

Algorithmic Trading

July 2025



Figure 1: Algorithmic Trading

Contents

1	WHAT IS FINANCIAL ANALYSIS AND QUANTITATIVE TRADING?	4
1.1	Financial Analysis	4
1.2	Quantitative Trading	4
2	STOCKS AND STOCK TRADING	4
2.1	Stocks	4
2.2	Stock Trading	4
3	EXTRACTING DATA FROM THE QUANDL API	5
3.1	QUANDL API FOR ALGORITHMIC TRADING	5
4	PROBLEM WITH EXTRACTING QUANDL API DATASET	6
5	EXTRACTING DATA FROM YAHOO FINANCE	6
6	EXPLORATORY DATA ANALYSIS ON STOCK PRICING DATA	7
6.1	Some important terminology	8
6.2	Getting a statistical summary of the data	9
6.3	Resampling the data	10
6.4	Calculating returns	11
7	MOVING AVERAGES IN TRADING	14
7.1	Calculating the rolling mean	14
8	FORMULATING A TRADING STRATEGY : Developing a momentum-based Simple Moving Average Crossover (SMAC) strategy	18
8.1	SMAC strategy	18
8.2	Lookback period	19
8.3	Buy and Sell signals	19
8.4	Implementing the code	19
8.4.1	Explanation of the code	21
8.5	Visualize the Performance of the Strategy on Quantopian	22
8.5.1	Key performance metrics explained	23
9	DESIGNING AND IMPLEMENTING A TRADING STRATEGY : SMAC + RSI hybrid	24
9.1	What is the strategy?	24
9.2	Logic behind the strategy	24
9.3	Why use this strategy?	25
9.4	How is the strategy implemented?	25

10 BACKTESTING THE TRADING STRATEGY	28
10.1 Conclusion	29
11 IMPROVING RETURNS OF THE TRADING STRATEGY	30
11.1 Tuning the parameters	30
11.2 Add a Signal Hold (Momentum Confirmation)	31
11.3 Change RSI Period (Momentum Tuning)	31
12 Conclusion and Future Work	32

1 WHAT IS FINANCIAL ANALYSIS AND QUANTITATIVE TRADING?

Financial analysis and quantitative trading are important aspects of the financial world today.

1.1 Financial Analysis

Financial analysis is the process of studying data such as stock prices, company reports, and market trends to make better investment decisions. It helps people understand how companies are performing and where the market might be headed.

1.2 Quantitative Trading

Quantitative trading, on the other hand, uses mathematics, statistics, and computer programs to trade stocks or other financial products. Traders create models that look for patterns in market data and use those patterns to decide when to buy or sell.

Thus, financial analysis provides the fundamental insights into a company's value, while quantitative trading uses mathematical models and data-driven strategies to exploit market inefficiencies—often building on the patterns identified through financial analysis.

2 STOCKS AND STOCK TRADING

2.1 Stocks

A stock is a representation of a share in the ownership of a corporation, which is issued at a certain amount. Companies issue stocks to raise funds to engage in projects that require more capital.

2.2 Stock Trading

Stock trading is the process of buying and selling existing and previously issued stocks. The price at which a stock can be bought or sold continues to fluctuate as demand and supply vary in the market. Traders use certain trading strategies and models to trade. One such strategy is short-selling: borrowing shares and immediately selling them in the hope of buying them later at a lower price, returning them to the lender, and making the margin.

3 EXTRACTING DATA FROM THE QUANDL API

3.1 QUANDL API FOR ALGORITHMIC TRADING

Quandl API is a tool that can be used to access financial, economic, and alternative datasets for algorithmic trading and quantitative research. It allows users to programmatically fetch historical and real-time market data in formats such as JSON and CSV. It can be installed using the following steps :

1. To create an account on Quandl API and get the API key, the following steps may be followed :

- (a) Create an account on the website [Quandl API](#)
- (b) Download the Nasdaq Data Link Python package from [download](#)

2. In the terminal, create a new directory for the project using the following command:

```
1 mkdir <directory_name>
```

3. Check if you have Python3 and virtualenv installed else install it using the following links :
[install Python](#)
[installation of virtual environment](#)

4. Create a new Python 3 virtual environment using the following command :

```
1 virtualenv <env_name>  
2 source <env_name>/bin/activate
```

5. Install jupyter-notebook using pip if not already installed:

```
1 pip install notebook
```

6. Install the pandas, quandl, and numpy packages if not already installed:

```
1 pip install pandas quandl numpy
```

7. Run your Jupyter notebook from the terminal using the following command :

```
1 jupyter notebook
```

8. Run the following command in terminal to install the package :

```
1 pip install nasdaq-data-link
```

9. In your notebook, import the required libraries and packages as follows :

```
1 import quandl as q
2 import pandas as pd
3 import numpy as np
```

10. Set API key and fetch dataset using :

```
1 # set the API key
2 q.ApiConfig.api_key = "<API key>"
3
4 #send a get request to query Microsoft's end of day stock prices
   from 1st Jan, 2010 to 1st Jan, 2019
5 msft_data = q.get("EOD/MSFT", start_date="2010-01-01", end_date
   ="2019-01-01")
6
7 # look at the first 5 rows of the dataframe
8 msft_data.head()
```

4 PROBLEM WITH EXTRACTING QUANDL API DATASET

If there is some issue with extracting data from this dataset, an alternative is the Yahoo Finance dataset. The yfinance API is a powerful tool for accessing financial data from Yahoo Finance. It allows users to download historical market data, retrieve financial information, and perform various financial analyses. This API is widely used in finance, investment, and trading applications for its ease of use and comprehensive data coverage.

5 EXTRACTING DATA FROM YAHOO FINANCE

To setup Yahoo Finance, use the following steps :

1. run the following command on terminal :

```
1 pip install yfinance
```

2. import library in Jupyter notebook using :

```
1 import yfinance as yf
```

- print the first 5 rows of requested historical stock price data for Microsoft using the following :

```
1 msft_data = yf.download("MSFT", start="2010-01-01", end
    ="2019-01-01")
2 print(msft_data.head())
```

```
[*****100%*****] 1 of 1 completed
```

Price	Close	High	Low	Open	Volume
Ticker	MSFT	MSFT	MSFT	MSFT	MSFT
Date					
2010-01-04	23.211443	23.323938	22.941455	22.963954	38409100
2010-01-05	23.218935	23.323931	22.978946	23.136440	49749600
2010-01-06	23.076450	23.308939	22.888959	23.158946	58182400
2010-01-07	22.836460	23.023952	22.641469	22.971453	50559700
2010-01-08	22.993946	23.158938	22.678960	22.708960	51197400

Figure 2: First 5 rows of Microsoft's historical stock price data (2010–2019) fetched using the yfinance library.

6 EXPLORATORY DATA ANALYSIS ON STOCK PRICING DATA

Before proceeding, it is important to understand what the data represent and what information they hold. This can be done using the `.info()` command in Python helps us to understand what all the dataset contains.

```
[*****100%*****] 1 of 1 completed
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2264 entries, 2010-01-04 to 2018-12-31
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   (Close, MSFT)    2264 non-null   float64
1   (High, MSFT)     2264 non-null   float64
2   (Low, MSFT)      2264 non-null   float64
3   (Open, MSFT)     2264 non-null   float64
4   (Volume, MSFT)   2264 non-null   int64
dtypes: float64(4), int64(1)
memory usage: 106.1 KB
None
```

Figure 3: Output of `msft_data.info()` showing the structure of the stock dataset

As seen in the above figure, the DataFrame contains DatetimeIndex, which means we are dealing with time-series data, which is a sequence of snapshots of prices taken at consecutive, equally spaced intervals of time. In trading, EOD stock pricing data captures the movement of the certain parameters about a stock, such as the stock price, over a specified period of time with data points recorded at regular intervals.

6.1 Some important terminology

1. **Open** : Captures the opening price of the stock, which is the price at which a stock first trades when the market opens on a given day.
2. **Closing** : Captures the closing price of the stock, which is the final price at which a stock trades when the market closes on a given day.
3. **Adj_Open / Adj_Close** : An adjusted opening/closing price is a stock's price on any given day of trading that has been revised to include any dividend distributions, stock splits, and other corporate actions that occurred at any time before the next day's open.
4. **Volume** : It records the number of shares which are being traded on any given day of trading.
5. **High** : It tracks the highest price of the stock during a particular day of trading
6. **Low** : It tracks the lowest price of the stock during a particular day of trading
7. **Dividend** : It is a part of the company's profit that it decides to give to its shareholders (the people who own its shares) like a reward for investing in the company.
8. **Split** : A stock split is when a company divides its existing shares into more shares to make them more affordable for people to buy. The total value stays the same, but the number of shares increases.
9. **Adj_Low / Adj_High**: It is the lowest or highest price of the stock for the day adjusted to account for things like dividends and stock splits that might have happened later.
10. **Adj_Vol** : It is the number of shares traded on a particular day adjusted for any stock splits.

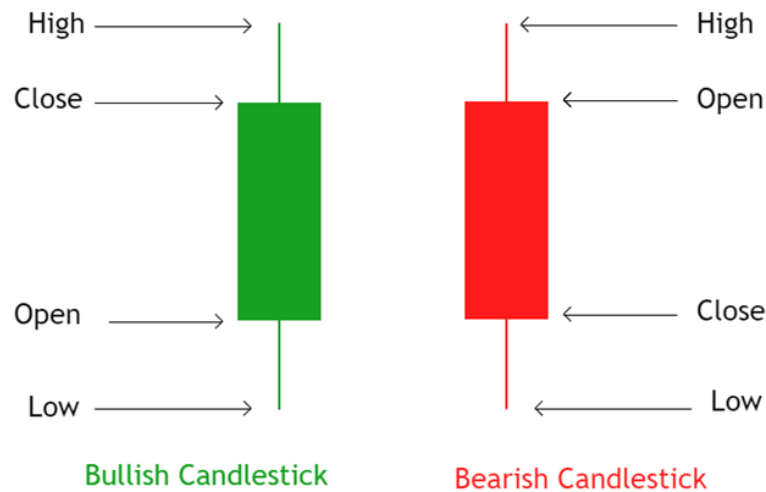


Figure 4: Candlestick chart illustrating Open, Close, High, and Low prices for a stock on a given day.

To better visualize the terminology of Open, Close, High, and Low, we refer to a candlestick chart. Each candlestick summarizes a single trading day, where:

1. The thick body shows the opening and closing prices.
2. The thin wicks (shadows) represent the day's high and low prices.
3. A green candle means the closing price was higher than the opening (bullish), while a red candle means it was lower (bearish).

This visual tool is essential for technical analysts as it quickly conveys market sentiment for each day.

6.2 Getting a statistical summary of the data

To get a statistical summary of the data, the following command can be run :

```
1 msft_data.describe()
```

It outputs a table consisting of some statistical records of the dataset such as mean, count, max, etc., which can give some superficial information about the dataset we are using.

[*****100%*****] 1 of 1 completed					
Price	Close	High	Low	Open	Volume
Ticker	MSFT	MSFT	MSFT	MSFT	MSFT
count	2264.000000	2264.000000	2264.000000	2264.000000	2.264000e+03
mean	41.835998	42.185959	41.450444	41.829326	4.163436e+07
std	23.656928	23.880381	23.421276	23.676843	2.348979e+07
min	17.415640	17.650274	17.203719	17.476194	7.425600e+06
25%	22.701962	22.892235	22.531265	22.707275	2.586055e+07
50%	35.238874	35.471747	34.940661	35.150745	3.620060e+07
75%	51.782544	52.015681	51.484373	51.818352	5.172612e+07
max	108.083809	108.616691	107.448066	107.906165	3.193179e+08

Figure 5: Statistical summary of Microsoft stock price dataset using the `.describe()` method.

6.3 Resampling the data

The `resample()` method in Pandas can be used to control how we organize time-based data, like grouping it by days, weeks, or months. As an example, running the following command :

```
1 msft_data.resample('ME').mean()
```

will tell Pandas to re-organize the data into monthly chunks, using the index (which is usually the Date in time-series data) to decide how to group the rows. The `.mean()` calculates the average of each numerical column (like Open, Close, etc.) for each month's group of days. Thus the `resample()` command allows us to analyse the data for specific timeframes.

```
[10]: msft_data.resample('ME').mean()
```

```
[10]:
```

	Price	Close	High	Low	Open	Volume
Ticker	MSFT	MSFT	MSFT	MSFT	MSFT	MSFT
Date						
2010-01-31	22.609099	22.937110	22.371873	22.714883	7.156057e+07	
2010-02-28	21.313821	21.469202	21.094702	21.289230	5.656017e+07	
2010-03-31	22.045631	22.210412	21.884126	22.041700	4.827118e+07	
2010-04-30	22.998142	23.169647	22.796857	22.934635	6.281093e+07	
2010-05-31	21.226182	21.603389	20.948449	21.353111	8.600651e+07	
...
2018-08-31	101.439971	101.924650	100.662176	101.224952	1.985340e+07	
2018-09-30	104.931227	105.593530	103.972218	104.781152	2.527661e+07	
2018-10-31	102.098004	103.803993	100.728985	102.644718	4.032817e+07	
2018-11-30	100.558963	101.658901	99.208348	100.517008	3.429660e+07	
2018-12-31	98.019299	100.033571	96.308454	98.610832	4.970077e+07	

108 rows × 5 columns

Figure 6: Sample output of `.resample()` method used to aggregate daily data into monthly means.

6.4 Calculating returns

Financial return is the money made or lost in an investment, which can be expressed nominally as the change in the amount of an investment over time. It can be calculated as the percentage derived from the ratio of profit to investment. For example, if one invests rupees 100 and later get rupees 110, the financial return is rupees 10. To better compare returns, we often convert this into a percentage. In this case, one makes rupees 10 on rupees 100, which is a 10% return. It simply shows how much the money has grown (or shrunk) over time.

We can calculate returns using the `pct_change()` command as follows :

```
1 # Import numpy package
2 import numpy as np
3
4 # assign 'Close' to 'daily_close'
5 daily_close = msft_data[['Close']]
6
7 # returns as fractional change
8 daily_return = daily_close.pct_change()
```

```

9
10 # replacing NA values with 0
11 daily_return.fillna(0, inplace=True)
12
13 print(daily_return)

```

Price Ticker	Close MSFT
Date	
2010-01-04	0.000000
2010-01-05	0.000323
2010-01-06	-0.006137
2010-01-07	-0.010400
2010-01-08	0.006896
...	...
2018-12-24	-0.041739
2018-12-26	0.068310
2018-12-27	0.006165
2018-12-28	-0.007808
2018-12-31	0.011754

[2264 rows x 1 columns]

Figure 7: Daily returns calculated from closing prices using the `pct_change()` method.

The formula used to calculate returns by this command is :

$$Return = \frac{(Price_t - Price_{t-1})}{Price_t}$$

To get monthly returns using this command, we can do the following:

```

1 mdata = msft_data.resample('ME').apply(lambda x: x.iloc[-1])
2 monthly_return = mdata.pct_change()
3
4 print(monthly_return)

```

Here the `.apply()` function has been used to get the last day of trading in the month

```
[39]: mdata = msft_data.resample('ME').apply(lambda x: x.iloc[-1])
monthly_return = mdata.pct_change()

print(monthly_return)
```

Price Ticker Date	Close MSFT	High MSFT	Low MSFT	Open MSFT	Volume MSFT
2010-01-31	NaN	NaN	NaN	NaN	NaN
2010-02-28	0.022146	-0.031253	0.035551	-0.037325	-0.791784
2010-03-31	0.021626	0.030156	0.023150	0.034555	0.579367
2010-04-30	0.042676	0.045760	0.046280	0.048245	-0.008551
2010-05-31	-0.151394	-0.155796	-0.155446	-0.164577	0.067739
...
2018-08-31	0.062993	0.060851	0.062338	0.052867	-0.160277
2018-09-30	0.018161	0.015872	0.019369	0.022383	-0.067817
2018-10-31	-0.066101	-0.056123	-0.072924	-0.076627	1.358780
2018-11-30	0.042684	0.030603	0.042153	0.054422	-0.340697
2018-12-31	-0.084047	-0.077228	-0.081566	-0.085005	-0.014608

[108 rows x 5 columns]

Figure 8: Monthly returns derived by resampling and applying percentage change

In addition to simple returns, we can also compute log returns using the formula :

$$Return = \ln\left(\frac{P_t}{P_{t-1}}\right)$$

Log returns are additive over time and are often preferred in mathematical modeling due to their statistical properties. While the values are similar for small changes, log returns are more accurate over longer time horizons or when volatility is high.

A comparison plot of simple vs log returns over the same stock data shows this subtle but important difference.

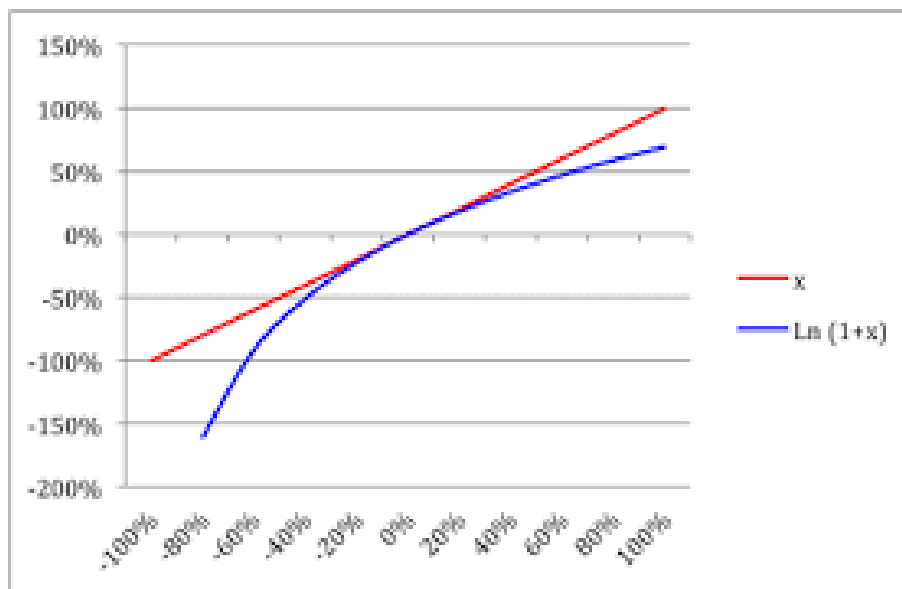


Figure 9: Comparison between simple and log returns

7 MOVING AVERAGES IN TRADING

Moving averages help us understand the general direction of a stock's price over time. In trading, people often look at the average price over the past few days, then move this time window forward day by day. This is called a moving window calculation.

7.1 Calculating the rolling mean

We can calculate the rolling mean using the `.rolling()` function. For example: To calculate the rolling mean over a window of 50 days and slide the window by 1 day, we can use the code given below.

```

1  # assigning closing prices to price
2  price = msft_data['Close']
3
4  # calculate the moving average
5  mav = price.rolling(window=50).mean()
6
7  # print the result
8  print(mav[-10:])

```

```

•[49]: # assigning closing prices to price
       price = msft_data['Close']

       # calculate the moving average
       mav = price.rolling(window=50).mean()

       # print the result
       print(mav[-10:])

```

Ticker	MSFT
Date	
2018-12-17	100.870918
2018-12-18	100.726741
2018-12-19	100.601239
2018-12-20	100.408435
2018-12-21	100.268094
2018-12-24	100.055435
2018-12-26	99.895090
2018-12-27	99.783221
2018-12-28	99.592944
2018-12-31	99.430249

Figure 10: 50-day rolling mean (Simple Moving Average) of Microsoft's stock prices to identify general trend.

The line `mav = price.rolling(window=50).mean()` creates a moving window of the last 50 values as it moves through the data set and takes the mean of those 50 values. The result is a new sequence whose first 49 elements are NaN and from the 50th element onward, it stores average of current and previous values.

To plot and see the difference, we use the matplotlib library as follows :

```

1 # import the matplotlib package to see the plot
2 import matplotlib.pyplot as plt
3 price.plot()

```

```
# import the matplotlib package to see the plot
import matplotlib.pyplot as plot
price.plot()
```

<Axes: xlabel='Date'>

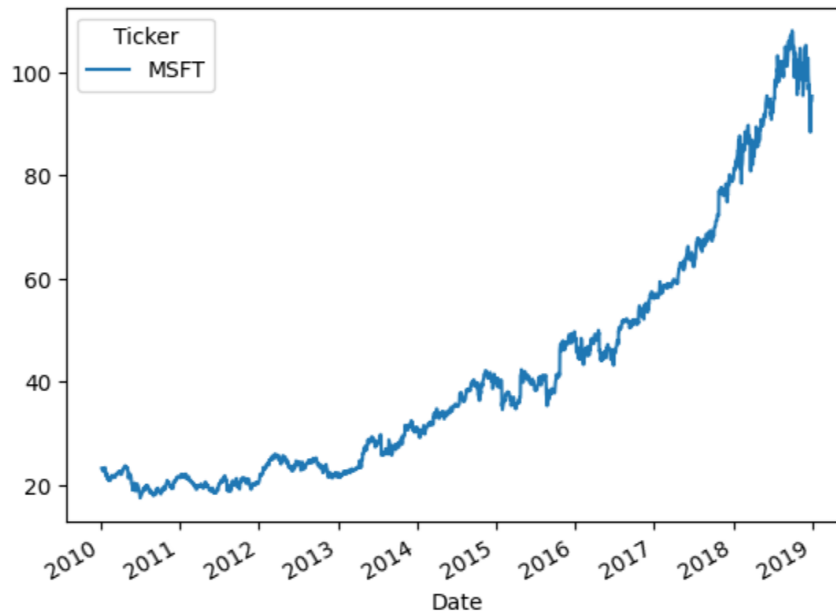


Figure 11: Microsoft stock price trend from 2010 to 2019, showing daily fluctuations.


```
mav.plot()
```

```
<Axes: xlabel='Date'>
```

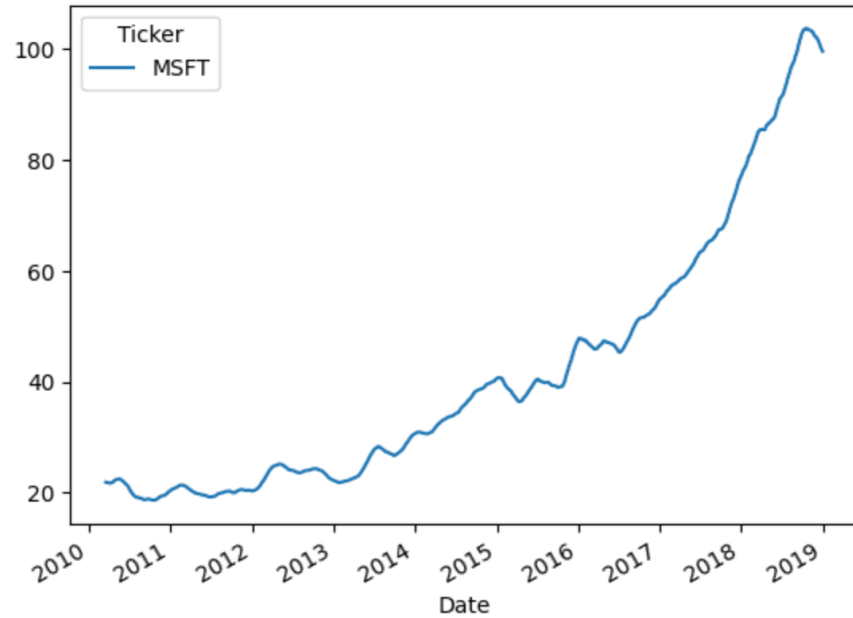


Figure 12: 50-day moving average of Microsoft stock price from 2010 to 2019, smoothing out short-term fluctuations.

From these graphs, we can easily interpret the general trend of the stock as the spikes in the data are consumed.

The choice of moving average window size significantly affects the strategy. Smaller windows (e.g., 20 days) react quickly to price changes but can produce false signals. Larger windows (e.g., 100 days) are more stable but slow to react.

A side-by-side comparison of 50-day, 100-day, and 200-day SMAs on the same price chart shows how responsive vs stable each curve is.



Figure 13: Comparison of 50-day, 100-day and 200-day SMAs

8 FORMULATING A TRADING STRATEGY : Developing a momentum-based Simple Moving Average Crossover (SMAC) strategy

Momentum-based strategies are a way of trading that puts emphasis on the idea that what is going up will likely go up and what is going down will likely going down. These strategies use technical indicators i.e. tools based on price and volume data, to identify trends.



Figure 14: Illustration of buy/sell logic in a momentum-based SMAC (Simple Moving Average Crossover) strategy.

In summary, we purchase securities that show an upwards trend and short-sell securities which show a downward trend.

8.1 SMAC strategy

The SMAC strategy is a long-only strategy i.e. it only buys stocks, not short-sells them. It uses the stock's price history to make decisions with no need for company reports, news, etc., assuming that past performance indicates future trend.

8.2 Lookback period

Lookback period is the amount of time we use to estimate the end. There are usually three types of lookback periods :

1. Short term : 1 week to 1 month
2. Intermediate term : a few months
3. Long term : a few years

In SMAC, we use a shorter period to capture the recent trend and a longer period to get the overall trend by calculating the moving average over each period.

8.3 Buy and Sell signals

Buy and Sell signals are where the "crossover" happens.

1. Buy signal : when the short-term moving average crosses above the long-term moving average i.e. recent prices are rising faster which means possible strong upward trend starting.
2. Sell signal : when the short-term moving average crosses below the long-term moving average i.e. recent prices are falling faster which means possible price drop.

8.4 Implementing the code

```
1 # initialize the short and long lookback periods
2 short_lb = 50
3 long_lb = 120
4
5 # initialize a new DataFrame called signal_df with the signal column
6 import pandas as pd
7 signal_df = pd.DataFrame(index=msft_data.index)
8 signal_df['signal'] = 0.0
9
10 # create a short simple moving average over the short lookback period
11 signal_df['short_mav'] = msft_data['Close'].rolling(window=short_lb,
12             min_periods=1, center=False).mean()
13
14 # create long simple moving average over the long lookback period
15 signal_df['long_mav'] = msft_data['Close'].rolling(window=long_lb,
16             min_periods=1, center=False).mean()
17
18 # generate the signals based on the conditional statement
19 signal_df['signal'][short_lb:] = np.where(signal_df['short_mav'][short_lb:]
20             > signal_df['long_mav'][short_lb:], 1.0, 0.0)
```

```

19 # create the trading orders based on the positions column
20 signal_df['positions'] = signal_df['signal'].diff()
21 signal_df[signal_df['positions'] == -1.0]

```

To visualize this :

```

1 fig = plt.figure()
2
3 # Add a subplot and label for y-axis
4 plt1 = fig.add_subplot(111, ylabel='Price in $')
5
6 msft_data['Close'].plot(ax=plt1, color='r', lw=2.)
7
8 # plot the short and long lookback moving averages
9 signal_df[['short_mav', 'long_mav']].plot(ax=plt1, lw=2., figsize=(12,8))
10
11 # plotting the sell signals
12 plt1.plot(signal_df.loc[signal_df.positions == -1.0].index,
13           signal_df.short_mav[signal_df.positions == -1.0],
14           'v', markersize=10, color='k')
15
16 # plotting the buy signals
17 plt1.plot(signal_df.loc[signal_df.positions == 1.0].index,
18           signal_df.short_mav[signal_df.positions == 1.0],
19           '^', markersize=10, color='m')
20
21 # Show the plot
22 plt.show()

```



Figure 15: SMAC strategy visualized using matplotlib: showing short and long moving averages with buy/sell signals.

8.4.1 Explanation of the code

1. We created two lookback periods, the shorter lookback period `short_lb` of 50 days and the longer lookback period `long_lb` of 120 days using :

```
1 short_lb = 50
2 long_lb = 120
```

2. We created a new dataframe named `signal_df` with the same index (dates) as `msft_data`. Further we add a column named `signal` where every entry is 0.0 (no signal). Our idea is to store buy and no-buy signals as 1 and 0 respectively.

```
1 import pandas as pd
2 signal_df = pd.DataFrame(index=msft_data.index)
3 signal_df['signal'] = 0.0
```

3. We calculate the rolling average of the closing price using a 50-day window which gives us short term moving average. It tracks the recent price trend. If it moves up faster than the long-term trend, it signals momentum.

```
1 signal_df['short_mav'] = msft_data['Close'].rolling(window=short_lb
, min_periods=1, center=False).mean()
```

4. Similarly, we calculate the long term moving average, which acts like a baseline.

```
1 signal_df['long_mav'] = msft_data['Close'].rolling(window=long_lb,  
    min_periods=1, center=False).mean()
```

5. We compare short term moving average and long term moving average. If short term is greater than long term, we give signal 1 i.e. buy signal

```
1 signal_df['signal'][short_lb:] = np.where(signal_df['short_mav'] [  
    short_lb:] > signal_df['long_mav'][short_lb:], 1.0, 0.0)
```

6. We then calculate the change in signal values from one day to the next. 1.0 means buy today and -1.0 means sell today. The last line filters and shows only the sell signals.

```
1 signal_df['positions'] = signal_df['signal'].diff()  
2 signal_df[signal_df['positions'] == -1.0]
```

So, in the graph, whenever the blue line (short term moving average) exceeds the orange line (long term moving average), there is a pink marker indicating to buy the stock and whenever the opposite happens, there is a black marker indicating selling the stock.

8.5 Visualize the Performance of the Strategy on Quantopian

Quantopian is a web-based platform built on top of Zipline, a Python library for backtesting trading strategies. It allows users to:

1. Write and test custom trading algorithms
2. Access historical market data for free
3. Backtest strategies using real market data
4. Share and collaborate with a community of quants

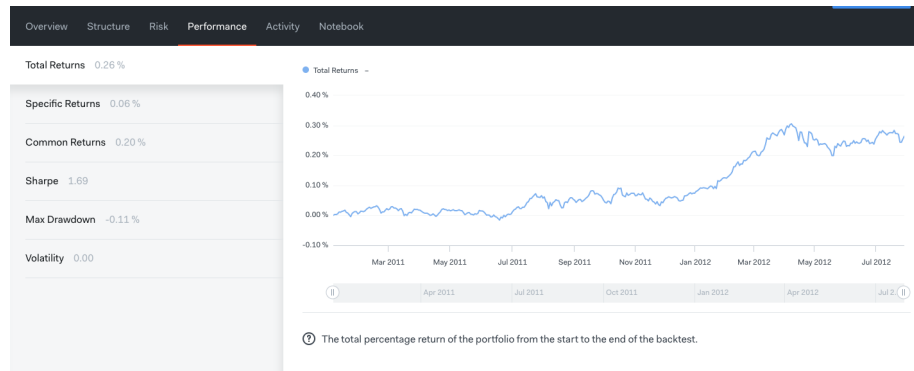


Figure 16: Quantopian backtest results of the SMAC strategy – showcasing returns and risk metrics.

8.5.1 Key performance metrics explained

1. Total return : tells how much the portfolio grew (or shrank) from the beginning to the end of the backtest. For example, a 40% return means the portfolio grew by 40%.
2. Specific Return : shows the part of the return that comes purely from strategy i.e. after removing returns that are due to broader market trends or common risks.
3. Common Return : return attributed to common market risk factors, such as sectors (like tech or healthcare) or styles (like value or growth investing). Quantopian breaks this down using 11 sector and 5 style factors.
4. Sharpe Ratio : measure of risk-adjusted return. It compares how much extra return one gets for each unit of risk. Quantopian calculates a 6-month rolling Sharpe ratio, which updates over time.
5. Max Drawdown : biggest drop from a high point to a low point in the portfolio's value. This shows the worst-case loss during the backtest.
6. Volatility : measures how much portfolio's returns fluctuate. A higher volatility means more ups and downs (more risk), while lower volatility means a smoother ride.

By running our SMAC strategy on Quantopian, we were able to understand not just how much return we got, but also how we got it, how risky it was, and how dependent it was on the overall market. These metrics help in refining and improving the strategy further.

9 DESIGNING AND IMPLEMENTING A TRADING STRATEGY : SMAC + RSI hybrid

9.1 What is the strategy?

The SMAC (Simple Moving Average Crossover) is a commonly used trend-following indicator. It smooths out price data by averaging a stock's price over a specified period. The Relative Strength Index (RSI) is a momentum oscillator that measures the speed and change of price movements. RSI values range from 0 to 100 and are typically used to identify overbought or oversold conditions.

9.2 Logic behind the strategy

Combine these two tools to create a more robust trading signal.

1. BUY signal :
 - (a) $RSI < 30$ i.e. stock is oversold
 - (b) price crosses above the SMA i.e. upward trend
2. SELL signal :
 - (a) $RSI > 70$ i.e. stock is overbought
 - (b) price crosses below the SMA i.e. downward trend

A plot of RSI with the 30/70 lines helps confirm whether a price move is gaining or losing momentum. Combining this with SMA crossover adds robustness to the strategy.



Figure 17: RSI plot with overbought oversold zones

This hybrid ensures we buy when the stock is gaining bullish momentum after being oversold and sell when it loses momentum after being overbought.

9.3 Why use this strategy?

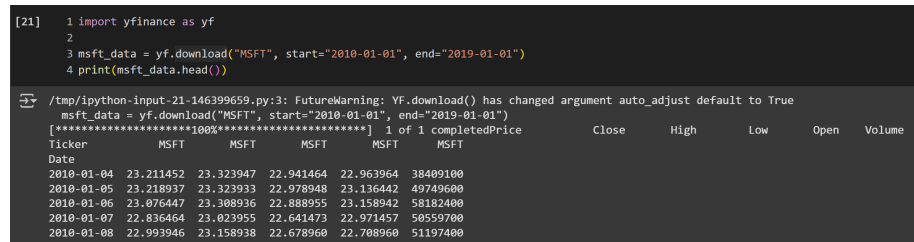
1. Complementary indicators: RSI detects momentum while SMA captures trends.
2. Fewer false signals: RSI confirms SMA-based trend reversals.

9.4 How is the strategy implemented?

1. Install the yfinance library using pip using the following command :

```
1 !pip install yfinance
```

2. Import the required dataset, in this case we have used msft dataset from yfinance



```
[21] 1 import yfinance as yf
2
3 msft_data = yf.download("MSFT", start="2010-01-01", end="2019-01-01")
4 print(msft_data.head())
```

```
/tmp/ipython-input-21-146399659.py:3: FutureWarning: YF.download() has changed argument auto_adjust default to True
msft_data = yf.download("MSFT", start="2010-01-01", end="2019-01-01")
[*****100%*****] 1 of 1 completedPrice      Close      High      Low      Open      Volume
Ticker
Date
2010-01-04  23.211452  23.323947  22.941464  22.963964  38409100
2010-01-05  23.218937  23.323933  22.978948  23.136442  49749600
2010-01-06  23.076447  23.308936  22.888955  23.158942  58182400
2010-01-07  22.836464  23.023955  22.641473  22.971457  50559700
2010-01-08  22.993946  23.158938  22.678960  22.708960  51197400
```

Figure 18: Importing Microsoft dataset (msft) from Yahoo Finance using yfinance for the RSI + SMAC hybrid strategy.

3. SMA CALCULATION :

- import the required libraries : namely numpy, matplotlib.pyplot and pandas

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
```

- Create 2 lookback periods, the shorter lookback period **short_lb** of 50 days and the longer lookback period **long_lb** of 120 days using :

```
1 short_lb = 50
2 long_lb = 120
```

- Create a new dataframe named **signal_df** with the same index (dates) as **msft_data**. Further add a column named signal where every entry is 0.0 (no signal).

```

1 import pandas as pd
2 signal_df = pd.DataFrame(index=msft_data.index)
3 signal_df['signal'] = 0.0

```

- Calculate the rolling average of the closing price using a 50-day window which gives us short term moving average. It tracks the recent price trend. If it moves up faster than the long-term trend, it signals momentum.

```

1 signal_df['short_mav'] = msft_data['Close'].rolling(window=
    short_lb, min_periods=1, center=False).mean()

```

- Similarly, calculate the long term moving average, which acts like a baseline.

```

1 signal_df['long_mav'] = msft_data['Close'].rolling(window=
    long_lb, min_periods=1, center=False).mean()

```

4. RSI calculation :

- Calculate difference between close price on consecutive dates using the `.diff()` command :

```

1 delta = msft_data['Close'].diff()

```

- Define the gain and loss as positive and negative delta respectively. In gains, the 0s correspond to losses and in loss, the 0s correspond to gains.

```

1 gain = delta.where(delta > 0, 0)
2 loss = -delta.where(delta < 0, 0)

```

- Calculate a 14 day rolling average of gain and loss

```

1 avg_gain = gain.rolling(window=14).mean()
2 avg_loss = loss.rolling(window=14).mean()

```

- Compute the relative strength i.e. the ratio of average gains to average loss

```

1 rs = avg_gain / avg_loss

```

- Convert RS into RSI value using the standard formula :

$$RSI = 100 - \left(\frac{100}{1 + rs} \right)$$

```

1 msft_data['RSI'] = 100 - (100 / (1 + rs))

```

5. Define the buy and sell condition :

- BUY condition :

```
1 buy_condition = (  
2     (signal_df['short_mav'] > signal_df['long_mav']) &  
3     (msft_data['RSI'] < 30)  
4 )
```

- SELL signal :

```
1 sell_condition = (  
2     (signal_df['short_mav'] < signal_df['long_mav']) &  
3     (msft_data['RSI'] > 70)  
4 )
```

6. Set buy and sell signals, identify trade entry/exit points and maintain positions between signals :

```
1 signal_df.loc[buy_condition, 'signal'] = 1.0  
2 signal_df.loc[sell_condition, 'signal'] = 0.0  
3  
4 signal_df['positions'] = signal_df['signal'].diff()  
5  
6 signal_df['signal'] = signal_df['signal'].ffill().fillna(0)
```

7. Visualize the strategy using a plot :

```
1 fig = plt.figure()  
2 plt1 = fig.add_subplot(111, ylabel='Price in $')  
3  
4 # Plot price and MAs  
5 msft_data['Close'].plot(ax=plt1, color='r', lw=2, label='MSFT')  
6 signal_df['short_mav'].plot(ax=plt1, color='blue', lw=2, label='short_mav')  
7 signal_df['long_mav'].plot(ax=plt1, color='orange', lw=2, label='long_mav')  
8  
9 # Plot buy signals  
10 buy_signals = signal_df[signal_df['positions'] == 1.0]  
11 plt1.plot(buy_signals.index, msft_data.loc[buy_signals.index, 'Close'],  
12           '^', markersize=10, color='m', label='Buy')  
13  
14 # Plot sell signals  
15 sell_signals = signal_df[signal_df['positions'] == -1.0]  
16 plt1.plot(sell_signals.index, msft_data.loc[sell_signals.index, 'Close'],  
17           'v', markersize=10, color='k', label='Sell')  
18
```

```

19 plt.legend()
20 plt.show()

```

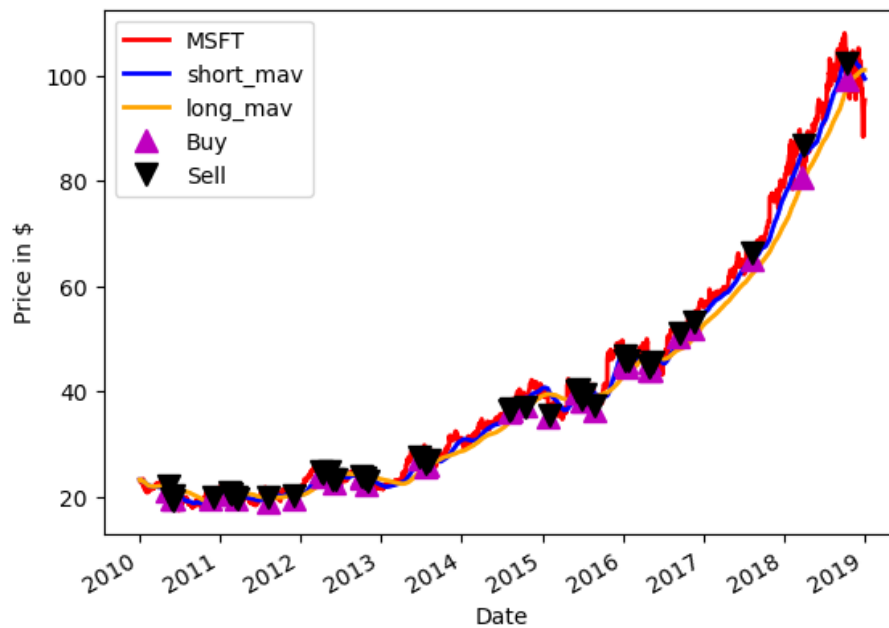


Figure 19: Combined RSI + SMAC strategy with buy (purple) and sell (black) signals plotted over price trends.

10 BACKTESTING THE TRADING STRATEGY

GOAL : Simulate how the strategy would have performed historically using `signal_df` and `msft_data`

1. calculate daily returns

```

1 msft_data['returns'] = msft_data['Close'].pct_change()

```

2. calculate strategy returns

```

1 signal_df['strategy_returns'] = msft_data['returns'] * signal_df['signal']

```

3. calculate cumulative returns

```

1 signal_df['cumulative_market_returns'] = (1 + msft_data['returns'])
  .cumprod()
2 signal_df['cumulative_strategy_returns'] = (1 + signal_df['
  strategy_returns']).cumprod()

```

4. plot the performance

```

1 plt.figure(figsize=(12,6))
2 plt.plot(signal_df['cumulative_market_returns'], label='Market
  Returns', linestyle='--')
3 plt.plot(signal_df['cumulative_strategy_returns'], label='Strategy
  Returns', color='green')
4 plt.title("Cumulative Returns: Market vs Strategy")
5 plt.xlabel("Date")
6 plt.ylabel("Cumulative Return")
7 plt.legend()
8 plt.grid(True)
9 plt.show()

```



Figure 20: Comparison of cumulative returns of market vs strategy from back-testing the hybrid strategy.

10.1 Conclusion

From the graph, we conclude that the current strategy failed to capture market gains. Some likely issues for this could be :

1. Too conservative RSI filter : Waiting for RSI ≥ 30 or ≤ 70 happens rarely. This means the strategy trades very infrequently — mostly missing big trends.
2. Late SMA Signals: SMAs lag behind price — so entries and exits may be delayed.
3. Not being in market too often

11 IMPROVING RETURNS OF THE TRADING STRATEGY

11.1 Tuning the parameters

Change the parameters as follows :

Parameter	Old value	New value
short_lb	50	20
long_lb	120	50
RSI buy limit	30	45
RSI sell limit	70	55

Backtesting the strategy and analyzing the plot generated after tuning the parameters, we conclude that the new set of parameters provides a better return than the old set of parameters.



Figure 21: Comparison of strategy performance after tuning moving average and RSI parameters.

11.2 Add a Signal Hold (Momentum Confirmation)

In trading, noisy or temporary signals are quite common i.e. as an example, suppose the RSI drops below the value for one day but the strategy reacts too quickly. To avoid this, we only add a buy or sell signal if the condition remains true for a few days, in this case 3 days. Therefore, we change the previous code block :

```
1 signal_df['signal'] = signal_df['signal'].ffill().fillna(0)
```

to the following code block :

```
1 signal_df['signal'] = signal_df['signal'].rolling(window=3).max()  
2 signal_df['signal'] = signal_df['signal'].ffill().fillna(0)
```

On backtesting, we observe that this strategy performs better than before with increased returns.



Figure 22: Improved results after applying signal hold (momentum confirmation) over a 3-day window.

11.3 Change RSI Period (Momentum Tuning)

The RSI (Relative Strength Index) is calculated using average gains and losses over a fixed number of days — by default, it's 14 days. The window or period controls how sensitive the RSI is.

RSI Period	Pros	Cons
Shorter	Catches momentum early	More noise and false signals
Longer	More reliable signals	May miss quick opportunities

Upon backtesting this strategy, we observe better returns.



Figure 23: Strategy results after modifying RSI period – showing improved responsiveness and timing of trades.

Upon retrospect, we can conclude that A shorter RSI (e.g. 8 or 10):

1. Reacts faster to recent price changes
2. Hits the oversold/overbought thresholds sooner than a longer RSI
3. Triggers BUY and SELL signals earlier

This means:

1. enter trades earlier when prices are still low (on a dip)
2. exit earlier when prices are still high (before reversal)

Thus timing advantage = better capture of price swings

12 Conclusion and Future Work

While the current hybrid strategy (SMAC + RSI) provides a structured approach to trading, there are several ways to improve:

1. Add Stop-Loss/Take-Profit rules to limit downside and lock profits.
2. Include position sizing and portfolio diversification to manage risk better.

3. Test on multiple stocks or sectors to assess robustness.
4. Apply machine learning techniques to predict market regime shifts.
5. Use technical indicators like MACD, Bollinger Bands, or ADX to enhance signal quality.

These extensions can further optimize returns and reduce drawdowns in live trading scenarios.

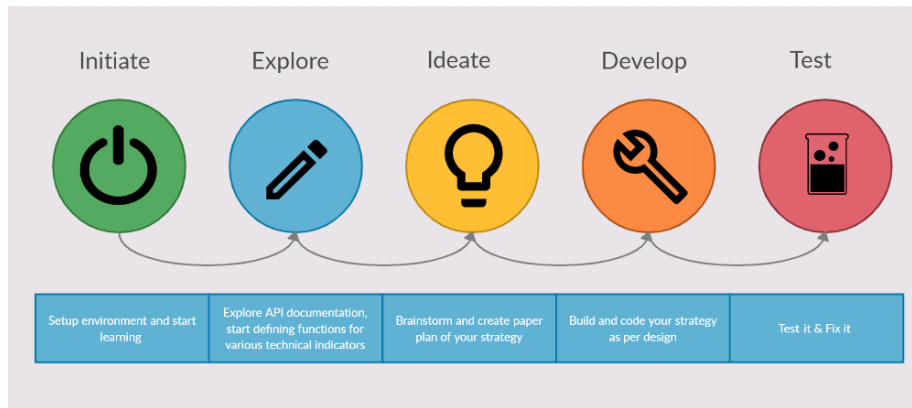


Figure 24: Algorithmic Trading Roadmap