종합실습2_1 따릉이

- 지금까지 배운 것을 총 복습 합니다.
- Data : 서울 공유 자전거
- 문제 : 2시간 후의 수요를 예측하고자 한다.

- OO시에서 에이블러 여러분에게 의뢰가 들어왔습니다.
- **공유 자전거 운영팀**에서는 공유자전거가 부족한 지역과 남는 지역에 대해서 판단하기 원합니다.
- 2시간 전에 **공유자전거 수요량**을 예측할 수 있다면, 이동시켜서 남거나 부족한 문제를 해결할수 있다고 합니다.

1.환경준비

(1) 라이브러리 로딩

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import *
from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential
from keras.layers import Dense
from keras.backend import clear_session
from keras.optimizers import Adam
```

- 함수 만들기

```
# 학습곡선 함수
def dl_history_plot(history):
    plt.figure(figsize=(10,6))
    plt.plot(history['loss'], label='train_err', marker = '.')
    plt.plot(history['val_loss'], label='val_err', marker = '.')

    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()
    plt.grid()
    plt.show()
```

(2) 데이터로딩

```
path = 'https://raw.githubusercontent.com/DA4BAM/dataset/master/SeoulBikeData2.csv'
data = pd.read_csv(path)
data['DateTime'] = pd.to_datetime(data['DateTime'])
data.drop(['Visibility','Solar'], axis = 1, inplace = True)
data.head(10)
```

	DateTime	Count	Temperature	Humidity	WindSpeed	Rainfall	Snowfall	Seasons	Holiday	Fun
0	2017-12-01 00:00:00	254	-5.2	37	2.2	0.0	0.0	Winter	No Holiday	
1	2017-12-01 01:00:00	204	-5.5	38	0.8	0.0	0.0	Winter	No Holiday	
2	2017-12-01 02:00:00	173	-6.0	39	1.0	0.0	0.0	Winter	No Holiday	
3	2017-12-01 03:00:00	107	-6.2	40	0.9	0.0	0.0	Winter	No Holiday	
4	2017-12-01	78	-6.0	36	2.3	0.0	0.0	Winter	No	

Next steps: Generate code with data View recommended plots

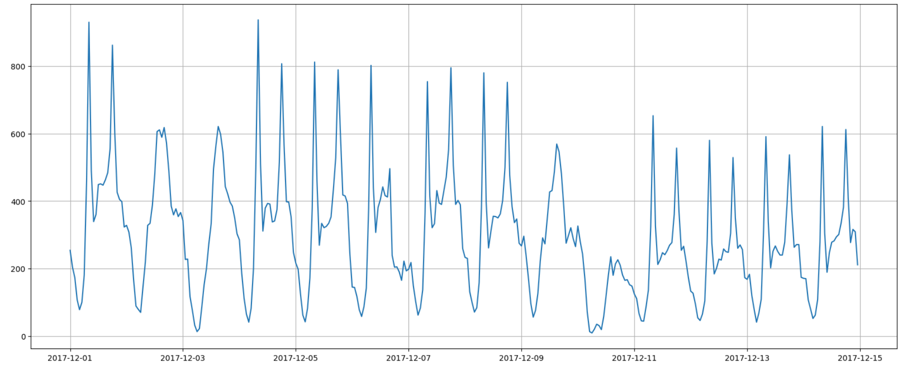
변수설명

- DateTime : year-month-day hh:mi:ss
- Count : 시간대별 수요량
- Temperature : 온도(섭씨)
- Humidity : 습도(%)
- WindSpeed : 풍속(m/s)
- Rainfall - mm
- Snowfall - cm
- Seasons - Winter, Spring, Summer, Autumn
- Holiday - Holiday / No holiday
- FuncDay - Yes / No

```
# 데이터 기간은 다음과 같습니다.
data.DateTime.min(), data.DateTime.max()
```

```
(Timestamp('2017-12-01 00:00:00'), Timestamp('2018-11-30 23:00:00'))
```

```
# 14일 동안의 수요량을 살펴 봅시다.
temp = data[:24*14]
plt.figure(figsize = (20,8))
plt.plot('DateTime', 'Count', data = temp)
plt.grid()
plt.show()
```



2.데이터 준비

(1) y 만들기

- 2시간 이후의 수요량을 예측해야 합니다.

```
data['y'] = data['Count'].shift(-2)
data.head()
```

	DateTime	Count	Temperature	Humidity	WindSpeed	Rainfall	Snowfall	Seasons	Holiday	Fun
0	2017-12-01 00:00:00	254	-5.2	37	2.2	0.0	0.0	Winter	No Holiday	
1	2017-12-01 01:00:00	204	-5.5	38	0.8	0.0	0.0	Winter	No Holiday	

Next steps: Generate code with data View recommended plots

```
# 2칸을 앞당겼기 때문에 하위 2행의 y값에 NaN이 표시되어 있습니다.
data.tail()
```

	DateTime	Count	Temperature	Humidity	WindSpeed	Rainfall	Snowfall	Seasons	Holiday
8755	2018-11-30 19:00:00	1003	4.2	34	2.6	0.0	0.0	Autumn	No Holiday
8756	2018-11-30 20:00:00	764	3.4	37	2.3	0.0	0.0	Autumn	No Holiday

```
# 하위 2행은 삭제합니다.
# 하위 2행 제외하고 다시 붙기
data = data.iloc[:-2]
```

(2) 데이터 정리

- 불필요한 변수 제거 : DateTime
- x, y 나누기

```
target = 'y'
x = data.drop(target, axis = 1)
y = data.loc[:, target]
```

```
# 날짜 데이터 제거
x.drop('DateTime', axis = 1, inplace = True)
x.head()
```

	Count	Temperature	Humidity	WindSpeed	Rainfall	Snowfall	Seasons	Holiday	FuncDay
0	254	-5.2	37	2.2	0.0	0.0	Winter	No Holiday	Yes
1	204	-5.5	38	0.8	0.0	0.0	Winter	No Holiday	Yes
2	173	-6.0	39	1.0	0.0	0.0	Winter	No Holiday	Yes

Next steps: Generate code with x View recommended plots

(3) NaN 조치

(4) 가변수화

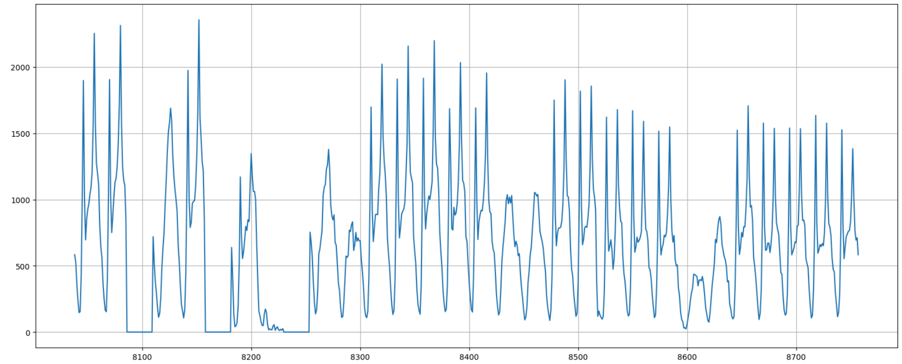
```
cat_cols = ['Seasons','Holiday','FuncDay']
x = pd.get_dummies(x, columns = cat_cols, drop_first = True)
```

(5) 데이터분할 : train : val

- 시계열 데이터이므로 시간의 흐름에 맞게 분할합시다.
    - 뒤에서 30일 : validaiton
    - 나머지 : train
    - 30일 : 시간단위 데이터이므로 24 * 30
- train_test_split : shuffle(뒤섞기) 옵션을 False로 하면 저장된 순서대로 자릅니다!

```
i = 30 * 24
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size = i, shuffle = False)
```

```
plt.figure(figsize = (20,8))
plt.plot(y_val)
plt.grid()
plt.show()
```



(6) Scaling

```
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)
```

3.모델링

- 히든레이어를 추가한 모델 두 개 이상을 생성한 후
- 성능을 비교하시오.
- 성능을 높이기 위해서 조절할 것들
  - 히든레이어 수
  - 히든레이어 노드수
  - epochs 수
  - learning_rate : 0.1 ~ 0.0001 사이에서 조정(예 Adam(learning_rate = 0.01))

(1) 모델1

```
nfeature = x_train.shape[1]
nfeature
```

```
11
```

```
clear_session()
model1 = Sequential([Dense(5, input_shape=(nfeature,), activation='relu'),
                     Dense(1)])
model1.summary()
```

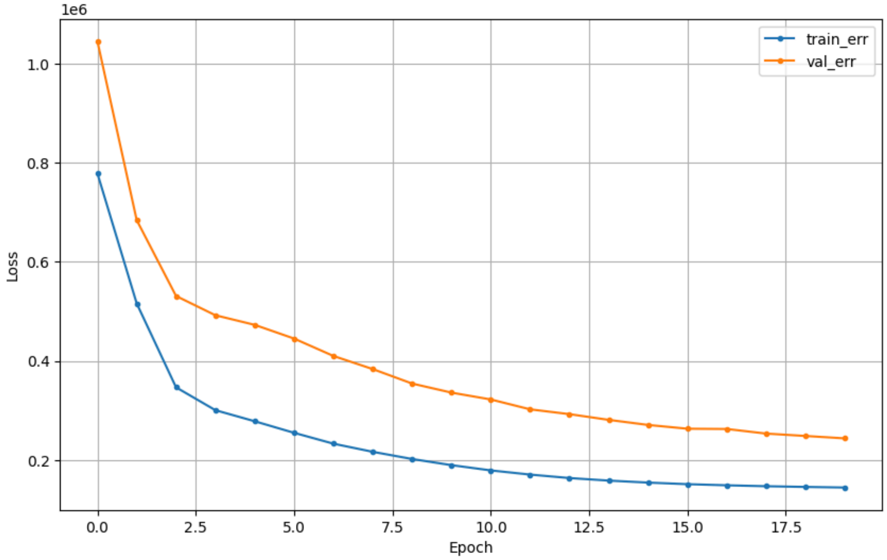
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 5)                 60

 dense_1 (Dense)             (None, 1)                 6
```

```
=================================================================
Total params: 66 (264.00 Byte)
Trainable params: 66 (264.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model1.compile(optimizer=Adam(0.01), loss='mse')
hist = model1.fit(x_train, y_train, epochs=20, validation_split=0.2).history
```

```
Epoch 1/20
201/201 [==============================] - 1s 3ms/step - loss: 779017.0000 - val_loss: 1046191.8125
Epoch 2/20
201/201 [==============================] - 1s 3ms/step - loss: 515960.5000 - val_loss: 684352.0625
Epoch 3/20
201/201 [==============================] - 1s 3ms/step - loss: 346456.4688 - val_loss: 531024.5625
Epoch 4/20
201/201 [==============================] - 0s 2ms/step - loss: 300263.3438 - val_loss: 492116.2500
Epoch 5/20
201/201 [==============================] - 0s 2ms/step - loss: 277862.7500 - val_loss: 472678.6875
Epoch 6/20
201/201 [==============================] - 0s 2ms/step - loss: 254529.0000 - val_loss: 445012.8438
Epoch 7/20
201/201 [==============================] - 1s 3ms/step - loss: 232573.6875 - val_loss: 409857.4688
Epoch 8/20
201/201 [==============================] - 1s 3ms/step - loss: 216120.7500 - val_loss: 383336.9062
Epoch 9/20
201/201 [==============================] - 1s 3ms/step - loss: 201618.3281 - val_loss: 354091.1562
Epoch 10/20
201/201 [==============================] - 1s 3ms/step - loss: 189206.6406 - val_loss: 335800.1875
Epoch 11/20
201/201 [==============================] - 1s 3ms/step - loss: 178559.2031 - val_loss: 321930.3750
Epoch 12/20
201/201 [==============================] - 0s 2ms/step - loss: 170119.7500 - val_loss: 301989.5312
Epoch 13/20
201/201 [==============================] - 0s 2ms/step - loss: 163135.4531 - val_loss: 292263.0312
Epoch 14/20
201/201 [==============================] - 0s 2ms/step - loss: 157962.9062 - val_loss: 280683.3750
Epoch 15/20
201/201 [==============================] - 0s 2ms/step - loss: 153933.0312 - val_loss: 270423.5000
Epoch 16/20
201/201 [==============================] - 0s 2ms/step - loss: 150733.4531 - val_loss: 262810.5625
Epoch 17/20
201/201 [==============================] - 0s 2ms/step - loss: 148469.1719 - val_loss: 262275.4375
Epoch 18/20
201/201 [==============================] - 0s 2ms/step - loss: 146575.0156 - val_loss: 252978.0312
Epoch 19/20
201/201 [==============================] - 0s 2ms/step - loss: 145165.2031 - val_loss: 248145.6875
Epoch 20/20
201/201 [==============================] - 0s 2ms/step - loss: 143924.1719 - val_loss: 243292.7344
```

```
dl_history_plot(hist)
```




```
pred = model1.predict(x_val)
```

```
23/23 [==============================] - 0s 1ms/step
```

```
print('RMSE:', mean_squared_error(y_val, pred))
print('MAE:', mean_absolute_error(y_val, pred))
print('MAPE:', mean_absolute_error(y_val, pred))
```

```
RMSE: 140893.1517956811
MAE: 252.517930677202
MAPE: 252.517930677202
```

코딩을 시작하거나 AI로 코드를 생성하세요.

(2) 모델2

```
clear_session()
model1 = Sequential([Dense(5, input_shape=(nfeature,), activation='relu'),
                     Dense(1)])
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 5)                 60

 dense_1 (Dense)             (None, 1)                 6

=================================================================
Total params: 66 (264.00 Byte)
Trainable params: 66 (264.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model1.compile(optimizer=Adam(0.01), loss='mse')
hist = model1.fit(x_train, y_train, epochs=50, validation_split=0.2).history
```

```
Epoch 25/50
201/201 [==============================] - 0s 2ms/step - loss: 138838.7188 - val_loss: 251029.5000
Epoch 26/50
201/201 [==============================] - 0s 2ms/step - loss: 138373.7969 - val_loss: 249263.2812
Epoch 27/50
201/201 [==============================] - 0s 2ms/step - loss: 137988.3750 - val_loss: 250561.9688
Epoch 28/50
201/201 [==============================] - 0s 2ms/step - loss: 137549.7969 - val_loss: 248817.0312
Epoch 29/50
201/201 [==============================] - 0s 2ms/step - loss: 137208.0469 - val_loss: 247192.8125
Epoch 30/50
201/201 [==============================] - 0s 2ms/step - loss: 137039.6562 - val_loss: 247646.8125
Epoch 31/50
201/201 [==============================] - 1s 2ms/step - loss: 136624.2344 - val_loss: 239406.0938
Epoch 32/50
201/201 [==============================] - 0s 2ms/step - loss: 136267.5938 - val_loss: 242659.5000
Epoch 33/50
201/201 [==============================] - 0s 2ms/step - loss: 135648.7344 - val_loss: 242946.4062
Epoch 34/50
201/201 [==============================] - 0s 2ms/step - loss: 135124.8594 - val_loss: 238030.8594
Epoch 35/50
201/201 [==============================] - 0s 2ms/step - loss: 134630.1875 - val_loss: 238949.3594
Epoch 36/50
201/201 [==============================] - 0s 2ms/step - loss: 134134.5000 - val_loss: 238801.0156
Epoch 37/50
201/201 [==============================] - 0s 2ms/step - loss: 133857.3594 - val_loss: 237318.1562
Epoch 38/50
201/201 [==============================] - 0s 2ms/step - loss: 133513.6094 - val_loss: 239043.2031
Epoch 39/50
201/201 [==============================] - 0s 2ms/step - loss: 133193.6406 - val_loss: 236826.1094
Epoch 40/50
201/201 [==============================] - 0s 2ms/step - loss: 133031.9062 - val_loss: 231592.8594
Epoch 41/50
201/201 [==============================] - 1s 3ms/step - loss: 132827.7344 - val_loss: 234429.2344
Epoch 42/50
201/201 [==============================] - 1s 3ms/step - loss: 132720.9844 - val_loss: 235419.7812
Epoch 43/50
201/201 [==============================] - 1s 3ms/step - loss: 132506.7812 - val_loss: 229710.5938
Epoch 44/50
201/201 [==============================] - 1s 4ms/step - loss: 132351.2188 - val_loss: 232192.5000
Epoch 45/50
201/201 [==============================] - 0s 2ms/step - loss: 132295.2812 - val_loss: 236861.9062
Epoch 46/50
201/201 [==============================] - 0s 2ms/step - loss: 131999.1094 - val_loss: 224222.3281
Epoch 47/50
201/201 [==============================] - 0s 2ms/step - loss: 132040.4375 - val_loss: 228974.9688
Epoch 48/50
201/201 [==============================] - 0s 2ms/step - loss: 131943.3125 - val_loss: 233416.5000
Epoch 49/50
201/201 [==============================] - 0s 2ms/step - loss: 131825.0625 - val_loss: 232616.6406
Epoch 50/50
201/201 [==============================] - 0s 2ms/step - loss: 131682.1719 - val_loss: 232294.8594
```

```
dl_history_plot(hist)
```