```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
from tqdm import tqdm
import pickle
```

데이터 처리

```
path = '/content/drive/MyDrive/통계적기계학습_팀플/데이터/대여이력정보/1~6'
forder = os.listdir(path)

data1 = pd.DataFrame()
for files in tqdm(forder):
    data_ = pd.read_csv(path+'/'+files, encoding='cp949')
    data1 = pd.concat([data1, data_])

path = '/content/drive/MyDrive/통계적기계학습_팀플/데이터/대여이력정보/7~12'
forder = os.listdir(path)

data2 = pd.DataFrame()
for files in tqdm(forder):
    data_ = pd.read_csv(path+'/'+files, encoding='cp949')
    data2 = pd.concat([data2, data_])
```

```
100%|██████████| 6/6 [01:15<00:00, 12.52s/it]
100%|██████████| 6/6 [01:58<00:00, 19.83s/it]
```

```
use = ['대여일시', '대여 대여소번호', '반납일시', '반납대여소번호', '이용시간', '이용거리']

# 강서구 대여소 번호
gangseogu_num = pd.read_excel('/content/drive/MyDrive/통계적기계학습_팀플/데이터/강서구대여소번호.xlsx', engine='openpyxl', header=None)
gangseogu_num = list(gangseogu_num[0].values)

data1 = data1[use]
data1 = data1[data1['대여 대여소번호'].isin(gangseogu_num)]
data2 = data2[['대여일시', '대여 대여소번호', '반납일시', '반납대여소번호', '이용시간(분)', '이용거리(M)']]
data2.columns = use
data2 = data2[data2['대여 대여소번호'].isin(gangseogu_num)]

data = pd.concat([data1, data2])


data['대여일시'] = data['대여일시'].apply(lambda x: x[:13])
data['반납일시'] = data['반납일시'].apply(lambda x: x[:13])

error_cnt = 0
def convert(x):
  global error_cnt
  try:
    x = str(int(x))
    return x
  except:
    pass

data['반납대여소번호'] = data['반납대여소번호'].apply(convert)
data['대여 대여소번호'] = data['대여 대여소번호'].apply(convert)


import pickle
with open('/content/drive/MyDrive/통계적기계학습_팀플/데이터/강서구_대여이력정보.pickle', 'wb') as f:
  pickle.dump(data, f)


from datetime import datetime
import pandas

def date_range(start, end):
    start = datetime.strptime(start, "%Y-%m-%d")
    end = datetime.strptime(end, "%Y-%m-%d")
    dates = []
    for date in pandas.date_range(start, periods=(end-start).days+1):
      date = date.strftime("%Y-%m-%d")
      for i in range(24):
        i = str(i)
        if int(i) < 10: i = '0' + i
        dates.append(date + ' ' + i)
```

```
    return dates

dates = date_range("2022-01-01", "2022-12-31")


rental_grouped = data.groupby(data['대여 대여소번호'])
return_grouped = data.groupby(data['반납대여소번호'])

final_data_dict = {}

for num in tqdm(gangseogu_num):
  try:
    rental_data = rental_grouped.get_group(str(num))
    return_data = return_grouped.get_group(str(num))

    rental_data_ = rental_data.groupby('대여일시')
    rental_data_dict = {}
    for d in rental_data_:
      time_df = d[1] # 특정 시간에서 데이터
      col_0 = time_df.iloc[0]['대여일시']
      col_1 = len(time_df)
      col_2 = time_df['이용거리'].sum()
      col_3 = time_df['이용시간'].sum()
      rental_data_dict[col_0] = [col_0, col_1, col_2, col_3]


    return_data_ = return_data.groupby('반납일시')
    return_data_dict = {}
    for d in return_data_:
      time_df = d[1]
      col_0 = time_df.iloc[0]['반납일시']
      col_1 = len(time_df)
      col_2 = time_df['이용거리'].sum()
      col_3 = time_df['이용시간'].sum()
      return_data_dict[col_0] = [col_0, col_1, col_2, col_3]

    new_data = []
    rental_data_keys = rental_data_dict.keys()
    return_data_keys = return_data_dict.keys()


    for t in dates:
      if t in rental_data_keys and t in return_data_keys:
        new_data.append(rental_data_dict[t] + return_data_dict[t][1:])
      elif t in rental_data_keys and t not in return_data_keys:
        new_data.append(rental_data_dict[t] + [0, 0, 0])
      elif t not in rental_data_keys and t in return_data_keys:
        new_data.append([t] + [0, 0, 0] + return_data_dict[t][1:])
      elif t not in rental_data_keys and t not in return_data_keys:
        new_data.append([t] + [0, 0, 0, 0, 0, 0])

    new_data = pd.DataFrame(
        new_data,
        columns=['시간대', '대여', '대여_이용거리', '대여_이용시간', '반납', '반납_이용거리', '반납_이용시간']
        )
    new_data['반납-대여'] = new_data['반납'] - new_data['대여']
    final_data_dict[str(num)] = new_data
  except Exception as e:
    print(e)
```

```
 72%|███████   | 136/188 [09:04<03:06,  3.59s/it]'3779'
100%|██████████| 188/188 [11:41<00:00,  3.73s/it]
```

모델학습

```
import torch
from torch.utils.data import Dataset, DataLoader, ConcatDataset, random_split
import torch.nn as nn

import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from tqdm import tqdm

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda:0
```

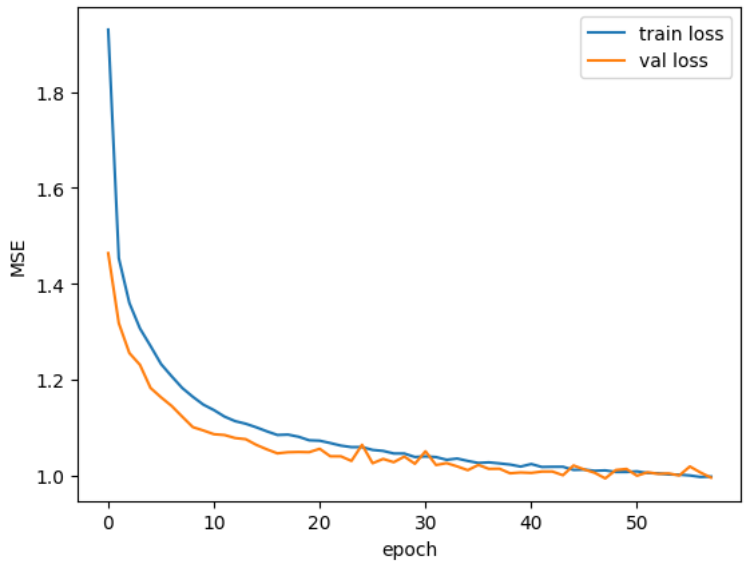
```
# 날씨(기온, 습도), 미세먼지(미세먼지, 초미세먼지) 데이터

weather = pd.read_csv("/content/drive/MyDrive/통계적기계학습_팀플/데이터/weather_2022_01_01to2022_12_31.csv")

for i in weather.index:
  w = weather.loc[i, 'Time']
  if w[-2:] != '00':
    weather.drop(i, axis=0, inplace=True)

weather = weather.reset_index(drop=True)

weather = weather[['Time', 'Temperature', 'Rel. humidity']]
weather['Temperature'] = weather['Temperature'].apply(lambda x: int(x[:-2]))
weather['Rel. humidity'] = weather['Rel. humidity'].apply(lambda x: int(x[:-1]))
weather['Time'] = weather['Time'].apply(lambda x: x[:2])

time = []
for t in range(24):
  if t < 10:
    time.append('0'+str(t))
  else:
    time.append(str(t))

time *= 365
i = 0
while i < 24*365:
  w = weather.loc[i, 'Time']
  if w != time[i]:
    temp = (weather.loc[i, 'Temperature'] + weather.loc[i+1, 'Temperature']) / 2
    humid = (weather.loc[i, 'Rel. humidity'] + weather.loc[i+1, 'Rel. humidity']) / 2
    weather.loc[i+0.5] = [time[i], temp, humid]
    weather = weather.sort_index()
    weather = weather.reset_index(drop=True)
    i+=1
  i+=1

weather = weather.drop('Time', axis=1)

dust = pd.read_csv("/content/drive/MyDrive/통계적기계학습_팀플/데이터/강서구 대기질 자료 제공_2022.csv")
dust = dust[['미세먼지(PM10)', '초미세먼지(PM2.5)']]


import pickle
with open('/content/drive/MyDrive/통계적기계학습_팀플/데이터/강서구_시간대여소별_대여이력정보.pickle', 'rb') as f:
  data = pickle.load(f)


class TimeSeriesDataset(Dataset):
  def __init__(self, data, scaler, window_size):
    self.scaler = scaler
    if self.scaler:
      data = self.scaler.transform(data)
    else:
      self.scaler = StandardScaler()
      data = self.scaler.fit_transform(data)

    x, y= [], []

    for i in range(len(data) - window_size):
      x.append(data[i:i+window_size])
      y.append(data[i+window_size, 0])

    self.x = torch.tensor(x, dtype=torch.float32, device=device)
    self.y = torch.tensor(y, dtype=torch.float32, device=device)

  def __len__(self):
    return len(self.x)

  def __getitem__(self, idx):
    return self.x[idx], self.y[idx]


class LSTMModel(nn.Module):
  def __init__(self, input_size, hidden_size, num_layers, output_size, dropout):
    super().__init__()

    self.num_layers = num_layers
    self.hidden_size = hidden_size

    self.lstm = nn.LSTM(input_size, hidden_size, num_layers, dropout=dropout, batch_first=True)
    self.linear = nn.Linear(hidden_size, output_size)
```

```
    def forward(self, x):
      h_0 = torch.zeros(self.num_layers, x.shape[0], self.hidden_size).to(device)
      c_0 = torch.zeros(self.num_layers, x.shape[0], self.hidden_size).to(device)

      x, _ = self.lstm(x, (h_0, c_0))
      x = self.linear(x[:, -1])
      return x


  class Trainer:
    def __init__(self, model, train_loader, val_loader):
      self.model = model
      self.train_loader = train_loader
      self.val_loader = val_loader

      self.loss = nn.MSELoss().to(device)
      self.optimizer = torch.optim.Adam(self.model.parameters(), lr=lr)

      self.loss_history = {'train': [], 'val': []}

    def train(self):
      min_val_loss = None
      cnt = 0

      for epoch in range(total_epochs):
        print(f'Epoch: {epoch:02}')

        train_loss = self.train_step()
        print(f'\tTrain Loss: {train_loss:.5f}')

        val_loss = self.val_step()
        print(f'\tVal Loss: {val_loss:.5f}')

        self.loss_history['train'].append(train_loss)
        self.loss_history['val'].append(val_loss)

        if min_val_loss is None or val_loss < min_val_loss:
          min_val_loss = val_loss
          cnt = 0
          torch.save(self.model.state_dict(), "/content/drive/MyDrive/통계적기계학습_팀플/lstm_all_features_minmaxscaler.pt")
        else:
          cnt += 1
          print(f'patiece: {cnt}')
          if cnt >= 10:
            print('Early stopped!')
            break

    def train_step(self):
      self.model.train()

      epoch_loss = 0
      bar = tqdm(self.train_loader)

      for batch in bar:
        self.optimizer.zero_grad()

        x, y = batch
        outputs = self.model(x)

        loss = self.loss(outputs.squeeze(), y)

        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        bar.set_description(f'Train Loss: {loss.item():.5f}')
        epoch_loss += loss.item()

      epoch_loss /= len(self.train_loader)
      return epoch_loss

    def val_step(self):
      self.model.eval()

      epoch_loss = 0
      bar = tqdm(self.val_loader)

      for batch in bar:
        x, y = batch
        outputs = self.model(x)

        loss = self.loss(outputs.squeeze(), y)
```

```
        bar.set_description(f'Val Loss: {loss.item():.5f}')
        epoch_loss += loss.item()

    epoch_loss /= len(self.val_loader)
    return epoch_loss


use_features = ['대여', '대여_이용거리', '대여_이용시간', '반납', '반납_이용거리', '반납_이용시간', '반납-대여']

window_size = 4  # 연속된 window_size개의 데이터를 가지고 다음 값 예측
input_size = len(use_features) + 4  # input 데이터 차원
hidden_size = 36  # lstm hidden state 차원
num_layers = 2  # lstm 쌓은 층
output_size = 1  # output 차원
dropout = 0.2
lr = 0.003
total_epochs = 500
batch_size = 10000
train_val_ratio = 0.9


# create dataset with each place and concat it
testdata_dict = {}
dataset_dict = {}
scaler = MinMaxScaler()
for k, d in tqdm(data.items()):
  d = pd.concat([d[use_features], weather, dust], axis=1)
  testdata_dict[k] = d.iloc[-24*7:, :].reset_index(drop=True) # last week is test data
  scaler.fit(d.iloc[:-24*7, :])

for k, d in tqdm(data.items()):
  d = pd.concat([d[use_features], weather, dust], axis=1)
  dataset_dict[k] = TimeSeriesDataset(d.iloc[:-24*7, :], scaler=scaler, window_size=window_size)

dataset = ConcatDataset(list(dataset_dict.values()))

      100%|██████████| 187/187 [00:00<00:00, 270.24it/s]
      100%|██████████| 187/187 [00:17<00:00, 10.87it/s]


train_dataset, val_dataset = random_split(dataset, [train_val_ratio, 1-train_val_ratio])
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

model = LSTMModel(input_size, hidden_size, num_layers, output_size, dropout)
model.to(device)

trainer = Trainer(model=model, train_loader=train_loader, val_loader=val_loader)

trainer.train()
```

```
Train Loss: 1.12729: 100%|██████████| 145/145 [00:28<00:00,  5.14it/s]
        Train Loss: 1.00125
Val Loss: 0.85342: 100%|██████████| 17/17 [00:03<00:00,  5.55it/s]
        Val Loss: 0.99953
patiece: 7
Epoch: 55
Train Loss: 0.97464: 100%|██████████| 145/145 [00:28<00:00,  5.18it/s]
        Train Loss: 0.99961
Val Loss: 1.08811: 100%|██████████| 17/17 [00:03<00:00,  5.64it/s]
        Val Loss: 1.01834
patiece: 8
Epoch: 56
Train Loss: 0.89190: 100%|██████████| 145/145 [00:27<00:00,  5.21it/s]
        Train Loss: 0.99642
Val Loss: 0.90966: 100%|██████████| 17/17 [00:03<00:00,  5.50it/s]
        Val Loss: 1.00577
patiece: 9
Epoch: 57
Train Loss: 0.94570: 100%|██████████| 145/145 [00:27<00:00,  5.23it/s]
        Train Loss: 0.99714
Val Loss: 0.78082: 100%|██████████| 17/17 [00:02<00:00,  5.69it/s]      Val Loss: 0.99497
patiece: 10
Early stopped!
```

```
import matplotlib.pyplot as plt

plt.plot(range(len(trainer.loss_history['train'])), trainer.loss_history['train'], label='train loss')
plt.plot(range(len(trainer.loss_history['train'])), trainer.loss_history['val'], label='val loss')
plt.xlabel('epoch')
plt.ylabel('MSE')
plt.legend()
plt.show()
```

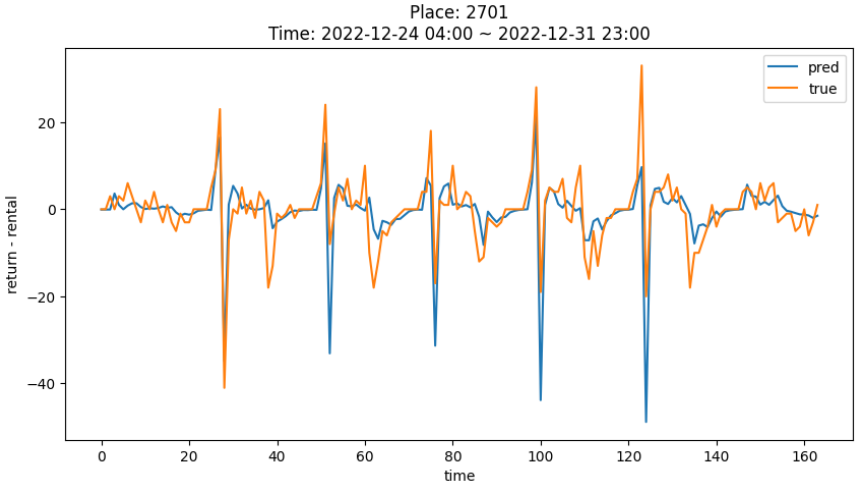


Test

```
testdataset_dict = {}
for k, v in testdata_dict.items():
  testdataset_dict[k] = TimeSeriesDataset(v, scaler=scaler, window_size=window_size)


def predict(model, rental_place):
  model.eval()
  pred = model(testdataset_dict[rental_place].x)

  pred_with_dummy = torch.cat([torch.zeros((164, 6)), pred.to('cpu'), torch.zeros((164, 4))], axis=1).detach().numpy()

  pred = testdataset_dict[rental_place].scaler.inverse_transform(pred_with_dummy)[:, 6]

  return pred

model = LSTMModel(input_size, hidden_size, num_layers, output_size, dropout)
model.load_state_dict(torch.load('/content/drive/MyDrive/통계적기계학습_팀플/lstm_all_features_minmax.pt'))
model.to(device)

target_place = '2701'

pred = predict(model, target_place)
```

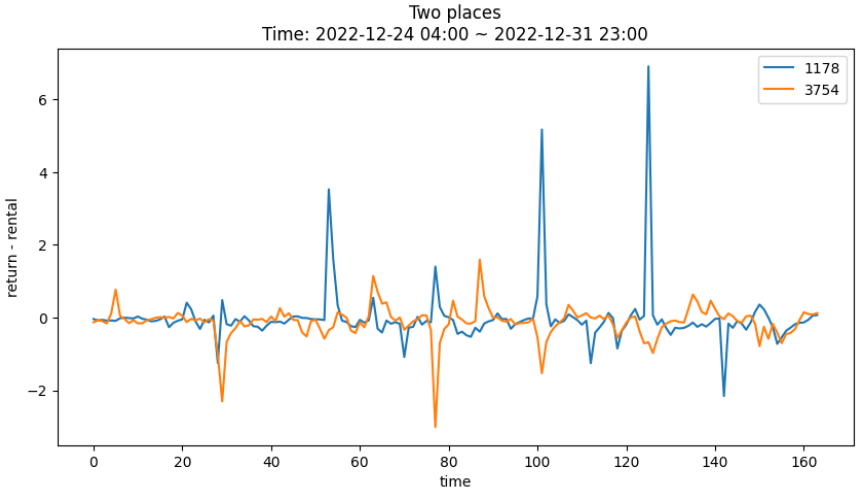
```
plt.figure(figsize=(10, 5))
plt.title(f'Place: {target_place}₩nTime: 2022-12-24 04:00 ~ 2022-12-31 23:00')
plt.xlabel('time')
plt.ylabel('return - rental')
plt.plot(pred, label='pred')
plt.plot(testdata_dict[target_place].loc[4:, '반납-대여'].values, label='true')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7eff305759c0>
```




```
pred1 = predict(model, '1178')
pred2 = predict(model, '3754')

plt.figure(figsize=(10,5))
plt.title('Two places₩nTime: 2022-12-24 04:00 ~ 2022-12-31 23:00')
plt.xlabel('time')
plt.ylabel('return - rental')
plt.plot(pred1, label='1178')
plt.plot(pred2, label='3754')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7eff65f9c520>
```



✓ 1초 오후 1:55에 완료됨