In [ ]:
```
import pyomo.environ as pyo
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn')
```

Load Data

In [ ]:
```
plant_properties = pd.read_csv('plantProps - Q1.csv').to_numpy()
plant_properties
```

Out[ ]:
```
array([['FuelCost ($/MWh)', 30, 25, 45, 60],
       ['OperatingCost ($/h)', 20, 30, 50, 10],
       ['StartupCost ($/switch)', 800, 650, 200, 80],
       ['MinGenerationLevel (MW)', 120, 90, 40, 3],
       ['MaxGenerationLevel (MW)', 800, 250, 200, 120],
       ['RampUpLimit (MW/h)', 30, 40, 80, 900],
       ['RampDownLimit (MW/h)', 50, 50, 100, 1800],
       ['MinimumUpTime (h)', 4, 3, 5, 1],
       ['MinimumDownTime (h)', 3, 1, 4, 1]], dtype=object)
```

In [ ]:
```
plant_properties = plant_properties[:,1:]
plant_properties
```

Out[ ]:
```
array([[30, 25, 45, 60],
       [20, 30, 50, 10],
       [800, 650, 200, 80],
       [120, 90, 40, 3],
       [800, 250, 200, 120],
       [30, 40, 80, 900],
       [50, 50, 100, 1800],
       [4, 3, 5, 1],
       [3, 1, 4, 1]], dtype=object)
```

In [ ]:
```
load_profile = pd.read_csv('loadProfile - Q1.csv').to_numpy()
load_profile = load_profile[:,1]
load_profile
```

Out[ ]:
```
array([185.4828134, 148.8480317, 136.006771 , 134.7357675, 137.3737109,
       153.9476645, 199.0182488, 243.3924932, 318.7554353, 385.2574395,
       413.3359044, 421.1049789, 405.6551978, 383.64289  , 362.4389527,
       378.3462924, 531.8660873, 641.4227544, 521.1043771, 474.3723767,
       422.713907 , 340.1245408, 250.1899275, 171.9553108, 139.0226624,
       119.2198799, 110.1121918, 100.       , 124.2801057, 182.8429973,
       264.5152453, 334.9288398, 406.9497271, 457.0100045, 480.2948178,
       492.3043364, 471.2083671, 447.5119297, 428.9733058, 463.8135369,
       627.205469 , 752.0483629, 597.8954387, 561.3042094, 488.68875  ,
       364.8736097, 264.2728658, 211.683914 , 192.6028372, 180.247302 ,
       195.7312391, 196.2442119, 259.2080246, 403.9510175, 666.6131807,
       754.9756656, 756.1715905, 737.0350648, 777.0478798, 732.1612308,
       687.980223 , 711.462273 , 683.291965 , 690.0044218, 858.9195055,
       943.4656682, 778.9301866, 726.9369313, 669.751109 , 563.0333569,
       456.9268876, 379.7749993, 309.8576303, 288.1278936, 299.885404 ,
       300.9178131, 340.9376981, 427.1840165, 709.9356043, 794.2653994,
       805.3991849, 796.2922314, 816.0451694, 779.920994 , 741.4746617,
       766.6398017, 731.5417434, 763.490417 , 887.1462983, 940.5195551,
       777.6267026, 727.9085663, 667.4751723, 553.1116525, 451.3369505,
       364.3918788, 302.1211214, 294.9899791, 290.0724265, 280.4694655,
       320.0899395, 424.4996604, 694.2194829, 782.8066284, 782.006771 ,
       774.8167323, 769.1420152, 718.7836219, 691.9367916, 723.0050217,
       690.9951876, 717.4708921, 840.7633513, 895.1722871, 755.2812914,
       702.8882875, 620.923672 , 508.5313102, 397.7433825, 325.7792816,
       269.6072762, 261.1742091, 255.3286223, 242.4778714, 285.7356282,
       376.922561 , 619.8940554, 733.2939768, 736.8796567, 739.0696823,
       735.2768633, 675.5912474, 636.425574 , 647.0037698, 615.4274244,
       636.7358391, 749.9615578, 822.2032241, 664.9177629, 619.3689298,
       558.8591735, 466.7119997, 387.3327111, 316.4655285, 255.2337216,
       244.9700388, 234.6334931, 221.4649393, 234.3533076, 257.4352364,
       332.0044674, 407.7408741, 472.7583122, 502.9667752, 523.5497127,
       528.8474206, 501.2162238, 477.5062167, 458.3627649, 467.7999841,
       591.3371511, 756.5988649, 606.2812131, 565.689144 , 514.6191725,
       446.9630254, 374.6387783, 307.8859006])
```

In [ ]:
```
n_hours = load_profile.shape[0]
time = np.arange(1,n_hours+1)
```

In [ ]:
```
generation_target = load_profile*1.05
```

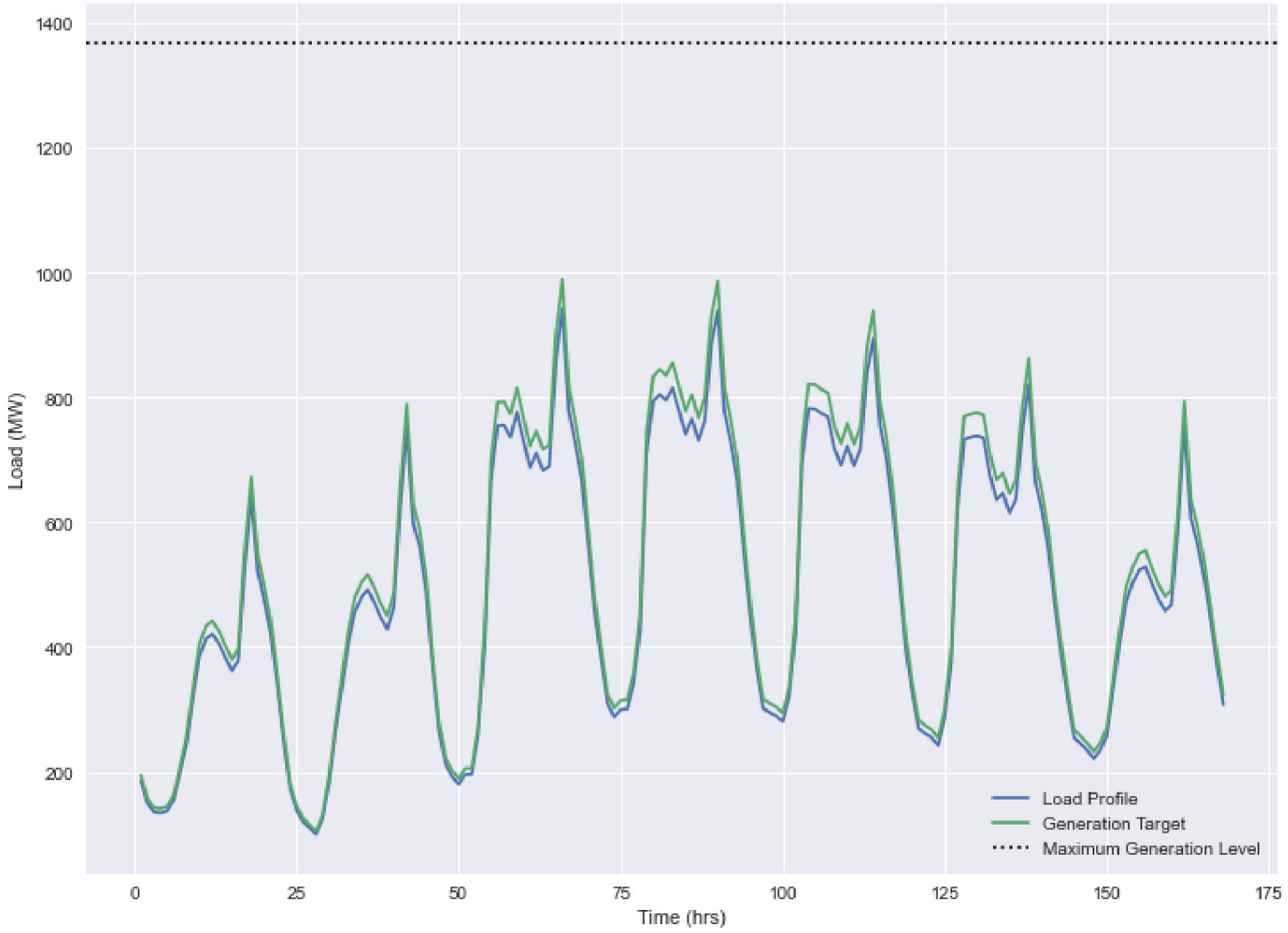
In [ ]:
```
installed_capacity = sum(plant_properties[4,:])
```

In [ ]:
```
if installed_capacity < max(generation_target):
    print('Error! Maximum of load profile {:2f} MW is above installed capacity {:2f} MW!'.format(max(generation_target)
```

```
    installed_capacity))
```

In [ ]:
```
# Visualize data
plt.figure(figsize=(12, 9))
plt.clf()
plt.plot(time, load_profile)
plt.plot(time, generation_target)
plt.axhline(y=installed_capacity, color="black", linestyle=":")
plt.xlabel('Time (hrs)')
plt.ylabel('Load (MW)')
plt.legend(['Load Profile','Generation Target','Maximum Generation Level'],loc='best')
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x1a6838fa040>
```



Define Constraints

In [ ]:
```
fuel_cost = plant_properties[0,:]
operating_cost = plant_properties[1,:]
startup_cost = plant_properties[2,:]
min_generation_level = plant_properties[3,:]
max_generation_level = plant_properties[4,:]
ramp_up_limit = plant_properties[5,:]
ramp_down_limit = plant_properties[6,:]
```

```
minimum_up_time = plant_properties[7,:]
minimum_down_time = plant_properties[8,:]
```

Define Optimization Problem

In [ ]:
```
model = pyo.ConcreteModel()

#index sets initialization
model.hours=pyo.Set(initialize=np.array([i for i in range(0,n_hours)]))
model.hours_1N=pyo.Set(initialize=np.array([i for i in range(1,n_hours)]))

#decision variables configuration
model.coal_old = pyo.Var(model.hours, domain = pyo.NonNegativeReals)
model.coal_new = pyo.Var(model.hours, domain = pyo.NonNegativeReals)
model.gas = pyo.Var(model.hours, domain = pyo.NonNegativeReals)
model.peaker = pyo.Var(model.hours, domain = pyo.NonNegativeReals)

model.max_generation=pyo.Var(model.hours, domain = pyo.NonNegativeReals)
model.total_generation=sum(model.coal_old[j]+model.coal_new[j]+model.gas[j]+model.peaker[j] for j in model.hours)
```

Define Objective Function

In [ ]:
```
model.obj=pyo.Objective(expr=sum(model.coal_old[i]*fuel_cost[0]+
model.coal_new[i]*fuel_cost[1]+
model.gas[i]*fuel_cost[2]+
model.peaker[i]*fuel_cost[3]
for i in model.hours))
```

Solve Optimization

In [ ]:
```
pyo.SolverFactory('glpk', executable="GPLK/w64/glpsol.exe").solve(model)
```

```
WARNING: Empty constraint block written in LP format - solver may error
```

Out[ ]:
{'Problem': [{'Name': 'unknown', 'Lower bound': 0.0, 'Upper bound': 0.0, 'Number of objectives': 1, 'Number of constrai
nts': 1, 'Number of variables': 673, 'Number of nonzeros': 1, 'Sense': 'minimize'}], 'Solver': [{'Status': 'ok', 'Termi
nation condition': 'optimal', 'Statistics': {'Branch and bound': {'Number of bounded subproblems': 0, 'Number of create
d subproblems': 0}}, 'Error rc': 0, 'Time': 0.0444483757019043}], 'Solution': [OrderedDict([('number of solutions', 0),
('number of solutions displayed', 0)])]}