Report on Project 2: Developing Optimized Trading Schedules for a Mutual Fund

by

Md Hishamur Rahman

Student ID: 1400077

Umail: hisham.rahman@utah.edu

University of Utah

10th October, 2022

Contents

Executive Summary

ME EN 5184 and 6184 Financial markets uses operations research for stock trading. A new mutual fund relying on a limited number of stocks is established in a stock trading company, which requires an optimized schedule for buying and selling stocks. This report developed a mathematical model based on linear programming that can be solved optimally using open-source software to produce the required buying and selling schedules for the stocks in the mutual fund. Experiment based on the historical year-long stock price data of the selected stocks revealed that the model developed in this report can yield $18.6 billion in a year from a $10 million starting investment.

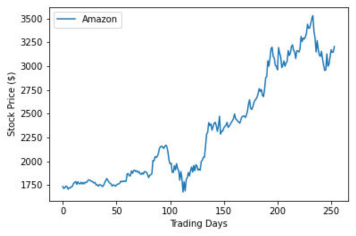
Introduction

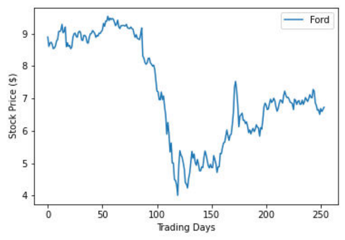
A new mutual fund is established with a limited number of stocks, i.e., Amazon, Ford, Facebook, Tesla, Walmart, Kroger, Comcast, which require an optimized schedule for determining when to buy and sell the stocks in order to maximize the worth at the end of the year. The mutual fund will buy and sell stocks at the end of the stock trading day based on the estimated closing price of stocks. Historical stock prices for a year are taken for demonstrating the feasibility of the proposed mutual fund. Investing in the Dow Jones industrial average (DJIA) for that year resulted in a loss of 2% on people's investment. However, the optimized trading methodology using linear programming is expected to yield better returns for the proposed mutual fund.

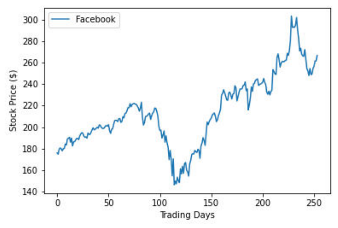
The abovementioned problem is addressed by modeling the stock buying and selling process mathematically using a linear programming approach. The model considered various constraints based on buying and selling limits and balance of stocks and cash flow through the year. The objective of the mathematical model is to maximize the worth at the end of year from buying and selling optimum number of stocks. The resulting mathematical model is solved optimally using Pulp, a Python package for solving linear programs. Experiments based on the historical price of the stocks shows that the proposed mathematical model can yield \$1.5 billion per month with a \$10 million starting investment.

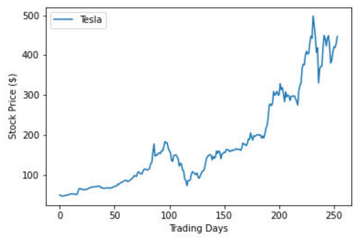
Data and Assumptions

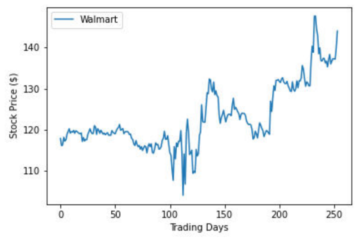
The data for this project includes historical stock price data spanning 254 trading days for Amazon, Ford, Facebook, Tesla, Walmart, Kroger, and Comcast. The year-long stock prices are not shown in the report for conciseness; however, the stock price trend for each company is shown below to provide an overall scenario. From the price trend it is evident that except Ford and Comcast, other companies have increasing price trend. Our mathematical model should be able to buy more shares of Amazon, Facebook, and Tesla between 100th and 150th trading days due to lowest prices during that period.

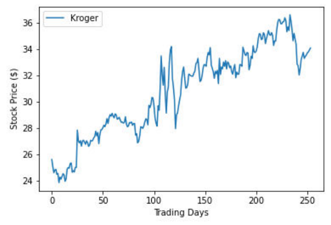


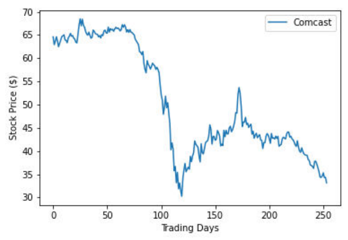












It is noteworthy to mention that any cash that is held for a trading day is assumed to earn 0.008% interest per trading day including weekends and holidays. For simplicity of the model, a \$10 million starting investment is assumed for the mutual fund. The trading fees are ignored in this report since the mutual fund is associated with a major trading company.

Mathematical Model

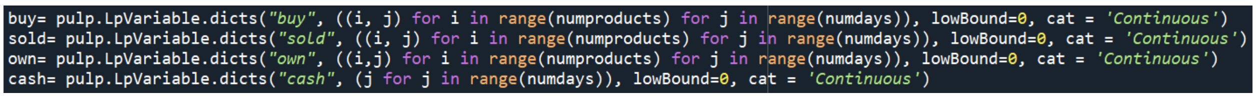
The mathematical model for this project is developed based on linear programming approach. The model includes a set of decision variables that needs to be solved, inequality constraints to meet the limits of buying and selling stocks, equality constraints to keep track of the owned stock balance and cash flow, and an objective function that involves maximization of the worth at the end of the year. They are described below with equation along with the Python code snippets.

Decision Variables

The decision variables for this model are as follows:

Number of stock i bought on day j → Bij
Number of stock i sold on day j → Sij
Number of stock i owned on day j → Oij
Cash at the end of day j → Cij

The above decision variables are coded in Pulp as follows:

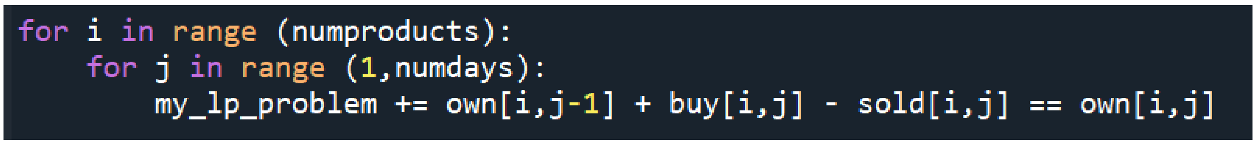


```
buy= pulp.LpVariable.dicts("buy", ((i, j) for i in range(numproducts) for j in range(numdays)), lowBound=0, cat = 'Continuous')
sold= pulp.LpVariable.dicts("sold", ((i, j) for i in range(numproducts) for j in range(numdays)), lowBound=0, cat = 'Continuous')
own= pulp.LpVariable.dicts("own", ((i,j) for i in range(numproducts) for j in range(numdays)), lowBound=0, cat = 'Continuous')
cash= pulp.LpVariable.dicts("cash", (j for j in range(numdays)), lowBound=0, cat = 'Continuous')
```

Constraints

For a stock i, the number of stocks owned in a trading day j would be the sum of the number of stocks owned from the previous trading day j-1 and the net number of stocks owned from buying and selling stocks in trading day j. This can be expressed as follows:

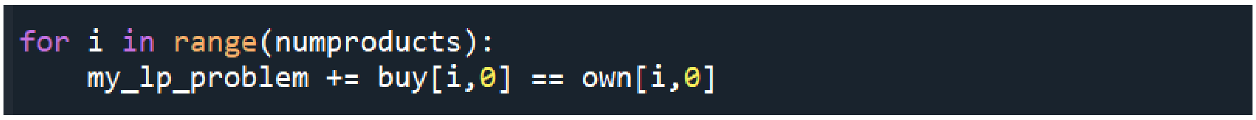
$$Oi,j-1+Bij-Sij=Oij \quad \forall i \geq 0, \forall j \geq 1 \qquad (1)$$



```
for i in range (numproducts):
    for j in range (1,numdays):
        my_lp_problem += own[i,j-1] + buy[i,j] - sold[i,j] == own[i,j]
```

Since the amount of stocks owned and sold on the first trading day (j=0) are zero, the amount of stocks owned on the first trading day would be amount of stocks that are bought on that day, which can be expressed as follows:

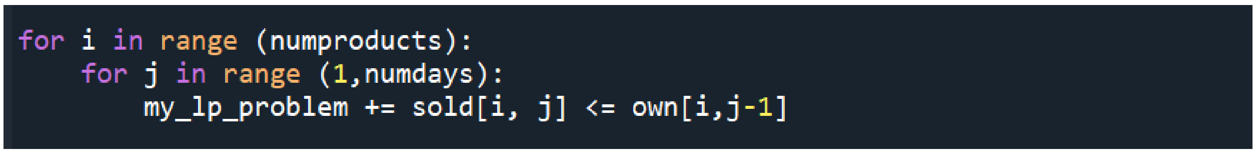
$$Bi,0=Oi,0 \quad \forall i \qquad (2)$$



```
for i in range(numproducts):
    my_lp_problem += buy[i,0] == own[i,0]
```

For any stock i, the number of stocks sold in a day j cannot be greater than the stocks owned from the previous day j-1. This can be expressed as follows:

$$Sij \leq Oi,j-1 \quad \forall i \geq 0, \forall j \geq 1 \qquad (3)$$

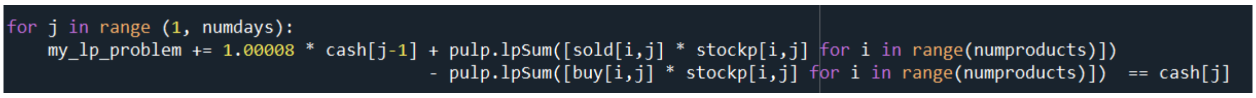


```
for i in range (numproducts):
    for j in range (1,numdays):
        my_lp_problem += sold[i, j] <= own[i,j-1]
```

The cash at the end of a trading day j would be the sum of the cash including any interest from the previous day j-1 and the net money earned after buying and selling stocks, which can be expressed as:

$$1.00008 \times Cj-1+\sum_i Sij \times Pij-\sum_i Bij \times Pij=Cj \quad \forall j \geq 1 \qquad (4)$$

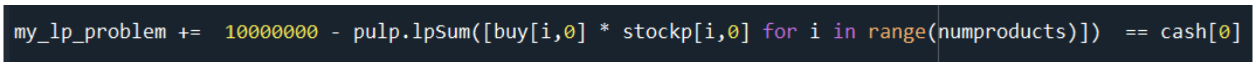
where Pij is the price of the stock i on day j.



```
for j in range (1, numdays):
    my_lp_problem += 1.00008 * cash[j-1] + pulp.lpSum([sold[i,j] * stockp[i,j] for i in range(numproducts)])
                                        - pulp.lpSum([buy[i,j] * stockp[i,j] for i in range(numproducts)])  == cash[j]
```

Since the above constraint is applied from 2nd day of the week (j≥1), we need specify the constraint for the first day separately (j=0). The first day will not have any cash earned from selling stocks. Therefore, the cash at the end of first trading day can be found by subtracting the money spent in buying stocks from the starting fund ($10 million). Therefore, the resulting constraint for the first day can be formulated and coded as follows:

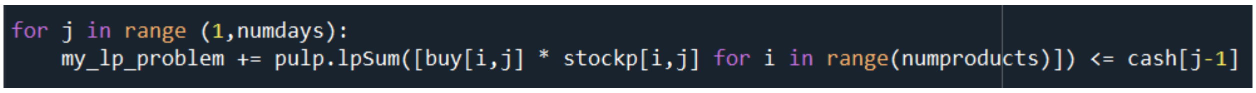
$$10000000 - \sum_i B_{i,0} \times P_{i,0} = C_0 \qquad (5)$$



```
my_lp_problem +=  10000000 - pulp.lpSum([buy[i,0] * stockp[i,0] for i in range(numproducts)])  == cash[0]
```

The money spent on a trading day j to buy stocks cannot exceed the cash owned from the previous day j-1, which can be expressed as follows:

$$\sum_i Bi,j \times Pi,j \leq Cj - 1 \quad \forall j \geq 1 \qquad (6)$$

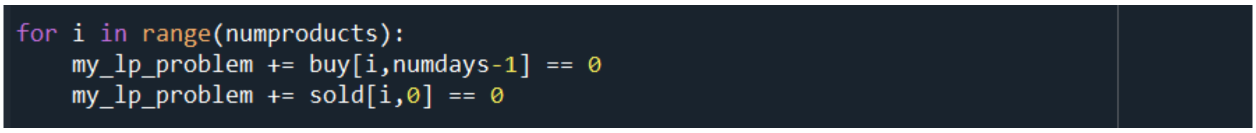


```
for j in range (1,numdays):
    my_lp_problem += pulp.lpSum([buy[i,j] * stockp[i,j] for i in range(numproducts)]) <= cash[j-1]
```

Finally, no stocks can be sold on the first trading day and no stocks should be bought on the final trading day (j=254). This can be expressed as follows:

$$B_{i,254} = 0 \quad \forall i \qquad (7)$$

$$S_{i,0} = 0 \quad \forall i \qquad (8)$$

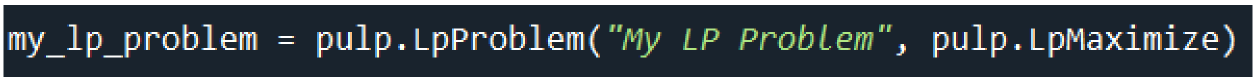


```
for i in range(numproducts):
    my_lp_problem += buy[i,numdays-1] == 0
    my_lp_problem += sold[i,0] == 0
```

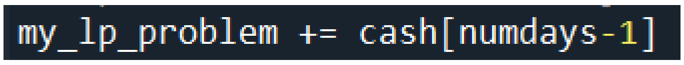
Objective Function

The objective of the mathematical model is to maximize the cash yielded at the end of the last trading day (j=254), which can be expressed as follows:

$$\max C_{254}$$



```
my_lp_problem = pulp.LpProblem("My LP Problem", pulp.LpMaximize)
```



```
my_lp_problem += cash[numdays-1]
```

The Python code for the mathematical model is provided in the Appendix.

Analysis

The developed mathematical model is optimized and solved using Pulp package in Python. The optimum solution revealed that it is possible to yield $18.6 billion in a year using the provided mathematical model. This indicates an average 3% growth of the money each trading day. With a 50% accuracy of the stock prices, it would still yield $439 million (considering a growth of 1.5% each day, 10,000,000*1.015^254=438,909,248).

The solution also provides the optimum buying, selling, and owning schedule for each stock on a trading day, which are tabulated as follows:

Bought:

Month	Day	Amazon	Ford	Facebook	Tesla	Walmart	Kroger	Comcast
10	1	0	0	0	0	0	0	0
10	2	0	0	57273.8	0	0	0	0
10	3	0	0	0	0	0	0	0

10	4	0	0	0	221980	0	0	0
10	7	0	0	0	17.2885	0	0	0
10	8	0	0	0	0.00137	0	0	0
10	9	0	6.1E-07	0	0	0	0	0
10	10	0	0	0	8.6E-12	0	0	0
10	11	0	0	0	0	0	0	0
10	14	0	0	0	0	0	0	0
10	15	0	0	0	0	0	0	0
10	16	0	6.1E-07	0	0	0	0	0
10	17	0	1265945	0	0	0	0	0
10	18	0	0	0	0	0	38.5389	0
10	21	0	0	0	0	0	487791	0
10	22	0	0	0	0	0	0	0
10	23	0	0	0	238014	0	0	0
10	24	0	0	0	16.1819	0	0	0
10	25	4.4E-05	0	0	0	0	0	0
10	28	0	0	0	0	0	0	0
10	29	8862.62	0	0	0	0	0	0
10	30	0	146.344	0	0	0	0	0
10	31	0	1836482	0	0	0	0	0
11	1	0	0	0	0	0	0	0
11	4	0	0	0	0	0	653417	0
11	5	0	0	0	20.59	0	0	0
11	6	0	0	0	278410	0	0	0
11	7	0	0	0	21.678	0	0	0
11	8	0	0	0	0.00173	0	0	0
11	11	0	0	4.9E-08	0	0	0	0
11	12	0	0	0	0	161335	0	0
11	13	0	0	0	0	0	0	0
11	14	0	2220683	0	0	0	0	0
11	15	0	0	0	0	0	0	0
11	18	0	0	0	283961	0	0	0
11	19	0	0	0	0	0	0	24.72242
11	20	0	0	0	0	0	0	316465
11	21	0	187.535	0	0	0	0	0
11	22	11978.9	0	0	0	0	0	0
11	25	0.94312	0	0	0	0	0	0
11	26	0	0	8.48345	0	0	0	0
11	27	0	0	0	0	0	0.00496	0
11	29	0	0	0	330168	0	0	0
12	2	0	0	0	0	0	0	0
12	3	0	0	0	0	0	0	343517.3
12	4	0	0	0	0	0	0	0
12	5	0	0	0	338515	0	0	0
12	6	0	0	0	26.6362	0	0	0
12	9	0	0	0	0.00211	0	0	0
12	10	0	0	0	1.6E-07	0	0	0
12	11	0	1E-10	0	0	0	0	0
12	12	0	0	0	0	0	0	0
12	13	0	0	0	339760	0	0	0

12	16	0	0	0	0	0	0	0
12	17	0	0	0	342036	0	0	0
12	18	0	0	0	26.3774	0	0	0
12	19	0	0	0	0.00205	0	0	0
12	20	0	0	0	1.6E-07	0	0	0
12	23	0	0	0	0	0	0	0
12	24	16029.4	0	0	0	0	0	0
12	26	0	0	0	0	0	0	0
12	27	0	0	0	0	0	1041637	0
12	30	0	0	0	28.8956	0	0	0
12	31	0	0	0	361795	0	0	0
1	2	0	0	0	28.1411	0	0	0
1	3	0	0	0	0.00219	0	0	0
1	6	0	0	0	1.7E-07	0	0	0
1	7	0	0	0	0	0	0	0
1	8	0	0	0	0	0	0	0
1	9	0	0	0	0	0	0	0
1	10	0	0	0	372470	0	0	0
1	13	0	0	0	27.1457	0	0	0
1	14	0	0	0.00104	0	0	0	0
1	15	21522.2	0	0	0	0	0	0
1	16	0	0	0	0	0	0	0
1	17	-3E-12	0	0	395892	0	0	0
1	21	0	0	0	29.5472	0	0	0
1	22	0	0	0	0.00227	0	0	0
1	23	0	0	0	0	0	0	0
1	24	0	0	0	0	0	0	0
1	27	0	0	0	406014	0	0	0
1	28	0	0	0	31.9724	0	0	0
1	29	0	0	0	0.0025	0	0	0
1	30	0	0	0	1.8E-07	0	0	0
1	31	0	0	0	1.4E-11	0	0	0
2	3	0	0	0	0	0	0	0
2	4	0	0	0	0	0	0	0
2	5	0	0	0	490251	0	0	0
2	6	2.8109	0	0	0	0	0	0
2	7	0	0	0	490834	0	0	0
2	10	0	0	0	0	0	208.772	0
2	11	0	0	0	0	0	0	0
2	12	0	0	0	495448	0	0	0
2	13	0	0	0	0	0	0	0
2	14	0	0	0	497945	0	0	0
2	18	0	0	0	37.1268	0	0	0
2	19	0	0	0	0	0	0.01726	0
2	20	0	0	0	0	0	3068223	0
2	21	0	0	0	0	0	0	0
2	24	0	0	0	0	0	0	0
2	25	47148.9	0	0	0	0	0	0
2	26	0	0	0	0	0	0	0
2	27	0	0	0	0	0	0	0

2	28	0	0	0	698741	0	0	0
3	2	0	0	0	0	0	0	0
3	3	0	0	0	0	0	3542164	0
3	4	0	0	0	0	0	268.545	0
3	5	0	0	0	0	0.00574	0	0
3	6	0	0	0	0	0	0	0
3	9	0	0	0	975121	0	0	0
3	10	0	0	0	0	0	0	0
3	11	0	0	0	0	0	0	0
3	12	0	0	814880	0	0	0	0
3	13	0	0	0	0	0	0	0
3	16	0	0	0	0	1299811	0	0
3	17	0	0	0	0	0	0	0
3	18	0	0	0	2145875	0	0	0
3	19	0	0	0	0	0	0	0
3	20	0	0	0	0	0	0	0
3	23	0	4.6E+07	0	0	0	0	0
3	24	0	0	0	0	0	0	430.7224
3	25	0	0	0.00752	0	0	0	0
3	26	0	0	0	0	0	0	0
3	27	0	0	1573859	0	0	0	0
3	30	0	0	0	196.575	0	0	0
3	31	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	7271798
4	2	0	0	0	231.074	0	0	0
4	3	0	0	0	2943869	0	0	0
4	6	0	4991.03	0	0	0	0	0
4	7	0	0.38402	0	0	0	0	0
4	8	0	0	0	1.3E-06	0	0	0
4	9	1.2E-13	0	0	219.082	0	0	0
4	13	0	0	0	0.01543	0	0	0
4	14	0	0	0	1.1E-06	0	0	0
4	15	0	0	0	0	0	0	0
4	16	0	0	0	0	0	0	11411145
4	17	14.4765	0	0	0	0	0	0
4	20	0	0	0	0	0	0	0
4	21	0	0	2778230	0	0	0	0
4	22	0	0	0	0	0	0	936.6535
4	23	0	0	0	3588395	0	0	0
4	24	0	0	0	549.296	0	0	0
4	27	0	0	0	0	0	0	0
4	28	0	0	3134513	0	0	0	0
4	29	0	0	236.195	0	0	0	0
4	30	0	0	0	0	0	0	0
5	1	0	0	0	4575048	0	0	0
5	4	0	0	0	337.216	0	0	0
5	5	0	0	0	0.02673	0	0	0
5	6	0	0	0	0	0	0	0
5	7	0	1.5E+08	0	0	0	0	0
5	8	24.0752	0	0	0	0	0	0

5	11	0	0	0	0	6230521	0	0
5	12	0	0	0	0	0	0	0
5	13	0	1.6E+08	0	0	0	0	0
5	14	0	12619	0	0	0	0	0
5	15	0	1.00746	0	0	0	0	0
5	18	0	0	1.9E-06	0	0	0	0
5	19	0	0	4001371	0	0	0	0
5	20	0	12645.8	0	0	0	0	0
5	21	0	0	0.024	0	0	0	0
5	22	0	7.9E-05	0	0	0	0	0
5	26	0	0	0	0	0	3E+07	0
5	27	0	0	0	0	0	0	0
5	28	0	0	0	6188457	0	0	0
5	29	0	0	0	477.77	0	0	0
6	1	0	1.08739	0	0	0	0	0
6	2	0	1.9E+08	0	0	0	0	0
6	3	0	14367.1	0	0	0	0	0
6	4	0	1.08289	0	0	0	0	0
6	5	0	0	0	3.2E-06	0	0	0
6	8	1.8E-11	0	0	0	0	0	0
6	9	0	0	0	7541875	0	0	0
6	10	0	0	0	0	0	0	0
6	11	0	2.5E+08	0	0	0	0	0
6	12	0	0	0	661.309	0	0	0
6	15	633392	0	0	0	0	0	0
6	16	99.9589	0	0	-7E-10	0	0	0
6	17	0.00792	0	0	0	0	0	0
6	18	6.3E-07	0	0	0	0	0	0
6	19	0	0	0	0	0	0	0
6	22	0	0	0	0	0	0	0
6	23	0	0	0	0	0	0	0
6	24	0	0	0	9112928	0	0	0
6	25	0	0	0	0	0	0	0
6	26	0	0	0	9362812	0	0	0
6	29	0	0	0	712.21	0	0	0
6	30	0	0	0	0.05326	0	0	0
7	1	0	0	0	4.1E-06	0	0	0
7	2	0	0	0	3E-10	0	0	0
7	6	0	0	0	0	5E-14	0	0
7	7	0	0	0	0	0	0	0
7	8	0	0	0	9527915	0	0	0
7	9	0	0	0	746.707	0	0	0
7	10	0	0	0	0	0	0	0
7	13	0	4.9E+08	0	0	0	0	0
7	14	0	37027.5	0	0	0	0	0
7	15	0	2.79519	0	0	0	0	0
7	16	0	0	0	5E-06	0	0	0
7	17	0	0	0	1.1E+07	0	0	0
7	20	0	0	0	0	0	0	6396.788
7	21	0	5.5E+08	0	0	0	0	0

7	22	0	83618.1	0	0	0	0	0
7	23	0	0	0	0	0	0	0
7	24	0	0	0	1.3E+07	0	0	0
7	27	0	0	0	0	0	0	0
7	28	0	0	1.8E+07	0	0	0	0
7	29	0	0	1420.66	0	0	0	0
7	30	0	0	0.11307	0	0	0	0
7	31	0	0	0	7.4E-06	0	0	0
8	3	0	0	0	0	0	0	1.1E+08
8	4	0	0	0	0	0	0	0
8	5	0	0	1.9E+07	0	0	0	0
8	6	0	0	0	0	0	0	0
8	7	0	7.3E+08	0	0	0	0	0
8	10	0	0	0	0	0	0	0
8	11	0	0	0	1.9E+07	0	0	0
8	12	0	0	0	1330.83	0	0	0
8	13	0	0	0	0.10212	0	0	0
8	14	0	0	0	8E-06	0	0	0
8	17	0	0	0	0	0	0	0
8	18	0	0	0	0	0	0	0
8	19	0	0	0	1.9E+07	0	0	0
8	20	0	0	0	0	0	0	0
8	21	0	1.1E+09	0	0	0	0	0
8	24	0	0	2231.39	0	0	0	0
8	25	0	0	0	2E+07	0	0	0
8	26	0	0	0	0	4855.97	0	0
8	27	0	0	0	0	-7E-13	0	0
8	28	0	0	0	2E+07	0	0	0
8	31	0	0	0	0	5058.35	0	0
9	1	0	0	0	0	0	2.8E+08	0
9	2	0	0	0	0	-7E-13	0	0
9	3	0	0	0	2.5E+07	0	0	0
9	4	0	0	0	0	0	0	0
9	8	0	0	0	3.2E+07	0	0	0
9	9	0	0	0	2294.3	0	0	0
9	10	0	-6E-08	0	0.18104	0	0	0
9	11	0	0	0	1.4E-05	0	0	0
9	14	0	0	0	0	0	0	0
9	15	0	0	0	0	0	0	0
9	16	0	0	0	0	0	0	0
9	17	0	0	0	3.4E+07	0	0	0
9	18	387.415	0	0	0	0	0	0
9	21	5047290	0	0	0	0	0	0
9	22	0	0	0	0	0	0	0
9	23	0	0	0	4.2E+07	0	0	0
9	24	0	0	0	3258.53	0	0	0
9	25	0	0	0	0.24817	0	0	0
9	28	0	0	0	0	0	0	0
9	29	0	0	0	4.2E+07	0	0	0
9	30	0	0	0	3261.96	0	0	0

10	1	0	0	0	0	0	0	0

Sold:

Month	Day	Amazon	Ford	Facebook	Tesla	Walmart	Kroger	Comcast
10	1	0	0	0	0	0	0	0
10	2	0	0	0	0	0	0	0
10	3	0	0	57273.8	0	0	0	0
10	4	0	0	0	0	0	0	0
10	7	0	0	0	0	0	0	0
10	8	0	0	0	0	0	0	0
10	9	0	0	0	0	0	0	0
10	10	0	0	0	0	0	0	0
10	11	0	0	0	0	0	0	0
10	14	0	0	0	0	0	0	0
10	15	0	6.1E-07	0	0	0	0	0
10	16	0	0	0	221997	0	0	0
10	17	0	0	0	0	0	0	0
10	18	0	1265945	0	0	0	0	0
10	21	0	0	0	0	0	0	0
10	22	0	0	0	0	0	487829	0
10	23	0	0	0	0	0	0	0
10	24	0	0	0	0	0	0	0
10	25	0	0	0	238030	0	0	0
10	28	4.4E-05	0	0	0	0	0	0
10	29	0	0	0	0	0	0	0
10	30	8862.62	0	0	0	0	0	0
10	31	0	0	0	0	0	0	0
11	1	0	1836629	0	0	0	0	0
11	4	0	0	0	0	0	0	0
11	5	0	0	0	0	0	653417	0
11	6	0	0	0	0	0	0	0
11	7	0	0	0	0	0	0	0
11	8	0	0	0	0	0	0	0
11	11	0	0	0	278452	0	0	0
11	12	0	0	4.9E-08	0	0	0	0
11	13	0	0	0	0	161335	0	0
11	14	0	0	0	0	0	0	0
11	15	0	2220683	0	0	0	0	0
11	18	0	0	0	0	0	0	0
11	19	0	0	0	283961	0	0	0
11	20	0	0	0	0	0	0	0
11	21	0	0	0	0	0	0	316489.7
11	22	0	0	0	0	0	0	0
11	25	0	187.535	0	0	0	0	0
11	26	0	0	0	0	0	0	0
11	27	11979.8	0	8.48345	0	0	0	0
11	29	0	0	0	0	0	0	0
12	2	0	0	0	330168	0	0.00496	0
12	3	0	0	0	0	0	0	0
12	4	0	0	0	0	0	0	343517.3

12	5	0	0	0	0	0	0	0
12	6	0	0	0	0	0	0	0
12	9	0	0	0	0	0	0	0
12	10	0	0	0	0	0	0	0
12	11	0	0	0	0	0	0	0
12	12	0	1E-10	0	338542	0	0	0
12	13	0	0	0	0	0	0	0
12	16	0	0	0	339760	0	0	0
12	17	0	0	0	0	0	0	0
12	18	0	0	0	0	0	0	0
12	19	0	0	0	0	0	3.7E-11	0
12	20	0	0	0	0	0	-4E-11	0
12	23	0	0	0	342063	0	0	0
12	24	0	0	0	0	0	0	0
12	26	16029.4	0	0	0	0	0	0
12	27	0	0	0	0	0	0	0
12	30	0	0	0	0	0	1041637	0
12	31	0	0	0	0	0	0	0
1	2	0	0	0	0	0	0	0
1	3	0	0	0	0	0	4.3E-11	0
1	6	0	0	0	0	0	-4E-11	0
1	7	0	0	0	0	0	0	0
1	8	0	0	0	361852	0	0	0
1	9	0	0	0	4.7E-11	0	0	0
1	10	0	0	0	0	0	0	0
1	13	0	0	0	0	0	0	0
1	14	0	0	0	372497	0	0	0
1	15	0	0	0	0	0	0	0
1	16	21522.2	0	0.00104	0	0	0	0
1	17	0	0	0	0	0	0	0
1	21	0	0	0	0	0	0	0
1	22	0	0	0	0	0	0	0
1	23	0	0	0	395921	0	0	0
1	24	0	0	0	0	0	0	0
1	27	0	0	0	0	0	0	0
1	28	0	0	0	0	0	0	0
1	29	0	0	0	0	0	0	0
1	30	0	0	0	0	0	0	0
1	31	0	0	0	0	0	0	0
2	3	0	0	0	0	0	0	0
2	4	0	0	0	406046	0	0	0
2	5	0	0	0	0	0	0	0
2	6	0	0	0	490251	0	0	0
2	7	0	0	0	0	0	0	0
2	10	0	0	0	0	0	0	0
2	11	2.8109	0	0	490834	0	208.772	0
2	12	0	0	0	0	0	0	0
2	13	0	0	0	495448	0	0	0
2	14	0	0	0	0	0	0	0
2	18	0	0	0	0	0	0	0

2	19	0	0	0	497982	0	0	0
2	20	0	0	0	0	0	0	0
2	21	0	0	0	0	0	3068223	0
2	24	0	0	0	0	0	0	0
2	25	0	0	0	0	0	0	0
2	26	47148.9	0	0	0	0	0	0
2	27	0	0	0	0	0	0	0
2	28	0	0	0	0	0	0	0
3	2	0	0	0	698741	0	0	0
3	3	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0
3	5	0	0	0	0	0	3542433	0
3	6	0	0	0	0	0.00574	0	0
3	9	0	0	0	0	0	0	0
3	10	0	0	0	975121	0	0	0
3	11	0	0	0	0	0	0	0
3	12	0	0	0	0	0	0	0
3	13	0	0	814880	0	0	0	0
3	16	0	0	0	0	0	0	0
3	17	0	0	0	0	1299811	0	0
3	18	0	0	0	0	0	0	0
3	19	0	0	0	2145875	0	0	0
3	20	0	0	0	0	0	0	0
3	23	0	0	0	0	0	0	0
3	24	0	0	0	0	0	0	0
3	25	0	4.6E+07	0	0	0	0	0
3	26	0	0	0.00752	0	0	0	430.7224
3	27	0	0	0	0	0	0	0
3	30	0	0	0	0	0	0	0
3	31	0	0	1573859	196.575	0	0	0
4	1	0	0	3.3E-10	0	0	0	0
4	2	0	0	0	0	0	0	7271798
4	3	0	0	0	0	0	3.9E-12	0
4	6	0	0	0	0	0	-4E-12	0
4	7	2.4E-13	0	0	0	0	0	0
4	8	0	4991.42	0	0	0	0	0
4	9	0	7.9E-13	0	0	0	0	0
4	13	0	-2E-12	0	0	0	-1E-12	0
4	14	-6E-12	1.7E-12	0	-2E-12	0	1.1E-12	0
4	15	5.8E-12	0	0	2944320	0	0	0
4	16	0	0	0	0	0	0	0
4	17	0	0	0	3.5E-10	0	0	11411145
4	20	14.4765	0	0	0	0	0	0
4	21	0	0	0	0	0	0	0
4	22	0	0	2778230	0	0	0	0
4	23	0	0	0	0	0	0	936.6535
4	24	0	0	0	0	0	0	0
4	27	0	0	0	3588944	0	0	0
4	28	0	0	0	0	0	0	0
4	29	0	0	0	0	0	0	0

4	30	0	0	3134749	0	0	0	0
5	1	0	0	0	0	0	0	0
5	4	0	0	0	0	0	0	0
5	5	0	0	0	0	0	0	0
5	6	0	0	0	4575385	0	0	0
5	7	0	0	0	0	0	0	0
5	8	0	1.5E+08	0	0	0	0	0
5	11	24.0752	0	0	0	0	0	0
5	12	0	0	0	0	6230521	0	0
5	13	0	0	0	0	0	0	0
5	14	0	0	0	0	0	0	0
5	15	0	0	0	0	0	0	0
5	18	0	1.6E+08	0	0	0	0	0
5	19	0	0	0	0	0	0	0
5	20	0	0	0	0	0	0	0
5	21	0	0	0	0	0	0	0
5	22	0	0	4001371	0	0	0	0
5	26	0	0	6.3E-10	0	0	0	0
5	27	0	12645.8	0	0	0	3E+07	0
5	28	0	0	0	0	0	0	0
5	29	0	0	0	0	0	0	0
6	1	0	0	0	6188934	0	0	0
6	2	0	0	0	0	0	0	0
6	3	0	0	0	0	0	0	0
6	4	0	0	0	0	0	0	0
6	5	0	0	0	0	0	0	0
6	8	1.8E-11	1.9E+08	0	3.2E-06	0	0	0
6	9	0	0	0	0	0	0	0
6	10	0	0	0	7541875	0	0	0
6	11	0	0	0	0	0	0	0
6	12	0	2.5E+08	0	0	0	0	0
6	15	0	0	0	661.309	0	0	0
6	16	0	0	0	0	0	0	0
6	17	0	0	0	0	0	0	0
6	18	0	0	0	-7E-10	0	0	0
6	19	0	0	0	0	0	0	0
6	22	0	0	0	0	0	0	0
6	23	633492	0	0	0	0	0	0
6	24	0	0	0	0	0	0	0
6	25	0	0	0	9112928	0	0	0
6	26	0	0	0	0	0	0	0
6	29	0	0	0	0	0	0	0
6	30	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0
7	2	0	0	0	0	0	0	0
7	6	0	0	0	0	0	0	0
7	7	0	0	0	9363525	5E-14	0	0
7	8	0	0	0	0	0	0	0
7	9	0	0	0	0	0	0	0
7	10	0	0	0	9528661	0	0	0

7	13	0	0	0	0	0	0	0
7	14	0	0	0	0	0	0	0
7	15	0	0	0	0	0	0	0
7	16	0	4.9E+08	0	0	0	0	0
7	17	0	0	0	0	0	0	0
7	20	0	0	0	1.1E+07	0	0	0
7	21	0	0	0	0	0	0	6396.788
7	22	0	0	0	0	0	0	0
7	23	0	5.5E+08	0	0	0	0	0
7	24	0	0	0	0	0	0	0
7	27	0	0	0	1.3E+07	0	0	0
7	28	0	0	0	0	0	0	0
7	29	0	0	0	0	0	0	0
7	30	0	0	0	0	0	0	0
7	31	0	0	1.8E+07	0	0	0	0
8	3	0	0	0	0	0	0	0
8	4	0	0	0	7.4E-06	0	0	1.1E+08
8	5	0	0	0	0	0	0	0
8	6	0	0	1.9E+07	0	0	0	0
8	7	0	0	0	0	0	0	0
8	10	0	7.3E+08	0	0	0	0	0
8	11	0	0	0	0	0	0	0
8	12	0	0	0	0	0	0	0
8	13	0	0	0	0	0	0	0
8	14	0	0	0	0	0	0	0
8	17	0	0	0	0	0	0	0
8	18	0	0	0	1.9E+07	0	0	0
8	19	0	0	0	0	0	0	0
8	20	0	0	0	1.9E+07	0	0	0
8	21	0	0	0	0	0	0	0
8	24	0	1.1E+09	0	0	0	0	0
8	25	0	0	0	0	0	0	0
8	26	0	0	2231.39	0	0	0	0
8	27	0	0	0	2E+07	4855.97	0	0
8	28	0	0	0	0	0	0	0
8	31	0	0	0	2E+07	0	0	0
9	1	0	0	0	0	0	0	0
9	2	0	0	0	0	5058.35	2.8E+08	0
9	3	0	0	0	0	0	0	0
9	4	0	0	0	2.5E+07	0	0	0
9	8	0	0	0	0	0	0	0
9	9	0	0	0	0	0	0	0
9	10	0	0	0	0	0	0	0
9	11	0	-6E-08	0	0	0	0	0
9	14	0	0	0	0	0	0	0
9	15	0	0	0	3.2E+07	0	0	0
9	16	0	0	0	0	0	0	0
9	17	0	0	0	0	0	0	0
9	18	0	0	0	3.4E+07	0	0	0
9	21	0	0	0	0	0	0	0

9	22	5047677	0	0	0	0	0	0
9	23	0	0	0	0	0	0	0
9	24	0	0	0	0	0	0	0
9	25	0	0	0	0	0	0	0
9	28	0	0	0	4.2E+07	0	0	0
9	29	0	0	0	0	0	0	0
9	30	0	0	0	0	0	0	0
10	1	0	0	0	4.2E+07	0	0	0

Owned:

Month	Day	Amazon	Ford	Facebook	Tesla	Walmart	Kroger	Comcast
10	1	0	0	0	0	0	0	0
10	2	0	0	57273.8	0	0	0	0
10	3	0	0	0	0	0	0	0
10	4	0	0	0	221980	0	0	0
10	7	0	0	0	221997	0	0	0
10	8	0	0	0	221997	0	0	0
10	9	0	6.1E-07	0	221997	0	0	0
10	10	0	6.1E-07	0	221997	0	0	0
10	11	0	6.1E-07	0	221997	0	0	0
10	14	0	6.1E-07	0	221997	0	0	0
10	15	0	0	0	221997	0	0	0
10	16	0	6.1E-07	0	0	0	0	0
10	17	0	1265945	0	0	0	0	0
10	18	0	0	0	0	0	38.5389	0
10	21	0	0	0	0	0	487829	0
10	22	0	0	0	0	0	0	0
10	23	0	0	0	238014	0	0	0
10	24	0	0	0	238030	0	0	0
10	25	4.4E-05	0	0	0	0	0	0
10	28	0	0	0	0	0	0	0
10	29	8862.62	0	0	0	0	0	0
10	30	0	146.344	0	0	0	0	0
10	31	0	1836629	0	0	0	0	0
11	1	0	0	0	0	0	0	0
11	4	0	0	0	0	0	653417	0
11	5	0	0	0	20.59	0	0	0
11	6	0	0	0	278430	0	0	0
11	7	0	0	0	278452	0	0	0
11	8	0	0	0	278452	0	0	0
11	11	0	0	4.9E-08	0	0	0	0
11	12	0	0	0	0	161335	0	0
11	13	0	0	0	0	0	0	0
11	14	0	2220683	0	0	0	0	0
11	15	0	0	0	0	0	0	0
11	18	0	0	0	283961	0	0	0
11	19	0	0	0	0	0	0	24.72242
11	20	0	0	0	0	0	0	316489.7
11	21	0	187.535	0	0	0	0	0

11	22	11978.9	187.535	0	0	0	0	0
11	25	11979.8	0	0	0	0	0	0
11	26	11979.8	0	8.48345	0	0	0	0
11	27	0	0	0	0	0	0.00496	0
11	29	0	0	0	330168	0	0.00496	0
12	2	0	0	0	0	0	0	0
12	3	0	0	0	0	0	0	343517.3
12	4	0	0	0	0	0	0	0
12	5	0	0	0	338515	0	0	0
12	6	0	0	0	338542	0	0	0
12	9	0	0	0	338542	0	0	0
12	10	0	0	0	338542	0	0	0
12	11	0	1E-10	0	338542	0	0	0
12	12	0	0	0	0	0	0	0
12	13	0	0	0	339760	0	0	0
12	16	0	0	0	0	0	0	0
12	17	0	0	0	342036	0	0	0
12	18	0	0	0	342063	0	0	0
12	19	0	0	0	342063	0	-4E-11	0
12	20	0	0	0	342063	0	0	0
12	23	0	0	0	0	0	0	0
12	24	16029.4	0	0	0	0	0	0
12	26	0	0	0	0	0	0	0
12	27	0	0	0	0	0	1041637	0
12	30	0	0	0	28.8956	0	0	0
12	31	0	0	0	361824	0	0	0
1	2	0	0	0	361852	0	0	0
1	3	0	0	0	361852	0	-4E-11	0
1	6	0	0	0	361852	0	0	0
1	7	0	0	0	361852	0	0	0
1	8	0	0	0	4.7E-11	0	0	0
1	9	0	0	0	0	0	0	0
1	10	0	0	0	372470	0	0	0
1	13	0	0	0	372497	0	0	0
1	14	0	0	0.00104	0	0	0	0
1	15	21522.2	0	0.00104	0	0	0	0
1	16	2.8E-12	0	0	0	0	0	0
1	17	0	0	0	395892	0	0	0
1	21	0	0	0	395921	0	0	0
1	22	0	0	0	395921	0	0	0
1	23	0	0	0	0	0	0	0
1	24	0	0	0	0	0	0	0
1	27	0	0	0	406014	0	0	0
1	28	0	0	0	406046	0	0	0
1	29	0	0	0	406046	0	0	0
1	30	0	0	0	406046	0	0	0
1	31	0	0	0	406046	0	0	0
2	3	0	0	0	406046	0	0	0
2	4	0	0	0	0	0	0	0
2	5	0	0	0	490251	0	0	0

2	6	2.8109	0	0	0	0	0	0
2	7	2.8109	0	0	490834	0	0	0
2	10	2.8109	0	0	490834	0	208.772	0
2	11	0	0	0	0	0	0	0
2	12	0	0	0	495448	0	0	0
2	13	0	0	0	0	0	0	0
2	14	0	0	0	497945	0	0	0
2	18	0	0	0	497982	0	0	0
2	19	0	0	0	0	0	0.01726	0
2	20	0	0	0	0	0	3068223	0
2	21	0	0	0	0	0	0	0
2	24	0	0	0	0	0	0	0
2	25	47148.9	0	0	0	0	0	0
2	26	0	0	0	0	0	0	0
2	27	0	0	0	0	0	0	0
2	28	0	0	0	698741	0	0	0
3	2	0	0	0	0	0	0	0
3	3	0	0	0	0	0	3542164	0
3	4	0	0	0	0	0	3542433	0
3	5	0	0	0	0	0.00574	0	0
3	6	0	0	0	0	0	0	0
3	9	0	0	0	975121	0	0	0
3	10	0	0	0	0	0	0	0
3	11	0	0	0	0	0	0	0
3	12	0	0	814880	0	0	0	0
3	13	0	0	0	0	0	0	0
3	16	0	0	0	0	1299811	0	0
3	17	0	0	0	0	0	0	0
3	18	0	0	0	2145875	0	0	0
3	19	0	0	0	0	0	0	0
3	20	0	0	0	0	0	0	0
3	23	0	4.6E+07	0	0	0	0	0
3	24	0	4.6E+07	0	0	0	0	430.7224
3	25	0	0	0.00752	0	0	0	430.7224
3	26	0	0	0	0	0	0	0
3	27	0	0	1573859	0	0	0	0
3	30	0	0	1573859	196.575	0	0	0
3	31	0	0	3.3E-10	0	0	0	0
4	1	0	0	0	0	0	0	7271798
4	2	0	0	0	231.074	0	0	0
4	3	0	0	0	2944100	0	-4E-12	0
4	6	0	4991.03	0	2944100	0	0	0
4	7	-2E-13	4991.42	0	2944100	0	0	0
4	8	-2E-13	7.9E-13	0	2944100	0	0	0
4	9	-1E-13	0	0	2944320	0	0	0
4	13	-1E-13	1.7E-12	0	2944320	0	1.1E-12	0
4	14	5.4E-12	0	0	2944320	0	0	0
4	15	-3E-13	0	0	3.5E-10	0	0	0
4	16	-3E-13	0	0	3.5E-10	0	0	11411145
4	17	14.4765	0	0	0	0	0	0

4	20	0	0	0	0	0	0	0
4	21	0	0	2778230	0	0	0	0
4	22	0	0	0	0	0	0	936.6535
4	23	0	0	0	3588395	0	0	0
4	24	0	0	0	3588944	0	0	0
4	27	0	0	0	0	0	0	0
4	28	0	0	3134513	0	0	0	0
4	29	0	0	3134749	0	0	0	0
4	30	0	0	0	0	0	0	0
5	1	0	0	0	4575048	0	0	0
5	4	0	0	0	4575385	0	0	0
5	5	0	0	0	4575385	0	0	0
5	6	0	0	0	0	0	0	0
5	7	0	1.5E+08	0	0	0	0	0
5	8	24.0752	0	0	0	0	0	0
5	11	0	0	0	0	6230521	0	0
5	12	0	0	0	0	0	0	0
5	13	0	1.6E+08	0	0	0	0	0
5	14	0	1.6E+08	0	0	0	0	0
5	15	0	1.6E+08	0	0	0	0	0
5	18	0	0	1.9E-06	0	0	0	0
5	19	0	0	4001371	0	0	0	0
5	20	0	12645.8	4001371	0	0	0	0
5	21	0	12645.8	4001371	0	0	0	0
5	22	0	12645.8	6.3E-10	0	0	0	0
5	26	0	12645.8	0	0	0	3E+07	0
5	27	0	0	0	0	0	0	0
5	28	0	0	0	6188457	0	0	0
5	29	0	0	0	6188934	0	0	0
6	1	0	1.08739	0	0	0	0	0
6	2	0	1.9E+08	0	0	0	0	0
6	3	0	1.9E+08	0	0	0	0	0
6	4	0	1.9E+08	0	0	0	0	0
6	5	0	1.9E+08	0	3.2E-06	0	0	0
6	8	0	0	0	0	0	0	0
6	9	0	0	0	7541875	0	0	0
6	10	0	0	0	0	0	0	0
6	11	0	2.5E+08	0	0	0	0	0
6	12	0	0	0	661.309	0	0	0
6	15	633392	0	0	0	0	0	0
6	16	633492	0	0	-7E-10	0	0	0
6	17	633492	0	0	-7E-10	0	0	0
6	18	633492	0	0	0	0	0	0
6	19	633492	0	0	0	0	0	0
6	22	633492	0	0	0	0	0	0
6	23	0	0	0	0	0	0	0
6	24	0	0	0	9112928	0	0	0
6	25	0	0	0	0	0	0	0
6	26	0	0	0	9362812	0	0	0
6	29	0	0	0	9363525	0	0	0

6	30	0	0	0	9363525	0	0	0
7	1	0	0	0	9363525	0	0	0
7	2	0	0	0	9363525	0	0	0
7	6	0	0	0	9363525	5E-14	0	0
7	7	0	0	0	0	0	0	0
7	8	0	0	0	9527915	0	0	0
7	9	0	0	0	9528661	0	0	0
7	10	0	0	0	0	0	0	0
7	13	0	4.9E+08	0	0	0	0	0
7	14	0	4.9E+08	0	0	0	0	0
7	15	0	4.9E+08	0	0	0	0	0
7	16	0	0	0	5E-06	0	0	0
7	17	0	0	0	1.1E+07	0	0	0
7	20	0	0	0	0	0	0	6396.788
7	21	0	5.5E+08	0	0	0	0	0
7	22	0	5.5E+08	0	0	0	0	0
7	23	0	0	0	0	0	0	0
7	24	0	0	0	1.3E+07	0	0	0
7	27	0	0	0	0	0	0	0
7	28	0	0	1.8E+07	0	0	0	0
7	29	0	0	1.8E+07	0	0	0	0
7	30	0	0	1.8E+07	0	0	0	0
7	31	0	0	0	7.4E-06	0	0	0
8	3	0	0	0	7.4E-06	0	0	1.1E+08
8	4	0	0	0	0	0	0	0
8	5	0	0	1.9E+07	0	0	0	0
8	6	0	0	0	0	0	0	0
8	7	0	7.3E+08	0	0	0	0	0
8	10	0	0	0	0	0	0	0
8	11	0	0	0	1.9E+07	0	0	0
8	12	0	0	0	1.9E+07	0	0	0
8	13	0	0	0	1.9E+07	0	0	0
8	14	0	0	0	1.9E+07	0	0	0
8	17	0	0	0	1.9E+07	0	0	0
8	18	0	0	0	0	0	0	0
8	19	0	0	0	1.9E+07	0	0	0
8	20	0	0	0	0	0	0	0
8	21	0	1.1E+09	0	0	0	0	0
8	24	0	0	2231.39	0	0	0	0
8	25	0	0	2231.39	2E+07	0	0	0
8	26	0	0	0	2E+07	4855.97	0	0
8	27	0	0	0	0	0	0	0
8	28	0	0	0	2E+07	0	0	0
8	31	0	0	0	0	5058.35	0	0
9	1	0	0	0	0	5058.35	2.8E+08	0
9	2	0	0	0	0	0	0	0
9	3	0	0	0	2.5E+07	0	0	0
9	4	0	0	0	0	0	0	0
9	8	0	0	0	3.2E+07	0	0	0
9	9	0	0	0	3.2E+07	0	0	0

9	10	0	-6E-08	0	3.2E+07	0	0	0
9	11	0	0	0	3.2E+07	0	0	0
9	14	0	0	0	3.2E+07	0	0	0
9	15	0	0	0	0	0	0	0
9	16	0	0	0	0	0	0	0
9	17	0	0	0	3.4E+07	0	0	0
9	18	387.415	0	0	0	0	0	0
9	21	5047677	0	0	0	0	0	0
9	22	0	0	0	0	0	0	0
9	23	0	0	0	4.2E+07	0	0	0
9	24	0	0	0	4.2E+07	0	0	0
9	25	0	0	0	4.2E+07	0	0	0
9	28	0	0	0	0	0	0	0
9	29	0	0	0	4.2E+07	0	0	0
9	30	0	0	0	4.2E+07	0	0	0
10	1	0	0	0	0	0	0	0

Cash:

	Money Made	18647455000
Month	Day	Cash ($)
10	1	10000000
10	2	800
10	3	10274569
10	4	821.96549
10	7	0.065757239
10	8	5.26058E-06
10	9	4.20846E-10
10	10	0
10	11	0
10	14	0
10	15	5.574E-06
10	16	11532762
10	17	922.62098
10	18	11760633
10	21	940.8506
10	22	12123496
10	23	969.87966
10	24	0.077590373
10	25	15620987
10	28	15622237
10	29	1249.7789
10	30	15775383
10	31	1262.0307
11	1	16328891
11	4	1306.3113
11	5	18184596
11	6	1454.7676
11	7	0.11638141
11	8	9.31051E-06

11	11	19218184
11	12	1537.4547
11	13	19519804
11	14	1561.5843
11	15	19876675
11	18	1590.134
11	19	20417903
11	20	1633.4323
11	21	20911754
11	22	1672.9403
11	25	1687.9513
11	26	0.13503611
11	27	21787125
11	29	1742.97
12	2	22114414
12	3	1769.1531
12	4	22367043
12	5	1789.3635
12	6	0.14314908
12	9	1.14519E-05
12	10	9.16154E-10
12	11	0
12	12	24353335
12	13	1948.2668
12	16	25925656
12	17	2074.0525
12	18	0.1659242
12	19	1.3275E-05
12	20	0
12	23	28679892
12	24	2294.3914
12	26	29957484
12	27	2396.5987
12	30	30269974
12	31	2421.5979
1	2	0.19372783
1	3	1.54995E-05
1	6	0
1	7	0
1	8	35616412
1	9	35619262
1	10	2849.5409
1	13	0.22796327
1	14	40074684
1	15	3205.9747
1	16	40420522
1	17	3233.6417
1	21	0.25869134
1	22	2.06953E-05
1	23	45309206

1	24	45312831
1	27	3625.0265
1	28	0.29000212
1	29	2.32002E-05
1	30	1.85601E-09
1	31	0
2	3	0
2	4	72037503
2	5	5763.0002
2	6	73435702
2	7	5874.8561
2	10	0.46998849
2	11	76030461
2	12	6082.4368
2	13	79674128
2	14	6373.9302
2	18	0.50991442
2	19	91371683
2	20	7309.7346
2	21	93005146
2	24	93012586
2	25	7441.0069
2	26	93342997
2	27	93350464
2	28	7468.0371
3	2	103927090
3	3	8314.1672
3	4	0.66513338
3	5	118565220
3	6	118574710
3	9	9485.9764
3	10	125864440
3	11	125874510
3	12	10069.961
3	13	138767830
3	16	11101.427
3	17	155026570
3	18	12402.126
3	19	183544770
3	20	183559450
3	23	14684.756
3	24	1.1747805
3	25	246729540
3	26	246765360
3	27	19741.229
3	30	1.5792983
3	31	262540310
4	1	21003.225
4	2	282617340
4	3	22609.387

4	6	1.808751
4	7	0.000144701
4	8	25106.836
4	9	2.0085469
4	13	0.000160684
4	14	0
4	15	429770550
4	16	34381.644
4	17	474449110
4	20	474521710
4	21	37961.737
4	22	506415790
4	23	79664.397
4	24	6.3731517
4	27	573333780
4	28	45866.702
4	29	3.6693362
4	30	641714530
5	1	51337.162
5	4	4.106973
5	5	0.000328558
5	6	716120980
5	7	57289.678
5	8	770528530
5	11	119639.53
5	12	771333540
5	13	61706.683
5	14	4.9365347
5	15	0.000394923
5	18	867817250
5	19	69425.38
5	20	5.5540304
5	21	0.000444322
5	22	939961960
5	26	75196.957
5	27	997344060
5	28	79787.525
5	29	6.383002
6	1	1111656400
6	2	88932.509
6	3	7.1146007
6	4	0.000569168
6	5	4.55334E-08
6	8	1418883200
6	9	113510.65
6	10	1546273300
6	11	123701.87
6	12	1629514800
6	15	261419.45
6	16	20.913556

6	17	0.001673085
6	18	0
6	19	0
6	22	0
6	23	1751231400
6	24	140098.51
6	25	1797173100
6	26	143773.85
6	29	11.501908
6	30	0.000920153
7	1	7.36122E-08
7	2	5.88898E-12
7	6	0
7	7	2602797500
7	8	208223.8
7	9	16.657904
7	10	2943689200
7	13	235495.14
7	14	18.839611
7	15	0.001507169
7	16	3332549100
7	17	266603.93
7	20	3648209200
7	21	571947.72
7	22	45.755818
7	23	3812634700
7	24	305010.77
7	27	4142812200
7	28	331424.98
7	29	26.513998
7	30	0.00212112
7	31	4567139500
8	3	365371.16
8	4	4699384700
8	5	375950.77
8	6	5004602000
8	7	400368.16
8	10	5172795200
8	11	413823.61
8	12	33.105889
8	13	0.002648471
8	14	0
8	17	0
8	18	7102948000
8	19	568235.84
8	20	7569728500
8	21	605578.28
8	24	7933439200
8	25	634675.14
8	26	678193.99

8	27	8779395400
8	28	702351.63
8	31	9882869000
9	1	790629.52
9	2	10219418000
9	3	817553.45
9	4	10504471000
9	8	840357.7
9	9	67.228616
9	10	0.005378289
9	11	0
9	14	0
9	15	14308567000
9	16	14309712000
9	17	1144777
9	18	14942350000
9	21	1195388
9	22	15795327000
9	23	1263626.2
9	24	101.09009
9	25	0.008087207
9	28	17492676000
9	29	1399414.1
9	30	111.95313
10	1	18647455000

Some sanity checks are done to check that the model works correctly by considering few rows from the solution. They are as follows:

1. Nothing sold on trading day 1 and nothing bought on last trading day (satisfies constraint 7 and 8).
2. Amount of stocks owned on day 1 is equal to the amount of stocks bought on that day, i.e., both are zero (satisfies constraint 2).
3. Amount of stocks owned on day 15 for Kroger is equal to sum of amount of stocks owned from previous day and after buying and selling stocks on that day, i.e., 39 + 487791 + 0 = 487829 (satisfies constraint 1).
4. Cash on first day is the starting investment minus the money spent in buying stocks, i.e., 10000000 – 0 = 10000000 (satisfies constraint 5)
5. Cash on trading day 2 is (1.00008*10000000) – (57273.8*174.6) + 0 = 800, which satisfies constraint 4. This also satisfies constraint 6, since 57273.8*174.6 < 10000000.
6. Amount of Facebook stocks sold on trading day 3 (57273.8) is equal to the amount of Facebook stocks from trading day 2 (57273.8), which satisfies constraint 3.

Maximum stocks are bought during the trading days 100-200, which was expected from the stock price data visualization.

Conclusion

This report provided a mathematical model for optimizing the buying and selling schedule of stocks in a mutual fund. The mathematical model used a linear programming approach that was coded and solved using the Pulp package in Python. The analysis based on historical stock price data for a year showed that the mathematical model is capable of producing $18.6 billion per year, providing better return than investing in DJIA.

In this report, we assumed a starting investment of $10 million. However, increasing the mutual fund would yield more money at the end of year. A sensitivity analysis can be provided based on the model if requested. Furthermore, more analysis can be done on a dataset over couple of years to assess the profitability trend. Additionally, the model improvement can be tested by imposing additional constraints on buying and selling stocks.

Appendix

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct  5 20:25:42 2022

@author: u1400077
"""

import pulp
import openpyxl
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Loading stock price data
stock = pd.read_excel('Stocks.xlsx')

# Processing data
numproducts = len(stock.columns.tolist()[2:])
numdays = len(stock)
names = stock.columns.tolist()[2:]
stockp = np.transpose(stock[names].values)
monthnames=stock['Month'].tolist()
daynames=stock['Day'].tolist()

# Visualizing stock prices
stocks=stock[names]
for i in stocks.columns.tolist():
    stocks[[i]].plot(xlabel='Trading Days', ylabel='Stock Price
($)').get_figure().savefig('X:/Documents/fig_'+i)


#Then instantiate a problem class, we'll name it "My LP problem" and
we're looking for an optimal maximum so we use LpMaximize

my_lp_problem = pulp.LpProblem("My LP Problem", pulp.LpMaximize)

# decision variables
buy= pulp.LpVariable.dicts("buy", ((i, j) for i in range(numproducts) for
j in range(numdays)), lowBound=0, cat = 'Continuous')
sold= pulp.LpVariable.dicts("sold", ((i, j) for i in range(numproducts)
for j in range(numdays)), lowBound=0, cat = 'Continuous')
own= pulp.LpVariable.dicts("own", ((i,j) for i in range(numproducts) for
j in range(numdays)), lowBound=0, cat = 'Continuous')
cash= pulp.LpVariable.dicts("cash", (j for j in range(numdays)),
lowBound=0, cat = 'Continuous')

# obj and constr are added using += to the my_lp_problem class.

#Pulp assumes that the the objective function is alsyws given first.
my_lp_problem += cash[numdays-1]

#constraints
for i in range (numproducts):
    for j in range (1,numdays):
        my_lp_problem += own[i,j-1] + buy[i,j] - sold[i,j] == own[i,j]

for i in range(numproducts):
    my_lp_problem += buy[i,0] == own[i,0]
```

```
for i in range (numproducts):
    for j in range (1,numdays):
        my_lp_problem += sold[i, j] <= own[i,j-1]

for j in range (1, numdays):
    my_lp_problem += 1.00008 * cash[j-1] + pulp.lpSum([sold[i,j] *
stockp[i,j] for i in range(numproducts)]) - pulp.lpSum([buy[i,j] *
stockp[i,j] for i in range(numproducts)])  == cash[j]


my_lp_problem +=  10000000 - pulp.lpSum([buy[i,0] * stockp[i,0] for i in
range(numproducts)])  == cash[0]

for j in range (1,numdays):
    my_lp_problem += pulp.lpSum([buy[i,j] * stockp[i,j] for i in
range(numproducts)]) <= cash[j-1]

for i in range(numproducts):
    my_lp_problem += buy[i,numdays-1] == 0
    my_lp_problem += sold[i,0] == 0

print (my_lp_problem)

status=my_lp_problem.solve()
if pulp.LpStatus[my_lp_problem.status] =='Infeasible':
    print ('INFEASIBLE ****************************************')

else:
    # To write out to an excel file


    wbr = openpyxl.Workbook()

    #Sheets start at 0

    wbr.create_sheet(index = 0, title = "Bought")
    wbr.create_sheet(index = 1, title = "Sold")
    wbr.create_sheet(index = 2, title = "Owned")
    wbr.create_sheet(index = 3, title = "Cash")
    sheetnames = wbr.sheetnames

    wsheet = wbr[sheetnames[0]]

    #Writing to individual cells
    # Cells start at 1, 1

    #set column widths
    letters=['A','B','C','D','E','F','G','H','I','J','K']
    widths = [8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]
    for i in range (0,9):
        wsheet.column_dimensions[letters[i]].width = widths[i]

    #I am using row to increase as I write out stuff and then columns.

    row =1

    for i in range (0, len(names)):
        wsheet.cell(row , i+4).value = names[i]
    wsheet.cell(row,1).value ='Bought'
    wsheet.cell(row , 2).value = 'Month'
wsheet.cell(row , 3).value = 'Day'

row+=1
for j in range (0,numdays):
    wsheet.cell(row, 2).value =monthnames[j]
    wsheet.cell(row, 3).value =daynames[j]
    for i in range (0,len(names)):
        wsheet.cell(row, i+4).value =buy[i,j].varValue

    row+=1

wsheet = wbr[sheetnames[1]]

#Writing to individual cells
# Cells start at 1, 1

#set column widths
letters=['A','B','C','D','E','F','G','H','I','J','K']
widths = [8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]
for i in range (0,9):
    wsheet.column_dimensions[letters[i]].width = widths[i]

#I am using row to increase as I write out stuff and then columns.

row =1

for i in range (0, len(names)):
    wsheet.cell(row , i+4).value = names[i]
wsheet.cell(row,1).value ='Sold'
wsheet.cell(row , 2).value = 'Month'
wsheet.cell(row , 3).value = 'Day'
#
row+=1
for j in range (0,numdays):
    wsheet.cell(row, 2).value =monthnames[j]
    wsheet.cell(row, 3).value =daynames[j]
    for i in range (0,len(names)):
        wsheet.cell(row, i+4).value =sold[i,j].varValue

    row+=1


wsheet = wbr[sheetnames[2]]

#Writing to individual cells
# Cells start at 1, 1

#set column widths
letters=['A','B','C','D','E','F','G','H','I','J','K']
widths = [8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]
for i in range (0,9):
    wsheet.column_dimensions[letters[i]].width = widths[i]

#I am using row to increase as I write out stuff and then columns.

row =1

for i in range (0, len(names)):
    wsheet.cell(row , i+4).value = names[i]
wsheet.cell(row,1).value ='Owned'
wsheet.cell(row , 2).value = 'Month'
wsheet.cell(row , 3).value = 'Day'

row+=1
for j in range (0,numdays):  wsheet.cell(row,
    2).value =monthnames[j] wsheet.cell(row,
    3).value =daynames[j] for i in range
    (0,len(names)):
        wsheet.cell(row, i+4).value =own[i,j].varValue row+=1

wsheet = wbr[sheetnames[3]]

#Writing to individual cells
# Cells start at 1, 1

#set column widths
letters=['A','B','C','D','E','F','G','H','I','J','K'] widths =
[8, 12, 8, 12, 8, 8, 8, 8, 8, 8, 8]
for i in range (0,9): wsheet.column_dimensions[letters[i]].width
    = widths[i]

#I am using row to increase as I write out stuff and then columns. row =1
wsheet.cell(row,1).value ='Money Made' wsheet.cell(row,2).value
=pulp.value(my_lp_problem.objective) row+=1


wsheet.cell(row, 1).value ='Cash' wsheet.cell(row ,
2).value = 'Month' wsheet.cell(row , 3).value =
'Day' wsheet.cell(row , 4).value = 'Cash ($)'
row+=1
for j in range (0,numdays):  wsheet.cell(row,
    2).value =monthnames[j] wsheet.cell(row,
    3).value =daynames[j]
    wsheet.cell(row, 4).value = cash[j].varValue row+=1

wbr.save('readinoutputexcel2.xlsx')
```