ExampleB

March 27, 2024

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.

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]:
```python
import enda
import pandas as pd
import os
```

[2]:
```python
enda.__file__
```

[2]:
```
'/Users/clement.jeannesson/Jobs/enda/enda/__init__.py'
```

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

[3]:
```python
# 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 = '.'
```

[4]:
```python
# 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.Resample.upsample_and_interpolate(
        portfolio_by_day,
        freq='30min',
        tz_info='Europe/Paris',
        forward_fill=True
    )

    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.TimezoneUtils.
 ↪convert_dtype_from_object_to_tz_aware(df['time'], tz_info = '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
```

[5]:
```
contracts, portfolio, historic_load_measured, weather_and_tso_forecasts =␣
 ↪read_data()
```

[6]:
```
contracts
```

```
[6]:        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]
```

```
[7]: portfolio
```

```
[7]:                            contracts_count       kva
     date
     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]
```

```
[8]: historic_load_measured
```

```
[8]:                                load_kw
     time
     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]
```

[9]:
```
# 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"])
```

[9]:
```
                           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
…                                           …          …         …
2020-12-07 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]
```

[10]:
```
# 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
)
```

[11]:
```
historic
```

[11]:
```
                           contracts_count          kva      load_kw  \
2015-01-01 00:00:00+01:00     21261.000000  167416.4000  2490.925806
2015-01-01 00:30:00+01:00     21261.020833  167417.4000  2412.623113
2015-01-01 01:00:00+01:00     21261.041667  167418.4000  2365.611276
```

```
2015-01-01 01:30:00+01:00     21261.062500  167419.4000  2336.141065
2015-01-01 02:00:00+01:00     21261.083333  167420.4000  2300.935642
…                                      …            …           …
2020-11-15 21:30:00+01:00     95509.041667  813616.3625  7657.293444
2020-11-15 22:00:00+01:00     95509.833333  813623.0500  7317.540759
2020-11-15 22:30:00+01:00     95510.625000  813629.7375  7580.051439
2020-11-15 23:00:00+01:00     95511.416667  813636.4250  7496.273993
2020-11-15 23:30:00+01:00     95512.208333  813643.1125  7376.005701

                           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
…                                           …          …       …
2020-11-15 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]
```

[12]:
```
# check that there is no NaN value
historic.isna().sum()
```

[12]:
```
contracts_count         0
kva                     0
load_kw                 0
tso_forecast_load_mw    0
t_weighted              0
t_smooth                0
dtype: int64
```

[13]:
```
# note that the type of the index is precise
historic.index.dtype, type(historic.index)
```

[13]:
```
(datetime64[ns, Europe/Paris], pandas.core.indexes.datetimes.DatetimeIndex)
```

[14]:
```
# check missing data in the timeseries (based on the time index only)
missing_periods= enda.TimeSeries.find_missing_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_excl_end_datetime = pd.to_datetime('2020-12-01 00:00:00+01:00').
  ↪astimezone('Europe/Paris'),
)
for missing_period in missing_periods:
    print("Missing data from {} to {}.".format(missing_period[0],␣
  ↪missing_period[1]))

duplicates, extra_points = enda.TimeSeries.find_duplicates_and_extra_points(
    dti=historic.index,
    expected_freq = '30min'
)

if len(duplicates) > 0 :
    print("Extra points found: {}".format(duplicates))

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 expect some missing data.

[15]:
```
# 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')]
```

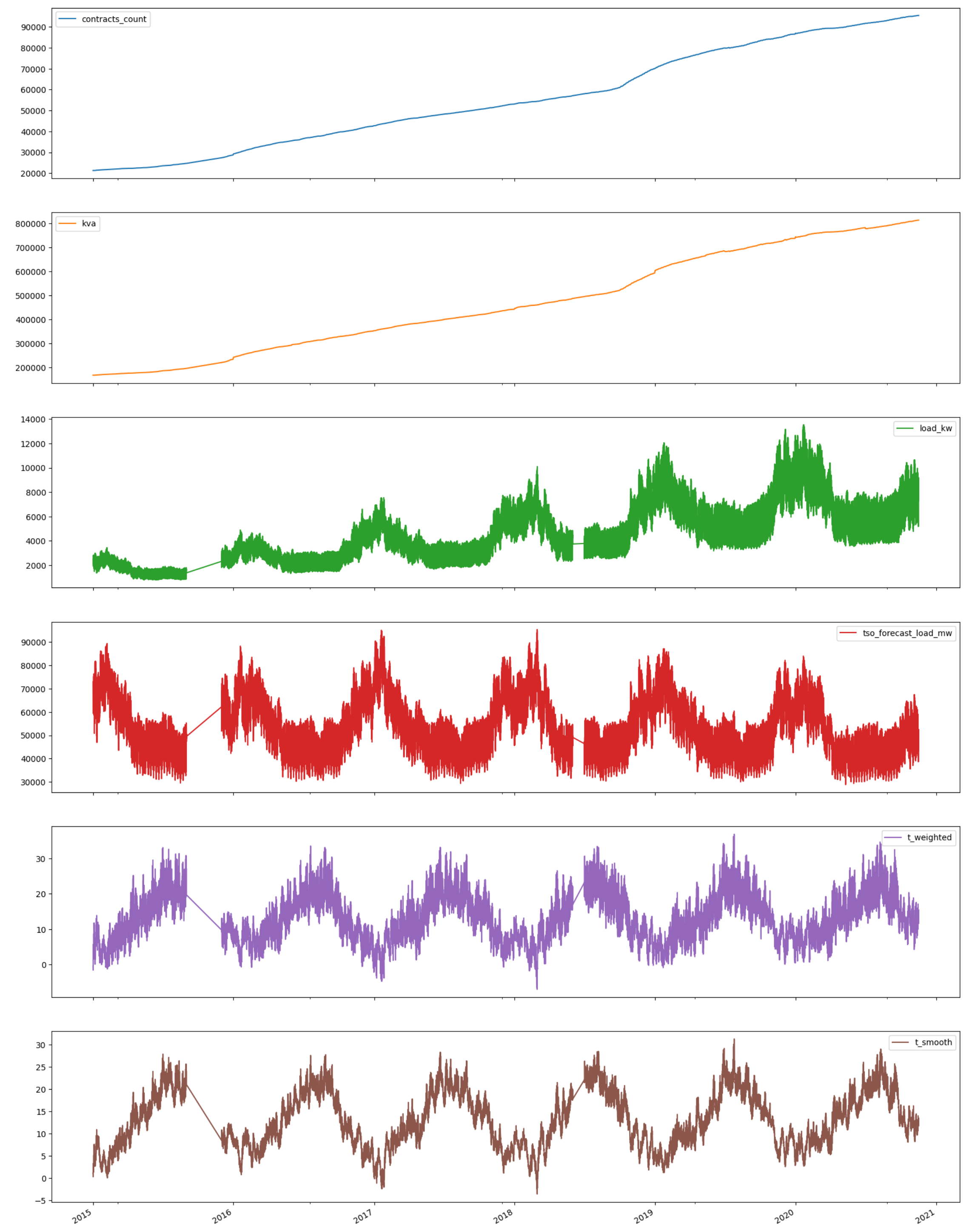
[15]:
```
                           contracts_count        kva      load_kw  \
2019-10-27 01:00:00+02:00         84133.24  716839.24  5179.955556
2019-10-27 01:30:00+02:00         84134.36  716850.66  5087.111111
2019-10-27 02:00:00+02:00         84135.48  716862.08  4898.400000
2019-10-27 02:30:00+02:00         84136.60  716873.50  4616.533333
2019-10-27 02:00:00+01:00         84137.72  716884.92  4259.822222
2019-10-27 02:30:00+01:00         84138.84  716896.34  4208.888889
2019-10-27 03:00:00+01:00         84139.96  716907.76  4137.955556

                           tso_forecast_load_mw  t_weighted  t_smooth
2019-10-27 01:00:00+02:00               41300.0       13.65     13.49
2019-10-27 01:30:00+02:00               40700.0       13.52     13.47
2019-10-27 02:00:00+02:00               36700.0       13.40     13.46
2019-10-27 02:30:00+02:00               36700.0       13.26     13.44
2019-10-27 02:00:00+01:00               36700.0       13.12     13.42
2019-10-27 02:30:00+01:00               36700.0       12.91     13.39
2019-10-27 03:00:00+01:00               36700.0       12.70     13.37
```

1.2 2. Visualize data

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

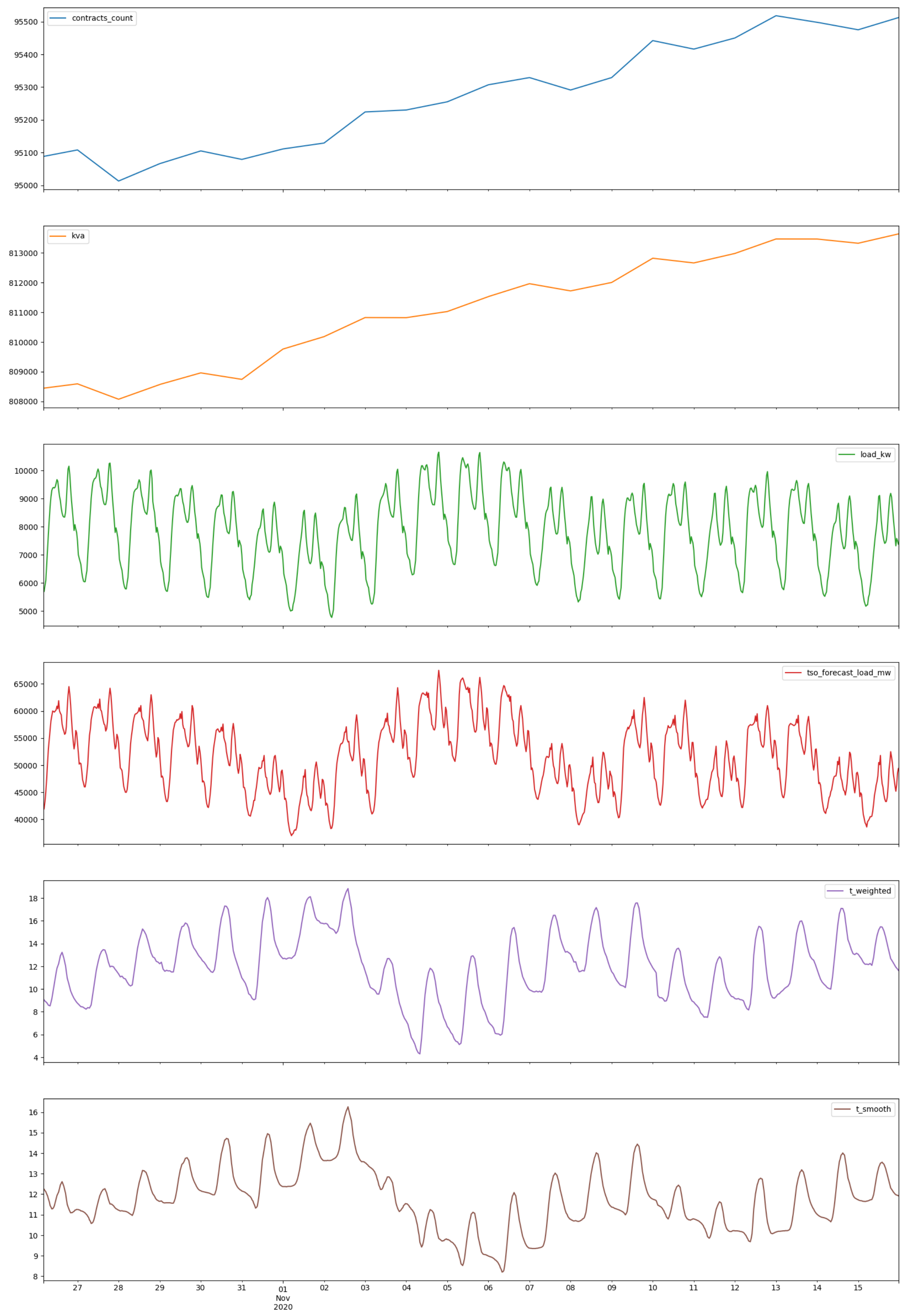
```
[16]: # Show full data set
      historic.plot(figsize=(20, 30), subplots=True)
```

```
[16]: array([<Axes: >, <Axes: >, <Axes: >, <Axes: >, <Axes: >, <Axes: >],
            dtype=object)
```



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

```
[17]: array([<Axes: >, <Axes: >, <Axes: >, <Axes: >, <Axes: >, <Axes: >],
            dtype=object)
```


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

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** :

[18]:
```
def featurize_datetime(df: pd.DataFrame) -> pd.DataFrame:
    """
    Featurize the input datframe with date/datetime-oriented features
    """

    # make a copy
    df = df.copy()

    df = enda.DatetimeFeature.split_datetime(
       df, split_list = ['minuteofday', 'dayofweek']
    )
    df = enda.DatetimeFeature.encode_cyclic_datetime_index(
        df, split_list = ['minuteofday', 'dayofweek', 'dayofyear']
    )

    # min and max years
    min_year = min(df.index).year
    max_year = max(df.index).year

    # add features about national holidays and school holidays (French holidays␣
 ↪here)
    special_days = enda.Calendar.feature_special_days(country='FR', years_list␣
 ↪= [min_year, max_year+1])
    df = pd.merge(
        df,
        special_days,
        how='left',
        left_index=True,
        right_index=True
    )

    return df
```

[19]:
```
# feature the historic dataframe. It makes it a train_set.
full_train_set = featurize_datetime(historic)
```

[20]:
```
full_train_set
```

```
[20]:                            contracts_count          kva       load_kw  \
2015-01-01 00:00:00+01:00     21261.000000  167416.4000  2490.925806
2015-01-01 00:30:00+01:00     21261.020833  167417.4000  2412.623113
2015-01-01 01:00:00+01:00     21261.041667  167418.4000  2365.611276
2015-01-01 01:30:00+01:00     21261.062500  167419.4000  2336.141065
2015-01-01 02:00:00+01:00     21261.083333  167420.4000  2300.935642
…                                        …            …            …
2020-11-15 21:30:00+01:00     95509.041667  813616.3625  7657.293444
2020-11-15 22:00:00+01:00     95509.833333  813623.0500  7317.540759
2020-11-15 22:30:00+01:00     95510.625000  813629.7375  7580.051439
2020-11-15 23:00:00+01:00     95511.416667  813636.4250  7496.273993
2020-11-15 23:30:00+01:00     95512.208333  813643.1125  7376.005701

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

                           minuteofday_sin  dayofweek_cos  dayofweek_sin  \
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  \
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  \
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
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 18 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).

[22]:
```
# 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
```

[22]:
```
                           contracts_count            kva
date
2020-12-01 00:00:00+01:00     96045.999741  819322.586540
2020-12-01 00:30:00+01:00     96046.649342  819329.296753
2020-12-01 01:00:00+01:00     96047.298944  819336.006967
2020-12-01 01:30:00+01:00     96047.948545  819342.717181
2020-12-01 02:00:00+01:00     96048.598146  819349.427394
…                                        …              …
2020-12-07 21:30:00+01:00     96261.017767  821543.667271
2020-12-07 22:00:00+01:00     96261.667368  821550.377484
2020-12-07 22:30:00+01:00     96262.316969  821557.087698
2020-12-07 23:00:00+01:00     96262.966571  821563.797912
2020-12-07 23:30:00+01:00     96263.616172  821570.508125

[336 rows x 2 columns]
```

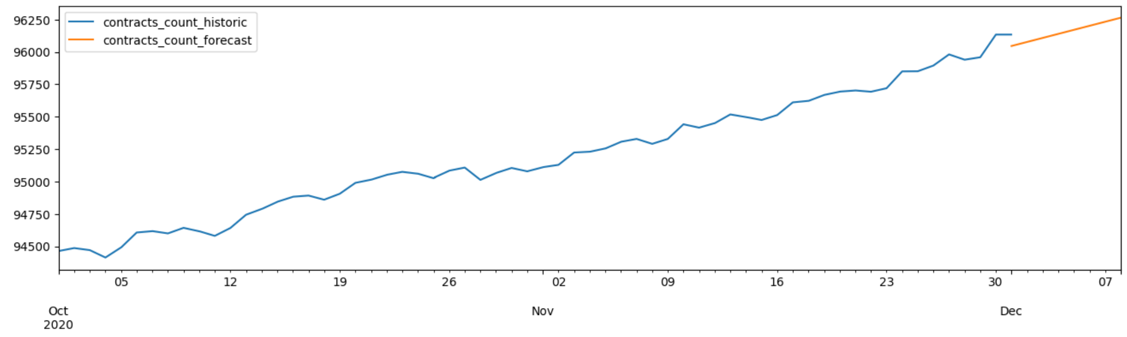
```
[23]: # add weather_and_tso_forecasts
forecast_input_data = pd.merge(
    forecast_portfolio,
    weather_and_tso_forecasts.dropna(subset=["tso_forecast_load_mw"]), #␣
 ↪forecast only where tso is not null for now
    how='inner', left_index=True, right_index=True
)
# add feature engineering
forecast_input_data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

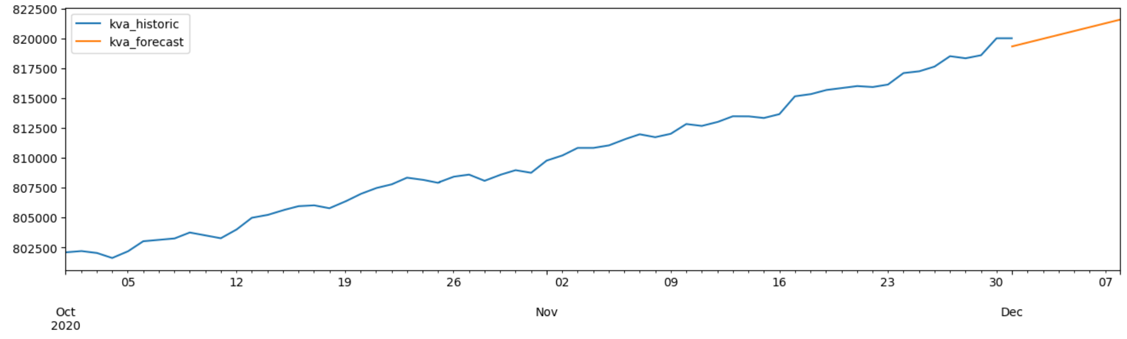
```
[23]:                            contracts_count            kva  \
2020-12-01 00:00:00+01:00     96045.999741  819322.586540
2020-12-01 00:30:00+01:00     96046.649342  819329.296753
2020-12-01 01:00:00+01:00     96047.298944  819336.006967
2020-12-01 01:30:00+01:00     96047.948545  819342.717181
2020-12-01 02:00:00+01:00     96048.598146  819349.427394
…                                      …              …
2020-12-07 21:30:00+01:00     96261.017767  821543.667271
2020-12-07 22:00:00+01:00     96261.667368  821550.377484
2020-12-07 22:30:00+01:00     96262.316969  821557.087698
2020-12-07 23:00:00+01:00     96262.966571  821563.797912
2020-12-07 23:30:00+01:00     96263.616172  821570.508125

                           tso_forecast_load_mw  t_weighted  t_smooth  \
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
…                                           …           …         …
2020-12-07 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

                           minuteofday  dayofweek  minuteofday_cos  \
2020-12-01 00:00:00+01:00            0          1         1.000000
2020-12-01 00:30:00+01:00           30          1         0.991445
2020-12-01 01:00:00+01:00           60          1         0.965926
2020-12-01 01:30:00+01:00           90          1         0.923880
2020-12-01 02:00:00+01:00          120          1         0.866025
…                                  …          …                …
2020-12-07 21:30:00+01:00         1290          0         0.793353
2020-12-07 22:00:00+01:00         1320          0         0.866025
2020-12-07 22:30:00+01:00         1350          0         0.923880
```

```
2020-12-07 23:00:00+01:00         1380         0         0.965926
2020-12-07 23:30:00+01:00         1410         0         0.991445

                           minuteofday_sin  dayofweek_cos  dayofweek_sin  \
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  \
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  \
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
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 17 columns]
```

[24]:
```
# 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))
```





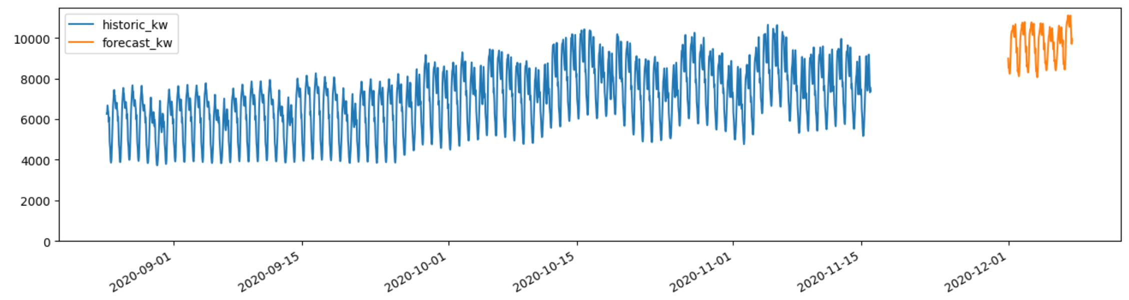
[25]:
```
# do the prediction
lin_reg_prediction = lin_reg.predict(forecast_input_data, target_col="load_kw")
```

[26]:
```
lin_reg_prediction
```

[26]:
```
                               load_kw
2020-12-01 00:00:00+01:00  8990.777945
2020-12-01 00:30:00+01:00  8786.112034
2020-12-01 01:00:00+01:00  8548.345484
2020-12-01 01:30:00+01:00  8630.046223
2020-12-01 02:00:00+01:00  8581.896054
…                                  …
2020-12-07 21:30:00+01:00  9891.252513
2020-12-07 22:00:00+01:00  9709.005148
2020-12-07 22:30:00+01:00  9744.242062
2020-12-07 23:00:00+01:00  9968.246215
2020-12-07 23:30:00+01:00  9882.876502

[336 rows x 1 columns]
```

[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"][-4000:].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]:
```
<Axes: >
```



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:

[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"])
benchmark
```

[28]:
```
                           actual_load_kw
2020-11-01 00:00:00+01:00     6817.332090
2020-11-01 00:30:00+01:00     6326.667322
2020-11-01 01:00:00+01:00     6172.223671
2020-11-01 01:30:00+01:00     6050.575318
2020-11-01 02:00:00+01:00     5898.881230
…                                       …
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

[720 rows x 1 columns]
```

[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']
```

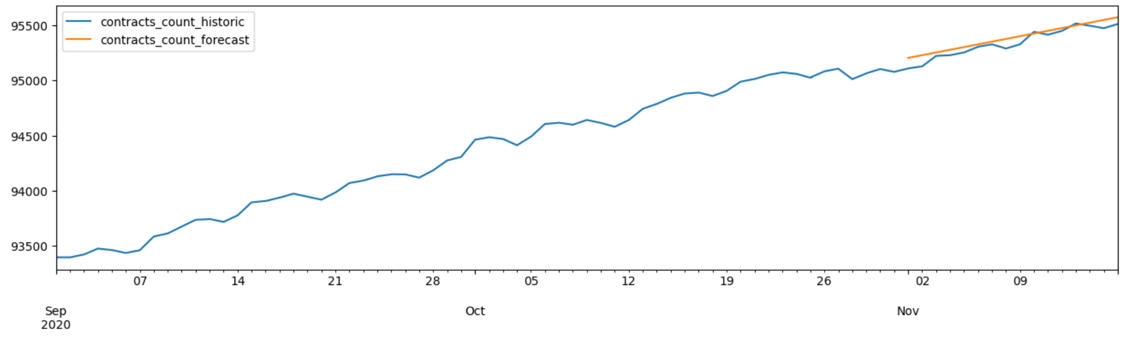
```
benchmark_test
```

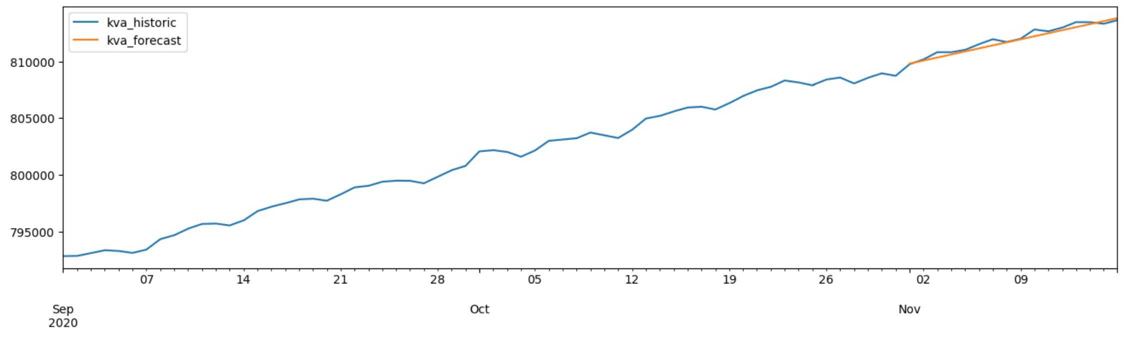
[30]:

```
                           contracts_count            kva  \
2020-11-01 00:00:00+01:00     95205.814480  809817.741508
2020-11-01 00:30:00+01:00     95206.326320  809823.316604
2020-11-01 01:00:00+01:00     95206.838160  809828.891701
2020-11-01 01:30:00+01:00     95207.350000  809834.466797
2020-11-01 02:00:00+01:00     95207.861839  809840.041893
…                                      …             …
2020-11-15 21:30:00+01:00     95571.779928  813803.935204
2020-11-15 22:00:00+01:00     95572.291767  813809.510300
2020-11-15 22:30:00+01:00     95572.803607  813815.085396
2020-11-15 23:00:00+01:00     95573.315447  813820.660492
2020-11-15 23:30:00+01:00     95573.827287  813826.235588

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

                           minuteofday_sin  dayofweek_cos  dayofweek_sin  \
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  \
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  \
2020-11-01 00:00:00+01:00             0.0                  0.0
2020-11-01 00:30:00+01:00             0.0                  0.0
2020-11-01 01:00:00+01:00             0.0                  0.0
2020-11-01 01:30:00+01:00             0.0                  0.0
2020-11-01 02:00:00+01:00             0.0                  0.0
…                                      …                   …
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
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 17 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_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…
not found.
Attempting to start a local H2O server…
  Java Version: openjdk version "21.0.1" 2023-10-17 LTS; OpenJDK Runtime
```

```
Environment Zulu21.30+15-CA (build 21.0.1+12-LTS); OpenJDK 64-Bit Server VM
Zulu21.30+15-CA (build 21.0.1+12-LTS, mixed mode, sharing)
  Starting server from /Users/clement.jeannesson/.pyenv/versions/3.9.10/envs/end
a_1.0.0_dev/lib/python3.9/site-packages/h2o/backend/bin/h2o.jar
  Ice root: /var/folders/pp/kyc80_js50g283hj0_c4yrhc0000gp/T/tmp9w493lg0
  JVM stdout: /var/folders/pp/kyc80_js50g283hj0_c4yrhc0000gp/T/tmp9w493lg0/h2o_c
lement_jeannesson_started_from_python.out
  JVM stderr: /var/folders/pp/kyc80_js50g283hj0_c4yrhc0000gp/T/tmp9w493lg0/h2o_c
lement_jeannesson_started_from_python.err
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321 … successful.

--------------------------  -----------------------------------------
H2O_cluster_uptime:         02 secs
H2O_cluster_timezone:       Europe/Paris
H2O_data_parsing_timezone:  UTC
H2O_cluster_version:        3.46.0.1
H2O_cluster_version_age:    13 days
H2O_cluster_name:           H2O_from_python_clement_jeannesson_xqjnib
H2O_cluster_total_nodes:    1
H2O_cluster_free_memory:    3.984 Gb
H2O_cluster_total_cores:    8
H2O_cluster_allowed_cores:  8
H2O_cluster_status:         locked, healthy
H2O_connection_url:         http://127.0.0.1:54321
H2O_connection_proxy:       {"http": null, "https": null}
H2O_internal_security:      False
Python_version:             3.9.10 final
--------------------------  -----------------------------------------
```

[39]:
```python
benchmark_train.iloc[0:10, 2:5]
```

[39]:
```
                               load_kw  tso_forecast_load_mw  t_weighted
2015-01-01 00:00:00+01:00  2490.925806               72900.0       -0.41
2015-01-01 00:30:00+01:00  2412.623113               71600.0       -0.48
2015-01-01 01:00:00+01:00  2365.611276               69900.0       -0.55
2015-01-01 01:30:00+01:00  2336.141065               70600.0       -0.66
2015-01-01 02:00:00+01:00  2300.935642               70500.0       -0.78
2015-01-01 02:30:00+01:00  2226.613719               69000.0       -0.89
2015-01-01 03:00:00+01:00  2166.173069               67200.0       -1.00
2015-01-01 03:30:00+01:00  2104.404493               65400.0       -1.11
2015-01-01 04:00:00+01:00  2064.678631               63800.0       -1.22
2015-01-01 04:30:00+01:00  2035.268532               62700.0       -1.25
```

[40]:
```python
benchmark_train.iloc[0:10, 2:5].to_csv("training_test.csv")
```

[41]:
```python
# 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))
```

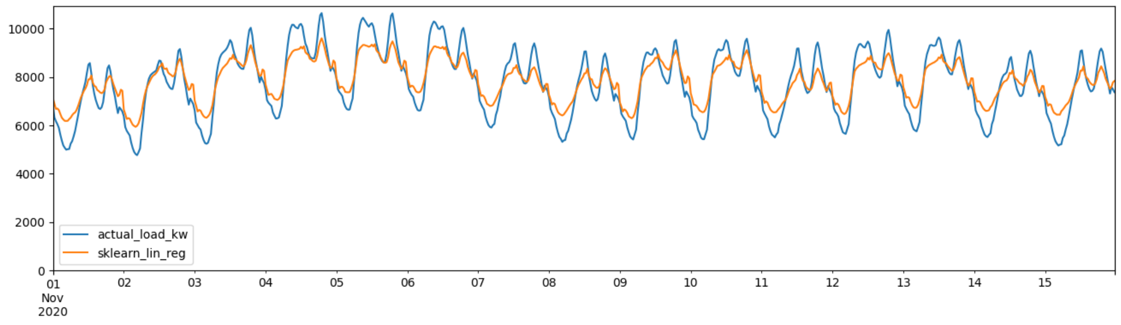
```
Benchmark with 8 models : ['sklearn_lin_reg', 'sklearn_sgd',
'sklearn_ada_boost', 'sklearn_nn', 'h2o_glm', 'h2o_rf', 'h2o_gbm', 'h2o_nn']

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 0.9 seconds.

Training sklearn_ada_boost before predicting with it..
sklearn_ada_boost took 44.4 seconds.

Training sklearn_nn before predicting with it..
sklearn_nn took 91.2 seconds.

Training h2o_glm before predicting with it..

<IPython.core.display.HTML object>

h2o_glm took 3.4 seconds.

Training h2o_rf before predicting with it..
h2o_rf took 15.6 seconds.

Training h2o_gbm before predicting with it..
h2o_gbm took 8.2 seconds.

Training h2o_nn before predicting with it..
h2o_nn took 18.3 seconds.
```

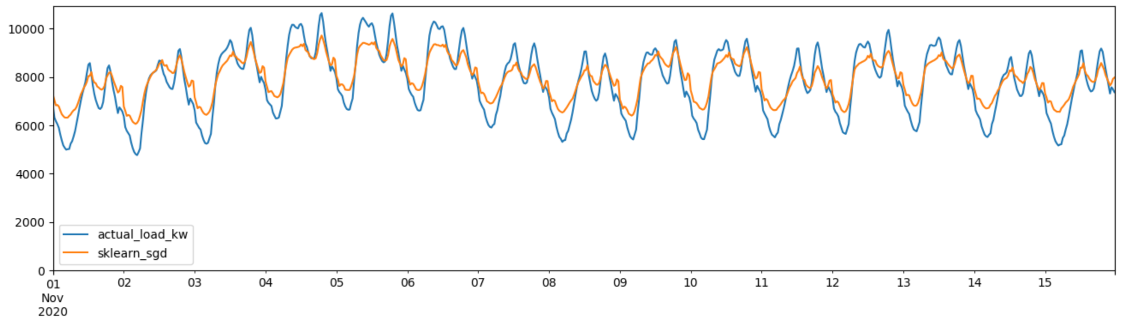
[42]:
```
benchmark
```

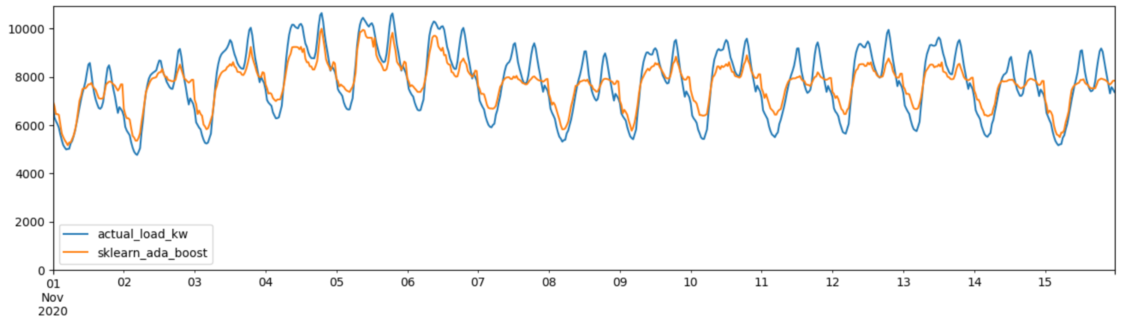
[42]:
```
                           actual_load_kw  sklearn_lin_reg  sklearn_sgd  \
2020-11-01 00:00:00+01:00     6817.332090      7116.262400  7268.504320
2020-11-01 00:30:00+01:00     6326.667322      6896.504843  7046.769174
2020-11-01 01:00:00+01:00     6172.223671      6682.516424  6830.605028
```

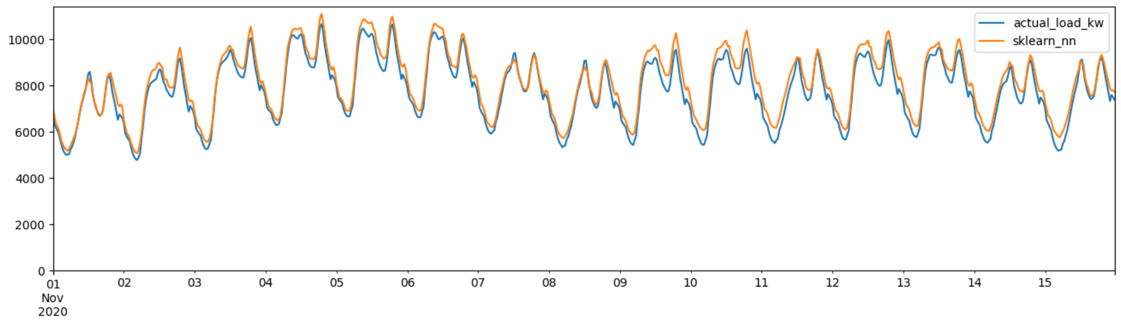
```
2020-11-01 01:30:00+01:00     6050.575318      6699.648917  6847.511206
2020-11-01 02:00:00+01:00     5898.881230      6635.737998  6782.325729
…                                       …            …          …
2020-11-15 21:30:00+01:00     7657.293444      7647.649943  7784.966759
2020-11-15 22:00:00+01:00     7317.540759      7516.196417  7653.352240
2020-11-15 22:30:00+01:00     7580.051439      7599.955734  7738.868558
2020-11-15 23:00:00+01:00     7496.273993      7784.720105  7926.057651
2020-11-15 23:30:00+01:00     7376.005701      7838.739518  7981.093363

                           sklearn_ada_boost   sklearn_nn      h2o_glm  \
2020-11-01 00:00:00+01:00        7004.055238  6978.222468  7411.859182
2020-11-01 00:30:00+01:00        6800.756289  6668.168002  7173.717434
2020-11-01 01:00:00+01:00        6457.625390  6356.734892  6935.575686
2020-11-01 01:30:00+01:00        6462.474690  6243.347042  6958.317705
2020-11-01 02:00:00+01:00        6430.078251  6060.438533  6878.974772
…                                        …          …          …
2020-11-15 21:30:00+01:00        7723.935769  8126.484696  7259.411106
2020-11-15 22:00:00+01:00        7688.075642  7852.186456  7146.039856
2020-11-15 22:30:00+01:00        7751.777186  7770.791269  7282.209600
2020-11-15 23:00:00+01:00        7841.638268  7786.286328  7531.807069
2020-11-15 23:30:00+01:00        7846.176657  7719.989282  7622.605723

                                h2o_rf      h2o_gbm       h2o_nn
2020-11-01 00:00:00+01:00  6618.429014  7238.106976  7134.318405
2020-11-01 00:30:00+01:00  6244.503840  6695.938679  6792.191181
2020-11-01 01:00:00+01:00  6056.103537  6378.550857  6490.857203
2020-11-01 01:30:00+01:00  5971.874814  6277.017389  6340.915400
2020-11-01 02:00:00+01:00  5830.723221  6196.947266  6155.570844
…                                  …          …          …
2020-11-15 21:30:00+01:00  7495.115877  7419.043766  8127.375499
2020-11-15 22:00:00+01:00  7393.640487  7141.754804  7934.052204
2020-11-15 22:30:00+01:00  7428.028846  7417.634841  7856.795770
2020-11-15 23:00:00+01:00  7434.956921  7220.599850  7806.171075
2020-11-15 23:30:00+01:00  7442.448993  7264.606470  7709.001933

[720 rows x 9 columns]
```

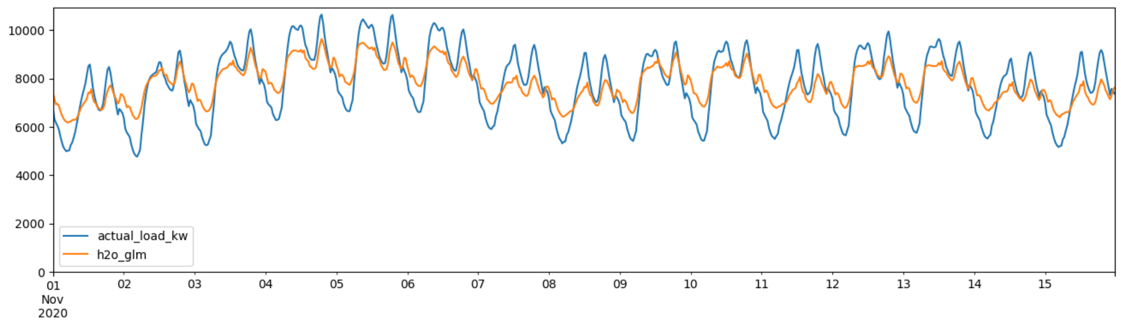
[43]:
```python
# visualize predictions
for c in benchmark.columns:
    if c != "actual_load_kw":
        to_plot = benchmark[["actual_load_kw", c]]
        to_plot.plot(ylim=0, figsize=(16, 4))
```


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

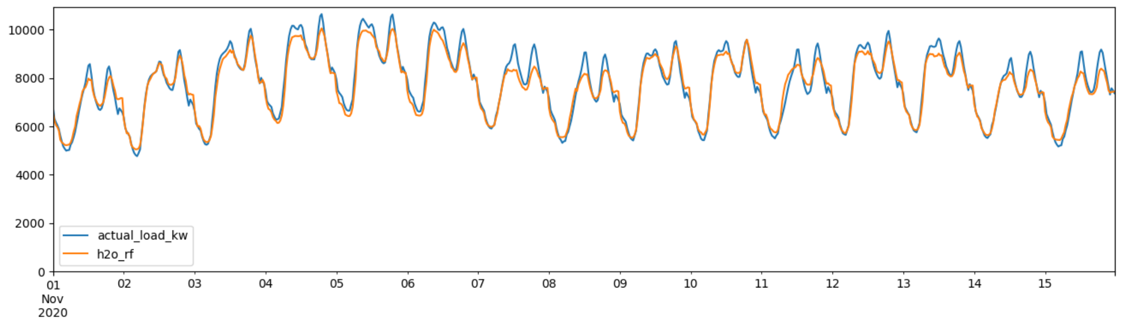

10000
8000
6000
4000
2000
0
actual_load_kw
sklearn_sgd
01
Nov
2020
02
03
04
05
06
07
08
09
10
11
12
13
14
15


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

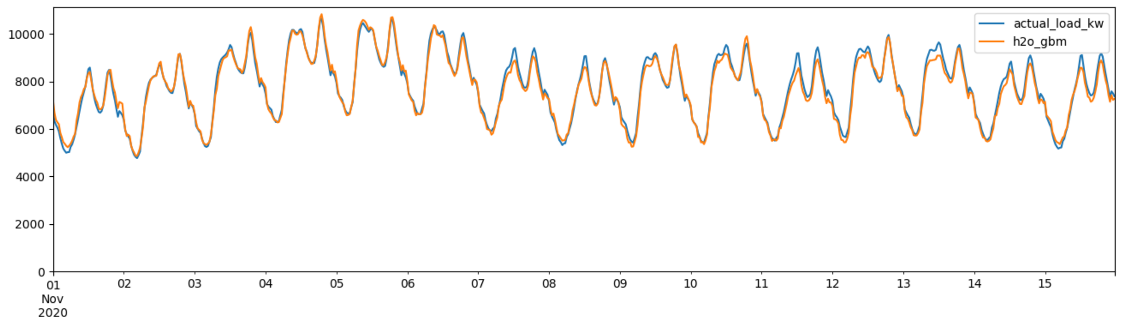

10000
8000
6000
4000
2000
0
actual_load_kw
sklearn_nn
01
Nov
2020
02
03
04
05
06
07
08
09
10
11
12
13
14
15



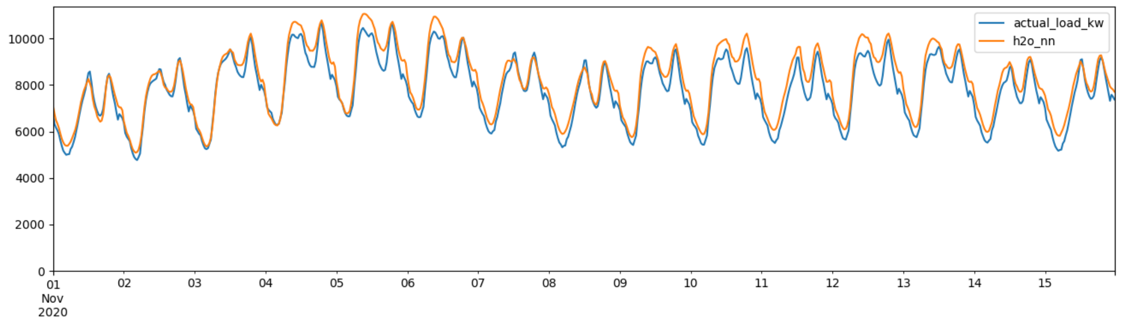
actual_load_kw
h2o_glm
0
2000
4000
6000
8000
10000
01 Nov 2020
02
03
04
05
06
07
08
09
10
11
12
13
14
15



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



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



actual_load_kw
h2o_nn
0
2000
4000
6000
8000
10000
01 Nov 2020
02
03
04
05
06
07
08
09
10
11
12
13
14
15

[44]:
```
# compute the mean absolute percentage error of each algo
scoring = enda.Scoring(predictions_df=benchmark, target="actual_load_kw")
scoring.mean_absolute_percentage_error().to_frame("mape")
```

[44]:
```
                       mape
sklearn_lin_reg    6.972668
sklearn_sgd        7.448227
sklearn_ada_boost  6.687777
sklearn_nn         4.821875
h2o_glm            9.056221
h2o_rf             2.832520
h2o_gbm            2.155135
h2o_nn             5.559592
```

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.

[45]:
```
all_models = dict()

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

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

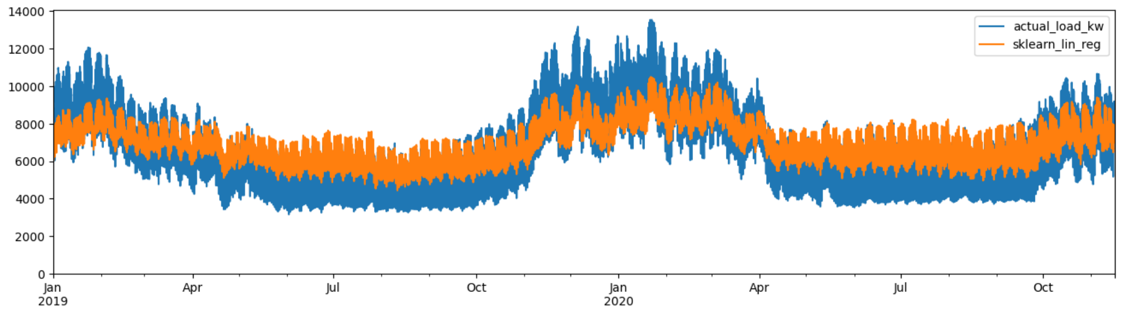
[46]:
```
from dateutil.relativedelta import relativedelta
portfolio_train_length = relativedelta(months=1)
```

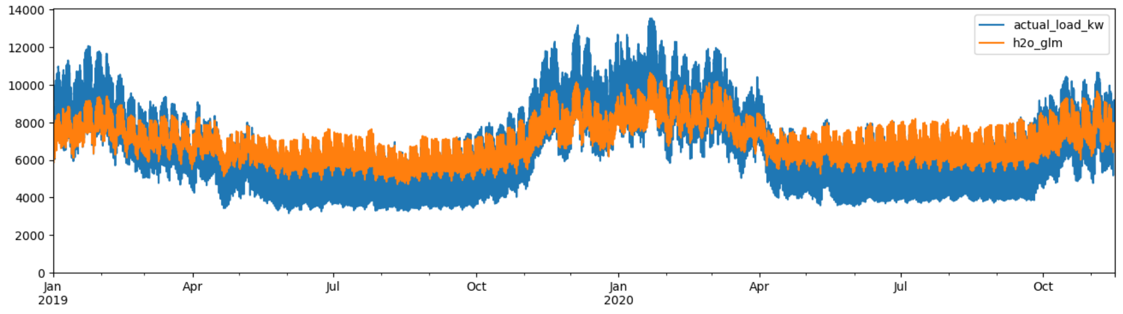
```
[47]: start_backtesting_dt = pd.to_datetime('2019-01-01 00:00:00+01:00').
       ↪tz_convert('Europe/Paris')
      benchmark = historic[historic.index>=start_backtesting_dt]["load_kw"].
       ↪to_frame("actual_load_kw")
      days_in_each_iteration = 28


      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
              test_set = test_set.drop(columns=["load_kw"])

              # forecast porfolio for the test_set
              pf_train_start = enda.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
```

[48]:
```
# 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))
```





[49]:
```python
# compute mean absolute percentage error
scoring = enda.Scoring(predictions_df=benchmark, target="actual_load_kw")
scoring.mean_absolute_percentage_error().to_frame("mape")
```

[49]:
```
                      mape
sklearn_lin_reg  13.830003
h2o_glm          14.362408
```

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.

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

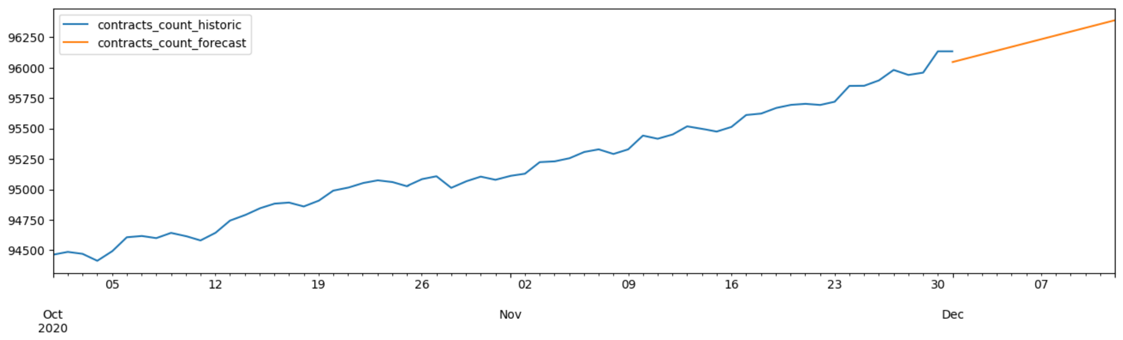
```
[51]: # create the forecast_input_data dataframe

      # 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_datetime(forecast_input_data)
      forecast_input_data
```

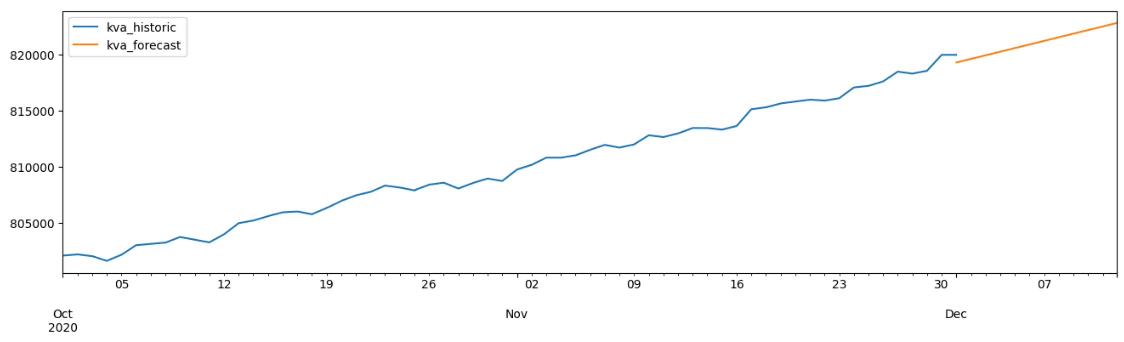
```
[51]:                            contracts_count            kva  \
      2020-12-01 00:00:00+01:00     96046.000857  819322.806873
      2020-12-01 00:30:00+01:00     96046.650461  819329.517545
      2020-12-01 01:00:00+01:00     96047.300064  819336.228217
      2020-12-01 01:30:00+01:00     96047.949668  819342.938889
      2020-12-01 02:00:00+01:00     96048.599272  819349.649561
      …                                      …              …
      2020-12-11 21:30:00+01:00     96385.743543  822832.488295
      2020-12-11 22:00:00+01:00     96386.393147  822839.198967
      2020-12-11 22:30:00+01:00     96387.042751  822845.909639
      2020-12-11 23:00:00+01:00     96387.692354  822852.620311
      2020-12-11 23:30:00+01:00     96388.341958  822859.330983

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

                           minuteofday_sin  dayofweek_cos  dayofweek_sin  \
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  \
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  \
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
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 17 columns]
```

[52]:
```
# 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))
```





[53]:
```
# tso data is missing after 2020-12-07 :
forecast_input_data[forecast_input_data.index>='2020-12-07 23:00:00+01:00'].
 ↪head()
```

[53]:
```
                           contracts_count            kva  \
2020-12-07 23:00:00+01:00     96262.968462  821564.171299
2020-12-07 23:30:00+01:00     96263.618065  821570.881971
2020-12-08 00:00:00+01:00     96264.267669  821577.592643
2020-12-08 00:30:00+01:00     96264.917273  821584.303315
2020-12-08 01:00:00+01:00     96265.566876  821591.013987

                           tso_forecast_load_mw  t_weighted  t_smooth  \
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  minuteofday_cos  \
2020-12-07 23:00:00+01:00         1380          0         0.965926
2020-12-07 23:30:00+01:00         1410          0         0.991445
2020-12-08 00:00:00+01:00            0          1         1.000000
2020-12-08 00:30:00+01:00           30          1         0.991445
2020-12-08 01:00:00+01:00           60          1         0.965926

                           minuteofday_sin  dayofweek_cos  dayofweek_sin  \
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  \
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  \
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
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
```

[54]:
```
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
)
```

[55]:
```
m.train(full_train_set, target_col='load_kw')
```

[56]:
```
import joblib
model_file_path = os.path.join(DIR, "gbm_with_fallback.pickle")
```

[57]:
```
# save the model for later
joblib.dump(m, filename=model_file_path)
```

[57]: ['./gbm_with_fallback.pickle']

[58]:
```
del m
```

[59]:
```
# load the model from disk (works even if you shutdown then restarted the H2O␣
↪server)
m2 = joblib.load(filename=model_file_path)
```

[60]:
```
m_prediction = m2.predict(forecast_input_data, target_col="load_kw")
```

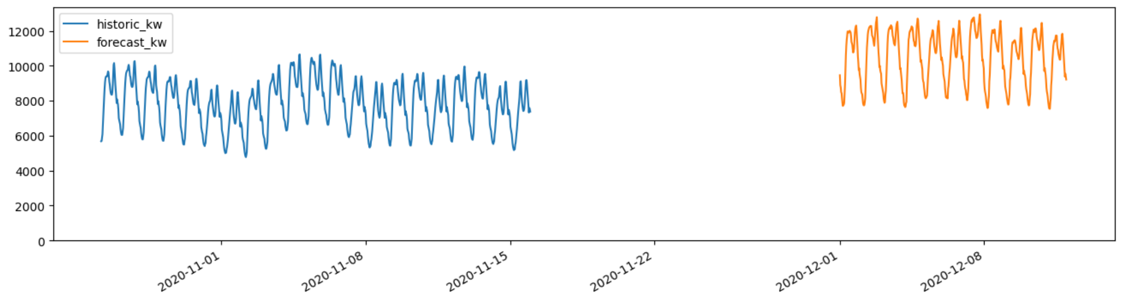
[61]:
```
# a good prediction is made until 2020-12-11
# even where TSO forecast is missing
m_prediction.tail()
```

[61]:
```
                               load_kw
2020-12-11 21:30:00+01:00  9787.209992
2020-12-11 22:00:00+01:00  9375.710344
2020-12-11 22:30:00+01:00  9541.818818
2020-12-11 23:00:00+01:00  9403.112938
2020-12-11 23:30:00+01:00  9208.277543
```

[ ]:
```

```

[62]:
```
# 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))
```

[62]: <Axes: >



[63]:
```
# 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_9c51 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: team-data@enercoop.org !