding\_period** (int): Holding period for calculating portfolio returns.

**Returns:**

* **float**: The weighted cumulative return of the long-short portfolio.

The code then calculates and prints the backtested returns for the long-short strategy using HRP weights, for portfolios formed by Cumulative, Mean, and Risk-Adjusted metrics.

### **8.4. Long-Short Trade Sheet Creation (trades\_sheet\_df\_ls, trades\_sheet\_df\_ls\_m, trades\_sheet\_df\_ls\_rad)**

This section generates trade sheets for the long-short strategy, similar to the long-only version, but now including both long and short positions and using HRP for weights.

* **trades\_sheet\_df\_ls**: DataFrame for long-short trades based on **Cumulative Returns**.
* **trades\_sheet\_df\_ls\_m**: DataFrame for long-short trades based on **Mean Returns**.
* **trades\_sheet\_df\_ls\_rad**: DataFrame for long-short trades based on **Risk Adjusted Returns**.

**Columns in Trade Sheets (same as long-only, plus 'Position'):**

* **Year** (str): The year of the trade.
* **Month** (str): The month of the trade.
* **stock** (str): The ticker symbol of the stock.
* **position** (str): 'Buy' for long positions, 'Sell' for short positions.
* **buy** (float): The price at which the stock was theoretically bought/shorted.
* **sell** (float): The price at which the stock was theoretically sold/covered.
* **drawdown** (float): The maximum drawdown during the holding period.
* **upside** (float): The maximum upside (percentage change) during the holding period.
* **Monthly\_Return** (float): The log return over the holding period.
* **Cumulative\_return** (float): The cumulative return of the stock in its formation period.
* **Mean\_Monthly\_Return** (float): The mean monthly return of the stock in its formation period.
* **Monthly\_Volatility** (float): *Same non-standard calculation as in long-only.*
* **Risk\_Adjusted\_Mean\_Return** (float): The Sharpe Ratio of the stock in its formation period.
* **Weight\_norm** (float): The normalized weight assigned to the stock (can be positive for long, negative for short).

For each rolling\_date:

1. Top 5 and Bottom 5 stocks are identified based on the respective metric.
2. HRP weights are calculated using calculate\_weights\_LongShort\_HRP. *Note: The weights are then divided by 2, possibly to represent a 50/50 allocation between the long and short sides of the portfolio.*
3. Trade details are recorded, including the position ('Buy' if weight\_norm > 0, 'Sell' if weight\_norm < 0).
4. Interactive sheets are generated for each of the long-short strategies (these are generally enabled and likely create Google Sheets).

## **9. Performance Evaluation (Long-Short)**

The returns\_performace function (already defined) is reused here to calculate the weighted average returns for the long-short strategies.

### **9.1. performance\_df\_longshort**

An empty DataFrame initialized to store the overall performance of each long-short strategy.

* **Columns**: ['Cumulative Returns', 'Mean Returns', 'Risk Adjusted Returns'].

The code then populates performance\_df\_longshort and prints which long-short strategy yielded the maximum returns and the corresponding value.

Individual interactive sheets for each stock within the best-performing long-short strategy are also generated.

## **10. Portfolio Sheet Generation**

This section generates monthly portfolio performance sheets, detailing the overall portfolio returns and the individual stock weights for each month, for both long-only and long-short strategies.

### **10.1. Portfolio Sheet Long Only (portfolio\_returns\_df)**

* **portfolio\_rows**: A list used to temporarily store data for each month's portfolio.
* **trades\_sheet\_df\_final**: Dynamically assigned to the long-only tradesheet DataFrame that yielded the maximum returns.
* **text**: Stores a string indicating the chosen metric (e.g., ' Cumulative Returns').
* Iterates through each rolling\_date (representing a month).
* Filters trades\_sheet\_df\_final to get trades for the current month.
* Calculates portfolio\_returns as the sum of (monthly return \* normalized weight) for that month's trades.
* Creates a dictionary row with 'Month', 'Year', and 'Portfolio\_Returns'.
* Adds the weight of \*all\* tickers to the row for the current month (0 if the stock was not in the portfolio).
* Appends row to portfolio\_rows.
* Finally, creates portfolio\_returns\_df from portfolio\_rows.
* An interactive sheet named 'Portfolio Sheet Long Only' is generated.

### **10.2. Portfolio Sheet Long Short (portfolio\_returns\_df\_ls)**

* **portfolio\_rows\_ls**: A list used to temporarily store data for each month's long-short portfolio.
* **trades\_sheet\_df\_final\_ls**: Dynamically assigned to the long-short tradesheet DataFrame that yielded the maximum returns.
* **text**: Stores a string indicating the chosen metric.
* The logic is identical to the long-only portfolio sheet generation, but it uses the long-short tradesheet data.
* An interactive sheet named 'Portfolio Sheet Long Short' is generated.

## **11. Conclusion**

This code provides a robust framework for backtesting price momentum strategies, incorporating different performance metrics and both long-only and long-short approaches. The use of yfinance allows for easy access to historical stock data, and the generation of detailed trade and portfolio sheets facilitates analysis of the strategy's performance over time. The implementation of Hierarchical Risk Parity for the long-short strategy aims to improve diversification and risk management compared to simpler weighting schemes.