In [1]:

```
# from nilm_metadata import get_appliance_types
# appliance_types = get_appliance_types()
# print(appliance_types)

# import os
# os.getcwd()
```

Carregando bibliotecas...

In [1]:

```
!pip install seaborn

import seaborn as sns

from matplotlib import rcParams
import matplotlib.pyplot as plt
import pandas as pd
import nilmtk
from nilmtk import MeterGroup
from nilmtk.api import API
import warnings
warnings.filterwarnings("ignore")

plt.style.use('ggplot')
rcParams['figure.figsize'] = (13, 10)

# import pathlib
# pathlib.Path().resolve()
```

```
Requirement already satisfied: seaborn in ./miniconda3/envs/nilm_0.4.
3/lib/python3.7/site-packages (0.11.2)
Requirement already satisfied: numpy>=1.15 in ./miniconda3/envs/nilm_
0.4.3/lib/python3.7/site-packages (from seaborn) (1.19.5)
Requirement already satisfied: scipy>=1.0 in ./miniconda3/envs/nilm_0.
4.3/lib/python3.7/site-packages (from seaborn) (1.7.1)
Requirement already satisfied: pandas>=0.23 in ./miniconda3/envs/nilm_
0.4.3/lib/python3.7/site-packages (from seaborn) (0.25.3)
Requirement already satisfied: matplotlib>=2.2 in ./miniconda3/envs/ni
lm_0.4.3/lib/python3.7/site-packages (from seaborn) (3.1.3)
Requirement already satisfied: kiwisolver>=1.0.1 in ./miniconda3/envs/
nilm_0.4.3/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn)
(1.3.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.
1 in ./miniconda3/envs/nilm_0.4.3/lib/python3.7/site-packages (from ma
tplotlib>=2.2->seaborn) (2.4.7)
Requirement already satisfied: python-dateutil>=2.1 in ./miniconda3/en
vs/nilm_0.4.3/lib/python3.7/site-packages (from matplotlib>=2.2->seabo
rn) (2.8.2)
Requirement already satisfied: cycler>=0.10 in ./miniconda3/envs/nilm_
0.4.3/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn) (0.1
0.0)
Requirement already satisfied: six in ./miniconda3/envs/nilm_0.4.3/li
b/python3.7/site-packages (from cycler>=0.10->matplotlib>=2.2->seabor
n) (1.16.0)
Requirement already satisfied: pytz>=2017.2 in ./miniconda3/envs/nilm_
0.4.3/lib/python3.7/site-packages (from pandas>=0.23->seaborn) (2021.
1)
```

Converter

In [ ]:

```
# from nilmtk.dataset_converters import convert_hb
# convert_hb('./BD/CASA/convert', './data/teste17.h5')
```

In [ ]:

```
# st = pd.HDFStore("./data/teste17.h5")
# print (st.keys())

# print (st['/building1/elec/meter1'].head())
# print (st['/building1/elec/meter2'].head())
# print (st['/building1/elec/meter3'].head())

# st.close()
```

Carregando dataset

In [2]:

```
from nilmtk.api import API
import warnings
warnings.filterwarnings("ignore")

from nilmtk import DataSet
from nilmtk.utils import print_dict

hb = DataSet('teste10.h5')
#iawe = DataSet('/data/iawe.h5')

print_dict(hb.metadata)
print_dict(hb.buildings)
```

- **name**: REDD
- **long_name**: The Reference Energy Disaggregation Data set
- **creators**:
  - Kolter, Zico
  - Johnson, Matthew
- **publication_date**: 2011
- **institution**: Massachusetts Institute of Technology (MIT)
- **contact**: zkolter@cs.cmu.edu
- **description**: Several weeks of power data for 6 different homes.
- **subject**: Disaggregated power demand from domestic buildings.
- **number_of_buildings**: 1
- **timezone**: US/Eastern
- **geo_location**:
  - **locality**: Massachusetts
  - **country**: US
  - **latitude**: 42.360091
  - **longitude**: -71.09416
- **related_documents**:
  - http://redd.csail.mit.edu (http://redd.csail.mit.edu)
  - J. Zico Kolter and Matthew J. Johnson. REDD: A public data set for energy disaggregation research. In proceedings of the SustKDD workshop on Data Mining Applications in Sustainability, 2011. http://redd.csail.mit.edu/kolter-kddsust11.pdf (http://redd.csail.mit.edu/kolter-kddsust11.pdf)
- **schema**: https://github.com/nilmtk/nilm_metadata/tree/v0.2 (https://github.com/nilmtk/nilm_metadata/tree/v0.2)
- **meter_devices**:
  - **eMonitor**:
    - **model**: eMonitor
    - **manufacturer**: Powerhouse Dynamics
    - **manufacturer_url**: http://powerhousedynamics.com (http://powerhousedynamics.com)
    - **description**: Measures circuit-level power demand. Comes with 24 CTs. This FAQ page suggests the eMonitor measures real (active) power: http://www.energycircle.com/node/14103 (http://www.energycircle.com/node/14103) although the REDD readme.txt says all channels record apparent power.
    - **sample_period**: 5
    - **max_sample_period**: 30
    - **measurements**:

- {'physical_quantity': 'power', 'type': 'active', 'upper_limit': 1142, 'lower_limit': 0}
- {'physical_quantity': 'power', 'type': 'apparent', 'upper_limit': 1215, 'lower_limit': 0}
- {'physical_quantity': 'power', 'type': 'reactive', 'upper_limit': 901, 'lower_limit': 0}
- {'physical_quantity': 'power factor', 'upper_limit': 1, 'lower_limit': 0}
- {'physical_quantity': 'voltage', 'upper_limit': 232, 'lower_limit': 0}
- {'physical_quantity': 'current', 'upper_limit': 6, 'lower_limit': 0}

- **wireless**: False

- **REDD_whole_house**:
  - **description**: REDD's DIY power meter used to measure whole-home AC waveforms at high frequency. To quote from their paper: "CTs from TED (http://www.theenergydetective.com (http://www.theenergydetective.com)) to measure current in the power mains, a Pico TA041 oscilloscope probe (http://www.picotechnologies.com (http://www.picotechnologies.com)) to measure voltage for one of the two phases in the home, and a National Instruments NI-9239 analog to digital converter to transform both these analog signals to digital readings. This A/D converter has 24 bit resolution with noise of approximately 70 µV, which determines the noise level of our current and voltage readings: the TED CTs are rated for 200 amp circuits and a maximum of 3 volts, so we are able to differentiate between currents of approximately ((200))(70 × 10−6)/(3) = 4.66mA, corresponding to power changes of about 0.5 watts. Similarly, since we use a 1:100 voltage stepdown in the oscilloscope probe, we can detect voltage differences of about 7mV."
  - **sample_period**: 0.5
  - **max_sample_period**: 30
  - **measurements**:
    - {'physical_quantity': 'voltage', 'upper_limit': 230, 'lower_limit': 0}
    - {'physical_quantity': 'current', 'upper_limit': 15, 'lower_limit': 0}
    - {'physical_quantity': 'power', 'type': 'active', 'upper_limit': 3016, 'lower_limit': 0}
    - {'physical_quantity': 'frequency', 'upper_limit': 61, 'lower_limit': 0}
    - {'physical_quantity': 'power factor', 'upper_limit': 1, 'lower_limit': 0}
  - **wireless**: False

- **1**: Building(instance=1, dataset='REDD')

Gráfico Geral

In [ ]:

```
build = 1
elec = hb.buildings[build].elec
elec.mains().power_series_all_data().head()
```

In [ ]:

```
sns.set_palette("Set2", n_colors=5)
elec.mains().plot()
elec['microwave'].plot()
elec['fan'].plot()

# Set a threshold to remove residual power noise when devices are off
elec.plot_when_on(on_power_threshold = 40) # Plot appliances when they are in use¶

# elec.draw_wiring_graph()
```

Dados

Proporção de energia submedida

In [ ]:

```
elec.proportion_of_energy_submetered()
```

Total Energy

In [ ]:

```
elec.mains().total_energy()
```

Energy per submeter

In [ ]:

```
energy_per_meter = elec.submeters().energy_per_meter() # kWh, again
energy_per_meter
```

Plot fraction of energy consumption of each appliance

In [ ]:

```
# fraction = elec.submeters().fraction_per_meter().dropna()
fraction = elec.fraction_per_meter().dropna()
# Create convenient labels
labels = elec.get_labels(fraction.index)
plt.figure(figsize=(10,30))
fraction.plot(kind='pie', labels=labels);
```

Quadro Geral

In [ ]:

```
print(elec)
elec.mains()
```

In [ ]:

```
from nilmtk.elecmeter import ElecMeterID##### Quadro Geral

meter1 = elec[ElecMeterID(instance=1, building=build, dataset='HB')]

next(meter1.load()).head()
```

In [ ]:

```
meter1.plot()
```

A taxa de abandono é um número entre 0 e 1 que especifica a proporção de amostras ausentes. Uma taxa de abandono de 0 significa que nenhuma amostra está faltando. Um valor de 1 significaria que todas as amostras estão faltando

In [ ]:

```
meter1.dropout_rate()
```

In [ ]:

```
good_sections = meter1.good_sections(full_results=True)
good_sections.plot()
```

In [ ]:

```
good_sections.combined()
```

Microondas

In [ ]:

```
microwave= elec['microwave']
#microwave.available_columns()
next(microwave.load()).head()
```

In [ ]:

```
microwave.plot()
```

A taxa de abandono é um número entre 0 e 1 que especifica a proporção de amostras ausentes. Uma taxa de abandono de 0 significa que nenhuma amostra está faltando. Um valor de 1 significaria que todas as amostras estão faltando

In [ ]:

```
microwave.dropout_rate()
```

In [ ]:

```
good_sections = microwave.good_sections(full_results=True)
good_sections.plot()
```

In [ ]:

```
good_sections.combined()
```

Ventilador

In [ ]:

```
fan = elec['fan']
#microwave.available_columns()
next(fan.load()).head()
```

In [ ]:

```
fan.plot()
```

In [ ]:

```
good_sections = fan.good_sections(full_results=True)
good_sections.plot()
```

A taxa de abandono é um número entre 0 e 1 que especifica a proporção de amostras ausentes. Uma taxa de abandono de 0 significa que nenhuma amostra está faltando. Um valor de 1 significaria que todas as amostras estão faltando

In [ ]:

```
fan.dropout_rate()
```

In [ ]:

```
good_sections.combined()
```

Autocorrelation Plot

In [ ]:

```
# from pandas.plotting import autocorrelation_plot
# elec.mains().plot_autocorrelation();
```

Dataframe de correlação dos aparelhos

In [ ]:

```
# correlation_df = elec.pairwise_correlation()
# correlation_df
```

Traçar dados submedidos em um 1 dia

In [ ]:

```
hb.set_window(start='2021-09-05', end='2021-09-07')
elec.plot();
plt.xlabel("Time");
```

In [ ]:

```
# hb.set_window(start='2021-09-05 00:00:00', end='2021-09-06 23:59:59')
hb.set_window(start='2021-09-05', end='2021-09-07')

# elec['microwave'].plot()
elec['fan'].plot()
plt.xlabel("Time");
```

Importamos os algoritmos que desejamos executar os experimentos:

Mean: Mean Algorithm

Hart's Algorithm

CO: Combinatorial Optimization

Discriminative Sparse Coding

Additive Factorial Hidden Markov Model

Additive Factorial Hidden Markov Model with Signal Aggregate Constraints

DSC: Discriminative Sparse Coding

RNN: Long short-term memory - LSTM

DAE: Denoising Auto Encoder

Seq2Point*

Seq2Seq

WindowGRU/Online GRU: Similar a LSTM, mas usa Gated Recurrent Unit (GRU)

ELM

In [3]:

```
from nilmtk.disaggregate import Mean,CO,Hart85
# from nilmtk_contrib.disaggregate import AFHMM,AFHMM_SAC,DSC,RNN,Seq2Point,Seq2Seq
from nilmtk_contrib.disaggregate import RNN,Seq2Point,WindowGRU
```

```
Using TensorFlow backend.
```

Em seguida, inserimos os valores para os diferentes parâmetros no dicionário. Como precisamos de vários aparelhos, inserimos os nomes de todos os aparelhos necessários no parâmetro 'appliances'.

Métricas: [https://github.com/nilmtk/nilmtk/blob/master/nilmtk/losses.py (https://github.com/nilmtk/nilmtk/blob/master/nilmtk/losses.py)](https://github.com/nilmtk/nilmtk/blob/master/nilmtk/losses.py)

Error: [https://github.com/nilmtk/nilmtk-contrib/issues/56 (https://github.com/nilmtk/nilmtk-contrib/issues/56)](https://github.com/nilmtk/nilmtk-contrib/issues/56)

In [4]:

```
d = {
  'power': {
    'mains': ['active'],
    'appliance': ['active']
#     'mains': ['active', 'frequency', 'power factor', 'current', 'voltage'],
#     'appliance': ['active', 'apparent', 'reactive', 'power factor', 'current', 'v
  },
  'sample_rate': 5,
  'display_predictions': True,
  'appliances': ['microwave', 'fan'],
  'methods': {
      'Mean':Mean({}),
#       "CO":CO({}),
      #'Hart85':Hart85({}),
      'RNN':RNN({'n_epochs':50,'batch_size':1024}),
      'Seq2Point':Seq2Point({'n_epochs':50,'batch_size':1024})
      #'Seq2Seq':Seq2Seq({'n_epochs':50,'batch_size':1024}),
      #'WindowGRU':WindowGRU({'n_epochs':30,'batch_size':1024})
  },
 'train': {
    'datasets': {
      'Redd': {
        'path': 'teste10.h5',
        'buildings': {
              1: {
                'start_time': '2021-09-02',
                'end_time': '2021-09-04'
              },

        }
      },

    }
  },

  'test': {
    'datasets': {
      'Redd': {
        'path': 'teste10.h5',
        'buildings': {
              1: {
                    'start_time': '2021-09-05',
                    'end_time': '2021-09-07'
          }
        }
      }
    },
    'metrics':['rmse', 'mae', 'relative_error', 'r2score', 'nde', 'nep', 'f1score']
  }
}
```

raiz do erro quadrático médio (RMSE) e o erro médio absoluto (MAE)

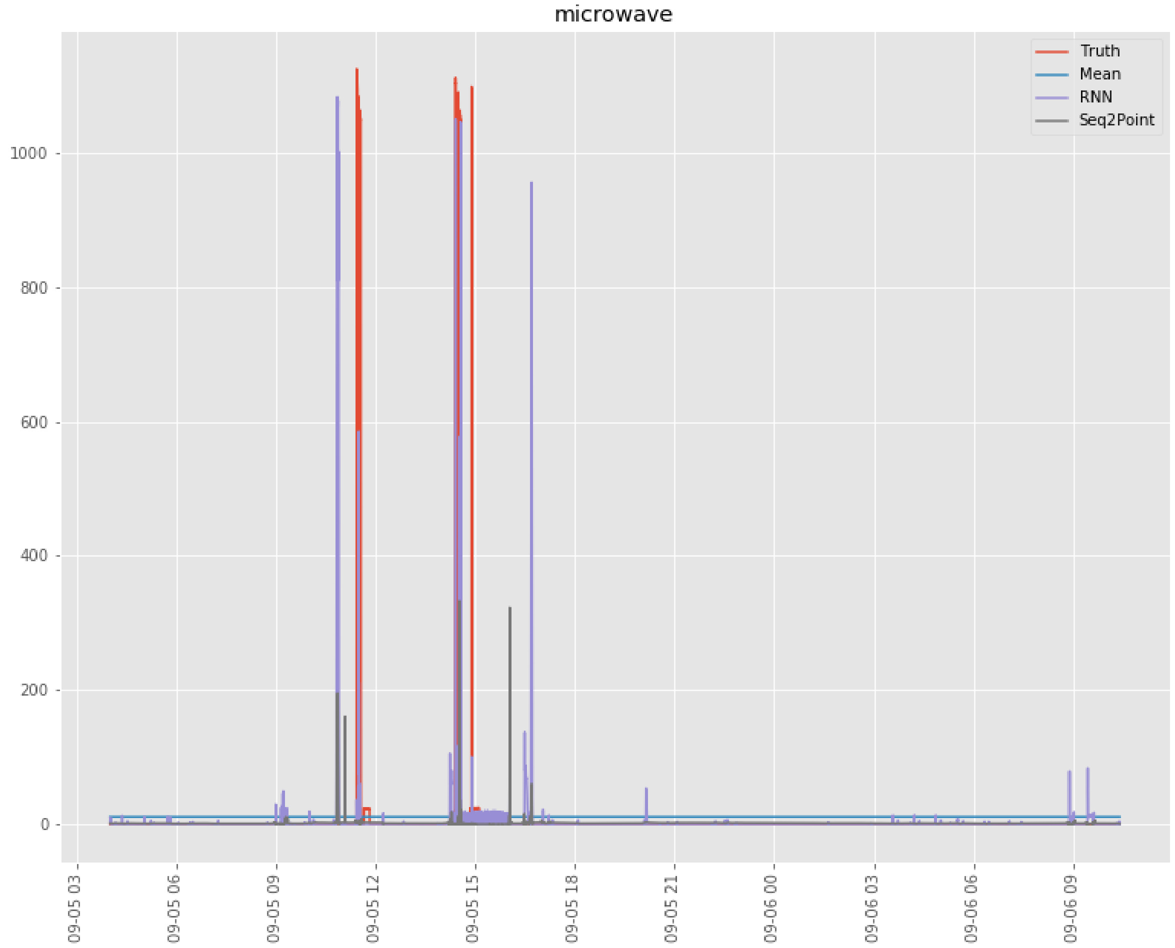
Quanto menor o seu valor, melhor é o modelo, já que a previsão se mostra mais próxima ao valor real. Comparando as duas métricas têm se que o RMSE penaliza desvios grandes, enquanto o MAE tem pesos iguais para todos os desvios.

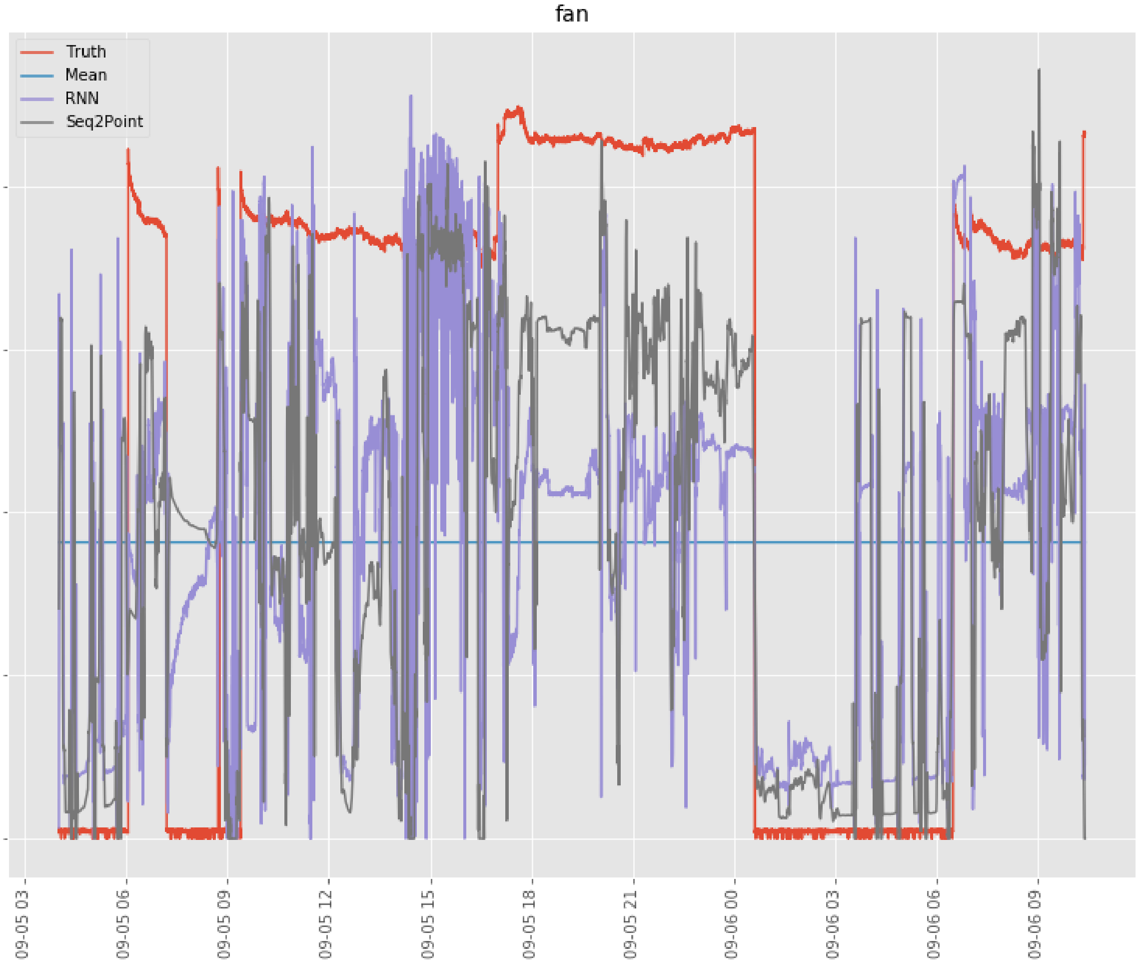
We can observe the prediction vs. truth graphs in the above cell. The accuracy metrics can be accessed using the following commands:

In [5]:

```
api_res = API(d)
```

```
Joint Testing for all algorithms
Loading data for  Redd  dataset
Dropping missing values
Generating predictions for : Mean
Generating predictions for : RNN
Generating predictions for : Seq2Point
............  rmse  ..............
                Mean        RNN  Seq2Point
microwave  72.196198  82.644827  72.262998
fan        40.406717  33.136142  32.572585
............  mae  ..............
                Mean        RNN  Seq2Point
microwave  15.895829   8.390965   7.145815
fan        39.986385  27.512135  26.085752
............  relative_error  ..............
               Mean       RNN  Seq2Point
microwave  1.314395  2.246470   2.809504
fan        1.071395  1.153426   1.722389
............  r2score  ..............
               Mean       RNN  Seq2Point
microwave -0.006476 -0.318882  -0.008339
fan       -0.203323  0.190757   0.218049
............  nde  ..............
               Mean       RNN  Seq2Point
microwave  1.000520  1.145321   1.001446
fan        0.626589  0.513844   0.505105
............  nep  ..............
               Mean       RNN  Seq2Point
microwave  2.997714  1.582410   1.347593
fan        0.755440  0.519771   0.492823
............  f1score  ..............
               Mean       RNN  Seq2Point
microwave  0.042654  0.201035   0.092784
fan        0.802654  0.872257   0.879705
```


microwave
Truth
Mean
RNN
Seq2Point
1000
800
600
400
200
0
09-05 03
09-05 06
09-05 09
09-05 12
09-05 15
09-05 18
09-05 21
09-06 00
09-06 03
09-06 06
09-06 09


fan
Truth
Mean
RNN
Seq2Point
0
20
40
60
80
09-05 03
09-05 06
09-05 09
09-05 12
09-05 15
09-05 18
09-05 21
09-06 00
09-06 03
09-06 06
09-06 09

In [6]:

```
import numpy as np
import pandas as pd

vals = np.concatenate([np.expand_dims(df.values,axis=2) for df in api_res.errors],a


cols = api_res.errors[0].columns
indexes = api_res.errors[0].index


mean = np.mean(vals,axis=2)
std = np.std(vals,axis=2)
print ('\n\n')
print ("Mean")
print (pd.DataFrame(mean,index=indexes,columns=cols))
print ('\n\n')
print ("Standard Deviation")
print (pd.DataFrame(std,index=indexes,columns=cols))
```

```



Mean
                Mean        RNN  Seq2Point
microwave  13.348691  13.698878  12.093114
fan        11.920837   9.128333   8.925201



Standard Deviation
                Mean        RNN  Seq2Point
microwave  24.580400  28.274081  24.669300
fan        17.887211  13.492315  13.028054
```

In [ ]: