W14_DataPreprocessing-Copy7

January 12, 2021

1 Complete DataPreprocessing for the final models

Last update: 17/12 >Jefry el Bhwash 16095065

Purpose: Using by the team and mostly self-written code to create a uniform notebook to do the datapreprocessing in one place.

Halfway through, the consumption seemed to be incorrect, that's why there's also some analysis in this notebook

---

This notebook makes use of the python markdown extension

[3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
from IPython.display import display, HTML
```

[4]:
```
#Variables
faster = False #goes around the excel reading and reads the pickle
features = False #adds new features
nearest = True

###############
#housenr = '028'
load_path = '/home/16095065/notebooks/zero/DATAX/'
sheet = ['smartMeter', 'solar']
```

2 Loading the original Data and adding a datetime

[5]:
```
houses = [28]
for housenr in houses:
    housenr = f'{housenr:03}'
    #smartMeter
```

```
    df_0 = pd.read_excel(load_path + f'{housenr:0>3}' + '.xlsx',␣
↪sheet_name=sheet[0], engine="openpyxl")
    df_0 = df_0.set_index(pd.DatetimeIndex(pd.
↪to_datetime(df_0['Timestamp'],unit='s').values))
    df_0.to_pickle(str(housenr) + 'df_0')

    #solar
    df_1 = pd.read_excel(load_path + f'{housenr:0>3}' + '.xlsx',␣
↪sheet_name=sheet[1], engine="openpyxl")
    df_1 = df_1.set_index(pd.DatetimeIndex(pd.
↪to_datetime(df_1['Timestamp'],unit='s').values))
    df_1.to_pickle(str(housenr) + 'df_1')

    print(f'Dataframe of {sheet[0]} has {len(df_0.columns)} columns and {df_0.
↪shape[0]} rows \nDataframe of {sheet[1]} has {len(df_1.columns)} columns and␣
↪{df_1.shape[0]} rows')
    print(f'Difference in rows: {df_0.shape[0] - df_1.shape[0]} = {round((df_0.
↪shape[0] - df_1.shape[0])/df_0.shape[0]*100, 5)}%')
    display(df_0.tail(2))
    display(df_1.tail(2))

    db_0 = df_0.copy().diff() #create raw data copies
    db_1 = df_1.copy().diff()
    if nearest:
        df_0 = df_0.diff().resample('5min').nearest(limit=1).dropna()␣
↪#resampling to 5 minutes so it can be combined with the other df
        df_1 = df_1.diff().resample('5min').nearest(limit=1).dropna()#
    if not nearest:
        df_0 = df_0.diff().resample('5min').sum().dropna() #resampling to 5␣
↪minutes so it can be combined with the other df
        df_1 = df_1.diff().resample('5min').sum().dropna()#

    #filter
    df_0 = df_0[['total_energy_out', 'total_energy_in']]
    df_1 = df_1[['total_energy_out']]
    #Renaming
    df_0.columns = ['smartMeter_energy_out', 'smartMeter_energy_in']
    df_1.columns = ['solar_energy_out']

    df = df_1.copy()
    df = df.join(df_0, on=df.index, how= 'inner').drop('key_0', axis=1)
    print(f"df 0 rows: {df_0.shape[0]} \ndf 1 rows: {df_1.shape[0]} \ndf rows  :
↪ {df.shape[0]}")
    print(f'\nTotal amount of possible 5 minute segments in this index:␣
↪{365*24*12-2} \nPercentage of that in this combined dataset (df): {round(df.
↪shape[0]/(365*24*12-2)*100, 3)}%')
```

```
    df.head(2)

    df['production'] = df['solar_energy_out']
    df.production.plot(marker='.')
    plt.title(sheet[1])
    plt.xlabel('Date')
    plt.ylabel('Energy [kWh]')
    plt.show()

    df['consumption'] = df['smartMeter_energy_in'] +␣
↪df['solar_energy_out']-df['smartMeter_energy_out']

    df.consumption.plot(marker='.', zorder=1)
    plt.hlines(0, df.index[0], df.index[-1], colors='k')
    plt.title(sheet[0])
    plt.xlabel('Date')
    plt.ylabel('Energy [kWh]')
    plt.show()

    peaklim = 600
    peak0 = db_0['Timestamp']>peaklim
    peak1 = db_1['Timestamp']>peaklim

    mt5m0 = db_0['Timestamp']-(db_0.Timestamp.mean()+2*db_0.Timestamp.std())>0
    mt5m1 = db_1['Timestamp']-(db_1.Timestamp.mean()+2*db_1.Timestamp.std())>0

    display(HTML(f'<h3> Plots </h3>'))
    #Plot 1
    plt.subplot(2,1,1)
    plt.plot(db_0.index, db_0.Timestamp)
    plt.title(f"{sheet[0]}: significant gap dots")

    plt.plot(db_0.Timestamp[mt5m0].index, db_0.Timestamp[mt5m0], 'o',␣
↪color='red')
    plt.xlabel('Date')
    plt.ylabel('Gap in the timestamp [s]')
    plt.tight_layout()
    plt.show()

    #Plot 2
    plt.subplot(2,1,2)
    plt.plot(db_1.index, db_1.Timestamp)
    plt.title(f"{sheet[1]}: significant gap dots")

    plt.plot(db_1.Timestamp[mt5m1].index, db_1.Timestamp[mt5m1], 'o',␣
↪color='red')
    plt.xlabel('Date')
```

```
    plt.ylabel('Gap in the timestamp [s]')
    plt.xlim()


    plt.tight_layout()
    plt.show()

    display(HTML('<hr>'))
    display(HTML(f'<h3> {sheet[0]} </h3>'))

    print(f"Mean of timestamps in {sheet[0]}: {db_0.Timestamp.mean()}\nStandard␣
↪Deviation timestamps of {sheet[0]}: {db_0.Timestamp.std()}\n")
    print(f"The amount of timestamp gaps significantly (2*std) above 5 minutes:␣
↪\t{len(np.where(np.array(mt5m0))[0])}")

    print('SmartMeter peaks:')
    print(f"{sheet[0]} peaks above {peaklim}: {len(np.where(np.
↪array(peak0))[0])} ")
    display(db_0.iloc[np.where(np.array(peak0))])

    display(HTML('<hr>'))
    display(HTML(f'<h3> {sheet[1]} </h3>'))

    print(f"Mean of timestamps in {sheet[1]}: {db_1.Timestamp.mean()}\nStandard␣
↪Deviation timestamps of {sheet[1]}: {db_1.Timestamp.std()}\n")
    print(f"The amount of timestamp gaps significantly (2*std) above 5 minutes:␣
↪\t{len(np.where(np.array(mt5m1))[0])}")


    print('Solar peaks:')
    print(f"{sheet[1]} peaks above {peaklim}: {len(np.where(np.
↪array(peak1))[0])} ")
    display(db_1.iloc[np.where(np.array(peak1))])

    peakdate = '2019-07-21'
    peak0index = db_0.iloc[np.where(np.array(peak0))].loc[peakdate].index
    peak1index = db_1.iloc[np.where(np.array(peak1))].loc[peakdate].index

    plt.figure(figsize=(10,5))

    plt.plot(df.consumption.loc[peakdate].index, df.consumption.loc[peakdate].
↪values, '.-', label='consumption')
    plt.plot(peak0index, df.consumption.truncate(before = peak0index.
↪strftime("%Y/%m/%d %H:%M")[0])[0] , 'o', label='Timestamp difference peak␣
↪smartMeter')
```

```
   plt.plot(peak1index, df.consumption.truncate(after = peak1index.
↪strftime("%Y/%m/%d %H:%M")[0])[-1], 'o', label='Timestamp difference peak␣
↪solar')
   plt.tight_layout()
   plt.xlabel('Date')
   plt.ylabel('Energy [kWh]')
   plt.legend()
   plt.title(f'Consumption of {peakdate} \nTogether with peaks of the␣
↪timestamp in {sheet[0]} and {sheet[1]}')
   plt.show()

   if nearest:
       print(f"Before the peak at {peak1index[0].hour}:00hr the values are␣
↪missing. This will result in 0 consumption at that time when resampling to␣
↪an hour")
   if not nearest:
       print(f"There's a few values 0 before there's a peak at {peak1index[0].
↪hour}:00hr. Those values are 0 because there's a gap of 1,5 hours in the␣
↪original dataset")
   print('The peak is most probably caused by a data freeze, however, the peak␣
↪is not high enough to destroy the models we are using')

   display(df.loc['2019-07-21 18:30':'2019-07-21 20:05'])
   print(f"The 5 minute aggregation method was set to: {'.nearest(limit=1)' if␣
↪nearest else '.mean()'} \nNearest results with a gap in the data, so it␣
↪stays closer to reality than using something like mean or sum")

   _ = df.consumption.plot()
   plt.hlines(-0.010, df.index[0], df.index[-1], colors='green', zorder=5)
   plt.ylim(-0.03,0)

   markers0 = [-0.011 for x in db_0.Timestamp[mt5m0].index]
   markers1 = [-0.011 for x in db_1.Timestamp[mt5m1].index]

   plt.scatter(db_0.Timestamp[mt5m0].index, markers0, marker='.',␣
↪color='orange', alpha=0.2, zorder=4)
   plt.scatter(db_1.Timestamp[mt5m1].index, markers1, marker='.', color='red',␣
↪alpha=0.6, zorder=3)

   plt.title(f'House {housenr} energy consumption below 0 \nwith significant␣
↪gap dots')
   plt.xlabel('Date')
   plt.ylabel('Energy [kWh]')
   plt.tight_layout()
   plt.show()
```

```
    belowzero = df['consumption']<0
    belowzero15 = df['consumption']<-0.01

    print(f"The amount of consumption below 0: \t\t{len(np.where(np.
↪array(belowzero))[0])}")
    print(f"The amount of consumption below -0.010: \t{len(np.where(np.
↪array(belowzero15))[0])}")

    #Consumption lower than 0 is set to 0 for now
    df['consumption'] = df['consumption'].apply(lambda x: 0 if x<0 else x)

    _ = df.consumption.plot(zorder=1)
    plt.hlines(0, df.index[0], df.index[-1], colors='k')
    plt.title(f'House {housenr}: Consumption after processing')
    plt.xlabel('Date')
    plt.ylabel('Energy [kWh]')
    plt.scatter(db_0.Timestamp[mt5m0].index, markers0, marker='.',␣
↪color='orange', alpha=1, zorder=4)
    plt.scatter(db_1.Timestamp[mt5m1].index, markers1, marker='.', color='red',␣
↪alpha=1, zorder=3)
    plt.show()
    print(f"{len(np.where(np.array(belowzero))[0])} values were set to 0")
    print(f"Which is {round(len(np.where(np.array(belowzero))[0])/len(df.
↪index)*100, 2)}% of the dataset")

    df = df[['consumption', 'production']].resample('1H').sum()

    print(f'The area under consumption = {round(df.consumption.sum(), 2)} kWh␣
↪\nThe area under production = {round(df.production.sum(), 2)} kWh')
    _ = df.plot(alpha = 0.5)
    plt.title(f'House {housenr}: Consumption and Production')
    plt.xlabel('Date')
    plt.ylabel('Energy [kWh]')
    plt.show()

    if features:
        #shifted
        def shifting(sf, shift:int):
            sf['cons_T-'+str(shift)] = sf['consumption'].shift(periods=shift,␣
↪freq='H')
            return sf

        temp_df = df.filter(items=['consumption'])
        day_temp_df = df.filter(items=['consumption'])

        shiftDagen = [24, 48, 72, 168]
```

```
        #week
        for i in range(24, 168+1):
            temp_df = shifting(temp_df, i)
        temp_df = temp_df.drop(['consumption'], axis=1)

        #day
        for i in range(24, 48+1):
            day_temp_df = shifting(day_temp_df, i)
        day_temp_df = day_temp_df.drop(['consumption'], axis=1)

        #Shifted days
        for i in shiftDagen:
            df = shifting(df, i)

        #columns added
        df['day_mean'] = day_temp_df.mean(axis=1, skipna=True)
        df['week_mean'] = temp_df.mean(axis=1, skipna=True)

        df = df.fillna(0)
        print('Features were added')
    else:
        print('No Features were added')

    display(df.tail())

    df.to_pickle(f'consumption_production_pickle_{housenr}')
```

```
Dataframe of smartMeter has 8 columns and 105071 rows
Dataframe of solar has 4 columns and 105063 rows
Difference in rows: 8 = 0.00761%
```

	Timestamp	energy_in_low	energy_in_norm	\
2019-12-31 22:55:00	1577832900	3683.150	2306.312	
2019-12-31 23:00:00	1577833200	3683.239	2306.312	

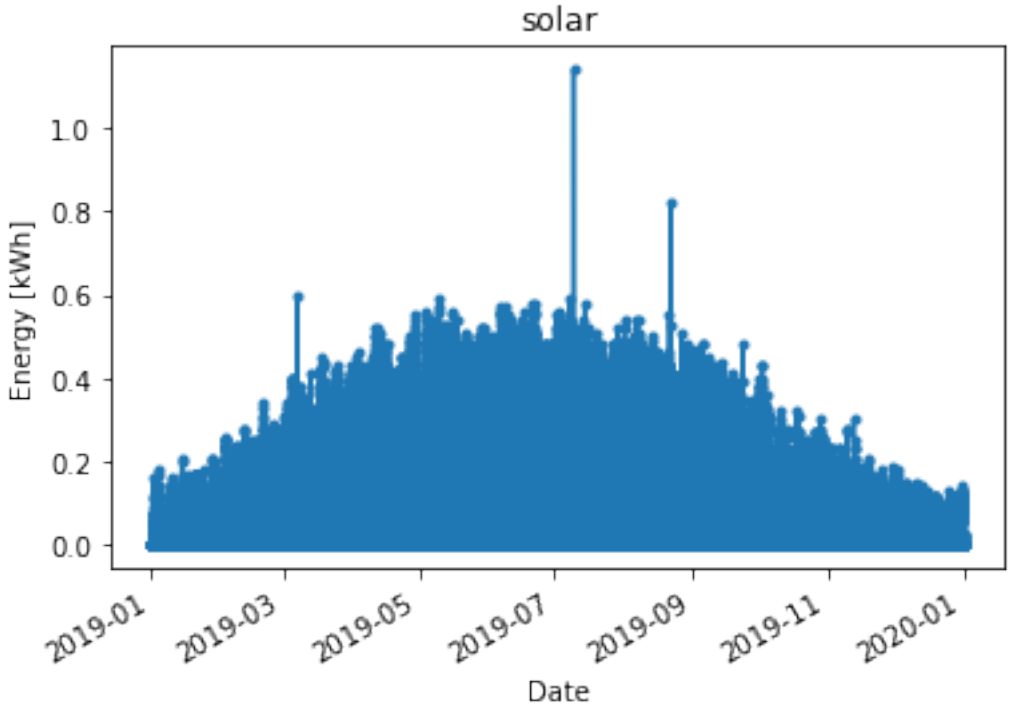
	energy_out_low	energy_out_norm	power	total_energy_in	\
2019-12-31 22:55:00	1728.566	4252.418	1091.0	5989.462	
2019-12-31 23:00:00	1728.566	4252.418	851.0	5989.551	

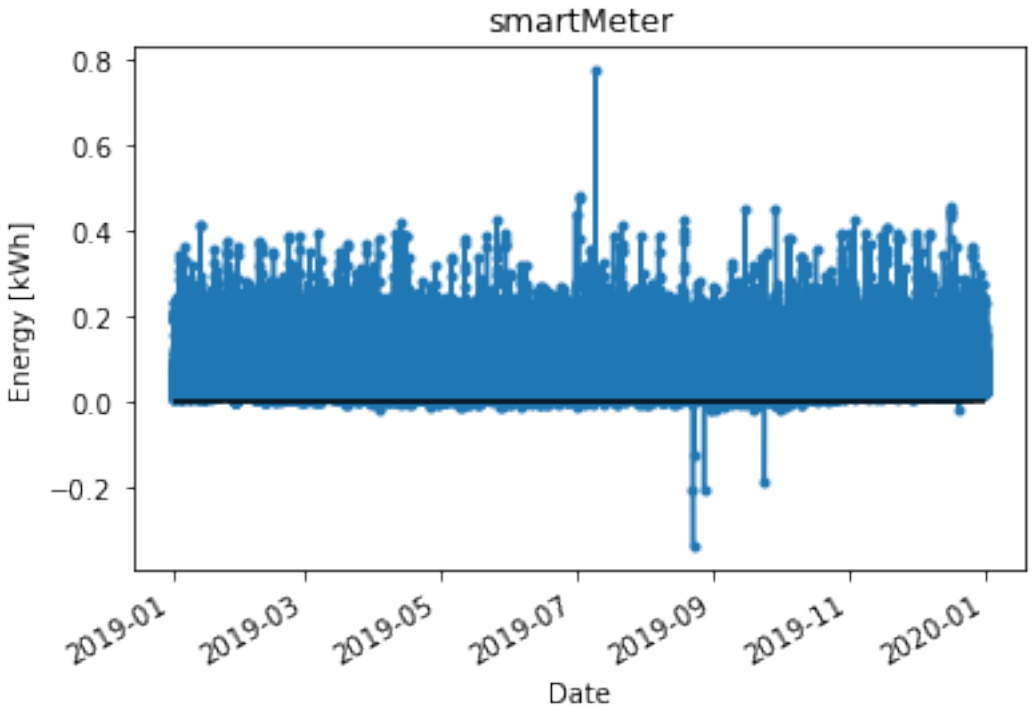
	total_energy_out
2019-12-31 22:55:00	5980.984
2019-12-31 23:00:00	5980.984

	Timestamp	power	total_energy_in	total_energy_out
2019-12-31 22:50:03	1577832603	2.62	20.5	7776.41
2019-12-31 22:55:01	1577832901	2.27	20.5	7776.41

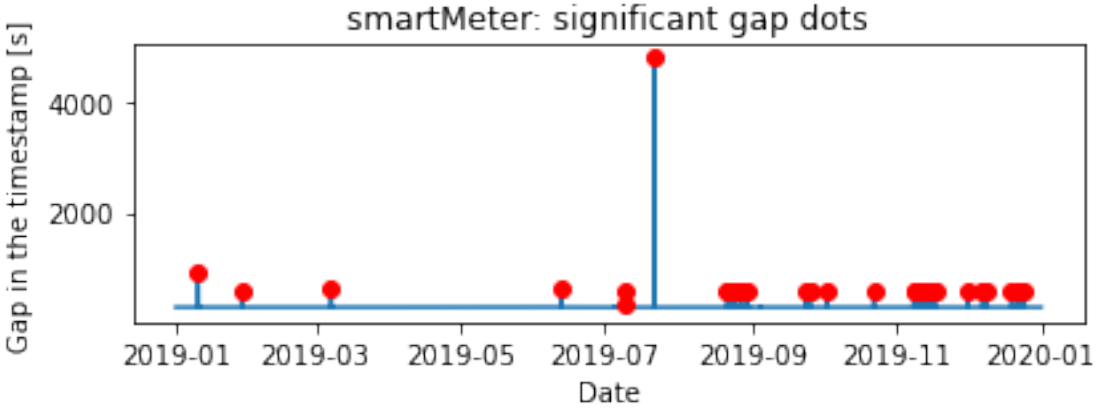
```
df 0 rows: 105102
```

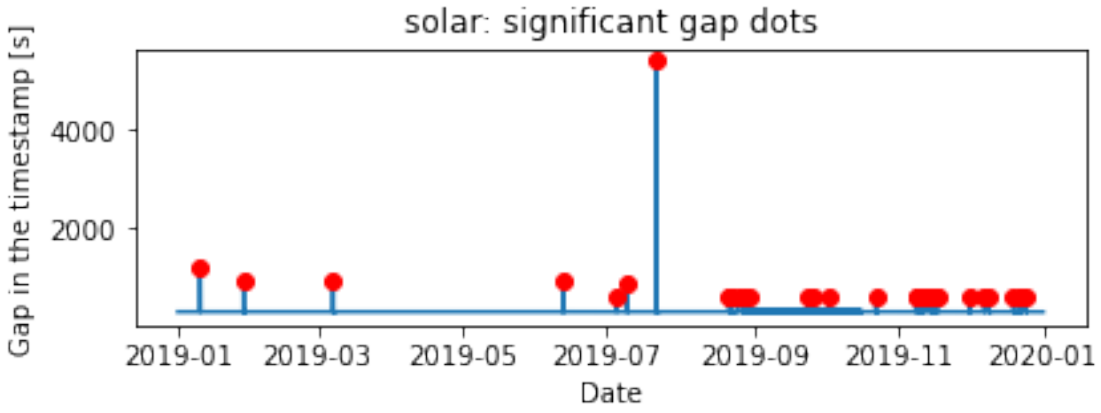
```
df 1 rows: 105092
df rows  : 105092

Total amount of possible 5 minute segments in this index: 105118
Percentage of that in this combined dataset (df): 99.975%
```





<IPython.core.display.HTML object>





<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

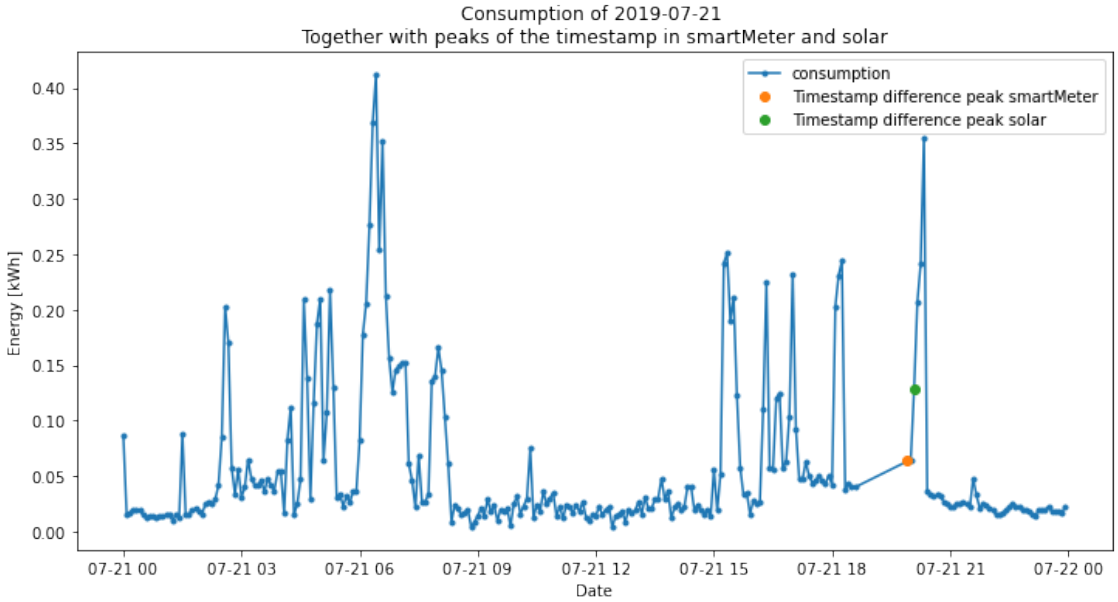
Mean of timestamps in smartMeter: 300.13998286856383
Standard Deviation timestamps of smartMeter: 15.00311893765971

The amount of timestamp gaps significantly (2*std) above 5 minutes: 35
SmartMeter peaks:
smartMeter peaks above 600: 12

	Timestamp	energy_in_low	energy_in_norm	energy_out_low	\
2019-01-10 07:25:00	910.0	0.000	0.177	0.0	
2019-01-28 15:50:01	608.0	0.000	0.035	0.0	
2019-03-07 09:35:00	620.0	0.000	0.000	0.0	
2019-06-12 11:10:01	643.0	0.000	0.011	0.0	
2019-07-21 19:55:00	4805.0	0.020	0.000	0.0	
2019-08-21 13:45:00	604.0	0.000	0.000	0.0	
2019-08-23 08:00:03	603.0	0.000	0.000	0.0	
2019-08-24 23:55:01	601.0	0.030	0.000	0.0	
2019-10-22 09:30:01	601.0	0.000	0.001	0.0	
2019-11-14 11:15:01	601.0	0.000	0.124	0.0	
2019-12-09 04:55:01	601.0	0.092	0.000	0.0	
2019-12-21 21:15:01	601.0	0.126	0.000	0.0	

	energy_out_norm	power	total_energy_in	\
2019-01-10 07:25:00	0.000	419.122222	0.177	
2019-01-28 15:50:01	0.000	497.200000	0.035	
2019-03-07 09:35:00	0.353	-687.066667	0.000	
2019-06-12 11:10:01	0.005	-120.255556	0.011	
2019-07-21 19:55:00	0.000	388.200000	0.020	
2019-08-21 13:45:00	0.531	127.900000	0.000	
2019-08-23 08:00:03	0.304	-94.000000	0.000	
2019-08-24 23:55:01	0.000	-82.000000	0.030	

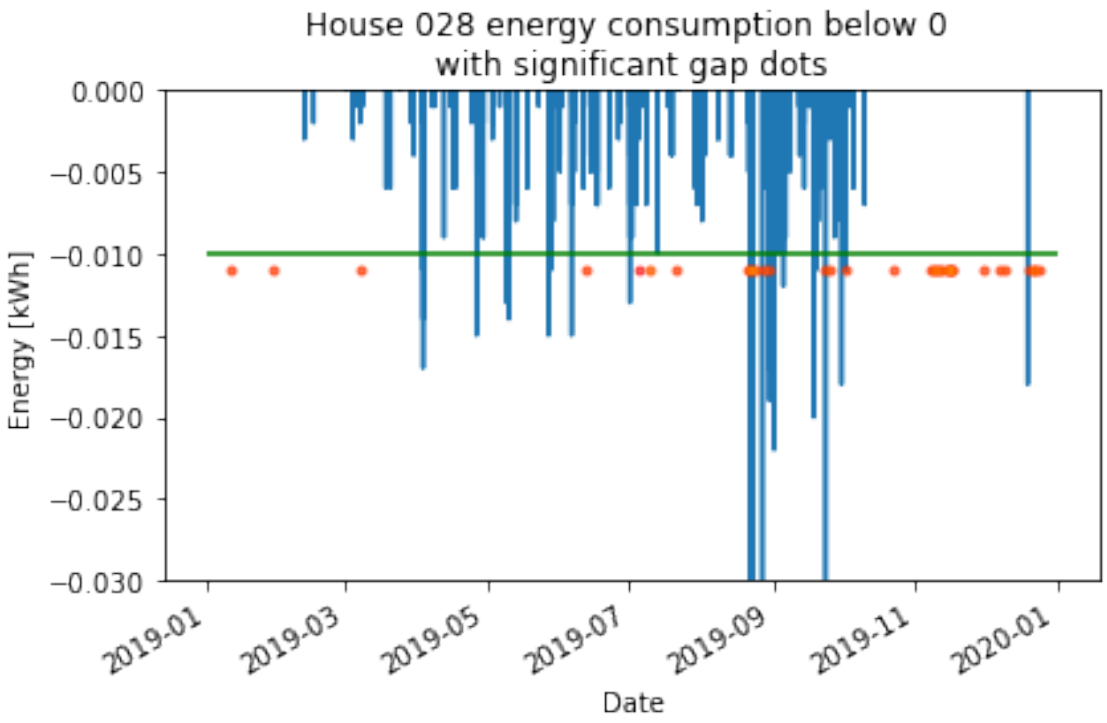
```
2019-10-22 09:30:01            0.045 -856.000000            0.001
2019-11-14 11:15:01            0.000  551.000000            0.124
2019-12-09 04:55:01            0.000 -428.000000            0.092
2019-12-21 21:15:01            0.000  457.000000            0.126

                    total_energy_out
2019-01-10 07:25:00             0.000
2019-01-28 15:50:01             0.000
2019-03-07 09:35:00             0.353
2019-06-12 11:10:01             0.005
2019-07-21 19:55:00             0.000
2019-08-21 13:45:00             0.531
2019-08-23 08:00:03             0.304
2019-08-24 23:55:01             0.000
2019-10-22 09:30:01             0.045
2019-11-14 11:15:01             0.000
2019-12-09 04:55:01             0.000
2019-12-21 21:15:01             0.000

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Mean of timestamps in solar: 300.1599912432659
Standard Deviation timestamps of solar: 17.276597542275262

The amount of timestamp gaps significantly (2*std) above 5 minutes:     34
Solar peaks:
solar peaks above 600: 15

                    Timestamp        power  total_energy_in  total_energy_out
2019-01-10 07:30:00     1211.0    -0.048111              0.0              0.00
2019-01-28 15:55:01      910.0   -55.548769              0.0              0.01
2019-03-07 09:40:01      922.0  -830.500500              0.0              0.60
2019-06-12 11:15:00      944.0  -271.290778              0.0              0.06
2019-07-04 23:35:00      608.0    -0.070000              0.0              0.00
2019-07-09 13:50:01      903.0  1360.885333              0.0              1.14
2019-07-21 20:05:00     5405.0   390.257421              0.0              0.01
2019-08-21 13:45:01      605.0   121.181000              0.0              0.55
2019-08-29 21:30:12      601.0     0.180000              0.0              0.00
2019-09-23 15:15:11      601.0  -232.800000              0.0              0.48
2019-11-14 11:15:03      601.0   224.710000              0.0              0.07
2019-11-15 13:20:01      601.0    78.650000              0.0              0.03
2019-11-16 08:35:02      601.0   182.070000              0.0              0.19
2019-12-07 05:10:04      603.0    -0.390000              0.0              0.00
2019-12-19 12:20:02      601.0   -62.130000              0.0              0.11
```



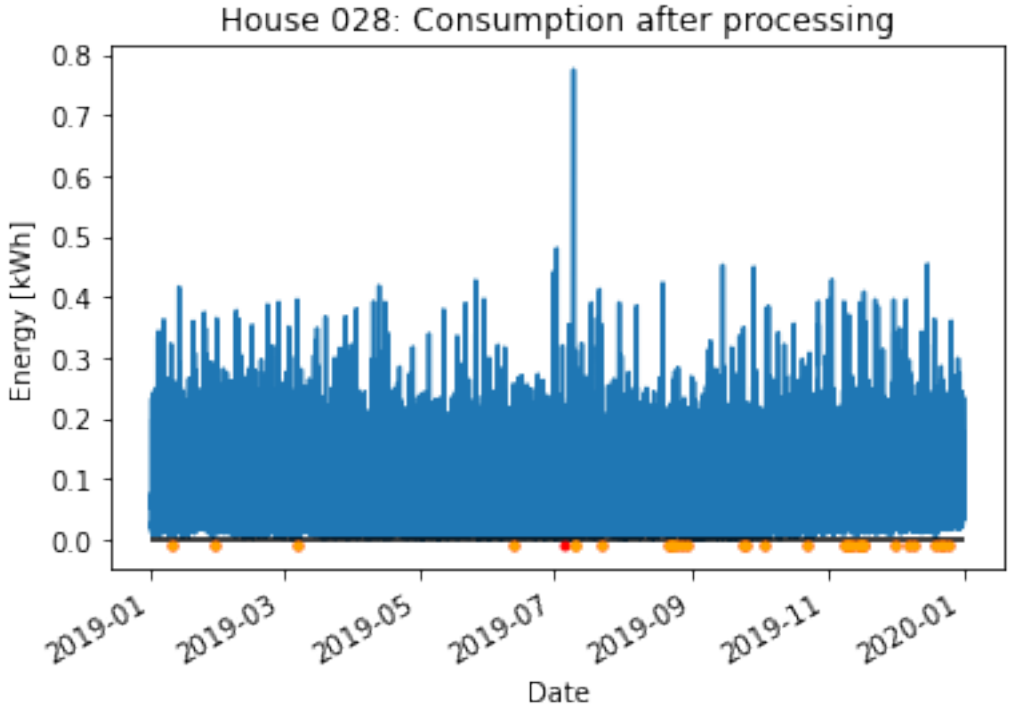
	solar_energy_out	smartMeter_energy_out	\
2019-07-21 18:30:00	0.03	0.0	
2019-07-21 18:35:00	0.03	0.0	
2019-07-21 20:00:00	0.01	0.0	
2019-07-21 20:05:00	0.01	0.0	

	smartMeter_energy_in	production	consumption
2019-07-21 18:30:00	0.010	0.03	0.040
2019-07-21 18:35:00	0.010	0.03	0.040
2019-07-21 20:00:00	0.055	0.01	0.065
2019-07-21 20:05:00	0.118	0.01	0.128

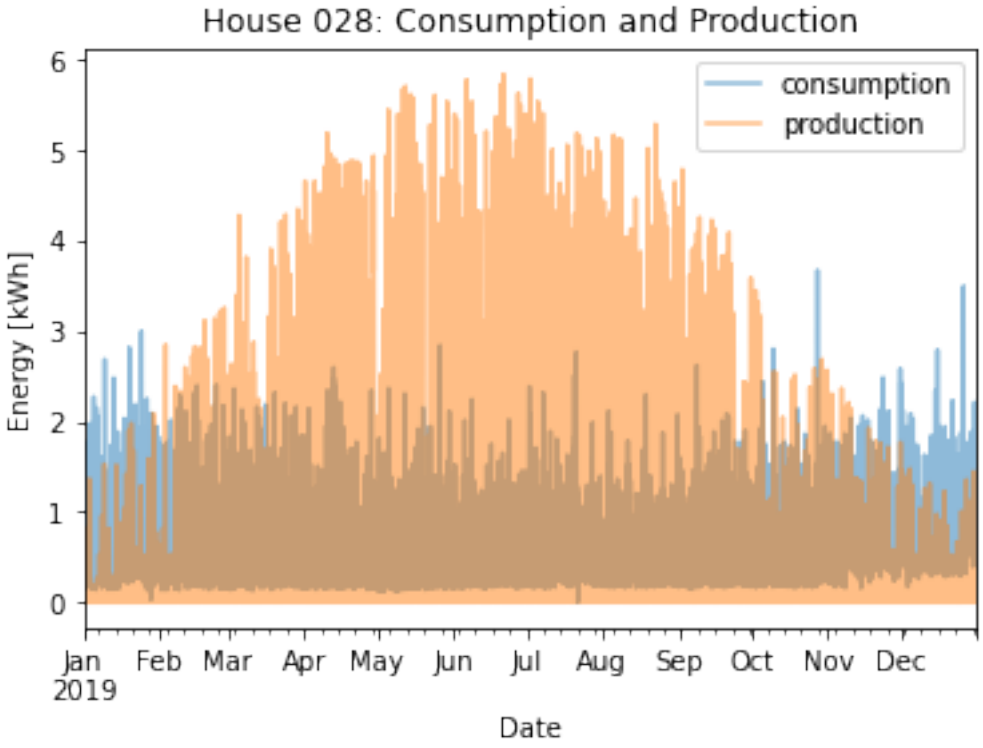
Before the peak at 20:00hr the values are missing. This will result in 0 consumption at that time when resampling to an hour
The peak is most probably caused by a data freeze, however, the peak is not high enough to destroy the models we are using
The 5 minute aggregation method was set to: .nearest(limit=1)
Nearest results with a gap in the data, so it stays closer to reality than using something like mean or sum




```
The amount of consumption below 0:              204
The amount of consumption below -0.010:         28
```



```
204 values were set to 0
Which is 0.19% of the dataset
The area under consumption = 5089.9 kWh
The area under production = 7464.21 kWh
```




```
No Features were added
```

	consumption	production
2019-12-31 18:00:00	0.918	0.0
2019-12-31 19:00:00	1.163	0.0
2019-12-31 20:00:00	1.315	0.0
2019-12-31 21:00:00	0.711	0.0
2019-12-31 22:00:00	1.179	0.0

[ ]: