```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os, sys

nb_path = '/content/drive/MyDrive/Colab Notebooks/site-packages'
sys.path.insert(0, nb_path)  # or append(nb_path)
```

```
# Standard library imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# First-party imports
from gluonts.dataset.common import ListDataset
import mxnet as mx
from mxnet import gluon
from gluonts.model.deepar import DeepAREstimator
from gluonts.mx.trainer import Trainer
```

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
mx.random.seed(0)
np.random.seed(0)
```

deepAR

```
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/태양열 발전량 예측/Timestamped.csv")
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
df
```

	Day	Hour	Minute	DHI	DNI	WS	RH	T	TARGET	Year	Month	Timestamp
0	1	0	0	0	0	1.5	69.08	-12	0.0	2016	1	2016-01-01 00:00:00
1	1	0	30	0	0	1.5	69.06	-12	0.0	2016	1	2016-01-01

```
df = df.set_index(df['Timestamp'])
cut_edge = pd.to_datetime('2018-12-31 23:30:00')
temp = df[{'TARGET', 'DHI', 'DNI', 'RH', 'T'}]
# train_end_stamp = pd.to_datetime('2016-06-27 23:30:00')
```

2016-01

```
prediction_window = 96 # 2 * 48

f1 = temp.DHI.values[:-prediction_window]
f2 = temp.DNI.values[:-prediction_window]
f3 = temp.RH.values[:-prediction_window]
f4 = temp['T'][:-prediction_window].values # callling .T transposes dataframe
```

```
train_ds = ListDataset(
    [{"start": temp.index[0],
      "target": np.array(temp.TARGET.values[:-prediction_window]),
      "feat_dynamic_real": [f1,f2,f3,f4] }],
    freq = "30min",
    one_dim_target= True
)
```

```
freq = "30min"

estimator = DeepAREstimator(prediction_length=prediction_window, use_feat_dynamic_real= True , f
                            trainer=Trainer(epochs=50, batch_size=128, ctx = mx.context.gpu(),))
```

```
predictor = estimator.train(training_data = train_ds)
```

```
  0%|          | 0/50 [00:00<?, ?it/s]learning rate from ``lr_scheduler`` has been overwritte
100%|██████████| 50/50 [00:42<00:00,  1.18it/s, epoch=1/50, avg_epoch_loss=3.95]
100%|██████████| 50/50 [00:19<00:00,  2.61it/s, epoch=2/50, avg_epoch_loss=2.73]
100%|██████████| 50/50 [00:19<00:00,  2.58it/s, epoch=3/50, avg_epoch_loss=2.26]
100%|██████████| 50/50 [00:19<00:00,  2.62it/s, epoch=4/50, avg_epoch_loss=2.01]
100%|██████████| 50/50 [00:19<00:00,  2.59it/s, epoch=5/50, avg_epoch_loss=1.87]
100%|██████████| 50/50 [00:19<00:00,  2.59it/s, epoch=6/50, avg_epoch_loss=1.74]
100%|██████████| 50/50 [00:19<00:00,  2.62it/s, epoch=7/50, avg_epoch_loss=1.71]
100%|██████████| 50/50 [00:19<00:00,  2.58it/s, epoch=8/50, avg_epoch_loss=1.59]
100%|██████████| 50/50 [00:19<00:00,  2.61it/s, epoch=9/50, avg_epoch_loss=1.49]
100%|██████████| 50/50 [00:19<00:00,  2.62it/s, epoch=10/50, avg_epoch_loss=1.43]
100%|██████████| 50/50 [00:19<00:00,  2.56it/s, epoch=11/50, avg_epoch_loss=1.38]
100%|██████████| 50/50 [00:18<00:00,  2.63it/s, epoch=12/50, avg_epoch_loss=1.34]
100%|██████████| 50/50 [00:19<00:00,  2.60it/s, epoch=13/50, avg_epoch_loss=1.29]
100%|██████████| 50/50 [00:18<00:00,  2.64it/s, epoch=14/50, avg_epoch_loss=1.24]
100%|██████████| 50/50 [00:19<00:00,  2.59it/s, epoch=15/50, avg_epoch_loss=1.21]
100%|██████████| 50/50 [00:19<00:00,  2.60it/s, epoch=16/50, avg_epoch_loss=1.18]
100%|██████████| 50/50 [00:19<00:00,  2.59it/s, epoch=17/50, avg_epoch_loss=1.16]
100%|██████████| 50/50 [00:19<00:00,  2.61it/s, epoch=18/50, avg_epoch_loss=1.12]
```

```
100%|██████████| 50/50 [00:19<00:00,  2.62it/s, epoch=19/50, avg_epoch_loss=1.1]
100%|██████████| 50/50 [00:19<00:00,  2.59it/s, epoch=20/50, avg_epoch_loss=1.08]
100%|██████████| 50/50 [00:19<00:00,  2.54it/s, epoch=21/50, avg_epoch_loss=1.06]
100%|██████████| 50/50 [00:19<00:00,  2.59it/s, epoch=22/50, avg_epoch_loss=1.02]
100%|██████████| 50/50 [00:19<00:00,  2.62it/s, epoch=23/50, avg_epoch_loss=1.03]
100%|██████████| 50/50 [00:19<00:00,  2.56it/s, epoch=24/50, avg_epoch_loss=0.998]
100%|██████████| 50/50 [00:19<00:00,  2.56it/s, epoch=25/50, avg_epoch_loss=0.991]
100%|██████████| 50/50 [00:19<00:00,  2.59it/s, epoch=26/50, avg_epoch_loss=0.965]
100%|██████████| 50/50 [00:19<00:00,  2.61it/s, epoch=27/50, avg_epoch_loss=0.961]
100%|██████████| 50/50 [00:19<00:00,  2.52it/s, epoch=28/50, avg_epoch_loss=0.945]
100%|██████████| 50/50 [00:18<00:00,  2.65it/s, epoch=29/50, avg_epoch_loss=0.92]
100%|██████████| 50/50 [00:19<00:00,  2.62it/s, epoch=30/50, avg_epoch_loss=0.918]
100%|██████████| 50/50 [00:19<00:00,  2.62it/s, epoch=31/50, avg_epoch_loss=0.897]
100%|██████████| 50/50 [00:18<00:00,  2.63it/s, epoch=32/50, avg_epoch_loss=0.876]
100%|██████████| 50/50 [00:19<00:00,  2.60it/s, epoch=33/50, avg_epoch_loss=0.867]
100%|██████████| 50/50 [00:19<00:00,  2.59it/s, epoch=34/50, avg_epoch_loss=0.858]
100%|██████████| 50/50 [00:19<00:00,  2.60it/s, epoch=35/50, avg_epoch_loss=0.843]
100%|██████████| 50/50 [00:18<00:00,  2.63it/s, epoch=36/50, avg_epoch_loss=0.835]
100%|██████████| 50/50 [00:19<00:00,  2.59it/s, epoch=37/50, avg_epoch_loss=0.811]
100%|██████████| 50/50 [00:19<00:00,  2.63it/s, epoch=38/50, avg_epoch_loss=0.811]
100%|██████████| 50/50 [00:18<00:00,  2.65it/s, epoch=39/50, avg_epoch_loss=0.794]
100%|██████████| 50/50 [00:19<00:00,  2.61it/s, epoch=40/50, avg_epoch_loss=0.782]
100%|██████████| 50/50 [00:19<00:00,  2.62it/s, epoch=41/50, avg_epoch_loss=0.762]
100%|██████████| 50/50 [00:19<00:00,  2.63it/s, epoch=42/50, avg_epoch_loss=0.767]
100%|██████████| 50/50 [00:18<00:00,  2.65it/s, epoch=43/50, avg_epoch_loss=0.743]
100%|██████████| 50/50 [00:19<00:00,  2.59it/s, epoch=44/50, avg_epoch_loss=0.725]
100%|██████████| 50/50 [00:18<00:00,  2.65it/s, epoch=45/50, avg_epoch_loss=0.725]
100%|██████████| 50/50 [00:19<00:00,  2.56it/s, epoch=46/50, avg_epoch_loss=0.718]
100%|██████████| 50/50 [00:18<00:00,  2.66it/s, epoch=47/50, avg_epoch_loss=0.72]
100%|██████████| 50/50 [00:19<00:00,  2.60it/s, epoch=48/50, avg_epoch_loss=0.703]
100%|██████████| 50/50 [00:19<00:00,  2.59it/s, epoch=49/50, avg_epoch_loss=0.707]
100%|██████████| 50/50 [00:18<00:00,  2.64it/s, epoch=50/50, avg_epoch_loss=0.688]
```

```
from pathlib import Path
predictor.serialize(Path("/content/drive/MyDrive/Colab Notebooks/태양열 발전량 예측/"))
```

```
WARNING:root:Serializing RepresentableBlockPredictor instances does not save the prediction n
```

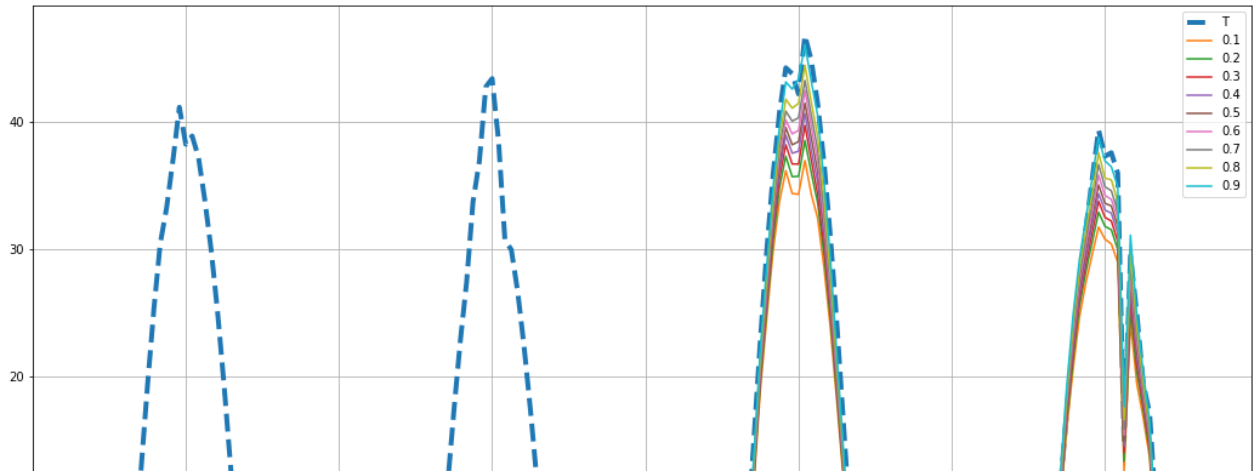
```
from gluonts.evaluation.backtest import make_evaluation_predictions, backtest_metrics

valid_ds = ListDataset(
    [{"start": temp.index[0],
      "target": temp.TARGET.values,
      "feat_dynamic_real": [temp.DHI.values, temp.DNI.values, temp.RH.values, temp['T'].values]
    freq = "30min",
    one_dim_target= True
)

# test_data의 target 시간 이후부터 예측을 실행하는 것으로 보임. train과 test는 무슨 연관이 있는

forecast_iter, ts_iter = make_evaluation_predictions(valid_ds, predictor=predictor, num_samples=
```

```
forecasts = next(forecast_iter) # forecasts: SampleForecast
tss = next(ts_iter) # tss : pd.DataFrame
```

```
quantile_forecasts = {}
quantiles = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

for quantile in quantiles:
    quantile_forecasts[str(quantile)] = forecasts.quantile(quantile)
```

```
def plot_prob_forecasts(
        ts_entry: Union[pd.DataFrame, pd.Series],
        quantile_forecasts: Union[dict, pd.DataFrame],
        prediction_length=96,
        return_forecasts=True):
    history_length = 48*4

    if isinstance(quantile_forecasts, pd.DataFrame):
        quantile_forecasts.index = ts_entry.index[-quantile_forecasts.prediction_length:]

    elif isinstance(quantile_forecasts, dict):
        quantile_forecasts = pd.DataFrame(quantile_forecasts, index=ts_entry.index[-prediction_l

    quantile_forecasts = quantile_forecasts[quantile_forecasts]
    fig, ax = plt.subplots(1, 1, figsize=(18, 10))
    ts_entry[-48 * 4:].plot(ax=ax, linewidth=4, linestyle='dashed')  # plot the history target
    plt.legend('True')
    quantile_forecasts.plot(y=quantile_forecasts.columns, ax=ax, linestyle='solid')

    plt.grid(which="both")
    plt.show()

    if return_forecasts:
        return quantile_forecasts
```

```
final_predictions = plot_prob_forecasts(tss, quantile_forecasts)
```




```
1 final_predictions
```

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	
2018-12-29 00:00:00	-0.140337	-0.114567	-0.099140	-0.084119	-0.071650	-0.057723	-0.040758	-0
2018-12-29 00:30:00	-0.135873	-0.111703	-0.093222	-0.080359	-0.065177	-0.051251	-0.037038	-0
2018-12-29 01:00:00	-0.127450	-0.106547	-0.090211	-0.079888	-0.066025	-0.052626	-0.038955	-0
2018-12-29 01:30:00	-0.132321	-0.111219	-0.094061	-0.079572	-0.063088	-0.051115	-0.033155	-0
2018-12-29 02:00:00	-0.132309	-0.106496	-0.090889	-0.077353	-0.063779	-0.050846	-0.034968	-0
...	...	...	...	...	...	...	...	
2018-								

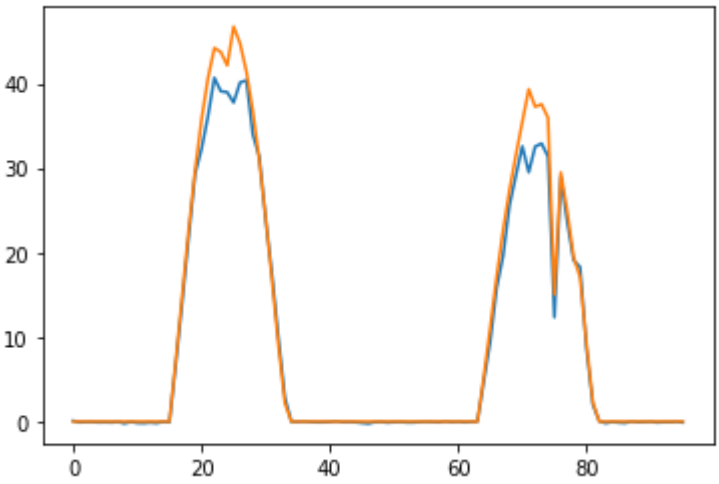
```
1 # 피쳐 4개  T저거 뺄까...
2 plt.plot(x,predict_entry, answer)
```

```
[<matplotlib.lines.Line2D at 0x7fac89e88898>,
 <matplotlib.lines.Line2D at 0x7fac89e88198>]
```

1

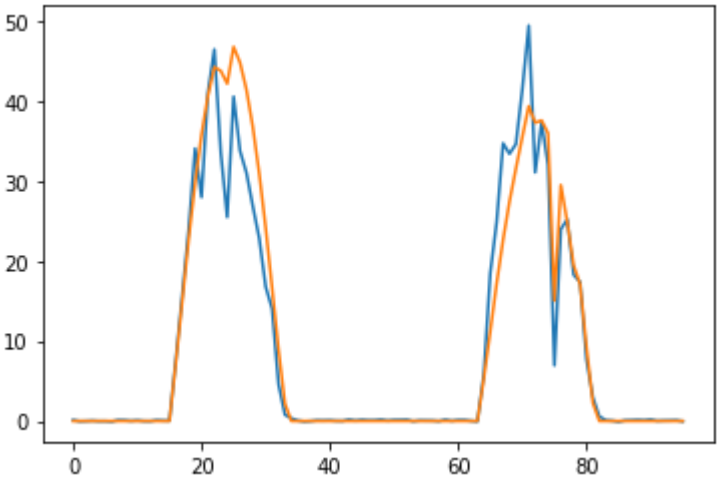
```
1 # 피쳐 3개가 제일 좋아보임
2 plt.plot(x,predict_entry, answer)
```

```
[<matplotlib.lines.Line2D at 0x7fac2779a8d0>,
 <matplotlib.lines.Line2D at 0x7fac2779aba8>]
```



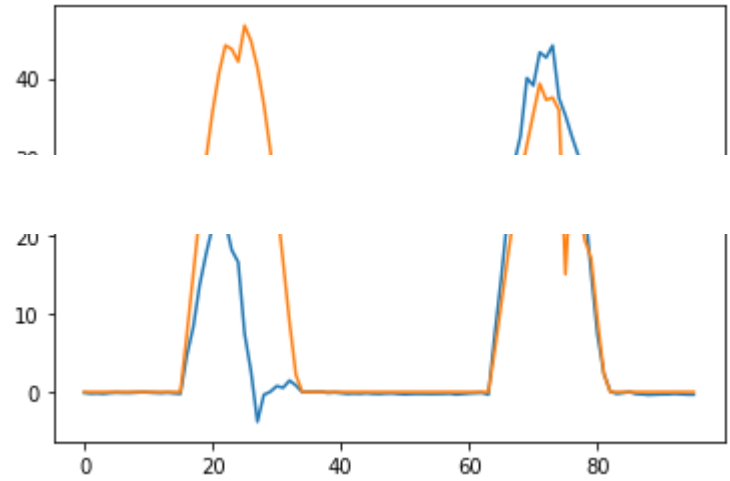

```
1 plt.plot(x,predict_entry, answer)
```

```
[<matplotlib.lines.Line2D at 0x7fac87cc1ba8>,
 <matplotlib.lines.Line2D at 0x7fac87cc1748>]
```




```
1 plt.plot(x,predict_entry, answer)
```

```
[<matplotlib.lines.Line2D at 0x7fac884d92e8>,
 <matplotlib.lines.Line2D at 0x7fac884d9400>]
```



1