Part I: Analysis of Dataset

We chose REFIT, a dataset of - 20 households (refit.buildings), recorded between - Oct 2013 - Jun 2015, with a sampling rate of - 8 seconds interval and - 9 possible appliances in each household (refit.elecs(): some of them are missing in house 12, 13 and 20).

The included raw electrical consumption data in Watt were collected during a project regarding research in the field of energy conservation and advanced energy services. More information can be found in:

https://pureportal.strath.ac.uk/en/datasets/refit-electrical-load-measurements

https://pure.strath.ac.uk/ws/portalfiles/portal/45410335/REFITREADME.txt

https://www.nature.com/articles/sdata2016122

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from nilmtk import DataSet

DATA_PATH = '.\data\REFIT.h5'
refit = DataSet(DATA_PATH)

type(refit)

nilmtk.dataset.DataSet
```

How does our dataset look like?

Number of Available Buildings

```
# easy way to find out the number of households
refit.buildings

OrderedDict([(1, Building(instance=1, dataset='REFIT')),
             (10, Building(instance=10, dataset='REFIT')),
             (11, Building(instance=11, dataset='REFIT')),
             (12, Building(instance=12, dataset='REFIT')),
             (13, Building(instance=13, dataset='REFIT')),
             (14, Building(instance=14, dataset='REFIT')),
             (15, Building(instance=15, dataset='REFIT')),
             (16, Building(instance=16, dataset='REFIT')),
             (17, Building(instance=17, dataset='REFIT')),
             (18, Building(instance=18, dataset='REFIT')),
             (19, Building(instance=19, dataset='REFIT')),
             (2, Building(instance=2, dataset='REFIT')),
             (20, Building(instance=20, dataset='REFIT')),
             (3, Building(instance=3, dataset='REFIT')),
             (4, Building(instance=4, dataset='REFIT')),
             (5, Building(instance=5, dataset='REFIT')),
```

```
              (6, Building(instance=6, dataset='REFIT')),
              (7, Building(instance=7, dataset='REFIT')),
              (8, Building(instance=8, dataset='REFIT')),
              (9, Building(instance=9, dataset='REFIT'))])
```

There are 20 buildings in the refit-Dataset.

Available Appliances

```
# electric meters and the appliances for each household (-> 9?!) were checked
with
refit.elecs()
#...
```

There seems to be some missing appliances in building 12, 13 and 20…

Characteristics of the Power Consumption

Now let's go more in detail: For analysing the dataset we choose two different time windows, both 4 months long - one is set during spring/summer 2014, the other one during autumn/winter 2014/15. The function describe() results in a first overview of all households:

```
refit.set_window(start='2014-04-01', end='2014-07-31')
refit.describe()
```

(Table output omitted due to very wide format)

```
refit.set_window(start='2014-10-01', end='2015-01-31')
refit.describe()
```

(Table output omitted due to very wide format)

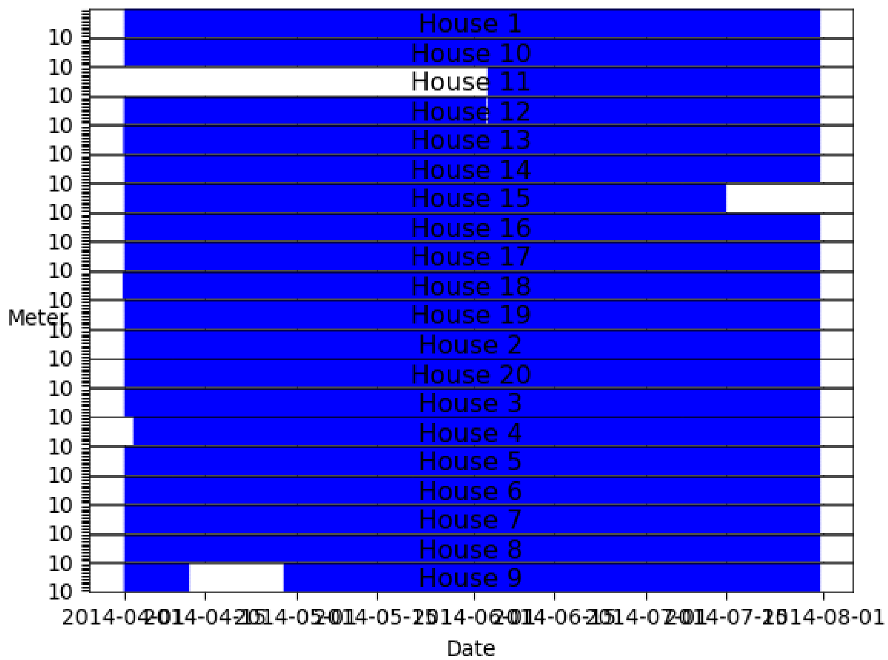
First impression: not all households have the same quality. Our focus is on duration and uptime, but also on dropout rates and correlation. There are differences during summertime and wintertime, too.

'Proportion of energy submetered' is quite low in all houses, therefore the amount of noise is quite high for all of them (but looking on the measured appliances itself, it seems that there are some important ones unmeasured). For the tasks of this Case Study - looking on appliances separately - this kind of noise shouldn't affect the results…

Although we have an impression, which households could fit for our project, we do some more investigation to learn about our data.

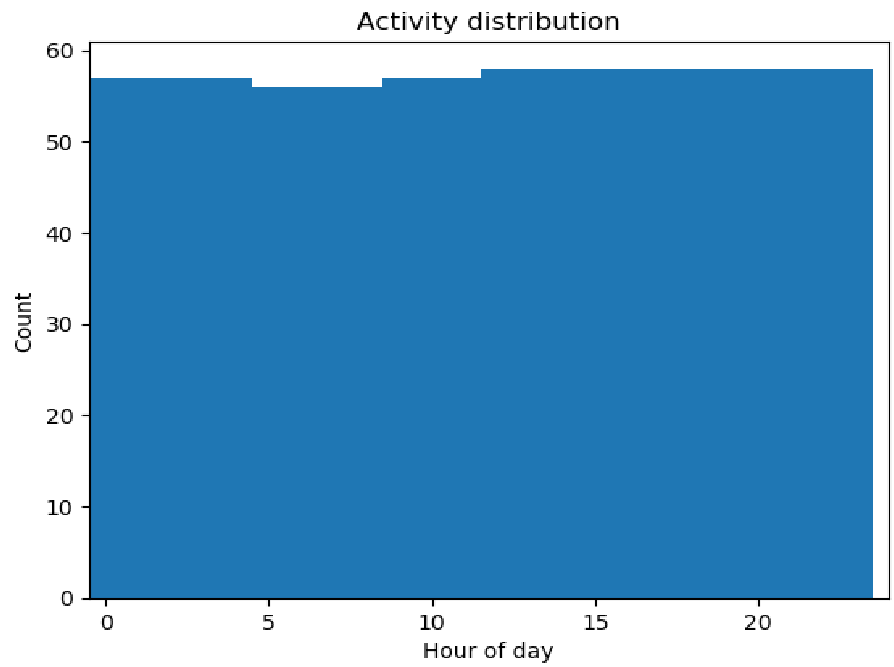
```
# back to the first time window
refit.set_window(start='2014-04-01', end='2014-07-31')

refit.plot_good_sections()

c:\Users\Chris\.conda\envs\case-study\lib\site-
packages\pandas\plotting\_matplotlib\converter.py:103: FutureWarning: Using
an implicitly registered datetime converter for a matplotlib plotting method.
The converter was registered by pandas on import. Future versions of pandas
```

```
will require you to explicitly register matplotlib converters.

To register the converters:
    >>> from pandas.plotting import register_matplotlib_converters
    >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
c:\Users\Chris\.conda\envs\case-study\lib\site-
packages\nilmtk\dataset.py:133: UserWarning: Tight layout not applied.
tight_layout cannot make axes height small enough to accommodate all axes
decorations
  plt.tight_layout()
```



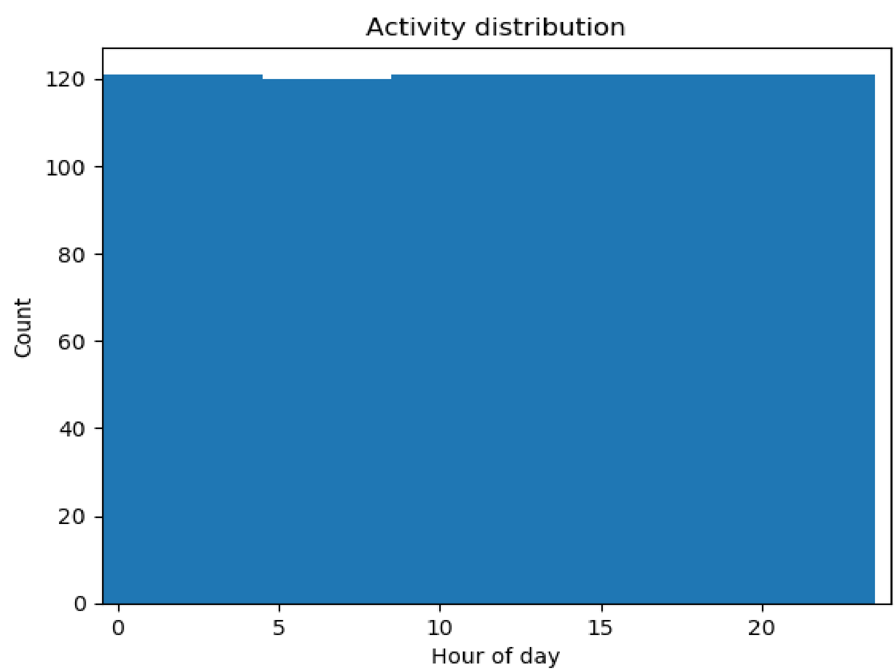
This plot strengthens the considerations about the problems of some households during the first time frame, which we saw also in the coresponding description table before. For example: 64 days are missing during the measurement of house 11. Let's check the activity histogram:

```
refit.buildings[11].elec.plot_activity_histogram()

Loading data for meter ElecMeterID(instance=10, building=11, dataset='REFIT')
Done loading data all meters for this chunk.

<matplotlib.axes._subplots.AxesSubplot at 0x170c8099040>
```



As we can see, the measurement amounted to only 58 days. Comparing to another building:

```
refit.buildings[5].elec.plot_activity_histogram()

Loading data for meter ElecMeterID(instance=10, building=5, dataset='REFIT')
Done loading data all meters for this chunk.

<matplotlib.axes._subplots.AxesSubplot at 0x170c84dee50>
```



The gap percentage is near zero for some of the buildings (e.g. house 5, 7, 10, 14), but higher for houses like 15, 9 and special for house 11 (good to see - for example in histograms above for house 11 and 5 - by values for 'Count').

Let's check out the dropout-rate for some buildings too. First we will ignore the gaps, and in the second step we won't ignore them. This will help us to decide on the right buildings where the measurements are good.

```
refit.buildings[1].elec.dropout_rate()

Calculating dropout_rate for ElecMeterID(instance=10, building=1,
dataset='REFIT') ...

0.0555345075877723

refit.buildings[1].elec.dropout_rate(ignore_gaps=False)

Calculating dropout_rate for ElecMeterID(instance=10, building=1,
dataset='REFIT') ...

0.040622436783764204

refit.buildings[5].elec.dropout_rate()

Calculating dropout_rate for ElecMeterID(instance=10, building=5,
dataset='REFIT') ...

0.0005233707743938547

refit.buildings[5].elec.dropout_rate(ignore_gaps=False)

Calculating dropout_rate for ElecMeterID(instance=10, building=5,
dataset='REFIT') ...

0.0

refit.buildings[7].elec.dropout_rate()

Calculating dropout_rate for ElecMeterID(instance=10, building=7,
dataset='REFIT') ...

0.010221714327823239

refit.buildings[7].elec.dropout_rate(ignore_gaps=False)

Calculating dropout_rate for ElecMeterID(instance=10, building=7,
dataset='REFIT') ...

0.0

refit.buildings[9].elec.dropout_rate()

Calculating dropout_rate for ElecMeterID(instance=10, building=9,
dataset='REFIT') ...

0.0020554369924269894
```

```python
refit.buildings[9].elec.dropout_rate(ignore_gaps=False)
```

```
Calculating dropout_rate for ElecMeterID(instance=10, building=9,
dataset='REFIT') ...

0.040298335571943733
```

```python
refit.buildings[12].elec.dropout_rate()
```

```
Calculating dropout_rate for ElecMeterID(instance=10, building=12,
dataset='REFIT') ...

6.508721234459649e-06
```

```python
refit.buildings[12].elec.dropout_rate(ignore_gaps=False)
```

```
Calculating dropout_rate for ElecMeterID(instance=10, building=12,
dataset='REFIT') ...

0.0
```

```python
refit.buildings[14].elec.dropout_rate()
```

```
Calculating dropout_rate for ElecMeterID(instance=10, building=14,
dataset='REFIT') ...

0.003631467858543161
```

```python
refit.buildings[14].elec.dropout_rate(ignore_gaps=False)
```

```
Calculating dropout_rate for ElecMeterID(instance=10, building=14,
dataset='REFIT') ...

0.0
```

```python
refit.buildings[20].elec.dropout_rate()
```

```
Calculating dropout_rate for ElecMeterID(instance=10, building=20,
dataset='REFIT') ...

0.08217909797525261
```

```python
refit.buildings[20].elec.dropout_rate(ignore_gaps=False)
```

```
Calculating dropout_rate for ElecMeterID(instance=10, building=20,
dataset='REFIT') ...

0.07861880099230989
```

Our previous research gave us a first impression of the households. Now we will focus on four households, which seem to be appropriate: 5, 7 and 14.

The buildings 5, 7 and 14

First we will look at the submeters, then we will calculate the total energy and finally look at the plots for each building.

```
refit.buildings[5].elec.submeters()
```

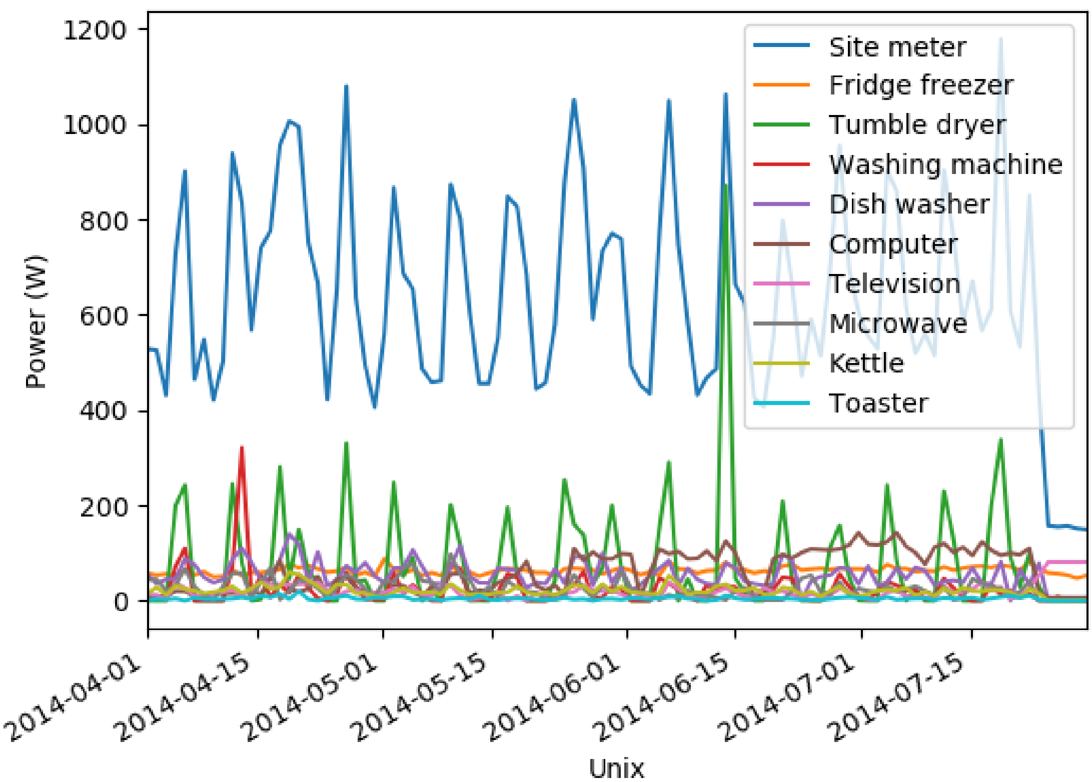
```
MeterGroup(meters=
  ElecMeter(instance=2, building=5, dataset='REFIT', appliances=[Appliance(type='fridge freezer', instance=1)])
  ElecMeter(instance=3, building=5, dataset='REFIT', appliances=[Appliance(type='tumble dryer', instance=1)])
  ElecMeter(instance=4, building=5, dataset='REFIT', appliances=[Appliance(type='washing machine', instance=1)])
  ElecMeter(instance=5, building=5, dataset='REFIT', appliances=[Appliance(type='dish washer', instance=1)])
  ElecMeter(instance=6, building=5, dataset='REFIT', appliances=[Appliance(type='computer', instance=1)])
  ElecMeter(instance=7, building=5, dataset='REFIT', appliances=[Appliance(type='television', instance=1)])
  ElecMeter(instance=8, building=5, dataset='REFIT', appliances=[Appliance(type='microwave', instance=1)])
  ElecMeter(instance=9, building=5, dataset='REFIT', appliances=[Appliance(type='kettle', instance=1)])
  ElecMeter(instance=10, building=5, dataset='REFIT', appliances=[Appliance(type='toaster', instance=1)])
)
```

```
refit.buildings[5].elec.total_energy()
```

```
Calculating total_energy for ElecMeterID(instance=10, building=5, dataset='REFIT') ...

active    2819.161941
dtype: float64
```

```
refit.buildings[5].elec.plot(ax=None, timeframe=None, plot_legend=True, unit='W', width=100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x170c90c9bb0>
```

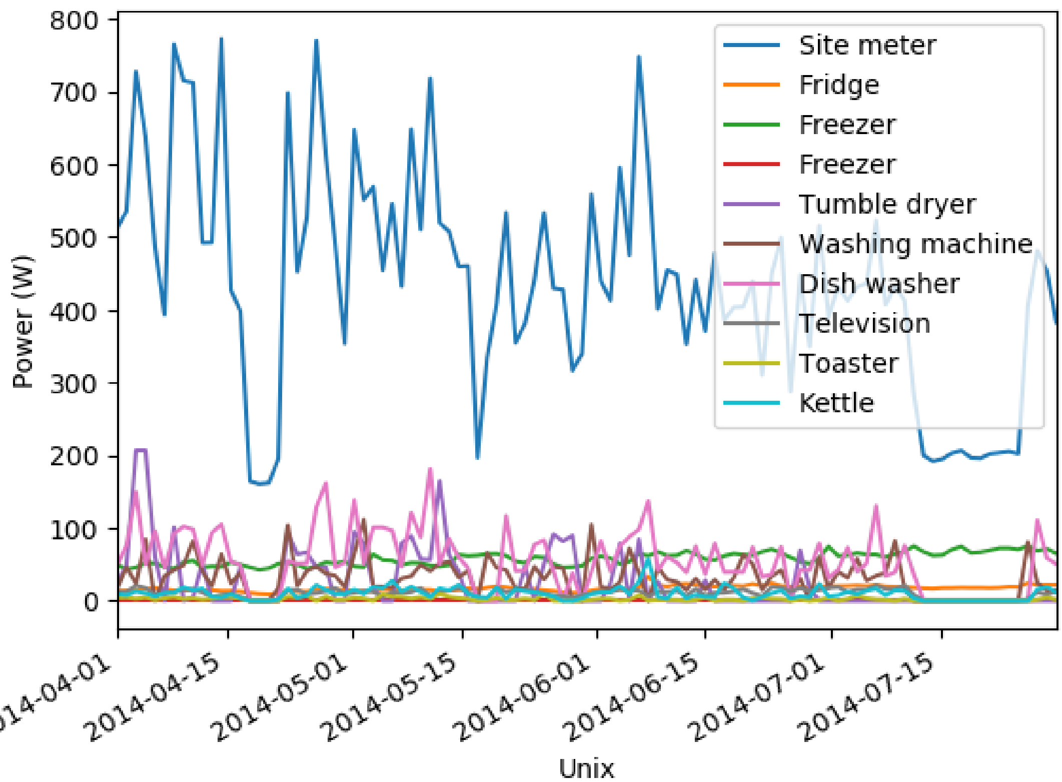



```
refit.buildings[7].elec.submeters()

MeterGroup(meters=
  ElecMeter(instance=2, building=7, dataset='REFIT',
appliances=[Appliance(type='fridge', instance=1)])
  ElecMeter(instance=3, building=7, dataset='REFIT',
appliances=[Appliance(type='freezer', instance=1)])
  ElecMeter(instance=4, building=7, dataset='REFIT',
appliances=[Appliance(type='freezer', instance=2)])
  ElecMeter(instance=5, building=7, dataset='REFIT',
appliances=[Appliance(type='tumble dryer', instance=1)])
  ElecMeter(instance=6, building=7, dataset='REFIT',
appliances=[Appliance(type='washing machine', instance=1)])
  ElecMeter(instance=7, building=7, dataset='REFIT',
appliances=[Appliance(type='dish washer', instance=1)])
  ElecMeter(instance=8, building=7, dataset='REFIT',
appliances=[Appliance(type='television', instance=1)])
  ElecMeter(instance=9, building=7, dataset='REFIT',
appliances=[Appliance(type='toaster', instance=1)])
  ElecMeter(instance=10, building=7, dataset='REFIT',
appliances=[Appliance(type='kettle', instance=1)])
)

refit.buildings[7].elec.total_energy()

Calculating total_energy for ElecMeterID(instance=10, building=7,
dataset='REFIT') ...
```

```
active    1836.118926
dtype: float64
```

```
refit.buildings[7].elec.plot(ax=None, timeframe=None, plot_legend=True,
unit='W', width=100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x170c8b49850>
```




```
refit.buildings[14].elec.submeters()
```

```
MeterGroup(meters=
  ElecMeter(instance=2, building=14, dataset='REFIT',
appliances=[Appliance(type='fridge freezer', instance=1)])
  ElecMeter(instance=3, building=14, dataset='REFIT',
appliances=[Appliance(type='tumble dryer', instance=1)])
  ElecMeter(instance=4, building=14, dataset='REFIT',
appliances=[Appliance(type='washing machine', instance=1)])
  ElecMeter(instance=5, building=14, dataset='REFIT',
appliances=[Appliance(type='dish washer', instance=1)])
  ElecMeter(instance=6, building=14, dataset='REFIT',
appliances=[Appliance(type='computer', instance=1)])
  ElecMeter(instance=7, building=14, dataset='REFIT',
appliances=[Appliance(type='television', instance=1)])
  ElecMeter(instance=8, building=14, dataset='REFIT',
appliances=[Appliance(type='microwave', instance=1)])
  ElecMeter(instance=9, building=14, dataset='REFIT',
appliances=[Appliance(type='audio system', instance=1)])
  ElecMeter(instance=10, building=14, dataset='REFIT',
```

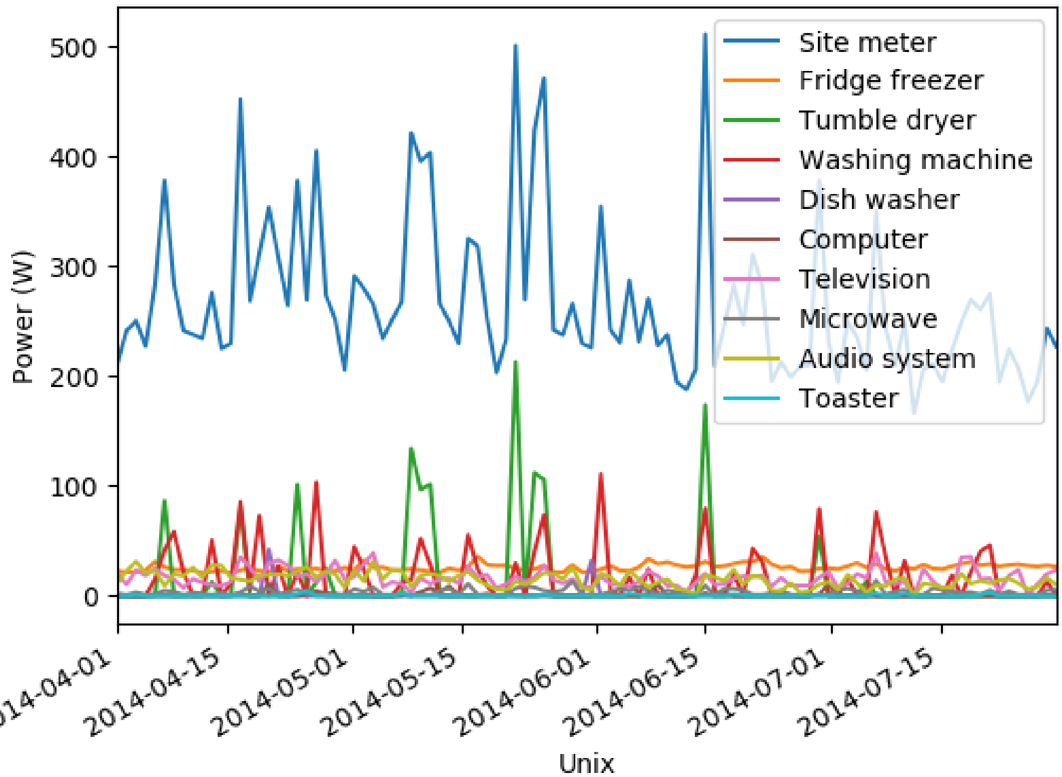
```
appliances=[Appliance(type='toaster', instance=1)])
)
```

```
refit.buildings[14].elec.total_energy()
```

```
Calculating total_energy for ElecMeterID(instance=10, building=14, dataset='REFIT') ...

active    4038.971498
dtype: float64
```

```
refit.buildings[14].elec.plot(ax=None, timeframe=None, plot_legend=True, unit='W', width=100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x170c9223b50>
```



For each of these buildings it would be possible to go more in detail - for example household 5:

```
refit.buildings[5].elec.appliances
```

```
[Appliance(type='television', instance=1),
 Appliance(type='microwave', instance=1),
 Appliance(type='kettle', instance=1),
 Appliance(type='dish washer', instance=1),
 Appliance(type='washing machine', instance=1),
 Appliance(type='toaster', instance=1),
 Appliance(type='tumble dryer', instance=1),
```

```
 Appliance(type='computer', instance=1),
 Appliance(type='fridge freezer', instance=1)]
```

```
refit.buildings[5].elec.submeters().energy_per_meter()
```

```
9/9 ElecMeter(instance=10, building=5, dataset='REFIT',
appliances=[Appliance(type='toaster', instance=1)])))))1)])
```

	(2, 5, REFIT)	(3, 5, REFIT)	(4, 5, REFIT)	(5, 5, REFIT)	(6, 5, REFIT)	(7, 5, REFIT)	(8, 5, REFIT)	(9, 5, REFIT)	(10, 5, REFIT)
active	182.59 8021	213.85 0364	66.053 814	155.13 1404	173.10 1041	59.511 101	72.94 899	59.675 328	12.943 093
appar ent	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
reacti ve	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

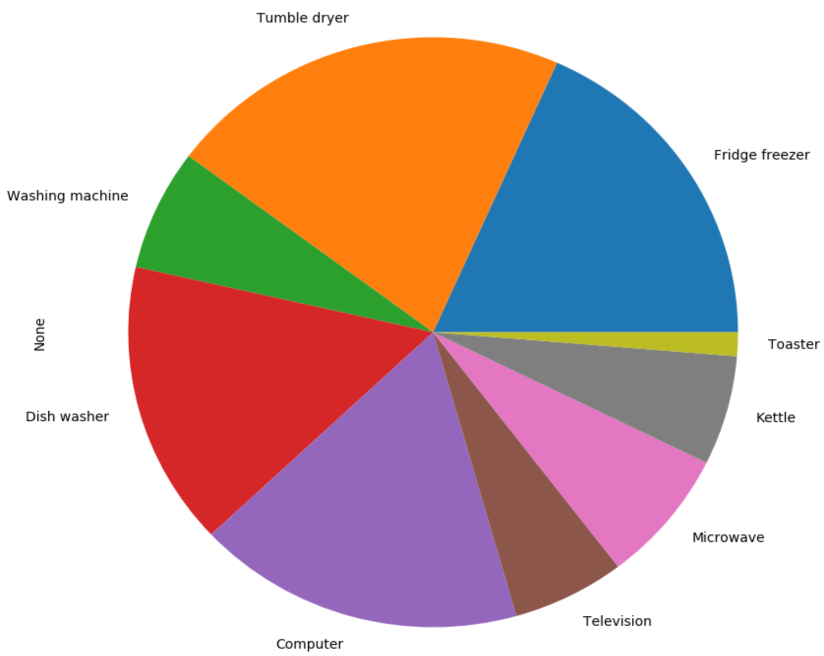
But more interesting are the proportions of the single appliances. Appliances with high power values shadow the smaller ones…

Let us look on our three houses:

```
fraction_5 =
refit.buildings[5].elec.submeters().fraction_per_meter().dropna()
labels_5 = refit.buildings[5].elec.get_labels(fraction_5.index)
plt.figure(figsize=(10,30))
fraction_5.plot(kind='pie', labels=labels_5)
```

```
9/9 ElecMeter(instance=10, building=5, dataset='REFIT',
appliances=[Appliance(type='toaster', instance=1)])))))1)])
```

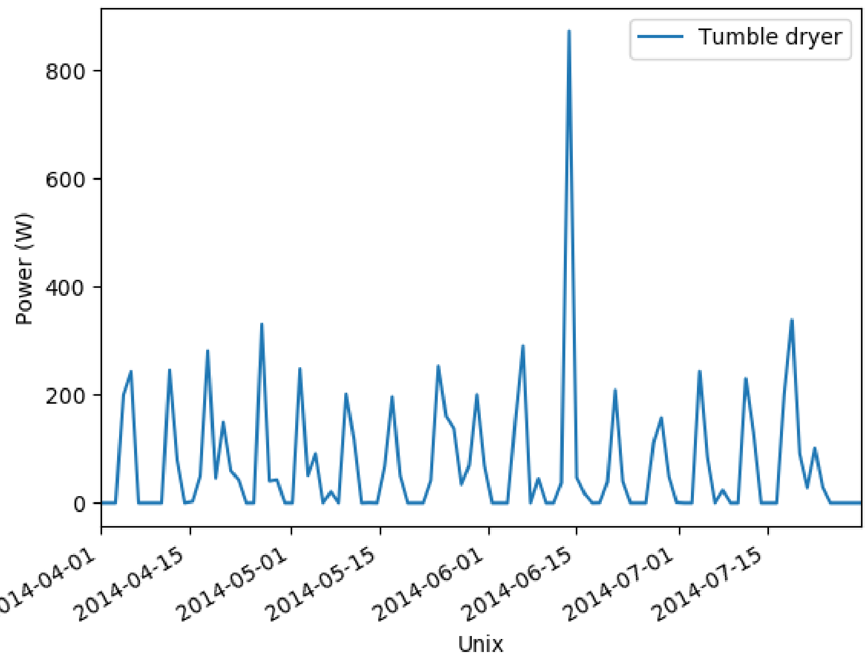
```
<matplotlib.axes._subplots.AxesSubplot at 0x2819ab8ec70>
```



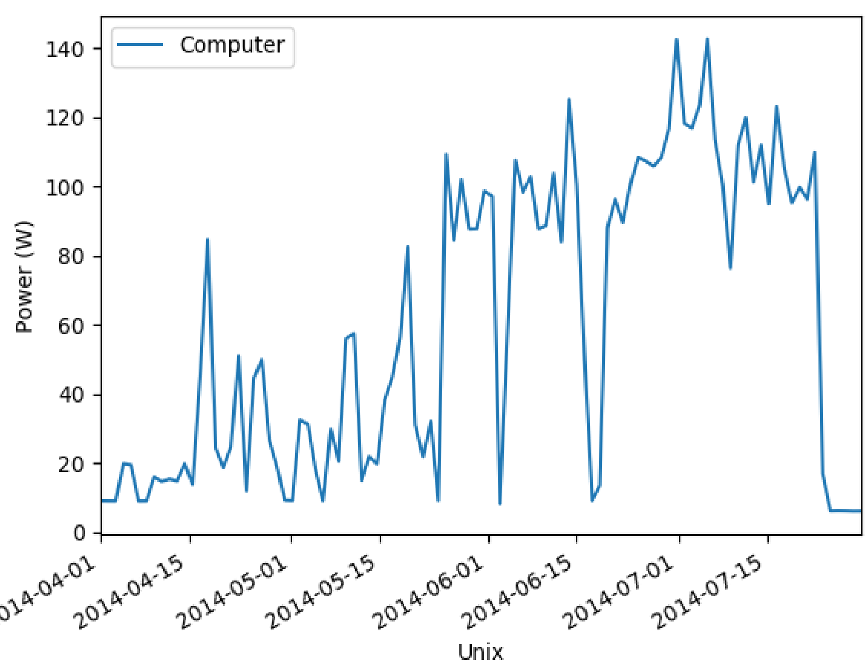

```
refit.buildings[5].elec.submeters().fraction_per_meter()

9/9 ElecMeter(instance=10, building=5, dataset='REFIT',
appliances=[Appliance(type='toaster', instance=1)]))))1)])

(2, 5, REFIT)     0.183366
(3, 5, REFIT)     0.214749
(4, 5, REFIT)     0.066332
(5, 5, REFIT)     0.155784
(6, 5, REFIT)     0.173829
(7, 5, REFIT)     0.059761
(8, 5, REFIT)     0.073256
(9, 5, REFIT)     0.059926
(10, 5, REFIT)    0.012998
dtype: float64
```

Interesting appliances of building 5: tumble dryer (instance 3) and computer (instance 6).

```
refit.buildings[5].elec[3].plot(ax=None, timeframe=None, plot_legend=True,
unit='W', width=100)

<matplotlib.axes._subplots.AxesSubplot at 0x170de7e0bb0>
```

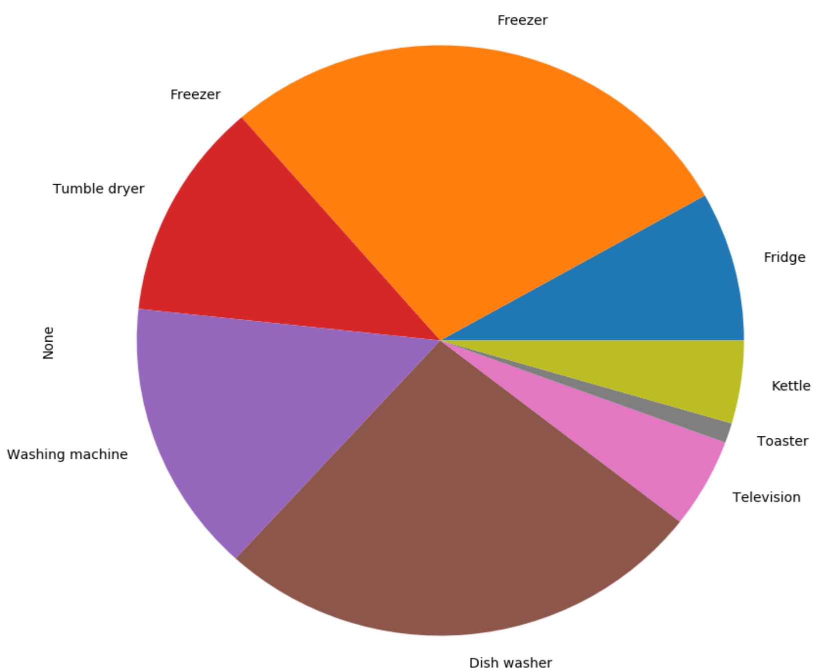



```
refit.buildings[5].elec[6].plot(ax=None, timeframe=None, plot_legend=True,
unit='W', width=100)

<matplotlib.axes._subplots.AxesSubplot at 0x170ca9d6e50>
```




```
fraction_7 =
refit.buildings[7].elec.submeters().fraction_per_meter().dropna()
labels_7 = refit.buildings[7].elec.get_labels(fraction_7.index)
plt.figure(figsize=(10,30))
fraction_7.plot(kind='pie', labels=labels_7)
```

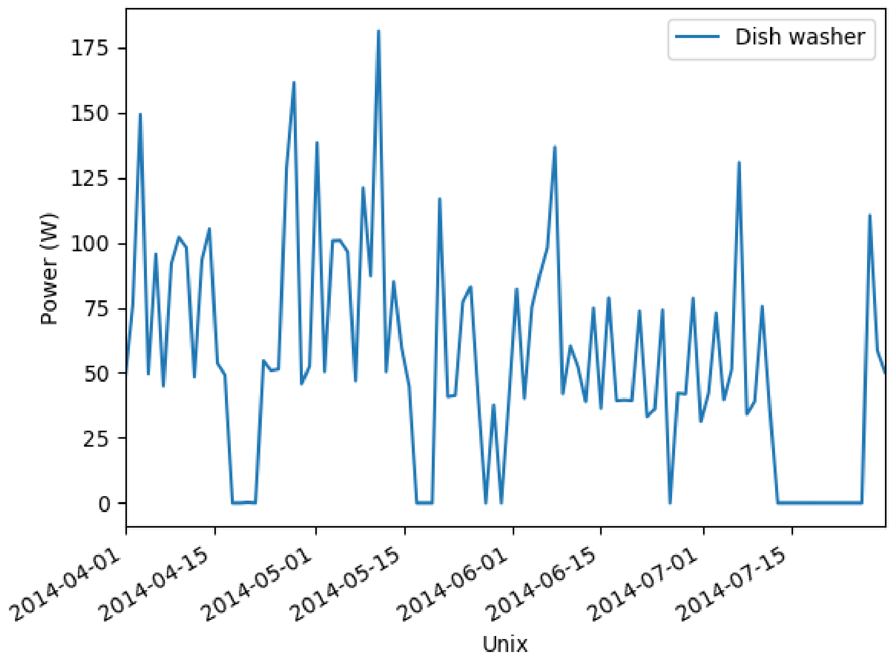
```
9/9 ElecMeter(instance=10, building=7, dataset='REFIT',
appliances=[Appliance(type='kettle', instance=1)]))]))1)])

<matplotlib.axes._subplots.AxesSubplot at 0x2819ac0ec40>
```




```
refit.buildings[7].elec.submeters().fraction_per_meter()

9/9 ElecMeter(instance=10, building=7, dataset='REFIT',
appliances=[Appliance(type='kettle', instance=1)]))]))1)])

(2, 7, REFIT)     8.169317e-02
(3, 7, REFIT)     2.819017e-01
(4, 7, REFIT)     7.059968e-09
(5, 7, REFIT)     1.192980e-01
(6, 7, REFIT)     1.493387e-01
(7, 7, REFIT)     2.625398e-01
(8, 7, REFIT)     4.903015e-02
(9, 7, REFIT)     1.086364e-02
(10, 7, REFIT)    4.533493e-02
dtype: float64
```

For us choosing are dataset interessting appliances of household 7 are: the dish washer (instance 7) and the kettle (instance 10).

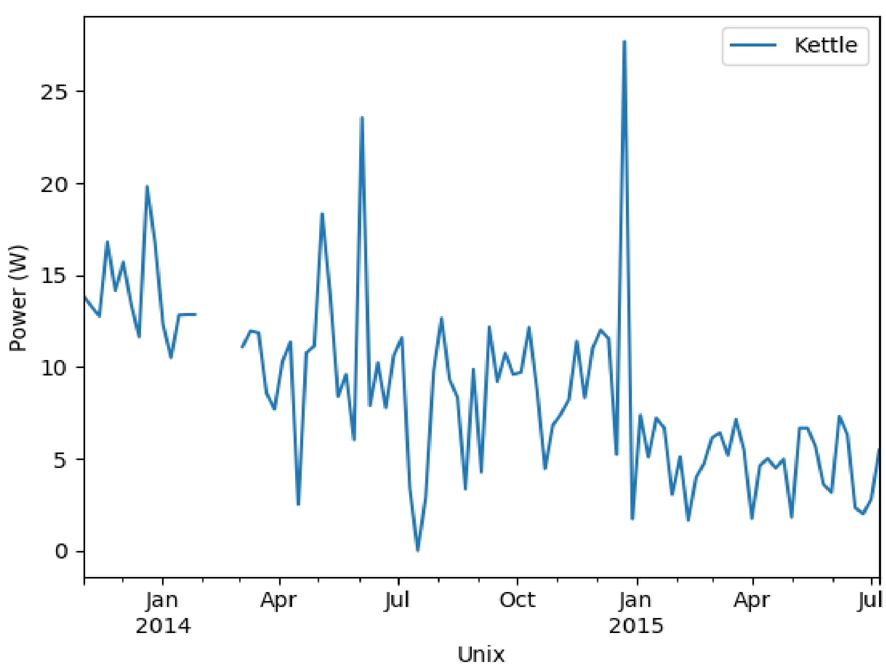
```
refit.buildings[7].elec[7].plot(ax=None, timeframe=None, plot_legend=True,
unit='W', width=100)

<matplotlib.axes._subplots.AxesSubplot at 0x170cabf1160>
```




```
refit.buildings[7].elec[10].plot(ax=None, timeframe=None, plot_legend=True,
unit='W', width=100)
```

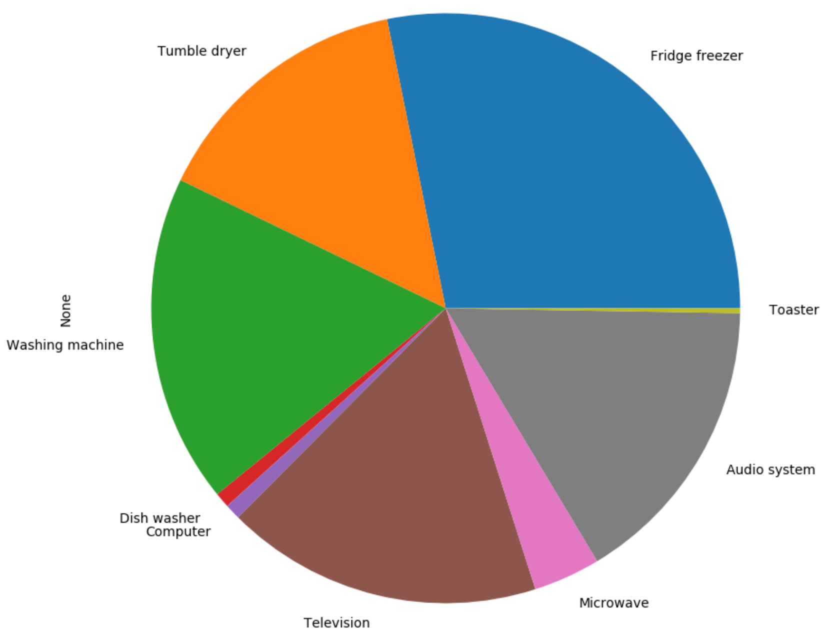
```
c:\Users\Chris\.conda\envs\case-study\lib\site-
packages\pandas\core\arrays\datetimes.py:1266: UserWarning: Converting to
PeriodArray/Index representation will drop timezone information.
  warnings.warn(
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x15117c027c0>
```



```python
fraction_14 =
refit.buildings[14].elec.submeters().fraction_per_meter().dropna()
labels_14 = refit.buildings[14].elec.get_labels(fraction_14.index)
plt.figure(figsize=(10,30))
fraction_14.plot(kind='pie', labels=labels_14)
```

```
9/9 ElecMeter(instance=10, building=14, dataset='REFIT',
appliances=[Appliance(type='toaster', instance=1)])1)]))])

<matplotlib.axes._subplots.AxesSubplot at 0x2819d224250>
```



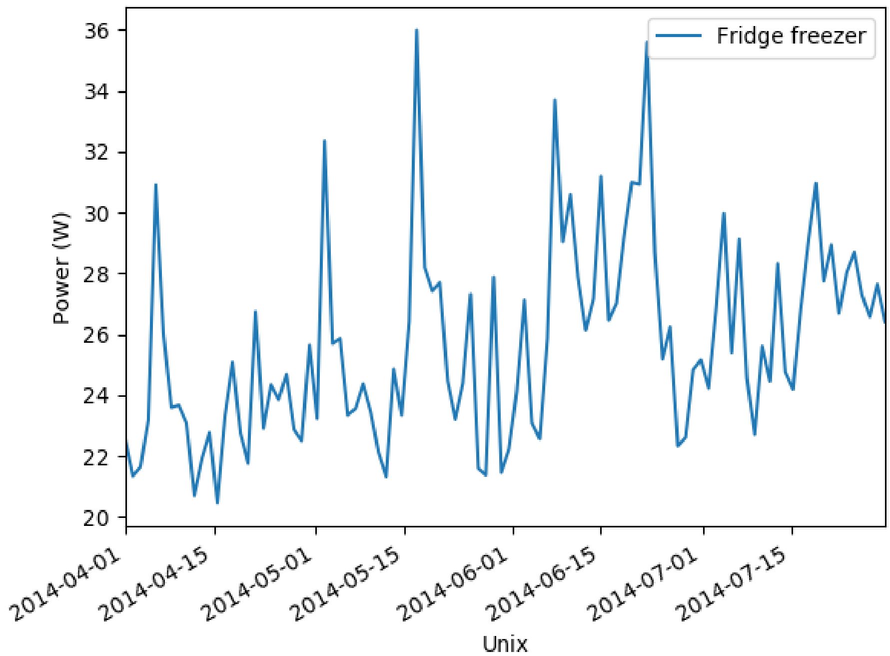

```python
refit.buildings[14].elec.submeters().fraction_per_meter()
```

```
9/9 ElecMeter(instance=10, building=14, dataset='REFIT',
appliances=[Appliance(type='toaster', instance=1)])1)]))])

(2, 14, REFIT)     0.281964
(3, 14, REFIT)     0.146541
(4, 14, REFIT)     0.180122
(5, 14, REFIT)     0.008278
(6, 14, REFIT)     0.008606
(7, 14, REFIT)     0.173769
(8, 14, REFIT)     0.036765
(9, 14, REFIT)     0.161324
(10, 14, REFIT)    0.002631
dtype: float64
```

Building 14 shows a different plot because of 3 very small parts. Appliances with high proportion are instance 2 - the fridge freezer, instance 4 - the washing machine, instance 7 - the television and instance 9 - the audio system.

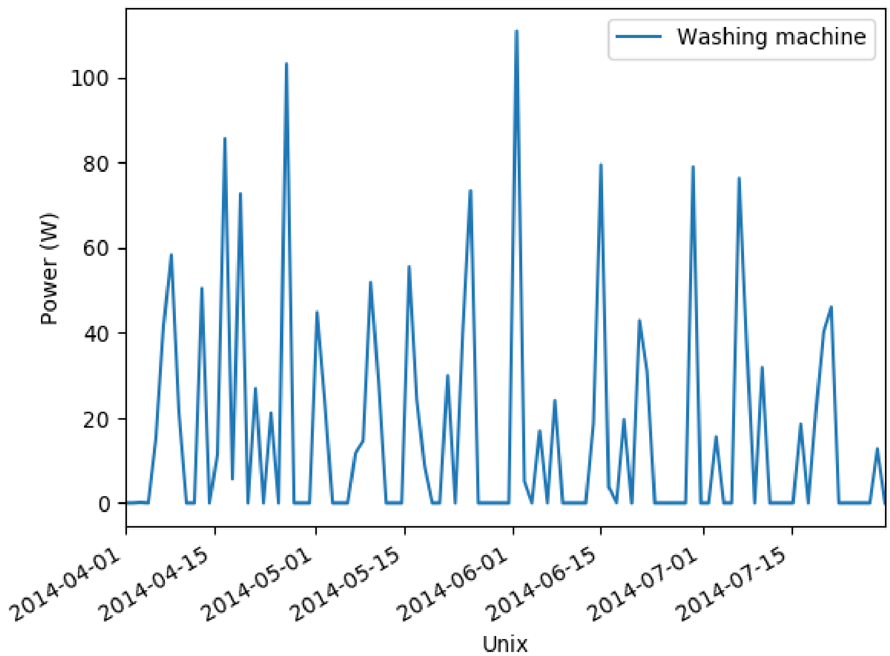
```
refit.buildings[14].elec[2].plot(ax=None, timeframe=None, plot_legend=True, unit='W', width=100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x170cab62d00>
```

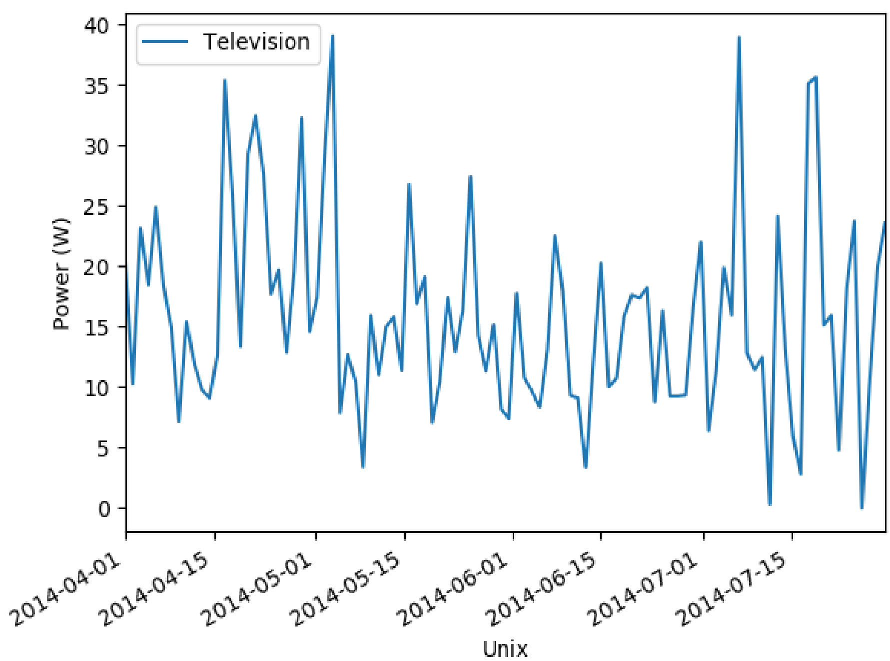



```
refit.buildings[14].elec[4].plot(ax=None, timeframe=None, plot_legend=True, unit='W', width=100)
```

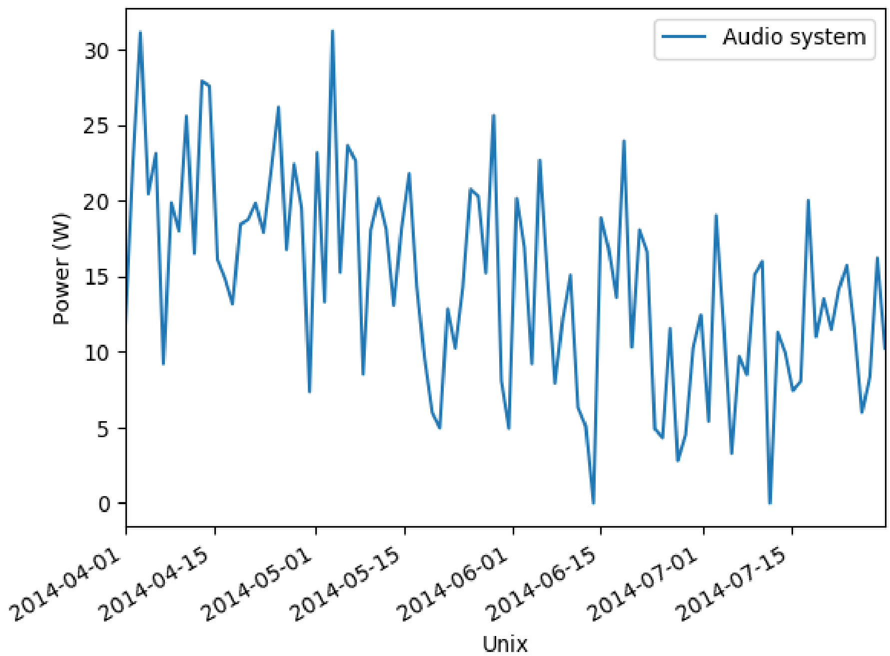
```
<matplotlib.axes._subplots.AxesSubplot at 0x170cadbe970>
```




```
refit.buildings[14].elec[7].plot(ax=None, timeframe=None, plot_legend=True,
unit='W', width=100)

<matplotlib.axes._subplots.AxesSubplot at 0x170cae3ad00>
```

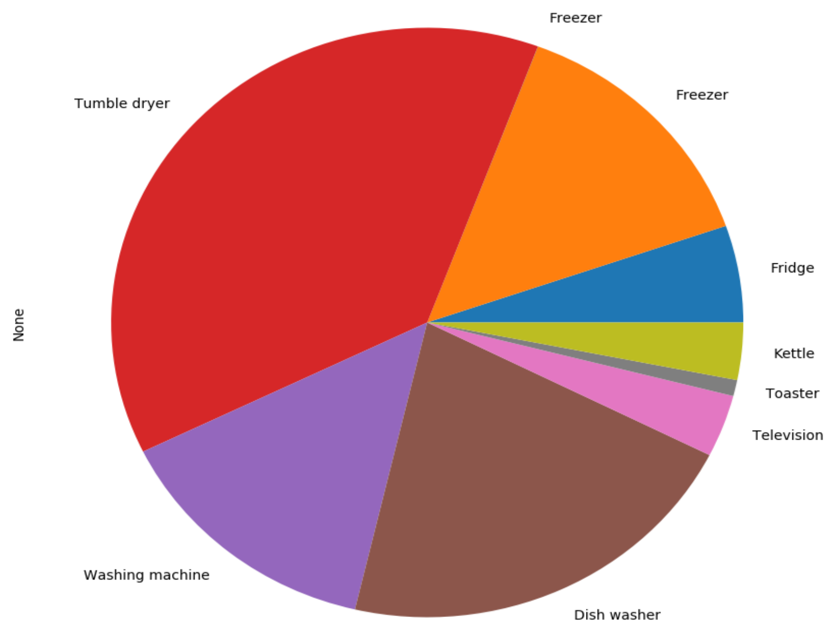



```
refit.buildings[14].elec[9].plot(ax=None, timeframe=None, plot_legend=True,
unit='W', width=100)

<matplotlib.axes._subplots.AxesSubplot at 0x170cae8d0d0>
```

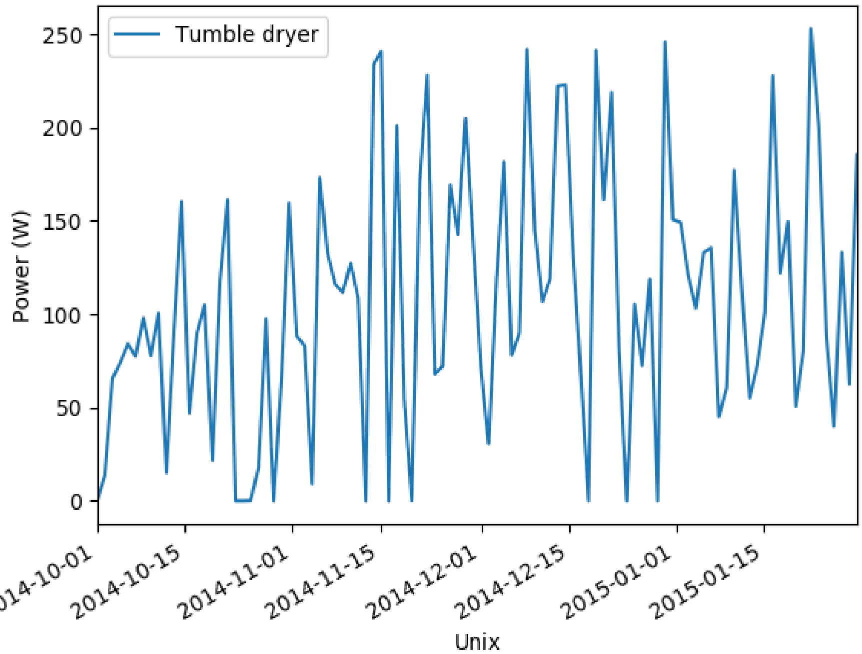


We also had a look on all the appliances during wintertime, but just household 7 shows an interesting change in the proportion of its items:

```
refit.set_window(start='2014-10-01', end='2015-01-31')

fraction_7 =
refit.buildings[7].elec.submeters().fraction_per_meter().dropna()
labels_7 = refit.buildings[7].elec.get_labels(fraction_7.index)
plt.figure(figsize=(10,30))
fraction_7.plot(kind='pie', labels=labels_7)

9/9 ElecMeter(instance=10, building=7, dataset='REFIT',
appliances=[Appliance(type='kettle', instance=1)]))]))1)])

<matplotlib.axes._subplots.AxesSubplot at 0x2819f545430>
```




```
refit.buildings[7].elec.submeters().fraction_per_meter()

9/9 ElecMeter(instance=10, building=7, dataset='REFIT',
appliances=[Appliance(type='kettle', instance=1)]))]))1)])

(2, 7, REFIT)     0.053146
(3, 7, REFIT)     0.139833
(4, 7, REFIT)     0.000077
(5, 7, REFIT)     0.379108
(6, 7, REFIT)     0.141110
(7, 7, REFIT)     0.212471
(8, 7, REFIT)     0.034119
(9, 7, REFIT)     0.008816
(10, 7, REFIT)    0.031320
dtype: float64

refit.buildings[7].elec[5].plot(ax=None, timeframe=None, plot_legend=True,
unit='W', width=100)

<matplotlib.axes._subplots.AxesSubplot at 0x170caf3cdf0>
```



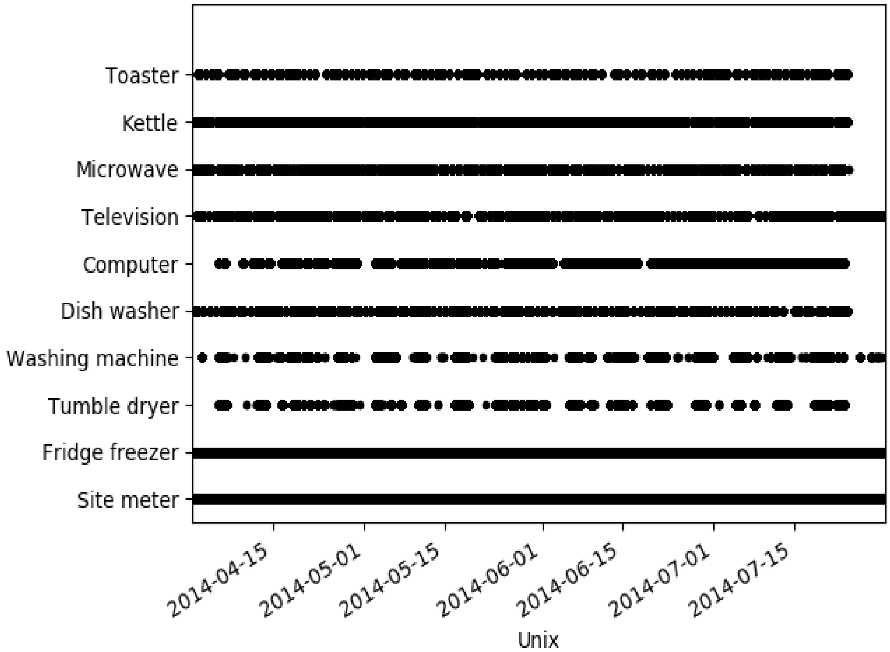
The tumble dryer (instance 5) was used very intensive in building 7 during October until end of January.

But lets go back to summertime:

```
refit.set_window(start='2014-04-01', end='2014-07-31')
```

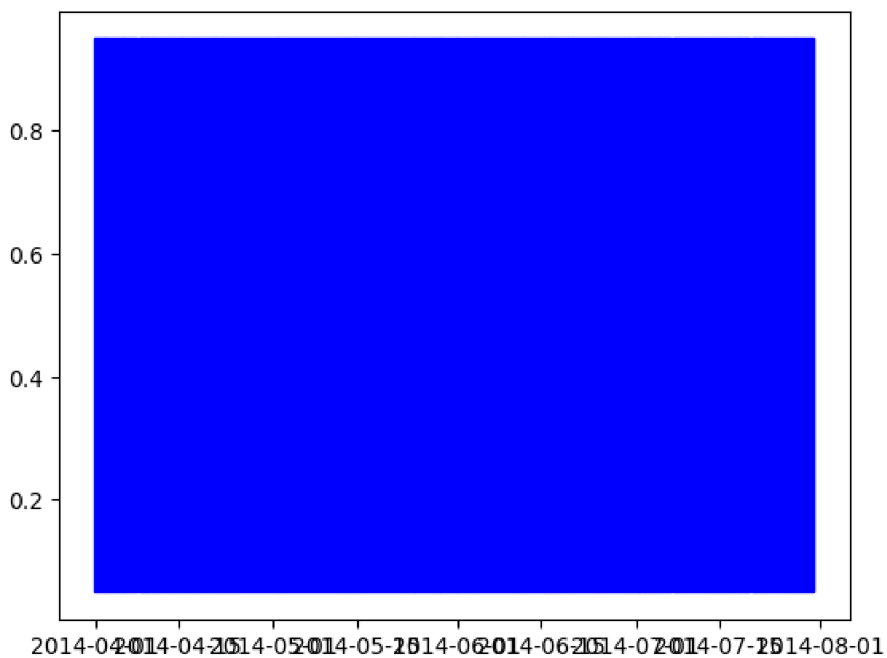
We will focus on building 5 again…

```
refit.buildings[5].elec.plot_when_on(on_power_threshold=40)

<matplotlib.axes._subplots.AxesSubplot at 0x2819a64fd90>
```



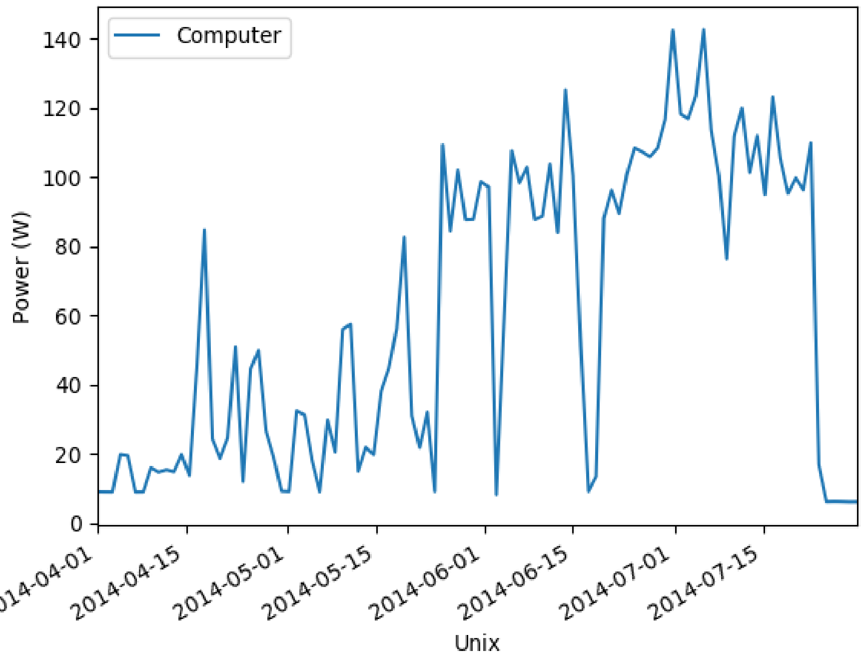
…and in detail on the computer…

```
refit.buildings[5].elec['computer'].good_sections(full_results=True).plot()
WARNING: search terms match 1 appliances. Instance 0 was selected
<matplotlib.axes._subplots.AxesSubplot at 0x2819ce74e50>
```



This looks fine.

```
refit.buildings[5].elec[6].plot(ax=None, timeframe=None, plot_legend=True, unit='W', width=100)

<matplotlib.axes._subplots.AxesSubplot at 0x170caa74730>
```



Checking the other appliances in a similiar way we are now ready to start with modelling…

Part II: Machine Learning

Google Colab Setup

Since training the models is quite expensive and time consuming on regular CPUs, we moved the training process to Google Colab using GPUs.

```
from google.colab import drive
drive.mount('/content/drive/')

Mounted at /content/drive/
```

First, we installed all requirements of the project without `nilmtk` and `nilm-metadata`.

```
!pip install -r ./drive/MyDrive/Energy/req_all_but_nilmtk.txt
```

Next, we cloned the repos of `nilmtk` and `nilm_metadata` from GitHub to install the packages from the folder as editables. The reason behind this is that the time consuming installation of `nilmtk` wastes “computing units” without doing any meaningful computations.

```
!pip install -e ./drive/MyDrive/Energy/packages/nilm_metadata
```

For `nilmtk`, we removed the outdated pins on `numpy`, `pandas`, `matplotlib` and `networkx` within `setup.py` to fasten the installation process from 60 minutes down to a few minutes, which saves “computing units” on Colab. Most of the time was used to build a wheel of the outdated `pandas=0.25.3`.

```
# nilmtk > setup.py
setup(
    # ...
    install_requires=[
        "pandas", #"pandas==0.25.3",
        "numpy",  # "numpy >= 1.13.3, < 1.20.0",
        "networkx",  #"networkx==2.1",
        "scipy",
        "tables",
        "scikit-learn>=0.21.2",
        "hmmlearn>=0.2.1",
        "pyyaml",
        "matplotlib", #"matplotlib==3.1.3",
        "jupyterlab"
    ],
    # ...
)

# Run the line below twice to get nilmtk installed
!pip install -e ./drive/MyDrive/Energy/packages/nilmtk
!pip install -e ./drive/MyDrive/Energy/packages/nilmtk
```

After the installation, *restart the Colab Runtime*.

Moreover, we have a Python version mismatch on Colab. `nilmtk` wants us to have `python=3.8`, but Colab uses `python=3.9+`. The `networkx` package might lead to problems, when `dag.py` is called, since `gcd` moved:

```
# networkx > dag.py
# ...
from fractions import gcd    # for python 3.8
from math import gcd         # for python 3.9+
# ...
```

An update to `dag.py` is necessary, if the code below shows an error. *Restart the Colab Runtime afterwards.*

```
# shouldn't error if the steps above are followed
from nilmtk import DataSet
```

NILMTK API

It's possible to use custom hand-crafted deep learning models and training procedures. However, `nilmtk` comes pre-packaged with a quite useful model training API, which we are going to use.

```
# Check if Google Drive is used
from pathlib import Path
gdrive = Path("./drive/MyDrive/Energy/data").exists()

# Load Data
from nilmtk import DataSet

data_path = "./drive/MyDrive/Energy/data" if gdrive else "./data"
file_path = f"{data_path}/REFIT.h5"  # google drive
refit = DataSet(file_path)

# Helper function
def ndir(x):
    """ Show properties and methods with no magic methods """
    return [x for x in dir(x) if not x.__contains__("__")]

# Load API and joblib (more efficient pickle replacement)
from nilmtk.api import API
import joblib
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
```

Models

Unfortunately, we have failed to install `nilmtk-contrib`, most likely due to a versioning problem of `nilmtk`. As a work-around, we have copied the three models `seq2point`, `seq2seq` and `BERT` verbatim from the GitHub repository. In the `BERT` code, we made slight adjustments to the import statements of `keras`, to reflect our newer version of `tensorflow`.

Also, we added a learning rate parameter for the Adam optimizer, to see if variations of the learning rate led to better results.

Looking at the code of the models, we see that the models take care of any pre-processing themselves.

```
# bert.py
"""
This code is copied verbatim from the nitlmk-contrib repo
https://github.com/nilmtk/nilmtk-
contrib/blob/master/nilmtk_contrib/disaggregate/bert.py

LICENCE: Apache License 2.0

-- Changes made --

* Due to the error:
    AttributeError: module 'tensorflow.compat.v2.__internal__' has no
attribute 'dispatch'
  => We switched the imports `from keras` to `from tensorflow.keras`

* Changed the variable file_path for the weights to reflect appliance name

* Added learning rate parameter, updated optimizer in model.compile()
"""

from __future__ import print_function, division
from warnings import warn

from nilmtk.disaggregate import Disaggregator
from tensorflow.keras.layers import Conv1D, Dense, Dropout, Reshape
from tensorflow.keras.layers import Flatten,Input,GlobalAveragePooling1D,
AveragePooling1D
import os
import pandas as pd
import numpy as np
import pickle
from collections import OrderedDict

from tensorflow.keras.optimizers import SGD
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import
Layer,MultiHeadAttention,LayerNormalization,Embedding
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow.keras.backend as K
import random
random.seed(10)
np.random.seed(10)
import tensorflow as tf
```

```
gpus=tf.config.experimental.list_physical_devices("GPU")
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu,True)

class SequenceLengthError(Exception):
    pass

class ApplianceNotFoundError(Exception):
    pass

#This code is inspired from :
# https://github.com/keras-team/keras-
io/blob/master/examples/nlp/text_classification_with_transformer.py

class TransformerBlock(Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super(TransformerBlock, self).__init__()
        self.att = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
        self.ffn = Sequential(
            [Dense(ff_dim, activation="relu"), Dense(embed_dim),]
        )
        self.layernorm1 = LayerNormalization(epsilon=1e-6)
        self.layernorm2 = LayerNormalization(epsilon=1e-6)
        self.dropout1 = Dropout(rate)
        self.dropout2 = Dropout(rate)

    def call(self, inputs, training):
        attn_output,att_weights = self.att(inputs,
inputs,return_attention_scores=True)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)

    def get_config(self):
        config = super().get_config().copy()
        config.update({
            'att'       : self.att,
            'ffn'       : self.ffn,
            'layernorm1': self.layernorm1,
            'layernorm2': self.layernorm2,
            'dropout1': self.dropout1,
            'dropout2': self.dropout2,
        })
        return config

class TokenAndPositionEmbedding(Layer):
    def __init__(self, maxlen, vocab_size, embed_dim):
        super(TokenAndPositionEmbedding, self).__init__()
```

```
        self.token_emb = Embedding(input_dim=vocab_size,
output_dim=embed_dim)
        self.pos_emb = Embedding(input_dim=maxlen, output_dim=embed_dim)

    def call(self, x):
        maxlen = tf.shape(x)[-1]
        positions = tf.range(start=0, limit=maxlen, delta=1)
        positions = self.pos_emb(positions)
        x = self.token_emb(x)
        return x + positions

    def get_config(self):
        config = super().get_config().copy()
        config.update({
            'token_emb' : self.token_emb,
            'pos_emb' : self.pos_emb,
        })
        return config


class LPpool(Layer):
    def __init__(self, pool_size, strides=None, padding='same'):
        super(LPpool,self).__init__()

self.avgpool=tf.keras.layers.AveragePooling1D(pool_size,strides,padding)

    def call(self, x):
        x = tf.math.pow(tf.math.abs(x), 2)
        x = self.avgpool(x)
        x = tf.math.pow(x, 1.0 / 2)
        return x

    def get_config(self):
        config = super().get_config().copy()
        config.update({
            'avgpool'       : self.avgpool,
        })
        return config

class BERT(Disaggregator):

    def __init__(self, params):

        self.MODEL_NAME = "BERT"
        self.chunk_wise_training = params.get('chunk_wise_training',False)
        self.sequence_length = params.get('sequence_length',99)
        self.n_epochs = params.get('n_epochs', 10)
        self.models = OrderedDict()
        self.mains_mean = 1800
        self.mains_std = 600
```

```
        self.batch_size = params.get('batch_size',512)
        self.appliance_params = params.get('appliance_params',{})
        if self.sequence_length%2==0:
            print ("Sequence length should be odd!")
            raise (SequenceLengthError)
        self.learning_rate = 0.001    # added

    def
partial_fit(self,train_main,train_appliances,do_preprocessing=True,**load_kwa
rgs):

        print("...............BERT partial_fit running...............")
        if len(self.appliance_params) == 0:
            self.set_appliance_params(train_appliances)

        if do_preprocessing:
            train_main, train_appliances = self.call_preprocessing(
                train_main, train_appliances, 'train')
        train_main = pd.concat(train_main,axis=0)
        train_main = train_main.values.reshape((-1,self.sequence_length,1))

        new_train_appliances = []
        for app_name, app_dfs in train_appliances:
            app_df = pd.concat(app_dfs,axis=0)
            app_df_values = app_df.values.reshape((-1,self.sequence_length))
            new_train_appliances.append((app_name, app_df_values))
        train_appliances = new_train_appliances

        for appliance_name, power in train_appliances:
            if appliance_name not in self.models:
                print("First model training for ", appliance_name)
                self.models[appliance_name] = self.return_network()
            else:
                print("Started Retraining model for ", appliance_name)

            model = self.models[appliance_name]
            if train_main.size > 0:
                # Sometimes chunks can be empty after dropping NANS
                if len(train_main) > 10:
                    # Do validation when you have sufficient samples
                    filepath = 'BERT-'+"_".join(appliance_name.split())+'.h5'
# change
                    checkpoint =
ModelCheckpoint(filepath,monitor='val_loss',verbose=1,save_best_only=True,mod
e='min')
                    train_x, v_x, train_y, v_y = train_test_split(train_main,
power, test_size=.15,random_state=10)

model.fit(train_x,train_y,validation_data=(v_x,v_y),epochs=self.n_epochs,call
backs=[checkpoint],batch_size=self.batch_size)
                    model.load_weights(filepath)
```

```
    def
disaggregate_chunk(self,test_main_list,model=None,do_preprocessing=True):

        if model is not None:
            self.models = model

        if do_preprocessing:
            test_main_list = self.call_preprocessing(
                test_main_list, submeters_lst=None, method='test')

        test_predictions = []
        for test_mains_df in test_main_list:

            disggregation_dict = {}
            test_main_array = test_mains_df.values.reshape((-1,
self.sequence_length, 1))

            for appliance in self.models:

                prediction = []
                model = self.models[appliance]
                prediction = model.predict(test_main_array
,batch_size=self.batch_size)

                #####################
                # This block is for creating the average of predictions over
the different sequences
                # the counts_arr keeps the number of times a particular
timestamp has occured
                # the sum_arr keeps the number of times a particular
timestamp has occured
                # the predictions are summed for  agiven time, and is divided
by the number of times it has occured

                l = self.sequence_length
                n = len(prediction) + l - 1
                sum_arr = np.zeros((n))
                counts_arr = np.zeros((n))
                o = len(sum_arr)
                for i in range(len(prediction)):
                    sum_arr[i:i + l] += prediction[i].flatten()
                    counts_arr[i:i + l] += 1
                for i in range(len(sum_arr)):
                    sum_arr[i] = sum_arr[i] / counts_arr[i]

                #################
                prediction = self.appliance_params[appliance]['mean'] +
(sum_arr * self.appliance_params[appliance]['std'])
                valid_predictions = prediction.flatten()
```

```
                valid_predictions = np.where(valid_predictions > 0,
valid_predictions, 0)
                df = pd.Series(valid_predictions)
                disggregation_dict[appliance] = df
            results = pd.DataFrame(disggregation_dict, dtype='float32')
            test_predictions.append(results)
        return test_predictions

    def return_network(self):
        '''Creates the BERT module
        '''
        embed_dim = 32  # Embedding size for each token
        num_heads = 2  # Number of attention heads
        ff_dim = 32  # Hidden layer size in feed forward network inside
transformer
        vocab_size = 20000 #vocab for different patterns in reading
        maxlen = self.sequence_length  #maxlength for attention

        model = Sequential()

model.add(Conv1D(16,4,activation="linear",input_shape=(self.sequence_length,1
),padding="same",strides=1))
        model.add(LPpool(pool_size=2))

        #Token and Positional embedding and Encoder part of the transformer
        model.add(TokenAndPositionEmbedding(maxlen, vocab_size, embed_dim))
        model.add(TransformerBlock(embed_dim, num_heads, ff_dim))

        #Fully connected layer
        model.add(Flatten())
        model.add(Dropout(0.1))
        model.add(Dense(self.sequence_length))
        model.add(Dropout(0.1))
        model.summary()
        model.compile(loss='mse',
                      optimizer=tf.keras.optimizers.Adam(self.learning_rate),
# changed
                      metrics=['mse'])
        return model

    def call_preprocessing(self, mains_lst, submeters_lst, method):

        if method == 'train':
            processed_mains_lst = []
            for mains in mains_lst:
                new_mains = mains.values.flatten()
                n = self.sequence_length
                units_to_pad = n // 2
                new_mains = np.pad(new_mains,
(units_to_pad,units_to_pad),'constant',constant_values = (0,0))
                new_mains = np.array([new_mains[i:i + n] for i in
```

```
range(len(new_mains) - n + 1)])
                new_mains = (new_mains - self.mains_mean) / self.mains_std
                processed_mains_lst.append(pd.DataFrame(new_mains))
            appliance_list = []
            for app_index, (app_name, app_df_lst) in
enumerate(submeters_lst):

                if app_name in self.appliance_params:
                    app_mean = self.appliance_params[app_name]['mean']
                    app_std = self.appliance_params[app_name]['std']
                else:
                    print ("Parameters for ", app_name ," were not found!")
                    raise ApplianceNotFoundError()


                processed_app_dfs = []
                for app_df in app_df_lst:
                    new_app_readings = app_df.values.flatten()
                    new_app_readings = np.pad(new_app_readings,
(units_to_pad,units_to_pad),'constant',constant_values = (0,0))
                    new_app_readings = np.array([new_app_readings[i:i + n]
for i in range(len(new_app_readings) - n + 1)])
                    new_app_readings = (new_app_readings - app_mean) /
app_std  # /self.max_val
                    processed_app_dfs.append(pd.DataFrame(new_app_readings))


                appliance_list.append((app_name, processed_app_dfs))


            return processed_mains_lst, appliance_list

        else:
            processed_mains_lst = []
            for mains in mains_lst:
                new_mains = mains.values.flatten()
                n = self.sequence_length
                units_to_pad = n // 2
                #new_mains = np.pad(new_mains,
(units_to_pad,units_to_pad),'constant',constant_values = (0,0))
                new_mains = np.array([new_mains[i:i + n] for i in
range(len(new_mains) - n + 1)])
                new_mains = (new_mains - self.mains_mean) / self.mains_std
                new_mains = new_mains.reshape((-1, self.sequence_length))
                processed_mains_lst.append(pd.DataFrame(new_mains))
            return processed_mains_lst

    def set_appliance_params(self,train_appliances):

        for (app_name,df_list) in train_appliances:
```

```
            l = np.array(pd.concat(df_list,axis=0))
            app_mean = np.mean(l)
            app_std = np.std(l)
            if app_std<1:
                app_std = 100

self.appliance_params.update({app_name:{'mean':app_mean,'std':app_std}})

# seq2seq.py
"""
This code is copied verbatim from the nitlmk-contrib repo
https://github.com/nilmtk/nilmtk-
contrib/blob/master/nilmtk_contrib/disaggregate/seq2seq.py

LICENCE: Apache License 2.0

-- Changes made: --
* Added learning rate parameter, updated optimizer in model.compile()

"""

from collections import OrderedDict
import numpy as np
import pandas as pd
from nilmtk.disaggregate import Disaggregator
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Conv1D, Dense, Dropout, Flatten
from tensorflow.keras.models import Sequential


class SequenceLengthError(Exception):
    pass

class ApplianceNotFoundError(Exception):
    pass



class Seq2Seq(Disaggregator):

    def __init__(self, params):

        self.MODEL_NAME = "Seq2Seq"
        self.file_prefix = "{}-temp-weights".format(self.MODEL_NAME.lower())
        self.chunk_wise_training = params.get('chunk_wise_training',False)
        self.sequence_length = params.get('sequence_length',99)
        self.n_epochs = params.get('n_epochs', 10)
        self.models = OrderedDict()
        self.mains_mean = 1800
        self.mains_std = 600
```

```
        self.batch_size = params.get('batch_size',512)
        self.appliance_params = params.get('appliance_params',{})
        if self.sequence_length%2==0:
            print ("Sequence length should be odd!")
            raise (SequenceLengthError)
        self.learning_rate = 0.001    # added

    def partial_fit(self, train_main, train_appliances,
do_preprocessing=True, current_epoch=0, **load_kwargs):
        print("...............Seq2Seq partial_fit running...............")
        if len(self.appliance_params) == 0:
            self.set_appliance_params(train_appliances)

        if do_preprocessing:
            train_main, train_appliances = self.call_preprocessing(
                train_main, train_appliances, 'train')

        train_main = pd.concat(train_main, axis=0)
        train_main = train_main.values.reshape((-1, self.sequence_length, 1))
        new_train_appliances = []
        for app_name, app_dfs in train_appliances:
            app_df = pd.concat(app_dfs, axis=0)
            app_df_values = app_df.values.reshape((-1, self.sequence_length))
            new_train_appliances.append((app_name, app_df_values))

        train_appliances = new_train_appliances
        for appliance_name, power in train_appliances:
            if appliance_name not in self.models:
                print("First model training for ", appliance_name)
                self.models[appliance_name] = self.return_network()
            else:
                print("Started Retraining model for ", appliance_name)

            model = self.models[appliance_name]
            if train_main.size > 0:
                # Sometimes chunks can be empty after dropping NANS
                if len(train_main) > 10:
                    # Do validation when you have sufficient samples
                    filepath = self.file_prefix + "-{}-epoch{}.h5".format(
                            "_".join(appliance_name.split()),
                            current_epoch,
                    )
                    checkpoint =
ModelCheckpoint(filepath,monitor='val_loss',verbose=1,save_best_only=True,mod
e='min')
                    model.fit(
                            train_main, power,
                            validation_split=.15,
                            epochs=self.n_epochs,
                            batch_size=self.batch_size,
                            callbacks=[ checkpoint ],
```

```
                    )
                    model.load_weights(filepath)


    def
disaggregate_chunk(self,test_main_list,model=None,do_preprocessing=True):
        if model is not None:
            self.models = model

        if do_preprocessing:
            test_main_list = self.call_preprocessing(
                test_main_list, submeters_lst=None, method='test')

        test_predictions = []
        for test_mains_df in test_main_list:

            disggregation_dict = {}
            test_main_array = test_mains_df.values.reshape((-1,
self.sequence_length, 1))

            for appliance in self.models:

                prediction = []
                model = self.models[appliance]
                prediction = model.predict(test_main_array
,batch_size=self.batch_size)

                #####################
                # This block is for creating the average of predictions over
the different sequences
                # the counts_arr keeps the number of times a particular
timestamp has occured
                # the sum_arr keeps the number of times a particular
timestamp has occured
                # the predictions are summed for  agiven time, and is divided
by the number of times it has occured

                l = self.sequence_length
                n = len(prediction) + l - 1
                sum_arr = np.zeros((n))
                counts_arr = np.zeros((n))
                o = len(sum_arr)
                for i in range(len(prediction)):
                    sum_arr[i:i + l] += prediction[i].flatten()
                    counts_arr[i:i + l] += 1
                for i in range(len(sum_arr)):
                    sum_arr[i] = sum_arr[i] / counts_arr[i]

                #################
                prediction = self.appliance_params[appliance]['mean'] +
```

```
(sum_arr * self.appliance_params[appliance]['std'])
                valid_predictions = prediction.flatten()
                valid_predictions = np.where(valid_predictions > 0,
valid_predictions, 0)
                df = pd.Series(valid_predictions)
                disggregation_dict[appliance] = df
            results = pd.DataFrame(disggregation_dict, dtype='float32')
            test_predictions.append(results)

        return test_predictions

    def return_network(self):

        model = Sequential()
        # 1D Conv

model.add(Conv1D(30,10,activation="relu",input_shape=(self.sequence_length,1)
,strides=2))
        model.add(Conv1D(30, 8, activation='relu', strides=2))
        model.add(Conv1D(40, 6, activation='relu', strides=1))
        model.add(Conv1D(50, 5, activation='relu', strides=1))
        model.add(Dropout(.2))
        model.add(Conv1D(50, 5, activation='relu', strides=1))
        model.add(Dropout(.2))
        model.add(Flatten())
        model.add(Dense(1024, activation='relu'))
        model.add(Dropout(.2))
        model.add(Dense(self.sequence_length))
        model.compile(loss='mse',
optimizer=tf.keras.optimizers.Adam(self.learning_rate))  # changed

        return model

    def call_preprocessing(self, mains_lst, submeters_lst, method):

        if method == 'train':
            processed_mains_lst = []
            for mains in mains_lst:
                new_mains = mains.values.flatten()
                n = self.sequence_length
                units_to_pad = n // 2
                new_mains = np.pad(new_mains,
(units_to_pad,units_to_pad),'constant',constant_values = (0,0))
                new_mains = np.array([new_mains[i:i + n] for i in
range(len(new_mains) - n + 1)])
                new_mains = (new_mains - self.mains_mean) / self.mains_std
                processed_mains_lst.append(pd.DataFrame(new_mains))
            #new_mains = pd.DataFrame(new_mains)
            appliance_list = []
            for app_index, (app_name, app_df_lst) in
enumerate(submeters_lst):
```

```
                if app_name in self.appliance_params:
                    app_mean = self.appliance_params[app_name]['mean']
                    app_std = self.appliance_params[app_name]['std']
                else:
                    print ("Parameters for ", app_name ," were not found!")
                    raise ApplianceNotFoundError()


                processed_app_dfs = []
                for app_df in app_df_lst:
                    new_app_readings = app_df.values.flatten()
                    new_app_readings = np.pad(new_app_readings,
(units_to_pad,units_to_pad),'constant',constant_values = (0,0))
                    new_app_readings = np.array([new_app_readings[i:i + n]
for i in range(len(new_app_readings) - n + 1)])
                    new_app_readings = (new_app_readings - app_mean) /
app_std  # /self.max_val
                    processed_app_dfs.append(pd.DataFrame(new_app_readings))


                appliance_list.append((app_name, processed_app_dfs))
                #new_app_readings = np.array([ new_app_readings[i:i+n] for i
in range(len(new_app_readings)-n+1) ])
                #print (new_mains.shape, new_app_readings.shape, app_name)

            return processed_mains_lst, appliance_list

        else:
            processed_mains_lst = []
            for mains in mains_lst:
                new_mains = mains.values.flatten()
                n = self.sequence_length
                units_to_pad = n // 2
                #new_mains = np.pad(new_mains,
(units_to_pad,units_to_pad),'constant',constant_values = (0,0))
                new_mains = np.array([new_mains[i:i + n] for i in
range(len(new_mains) - n + 1)])
                new_mains = (new_mains - self.mains_mean) / self.mains_std
                new_mains = new_mains.reshape((-1, self.sequence_length))
                processed_mains_lst.append(pd.DataFrame(new_mains))
            return processed_mains_lst

    def set_appliance_params(self,train_appliances):

        for (app_name,df_list) in train_appliances:
            l = np.array(pd.concat(df_list,axis=0))
            app_mean = np.mean(l)
            app_std = np.std(l)
            if app_std<1:
```

```
                app_std = 100

self.appliance_params.update({app_name:{'mean':app_mean,'std':app_std}})

# seq2point.py
"""
This code is copied verbatim from the nitlmk-contrib repo
https://github.com/nilmtk/nilmtk-
contrib/blob/master/nilmtk_contrib/disaggregate/seq2point.py

LICENCE: Apache License 2.0

-- Changes made: --
* Added learning rate parameter, updated optimizer in model.compile()

"""

from collections import OrderedDict
import numpy as np
import pandas as pd
from nilmtk.disaggregate import Disaggregator
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Conv1D, Dense, Dropout, Reshape, Flatten
from tensorflow.keras.models import Sequential


class SequenceLengthError(Exception):
    pass

class ApplianceNotFoundError(Exception):
    pass

class Seq2Point(Disaggregator):

    def __init__(self, params):
        """
        Parameters to be specified for the model
        """

        self.MODEL_NAME = "Seq2Point"
        self.models = OrderedDict()
        self.file_prefix = "{}-temp-weights".format(self.MODEL_NAME.lower())
        self.chunk_wise_training = params.get('chunk_wise_training',False)
        self.sequence_length = params.get('sequence_length',99)
        self.n_epochs = params.get('n_epochs', 10 )
        self.batch_size = params.get('batch_size',512)
        self.appliance_params = params.get('appliance_params',{})
        self.mains_mean = params.get('mains_mean',1800)
        self.mains_std = params.get('mains_std',600)
        if self.sequence_length%2==0:
```

```
            print ("Sequence length should be odd!")
            raise (SequenceLengthError)
        self.learning_rate = 0.001    # added

    def partial_fit(self, train_main, train_appliances,
do_preprocessing=True, current_epoch=0, **load_kwargs):
        # If no appliance wise parameters are provided, then copmute them
using the first chunk
        if len(self.appliance_params) == 0:
            self.set_appliance_params(train_appliances)

        print("...............Seq2Point partial_fit running...............")
        # Do the pre-processing, such as  windowing and normalizing
        if do_preprocessing:
            train_main, train_appliances = self.call_preprocessing(
                train_main, train_appliances, 'train')

        train_main = pd.concat(train_main, axis=0)
        train_main = train_main.values.reshape((-1, self.sequence_length, 1))
        new_train_appliances = []
        for app_name, app_df in train_appliances:
            app_df = pd.concat(app_df, axis=0)
            app_df_values = app_df.values.reshape((-1, 1))
            new_train_appliances.append((app_name, app_df_values))
        train_appliances = new_train_appliances

        for appliance_name, power in train_appliances:
            # Check if the appliance was already trained. If not then create
a new model for it
            if appliance_name not in self.models:
                print("First model training for", appliance_name)
                self.models[appliance_name] = self.return_network()
            # Retrain the particular appliance
            else:
                print("Started Retraining model for", appliance_name)

            model = self.models[appliance_name]
            if train_main.size > 0:
                # Sometimes chunks can be empty after dropping NANS
                if len(train_main) > 10:
                    # Do validation when you have sufficient samples
                    filepath = self.file_prefix + "-{}-epoch{}.h5".format(
                            "_".join(appliance_name.split()),
                            current_epoch,
                    )
                    checkpoint =
ModelCheckpoint(filepath,monitor='val_loss',verbose=1,save_best_only=True,mod
e='min')
                    model.fit(
                            train_main, power,
                            validation_split=0.15,
```

```
                            epochs=self.n_epochs,
                            batch_size=self.batch_size,
                            callbacks=[checkpoint],
                    )
                    model.load_weights(filepath)


    def
disaggregate_chunk(self,test_main_list,model=None,do_preprocessing=True):
        if model is not None:
            self.models = model

        # Preprocess the test mains such as windowing and normalizing

        if do_preprocessing:
            test_main_list = self.call_preprocessing(test_main_list,
submeters_lst=None, method='test')

        test_predictions = []
        for test_main in test_main_list:
            test_main = test_main.values
            test_main = test_main.reshape((-1, self.sequence_length, 1))
            disggregation_dict = {}
            for appliance in self.models:
                prediction =
self.models[appliance].predict(test_main,batch_size=self.batch_size)
                prediction = self.appliance_params[appliance]['mean'] +
prediction * self.appliance_params[appliance]['std']
                valid_predictions = prediction.flatten()
                valid_predictions = np.where(valid_predictions > 0,
valid_predictions, 0)
                df = pd.Series(valid_predictions)
                disggregation_dict[appliance] = df
            results = pd.DataFrame(disggregation_dict, dtype='float32')
            test_predictions.append(results)
        return test_predictions

    def return_network(self):
        # Model architecture
        model = Sequential()

model.add(Conv1D(30,10,activation="relu",input_shape=(self.sequence_length,1)
,strides=1))
        model.add(Conv1D(30, 8, activation='relu', strides=1))
        model.add(Conv1D(40, 6, activation='relu', strides=1))
        model.add(Conv1D(50, 5, activation='relu', strides=1))
        model.add(Dropout(.2))
        model.add(Conv1D(50, 5, activation='relu', strides=1))
        model.add(Dropout(.2))
        model.add(Flatten())
        model.add(Dense(1024, activation='relu'))
```

```
        model.add(Dropout(.2))
        model.add(Dense(1))
        model.compile(loss='mse',
optimizer=tf.keras.optimizers.Adam(self.learning_rate))  #
,metrics=[self.mse])
        return model

    def call_preprocessing(self, mains_lst, submeters_lst, method):

        if method == 'train':
            # Preprocessing for the train data
            mains_df_list = []
            for mains in mains_lst:
                new_mains = mains.values.flatten()
                n = self.sequence_length
                units_to_pad = n // 2
                new_mains =
np.pad(new_mains,(units_to_pad,units_to_pad),'constant',constant_values=(0,0)
)
                new_mains = np.array([new_mains[i:i + n] for i in
range(len(new_mains) - n + 1)])
                new_mains = (new_mains - self.mains_mean) / self.mains_std
                mains_df_list.append(pd.DataFrame(new_mains))

            appliance_list = []
            for app_index, (app_name, app_df_list) in
enumerate(submeters_lst):
                if app_name in self.appliance_params:
                    app_mean = self.appliance_params[app_name]['mean']
                    app_std = self.appliance_params[app_name]['std']
                else:
                    print ("Parameters for ", app_name ," were not found!")
                    raise ApplianceNotFoundError()

                processed_appliance_dfs = []

                for app_df in app_df_list:
                    new_app_readings = app_df.values.reshape((-1, 1))
                    # This is for choosing windows
                    new_app_readings = (new_app_readings - app_mean) /
app_std
                    # Return as a list of dataframe

processed_appliance_dfs.append(pd.DataFrame(new_app_readings))
                appliance_list.append((app_name, processed_appliance_dfs))
            return mains_df_list, appliance_list

        else:
            # Preprocessing for the test data
            mains_df_list = []
```

```
            for mains in mains_lst:
                new_mains = mains.values.flatten()
                n = self.sequence_length
                units_to_pad = n // 2
                new_mains = 
np.pad(new_mains,(units_to_pad,units_to_pad),'constant',constant_values=(0,0)
)
                new_mains = np.array([new_mains[i:i + n] for i in 
range(len(new_mains) - n + 1)])
                new_mains = (new_mains - self.mains_mean) / self.mains_std
                mains_df_list.append(pd.DataFrame(new_mains))
            return mains_df_list

    def set_appliance_params(self,train_appliances):
        # Find the parameters using the first
        for (app_name,df_list) in train_appliances:
            l = np.array(pd.concat(df_list,axis=0))
            app_mean = np.mean(l)
            app_std = np.std(l)
            if app_std<1:
                app_std = 100

self.appliance_params.update({app_name:{'mean':app_mean,'std':app_std}})
        print (self.appliance_params)
```

Structure of the models:

- `Seq2Seq` is the smallest of the three models with about 447k parameters.
- According to the summary `BERT` is the second largest model with 3.1M parameters. Though we are not sure whether `keras` is calculating the number of parameters correctly as it has some custom layers.
- The largest model seems to be the `Seq2Point` model with 3.6M parameters
- All models make heavy use of convolutional layers which work very well for computer vision tasks.
- The `BERT` model uses a transformer architecture with the so-called “attention” mechanism

```
Seq2Point({"n_epochs": 5, "learning_rate": 0.001}).return_network().summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1d_5 (Conv1D)           (None, 90, 30)            330       
                                                                 
 conv1d_6 (Conv1D)           (None, 83, 30)            7230      
                                                                 
 conv1d_7 (Conv1D)           (None, 78, 40)            7240      
                                                                 
 conv1d_8 (Conv1D)           (None, 74, 50)            10050     
```

```
 dropout_3 (Dropout)          (None, 74, 50)           0

 conv1d_9 (Conv1D)            (None, 70, 50)           12550

 dropout_4 (Dropout)          (None, 70, 50)           0

 flatten_1 (Flatten)          (None, 3500)             0

 dense_2 (Dense)              (None, 1024)             3585024

 dropout_5 (Dropout)          (None, 1024)             0

 dense_3 (Dense)              (None, 1)                1025

=================================================================
Total params: 3,623,449
Trainable params: 3,623,449
Non-trainable params: 0
_________________________________________________________________
```

```
Seq2Seq({"n_epochs": 5, "learning_rate": 0.001}).return_network().summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                 Output Shape             Param #
=================================================================
 conv1d_10 (Conv1D)           (None, 45, 30)           330

 conv1d_11 (Conv1D)           (None, 19, 30)           7230

 conv1d_12 (Conv1D)           (None, 14, 40)           7240

 conv1d_13 (Conv1D)           (None, 10, 50)           10050

 dropout_6 (Dropout)          (None, 10, 50)           0

 conv1d_14 (Conv1D)           (None, 6, 50)            12550

 dropout_7 (Dropout)          (None, 6, 50)            0

 flatten_2 (Flatten)          (None, 300)              0

 dense_4 (Dense)              (None, 1024)             308224

 dropout_8 (Dropout)          (None, 1024)             0

 dense_5 (Dense)              (None, 99)               101475

=================================================================
```

```
Total params: 447,099
Trainable params: 447,099
Non-trainable params: 0
_________________________________________________________________
```

```
BERT({"n_epochs": 5, "learning_rate": 0.001}).return_network().summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1d_15 (Conv1D)          (None, 99, 16)            80        
                                                                 
 l_ppool (LPpool)            (None, 50, 16)            0         
                                                                 
 token_and_position_embeddin  (None, 50, 16, 32)       643168    
 g (TokenAndPositionEmbeddin                                     
 g)                                                              
                                                                 
 transformer_block (Transfor  (None, 50, 16, 32)       10656     
 merBlock)                                                       
                                                                 
 flatten_3 (Flatten)         (None, 25600)             0         
                                                                 
 dropout_11 (Dropout)        (None, 25600)             0         
                                                                 
 dense_8 (Dense)             (None, 99)                2534499   
                                                                 
 dropout_12 (Dropout)        (None, 99)                0         
                                                                 
=================================================================
Total params: 3,188,403
Trainable params: 3,188,403
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1d_15 (Conv1D)          (None, 99, 16)            80        
                                                                 
 l_ppool (LPpool)            (None, 50, 16)            0         
                                                                 
 token_and_position_embeddin  (None, 50, 16, 32)       643168    
 g (TokenAndPositionEmbeddin                                     
 g)                                                              
                                                                 
 transformer_block (Transfor  (None, 50, 16, 32)       10656     
 merBlock)                                                       
```

```
 flatten_3 (Flatten)         (None, 25600)             0

 dropout_11 (Dropout)        (None, 25600)             0

 dense_8 (Dense)             (None, 99)                2534499

 dropout_12 (Dropout)        (None, 99)                0

=================================================================
Total params: 3,188,403
Trainable params: 3,188,403
Non-trainable params: 0
_________________________________________________________________
```

Building 5

The first builiding we tried to analyse was building 5, as it is one of the more interesting ones according to our data analysis. On building 5 we modeled the tumble dryer and the computer.

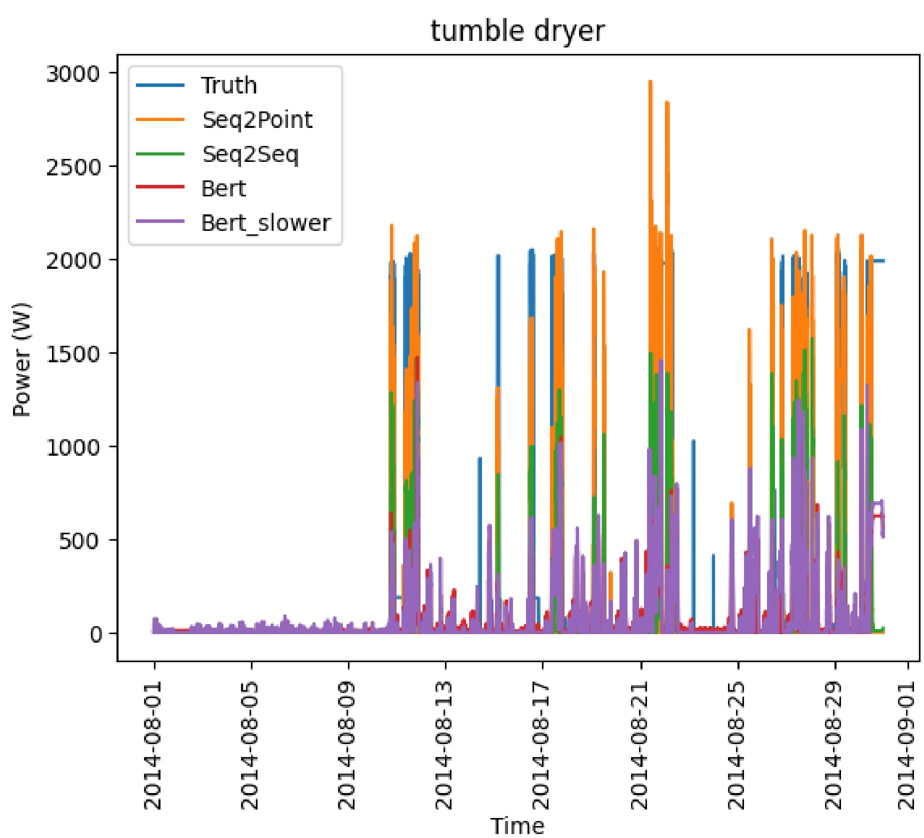
```
refit.buildings[5].elec

MeterGroup(meters=
  ElecMeter(instance=1, building=5, dataset='REFIT', site_meter,
appliances=[])
  ElecMeter(instance=2, building=5, dataset='REFIT',
appliances=[Appliance(type='fridge freezer', instance=1)])
  ElecMeter(instance=3, building=5, dataset='REFIT',
appliances=[Appliance(type='tumble dryer', instance=1)])
  ElecMeter(instance=4, building=5, dataset='REFIT',
appliances=[Appliance(type='washing machine', instance=1)])
  ElecMeter(instance=5, building=5, dataset='REFIT',
appliances=[Appliance(type='dish washer', instance=1)])
  ElecMeter(instance=6, building=5, dataset='REFIT',
appliances=[Appliance(type='computer', instance=1)])
  ElecMeter(instance=7, building=5, dataset='REFIT',
appliances=[Appliance(type='television', instance=1)])
  ElecMeter(instance=8, building=5, dataset='REFIT',
appliances=[Appliance(type='microwave', instance=1)])
  ElecMeter(instance=9, building=5, dataset='REFIT',
appliances=[Appliance(type='kettle', instance=1)])
  ElecMeter(instance=10, building=5, dataset='REFIT',
appliances=[Appliance(type='toaster', instance=1)])
)
```

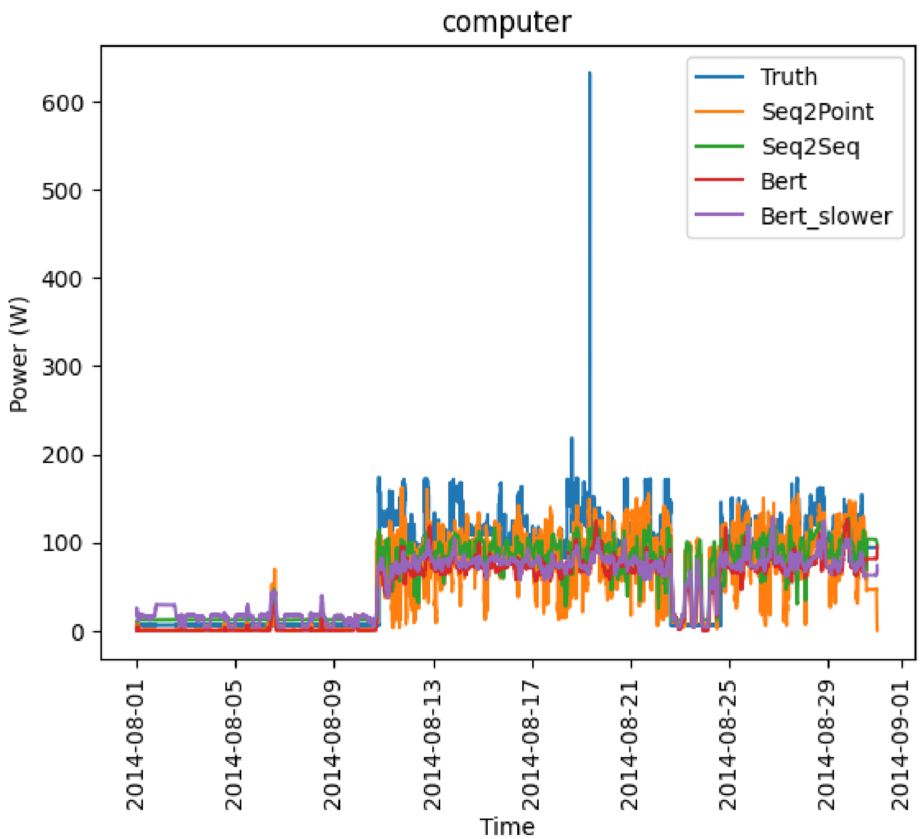
Here, we specified the parameters needed for the API. For the `Seq2Seq` and `Seq2Point` we used 20 epochs and for the computationally intensive `BERT` models we used only 10 epochs each. More epochs could lead to a better model performance, but we were constrained by Colab “computing units” and tried to be economical. The second `BERT` model also has a slower learning rate (reduced by 50%). We wanted to see if it performed better than with the default learning rate.

For the training process we used a sample rate of 60 (which means one data point every minute), a training period of four months (April - July 2014), and a testing period of one month (August 2014).

```
building5_param = {
  "power": {"mains": ["apparent","active"],"appliance":
["apparent","active"]},
  "sample_rate": 60,
  "appliances": [ "tumble dryer", "computer" ],
  "methods": {"Seq2Point": Seq2Point({"n_epochs": 20}),
              "Seq2Seq": Seq2Seq({"n_epochs": 20}),
              "Bert": BERT({"n_epochs": 10}),
              "Bert_slower": BERT({"n_epochs": 10, "learning_rate": 0.0005})
              },
  "display_predictions": True,
  "train": {
    "datasets": {
        "Dataport": {
            "path": file_path,
            "buildings": {
                5: {
                    "start_time": "2014-04-01",
                    "end_time": "2014-07-31"
                    }
                }
            }
        }
    },
  "test": {
    "datasets": {
        "Dataport": {
            "path": file_path,
            "buildings": {
                5: {
                    "start_time": "2014-08-01",
                    "end_time": "2014-08-31"
                    }
                }
            }
        },
        "metrics":["rmse"]
    }
  }

# Model Training. Saving results to a file
if Path(f"{data_path}/building5.joblib").exists() == False:
    building5_mod = API(building5_param)
    results = {
        "pred_overall": building5_mod.pred_overall,
        "errors": building5_mod.errors,
        "test_mains": building5_mod.test_mains,
```

```
        "test_submeters": building5_mod.test_submeters
    }
    with open(f"{data_path}/building5.joblib", "wb") as f:
        joblib.dump(results, f)

Joint Testing for all algorithms
Loading data for  Dataport  dataset
Dropping missing values
Generating predictions for : Seq2Point
85/85 [==============================] - 0s 2ms/step
85/85 [==============================] - 0s 2ms/step
Generating predictions for : Seq2Seq
84/84 [==============================] - 0s 2ms/step
84/84 [==============================] - 0s 2ms/step
Generating predictions for : BERT
84/84 [==============================] - 3s 36ms/step
84/84 [==============================] - 3s 36ms/step
Generating predictions for : BERT
84/84 [==============================] - 3s 36ms/step
84/84 [==============================] - 3s 36ms/step
............  rmse  ..............
               Seq2Point     Seq2Seq        Bert  Bert_slower
tumble dryer  405.726374  400.311207  386.986838   387.437535
computer       31.641851   28.285084   30.947019    30.834153
```






```
with open(f"{data_path}/building5.joblib", "rb") as f:
    results = joblib.load(f)
print(results["errors"])

[                Seq2Point     Seq2Seq        Bert  Bert_slower
tumble dryer  405.726374  400.311207  386.986838   387.437535
computer       31.641851   28.285084   30.947019    30.834153]
```

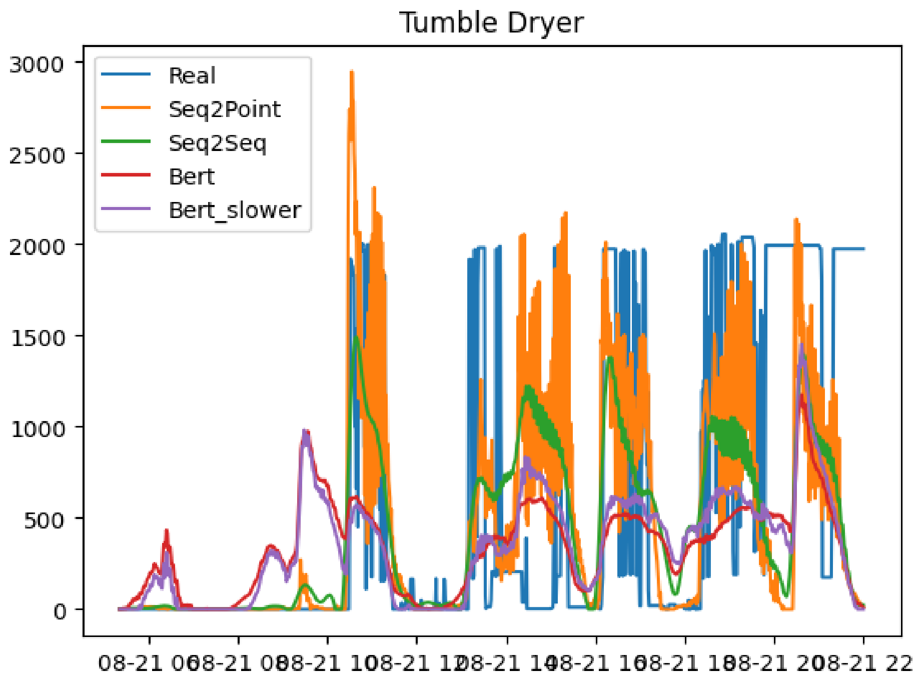
Above we reported the rmse errors. The regular BERT performed best for the tumble dryer, while the Seq2Seq model was best for the computer. Interestingly, the slower learning rate for the BERT didn´t really materialize into a significantly better prediction performance.

Looking at the plots the sequence models Seq2Point and Seq2Seq seemed to better capture the spikes (variance), while the BERT predictions had less variance.

Zooming into an arbitrary window leads to the following plots:

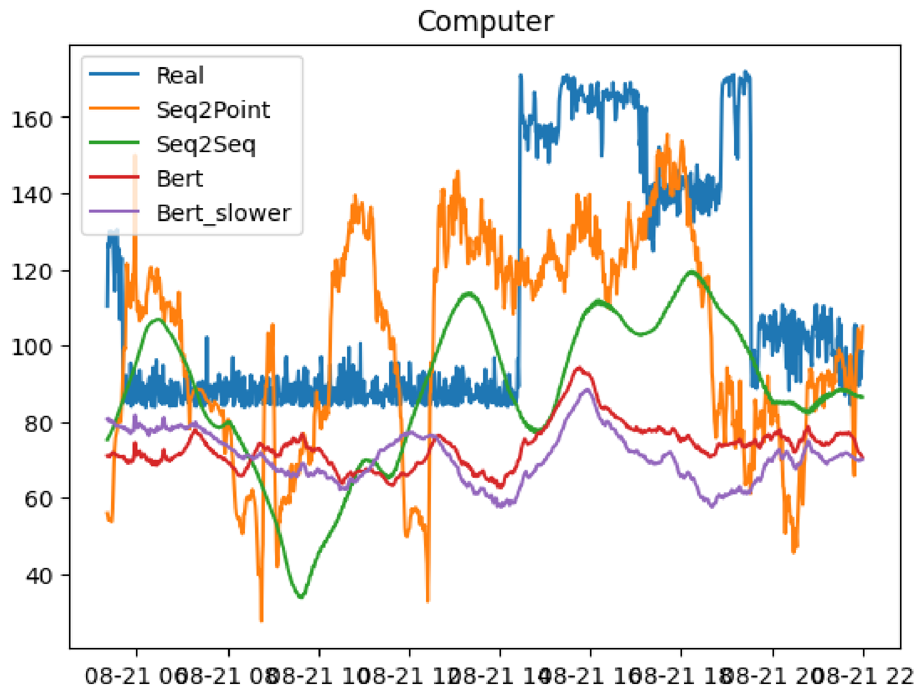
```
a = 29000
b = 30000
col = 0   # Tumble Dryer

plt.plot(results["test_submeters"][col][1][0][a:b], label = "Real")
plt.plot(results["pred_overall"]["Seq2Point"].iloc[a:b,col], label =
"Seq2Point")
plt.plot(results["pred_overall"]["Seq2Seq"].iloc[a:b,col], label = "Seq2Seq")
plt.plot(results["pred_overall"]["Bert"].iloc[a:b,col], label = "Bert")
```

```
plt.plot(results["pred_overall"]["Bert_slower"].iloc[a:b,col], label =
"Bert_slower")
plt.title("Tumble Dryer")
plt.legend()
plt.plot()
```



Again, we see the same pattern: BERT models capture the location of the spikes but not the complete magnitude. Moreover, it seems that the BERT models sometimes captured random noise.

```
a = 29000
b = 30000
col = 1   # Computer

plt.plot(results["test_submeters"][col][1][0][a:b], label = "Real")
plt.plot(results["pred_overall"]["Seq2Point"].iloc[a:b,col], label =
"Seq2Point")
plt.plot(results["pred_overall"]["Seq2Seq"].iloc[a:b,col], label = "Seq2Seq")
plt.plot(results["pred_overall"]["Bert"].iloc[a:b,col], label = "Bert")
plt.plot(results["pred_overall"]["Bert_slower"].iloc[a:b,col], label =
"Bert_slower")
plt.title("Computer")
plt.legend()
plt.plot()
```



For the computer the `BERT` models were much more consistent than the sequence models. The `Seq2Seq` and `Seq2Point` seemed to struggle with noise in the time series leading to random spikes, making them rather impractical for the computer.

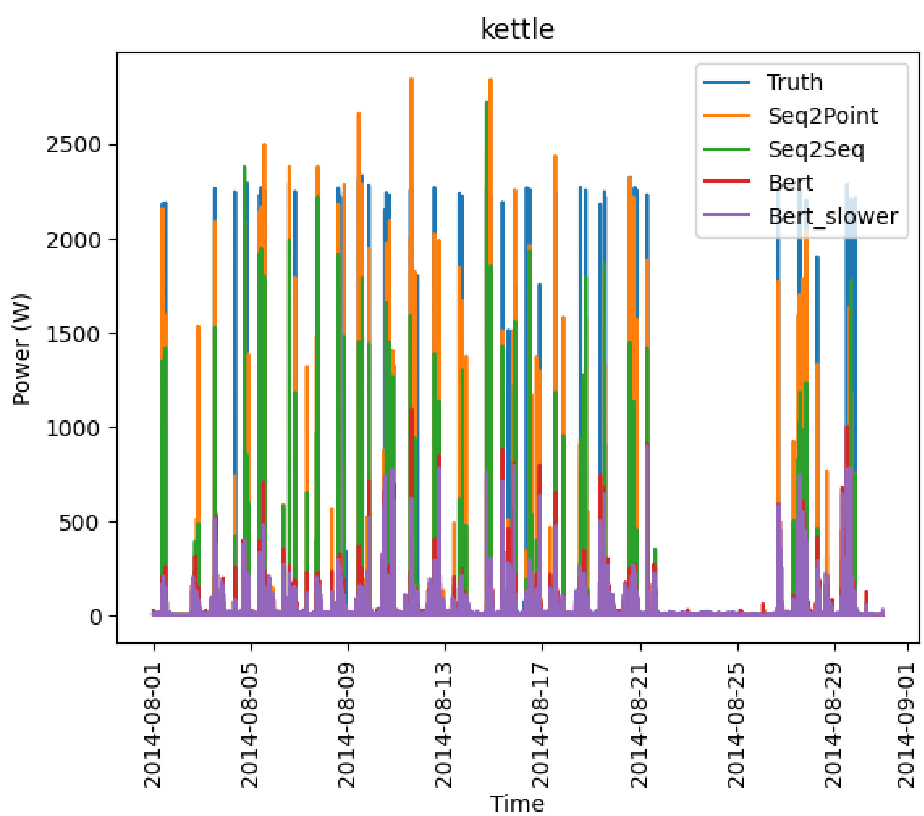
Building 7

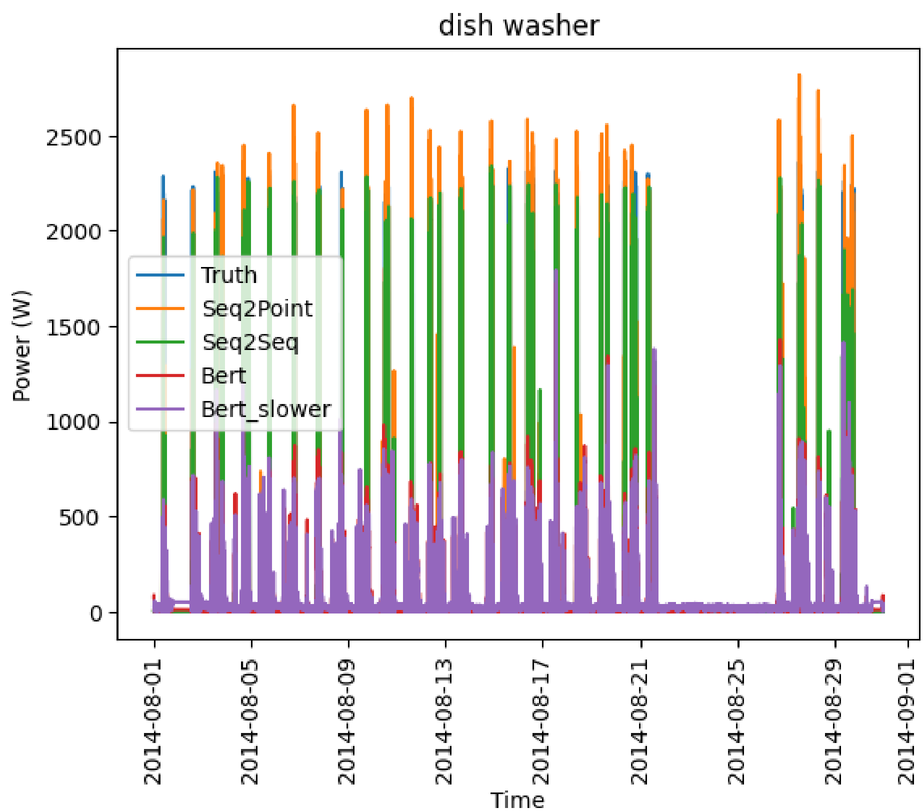
```
refit.buildings[7].elec

MeterGroup(meters=
  ElecMeter(instance=1, building=7, dataset='REFIT', site_meter,
appliances=[])
  ElecMeter(instance=2, building=7, dataset='REFIT',
appliances=[Appliance(type='fridge', instance=1)])
  ElecMeter(instance=3, building=7, dataset='REFIT',
appliances=[Appliance(type='freezer', instance=1)])
  ElecMeter(instance=4, building=7, dataset='REFIT',
appliances=[Appliance(type='freezer', instance=2)])
  ElecMeter(instance=5, building=7, dataset='REFIT',
appliances=[Appliance(type='tumble dryer', instance=1)])
  ElecMeter(instance=6, building=7, dataset='REFIT',
appliances=[Appliance(type='washing machine', instance=1)])
  ElecMeter(instance=7, building=7, dataset='REFIT',
appliances=[Appliance(type='dish washer', instance=1)])
  ElecMeter(instance=8, building=7, dataset='REFIT',
appliances=[Appliance(type='television', instance=1)])
  ElecMeter(instance=9, building=7, dataset='REFIT',
appliances=[Appliance(type='toaster', instance=1)])
  ElecMeter(instance=10, building=7, dataset='REFIT',
appliances=[Appliance(type='kettle', instance=1)])
)
```

For building 7 we tried to predict the kettle and the dish washer. Both appliances are used for a very short period of time only.

```
building7_param = {
  "power": {"mains": ["apparent","active"],"appliance":
["apparent","active"]},
  "sample_rate": 60,
  "appliances": ["kettle", "dish washer"],
  "methods": {"Seq2Point": Seq2Point({"n_epochs": 20}),
              "Seq2Seq": Seq2Seq({"n_epochs": 20}),
              "Bert": BERT({"n_epochs": 10}),
              "Bert_slower": BERT({"n_epochs": 10, "learning_rate": 0.0005})
              },
  "display_predictions": True,
  "train": {
    "datasets": {
        "Dataport": {
            "path": file_path,
            "buildings": {
                7: {
                    "start_time": "2014-04-01",
                    "end_time": "2014-07-31"
                    }
                }
            }
        }
    },
  "test": {
    "datasets": {
        "Dataport": {
            "path": file_path,
            "buildings": {
                7: {
                    "start_time": "2014-08-01",
                    "end_time": "2014-08-31"
                    }
                }
            }
        },
        "metrics":["rmse"]
    }
  }

if Path(f"{data_path}/building7.joblib").exists() == False:
    building7_mod = API(building7_param)
    results = {
        "pred_overall": building7_mod.pred_overall,
        "errors": building7_mod.errors,
        "test_mains": building7_mod.test_mains,
        "test_submeters": building7_mod.test_submeters
    }
```

```
    with open(f"{data_path}/building7.joblib", "wb") as f:
        joblib.dump(results, f)

Joint Testing for all algorithms
Loading data for  Dataport  dataset
Dropping missing values
Generating predictions for : Seq2Point
84/84 [==============================] - 0s 3ms/step
84/84 [==============================] - 0s 2ms/step
Generating predictions for : Seq2Seq
84/84 [==============================] - 0s 2ms/step
84/84 [==============================] - 0s 2ms/step
Generating predictions for : BERT
84/84 [==============================] - 3s 36ms/step
84/84 [==============================] - 3s 36ms/step
Generating predictions for : BERT
84/84 [==============================] - 3s 36ms/step
84/84 [==============================] - 3s 36ms/step
............  rmse  .............
              Seq2Point      Seq2Seq         Bert  Bert_slower
kettle        87.887507    86.512243   110.119227   112.316187
dish washer  133.175607   125.308619   259.270627   264.857701
```

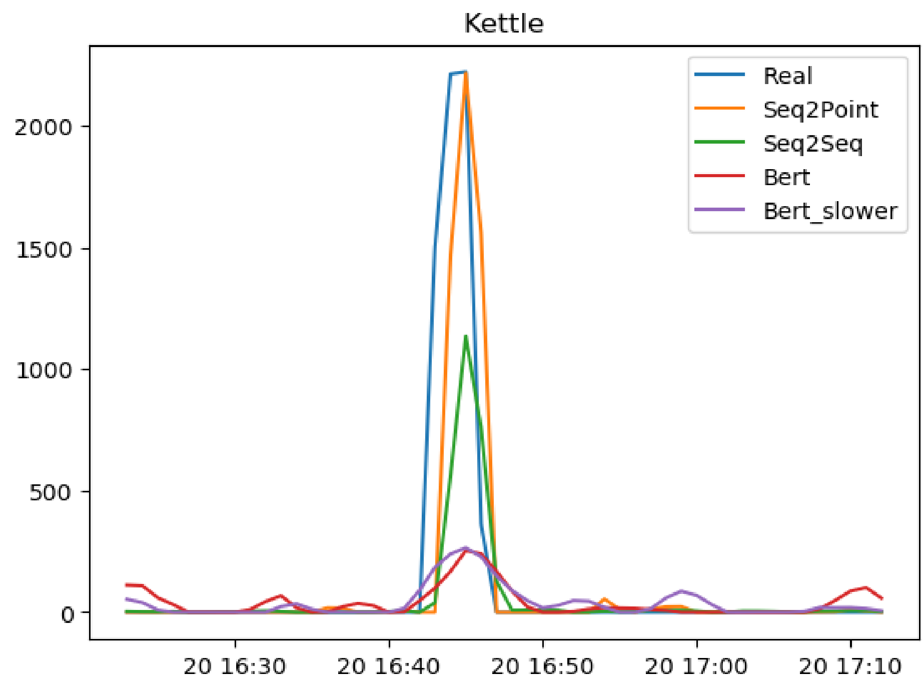




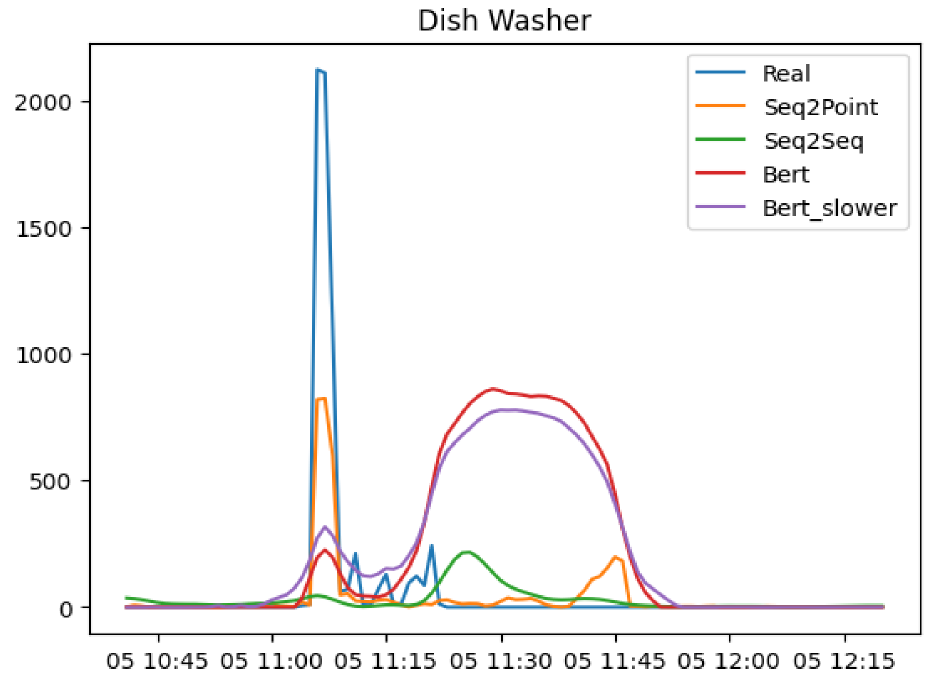

```
with open(f"{data_path}/building7.joblib", "rb") as f:
    results = joblib.load(f)
print(results["errors"])

[               Seq2Point      Seq2Seq         Bert   Bert_slower
kettle         87.887507    86.512243   110.119227   112.316187
dish washer   133.175607   125.308619   259.270627   264.857701]
```

Looking at the errors, the `Seq2Seq` and `Seq2Point` clearly outperformed the `BERT` transformer models. It's likely that the outperformance is due to the architecture of the models.

```
a = 28250
b = 28300
col = 0    # Kettle

plt.plot(results["test_submeters"][col][1][0][a:b], label = "Real")
plt.plot(results["pred_overall"]["Seq2Point"].iloc[a:b,col], label =
"Seq2Point")
plt.plot(results["pred_overall"]["Seq2Seq"].iloc[a:b,col], label = "Seq2Seq")
plt.plot(results["pred_overall"]["Bert"].iloc[a:b,col], label = "Bert")
plt.plot(results["pred_overall"]["Bert_slower"].iloc[a:b,col], label =
"Bert_slower")
plt.title("Kettle")
plt.legend()
plt.plot()
```




```
a = 6400
b = 6500
col = 1   # Dish Washer

plt.plot(results["test_submeters"][col][1][0][a:b], label = "Real")
plt.plot(results["pred_overall"]["Seq2Point"].iloc[a:b,col], label =
"Seq2Point")
plt.plot(results["pred_overall"]["Seq2Seq"].iloc[a:b,col], label = "Seq2Seq")
plt.plot(results["pred_overall"]["Bert"].iloc[a:b,col], label = "Bert")
plt.plot(results["pred_overall"]["Bert_slower"].iloc[a:b,col], label =
"Bert_slower")
plt.title("Dish Washer")
plt.legend()
plt.plot()
```



The transformer models predicted a second "bump" for the dish washer probably due to some random noise.

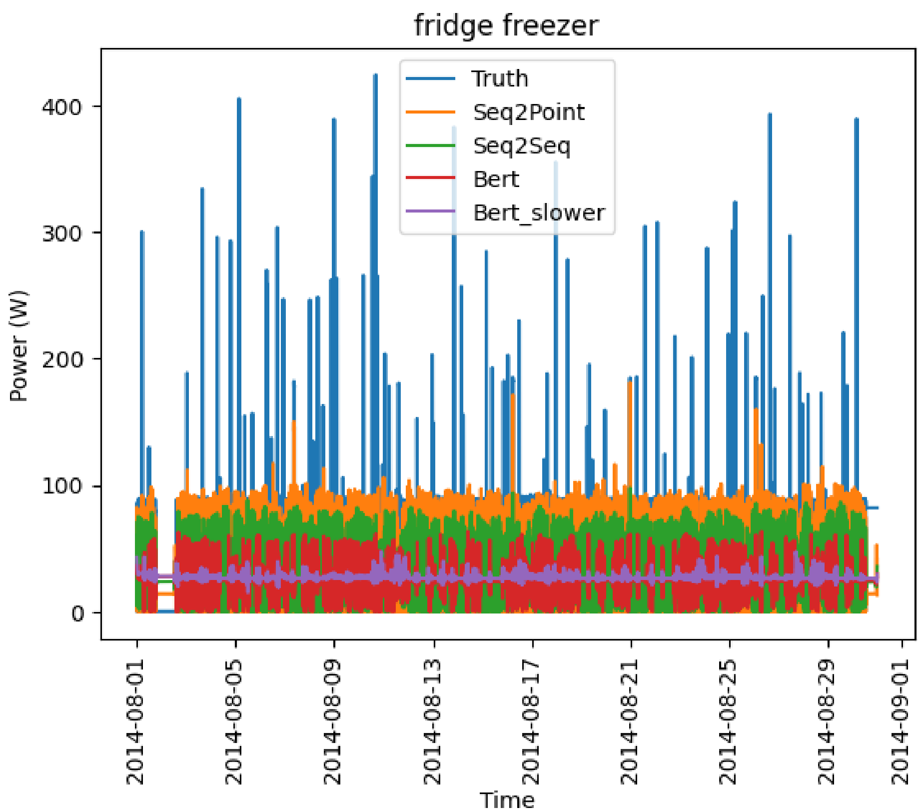
Building 14

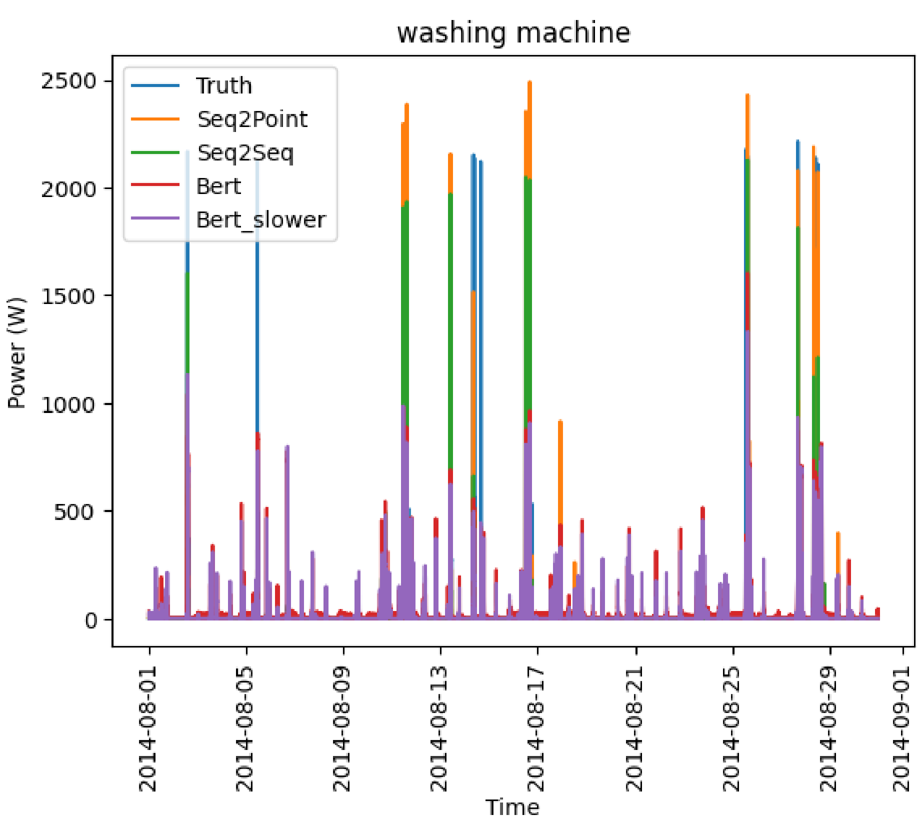
```
refit.buildings[14].elec

MeterGroup(meters=
  ElecMeter(instance=1, building=14, dataset='REFIT', site_meter,
appliances=[])
  ElecMeter(instance=2, building=14, dataset='REFIT',
appliances=[Appliance(type='fridge freezer', instance=1)])
  ElecMeter(instance=3, building=14, dataset='REFIT',
appliances=[Appliance(type='tumble dryer', instance=1)])
  ElecMeter(instance=4, building=14, dataset='REFIT',
appliances=[Appliance(type='washing machine', instance=1)])
  ElecMeter(instance=5, building=14, dataset='REFIT',
appliances=[Appliance(type='dish washer', instance=1)])
  ElecMeter(instance=6, building=14, dataset='REFIT',
appliances=[Appliance(type='computer', instance=1)])
  ElecMeter(instance=7, building=14, dataset='REFIT',
appliances=[Appliance(type='television', instance=1)])
  ElecMeter(instance=8, building=14, dataset='REFIT',
appliances=[Appliance(type='microwave', instance=1)])
  ElecMeter(instance=9, building=14, dataset='REFIT',
appliances=[Appliance(type='audio system', instance=1)])
  ElecMeter(instance=10, building=14, dataset='REFIT',
appliances=[Appliance(type='toaster', instance=1)])
)
```

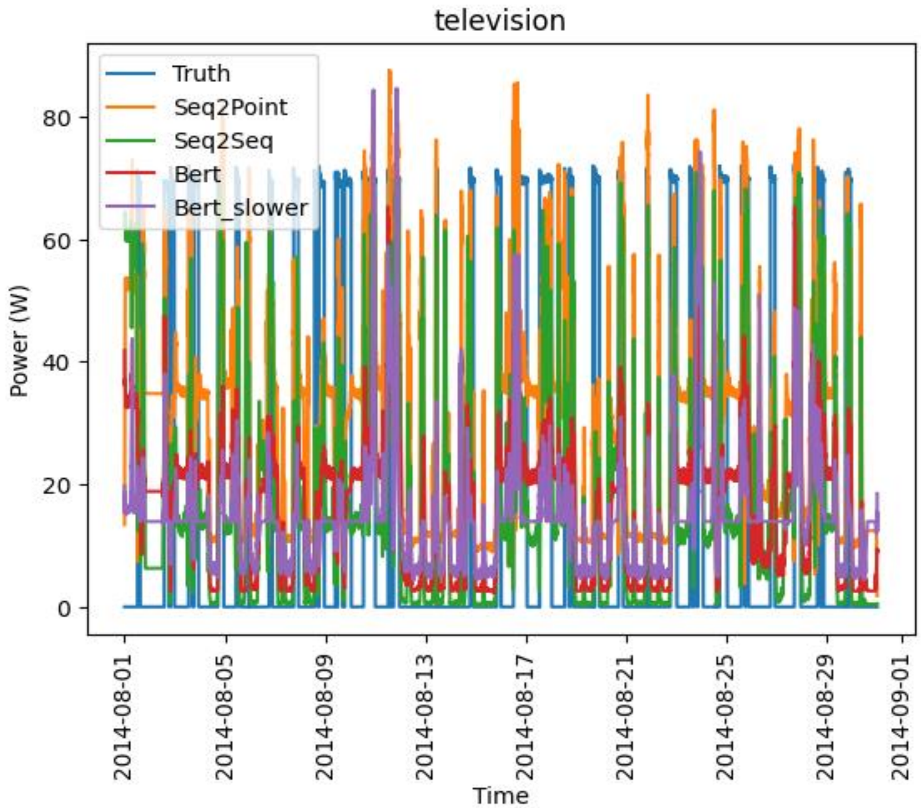
For building 14 we tried the models on a varity of appliance patterns.

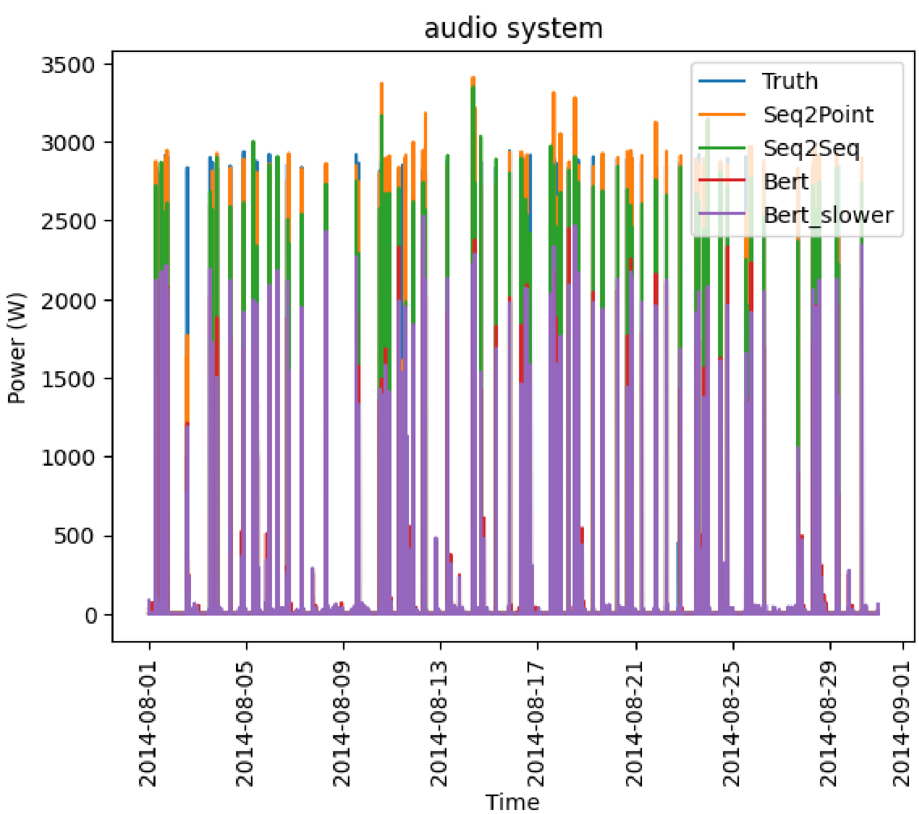
```python
building14_param = {
  "power": {"mains": ["apparent","active"],"appliance":
["apparent","active"]},
  "sample_rate": 60,
  "appliances": [ "fridge freezer", "washing machine", "television", "audio
system" ],
  "methods": {"Seq2Point": Seq2Point({"n_epochs": 20}),
              "Seq2Seq": Seq2Seq({"n_epochs": 20}),
              "Bert": BERT({"n_epochs": 10}),
              "Bert_slower": BERT({"n_epochs": 10, "learning_rate": 0.0005})
              },
  "display_predictions": True,
  "train": {
    "datasets": {
        "Dataport": {
            "path": file_path,
            "buildings": {
                14: {
                    "start_time": "2014-04-01",
                    "end_time": "2014-07-31"
                    }
                }
            }
        }
    },
  "test": {
    "datasets": {
        "Dataport": {
            "path": file_path,
            "buildings": {
                14: {
                    "start_time": "2014-08-01",
                    "end_time": "2014-08-31"
                    }
                }
            }
        },
        "metrics":["rmse"]
    }
  }

if Path(f"{data_path}/building14.joblib").exists() == False:
    building14_mod = API(building14_param)
    results = {
        "pred_overall": building14_mod.pred_overall,
        "errors": building14_mod.errors,
        "test_mains": building14_mod.test_mains,
        "test_submeters": building14_mod.test_submeters
    }
    with open(f"{data_path}/building14.joblib", "wb") as f:
        joblib.dump(results, f)
```

```
Joint Testing for all algorithms
Loading data for  Dataport  dataset
Dropping missing values
Generating predictions for : Seq2Point
85/85 [==============================] - 1s 6ms/step
85/85 [==============================] - 1s 5ms/step
85/85 [==============================] - 1s 4ms/step
85/85 [==============================] - 1s 4ms/step
Generating predictions for : Seq2Seq
85/85 [==============================] - 0s 4ms/step
85/85 [==============================] - 0s 3ms/step
85/85 [==============================] - 0s 2ms/step
85/85 [==============================] - 0s 2ms/step
Generating predictions for : BERT
85/85 [==============================] - 18s 215ms/step
85/85 [==============================] - 19s 218ms/step
85/85 [==============================] - 19s 215ms/step
85/85 [==============================] - 18s 214ms/step
Generating predictions for : BERT
85/85 [==============================] - 18s 214ms/step
85/85 [==============================] - 19s 215ms/step
85/85 [==============================] - 18s 215ms/step
85/85 [==============================] - 18s 214ms/step
............  rmse  ..............
                 Seq2Point    Seq2Seq        Bert  Bert_slower
fridge freezer   21.736150  25.764762   34.535043    38.285467
washing machine  52.042083  61.049155  104.372881   104.982779
television       29.410966  24.490124   26.568183    27.223311
audio system     47.394926  50.714894   94.528559    90.211560
```


fridge freezer
Truth
Seq2Point
Seq2Seq
Bert
Bert_slower
Power (W)
400
300
200
100
0
2014-08-01
2014-08-05
2014-08-09
2014-08-13
2014-08-17
2014-08-21
2014-08-25
2014-08-29
2014-09-01
Time


washing machine
Truth
Seq2Point
Seq2Seq
Bert
Bert_slower
Power (W)
2500
2000
1500
1000
500
0
2014-08-01
2014-08-05
2014-08-09
2014-08-13
2014-08-17
2014-08-21
2014-08-25
2014-08-29
2014-09-01
Time


television
Truth
Seq2Point
Seq2Seq
Bert
Bert_slower
Power (W)
0
20
40
60
80
2014-08-01
2014-08-05
2014-08-09
2014-08-13
2014-08-17
2014-08-21
2014-08-25
2014-08-29
2014-09-01
Time

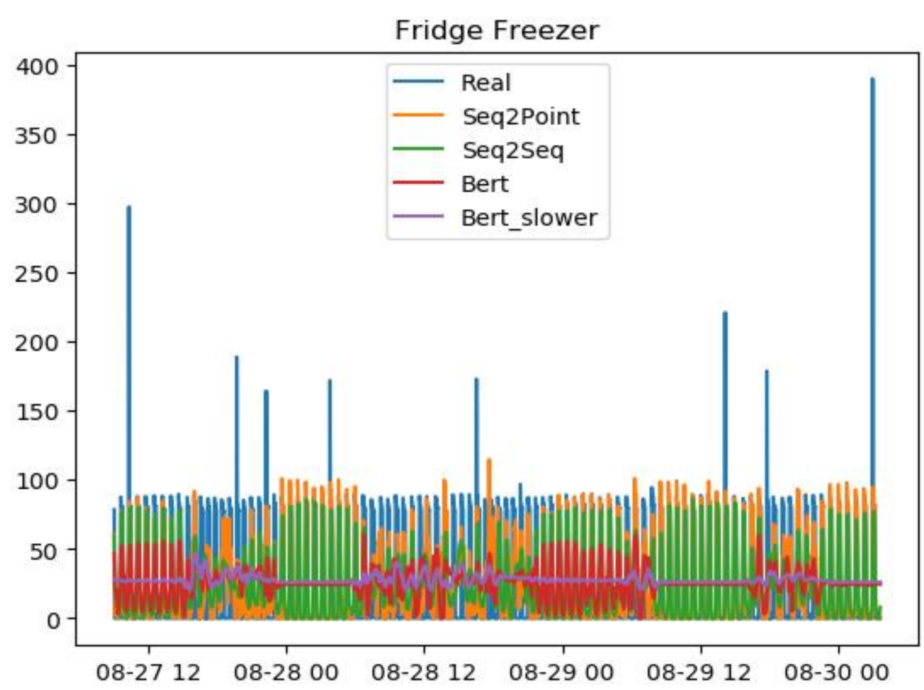

audio system
Truth
Seq2Point
Seq2Seq
Bert
Bert_slower
Power (W)
0
500
1000
1500
2000
2500
3000
3500
2014-08-01
2014-08-05
2014-08-09
2014-08-13
2014-08-17
2014-08-21
2014-08-25
2014-08-29
2014-09-01
Time

```
with open(f"{data_path}/building14.joblib", "rb") as f:
    results = joblib.load(f)
print(results["errors"])
```

```
[                  Seq2Point    Seq2Seq        Bert  Bert_slower
fridge freezer   21.736150  25.764762   34.535043    38.285467
washing machine  52.042083  61.049155  104.372881   104.982779
television       29.410966  24.490124   26.568183    27.223311
audio system     47.394926  50.714894   94.528559    90.211560]
```

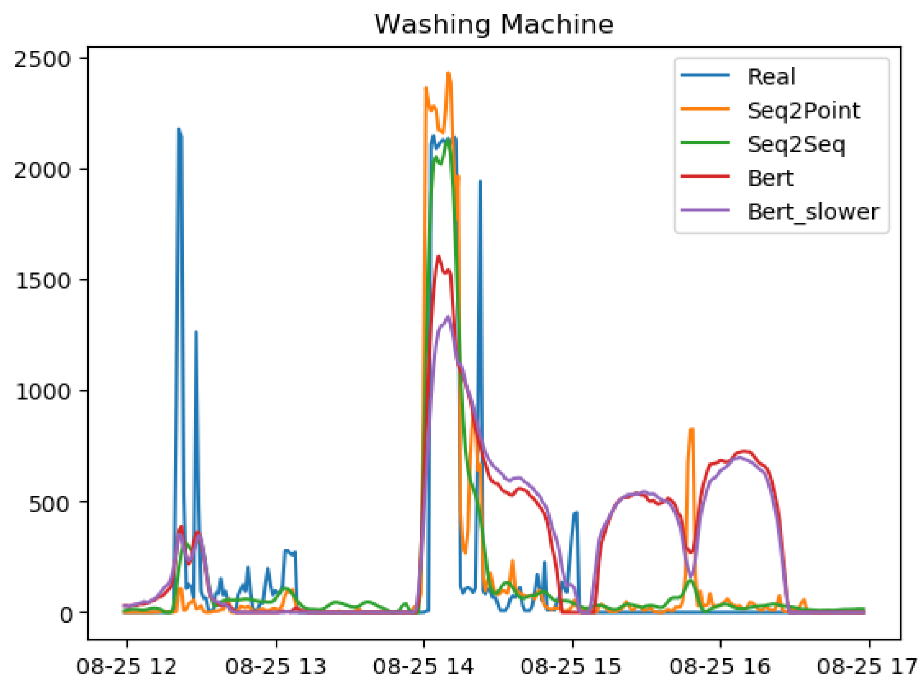
The sequence models again outperformed the transformer models on the RMSE metric. The transformer models seemed to be able to predict the time of the spikes but not their entire magnitude.

```
a = 38000
b = 42000
col = 0   # Fridge Freezer

plt.plot(results["test_submeters"][col][1][0][a:b], label = "Real")
plt.plot(results["pred_overall"]["Seq2Point"].iloc[a:b,col], label =
"Seq2Point")
plt.plot(results["pred_overall"]["Seq2Seq"].iloc[a:b,col], label = "Seq2Seq")
plt.plot(results["pred_overall"]["Bert"].iloc[a:b,col], label = "Bert")
plt.plot(results["pred_overall"]["Bert_slower"].iloc[a:b,col], label =
"Bert_slower")
plt.title("Fridge Freezer")
plt.legend()
plt.plot()
```



The fridge has an oscillating pattern, which works well for the sequence models but not for the transformer models. Sometimes the BERT predictions were oscillating and sometimes the prediction was only a flat line. Neither architecture recognized the high magnitude spikes. A slower learning rate worsened the performance.

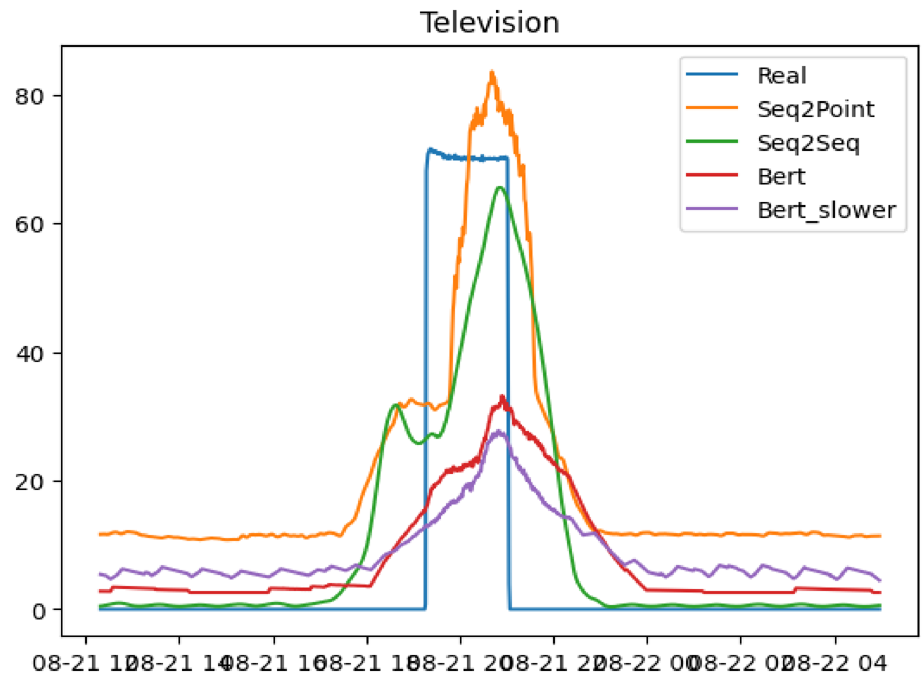
```
a = 35300
b = 35600
col = 1   # Washing Machine

plt.plot(results["test_submeters"][col][1][0][a:b], label = "Real")
plt.plot(results["pred_overall"]["Seq2Point"].iloc[a:b,col], label =
"Seq2Point")
plt.plot(results["pred_overall"]["Seq2Seq"].iloc[a:b,col], label = "Seq2Seq")
plt.plot(results["pred_overall"]["Bert"].iloc[a:b,col], label = "Bert")
plt.plot(results["pred_overall"]["Bert_slower"].iloc[a:b,col], label =
"Bert_slower")
plt.title("Washing Machine")
plt.legend()
plt.plot()
```



The washing mashine pattern is a typical example for the transformer models predicting a second "bump" after a real high. Maybe due to the self-attention mechanism?

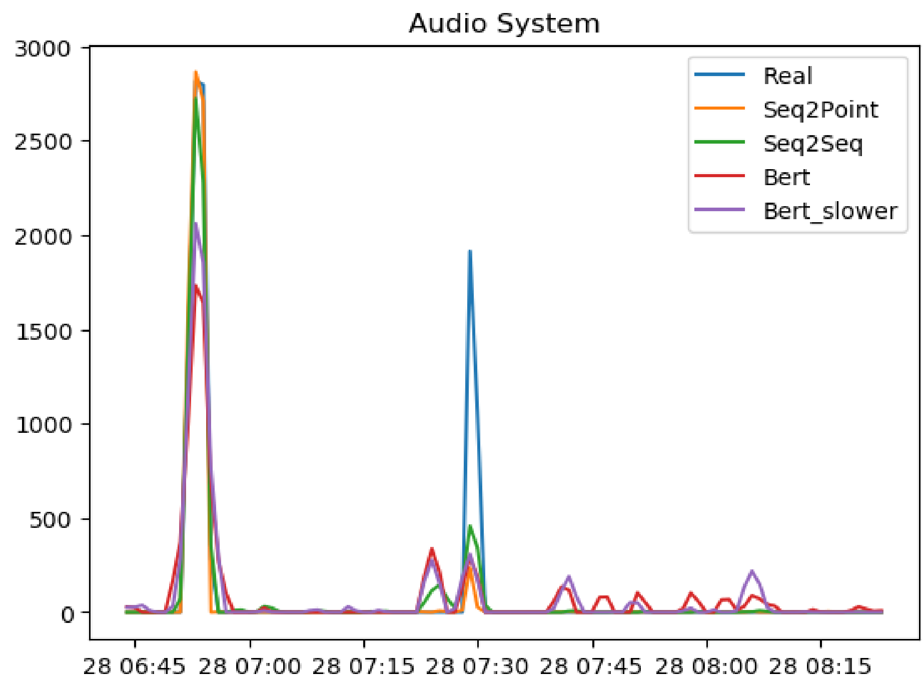
```
a = 29500
b = 30500
col = 2   # Television

plt.plot(results["test_submeters"][col][1][0][a:b], label = "Real")
plt.plot(results["pred_overall"]["Seq2Point"].iloc[a:b,col], label =
"Seq2Point")
plt.plot(results["pred_overall"]["Seq2Seq"].iloc[a:b,col], label = "Seq2Seq")
plt.plot(results["pred_overall"]["Bert"].iloc[a:b,col], label = "Bert")
```

```
plt.plot(results["pred_overall"]["Bert_slower"].iloc[a:b,col], label =
"Bert_slower")
plt.title("Television")
plt.legend()
plt.plot()
```



The sequence models responded faster to a high, while transformers were slower and responded with a lower magnitude.

```
a = 39300
b = 39400
col = 3   # Audio System

plt.plot(results["test_submeters"][col][1][0][a:b], label = "Real")
plt.plot(results["pred_overall"]["Seq2Point"].iloc[a:b,col], label =
"Seq2Point")
plt.plot(results["pred_overall"]["Seq2Seq"].iloc[a:b,col], label = "Seq2Seq")
plt.plot(results["pred_overall"]["Bert"].iloc[a:b,col], label = "Bert")
plt.plot(results["pred_overall"]["Bert_slower"].iloc[a:b,col], label =
"Bert_slower")
plt.title("Audio System")
plt.legend()
plt.plot()
```



The audio system had a very short usage-time, and all models recognized the spikes. The BERT models fit some random noise and as a result, predicted phantom spikes.

Conclusion

- It's not clear whether the transformer architecture is superior to the sequence models. Quite often the transformers struggle with oscillating patterns or very high magnitudes. Also, their training is computationally much more expensive than sequence models.
- More epochs for the BERT models would have been better, but we were constrained by Google Colab computing units.
- There is room for a lot more experiments, for example by changing learning rates or the pre-processing functions.