

▼ 세팅

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

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

```
import os
os.chdir('/content/drive/MyDrive/Colab Notebooks')
```

```
!sudo apt-get install -y fonts-nanum
```

Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
The following NEW packages will be installed:  
 fonts-nanum  
0 upgraded, 1 newly installed, 0 to remove and 41 not upgraded.  
Need to get 10.3 MB of archives.  
After this operation, 34.1 MB of additional disk space will be used.  
Get:1 <http://archive.ubuntu.com/ubuntu> jammy/universe amd64 fonts-nanum all 20200506-1 [10.3 MB]  
Fetched 10.3 MB in 3s (3,546 kB/s)  
debconf: unable to initialize frontend: Dialog  
debconf: (No usable dialog-like program is installed, so the dialog based frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 78, <> line 1.)  
debconf: falling back to frontend: Readline  
debconf: unable to initialize frontend: Readline  
debconf: (This frontend requires a controlling tty.)  
debconf: falling back to frontend: Teletype  
dpkg-preconfigure: unable to re-open stdin:  
Selecting previously unselected package fonts-nanum.  
(Reading database ... 121713 files and directories currently installed.)  
Preparing to unpack .../fonts-nanum\_20200506-1\_all.deb ...  
Unpacking fonts-nanum (20200506-1) ...  
Setting up fonts-nanum (20200506-1) ...  
Processing triggers for fontconfig (2.13.1-4.2ubuntu5) ...

```
pip install pytimekr
```

Collecting pytimekr  
 Downloading pytimekr-0.1.0.tar.gz (7.3 kB)  
 Preparing metadata (setup.py) ... done  
Collecting lunardate>=0.1.5 (from pytimekr)  
 Downloading lunardate-0.2.2-py3-none-any.whl.metadata (3.6 kB)  
Downloading lunardate-0.2.2-py3-none-any.whl (18 kB)  
Building wheels for collected packages: pytimekr  
 Building wheel for pytimekr (setup.py) ... done  
 Created wheel for pytimekr: filename=pytimekr-0.1.0-py3-none-any.whl size=7923 sha256=7726fbe19e7835dd66acf57a90045063d2314f64c6db7f76651ca4420a2c7e8d  
 Stored in directory: /root/.cache/pip/wheels/df/bb/74/a8776ef7cbb5a05ebbcff3c386b16d8f5f08760a393ebec014  
Successfully built pytimekr  
Installing collected packages: lunardate, pytimekr  
Successfully installed lunardate-0.2.2 pytimekr-0.1.0

```
pip install pytorch_forecasting
```

Downloading lightning\_utilities-0.15.2-py3-none-any.whl.metadata (5.7 kB)  
Requirement already satisfied: packaging<27.0,>=20.0 in /usr/local/lib/python3.12/dist-packages (from lightning<3.0.0,>=2.0.0->pytorch\_forecasting) (25.0)  
Collecting torchmetrics<3.0,>0.7.0 (from lightning<3.0.0,>=2.0.0->pytorch\_forecasting)  
 Downloading torchmetrics-1.8.2-py3-none-any.whl.metadata (22 kB)  
Requirement already satisfied: tqdm<6.0,>=4.57.0 in /usr/local/lib/python3.12/dist-packages (from lightning<3.0.0,>=2.0.0->pytorch\_forecasting) (4.67.1)  
Requirement already satisfied: typing-extensions<6.0,>4.5.0 in /usr/local/lib/python3.12/dist-packages (from lightning<3.0.0,>=2.0.0->pytorch\_forecasting) (4.15.0)  
Collecting pytorch-lightning (from lightning<3.0.0,>=2.0.0->pytorch\_forecasting)  
 Downloading pytorch\_lightning-2.6.0-py3-none-any.whl.metadata (21 kB)  
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas<3.0.0,>=1.3.0->pytorch\_forecasting) (2.9.0.post0)  
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas<3.0.0,>=1.3.0->pytorch\_forecasting) (2025.2)  
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas<3.0.0,>=1.3.0->pytorch\_forecasting) (2025.2)  
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn<2.0,>=1.2->pytorch\_forecasting) (1.5.2)  
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn<2.0,>=1.2->pytorch\_forecasting) (3.6.0)  
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (3.20.0)  
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (75.2.0)  
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (1.14.0)  
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (3.6)  
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (3.1.6)  
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (12.6.77)  
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (12.6.77)  
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (12.6.80)  
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (9.10.2.21)  
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (12.6.4.1)  
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (11.3.0.4)  
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (10.3.7.77)  
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (11.7.1.2)  
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (0.7.1)  
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (2.27.5)  
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (3.3.20)  
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (12.6.77)  
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (12.6.85)  
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (1.11.1.6)  
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3.12/dist-packages (from torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (3.5.0)  
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]<2027.0,>=2022.5.0->lightning<3.0.0,>=2.0.0->pytorch\_forecasting  
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas<3.0.0,>=1.3.0->pytorch\_forecasting) (1.17.0)  
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (1.3.0)  
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from Jinja2->torch!=2.0.1,<3.0.0,>=2.0.0->pytorch\_forecasting) (3.0.3)  
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!>=4.0.0a1->fsspec[http]<2027.0,>=2022.5.0->lightning<3.0.0,>=2.0.0->p  
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!>=4.0.0a1->fsspec[http]<2027.0,>=2022.5.0->lightning<3.0.0,>=2.0.0->p  
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!>=4.0.0a1->fsspec[http]<2027.0,>=2022.5.0->lightning<3.0.0,>=2.0.0->pyto  
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!>=4.0.0a1->fsspec[http]<2027.0,>=2022.5.0->lightning<3.0.0,>=2.0.0->

```
Installing collected packages: lightning-utilities, torchmetrics, pytorch-lightning, lightning, pytorch_forecasting
Successfully installed lightning-2.6.0 lightning-utilities-0.15.2 pytorch-lightning-2.6.0 pytorch_forecasting-1.5.0 torchmetrics-1.8.2
```

```
import pandas as pd
import datetime as dt
import itertools
from pytimekr import pytimekr
from pytorch_forecasting import TimeSeriesDataSet
from pytorch_forecasting.data import GroupNormalizer
import lightning.pytorch as pl
from lightning.pytorch.callbacks import EarlyStopping, LearningRateMonitor # Changed for consistency
from pytorch_forecasting import TemporalFusionTransformer, QuantileLoss, Baseline
from pytorch_lightning.loggers import TensorBoardLogger
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt

plt.rc('font', family='NanumBarunGothic')
```

## 데이터 불러오기

```
electrade_2013 = pd.read_csv('2013_2016 일별 시간대별 연료원별 전력거래량.csv', encoding='cp949')
electrade_2017 = pd.read_csv('2017~2020 일별 시간대별 연료원별 전력거래량.csv', encoding='cp949')
electrade_2021 = pd.read_csv('일별 시간대별 연료원별 거래량_20220430.csv', encoding='cp949')
electrade_2022 = pd.read_csv('한국전력거래소_연료원별 전력거래량_20231031.csv', encoding='cp949')
electrade_2023 = pd.read_csv('한국전력거래소_연료원별 전력거래량_20231231.csv', encoding='cp949')
```

# 데이터 병합

```
electrade_2013.columns = ['거래일', '거래시간', '연료원', '전력거래량']
electrade_2017.columns = ['거래일', '거래시간', '연료원', '전력거래량']
electrade_2021 = electrade_2021[['거래일', '거래시간', '연료원', '전력거래량']]
electrade_2022 = electrade_2022[['거래일', '거래시간', '연료원', '전력거래량(MWh)']]
electrade_2022.columns = ['거래일', '거래시간', '연료원', '전력거래량']
electrade_2023 = electrade_2023[['거래일', '거래시간', '연료원', '전력거래량(MWh)']]
electrade_2023.columns = ['거래일', '거래시간', '연료원', '전력거래량']
```

```
electrade = pd.concat([electrade_2013, electrade_2017, electrade_2021, electrade_2022, electrade_2023]).reset_index(drop=True)
# electrade = pd.concat([electrade_2021, electrade_2022, electrade_2023]).reset_index(drop=True)
```

```
# electrade.loc[electrade['거래시간'] == 24, '거래시간'] = 0 # 자정 거래시간이 00이랑 24로 중복 표기되어있음 -> 0으로 통일
# electrade.거래일 = pd.to_datetime(electrade.거래일)
```

```
# # Combine date and hour to create a full datetime timestamp for accurate time indexing
# electrade['full_timestamp'] = electrade['거래일'] + pd.to_timedelta(electrade['거래시간'], unit='h')
```

```
# # Sort data to ensure correct chronological order within each group (연료원)
# electrade = electrade.sort_values(by=['full_timestamp', '연료원']).reset_index(drop=True)
```

electrade

	거래일	거래시간	연료원	전력거래량	
0	2013-01-01	0	원자력	16232.160050	
1	2013-01-01	0	석탄	23638.589390	
2	2013-01-01	0	LNG	13380.994560	
3	2013-01-01	0	유류	697.044902	
4	2013-01-01	0	양수	855.181596	
...	...	...	...	...	
1658011	2023-12-31	20	기타	42.000000	
1658012	2023-12-31	21	기타	42.000000	
1658013	2023-12-31	22	기타	41.000000	
1658014	2023-12-31	23	기타	39.000000	
1658015	2023-12-31	24	기타	38.000000	

1658016 rows × 4 columns

# 공휴일 리스트

```
holidays = []
for i in range(2013,2024):
    holidays.append(pytimekr.holidays(i))
holidays = list(itertools.chain(*holidays))
```

## 시간별, 연료원별 전력거래량(단기 예측)

```
electrade_short = pd.concat([electrade_2021, electrade_2022, electrade_2023]).reset_index(drop=True)
electrade_short.replace('유연탄', '석탄', inplace = True)
electrade_short.replace('무연탄', '석탄', inplace = True)
electrade_short = electrade_short.groupby(['거래일', '거래시간', '연료원']).sum().reset_index()
electrade_short.loc[electrade_short['거래시간'] == 24, '거래시간'] = 0 # 자정 거래시간이 00이랑 24로 중복 표기되어있음 -> 0으로 통일
electrade_short.거래일 = pd.to_datetime(electrade_short.거래일)
```

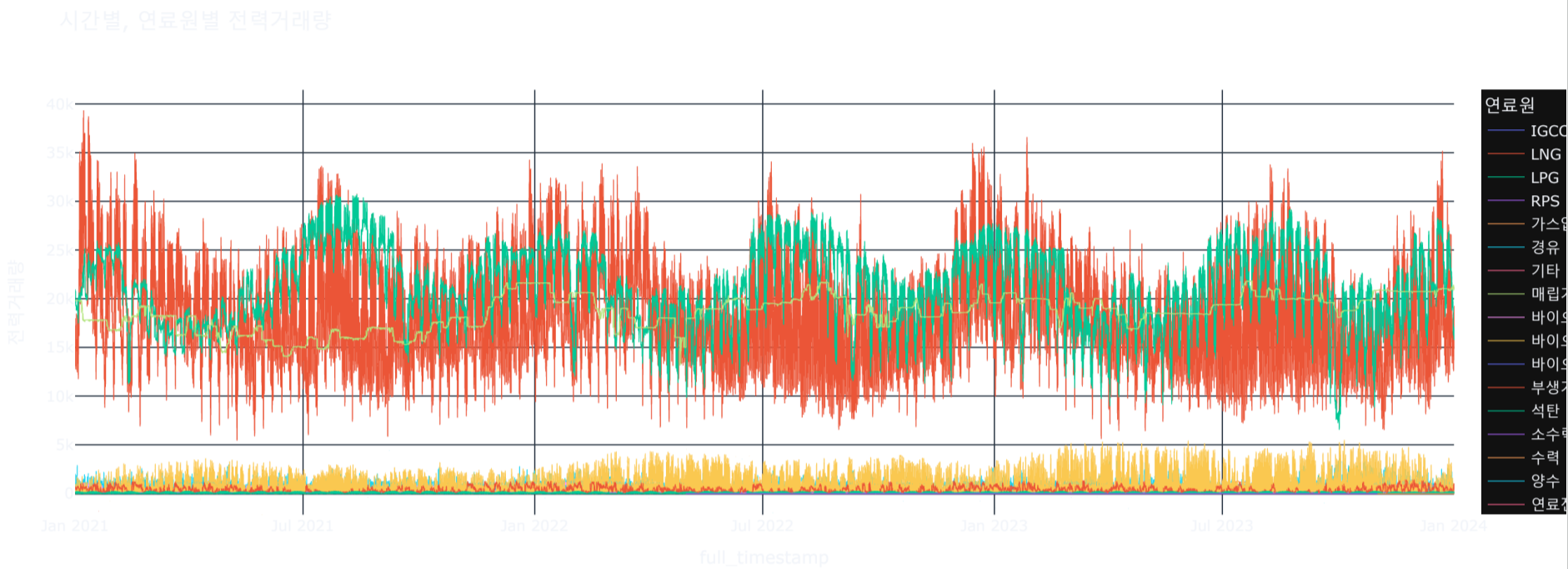
```
# Combine date and hour to create a full datetime timestamp for accurate time indexing
electrade_short['full_timestamp'] = electrade_short['거래일'] + pd.to_timedelta(electrade_short['거래시간'], unit='h')
```

```
# Sort data to ensure correct chronological order within each group (연료원)
electrade_short = electrade_short.sort_values(by=['full_timestamp', '연료원']).reset_index(drop=True)
electrade_short
```

	거래일	거래시간	연료원	전력거래량	full_timestamp	
	0	2021-01-01	0	IGCC	237.628160	2021-01-01 00:00:00
	1	2021-01-01	0	LNG	16679.367610	2021-01-01 00:00:00
	2	2021-01-01	0	LPG	40.633488	2021-01-01 00:00:00
	3	2021-01-01	0	RPS	0.000000	2021-01-01 00:00:00
	4	2021-01-01	0	가스압	0.000000	2021-01-01 00:00:00
	...	...	...	...	...	...
	613219	2023-12-31	23	중유	32.000000	2023-12-31 23:00:00
	613220	2023-12-31	23	태양광	1.000000	2023-12-31 23:00:00
	613221	2023-12-31	23	폐기물	122.000000	2023-12-31 23:00:00
	613222	2023-12-31	23	풍력	337.000000	2023-12-31 23:00:00
	613223	2023-12-31	23	해양에너지	17.000000	2023-12-31 23:00:00

시각화 24 rows × 5 columns

```
fig = px.line(electrade_short, x="full_timestamp", y="전력거래량", color='연료원', title='시간별, 연료원별 전력거래량', template = 'plotly_dark')
fig.update_traces(line_width= 0.7)
fig
```



```
electrade['time_idx_numeric'] = electrade.groupby('연료원')['full_timestamp'].transform(lambda x: (x - x.min()).dt.total_seconds() / 3600).astype(int)
```

```
electrade['월'] = electrade.거래일.dt.month.astype(str)
electrade['요일'] = electrade.거래일.dt.weekday.astype(str)
electrade['공휴일여부'] = electrade.거래일.isin(holidays)*1
electrade['공휴일여부'] = electrade['공휴일여부'].astype(str)
electrade['거래시간'] = electrade['거래시간'].astype(str)
# Rename columns to match the desired input for TimeSeriesDataSet and general clarity
electrade.rename(columns={
    '거래일': 'date_original', # Keeping original date for context, if needed
    '거래시간': 'hour',
    '연료원': 'fuel_type',
    '전력거래량': 'volume',
    'time_idx_numeric': 'time_idx' # This is the integer time index for TimeSeriesDataSet
}, inplace=True)
```

```
# Drop the 'full_timestamp' column as it's no longer needed after creating 'time_idx'
# And reorder columns for consistency
electrade.columns = ['date_original', 'hour', 'fuel_type', 'volume', 'full_timestamp', 'time_idx', 'month', 'day_of_week', 'is_holiday']
```

```
-----
KeyError                                Traceback (most recent call last)
/tmp/ipython-input-1969190317.py in <cell line: 0>()
----> 1 electrade['time_idx_numeric'] = electrade.groupby('연료원')['full_timestamp'].transform(lambda x: (x - x.min()).dt.total_seconds() / 3600).astype(int)
      2
      3 electrade['월'] = electrade.거래일.dt.month.astype(str)
      4 electrade['요일'] = electrade.거래일.dt.weekday.astype(str)
      5 electrade['공휴일여부'] = electrade.거래일.isin(holidays)*1
```

```
-----
1 frames -----
/usr/local/lib/python3.12/dist-packages/pandas/core/base.py in __getitem__(self, key)
    242     else:
    243         if key not in self.obj:
-> 244             raise KeyError(f"Column not found: {key}")
    245         ndim = self.obj[key].ndim
    246         return self._getitem(key, ndim=ndim)
```

```
KeyError: 'Column not found: full_timestamp'
```

다음 단계: 오류 설명

모델링

```
electrade_short['time_idx_numeric'] = electrade_short.groupby('연료원')['full_timestamp'].transform(lambda x: (x - x.min()).dt.total_seconds() / 3600).astype(int)
```

```
electrade_short['월'] = electrade_short.거래일.dt.month.astype(str)
electrade_short['요일'] = electrade_short.거래일.dt.weekday.astype(str)
```




```
electrade_short['공휴일여부'] = electrade_short.거래일.isin(holidays)*1
electrade_short['공휴일여부'] = electrade_short['공휴일여부'].astype(str)
electrade_short['거래시간'] = electrade_short['거래시간'].astype(str)
# Rename columns to match the desired input for TimeSeriesDataSet and general clarity
electrade_short.rename(columns={
    '거래일': 'date_original', # Keeping original date for context, if needed
    '거래시간': 'hour',
    '연료원': 'fuel_type',
    '전력거래량': 'volume',
    'time_idx_numeric': 'time_idx', # This is the integer time index for TimeSeriesDataSet
    '월': 'month',
    '요일': 'day_of_week',
    '공휴일여부': 'is_holiday'
}, inplace=True)

# Drop the 'full_timestamp' column as it's no longer needed after creating 'time_idx'
# And reorder columns for consistency
# electrade_short.columns = ['date_original', 'hour', 'fuel_type', 'volume', 'full_timestamp', 'time_idx', 'month', 'day_of_week', 'is_holiday']
```

/tmp/ipython-input-81997624.py:5: FutureWarning:

The behavior of 'isin' with dtype=datetime64[ns] and castable values (e.g. strings) is deprecated. In a future version, these will not be considered matching by isin. Explicitly cast to

electrade\_short

	date_original	hour	fuel_type	volume	full_timestamp	time_idx	month	day_of_week	is_holiday	
0	2021-01-01	0	IGCC	237.628160	2021-01-01 00:00:00	0	1	4	1	
1	2021-01-01	0	LNG	16679.367610	2021-01-01 00:00:00	0	1	4	1	
2	2021-01-01	0	LPG	40.633488	2021-01-01 00:00:00	0	1	4	1	
3	2021-01-01	0	RPS	0.000000	2021-01-01 00:00:00	0	1	4	1	
4	2021-01-01	0	가스압	0.000000	2021-01-01 00:00:00	0	1	4	1	
...	...	...	...	...	...	...	...	...	...	
613219	2023-12-31	23	중유	32.000000	2023-12-31 23:00:00	26279	12	6	0	
613220	2023-12-31	23	태양광	1.000000	2023-12-31 23:00:00	26279	12	6	0	
613221	2023-12-31	23	폐기물	122.000000	2023-12-31 23:00:00	26279	12	6	0	
613222	2023-12-31	23	풍력	337.000000	2023-12-31 23:00:00	26279	12	6	0	
613223	2023-12-31	23	해양에너지	17.000000	2023-12-31 23:00:00	26279	12	6	0	

613224 rows × 9 columns

```
# 모델 설정
max_prediction_length = 24 # 향후 1일(24시간)을 예측 (decoder 길이)
max_encoder_length = 48 # 과거 2일을 참조 (encoder 길이)
training_cutoff = electrade_short.time_idx.max() - max_prediction_length # 학습 데이터 자르는 시점

training = TimeSeriesDataSet(
    electrade_short[lambdax: x.time_idx <= training_cutoff], # 학습용 데이터만 필터링

    # ---기본 설정---
    time_idx = "time_idx",
    target = "volume",
    group_ids= ["fuel_type"],

    # ---시퀀스 길이 설정---
    min_encoder_length = max_encoder_length//2, # 최소 참조 기간(가변적일 수 있음)
    max_encoder_length = max_encoder_length, # 최대 참조 기간
    min_prediction_length = 1, # 최소 예측 기간
    max_prediction_length = max_prediction_length, # 최대 예측 기간

    # ---변수 매핑(중요)---

    # 1. 정적 변수(static)
    static_categoricals = ['fuel_type'],

    # 2. 미래를 아는 변수(known inputs)
    time_varying_known_categoricals = ['hour', 'day_of_week', 'month', 'is_holiday'],
    time_varying_known_reals = ['time_idx'], # time_idx 여기에 포함시키기

    # 3. 과거만 아는 변수
    time_varying_unknown_categoricals = [],
    time_varying_unknown_reals = ['volume'], # 타겟도 포함시켜야함

    # ---전처리 옵션---
    target_normalizer = GroupNormalizer(
        groups=["fuel_type"], transformation="log1p"
    ),
    add_relative_time_idx = True, # 시퀀스 내 상대적 시간 인덱스 자동 추가
    add_target_scales = True, # 타겟의 스케일 정보 추가(정규화 복원 시 필요)
    add_encoder_length = True, # 인코더 길이 정보 추가
)
```

batch\_size = 64 # GPU 메모리에 맞춰 조절

```
# -----
# 1단계: DataLoader 만들기
# -----

# 검증 데이터셋(validation set) 분리: 마지막 시간을 기준으로 나눔
validation = TimeSeriesDataSet.from_dataset(
    training, electrade_short, predict=True,
    stop_randomization=True
)

# 학습용 로더
train_dataloader = training.to_dataloader(
```

```
        train = True,
        batch_size = batch_size,
        num_workers = 11
    )

# 검증용 로더
val_dataloader = validation.to_dataloader(
    train = False,
    batch_size = batch_size * 10,
    num_workers = 11
)
```

```
# baseline
import torch
actuals = torch.cat([y for x, (y, weight) in iter(val_dataloader)])
baseline_predictions = Baseline().predict(val_dataloader)
(actuals - baseline_predictions).abs().mean().item()
```

```
# -----
# 2단계: TFT 모델 생성
# -----

tft = TemporalFusionTransformer.from_dataset(
    training,
    # [학습 파라미터]
    learning_rate = 0.03, # 0.01~0.05 사이로 설정
    hidden_size = 64, # 모델의 크기(데이터가 적으면 16, 많으면 64~128)
    attention_head_size = 1, # 어텐션 헤드 수(보통 1~4)
    dropout = 0.0, # 과적합 방지(0.1~0.3)
    hidden_continuous_size = 8, # 연속형 변수 처리 크기

    # [손실 함수] -> tft는 확률적 예측을 하므로 QuantileLoss 사용
    loss = QuantileLoss(),

    # [로그 및 최적화 설정]
    log_interval = 10,
    optimizer = "Adam", # # tft 저자가 추천함..아니면 torch.optim.Adam
    reduce_on_plateau_patience = 4, # 성능이 안 오르면 학습률을 줄임
)

print(f"모델 파라미터 개수: {tft.size()/1e3:.1f}k")
```

```
# -----
# 3단계: 트레이너(Trainer) 설정 및 학습 시작
# -----

# 과적합 방지: 검증 성능이 10번 연속 좋아지지 않으면 조기 종료
early_stop_callback = EarlyStopping(
    monitor = "val_loss",
    min_delta = 1e-4,
    patience = 5,
    verbose = False,
    mode = "min"
)

lr_logger = pl.callbacks.LearningRateMonitor() # 학습률 변화 기록
tb_logger = TensorBoardLogger("lightning_logs", name="my_tft_model")

trainer = pl.Trainer(
    max_epochs = 3, # 최대 30번 학습(얼리스타핑으로 일찍 끝날 수 있음)
    accelerator = "gpu", # gpu가 있으면 자동 사용, 없으면 cpu 사용
    devices = 1, # 사용할 장치 개수
    # precision = 'bf16-mixed',
    enable_model_summary = True,
    logger = tb_logger,
    gradient_clip_val = 0.1, # 학습 안정화를 위해 기울기 자르기
    callbacks = [lr_logger, early_stop_callback]
)
```

```
# 학습 시작

print("학습을 시작합니다...")
trainer.fit(
    tft,
    train_dataloaders=train_dataloader,
    val_dataloaders=val_dataloader
)
```

```
# Best Model 로드
best_model_path = trainer.checkpoint_callback.best_model_path
best_tft = TemporalFusionTransformer.load_from_checkpoint(best_model_path, weights_only=False)
```

```
# calculate MAE on validation set
actuals = torch.cat([y[0] for x, y in iter(val_dataloader)])
predictions = best_tft.predict(val_dataloader)
(actuals - predictions).abs().mean()
```

```
%load_ext tensorboard

%tensorboard --logdir lightning_logs
```

```
# 예측 시각화

raw_predictions, a,b, x,y = best_tft.predict(val_dataloader, mode="raw", return_x=True)
forname = best_tft.predict(val_dataloader, mode="prediction", return_index=True)
name = forname[2]['fuel_type']
for i in range(25):
    best_tft.plot_prediction(a, raw_predictions, idx=i, add_loss_to_title=False)
    plt.title(f"{name[i]} 24시간 거래 예측값")
    plt.show()
```

일별 총 전력거래량(시각화만)

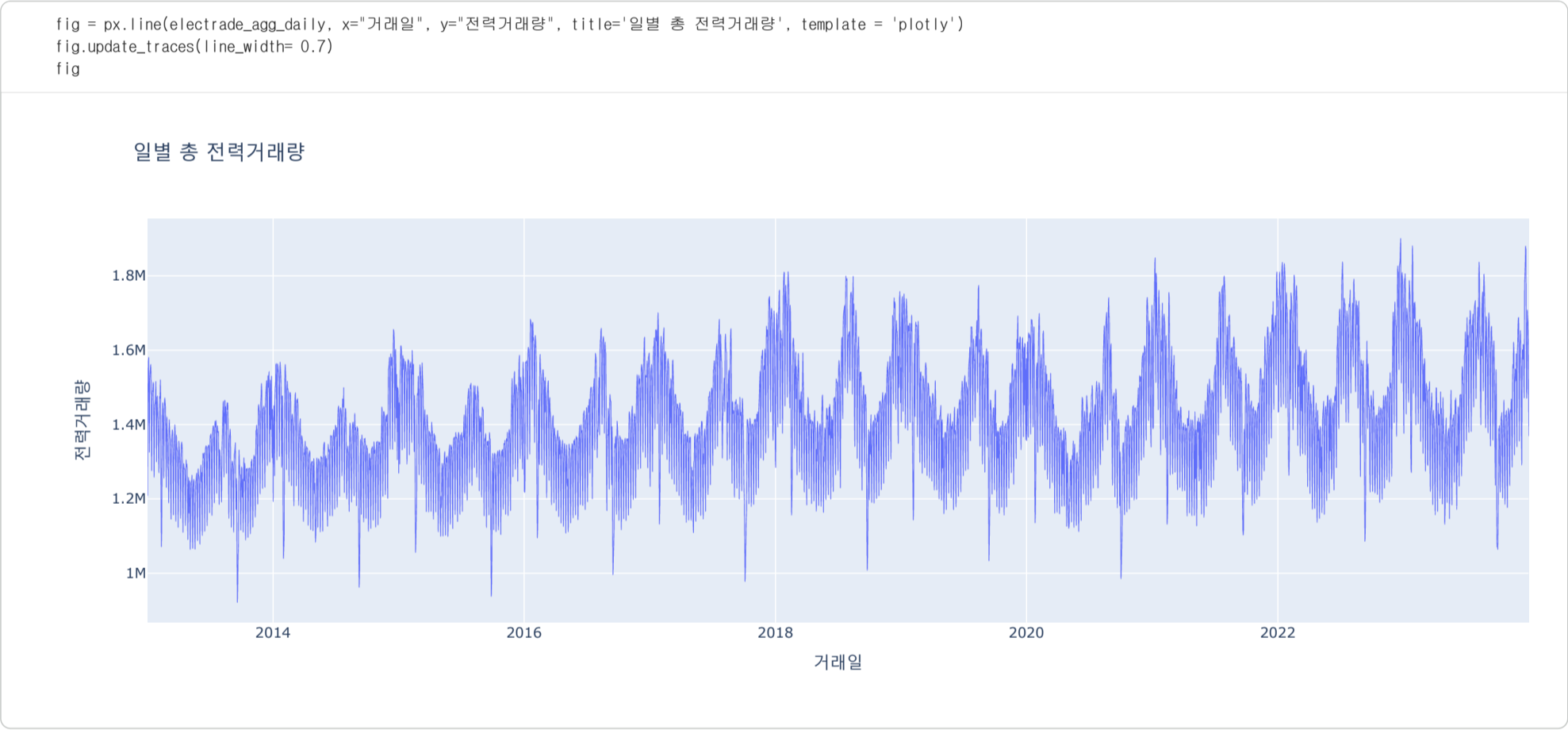
# 일별 총 전력거래량

electrade\_agg\_daily = electrade.groupby('거래일').sum().reset\_index()  
electrade\_agg\_daily.drop(['거래시간', '연료원'], inplace=True,axis=1)  
electrade\_agg\_daily

	거래일	전력거래량	
0	2013-01-01	1.208822e+06	
1	2013-01-02	1.470775e+06	
2	2013-01-03	1.569730e+06	
3	2013-01-04	1.579830e+06	
4	2013-01-05	1.455056e+06	
...	...	...	
4012	2023-12-27	1.685486e+06	
4013	2023-12-28	1.674182e+06	
4014	2023-12-29	1.614603e+06	
4015	2023-12-30	1.457736e+06	
4016	2023-12-31	1.369642e+06	

4017 rows × 2 columns

다음 단계: [electrade\\_agg\\_daily 변수로 코드 생성](#) [New interactive sheet](#)



일별, 원료별 전력거래량

# 일별, 원료별 전력거래량

electrade\_type\_daily = electrade.groupby(['거래일', '연료원']).sum().reset\_index()  
electrade\_type\_daily.drop('거래시간', inplace=True,axis=1)  
electrade\_type\_daily.replace('유연탄', '석탄', inplace = True)  
electrade\_type\_daily.replace('무연탄', '석탄', inplace = True)  
electrade\_type\_daily = electrade\_type\_daily.groupby(['거래일', '연료원']).sum().reset\_index()  
electrade\_type\_daily

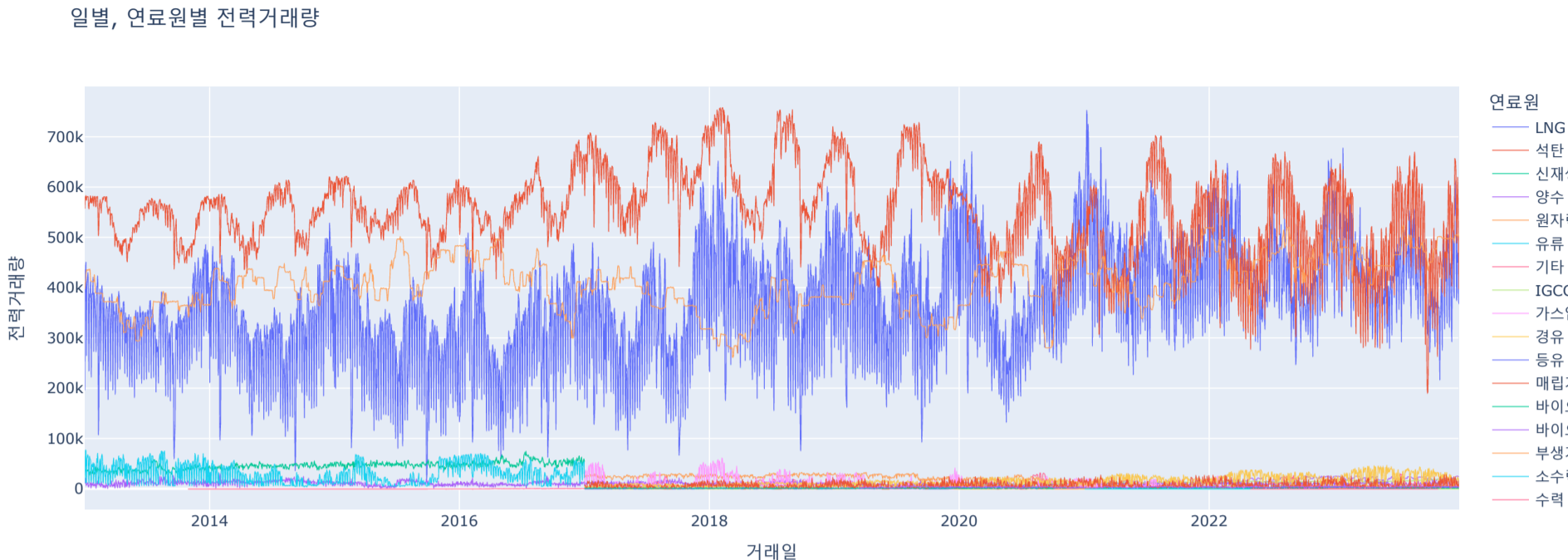
	거래일	연료원	전력거래량	
0	2013-01-01	LNG	196479.142825	
1	2013-01-01	석탄	556980.383040	
2	2013-01-01	신재생	34490.875819	
3	2013-01-01	양수	10953.985007	
4	2013-01-01	원자력	397446.705810	
...	...	...	...	
66523	2023-12-31	중유	740.000000	
66524	2023-12-31	태양광	4721.000000	
66525	2023-12-31	폐기물	2934.000000	
66526	2023-12-31	풍력	16964.000000	
66527	2023-12-31	해양에너지	1315.000000	

66528 rows × 3 columns

다음 단계: [electrade\\_type\\_daily 변수로 코드 생성](#) [New interactive sheet](#)

시각화

```
fig = px.line(electrade_type_daily, x="거래일", y="전력거래량", color='연료원', title='일별, 연료원별 전력거래량', template = 'plotly')
fig.update_traces(line_width= 0.7)
fig
```



모델링

코딩을 시작하거나 AI로 코드를 생성하세요.

○○

```
electrade.loc[electrade['거래시간'] == 24, '거래시간'] = 0 # 자정 거래시간이 00이랑 24로 중복 표기되어있음 -> 0으로 통일
electrade.거래일 = pd.to_datetime(electrade.거래일)

# Combine date and hour to create a full datetime timestamp for accurate time indexing
electrade['full_timestamp'] = electrade['거래일'] + pd.to_timedelta(electrade['거래시간'], unit='h')

# Sort data to ensure correct chronological order within each group (연료원)
electrade = electrade.sort_values(by=['연료원', 'full_timestamp']).reset_index(drop=True)

# Create a numerical time index (integer) as required by TimeSeriesDataSet
# This calculates the number of hours from the earliest timestamp within each fuel_type group.
# We use dt.total_seconds() / 3600 to get hours and then convert to int.
electrade['time_idx_numeric'] = electrade.groupby('연료원')['full_timestamp'].transform(lambda x: (x - x.min()).dt.total_seconds() / 3600).astype(int)

electrade['월'] = electrade.거래일.dt.month.astype(str)
electrade['요일'] = electrade.거래일.dt.weekday.astype(str)
electrade['공휴일여부'] = electrade.거래일.isin(holidays)*1
electrade['공휴일여부'] = electrade['공휴일여부'].astype(str)
electrade['거래시간'] = electrade['거래시간'].astype(str)
# Rename columns to match the desired input for TimeSeriesDataSet and general clarity
electrade.rename(columns={
    '거래일': 'date_original', # Keeping original date for context, if needed
    '거래시간': 'hour',
    '연료원': 'fuel_type',
    '전력거래량': 'volume',
    'time_idx_numeric': 'time_idx' # This is the integer time index for TimeSeriesDataSet
}, inplace=True)

# Drop the 'full_timestamp' column as it's no longer needed after creating 'time_idx'
# And reorder columns for consistency
electrade.columns = ['date_original', 'hour', 'fuel_type', 'volume', 'full_timestamp', 'time_idx', 'month', 'day_of_week', 'is_holiday']

electrade
```

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=use_spl_gen['splym'], y=use_spl_gen['공급량'],mode = 'lines',name='발전용가스'))
fig.add_trace(go.Scatter(x=use_spl_city['splym'], y=use_spl_city['공급량'],mode = 'lines',name='도시가스'))
fig
```

```
# 모델 설정
max_prediction_length = 24 # 향후 7일(168시간)을 예측 (decoder 길이)
max_encoder_length = 48 # 과거 14일을 참조 (encoder 길이)
training_cutoff = electrade.time_idx.max() - max_prediction_length # 학습 데이터 자르는 시점

training = TimeSeriesDataSet(
    electrade[lambdax: x.time_idx <= training_cutoff], # 학습용 데이터만 필터링

    # ---기본 설정---
    time_idx = "time_idx",
    target = "volume",
    group_ids= ["fuel_type"],

    # ---시퀀스 길이 설정---
    min_encoder_length = max_encoder_length//2, # 최소 참조 기간(가변적일 수 있음)
    max_encoder_length = max_encoder_length, # 최대 참조 기간
    min_prediction_length = 1, # 최소 예측 기간
    max_prediction_length = max_prediction_length, # 최대 예측 기간
```

```
# ---변수 매핑(중요)---

# 1. 정적 변수(static)
static_categoricals = ['fuel_type'],

# 2. 미래를 아는 변수(known inputs)
time_varying_known_categoricals = ['hour', 'day_of_week', 'month', 'is_holiday'],
time_varying_known_reals = ['time_idx'], # time_idx 여기에 포함시키기

# 3. 과거만 아는 변수
# time_varying_unknown_categoricals = [],
time_varying_unknown_reals = ['volume'], # 타겟도 포함시켜야함

# ---전처리 옵션---
target_normalizer = GroupNormalizer(
    groups=["fuel_type"], transformation="log1p"
),
add_relative_time_idx = True, # 시퀀스 내 상대적 시간 인덱스 자동 추가
add_target_scales = True,    # 타겟의 스케일 정보 추가(정규화 복원 시 필요)
add_encoder_length = True,    # 인코더 길이 정보 추가
)
```

```
# 실제 사용을 위한 DataLoader 변환
batch_size = 256 # GPU 메모리에 맞춰 조절
```

```
# -----
# 1단계: DataLoader 만들기
# -----

# 검증 데이터셋(validation set) 분리: 마지막 시간을 기준으로 나눔
validation = TimeSeriesDataSet.from_dataset(
    training, electrade, predict=True,
    stop_randomization=True
)

# 학습용 로더
train_dataloader = training.to_dataloader(
    train = True,
    batch_size = batch_size,
    num_workers = 11
)

# 검증용 로더
val_dataloader = validation.to_dataloader(
    train = False,
    batch_size = batch_size * 10,
    num_workers = 11
)
```

```
# -----
# 2단계: TFT 모델 생성
# -----

tft = TemporalFusionTransformer.from_dataset(
    training,
    # [학습 파라미터]
    learning_rate = 0.03, # 0.01~0.05 사이로 설정
    hidden_size = 64, # 모델의 크기(데이터가 적으면 16, 많으면 64~128)
    attention_head_size = 1, # 어텐션 헤드 수(보통 1~4)
    dropout = 0.0, # 과적합 방지(0.1~0.3)
    hidden_continuous_size = 8, # 연속형 변수 처리 크기

    # [손실 함수] -> tft는 확률적 예측을 하므로 QuantileLoss 사용
    loss = QuantileLoss(),

    # [로그 및 최적화 설정]
    log_interval = 10,
    optimizer = "Adam", ## tft 저자가 추천함..아니면 torch.optim.Adam
    reduce_on_plateau_patience = 4, # 성능이 안 오르면 학습률을 줄임
)

print(f"모델 파라미터 개수: {tft.size()}/1e3:.1f}k")
```

```
# -----
# 3단계: 트레이너(Trainer) 설정 및 학습 시작
# -----

# 과적합 방지: 검증 성능이 10번 연속 좋아지지 않으면 조기 종료
early_stop_callback = EarlyStopping(
    monitor = "val_loss",
    min_delta = 1e-4,
    patience = 5,
    verbose = False,
    mode = "min"
)

lr_logger = pl.callbacks.LearningRateMonitor() # 학습률 변화 기록

trainer = pl.Trainer(
    max_epochs = 8, # 최대 30번 학습(알리스타핑으로 일찍 끝날 수 있음)
    accelerator = "gpu", # gpu가 있으면 자동 사용, 없으면 cpu 사용
    devices = 1, # 사용할 장치 개수
    # precision = 'bf16-mixed',
    enable_model_summary = True,
    gradient_clip_val = 0.1, # 학습 안정화를 위해 기울기 자르기
    callbacks = [lr_logger, early_stop_callback]
)
```

```
# 학습 시작
```

```
print("학습을 시작합니다...")
trainer.fit(
    tft,
    train_data loaders=train_data loader,
    val_data loaders=val_data loader
)
```

```
# Best Model 로드
```

```
best_model_path = trainer.checkpoint_callback.best_model_path
best_tft = TemporalFusionTransformer.load_from_checkpoint(best_model_path, weights_only=False)
```

```
# 검증 데이터에 대해 예측 수행(mode="prediction"이면 점추정(quantile 중 0.5, 즉 중앙값 반환). mode="quantiles"면 구간추정(모든 quantile 반환))
raw_predictions, x_decoder, x_encoder = best_tft.predict(
    val_data loader,
    mode = "raw",
    return_x = True
)
```

```
# 결과 시각화
```

```
best_tft.plot_prediction(x_decoder, raw_predictions, idx=0, add_loss_to_title=True)
```

```
# Best Model 로드
```

```
best_model_path = trainer.checkpoint_callback.best_model_path
best_tft = TemporalFusionTransformer.load_from_checkpoint(best_model_path, weights_only=False)
```

```
predictions, index = best_tft.predict(val_data loader, mode='prediction', return_index = True)
results = index.copy()
```

```
results['predicted_volume'] = predictions.tolist()
print(results.head())
```

```
lng_preds = results[results['fuel_type'] == 'LNG']
print('LNG 향후 24시간 거래 예측값:', lng_preds.iloc[0]['predicted_volume'])
```

```
import matplotlib.pyplot as plt
```

```
target_fuel = "LNG"
```

```
best_tft.plot_prediction(
    val_data loader,
    predictions,
    index = index[index['fuel_type']==target_fuel].index[0],
    add_loss_to_title = True
)
plt.title(f"{target_fuel} 24시간 거래 예측값")
plt.show()
```