실제 데이터로 만들어 보는 모델

+ 코드 + 텍스트

1. 데이터 파악하기

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd
import numpy as np

# 집 값 데이터를 불러옵니다.
df = pd.read_csv("./data/house_train.csv")

# 데이터를 미리 살펴보겠습니다.
df
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	Po
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	
				...	...	...	...	...	...	...	...	...	...
			RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...	0	
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...	0	
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...	0	
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...	0	
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...	0	

1460 rows × 81 columns

저장이 완료되었습니다.

```
# 데이터가 어떤 유형으로 이루어져 있는지 알아봅니다.
df.dtypes
```

```
Id               int64
MSSubClass       int64
MSZoning        object
LotFrontage    float64
LotArea          int64
                ...
MoSold           int64
YrSold           int64
SaleType        object
SaleCondition   object
SalePrice        int64
Length: 81, dtype: object
```

2. 결측치, 카테고리 변수 처리하기

```
# 속성별로 결측치가 몇 개인지 확인합니다.
df.isnull().sum().sort_values(ascending=False).head(20)
```

```
PoolQC          1453
MiscFeature     1406
Alley           1369
Fence           1179
FireplaceQu      690
LotFrontage      259
GarageYrBlt       81
GarageCond        81
GarageType        81
GarageFinish      81
GarageQual        81
BsmtFinType2      38
BsmtExposure      38
BsmtQual          37
BsmtCond          37
BsmtFinType1      37
MasVnrArea         8
MasVnrType         8
Electrical         1
Id                 0
dtype: int64
```

```
# 카테고리형 변수를 0과 1로 이루어진 변수로 바꾸어 줍니다.(12장 3절)
df = pd.get_dummies(df)

# 결측치를 전체 칼럼의 평균으로 대체하여 채워줍니다.
df = df.fillna(df.mean())

# 업데이트된 데이터 프레임을 출력해봅니다.
df
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuil
0	1	60	65.0	8450	7	5	20(
1	2	20	80.0	9600	6	8	197
			68.0	11250	7	5	20(
			60.0	9550	7	5	19
4	5	60	84.0	14260	8	5	20(
...	...	...	...	...	...	...	
1455	1456	60	62.0	7917	6	5	199
1456	1457	20	85.0	13175	6	6	197
1457	1458	70	66.0	9042	7	9	194
1458	1459	20	68.0	9717	5	6	195
1459	1460	20	75.0	9937	5	6	196

1460 rows × 290 columns

저장이 완료되었습니다.

3. 속성별 관련도 추출하기

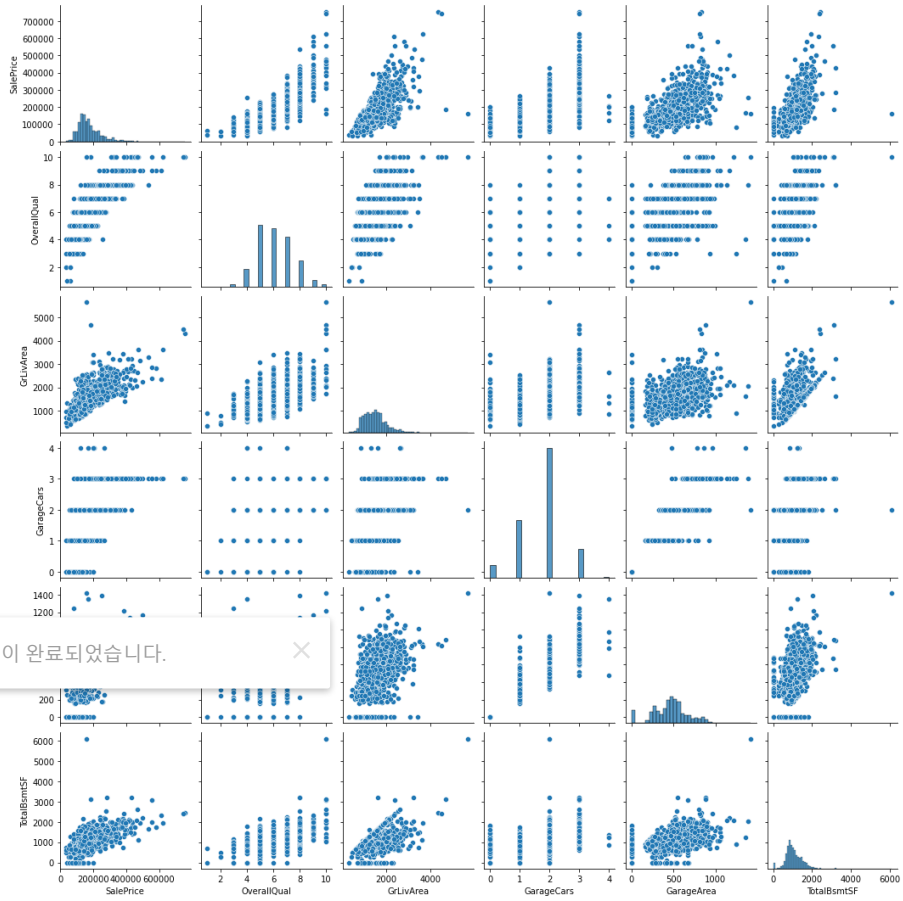
```
# 데이터 사이의 상관 관계를 저장합니다.
df_corr=df.corr()

# 집 값과 관련이 큰 것부터 순서대로 저장합니다.
df_corr_sort=df_corr.sort_values('SalePrice', ascending=False)

# 집 값과 관련도가 가장 큰 10개의 속성들을 출력합니다.
df_corr_sort['SalePrice'].head(10)
```

```
SalePrice       1.000000
OverallQual     0.790982
GrLivArea       0.708624
GarageCars      0.640409
GarageArea      0.623431
TotalBsmtSF     0.613581
1stFlrSF        0.605852
FullBath        0.560664
BsmtQual_Ex     0.553105
TotRmsAbvGrd    0.533723
Name: SalePrice, dtype: float64
```

```
# 집 값과 관련도가 가장 높은 속성들을 추출해서 상관도 그래프를 그려봅니다.
cols=['SalePrice','OverallQual','GrLivArea','GarageCars','GarageArea','TotalBsmtSF']
sns.pairplot(df[cols])
plt.show();
```



4. 주택 가격 예측 모델

```
# 집 값을 제외한 나머지 열을 저장합니다.
cols_train=['OverallQual','GrLivArea','GarageCars','GarageArea','TotalBsmtSF']
X_train_pre = df[cols_train]

# 집 값을 저장합니다.
y = df['SalePrice'].values

# 전체의 80%를 학습셋으로, 20%를 테스트셋으로 지정합니다.
X_train, X_test, y_train, y_test = train_test_split(X_train_pre, y, test_size=0.2)

# 모델의 구조를 설정합니다.
model = Sequential()
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(40, activation='relu'))
model.add(Dense(1))
model.summary()

# 모델을 실행합니다.
model.compile(optimizer ='adam', loss = 'mean_squared_error')

# 20회 이상 결과가 향상되지 않으면 자동으로 중단되게끔 합니다.
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=20)

# 모델의 이름을 정합니다.
modelpath="./data/model/Ch15-house.hdf5"

# 최적화 모델을 업데이트하고 저장합니다.
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=0, save_best_only=Tr

# 실행 관련 설정을 하는 부분입니다. 전체의 20%를 검증셋으로 설정합니다.
                , y_train, validation_split=0.25, epochs=2000, batch_size=32, callbacks=
```

```
Epoch 132/2000
28/28 [==============================] - 0s 4ms/step - loss: 1857990656.0000 - val_loss: 2118029440.0000
Epoch 133/2000
28/28 [==============================] - 0s 4ms/step - loss: 1862326144.0000 - val_loss: 2129511808.0000
Epoch 134/2000
28/28 [==============================] - 0s 4ms/step - loss: 1865215744.0000 - val_loss: 2144991488.0000
Epoch 135/2000
28/28 [==============================] - 0s 5ms/step - loss: 1858451840.0000 - val_loss: 2125956480.0000
Epoch 136/2000
28/28 [==============================] - 0s 4ms/step - loss: 1861348736.0000 - val_loss: 2140236032.0000
Epoch 137/2000
28/28 [==============================] - 0s 4ms/step - loss: 1855917824.0000 - val_loss: 2143077504.0000
Epoch 138/2000
28/28 [==============================] - 0s 5ms/step - loss: 1856772992.0000 - val_loss: 2121010944.0000
Epoch 139/2000
28/28 [==============================] - 0s 4ms/step - loss: 1857664896.0000 - val_loss: 2182798336.0000
Epoch 140/2000
28/28 [==============================] - 0s 4ms/step - loss: 1863239552.0000 - val_loss: 2129678336.0000
Epoch 141/2000
28/28 [==============================] - 0s 4ms/step - loss: 1858560128.0000 - val_loss: 2139776640.0000
```

```
# 예측 값과 실제 값, 실행 번호가 들어갈 빈 리스트를 만듭니다.
real_prices =[]
pred_prices = []
X_num = []

# 25개의 샘플을 뽑아 실제 값, 예측 값을 출력해 봅니다.
n_iter = 0
Y_prediction = model.predict(X_test).flatten()
for i in range(25):
    real = y_test[i]
    prediction = Y_prediction[i]
    print("실제가격: {:.2f}, 예상가격: {:.2f}".format(real, prediction))
    real_prices.append(real)
    pred_prices.append(prediction)
    n_iter = n_iter + 1
    X_num.append(n_iter)
```

```
10/10 [==============================] - 0s 2ms/step
실제가격: 204000.00, 예상가격: 0573.16
실제가격: 143000.00, 예상가격: 26.98
실제가격: 149350.00, 예상가격: 161655.14
실제가격: 204000.00, 예상가격: 186832.89
실제가격: 159950.00, 예상가격: 160576.06
실제가격: 143000.00, 예상가격: 150491.38
실제가격: 90350.00, 예상가격: 94809.02
실제가격: 98000.00, 예상가격: 97495.51
실제가격: 320000.00, 예상가격: 282381.22
실제가격: 151000.00, 예상가격: 158318.52
실제가격: 174900.00, 예상가격: 171337.38
실제가격: 194500.00, 예상가격: 193063.95
실제가격: 151400.00, 예상가격: 248888.86
실제가격: 180500.00, 예상가격: 193406.44
실제가격: 120500.00, 예상가격: 153792.59
실제가격: 285000.00, 예상가격: 246126.97
실제가격: 93000.00, 예상가격: 147456.83
실제가격: 301500.00, 예상가격: 283128.03
실제가격: 257000.00, 예상가격: 261037.66
실제가격: 147000.00, 예상가격: 157420.62
실제가격: 153337.00, 예상가격: 175792.61
실제가격: 202665.00, 예상가격: 194942.06
실제가격: 339750.00, 예상가격: 249999.86
실제가격: 190000.00, 예상가격: 204532.78
실제가격: 82000.00, 예상가격: 113534.34
```



```
# 그래프를 통해 샘플로 뽑은 25개의 값을 비교해 봅니다.

plt.plot(X_num, pred_prices, label='predicted price')
plt.plot(X_num, real_prices, label='real price')
plt.legend()
plt.show()
```

0초 오후 3:40에 완료됨

저장이 완료되었습니다.