NATIONAL TECHNICAL UNIVERSITY OF ATHENS

MSc DATA SCIENCE AND MACHINE LEARNING

PROGRAMMING TOOLS AND TECHNOLOGIES FOR DATA SCIENCE

Investing in the stock market
(The time traveler way)

Pavlos Kalfantis
MSc Student
pavloskalfantis@mail.ntua.gr
Student ID: 03400134

February 2022

1 Introduction

Imagine having the power to travel back in time. The possibilities would be endless! As part of the graduate course 'Programming Tools and Technologies for Data Science', we were lucky enough to be given the power, but for a specific reason. Try to make as much money as possible on New York's stock exchanges! In this hypothetical scenario, we are given $1 on 1/1/1960 and we are asked to find the best moves (i.e. buying and selling stocks of companies listed on NYSE and NASDAQ), such that the return is a large as possible. The caveat? Each day, you are only allowed to buy one more stock of a specific company than the amount of stocks you had of that company the previous day! There are some additional constraints, such as the amount of stocks traded is capped at 10% of the number of stocks traded on every day. However, even after these constraints, the amount of (hypothetical) money that can be made is pretty large! This very fun and interesting data analysis and algorithmic problem was done using the `Python` programming language and on the subsequent sections, the methodology that was used is presented and analyzed.

2 The Dataset

We are given a fairly big dataset of more than 7000 stocks, containing prices and volumes for days in the past. While doing exploratory data analysis and trying different methodologies to obtain a high return on our 1$ investment, several anomalies are observed. That is why some data cleaning was done before producing the final sequences. Firstly, only stocks with non negative values were kept, and rows that the Volume of stocks traded that day is zero are deleted. Secondly, there are several stocks that have a typo on one row of the dataset. More specifically, some stocks have an erroneous row where the Low price of a given day is very small (very close to zero), when the rest of the values do not have the same problem. These rows are removed given a threshold. The threshold is that if the lowest price of the stock is more than 1000% smaller than the second lowest, that row will be removed. In addition, the same thing is done when the maximum value of the High daily price is more than 1000% bigger than the next one. Finally, in order to decrease the size of our dataset, only stocks that their biggest overall price is observed later in time than the smallest. These steps are shown on the code below.

```
#Step 1

# get data file names and keep valid files
#i.e. not empty or without zero and negative price
# data are saved on a list in the form of (stock name, dataframe)

path =r'archive\Stocks'
filenames = glob.glob(path + "/*.txt")
valid_dfs = []
performances = []
corrected_stocks = []
```

```
for filename in filenames:
    try:
        stock = filename.split('\\')[-1].split('.')[0]
        df = pd.read_csv(filename, index_col=None, header=0)
        #Keep files with non zero and negative prices
        if min(df.Low)>0 and min(df.High)>0and min(df.Open)>0 and min(df.Close)>0:
            df['stock'] = stock
            df['Date']= pd.to_datetime(df['Date'])
            df.set_index('Date',inplace=True)

            #delete zero volume days
            df = df[df.Volume!=0]

            #Delete problematic rows with small Low value
            check1 = (df.sort_values(by='Low').Low[1]-df.sort_values\
                      (by='Low').Low[0])/df.sort_values(by='Low').Low[0]
            if check1>10:
                df = df[df.Low!=min(df.Low)] #Delete problematic row
                corrected_stocks.append(stock)

            #Delete problematic rows with large High value
            check2 = (df.sort_values(by='High').High[-1]-df.sort_values\
                      (by='High').High[-2])/df.sort_values(by='High').High[-2]
            if check2>10:
                df = df[df.High!=max(df.High)] #Delete problematic row
                corrected_stocks.append(stock)

            #Find max and minimum price and corresponding date over the years
            max_price = max(df.High)
            min_price = min(df.Low)
            max_date = df[df.High==max_price].iloc[[-1]].index
            min_date = df[df.Low==min_price].iloc[[0]].index

            #Keep stocks that maximum overall price happened after minimum
            if max_date >  min_date:
                max_pct = (max_price-min_price)/min_price
                max_abs = (max_price-min_price)
                performances.append((stock,max_pct,max_abs))
                valid_dfs.append((stock,df))

    except:
        pass
```

3 The Methodology

The methodology to obtain a sequence of actions on stocks of the dataset, with the end goal of having a large profit given the constraints is based on selecting some 'good' stocks, which have demonstrated good performance over the years. The overall performance is measured as the percentage that the stock has risen to its highest value from its lowest value over the years. This information is stored on the `pandas DataFrame 'best_performances_df'`. Then, for a certain number of these stocks, we are exploring good moves for several windows, i.e. moves that have a high percentage return on our investment for a specific period of time. For this purpose, while iterating through the selected stocks, we are using the function `pd.series.pct_change` of the `pandas` library in `python`, which calculates the percentage change of values of a specific column. This

column is the column with the high prices of each stock on each day. All these moves are stored on the `good_moves DataFrame`.

```
#Step 2

#Select some good trades (buy and sell dates) for each of the good stocks
#Also create the concatenation of dataframes of good stocks

good_moves_li = []
good_stocks_df_li = []

for (name,df) in valid_dfs:
    if name in good_stock_names:
        #Create dataframe with all good stocks to plot performance
        good_stocks_df_li.append(df)

        #Check the percentage change for multiple periods
        for period in range(1,len(df)//2,2):
            df['change_high']=df.High.pct_change(periods=period)
            for sell_date in df.sort_values(by=['change_high'],ascending=False)\
            .iloc[0:10].index:

                idx = df.index.get_loc(sell_date)
                buy_date = df.iloc[idx-period].name
                low_price = df.iloc[idx-period].Low

                vol_buy_date = df.iloc[idx-period].Volume
                high_price = df.iloc[idx].High
                vol_sell_date = df.iloc[idx].Volume

                good_moves_li.append((
                name,buy_date,vol_buy_date,low_price,'buy'))
                good_moves_li.append((
                name,sell_date,vol_sell_date,high_price,'sell'))

#Concatenated dataframe of all used stocks to look up
#closing prices on each day and calcuate portfolio value

good_stocks_df= pd.concat(good_stocks_df_li)

#Create dataframe of the good moves to iterate through and perform the functions
good_moves = pd.DataFrame(good_moves_li,columns=[
'stock','date','volume','price','action'])

#Sort by date and always buy before sell on a given day
good_moves.sort_values(by=['date','action'],inplace=True)

# Delete duplicates (buy or sell the same stock on the same day)
good_moves.drop_duplicates(inplace=True)
good_moves = good_moves.drop_duplicates(subset=['stock', 'date'],keep='first')
```

Final step is to iterate through the good moves and perform two actions. Either use the function `buy_low` that tries to buy the specific stock on that day at its lowest value (given the constraints discussed on the introduction), or use the function `sell_high` that tries to sell the stock given the constraints. The methodology used is that we buy or sell the maximum number allowed on each day that the analysis has indicated is a good day to perform the action. Our portfolio is saved on the `portfolio dictionary`, with our available cash balance and stocks as keys. The last thing being done on this iteration is to save the portfolio value of each time step, in order to plot the valuation

diagram of our sequence. This is being done by looking the closing price of each stock on the portfolio, multiplying with the number of stocks and adding the cash balance.

```
#Step 3
#Iterate through the good moves and buy and sell the stocks.
#Also Keeping the values of cash balance and portfolio value of each day to plot

portfolio = {'cash':1}
transactions = []
time_values = []

for index,row in good_moves.iterrows():
    if row.action=='buy':
        buy_low(portfolio,transactions,row.stock,row.date,row.price,row.volume)
    else:
        sell_high(portfolio,transactions,row.stock,row.date,row.price,row.volume)

    #Get closing value of portfolio for certain dates
    portf_value = portfolio_closing_value(portfolio,str(row.date))
    time_values.append((row.date,portfolio['cash'],portf_value))

#Functions used through the iteration

#'Buy Low' Function to update portfolio and append a transaction
def buy_low(portfolio,transactions,stock,date,price,vol):
    if stock in portfolio.keys():
        num = min(portfolio[stock]+1,portfolio['cash']//price,vol//10)
        portfolio[stock]+=num
    else:
        num = min(1,portfolio['cash']//price,vol//10)
        portfolio[stock]=num
    if num>0:
        portfolio['cash']-=price*num
        transactions.append((date,'buy-low',stock,num))

#'Sell High' Function to update portfolio and append a transaction
def sell_high(portfolio,transactions,stock,date,price,vol):
    if stock in portfolio.keys():
        num = min(portfolio[stock],vol//10)
        portfolio[stock]-=num
        portfolio['cash']+=price*num
        if num>0:
            transactions.append((date,'sell-high',stock,num))

#Function to calculate the closing price of a portfolio on a given day
def portfolio_closing_value(portfolio,date):
    value = portfolio['cash']
    for stock in portfolio.keys():
        if stock in good_stocks_df.loc[date].stock.values:
            stock_closing_price = good_stocks_df[good_stocks_df.stock==stock]\
            .loc[date].Close
            value += stock_closing_price*portfolio[stock]
    return value
```

4 The Results

In order to produce the two sequences, 10 and 100 stocks are kept respectively. This changes the search space (and subsequently the size of the `good_moves` DataFrame) and produces a different number of moves (455 and 2858 respectively). The profit for the small sequence, as shown on the `small.txt` was $5m and the profit for the large sequence, as shown on the `large.txt` was $831m. The last part of the analysis was to plot the portfolio value and the cash balance through time. The two plotes are shown on figures 1 and 2.

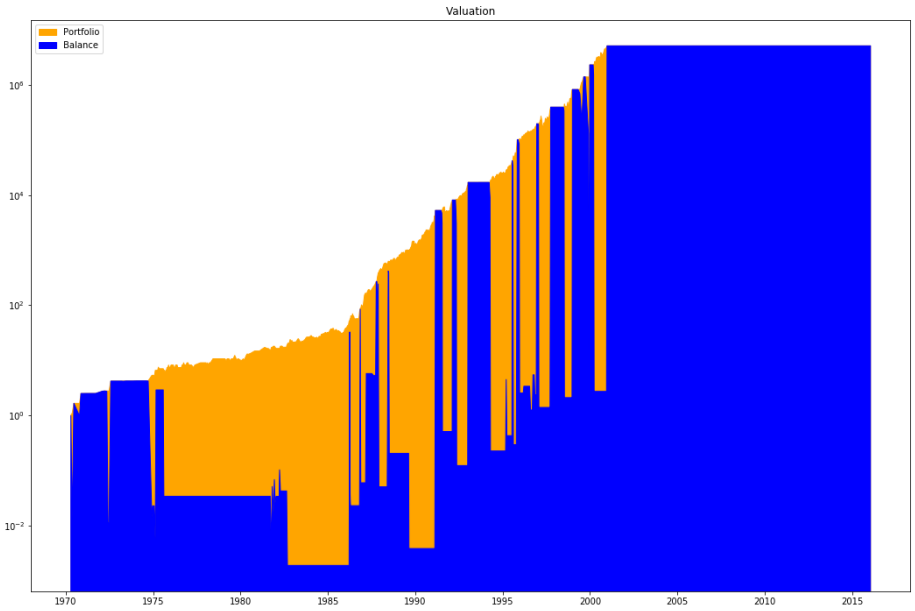

Figure 1: The valuation diagram of our portfolio with 455 transactions

```
#Plot valuation diagram

time_series = pd.DataFrame(time_values,columns=['date','balance','portfolio'])
x = time_series.date
y = time_series.balance
z = time_series.portfolio
plt.rcParams["figure.figsize"] = (18,12)
plt.fill_between(x, z,label='Portfolio',color='orange')
plt.fill_between(x, y,label='Balance',color='b')
plt.legend(loc='upper left')
plt.title('Valuation')
plt.yscale('log')
plt.savefig('figs/plot_small')
```

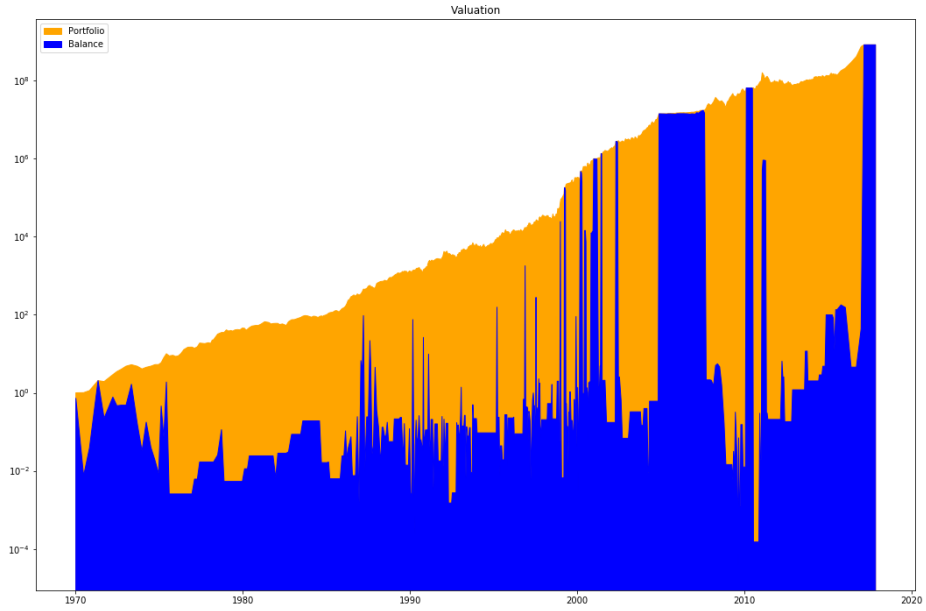

Figure 2: The valuation diagram of our portfolio with 2858 transactions