```
In [104…
#import data from sources
import pandas as pd
from functools import reduce
import requests
import os
import pathlib
from datetime import datetime
import matplotlib.pyplot as plt
from numpy import random
from tabulate import tabulate
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
import seaborn as sns
from itertools import combinations
import math
import bisect
import sys
```

Simulation of Prosumers in a Shapley Coalition Utilizing NRG Market Model

Description:

A game theory cooperative coalition designed using an NRG pay out model for prosumer net generation and consumer consumption.

What is NRG-X-Change? "In this paper we propose NRG-X-Change — anovel mechanism for trading of locally produced re-newable energy that does not rely on an energy mar-ket or matching of orders ahead of time. In our modellocally produced energy is continuously fed into thegrid and payment is received based on actual usage,rather than predicted, as consumption is measured bythe DSO (Distribution Service Operator) and billed in near real-time." (Mihail Mihaylov)

NRG-Xchange is a Non-Linear Payout:

The NRG-X-Change as described by the authors performs a dynamic payment to prosumers that are capable of meeting the demand of the micro-grid. The micro-grid is made of prosumers and consumers. As the load demand spikes the pricing for net generation also spikes to meet the demand. When there is too much generation on the grid the pricing drops encouraging prosumers to generate less and consumer to consume more. The payout function g(.) , utilizes a normalization component in the denominator to account for over or under generation distributing the payout along the curve. The payment is at its highest when generation meets the total demand and at its lowest as generation starts to saturate the market because of low demand.

$$g(x, t_p, t_c) = \frac{x^n * q_{t_p = t_c}}{e^{\frac{(t_p - t_c)^2}{a}}}$$

Where x, is the net energy of the prosumer. q, is the maximum price allowed. t_p ,is the total produced energy of all prosumers. t_c is the total consumption of all the prosumers. a , is a scaling constant to adjust the pay out. Similarly lets consider the cost of energy for consumers to purchase based on pricing set by the h(.) function. In tandem these incentives are non-linear because of the distribution curve. The shape of that curve can be adjusted to the size of the network and the volatility of the network.

$$h(y, t_p, t_c) = \frac{y * r_{t_c >> t_p} * t_c}{t_c + t_p}$$

Where y is the withdrawn energy, and $r_{t_c >> t_p}$ is the maximum cost of energy delivered by the utility when the energy supply by prosumers is low. Again, t_p is the total production and t_c is the total consumption of the prosumers in the network. The minimum payment by the utility in the historical payment prices would indicate the minimum amount willing to charge customers for energy in order to cover the cost of delivering the energy. We will use the minimum price in our list for r.

In [105…

```
# NRGXChange g(.) Function
def g(price,p,tp,tc,a,n):
    x = p
    q = (0.01*price)
    try:
        #print(f"n{n},tp{tp},tc{tc},a{a},q{q}")
        pay = abs((pow(x,n)*q)/math.exp(pow((tp-tc),2)/a))
    except OverflowError:
        pay = float('inf')
    return pay

# NRGXChange h(.) Function
def h(price,c,tp,tc):
    y = c
    r = (0.01*price)
    try:
        cost = (y*r*tc)/(tc+tp)
    except OverflowError:
        cost = float('inf')
    return cost
```

Cooperative Game for Prosumers with Non-Linear Payout

The game is in terms of a **characteristic function**, which specfies for every group of players the total payoff that the members of S can by signing an greement among themselves; this payoff is available for distribution among the members of the group. A coalitional game with transferable payoff is a pair $< N, v >$ where $N = \{1, \ldots, n\}$ is the set of players and for every subset S of I (called a coalition) $v(S) \in \mathbb{R}$ is the total payoff that is available for division among members of S (called the worth of S). We assume that the larger the coalition the larger the payoff (this property is called superadditivity).

An agreement amongst players is a list $(x_1, x_1, \ldots, x_n)$ where x_1, is the proposed payoff to individual i. Shapley value is interpreted in terms of **expected marginal contribution**. It is calculated by considering all the possible orders of arrival of the players into a room and giving each player his marginal contribution.

Example of Shapley Contributions

Assume there are two players that are prosumers on a micro-grid. The prosumers contribute to the micro-grid in order to meet the demand of a consumer on the grid. In this example the consumer is not treated as a player. The prosumers can supply the demand in the two different ways based on order of arrival.

1. Prosumer 1 can supply a partial amount of the demand first and then Prosumer 2 can supply the rest.
2. Prosumer 2 can supply a partial amount of the demand first and then Prosumer 1 can supply the rest.

If the consumer payment is not linear but increases the more the total demand is met then there is an incentive for the generation to be completely filled to get back more money for it. That means that as the generation nears 100% of the demand the pricing will be more therefore the combination of the two prosumers generation becomes a superadditive payout.

Let's choose a moment in time that the consumer is demanding energy to be placed on the micro-grid. At that moment, Prosumer 1, would only be able to fulfill 75% of the request and would be paid at the going price times the max amount provided, $v(1) = 0.77$. Prosumer 2, would only be able to fulfill 50% of the demand (less effecient generation), it would also be paid at the going price times the max amount of energy provided,$v(1) = 0.51$. If the complete demand is met then the payout would be $v(12) = 1.04$, where the notation for contributions of both is interchangeable since it results in the same total payout, i.e. $v(12) == v(21)$

Consider Prosumer 1 fulfills the demand at 75%, and Prosumer 2 fulfills the demand at 25% (with a surples of 25%), compared to Prosumer 2 fulfilling 50% of the demand and then Prosumer 1 fulfilling the remaining 50% (with a surplus of 25%). The marginal contribution of each player can be different depending on how much they have to provide. Since the one that can provide the most can capture more of the superadditive pricing then the one who cannot provide the most would not be subject to the same superadditive pricing and needs to be paid according to its capacity to contribute to the overall worth of the coalition.

If Prosumer 1 contributes first then Prosumer 2, Prosumer 1's contribution is $v(1) = 0.77$ when Prosumer 2 arrives the surplus increases from 0.77 to 1.04 and therefore Prosumer 2's marginal contribution is $v(2) = v(12) - v(1)$, $0.27 = 1.04 - 0.77$.

If Prosumer 2 contributes first then Prosumer 1, Prosumer 2's contribution is $v(2) = 0.51$ when Prosumer 1 arrives the surplus increases from 0.51 to 1.04 and therefore Prosumer 1's marginal contribution is $v(1) = v(12) - v(2)$, $0.53 = 1.04 - 0.51$.

Probability	Order of Arrival	P1 M.C.	P2 M.C.
1/2	First 1 then 2	0.77	0.27
1/2	First 2 then 1	0.53	0.51

Expected Marginal Contribution (Shapley Value Calculation)

	P1 Shapley		P2 Shapley
$\frac{1}{2} * 0.77 + \frac{1}{2} * 0.53$	= 0.65	$\frac{1}{2} * 0.27 + \frac{1}{2} * 0.51$	= 0.39

In conclusion, if Prosumer 1 and Prosumer 2 have the capacity to fulfill the need of the demand at a given moment in time, they can either request individually to supplement the need and recieve individual payment based on what they offer or join in a coalition to supplement more of the demand together and get paid through marginal contributions.

Follow-up to the Example : What Happens to Excess Energy?

Prosumers are generating energy for the demand but the joined contribution to the demand of multiple prosumers can become a surplus to the demand requested. At the point that the net energy is more than the demand it must be dissipated or "curtailed". In the example prosumers would first find consensus on contributing a given amount based on its overall capacity then a setpoint would allow the prosumers to feed the power based on the agreed setpoint. The curtailment of the delivered energy can be done in a system that converts DC to AC and limits the injection of power to the Grid. The curtailment would shed the excess energy, by limiting or temporarily disconnecting the solar panels. Another option would be to store the excess energy into battery or hydrogen storage to be retrieved later. This scenario is not explored in this study but can provide additional benefits.

Shapley Value Calculation

We can model the a shapley value calculation based on the number of players and the marginal contribution logic described in the previous example. Each players characteristic function will determine the players max contribution. The order of arrival for each player is a combinatorial assesment based on number of players , N . The probability of each combination is $1/N!$. The probability will be multiplied by the marginal contribution of each player for every combination of the other players order of arrival. It will then be summed up for that player. The final result would be the calcualted shapley value for the player. The function below recieves a list of the players characteristic function output and the number of players in order to return the shapley values.

In [106…

```
# Shapley Value Python Logic
# Authored by Susobhan Ghosh
# https://github.com/susobhang70
# Committed on 02/01/2020

def power_set(List):
    PS = [list(j) for i in range(len(List)) for j in combinations(List, i+1)]
    return PS

def get_shapley(n,v):
    tempList = list([i for i in range(n)])
    N = power_set(tempList)
    shapley_values = []
    for i in range(n):
        shapley = 0
        for j in N:
            if i not in j:
                cmod = len(j)
                Cui = j[:]
                bisect.insort_left(Cui,i)
                l = N.index(j)
                k = N.index(Cui)
                temp = float(float(v[k]) - float(v[l])) *\
                           float(math.factorial(cmod) * math.factorial(n - cmod - 1)) / float(math.factorial(n)
                shapley += temp
        cmod = 0
        Cui = [i]
```

```
            k = N.index(Cui)
            temp = float(v[k]) * float(math.factorial(cmod) * math.factorial(n - cmod - 1)) / float(math.factorial(
            shapley += temp
            shapley_values.append(shapley)
        return shapley_values
```

Lets test the shapley function by leveraging our previous example. The number of players is $n = 2$ and the characterstic input that we used in the example is $v = [v(1), v(2), v(1,2)], v = [0.10, 0.12, 0.23]$. Note that the result of the scripted function matches the example provided in the text. We can now use it for more than 2 players and still calculate the shapley values accurately.

In [107…

```
n=2
v=[0.77,0.51,1.04]
vals = get_shapley(n,v)
print(f"Prosumer-1: ${vals[0]}, Prosumer-2: ${vals[1]}")
```

```
Prosumer-1: $0.65, Prosumer-2: $0.39
```

Modeling a Prosumer Market with Coalitions

We will leverage an NRGX-Change based market of prosumers on an micro-grid. Any excess energy that is not consumed by the micro-grid is not considered at this time. The goal of each prosumer would be to offset the demand of the micro-grid. In most cases the total demand would need to be supplemented by the larger Grid at some retail electricity price. The network will use the payout and consumption functions for NRGX-Change as each prosumer generates every month.

Data Gathering

The data gathered is from EIA.gov. The data was then used to synthesize typical prosumer consumption and generation over the course of 12 months. Volatitlity in the usage and generation was added as a normal distribution with a given variance to simulate real world conditions. The data is pulled from a local file and then sorted by timestamp. The fields are grouped by id and by time. Grouping by time allows for settlement calculations to occur at each time interval.

In [108…

```
#Import the data to local csv
all_prosumers_data = pd.read_csv('data/prosumer_N3_model_20210129_1416.csv')
all_prosumers_data["time"] = pd.to_datetime(all_prosumers_data['time'], format='%Y-%m-%d %H:%M')
all_prosumers_data.sort_values(by='time')
prosumer_data_by_id = [pd.DataFrame(y) for x, y in all_prosumers_data.groupby('id', as_index=False)]
N=len(prosumer_data_by_id)
```

```
prosumers_at_t = [pd.DataFrame(y) for x, y in all_prosumers_data.groupby('time', as_index=False)]
prosumers_at_t[0]
```

Out[108…

	time	demand	generation	consumption	net_energy	price	id
13	2019-10-01	785.747309	567.176089	218.571220	0.000000	11.66	1
27	2019-10-01	706.638381	897.462539	0.000000	-190.824158	11.66	2
41	2019-10-01	996.729985	392.570712	604.159273	0.000000	11.66	3

Apply NRGX-Change Settlement

There are lower layers to the actual movement of electricity between the prosumers and the DSO. We will stay at the financial transaction layer and not consider the interconnection, blockchain transactions and controls at this time. The financial transactions for net energy produced by prosumers is used in the payment function along with the net energy procued by other prosumers. If the demand for energy was great at that time and other prosumers where not filling in for the need then the price for the energy contributed goes up. This scheme relies on each prosumer not coordinating with others. Over-producing in this market will lead to lesser payment. By producing the exact amount that is needed then the payment is maximized.

Lets apply the payment functions and determine the profits of each prosumer accordingly.

1. Determine the max/min pricing to set the max/min pricing limits for the payout/cost functions. These limits are addresing the total payment allowed for prosumers and the minimum payment needed to deliver power, respectively.
2. Calculate the t_c total consumption and t_p total production of the set of prosumers.
3. Apply g(.) to any prosumer who is providing net_energy in order to calculate the payout for the net energy.
4. Apply h(.) to any prosumer that has not covered their own consumption needs and needs to buy the energy from the NRGX-Changes Market's DSO who would then use that demand. When consumption is high the pricing function for net energy pay out is automatically higher, and the cost for hte consumption is automatically higher.

In [109…

```
#historical pricing
max_price = all_prosumers_data['price'].max()
min_price = all_prosumers_data['price'].min()

#calculate the payment at time t, for all prosumers
payments=[]
```

```
t_periods = len(prosumers_at_t)
for i in range(t_periods):
    tc = prosumers_at_t[i]['consumption'].sum()
    tp = abs(prosumers_at_t[i]['net_energy'].sum())
    payments = []
    for index, prosumer in prosumers_at_t[i].iterrows():
        pay = 0
        p = abs(prosumer['net_energy'])
        if abs(prosumer['net_energy']) > 0:
            pay = g(price=max_price,p=p,tc=tc,tp=tp,n=1,a=1000000)
        else:
            # payment is negative to show debt by consumer
            pay = -(h(price=min_price,c=prosumer['consumption'],tc=tc,tp=tp))
        payments.append(pay)
    prosumers_at_t[i]['nrg_cash'] = payments

#view the sample at time 0
prosumers_at_t[0]
```

Out[109…

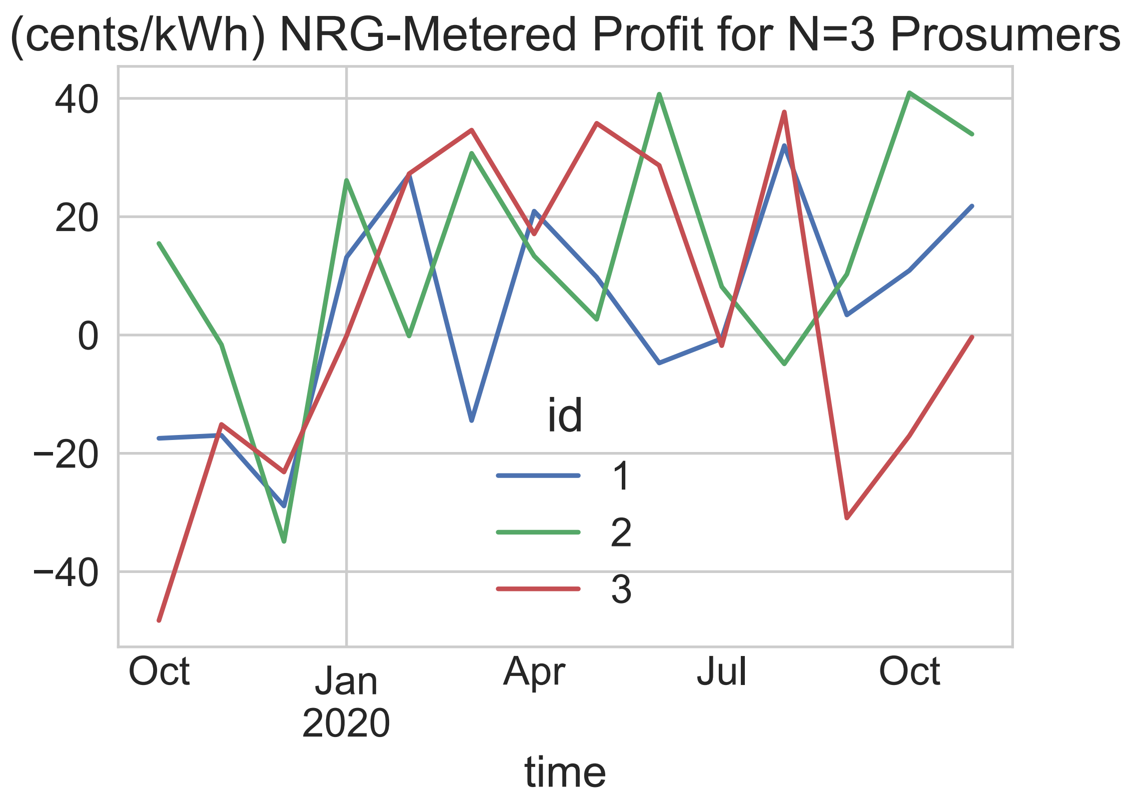
	time	demand	generation	consumption	net_energy	price	id	nrg_cash
13	2019-10-01	785.747309	567.176089	218.571220	0.000000	11.66	1	-17.458161
27	2019-10-01	706.638381	897.462539	0.000000	-190.824158	11.66	2	15.475455
41	2019-10-01	996.729985	392.570712	604.159273	0.000000	11.66	3	-48.256628

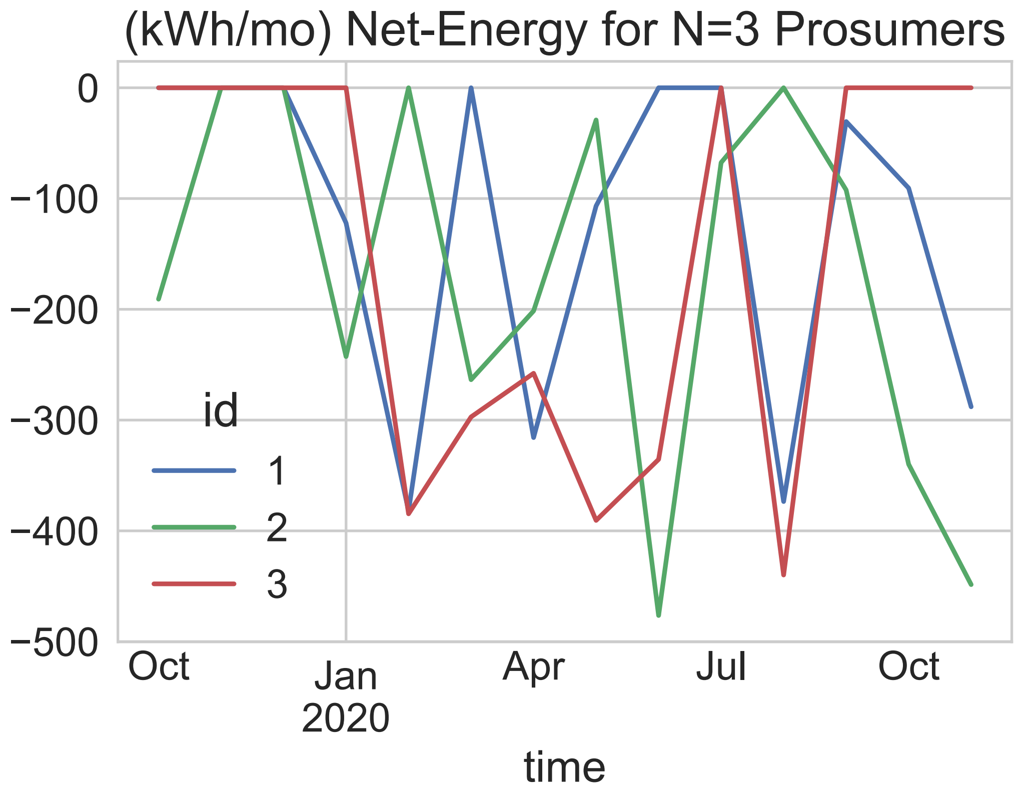
In [110…

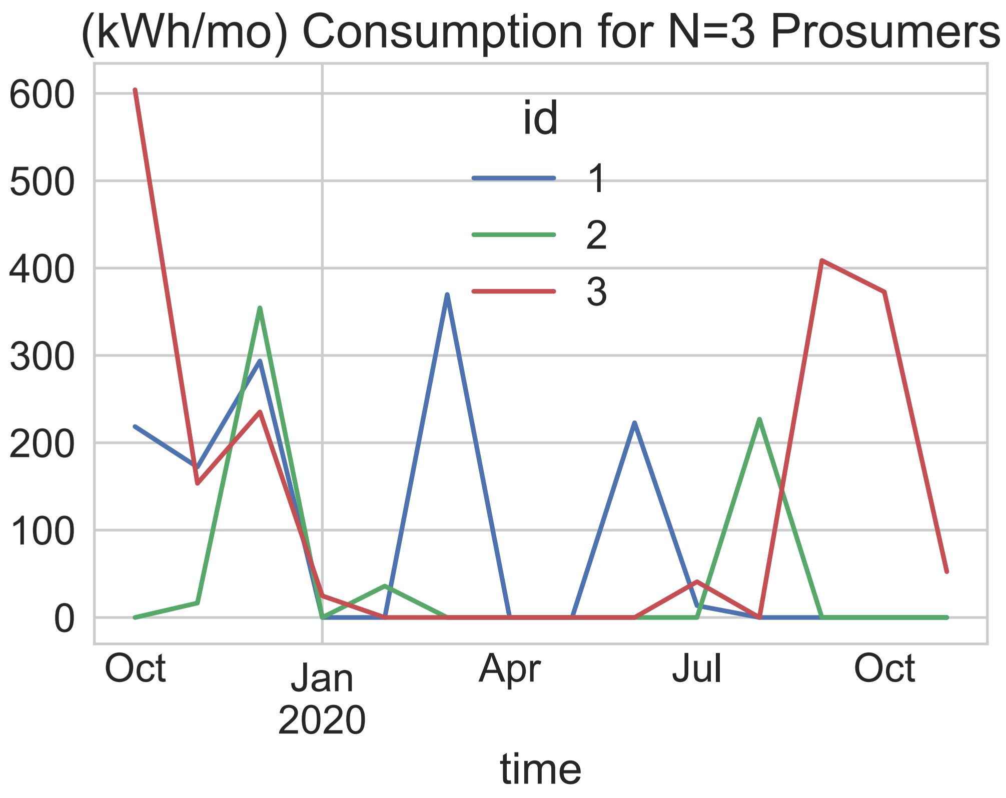
```
#Plot the trends for all prosumers.
df = pd.concat(prosumers_at_t)
df = df.pivot(index='time', columns='id', values='nrg_cash')
df.plot(title=f"(cents/kWh) NRG-Metered Profit for N=3 Prosumers")
df = pd.concat(prosumers_at_t)
df = df.pivot(index='time', columns='id', values='net_energy')
df.plot(title=f"(kWh/mo) Net-Energy for N=3 Prosumers")
df = pd.concat(prosumers_at_t)
df = df.pivot(index='time', columns='id', values='consumption')
df.plot(title=f"(kWh/mo) Consumption for N=3 Prosumers")
```

Out[110…

```
<matplotlib.axes._subplots.AxesSubplot at 0x12a588dd8>
```


(cents/kWh) NRG-Metered Profit for N=3 Prosumers
40
20
0
−20
−40
id
1
2
3
Oct
Jan
2020
Apr
Jul
Oct
time


(kWh/mo) Net-Energy for N=3 Prosumers
0
−100
−200
−300
−400
−500
Oct
Jan
2020
Apr
Jul
Oct
time
id
1
2
3



At every settlement period (monthly) we will determine if more than one prosumer is providing net_energy to the grid. If there is a situation like that then the group of prosumers can participate in a coalitional payout instead of just the NRGX-change payout. The idea would be that the NRGX-Change will see two prosumers as a single prosumer instead of two individuals.

We would need to modify the dataset to combine prosumers at the times that more than one is generating. Then we could re-apply NRG payouts and use the marginal contribution to determine the benefit of coalitional payout.

Modify dataset to merge into coalitional groups.

In [111…

```
coalitional_prosumers_at_t = []
for t in prosumers_at_t:
    p = t.loc[abs(t['net_energy']) > 0]
    p['coalitional_consumption'] = t['consumption'].sum()
    if len(p) > 1:
        coalitional_prosumers_at_t.append(p)

print(coalitional_prosumers_at_t[0])
```

```
         time      demand  generation  consumption  net_energy  price  id  \
10 2020-01-01  666.182080  788.227717          0.0 -122.045637  11.73   1   
24 2020-01-01  637.803627  880.472852          0.0 -242.669225  11.73   2   

     nrg_cash  coalitional_consumption  
10  13.144839                24.752677  
24  26.136518                24.752677  
```

In [112…

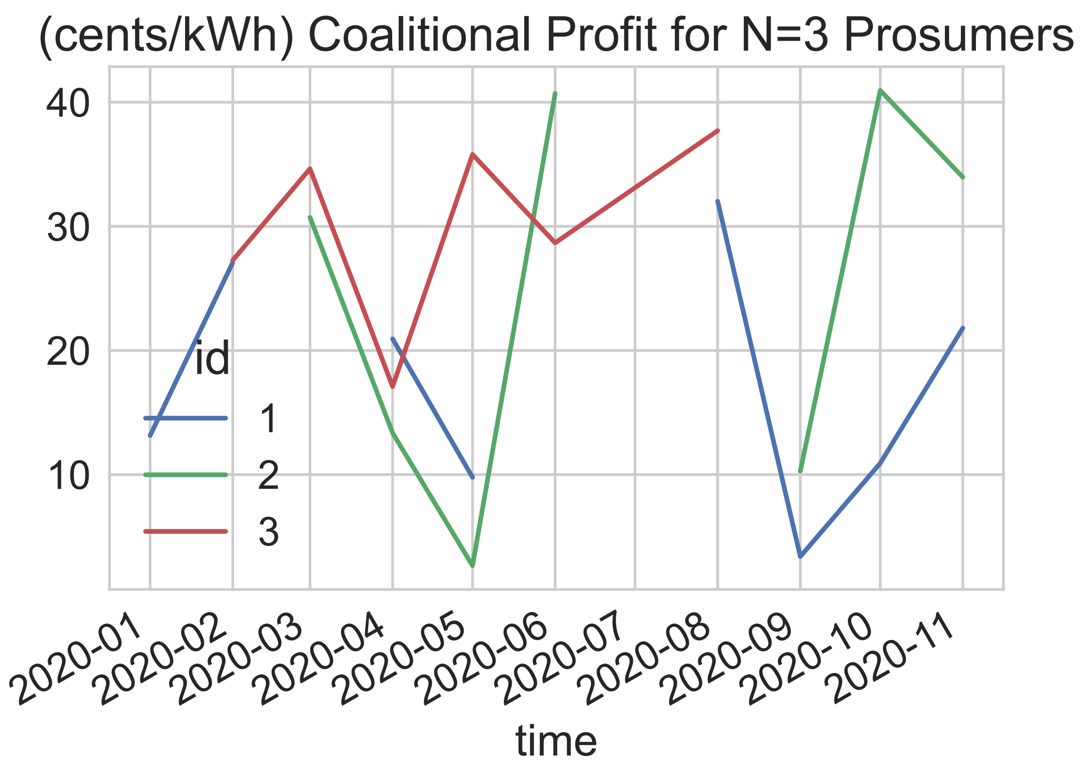
```
for t in coalitional_prosumers_at_t:
    t['coalition_energy'] = abs(t['net_energy']).sum()
    n=len(t['id'])
    tc = t['coalitional_consumption'].mean()
    if n ==2 :
        v1 = t['nrg_cash'].iloc[0]
        v2 = t['nrg_cash'].iloc[1]
        p = abs(t['net_energy']).sum()
        v12 = g(price=max_price,p=p,tc=tc,tp=p,n=1,a=1000000)
        vals = get_shapley(n,[v1,v2,v12])

    if n==3 :
        v1 = t['nrg_cash'].iloc[0]
        v2 = t['nrg_cash'].iloc[1]
        v3 = t['nrg_cash'].iloc[2]
        v12p = abs(t.loc[t['id'].isin([1,2])]['net_energy']).sum()
        v12 = g(price=max_price,p=v12p,tc=tc,tp=p,n=1,a=1000000)
        v13p = abs(t.loc[t['id'].isin([1,3])]['net_energy']).sum()
        v13 = g(price=max_price,p=v13p,tc=tc,tp=p,n=1,a=1000000)
        v23p = abs(t.loc[t['id'].isin([2,3])]['net_energy']).sum()
        v23 = g(price=max_price,p=v23p,tc=tc,tp=p,n=1,a=1000000)
        p = abs(t['net_energy']).sum()
        v123 = g(price=max_price,p=p,tc=tc,tp=p,n=1,a=1000000)
        get_shapley(n,[v1,v2,v3,v12,v13,v23,v123])


print(coalitional_prosumers_at_t[0])
```

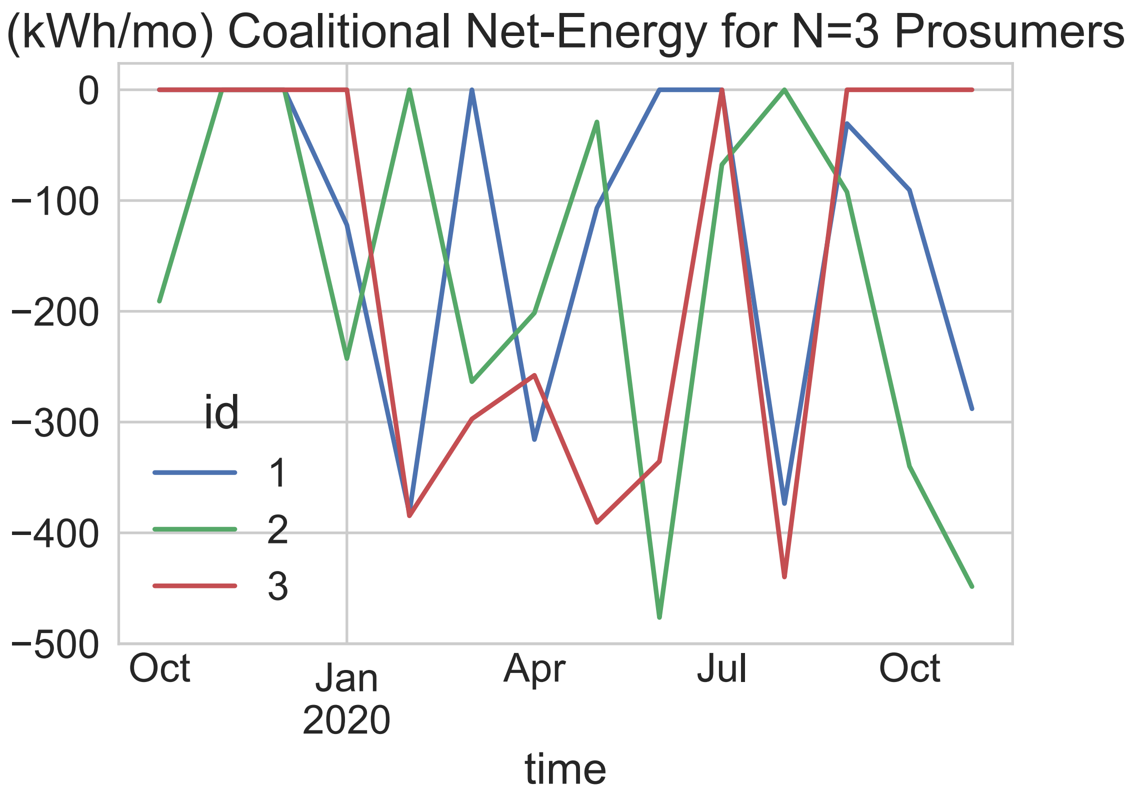
```
         time      demand  generation  consumption  net_energy  price  id  \
10 2020-01-01  666.182080  788.227717          0.0 -122.045637  11.73   1   
24 2020-01-01  637.803627  880.472852          0.0 -242.669225  11.73   2   

     nrg_cash  coalitional_consumption  coalition_energy  
10  13.144839                24.752677        364.714861  
24  26.136518                24.752677        364.714861  
```

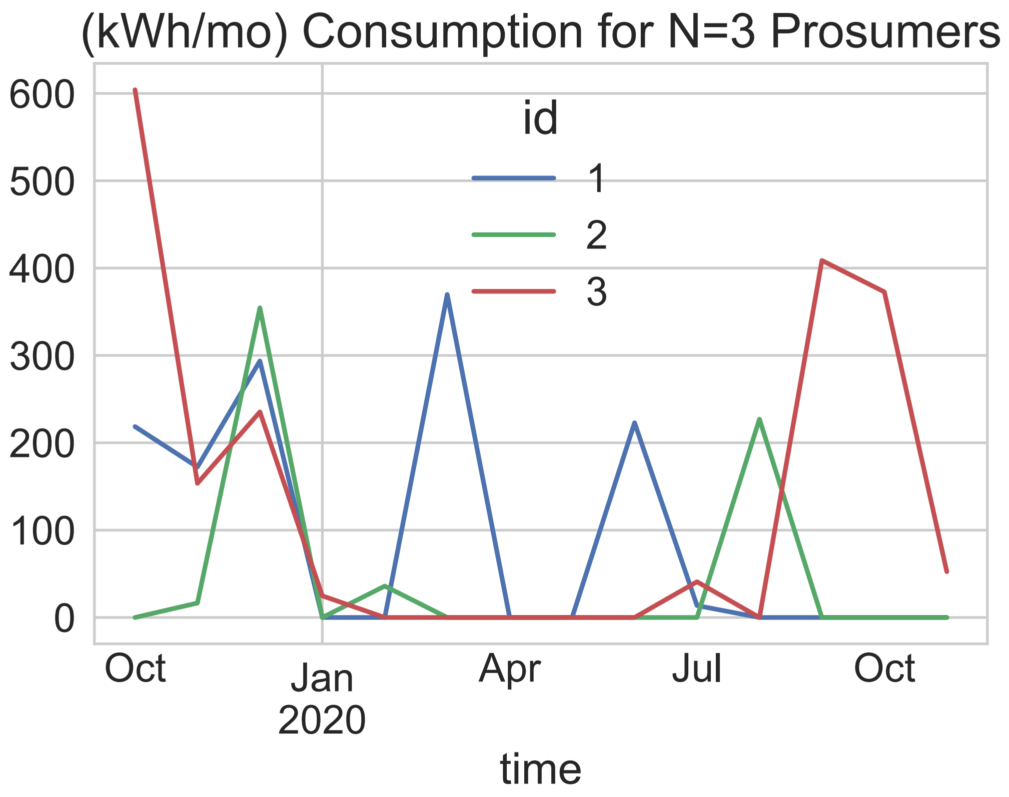
In [113…

```
#Plot the trends for all prosumers.
df = pd.concat(coalitional_prosumers_at_t)
df = df.pivot(index='time', columns='id', values='nrg_cash')
df.plot(title=f"(cents/kWh) Coalitional Profit for N=3 Prosumers")
df = pd.concat(prosumers_at_t)
df = df.pivot(index='time', columns='id', values='net_energy')
df.plot(title=f"(kWh/mo) Coalitional Net-Energy for N=3 Prosumers")
df = pd.concat(prosumers_at_t)
df = df.pivot(index='time', columns='id', values='consumption')
df.plot(title=f"(kWh/mo) Consumption for N=3 Prosumers")
```

```
Out[113… <matplotlib.axes._subplots.AxesSubplot at 0x1293ecf28>
```




(kWh/mo) Coalitional Net-Energy for N=3 Prosumers
0
−100
−200
−300
−400
−500
id
1
2
3
Oct
Jan
2020
Apr
Jul
Oct
time


(kWh/mo) Consumption for N=3 Prosumers
600
500
400
300
200
100
0
id
1
2
3
Oct
Jan
2020
Apr
Jul
Oct
time

```
In [ ]:
```