

# Decision Analytics for Business and Policy Final Project

## Real Time Arbitrage of Energy Prices with a Battery Network

**Justin Poser**

AndrewID: justinpo

**Ryan Shen**

AndrewID: ryanshen

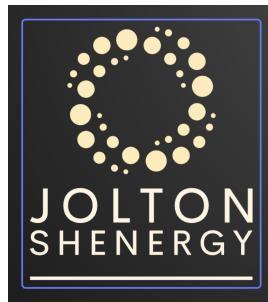
**Colton Lapp**

AndrewID: cglapp

**Date:**

January 6, 2024

**Team Contributions:** Justin spearheaded the coding, Colton focused on the mathematical formulation, and Ryan produced the problem formulation, and domain research and led the Optiguide exploration - but all members assisted each other with all aspects.



# 1 Executive Summary

Real-time temporal arbitrage is seen as a key source of revenue for battery storage systems. With real-time temporal arbitrage, charging happens at periods when electricity prices are cheapest, and energy is discharged from batteries to the grid during peak demand periods when electricity fetches high prices. The New York Independent Service Operator (NYISO) operates the state's wholesale power market, including power markets, contracts that control transmission, and related products.

For our project, we assume the role of a New York ISO contractor (Jolton Shenergy Inc.) which can adjust the number of batteries, the type of battery, and the quantity of warehouse space that should be reserved for a battery farm. Our goal is to maximize daily arbitrage profits by buying and selling power to the grid subject to a forecast of hourly energy prices. To complicate our model, we incorporate the fixed costs of monthly warehouse and battery rentals and also include nuanced battery characteristics so that our model can differentiate what the optimal mix of battery types should be. As a result, our model is not only able to plan buying/selling schedules subject to price forecasts, but it's also able to inform us what the optimal mix of batteries and warehouses is given our parameterization. For demonstrative purposes, we chose to fabricate the specs of the battery types to be highly differentiated to arrive at a solution that invests in all three types. The characteristics we model, such as size, cost, round trip efficiency (amount of electricity retained when storing energy into the battery), charge capacity, and charge rates, however, are very real. We also model the marginal cost of warehouse rental as upwards increasing to discourage infinite investment. We choose to attach the battery farm to one generator location for simplicity, which in this case is by Hudson Falls.

Our optimization problem attempts to tackle energy arbitrage in two stages. In the first stage, we optimize for profits over a month of historical data and record the optimal amount of fixed investment (warehouse and battery rentals). In the second stage, we fix our investment decisions using our historical optimization results and then optimize for daily profits given 24 hours of prices. We assume that we have access to an accurate forecast even though energy prices are notoriously difficult to predict.

We demonstrate that after using October 2023's price data to determine fixed investments, we can make enough arbitrage revenue in November's daily data to profit overall. We ultimately chose all three of our batteries because of their respective strengths. Lithium-ion batteries minimize electricity loss but are larger and more expensive than the alternatives. Lead batteries offer high storage at a low cost but have slow charge times. Meanwhile, palladium batteries offer superior cost and superior charge times but are largely inefficient.

Our optiguide formulations envision scenarios where the contractor would face a variety of supply-side constraints affecting revenue or cost. We determine that palladium is the strongest battery by far on account of its small size and low cost. Size is a major determinant of profit. In scenarios where we slightly reduce warehouse space by installing fans, our profit, which is close to 10,000 before running Optiguide simulations, does not exceed 9,000.

The selling patterns under each of the Optiguide scenarios are also similar, where we see buying occurring earlier in the day and selling during high-demand periods closer to the evening.

## 2 Analytical Model and Data

Our team solved the same energy arbitrage optimization problem two times. In the first optimization round, our goal was to understand the optimal amount of investments in long-term rentals for the next period. We utilized historical data in the first round to optimize for maximum hypothetical historical profits, given the flexibility to optimally choose fixed long-term decision variables (namely the number of warehouses and batteries to rent). That round of optimization was unrealistic because we had price data for the entire long time period, whereas we only had a daily forecast during deployment. After that first round of optimization, we acquired the optimal amount of historical investment under perfect information.

After determining the optimal historical investment amounts, we committed ourselves to a future investment, under the assumption that the historical data would provide a good approximation for the time period to come. In the time period ahead, daily price schedules are released every 24 hours, which we perform energy arbitrage on given our fixed investments. More specifically, we choose a daily schedule of purchasing and selling power from the grid given our warehouse/battery setup.

### 2.1 Data Summary

New York ISO data is separated by date and generator. It's assumed that generators are an accurate proxy for location based market prices.

There are two datasets, the day ahead market prices and the real time market prices. While the New York ISO is responsible for generating forecasts 24 hours in advance, real-time energy prices are determined in 6 minute intervals through a complicated multi-stage bid process. We use real time market data wherever we have access. The data is downloaded through a robust web scraping routine implemented in Python.

The screenshot shows a table titled "Day Ahead Market Generator LBMP". The table has two main sections: "Generator Prices" and "Marginal Cost of Losses". The "Generator Prices" section lists generators by Name and PTID, with columns for each hour from 00:00 to 11:00 EST. The "Marginal Cost of Losses" section lists the same generators with columns for each hour from 00:00 to 11:00 EST. The data includes numerical values for price and cost.

Generator Prices	... LBMP \$												... Marginal Cost of Losses											
	00:00 EST	01:00 EST	02:00 EST	03:00 EST	04:00 EST	05:00 EST	06:00 EST	07:00 EST	08:00 EST	09:00 EST	10:00 EST	11:00 EST	00:00 EST	01:00 EST	02:00 EST	03:00 EST	04:00 EST	05:00 EST	06:00 EST	07:00 EST	08:00 EST	09:00 EST	10:00 EST	11:00 EST
59TH STREET_GT_1 24138	44.39 3.08 -0.21	40.39 2.78 0.00	38.61 2.59 -0.50	38.45 2.43 -2.21	39.09 2.50 -2.30	45.97 4.33 -7.87	75.29 6.09 0.00	97.78 6.52 13.40	92.57 5.95 15.75	83.07 4.74 14.39	65.16 4.23 -0.37	61.77 4.23 -4.68												
74TH STREET_GT_1 24260	44.43 3.12 -0.21	40.43 2.82 0.00	38.65 2.63 -0.50	38.48 2.47 -2.21	39.13 2.54 -2.30	46.00 4.33 -7.87	75.29 5.99 0.00	103.20 6.31 7.87	98.86 5.77 9.26	88.82 4.68 8.46	66.64 4.18 -1.91	63.05 -6.02												
74TH STREET_GT_2 24261	44.43 3.12 -0.21	40.43 2.82 0.00	38.65 2.63 -0.50	38.48 2.47 -2.21	39.13 2.54 -2.30	46.00 4.33 -7.87	75.29 5.99 0.00	103.20 6.31 7.87	98.86 5.77 9.26	88.82 4.68 8.46	66.64 4.18 -1.91	63.05 -6.02												
ADK HUDSON_FALLS 24011	44.03 2.67 -0.26	40.17 2.56 0.00	38.52 2.38 -0.62	38.81 2.26 -2.74	39.55 2.40 -2.86	59.38 2.63 -21.18	75.57 4.61 0.00	105.26 6.73 6.55	100.21 6.11 7.70	88.69 4.21 7.03	67.35 3.96 -3.34	63.91 3.33 -7.72												

(a) Data Example

The screenshot shows a table titled "Day-Ahead Market LBMP - Generator". The table has three columns: "CSV Files", "HTML Files", and "Last Updated". It lists three rows of data corresponding to the dates 12-07-2023, 12-06-2023, and 12-05-2023, with links to the respective files.

CSV Files	HTML Files	Last Updated
<a href="#">12-07-2023</a>	<a href="#">12-07-2023</a>	12/06/23 09:32 EST
<a href="#">12-06-2023</a>	<a href="#">12-06-2023</a>	12/05/23 09:34 EST
<a href="#">12-05-2023</a>	<a href="#">12-05-2023</a>	12/04/23 09:32 EST
<a href="#">12-04-2023</a>	<a href="#">12-04-2023</a>	12/03/23 09:32 EST

(b) Data Source Webpage

Figure 1: Examples of data.

## 2.2 Decision Variables and Explanations

The decision variables in the first round of optimization are:

$N_b$  : The number of batteries to rent of type  $b$

$W$  : The number of warehouses to rent

$B_{t,b}$  : The amount of power to purchase at time  $t$  for batteries  $b$

$S_{t,b}$  : The amount of power to sell at time  $t$  from batteries  $b$

After the first round of optimization on historical data, we decide on the amount of batteries and warehouses to rent using the optimal value from the historical analysis, fixing them for the next time period:

Fix :  $N_b = N_b^* \quad \forall b \in \{b_1 = \text{Lithium}, b_2 = \text{Lead}, b_3 = \text{Palladium}\}$

Fix :  $W = W^*$

For the next time month, we are then in the deployment setting and cannot change our investments. Every day at 11am, a 24 hour schedule of price data is released by the NY ISO. Given our fixed investments, we choose the optimal amount of power to buy and sell for each battery type:

- For each day  $d \in [1, 2, \dots, D=31]$  :
  - Solve a 24 hour optimization problem given price forecasts on that day, determining:
    - $S_{t,b} \quad \forall t \in \{1, 2, \dots, 24\} \quad \forall b \in \{b_1 = \text{Lithium}, b_2 = \text{Lead}, b_3 = \text{Palladium}\}$
    - $B_{t,b} \quad \forall t \in \{1, 2, \dots, 24\} \quad \forall b \in \{b_1 = \text{Lithium}, b_2 = \text{Lead}, b_3 = \text{Palladium}\}$

## 2.3 Objective Function

In the first optimization stage, we are allowed to choose the number of batteries for each type and number of warehouses to invest in. Our goal is to maximize historical profits, i.e.:

### Objective Function Stage 1

$$\begin{aligned}
 & \max_{S_{t,b}, B_{t,b}, N_b, W} \sum_{t=1}^T \sum_{b=1}^B [\text{Arbitrage Profits}] - \sum_{b=1}^B \text{Battery Costs} - \sum_{w=1}^W \text{Warehouse Costs} \\
 &= \max_{S_{t,b}, B_{t,b}, N_b, W} \sum_{t=1}^T \sum_{b=1}^B [(S_{t,b} - B_{t,b}) * P_t] - \sum_{b=1}^B N_b C_b - \sum_{w=1}^W Z_w C_w
 \end{aligned} \tag{1}$$

where:  $C_b$  is the cost of a battery type  $b$

$Z_w$  is a binary auxiliary variable equal to 1 if we rented warehouse  $w$

$C_w$  is the cost of warehouse  $w$

$P_t$  is the price of power at time  $t$

## Objective Function Stage 2

In the second stage of optimization, we ignore the fixed costs associated with investments already made, so the only decision variables we can change are  $S_{t,b}$  and  $B_{t,b}$ .

$$= \max_{S_{t,b}, B_{t,b}} \sum_{t=1}^{24} \sum_{b=1}^B [(S_{t,b} - B_{t,b}) * P_t] \quad (2)$$

## 2.4 Constraints

Charge Flow:  $0 \leq \sum_0^t [L_b B_{t,b} - S_{t,b}] \leq N_b A_b \quad \forall b \in B, \quad \forall t \in T$

Charge can't be negative or exceed capacity

Charging Rate:  $L_b B_{t,b} \leq M_b \quad \forall b \in B, \quad \forall t \in T$

Cannot charge faster than capacity

Discharge Rate:  $S_{t,b} \leq N_b \quad \forall b \in B, \quad \forall t \in T$

Cannot discharge faster than capacity

Warehouse Capacity:  $W * f \geq \sum_{b=1}^B V_b N_b$

Cannot store more batteries than we have warehouse space

Auxiliary Vars:  $W = \sum_{z=1}^Z Z_w$

Constraint for auxiliary vars

Where constants above are:

$A_b$  : Is the charge capacity of battery type  $b$

$L_b$  : Is the charging efficiency of battery type  $b$

$M_b$  : Is the maximum charge amount in one time period for battery type  $b$

$N_b$  : Is the maximum discharge amount in one time period for battery type  $b$

$V_b$  : Is the space (square feet) taken up for battery type  $b$

$f$  : Is the amount of square feet provided by a warehouse

## 2.5 Parameterization

Because it is harder to find a good next warehouse and takes labor to secure the rental contract, we model the rental costs as upwards increasing for each incremental warehouse.

Warehouse Number	Price
$Z_w = 1$	\$30,000
$Z_w = 2$	\$50,000
$Z_w = 3$	\$100,000
$Z_w = 4$	\$300,000
$Z_w = 5$	\$8,000,000

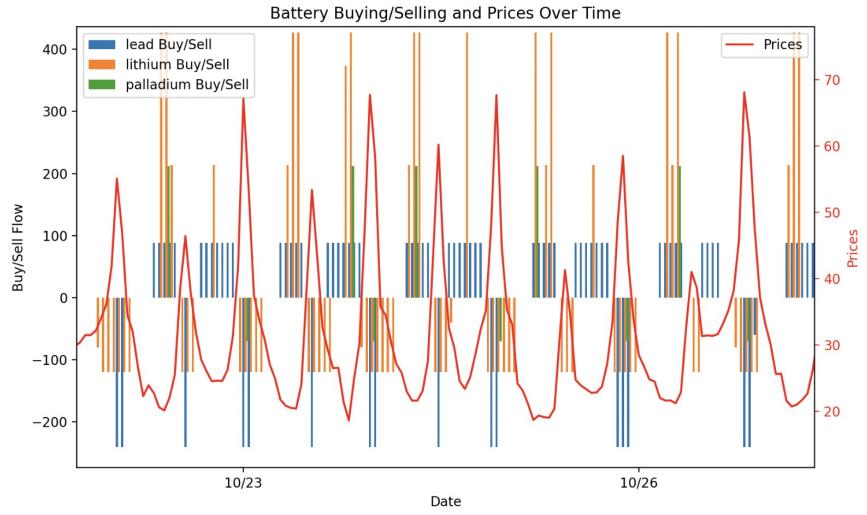
Figure 2: Warehouse Cost Schedule

Characteristic	Lithium	Lead	Palladium
<b>Size</b>	22.1	20.3	.1
<b>Capacity</b>	100	350	5
<b>Round Trip Efficiency</b>	.75	.68	.33
<b>Max Charge</b>	40	10	5
<b>Max Discharge</b>	15	40	5
<b>Cost</b>	\$12500	\$1100	\$50

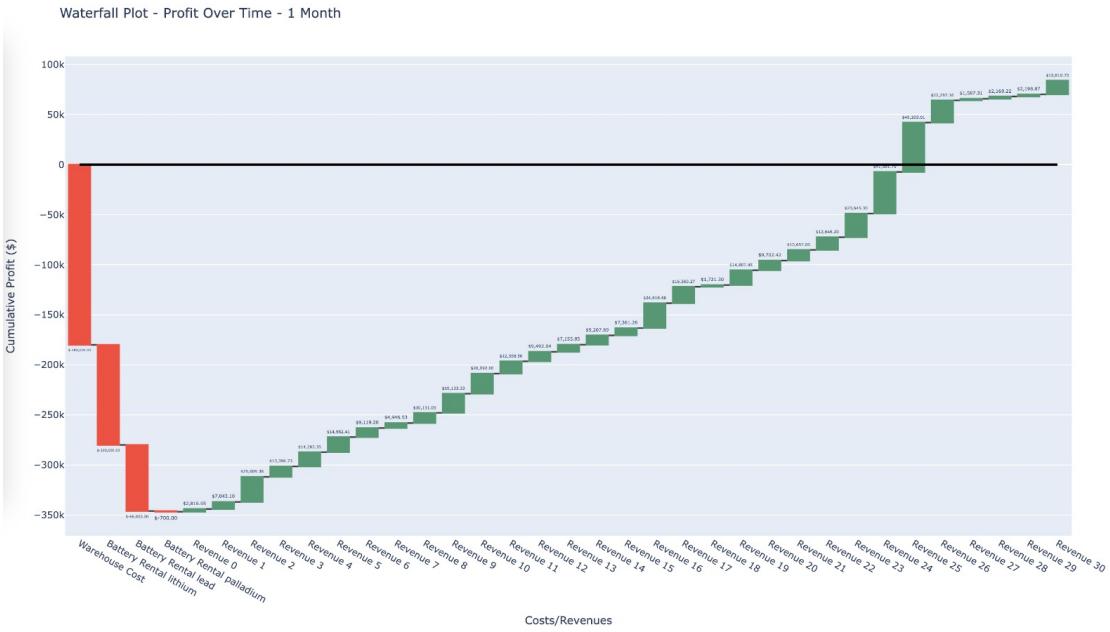
Figure 3: Battery Specs

## 2.6 Results

We tinkered with the parameterization in order to arrive at a battery investment plan that used all three types. We also adjusted costs to ensure that our profits were reasonable. Overall, it appears that using historical data to determine investments for future periods is successful as demonstrated in the waterfall plot of costs and revenues. Additionally, the time series of buying/selling decisions by batteries shows how the three battery types are utilized differently according to their specifications: some charge quickly, while others discharge quickly, and their round trip efficiencies raise or limit the price ratio differences needed in order to make the arbitrage profitable. The overall pattern is to buy when prices are cheap and sell when they are expensive, as expected. Our optimization formulation is successfully able to determine which batteries are best able to exploit price changes, however, and what the break-even point is with regard to balancing the marginal cost of another battery vs the revenue it could earn. *see plots below*



(a) Purchasing Plan Historical Data



(b) Profits in Month Ahead

Figure 4: Optimization Results

### 3 Optiguide Examples

#### 3.1 Overview

Attempting to assess the robustness of our model using Optiguide proved challenging due to the initial incompatibility with our code's webscraping components. We were able to overcome this obstacle, however, and extracted 10 insightful responses from optiguide. The results consistently highlighted a combination of batteries as the optimal choice across diverse scenarios, with our hypothetical battery, palladium, emerging as one of the top-performing types across the board.

#### 3.2 Example 1

What if warehouse costs were to be 13% more expensive?

Training:

```
Question: What if all warehouse costs are 15% more expensive?"  
Answer Code:  
```python  
for w in range(len(parameters['warehouse_data'])):  
    parameters['warehouse_data'][w]['cost'] *= 1.15  
-----
```

Output:

```
```python  
if warehouses_used is None:  
    warehouses_used = model.addVars(len(warehouse_data), vtype=GRB.BINARY, name=f'Number of warehouses')  
    decision_var_dict['warehouses_used'] = warehouses_used  
  
model.update()  
  
model.addConstr(  
    gp.quicksum(  
        warehouse_data[i]['area'] * warehouses_used[i] for i, warehouse in enumerate(warehouse_data)  
    ) >= total_area_needed,  
    name="Area_constraint"  
)  
  
# Update the 'obj' to reflect modified warehouse costs  
objs = objs + [(warehouse_data[i]['cost'] * 1.13) * warehouses_used[i] * -1 for i, warehouse in enumerate(warehouse_data)] # Increased warehouse costs by 13%  
  
model.update()  
  
Solved in 7 iterations and 0.01 seconds (0.00 work units)  
Optimal objective 8.493664385e+04  
[24576.635294117656, 7899.317647058826, 12628.023529411763, 11615.223529411767, 6748.799999999999, 3850.65882352941  
5, 9841.835294117653, 16403.999999999996, 19362.141176470584, 9837.388235294122, 7648.870588235302, 5843.4352941176  
44, 7130.047058823533, 5954.823529411762, 22108.023529411774, 13071.74117647059, 556.729411764707, 10560.4941176470  
64, 7416.564705882356, 7945.200000000001, 9458.541176470586, 19520.4705882353, 37805.30213903742, 43590.28235294117  
6, 17330.047058823533, 283.41176470588107, 1159.7647058823532, 1437.7411764705867, 10067.858823529412, 18249.176470  
58824, 84936.64385026737]  
{'lithium': 0.0, 'lead': 12.0, 'palladium': 114.0}
```

Figure 5: Optiguide Example 1

### 3.3 Example 2

What if our warehouses needed to install an additional 15 feet of cooling towers per warehouse?

Training:

```
Question: "What if our warehouses needed to install an additional 20 square feet for cooling towers?"  
Answer Code:  
```python  
dec = 20  
for p in range(num_periods):  
    for w in len(parameters['warehouses']):  
        parameters['warehouse_data'][w]['area'] -= dec  
...  
-----
```

```
OptiGuide Energy Arbitrage Example (to safeguard):
```

```
--- Code ---  
for w in parameters['warehouse_data']:  
    w['area'] -= 15  
--- One-Word Answer: SAFE or DANGER ---
```

Output:

```
Solved in 7 iterations and 0.01 seconds (0.00 work units)  
Optimal objective 8.493664385e+04  
[24576.635294117656, 7899.317647058826, 12628.023529411763, 11615.223529411767, 6748.799999999999, 3850.65882352941  
5, 9841.835294117653, 16403.999999999996, 19362.141176470584, 9837.388235294122, 7648.870588235302, 5843.4352941176  
44, 7130.047058823533, 5954.823529411762, 22108.023529411774, 13071.74117647059, 556.729411764707, 10560.4941176470  
64, 7416.564705882356, 7945.200000000001, 9458.541176470586, 19520.4705882353, 37805.30213903742, 43590.28235294117  
6, 17330.047058823533, 283.41176470588107, 1159.7647058823532, 1437.7411764705867, 10067.858823529412, 18249.176470  
58824, 84936.64385026737]  
{'lithium': 0.0, 'lead': 12.0, 'palladium': 114.0}
```

Figure 6: Optiguide Example 2

### 3.4 Example 3

What if our procurement team told us we had to have a minimum of 30 batteries and a maximum of 70 batteries?

Training:

```
Question: What if our procurement team told us we had to have a minimum of 30 batteries and a maximum of 70 batteries
Answer Code:
```python
model.addConstraint(decision_var_dict['battery_counts'] >= 30, f'MinBatteriesConstraint')
model.addConstraint(decision_var_dict['battery_counts'] <= 70, f'MaxBatteriesConstraint')
```
-----
```

Output:

```
--- Code ---
min_batteries = 20
max_batteries = 50
for battery_type in parameters['battery_types_used']:
    model.addConstr(decision_var_dict['battery_counts'][battery_type] >= min_batteries, f'MinBatteriesConstraint_{battery_type}')
    model.addConstr(decision_var_dict['battery_counts'][battery_type] <= max_batteries, f'MaxBatteriesConstraint_{battery_type}')

Optimal objective 1.019712360e+05
[28672.741176470598, 9215.870588235295, 14732.694117647066, 13551.094117647062, 7873.6, 4492.435294117644, 11482.14
1176470595, 19138.0, 22589.16470588236, 11476.952941176469, 8923.682352941169, 6817.341176470596, 8318.38823529412
2, 6947.2941176470595, 25792.694117647086, 15250.364705882355, 649.5176470588267, 12320.576470588236, 8652.65882352
9411, 9269.399999999998, 11034.964705882347, 22773.882352941182, 44202.890374331524, 50855.32941176471, 20218.38823
529411, 330.64705882353155, 1353.0588235294126, 1677.3647058823517, 11745.83529411765, 21290.705882352937, 101971.2
360071301]
{'lithium': 0.0, 'lead': 14.0, 'palladium': 158.0}
```

Figure 7: Optiguide Example 3

### 3.5 Example 4

What if we switch our generator from ADK HUDSON\_\_FALLS to ADK S GLENS\_\_FALLS?

Training:

Question: What if we switch our generator from ADK HUDSON\_\_FALLS to CORNELL\_\_?

Answer Code:

```
```python
parameters['generator_name'] = 'CORNELL_'
```

--- Code ---

```
parameters['generator_name'] = 'ADK S GLENS__FALLS'
```

Output:

```
Solved in 7 iterations and 0.00 seconds (0.00 work units)
Optimal objective 1.019712360e+05
[28672.741176470598, 9215.870588235295, 14732.694117647066, 13551.094117647062, 7873.6, 4492.435294117644, 11482.14
1176470595, 19138.0, 22589.16470588236, 11476.952941176469, 8923.682352941169, 6817.341176470596, 8318.38823529412
2, 6947.2941176470595, 25792.694117647086, 15250.364705882355, 649.5176470588267, 12320.576470588236, 8652.65882352
9411, 9269.399999999998, 11034.964705882347, 22773.882352941182, 44202.890374331524, 50855.32941176471, 20218.38823
529411, 330.64705882353155, 1353.0588235294126, 1677.3647058823517, 11745.83529411765, 21290.705882352937, 101971.2
360071301]
{'lithium': 0.0, 'lead': 14.0, 'palladium': 158.0}
```

Figure 8: Optiguide Example 4

### 3.6 Example 5

What if the battery charge capacity for Palladium was only 25% instead of 50%?

```
Question: What if the battery charge capacity for Lithium was only 70 percent instead of 95 percent?
```

```
Answer Code:
```

```
```python
charge_pct = 70
parameters['lithium']['charge_loss'] = charge_pct/100
```
```

--- Code ---

```
parameters['battery_types']['palladium']['charge_loss'] = 0.25
```

--- One-Word Answer: SAFE or DANGER ---

Output:

```
Solved in 10 iterations and 0.00 seconds (0.00 work units)
Optimal objective 1.52693333e+06
[840800.0, 445866.666666665, 662000.0, 541333.333333333, 342800.0, 231200.0, 388866.666666666, 776600.0, 992533.
3333333333, 613600.0, 471133.333333326, 436600.000000001, 366533.333333334, 327266.666666666, 957466.666666666
5, 765600.0, 121733.3333333337, 399533.333333326, 360933.3333333326, 380133.3333333326, 445733.3333333326, 71
0866.666666665, 1177600.0, 1340800.000000002, 967866.666666667, 113200.0, 127866.6666666663, 141266.666666666
3, 477933.333333326, 669200.0, 1526933.333333333]
{'lithium': 0.0, 'lead': 0.0, 'palladium': 4000.0}
```

Figure 9: Optiguide Example 5

### 3.7 Example 6

What if you added a vanadium flow battery and knew the specs as follows:

- size = 15
- capacity = 200
- charge loss = 80
- max charge = 140
- max\_discharge = 140
- cost = 550

#### Training

```
Question: What if you added a vanadium flow battery and knew the specs as follows:  
size = 18  
capacity = 270  
charge loss = 30%  
max charge = 1000  
max_discharge = 10  
cost = 800  
Answer Code:  
```python  
parameters['battery_type']['vanadium'] = {  
    'size': 18,  
    'capacity': 270,  
    'charge_loss': 0.3,  
    'max_charge': 1000,  
    'max_discharge': 10,  
    'cost': 800  
}  
parameters['battery_types_used'].append('vanadium')  
  
--- Code ---  
parameters['battery_type']['vanadium'] = {  
    'size': 15,  
    'capacity': 200,  
    'charge_loss': 0.8,  
    'max_charge': 140,  
    'max_discharge': 140,  
    'cost': 550  
}  
parameters['battery_types_used'].append('vanadium')  
  
Solved in 12 iterations and 0.00 seconds (0.00 work units)  
Optimal objective 4.026202394e+05  
[209215.5, 117301.5999999999, 156795.5999999998, 126430.2000000001, 84698.9, 63887.2000000001, 100287.2, 20801  
9.5, 236168.4000000002, 160997.2, 131353.3000000005, 120569.7999999997, 91518.7000000001, 84988.7999999999, 24  
6483.9000000002, 196821.3000000002, 37822.2000000001, 110419.4, 102427.0, 102049.9999999999, 116257.7000000000  
3, 198750.5, 305793.218181814, 372645.0, 254142.1999999995, 39126.1, 39731.9000000001, 40297.3999999994, 1207  
99.9000000001, 174557.5, 402620.2393939394]  
{'lithium': 0.0, 'lead': 0.0, 'palladium': 100.0, 'vanadium': 26.0}
```

Figure 10: Optiguide Example 6

### 3.8 Example 7

What if our procurement team told us we had to have a minimum of 5 batteries but doing so would get us a bulk discount of 20%?

```
Question: What if our procurement team told us we had to have a minimum of 20 but doing so would get us a bulk
discount of 10%?
Answer Code:
```python
discount = 10
parameters['battery_type']['cost'] *= (100-discount)/100
model.addConstraint(decision_var_dict['battery_counts'] >= 20, f'MinBatteriesConstraint')
```

--- Code ---
discount = 20
parameters['battery_type']['cost'] *= (100-discount)/100
model.addConstraint(decision_var_dict['battery_counts'] >= 5, f'MinBatteriesConstraint')

Solved in 7 iterations and 0.01 seconds (0.00 work units)
Optimal objective 1.019712360e+05
[28672.741176470598, 9215.870588235295, 14732.694117647066, 13551.094117647062, 7873.6, 4492.435294117644, 11482.14
1176470595, 19138.0, 22589.16470588236, 11476.952941176469, 8923.682352941169, 6817.341176470596, 8318.38823529412
2, 6947.2941176470595, 25792.694117647086, 15250.364705882355, 649.5176470588267, 12320.576470588236, 8652.65882352
9411, 9269.399999999998, 11034.964705882347, 22773.882352941182, 44202.890374331524, 50855.32941176471, 20218.38823
529411, 330.64705882353155, 1353.0588235294126, 1677.3647058823517, 11745.83529411765, 21290.705882352937, 101971.2
360071301]
{'lithium': 0.0, 'lead': 14.0, 'palladium': 158.0}
```

Figure 11: Optiguide Example 7

### 3.9 Example 8

What if every 10 periods, we can't sell electricity to the grid because the grid is undergoing maintenance?

```
Question: "What if every 14 periods, we can't sell electricity to the grid?"  
Answer Code:  
```python  
break_num = 14  
for p in range(num_periods):  
    if p%break_num == 0:  
        model.addConstr(sell[p] = 0, f'DischargeConstraint_period_{p+1}')  
...  
  
--- Code ---  
break_num = 10  
for p in range(num_periods):  
    if p%break_num == 0:  
        model.addConstr(sell[p] = 0, f'DischargeConstraint_period_{p+1}')  
  
Solved in 7 iterations and 0.00 seconds (0.00 work units)  
Optimal objective 1.019712360e+05  
[28672.741176470598, 9215.870588235295, 14732.694117647066, 13551.094117647062, 7873.6, 4492.435294117644, 11482.14  
1176470595, 19138.0, 22589.16470588236, 11476.952941176469, 8923.682352941169, 6817.341176470596, 8318.38823529412  
2, 6947.2941176470595, 25792.694117647086, 15250.364705882355, 649.5176470588267, 12320.576470588236, 8652.65882352  
9411, 9269.399999999998, 11034.964705882347, 22773.882352941182, 44202.890374331524, 50855.32941176471, 20218.38823  
529411, 330.64705882353155, 1353.0588235294126, 1677.3647058823517, 11745.83529411765, 21290.705882352937, 101971.2  
360071301]  
{'lithium': 0.0, 'lead': 14.0, 'palladium': 158.0}
```

Figure 12: Optiguide Example 8

### 3.10 Example 9

What if we're looking to maximize our total profits exclusively within the last 15 days of the month?

```
-----
Question: What if our procurement team told us we had to have a minimum of 20 but doing so would get us a bulk
discount of 10%?
Answer Code:
```python
discount = 10
parameters['battery_type']['cost'] *= (100-discount)/100
model.addConstraint(decision_var_dict['battery_counts'] >= 20, f'MinBatteriesConstraint')
.....
```

```
--- Code ---
discount = 20
parameters['battery_type']['cost'] *= (100-discount)/100
model.addConstraint(decision_var_dict['battery_counts'] >= 5, f'MinBatteriesConstraint')

Solved in 7 iterations and 0.01 seconds (0.00 work units)
Optimal objective 1.019712360e+05
[28672.741176470598, 9215.870588235295, 14732.694117647066, 13551.094117647062, 7873.6, 4492.435294117644, 11482.14
1176470595, 19138.0, 22589.16470588236, 11476.952941176469, 8923.682352941169, 6817.341176470596, 8318.38823529412
2, 6947.2941176470595, 25792.694117647086, 15250.364705882355, 649.5176470588267, 12320.576470588236, 8652.65882352
9411, 9269.399999999998, 11034.964705882347, 22773.882352941182, 44202.890374331524, 50855.32941176471, 20218.38823
529411, 330.64705882353155, 1353.0588235294126, 1677.3647058823517, 11745.83529411765, 21290.705882352937, 101971.2
360071301]
{'lithium': 0.0, 'lead': 14.0, 'palladium': 158.0}
```

Figure 13: Optiguide Example 9

### 3.11 Example 10

What if you added a vanadium flow battery and knew the specs as follows:

- size = 15
- capacity = 200
- charge loss = 80
- max charge = 140
- max\_discharge = 140
- cost = 550

Then, our procurement team tells us **we have to order a minimum of 20 vanadium batteries.**

#### Training:

```
Question: What if you added a vanadium flow battery and knew the specs as follows:  
size = 18  
capacity = 270  
charge_loss = 30%  
max_charge = 1000  
max_discharge = 10  
cost = 800  
Answer Code:  
```python  
parameters['battery_type']['vanadium'] = {  
    'size': 18,  
    'capacity': 270,  
    'charge_loss': 0.3,  
    'max_charge': 1000,  
    'max_discharge': 10,  
    'cost': 800  
}  
parameters['battery_types_used'].append('vanadium')  
  
--- Code ---  
# Adding the specifications of the vanadium battery to the battery_types dictionary in parameters  
parameters['battery_types']['vanadium'] = {  
    'size': 15,  
    'capacity': 200,  
    'charge_loss': 0.8,  
    'max_charge': 140,  
    'max_discharge': 140,  
    'cost': 550  
}  
  
# Including vanadium as one of the battery types used  
parameters['battery_types_used'].append('vanadium')  
  
# if still in the setup phase and not yet in the run phase  
# we specify it in the setup phase like this:  
parameters['battery_counts'] = {'vanadium': 20}  
  
# But, if we're already in the run phase where the model is defined,  
# we can add a constraint that the minimum number of vanadium batteries used should be 20  
model.addConstr(decision_var_dict['battery_counts']['vanadium'] >= 20, "MinNumberOfVanadiumBatteries")
```

Figure 14: Optiguide Example 10

### 3.12 Optiguide Conclusion

Regardless of how we adjust our battery related constraints for this exercise, Palladium is universally the best performing battery.

It seems as though reducing the amount of warehouse space has the largest impact on our profit. When we install 15 square feet of cooling towers or increase warehouse costs by small degrees, our total profit does not exceed \$9,000. Additionally, because palladium batteries, in this scenario, are near 1/50 of the cost of their alternatives, it's plausible that we as the contractors would purchase more Palladium batteries overall. However, we also purchase lead batteries because of their profitability per capacity, which performs best among the three battery types. Palladium generates almost a third of the lead battery's profit per capacity.

In all of our optiguide scenarios, we don't purchase lithium batteries. Even though lithium batteries boast the fastest charge rate and the best battery efficiency, over a month-long period, given the battery's large size, the lithium battery's strengths profit generating than the lead battery's ability to store more charge with overall lower battery costs.

While the choices to buy or sell are not represented within our optiguide interaction screenshots, the general pattern remains the same. Our contractor is incentivized to buy electricity during low-demand periods earlier in the day and sell closer to the evening when we see increases in residential electricity consumption after office hours.

Even though the conclusions from our optiguide simulation are straightforward, we faced several challenges working with optiguide's functions. Because of our program's computational intensity, we quickly burned through \$20 running through optiguide simulations. We were wary of making adjustments to the demand side of the data by accessing the web scraping elements because we were afraid that doing so would take several attempts and rapidly deplete our budget. Therefore, the majority of our optiguide interactions focus on altering factors tied to the first part of our two-stage problem.

## 4 Appendix

[https://github.com/colton-lapp/EnergyArbitrage/blob/complicate-model/Code/optiguide\\_example.ipynb](https://github.com/colton-lapp/EnergyArbitrage/blob/complicate-model/Code/optiguide_example.ipynb)