Mini Project Checkpoint Report

Introduction and Overview

This assignment is about stock sell/buy prediction based on real stock data. In this assignment, we are asked to pick five stocks that had a good performance during the past year from NASDAQ, and almost all the tasks were based on these five stocks. The datasets record the data including date, opening price, high price, low price, closing price, volume. They could be downloaded from alpaca. markets used alpaca API. The content of the checkpoint report could be divided into three parts: firstly, show the program and modules (especially SMA and EMA) for baseline in stock prediction; additionally, explain the methods and metrics of back-test of each of 5 stocks and across them; finally, mention contribution of each team member.

Baseline and ML model

Download the dataset

First of all, the five stocks that we have chosen from NASDAQ are AAPL, TSLA, NFLX, GOOG, and NVDA. These five all have good performance during the past year and we suppose that we can have much more benefit on them.

To get the prices of these five stocks, we defined a class named Get_prices, and in this class, we had some functions that obtaining the past year opening data, closing data, and complete data.

```
ENDPOINT="https://paper-api.alpaca.markets"
API_KEY_ID="PKTCAGBR2D97MZXNAQMF" # Put in yours here - Needed for paper trading
SECRET_KEY="PDuEwQYloK9uBw8nJDqeVkMqJJ3alIe8KT20eahJ" # Put in yours here - Needed for paper trading

class Get_prices:
    def __init__(self,STOCK):
        self.api = alpaca.REST(API_KEY_ID, SECRET_KEY, ENDPOINT)
        self.STOCK = STOCK
        # Anything else you want to initialize or a method you want to
        # call during initialization of class - Feel free to add

        # Get past 90 days closing prices
        # self.get_pastyear_closing_prices()


    def get_pastyear_closing_prices(self):
        # barset = self.api.get_barset(self.STOCK, 'day', limit=255).df
        barset = self.api.get_barset(self.STOCK, 'day', limit=365)
        bars = barset[self.STOCK]
        pastyear_closing_prices = [bars[index].c for index in range(len(bars))]
        return pastyear_closing_prices

    def get_pastyear_opening_prices(self):
        # barset = self.api.get_barset(self.STOCK, 'day', limit=255).df
        barset = self.api.get_barset(self.STOCK, 'day', limit=365)
        bars = barset[self.STOCK]
        pastyear_opening_prices = [bars[index].o for index in range(len(bars))]
        return pastyear_opening_prices

    def get_pastyear_prices(self):
        # barset = self.api.get_barset(self.STOCK, 'day', limit=255).df
        return self.api.get_barset(self.STOCK, 'day', limit=365)
```

Then called this class to download the datasets and saved them as CSV files successfully.

```
aapl =  Get_prices('AAPL')
aapl_opening_prices = aapl.get_pastyear_opening_prices()
aapl_closing_prices = aapl.get_pastyear_closing_prices()

tsla =  Get_prices('TSLA')
tsla_opening_prices = tsla.get_pastyear_opening_prices()
tsla_closing_prices = tsla.get_pastyear_closing_prices()

nflx =  Get_prices('NFLX')
nflx_opening_prices = nflx.get_pastyear_opening_prices()
nflx_closing_prices = nflx.get_pastyear_closing_prices()

nvda =  Get_prices('NVDA')
nvda_opening_prices = nvda.get_pastyear_opening_prices()
nvda_closing_prices = nvda.get_pastyear_closing_prices()

goog =  Get_prices('GOOG')
goog_opening_prices = goog.get_pastyear_opening_prices()
goog_closing_prices = goog.get_pastyear_closing_prices()

tsla.get_pastyear_prices().df['TSLA'].to_csv('tsla.csv')
nflx.get_pastyear_prices().df['NFLX'].to_csv('nflx.csv')
aapl.get_pastyear_prices().df['AAPL'].to_csv('aapl.csv')
nvda.get_pastyear_prices().df['NVDA'].to_csv('nvda.csv')
goog.get_pastyear_prices().df['GOOG'].to_csv('goog.csv')
```

SMA and EMA

Simple moving average (SMA) Simple and easy to understand average is one of the most basic technical indicators. Simple and easy-to-understand average means that we add up the prices over time and divide by the length of the period to get it. The idea of a simple moving average is that we give each day the same weight because we assume that whether it is a date closer to our current date or date further away, the impact on our current price is the same. When our date keeps moving backward, our "window" is also sliding at the same time, so it is called moving average.

If expressed as a formula, it is the following formula:

$$SMA = \sum_{i=1}^{N} P_i / N$$

$P_i$represents the price of the i-th day.
N represents the number of time windows.

As for EMA, the exponential moving averages are a staple of technical analysis and are used in countless technical indicators. However, the exponential moving average is a cumulative calculation and includes all data. The past ones contribute less and less to the mean, while the recent values contribute more. This approach allows moving averages to be more sensitive to changes in the data.

$$K = \frac{2}{(n+1)}$$

$$EMA = EMA_{[-1]} + K \times (input - EMA_{[-1]})$$

or

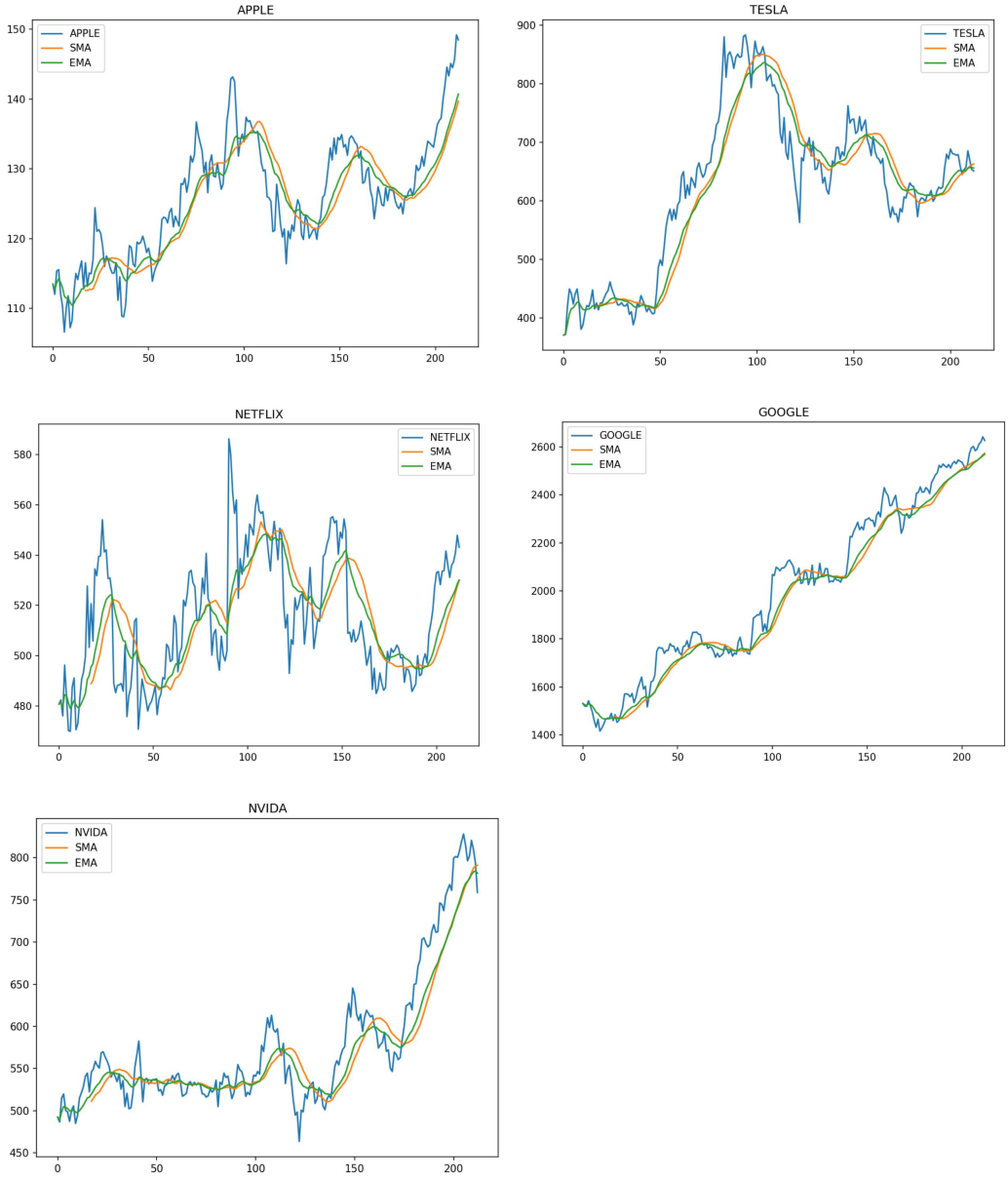
$$EMA = K \times input + (1 - K) \times EMA_{[-1]}$$

The two calculations are mathematically equivalent.

AS for them, we have defined a function named SMA_EMA_plot which could build a dataframe to store the prices of closing. And then we used rolling(), ewm(), and mean() functions to acquire SMA and EMA values. After that, plot them. The period of them is 18, which we came up with after many attempts.

```
def SMA_EMA_plot(data,window,name):

    data_df = pd.DataFrame(columns=['close'])
    data_df['close'] = data

    SMA = pd.DataFrame(columns = ['close'])
    EMA = pd.DataFrame(columns=['close'])

    SMA['close'] = data_df['close'].rolling(window = window).mean()
    EMA['close'] = data_df['close'].ewm(span = window).mean()


    plt.figure(figsize=(8, 6))
    plt.plot(data, label = name)
    plt.plot(SMA['close'],label = 'SMA')
    plt.plot(EMA['close'],label = 'EMA')
    plt.title(name)
    plt.legend()
    plt.show()
```

The results of the five stocks are as follows:



The blue line represents the stock closing price, orange for SMA, and green for EMA. EMA reflects the very recent price movements. Though SMA reflects more recent time than MA, its weights of recent time are not as heavy as exponential weight. Thus, when SMA and EMA have intersections, there will be price fluctuations. There will be a downward trend if EMA is less than SMA after the crossing point since EMA is more sensitive to recent prices than SMA. Otherwise, EMA is more, an upward tendency will appear of the price.

LR model

To build a supervised machine learning model, we need to set a series of labels. To get them, we used SMA and EMA values, compared them, and set 0 and 1 as buy and sell signals. If SMA is more, it is an indicator of sell, because the price will go down after a few days, so we append value 1 to the list. Or, append value 0. After that, build a dataframe that is composed of closing prices and a buy-sell signal list.

In this way, we finally obtained the whole dataset and split it into four subsets: x_train, x_test, y_train, and y_test. Fit them in the LogisticRegression() and score the model, which is used to calculate the accuracy of the prediction model. The result was 0.62857, I suppose that it is not bad in the stock forecast.

```
SMA_18 = pd.DataFrame(columns=['close'])
EMA_18 = pd.DataFrame(columns=['close'])
aapl_df = pd.DataFrame(columns=['open','close'])
aapl_df['close'] = aapl_closing_prices
SMA_18['close'] = aapl_df['close'].rolling(window = 18).mean()
EMA_18['close'] = aapl_df['close'].ewm(span = 18).mean()
```

```
buy_sell = []
for i in range(len(SMA_18)):

      if i > 17:

        if float(SMA_18.loc[i]) > float(EMA_18.loc[i]):

              buy_sell.append(1)

        elif float(SMA_18.loc[i]) < float(EMA_18.loc[i]):

              buy_sell.append(0)

data = pd.DataFrame(columns=['price','label'])
data['price'] = aapl_closing_prices[18:]
data['label'] = buy_sell
```

```
x_train,x_test, y_train, y_test = train_test_split(data['price'],data['label'],test_size=0.2)
x_train = x_train.values.reshape(-1,1)
x_test = x_test.values.reshape(-1,1)
LR = LogisticRegression()
LR.fit(x_train,y_train)
LR.score(x_test,y_test)
```

```
0.6285714285714286
```

back-test

back-test of 5 stocks

In this part, we back-tested our strategy on the each of 5 stocks chosen and implemented another test across them.

Before that, an appropriate strategy is a must of the back-test. There are some requirements of the strategy: no day-trading, buy and sell automatically, make sure the profit margin of at least 0.5%.

We defined another class which is named TestStrategy based on the backtrader library. The log function records the data of the operations. The notify_order function shows the order processing status: if the order has already been processed, do nothing; if the order is complete, check it; if the order is rejected, show the reason for it (Order Canceled/Margin/Rejected); and if order status processing completed, set to None. The notify_trade shows the transaction results like the gross margin and net profit for transactions. The nest function includes the next steps: recording closing price; Checking whether the order is being placed, if so, the second-order cannot be submitted; buying if EMA > SMA and selling if EMA < SMA; at last, show the final value of the whole progress.

```
class TestStrategy(bt.Strategy):

    def log(self, txt, dt=None, doprint=False):

        if doprint:
            dt = dt or self.datas[0].datetime.date(0)
            print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):

        # initialize
        self.dataclose = self.datas[0].close
        self.order = None
        self.buyprice = None
        self.buycomm = None

        # 18-EMA
        self.ema18 = bt.indicators.ExponentialMovingAverage(
            self.datas[0], period=18)
        # 18-SMA
        self.sma18 = bt.indicators.SimpleMovingAverage(
            self.datas[0], period=18)

    def notify_order(self, order):
        """
        Order status processing

        Arguments:
            order {object} -- Order status
        """
        if order.status in [order.Submitted, order.Accepted]:
            # If the order has already been processed, do nothing
            return

        # Check if the order is complete
        if order.status in [order.Completed]:
            if order.isbuy():
                self.buyprice = order.executed.price
                self.buycomm = order.executed.comm
            self.bar_executed = len(self)

        # Order rejected due to lack of funds etc.
        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')
```

```
        # Order status processing completed, set to None
        self.order = None

    def notify_trade(self, trade):
        """
        Transaction results

        Arguments:
            trade {object} -- Transaction status
        """
        if not trade.isclosed:
            return

        # Displays gross margin and net profit for transactions
        self.log('OPERATION PROFIT, GROSS %.2f, NET %.2f' %
                 (trade.pnl, trade.pnlcomm), doprint=True)

    def next(self):
        ''' next process '''

        # record closing price
        self.log('Close, %.2f' % self.dataclose[0])

        # Whether the order is being placed, if so,
        # the second order cannot be submitted
        if self.order:
            return

        # have you bought
        if not self.position:
            # Haven't bought yet, if EMA > SMA indicates uptrend, buy
            if self.ema18[0] > self.sma18[0]:
                self.log('BUY CREATED, %.2f' %self.dataclose[0], doprint=True)
                self.order = self.buy()
        else:
            # Already bought, if EMA < SMA, indicating a downtrend, sell
            if self.ema18[0] < self.sma18[0]:
                self.log('SELL CREATED, %.2f' %self.dataclose[0], doprint=True)
                self.order = self.sell()

    def stop(self):
        self.log(u' Ending Value %.2f' %
                 (self.broker.getvalue()), doprint=True)
```

Then, define a function of run_model. Initialize the model, build strategy, set the number of the stack of each operation, load data into the model, set initial capital and commission, run it, and plot the result.

```
def run_model(dataname,fromdate,todate,startcash,stake):

      if __name__ == '__main__':

    # Initialize the model
        cerebro = bt.Cerebro()

    # build strategy
        strats = cerebro.addstrategy(TestStrategy)
    # Buy stake shares each time
        cerebro.addsizer(bt.sizers.FixedSize, stake=stake)

    # Load data into the model
        data = bt.feeds.GenericCSVData(dataname = dataname,
                            fromdate = fromdate,
                            todate = todate,
                            datetime = 0,
                            open = 1,
                            high = 2,
                            low = 3,
                            close = 4,
                            volume = 5,
                            dtformat='%Y%m%d')
        cerebro.adddata(data)

    # Set initial capital and commission
        cerebro.broker.setcash(startcash)
#    cerebro.broker.setcommission(0.005)

    # Funding before strategy execution
        print('startcash: %.2f' % cerebro.broker.getvalue())

    # Policy enforcement
        cerebro.run()
        plt.rcParams['figure.figsize'] = [12, 8]
        cerebro.plot(iplot = True)
```

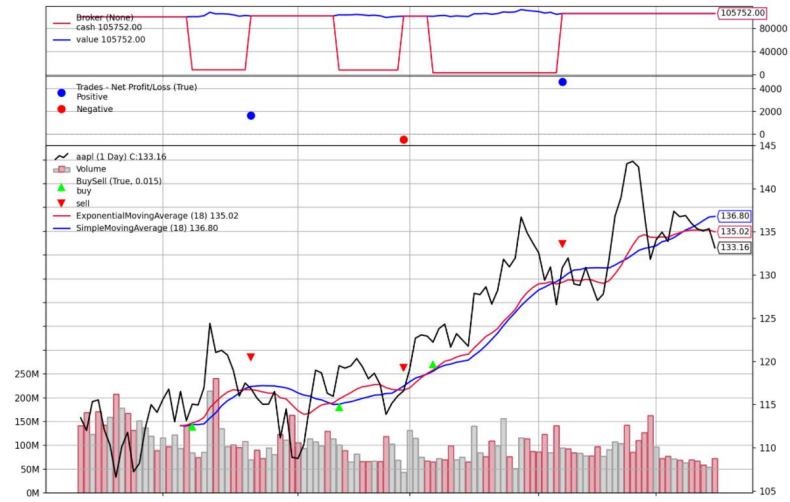
Set the fromdate, todate, and start cash.

```
fromdate = datetime(2020,7,14)
todate = datetime(2021,2,17)
startcash = 100000
```

Then, call the run_model of each of the stocks. Here are the results.

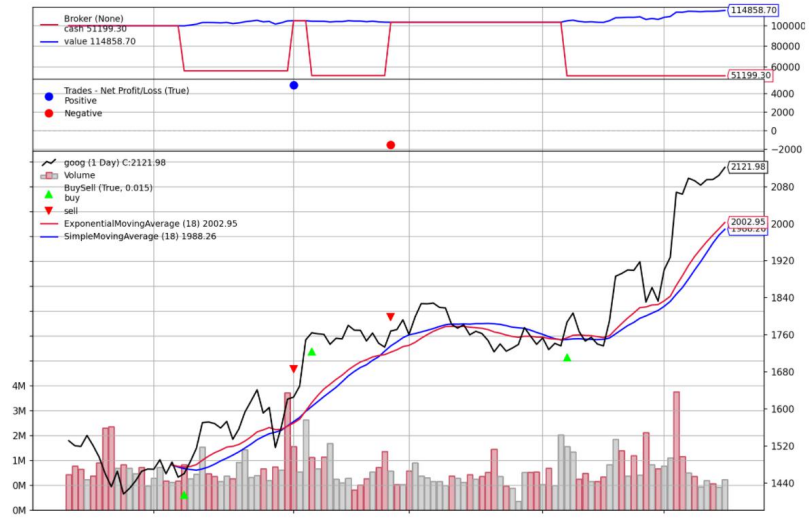
```
#AAPLE
run_model('aapl.csv',fromdate,todate,startcash,stake = 800)
```

```
startcash: 100000.00
2020-10-06, BUY CREATED, 113.16
2020-10-20, SELL CREATED, 117.50
2020-10-21, OPERATION PROFIT, GROSS 1640.00, NET 1640.00
2020-11-10, BUY CREATED, 115.97
2020-11-25, SELL CREATED, 116.02
2020-11-27, OPERATION PROFIT, GROSS -496.00, NET -496.00
2020-12-03, BUY CREATED, 122.94
2021-01-06, SELL CREATED, 126.58
2021-01-07, OPERATION PROFIT, GROSS 4608.00, NET 4608.00
2021-01-25, BUY CREATED, 142.90
2021-01-26, BUY CREATED, 143.19
2021-01-27, BUY CREATED, 142.53
2021-01-28, BUY CREATED, 137.04
2021-01-29, BUY CREATED, 131.83
2021-02-01, BUY CREATED, 134.10
2021-02-02, BUY CREATED, 134.98
2021-02-03, BUY CREATED, 133.90
2021-02-04, BUY CREATED, 137.40
2021-02-05, BUY CREATED, 136.77
2021-02-16,  Ending Value 105752.00
```



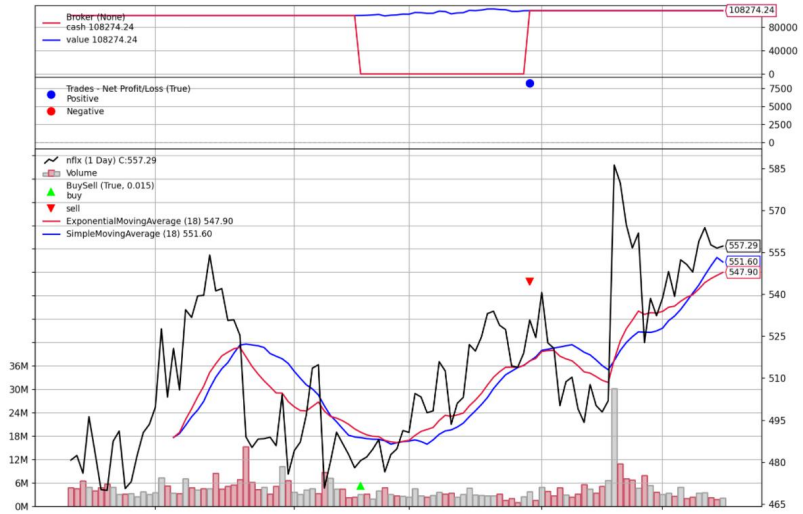

```
#GOOGLE
run_model('goog.csv',fromdate,todate,startcash,stake =30)
```

```
startcash: 100000.00
2020-10-06, BUY CREATED, 1452.47
2020-10-30, SELL CREATED, 1621.08
2020-11-02, OPERATION PROFIT, GROSS 4916.10, NET 4916.10
2020-11-04, BUY CREATED, 1749.14
2020-11-23, SELL CREATED, 1733.52
2020-11-24, OPERATION PROFIT, GROSS -1515.00, NET -1515.00
2021-01-06, BUY CREATED, 1736.19
2021-02-16,  Ending Value 114858.70
```



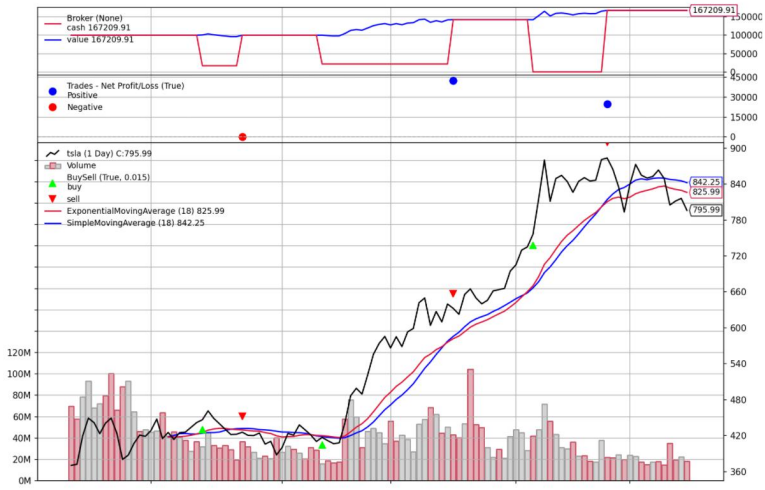
```
#NETFLIX
run_model('nflx.csv',fromdate,todate,startcash,stake =208)
```

```
startcash: 100000.00
2020-10-06, BUY CREATED, 505.87
2020-10-07, BUY CREATED, 534.54
2020-10-08, BUY CREATED, 531.79
2020-10-09, BUY CREATED, 539.45
2020-10-12, BUY CREATED, 539.70
2020-10-13, BUY CREATED, 554.08
2020-10-14, BUY CREATED, 541.26
2020-10-15, BUY CREATED, 542.09
2020-10-16, BUY CREATED, 530.70
2020-10-19, BUY CREATED, 530.92
2020-11-11, BUY CREATED, 490.76
2020-11-12, BUY CREATED, 486.77
2020-11-13, BUY CREATED, 482.82
2020-11-16, BUY CREATED, 478.04
2020-12-28, SELL CREATED, 519.01
2020-12-29, OPERATION PROFIT, GROSS 8274.24, NET 8274.24
2020-12-29, BUY CREATED, 530.87
2021-01-20, BUY CREATED, 586.25
2021-01-21, BUY CREATED, 579.84
2021-01-22, BUY CREATED, 564.91
2021-01-25, BUY CREATED, 556.63
2021-01-26, BUY CREATED, 561.94
2021-01-27, BUY CREATED, 522.80
2021-01-28, BUY CREATED, 538.59
2021-01-29, BUY CREATED, 532.39
2021-02-01, BUY CREATED, 538.71
2021-02-02, BUY CREATED, 548.17
2021-02-03, BUY CREATED, 539.30
2021-02-04, BUY CREATED, 552.37
2021-02-05, BUY CREATED, 550.73
2021-02-16,  Ending Value 108274.24
```



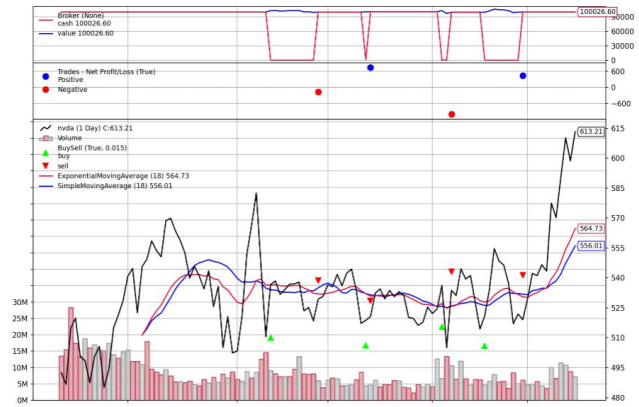

```
#TESLA
run_model('tsla.csv',fromdate,todate,startcash,stake =187)
```

```
startcash: 100000.00
2020-10-12, BUY CREATED, 442.00
2020-10-21, SELL CREATED, 422.50
2020-10-22, OPERATION PROFIT, GROSS -37.16, NET -37.16
2020-11-10, BUY CREATED, 410.41
2020-12-14, SELL CREATED, 639.79
2020-12-15, OPERATION PROFIT, GROSS 42396.64, NET 42396.64
2021-01-05, BUY CREATED, 735.05
2021-01-25, SELL CREATED, 880.81
2021-01-26, OPERATION PROFIT, GROSS 24850.43, NET 24850.43
2021-02-16,  Ending Value 167209.91
```



```
#NVIDA
run_model('nvda.csv',fromdate,todate,startcash,stake =190)
```

```
startcash: 100000.00
2020-10-06, BUY CREATED, 549.36
2020-10-07, BUY CREATED, 558.56
2020-10-08, BUY CREATED, 553.79
2020-10-09, BUY CREATED, 550.54
2020-10-12, BUY CREATED, 568.71
2020-10-13, BUY CREATED, 569.91
2020-10-14, BUY CREATED, 563.61
2020-10-15, BUY CREATED, 558.85
2020-10-16, BUY CREATED, 552.46
2020-11-06, BUY CREATED, 582.59
2020-11-09, BUY CREATED, 545.41
2020-11-10, BUY CREATED, 510.55
2020-11-24, SELL CREATED, 518.46
2020-11-25, OPERATION PROFIT, GROSS -161.50, NET -161.50
2020-12-03, BUY CREATED, 536.12
2020-12-04, BUY CREATED, 542.39
2020-12-07, BUY CREATED, 544.43
2020-12-08, BUY CREATED, 533.84
2020-12-09, BUY CREATED, 517.23
2020-12-10, SELL CREATED, 518.72
2020-12-11, OPERATION PROFIT, GROSS 746.70, NET 746.70
2020-12-15, BUY CREATED, 534.49
2020-12-16, BUY CREATED, 529.49
2020-12-17, BUY CREATED, 533.75
2021-01-04, BUY CREATED, 524.54
2021-01-06, SELL CREATED, 504.96
2021-01-07, OPERATION PROFIT, GROSS -1005.10, NET -1005.10
2021-01-11, BUY CREATED, 544.57
2021-01-12, BUY CREATED, 539.40
2021-01-13, BUY CREATED, 541.22
2021-01-14, BUY CREATED, 527.83
2021-01-15, BUY CREATED, 514.38
2021-01-28, SELL CREATED, 521.93
2021-01-29, OPERATION PROFIT, GROSS 446.50, NET 446.50
2021-02-05, BUY CREATED, 543.40
2021-02-08, BUY CREATED, 577.51
2021-02-09, BUY CREATED, 570.25
2021-02-10, BUY CREATED, 590.57
2021-02-11, BUY CREATED, 610.21
2021-02-12, BUY CREATED, 598.64
2021-02-16, BUY CREATED, 613.21
2021-02-16,  Ending Value 100026.60
```



In these graphs, there are some signals about prices, SMA, EMA, buy and sell, volume, positive and negative points, and cash lines. At the date of each red and green square, there are good chances to buy or sell. The window of SMA and EMA is 18, this value is a relatively good indicator that I have come up with after many attempts, which is the same as the SMA and EMA training parts above. And we can get the volume of final cash in the each of graphs.

Back-test across 5 stocks

Similar to the strategy above, we build another strategy which is named TestStrategy_5stocks for this task. In this part, we assume the period is still 18, however, the stake is 50, since the start cash is 100,000, if the number of stakes is too large, it will not able to afford the expenditure (e.g., GOOG). So, to avoid this problem, we choose a small value of it. Same to before, we use SMA and EMA to determine to buy or sell on every transaction day by a loop structure through each stock. Furthermore, show the order and trade status like before.

```
class TestStrategy_5stocks(bt.Strategy):

    params = dict(period = 18,
                  stake = 50)

    def log(self, txt, dt=None, doprint=False):
        if doprint:
            dt = dt or self.datas[0].datetime.date(0)
            print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):

        self.dataclose = self.datas[0].close
        self.order = None
        self.buyprice = None
        self.buycomm = None
        self.smainds = dict()
        self.emainds = dict()

        for i, d in enumerate(self.datas):
            self.smainds[d] = bt.ind.SMA(d.close, period=self.p.period)
            self.emainds[d] = bt.ind.EMA(d.close, period=self.p.period)

    def next(self):
        for i, d in enumerate(self.datas):
            pos = self.getposition(d)
            if not len(pos):
                if self.emainds[d][0] > self.smainds[d][0]: # meet buy condition
                    self.log('BUY CREATED, %.2f' %self.dataclose[0], doprint=True)
                    self.buy(data = d, size = self.p.stake)
            elif self.emainds[d][0] < self.smainds[d][0]: # meet sell condition
                self.log('SELL CREATED, %.2f' %self.dataclose[0], doprint=True)
                self.sell(data = d)


    def notify_order(self, order):

        if order.status in [order.Submitted, order.Accepted]:
            return

        if order.status in [order.Completed]:
            if order.isbuy():
                self.buyprice = order.executed.price
                self.buycomm = order.executed.comm
            self.bar_executed = len(self)

        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')

        self.order = None

    def notify_trade(self, trade):

        if not trade.isclosed:
            return

        self.log('OPERATION PROFIT, GROSS %.2f, NET %.2f' %
                 (trade.pnl, trade.pnlcomm), doprint=True)

    def stop(self):
        self.log(u' Ending Value %.2f' %
                 (self.broker.getvalue()), doprint=True)
```

Next, initialize parameters, run the model and plot it.

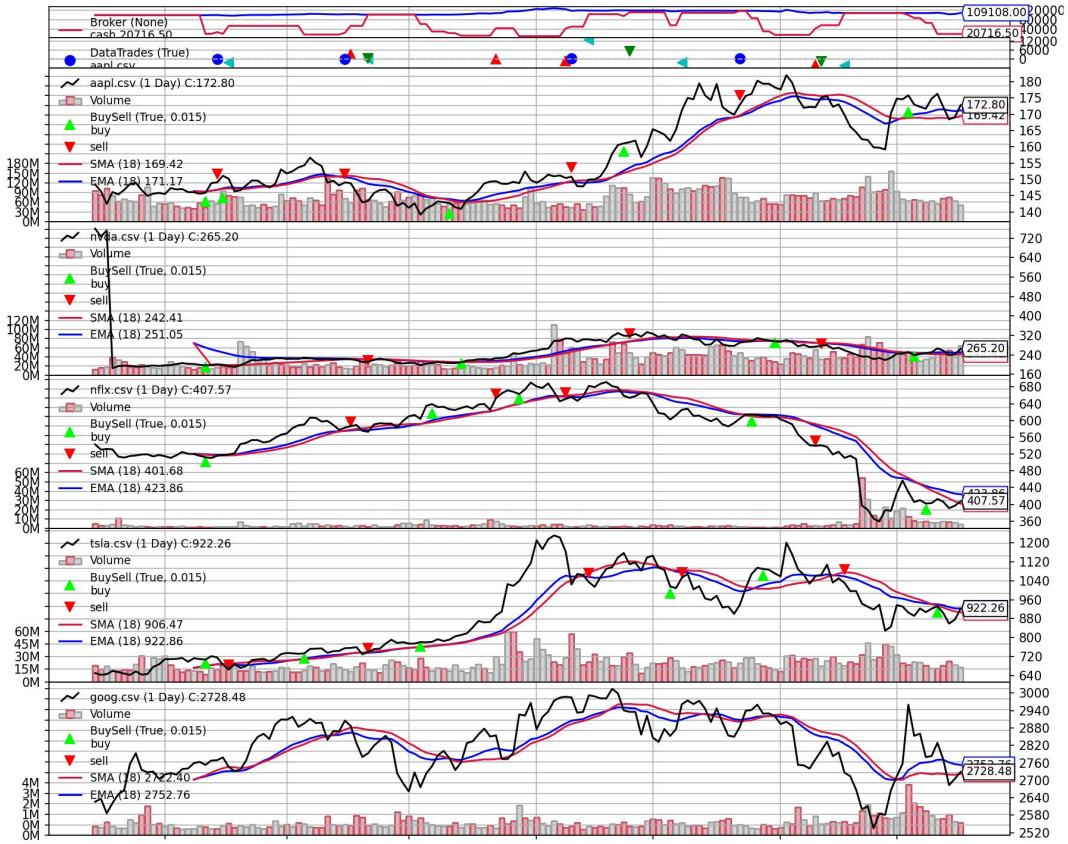
```
def run_model_5stocks(strategy,datalist, stakenum):

      cerebro = bt.Cerebro()

      cerebro.addstrategy(TestStrategy_5stocks)

      cerebro.addsizer(bt.sizers.FixedSize, stake=stakenum)

      cerebro.broker.setcash(100000)

      fromdate = datetime(2021,7,15)
      todate = datetime(2022,2,16)

      for key in datalist:
            data = bt.feeds.GenericCSVData(dataname = key,
                            fromdate = fromdate,
                            todate = todate,
                            datetime = 0,
                            open = 1,
                            high = 2,
                            low = 3,
                            close = 4,
                            volume = 5,
                            dtformat='%Y%m%d')

            cerebro.adddata(data, name = key)

      results = cerebro.run()

      portvalue = cerebro.broker.getvalue()
      pnl = portvalue - startcash

      print(f'total: {round(portvalue,2)}')
      print(f'benefit: {round(pnl,2)}')

      plt.rcParams['figure.figsize'] = [12, 10]
      cerebro.plot(iplot = True)
```

Finally, we can easily get the result of the whole progress.

```
data_list = ['aapl.csv','nvda.csv','nflx.csv','tsla.csv','goog.csv']
stakenum = 50
run_model_5stocks(TestStrategy_5stocks,data_list,stakenum)
```

```
2021-08-10, BUY CREATED, 145.61
2021-08-10, BUY CREATED, 145.61
2021-08-10, BUY CREATED, 145.61
2021-08-10, BUY CREATED, 145.61
2021-08-12, SELL CREATED, 148.88
2021-08-13, OPERATION PROFIT, GROSS 145.00, NET 145.00
2021-08-13, BUY CREATED, 149.10
2021-08-16, SELL CREATED, 151.12
2021-08-17, OPERATION PROFIT, GROSS -2002.00, NET -2002.00
2021-08-24, BUY CREATED, 149.63
2021-08-25, BUY CREATED, 148.37
2021-08-26, BUY CREATED, 147.54
2021-08-27, BUY CREATED, 148.60
2021-08-30, BUY CREATED, 153.10
2021-08-31, BUY CREATED, 151.83
2021-09-01, BUY CREATED, 152.54
2021-09-02, BUY CREATED, 153.71
2021-09-02, BUY CREATED, 153.71
2021-09-03, BUY CREATED, 154.31
2021-09-07, BUY CREATED, 156.70
2021-09-08, BUY CREATED, 155.12
2021-09-09, BUY CREATED, 154.10
2021-09-10, BUY CREATED, 149.00
2021-09-14, SELL CREATED, 148.19
2021-09-15, OPERATION PROFIT, GROSS -1.50, NET -1.50
2021-09-15, SELL CREATED, 149.10
2021-09-16, OPERATION PROFIT, GROSS 3361.50, NET 3361.50
2021-09-20, SELL CREATED, 142.97
2021-09-20, SELL CREATED, 142.97
2021-09-21, OPERATION PROFIT, GROSS 681.00, NET 681.00
2021-09-21, OPERATION PROFIT, GROSS 161.00, NET 161.00
2021-10-01, BUY CREATED, 142.66
2021-10-05, BUY CREATED, 141.10
2021-10-08, BUY CREATED, 143.01
2021-10-11, BUY CREATED, 142.80
2021-10-12, BUY CREATED, 141.46
2021-10-12, BUY CREATED, 141.46
2021-10-13, BUY CREATED, 140.96
2021-10-14, BUY CREATED, 143.76
2021-10-15, BUY CREATED, 144.83
2021-10-18, BUY CREATED, 146.54
2021-10-19, BUY CREATED, 148.78
2021-10-20, SELL CREATED, 149.35
2021-10-20, BUY CREATED, 149.35
2021-10-21, OPERATION PROFIT, GROSS 18.50, NET 18.50
2021-10-21, BUY CREATED, 149.46
2021-10-22, BUY CREATED, 148.69
2021-10-25, BUY CREATED, 148.65
2021-10-26, BUY CREATED, 149.26
2021-10-26, BUY CREATED, 149.26
2021-10-27, BUY CREATED, 148.85
2021-10-28, BUY CREATED, 152.48
2021-10-29, BUY CREATED, 149.80
2021-11-01, BUY CREATED, 148.99
2021-11-02, BUY CREATED, 150.00
2021-11-03, BUY CREATED, 151.49
2021-11-04, BUY CREATED, 150.96
2021-11-05, SELL CREATED, 151.24
```

```
2021-11-05, BUY CREATED, 151.24
2021-11-08, OPERATION PROFIT, GROSS -867.00, NET -867.00
2021-11-08, SELL CREATED, 150.45
2021-11-08, BUY CREATED, 150.45
2021-11-09, OPERATION PROFIT, GROSS 394.50, NET 394.50
2021-11-09, BUY CREATED, 150.83
2021-11-10, BUY CREATED, 147.92
2021-11-11, SELL CREATED, 147.88
2021-11-11, BUY CREATED, 147.88
2021-11-12, OPERATION PROFIT, GROSS 12549.00, NET 12549.00
2021-11-12, BUY CREATED, 149.99
2021-11-15, BUY CREATED, 150.00
2021-11-16, BUY CREATED, 151.00
2021-11-17, BUY CREATED, 153.65
2021-11-19, BUY CREATED, 160.56
2021-11-22, SELL CREATED, 161.08
2021-11-23, OPERATION PROFIT, GROSS 5305.00, NET 5305.00
2021-12-02, BUY CREATED, 163.79
2021-12-06, SELL CREATED, 165.29
2021-12-07, OPERATION PROFIT, GROSS -2012.00, NET -2012.00
2021-12-13, BUY CREATED, 175.82
2021-12-14, BUY CREATED, 174.34
2021-12-15, BUY CREATED, 179.30
2021-12-16, BUY CREATED, 172.25
2021-12-17, BUY CREATED, 170.86
2021-12-20, SELL CREATED, 169.88
2021-12-20, BUY CREATED, 169.88
2021-12-21, OPERATION PROFIT, GROSS 493.50, NET 493.50
2021-12-21, BUY CREATED, 172.98
2021-12-22, BUY CREATED, 175.63
2021-12-22, BUY CREATED, 175.63
2021-12-23, BUY CREATED, 176.29
2021-12-27, BUY CREATED, 180.33
2021-12-27, BUY CREATED, 180.33
2021-12-28, BUY CREATED, 179.39
2021-12-29, BUY CREATED, 179.38
2022-01-07, SELL CREATED, 172.25
2022-01-10, OPERATION PROFIT, GROSS -3889.00, NET -3889.00
2022-01-10, SELL CREATED, 172.17
2022-01-11, OPERATION PROFIT, GROSS -1248.00, NET -1248.00
2022-01-14, SELL CREATED, 173.05
2022-01-18, OPERATION PROFIT, GROSS -4191.50, NET -4191.50
2022-01-31, BUY CREATED, 174.80
2022-02-01, BUY CREATED, 174.60
2022-02-01, BUY CREATED, 174.60
2022-02-02, BUY CREATED, 175.83
2022-02-02, BUY CREATED, 175.83
2022-02-03, BUY CREATED, 173.00
2022-02-04, BUY CREATED, 172.32
2022-02-04, BUY CREATED, 172.32
2022-02-07, BUY CREATED, 171.71
2022-02-08, BUY CREATED, 174.82
2022-02-08, BUY CREATED, 174.82
2022-02-09, BUY CREATED, 176.28
2022-02-10, BUY CREATED, 172.26
2022-02-11, BUY CREATED, 168.44
2022-02-14, BUY CREATED, 169.13
2022-02-15, BUY CREATED, 172.80
2022-02-15,  Ending Value 109108.00
total: 109108.0
benefit: 9108.0
```



contribution of each team member

In the mini-project, there are two people in our team: Xiangyu Gao and Rui Li. Xiangyu is mainly responsible for the model establishment, writing main codes, and parameter debugging, while Rui is mainly responsible for finding reference materials, correcting codes, and selecting model methods. Throughout the project, we found each other online and decided to team up together. During this time, we were coding face-to-face three times a week, because face-to-face communication allowed us to program more efficiently. Although we have our division of labor, the overall procedure is the result of both of us. We all feel very happy to work with each other.