

# ExampleA

March 25, 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
jupyter notebook # launch jupyter
```

Then you can download the notebook (ExampleA.ipynb) and the dataset (example\_a.zip) to your local machine. 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 30 min time-step. The desired time-zone is ‘Europe/Paris’. 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	PR01	
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]: # 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]: # 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]: # 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]: 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]: # 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='30min',
    tz='Europe/Paris'
)
```

```
[12]: portfolio_slp
```

```
[12]:
```

	contracts_count	subscribed_power_kva \
time		
2020-09-16 00:00:00+02:00	1.0	6.0
2020-09-16 00:30:00+02:00	1.0	6.0
2020-09-16 01:00:00+02:00	1.0	6.0
2020-09-16 01:30:00+02:00	1.0	6.0
2020-09-16 02:00:00+02:00	1.0	6.0
...	...	...
2020-09-23 21:30:00+02:00	4.0	36.0
2020-09-23 22:00:00+02:00	4.0	36.0
2020-09-23 22:30:00+02:00	4.0	36.0

2020-09-23 23:00:00+02:00	4.0	36.0
2020-09-23 23:30:00+02:00	4.0	36.0

	estimated_annual_consumption_kwh
time	
2020-09-16 00:00:00+02:00	4500.0
2020-09-16 00:30:00+02:00	4500.0
2020-09-16 01:00:00+02:00	4500.0
2020-09-16 01:30:00+02:00	4500.0
2020-09-16 02:00:00+02:00	4500.0
...	...
2020-09-23 21:30:00+02:00	23500.0
2020-09-23 22:00:00+02:00	23500.0
2020-09-23 22:30:00+02:00	23500.0
2020-09-23 23:00:00+02:00	23500.0
2020-09-23 23:30:00+02:00	23500.0

[384 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/Paris' to make sure the df
    ↳ has a pandas.DatetimeIndex
    df['time'] = df.align_timezone(df['time'], tzinfo = 'Europe/
    ↳ Paris')
    df.set_index('time', inplace=True)
```

```
[15]: historic_load_measured
```

	smart_metered_kw	slp_kw
time		
2020-09-16 00:00:00+02:00	0.000000	1.506618
2020-09-16 00:30:00+02:00	0.000000	1.408237
2020-09-16 01:00:00+02:00	0.000000	1.327301

2020-09-16 01:30:00+02:00	0.000000	1.269845
2020-09-16 02:00:00+02:00	0.000000	1.219173
...	...	...
2020-09-19 21:30:00+02:00	4.547055	11.590506
2020-09-19 22:00:00+02:00	4.389772	10.854016
2020-09-19 22:30:00+02:00	4.244040	10.139445
2020-09-19 23:00:00+02:00	4.053145	9.341356
2020-09-19 23:30:00+02:00	3.915264	8.406317

[192 rows x 2 columns]

```
[16]: weather_and_tso_forecasts
```

```
[16]:
```

time	tso_forecast_load_mw	t_weighted
2020-09-16 00:00:00+02:00	44700.0	20.69
2020-09-16 00:30:00+02:00	42000.0	20.41
2020-09-16 01:00:00+02:00	39800.0	20.13
2020-09-16 01:30:00+02:00	40100.0	19.73
2020-09-16 02:00:00+02:00	39600.0	19.33
...	...	...
2020-09-23 21:30:00+02:00	45000.0	17.34
2020-09-23 22:00:00+02:00	43300.0	17.05
2020-09-23 22:30:00+02:00	44000.0	16.76
2020-09-23 23:00:00+02:00	46300.0	16.48
2020-09-23 23:30:00+02:00	44800.0	16.08

[384 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 00:30:00+02:00	1.0	6.0	
2020-09-16 01:00:00+02:00	1.0	6.0	
2020-09-16 01:30:00+02:00	1.0	6.0	
2020-09-16 02:00:00+02:00	1.0	6.0	
...	...	...	
2020-09-19 21:30:00+02:00	3.0	30.0	
2020-09-19 22:00:00+02:00	3.0	30.0	
2020-09-19 22:30:00+02:00	3.0	30.0	
2020-09-19 23:00:00+02:00	3.0	30.0	
2020-09-19 23:30:00+02:00	3.0	30.0	

	estimated_annual_consumption_kwh	slp_kw	\
time			
2020-09-16 00:00:00+02:00	4500.0	1.506618	
2020-09-16 00:30:00+02:00	4500.0	1.408237	
2020-09-16 01:00:00+02:00	4500.0	1.327301	
2020-09-16 01:30:00+02:00	4500.0	1.269845	
2020-09-16 02:00:00+02:00	4500.0	1.219173	
...	...	...	
2020-09-19 21:30:00+02:00	19500.0	11.590506	
2020-09-19 22:00:00+02:00	19500.0	10.854016	
2020-09-19 22:30:00+02:00	19500.0	10.139445	
2020-09-19 23:00:00+02:00	19500.0	9.341356	
2020-09-19 23:30:00+02:00	19500.0	8.406317	

	tso_forecast_load_mw	t_weighted
time		
2020-09-16 00:00:00+02:00	44700.0	20.69
2020-09-16 00:30:00+02:00	42000.0	20.41
2020-09-16 01:00:00+02:00	39800.0	20.13
2020-09-16 01:30:00+02:00	40100.0	19.73
2020-09-16 02:00:00+02:00	39600.0	19.33
...	...	...
2020-09-19 21:30:00+02:00	41800.0	19.64
2020-09-19 22:00:00+02:00	40700.0	19.35
2020-09-19 22:30:00+02:00	41900.0	19.00
2020-09-19 23:00:00+02:00	44000.0	18.65
2020-09-19 23:30:00+02:00	43600.0	18.36

[192 rows x 6 columns]

```
[18]: # 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 00:30:00+02:00           3.0          30.0
2020-09-21 01:00:00+02:00           3.0          30.0
2020-09-21 01:30:00+02:00           3.0          30.0
2020-09-21 02:00:00+02:00           3.0          30.0
...
2020-09-23 21:30:00+02:00           4.0          36.0
2020-09-23 22:00:00+02:00           4.0          36.0
2020-09-23 22:30:00+02:00           4.0          36.0
2020-09-23 23:00:00+02:00           4.0          36.0
2020-09-23 23:30:00+02:00           4.0          36.0

```

```

estimated_annual_consumption_kwh  \
time
2020-09-21 00:00:00+02:00          19500.0
2020-09-21 00:30:00+02:00          19500.0
2020-09-21 01:00:00+02:00          19500.0
2020-09-21 01:30:00+02:00          19500.0
2020-09-21 02:00:00+02:00          19500.0
...
2020-09-23 21:30:00+02:00          23500.0
2020-09-23 22:00:00+02:00          23500.0
2020-09-23 22:30:00+02:00          23500.0
2020-09-23 23:00:00+02:00          23500.0
2020-09-23 23:30:00+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 00:30:00+02:00          38500.0          18.00
2020-09-21 01:00:00+02:00          36400.0          17.64
2020-09-21 01:30:00+02:00          36400.0          17.42
2020-09-21 02:00:00+02:00          35800.0          17.19
...
2020-09-23 21:30:00+02:00          45000.0          17.34
2020-09-23 22:00:00+02:00          43300.0          17.05

```

2020-09-23 22:30:00+02:00	44000.0	16.76
2020-09-23 23:00:00+02:00	46300.0	16.48
2020-09-23 23:30:00+02:00	44800.0	16.08

[144 rows x 5 columns]

```
[19]: # 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]: 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 `sklearn` estimator : `enda.ml_backends.sklearn_estimator.SklearnEstimator`. It makes it easier to deal with timeseries and pandas dataframes. To save a trained model we will use `joblib`.

Install the requirements:

```
pip install scikit-learn joblib
```

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

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

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

```
[24]: # 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    8.885987
2020-09-21 00:30:00+02:00    8.676176
```

```

2020-09-21 01:00:00+02:00    8.561693
2020-09-21 01:30:00+02:00    8.571478
2020-09-21 02:00:00+02:00    8.612515
...
2020-09-23 21:30:00+02:00   13.717845
2020-09-23 22:00:00+02:00   13.643261
2020-09-23 22:30:00+02:00   13.731435
2020-09-23 23:00:00+02:00   14.086547
2020-09-23 23:30:00+02:00   13.943036

```

```
[144 rows x 1 columns]
```

To visualize pandas dataframes, we use `matplotlib` as backend :

```
pip install matplotlib
```

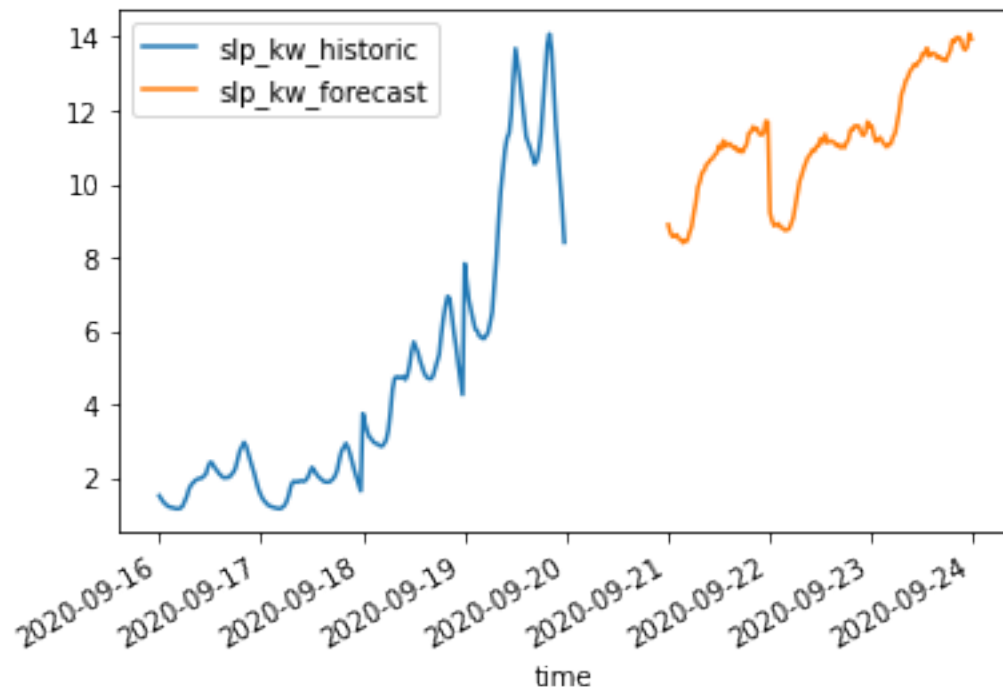
```
[26]: import matplotlib.pyplot as plt
```

```

[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()

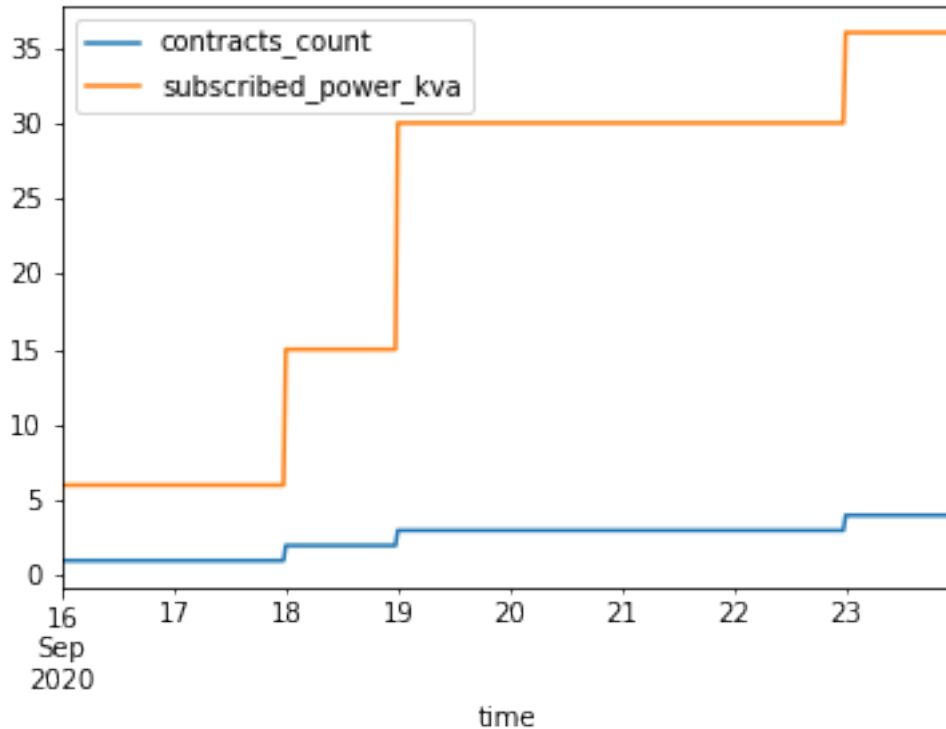
```

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



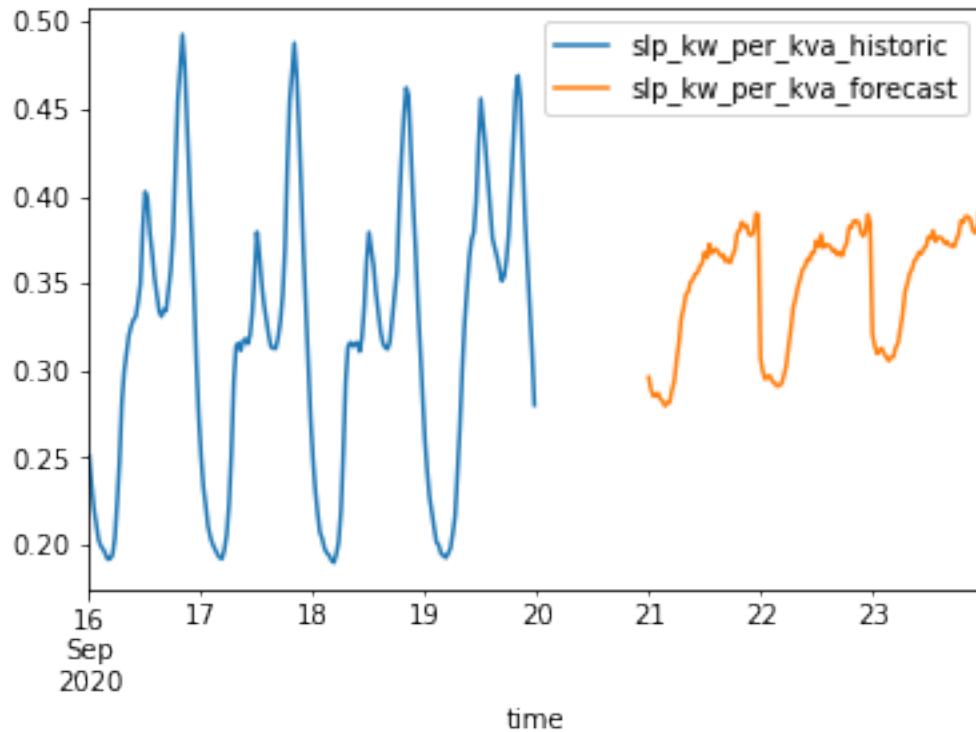
```
[28]: # plot the size of the portfolio of SLP customers over time
portfolio_slp[["contracts_count", "subscribed_power_kva"]].plot()
```

```
[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

That's 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 feedback at: [emmanuel.charon@enercoop.org](mailto:emmanuel.charon@enercoop.org) !