Pytorch Template

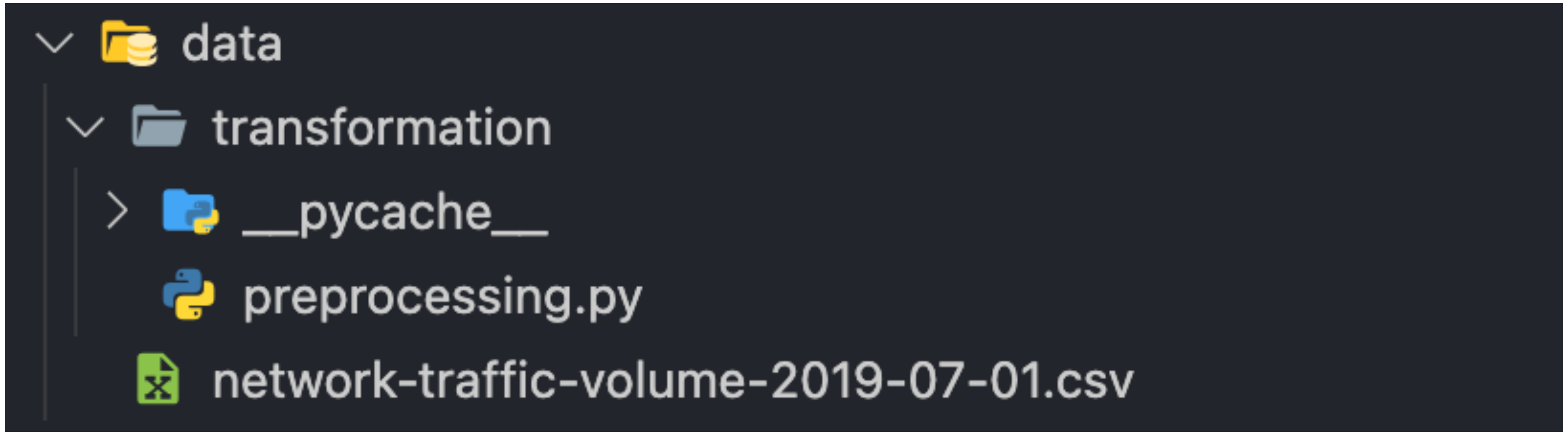
apply to LSTM model

Prepare to use template

- Clone the git repository victoresque/pytorch-template
- Create new LSTM project (new_poject.py)

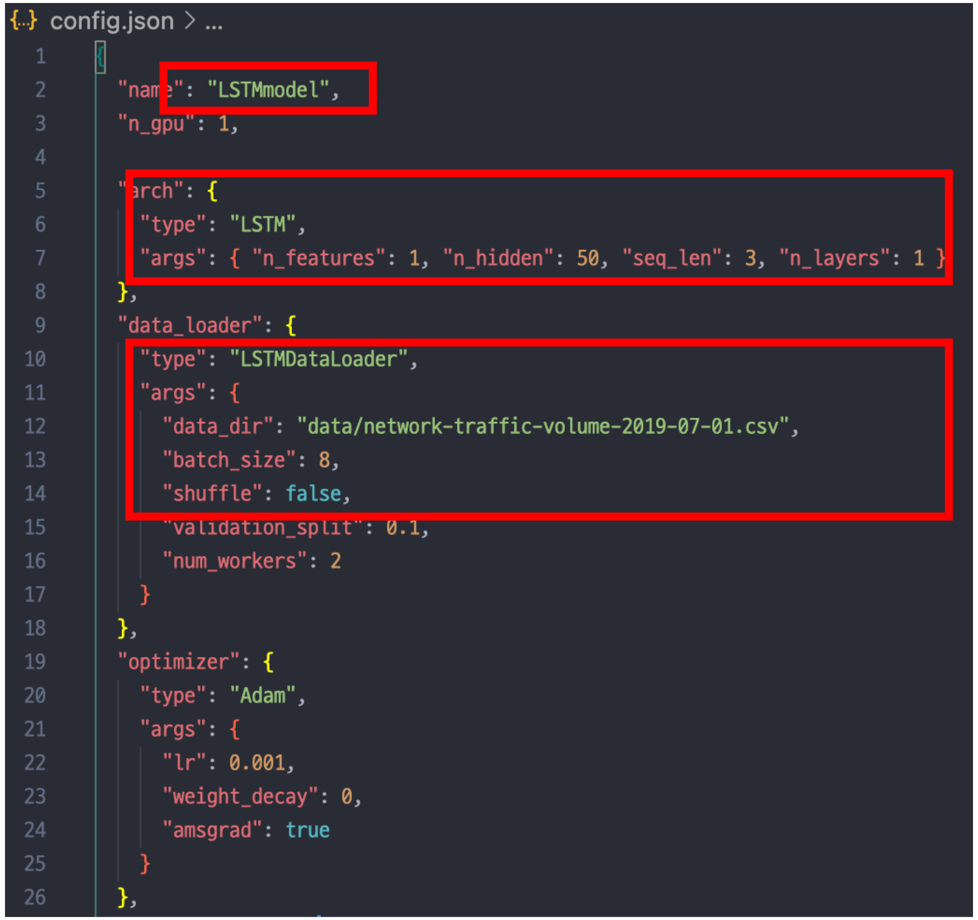
Data folder

- Position our data in the folder
- Create transformaion folder and preprocessing.py

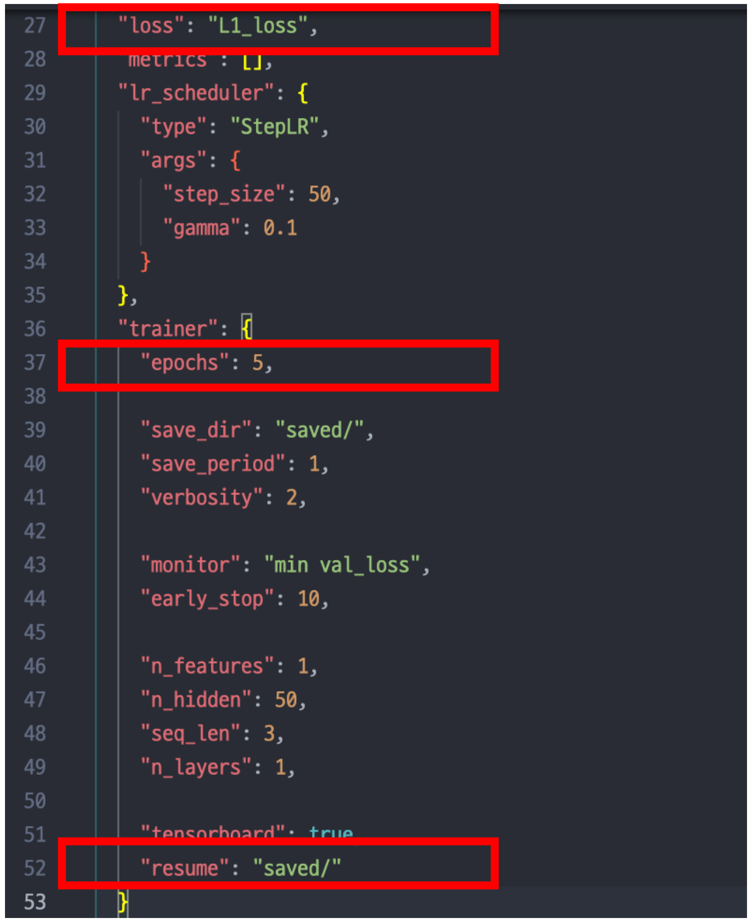


```
data
  transformation
    __pycache__
    preprocessing.py
  network-traffic-volume-2019-07-01.csv
```

Config.json

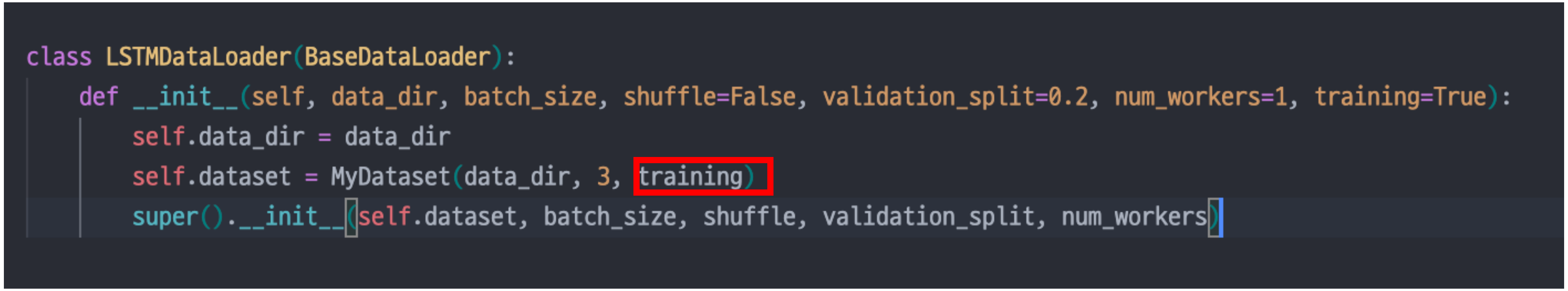


```
{
  "name": "LSTMmodel",
  "n_gpu": 1,

  "arch": {
    "type": "LSTM",
    "args": { "n_features": 1, "n_hidden": 50, "seq_len": 3, "n_layers": 1 }
  },
  "data_loader": {
    "type": "LSTMDataLoader",
    "args": {
      "data_dir": "data/network-traffic-volume-2019-07-01.csv",
      "batch_size": 8,
      "shuffle": false,
      "validation_split": 0.1,
      "num_workers": 2
    }
  },
  "optimizer": {
    "type": "Adam",
    "args": {
      "lr": 0.001,
      "weight_decay": 0,
      "amsgrad": true
    }
  },
```



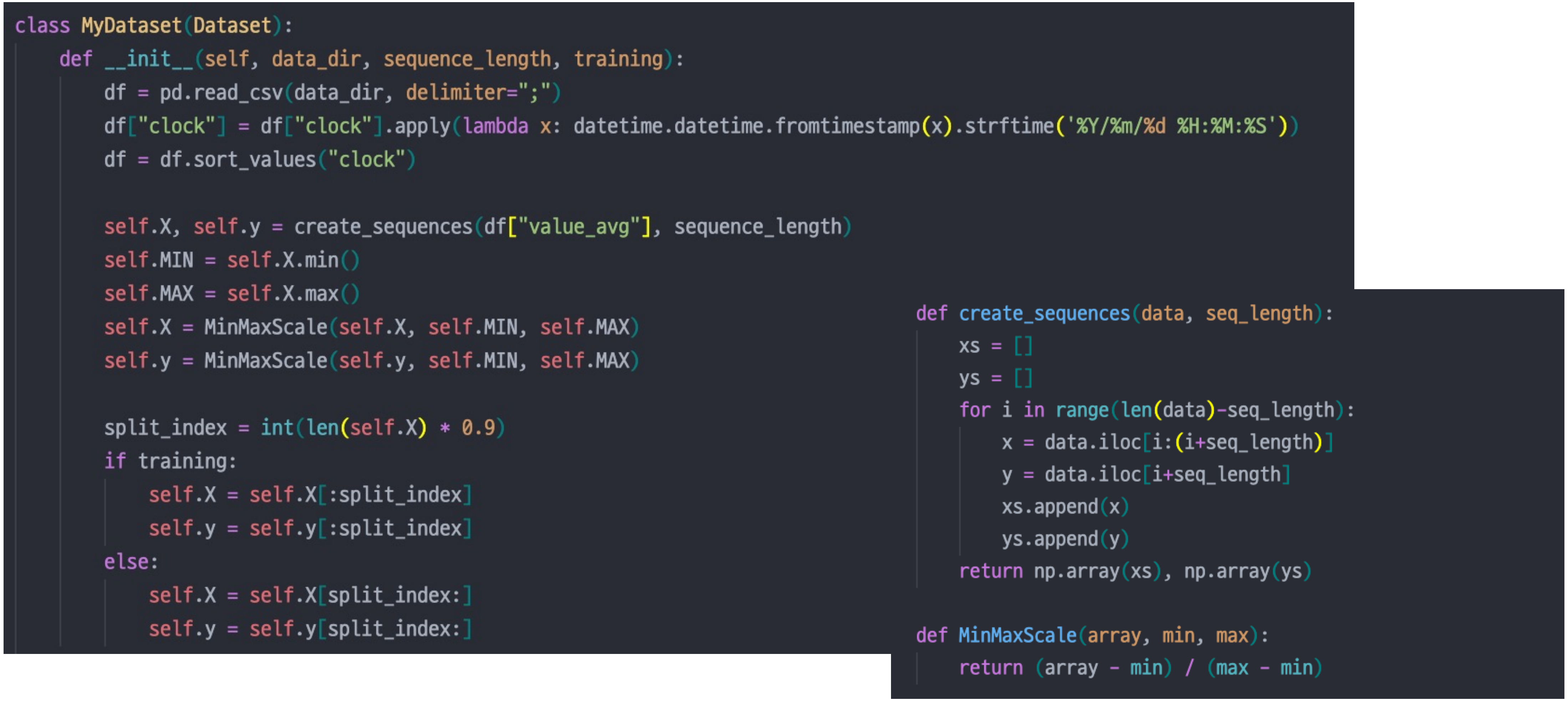
```
  "loss": "L1_loss",
  "metrics": [],
  "lr_scheduler": {
    "type": "StepLR",
    "args": {
      "step_size": 50,
      "gamma": 0.1
    }
  },
  "trainer": {
    "epochs": 5,

    "save_dir": "saved/",
    "save_period": 1,
    "verbosity": 2,

    "monitor": "min val_loss",
    "early_stop": 10,

    "n_features": 1,
    "n_hidden": 50,
    "seq_len": 3,
    "n_layers": 1,

    "tensorboard": true
    "resume": "saved/"
}
```

Data_loaders.py



```python
class LSTMDataLoader(BaseDataLoader):
    def __init__(self, data_dir, batch_size, shuffle=False, validation_split=0.2, num_workers=1, training=True):
        self.data_dir = data_dir
        self.dataset = MyDataset(data_dir, 3, training)
        super().__init__(self.dataset, batch_size, shuffle, validation_split, num_workers)
```

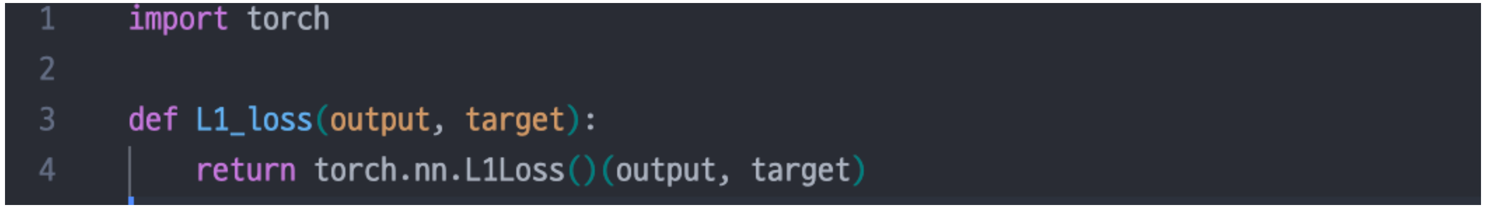
Preprocessing.py



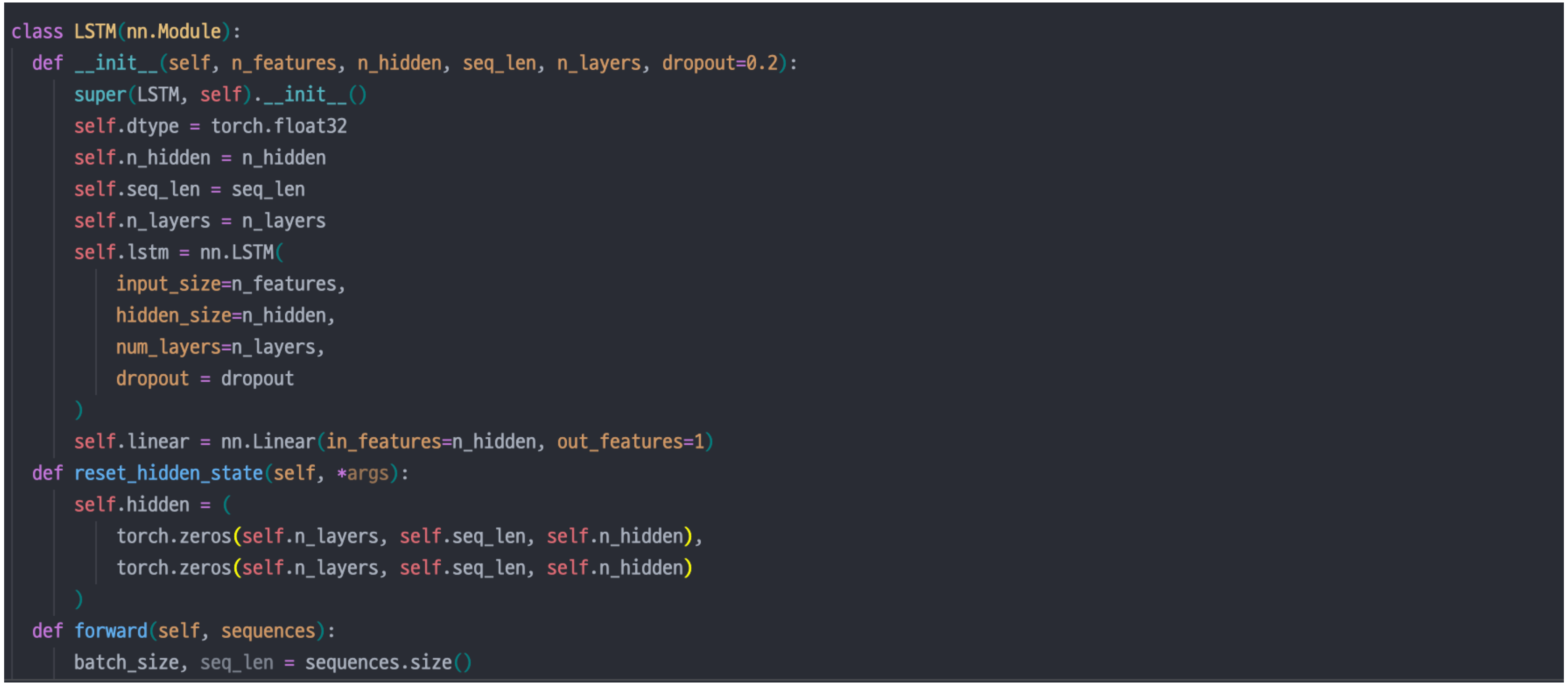
```python
class MyDataset(Dataset):
    def __init__(self, data_dir, sequence_length, training):
        df = pd.read_csv(data_dir, delimiter=";")
        df["clock"] = df["clock"].apply(lambda x: datetime.datetime.fromtimestamp(x).strftime('%Y/%m/%d %H:%M:%S'))
        df = df.sort_values("clock")

        self.X, self.y = create_sequences(df["value_avg"], sequence_length)
        self.MIN = self.X.min()
        self.MAX = self.X.max()
        self.X = MinMaxScale(self.X, self.MIN, self.MAX)
        self.y = MinMaxScale(self.y, self.MIN, self.MAX)

        split_index = int(len(self.X) * 0.9)
        if training:
            self.X = self.X[:split_index]
            self.y = self.y[:split_index]
        else:
            self.X = self.X[split_index:]
            self.y = self.y[split_index:]
```

```python
def create_sequences(data, seq_length):
    xs = []
    ys = []
    for i in range(len(data)-seq_length):
        x = data.iloc[i:(i+seq_length)]
        y = data.iloc[i+seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

def MinMaxScale(array, min, max):
    return (array - min) / (max - min)
```

Loss.py / Model.py

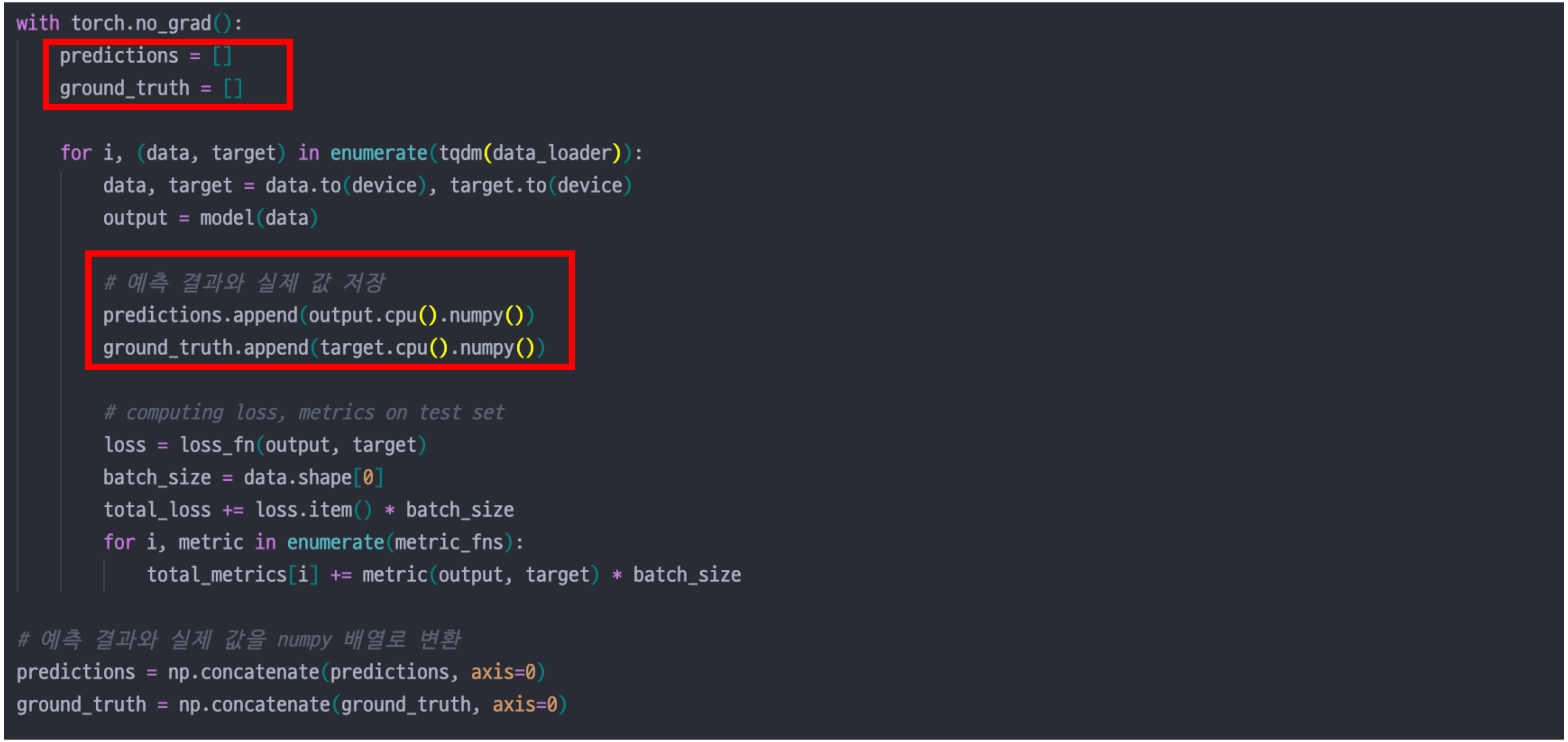


```python
import torch

def L1_loss(output, target):
    return torch.nn.L1Loss()(output, target)
```



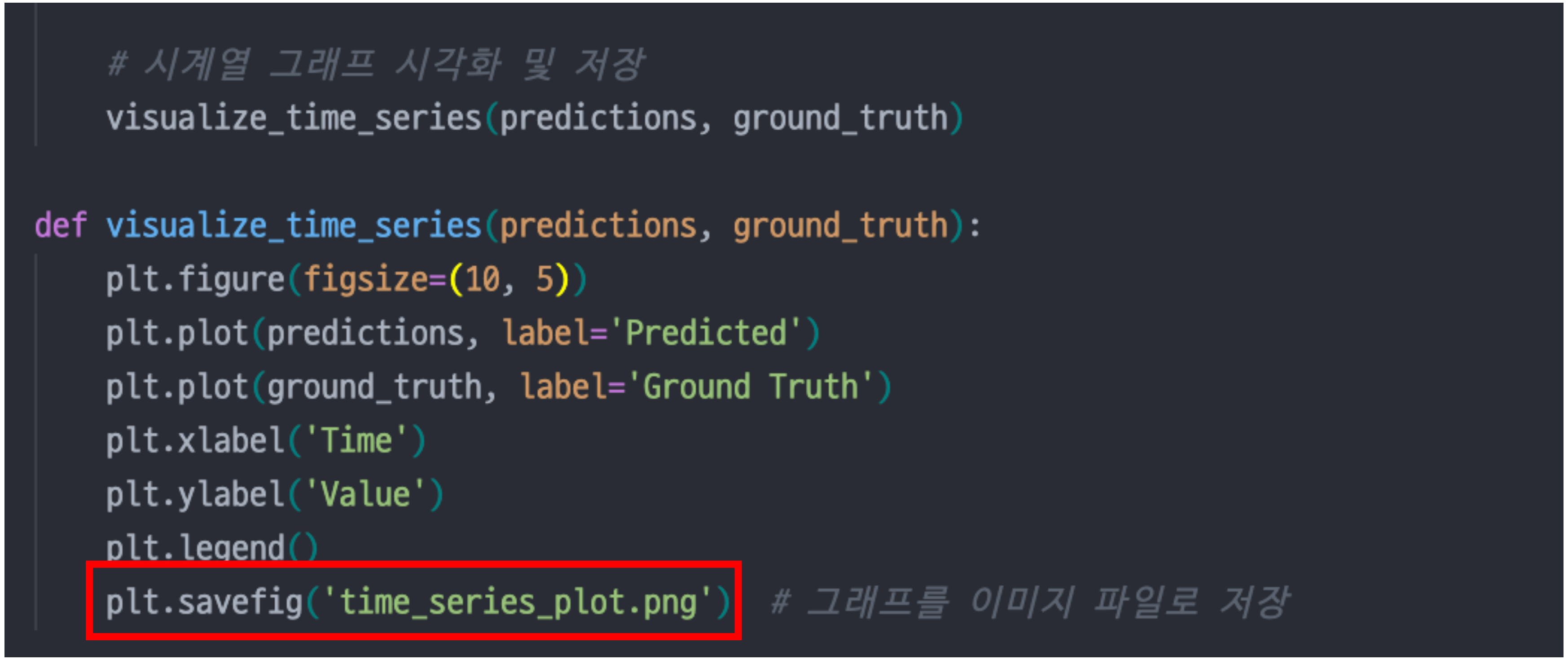
```python
class LSTM(nn.Module):
  def __init__(self, n_features, n_hidden, seq_len, n_layers, dropout=0.2):
      super(LSTM, self).__init__()
      self.dtype = torch.float32
      self.n_hidden = n_hidden
      self.seq_len = seq_len
      self.n_layers = n_layers
      self.lstm = nn.LSTM(
          input_size=n_features,
          hidden_size=n_hidden,
          num_layers=n_layers,
          dropout = dropout
      )
      self.linear = nn.Linear(in_features=n_hidden, out_features=1)
  def reset_hidden_state(self, *args):
      self.hidden = (
          torch.zeros(self.n_layers, self.seq_len, self.n_hidden),
          torch.zeros(self.n_layers, self.seq_len, self.n_hidden)
      )
  def forward(self, sequences):
      batch_size, seq_len = sequences.size()
```

Test.py



```python
with torch.no_grad():
    predictions = []
    ground_truth = []

    for i, (data, target) in enumerate(tqdm(data_loader)):
        data, target = data.to(device), target.to(device)
        output = model(data)

        # 예측 결과와 실제 값 저장
        predictions.append(output.cpu().numpy())
        ground_truth.append(target.cpu().numpy())

        # computing loss, metrics on test set
        loss = loss_fn(output, target)
        batch_size = data.shape[0]
        total_loss += loss.item() * batch_size
        for i, metric in enumerate(metric_fns):
            total_metrics[i] += metric(output, target) * batch_size

# 예측 결과와 실제 값을 numpy 배열로 변환
predictions = np.concatenate(predictions, axis=0)
ground_truth = np.concatenate(ground_truth, axis=0)
```

Test.py



```
    # 시계열 그래프 시각화 및 저장
    visualize_time_series(predictions, ground_truth)

def visualize_time_series(predictions, ground_truth):
    plt.figure(figsize=(10, 5))
    plt.plot(predictions, label='Predicted')
    plt.plot(ground_truth, label='Ground Truth')
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.legend()
    plt.savefig('time_series_plot.png')  # 그래프를 이미지 파일로 저장
```

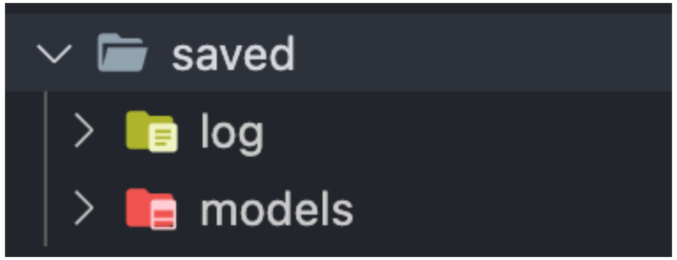
Training & Testing

- Train

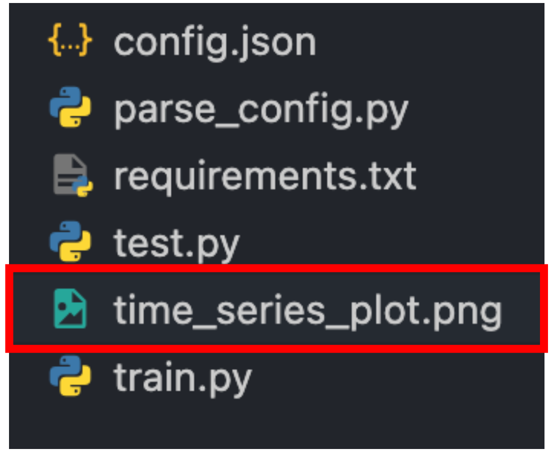
  ```
  ! python train.py -c config.json
  ```

- Test

  ```
  ! python test.py --resume saved/models
  /{date_time}/model_best.pth
  ```

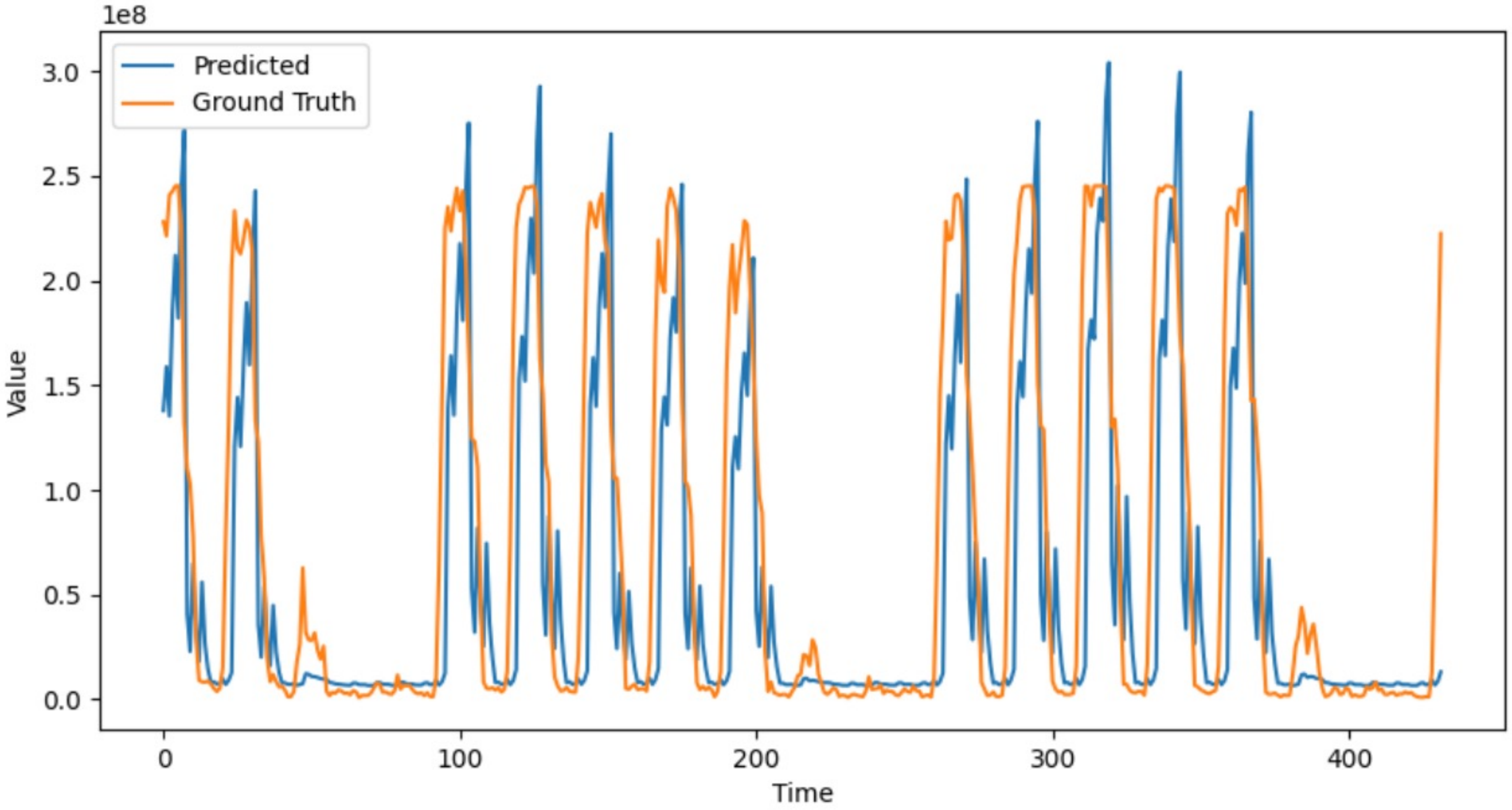


saved
log
models



config.json
parse_config.py
requirements.txt
test.py
time_series_plot.png
train.py

Result



1e8
Predicted
Ground Truth
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Value
0
100
200
300
400
Time