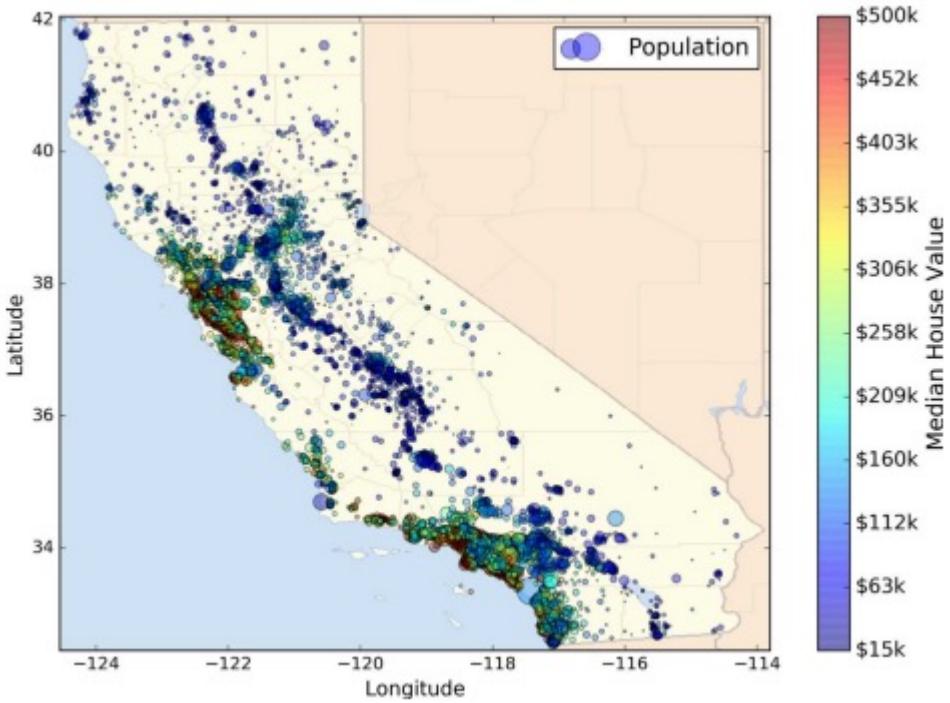
CUAI 스터디_MTs - 머신러닝 프로젝트

2022.03.19

발표자 : 이하은

캘리포니아 주택 가격 예측 모델



구역 별 포함된 데이터

- 인구
- 중간 소득
- 중간 주택 가격 등

문제 정의

- 목적 파악
  - 문제를 어떻게 구성할지, 어떤 알고리즘을 선택할지, 모델 평가에 어떤 성능 지표를 사용할지, 모델 튜닝을 위해 얼마나 노력을 투여할지 등을 결정
  - 파이프라인에서의 인풋

- 현재 솔루션의 구성 파악
  ex. 전문가들의 수동 측정

- 구체적인 문제 정의
  ex.
  - 레이블된 훈련 샘플 : 지도학습
  - 값 예측 : 회귀 문제
  - 예측에 사용할 특성이 여러 개 : 다중 회귀 문제
  - 구역마다 하나의 값 예측 : 단변량 회귀 문제
  - 데이터에 플로우 X, 빠르게 변화X, 데이터 크기가 작음 : 일반적인 배치 학습

성능 측정 지표 선택

- 평균 제곱근 오차 (RMSE)

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

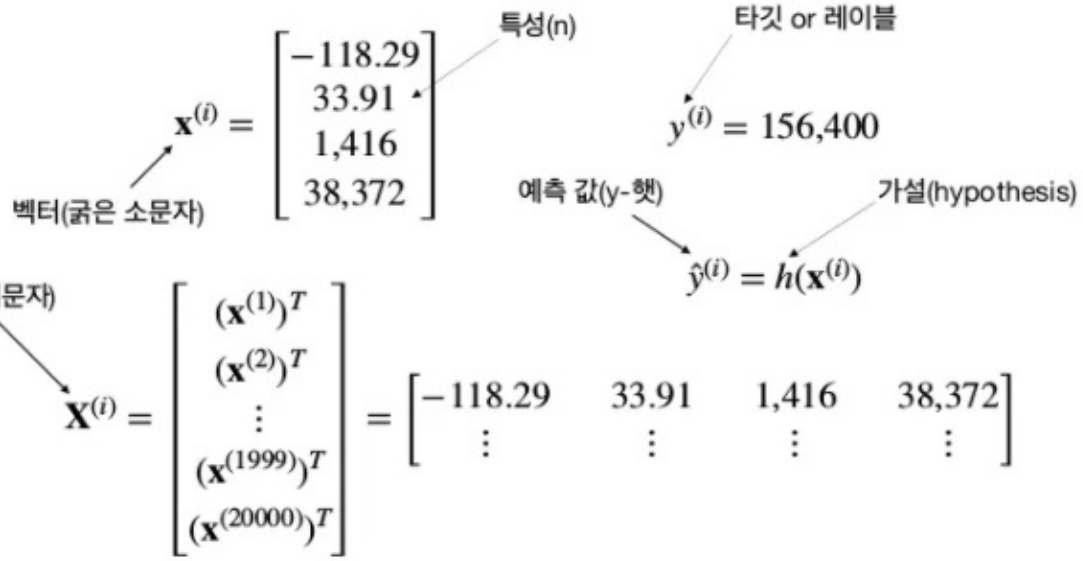
회귀 문제의 전형적인 성능 지표
오차가 커질 수록 더욱 값이 커지므로, 예측의 오류가 얼마나 많은지 가늠.

- 평균 절대 오차 (MAE)
  이상치로 보이는 구역이 많을 때, 회귀 문제의 성능 지표

성능 측정 지표 선택

- 표기법

어떤 구역의 경도가 -118.29, 위도 33.91, 주민수 1,416, 중간 소득 $38,372이고 중간 주택 가격이 $156,400이라면,



데이터 구조 훑어보기

In [6]:
```
housing = load_housing_data()
housing.head()
```

Out [6]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	house
0	-122.23	37.88	41.0	880.0	129.0	322.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1
2	-122.24	37.85	52.0	1467.0	190.0	496.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	

데이터 구조 훑어보기

```
In [7]: housing.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
In [8]: housing["ocean_proximity"].value_counts()

Out[8]: <1H OCEAN     9136
        INLAND        6551
        NEAR OCEAN    2658
        NEAR BAY      2290
        ISLAND           5
        Name: ocean_proximity, dtype: int64
```

데이터 구조 훑어보기

```
In [9]: housing.describe()
```

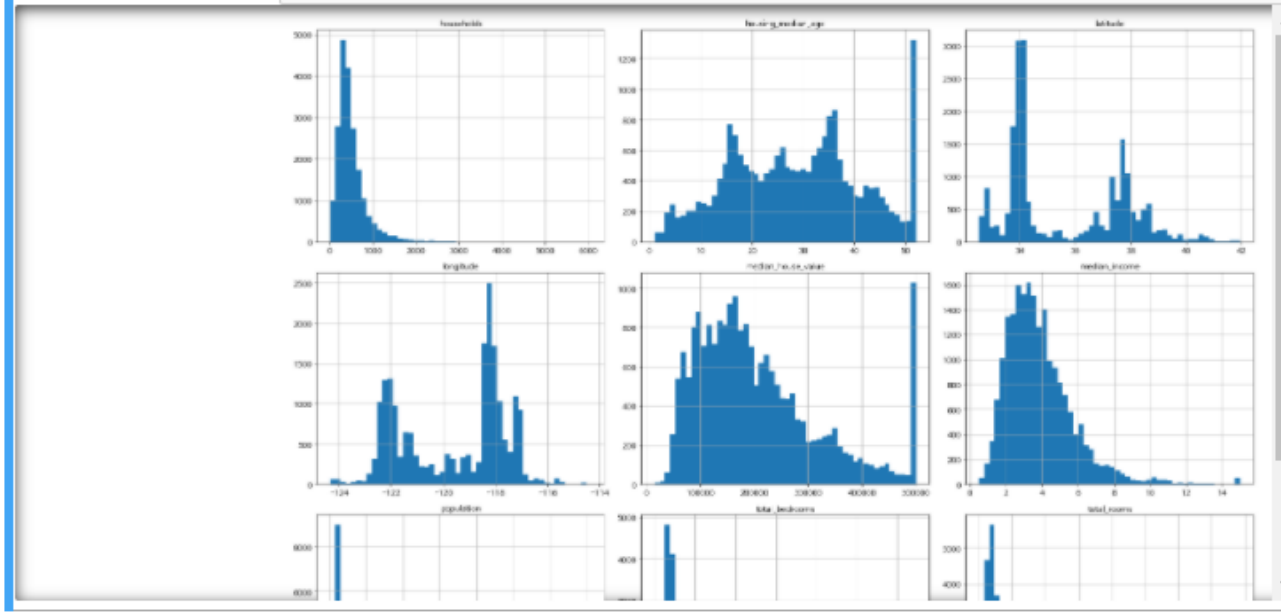
Out [9]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	206
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	14
std	2.003532	2.135952	12.585558	2181.615252	421.385070	11
min	-124.350000	32.540000	1.000000	2.000000	1.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	7
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	11
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	17
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	356

데이터 구조 훑어보기

In [10]:

```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
save_fig("attribute_histogram_plots")
plt.show()
```



테스트 세트 생성

- 데이터 스누핑 편향
  ; 데이터를 보고 편견이 생겨 적합하지 못한 알고리즘을 쓰게 되었을 때, 기대한 성능이 나오지 않는 것.

In [12]:
```
import numpy as np

# 예시를 위해서 만든 것입니다. 사이킷런에는 train_test_split() 함수가 있습니다.
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

In [13]:
```
train_set, test_set = split_train_test(housing, 0.2)
print(len(train_set), "train +", len(test_set), "test")
```

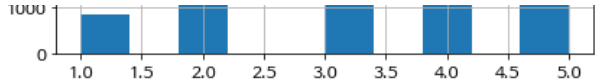
```
16512 train + 4128 test
```

보통 데이터 세트의 약 20% -> 해당 테스트 세트는 별도 저장 필요

계층적 샘플링

- 데이터의 양이 방대하지 않다면, 테스트 세트를 선정할 때 데이터의 편향성이 드러나지 않아야함.

	Overall	Stratified	Random	Rand. %error	Strat. %error
1.0	0.039826	0.039729	0.040213	0.973236	-0.243309
2.0	0.318847	0.318798	0.324370	1.732260	-0.015195
3.0	0.350581	0.350533	0.358527	2.266446	-0.013820
4.0	0.176308	0.176357	0.167393	-5.056334	0.027480
5.0	0.114438	0.114583	0.109496	-4.318374	0.127011



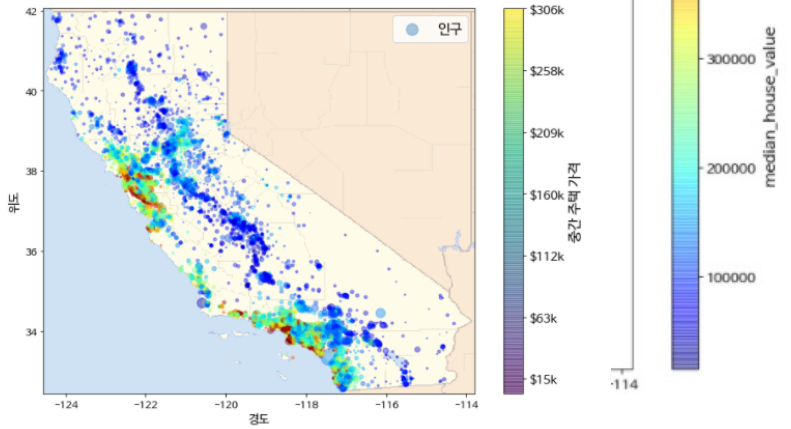
데이터 시각화

- 지리정보를 활용한 지리적 데이터 시각화

In [35]:

In [36]:
```
import matplotlib.image as mpimg
california_img=mpimg.imread(PROJECT_ROOT_DIR + '/images/end_to_end_project/califo
ax = housing.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                  s=housing['population']/100, label="인구",
                  c="median_house_value", cmap=plt.get_cmap("jet"),
                  colorbar=False, alpha=0.4,
                 )
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05], alpha=0.5)
plt.ylabel("위도", fontsize=14)
plt.xlabel("경도", fontsize=14)

prices = housing["median_house_value"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cbar = plt.colorbar()
cbar.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], fontsize=
cbar.set_label('중간 주택 가격', fontsize=16)

plt.legend(fontsize=16)
save_fig("california_housing_prices_plot")
plt.show()
```

```
alpha=0.4,

True,

de", alpha=0.1)
```

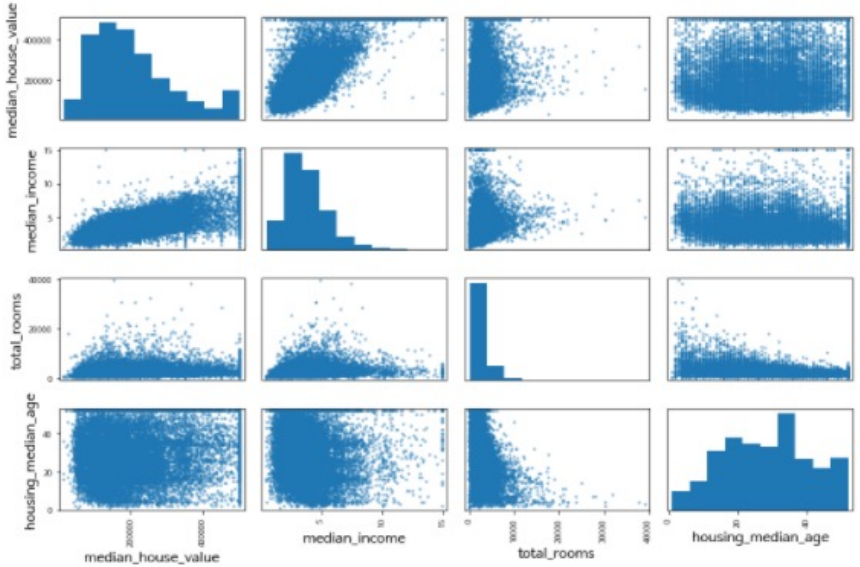


상관관계 조사

- 특성 간의 표준 상관계수(정비례/반비례 관계) 계산

In [39]:

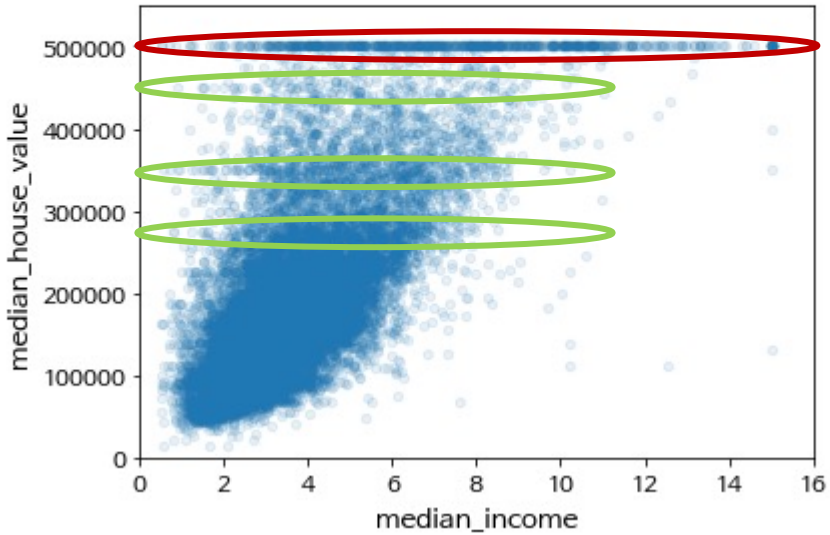
```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
save_fig("scatter_matrix_plot")
```



상관관계 조사

- 특성 간의 표준 상관계수(정비례/반비례 관계) 계산

```
In [40]:
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
plt.axis([0, 16, 0, 550000])
save_fig("income_vs_house_value_scatterplot")
```



특성 조합 실험

- 가구당 방 개수, 방 별 침대 개수, 가구당 인원 특성 조합

In [41]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

In [42]:
```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[42]:
```
median_house_value          1.000000
median_income               0.687160
rooms_per_household         0.146285
total_rooms                 0.135097
housing_median_age          0.114110
households                  0.064506
total_bedrooms              0.047689
population_per_household   -0.021985
population                 -0.026920
longitude                  -0.047432
latitude                   -0.142724
bedrooms_per_room          -0.259984
Name: median_house_value, dtype: float64
```

머신러닝 알고리즘을 위한 데이터 준비

- 집의 가격을 예측할 것이므로 house_value 행 삭제

```
In [45]:
housing = strat_train_set.drop("median_house_value", axis=1) # 훈련 세트를 위해 ㄷ
housing_labels = strat_train_set["median_house_value"].copy()
```

- NULL값 확인

```
In [46]:
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

Out [46]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	h
4629	-118.30	34.07	18.0	3759.0	NaN	3296.0	
6068	-117.86	34.01	16.0	4632.0	NaN	3038.0	
17923	-121.97	37.35	30.0	1955.0	NaN	999.0	
13656	-117.30	34.05	6.0	2155.0	NaN	1039.0	
19252	-122.79	38.48	7.0	6837.0	NaN	3468.0	

머신러닝 알고리즘을 위한 데이터 준비

- NULL값 제거

In [47]:
```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # 옵션 1
```

Out[47]:

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househ

In [48]:
```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # 옵션 2
```

Out[48]:

	longitude	latitude	housing_median_age	total_rooms	population	households	media
4629	-118.30	34.07	18.0	3759.0	3296.0	1462.0	
6068	-117.86	34.01	16.0	4632.0	3038.0	727.0	
17923	-121.97	37.35	30.0	1955.0	999.0	386.0	
13656	-117.30	34.05	6.0	2155.0	1039.0	391.0	
19252	-122.79	38.48	7.0	6837.0	3468.0	1405.0	

In [49]:
```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # 옵션 3
sample_incomplete_rows
```

Out[49]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	h
4629	-118.30	34.07	18.0	3759.0	433.0	3296.0	
6068	-117.86	34.01	16.0	4632.0	433.0	3038.0	
17923	-121.97	37.35	30.0	1955.0	433.0	999.0	
13656	-117.30	34.05	6.0	2155.0	433.0	1039.0	
19252	-122.79	38.48	7.0	6837.0	433.0	3468.0	

머신러닝 알고리즘을 위한 데이터 준비

- 사이킷 런의 imputer 메소드 사용

In [50]:
```
#from sklearn.preprocessing import Imputer
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
```

중간값이 수치형 특성에서만 계산될 수 있기 때문에 텍스트 특성을 삭제합니다:

In [51]:
```
housing_num = housing.drop('ocean_proximity', axis=1)
# 다른 방법: housing_num = housing.select_dtypes(include=[np.number])
```

In [52]:
```
imputer.fit(housing_num)
```

Out[52]:
```
SimpleImputer(add_indicator=False, copy=True, fill_value=None,
              missing_values=nan, strategy='median', verbose=0)
```

In [53]:
```
imputer.statistics_
```

Out[53]:
```
array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

각 특성의 중간 값이 수동으로 계산한 것과 같은지 확인해 보세요:

In [54]:
```
housing_num.median().values
```

Out[54]:
```
array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

훈련 세트 변환:

머신러닝 알고리즘을 위한 데이터 준비

- 텍스트와 범주형 특성 전처리 - factorize

In [60]:
```
housing_cat = housing['ocean_proximity']
housing_cat.head(10)
```

Out[60]:
```
17606     <1H OCEAN
18632     <1H OCEAN
14650    NEAR OCEAN
3230         INLAND
3555      <1H OCEAN
19480        INLAND
8879      <1H OCEAN
13685        INLAND
4937      <1H OCEAN
4861      <1H OCEAN
Name: ocean_proximity, dtype: object
```

판다스의 `factorize()` 메소드는 문자열 범주형 특성을 머신러닝 알고리즘이 다루기 쉬운 숫자 범주형 특성으로 변환시켜 줍니다:

In [61]:
```
housing_cat_encoded, housing_categories = housing_cat.factorize()
housing_cat_encoded[:10]
```

Out[61]:
```
array([0, 0, 1, 2, 0, 2, 0, 2, 0, 0])
```

In [62]:
```
housing_categories
```

Out[62]:
```
Index(['<1H OCEAN', 'NEAR OCEAN', 'INLAND', 'NEAR BAY', 'ISLAND'], dtype='objec
t')
```

`OneHotEncoder`를 사용하여 범주형 값을 원-핫 벡터로 변경합니다:

머신러닝 알고리즘을 위한 데이터 준비

- 특성 스케일링 – 특성별 스케일이 너무 차이가 나는 경우 정규화 혹은 표준화 진행

In [81]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

In [82]:
```
housing_num_tr
```

Out[82]:
```
array([[-1.15604281,  0.77194962,  0.74333089, ..., -0.31205452,
        -0.08649871,  0.15531753],
       [-1.17602483,  0.6596948 , -1.1653172 , ...,  0.21768338,
        -0.03353391, -0.83628902],
       [ 1.18684903, -1.34218285,  0.18664186, ..., -0.46531516,
        -0.09240499,  0.4222004 ],
       ...,
       [ 1.58648943, -0.72478134, -1.56295222, ...,  0.3469342 ,
        -0.03055414, -0.52177644],
       [ 0.78221312, -0.85106801,  0.18664186, ...,  0.02499488,
         0.06150916, -0.30340741],
       [-1.43579109,  0.99645926,  1.85670895, ..., -0.22852947,
        -0.09586294,  0.10180567]])
```

모델 선택과 훈련

- 훈련 세트에서 훈련 및 평가 – 선형회귀모델 훈련

In [90]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

Out[90]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [91]:
```
# 훈련 샘플 몇 개를 사용해 전체 파이프라인을 적용해 보겠습니다.
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)

print("예측:", lin_reg.predict(some_data_prepared))
```

```
예측: [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
```

실제 값과 비교합니다:

In [92]:
```
print("레이블:", list(some_labels))
```

```
레이블: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

모델 선택과 훈련

- 훈련 세트에서 훈련 및 평가 – rmse값 계산

In [94]:
```
from sklearn.metrics import mean_squared_error

housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out [94]: 68628.19819848923

모델 선택과 훈련

- 훈련 세트에서 훈련 및 평가 – DecisionTreeRegressor

In [96]:
```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

Out [96]:
```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=42, splitter='best')
```

In [97]:
```
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out [97]:
```
0.0
```

모델 선택과 훈련

- 교차검증을 사용한 평가 – K-겹 교차 검증

In [98]:
```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
```

In [99]:
```
def display_scores(scores):
    print("점수:", scores)
    print("평균:", scores.mean())
    print("표준편차:", scores.std())

display_scores(tree_rmse_scores)
```

```
점수: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
 71115.88230639 75585.14172901 70262.86139133 70273.6325285
 75366.87952553 71231.65726027]
평균: 71407.68766037929
표준편차: 2439.4345041191004
```

In [100]:
```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
점수: [66782.73843989 66960.118071   70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
평균: 69052.46136345083
표준편차: 2731.674001798349
```

모델 선택과 훈련

- 랜덤 포레스트 – 특성을 무작위로 선택해 많은 결정 트리를 만들고 그 예측을 평균내는 방식

In [103]:
```
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
점수: [51646.44545909 48940.60114882 53050.86323649 54408.98730149
 50922.14870785 56482.50703987 51864.52025526 49760.85037653
 55434.21627933 53326.10093303]
평균: 52583.72407377466
표준편차: 2298.353351147122
```

- 앙상블 학습 – 여러 다른 모델을 모아서 하나의 모델을 만드는 것으로, 머신러닝 알고리즘 성능을 극대화하는 방법

모델 세부 튜닝

- 그리드 탐색 : 모델에게 가장 적합한 하이퍼 파라미터 조합 찾기

```
In [107]: grid_search.best_params_
Out[107]: {'max_features': 8, 'n_estimators': 30}

In [108]: grid_search.best_estimator_
Out[108]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                                max_features=8, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=30,
                                n_jobs=None, oob_score=False, random_state=42, verbose=0,
                                warm_start=False)
```

그리드서치에서 테스트한 하이퍼파라미터 조합의 점수를 확인합니다:

```
In [109]: cvres = grid_search.cv_results_
          for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
              print(np.sqrt(-mean_score), params)

63669.05791727153 {'max_features': 2, 'n_estimators': 3}
55627.16171305252 {'max_features': 2, 'n_estimators': 10}
53384.57867637289 {'max_features': 2, 'n_estimators': 30}
60965.99185930139 {'max_features': 4, 'n_estimators': 3}
52740.98248528835 {'max_features': 4, 'n_estimators': 10}
50377.344409590376 {'max_features': 4, 'n_estimators': 30}
58663.84733372485 {'max_features': 6, 'n_estimators': 3}
52006.15355973719 {'max_features': 6, 'n_estimators': 10}
50146.465964159885 {'max_features': 6, 'n_estimators': 30}
57869.25504027614 {'max_features': 8, 'n_estimators': 3}
51711.09443660957 {'max_features': 8, 'n_estimators': 10}
49682.25345942335 {'max_features': 8, 'n_estimators': 30}
62895.088889905004 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54658.14484390074 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59470.399594730654 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52725.01091081235 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57490.612956065226 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51009.51445842374 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```