ExampleA

April 1, 2021

1 Project enda : Example A

In this example notebook, we will show how to read and manipulate contracts data on a small sample. Then we will show how to align it with consumption, weather and TSO forecast data in order to train it and make a load forecast.

To start, you will need a python 3 installation (use a virtual environment), and to install some packages :

```
# create virtualenv, can use for instance {path_to_python3.9} instead of just "python3"
python3 -m venv {path-to-venv}
source {path-to-venv}/bin/activate
which python  # check python path
python --version  # check python version
pip install --upgrade pip  # upgrade pip, the package manager
pip install pandas enda jupyter
pip install numexpr bottleneck  # optional, pandas speed boost
pip install scikit-learn joblib matplotlib # for some steps in this tutorial
which jupyter  # check that the jupyter program you are using is the one in this venv
jupyter notebook  # lauch jupyter
```

Then you can download `example_a.zip` to your local machine. It contains this notebook (`ExampleA.ipynb`) and the dataset (`contracts.csv`, `historic_load_measured.csv`, `weather_and_tso_forecasts.csv`). Open `ExampleA.ipynb` with jupyter and follow the tutorial there instead of the pdf/html. The dataset is a micro-example of the data we typically deal with.

We here pretend **we are exactly on '2020-09-20'** and want to predict our SLP (synthetic load profiles) customers load for the next 3 days, from '2020-09-21' to '2020-09-23' at a 15 min time-step. The desired time-zone is 'Europe/Berlin'. This load may depend on several factors such as the number of customer or the weather. In this example, we have only 3 days of training data, from '2020-09-16' to '2020-09-19'.

Data from '2020-09-20' is not available because we do not have the most recent measured consumption data : there is a time-gap between the latest time for which we have an actual measure and the next time we want to predict. In a more realistic example, this gap may be a few days or weeks.

The files are : - `contracts.csv` : contains a list of 7 electricity customer contracts with different characteristics. - `historic_load_measured.csv` : the past load for 2 groups of customers : `smart_metered` and `slp`, from '2020-09-16' to '2020-09-19'. - `weather_and_tso_forecasts.csv` : 2 external forecasts, the temperature and the total load on our TSO's grid, available in the past and in the future : from '2020-09-16' to '2020-09-23'.

You can now follow this tutorial step by step. It is divided in 3 parts: 1. Deal with contracts data 2. Make a really basic prediction 3. Try it yourself

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

[2]:
```
# replace with the folder path where you put example_a
DIR = '/Users/emmanuel.charon/Documents/CodeProjects/enercoop/enda/data/
↪example_a'
```

1.1 1. Deal with contracts data

[3]:
```
contracts = enda.Contracts.read_contracts_from_file(os.path.join(DIR,␣
↪"contracts.csv"))
```

[4]:
```
contracts
# When date_end_exclusive = NaT, this means the contract is still active today␣
↪and has no planned end date.
# Note that lines 1 and 2 are about the same customer with customer_id=1. They␣
↪changed their subscribed power,
#   so we counted it as a new contract (contract_id=1-a then 1-b).
# Note that some have a start date or an end date in the 'future' (after␣
↪'2020-09-20').
```

[4]:
```
   customer_id contract_id date_start date_end_exclusive  \
0            1         1-a 2020-09-16         2020-09-19
1            1         1-b 2020-09-19                NaT
2            2         2-a 2020-09-17         2020-09-21
3            3         3-a 2020-09-18                NaT
4            4         4-a 2020-09-19                NaT
5            5         5-a 2020-09-18         2020-09-26
6            6         6-a 2020-09-23                NaT

    sub_contract_end_reason  subscribed_power_kva  smart_metered profile  \
0  changed subscribed power                     6          False    RES2
1                       NaN                     9          False    RES2
2              contract end                    15           True     NaN
3                       NaN                     3           True     NaN
4                       NaN                    12          False    PRO1
5              contract end                     9          False    RES2
6                       NaN                     6          False    RES2

   customer_type  specific_price  estimated_annual_consumption_kwh  \
0    residential           False                              4500
1    residential           False                              4500
2  professionnal            True                             20000
```

```
3   residential           False                           3000
4  professionnal          False                          10000
5   residential            True                           5000
6   residential            True                           4000

        tension
0  BT<=36kVA RES
1  BT<=36kVA RES
2  BT<=36kVA PRO
3  BT<=36kVA RES
4  BT<=36kVA PRO
5  BT<=36kVA RES
6  BT<=36kVA RES
```

[5]:
```
# we are only interested in SLP customers here
contracts_slp = contracts[~contracts["smart_metered"]].copy() # drop␣
↪smart-metered contracts
# add a variable to count the number of active contracts
contracts_slp["contracts_count"] = 1
```

[6]:
```
contracts_slp
```

[6]:
```
   customer_id contract_id date_start date_end_exclusive  \
0            1         1-a 2020-09-16         2020-09-19
1            1         1-b 2020-09-19                NaT
4            4         4-a 2020-09-19                NaT
5            5         5-a 2020-09-18         2020-09-26
6            6         6-a 2020-09-23                NaT

    sub_contract_end_reason  subscribed_power_kva  smart_metered profile  \
0  changed subscribed power                     6          False    RES2
1                       NaN                     9          False    RES2
4                       NaN                    12          False    PRO1
5              contract end                     9          False    RES2
6                       NaN                     6          False    RES2

   customer_type  specific_price  estimated_annual_consumption_kwh  \
0   residential           False                           4500
1   residential           False                           4500
4  professionnal          False                          10000
5   residential            True                           5000
6   residential            True                           4000

        tension  contracts_count
0  BT<=36kVA RES                1
1  BT<=36kVA RES                1
4  BT<=36kVA PRO                1
```

```
5  BT<=36kVA RES               1
6  BT<=36kVA RES               1
```

[7]:
```python
# count the running total of ["contracts_count", "subscribed_power_kva",␣
↪"estimated_annual_consumption_kwh"]  each day
portfolio_slp_by_day = enda.Contracts.compute_portfolio_by_day(
    contracts_slp,
    columns_to_sum = ["contracts_count", "subscribed_power_kva",␣
↪"estimated_annual_consumption_kwh"],
    date_start_col="date_start",
    date_end_exclusive_col="date_end_exclusive"
)
```

[8]:
```python
# note that portfolio_by_day can have dates in the future (after 2020-09-20) if␣
↪some contracts have a future date_end
portfolio_slp_by_day
```

[8]:
```
            contracts_count  subscribed_power_kva  \
date
2020-09-16              1.0                   6.0
2020-09-17              1.0                   6.0
2020-09-18              2.0                  15.0
2020-09-19              3.0                  30.0
2020-09-20              3.0                  30.0
2020-09-21              3.0                  30.0
2020-09-22              3.0                  30.0
2020-09-23              4.0                  36.0
2020-09-24              4.0                  36.0
2020-09-25              4.0                  36.0
2020-09-26              3.0                  27.0

            estimated_annual_consumption_kwh
date
2020-09-16                            4500.0
2020-09-17                            4500.0
2020-09-18                            9500.0
2020-09-19                           19500.0
2020-09-20                           19500.0
2020-09-21                           19500.0
2020-09-22                           19500.0
2020-09-23                           23500.0
2020-09-24                           23500.0
2020-09-25                           23500.0
2020-09-26                           18500.0
```

[9]:
```python
# restrict/extend the portfolio_by_day to desired dates
portfolio_slp_by_day = enda.Contracts.get_portfolio_between_dates(
    portfolio_slp_by_day,
    start_datetime = pd.to_datetime('2020-09-16'),
    end_datetime_exclusive = pd.to_datetime('2020-09-24')
)
```

[10]:
```python
portfolio_slp_by_day
```

[10]:
```
            contracts_count  subscribed_power_kva  \
date
2020-09-16              1.0                   6.0
2020-09-17              1.0                   6.0
2020-09-18              2.0                  15.0
2020-09-19              3.0                  30.0
2020-09-20              3.0                  30.0
2020-09-21              3.0                  30.0
2020-09-22              3.0                  30.0
2020-09-23              4.0                  36.0

            estimated_annual_consumption_kwh
date
2020-09-16                            4500.0
2020-09-17                            4500.0
2020-09-18                            9500.0
2020-09-19                           19500.0
2020-09-20                           19500.0
2020-09-21                           19500.0
2020-09-22                           19500.0
2020-09-23                           23500.0
```

[11]:
```python
# turn the portfolio_by_day into a portfolio timeseries with our desired freq␣
↪and timezone
portfolio_slp = enda.TimeSeries.interpolate_daily_to_sub_daily_data(
    portfolio_slp_by_day,
    freq='15min',
    tz='Europe/Berlin'
)
```

[12]:
```python
portfolio_slp
```

[12]:
```
                           contracts_count  subscribed_power_kva  \
time
2020-09-16 00:00:00+02:00              1.0                   6.0
2020-09-16 00:15:00+02:00              1.0                   6.0
2020-09-16 00:30:00+02:00              1.0                   6.0
2020-09-16 00:45:00+02:00              1.0                   6.0
```

```
2020-09-16 01:00:00+02:00              1.0                    6.0
…                                    …                    …
2020-09-23 22:45:00+02:00              4.0                   36.0
2020-09-23 23:00:00+02:00              4.0                   36.0
2020-09-23 23:15:00+02:00              4.0                   36.0
2020-09-23 23:30:00+02:00              4.0                   36.0
2020-09-23 23:45:00+02:00              4.0                   36.0

                           estimated_annual_consumption_kwh
time
2020-09-16 00:00:00+02:00                            4500.0
2020-09-16 00:15:00+02:00                            4500.0
2020-09-16 00:30:00+02:00                            4500.0
2020-09-16 00:45:00+02:00                            4500.0
2020-09-16 01:00:00+02:00                            4500.0
…                                                     …
2020-09-23 22:45:00+02:00                           23500.0
2020-09-23 23:00:00+02:00                           23500.0
2020-09-23 23:15:00+02:00                           23500.0
2020-09-23 23:30:00+02:00                           23500.0
2020-09-23 23:45:00+02:00                           23500.0

[768 rows x 3 columns]
```

1.2 2. Make a really basic prediction

[13]:
```
# read historical load, weather and TSO forecast data
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"))
```

[14]:
```
# 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'])
    # for now df['time'] can be of dtype "object" because there are 2 french␣
 ↪timezones: +60min and +120min.
    # it is important to align time-zone to 'Europe/Berlin' to make sure the df␣
 ↪has a pandas.DatetimeIndex
    df['time'] = enda.TimeSeries.align_timezone(df['time'], tzinfo = 'Europe/
 ↪Berlin')
    df.set_index('time', inplace=True)
```

[15]:
```
historic_load_measured
```

[15]:
```
                           smart_metered_kw  slp_kw
time
2020-09-16 00:00:00+02:00            0.0000  1.5066
2020-09-16 00:15:00+02:00            0.0000  1.4574
2020-09-16 00:30:00+02:00            0.0000  1.4082
2020-09-16 00:45:00+02:00            0.0000  1.3678
2020-09-16 01:00:00+02:00            0.0000  1.3273
…                                     …       …
2020-09-19 22:45:00+02:00            4.1486  9.7404
2020-09-19 23:00:00+02:00            4.0531  9.3414
2020-09-19 23:15:00+02:00            3.9842  8.8738
2020-09-19 23:30:00+02:00            3.9153  8.4063
2020-09-19 23:45:00+02:00            3.8018  8.2067

[384 rows x 2 columns]
```

[16]:
```
weather_and_tso_forecasts
```

[16]:
```
                           tso_forecast_load_mw  t_weighted
time
2020-09-16 00:00:00+02:00               44700.0       20.69
2020-09-16 00:15:00+02:00               43350.0       20.55
2020-09-16 00:30:00+02:00               42000.0       20.41
2020-09-16 00:45:00+02:00               40900.0       20.27
2020-09-16 01:00:00+02:00               39800.0       20.13
…                                         …          …
2020-09-23 22:45:00+02:00               45150.0       16.62
2020-09-23 23:00:00+02:00               46300.0       16.48
2020-09-23 23:15:00+02:00               45550.0       16.28
2020-09-23 23:30:00+02:00               44800.0       16.08
2020-09-23 23:45:00+02:00               43900.0       15.88

[768 rows x 2 columns]
```

[17]:
```
# lets create the train set with historical data

portfolio_slp_historic = portfolio_slp[portfolio_slp.index <=␣
 ↪historic_load_measured.index.max()]

slp_historic = pd.merge(
    portfolio_slp_historic,
    historic_load_measured[['slp_kw']],
    how='inner', left_index=True, right_index=True
)

slp_historic = pd.merge(
    slp_historic,
```

```
    weather_and_tso_forecasts,
    how='inner', left_index=True, right_index=True
)

slp_historic
```

[17]:

```
                           contracts_count  subscribed_power_kva  \
time
2020-09-16 00:00:00+02:00              1.0                   6.0
2020-09-16 00:15:00+02:00              1.0                   6.0
2020-09-16 00:30:00+02:00              1.0                   6.0
2020-09-16 00:45:00+02:00              1.0                   6.0
2020-09-16 01:00:00+02:00              1.0                   6.0
…                                    …                   …
2020-09-19 22:45:00+02:00              3.0                  30.0
2020-09-19 23:00:00+02:00              3.0                  30.0
2020-09-19 23:15:00+02:00              3.0                  30.0
2020-09-19 23:30:00+02:00              3.0                  30.0
2020-09-19 23:45:00+02:00              3.0                  30.0

                           estimated_annual_consumption_kwh  slp_kw  \
time
2020-09-16 00:00:00+02:00                            4500.0  1.5066
2020-09-16 00:15:00+02:00                            4500.0  1.4574
2020-09-16 00:30:00+02:00                            4500.0  1.4082
2020-09-16 00:45:00+02:00                            4500.0  1.3678
2020-09-16 01:00:00+02:00                            4500.0  1.3273
…                                                       …     …
2020-09-19 22:45:00+02:00                           19500.0  9.7404
2020-09-19 23:00:00+02:00                           19500.0  9.3414
2020-09-19 23:15:00+02:00                           19500.0  8.8738
2020-09-19 23:30:00+02:00                           19500.0  8.4063
2020-09-19 23:45:00+02:00                           19500.0  8.2067

                           tso_forecast_load_mw  t_weighted
time
2020-09-16 00:00:00+02:00               44700.0      20.690
2020-09-16 00:15:00+02:00               43350.0      20.550
2020-09-16 00:30:00+02:00               42000.0      20.410
2020-09-16 00:45:00+02:00               40900.0      20.270
2020-09-16 01:00:00+02:00               39800.0      20.130
…                                         …         …
2020-09-19 22:45:00+02:00               42950.0      18.825
2020-09-19 23:00:00+02:00               44000.0      18.650
2020-09-19 23:15:00+02:00               43800.0      18.505
2020-09-19 23:30:00+02:00               43600.0      18.360
2020-09-19 23:45:00+02:00               42700.0      18.220
```

```
[384 rows x 6 columns]
```

[18]:
```python
# lets create the input data for our forecast
portfolio_slp_forecast = portfolio_slp[portfolio_slp.index >= pd.
 ↪to_datetime('2020-09-21 00:00:00+02:00')]

slp_forecast_input = pd.merge(
    portfolio_slp_forecast,
    weather_and_tso_forecasts,
    how='inner', left_index=True, right_index=True
)

slp_forecast_input
```

[18]:
```
                           contracts_count  subscribed_power_kva  \
time
2020-09-21 00:00:00+02:00              3.0                  30.0
2020-09-21 00:15:00+02:00              3.0                  30.0
2020-09-21 00:30:00+02:00              3.0                  30.0
2020-09-21 00:45:00+02:00              3.0                  30.0
2020-09-21 01:00:00+02:00              3.0                  30.0
…                                      …                     …
2020-09-23 22:45:00+02:00              4.0                  36.0
2020-09-23 23:00:00+02:00              4.0                  36.0
2020-09-23 23:15:00+02:00              4.0                  36.0
2020-09-23 23:30:00+02:00              4.0                  36.0
2020-09-23 23:45:00+02:00              4.0                  36.0

                           estimated_annual_consumption_kwh  \
time
2020-09-21 00:00:00+02:00                           19500.0
2020-09-21 00:15:00+02:00                           19500.0
2020-09-21 00:30:00+02:00                           19500.0
2020-09-21 00:45:00+02:00                           19500.0
2020-09-21 01:00:00+02:00                           19500.0
…                                                      …
2020-09-23 22:45:00+02:00                           23500.0
2020-09-23 23:00:00+02:00                           23500.0
2020-09-23 23:15:00+02:00                           23500.0
2020-09-23 23:30:00+02:00                           23500.0
2020-09-23 23:45:00+02:00                           23500.0

                           tso_forecast_load_mw  t_weighted
time
2020-09-21 00:00:00+02:00               40600.0       18.36
2020-09-21 00:15:00+02:00               39550.0       18.18
```

```
2020-09-21 00:30:00+02:00                    38500.0       18.00
2020-09-21 00:45:00+02:00                    37450.0       17.82
2020-09-21 01:00:00+02:00                    36400.0       17.64
…                                              …         …
2020-09-23 22:45:00+02:00                    45150.0       16.62
2020-09-23 23:00:00+02:00                    46300.0       16.48
2020-09-23 23:15:00+02:00                    45550.0       16.28
2020-09-23 23:30:00+02:00                    44800.0       16.08
2020-09-23 23:45:00+02:00                    43900.0       15.88

[288 rows x 5 columns]
```

[19]:
```python
# create minimalistic features, for the example, just the hour
def featurize(df):
    df = df.copy(deep=True)
    df["hour"] = df.index.hour
    return df
```

[20]:
```python
slp_historic = featurize(slp_historic)
slp_forecast_input = featurize(slp_forecast_input)
```

In this example we will use a simple linear regression using the implementation in `sklearn`. Enda has a wrapper that works with any scikit-learn estimator : `enda.ml_backends.sklearn_estimator.EndaSklearnEstimator`. It makes it easier to deal with timeseries and pandas dataframes. It also allows to use estimators in more advanced models defined in enda.

To save a trained model we will use `joblib`.

Install the requirements if you haven't already :

`pip install scikit-learn joblib`

[21]:
```python
from enda.ml_backends.sklearn_estimator import EndaSklearnEstimator
from sklearn.linear_model import LinearRegression
import joblib
```

[22]:
```python
lin_reg = EndaSklearnEstimator(LinearRegression())
lin_reg.train(slp_historic, target_col='slp_kw')
```

[23]:
```python
# save model to a file
model_path = os.path.join(DIR, "lin_reg.joblib")
joblib.dump(lin_reg, model_path)
del lin_reg
```

[24]:
```python
# load model from the file
lin_reg = joblib.load(model_path)
prediction = lin_reg.predict(slp_forecast_input, target_col='slp_kw')
```

```
assert (prediction.index == slp_forecast_input.index).all()  # verify that the␣
↪pandas.DatetimeIndex is conserved
```

[25]:
```
prediction
```

[25]:
```
                              slp_kw
time
2020-09-21 00:00:00+02:00   8.890794
2020-09-21 00:15:00+02:00   8.785566
2020-09-21 00:30:00+02:00   8.680339
2020-09-21 00:45:00+02:00   8.575111
2020-09-21 01:00:00+02:00   8.564227
…                                …
2020-09-23 22:45:00+02:00  13.834379
2020-09-23 23:00:00+02:00  14.058107
2020-09-23 23:15:00+02:00  13.985939
2020-09-23 23:30:00+02:00  13.913771
2020-09-23 23:45:00+02:00  13.825492

[288 rows x 1 columns]
```

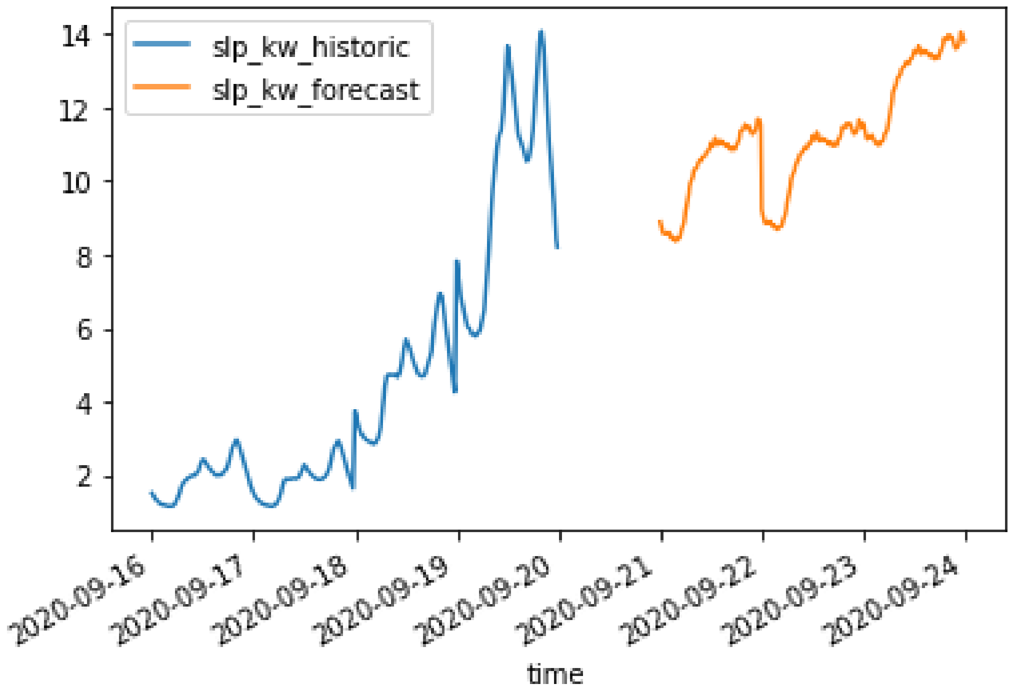
To visualize pandas dataframes, we use `matplotlib` as backend. Install it if you haven't already :

`pip install matplotlib`

[26]:
```
import matplotlib.pyplot as plt
```

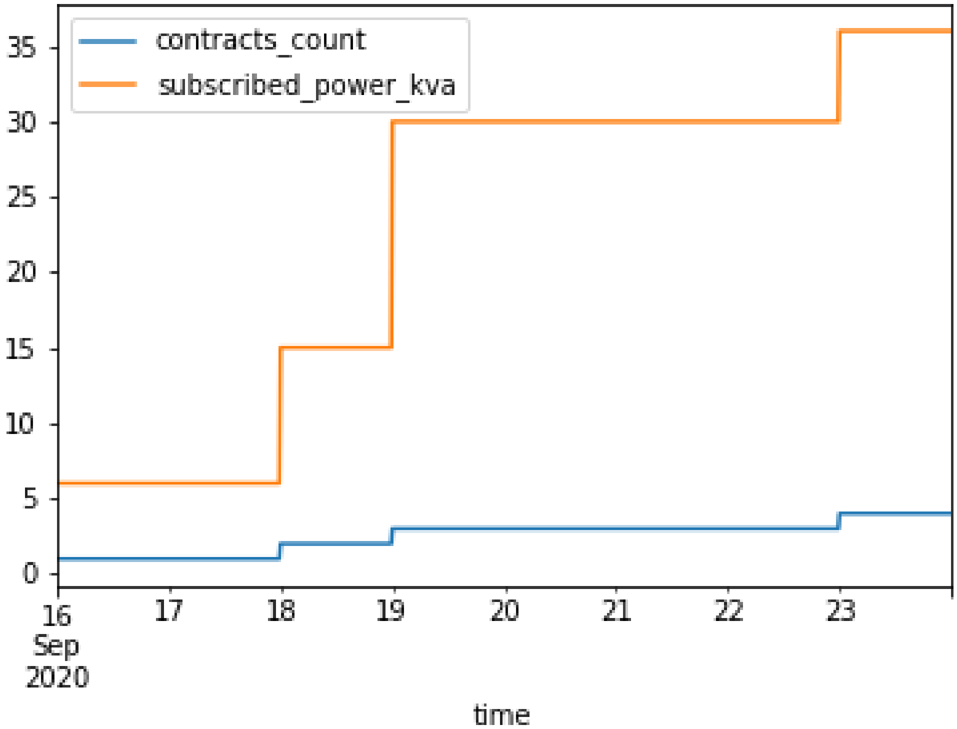
[27]:
```
# plot consumption : historic and forecast
to_plot = pd.merge(
    slp_historic["slp_kw"].to_frame("slp_kw_historic"),
    prediction.rename(columns={"slp_kw": "slp_kw_forecast"}),
    how='outer', left_index=True, right_index=True
)
to_plot.plot()
```

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



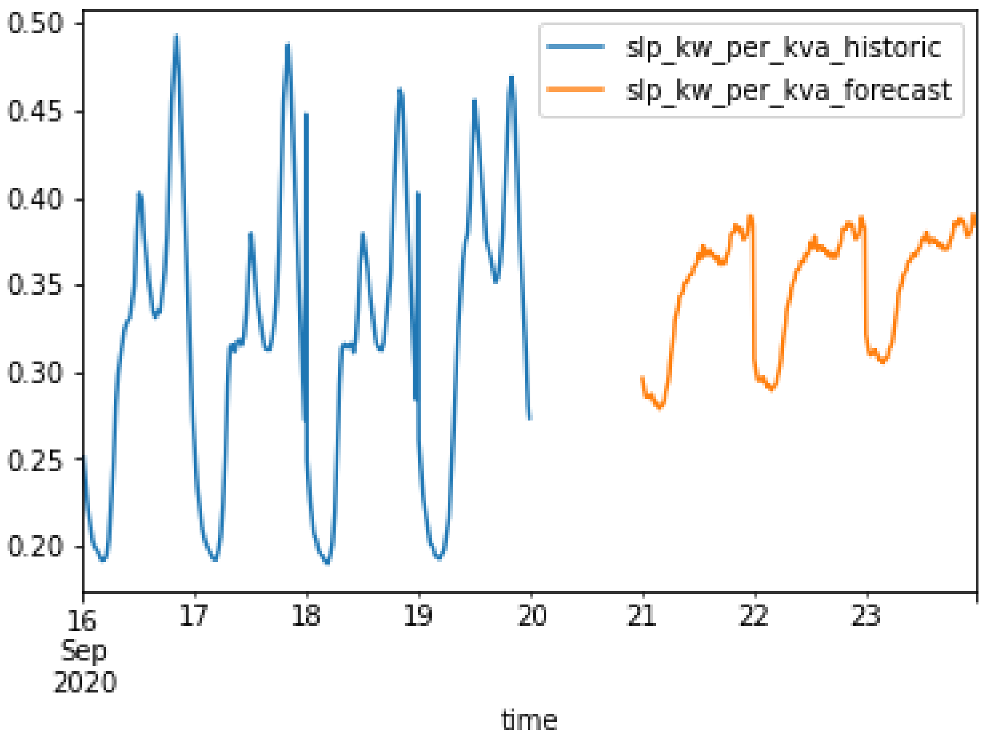
[28]:
```
# plot the size of the portfolio of SLP customers over time
portfolio_slp[["contracts_count", "subscribed_power_kva"]].plot()
```

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



[29]:
```
# plot consumption per kva: historic and forecast
to_plot = pd.merge(
    (slp_historic["slp_kw"]/slp_historic["subscribed_power_kva"]).
 ↪to_frame("slp_kw_per_kva_historic"),
    (prediction["slp_kw"]/portfolio_slp["subscribed_power_kva"]).
 ↪to_frame("slp_kw_per_kva_forecast"),
    how='outer', left_index=True, right_index=True
)
to_plot.plot()
```

[29]:
```
<AxesSubplot:xlabel='time'>
```



1.3 3. Try it yourself

As an exercise, you should repeat the previous analysis/prediction but this time on `smart-metered` customers.

```
[ ]:
```

1.4 Conclusion

Thats all for this introduction. Go to Example B for a more complete and in-depth example. Thanks for reading and don't hesitate to send feeback at: emmanuel.charon@enercoop.org !