Kronos-Backtester Documentation

Overview

"Kronos-Backtester" is a Python-based toolkit for financial data analysis and backtesting of trading strategies. It provides an effective platform for simulating trading strategies using historical data to evaluate their performance and potential profitability.

Installation Guide:

```bash
pip install kronos-backtester
```

This command installs "Kronos-Backtester" along with its necessary dependencies.

Requirements and Dependencies

- Python 3.x
- Pandas
- yfinance (for fetching financial data)
- Additional Python libraries as needed for specific strategies.

Code Documentation

1. Function: `momentum_trading_strategy` (Example Strategy)

- Description: This function demonstrates a momentum trading strategy. It is an example of how a strategy function should be structured for use with

"Kronos-Backtester." The function calculates moving averages over specified windows to generate trading signals.

- Parameters:
  - `data`: DataFrame with 'Date', 'Price', and additional columns for analysis.
  - `short_window`: Short-term moving average window.
  - `long_window`: Long-term moving average window.
  - `entry_threshold`: Minimum price change to trigger a buy/sell signal (percentage).
  - `exit_threshold`: Minimum price change to exit a position (percentage).
- Returns: DataFrame with buy/sell signals. The function must return:
  - `1` for a buy signal.
  - `-1` for a sell signal.
  - `0` to hold (no action).
- Note: This strategy serves as a template. Users can develop their own strategies ensuring they return the specified signal values.

2. Class: `Backtester`

- Method: `__init__`
  - Description: Initializes the Backtester class with a given strategy function.
  - Parameters:
    - `strategy`: A callable strategy function to be tested.
- Method: `testTickerReport`
  - Description: Tests the given strategy on a stock ticker over a period and generates a performance report.
  - Parameters: Ticker symbol, start date, end date.
  - Returns: A report detailing the performance of the strategy, including metrics like net worth, equity final, max drawdown, and Sharpe Ratio.

3. Function: `wrapper`

- Description: A wrapper function for the strategy function, setting default parameters for ease of use.
- Parameters: The wrapper function should accept the same parameters as the strategy function.
- Usage Example:

python Copy code

```python
bt = Backtester(wrapper)
test = bt.testTickerReport('AAPL', '2010-01-01', '2020-01-01')
for key in test:
    print(key, test[key])
```

Usage and Examples

Initializing the Backtester with a Strategy

To use "Kronos-Backtester," you first need to initialize it with a trading strategy. Here's how you can do it:

python Copy code

```python
from kronos_backtester import Backtester

# Define your strategy function here
def your_strategy_function(data):
    # Implementation of your strategy
    # ...

# Initialize the Backtester with your strategy
bt = Backtester(your_strategy_function)
```

After initializing, you can perform a backtest on historical stock data:

python Copy code

```python
# Perform a backtest on a specific stock ticker and date range
report = bt.testTickerReport('AAPL', '2010-01-01', '2020-01-01')

# Print the backtest report
for key, value in report.items():
    print(f"{key}: {value}")
```

Analyzing the Output and Understanding Performance Metrics

The backtest report includes various performance metrics. Key metrics to consider:

- *Net Worth:* Total value of the portfolio at the end of the backtest period.
- Equity Final: The final equity value.
- *Max Drawdown*: The maximum observed loss from a peak to a trough, before a new peak is attained.
- *Sharpe Ratio:* Measures the performance of the investment compared to a risk-free asset, after adjusting for its risk.

Troubleshooting and FAQs

Common Questions and Issues

- Q: What if I encounter an error regarding missing data?
    - A: Ensure that all required data fields are present in your dataset. Missing data can often lead to errors during the backtesting process.
- Q: How do I handle a strategy that requires multiple stock tickers?
    - A: Modify your strategy function to accept and process multiple tickers. Ensure that your backtester is provided with the correct data format.
- Q: The backtester is running very slow. How can I improve its performance?
    - A: Performance can be improved by optimizing your strategy code. Consider reducing the complexity of calculations or using efficient data structures.

License and Credits