Momentum Trading Strategy

Giulio Caputi, December 2024

1. Introduction

A momentum trading strategy seeks to profit by following trends. In this specific example, we develop a moving-average crossover algorithm, i.e., a strategy that uses two moving averages (a "short" and a "long" window) to generate trade signals. Specifically, we buy when the short-term moving average (SMA) is above the long-term moving average (LMA), and we sell when the opposite happens. We manage risk by incorporating volatility-based position sizing (dynamically adjusting positions in response to volatility changes) and stop-losses. Lastly, we avoid look-ahead bias by shifting the signals by one day, ensuring that buy/sell decisions are made only with information actually available at the time of trading.

2. Setting the Parameters

We test this strategy with Intel Corp over the 5-year period from 2019-01-01 to 2023-12-31. We choose 50 days as the window for our short moving average, and 200 days for the long moving average (a common choice). As for adjusting position sizing, we examine the past 20 days to estimate volatility. We place a stop-loss 15% below the entry price for long positions (and above the entry price for short positions). Our initial capital is $100,000, and we assume no prior position when starting the backtest. Lastly, we work with a 1% transaction cost, which incorporates expenses like commissions, slippage, bid-ask spread, and other kinds of trading costs. The data is downloaded from the *yfinance* Python library.

3. Feature Engineering

For each day in our sample, we compute the daily return of the stock, the standard deviation of the previous 20 daily returns (as our estimate for volatility), the short-term and long-term moving averages of the stock price, and our trading signal. We want to be long when the SMA is above the LMA, otherwise we want to be short. We shift these signals by 1 day to avoid acting on data we do not have until after the close. To reduce overexposure during volatile markets, we size positions according to recent volatility. Specifically, our position size is $\frac{risk\_fraction * capital}{volatility * price}$, where $risk\_fraction$ is the percentage of capital we are willing to risk per trade (1% in our case), which is scaled by volatility (estimated as the rolling 20-day unbiased standard deviation of returns) and the asset's current price.

4. Backtest

We simulate the trading day-by-day, obtaining a trading signal for each day. We start with 100% in cash (so with no positions) and total equity equal to the initial capital. On each day, we observe the signal (+1 means being long, -1 means being short, and 0 means being neutral). If we already have a position, we check whether the price has hit the stop-loss threshold. If it has, we exit, setting the signal to 0 to indicate this. Each day t we compute equity as $equity_t = cash_t + (position_t * price_t)$. This updated equity becomes the starting point for the next day. We execute trades in this way:

- If signal == 1 → If we are currently short or flat, we close the short (if any), and then buy shares. The number of shares is determined by volatility-based sizing as explained above. Every time we trade, we deduct 1% of the notional value to take transaction costs (e.g., commissions, slippage, bid-ask spread) into account.
- If signal == -1 → If we are currently long or flat, we close the long (if any), and then sell short. Again, we adjust the position size dynamically, taking transaction costs into account.
- If signal == 0 → If we have a position (long or short), we close it. Also in this case we consider the impact of transaction costs.

5. Results

Throughout the backtest, we track daily equity, then compute the equity curve, total PnL (+228.68%), cumulative returns, the annualised Sharpe and Sortino ratios (0.67 and 1.04, respectively), the Compound Annual Growth Rate ($CAGR = (\frac{final\_equity}{initial\_capital})^{1/years} - 1 = 0.2691$), the maximum drawdown (-52.63%), and the Calmar Ratio ($\frac{CAGR}{max\_drawdown} = 0.51$). Results are summarised in Figures (a)-(e) and Table 1 below the Python code.

6. Limitations

Given the 1-page constraint, this work has several limitations. First, the choice of the moving average windows (50 and 200 days) is arbitrary, and we could use walk-forward optimisation or cross-validation to tune these parameters. Moreover, true shorting requires margin, borrow fees, and can be restricted under certain market conditions. Our simplified backtest ignores these real-world details. Additionally, we rely on a 20-day rolling standard deviation of daily returns, which might fail to fully capture future volatility, especially during rapidly changing market conditions. We also assume it is possible to buy/sell fractional shares, as in reality we should probably discretise the number of shares we trade. Furthermore, the 15% stop-loss is a simple approach, and in real trading we would use trailing stops or volatility-based stops. In addition, the analysis is limited to only one company, but in reality it should be applied to a wide range of products, dynamically adjusting weights. Lastly, we probably perform too many position adjustments, so we may decide to resize a position only when volatility changes by at least a certain percentage.

7. Python Code

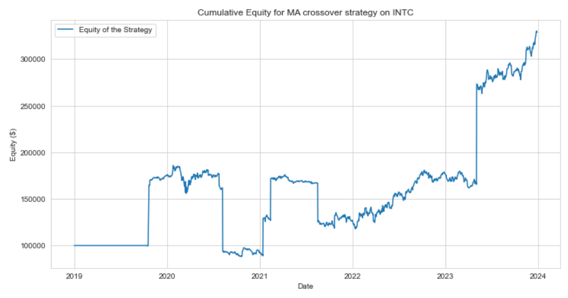
```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import yfinance as yf

sns.set_style('whitegrid')


# In[2]:


##############################
### SETTING THE PARAMETERS ###
##############################


ticker = 'INTC'
start_date = '2019-01-01'
end_date = '2023-12-31'

# These are the moving average windows
ma_short = 50
ma_long = 200

# We will adjust our positions based on the past 20-day volatility
vol_lookback = 20

# We set a stop-loss threshold of 15% the entry price
stop_loss_pct = 0.15

initial_capital = 100000

# We assume transaction costs of 1%. These include things like
# commissions, slippage, bid-ask spread, and other kinds of trading costs
transaction_costs = 0.01


# In[3]:


######################
### DATA RETRIEVAL ###
######################


df = yf.download(ticker, start=start_date, end=end_date)
df = df[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']].dropna()


# # Feature Engineering

# In[4]:
```

```
###########################
### FEATURE ENGINEERING ###
###########################


df['Returns'] = df['Adj Close'].pct_change()

df['MA_Short'] = df['Adj Close'].rolling(ma_short).mean()
df['MA_Long'] = df['Adj Close'].rolling(ma_long).mean()

# This is used only for dynamic position sizing
df['Volatility'] = df['Returns'].rolling(vol_lookback).std(ddof=0)

# Signal: 1 if MA_Short > MA_Long, -1 if MA_Short < MA_Long, 0 otherwise
df['Signal'] = 0
df.loc[df['MA_Short'] > df['MA_Long'], 'Signal'] = 1
df.loc[df['MA_Short'] < df['MA_Long'], 'Signal'] = -1

# We shift signals by 1 day to avoid lookahead bias
df['Signal'] = df['Signal'].shift(1)

# This is for our volatility-based position sizing
risk_fraction = 0.01  # risk 1% of capital per trade
df['Position_Size'] = np.nan


# In[5]:


################
### BACKTEST ###
################


df['Position'] = 0.0       # how many shares we actually hold
df['Cash'] = 0.0           # how much cash we hold
df['Equity'] = 0.0         # total equity = cash + position * price
df['Trade_Price'] = np.nan # price at which we entered the trade
df['Stop_Loss'] = np.nan   # stop-loss price

capital = initial_capital
position = 0
entry_price = 0
stop_loss = 0

for i in range(len(df)):
    price = df['Adj Close'].iloc[i]

    if i == 0:
        # On the first day we initialise as follows
        df.at[df.index[i], 'Cash'] = capital
        df.at[df.index[i], 'Position'] = 0
        df.at[df.index[i], 'Equity'] = capital
        continue

    # This retrieves yesterday's equity, position, and cash
    prev_equity = df['Equity'].iloc[i-1]
    prev_position = df['Position'].iloc[i-1]
    prev_cash = df['Cash'].iloc[i-1]

    signal = df['Signal'].iloc[i]
    vol = df['Volatility'].iloc[i]
```

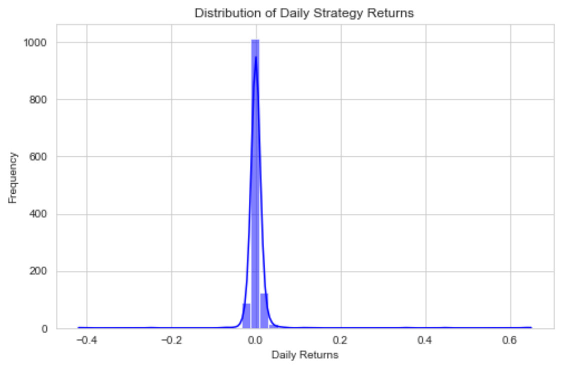
```python
# We compute today's position size (in number of shares) based on yesterday's equity
if vol == 0 or np.isnan(vol):
    position_size = prev_position
else:
    position_size = (risk_fraction * prev_equity) / (vol * price)

# Here we implement our stop-loss
if prev_position != 0:
    # If we're long and price < stop_loss, we exit
    if prev_position > 0 and price < stop_loss:
        signal = 0
    # If we're short and price > stop_loss, we exit
    if prev_position < 0 and price > stop_loss:
        signal = 0

# Here we implement the trading logic
if signal == 1:
    # If we are not already long, we go long
    if prev_position <= 0:
        # 1) Close any short positions first
        realized_pnl = 0
        if prev_position < 0:
            # Closing short => buy back shares
            # Number of shares to buy back = abs(prev_position)
            # PnL from the short = (Entry_Price - Current_Price) * abs(Position)
            shares_to_buy = abs(prev_position)
            close_amount = shares_to_buy * price
            commission_closing = close_amount * transaction_costs
            realized_pnl = (prev_position * (price - df['Trade_Price'].iloc[i-1]))
            prev_cash = prev_cash + realized_pnl - commission_closing

        # 2) Open the new long
        new_position = position_size
        cost = new_position * price
        commission_opening = cost * transaction_costs
        total_outflow = cost + commission_opening

        if total_outflow > prev_cash:
            new_position = (prev_cash / (price * (1 + transaction_costs)))
            cost = new_position * price
            commission_opening = cost * transaction_costs
            total_outflow = cost + commission_opening

        new_cash = prev_cash - total_outflow
        entry_price = price
        stop_loss = price * (1 - stop_loss_pct)
        position = new_position

    else:
        # If we are already long, we don't do anything
        new_cash = prev_cash
        position = prev_position
        entry_price = df['Trade_Price'].iloc[i-1]
        stop_loss = df['Stop_Loss'].iloc[i-1]

elif signal == -1:
    # If we are not already short, we go short
    if prev_position >= 0:
        # 1) Close any long positions first
        realized_pnl = 0
        if prev_position > 0:
                # Sell the shares
                shares_to_sell = prev_position
                sell_amount = shares_to_sell * price
                commission_closing = sell_amount * transaction_costs
                realized_pnl = (prev_position * (price - df['Trade_Price'].iloc[i-1]))
                prev_cash = prev_cash + realized_pnl - commission_closing

            # 2) Open new short
            new_position = -position_size
            proceeds = abs(new_position) * price
            commission_opening = proceeds * transaction_costs
            total_inflow = proceeds - commission_opening

            new_cash = prev_cash + total_inflow
            entry_price = price
            stop_loss = price * (1 + stop_loss_pct)
            position = new_position

        else:
            # In case we are already short, we don't do anything
            new_cash = prev_cash
            position = prev_position
            entry_price = df['Trade_Price'].iloc[i-1]
            stop_loss = df['Stop_Loss'].iloc[i-1]

    else:
        # if the signal is 0, we want to be flat
        # If we have a position, close it
        if prev_position != 0:
            if prev_position > 0:
                # We sell all shares at current price
                sell_amount = prev_position * price
                commission_close = sell_amount * transaction_costs
                realized_pnl = prev_position * (price - df['Trade_Price'].iloc[i-1])
                new_cash = prev_cash + realized_pnl - commission_close
            else:
                # We buy back short
                shares_to_buy = abs(prev_position)
                buy_amount = shares_to_buy * price
                commission_close = buy_amount * transaction_costs
                realized_pnl = prev_position * (price - df['Trade_Price'].iloc[i-1])
                new_cash = prev_cash + realized_pnl - commission_close
            position = 0
            entry_price = np.nan
            stop_loss = np.nan
        else:
            # We remain flat
            new_cash = prev_cash
            position = prev_position
            entry_price = df['Trade_Price'].iloc[i-1]
            stop_loss = df['Stop_Loss'].iloc[i-1]

    df.at[df.index[i], 'Position'] = position
    df.at[df.index[i], 'Trade_Price'] = entry_price
    df.at[df.index[i], 'Stop_Loss'] = stop_loss
    df.at[df.index[i], 'Cash'] = new_cash

    equity = new_cash + position * price
    df.at[df.index[i], 'Equity'] = equity


# In[6]:
```

```
###########################
### PERFORMANCE METRICS ###
###########################


df['Equity_Change'] = df['Equity'].pct_change().fillna(0)
df['Strategy_Returns'] = df['Equity_Change']
df['Cumulative_Strategy_Returns'] = (1 + df['Strategy_Returns']).cumprod() - 1

final_equity = df['Equity'].iloc[-1]
total_pnl = final_equity - initial_capital
pct_gain = total_pnl / initial_capital

daily_return_mean = df['Strategy_Returns'].mean()
daily_return_std = df['Strategy_Returns'].std(ddof=1)

sharpe_ratio = (daily_return_mean / daily_return_std) * np.sqrt(252)

negative_returns = df['Strategy_Returns'][df['Strategy_Returns'] < 0]
downside_std = negative_returns.std(ddof=1)
sortino_ratio = (daily_return_mean / downside_std) * np.sqrt(252)

num_days = (df.index[-1] - df.index[0]).days
years = num_days / 365.25
cagr = (final_equity / initial_capital)**(1/years) - 1

df['RollingMax'] = df['Equity'].cummax()
df['Drawdown'] = df['Equity'] / df['RollingMax'] - 1
max_drawdown = df['Drawdown'].min()

calmar_ratio = (cagr) / abs(max_drawdown)


# In[7]:


########################
### PLOTTING RESULTS ###
########################


# Plot the equity resulting from the strategy
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Equity'], label='Equity of the Strategy')
plt.title(f'Cumulative Equity for MA crossover strategy on {ticker}')
plt.xlabel('Date')
plt.ylabel('Equity ($)')
plt.legend()
plt.show()

# Plot the drawdown
plt.figure(figsize=(12, 4))
plt.plot(df.index, df['Drawdown'] * 100, color='red', label='Drawdown (%)')
plt.title('Drawdown Over Time')
plt.xlabel('Date')
plt.ylabel('Drawdown (%)')
plt.legend()
plt.show()

# Plot daily strategy returns over time
```

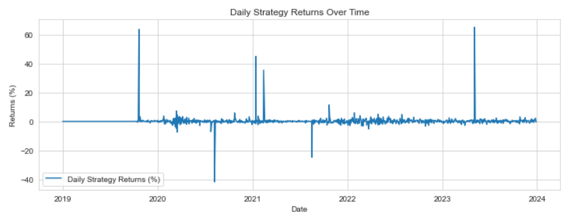
```
plt.figure(figsize=(12, 4))
plt.plot(df.index, df['Strategy_Returns']*100, label='Daily Strategy Returns (%)')
plt.title('Daily Strategy Returns Over Time')
plt.xlabel('Date')
plt.ylabel('Returns (%)')
plt.legend()
plt.show()

# Distribution of daily returns
plt.figure(figsize=(8, 5))
sns.histplot(df['Strategy_Returns'], bins=50, kde=True, color='blue')
plt.title('Distribution of Daily Strategy Returns')
plt.xlabel('Daily Returns')
plt.ylabel('Frequency')
plt.show()

# Rolling Sharpe ratio (computed with a 30-day window)
roll_window = 30
df['RollMean'] = df['Strategy_Returns'].rolling(roll_window).mean()
df['RollStd'] = df['Strategy_Returns'].rolling(roll_window).std(ddof=1)
df['Rolling_Sharpe'] = (df['RollMean'] / df['RollStd']) * np.sqrt(252)

plt.figure(figsize=(12, 4))
plt.plot(df.index, df['Rolling_Sharpe'], label=f'{roll_window}-Day Rolling Sharpe')
plt.axhline(0, color='black', linewidth=1)
plt.title(f'{roll_window}-Day Rolling Sharpe Ratio')
plt.xlabel('Date')
plt.ylabel('Sharpe Ratio')
plt.legend()
plt.show()


# In[8]:


print(f"Ticker: {ticker}")
print(f"Period: {start_date} to {end_date}")
print(f"MA Short: {ma_short}, MA Long: {ma_long}")
print(f"Transaction costs: {transaction_costs*100:.2f}% per trade")
print(f"Starting capital: ${initial_capital:,.2f}")
print(f"Final Equity: ${final_equity:,.2f}")
print(f"Total PnL: ${total_pnl:,.2f}")
print(f"Gain: {pct_gain*100:,.2f}%")
print(f"Daily Mean Return: {daily_return_mean*100:.3f}%")
print(f"Daily Std Dev: {daily_return_std*100:.3f}%")
print(f"Annualised Sharpe Ratio: {sharpe_ratio:.2f}")
print(f"Sortino Ratio: {sortino_ratio:.2f}")
print(f"CAGR: {cagr*100:.2f}%")
print(f"Calmar Ratio: {calmar_ratio:.2f}")
print(f"Max Drawdown: {max_drawdown*100:.2f}%")
```



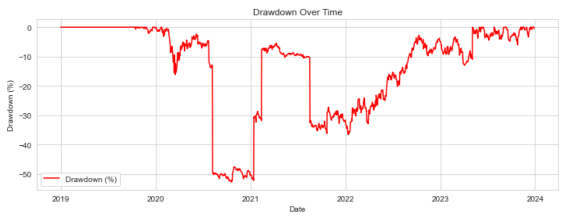
(a) Cumulative equity of the strategy



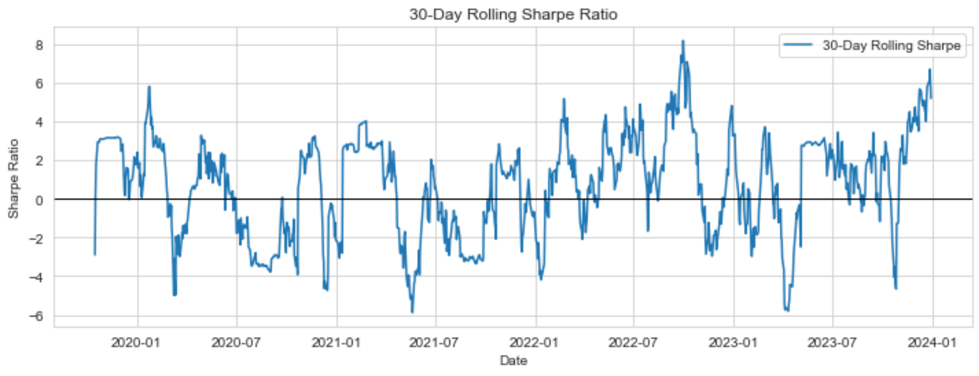
(b) Distribution of daily returns



(c) Daily returns



(d) Drawdown



(e) 30-day rolling Sharpe ratio

Ticker	INTC
Period	2019-01-01 to 2023-12-31
MA Short	50
MA Long	200
Transaction costs	1.00% per trade
Starting capital	\$100,000.00
Final Equity	\$328,676.81
Total PnL	\$228,676.81
Gain	228.68%
Daily Mean Return	0.147%
Daily Std Dev	3.495%
Annualised Sharpe Ratio	0.67
Sortino Ratio	1.04
CAGR	26.91%
Calmar Ratio	0.51
Max Drawdown	-52.63%

Table 1: Performance Metrics