Part 1.a

New objective function:

$$\text{Cost} = Q_A\text{Cost}_A + Q_B\text{Cost}_B + q\text{Cost}_q - C_C(Q_A + Q_B)\text{Cost}_C$$

$$Q_A + Q_B = C_{\text{valve}}\sqrt{h}$$

```python
# Define the objective function to minimize cost
def objFunc(x,*args):
    Ca, Cb, Cc, T, h, Qa, Qb, q = x
    costA, costB, costC, costq = args
    cost = (Qa*costA + Qb*costB + q*costq) - Cc*(Qa+Qb)*costC
    return cost
```

Part 1.b

New constraint functions.

$$V = h(\frac{\pi d^2}{4})$$

$$Q_{\text{in}} = Q_A + Q_B$$

$$Q_{\text{out}} = C_{\text{valve}}\sqrt{h}$$

$$R_{\text{rxn}} = k_0 e^{\frac{-E_a}{RT}} C_A C_B V$$

$$0 = Q_{\text{in}} - Q_{\text{out}}$$

$$0 = Q_A C_{A0} - Q_{\text{out}} C_A - R_{\text{rxn}}$$

$$0 = Q_B C_{B0} - Q_{\text{out}} C_B - R_{\text{rxn}}$$

$$0 = -Q_{\text{out}} C_c + R_{\text{rxn}}$$

$$0 = Q_A T_A + Q_B T_B + \frac{q}{\rho C_p} - Q_{\text{out}} T - \frac{dH R_{\text{rxn}}}{\rho C_p}$$

```python
# Define the constraints
def constraints(decisionVar):
    Ca, Cb, Cc, T, h, Qa, Qb, q = decisionVar
    V    = h*(np.pi*d*d/4)
    Qin  = Qa + Qb
    Qout = Cvalve*np.sqrt(h)
    Rrxn = k0*np.exp(-Ea/R/T)*Ca*Cb*V
    q    = q*1e6  # convert GJ to kJ

    g0   = Qin - Qout
    g1   = Qa*Ca0 - Qout*Ca - Rrxn
    g2   = Qb*Cb0 - Qout*Cb - Rrxn
    g3   = -Qout*Cc + Rrxn
    g4   = Qa*Ta + Qb*Tb + q/rho/Cp - Qout*T - dH*Rrxn/rho/Cp

    return g0,g1,g2,g3,g4
```

Part 1.c

```
# Define the bounds for the decision variables
# Ca, Cb, Cc, T, h, Qa, Qb, q
Ca_bound = (0,20)
Cb_bound = (0,20)
Cc_bound = (0,10)
T_bound  = (300,620)
h_bound  = (1,8)
Qa_bound = (1,8)
Qb_bound = (1,8)
q_bound  = (0,12)
bounds = [Ca_bound,Cb_bound,Cc_bound,T_bound,h_bound,Qa_bound,Qb_bound,q_bound]

# Define the inequality contraints
constraints = ({'type': 'eq', 'fun': constraints})

# Initial guess for the decision variables
Ca_g = 2
Cb_g = 2
Cc_g = 5
T_g  = 400
h_g  = 8
Qa_g = 6
Qb_g = 7
q_g  = 10

guess  = [Ca_g,Cb_g,Cc_g,T_g,h_g,Qa_g,Qb_g,q_g]

# Minimize command
def optFunc(otherParams, guess):
    opt =  minimize(objFunc,guess,args=(otherParams),method='SLSQP', bounds=bounds, cons
#     print('Decision variable = ', opt.x)
#     print('Profit = ', opt.fun)
    return opt.x[4],opt.x[6],opt.x[7]
```

Part 2.a

Cost A	Cost B	Cost C	Cost q	Q_B^*	q^*	h^*	Profit ($/min)
2	5	5	2	4.41	11.05	8.0	139.00
2	50	20	2	4.08	10.97	8.0	523.64
2	10	5	20	2.90	0	8.0	30.97*
20	50	2	20	1	0	1.0	-87.41

Part 2.b

In the last scenario, the reactor is trying to minimize the loss on the days that the costs of reactants are high by putting the least amount of Qa, Qb, q, and setting the lowest h.

Part 3.a

Total profit of the plant in 1 year: $ 83841662.81

Part 3.b

Percentage increase in profit compared to not optimizing: 68%

```
not_optimized = 49906368.896

not_opt_increased = (new_optimized - not_optimized)/not_optimized*100
print(not_opt_increased)
```

```
67.99792224596563
```

Part 3.c

Percentage increase in profit compared to not optimizing: 18%

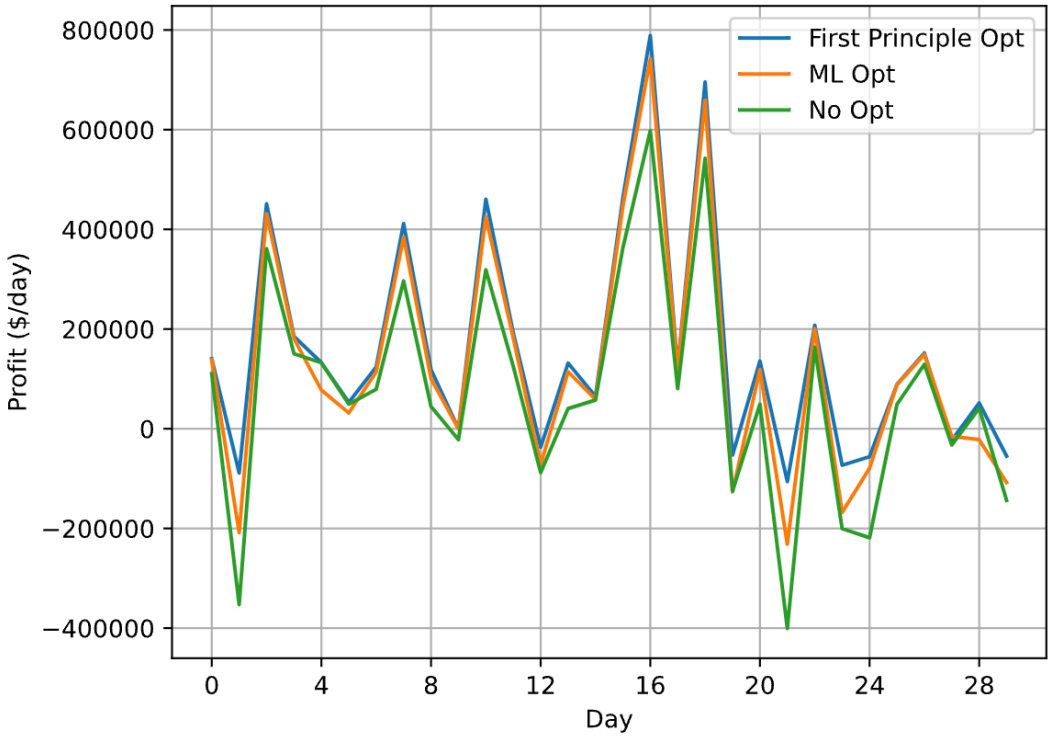
```
old_optimized = 70951383.737

old_increased = (new_optimized - old_optimized)/old_optimized*100
print(old_increased)
```

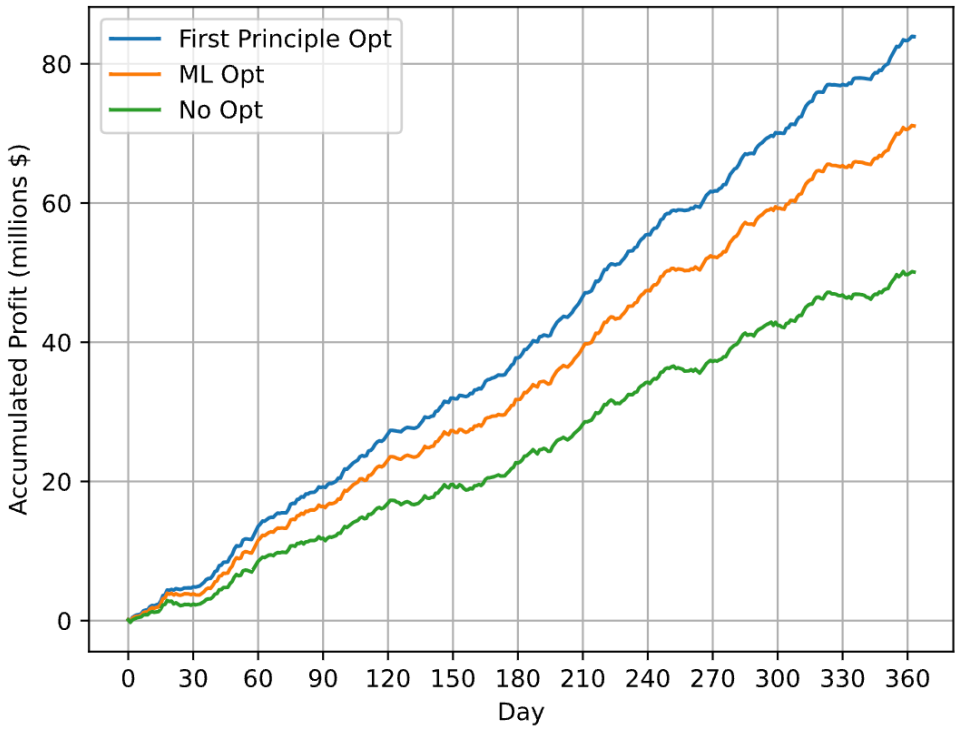
```
18.16776276621758
```

Part 3.d

The first 30 days of operation.



Part 3.e



Part 3.f

The profit from first principle optimization has a 68% increase compared to the non-optimization method, and 18% increased compared to the machine learning method. Both optimization methods brought in higher profits because they changed the plant's setpoints according to the daily fluctuation of the price of reactants. Going beyond that, the first principle optimization is better than the machine learning method because the parameters were not only tuned to decrease the cost of production, but also to minimize the inevitable loss at certain days.