

# Momentum Trading Strategy: MA Crossover

## Overview

This strategy implements and backtests a momentum trading approach using moving average crossovers and incorporates risk management techniques: volatility normalization and stop-loss mechanisms. Metrics such as the Sharpe Ratio and total PnL are calculated to evaluate performance. A rolling window backtesting framework ensures robustness and minimizes overfitting and lookahead bias.

## Strategy Workflow

1. **Data Acquisition:** Historical stock price data is fetched using `yfinance` for a specified ticker ( Apple Inc., `AAPL`) over 2020–2023. Key attributes include open, high, low, close prices, and trading volume.
2. **Preprocessing:** Daily returns are computed as the percentage change in closing prices. A rolling window approach splits the data into training and testing segments, ensuring that each training period consists only of historical data. (Prevent Looking Ahead bias)
3. **Hyperparameter Grid:** Applied Grid-Search on following parameters: Short/long MA windows, Stop-loss percentage, Volatility norm window.
4. **Training:** Generate Buy/Sell signals based on moving average historical prices, Signals are normalized using volatility-adjusted returns to determine position sizes (Buy if `Short.MA > Long.MA` and vice versa). A backtesting engine computes sharpe ratio for training data across hyperparameter combinations, the parameter set with the highest sharpe ratio is selected as the `best_params`.
5. **Testing:** Optimal parameters are applied to testing data. Signals for testing are shifted by one day to ensure that decisions are made only with information available up to the previous day, fully mitigating any potential lookahead bias. Performance matrix including sharpe ratio, maximum drawdown, and cumulative returns on the test data are calculated.
6. **Performance Metrics on Test Data:** A total of 36 rolling windows were created, each corresponding to a test interval of 21 tradable days. For each interval, key performance metrics such as the Sharpe ratio, maximum drawdown, and cumulative returns were calculated. The average results across all test intervals are as follows:
  - Average Test data Sharpe Ratio: 1.14
  - Average Test data Maximum Drawdown: 5.59%
  - Average Test data Cumulative Returns: 0.20%

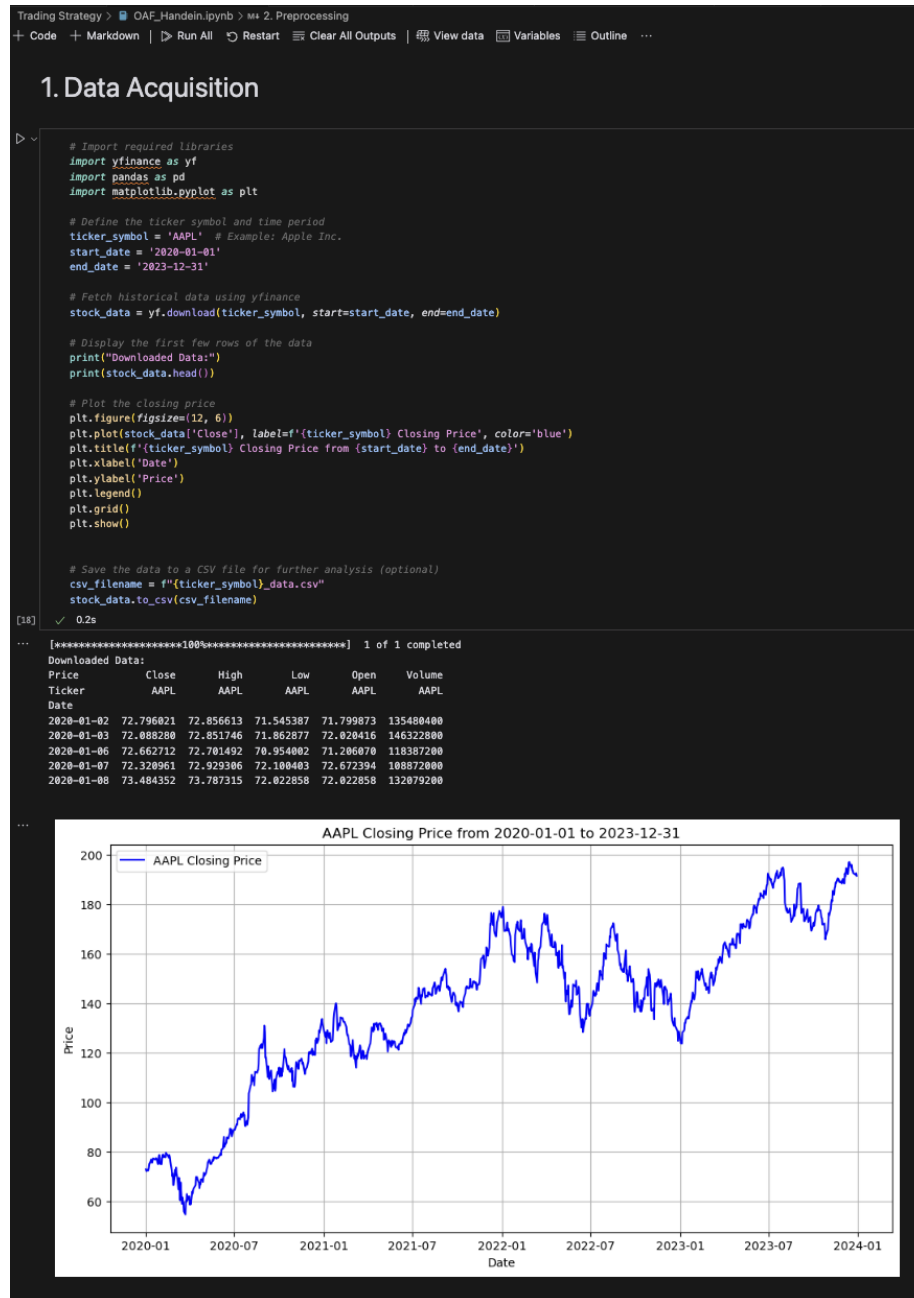


Figure 1: Data Acquisition

## 2. Preprocessing

```
# Include 'Date' column in the new DataFrame
new_df = pd.DataFrame({
    'Date': stock_data.index,
    'Open': stock_data['Open'].values.ravel(),
    'High': stock_data['High'].values.ravel(),
    'Low': stock_data['Low'].values.ravel(),
    'Close': stock_data['Close'].values.ravel(),
    'Volume': stock_data['Volume'].values.ravel()
})

new_df['Close']

stock_data = new_df

# Calculate daily returns
stock_data['Daily_Returns'] = stock_data['Close'].pct_change()
initial_capital = 100000 # Initial investment capital in USD
```

✓ 0.0s

## 3. Hyperparameter Grid

```
import pandas as pd
from sklearn.model_selection import ParameterGrid
import numpy as np

# Define hyperparameter grid
param_grid = {
    'short_window': [5, 10, 15],
    'long_window': [30, 50, 70],
    'stop_loss_pct': [0.02, 0.05, 0.1],
    'volatility_window': [10, 20, 30]
}

# Generate all combinations of hyperparameters
grid = ParameterGrid(param_grid)

# Set rolling window parameters
rolling_window_size = 252 # 1 year of trading days
step_size = 21 # Monthly step size
```

✓ 0.5s

Figure 2: Preprocessing and Hyperparameter Grid



## 4/5 Training and Testing

```
# Store results
results = []
count = 0
test_period_results = []

# Function to calculate maximum drawdown
def calculate_max_drawdown(portfolio_values):
    peak = portfolio_values[0]
    max_drawdown = 0
    for value in portfolio_values:
        if value > peak:
            peak = value
        drawdown = (peak - value) / peak
        max_drawdown = max(max_drawdown, drawdown)
    return max_drawdown

# Perform rolling window backtesting
for start_idx in range(0, len(stock_data) - rolling_window_size, step_size):
    # Define training and testing period
    train_data = stock_data.iloc[start_idx:start_idx + rolling_window_size].copy()
    test_data = stock_data.iloc[start_idx + rolling_window_size : max(param_grid['long_window']):start_idx + rolling_window_size + step_size].copy()

    # Evaluate each hyperparameter combination on the training data
    for params in grid:
        # Extract hyperparameters
        short_window = params['short_window']
        long_window = params['long_window']
        stop_loss_pct = params['stop_loss_pct']
        volatility_window = params['volatility_window']

        # Apply strategy on training data
        train_data['Short_MA'] = train_data['Close'].rolling(window=short_window).mean()
        train_data['Long_MA'] = train_data['Close'].rolling(window=long_window).mean()
        train_data['Signal'] = 0
        train_data.loc[train_data['Short_MA'] > train_data['Long_MA'], 'Signal'] = 1
        train_data.loc[train_data['Short_MA'] <= train_data['Long_MA'], 'Signal'] = -1
        train_data['Volatility'] = train_data['Daily_Returns'].rolling(window=volatility_window).std()
        train_data['Normalized_Returns'] = train_data['Daily_Returns'] / train_data['Volatility']
        train_data['Position_Size'] = train_data['Signal'] * train_data['Normalized_Returns']

        # Perform backtest on training data
        cash = initial_capital
        stock_value = 0
        quantity = 0
        portfolio_values = []

        for i in range(1, len(train_data)):
            if train_data['Signal'].iloc[i] == 1: # Buy signal
                entry_price = train_data['Close'].iloc[i]
                position_size = train_data['Position_Size'].iloc[i] * cash
                quantity = position_size / entry_price
                cash -= position_size
            elif train_data['Signal'].iloc[i] == -1: # Sell signal
                exit_price = train_data['Close'].iloc[i]
                stock_value = quantity * exit_price
                cash += stock_value
                quantity = 0

            # Calculate current stock value for unrealized gain/loss
            stock_value = quantity * train_data['Close'].iloc[i]
            total_capital = cash + stock_value
            portfolio_values.append(total_capital)

        max_drawdown = calculate_max_drawdown(portfolio_values)
        cumulative_returns = (portfolio_values[-1] - initial_capital) / initial_capital
        sharpe_ratio = np.mean(train_data['Daily_Returns'].dropna()) / np.std(train_data['Daily_Returns'].dropna()) * np.sqrt(252)

    # Store results
    results.append({
        'params': params,
        'Sharpe Ratio': sharpe_ratio,
        'Max Drawdown': max_drawdown,
        'Cumulative Returns': cumulative_returns
    })

# Apply the best hyperparameters to test data
best_params = max(results, key=lambda x: x['Sharpe Ratio'])['params']

# Apply strategy on test data
test_data['Short_MA'] = test_data['Close'].rolling(window=best_params['short_window']).mean()
test_data['Long_MA'] = test_data['Close'].rolling(window=best_params['long_window']).mean()
test_data['Signal'] = 0
test_data.loc[test_data['Short_MA'] > test_data['Long_MA'], 'Signal'] = 1
test_data.loc[test_data['Short_MA'] <= test_data['Long_MA'], 'Signal'] = -1
test_data['Volatility'] = test_data['Daily_Returns'].rolling(window=best_params['volatility_window']).std()
test_data['Normalized_Returns'] = test_data['Daily_Returns'] / test_data['Volatility']
test_data['Position_Size'] = test_data['Signal'] * test_data['Normalized_Returns']

test_data = test_data.iloc[max(param_grid['long_window']):].copy()
cash = initial_capital
stock_value = 0
quantity = 0
portfolio_values = []

for i in range(1, len(test_data)):
    if test_data['Signal'].iloc[i] == 1: # Buy signal
        entry_price = test_data['Close'].iloc[i]
        position_size = test_data['Position_Size'].iloc[i] * cash
        quantity = position_size / entry_price
        cash -= position_size
    elif test_data['Signal'].iloc[i] == -1: # Sell signal
        exit_price = test_data['Close'].iloc[i]
        stock_value = quantity * exit_price
        cash += stock_value
        quantity = 0

    # Calculate current stock value for unrealized gain/loss
    stock_value = quantity * test_data['Close'].iloc[i]
    total_capital = cash + stock_value
    portfolio_values.append(total_capital)

max_drawdown = calculate_max_drawdown(portfolio_values)
cumulative_returns = (portfolio_values[-1] - initial_capital) / initial_capital
sharpe_ratio = np.mean(test_data['Daily_Returns'].dropna()) / np.std(test_data['Daily_Returns'].dropna()) * np.sqrt(252)

test_period_results.append({
    'Sharpe Ratio': sharpe_ratio,
    'Max Drawdown': max_drawdown,
    'Cumulative Returns': cumulative_returns
})

# Calculate average metrics across test periods
average_sharpe_ratio = np.mean([result['Sharpe Ratio'] for result in test_period_results])
average_max_drawdown = np.mean([result['Max Drawdown'] for result in test_period_results])
average_cumulative_returns = np.mean([result['Cumulative Returns'] for result in test_period_results])

print("\nPerformance Summary Across Test Periods:")
print(f"Average Sharpe Ratio: {average_sharpe_ratio:.2f}")
print(f"Average Maximum Drawdown: {average_max_drawdown:.2%}")
print(f"Average Cumulative Returns: {average_cumulative_returns:.2%}")
```

✓ B0s

Performance Summary Across Test Periods:  
Average Sharpe Ratio: 1.14  
Average Maximum Drawdown: 5.59%  
Average Cumulative Returns: 6.20%

## Appendix 1: Data Acquisition

```
1 # Import required libraries
2 import yfinance as yf
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6 # Define the ticker symbol and time period
7 ticker_symbol = 'AAPL' # Example: Apple Inc.
8 start_date = '2020-01-01'
9 end_date = '2023-12-31'
10
11 # Fetch historical data using yfinance
12 stock_data = yf.download(ticker_symbol, start=start_date, end=
    end_date)
13
14 # Display the first few rows of the data
15 print("Downloaded Data:")
16 print(stock_data.head())
17
18 # Plot the closing price
19 plt.figure(figsize=(12, 6))
20 plt.plot(stock_data['Close'], label=f'{ticker_symbol} Closing Price',
    color='blue')
21 plt.title(f'{ticker_symbol} Closing Price from {start_date} to {
    end_date}')
22 plt.xlabel('Date')
23 plt.ylabel('Price')
24 plt.legend()
25 plt.grid()
26 plt.show()
27
28
29 # Save the data to a CSV file for further analysis (optional)
30 csv_filename = f"{ticker_symbol}_data.csv"
31 stock_data.to_csv(csv_filename)
```

## Appendix 2: Preprocessing

```
1 # Include 'Date' column in the new DataFrame
2 new_df = pd.DataFrame({
3     'Date': stock_data.index,
4     'Open': stock_data['Open'].values.ravel(),
5     'High': stock_data['High'].values.ravel(),
6     'Low': stock_data['Low'].values.ravel(),
7     'Close': stock_data['Close'].values.ravel(),
8     'Volume': stock_data['Volume'].values.ravel()
9 })
10
11 new_df['Close']
12
13 stock_data = new_df
14
15 # Calculate daily returns
```

```

16 stock_data['Daily>Returns'] = stock_data['Close'].pct_change()
17 initial_capital = 100000 # Initial investment capital in USD

```

## Appendix 3: Hyperparameter Grid

```

1 import pandas as pd
2 from sklearn.model_selection import ParameterGrid
3 import numpy as np
4
5 # Define hyperparameter grid
6 param_grid = {
7     'short_window': [5, 10, 15],
8     'long_window': [30, 50, 70],
9     'stop_loss_pct': [0.02, 0.05, 0.1],
10    'volatility_window': [10, 20, 30]
11 }
12
13 # Generate all combinations of hyperparameters
14 grid = ParameterGrid(param_grid)
15
16 # Set rolling window parameters
17 rolling_window_size = 252 # 1 year of trading days
18 step_size = 21 # Monthly step size

```

## Appendix 4: Training and Testing

```

1 # Store results
2 results = []
3 count = 0
4 test_period_results = []
5
6 # Function to calculate maximum drawdown
7 def calculate_max_drawdown(portfolio_values):
8     peak = portfolio_values[0]
9     max_drawdown = 0
10    for value in portfolio_values:
11        if value > peak:
12            peak = value
13            drawdown = (peak - value) / peak
14            max_drawdown = max(max_drawdown, drawdown)
15    return max_drawdown
16
17 # Perform rolling window backtesting
18 for start_idx in range(0, len(stock_data) - rolling_window_size,
19    step_size):
20    # Define training and testing period
21    train_data = stock_data.iloc[start_idx:start_idx +
22    rolling_window_size].copy()
23    test_data = stock_data.iloc[start_idx + rolling_window_size -
24    max(param_grid['long_window']):start_idx + rolling_window_size
25    + step_size].copy()
26
27

```

```

23 # Evaluate each hyperparameter combination on the training data
24 for params in grid:
25     # Extract hyperparameters
26     short_window = params['short_window']
27     long_window = params['long_window']
28     stop_loss_pct = params['stop_loss_pct']
29     volatility_window = params['volatility_window']
30
31     # Apply strategy on training data
32     train_data['Short_MA'] = train_data['Close'].rolling(window
=short_window).mean()
33     train_data['Long_MA'] = train_data['Close'].rolling(window=
long_window).mean()
34     train_data['Signal'] = 0
35     train_data.loc[train_data['Short_MA'] > train_data['Long_MA
'], 'Signal'] = 1
36     train_data.loc[train_data['Short_MA'] <= train_data['
Long_MA'], 'Signal'] = -1
37     train_data['Volatility'] = train_data['Daily_Returns'].
rolling(window=volatility_window).std()
38     train_data['Normalized_Returns'] = train_data['
Daily_Returns'] / train_data['Volatility']
39     train_data['Position_Size'] = train_data['Signal'] *
train_data['Normalized_Returns']
40
41     # Perform backtest on training data
42     cash = initial_capital
43     stock_value = 0
44     quantity = 0
45     portfolio_values = []
46
47     for i in range(1, len(train_data)):
48         if train_data['Signal'].iloc[i] == 1: # Buy signal
49             entry_price = train_data['Close'].iloc[i]
50             position_size = train_data['Position_Size'].iloc[i]
51
52             * cash
53             quantity += position_size / entry_price
54             cash -= position_size
55             elif train_data['Signal'].iloc[i] == -1: # Sell signal
56                 exit_price = train_data['Close'].iloc[i]
57                 stock_value = quantity * exit_price
58                 cash += stock_value
59                 quantity = 0
60
61             # Calculate current stock value for unrealized gain/
62             loss
63             stock_value = quantity * train_data['Close'].iloc[i]
64             total_capital = cash + stock_value
65             portfolio_values.append(total_capital)
66
67             max_drawdown = calculate_max_drawdown(portfolio_values)
68             cumulative_returns = (portfolio_values[-1] -
initial_capital) / initial_capital
69             sharpe_ratio = np.mean(train_data['Daily_Returns'].dropna()
) / np.std(train_data['Daily_Returns'].dropna()) * np.sqrt(252)
70
71             # Store results

```



```

69         results.append({
70             'params': params,
71             'Sharpe Ratio': sharpe_ratio,
72             'Max Drawdown': max_drawdown,
73             'Cumulative Returns': cumulative_returns
74         })
75
76     # Apply the best hyperparameters to test data
77     best_params = max(results, key=lambda x: x['Sharpe Ratio'])['
params']
78
79     # Apply strategy on test data
80     test_data['Short_MA'] = test_data['Close'].rolling(window=
best_params['short_window']).mean()
81     test_data['Long_MA'] = test_data['Close'].rolling(window=
best_params['long_window']).mean()
82     test_data['Signal'] = 0
83     test_data.loc[test_data['Short_MA'] > test_data['Long_MA'], '
Signal'] = 1
84     test_data.loc[test_data['Short_MA'] <= test_data['Long_MA'], '
Signal'] = -1
85     test_data['Volatility'] = test_data['Daily>Returns'].rolling(
window=best_params['volatility_window']).std()
86     test_data['Normalized>Returns'] = test_data['Daily>Returns'] /
test_data['Volatility']
87     test_data['Position_Size'] = test_data['Signal'] * test_data['
Normalized>Returns']
88
89     test_data = test_data.iloc[max(param_grid['long_window']):].
copy()
90     cash = initial_capital
91     stock_value = 0
92     quantity = 0
93     portfolio_values = []
94
95     for i in range(1, len(test_data)):
96         if test_data['Signal'].iloc[i] == 1: # Buy signal
97             entry_price = test_data['Close'].iloc[i]
98             position_size = test_data['Position_Size'].iloc[i] *
cash
99             quantity += position_size / entry_price
100             cash -= position_size
101         elif test_data['Signal'].iloc[i] == -1: # Sell signal
102             exit_price = test_data['Close'].iloc[i]
103             stock_value = quantity * exit_price
104             cash += stock_value
105             quantity = 0
106
107         # Calculate current stock value for unrealized gain/loss
108         stock_value = quantity * test_data['Close'].iloc[i]
109         total_capital = cash + stock_value
110         portfolio_values.append(total_capital)
111
112     max_drawdown = calculate_max_drawdown(portfolio_values)
113     cumulative_returns = (portfolio_values[-1] - initial_capital) /
initial_capital
114     sharpe_ratio = np.mean(test_data['Daily>Returns'].dropna()) /

```

```

115     np.std(test_data['Daily_Returns'].dropna()) * np.sqrt(252)
116     test_period_results.append({
117         'Sharpe Ratio': sharpe_ratio,
118         'Max Drawdown': max_drawdown,
119         'Cumulative Returns': cumulative_returns
120     })
121
122 # Calculate average metrics across test periods
123 average_sharpe_ratio = np.mean([result['Sharpe Ratio'] for result
124     in test_period_results])
124 average_max_drawdown = np.mean([result['Max Drawdown'] for result
125     in test_period_results])
125 average_cumulative_returns = np.mean([result['Cumulative Returns']
126     for result in test_period_results])
126
127 print("\nPerformance Summary Across Test Periods:")
128 print(f"Average Sharpe Ratio: {average_sharpe_ratio:.2f}")
129 print(f"Average Maximum Drawdown: {average_max_drawdown:.2%}")
130 print(f"Average Cumulative Returns: {average_cumulative_returns
131     :.2%}")

```