



넷째마당

딥러닝 기본기 다지기

15장 실제 데이터로 만들어 보는 모델

- 1 데이터 파악하기
- 2 결측치, 카테고리 변수 처리하기
- 3 속성별 관련도 추출하기
- 4 주택 가격 예측 모델

실제 데이터로 만들어 보는 모델

- 실제 데이터로 만들어 보는 모델



실제 데이터로 만들어 보는 모델

● 실제 데이터로 만들어 보는 모델

- 지금까지 한 실습은 참 또는 거짓을 맞히거나 여러 개의 보기 중 하나를 예측하는 분류 문제
- 이번에는 수치를 예측하는 문제
- 준비된 데이터는 아이오와주 에임스 지역에서 2006년부터 2010년까지 거래된 실제 부동산 판매 기록
- 주거 유형, 차고, 자재 및 환경에 관한 80개의 서로 다른 속성을 이용해 집의 가격을 예측해 볼 예정
- 오랜 시간 사람이 일일이 기록하다 보니 빠진 부분도 많고, 집에 따라 어떤 항목은 범위에서 너무 벗어나 있기도 하며, 또 가격과는 관계가 없는 정보가 포함되어 있기도 함
- 실제 현장에서 만나게 되는 이런 류의 데이터를 어떻게 다루어야 하는지 이장에서 학습해 보자

실제 데이터로 만들어 보는 모델

- 총 80개 특성(feature) :
- <https://nanometre380.tistory.com/12>,
- <https://www.kaggle.com/c/house-prices-advanced-regression-techniques>
 - MSSubClass : 매매와 관련된 주거 타입 식별
 - MSZoning : 매매의 일반적인 지역 분류
 - LotFrontage : 부동산과 연결된 도로의 선형 피트
 - LotArea : 평방피트 단위의 부지 크기
 - Street : 부동산으로의 도로(street) 접근 유형
 - Alley : 부동산으로의 골목(alley) 접근 유형
 - LotShape : 부동산의 일반적인 모양
 - LandContour : 부동산의 평탄도
 - Utilities : 사용 가능한 유틸리티(수도전기가스) 유형
 - LotConfig : 지역 구성
 - LandSlope : 부동산의 경사
 - Neighborhood : 에임스 시 경계 내의 물리적인 위치
 - Condition1 : 주요 도로 및 철도와의 근접성
 - Condition2 : 주요 도로 및 철도와의 근접성 (두 개 이상 존재하는 경우)



1 데이터 파악하기

1 데이터 파악하기

- 데이터 파악하기

- 먼저 데이터를 불러와 확인해 보자

```
import pandas as pd

# 깃허브에 준비된 데이터를 가져옵니다.
!git clone https://github.com/taehojo/data.git

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



1 데이터 파악하기

- 데이터 파악하기

- 데이터를 미리 살펴보자

```
df
```


1 데이터 파악하기

● 데이터 파악하기

실행 결과

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...
...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...

1 데이터 파악하기

● 데이터 파악하기

PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000
...
0	NaN	NaN	NaN	0	8	2007	WD	Normal	175000
0	NaN	MnPrv	NaN	0	2	2010	WD	Normal	210000
0	NaN	GdPrv	Shed	2500	5	2010	WD	Normal	266500
0	NaN	NaN	NaN	0	4	2010	WD	Normal	142125
0	NaN	NaN	NaN	0	6	2008	WD	Normal	147500

1460 rows × 81 columns



1 데이터 파악하기

- 데이터 파악하기

- 총 80개의 속성으로 이루어져 있고 마지막 열이 우리의 타깃인 집 값(SalePrice)
- 모두 1,460개의 샘플이 들어 있음

1 데이터 파악하기

- 데이터 파악하기

- 이제 각 데이터가 어떤 유형으로 되어 있는지 알아보자

```
df.dtypes
```

1 데이터 파악하기

● 데이터 파악하기

실행 결과

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



1 데이터 파악하기

- 데이터 파악하기

- 정수형(int64)과 실수형(float64), 그리고 오브젝트형(object) 등 여러 유형이 있음을 알 수 있음



2 결측치, 카테고리 변수 처리하기

2 결측치, 카테고리 변수 처리하기

- 결측치, 카테고리 변수 처리하기

- 앞 장에서 다루었던 데이터와 차이점은 아직 전처리가 끝나지 않은 상태의 데이터라 측정 값이 없는 결측치가 있다는 것
- 결측치가 있는지 알아보는 함수는 isnull()
- 결측치가 모두 몇 개인지 세어 가장 많은 것부터 순서대로 나열한 후 처음 20개만 출력하는 코드는 다음과 같음

```
df.isnull().sum().sort_values(ascending=False).head(20)
```


2 결측치, 카테고리 변수 처리하기

- 결측치, 카테고리 변수 처리하기

실행 결과

PoolQC	1453
MiscFeature	1406
Alley	1369
Fence	1179
FireplaceQu	690
LotFrontage	259
GarageYrBlt	81
GarageCond	81
GarageType	81
GarageFinish	81



2 결측치, 카테고리 변수 처리하기

- 결측치, 카테고리 변수 처리하기

```
GarageQual      81
BsmtFinType2    38
BsmtExposure    38
BsmtQual        37
BsmtCond        37
BsmtFinType1    37
MasVnrArea      8
MasVnrType      8
Electrical      1
Id              0
dtype: int64
```

2 결측치, 카테고리 변수 처리하기

- 결측치, 카테고리 변수 처리하기

- 결측치가 많은 항목은 1,460개의 샘플 중에서 1,453개나 비어 있을 만큼 빠진 곳이 많은 것을 확인할 수 있음

2 결측치, 카테고리 변수 처리하기

- 결측치, 카테고리 변수 처리하기

- 이제 모델을 만들기 위해 데이터를 전처리하겠음
- 먼저 12.3절에서 소개되었던 판다스의 `get_dummies()` 함수를 이용해 카테고리형 변수를 0과 1로 이루어진 변수로 바꾸어 줌

```
df = pd.get_dummies(df)
```

2 결측치, 카테고리 변수 처리하기

- 결측치, 카테고리 변수 처리하기

- 결측치를 채워 줌
- 결측치를 채워 주는 함수는 판다스의 fillna()
- 괄호 안에 df.mean()을 넣어 주면 평균값으로 채워 줌

```
df = df.fillna(df.mean())
```

2 결측치, 카테고리 변수 처리하기

● 결측치, 카테고리 변수 처리하기

- 특정한 값으로 대체하려면 `fillna()` 함수의 괄호 안에 해당 값을 적으면 됨
- 예를 들어 결측치를 모두 0으로 바꾸려면 `fillna(0)`이 됨
- `dropna()`를 사용하면 결측치가 있는 속성을 제거
- 이때 `dropna(how='any')`는 결측치가 하나라도 있으면 삭제하라는 의미이고, `dropna(how='all')`은 모든 값이 결측치일 때 삭제하라는 의미



2 결측치, 카테고리 변수 처리하기

- 결측치, 카테고리 변수 처리하기
 - 이제 업데이트된 데이터 프레임을 출력해 보자

```
df
```

2 결측치, 카테고리 변수 처리하기

● 결측치, 카테고리 변수 처리하기

실행 결과

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFin SF1	...
0	1	60	65.0	8450	7	5	2003	2003	196.0	706	...
1	2	20	80.0	9600	6	8	1976	1976	0.0	978	...
2	3	60	68.0	11250	7	5	2001	2002	162.0	486	...
3	4	70	60.0	9550	7	5	1915	1970	0.0	216	...
4	5	60	84.0	14260	8	5	2000	2000	350.0	655	...
...
1455	1456	60	62.0	7917	6	5	1999	2000	0.0	0	...
1456	1457	20	85.0	13175	6	6	1978	1988	119.0	790	...
1457	1458	70	66.0	9042	7	9	1941	2006	0.0	275	...
1458	1459	20	68.0	9717	5	6	1950	1996	0.0	49	...
1459	1460	20	75.0	9937	5	6	1965	1965	0.0	830	...

2 결측치, 카테고리 변수 처리하기

- 결측치, 카테고리 변수 처리하기

SaleType_WD	SaleCondition_Abnorml	SaleCondition_AdjLand	SaleCondition_Alloca	SaleCondition_Family	SaleCondition_Normal	SaleCondition_Partial
1	0	0	0	0	1	0
1	0	0	0	0	1	0
1	0	0	0	0	1	0
1	1	0	0	0	0	0
1	0	0	0	0	1	0
...
1	0	0	0	0	1	0
1	0	0	0	0	1	0
1	0	0	0	0	1	0
1	0	0	0	0	1	0
1	0	0	0	0	1	0

1460 rows × 290 columns

- 결측치는 보이지 않으며, 카테고리형 변수를 모두 원-핫 인코딩 처리하므로 전체 열이 81개에서 290개로 늘었음



3 속성별 관련도 추출하기

3 속성별 관련도 추출하기

● 속성별 관련도 추출하기

- 이 중에서 우리에게 필요한 정보를 추출해 보자
- 먼저 ❶ 데이터 사이의 상관관계를 df_corr 변수에 저장
- ❷ 집 값과 관련이 큰 것부터 순서대로 정렬해 df_corr_sort 변수에 저장
- ❸ 집 값과 관련도가 가장 큰 열 개의 속성들을 출력

```
df_corr = df.corr() ..... ❶  
df_corr_sort = df_corr.sort_values('SalePrice', ascending=False) ..... ❷  
df_corr_sort['SalePrice'].head(10) ..... ❸
```

3 속성별 관련도 추출하기

- 속성별 관련도 추출하기

실행 결과

```
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
```

3 속성별 관련도 추출하기

- 속성별 관련도 추출하기

- 추출된 속성들과 집 값의 관련도를 시각적으로 확인하기 위해 상관도 그래프를 그려 보자

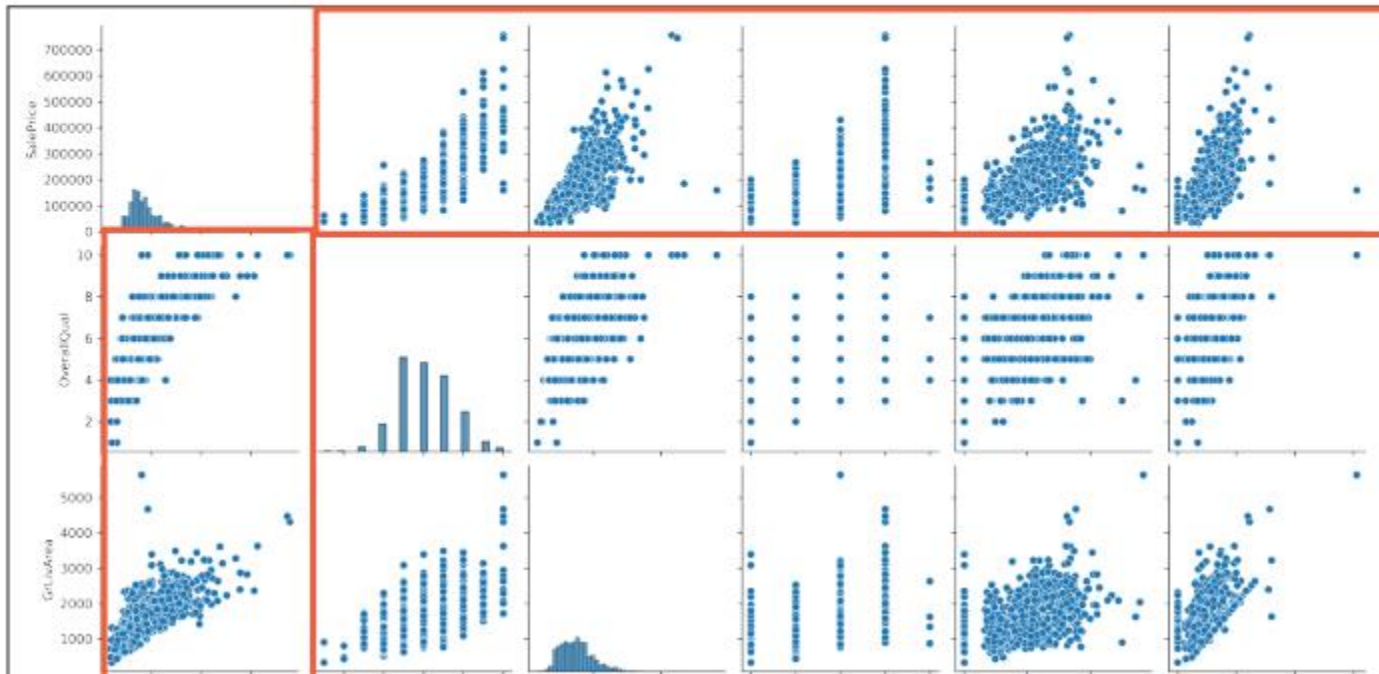
```
cols = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea',  
        'TotalBsmtSF']  
sns.pairplot(df[cols])  
plt.show();
```

3 속성별 관련도 추출하기

● 속성별 관련도 추출하기

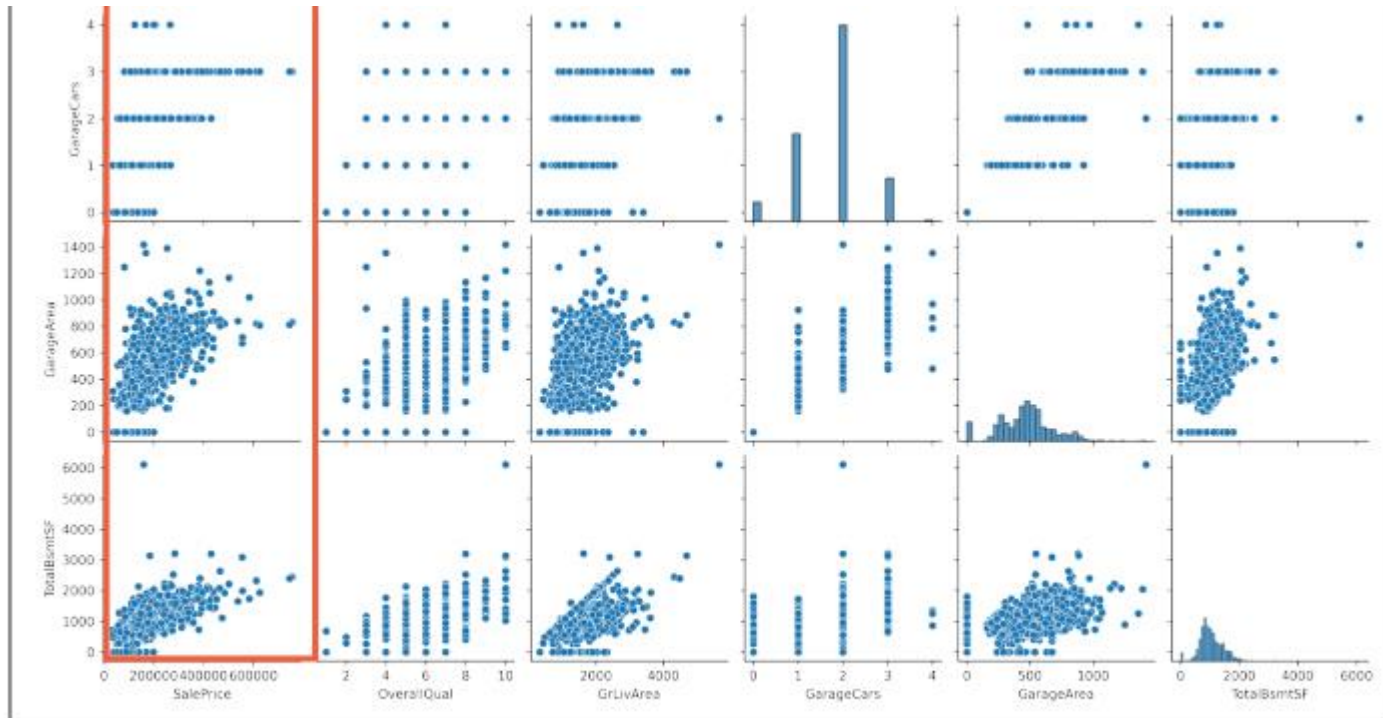
실행 결과

▼ 그림 15-1 | 추출된 속성들과 집 값의 관련도



3 속성별 관련도 추출하기

- 속성별 관련도 추출하기



- 선택된 속성들이 집 값(SalePrice)과 양의 상관관계가 있음을 확인할 수 있음(빨간색 사각형으로 표시한 부분)



4 주택 가격 예측 모델

4 주택 가격 예측 모델

● 주택 가격 예측 모델

- 이제 앞서 구한 중요 속성을 이용해 학습셋과 테스트셋을 만들어 보자
- 집 값을 y로, 나머지 열을 X_train_pre로 저장한 후 전체의 80%를 학습셋으로, 20%를 테스트셋으로 지정

```
cols_train = ['OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea', 'TotalB
smtSF']
X_train_pre = df[cols_train]
y = df['SalePrice'].values
X_train, X_test, y_train, y_test = train_test_split(X_train_pre, y, test_
size=0.2)
```

4 주택 가격 예측 모델

● 주택 가격 예측 모델

- 모델의 구조와 실행 옵션을 설정
- 입력될 속성의 개수를 `X_train.shape[1]`로 지정해 자동으로 세도록 했음

```
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()
```

4 주택 가격 예측 모델

- 주택 가격 예측 모델

- 실행에서 달라진 점은 손실 함수
- 선형 회귀이므로 평균 제곱 오차(mean_squared_error)를 적용

```
model.compile(optimizer='adam', loss='mean_squared_error')
```

4 주택 가격 예측 모델

● 주택 가격 예측 모델

- 20번 이상 결과가 향상되지 않으면 자동으로 중단되게끔 함
- 저장될 모델 이름을 'Ch15-house.hdf5'로 정함
- 모델은 차후 '22장. 캐글로 시작하는 새로운 도전'에서 다시 사용(검증셋을 추가하고 싶을 경우 앞서와 마찬가지로 학습셋, 검증셋, 테스트셋의 비율을 각각 60%, 20%, 20%로 정하면 됨)

```
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=20)
```

```
modelpath = "./data/model/Ch15-house.hdf5"
```

```
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss',  
verbose=0, save_best_only=True)
```

```
history = model.fit(X_train, y_train, validation_split=0.25, epochs=2000,  
batch_size=32, callbacks=[early_stopping_callback,checkpointer])
```

4 주택 가격 예측 모델

- 주택 가격 예측 모델
 - 모든 코드를 실행하면 다음과 같음

실습 | 주택 가격 예측하기



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
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
```

4 주택 가격 예측 모델

● 주택 가격 예측 모델

깃허브에 준비된 데이터를 가져옵니다.

```
!git clone https://github.com/taehojo/data.git
```

집 값 데이터를 불러옵니다.

```
df = pd.read_csv("./data/house_train.csv")
```

카테고리형 변수를 0과 1로 이루어진 변수로 바꾸어 줍니다.

```
df = pd.get_dummies(df)
```

결측치를 전체 칼럼의 평균으로 대체해 채워 줍니다.

```
df = df.fillna(df.mean())
```

4 주택 가격 예측 모델

● 주택 가격 예측 모델

```
# 데이터 사이의 상관관계를 저장합니다.
```

```
df_corr = df.corr()
```

```
# 집 값과 관련이 큰 것부터 순서대로 저장합니다.
```

```
df_corr_sort = df_corr.sort_values('SalePrice', ascending=False)
```

```
# 집 값을 제외한 나머지 열을 저장합니다.
```

```
cols_train = ['OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea', 'TotalB  
smtSF']
```

```
X_train_pre = df[cols_train]
```

```
# 집 값을 저장합니다.
```

```
y = df['SalePrice'].values
```

4 주택 가격 예측 모델

● 주택 가격 예측 모델

```
# 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()
```


4 주택 가격 예측 모델

● 주택 가격 예측 모델

모델을 실행합니다.

```
model.compile(optimizer='adam', loss='mean_squared_error')
```

20번 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=True)
```



4 주택 가격 예측 모델

● 주택 가격 예측 모델

```
# 실행 관련 설정을 하는 부분입니다. 전체의 20%를 검증셋으로 설정합니다.  
history = model.fit(X_train, y_train, validation_split=0.25, epochs=2000,  
                    batch_size=32, callbacks=[early_stopping_callback,checkpointer])
```

4 주택 가격 예측 모델

● 주택 가격 예측 모델

실행 결과

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
dense (Dense)	(None, 10)	60
dense_1 (Dense)	(None, 30)	330
dense_2 (Dense)	(None, 40)	1240
dense_3 (Dense)	(None, 1)	41
=====		

4 주택 가격 예측 모델

- 주택 가격 예측 모델

```
Total params: 1,671
```

```
Trainable params: 1,671
```

```
Non-trainable params: 0
```

```
Epoch 1/2000
```

```
28/28 [=====] - 0s 5ms/step - loss:
```

```
39256875008.0000 - val_loss: 38050066432.0000
```

```
... (중략) ...
```

```
Epoch 145/2000
```

```
28/28 [=====] - 0s 2ms/step - loss:
```

```
1962943104.0000 - val_loss: 2011970944.0000
```

- 145번째에서 학습이 중단



4 주택 가격 예측 모델

- 주택 가격 예측 모델
 - 학습 중단 시점은 실행할 때마다 다를 수 있음

4 주택 가격 예측 모델

● 주택 가격 예측 모델

- 학습 결과를 시각화하기 위해 예측 값과 실제 값, 실행 번호가 들어갈 빈 리스트를 만들고 25개의 샘플로부터 얻은 결과를 채워 넣겠음

```
real_prices = []
pred_prices = []
X_num = []

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)
```



4 주택 가격 예측 모델

- 주택 가격 예측 모델

```
pred_prices.append(prediction)
n_iter = n_iter + 1
X_num.append(n_iter)
```



4 주택 가격 예측 모델

● 주택 가격 예측 모델

실행 결과

실제가격: 262500.00, 예상가격: 240051.36

실제가격: 78000.00, 예상가격: 118369.56

... (중략) ...

실제가격: 127000.00, 예상가격: 116693.46

실제가격: 485000.00, 예상가격: 357789.88

4 주택 가격 예측 모델

- 주택 가격 예측 모델

- 그래프를 통해 샘플로 뽑은 25개의 값을 비교해 보자

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

4 주택 가격 예측 모델

● 주택 가격 예측 모델

실행 결과

▼ 그림 15-2 | 실제 집 값과 예측된 집 값 비교



4 주택 가격 예측 모델

● 주택 가격 예측 모델

- 예측된 집 값의 곡선이 실제 집 값의 곡선과 유사하게 움직이고 있음을 볼 수 있음
- 그림 15-2에 출력되는 곡선의 전체적인 형태는 실행할 때마다 달라질 수 있음
- 예측된 집 값의 곡선과 실제 집 값의 곡선이 유사하게 움직이면 잘 학습된 것