Final Report

Algorithmic trading using python

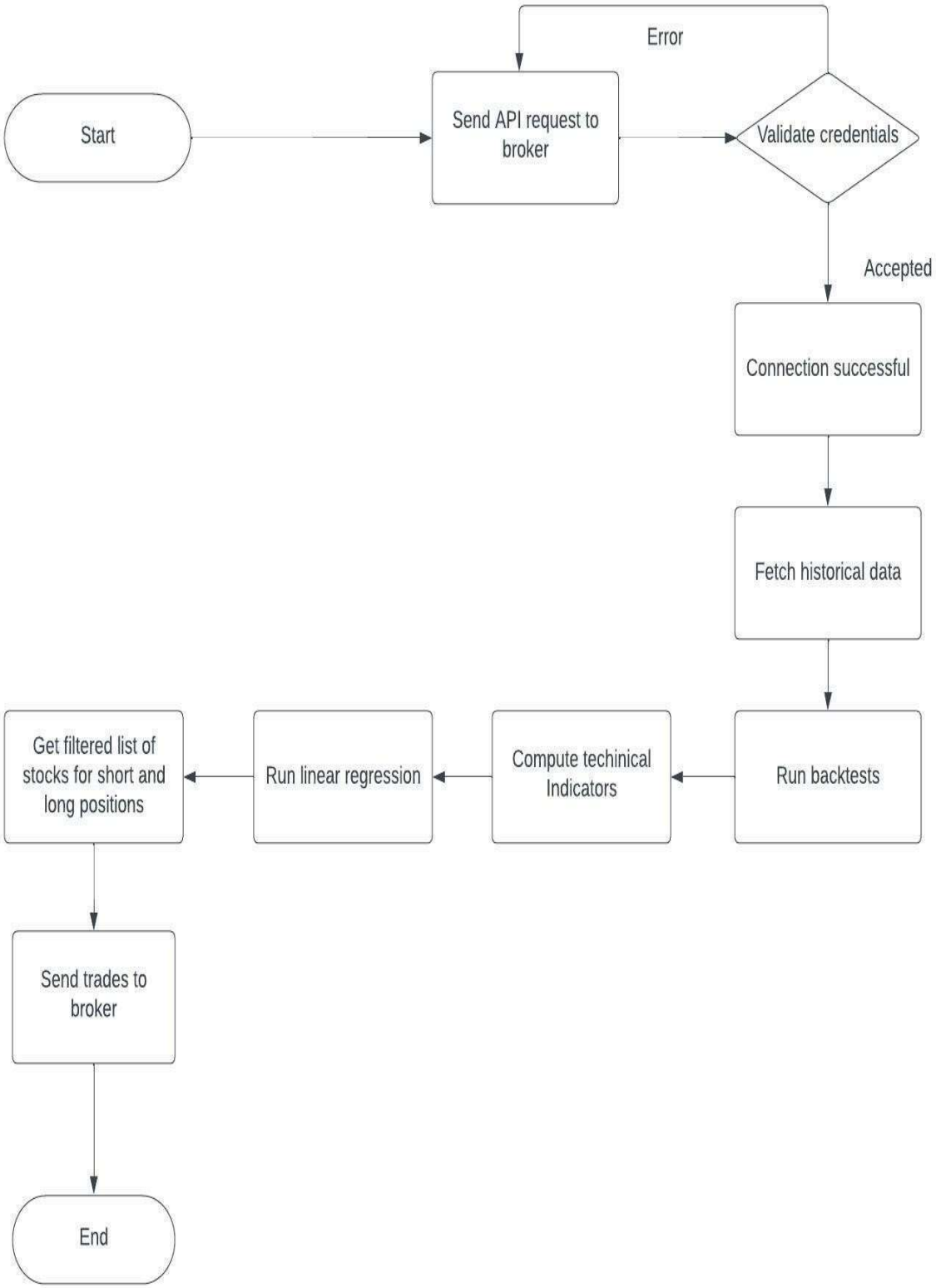
Team: Prasham Parekh and Venkatesh Jain

Contents

Introduction to Algorithmic trading

Algorithmic trading is a method of executing orders using automated pre-programmed trading instructions accounting for variables such as time, price, and volume. This type of trading attempts to leverage the speed and computational resources of computers relative to human traders. In the twenty-first century, algorithmic trading has been gaining traction with both retail and institutional traders. It is widely used by investment banks, pension funds, mutual funds, and hedge funds that may need to spread out the execution of a larger order or perform trades too fast for human traders to react to. A study in 2019 showed that around 92% of trading in the Forex market was performed by trading algorithms rather than humans. The term algorithmic trading is often used synonymously with the automated trading system. These encompass a variety of trading strategies, some of which are based on formulas and results from mathematical finance, and often rely on specialized software.

Flowchart for the system



Connecting to dummy broker API

We start our system by asking the user for credentials to log in to the broker's account for trading. The connection is established by an endpoint provided by the broker.

Our system has a dummy broker API with endpoint "'http://api.open-notify.org/astros.json"

The system checks for the response code of 200. If it gets any other response code it throws an error.

Code snippet:

```
class BrokerConnection:
    def __init__(self, username, password):
        self.username = username
        self.password = password

    # Instance method
    def createconnection(self):
        query = {'Username': self.username, 'Password': self.password}
        response = requests.get('http://api.open-notify.org/astros.json', query)
        if(response.status_code == 200):
            return f"Broker connection successful for username: {self.username}"
        else:
            return f"Some error occured, try again."
```

Fetching historical candlestick data

Yahoo! Finance is a media property that is part of the Yahoo! network. It provides financial news, data and commentary including stock quotes, press releases and financial reports.

We use the free API layer to fetch OHLC(open, high, low, close) data for technical analysis and stock selection.

Code snippet:

```
import yfinance as yf

def getOHLC(stock, period, interval, suffix = ""):
    stock = stock+suffix
    tickr = yf.Ticker(stock)
    hist = tickr.history(period, interval)
    hist=hist.drop(['Dividends', 'Stock Splits'], axis=1)
    return hist
```

Backtesting for estimated returns

Backtesting involves applying a strategy or predictive model to historical data to determine its accuracy. It allows traders to test trading strategies without the need to risk capital. Common backtesting measures include net profit/loss, return, risk-adjusted return, market exposure, and volatility.

Our system only focuses on expected monthly returns. As the analysis is on the entire market to filter out stocks and not to test the strategy.

We work with 15min ohlc data and backtest it for 1 month of historical data.

Code snippet:

```
def backtest(stockname):
    suff = ''

    stock = stockname

    interval = '15m'
    period = '1mo'

    price = getOHLC(stock, period, interval, suffix = suff) price =
    price.drop(columns = ['High', 'Low', 'Volume'], axis = 1)
    pd.set_option('display.max_rows', 1000)
    pd.set_option('display.max_columns', 500)
    pd.set_option('display.width', 1000)

    balance = 10000
    price['Signal'] = 'No Position'


    target = 1.5
    stoploss = -1

    price = getBollingerBands(price)
    price['50SMA'] = price['Close'].rolling(window = 50).mean()
    price['Sys_B'] = balance
    price['PeriodPnL'] = ((price['Open']/price['Open'].shift(1))-1)*100
    price['TradePnL'] = 0

    for i in range(1,len(price)):
        MyPnL = 0
        condition1 = price['Upper Band'][i] < price['50SMA'][i] and price['MA'][i] < price['Close'][i]


        if(condition1):

            while(price['TradePnL'][i] < target and price['TradePnL'][i] > stoploss and i in range(i,len(price)-1)):

                price.iloc[i, price.columns.get_loc('Signal')] = "BUY" price.iloc[i+1, price.columns.get_loc('TradePnL')] = ((1 +
                (price['TradePnL'][i]/100)) * (1 + (price['PeriodPnL'][i+1]/100)) - 1) * 100 if(price['TradePnL'][i+1]>=target or
                price['TradePnL'][i+1]<=stoploss): price.iloc[i+1:len(price)-1 , price.columns.get_loc('Sys_B')] = price['Sys_B'][i] +
                price['Sys_B'][i+1]*(price['TradePnL'][i+1]/100) i += 1

        else:
            pass

    winners = price[price['TradePnL']>=target]
    losers =
    price[price['TradePnL']<=stoploss]
    duration = price[price['TradePnL']!= 0]
    avg_gain = price['Sys_B'][-2] return
    str((((avg_gain)/10000)-1)*100)
```

Technical Indicators

Technical indicators are heuristic or pattern-based signals produced by the price, volume, and/or open interest of a security or contract used by traders who follow technical analysis. By

analyzing historical data, technical analysts use indicators to predict future price movements.

We are using the following technical indicators:

1. Moving average: A moving average is a statistic that captures the average change in a data series over time. In finance, moving averages are often used by technical analysts to keep track of prices trends for specific securities.
2. Bollinger bands: Bollinger Bands are envelopes plotted at a standard deviation level above and below a simple moving average of the price. Because the distance of the bands is based on standard deviation, they adjust to volatility swings in the underlying price.
3. Relative Strength Index: The relative strength index is a technical indicator used in the analysis of financial markets. It is intended to chart the current and historical strength or weakness of a stock or market based on the closing prices of a recent trading period.
4. MACD: Moving average convergence divergence (MACD) is a trend-following momentum indicator that shows the relationship between two moving averages of a security's price. The MACD is calculated by subtracting the 26-period exponential moving average (EMA) from the 12-period EMA.

Computing Techinical Indicators

Code snippet:

```
def getMACD(price, slow = 26, fast = 12, signal = 9):
    price['macd_line'] = price['Close'].ewm(span = fast).mean() - price['Close'].ewm(span =
slow).mean()
    price['signal_line'] = price['macd_line'].ewm(span = signal).mean()
    price['macd_diff'] = price['macd_line'] - price['signal_line']
    return price

def getRSI(price, period = 14):
    price['upmove'] = 0
    price['downmove'] = 0
    for i in range(len(price['Close'])):
        # if(price['Close'][i]>price['Close'][i-1]):
        price['upmove'] = np.where(price['Close'] > price['Close'].shift(1), price['Close'] -
price['Close'].shift(1), 0)
        # else:
        price['downmove'] = np.where(price['Close'] < price['Close'].shift(1),
price['Close'].shift(1) - price['Close'], 0)

    price['Avg_upmove'] = price['upmove'].rolling(window = period).mean()
    price['Avg_downmove'] = price['downmove'].rolling(window = period).mean()
    price['RS'] = price['Avg_upmove']/price['Avg_downmove']
    price['RSI'] = 100 - (100/(1+price['RS']))
    price = price.drop(columns = ['Avg_upmove', 'Avg_downmove', 'RS'], axis=1)
    return price

def getBollingerBands(price):
    price['MA']=price['Close'].rolling(window=20).mean()
    price['StdDev']=price['Close'].rolling(window=20).std()
    price['Upper Band'] = price['MA'] + (price['StdDev'] * 2)
    price['Lower Band'] = price['MA'] - (price['StdDev'] * 2)
    return price
```

Linear Regression

In statistics, linear regression is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables. The case of one explanatory variable is called simple linear regression; for more than one, the process is called multiple linear regression

We use linear regression to select stocks with higher volatility. Such stocks are expected to perform better in a mean reverting market.

Code snippet:

```
def OLS_max(ohlc):
    ohlc1 = pd.DataFrame(ohlc[-24:])
    ohlc1['Step'] = [i for i in range(len(ohlc1['Close']))]
    result = sm.ols(formula="Close ~ Step", data=ohlc1).fit()
    ohlc1['Predicted'] = result.params['Intercept'] + result.params['Step']*ohlc1['Step']
    ohlc1['Residual%'] = abs(((ohlc1['Predicted'] - ohlc1['Close'])/ohlc1['Close'])*100)
    return max(ohlc1['Residual%'])

def regression_channel(ohlc):
    ohlc1 = pd.DataFrame(ohlc[-24:])
    ohlc1['Step'] = [i for i in range(len(ohlc1['Close']))]
    result = sm.ols(formula="Close ~ Step", data=ohlc1).fit()
    ohlc1['Predicted'] = result.params['Intercept'] + result.params['Step']*ohlc1['Step']
    ohlc1['UC'] = ohlc1['Predicted'] + 1 * ohlc1['Close'].std()
    ohlc1['LC'] = ohlc1['Predicted'] - 1 * ohlc1['Close'].std()
    return ohlc1
```

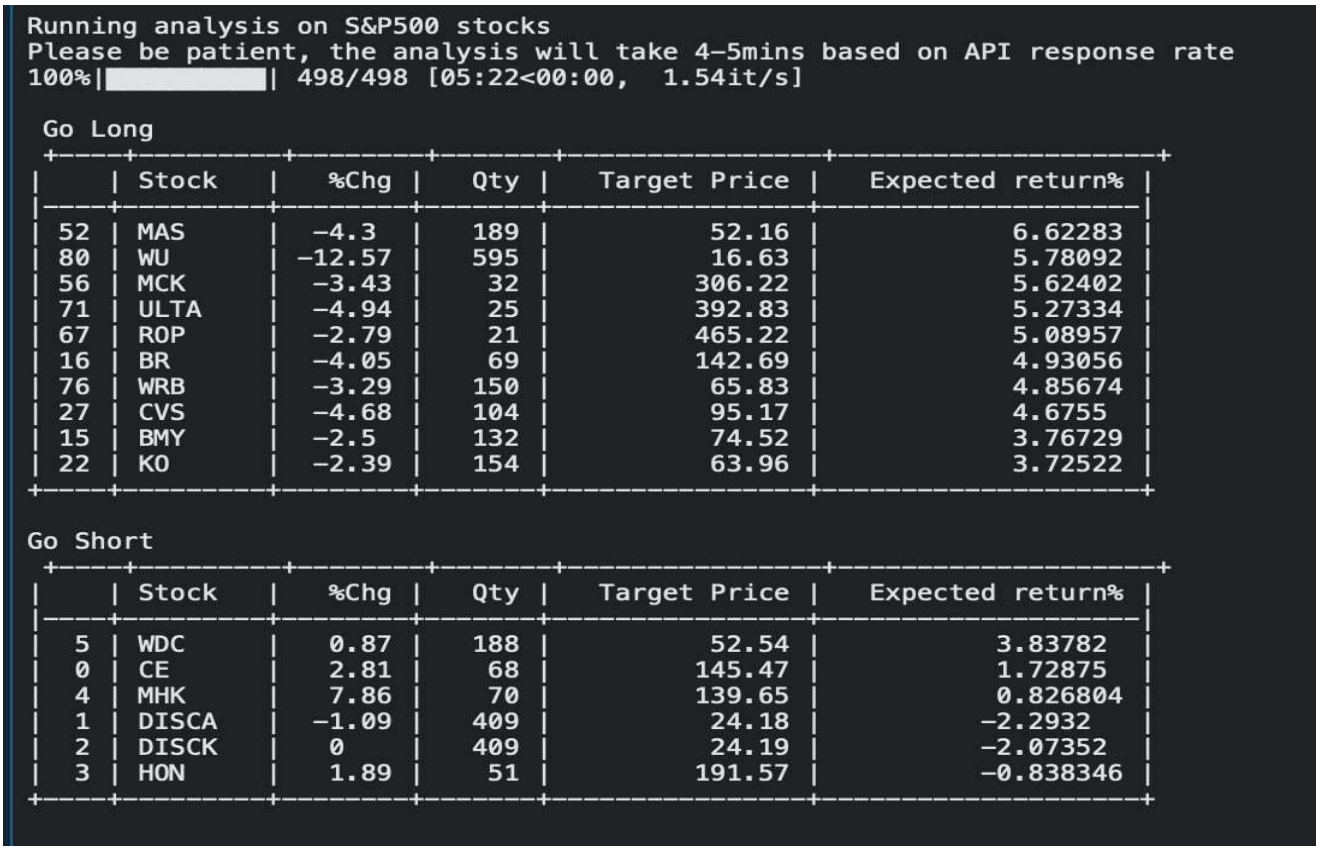
List of stocks

The Nasdaq-100 is a stock market index made up of 102 equity securities issued by 100 of the largest non-financial companies listed on the Nasdaq stock market. It is a modified capitalization-weighted index. The stocks' weights in the index are based on their market capitalizations, with certain rules capping the influence of the largest components. It is based on exchange, and it does not have any financial companies.

The complete list is available at: https://www.slickcharts.com/sp500

Sample results

Output:



```
Running analysis on S&P500 stocks
Please be patient, the analysis will take 4-5mins based on API response rate
100%|██████████| 498/498 [05:22<00:00,  1.54it/s]

 Go Long
 +----+---------+--------+-------+----------------+--------------------+
|    | Stock   |   %Chg |   Qty |   Target Price |   Expected return% |
|----+---------+--------+-------+----------------+--------------------|
| 52 | MAS     |   -4.3 |   189 |          52.16 |            6.62283 |
| 80 | WU      | -12.57 |   595 |          16.63 |            5.78092 |
| 56 | MCK     |  -3.43 |    32 |         306.22 |            5.62402 |
| 71 | ULTA    |  -4.94 |    25 |         392.83 |            5.27334 |
| 67 | ROP     |  -2.79 |    21 |         465.22 |            5.08957 |
| 16 | BR      |  -4.05 |    69 |         142.69 |            4.93056 |
| 76 | WRB     |  -3.29 |   150 |          65.83 |            4.85674 |
| 27 | CVS     |  -4.68 |   104 |          95.17 |            4.6755  |
| 15 | BMY     |  -2.5  |   132 |          74.52 |            3.76729 |
| 22 | KO      |  -2.39 |   154 |          63.96 |            3.72522 |
+----+---------+--------+-------+----------------+--------------------+

Go Short
 +----+---------+--------+-------+----------------+--------------------+
|    | Stock   |   %Chg |   Qty |   Target Price |   Expected return% |
|----+---------+--------+-------+----------------+--------------------|
|  5 | WDC     |   0.87 |   188 |          52.54 |            3.83782 |
|  0 | CE      |   2.81 |    68 |         145.47 |            1.72875 |
|  4 | MHK     |   7.86 |    70 |         139.65 |            0.826804 |
|  1 | DISCA   |  -1.09 |   409 |          24.18 |           -2.2932  |
|  2 | DISCK   |   0    |   409 |          24.19 |           -2.07352 |
|  3 | HON     |   1.89 |    51 |         191.57 |           -0.838346 |
+----+---------+--------+-------+----------------+--------------------+
```

Miscellaneous information and data sources

- The strategy is a mean reverting strategy that capitalizes on the market moving sideways.
- The system replicated an actual trading system used at sophisticated trading firms and hedge funds ● Output:
    - Table ‘Go Long’ gives tickr symbol of stocks to take a long position in.
    - Table ‘Go Short gives tickr symbol of stocks to take a short position in.

    - Column:
        - ‘Stock’: The tickr name of the companies.
        - ‘%Chg’: Change in price from previous day.
        - ‘Qty’: Amount to buy/sell assuming $10000/stock.
        - ‘Target Price’: Price to close the trade.
        - ‘Expected return’: 1 month expected return % based on backtested data.
- Data Sources:
    - YahooFinance API
    - Wikipedia
    - Slickcharts

Challenges

- The stock market is very volatile. The proposed system gives expected returns which may not be true.
- The entire system is heavily dependent on a single data source. If that data source fails, the entire system will stop working
- The system relies on API for fetching data and not any locally sourced data, this makes the processing tedious and calculation-intensive

Future Work

- Make the system more robust and secure.
- Make the system fetch data from multiple sources so the dependencies is reduced.
- Add feature such as different time periods and intervals for investment.