ExampleB

April 1, 2021

1 Project enda : Example B

If you haven't already, read Example A first, it is not long. Download `example_b.zip` and run this notebook in the correct python environment.

Install all the required packages in your python virtualenv:

```
pip install numexpr bottleneck pandas enda jupyter h2o scikit-learn statsmodels matplotlib jobl
# more packages used for feature engineering below
pip install jours-feries-france vacances-scolaires-france Unidecode
```

In this example we will go more in depth, with realistic data and more historical data (~4-5 years). This example is divided in 7 parts: 1. Read and prepare data, check for missing values and gaps 2. Visualize data 3. Feature engineering : datetime and calendar features 4. Portfolio forecast & basic prediction 5. Benchmark with simple evaluation 6. Benchmark with Backtesting 7. Make the prediction

We set ourselves in a setup as if we were **exactly on 2020-11-30**. We want to predict the total consumption of customers for the next few days starting 2020-12-01 at a 30min time-step. We have: - our customer contracts until 2020-11-30 included. - historical load data from 2015-01-01 until 2020-11-15 included. There is a ~15 day time-gap between the last moment for which we have an actual load measure and 'today' (2020-11-30). - weather forecast until 2020-12-11 (11 days). - our TSO's network load forecast until 2020-12-7 (7 days).

In here (example B), we will put all our customers in only 1 group and forecast the next 7 days. We will first construct the dataset and the forecast input data and test it with a basic linear regressor. We will then try various algorithms and compare them. Finally we will give an example of backtesting on the data.

[1]:
```
import enda
import pandas as pd
import os
```

1.1 1. Read and prepare data, check for missing values and gaps

[2]:
```
# Replace this with the path to your example_b directory.
# You should have ExampleB.ipynb opened in jupiter, so you can run each step
DIR = '/Users/emmanuel.charon/Documents/CodeProjects/enercoop/enda/data/
 ↪example_b'
```

[3]:
```
# Get the 30min time-step data just like in Example A
# (columns are a bit different and there is more data)
# Here we consider all customers in one big group.
def read_data():
    contracts = enda.Contracts.read_contracts_from_file(os.path.join(DIR,␣
↪"contracts.csv"))
    contracts["contracts_count"] = 1
    portfolio_by_day = enda.Contracts.compute_portfolio_by_day(
        contracts,
        columns_to_sum = ["contracts_count", "kva"],
        date_start_col="date_start",
        date_end_exclusive_col="date_end_exclusive",
    )
    portfolio = enda.TimeSeries.interpolate_daily_to_sub_daily_data(
        portfolio_by_day,
        freq='30min',
        tz='Europe/Paris'
    )

    historic_load_measured = pd.read_csv(os.path.join(DIR,␣
↪"historic_load_measured.csv"))
    weather_and_tso_forecasts = pd.read_csv(os.path.join(DIR,␣
↪"weather_and_tso_forecasts.csv"))
    # correctly format 'time' as a pandas.DatetimeIndex of dtype: datetime[ns,␣
↪tzinfo]
    for df in [historic_load_measured, weather_and_tso_forecasts]:
        df['time'] = pd.to_datetime(df['time'])
        df['time'] = enda.TimeSeries.align_timezone(df['time'], tzinfo =␣
↪'Europe/Paris')
        df.set_index('time', inplace=True)

    # keep only where both loads are known
    historic_load_measured = historic_load_measured.dropna()
    historic_load_measured["load_kw"] =␣
↪historic_load_measured["smart_metered_kw"] + historic_load_measured["slp_kw"]
    # keep only the full load
    historic_load_measured = historic_load_measured[["load_kw"]]


    return contracts, portfolio, historic_load_measured,␣
↪weather_and_tso_forecasts
```

[4]:
```
contracts, portfolio, historic_load_measured, weather_and_tso_forecasts =␣
↪read_data()
```

[5]:
```
contracts
```

[5]:

	date_start	date_end_exclusive	kva	meter_reading_type	contracts_count
0	2006-08-09	NaT	12.0	PROFILE	1
1	2006-09-01	2006-11-23	6.0	PROFILE	1
2	2006-09-01	2007-11-01	3.0	PROFILE	1
3	2006-09-01	2007-12-19	12.0	PROFILE	1
4	2006-09-01	2008-06-28	12.0	PROFILE	1
…	…	…	…	…	…
162598	2020-11-30	NaT	6.0	PROFILE	1
162599	2020-11-30	NaT	6.0	PROFILE	1
162600	2020-11-30	NaT	6.0	PROFILE	1
162601	2020-11-30	NaT	6.0	PROFILE	1
162602	2020-11-30	NaT	6.0	PROFILE	1

[162603 rows x 5 columns]

[6]:
```
portfolio
```

[6]:

time	contracts_count	kva
2006-08-09 00:00:00+02:00	1.0	12.0
2006-08-09 00:30:00+02:00	1.0	12.0
2006-08-09 01:00:00+02:00	1.0	12.0
2006-08-09 01:30:00+02:00	1.0	12.0
2006-08-09 02:00:00+02:00	1.0	12.0
…	…	…
2020-11-30 21:30:00+01:00	96134.0	820005.7
2020-11-30 22:00:00+01:00	96134.0	820005.7
2020-11-30 22:30:00+01:00	96134.0	820005.7
2020-11-30 23:00:00+01:00	96134.0	820005.7
2020-11-30 23:30:00+01:00	96134.0	820005.7

[250946 rows x 2 columns]

[7]:
```
historic_load_measured
```

[7]:

time	load_kw
2015-01-01 00:00:00+01:00	2490.925806
2015-01-01 00:30:00+01:00	2412.623113
2015-01-01 01:00:00+01:00	2365.611276
2015-01-01 01:30:00+01:00	2336.141065
2015-01-01 02:00:00+01:00	2300.935642
…	…
2020-11-15 21:30:00+01:00	7657.293444
2020-11-15 22:00:00+01:00	7317.540759
2020-11-15 22:30:00+01:00	7580.051439
2020-11-15 23:00:00+01:00	7496.273993

```
2020-11-15 23:30:00+01:00  7376.005701

[97198 rows x 1 columns]
```

[8]:
```
# t_weighted is the average french temperature weighted by population density
# t_smooth is a smoothing computed over t_weighted to take into account␣
 ↪building calorific inertia
# (t_smooth is computed out of enda here)

# some tso_forecast_load_mw is missing at the end (we don't show it here)
weather_and_tso_forecasts.dropna(subset=["tso_forecast_load_mw"])
```

[8]:

time	tso_forecast_load_mw	t_weighted	t_smooth
2015-01-01 00:00:00+01:00	72900.0	-0.41	1.17
2015-01-01 00:30:00+01:00	71600.0	-0.48	1.17
2015-01-01 01:00:00+01:00	69900.0	-0.55	1.15
2015-01-01 01:30:00+01:00	70600.0	-0.66	1.14
2015-01-01 02:00:00+01:00	70500.0	-0.78	1.11
…	…	…	…
2020-12-07 21:30:00+01:00	68400.0	4.20	4.13
2020-12-07 22:00:00+01:00	66900.0	4.12	4.10
2020-12-07 22:30:00+01:00	67600.0	4.03	4.08
2020-12-07 23:00:00+01:00	70200.0	3.94	4.07
2020-12-07 23:30:00+01:00	69600.0	3.94	4.07

[104064 rows x 3 columns]

[9]:
```
# lets create the train set with historical data
historic = pd.merge(
    portfolio,
    historic_load_measured, # here we select only the load of the desired group
    how='inner', left_index=True, right_index=True
)

historic = pd.merge(
    historic,
    weather_and_tso_forecasts,
    how='inner', left_index=True, right_index=True
)
```

[10]:
```
historic
```

[10]:

time	contracts_count	kva	load_kw	\
2015-01-01 00:00:00+01:00	21261.0	167416.4	2490.925806	
2015-01-01 00:30:00+01:00	21261.0	167416.4	2412.623113	

```
2015-01-01 01:00:00+01:00          21261.0  167416.4  2365.611276
2015-01-01 01:30:00+01:00          21261.0  167416.4  2336.141065
2015-01-01 02:00:00+01:00          21261.0  167416.4  2300.935642
…                                      …         …           …
2020-11-15 21:30:00+01:00          95475.0  813328.8  7657.293444
2020-11-15 22:00:00+01:00          95475.0  813328.8  7317.540759
2020-11-15 22:30:00+01:00          95475.0  813328.8  7580.051439
2020-11-15 23:00:00+01:00          95475.0  813328.8  7496.273993
2020-11-15 23:30:00+01:00          95475.0  813328.8  7376.005701

                           tso_forecast_load_mw  t_weighted  t_smooth
time
2015-01-01 00:00:00+01:00               72900.0       -0.41      1.17
2015-01-01 00:30:00+01:00               71600.0       -0.48      1.17
2015-01-01 01:00:00+01:00               69900.0       -0.55      1.15
2015-01-01 01:30:00+01:00               70600.0       -0.66      1.14
2015-01-01 02:00:00+01:00               70500.0       -0.78      1.11
…                                           …           …          …
2020-11-15 21:30:00+01:00               46200.0       12.05     12.01
2020-11-15 22:00:00+01:00               45200.0       11.92     11.97
2020-11-15 22:30:00+01:00               46400.0       11.84     11.96
2020-11-15 23:00:00+01:00               48600.0       11.75     11.94
2020-11-15 23:30:00+01:00               49400.0       11.64     11.92

[97198 rows x 6 columns]
```

[11]:
```
# check that there is no NaN value
historic.isna().sum()
```

[11]:
```
contracts_count         0
kva                     0
load_kw                 0
tso_forecast_load_mw    0
t_weighted              0
t_smooth                0
dtype: int64
```

[12]:
```
# note that the type of the index is precise
historic.index.dtype, type(historic.index)
```

[12]:
```
(datetime64[ns, Europe/Paris], pandas.core.indexes.datetimes.DatetimeIndex)
```

[13]:
```
# check missing data in the timeseries (based on the time index only)
freq, missing_periods, extra_points = enda.TimeSeries.
 ↪find_missing_and_extra_periods(
    dti=historic.index,
    expected_freq = '30min',
```

```
    expected_start_datetime = pd.to_datetime('2015-01-01 00:00:00+01:00').
↪astimezone('Europe/Paris'),
    expected_end_datetime = pd.to_datetime('2020-11-30 23:30:00+01:00').
↪astimezone('Europe/Paris')
)
for missing_period in missing_periods:
    print("Missing data from {} to {}.".format(missing_period[0],␣
↪missing_period[1]))
if len(extra_points) > 0 :
    print("Extra points found: {}".format(extra_points))
```

```
Missing data from 2015-09-01 00:00:00+02:00 to 2015-11-30 23:30:00+01:00.
Missing data from 2018-06-01 00:00:00+02:00 to 2018-06-30 23:30:00+02:00.
Missing data from 2020-11-16 00:00:00+01:00 to 2020-11-30 23:30:00+01:00.
```

We expected the missing data from 2020-11-16 to 2020-11-30, but not from the rest.

[14]:
```
# Zoom on a daylight savings time change to double-check that it was handled␣
↪correctly
historic[(historic.index >= '2019-10-27 01:00:00+02:00') & (historic.index <␣
↪'2019-10-27 03:30:00+01:00')]
```

[14]:

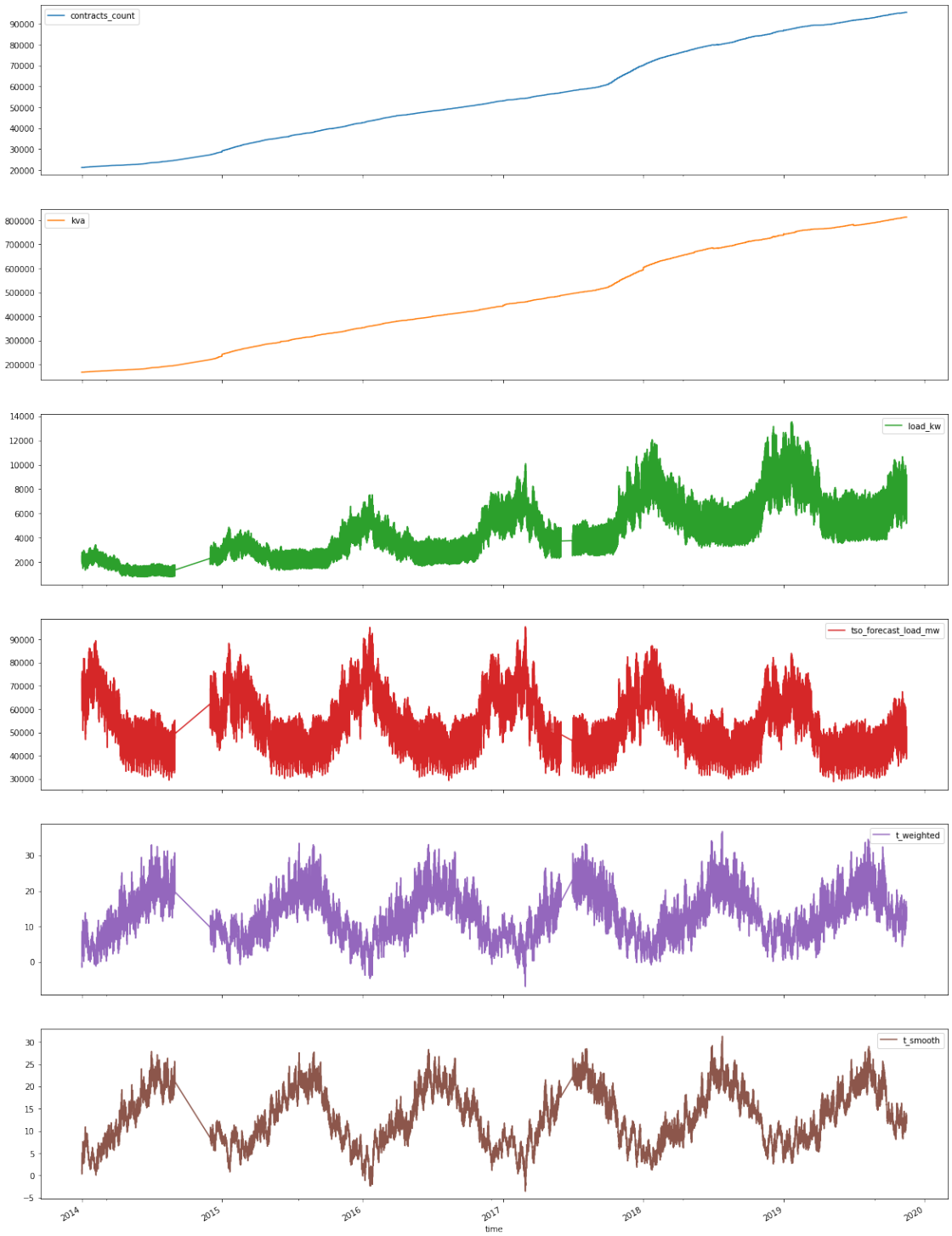
time	contracts_count	kva	load_kw	tso_forecast_load_mw	t_weighted	t_smooth
2019-10-27 01:00:00+02:00	84131.0	716816.4	5179.955556	41300.0	13.65	13.49
2019-10-27 01:30:00+02:00	84131.0	716816.4	5087.111111	40700.0	13.52	13.47
2019-10-27 02:00:00+02:00	84131.0	716816.4	4898.400000	36700.0	13.40	13.46
2019-10-27 02:30:00+02:00	84131.0	716816.4	4616.533333	36700.0	13.26	13.44
2019-10-27 02:00:00+01:00	84131.0	716816.4	4259.822222	36700.0	13.12	13.42
2019-10-27 02:30:00+01:00	84131.0	716816.4	4208.888889	36700.0	12.91	13.39
2019-10-27 03:00:00+01:00	84131.0	716816.4	4137.955556	36700.0	12.70	13.37

1.2 2. Visualize data

In order to visualise using pandas, we use the `matplotlib` backend.

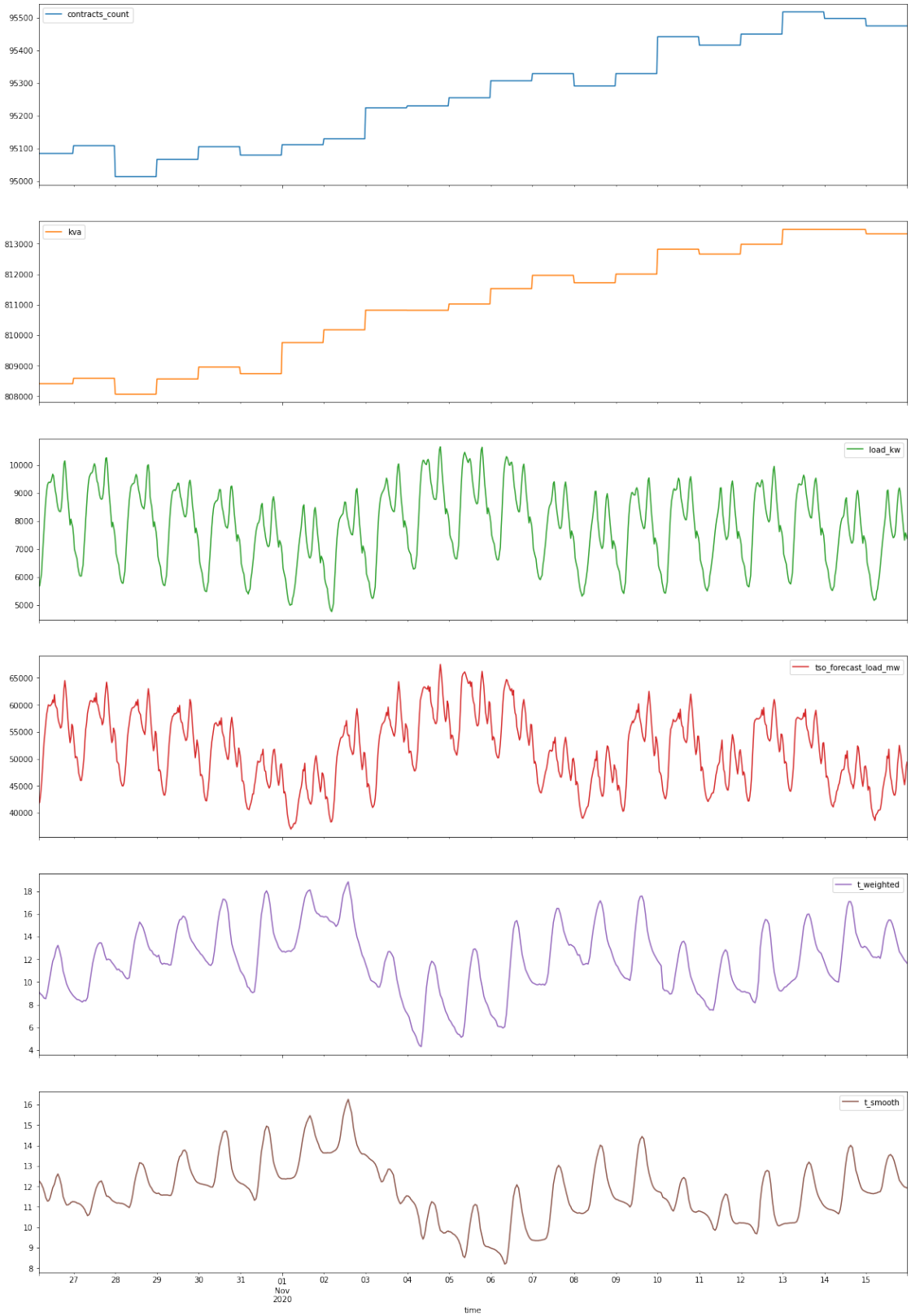
[15]:
```
# Show full data set
historic.plot(figsize=(20, 30), subplots=True)
```

[15]:
```
array([<AxesSubplot:xlabel='time'>, <AxesSubplot:xlabel='time'>,
       <AxesSubplot:xlabel='time'>, <AxesSubplot:xlabel='time'>,
       <AxesSubplot:xlabel='time'>, <AxesSubplot:xlabel='time'>],
      dtype=object)
```



[16]:
```
# Show recent data
historic[-1000:].plot(figsize=(20, 30), subplots=True)
```

[16]:
```
array([<AxesSubplot:xlabel='time'>, <AxesSubplot:xlabel='time'>,
       <AxesSubplot:xlabel='time'>, <AxesSubplot:xlabel='time'>,
       <AxesSubplot:xlabel='time'>, <AxesSubplot:xlabel='time'>],
      dtype=object)
```


contracts_count
95500
95400
95300
95200
95100
95000
kva
813000
812000
811000
810000
809000
808000
load_kw
10000
9000
8000
7000
6000
5000
tso_forecast_load_mw
65000
60000
55000
50000
45000
40000
t_weighted
18
16
14
12
10
8
6
4
t_smooth
16
15
14
13
12
11
10
9
8
27
28
29
30
31
01
Nov
2020
02
03
04
05
06
07
08
09
10
11
12
13
14
15
time

Don't hesitate to add your own visualisations!

[ ]:

1.3 3. Feature engineering

Before we train, we will add some features based on the `datetime`, and some calendar features related to national holidays or school holydays.

We use some packages for the holidays, which are used in **enda.feature_engineering.calendar** :

`pip install jours-feries-france vacances-scolaires-france Unidecode`

[17]:
```
import enda.feature_engineering.calendar
```

[18]:
```
# define the features we want to add before training/predicting
def featurize(df):
    # put datetime features to capture the data frequencies: daily, weekly and yearly periods.
    df = enda.DatetimeFeature.split_datetime(
        df, split_list = ['minuteofday', 'dayofweek', 'month']
    )
    df = enda.DatetimeFeature.encode_cyclic_datetime_index(
        df, split_list = ['minuteofday', 'dayofweek', 'dayofyear']
    )

    # add features about national holidays and school holidays (French holidays here)
    special_days = enda.feature_engineering.calendar.Calendar().get_french_special_days()
    df = pd.merge(
        df, special_days,
        how='left', left_index=True, right_index=True
    )
    return df
```

[19]:
```
full_train_set = featurize(historic)
```

[20]:
```
full_train_set
```

[20]:

time	contracts_count	kva	load_kw	\
2015-01-01 00:00:00+01:00	21261.0	167416.4	2490.925806	
2015-01-01 00:30:00+01:00	21261.0	167416.4	2412.623113	
2015-01-01 01:00:00+01:00	21261.0	167416.4	2365.611276	
2015-01-01 01:30:00+01:00	21261.0	167416.4	2336.141065	

```
2015-01-01 02:00:00+01:00          21261.0  167416.4  2300.935642
…                                      …       …          …
2020-11-15 21:30:00+01:00          95475.0  813328.8  7657.293444
2020-11-15 22:00:00+01:00          95475.0  813328.8  7317.540759
2020-11-15 22:30:00+01:00          95475.0  813328.8  7580.051439
2020-11-15 23:00:00+01:00          95475.0  813328.8  7496.273993
2020-11-15 23:30:00+01:00          95475.0  813328.8  7376.005701

                           tso_forecast_load_mw  t_weighted  t_smooth  \
time
2015-01-01 00:00:00+01:00               72900.0       -0.41      1.17
2015-01-01 00:30:00+01:00               71600.0       -0.48      1.17
2015-01-01 01:00:00+01:00               69900.0       -0.55      1.15
2015-01-01 01:30:00+01:00               70600.0       -0.66      1.14
2015-01-01 02:00:00+01:00               70500.0       -0.78      1.11
…                                           …          …         …
2020-11-15 21:30:00+01:00               46200.0       12.05     12.01
2020-11-15 22:00:00+01:00               45200.0       11.92     11.97
2020-11-15 22:30:00+01:00               46400.0       11.84     11.96
2020-11-15 23:00:00+01:00               48600.0       11.75     11.94
2020-11-15 23:30:00+01:00               49400.0       11.64     11.92

                           minuteofday  dayofweek  month  minuteofday_cos  \
time
2015-01-01 00:00:00+01:00            0          3      1         1.000000
2015-01-01 00:30:00+01:00           30          3      1         0.991445
2015-01-01 01:00:00+01:00           60          3      1         0.965926
2015-01-01 01:30:00+01:00           90          3      1         0.923880
2015-01-01 02:00:00+01:00          120          3      1         0.866025
…                                    …        …      …              …
2020-11-15 21:30:00+01:00         1290          6     11         0.793353
2020-11-15 22:00:00+01:00         1320          6     11         0.866025
2020-11-15 22:30:00+01:00         1350          6     11         0.923880
2020-11-15 23:00:00+01:00         1380          6     11         0.965926
2020-11-15 23:30:00+01:00         1410          6     11         0.991445

                           minuteofday_sin  dayofweek_cos  dayofweek_sin  \
time
2015-01-01 00:00:00+01:00         0.000000      -0.900969       0.433884
2015-01-01 00:30:00+01:00         0.130526      -0.900969       0.433884
2015-01-01 01:00:00+01:00         0.258819      -0.900969       0.433884
2015-01-01 01:30:00+01:00         0.382683      -0.900969       0.433884
2015-01-01 02:00:00+01:00         0.500000      -0.900969       0.433884
…                                      …             …              …
2020-11-15 21:30:00+01:00        -0.608761       0.623490      -0.781831
2020-11-15 22:00:00+01:00        -0.500000       0.623490      -0.781831
2020-11-15 22:30:00+01:00        -0.382683       0.623490      -0.781831
```

```
2020-11-15 23:00:00+01:00        -0.258819       0.623490      -0.781831
2020-11-15 23:30:00+01:00        -0.130526       0.623490      -0.781831

                           dayofyear_cos  dayofyear_sin  lockdown  \
time
2015-01-01 00:00:00+01:00       1.000000       0.000000       0.0
2015-01-01 00:30:00+01:00       1.000000       0.000000       0.0
2015-01-01 01:00:00+01:00       1.000000       0.000000       0.0
2015-01-01 01:30:00+01:00       1.000000       0.000000       0.0
2015-01-01 02:00:00+01:00       1.000000       0.000000       0.0
…                                  …              …        …
2020-11-15 21:30:00+01:00       0.691771      -0.722117       0.0
2020-11-15 22:00:00+01:00       0.691771      -0.722117       0.0
2020-11-15 22:30:00+01:00       0.691771      -0.722117       0.0
2020-11-15 23:00:00+01:00       0.691771      -0.722117       0.0
2020-11-15 23:30:00+01:00       0.691771      -0.722117       0.0

                           public_holiday  nb_school_areas_off  \
time
2015-01-01 00:00:00+01:00             1.0                  3.0
2015-01-01 00:30:00+01:00             1.0                  3.0
2015-01-01 01:00:00+01:00             1.0                  3.0
2015-01-01 01:30:00+01:00             1.0                  3.0
2015-01-01 02:00:00+01:00             1.0                  3.0
…                                     …                    …
2020-11-15 21:30:00+01:00             0.0                  0.0
2020-11-15 22:00:00+01:00             0.0                  0.0
2020-11-15 22:30:00+01:00             0.0                  0.0
2020-11-15 23:00:00+01:00             0.0                  0.0
2020-11-15 23:30:00+01:00             0.0                  0.0

                           extra_long_weekend
time
2015-01-01 00:00:00+01:00                 0.0
2015-01-01 00:30:00+01:00                 0.0
2015-01-01 01:00:00+01:00                 0.0
2015-01-01 01:30:00+01:00                 0.0
2015-01-01 02:00:00+01:00                 0.0
…                                         …
2020-11-15 21:30:00+01:00                 0.0
2020-11-15 22:00:00+01:00                 0.0
2020-11-15 22:30:00+01:00                 0.0
2020-11-15 23:00:00+01:00                 0.0
2020-11-15 23:30:00+01:00                 0.0

[97198 rows x 19 columns]
```

[21]:
```
# train a basic scikit-learn LinearRegression
from enda.ml_backends.sklearn_estimator import EndaSklearnEstimator
from sklearn.linear_model import LinearRegression

lin_reg = EndaSklearnEstimator(LinearRegression())
lin_reg.train(full_train_set, target_col='load_kw')
```

1.4 4. Portfolio forecast & basic prediction

We need an estimate of our portfolio in the next few days, the tso_load and weather forecasts.

In order to get our portfolio in the next few days, here we will just consider the latest trends in our portfolio.

In another setup, you might want to connect to your sales software or ERP and take into account contracts that will end or start soon.

We will use `enda.Contracts.forecast_portfolio_linear` (which requires the `sklearn` package).

[23]:
```
# we will forecast the portfolio using a linear method
forecast_portfolio = enda.Contracts.forecast_portfolio_linear(
    portfolio_df=portfolio[portfolio.index >= "2020-11-01 00:00:00+02:00"], # #␣
 ↪only use recent portfolio trend to forecast the next few days
    start_forecast_date=pd.to_datetime("2020-12-01 00:00:00+01:00").
 ↪tz_convert("Europe/Paris"),
    end_forecast_date_exclusive=pd.to_datetime("2020-12-08 00:00:00+01:00").
 ↪tz_convert("Europe/Paris"),
    freq='30min',
    tzinfo='Europe/Paris'
)
forecast_portfolio
```

[23]:

time	contracts_count	kva
2020-12-01 00:00:00+01:00	96024.460397	819113.699479
2020-12-01 00:30:00+01:00	96025.103312	819120.353482
2020-12-01 01:00:00+01:00	96025.746226	819127.007485
2020-12-01 01:30:00+01:00	96026.389140	819133.661488
2020-12-01 02:00:00+01:00	96027.032054	819140.315491
…	…	…
2020-12-07 21:30:00+01:00	96237.265007	821316.174461
2020-12-07 22:00:00+01:00	96237.907922	821322.828464
2020-12-07 22:30:00+01:00	96238.550836	821329.482467
2020-12-07 23:00:00+01:00	96239.193750	821336.136470
2020-12-07 23:30:00+01:00	96239.836664	821342.790473

[336 rows x 2 columns]

[24]:
```
# add weather_and_tso_forecasts
forecast_input_data = pd.merge(
    forecast_portfolio,
    weather_and_tso_forecasts.dropna(subset=["tso_forecast_load_mw"]), #␣
 ↪forecast only where tso is not null for now
    how='inner', left_index=True, right_index=True
)
# add feature engineering
forecast_input_data = featurize(forecast_input_data)
forecast_input_data
```

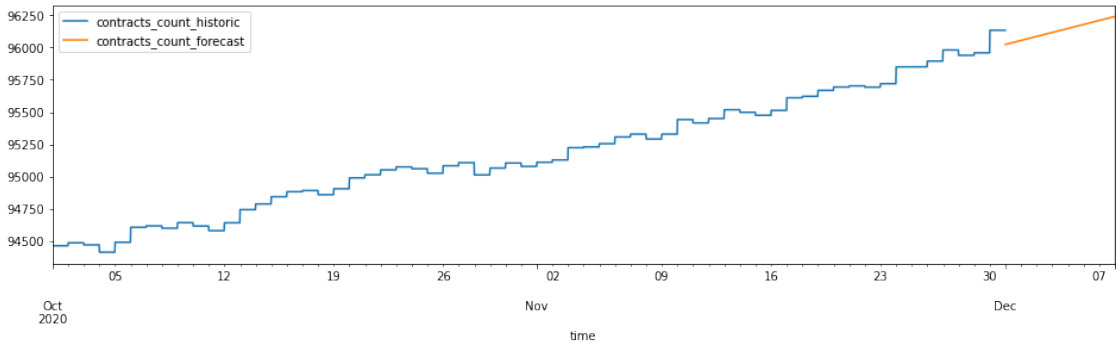
[24]:

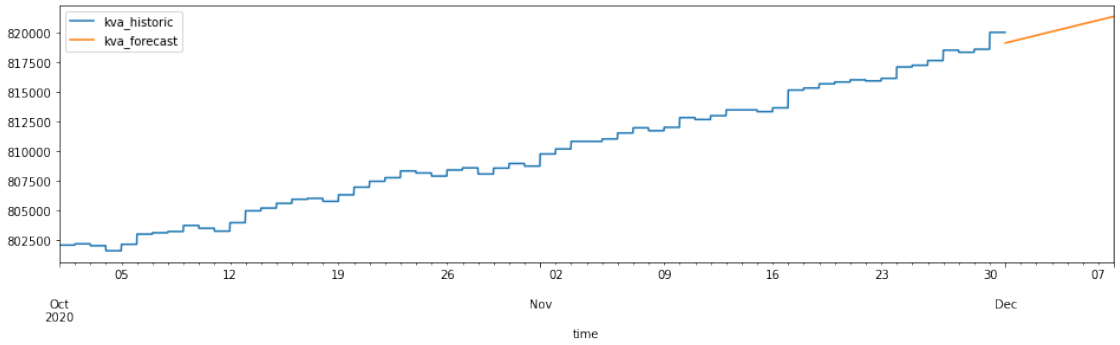
time	contracts_count	kva	tso_forecast_load_mw	t_weighted	t_smooth	minuteofday	dayofweek	month	minuteofday_cos
2020-12-01 00:00:00+01:00	96024.460397	819113.699479	66100.0	4.69	5.08	0	1	12	1.000000
2020-12-01 00:30:00+01:00	96025.103312	819120.353482	64200.0	4.82	5.10	30	1	12	0.991445
2020-12-01 01:00:00+01:00	96025.746226	819127.007485	61900.0	4.96	5.12	60	1	12	0.965926
2020-12-01 01:30:00+01:00	96026.389140	819133.661488	62800.0	5.04	5.13	90	1	12	0.923880
2020-12-01 02:00:00+01:00	96027.032054	819140.315491	62300.0	5.13	5.14	120	1	12	0.866025
…	…	…	…	…	…	…	…	…	…
2020-12-07 21:30:00+01:00	96237.265007	821316.174461	68400.0	4.20	4.13				
2020-12-07 22:00:00+01:00	96237.907922	821322.828464	66900.0	4.12	4.10				
2020-12-07 22:30:00+01:00	96238.550836	821329.482467	67600.0	4.03	4.08				
2020-12-07 23:00:00+01:00	96239.193750	821336.136470	70200.0	3.94	4.07				
2020-12-07 23:30:00+01:00	96239.836664	821342.790473	69600.0	3.94	4.07				

```
2020-12-07 21:30:00+01:00         1290          0     12         0.793353
2020-12-07 22:00:00+01:00         1320          0     12         0.866025
2020-12-07 22:30:00+01:00         1350          0     12         0.923880
2020-12-07 23:00:00+01:00         1380          0     12         0.965926
2020-12-07 23:30:00+01:00         1410          0     12         0.991445

                           minuteofday_sin  dayofweek_cos  dayofweek_sin  \
time
2020-12-01 00:00:00+01:00         0.000000        0.62349       0.781831
2020-12-01 00:30:00+01:00         0.130526        0.62349       0.781831
2020-12-01 01:00:00+01:00         0.258819        0.62349       0.781831
2020-12-01 01:30:00+01:00         0.382683        0.62349       0.781831
2020-12-01 02:00:00+01:00         0.500000        0.62349       0.781831
…                                      …              …              …
2020-12-07 21:30:00+01:00        -0.608761        1.00000       0.000000
2020-12-07 22:00:00+01:00        -0.500000        1.00000       0.000000
2020-12-07 22:30:00+01:00        -0.382683        1.00000       0.000000
2020-12-07 23:00:00+01:00        -0.258819        1.00000       0.000000
2020-12-07 23:30:00+01:00        -0.130526        1.00000       0.000000

                           dayofyear_cos  dayofyear_sin  lockdown  \
time
2020-12-01 00:00:00+01:00       0.861702      -0.507415       0.0
2020-12-01 00:30:00+01:00       0.861702      -0.507415       0.0
2020-12-01 01:00:00+01:00       0.861702      -0.507415       0.0
2020-12-01 01:30:00+01:00       0.861702      -0.507415       0.0
2020-12-01 02:00:00+01:00       0.861702      -0.507415       0.0
…                                    …             …       …
2020-12-07 21:30:00+01:00       0.909308      -0.416125       0.0
2020-12-07 22:00:00+01:00       0.909308      -0.416125       0.0
2020-12-07 22:30:00+01:00       0.909308      -0.416125       0.0
2020-12-07 23:00:00+01:00       0.909308      -0.416125       0.0
2020-12-07 23:30:00+01:00       0.909308      -0.416125       0.0

                           public_holiday  nb_school_areas_off  \
time
2020-12-01 00:00:00+01:00             0.0                  0.0
2020-12-01 00:30:00+01:00             0.0                  0.0
2020-12-01 01:00:00+01:00             0.0                  0.0
2020-12-01 01:30:00+01:00             0.0                  0.0
2020-12-01 02:00:00+01:00             0.0                  0.0
…                                     …                    …
2020-12-07 21:30:00+01:00             0.0                  0.0
2020-12-07 22:00:00+01:00             0.0                  0.0
2020-12-07 22:30:00+01:00             0.0                  0.0
2020-12-07 23:00:00+01:00             0.0                  0.0
2020-12-07 23:30:00+01:00             0.0                  0.0
```

```
                           extra_long_weekend
time
2020-12-01 00:00:00+01:00                 0.0
2020-12-01 00:30:00+01:00                 0.0
2020-12-01 01:00:00+01:00                 0.0
2020-12-01 01:30:00+01:00                 0.0
2020-12-01 02:00:00+01:00                 0.0
…                                         …
2020-12-07 21:30:00+01:00                 0.0
2020-12-07 22:00:00+01:00                 0.0
2020-12-07 22:30:00+01:00                 0.0
2020-12-07 23:00:00+01:00                 0.0
2020-12-07 23:30:00+01:00                 0.0

[336 rows x 18 columns]
```

[25]:
```python
# show recent portfolio and forecast
for c in ["contracts_count", "kva"]:
    to_plot = pd.merge(
        portfolio[(portfolio.index >= '2020-10-01')][c].to_frame(c+"_historic"),
        forecast_input_data[c].to_frame(c+"_forecast"),
        how='outer', left_index=True, right_index=True
    )

    to_plot.plot(figsize=(16, 4))
```

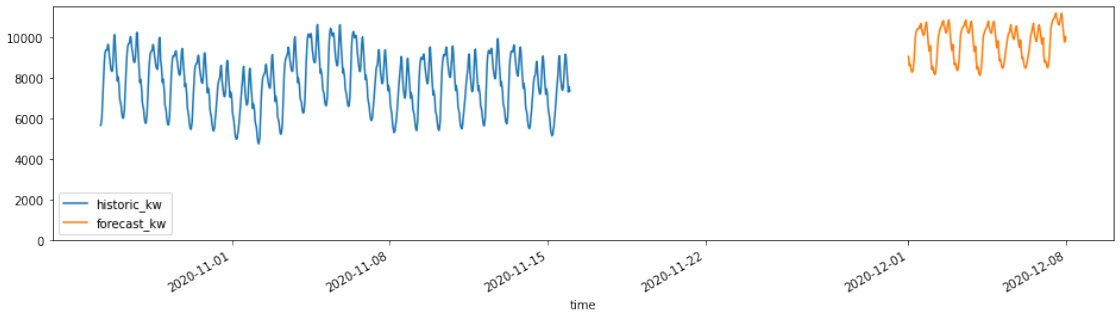




[26]:
```
# do the prediction
lin_reg_prediction = lin_reg.predict(forecast_input_data, target_col="load_kw")
```

[27]:
```
# visualize recent load along with our forecast.
# remember we don't have recent actual load so there is a time-gap.
to_plot = pd.merge(
    historic["load_kw"][-1000:].to_frame("historic_kw"),
    lin_reg_prediction.rename(columns={"load_kw": "forecast_kw"}),
    how='outer', left_index=True, right_index=True
)
to_plot.plot(ylim=0, figsize=(16, 4))
```

[27]: <AxesSubplot:xlabel='time'>



1.5 5. Benchmark with simple evaluation

The previous forecast based on linear regression is very limited. Let's try and use a better algorithm !

We will define some algorithms using `scikit-klearn` as a machine learning backend and others using `h2o`.

For that we need the `h2o` package:

```
pip install h2o
```

[28]:
```
# here we do a benchmark, we want to compare with actual data,
# lets say from 2020-11-01 to 2020-11-15
benchmark_train = full_train_set[full_train_set.index < '2020-11-01']
benchmark_test = full_train_set[full_train_set.index >= '2020-11-01']

# save the actual_load in a 'benchmark' dataframe,
# we will add the predictions of each algo to 'benchmark'
benchmark = benchmark_test["load_kw"].to_frame("actual_load_kw")

benchmark_test = benchmark_test.drop(columns=["load_kw"])
```

[29]:
```
# some parts give ConvergenceWarnings here and we'll ignore them.
import warnings
warnings.filterwarnings('ignore')
```

[30]:
```
# use the same method as before to predict a portfolio for 2020-11-01 ->␣
↪2020-11-15
benchmark_test_portfolio = forecast_portfolio = enda.Contracts.
↪forecast_portfolio_linear(
    portfolio_df=portfolio[(portfolio.index >= '2020-10-01') & (portfolio.index␣
↪< '2020-11-01')],
    start_forecast_date=pd.to_datetime("2020-11-01 00:00:00+01:00").
↪tz_convert("Europe/Paris"),
    end_forecast_date_exclusive=pd.to_datetime("2020-11-16 00:00:00+01:00").
↪tz_convert("Europe/Paris"),
    freq='30min',
    tzinfo='Europe/Paris'
)

benchmark_test['kva'] = benchmark_test_portfolio['kva']
benchmark_test['contracts_count'] = benchmark_test_portfolio['contracts_count']
benchmark_test
```

[30]:

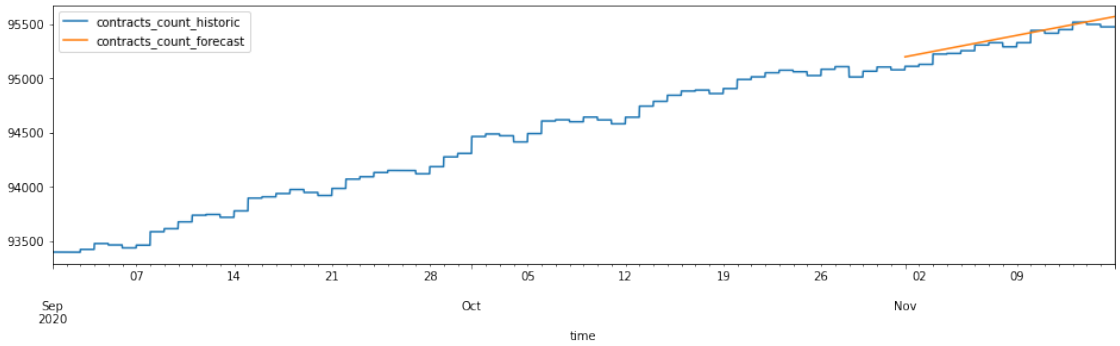
time	contracts_count	kva
2020-11-01 00:00:00+01:00	95198.664499	809667.353998
2020-11-01 00:30:00+01:00	95199.180488	809672.890306
2020-11-01 01:00:00+01:00	95199.696477	809678.426615
2020-11-01 01:30:00+01:00	95200.212466	809683.962923
2020-11-01 02:00:00+01:00	95200.728455	809689.499231
…	…	…
2020-11-15 21:30:00+01:00	95567.596700	813625.814355
2020-11-15 22:00:00+01:00	95568.112689	813631.350663
2020-11-15 22:30:00+01:00	95568.628678	813636.886971
2020-11-15 23:00:00+01:00	95569.144667	813642.423280

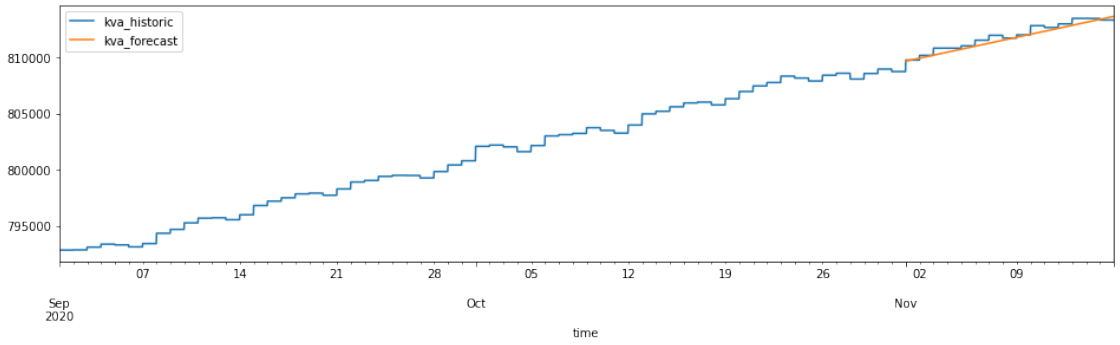
```
2020-11-15 23:30:00+01:00     95569.660656  813647.959588

                          tso_forecast_load_mw  t_weighted  t_smooth  \
time
2020-11-01 00:00:00+01:00               47900.0       12.67     12.37
2020-11-01 00:30:00+01:00               45800.0       12.68     12.37
2020-11-01 01:00:00+01:00               43700.0       12.70     12.37
2020-11-01 01:30:00+01:00               43900.0       12.66     12.37
2020-11-01 02:00:00+01:00               43200.0       12.63     12.36
…                                           …           …         …
2020-11-15 21:30:00+01:00               46200.0       12.05     12.01
2020-11-15 22:00:00+01:00               45200.0       11.92     11.97
2020-11-15 22:30:00+01:00               46400.0       11.84     11.96
2020-11-15 23:00:00+01:00               48600.0       11.75     11.94
2020-11-15 23:30:00+01:00               49400.0       11.64     11.92

                          minuteofday  dayofweek  month  minuteofday_cos  \
time
2020-11-01 00:00:00+01:00           0          6     11         1.000000
2020-11-01 00:30:00+01:00          30          6     11         0.991445
2020-11-01 01:00:00+01:00          60          6     11         0.965926
2020-11-01 01:30:00+01:00          90          6     11         0.923880
2020-11-01 02:00:00+01:00         120          6     11         0.866025
…                                   …        …     …              …
2020-11-15 21:30:00+01:00        1290          6     11         0.793353
2020-11-15 22:00:00+01:00        1320          6     11         0.866025
2020-11-15 22:30:00+01:00        1350          6     11         0.923880
2020-11-15 23:00:00+01:00        1380          6     11         0.965926
2020-11-15 23:30:00+01:00        1410          6     11         0.991445

                          minuteofday_sin  dayofweek_cos  dayofweek_sin  \
time
2020-11-01 00:00:00+01:00         0.000000        0.62349      -0.781831
2020-11-01 00:30:00+01:00         0.130526        0.62349      -0.781831
2020-11-01 01:00:00+01:00         0.258819        0.62349      -0.781831
2020-11-01 01:30:00+01:00         0.382683        0.62349      -0.781831
2020-11-01 02:00:00+01:00         0.500000        0.62349      -0.781831
…                                      …              …              …
2020-11-15 21:30:00+01:00        -0.608761        0.62349      -0.781831
2020-11-15 22:00:00+01:00        -0.500000        0.62349      -0.781831
2020-11-15 22:30:00+01:00        -0.382683        0.62349      -0.781831
2020-11-15 23:00:00+01:00        -0.258819        0.62349      -0.781831
2020-11-15 23:30:00+01:00        -0.130526        0.62349      -0.781831

                          dayofyear_cos  dayofyear_sin  lockdown  \
time
2020-11-01 00:00:00+01:00       0.500000      -0.866025       0.0
```

```
2020-11-01 00:30:00+01:00       0.500000      -0.866025       0.0
2020-11-01 01:00:00+01:00       0.500000      -0.866025       0.0
2020-11-01 01:30:00+01:00       0.500000      -0.866025       0.0
2020-11-01 02:00:00+01:00       0.500000      -0.866025       0.0
…                                    …             …       …
2020-11-15 21:30:00+01:00       0.691771      -0.722117       0.0
2020-11-15 22:00:00+01:00       0.691771      -0.722117       0.0
2020-11-15 22:30:00+01:00       0.691771      -0.722117       0.0
2020-11-15 23:00:00+01:00       0.691771      -0.722117       0.0
2020-11-15 23:30:00+01:00       0.691771      -0.722117       0.0

                           public_holiday  nb_school_areas_off  \
time
2020-11-01 00:00:00+01:00             1.0                  3.0
2020-11-01 00:30:00+01:00             1.0                  3.0
2020-11-01 01:00:00+01:00             1.0                  3.0
2020-11-01 01:30:00+01:00             1.0                  3.0
2020-11-01 02:00:00+01:00             1.0                  3.0
…                                     …                    …
2020-11-15 21:30:00+01:00             0.0                  0.0
2020-11-15 22:00:00+01:00             0.0                  0.0
2020-11-15 22:30:00+01:00             0.0                  0.0
2020-11-15 23:00:00+01:00             0.0                  0.0
2020-11-15 23:30:00+01:00             0.0                  0.0

                           extra_long_weekend
time
2020-11-01 00:00:00+01:00                 0.0
2020-11-01 00:30:00+01:00                 0.0
2020-11-01 01:00:00+01:00                 0.0
2020-11-01 01:30:00+01:00                 0.0
2020-11-01 02:00:00+01:00                 0.0
…                                         …
2020-11-15 21:30:00+01:00                 0.0
2020-11-15 22:00:00+01:00                 0.0
2020-11-15 22:30:00+01:00                 0.0
2020-11-15 23:00:00+01:00                 0.0
2020-11-15 23:30:00+01:00                 0.0

[720 rows x 18 columns]
```

[31]:
```
# compare portfolio forecast to reality
for c in ["contracts_count", "kva"]:
    to_plot = pd.merge(
        portfolio[(portfolio.index >= '2020-09-01') & (portfolio.index <␣
↪'2020-11-16')][c].to_frame(c+"_historic"),
        benchmark_test[c].to_frame(c+"_forecast"),
```

```
        how='outer', left_index=True, right_index=True
    )

    to_plot.plot(figsize=(16, 4))
```





Lets define some algorithms then train and predict with them. All the models we define implement the `enda.estimators.EndaEstimator` abstract class (see the docs).

Enda comes with wrappers around scikit-learn and H2O estimators : - sklearn: `enda.ml_backends.sklearn_estimator.EndaSklearnEstimator` - H2O: `enda.ml_backends.h2o_estimator.EndaH2OEstimator`

[32]:
```
import time
import h2o
import random
import numpy

from sklearn.linear_model import LinearRegression, SGDRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

```
from enda.ml_backends.h2o_estimator import EndaH2OEstimator  # enda's wrapper␣
↪around H2O models
from h2o.estimators import H2OGeneralizedLinearEstimator
from h2o.estimators import H2OXGBoostEstimator
from h2o.estimators import H2OGradientBoostingEstimator
from h2o.estimators import H2ORandomForestEstimator
from h2o.estimators import H2ODeepLearningEstimator
```

[33]:
```
random.seed(17)  # set random seed for reproducibility
numpy.random.seed(17) # for sklearn
# for h2o we will define it in each model
```

[34]:
```
all_models = dict()
```

[35]:
```
# Some models with the sklearn machine learning backend

all_models['sklearn_lin_reg'] = EndaSklearnEstimator(LinearRegression())

all_models['sklearn_sgd'] = EndaSklearnEstimator(
    Pipeline([('standard_scaler', StandardScaler()),
              ('sgd', SGDRegressor())
             ]
            )
)

all_models['sklearn_ada_boost'] = EndaSklearnEstimator(AdaBoostRegressor(
    n_estimators=500,
    loss='square',
    learning_rate=0.8)
)

all_models['sklearn_nn'] = EndaSklearnEstimator(
    Pipeline([('standard_scaler', StandardScaler()),
              ('mlp', MLPRegressor(
                  solver='adam',
                  activation='relu',
                  hidden_layer_sizes=[48, 48, 24],
                  max_iter=150
              ))
             ]
            )
)
```

[36]:
```
# Some models with the h2o machine learning backend

all_models['h2o_glm'] = EndaH2OEstimator(H2OGeneralizedLinearEstimator(
```

```
    standardize=False,
    intercept=True,
    seed=17)
)

all_models['h2o_rf'] = EndaH2OEstimator(H2ORandomForestEstimator(
    ntrees=300,
    max_depth=15,
    sample_rate=0.8,
    min_rows=10,
    nbins=52,
    mtries=3,
    seed=17
))

all_models['h2o_gbm'] = EndaH2OEstimator(H2OGradientBoostingEstimator(
    ntrees=500,
    max_depth=5,
    sample_rate=0.5,
    min_rows=5,
    seed=17
))

all_models['h2o_xgboost'] = EndaH2OEstimator(H2OXGBoostEstimator(
    **{
        "ntrees": 500,
        "max_depth": 5,
        "sample_rate": 0.8,
        "min_rows": 10,
        "seed": 17
    }
))

all_models['h2o_nn'] = EndaH2OEstimator(H2ODeepLearningEstimator(
    **{
        "activation": "Tanh",
        "hidden": [48, 48, 24],
        "distribution": "gaussian",
        "epochs": 20,
        "seed": 17
    }
))
```

[37]:
```
# You can add more models to the benchmark here if you like
```

[38]:
```
# to train or predict with H2O models, we boot up a local h2o server
h2o.init(nthreads=-1)
```

```
h2o.no_progress()
```

```
Checking whether there is an H2O instance running at http://localhost:54321 .
connected.

-------------------------- ␣
 ↪------------------------------------------------------------------
H2O_cluster_uptime:         2 hours 29 mins
H2O_cluster_timezone:       Europe/Paris
H2O_data_parsing_timezone:  UTC
H2O_cluster_version:        3.32.0.4
H2O_cluster_version_age:    2 months
H2O_cluster_name:           H2O_from_python_emmanuel_charon_xqjnib
H2O_cluster_total_nodes:    1
H2O_cluster_free_memory:    3.861 Gb
H2O_cluster_total_cores:    4
H2O_cluster_allowed_cores:  4
H2O_cluster_status:         locked, healthy
H2O_connection_url:         http://localhost:54321
H2O_connection_proxy:       {"http": null, "https": null}
H2O_internal_security:      False
H2O_API_Extensions:         Amazon S3, XGBoost, Algos, AutoML, Core V3,␣
 ↪TargetEncoder, Core V4
Python_version:             3.7.6 final
-------------------------- ␣
 ↪------------------------------------------------------------------
```

[39]:
```
# this should take between 5 and 15 minutes to run (in function of your␣
 ↪hardware)
print("Benchmark with {} models : {}\n".format(len(all_models), list(all_models.
 ↪keys())))
for model_name, model in all_models.items():
    model_start_time = time.time()
    print("Training {} before predicting with it..".format(model_name))
    model.train(benchmark_train, target_col='load_kw')
    model_prediction = model.predict(benchmark_test, target_col='load_kw')
    benchmark[model_name] = model_prediction
    print("{} took {:.1f} seconds.\n".format(model_name, time.
 ↪time()-model_start_time))
```

```
Benchmark with 9 models : ['sklearn_lin_reg', 'sklearn_sgd',
'sklearn_ada_boost', 'sklearn_nn', 'h2o_glm', 'h2o_rf', 'h2o_gbm',
'h2o_xgboost', 'h2o_nn']

Training sklearn_lin_reg before predicting with it..
sklearn_lin_reg took 0.1 seconds.

Training sklearn_sgd before predicting with it..
```

```
sklearn_sgd took 1.5 seconds.

Training sklearn_ada_boost before predicting with it..
sklearn_ada_boost took 62.8 seconds.

Training sklearn_nn before predicting with it..
sklearn_nn took 62.7 seconds.

Training h2o_glm before predicting with it..
h2o_glm took 2.5 seconds.

Training h2o_rf before predicting with it..
h2o_rf took 35.4 seconds.

Training h2o_gbm before predicting with it..
h2o_gbm took 22.9 seconds.

Training h2o_xgboost before predicting with it..
h2o_xgboost took 56.5 seconds.

Training h2o_nn before predicting with it..
h2o_nn took 87.4 seconds.
```

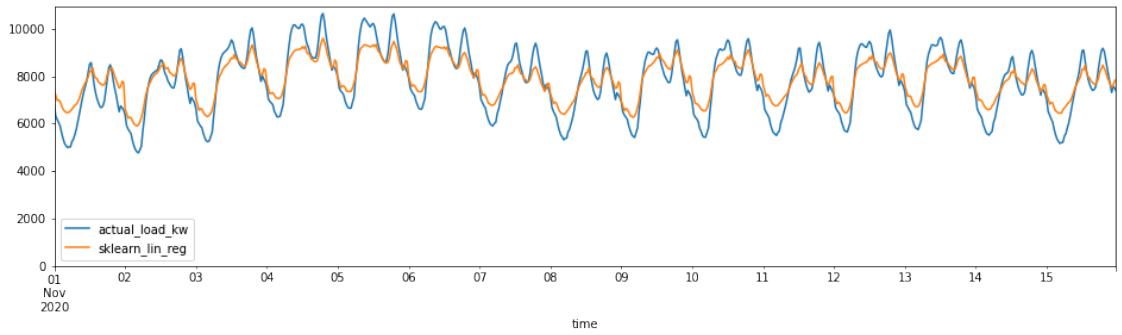
[40]:
```
benchmark
```

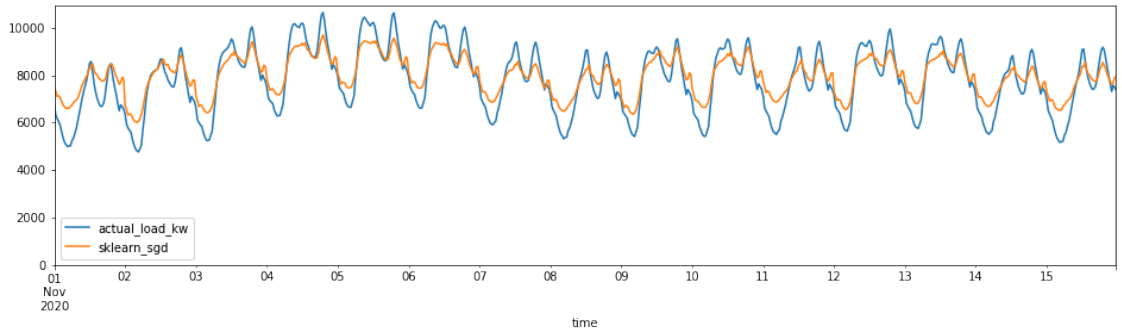
[40]:

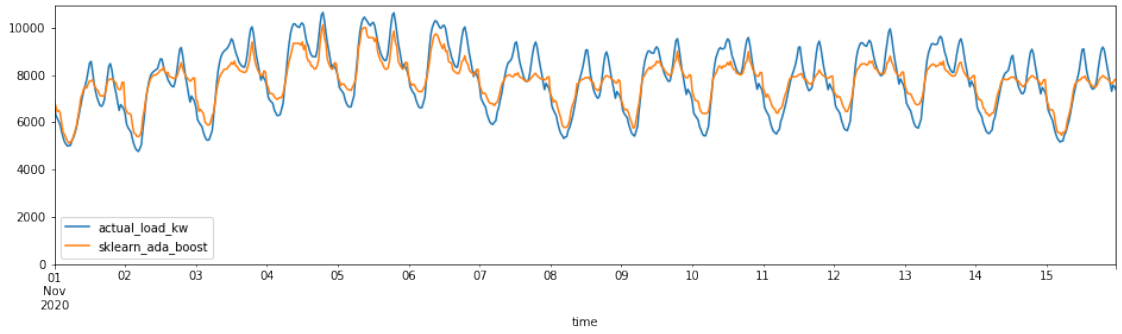
time	actual_load_kw	sklearn_lin_reg	sklearn_sgd	sklearn_ada_boost	sklearn_nn	h2o_glm
2020-11-01 00:00:00+01:00	6817.332090	7422.322246	7551.289345	6928.529317	7359.855183	7411.803676
2020-11-01 00:30:00+01:00	6326.667322	7198.905565	7328.454590	6704.866625	7017.206346	7173.577001
2020-11-01 01:00:00+01:00	6172.223671	6981.068131	7111.187034	6459.074008	6675.912928	6935.350325
2020-11-01 01:30:00+01:00	6050.575318	6999.743234	7131.405011	6501.046058	6554.039623	6958.100006
2020-11-01 02:00:00+01:00	5898.881230	6934.923230	7067.670349	6408.101013	6336.996037	6878.728504
…	…	…	…			
2020-11-15 21:30:00+01:00	7657.293444	7662.094726	7735.127624			
2020-11-15 22:00:00+01:00	7317.540759	7529.936137	7603.660786			
2020-11-15 22:30:00+01:00	7580.051439	7618.314141	7693.768759			
2020-11-15 23:00:00+01:00	7496.273993	7809.893704	7887.543159			
2020-11-15 23:30:00+01:00	7376.005701	7867.426287	7946.817362			

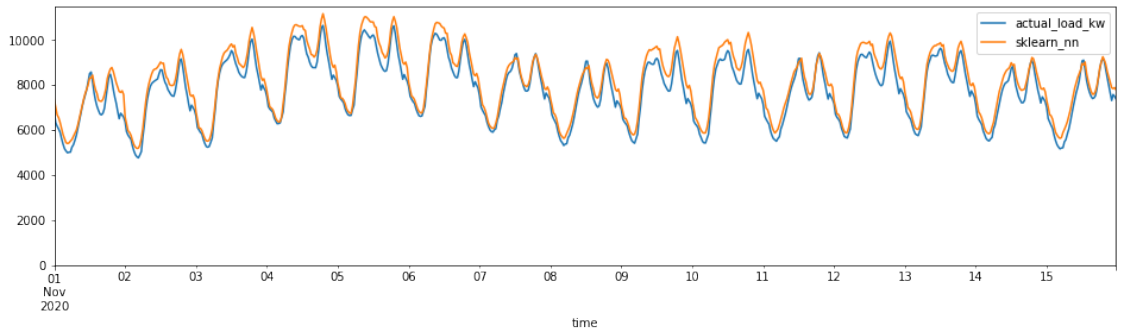
```
…                                    …          …          …
2020-11-15 21:30:00+01:00        7705.219899  8127.213107  7259.008463
2020-11-15 22:00:00+01:00        7621.020816  7883.145256  7145.596567
2020-11-15 22:30:00+01:00        7699.113183  7843.329704  7281.814228
2020-11-15 23:00:00+01:00        7802.525684  7890.337302  7531.499870
2020-11-15 23:30:00+01:00        7810.880919  7789.970024  7622.330339

                                     h2o_rf      h2o_gbm  h2o_xgboost       h2o_nn
time
2020-11-01 00:00:00+01:00  6537.198906  7020.909350  7064.977051  6817.101140
2020-11-01 00:30:00+01:00  6228.407756  6418.836376  6525.090332  6524.103421
2020-11-01 01:00:00+01:00  6058.203420  6217.635216  6384.663086  6226.189696
2020-11-01 01:30:00+01:00  6019.727900  6065.009571  6292.012207  6127.618693
2020-11-01 02:00:00+01:00  5839.442222  5972.858393  6198.620605  5981.783909
…                                  …            …            …            …
2020-11-15 21:30:00+01:00  7483.921108  7272.697440  7436.578125  7847.286864
2020-11-15 22:00:00+01:00  7369.696343  7050.622522  7187.884766  7610.507253
2020-11-15 22:30:00+01:00  7411.223903  7247.823333  7486.856934  7563.156549
2020-11-15 23:00:00+01:00  7425.497777  7294.028407  7286.999512  7598.105915
2020-11-15 23:30:00+01:00  7458.976123  7117.461922  7210.519043  7537.384211

[720 rows x 10 columns]
```

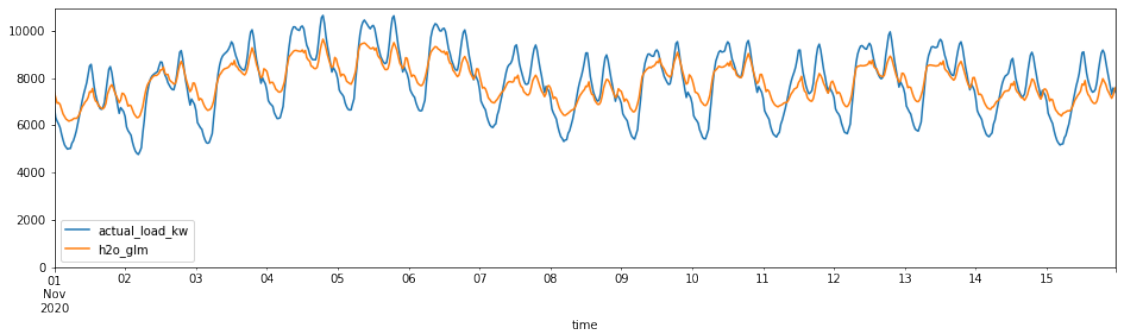
[41]:
```
# visualize predictions
for c in benchmark.columns:
    if c != "actual_load_kw":
        to_plot = benchmark[["actual_load_kw", c]]
        to_plot.plot(ylim=0, figsize=(16, 4))
```

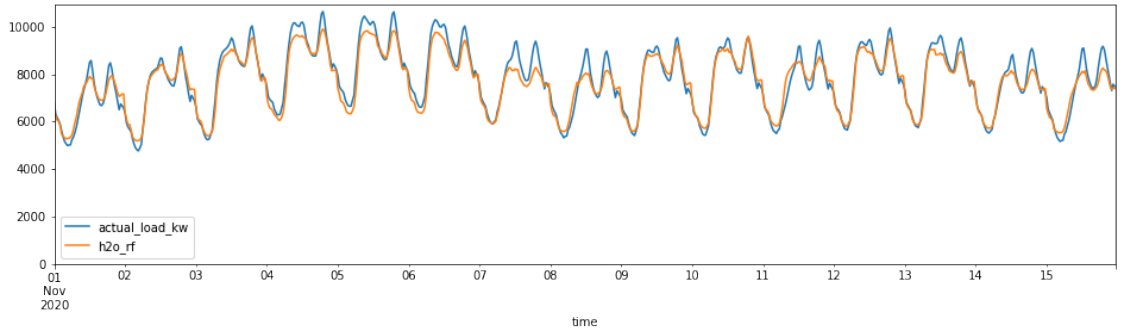


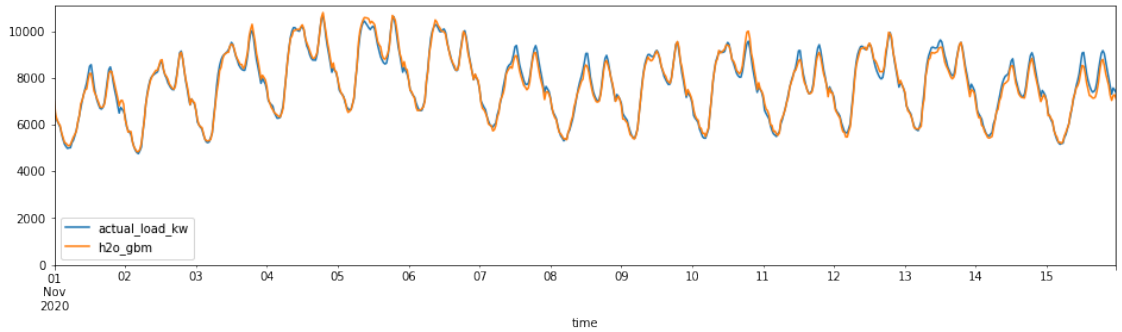

actual_load_kw
sklearn_sgd
time

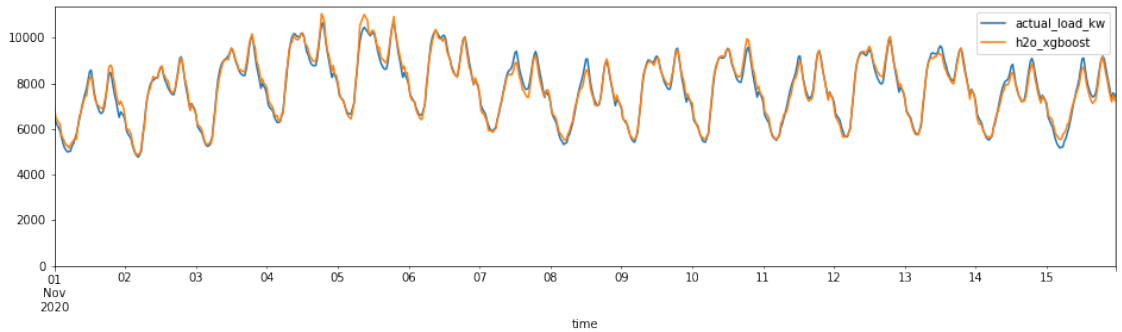

actual_load_kw
sklearn_ada_boost
time

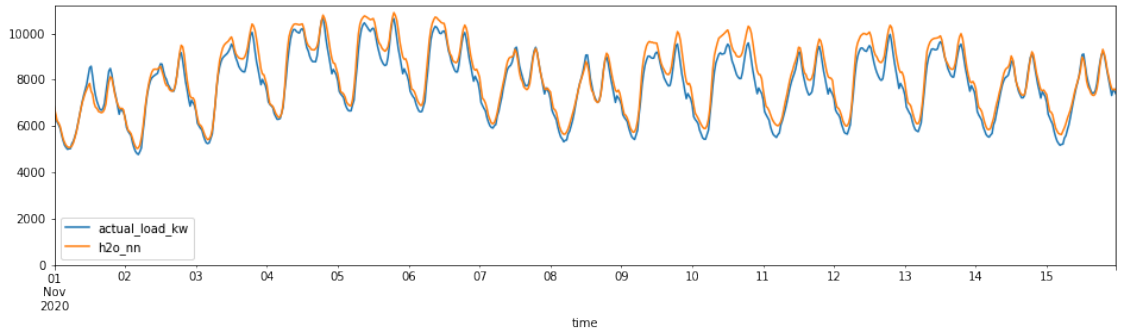

actual_load_kw
sklearn_nn
time


actual_load_kw
h2o_glm
time


10000
8000
6000
4000
2000
0
actual_load_kw
h2o_rf
01
Nov
2020
02
03
04
05
06
07
08
09
10
11
12
13
14
15
time


10000
8000
6000
4000
2000
0
actual_load_kw
h2o_gbm
01
Nov
2020
02
03
04
05
06
07
08
09
10
11
12
13
14
15
time


10000
8000
6000
4000
2000
0
actual_load_kw
h2o_xgboost
01
Nov
2020
02
03
04
05
06
07
08
09
10
11
12
13
14
15
time



[42]:
```
# compute the mean absolute percentage error of each algo
from enda.scoring import Scoring
```

[43]:
```
scoring = Scoring(predictions_df=benchmark, target="actual_load_kw")
scoring.mean_absolute_percentage_error().to_frame("mape")
```

[43]:

	mape
sklearn_lin_reg	7.198416
sklearn_sgd	7.614963
sklearn_ada_boost	6.458338
sklearn_nn	5.454028
h2o_glm	9.054567
h2o_rf	3.425527
h2o_gbm	1.722011
h2o_xgboost	2.003797
h2o_nn	4.469578

1.6 6. Benchmark with Backtesting

In traditional machine learning, we need more than just 1 evaluation to test an algorithm. We typically use cross-validation to see if the algorithm is not biased and if it can be expected to work well in most cases. For time-series predictions we cannot do a regular cross-validation because it is not realistic : we always want to train using historical data that happened before the prediction.

Here we will do **backtesting** week after week. With the given dataset, this means : - for each week w from early 2019 until the end of the dataset : train using data from the beginning of the dataset (early 2015) until a few days before week w, then eval on w. - the first iteration will train an algorithm using data from 2015 to 2018, then eval on the first week of 2019 - the second iteration will train using data from 2015 to a bit before the first week of 2019, then eval on the second week of 2019 - and so on… - keep the predictions of each time-step using this method, from early 2019 to november 2020.
- then compare these predictions to the historic data to evaluate the quality of each algorithm.

This makes most sense if in your production environment, you plan to retrain the algorithm regularly with recent data.

Backtesting can take a significant amount of time. We backtest only 2 linear regressions below in order to have an example that runs fast. Don't hesitate to add other algorithms.

[44]:
```
all_models = dict()

all_models['sklearn_lin_reg'] = EndaSklearnEstimator(LinearRegression())

all_models['h2o_glm'] =␣
↪EndaH2OEstimator(H2OGeneralizedLinearEstimator(standardize=False,␣
↪intercept=True))
```

[45]:
```
from dateutil.relativedelta import relativedelta
from enda.timezone_utils import TimezoneUtils
portfolio_train_length = relativedelta(months=1)
```

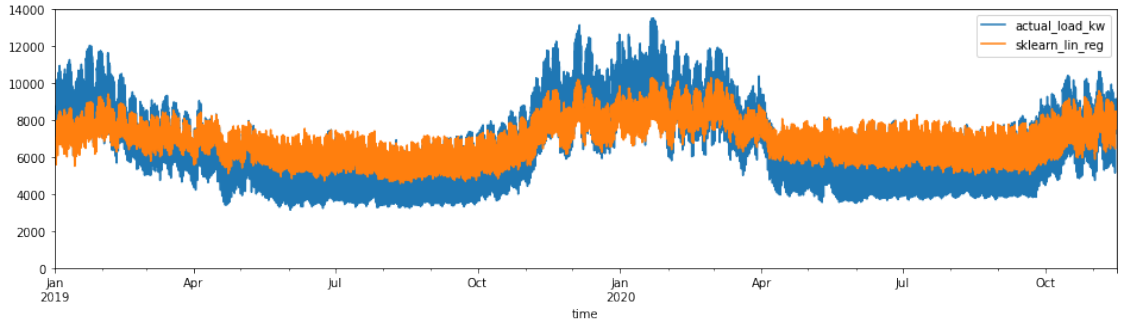
[46]:
```
start_backtesting_dt = pd.to_datetime('2019-01-01 00:00:00+01:00').
↪tz_convert('Europe/Paris')
benchmark = historic[historic.index>=start_backtesting_dt]["load_kw"].
↪to_frame("actual_load_kw")
days_in_each_iteration = 28


for model_name, model in all_models.items():

    count_iterations = 0

    model_predictions = []

    for train_set, test_set in enda.BackTesting.yield_train_test(
        historic,
        start_eval_datetime=start_backtesting_dt,
        days_between_trains=days_in_each_iteration,
        gap_days_between_train_and_eval=14
    ):
        count_iterations += 1
        if count_iterations <= 2 or count_iterations % 10 == 0:
            print("Model {}, backtesting iteration {}, train set {}->{}, test␣
↪set {}->{}\n".format(
                    model_name, count_iterations,
                    train_set.index.min(), train_set.index.max(),
                    test_set.index.min(), test_set.index.max()))

        # featurize
        train_set = featurize(train_set)
        test_set = test_set.drop(columns=["load_kw"])
        test_set = featurize(test_set)
```

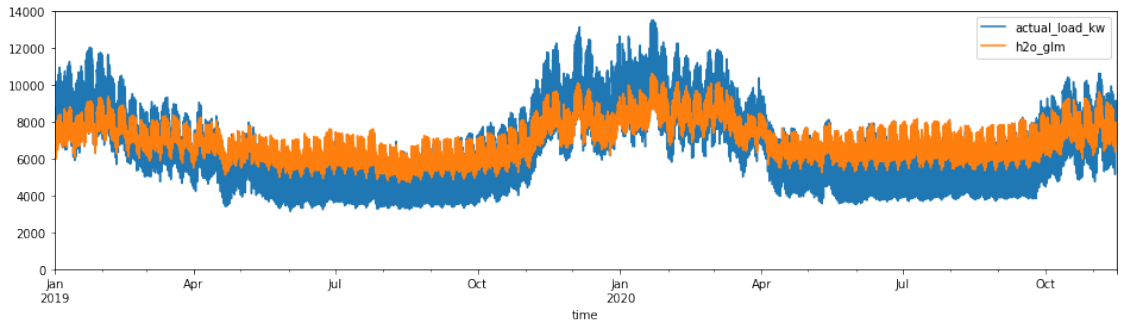
```
        # forecast porfolio for the test_set
        pf_train_start = TimezoneUtils.add_interval_to_day_dt(
            day_dt=test_set.index.min(),
            interval=-portfolio_train_length,
        )
        pf_train = portfolio[(portfolio.index >= pf_train_start) & (portfolio.
↪index < test_set.index.min())]


        forecast_portfolio = enda.Contracts.forecast_portfolio_linear(
            portfolio_df=pf_train,
            start_forecast_date=test_set.index.min(),
            end_forecast_date_exclusive=test_set.index.
↪max()+relativedelta(minutes=30),
            freq='30min',
            tzinfo='Europe/Paris'
        )  # recent portfolio trend

        test_set['kva'] = forecast_portfolio['kva']
        test_set['contracts_count'] = forecast_portfolio['contracts_count']

        # train and predict
        model.train(train_set, target_col='load_kw')
        model_predictions.append(model.predict(test_set, target_col='load_kw'))

    benchmark[model_name] = pd.concat(model_predictions)
```

```
Model sklearn_lin_reg, backtesting iteration 1, train set 2015-01-01
00:00:00+01:00->2018-12-17 23:30:00+01:00, test set 2019-01-01
00:00:00+01:00->2019-01-28 23:30:00+01:00

Model sklearn_lin_reg, backtesting iteration 2, train set 2015-01-01
00:00:00+01:00->2019-01-14 23:30:00+01:00, test set 2019-01-29
00:00:00+01:00->2019-02-25 23:30:00+01:00

Model sklearn_lin_reg, backtesting iteration 10, train set 2015-01-01
00:00:00+01:00->2019-08-26 23:30:00+02:00, test set 2019-09-10
00:00:00+02:00->2019-10-07 23:30:00+02:00

Model sklearn_lin_reg, backtesting iteration 20, train set 2015-01-01
00:00:00+01:00->2020-06-01 23:30:00+02:00, test set 2020-06-16
00:00:00+02:00->2020-07-13 23:30:00+02:00

Model h2o_glm, backtesting iteration 1, train set 2015-01-01
00:00:00+01:00->2018-12-17 23:30:00+01:00, test set 2019-01-01
00:00:00+01:00->2019-01-28 23:30:00+01:00
```

```
Model h2o_glm, backtesting iteration 2, train set 2015-01-01
00:00:00+01:00->2019-01-14 23:30:00+01:00, test set 2019-01-29
00:00:00+01:00->2019-02-25 23:30:00+01:00

Model h2o_glm, backtesting iteration 10, train set 2015-01-01
00:00:00+01:00->2019-08-26 23:30:00+02:00, test set 2019-09-10
00:00:00+02:00->2019-10-07 23:30:00+02:00

Model h2o_glm, backtesting iteration 20, train set 2015-01-01
00:00:00+01:00->2020-06-01 23:30:00+02:00, test set 2020-06-16
00:00:00+02:00->2020-07-13 23:30:00+02:00
```

[47]:
```
# visualize predictions
for c in benchmark.columns:
    if c != "actual_load_kw":
        to_plot = benchmark[["actual_load_kw", c]]
        to_plot.plot(ylim=0, figsize=(16, 4))
```





[48]:
```
# compute mean absolute percentage error
scoring = Scoring(predictions_df=benchmark, target="actual_load_kw")
scoring.mean_absolute_percentage_error().to_frame("mape")
```

[48]:
```
                      mape
sklearn_lin_reg  13.376039
h2o_glm          14.365445
```

If you have time/computing power: - try more algorithms in the backtesting benchmark, this is longer but more reliable than a simple benchmark (think of it as crossval versus single eval in a non-time-series setup). - reduce the "days_in_each_iteration" down to 7 if you think you can have a weekly training in your production environment.

1.7 7. Make the prediction

Seeing the results from just the basic benchmark, we here decide to predict using h2o's gbm (and our set of hyperparameters). We now need to train it on the full dataset and make the prediction.

In the input data, the TSO forecast is only available for the next 7 days but the weather forecast is available for the next 11 days.

We use **EndaEstimatorWithFallback** to be able to predict with or without TSO data.

Checkout more EndaEstimators here: https://github.com/enercoop/enda/blob/main/enda/estimators.py . They work on top of all supported machine learning backends.

[49]:
```
from enda.estimators import EndaEstimatorWithFallback
```

[50]:
```
# create the forecast_input_data dataframe

# we will forecast the portfolio for the next 11 days
forecast_portfolio =  enda.Contracts.forecast_portfolio_linear(
    portfolio_df=portfolio[portfolio.index >= '2020-11-01 00:00:00+01:00'],
    start_forecast_date=pd.to_datetime("2020-12-01 00:00:00+01:00").
↪tz_convert("Europe/Paris"),
    end_forecast_date_exclusive=pd.to_datetime("2020-12-12 00:00:00+01:00").
↪tz_convert("Europe/Paris"),
    freq='30min',
    tzinfo='Europe/Paris'
)

# this time we don't remove rows where tso_forecast is missing
forecast_input_data = pd.merge(
    forecast_portfolio,
    weather_and_tso_forecasts,
    how='inner', left_index=True, right_index=True
)
# add feature engineering
forecast_input_data = featurize(forecast_input_data)
forecast_input_data
```

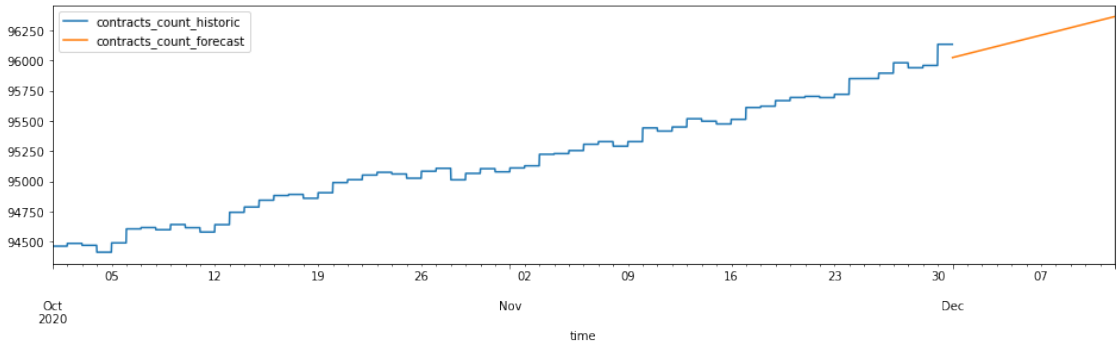
[50]:
```
                           contracts_count           kva  \
time
```

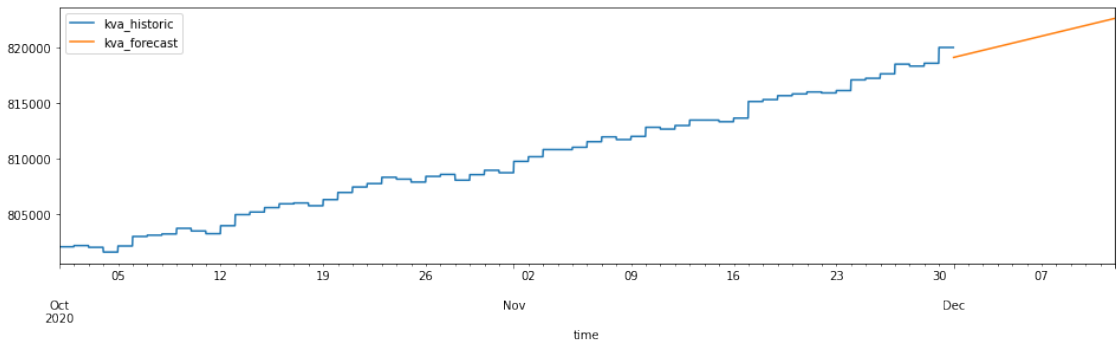
```
2020-12-01 00:00:00+01:00     96024.408293  819111.520044
2020-12-01 00:30:00+01:00     96025.051099  819118.169514
2020-12-01 01:00:00+01:00     96025.693905  819124.818985
2020-12-01 01:30:00+01:00     96026.336710  819131.468455
2020-12-01 02:00:00+01:00     96026.979516  819138.117925
…                                      …           …
2020-12-11 21:30:00+01:00     96360.595760  822589.193030
2020-12-11 22:00:00+01:00     96361.238566  822595.842500
2020-12-11 22:30:00+01:00     96361.881372  822602.491971
2020-12-11 23:00:00+01:00     96362.524178  822609.141441
2020-12-11 23:30:00+01:00     96363.166984  822615.790911

                           tso_forecast_load_mw  t_weighted  t_smooth  \
time
2020-12-01 00:00:00+01:00                66100.0        4.69      5.08
2020-12-01 00:30:00+01:00                64200.0        4.82      5.10
2020-12-01 01:00:00+01:00                61900.0        4.96      5.12
2020-12-01 01:30:00+01:00                62800.0        5.04      5.13
2020-12-01 02:00:00+01:00                62300.0        5.13      5.14
…                                            …          …         …
2020-12-11 21:30:00+01:00                   NaN        8.25      6.03
2020-12-11 22:00:00+01:00                   NaN        8.22      5.94
2020-12-11 22:30:00+01:00                   NaN        8.16      5.83
2020-12-11 23:00:00+01:00                   NaN        8.11      5.78
2020-12-11 23:30:00+01:00                   NaN        8.11      5.73

                           minuteofday  dayofweek  month  minuteofday_cos  \
time
2020-12-01 00:00:00+01:00            0          1     12         1.000000
2020-12-01 00:30:00+01:00           30          1     12         0.991445
2020-12-01 01:00:00+01:00           60          1     12         0.965926
2020-12-01 01:30:00+01:00           90          1     12         0.923880
2020-12-01 02:00:00+01:00          120          1     12         0.866025
…                                    …          …     …              …
2020-12-11 21:30:00+01:00         1290          4     12         0.793353
2020-12-11 22:00:00+01:00         1320          4     12         0.866025
2020-12-11 22:30:00+01:00         1350          4     12         0.923880
2020-12-11 23:00:00+01:00         1380          4     12         0.965926
2020-12-11 23:30:00+01:00         1410          4     12         0.991445

                           minuteofday_sin  dayofweek_cos  dayofweek_sin  \
time
2020-12-01 00:00:00+01:00         0.000000       0.623490       0.781831
2020-12-01 00:30:00+01:00         0.130526       0.623490       0.781831
2020-12-01 01:00:00+01:00         0.258819       0.623490       0.781831
2020-12-01 01:30:00+01:00         0.382683       0.623490       0.781831
2020-12-01 02:00:00+01:00         0.500000       0.623490       0.781831
```

```
…                                    …           …           …
2020-12-11 21:30:00+01:00       -0.608761   -0.900969   -0.433884
2020-12-11 22:00:00+01:00       -0.500000   -0.900969   -0.433884
2020-12-11 22:30:00+01:00       -0.382683   -0.900969   -0.433884
2020-12-11 23:00:00+01:00       -0.258819   -0.900969   -0.433884
2020-12-11 23:30:00+01:00       -0.130526   -0.900969   -0.433884

                           dayofyear_cos  dayofyear_sin  lockdown  \
time
2020-12-01 00:00:00+01:00       0.861702      -0.507415       0.0
2020-12-01 00:30:00+01:00       0.861702      -0.507415       0.0
2020-12-01 01:00:00+01:00       0.861702      -0.507415       0.0
2020-12-01 01:30:00+01:00       0.861702      -0.507415       0.0
2020-12-01 02:00:00+01:00       0.861702      -0.507415       0.0
…                                  …           …       …
2020-12-11 21:30:00+01:00       0.935717      -0.352752       0.0
2020-12-11 22:00:00+01:00       0.935717      -0.352752       0.0
2020-12-11 22:30:00+01:00       0.935717      -0.352752       0.0
2020-12-11 23:00:00+01:00       0.935717      -0.352752       0.0
2020-12-11 23:30:00+01:00       0.935717      -0.352752       0.0

                           public_holiday  nb_school_areas_off  \
time
2020-12-01 00:00:00+01:00             0.0                  0.0
2020-12-01 00:30:00+01:00             0.0                  0.0
2020-12-01 01:00:00+01:00             0.0                  0.0
2020-12-01 01:30:00+01:00             0.0                  0.0
2020-12-01 02:00:00+01:00             0.0                  0.0
…                                     …                    …
2020-12-11 21:30:00+01:00             0.0                  0.0
2020-12-11 22:00:00+01:00             0.0                  0.0
2020-12-11 22:30:00+01:00             0.0                  0.0
2020-12-11 23:00:00+01:00             0.0                  0.0
2020-12-11 23:30:00+01:00             0.0                  0.0

                           extra_long_weekend
time
2020-12-01 00:00:00+01:00                 0.0
2020-12-01 00:30:00+01:00                 0.0
2020-12-01 01:00:00+01:00                 0.0
2020-12-01 01:30:00+01:00                 0.0
2020-12-01 02:00:00+01:00                 0.0
…                                         …
2020-12-11 21:30:00+01:00                 0.0
2020-12-11 22:00:00+01:00                 0.0
2020-12-11 22:30:00+01:00                 0.0
2020-12-11 23:00:00+01:00                 0.0
```

```
2020-12-11 23:30:00+01:00                  0.0

[528 rows x 18 columns]
```

[51]:
```
# show recent portfolio and forecast
for c in ["contracts_count", "kva"]:
    to_plot = pd.merge(
        portfolio[(portfolio.index >= '2020-10-01')][c].to_frame(c+"_historic"),
        forecast_input_data[c].to_frame(c+"_forecast"),
        how='outer', left_index=True, right_index=True
    )

    to_plot.plot(figsize=(16, 4))
```





[52]:
```
# tso data is missing after 2020-12-07 :
forecast_input_data[forecast_input_data.index>='2020-12-07 23:00:00+01:00'].
 ↪head()
```

[52]:
```
                           contracts_count            kva  \
time
2020-12-07 23:00:00+01:00     96239.105452  821332.443136
```

```
2020-12-07 23:30:00+01:00    96239.748258  821339.092607
2020-12-08 00:00:00+01:00    96240.391064  821345.742077
2020-12-08 00:30:00+01:00    96241.033869  821352.391547
2020-12-08 01:00:00+01:00    96241.676675  821359.041018

                         tso_forecast_load_mw  t_weighted  t_smooth  \
time
2020-12-07 23:00:00+01:00               70200.0        3.94      4.07
2020-12-07 23:30:00+01:00               69600.0        3.94      4.07
2020-12-08 00:00:00+01:00                   NaN        3.95      4.07
2020-12-08 00:30:00+01:00                   NaN        3.88      4.06
2020-12-08 01:00:00+01:00                   NaN        3.81      4.05

                         minuteofday  dayofweek  month  minuteofday_cos  \
time
2020-12-07 23:00:00+01:00        1380          0     12         0.965926
2020-12-07 23:30:00+01:00        1410          0     12         0.991445
2020-12-08 00:00:00+01:00           0          1     12         1.000000
2020-12-08 00:30:00+01:00          30          1     12         0.991445
2020-12-08 01:00:00+01:00          60          1     12         0.965926

                         minuteofday_sin  dayofweek_cos  dayofweek_sin  \
time
2020-12-07 23:00:00+01:00       -0.258819        1.00000       0.000000
2020-12-07 23:30:00+01:00       -0.130526        1.00000       0.000000
2020-12-08 00:00:00+01:00        0.000000        0.62349       0.781831
2020-12-08 00:30:00+01:00        0.130526        0.62349       0.781831
2020-12-08 01:00:00+01:00        0.258819        0.62349       0.781831

                         dayofyear_cos  dayofyear_sin  lockdown  \
time
2020-12-07 23:00:00+01:00      0.909308      -0.416125       0.0
2020-12-07 23:30:00+01:00      0.909308      -0.416125       0.0
2020-12-08 00:00:00+01:00      0.916317      -0.400454       0.0
2020-12-08 00:30:00+01:00      0.916317      -0.400454       0.0
2020-12-08 01:00:00+01:00      0.916317      -0.400454       0.0

                         public_holiday  nb_school_areas_off  \
time
2020-12-07 23:00:00+01:00            0.0                  0.0
2020-12-07 23:30:00+01:00            0.0                  0.0
2020-12-08 00:00:00+01:00            0.0                  0.0
2020-12-08 00:30:00+01:00            0.0                  0.0
2020-12-08 01:00:00+01:00            0.0                  0.0

                         extra_long_weekend
time
```

```
2020-12-07 23:00:00+01:00                0.0
2020-12-07 23:30:00+01:00                0.0
2020-12-08 00:00:00+01:00                0.0
2020-12-08 00:30:00+01:00                0.0
2020-12-08 01:00:00+01:00                0.0
```

[53]:
```
gbm_1 = EndaH2OEstimator(H2OGradientBoostingEstimator(
    ntrees=500,
    max_depth=5,
    sample_rate=0.5,
    min_rows=5
))

gbm_2 = EndaH2OEstimator(H2OGradientBoostingEstimator(
    ntrees=500,
    max_depth=5,
    sample_rate=0.5,
    min_rows=5
))

m = EndaEstimatorWithFallback(
    resilient_column="tso_forecast_load_mw",
    estimator_with=gbm_1,
    estimator_without=gbm_2
)
```

[54]:
```
m.train(full_train_set, target_col='load_kw')
```

[55]:
```
import joblib
model_file_path = os.path.join(DIR, "gbm_with_fallback.pickle")
```

[56]:
```
# save the model for later
joblib.dump(m, filename=model_file_path)
```

[56]:
```
['/Users/emmanuel.charon/Documents/CodeProjects/enercoop/enda/data/example_b/gbm
_with_fallback.pickle']
```

[57]:
```
del m
```

[58]:
```
# load the model from disk (works even if you shutdown then restarted the H2O␣
 ↪server)
m2 = joblib.load(filename=model_file_path)
```

[59]:
```
m_prediction = m2.predict(forecast_input_data, target_col="load_kw")
```

[60]:
```
# a good prediction is made until 2020-12-11
# even where TSO forecast is missing
```

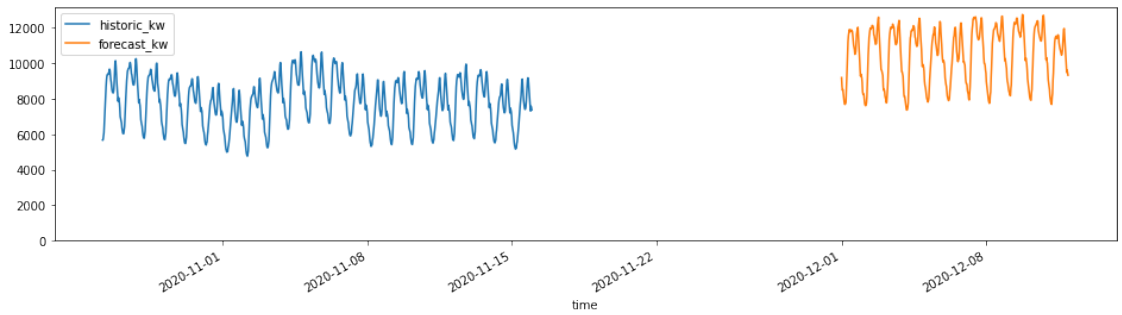
```
m_prediction.tail()
```

[60]:

time	load_kw
2020-12-11 21:30:00+01:00	9828.627576
2020-12-11 22:00:00+01:00	9495.561816
2020-12-11 22:30:00+01:00	9612.761129
2020-12-11 23:00:00+01:00	9455.096895
2020-12-11 23:30:00+01:00	9335.398414

[61]:
```
# visualize recent load along with our forecast; remember we don't have recent␣
↪actual load so there is a time-gap.
# (remember that the prediction takes weather forecast and more information␣
↪into account)
to_plot = pd.merge(
    historic["load_kw"][-1000:].to_frame("historic_kw"),
    m_prediction.rename(columns={"load_kw": "forecast_kw"}),
    how='outer', left_index=True, right_index=True
)
to_plot.plot(ylim=0, figsize=(16, 4))
```

[61]: <AxesSubplot:xlabel='time'>



[62]:
```
# don't forget to shutdown your h2o local server
h2o.cluster().shutdown()
# wait for h2o to really finish shutting down
time.sleep(5)
```

```
H2O session _sid_b8ce closed.
```

1.8 Conclusion

Thats all for Example B. Check out Example C next. Thanks for reading and don't hesitate to send feeback at: emmanuel.charon@enercoop.org !