PYTHON PROJECT PROPOSAL

Team

Rui Zong

Topic:

Moving Average Trading Strategy

TABLE OF CONTENTS

THE MOVING AVERAGE TRADING STRATEGY

INTRODUCTION

We aim to design a trading strategy that will make it easier for traders and investors to formulate their trading style. This will be done by taking into account and using the technical analysis indicator known as Moving Average.

The use of technical analysis evaluates investments and identifies the optimum trade opportunities by observing price trends and patterns on charts. Technical analysts trust past historical figures and movements in prices of securities to forecast future prices.

This strategy will be interesting as a common problem encountered during trading is whether the trader should buy or sell a stock. By considering Moving Average and understanding the dynamics of this model, we aim to mitigate this issue.

MOVING AVERAGE

Moving average involves creating a constantly updated average price that is based on historical data to signal optimal buy and sell choices. The importance of the model is that these buy and sell signals assist investors in making important decisions.

By analyzing a short-term and long-term moving average for historical data over a 5-day and 10-day period, profitable investment decisions can be assessed, and this helps determine the trend direction.

At the point when the price crosses above the moving average, this will indicate a buy strategy and an upward trend. On the other hand, when it crosses below the moving average, a sell strategy or downward trend will prove to generate a higher return.

The calculation involves calculating the average within a range of prices multiplied by the number of periods within that particular range.

This arithmetic indicator observes a death and golden cross. The death cross occurs at the point when the short-term moving average crosses below the long-term moving average thus indicating a bearish signal. On the other hand, a golden cross occurs at the point when the short-term moving average crosses above the long-term moving average which indicates a bull signal.

A bear signal is a sign of decline or losses whereas a bull signal indicates rising prices that are more optimistic.

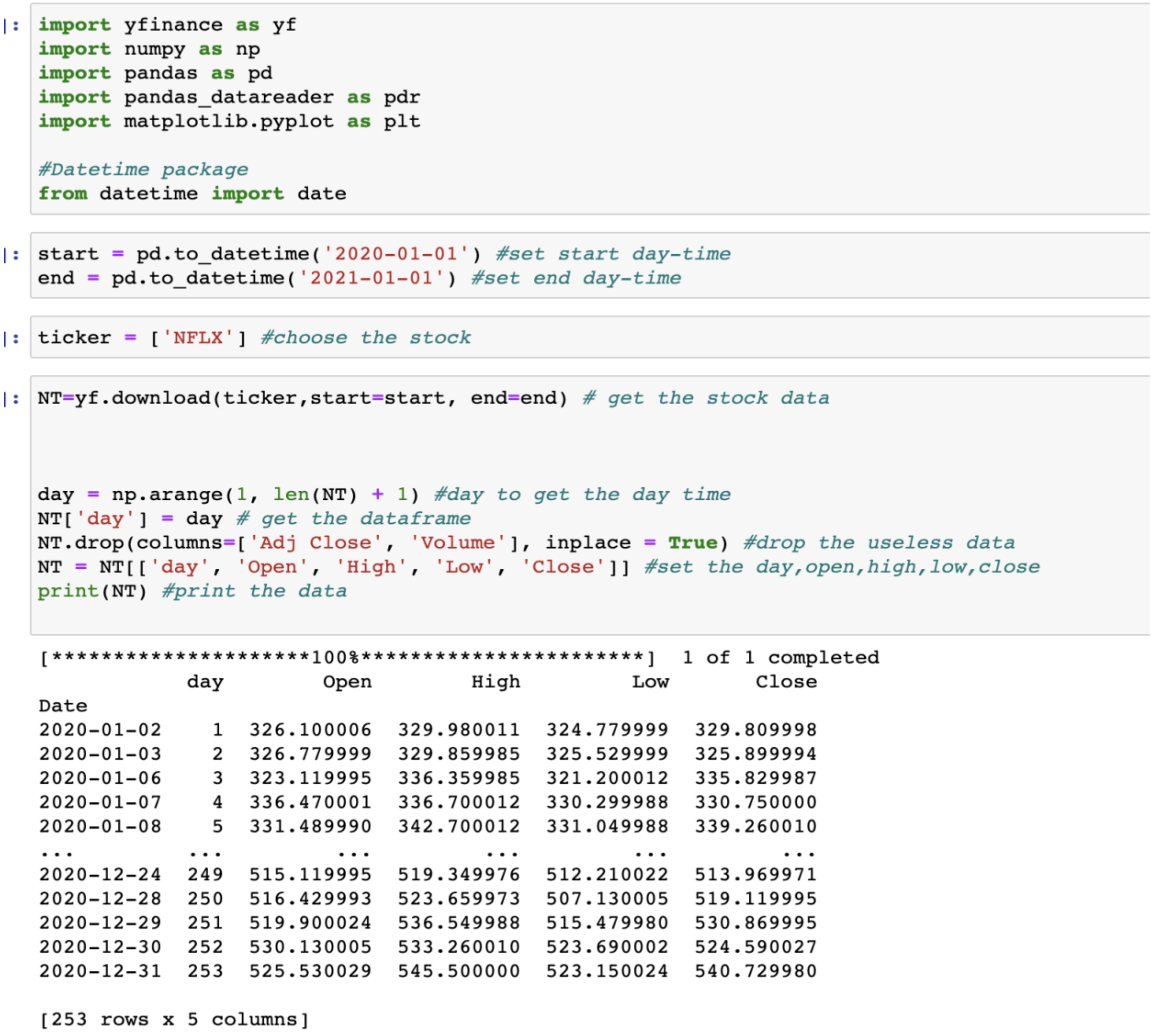
METHODOLOGY

The steps we carried out to code our trading strategy are highlighted below:

i. **Import pandas, pandas datareader, numpy, yfinance, pyplot and matplotib**
ii. **Data time set for one year from 1 January 2020 to 1 January 2021**
iii. **Data frame created and data generated from yahoo finance based on date to gather Open, High, Low and Closing price for stock**
iv. **Two variables created – 5-day and 10-day moving averages**
v. **Rolling widow of 10 for long-term moving average and 5 for short-term moving average**
vi. **Flags then added to determine when two variables cross over**
vii. **Data plotted using matplotlip.pyplot**

The stock selected was Netflix (NFLX) and the above steps were carried out.

CODE



```
]: import yfinance as yf
   import numpy as np
   import pandas as pd
   import pandas_datareader as pdr
   import matplotlib.pyplot as plt

   #Datetime package
   from datetime import date
```

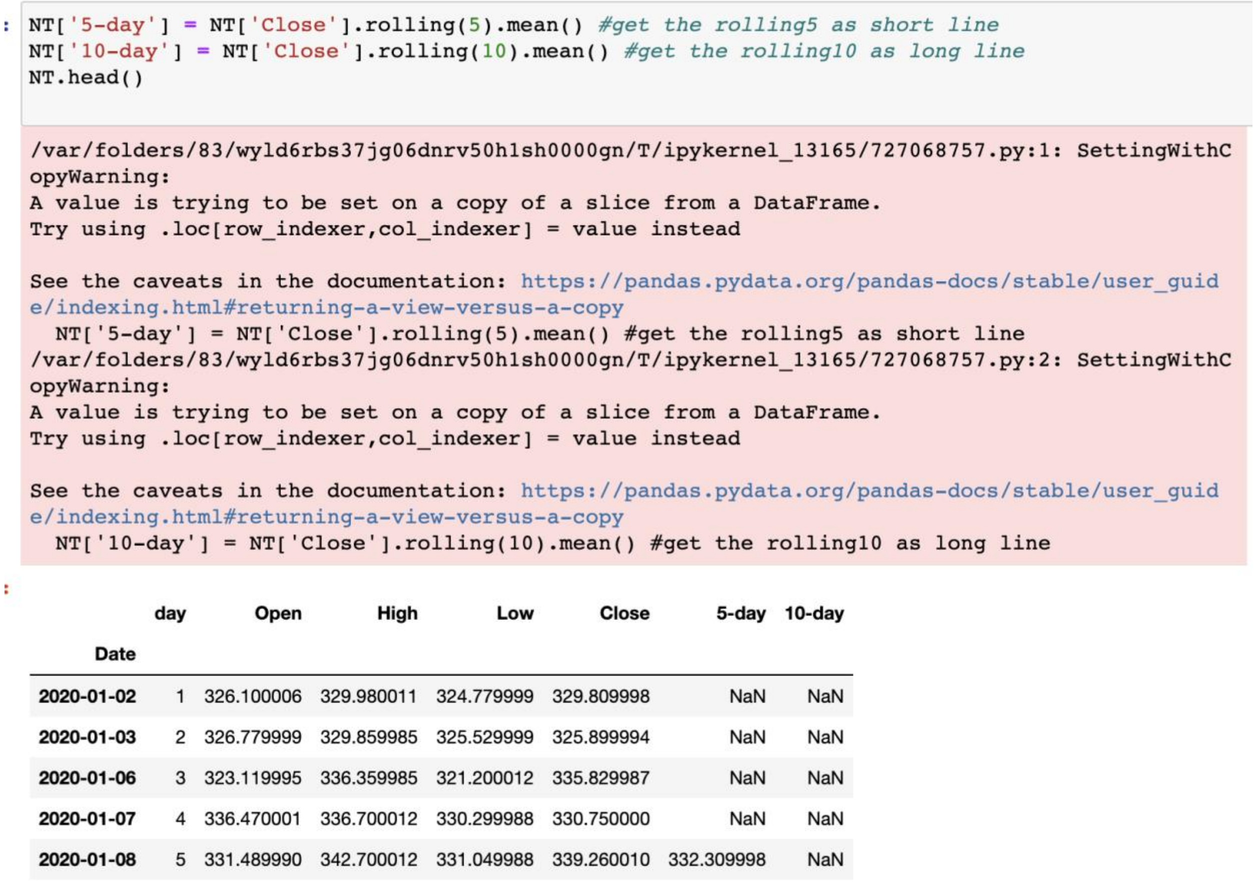
```
]: start = pd.to_datetime('2020-01-01') #set start day-time
   end = pd.to_datetime('2021-01-01') #set end day-time
```

```
]: ticker = ['NFLX'] #choose the stock
```

```
]: NT=yf.download(ticker,start=start, end=end) # get the stock data



   day = np.arange(1, len(NT) + 1) #day to get the day time
   NT['day'] = day # get the dataframe
   NT.drop(columns=['Adj Close', 'Volume'], inplace = True) #drop the useless data
   NT = NT[['day', 'Open', 'High', 'Low', 'Close']] #set the day,open,high,low,close
   print(NT) #print the data
```

```
[*********************100%***********************]  1 of 1 completed
            day        Open        High         Low       Close
Date
2020-01-02    1  326.100006  329.980011  324.779999  329.809998
2020-01-03    2  326.779999  329.859985  325.529999  325.899994
2020-01-06    3  323.119995  336.359985  321.200012  335.829987
2020-01-07    4  336.470001  336.700012  330.299988  330.750000
2020-01-08    5  331.489990  342.700012  331.049988  339.260010
...         ...         ...         ...         ...         ...
2020-12-24  249  515.119995  519.349976  512.210022  513.969971
2020-12-28  250  516.429993  523.659973  507.130005  519.119995
2020-12-29  251  519.900024  536.549988  515.479980  530.869995
2020-12-30  252  530.130005  533.260010  523.690002  524.590027
2020-12-31  253  525.530029  545.500000  523.150024  540.729980

[253 rows x 5 columns]
```

In the above code, we downloaded the specific ticker using yahoo finance and then cleaned the data to obtain open, high, low, close with the start date of 01/01/2020 and end date of 01/01/2021.

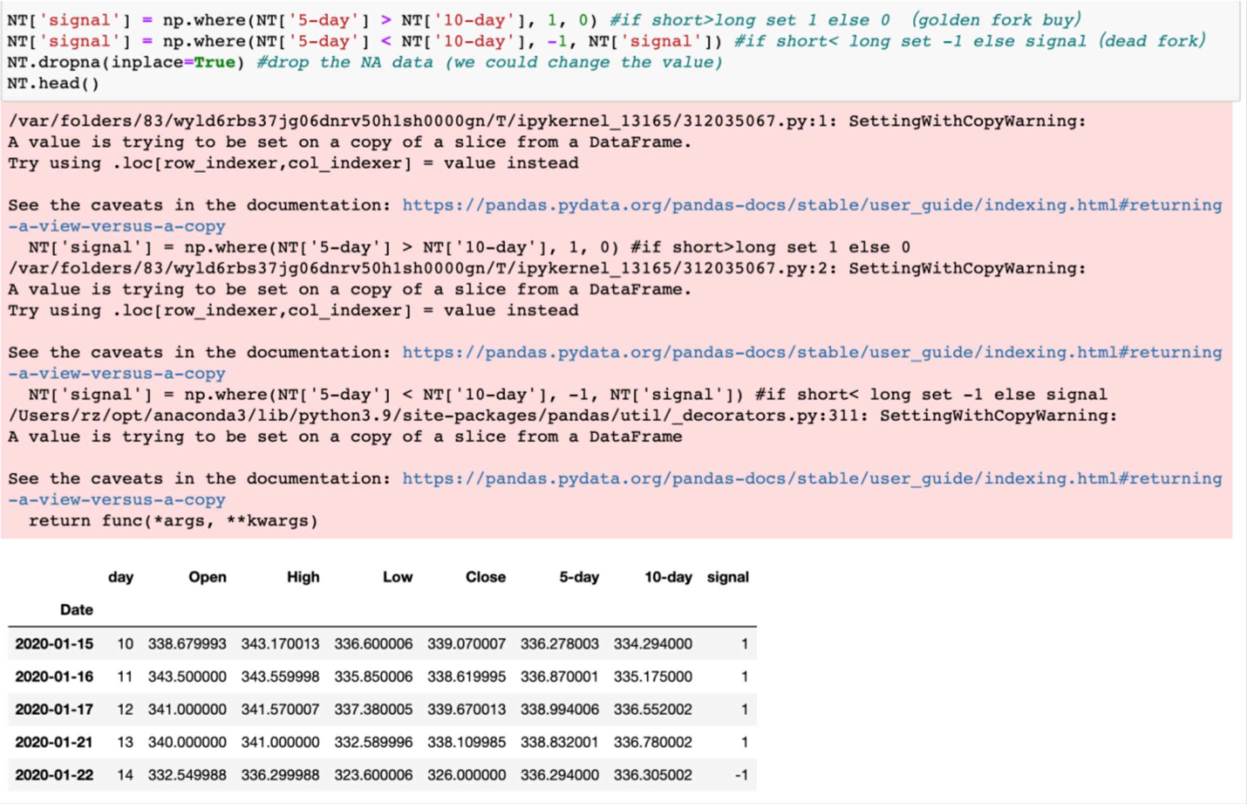


```
NT['5-day'] = NT['Close'].rolling(5).mean() #get the rolling5 as short line
NT['10-day'] = NT['Close'].rolling(10).mean() #get the rolling10 as long line
NT.head()
```

```
/var/folders/83/wyld6rbs37jg06dnrv50h1sh0000gn/T/ipykernel_13165/727068757.py:1: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy
  NT['5-day'] = NT['Close'].rolling(5).mean() #get the rolling5 as short line
/var/folders/83/wyld6rbs37jg06dnrv50h1sh0000gn/T/ipykernel_13165/727068757.py:2: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy
  NT['10-day'] = NT['Close'].rolling(10).mean() #get the rolling10 as long line
```

	day	Open	High	Low	Close	5-day	10-day
Date							
2020-01-02	1	326.100006	329.980011	324.779999	329.809998	NaN	NaN
2020-01-03	2	326.779999	329.859985	325.529999	325.899994	NaN	NaN
2020-01-06	3	323.119995	336.359985	321.200012	335.829987	NaN	NaN
2020-01-07	4	336.470001	336.700012	330.299988	330.750000	NaN	NaN
2020-01-08	5	331.489990	342.700012	331.049988	339.260010	332.309998	NaN

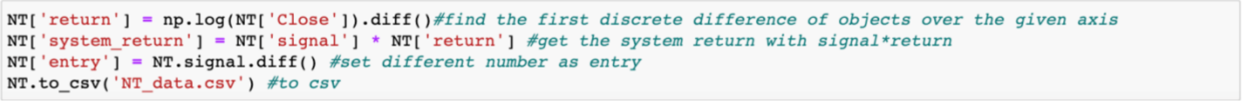
Further, as indicated above we get the line graph for 5-day and 10-day mean, where 5-day is our short line and 10-day is the long line. To analyze the moving average, we set some specifications i.e., to long the stock if short line rises above the long line (signal=1) and short the stock if short line dips below the long line (signal= -1). The code for getting this result is indicated below:



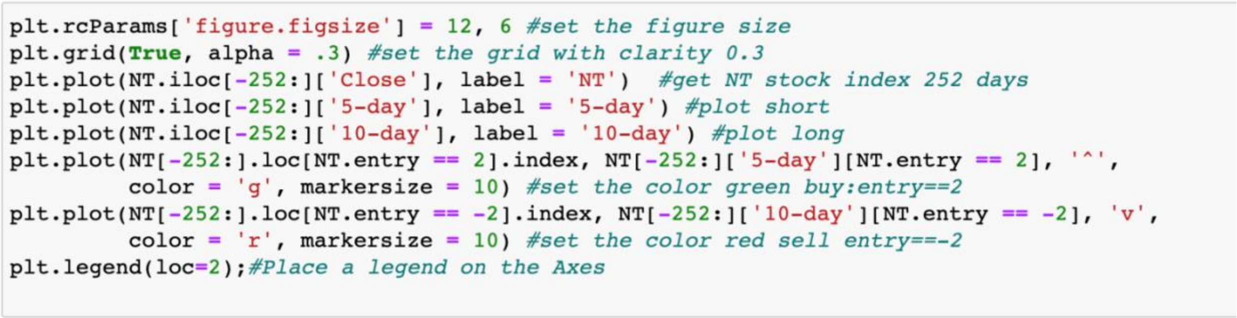
```
NT['signal'] = np.where(NT['5-day'] > NT['10-day'], 1, 0) #if short>long set 1 else 0  (golden fork buy)
NT['signal'] = np.where(NT['5-day'] < NT['10-day'], -1, NT['signal']) #if short< long set -1 else signal (dead fork)
NT.dropna(inplace=True) #drop the NA data (we could change the value)
NT.head()
```

```
/var/folders/83/wyld6rbs37jg06dnrv50h1sh0000gn/T/ipykernel_13165/312035067.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
  NT['signal'] = np.where(NT['5-day'] > NT['10-day'], 1, 0) #if short>long set 1 else 0
/var/folders/83/wyld6rbs37jg06dnrv50h1sh0000gn/T/ipykernel_13165/312035067.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
  NT['signal'] = np.where(NT['5-day'] < NT['10-day'], -1, NT['signal']) #if short< long set -1 else signal
/Users/rz/opt/anaconda3/lib/python3.9/site-packages/pandas/util/_decorators.py:311: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
  return func(*args, **kwargs)
```

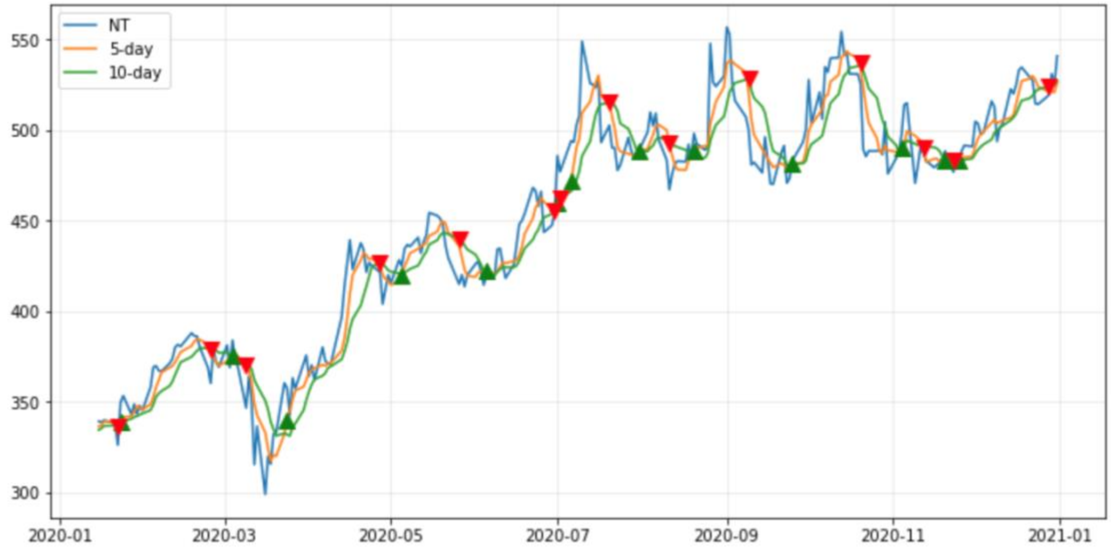
	day	Open	High	Low	Close	5-day	10-day	signal
Date								
2020-01-15	10	338.679993	343.170013	336.600006	339.070007	336.278003	334.294000	1
2020-01-16	11	343.500000	343.559998	335.850006	338.619995	336.870001	335.175000	1
2020-01-17	12	341.000000	341.570007	337.380005	339.670013	338.994006	336.552002	1
2020-01-21	13	340.000000	341.000000	332.589996	338.109985	338.832001	336.780002	1
2020-01-22	14	332.549988	336.299988	323.600006	326.000000	336.294000	336.305002	-1



```
NT['return'] = np.log(NT['Close']).diff()#find the first discrete difference of objects over the given axis
NT['system_return'] = NT['signal'] * NT['return'] #get the system return with signal*return
NT['entry'] = NT.signal.diff() #set different number as entry
NT.to_csv('NT_data.csv') #to csv
```



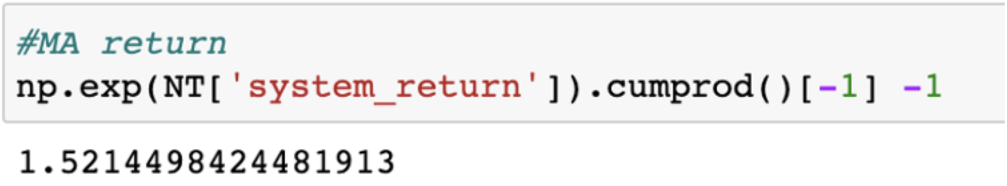
```
plt.rcParams['figure.figsize'] = 12, 6 #set the figure size
plt.grid(True, alpha = .3) #set the grid with clarity 0.3
plt.plot(NT.iloc[-252:]['Close'], label = 'NT')  #get NT stock index 252 days
plt.plot(NT.iloc[-252:]['5-day'], label = '5-day') #plot short
plt.plot(NT.iloc[-252:]['10-day'], label = '10-day') #plot long
plt.plot(NT[-252:].loc[NT.entry == 2].index, NT[-252:]['5-day'][NT.entry == 2], '^',
         color = 'g', markersize = 10) #set the color green buy:entry==2
plt.plot(NT[-252:].loc[NT.entry == -2].index, NT[-252:]['10-day'][NT.entry == -2], 'v',
         color = 'r', markersize = 10) #set the color red sell entry==-2
plt.legend(loc=2);#Place a legend on the Axes
```



In this code, we assign colors to the plotted line graphs as you can see in the screenshot above. Additionally, the red and green triangles that you see indicate a bullish or a bearish trend, green triangle indicating the upward trend and red triangle indicating the downward trend respectively. The green arrow can also be interpreted as a signal for 'buy(entry=2)' whereas the inference for the red arrow is a 'sell(entry=-2)' signal.

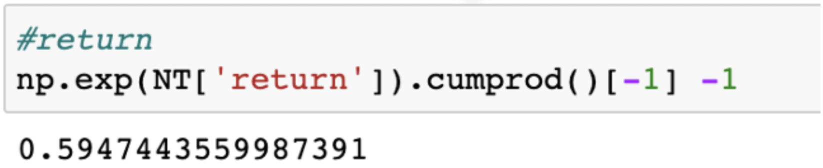
BACK-TESTING

Following is the code for Back -Testing, to verify if our strategy and code give us reliable results going forward.



```
#MA return
np.exp(NT['system_return']).cumprod()[-1] -1
```

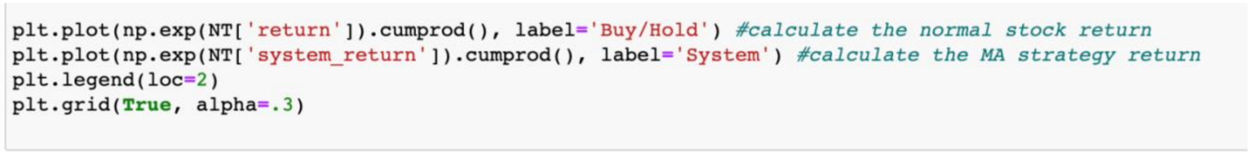
```
1.5214498424481913
```



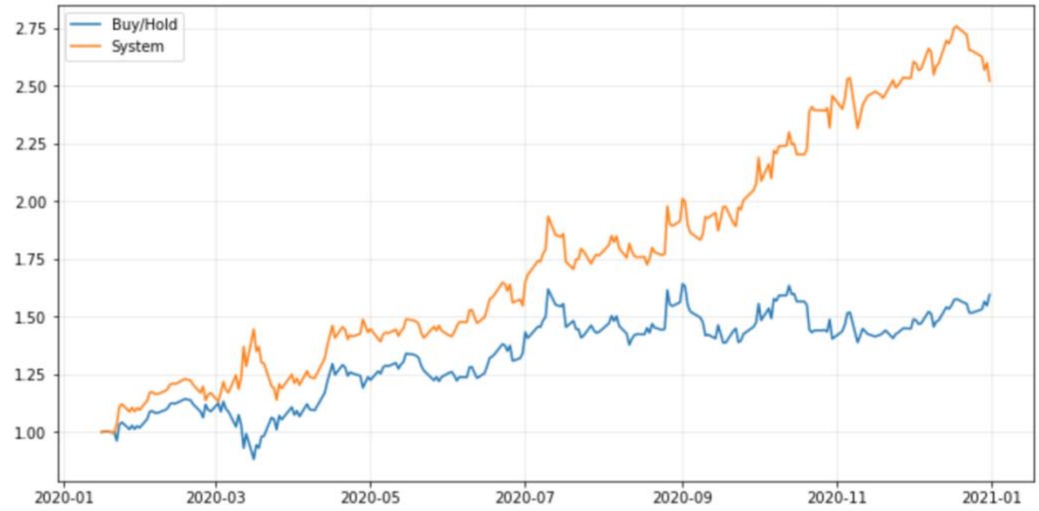
```
#return
np.exp(NT['return']).cumprod()[-1] -1
```

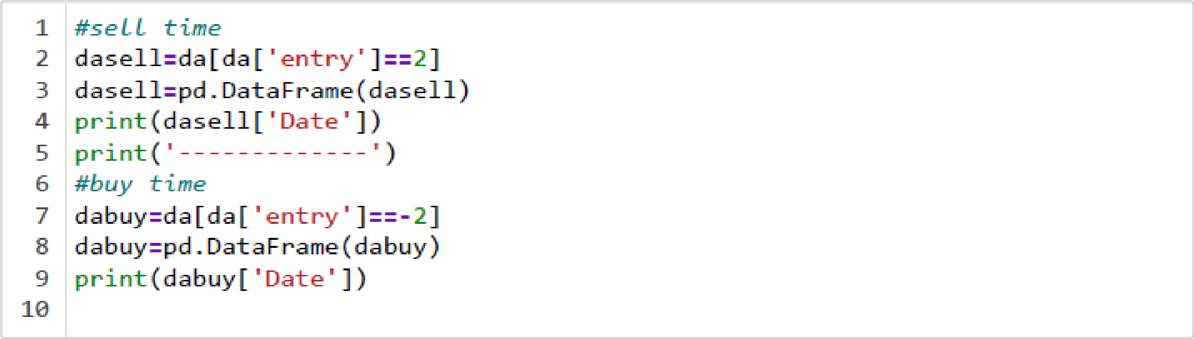
```
0.5947443559987391
```

We defined a function to calculate the MA return 152%, the normal return 59% and plotted these data to compare in different time periods. It’s clear to see that the MA strategy performed well during the pandemic (2020-03).



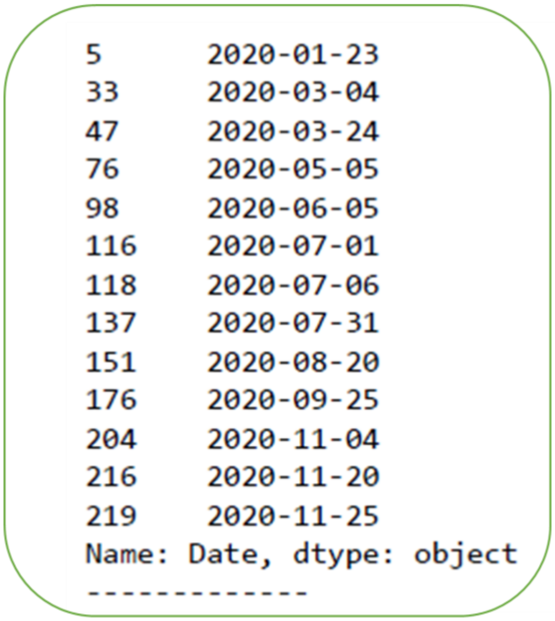
```
plt.plot(np.exp(NT['return']).cumprod(), label='Buy/Hold') #calculate the normal stock return
plt.plot(np.exp(NT['system_return']).cumprod(), label='System') #calculate the MA strategy return
plt.legend(loc=2)
plt.grid(True, alpha=.3)
```





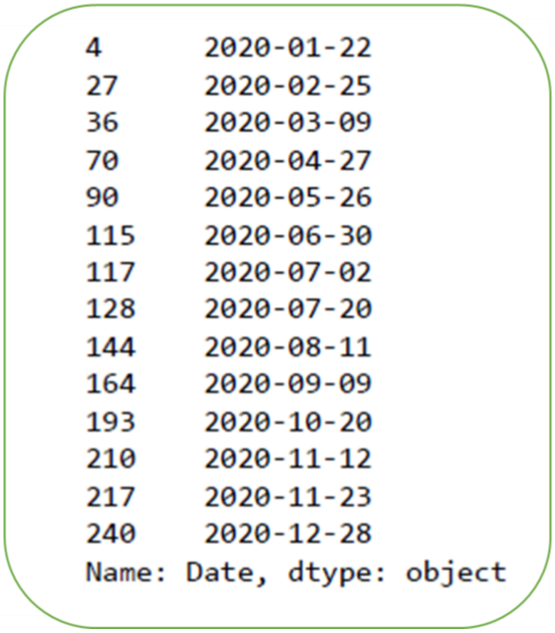
```python
#sell time
dasell=da[da['entry']==2]
dasell=pd.DataFrame(dasell)
print(dasell['Date'])
print('-------------')
#buy time
dabuy=da[da['entry']==-2]
dabuy=pd.DataFrame(dabuy)
print(dabuy['Date'])

```

SELL



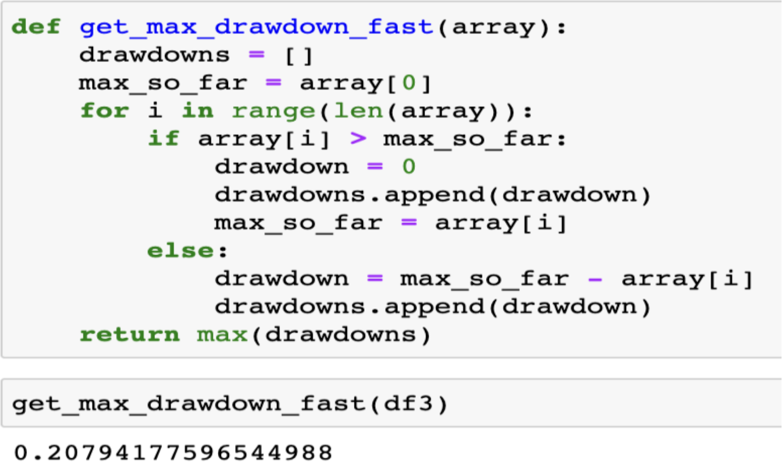
```
5       2020-01-23
33      2020-03-04
47      2020-03-24
76      2020-05-05
98      2020-06-05
116     2020-07-01
118     2020-07-06
137     2020-07-31
151     2020-08-20
176     2020-09-25
204     2020-11-04
216     2020-11-20
219     2020-11-25
Name: Date, dtype: object
-------------
```

BUY



```
4       2020-01-22
27      2020-02-25
36      2020-03-09
70      2020-04-27
90      2020-05-26
115     2020-06-30
117     2020-07-02
128     2020-07-20
144     2020-08-11
164     2020-09-09
193     2020-10-20
210     2020-11-12
217     2020-11-23
240     2020-12-28
Name: Date, dtype: object
```

As you can see above, two prices for the same dates are compared. For example, buy stock on 01/22/2020 for 4 and sell on 01/23/2020. The general idea is to buy low and sell higher, to earn profits.

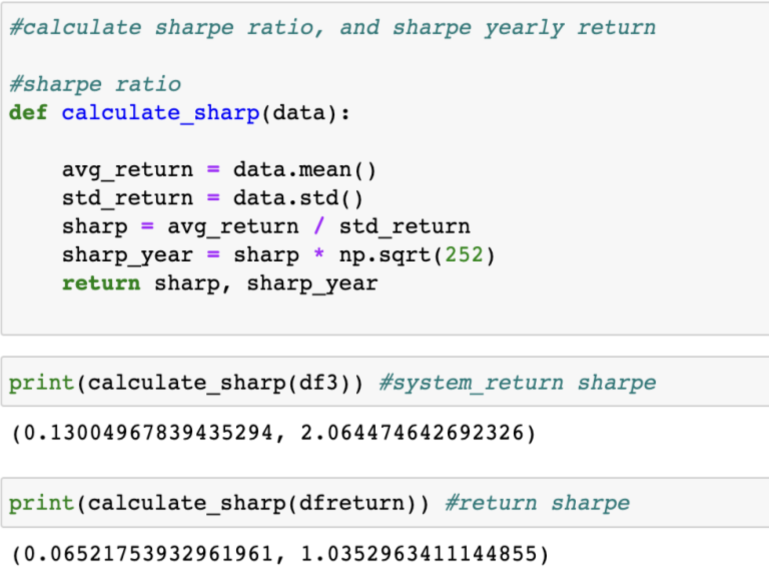


```
def get_max_drawdown_fast(array):
    drawdowns = []
    max_so_far = array[0]
    for i in range(len(array)):
        if array[i] > max_so_far:
            drawdown = 0
            drawdowns.append(drawdown)
            max_so_far = array[i]
        else:
            drawdown = max_so_far - array[i]
            drawdowns.append(drawdown)
    return max(drawdowns)
```

```
get_max_drawdown_fast(df3)
```

```
0.20794177596544988
```

Another interesting part of our project was to find out the maximum fall in the Netflix stock, which is given as a difference between the lowest trough and the highest peak before the trough in the moving average. The code to find the results is indicated above. The max drawdown is 20% which is considered a little bit high. But it’s always better than the original holding strategy.



```
#calculate sharpe ratio, and sharpe yearly return

#sharpe ratio
def calculate_sharp(data):

    avg_return = data.mean()
    std_return = data.std()
    sharp = avg_return / std_return
    sharp_year = sharp * np.sqrt(252)
    return sharp, sharp_year
```

```
print(calculate_sharp(df3)) #system_return sharpe
```

```
(0.13004967839435294, 2.064474642692326)
```

```
print(calculate_sharp(dfreturn)) #return sharpe
```

```
(0.06521753932961961, 1.0352963411144855)
```

The code to calculate Sharpe ratio is shown, as we considered Sharpe Ratio to be one of the important factors in deciding how much return we are getting for every unit of risk taken. As you can see, sharp yearly return is between 1-2, which implies that one should invest in the Netflix stock for positive return.

OTHER INDICATORS

It is important to note that using more than one indicator in technical analysis has been proven to be more profitable in the past. With this in mind, we suggest adding advanced indicators to better judge the stock market.

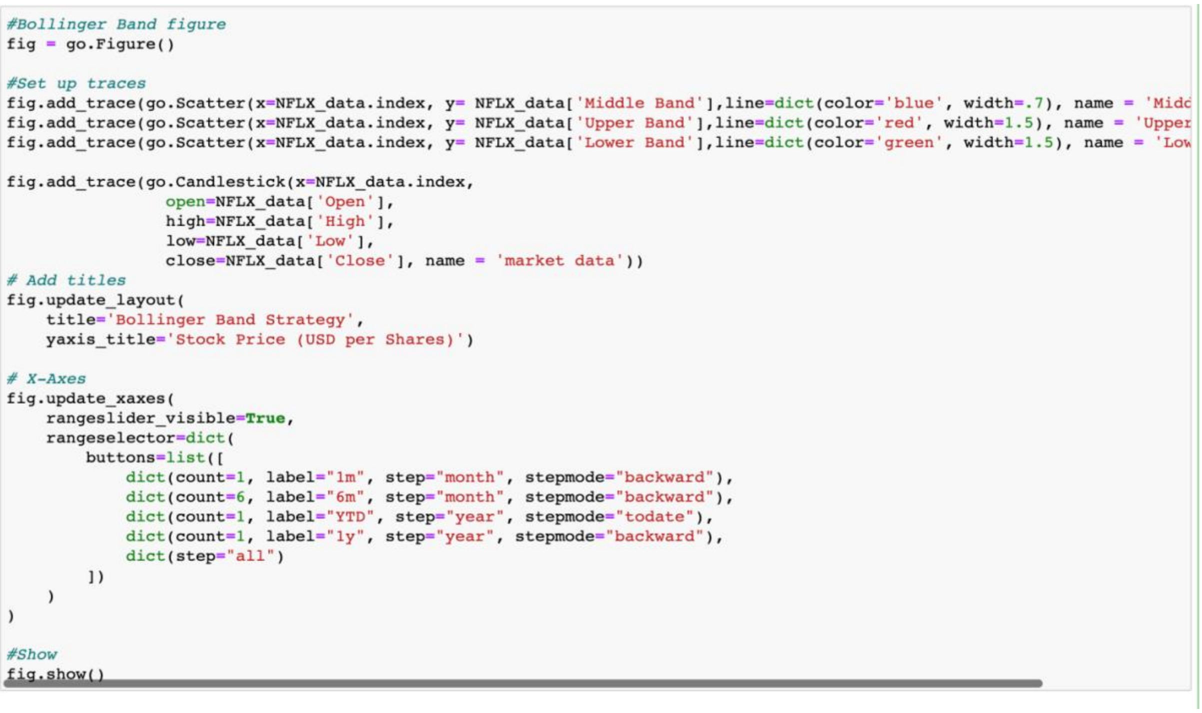
Moving average convergence divergence (MACD)

A valuable tool to use in trading strategies is MACD which is a trend-following momentum indicator observing the relationship between two moving averages.

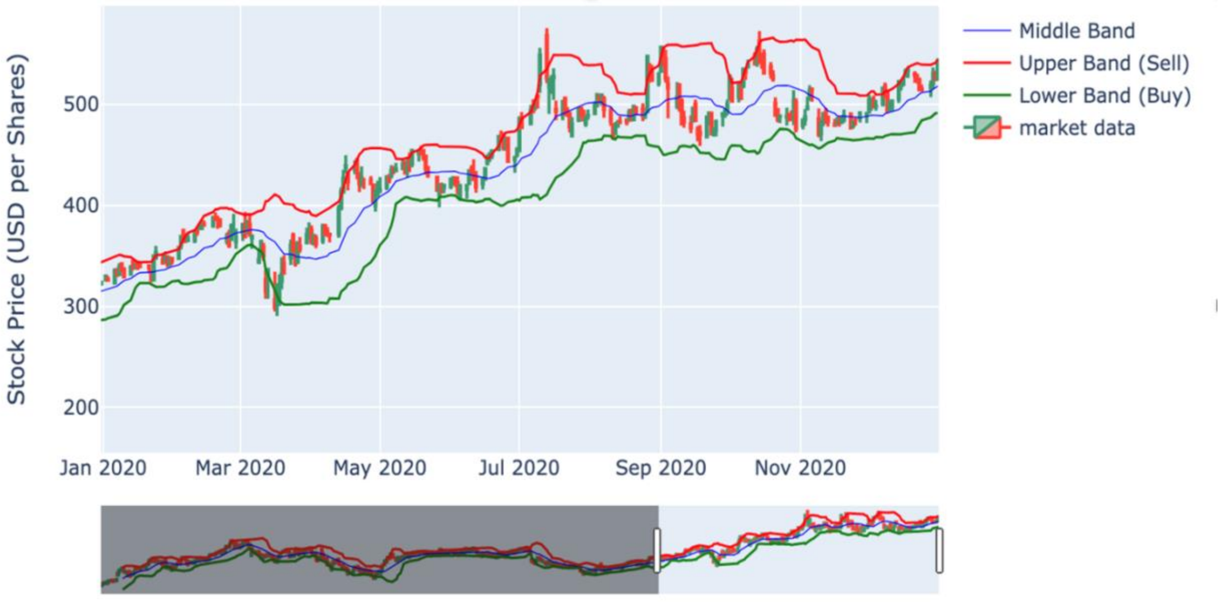
The idea of MACD involves using the two moving averages along with histograms and signals. Therefore, by observing MACD, signals and histograms for stocks and analyzing their short-term and long-term moving averages, trade decisions can be made. At the point when the MACD line crosses above the signal, this will indicate a buy strategy whereas when it crosses below the signal line, a sell strategy will prove profitable.

By means of machine learning and MACD, data can be gathered much faster with trade strategies identified using optimal values. This makes the indicator an advanced strategy that is based off moving average.

Bollinger bands



```
#Bollinger Band figure
fig = go.Figure()

#Set up traces
fig.add_trace(go.Scatter(x=NFLX_data.index, y= NFLX_data['Middle Band'],line=dict(color='blue', width=.7), name = 'Midc
fig.add_trace(go.Scatter(x=NFLX_data.index, y= NFLX_data['Upper Band'],line=dict(color='red', width=1.5), name = 'Upper
fig.add_trace(go.Scatter(x=NFLX_data.index, y= NFLX_data['Lower Band'],line=dict(color='green', width=1.5), name = 'Low

fig.add_trace(go.Candlestick(x=NFLX_data.index,
                open=NFLX_data['Open'],
                high=NFLX_data['High'],
                low=NFLX_data['Low'],
                close=NFLX_data['Close'], name = 'market data'))
# Add titles
fig.update_layout(
    title='Bollinger Band Strategy',
    yaxis_title='Stock Price (USD per Shares)')

# X-Axes
fig.update_xaxes(
    rangeslider_visible=True,
    rangeselector=dict(
        buttons=list([
            dict(count=1, label="1m", step="month", stepmode="backward"),
            dict(count=6, label="6m", step="month", stepmode="backward"),
            dict(count=1, label="YTD", step="year", stepmode="todate"),
            dict(count=1, label="1y", step="year", stepmode="backward"),
            dict(step="all")
        ])
    )
)

#Show
fig.show()
```



Like MACD, another useful strategy is to use Bollinger Bands. The strategy focuses primarily on price and volatility and is used to detect market conditions when the stock is overbought or oversold. Bollinger Bands play a vital role in determining exit and entry points for a trade. It relies on the concept of mean reversion of the price, which means prices will eventually revert back to the mean price, in case they deviate substantially when overbought or oversold. In the above code, we introduced 3 bands, namely Middle Band, Upper Band and Lower Band and added Candlestick for Open, High, Low and Close.

CONCLUSION AND IMPROVEMENTS

First, the moving average trading strategy proves to be a useful oscillator in determining buy and sell decisions, however, it is important to use more indicators to further support the trading choice.
Second, many traders are under the assumption that markets are efficient. This means that current market prices already reflect all the information that is available, that is, there is symmetric information in the market. Therefore, historical data should not be used to indicate future price movements.
Third, one useful strategy that can be implemented is using a combination of indicators including Moving Average, MACD and Bollinger Bands.
If there is extra time, we could explore improvements further along different directions.

1. We assumed zero interest rates, no transaction costs. These numbers could be found in 13-week T-bill rate and specific stock exchange.
2. We didn’t consider the turnover rate and the other risks. Var could be added as a factor.
3. This trading strategy can’t pinpoint the exact minutes and seconds to trade just show the date to buy or sell your asset. HFT strategy and more advanced algorithms should be applied.
4. Transactions need to be made by hands and it’s not automatic without the Bloomberg API.
5. More stocks could be added as a portfolio. Each ratio should be calculated with the buy time and sell time.