Simulate Prosumers in a NRG-X-Change Connected Market

Description:

Prosumers are collected as a model with the consumption, generation, net energy and pricing data avaialble. The prosumer data has been built based on Tier1 installations for the purposes of modeling a typical prosumer. The limits on the capacity for Teir1 are also considered. In this simulation parameters for the gross power ratings, AC/DC conversion, will be considered as well as the generation limits.

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 and billed in near real-time." (Mihail Mihaylov)

Steps:

1. Data Gathering

2. Build the NRG-X-Change Simulation

3. Analyze Results

4. Summary and Further Analysis

In [1]:

```
#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 math
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
import seaborn as sns
```

Step 1 : Data Gathering

1.1 Collecting Data From Local Dataset

The prosumer data has been built into a Comma-Sperated file that can be parsed with the following format:

time	demand	generation	consumption	net_energy	price	id
2020-11-01 00:00:00	621.143	1115.12	0	-493.98	12	1
2020-10-01 00:00:00	1110.06	1178.87	0	-68.8135	11.49	1
2020-09-01 00:00:00	1226.16	1026.59	199.568	0	11.97	1
2020-08-01 00:00:00	1002.64	972.796	29.8479	0	11.61	1

...(continued)

Parsing the dataset by the 'id' col. allows us to collect individual prosumer data. The next step is to put the prosumer data into an object class that contains some of the functionality we will perform on the data.

In [7]:

```
#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='
all_prosumers_data.sort_values(by='time')
prosumer_data_by_id = [pd.DataFrame(y) for x, y in all_prosumers_data.groupby('i
N=len(prosumer_data_by_id)
prosumers_at_t = [pd.DataFrame(y) for x, y in all_prosumers_data.groupby('time',

#print(prosumer_data[0]) #show the first prosumer in the dataset
print(all_prosumers_data)
```

```
          time       demand   generation  consumption  net_energy  price  id
0   2020-11-01   713.636892  1001.611236     0.000000 -287.974344  12.00   1
1   2020-10-01  1033.715254  1124.283176     0.000000  -90.567922  11.49   1
2   2020-09-01  1169.504690  1200.000000     0.000000  -30.495310  11.97   1
3   2020-08-01   745.056436  1118.585765     0.000000 -373.529328  11.61   1
4   2020-07-01  1088.967130  1075.337671    13.629459    0.000000  11.71   1
5   2020-06-01  1202.880127   979.940026   222.940101    0.000000  11.53   1
6   2020-05-01   839.217153   945.914118     0.000000 -106.696964   9.84   1
7   2020-04-01   726.229347  1042.016842     0.000000 -315.787495  11.71   1
8   2020-03-01  1136.909963   767.108299   369.801664    0.000000  11.64   1
9   2020-02-01   405.862473   787.888820     0.000000 -382.026347  11.76   1
10  2020-01-01   666.182080   788.227717     0.000000 -122.045637  11.73   1
11  2019-12-01   785.346708   491.636087   293.710621    0.000000  11.62   1
12  2019-11-01   638.907999   466.653885   172.254114    0.000000  12.09   1
13  2019-10-01   785.747309   567.176089   218.571220    0.000000  11.66   1
14  2020-11-01   751.491540  1200.000000     0.000000 -448.508460  12.00   2
15  2020-10-01   762.847690  1102.706306     0.000000 -339.858616  11.49   2
16  2020-09-01   985.274160  1077.522948     0.000000  -92.248789  11.97   2
17  2020-08-01  1231.715100  1004.639607   227.075492    0.000000  11.61   2
18  2020-07-01  1132.463515  1200.000000     0.000000  -67.536485  11.71   2
19  2020-06-01   723.598878  1200.000000     0.000000 -476.401122  11.53   2
20  2020-05-01   935.236513   964.290192     0.000000  -29.053679   9.84   2
21  2020-04-01   686.698848   888.174155     0.000000 -201.475307  11.71   2
22  2020-03-01   650.421040   913.950349     0.000000 -263.529309  11.64   2
23  2020-02-01   754.921915   718.946917    35.974998    0.000000  11.76   2
24  2020-01-01   637.803627   880.472852     0.000000 -242.669225  11.73   2
25  2019-12-01   793.764068   439.250888   354.513179    0.000000  11.62   2
26  2019-11-01   638.090831   621.538424    16.552407    0.000000  12.09   2
```

27	2019-10-01	706.638381	897.462539	0.000000	-190.824158	11.66	2
28	2020-11-01	783.506280	731.021485	52.484795	0.000000	12.00	3
29	2020-10-01	1256.561563	883.816246	372.745318	0.000000	11.49	3
30	2020-09-01	1338.021939	929.304281	408.717657	0.000000	11.97	3
31	2020-08-01	737.868197	1177.708923	0.000000	-439.840726	11.61	3
32	2020-07-01	732.703553	691.806280	40.897274	0.000000	11.71	3
33	2020-06-01	845.578525	1181.059231	0.000000	-335.480705	11.53	3
34	2020-05-01	809.429409	1200.000000	0.000000	-390.570591	9.84	3
35	2020-04-01	942.248731	1200.000000	0.000000	-257.751269	11.71	3
36	2020-03-01	648.239619	945.389520	0.000000	-297.149901	11.64	3
37	2020-02-01	715.936170	1100.551040	0.000000	-384.614870	11.76	3
38	2020-01-01	687.859185	663.106509	24.752677	0.000000	11.73	3
39	2019-12-01	917.458586	682.124809	235.333777	0.000000	11.62	3
40	2019-11-01	818.159911	664.579529	153.580382	0.000000	12.09	3
41	2019-10-01	996.729985	392.570712	604.159273	0.000000	11.66	3

2. Build the NRG-X-Change Simulation

2.1 Define the Energy Pay-Out Function, g(.)

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

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. The NRG exchange cost mechanisim includes a wheighted distribution to adjust for the need of energy consumption when the demand is the highest and drops the pricing when demand is the lowest. We will use the largest value of our pricing history to determine the q parameter. The maximum payout for the price of electricity over the course of the year can vary so it is taken as a historical maximum and assumed to continue in the future based on regulatory rate cases.

In [8]:

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

2.2 Define the Energy Cost Function, h(.)

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

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 [9]:

```
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
```

2.3 Generate Market Results

In [10]:

```
#split prosumers by time slice
#prosumer_data_by_id[0]
prosumers_at_t[0]
```

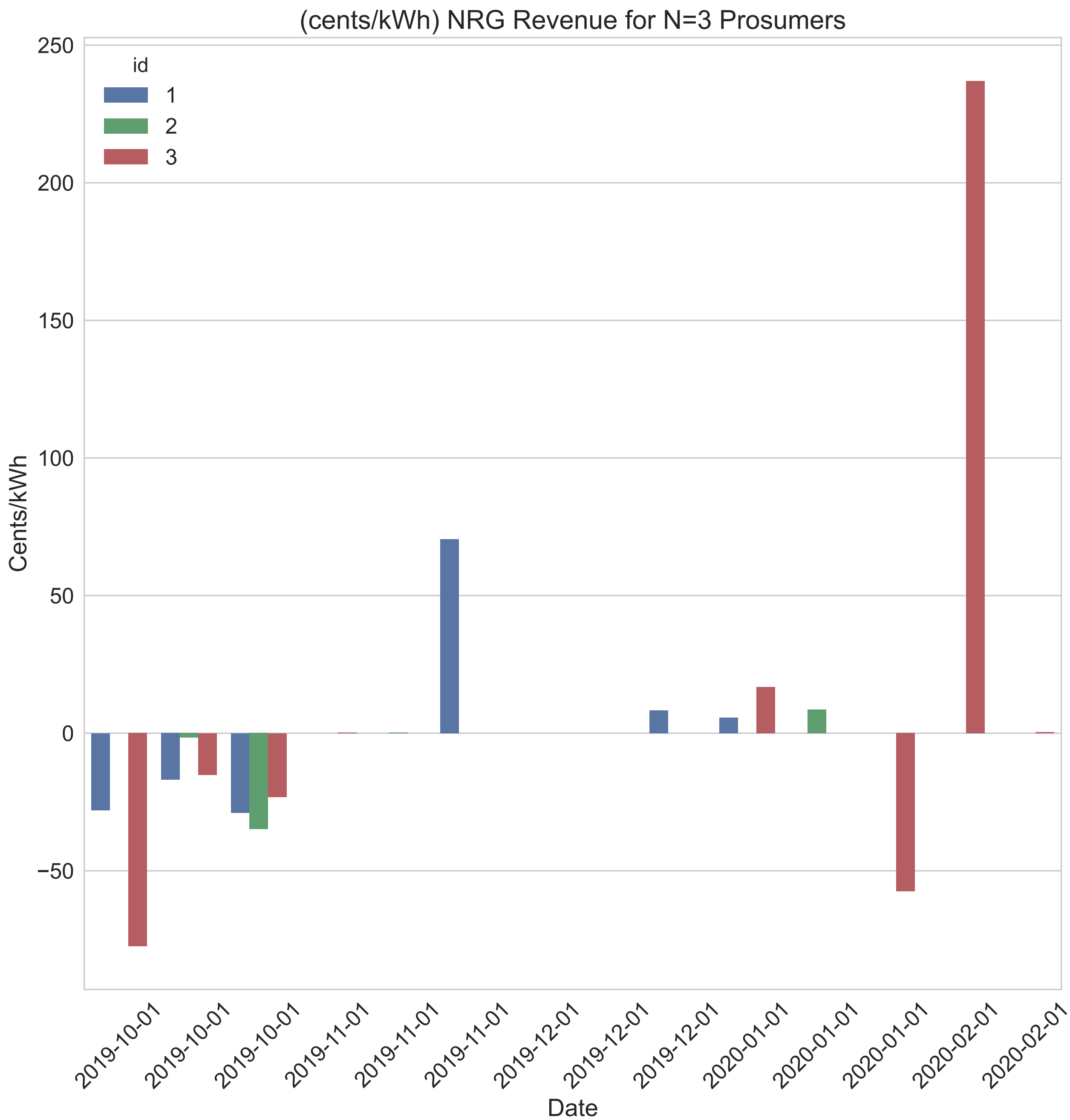
Out[10]:

	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

In [11]:

```
#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 = prosumers_at_t[i]['net_energy'].sum()
    k = 0
    for index, prosumer in prosumers_at_t[i].iterrows():
        pay = 0
        if prosumer['net_energy'] > 0:
            pay = g(price=max_price,p=prosumer['net_energy'],tc=tc,tp=tp)
        else:
            # payment is negative to show debt by consumer
            pay = -(h(price=min_price,c=prosumer['consumption'],tc=tc,tp=tp))
        time = prosumers_at_t[i]['time'].values[k]
        id = prosumers_at_t[i]['id'].values[k]
        payments.append([time,id,pay])
        k=k+1

#print(payments_df)
payments_df = pd.DataFrame(payments,columns=['time','id','nrg_pay'])
payments_df['time'] = pd.to_datetime(payments_df['time'], format='%Y-%m-%d')

# Use white grid plot background from seaborn
sns.set(font_scale=1.5, style="whitegrid")
fig, ax = plt.subplots(figsize=(12, 12))
```

```python
# Set title and labels for axes
ax = sns.barplot(x="time", y="nrg_pay", hue="id", data=payments_df)
ax.set(xlabel="Date",
       ylabel="Cents/kWh",
       title=f"(cents/kWh) NRG Revenue for N={N} Prosumers")
ax.xaxis_date()


# Ensure a major tick for each week using (interval=1)
#ax.xaxis.set_major_locator(mdates.MonthLocator(interval=1))
ax.xaxis.set_major_formatter(DateFormatter('%Y-%m-%d'))
ax.set_xticklabels([pd.to_datetime(tm).strftime('%Y-%m-%d') for tm in payments_d
plt.xticks(rotation=45)
plt.show()
```



Step 3: Analysis of Performance

2.1 Analyze the Performance by Prosumer

Durng an NRG market teh prosumers where achieving the greatest returns when genreation was the lowest.

2.3 Analysis of the Market Performance for 100 Prosumers

To understand the scalability of the market, we will simulate N=100 prosumers. The charts will show the same volatility curves and calcualte the revenue, from the profit and expenses during the demand and generation curves.

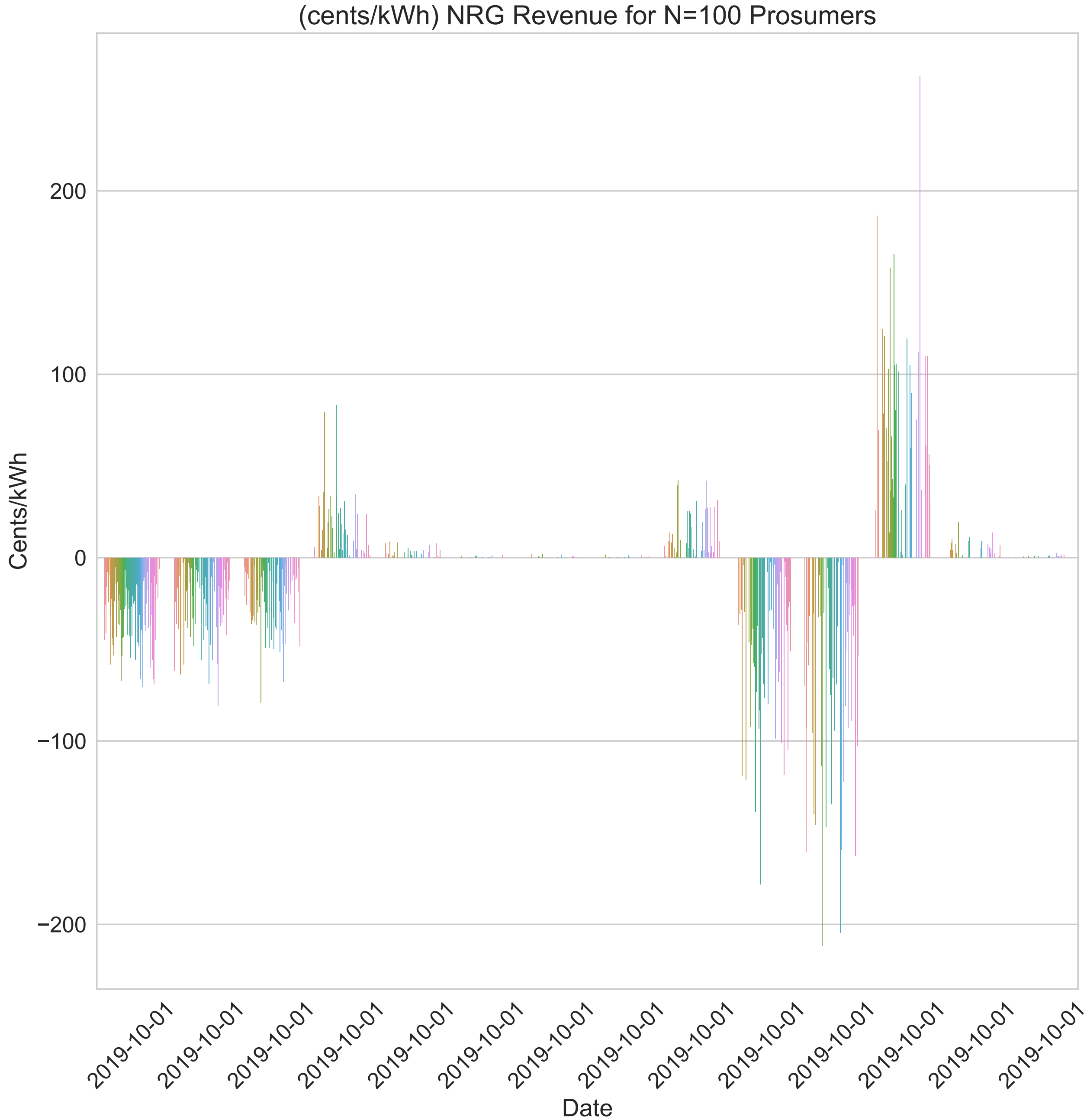
In [15]:

```
#Import the data to local csv
all_prosumers_data = pd.read_csv('data/prosumer_N100_model_20210129_1414.csv')
all_prosumers_data["time"] = pd.to_datetime(all_prosumers_data['time'], format='
all_prosumers_data.sort_values(by='time')
prosumer_data_by_id = [pd.DataFrame(y) for x, y in all_prosumers_data.groupby('i
N=len(prosumer_data_by_id)
prosumers_at_t = [pd.DataFrame(y) for x, y in all_prosumers_data.groupby('time',

#print(prosumer_data[0]) #show the first prosumer in the dataset
print(all_prosumers_data)

#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 = prosumers_at_t[i]['net_energy'].sum()
    k = 0
    for index, prosumer in prosumers_at_t[i].iterrows():
        pay = 0
        if prosumer['net_energy'] > 0:
            pay = g(price=max_price,p=prosumer['net_energy'],tc=tc,tp=tp)
        else:
            # payment is negative to show debt by consumer
            pay = -(h(price=min_price,c=prosumer['consumption'],tc=tc,tp=tp))
        time = prosumers_at_t[i]['time'].values[k]
        id = prosumers_at_t[i]['id'].values[k]
        payments.append([time,id,pay])
        k=k+1

#print(payments_df)
payments_df = pd.DataFrame(payments,columns=['time','id','nrg_pay'])
payments_df['time'] = pd.to_datetime(payments_df['time'], format='%Y-%m-%d')
print(payments_df)
```

```
         time       demand   generation  consumption  net_energy  price   id
0  2020-11-01   928.558449  1061.810263     0.000000 -133.251814  12.00    1
1  2020-10-01   952.496370   945.671067     6.825303    0.000000  11.49    1
2  2020-09-01   953.234234  1200.000000     0.000000 -246.765766  11.97    1
3  2020-08-01  1135.378044   901.152036   234.226009    0.000000  11.61    1
```

```
4     2020-07-01   648.729055  1074.617534    0.000000 -425.888479  11.71    1
5     2020-06-01  1253.202416  1189.318132   63.884284    0.000000  11.53    1
6     2020-05-01   775.415108  1139.274230    0.000000 -363.859121   9.84    1
7     2020-04-01   717.336216   868.937997    0.000000 -151.601781  11.71    1
8     2020-03-01   690.381161  1200.000000    0.000000 -509.618839  11.64    1
9     2020-02-01   638.943695   787.545678    0.000000 -148.601983  11.76    1
10    2020-01-01   696.291964   652.057768   44.234197    0.000000  11.73    1
11    2019-12-01   674.696334   640.294779   34.401554    0.000000  11.62    1
12    2019-11-01  1032.149550   546.697488  485.452062    0.000000  12.09    1
13    2019-10-01   887.985834   719.246104  168.739730    0.000000  11.66    1
14    2020-11-01   721.998741   667.913049   54.085691    0.000000  12.00    2
15    2020-10-01   974.749027  1114.995213    0.000000 -140.246185  11.49    2
16    2020-09-01  1274.048466  1200.000000   74.048466    0.000000  11.97    2
17    2020-08-01   692.662916  1200.000000    0.000000 -507.337084  11.61    2
18    2020-07-01   940.482185  1200.000000    0.000000 -259.517815  11.71    2
19    2020-06-01  1193.963915  1188.613887    5.350028    0.000000  11.53    2
20    2020-05-01  1160.926628  1200.000000    0.000000  -39.073372   9.84    2
21    2020-04-01   816.605506  1200.000000    0.000000 -383.394494  11.71    2
22    2020-03-01   730.403562   913.333870    0.000000 -182.930308  11.64    2
23    2020-02-01   747.774449   868.492503    0.000000 -120.718054  11.76    2
24    2020-01-01   586.926927   629.018467    0.000000  -42.091540  11.73    2
25    2019-12-01   545.518063   396.127355  149.390708    0.000000  11.62    2
26    2019-11-01   765.875227   623.786950  142.088277    0.000000  12.09    2
27    2019-10-01  1034.704050   593.269981  441.434068    0.000000  11.66    2
28    2020-11-01   604.981720   888.988688    0.000000 -284.006968  12.00    3
29    2020-10-01   930.967344  1190.899825    0.000000 -259.932481  11.49    3
...          ...          ...          ...         ...         ...    ...  ...
1370 2019-11-01   709.447670   570.010128  139.437542    0.000000  12.09   98
1371 2019-10-01   762.373969   830.003934    0.000000  -67.629965  11.66   98
1372 2020-11-01   514.236791  1176.090012    0.000000 -661.853221  12.00   99
1373 2020-10-01  1015.581310   785.477408  230.103902    0.000000  11.49   99
1374 2020-09-01  1285.389784  1200.000000   85.389784    0.000000  11.97   99
1375 2020-08-01  1025.332968  1200.000000    0.000000 -174.667032  11.61   99
1376 2020-07-01   781.929030   701.218027   80.711003    0.000000  11.71   99
1377 2020-06-01   859.199950  1200.000000    0.000000 -340.800050  11.53   99
1378 2020-05-01   644.116139  1200.000000    0.000000 -555.883861   9.84   99
1379 2020-04-01   777.500348  1175.079684    0.000000 -397.579336  11.71   99
1380 2020-03-01   531.493428   992.057616    0.000000 -460.564189  11.64   99
1381 2020-02-01   778.250313   758.751752   19.498562    0.000000  11.76   99
1382 2020-01-01   755.890988   837.472767    0.000000  -81.581779  11.73   99
1383 2019-12-01   460.098778   594.467462    0.000000 -134.368684  11.62   99
1384 2019-11-01   787.298847   824.722009    0.000000  -37.423162  12.09   99
1385 2019-10-01   859.277800   872.884662    0.000000  -13.606862  11.66   99
1386 2020-11-01   795.095056  1146.659698    0.000000 -351.564642  12.00  100
1387 2020-10-01   883.494737  1120.568418    0.000000 -237.073682  11.49  100
1388 2020-09-01   956.560819  1200.000000    0.000000 -243.439181  11.97  100
1389 2020-08-01   690.167053  1064.871652    0.000000 -374.704599  11.61  100
1390 2020-07-01  1056.994041   885.642183  171.351859    0.000000  11.71  100
1391 2020-06-01   931.927599  1200.000000    0.000000 -268.072401  11.53  100
1392 2020-05-01  1199.296961  1200.000000    0.000000   -0.703039   9.84  100
1393 2020-04-01   790.230195  1200.000000    0.000000 -409.769805  11.71  100
1394 2020-03-01   528.691942  1096.088186    0.000000 -567.396244  11.64  100
1395 2020-02-01   633.236470   470.214966  163.021505    0.000000  11.76  100
1396 2020-01-01   817.482665   809.957808    7.524857    0.000000  11.73  100
1397 2019-12-01   799.119400   449.548868  349.570532    0.000000  11.62  100
1398 2019-11-01   613.511616   516.360955   97.150660    0.000000  12.09  100
1399 2019-10-01   697.072000   638.602892   58.469108    0.000000  11.66  100

[1400 rows x 7 columns]
           time   id    nrg_pay
0    2019-10-01    1 -17.103135
1    2019-10-01    2 -44.742911
2    2019-10-01    3 -25.525120
3    2019-10-01    4 -41.269983
4    2019-10-01    5  -4.823585
```

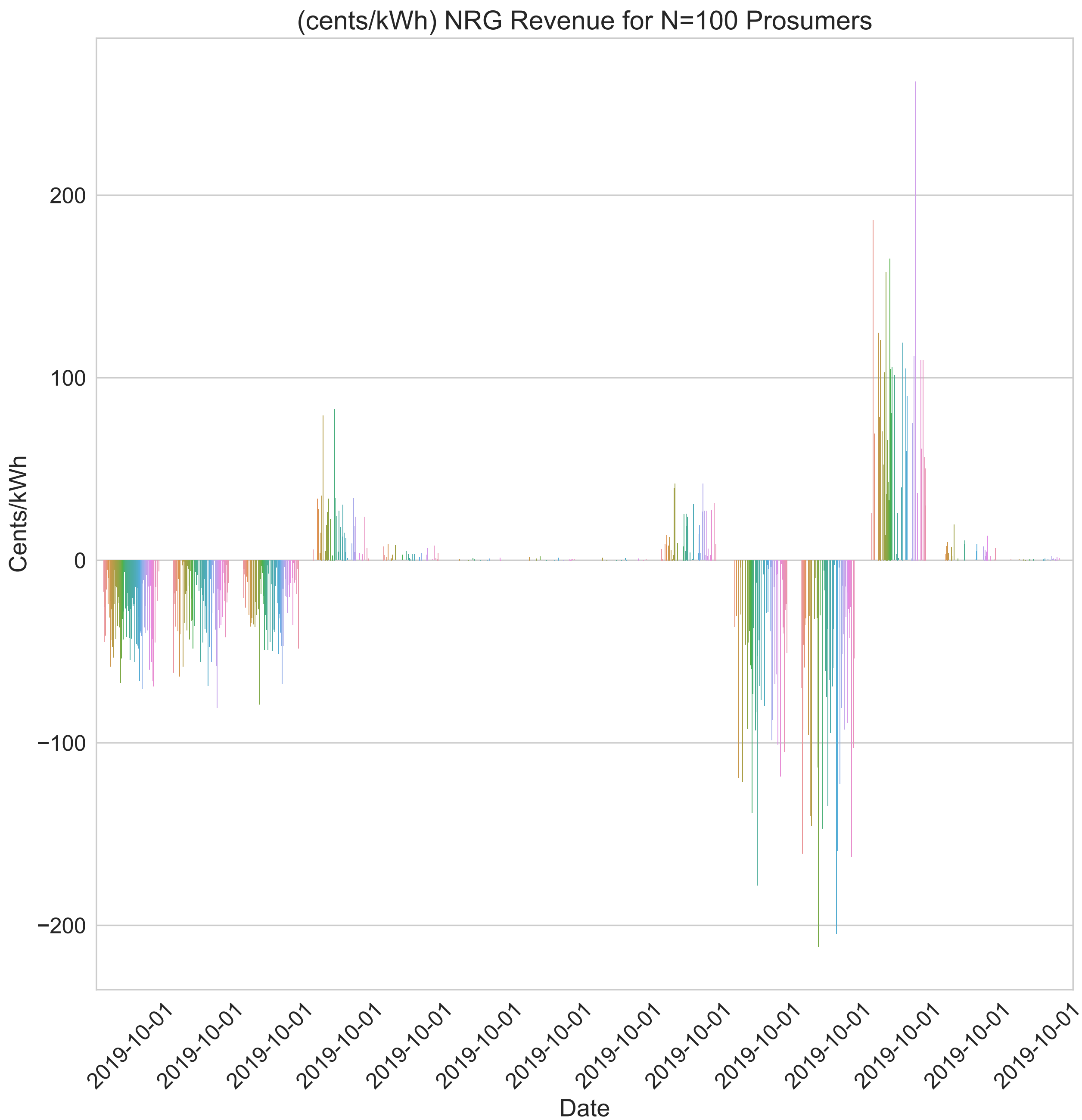
```
5    2019-10-01    6 -15.975918
6    2019-10-01    7  -7.822187
7    2019-10-01    8  -5.029514
8    2019-10-01    9 -23.921344
9    2019-10-01   10  -9.591743
10   2019-10-01   11  -0.000000
11   2019-10-01   12 -31.345202
12   2019-10-01   13 -58.162983
13   2019-10-01   14 -22.259031
14   2019-10-01   15 -26.908223
15   2019-10-01   16 -43.651398
16   2019-10-01   17 -47.528397
17   2019-10-01   18 -53.213327
18   2019-10-01   19  -2.414586
19   2019-10-01   20 -23.741683
20   2019-10-01   21  -4.986783
21   2019-10-01   22 -14.118163
22   2019-10-01   23 -43.045143
23   2019-10-01   24 -14.839750
24   2019-10-01   25 -13.135809
25   2019-10-01   26 -35.984835
26   2019-10-01   27 -23.275874
27   2019-10-01   28 -19.987208
28   2019-10-01   29 -36.702079
29   2019-10-01   30 -28.361928
...         ...  ...        ...
1370 2020-11-01   71   0.000000
1371 2020-11-01   72   0.095687
1372 2020-11-01   73   0.000000
1373 2020-11-01   74   0.000000
1374 2020-11-01   75   2.258042
1375 2020-11-01   76   0.698954
1376 2020-11-01   77   0.000000
1377 2020-11-01   78   0.922035
1378 2020-11-01   79   0.000000
1379 2020-11-01   80   0.000000
1380 2020-11-01   81   0.764319
1381 2020-11-01   82   0.000000
1382 2020-11-01   83   0.000000
1383 2020-11-01   84   1.681010
1384 2020-11-01   85   0.000000
1385 2020-11-01   86   0.000000
1386 2020-11-01   87   0.000000
1387 2020-11-01   88   1.130290
1388 2020-11-01   89   0.000000
1389 2020-11-01   90   0.000000
1390 2020-11-01   91   0.000000
1391 2020-11-01   92   0.000000
1392 2020-11-01   93   0.000000
1393 2020-11-01   94   0.000000
1394 2020-11-01   95   0.000000
1395 2020-11-01   96   0.000000
1396 2020-11-01   97   0.000000
1397 2020-11-01   98   0.000000
1398 2020-11-01   99   0.000000
1399 2020-11-01  100   0.000000

[1400 rows x 3 columns]
```



In [16]:

```
# Use white grid plot background from seaborn
sns.set(font_scale=1.5, style="whitegrid")
fig, ax = plt.subplots(figsize=(12, 12))

# Set title and labels for axes
ax = sns.barplot(x="time", y="nrg_pay", hue="id", data=payments_df)
ax.set(xlabel="Date",
       ylabel="Cents/kWh",
       title=f"(cents/kWh) NRG Revenue for N={N} Prosumers")
ax.xaxis_date()


# Ensure a major tick for each week using (interval=1)
#ax.xaxis.set_major_locator(mdates.MonthLocator(interval=1))
ax.xaxis.set_major_formatter(DateFormatter('%Y-%m-%d'))
ax.set_xticklabels([pd.to_datetime(tm).strftime('%Y-%m-%d') for tm in payments_d
plt.xticks(rotation=45)
```

```
plt.legend('')
plt.show()
```



Step 4: Summary and Further Studies

TBD