Mini Project 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 the following parts: firstly, show the program and modules in stock prediction; additionally, explain the methods and metrics of back-test of each of 5 stocks and across them; besides, improved the model to the neural network and trained it; furthermore, applied the model on paper trading and ran it live; finally, mention contribution of each team member.

Trading system development

The main idea is to construct an abstract TradingSystem class so that we can implement custom rulesets for each type of system we wish to trade. The code is fairly straightforward, allowing us to initialize the system and thread an infinite loop.

```
class TradingSystem(abc.ABC):

    def __init__(self, api, symbol, time_frame, system_id, system_label):
        self.api = api
        self.symbol = symbol
        self.time_frame = time_frame
        self.system_id = system_id
        self.system_label = system_label
        thread = threading.Thread(target=self.system_loop)
        thread.start()

    @abc.abstractmethod
    def place_buy_order(self):
        pass

    @abc.abstractmethod
    def place_sell_order(self):
        pass

    @abc.abstractmethod
    def system_loop(self):
        pass
```

AI trading model development

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.

Connect to a brokerage firm, which will allow us to receive real-time data on transactions in securities of our interest. In this article, I'll be using Alpaca, the easiest free way to get started with algorithmic trading, and for our purposes, AI trading. Create an account and go to the dashboard to generate an API key.

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 date, 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')
```

Preprocess the dataset

For this system, I will build and train an AI model as a portfolio manager for my system. The idea is to train a neural network to buy at a certain negative change threshold and sell stock prices at a certain positive change threshold. We are essentially teaching our AI to buy sediment and sell sediment. To train this neural network, I will build and annotate a dataset based on weekly historical market data for stocks and create a function called a signal that will produce values in the set {0,1, -1} based on changing thresholds.

To obtain an intact dataset of neural network training, including labels and predictors, we preprocessed the dataset: let the label be three different values (-1, 0, 1 separately). If the stock's closing price on the current day is lower than the stock's closing price 7 days ago by $0.5, it means that the stock is moving lower, and the label is assigned as 1, the stock needs to be sold. Or if the price is higher than that of a week ago about $0.5, which means it is moving higher, the label is set as -1 and we need to buy it instantly.

	Delta Close	Signal
1		
2	-2.559997	1
3	3.410003	-1
4	3.679993	-1
5	3.25	-1
6	-1.709992	1
7	-3.440002	1
8	-4.220001	1
9	8.600006	-1
10	-2.480011	1
11	0.050003	0
12	2.410004	-1
13	-2.190003	1
14	9.110001	-1
15	3.889999	-1
16	-0.990005	0
17	0.580002	0
18	-1.039994	1

Structure of neural network

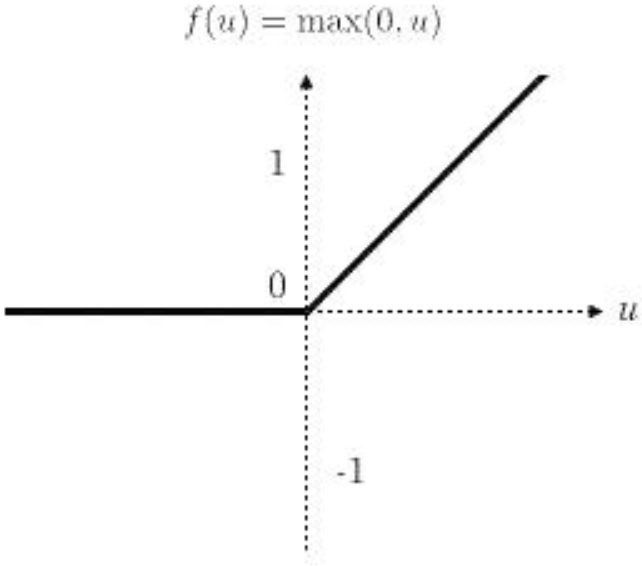
Now consider the architecture of this neural network. We will save the weights after backpropagation so that after successfully testing the model, we can deploy it.

```
class PMModelDevelopment:

    def __init__(self,STOCK_filename):
        data = pd.read_csv(STOCK_filename)
        X = data['Delta Close']
        y = data.drop(['Delta Close'], axis=1)

        X_train, X_test, y_train, y_test = train_test_split(X, y)

        network = Sequential()

        network.add(Dense(1, input_shape=(1,), activation='relu'))
        network.add(Dense(3, activation='relu'))
        network.add(Dense(3, activation='relu'))
        network.add(Dense(3, activation='relu'))
        network.add(Dense(1, activation='relu'))

        network.compile(
                    optimizer='rmsprop',
                    loss='hinge',
                    metrics=['accuracy']
                    )
        network.fit(X_train.values, y_train.values, epochs=100)

        y_pred = network.predict(X_test.values)
        y_pred = np.around(y_pred, 0)
        print(classification_report(y_test, y_pred))

        model = network.to_json()
        with open("model.json", "w") as json_file:
            json_file.write(model)

        network.save_weights("weights.h5")
```

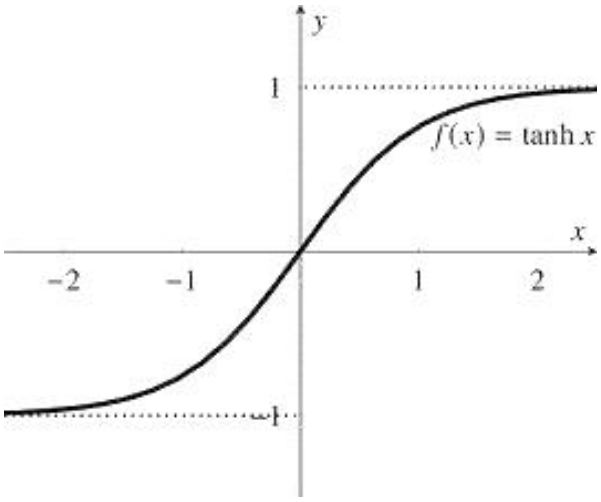
We will use a sequence model, in this case, the choice of activation function is crucial, if we close our eyes and choose ReLu with a binary cross-entropy loss function, we will get a confusion matrix that looks similar to the following:

	precision	recall	f1-score	support
-1	0.00	0.00	0.00	21
0	0.33	1.00	0.50	22
1	0.00	0.00	0.00	23
accuracy			0.33	66
macro avg	0.11	0.33	0.17	66
weighted avg	0.11	0.33	0.17	66

Actually, let's visualize the ReLu function, we can easily get this graph:



If we set negative input to zero, or expect binary output using loss function, hyperbolic tangent function, and hinge loss help here:



Updating our neural network, recognizing where we went wrong, we have the following model:

```
class PMModelDevelopment:

    def __init__(self,filename,stockname):#filename is a string
        data = pd.read_csv(filename)
        X = data['Delta Close']
        y = data.drop(['Delta Close'], axis=1)

        X_train, X_test, y_train, y_test = train_test_split(X, y)

        network = Sequential()

        network.add(Dense(1, input_shape=(1,), activation='tanh'))
        network.add(Dense(3, activation='tanh'))
        network.add(Dense(3, activation='tanh'))
        network.add(Dense(3, activation='tanh'))
        network.add(Dense(1, activation='tanh'))

        network.compile(
                    optimizer='rmsprop',
                    loss='hinge',
                    metrics=['accuracy']
                    )
        network.fit(X_train.values, y_train.values, epochs=100)

        y_pred = network.predict(X_test.values)
        y_pred = np.around(y_pred, 0)
        print(classification_report(y_test, y_pred))

        model = network.to_json()
        with open((stockname + "_model.json"), "w") as json_file:
            json_file.write(model)

        network.save_weights((stockname + "_weights.h5"))
```

After training, we found a significant improvement in classification reports.

	precision	recall	f1-score	support
	-1	0.83	1.00	0.91	25
	0	1.00	0.06	0.12	16
	1	0.71	1.00	0.83	25
accuracy				0.77	66
macro avg		0.85	0.69	0.62	66
weighted avg		0.83	0.77	0.69	66

AI Portfolio Manager

We've successfully developed our model by saving the model and loading it into the class that

specifically hosts it

```
class PortfolioManagementModel:

    def __init__(self,filename,stockname):
        data = pd.read_csv(filename)
        X = data['Delta Close']
        y = data.drop(['Delta Close'], axis=1)

        json_file = open((stockname + '_model.json'), 'r')
        json = json_file.read()
        json_file.close()
        self.network = model_from_json(json)

        self.network.load_weights((stockname + "_weights.h5"))

        y_pred = self.network.predict(X.values)
        y_pred = np.around(y_pred, 0)
        print(classification_report(y, y_pred))
```

We verify the structure of the neural network and correctly loaded weights by looking at the classification reports for the entire dataset.

	precision	recall	f1-score	support
-1	0.73	1.00	0.84	95
0	1.00	0.07	0.14	69
1	0.77	1.00	0.87	97
accuracy			0.75	261
macro avg	0.83	0.69	0.62	261
weighted avg	0.82	0.75	0.67	261

The result of training is as follows:

```
filename ='IBM.csv'
stockname ='IBM'
PMModelDevelopment(filename,stockname)
PortfolioManagementModel(filename,stockname)
```

```
Epoch 1/50
7/7 [==============================] - 1s 1ms/step - loss: 1.2508 - accuracy: 0.2462
Epoch 2/50
7/7 [==============================] - 0s 916us/step - loss: 1.2213 - accuracy: 0.2462
Epoch 3/50
7/7 [==============================] - 0s 964us/step - loss: 1.1999 - accuracy: 0.2462
Epoch 4/50
7/7 [==============================] - 0s 1ms/step - loss: 1.1799 - accuracy: 0.2462
Epoch 5/50
7/7 [==============================] - 0s 1ms/step - loss: 1.1607 - accuracy: 0.2462
Epoch 6/50
7/7 [==============================] - 0s 1ms/step - loss: 1.1411 - accuracy: 0.2462
Epoch 7/50
7/7 [==============================] - 0s 1ms/step - loss: 1.1226 - accuracy: 0.2462
Epoch 8/50
7/7 [==============================] - 0s 1ms/step - loss: 1.1044 - accuracy: 0.2462
Epoch 9/50
7/7 [==============================] - 0s 2ms/step - loss: 1.0883 - accuracy: 0.2462
Epoch 10/50
7/7 [==============================] - 0s 1ms/step - loss: 1.0710 - accuracy: 0.2462
Epoch 11/50
7/7 [==============================] - 0s 790us/step - loss: 1.0545 - accuracy: 0.2462
Epoch 12/50
7/7 [==============================] - 0s 981us/step - loss: 1.0372 - accuracy: 0.2462
Epoch 13/50
7/7 [==============================] - 0s 3ms/step - loss: 1.0208 - accuracy: 0.2462
Epoch 14/50
7/7 [==============================] - 0s 1ms/step - loss: 1.0046 - accuracy: 0.2462
Epoch 15/50
7/7 [==============================] - 0s 2ms/step - loss: 0.9886 - accuracy: 0.2462
Epoch 16/50
7/7 [==============================] - 0s 1ms/step - loss: 0.9731 - accuracy: 0.2462
Epoch 17/50
7/7 [==============================] - 0s 1ms/step - loss: 0.9580 - accuracy: 0.2462
Epoch 18/50
7/7 [==============================] - 0s 1ms/step - loss: 0.9427 - accuracy: 0.2462
Epoch 19/50
7/7 [==============================] - 0s 1ms/step - loss: 0.9268 - accuracy: 0.2462
Epoch 20/50
7/7 [==============================] - 0s 2ms/step - loss: 0.9108 - accuracy: 0.2462
Epoch 21/50
7/7 [==============================] - 0s 1ms/step - loss: 0.8944 - accuracy: 0.2462
Epoch 22/50
7/7 [==============================] - 0s 1ms/step - loss: 0.8804 - accuracy: 0.2462
Epoch 23/50
7/7 [==============================] - 0s 931us/step - loss: 0.8632 - accuracy: 0.2462
Epoch 24/50
7/7 [==============================] - 0s 1ms/step - loss: 0.8464 - accuracy: 0.2462
Epoch 25/50
7/7 [==============================] - 0s 1ms/step - loss: 0.8281 - accuracy: 0.2462
Epoch 26/50
7/7 [==============================] - 0s 1ms/step - loss: 0.8101 - accuracy: 0.2462
Epoch 27/50
7/7 [==============================] - 0s 1ms/step - loss: 0.7946 - accuracy: 0.2462
Epoch 28/50
7/7 [==============================] - 0s 978us/step - loss: 0.7766 - accuracy: 0.2462
Epoch 29/50
7/7 [==============================] - 0s 1ms/step - loss: 0.7592 - accuracy: 0.2462
Epoch 30/50
7/7 [==============================] - 0s 1ms/step - loss: 0.7428 - accuracy: 0.2462
```

```
Epoch 31/50
7/7 [==============================] - 0s 1ms/step - loss: 0.7256 - accuracy: 0.2462
Epoch 32/50
7/7 [==============================] - 0s 1ms/step - loss: 0.7095 - accuracy: 0.2462
Epoch 33/50
7/7 [==============================] - 0s 1ms/step - loss: 0.6905 - accuracy: 0.2462
Epoch 34/50
7/7 [==============================] - 0s 2ms/step - loss: 0.6762 - accuracy: 0.2462
Epoch 35/50
7/7 [==============================] - 0s 1ms/step - loss: 0.6610 - accuracy: 0.2462
Epoch 36/50
7/7 [==============================] - 0s 1ms/step - loss: 0.6480 - accuracy: 0.2462
Epoch 37/50
7/7 [==============================] - 0s 2ms/step - loss: 0.6324 - accuracy: 0.2462
Epoch 38/50
7/7 [==============================] - 0s 916us/step - loss: 0.6176 - accuracy: 0.2462
Epoch 39/50
7/7 [==============================] - 0s 2ms/step - loss: 0.6032 - accuracy: 0.2462
Epoch 40/50
7/7 [==============================] - 0s 936us/step - loss: 0.5885 - accuracy: 0.5897
Epoch 41/50
7/7 [==============================] - 0s 2ms/step - loss: 0.5754 - accuracy: 0.5744
Epoch 42/50
7/7 [==============================] - 0s 1ms/step - loss: 0.5628 - accuracy: 0.5692
Epoch 43/50
7/7 [==============================] - 0s 934us/step - loss: 0.5504 - accuracy: 0.5692
Epoch 44/50
7/7 [==============================] - 0s 2ms/step - loss: 0.5381 - accuracy: 0.5590
Epoch 45/50
7/7 [==============================] - 0s 920us/step - loss: 0.5270 - accuracy: 0.5487
Epoch 46/50
7/7 [==============================] - 0s 926us/step - loss: 0.5157 - accuracy: 0.5333
Epoch 47/50
7/7 [==============================] - 0s 979us/step - loss: 0.5043 - accuracy: 0.5333
Epoch 48/50
7/7 [==============================] - 0s 951us/step - loss: 0.4940 - accuracy: 0.5333
Epoch 49/50
7/7 [==============================] - 0s 931us/step - loss: 0.4840 - accuracy: 0.5282
Epoch 50/50
7/7 [==============================] - 0s 1ms/step - loss: 0.4779 - accuracy: 0.5282
              precision    recall  f1-score   support

          -1       0.70      1.00      0.83        19
           0       1.00      0.29      0.44        21
           1       0.79      1.00      0.88        26

    accuracy                           0.77        66
   macro avg       0.83      0.76      0.72        66
weighted avg       0.83      0.77      0.73        66

              precision    recall  f1-score   support

          -1       0.81      1.00      0.89        95
           0       1.00      0.33      0.50        69
           1       0.81      1.00      0.89        97

    accuracy                           0.82       261
   macro avg       0.87      0.78      0.76       261
weighted avg       0.86      0.82      0.79       261


<__main__.PortfolioManagementModel at 0x1f4873c1f10>
```

Deploy AI models to trading systems

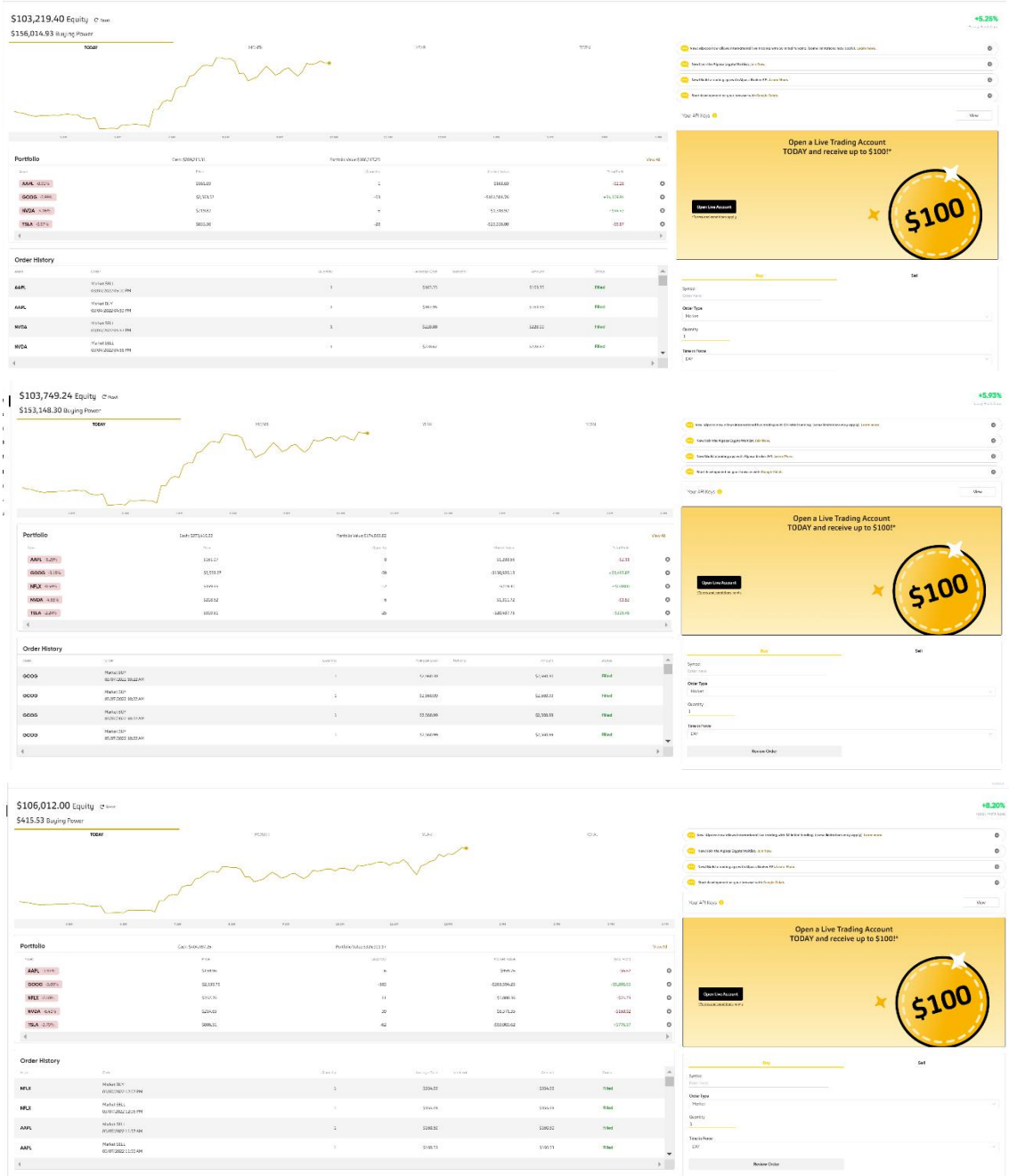
When rethinking the implementation of the abstract TradingSystem class, we have PortfolioManagementSystem. We updated the abstract functions to fulfill their respective purposes.

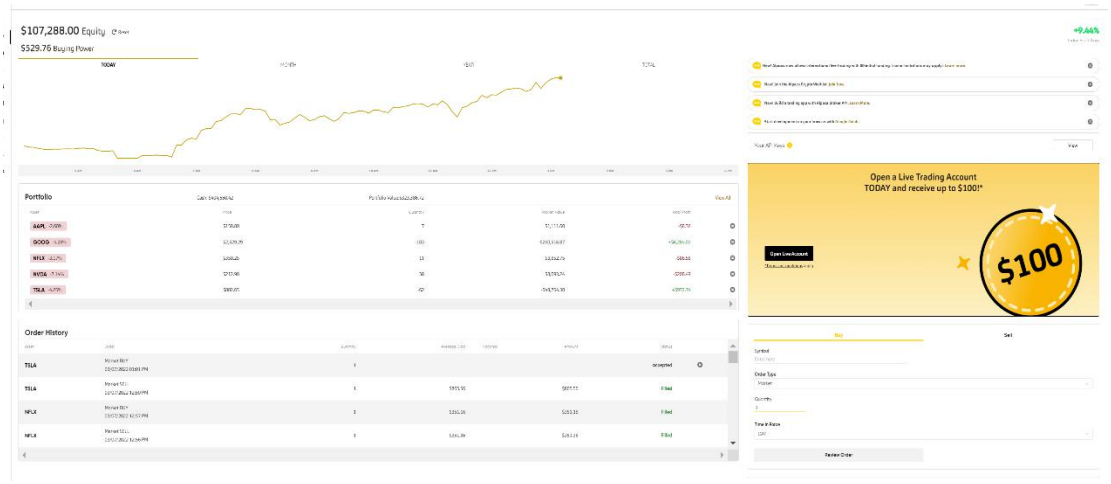
```
class PortfolioManagementSystem(TradingSystem):

    def __init__(self,filename,stockname):
        super().__init__(AlpacaPaperSocket(), stockname, 86400, 1, 'AI_PM')
        self.AI = PortfolioManagementModel(filename,stockname)

    def place_buy_order(self):
        self.api.submit_order(
                        symbol=stockname,
                        qty=1,
                        side='buy',
                        type='market',
                        time_in_force='day',
                    )

    def place_sell_order(self):
        self.api.submit_order(
                        symbol=stockname,
                        qty=1,
                        side='sell',
                        type='market',
                        time_in_force='day',
                    )

    def system_loop(self):
        this_weeks_close = 0
        last_weeks_close = 0
        delta = 0
        day_count = 0
        while(True):
            time.sleep(20)
            data_req = self.api.get_barset(stockname, timeframe='day', limit=1).df
            x = pd.DataFrame(
                data=[[
                    data_req[stockname]['close'][0]]], columns='Close'.split()
            )
            day_count += 1
            if(day_count == 5):
                day_count = 0
                last_weeks_close = this_weeks_close
                this_weeks_close = x['Close']
                delta = this_weeks_close - last_weeks_close
                if np.around(self.AI.network.predict([delta])) <= -.5:
                    self.place_sell_order()
                elif np.around(self.AI.network.predict([delta])) >= .5:
                    self.place_buy_order()
```

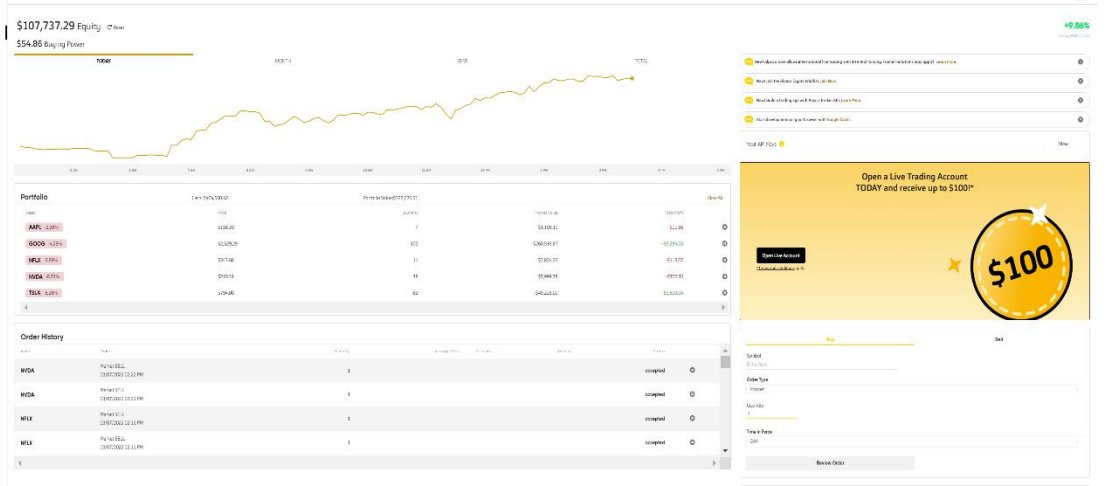
Let's talk about system_loop. system_loop initializes variables for close this week, close last week, current delta, and days. The infinite loop (the thread of the concurrent system) is responsible for collecting data once a day and determining if we have reached the weekly split. After reaching the weekly split, the variable is updated and we consult our AI whether to buy or sell.

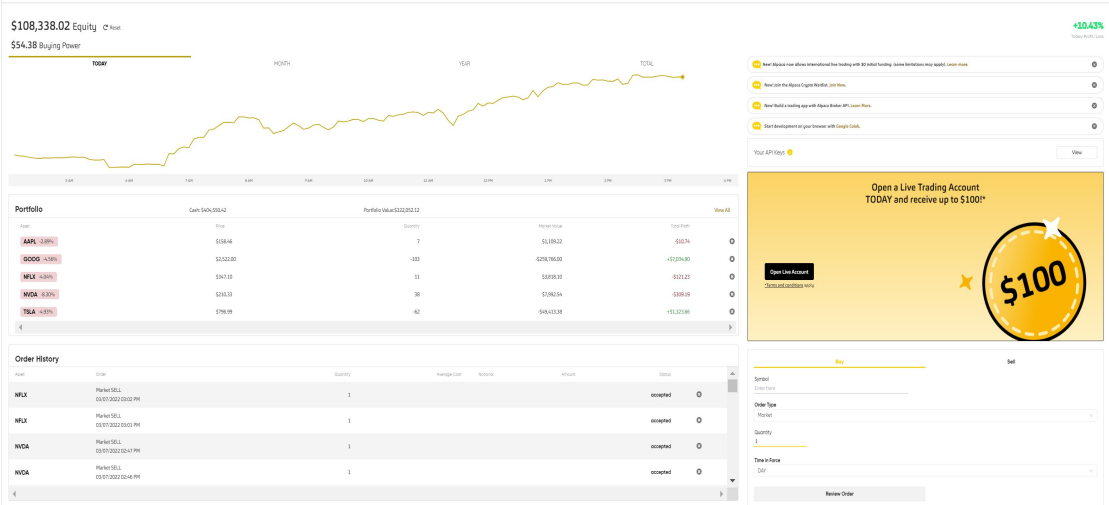
Paper trading

The result of paper trading is as follows:




$107,288.00 Equity
$529.76 Buying Power
+9.44%
TODAY
MONTH
YEAR
TOTAL
Portfolio
Order History
Open a Live Trading Account
TODAY and receive up to $100!*
$100
Open Live Account
Buy
Sell
Review Order


$107,737.29 Equity
$54.86 Buying Power
+9.86%
TODAY
MONTH
YEAR
TOTAL
Portfolio
Order History
Open a Live Trading Account
TODAY and receive up to $100!*
$100
Open Live Account
Buy
Sell
Review Order


$108,338.02 Equity
$54.38 Buying Power
+10.43%
TODAY
MONTH
YEAR
TOTAL
Portfolio
Order History
Open a Live Trading Account
TODAY and receive up to $100!*
$100
Open Live Account
Buy
Sell
Review Order

Alpaca

Paper Trading

- Paper Overview
- Paper Positions
- Paper Orders
- Paper Account
- OAuth Apps
- Documentation
- Status Page
- Community Slack
- Community Forum

Closed Orders

Symbol	Qty	Filled Qty	Side	Type	Time in Force	Limit Price	Stop Price	Filled Avg Price	Notional	Amount	Status	Submitted at	Filled at	Canceled at	Replaced at
TSLA	1	1	sell	market	day	-	-	$805.56	-	$805.56	filled	2022-03-07 12:59:26	2022-03-07 12:59:26		
NFLX	1	1	buy	market	day	-	-	$351.16	-	$351.16	filled	2022-03-07 12:57:45	2022-03-07 12:57:45		
NFLX	1	1	sell	market	day	-	-	$351.36	-	$351.36	filled	2022-03-07 12:56:04	2022-03-07 12:56:04		
AAPL	1	1	buy	market	day	-	-	$159.86	-	$159.86	filled	2022-03-07 12:47:53	2022-03-07 12:47:54		
AAPL	1	1	buy	market	day	-	-	$159.89	-	$159.89	filled	2022-03-07 12:47:48	2022-03-07 12:47:48		
AAPL	1	1	buy	market	day	-	-	$159.83	-	$159.83	filled	2022-03-07 12:47:45	2022-03-07 12:47:45		
AAPL	1	1	buy	market	day	-	-	$159.94	-	$159.94	filled	2022-03-07 12:46:12	2022-03-07 12:46:12		
AAPL	1	1	sell	market	day	-	-	$159.90	-	$159.90	filled	2022-03-07 12:46:07	2022-03-07 12:46:07		
AAPL	1	1	sell	market	day	-	-	$159.89	-	$159.89	filled	2022-03-07 12:46:04	2022-03-07 12:46:04		
AAPL	1	1	buy	market	day	-	-	$160.25	-	$160.25	filled	2022-03-07 12:44:31	2022-03-07 12:44:31		
AAPL	1	1	buy	market	day	-	-	$160.20	-	$160.20	filled	2022-03-07 12:44:26	2022-03-07 12:44:26		
NVDA	1	1	sell	market	day	-	-	$214.51	-	$214.51	filled	2022-03-07 12:42:50	2022-03-07 12:42:50		
NVDA	1	1	sell	market	day	-	-	$214.53	-	$214.53	filled	2022-03-07 12:42:44	2022-03-07 12:42:44		
NVDA	1	1	buy	market	day	-	-	$214.33	-	$214.33	filled	2022-03-07 12:41:09	2022-03-07 12:41:10		
NVDA	1	1	sell	market	day	-	-	$214.32	-	$214.32	filled	2022-03-07 12:41:03	2022-03-07 12:41:04		
NVDA	1	1	buy	market	day	-	-	$214.59	-	$214.59	filled	2022-03-07 12:39:28	2022-03-07 12:39:28		
TSLA	1	1	buy	market	day	-	-	$809.56	-	$809.56	filled	2022-03-07 12:37:47	2022-03-07 12:37:47		
TSLA	1	1	buy	market	day	-	-	$809.48	-	$809.48	filled	2022-03-07 12:36:06	2022-03-07 12:36:06		
TSLA	1	1	sell	market	day	-	-	$811.72	-	$811.72	filled	2022-03-07 12:34:25	2022-03-07 12:34:25		

Alpaca

Paper Trading

- Paper Overview
- Paper Positions
- Paper Orders
- Paper Account
- OAuth Apps
- Documentation
- Status Page
- Community Slack
- Community Forum

Search

Positions

All

- Cash +$404,550.42
- Stocks +$322,147.03

Stocks

- GOOG -$259,869.00
- TSLA -$49,352.00
- NVDA +$7,999.76

Liquidate Selected Positions

Stock	Shares	Price	Avg Entry	Cost Basis	Market Value	Today's P/L (%)	Today's P/L ($)	Total P/L (%)	Total P/L ($)
AAPL	7	$158.31	$159.99	$1,119.96	$1,108.17	-1.05%	-$11.79	-1.05%	-$11.79
GOOG	-103	$2,523.00	$2,590.30	-$266,800.90	-$259,869.00	+2.86%	+$7,664.70	+2.60%	+$6,931.90
NFLX	11	$347.10	$358.12	$3,939.33	$3,818.10	-3.08%	-$121.23	-3.08%	-$121.23
NVDA	38	$210.52	$218.47	$8,301.73	$7,999.76	-3.64%	-$301.97	-3.64%	-$301.97
TSLA	-62	$796.00	$818.34	-$50,737.04	-$49,352.00	+2.76%	+$1,398.71	+2.73%	+$1,385.04

Alpaca

Paper Trading

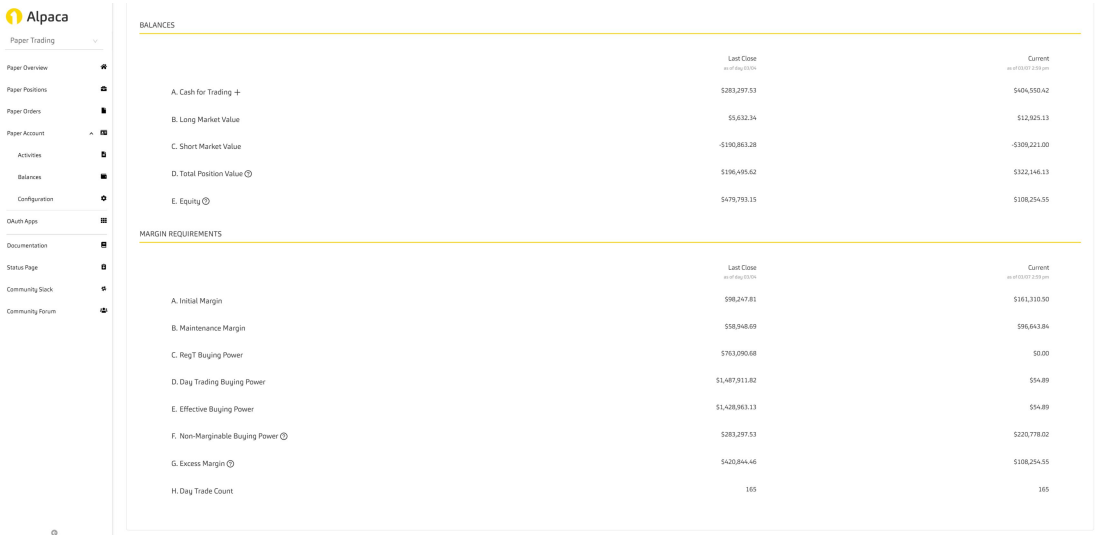
- Paper Overview
- Paper Positions
- Paper Orders
- Paper Account
  - Activities
  - Balances
  - Configuration
- OAuth Apps
- Documentation
- Status Page
- Community Slack
- Community Forum

Account Activities

Transfers | Dividends | Stock Trades | Crypto Trades | All

Select date

Activity	Date	Amount
Sell_short 1 TSLA	3/7/2022, 12:59:26 PM	+$805.56
Buy 1 NFLX	3/7/2022, 12:57:45 PM	-$351.16
Sell 1 NFLX	3/7/2022, 12:56:04 PM	+$351.36
Buy 1 AAPL	3/7/2022, 12:47:54 PM	-$159.86
Buy 1 AAPL	3/7/2022, 12:47:48 PM	-$159.89
Buy 1 AAPL	3/7/2022, 12:47:45 PM	-$159.83
Buy 1 AAPL	3/7/2022, 12:46:12 PM	-$159.94
Sell 1 AAPL	3/7/2022, 12:46:07 PM	+$159.9
Sell 1 AAPL	3/7/2022, 12:46:04 PM	+$159.89
Buy 1 AAPL	3/7/2022, 12:44:31 PM	-$160.25
Buy 1 AAPL	3/7/2022, 12:44:26 PM	-$160.2
Sell 1 NVDA	3/7/2022, 12:42:50 PM	+$214.51
Sell 1 NVDA	3/7/2022, 12:42:44 PM	+$214.53
Buy 1 NVDA	3/7/2022, 12:41:10 PM	-$214.33
Sell 1 NVDA	3/7/2022, 12:41:04 PM	+$214.32



As of March 7, the final total asset is $108338 and the ratio of Profit/Loss is 10.43%.It can be seen that the model works well.

Conclusion and contribution of each team member

We discussed how to connect to a brokerage, specifically this example of Alpaca. We then created the TradingSystem class itself and its inherited fields and implemented the class in a system dedicated to portfolio management. Afterward, we built an AI model to make trading decisions and discussed the lack of mathematical understanding behind the scenes. Finally, we deploy the model into the implemented system, giving our AI the ability to buy, sell, and hold.

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.