통계적 머신러닝 Final report code

2019150445 신백록

1. Data Loading & Preprocessing

In [1]:
```
import pandas as pd
housing = pd.read_csv('housing.csv')
```

In [2]:
```
housing.head()
```

Out[2]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hous
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	
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	

In [3]:
```
housing.describe()
```

Out[3]:

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

In [4]:
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [5]:
```
y = housing.iloc[:, -2]
X = pd.concat([housing.iloc[:,:-2], housing.iloc[:, -1]],axis=1)
```

In [6]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [7]:
```
X.head()
```

Out[7]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hous
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	
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	

In [8]:
```
X.isnull().sum()
```

Out[8]:

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms      207
population            0
households            0
median_income         0
ocean_proximity       0
dtype: int64
```

In [9]:

```
X[X.isna().any(axis=1)]
```

Out[9]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
290	-122.16	37.77	47.0	1256.0	NaN	570.0
341	-122.17	37.75	38.0	992.0	NaN	732.0
538	-122.28	37.78	29.0	5154.0	NaN	3741.0
563	-122.24	37.75	45.0	891.0	NaN	384.0
696	-122.10	37.69	41.0	746.0	NaN	387.0
...	...	...	...	...	...	...
20267	-119.19	34.20	18.0	3620.0	NaN	3171.0
20268	-119.18	34.19	19.0	2393.0	NaN	1938.0
20372	-118.88	34.17	15.0	4260.0	NaN	1701.0
20460	-118.75	34.29	17.0	5512.0	NaN	2734.0
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0

207 rows × 9 columns

In [10]:

```
#corr for missing values
X.corr() #total bedrooms가 207개 missing이기에 비슷한 Total_rooms 가지고 추정
```

Out[10]:

	longitude	latitude	housing_median_age	total_rooms	total_bedro
longitude	1.000000	-0.924664	-0.108197	0.044568	0.069
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.066
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.32(
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.930
total_bedrooms	0.069608	-0.066983	-0.320451	0.930380	1.000
population	0.099773	-0.108785	-0.296244	0.857126	0.87
households	0.055310	-0.071035	-0.302916	0.918484	0.979
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.007

In [11]:

```
X['total_rooms_cat'] = pd.qcut(X['total_rooms'], 4)
```

In [12]:

```
X['total_rooms_cat'].unique()
```

Out[12]:

```
[(1.999, 1447.75], (3148.0, 39320.0], (1447.75, 2127.0], (2127.0, 3148.0]]
Categories (4, interval[float64, right]): [(1.999, 1447.75] < (1447.75, 212
7.0] < (2127.0, 3148.0] < (3148.0, 39320.0]]
```

In [13]:

```
X['total_bedrooms'] = X['total_bedrooms'].fillna(X.groupby('total_rooms_cat
```

In [14]:

```
X[X['total_bedrooms'].isna()]
```

Out[14]:

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	house

In [15]:

```
X.iloc[[290,341,20484]]
```

Out[15]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
290	-122.16	37.77	47.0	1256.0	222.0	570.0
341	-122.17	37.75	38.0	992.0	222.0	732.0
20484	-118.72	34.28	17.0	3051.0	512.0	1705.0

In [16]:

```
X = X.drop('total_rooms_cat', axis=1)
```

In [17]:

```
X.isna().sum()
```

Out[17]:
```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
ocean_proximity       0
dtype: int64
```

In [18]:
```
# Missing Automation

from sklearn.base import TransformerMixin
class NullValueImputer(TransformerMixin):
    def __init__(self):
        None
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        for column in X.columns.tolist():
            if column in X.columns[X.dtypes==object].tolist():
                X[column] = X[column].fillna(X[column].mode())
            elif column in ['total_bedrooms', 'population', 'households']:
                X['total_rooms_cat'] = pd.qcut(X['total_rooms'], 4)
                X[column] = X[column].fillna(X.groupby('total_rooms_cat')[c
                X.drop('total_rooms_cat', axis=1, inplace=True)
            else:
                X[column] = X[column].fillna(X[column].median())
        return X
```

In [ ]:

In [19]:
```
#cat
X['ocean_proximity'].unique()
```

Out[19]:
```
array(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'],
      dtype=object)
```

In [20]:
```
#One Hot encoding
import numpy as np
from sklearn.preprocessing import OneHotEncoder

cat = np.array(X['ocean_proximity'])
cat = cat.reshape(-1,1)

ohe = OneHotEncoder()
cat = ohe.fit_transform(cat)
cat_df = pd.DataFrame(cat.toarray())
cat_df.head()
```

Out[20]:

	0	1	2	3	4
0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	1.0	0.0

In [21]:

```
numeric_df = X.select_dtypes(exclude=['object'])
numeric_df.head()
```

Out[21]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hous
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	
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	

In [22]:

```
from scipy.sparse import csr_matrix, hstack
numeric = csr_matrix(numeric_df)
sparse_mat = hstack((cat, numeric))
sparse_df = pd.DataFrame(sparse_mat.toarray())
sparse_df.head()
```

Out[22]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.0	0.0	0.0	1.0	0.0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252
1	0.0	0.0	0.0	1.0	0.0	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014
2	0.0	0.0	0.0	1.0	0.0	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574
3	0.0	0.0	0.0	1.0	0.0	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431
4	0.0	0.0	0.0	1.0	0.0	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462

In [23]:

```
# OHE Automation
class SparseMatrix(TransformerMixin):
    def __init__(self):
        None
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        if len(X.columns[X.dtypes==object])==1:
            cat_column = X.columns[X.dtypes==object]
            cat = np.array(X[cat_column])
            cat = cat.reshape(-1,1)

            ohe = OneHotEncoder()
            cat = ohe.fit_transform(cat)
        else:
            categorical_columns = X.columns[X.dtypes==object].tolist()
            ohe = OneHotEncoder()
            cat = ohe.fit_transform(X[categorical_columns])
        numeric_df = X.select_dtypes(exclude=["object"])
        numeric = csr_matrix(numeric_df)
        final_sparse_matrix = hstack((cat, numeric))
        return final_sparse_matrix
```

In [24]:

```
# OHE Not sparse Automation
class OHEMatrix(TransformerMixin):
    def __init__(self):
        None
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        if len(X.columns[X.dtypes==object])==1:
            cat_column = X.columns[X.dtypes==object]
            cat = np.array(X[cat_column])
            cat = cat.reshape(-1,1)

            ohe = OneHotEncoder(sparse=False)
            cat = pd.DataFrame(ohe.fit_transform(cat))
        else:
            categorical_columns = X.columns[X.dtypes==object].tolist()
            ohe = OneHotEncoder(sparse=False)
            cat = pd.DataFrame(ohe.fit_transform(X[categorical_columns]))
        numeric = X.select_dtypes(exclude=["object"])
        numeric = pd.DataFrame(np.array(numeric)) # to rearrange index

        final_matrix = pd.concat([numeric, cat],ignore_index=True, axis=1)
        return final_matrix
```

In [25]:

```
sm = SparseMatrix().fit_transform(X)
print(sm)
```

```
  (0, 3)	1.0
  (1, 3)	1.0
  (2, 3)	1.0
  (3, 3)	1.0
  (4, 3)	1.0
  (5, 3)	1.0
  (6, 3)	1.0
  (7, 3)	1.0
  (8, 3)	1.0
  (9, 3)	1.0
  (10, 3)	1.0
  (11, 3)	1.0
  (12, 3)	1.0
  (13, 3)	1.0
  (14, 3)	1.0
  (15, 3)	1.0
  (16, 3)	1.0
  (17, 3)	1.0
  (18, 3)	1.0
  (19, 3)	1.0
  (20, 3)	1.0
  (21, 3)	1.0
  (22, 3)	1.0
  (23, 3)	1.0
  (24, 3)	1.0
  :	:
  (20636, 12)	2.5568
  (20637, 5)	-121.22
  (20637, 6)	39.43
  (20637, 7)	17.0
  (20637, 8)	2254.0
  (20637, 9)	485.0
  (20637, 10)	1007.0
  (20637, 11)	433.0
  (20637, 12)	1.7
  (20638, 5)	-121.32
  (20638, 6)	39.43
  (20638, 7)	18.0
  (20638, 8)	1860.0
  (20638, 9)	409.0
  (20638, 10)	741.0
  (20638, 11)	349.0
  (20638, 12)	1.8672
  (20639, 5)	-121.24
  (20639, 6)	39.37
  (20639, 7)	16.0
  (20639, 8)	2785.0
  (20639, 9)	616.0
  (20639, 10)	1387.0
  (20639, 11)	530.0
  (20639, 12)	2.3886
```

In [26]:

```
non_sm = OHEMatrix().fit_transform(X_train)
non_sm
```

Out[26]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	-119.84	36.77	6.0	1853.0	473.0	1397.0	417.0	1.4817	0.0	1.0	0.0	0.0	0.0
1	-117.80	33.68	8.0	2032.0	349.0	862.0	340.0	6.9133	1.0	0.0	0.0	0.0	0.0
2	-120.19	36.60	25.0	875.0	214.0	931.0	214.0	1.5536	0.0	1.0	0.0	0.0	0.0
3	-118.32	34.10	31.0	622.0	229.0	597.0	227.0	1.5284	1.0	0.0	0.0	0.0	0.0
4	-121.23	37.79	21.0	1922.0	373.0	1130.0	372.0	4.0815	0.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
16507	-121.90	39.59	20.0	1465.0	278.0	745.0	250.0	3.0625	0.0	1.0	0.0	0.0	0.0
16508	-122.25	38.11	49.0	2365.0	504.0	1131.0	458.0	2.6133	0.0	0.0	0.0	1.0	0.0
16509	-121.22	38.92	19.0	2531.0	461.0	1206.0	429.0	4.4958	0.0	1.0	0.0	0.0	0.0
16510	-118.14	34.16	39.0	2776.0	840.0	2546.0	773.0	2.5750	1.0	0.0	0.0	0.0	0.0
16511	-124.13	40.80	31.0	2152.0	462.0	1259.0	420.0	2.2478	0.0	0.0	0.0	0.0	1.0

16512 rows × 13 columns

In [27]:

```
from sklearn.pipeline import Pipeline
data_pipeline = Pipeline([('null_imputer', NullValueImputer()), ('ohe',OHEI
X_train_transformed = data_pipeline.fit_transform(X_train)
```

In [28]:

```
X_train_transformed
```

Out[28]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	-119.84	36.77	6.0	1853.0	473.0	1397.0	417.0	1.4817	0.0	1.0	0.0	0.0	0.0
1	-117.80	33.68	8.0	2032.0	349.0	862.0	340.0	6.9133	1.0	0.0	0.0	0.0	0.0
2	-120.19	36.60	25.0	875.0	214.0	931.0	214.0	1.5536	0.0	1.0	0.0	0.0	0.0
3	-118.32	34.10	31.0	622.0	229.0	597.0	227.0	1.5284	1.0	0.0	0.0	0.0	0.0
4	-121.23	37.79	21.0	1922.0	373.0	1130.0	372.0	4.0815	0.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
16507	-121.90	39.59	20.0	1465.0	278.0	745.0	250.0	3.0625	0.0	1.0	0.0	0.0	0.0
16508	-122.25	38.11	49.0	2365.0	504.0	1131.0	458.0	2.6133	0.0	0.0	0.0	1.0	0.0
16509	-121.22	38.92	19.0	2531.0	461.0	1206.0	429.0	4.4958	0.0	1.0	0.0	0.0	0.0
16510	-118.14	34.16	39.0	2776.0	840.0	2546.0	773.0	2.5750	1.0	0.0	0.0	0.0	0.0
16511	-124.13	40.80	31.0	2152.0	462.0	1259.0	420.0	2.2478	0.0	0.0	0.0	0.0	1.0

16512 rows × 13 columns

In [29]:

```
X_train_transformed.columns=['longitude', 'latitude', 'housing_median_age',
       'total_bedrooms', 'population', 'households', 'median_income', 'ocea
X_train_transformed
```

Out[29]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-119.84	36.77	6.0	1853.0	473.0	1397.0
1	-117.80	33.68	8.0	2032.0	349.0	862.0
2	-120.19	36.60	25.0	875.0	214.0	931.0
3	-118.32	34.10	31.0	622.0	229.0	597.0
4	-121.23	37.79	21.0	1922.0	373.0	1130.0
...	...	...	...	...	...	...
16507	-121.90	39.59	20.0	1465.0	278.0	745.0
16508	-122.25	38.11	49.0	2365.0	504.0	1131.0
16509	-121.22	38.92	19.0	2531.0	461.0	1206.0
16510	-118.14	34.16	39.0	2776.0	840.0	2546.0
16511	-124.13	40.80	31.0	2152.0	462.0	1259.0

16512 rows × 13 columns

In [30]:

```
X_test_transformed = data_pipeline.transform(X_test)
```

In [31]:

```
X_test_transformed
```

Out[31]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	-117.65	33.60	15.0	5736.0	837.0	2529.0	762.0	6.4114	1.0	0.0	0.0	0.0	0.0
1	-120.91	38.62	12.0	4545.0	748.0	2033.0	718.0	4.1843	0.0	1.0	0.0	0.0	0.0
2	-118.23	33.93	35.0	1149.0	277.0	909.0	214.0	1.7411	1.0	0.0	0.0	0.0	0.0
3	-122.37	37.59	39.0	4645.0	1196.0	2156.0	1113.0	3.4412	0.0	0.0	0.0	0.0	1.0
4	-117.98	33.70	16.0	5127.0	631.0	2142.0	596.0	7.8195	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4123	-117.12	34.04	25.0	2495.0	438.0	1071.0	405.0	4.8173	0.0	1.0	0.0	0.0	0.0
4124	-121.57	39.12	30.0	2601.0	534.0	1702.0	506.0	2.0800	0.0	1.0	0.0	0.0	0.0
4125	-118.16	33.97	30.0	2419.0	715.0	3208.0	719.0	2.1743	1.0	0.0	0.0	0.0	0.0
4126	-117.14	32.70	32.0	1280.0	353.0	1335.0	330.0	1.6023	0.0	0.0	0.0	0.0	1.0
4127	-119.19	34.21	28.0	4194.0	811.0	2556.0	856.0	4.2227	0.0	0.0	0.0	0.0	1.0

4128 rows × 13 columns

2. Model Selection

In [32]:

```
from sklearn.model_selection import cross_val_score, KFold
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
#from catboost import CatBoostRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso, Ridge
from sklearn.linear_model import RANSACRegressor
import warnings
warnings.filterwarnings('ignore')
```

In [33]:

```
kfold = KFold(n_splits=10, shuffle=True, random_state=0)
def regression_model(model):

    scores = cross_val_score(model, X_train_transformed, y_train, scoring=

    rmse = (-scores)**0.5

    # Return mean score
    return rmse.mean()
```

In [58]:
```
rmse_lr = regression_model(LinearRegression())
print(rmse_lr)
```

68558.3849715519

In [59]:
```
rmse_lasso = regression_model(Lasso())
print(rmse_lasso)
```

68557.53494336414

In [60]:
```
rmse_ridge = regression_model(Ridge())
print(rmse_ridge)
```

68552.72895813311

In [63]:
```
rmse_ransac = regression_model(RANSACRegressor())
print(rmse_ransac)
```

76001.17988173329

In [64]:
```
rmse_rf = regression_model(RandomForestRegressor())
print(rmse_rf)
```

49116.946956809625

In [65]:
```
rmse_xgb_gblinear = regression_model(XGBRegressor(booster='gblinear'))
print(rmse_xgb_gblinear)
```

69699.32045238555

In [66]:
```
rmse_xgb_gbtree = regression_model(XGBRegressor(booster='gbtree'))
print(rmse_xgb_gbtree)
```

47427.75538243391

In [67]:
```
rmse_xgb_dart = regression_model(XGBRegressor(booster='dart'))
print(rmse_xgb_dart)
```

47427.75538243391

In [129...
```
rmse_lgbm_gbdt = regression_model(LGBMRegressor(boosting_type='gbdt'))
print(rmse_lgbm_gbdt)
```

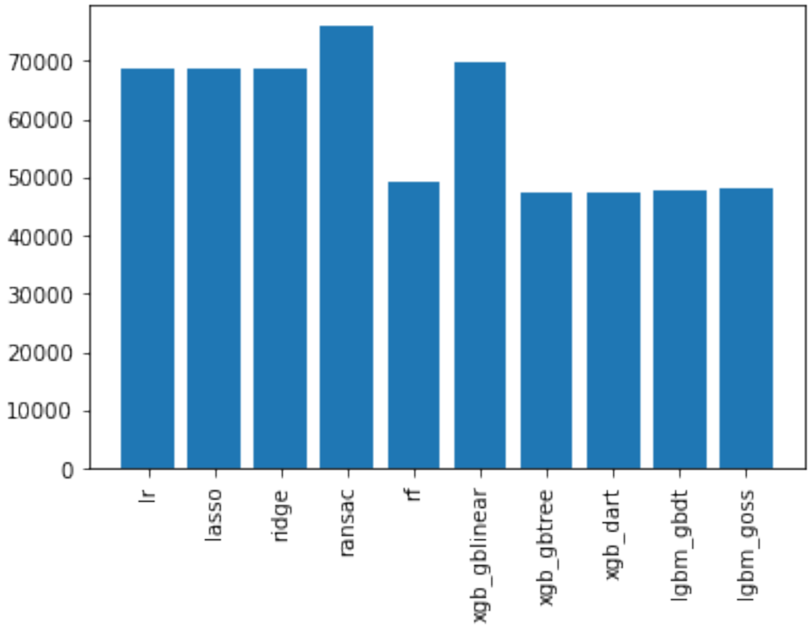
47660.42925114799

In [130...
```
rmse_lgbm_goss = regression_model(LGBMRegressor(boosting_type='goss'))
print(rmse_lgbm_goss)
```

47973.44456533641

In [69]:
```
#regression_model(CatBoostRegressor(verbose=0))
```

In [131…
```
from matplotlib import pyplot as plt
rmse = [rmse_lr, rmse_lasso, rmse_ridge, rmse_ransac, rmse_rf, rmse_xgb_gbl
label = ['lr', 'lasso', 'ridge', 'ransac', 'rf', 'xgb_gblinear', 'xgb_gbtre
plt.bar(label, rmse)
plt.xticks(rotation=90)
plt.show()
```



In [70]:
```
## Select RandomForestRegressor, XGBRegressor with gbtree, LGBMRegressor w
```

3. Hyper parameter tuning

In [34]:
```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import mean_squared_error as MSE
```

In [35]:
```
kfold = KFold(n_splits=10, shuffle=True, random_state=0)
def grid_search(model, params):

    grid_reg = GridSearchCV(model, params, scoring='neg_mean_squared_error

    grid_reg.fit(X_train_transformed, y_train)

    best_params = grid_reg.best_params_
    best_score=grid_reg.best_score_
    rmse=np.sqrt(-best_score)
    print("Best params:", best_params)
    print("Best score:",rmse.round(2))
```

In [36]:

```
def randomized_search(model, params):

    rand_reg = RandomizedSearchCV(model, params, n_iter=20, scoring='neg_me
                                  cv=kfold, n_jobs=-1)

    rand_reg.fit(X_train_transformed, y_train)

    best_model = rand_reg.best_estimator_
    best_params = rand_reg.best_params_
    print("Best params:", best_params)
    best_score = np.sqrt(-rand_reg.best_score_)
    print("Training score: {:.3f}".format(best_score))

    return best_model
```

In [37]:

```
def model_evaluation(model):
    y_train_pred = model.predict(X_train_transformed)
    y_pred = model.predict(X_test_transformed)

    train_rmse = np.sqrt(MSE(y_train, y_train_pred))
    test_rmse = np.sqrt(MSE(y_test, y_pred))

    print('Train rmse:', train_rmse)
    print('Test rmse:', test_rmse)
```

In [52]:

```
#Random Forest
rf = RandomForestRegressor(random_state=1, n_jobs=-1)
```

In [74]:

```
grid_search(rf, {'n_estimators': range(50,400,50)})
```

```
Best params: {'n_estimators': 350}
Best score: 48959.12
```

In [76]:

```
grid_search(rf, {'min_weight_fraction_leaf': np.arange(0, 0.01, 0.001)})
```

```
Best params: {'min_weight_fraction_leaf': 0.0}
Best score: 49166.45
```

In [77]:

```
grid_search(rf, {'max_features': ['auto', 0.3, 0.5, 0.7, 0.9]})
```

```
Best params: {'max_features': 0.7}
Best score: 48629.61
```

In [78]:

```
grid_search(rf, {'max_depth': [None, 2, 5, 7, 10, 15]})
```

```
Best params: {'max_depth': None}
Best score: 49166.45
```

In [53]:
```
#All in one & param 세분화 with randomizedcv
param_grid = {'n_estimators': range(200, 400, 50), 'min_weight_fraction_lea
              'max_features': [0.4, 0.5, 0.6], 'max_depth': [None]}
rf_best = randomized_search(rf, param_grid)
```

```
Best params: {'n_estimators': 350, 'min_weight_fraction_leaf': 0.0001, 'max
_features': 0.6, 'max_depth': None}
Training score: 48298.436
```

In [54]:
```
model_evaluation(rf_best)
#Overfitting 많이 존재 but 성능 좋아짐
```

```
Train rmse: 22028.451559784986
Test rmse: 48323.154093197896
```

In [ ]:
```

```

In [47]:
```
#XGBoost with gbtree(default)
xgb = XGBRegressor(random_state = 1)
```

In [109…
```
grid_search(xgb, {'n_estimators': range(50,400,50)})
```

```
Best params: {'n_estimators': 250}
Best score: 47091.08
```

In [110…
```
grid_search(xgb, {'learning_rate':[0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.(
```

```
Best params: {'learning_rate': 0.2}
Best score: 47120.58
```

In [111…
```
grid_search(xgb, {'max_depth': [3, 4, 6, 8, None]})
```

```
Best params: {'max_depth': 6}
Best score: 47455.56
```

In [112…
```
grid_search(xgb, {'gamma': [0, 0.1, 0.3, 0.5 ,1]})
```

```
Best params: {'gamma': 0}
Best score: 47455.56
```

In [113…
```
grid_search(xgb, {'min_child_weight': range(1,5)})
```

```
Best params: {'min_child_weight': 1}
Best score: 47455.56
```

In [116…
```
grid_search(xgb, {'subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1]})
```

```
Best params: {'subsample': 1}
Best score: 47455.56
```

In [117…

```python
grid_search(xgb, {'colsample_bytree': [0.5, 0.6, 0.7, 0.8, 0.9, 1]})
```

```
Best params: {'colsample_bytree': 1}
Best score: 47455.56
```

In [48]:

```python
#All in one & param 세분화 with randomizedcv
param_grid = {'n_estimators': range(220, 280, 10), 'learning_rate': [0.05,
              'max_depth': [5, 6, 7], 'gamma': [0, 0.05, 1],
              'min_child_weight': [1, 2], 'subsample': [0.9, 0.95, 1],
              'colsample_bytree': [0.9, 0.95, 1]}
xgb_best = randomized_search(xgb, param_grid)
```

```
Best params: {'subsample': 0.9, 'n_estimators': 270, 'min_child_weight': 1,
'max_depth': 7, 'learning_rate': 0.05, 'gamma': 1, 'colsample_bytree': 0.95
}
Training score: 46142.069
```

In [49]:

```python
model_evaluation(xgb_best)
#Overfitting 많이 나아짐 and 성능 좋아짐
```

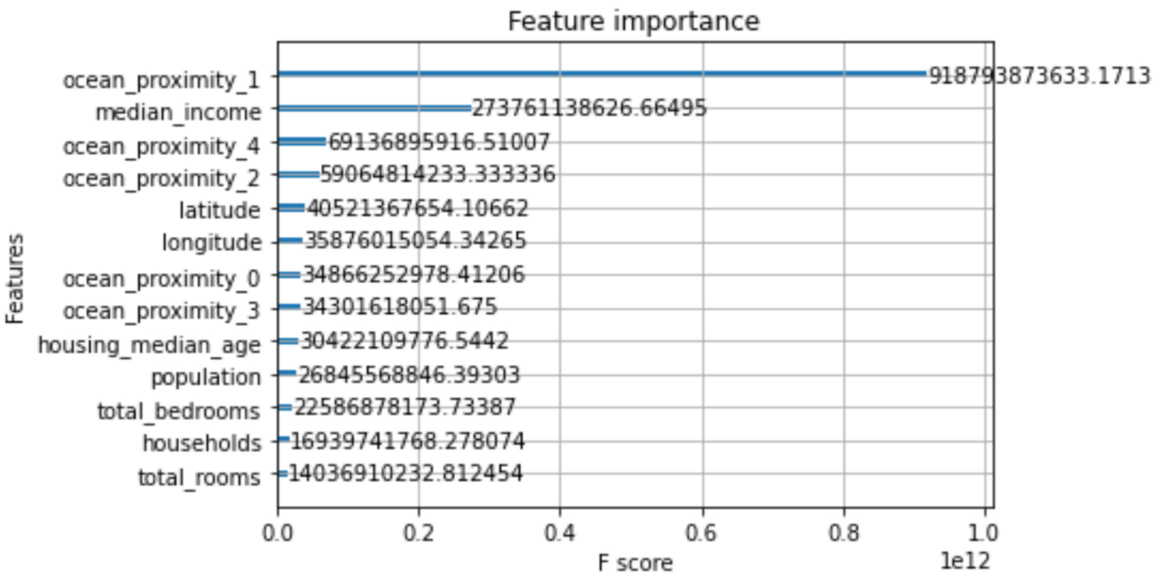
```
Train rmse: 29763.99882834021
Test rmse: 46603.74984376612
```

In [50]:

```python
#partial dependency, feature importance
```

In [51]:

```python
from xgboost import plot_importance
from matplotlib import pyplot
plot_importance(xgb_best,max_num_features=None,importance_type='gain')
pyplot.show()
```



In [ ]:

In [38]:

```
#lgbm regressor
lgbm_gbdt = LGBMRegressor(boosting_type='gbdt', random_state=1)
```

In [133…

```
grid_search(lgbm_gbdt, {'n_estimators': range(50, 400, 50)})
```

```
Best params: {'n_estimators': 350}
Best score: 45875.67
```

In [134…

```
grid_search(lgbm_gbdt, {'learning_rate': [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, (
```

```
Best params: {'learning_rate': 0.2}
Best score: 46994.11
```

In [135…

```
grid_search(lgbm_gbdt, {'max_depth': [3, 4, 6, 8, None]})
```

```
Best params: {'max_depth': None}
Best score: 47678.48
```

In [137…

```
grid_search(lgbm_gbdt, {'min_child_weight': range(1,5)})
```

```
Best params: {'min_child_weight': 1}
Best score: 47678.48
```

In [138…

```
grid_search(lgbm_gbdt, {'subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1]})
```

```
Best params: {'subsample': 0.5}
Best score: 47678.48
```

In [139…

```
grid_search(lgbm_gbdt, {'colsample_bytree': [0.5, 0.6, 0.7, 0.8, 0.9, 1]})
```

```
Best params: {'colsample_bytree': 0.8}
Best score: 47081.62
```

In [44]:

```
#All in one & param 세분화 with randomizedcv
param_grid = {'n_estimators': range(300, 400, 10), 'learning_rate': [0.05,
              'max_depth': [8, 9, None],'min_child_weight': [1, 2], 'subsa
              'colsample_bytree': [0.7, 0.75, 0.8, 0.85]}
lgbm_gbdt_best = randomized_search(lgbm_gbdt, param_grid)
```

```
Best params: {'subsample': 0.55, 'n_estimators': 370, 'min_child_weight': 2
, 'max_depth': 8, 'learning_rate': 0.1, 'colsample_bytree': 0.8}
Training score: 45422.949
```

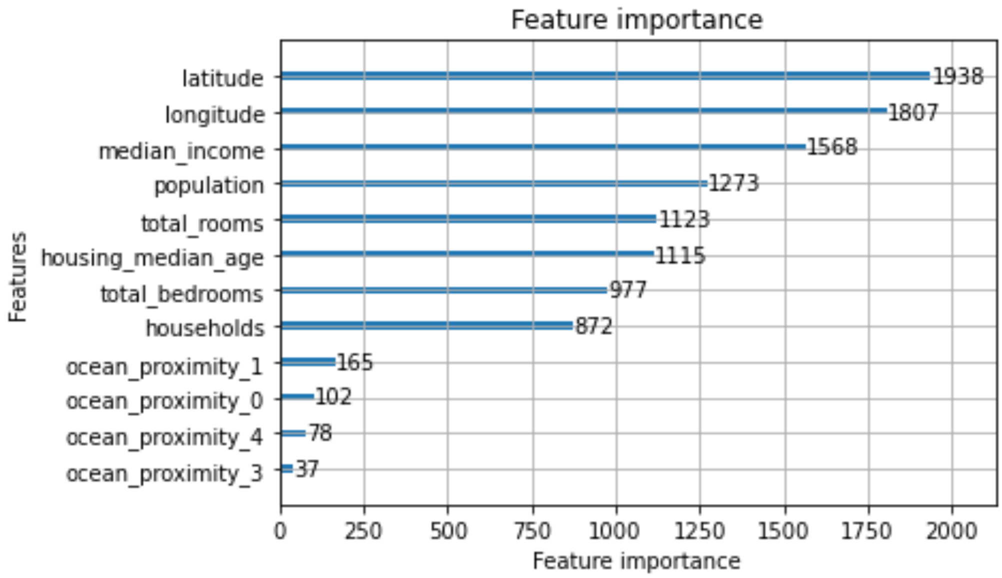
In [45]:

```
model_evaluation(lgbm_gbdt_best)
#성능 good
```

```
Train rmse: 31311.132601504098
Test rmse: 45739.12327785611
```

In [46]:

```
from lightgbm import plot_importance
from matplotlib import pyplot
plot_importance(lgbm_gbdt_best,max_num_features=None)
pyplot.show()
```

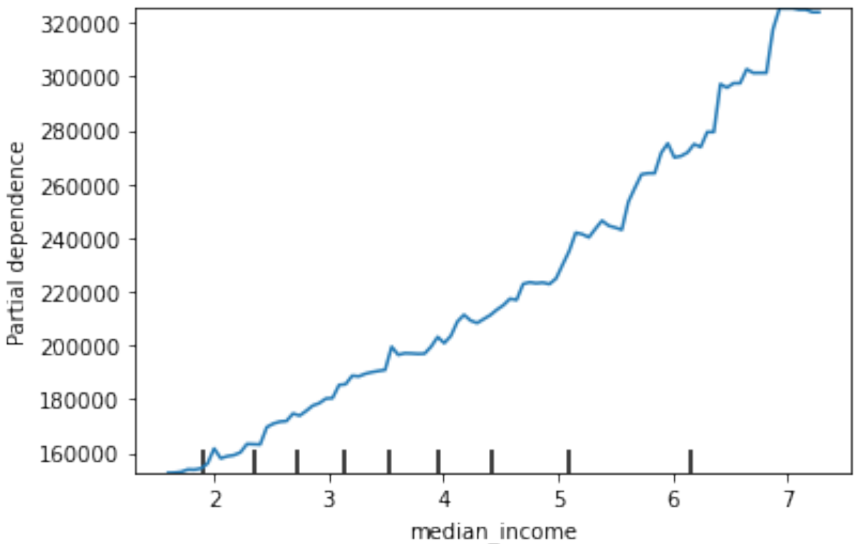


In [186…

```
from sklearn.inspection import plot_partial_dependence
plot_partial_dependence(lgbm_gbdt_best,X_train_transformed,[7])
#매우 직관적인 결과
```

Out[186…

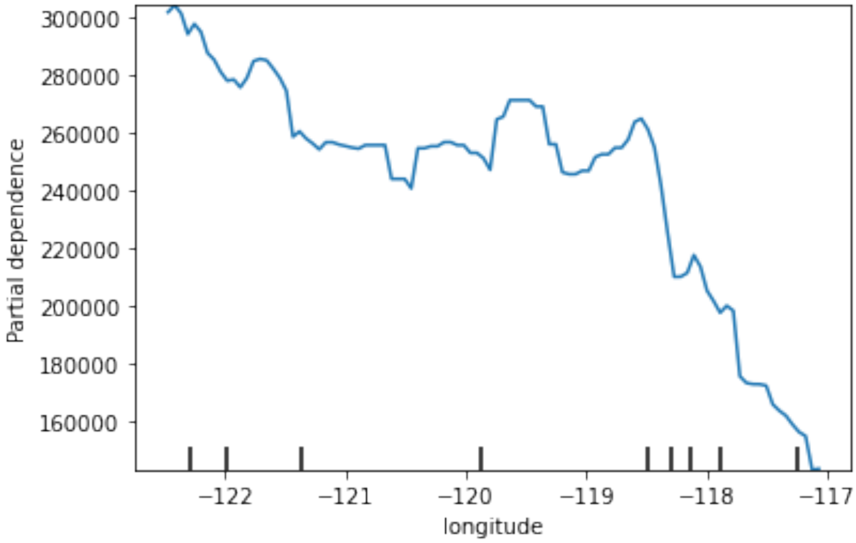
```
<sklearn.inspection._plot.partial_dependence.PartialDependenceDisplay at 0x14fdc5c70>
```



In [187…

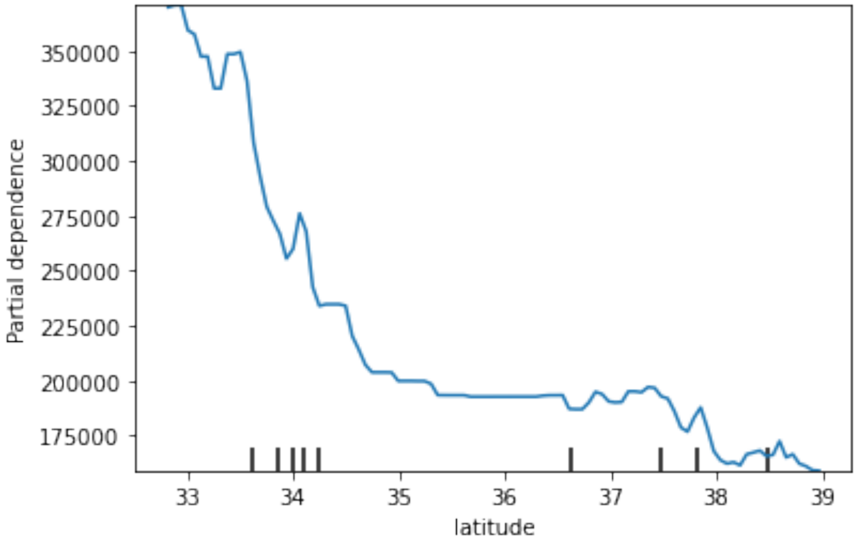
```
plot_partial_dependence(lgbm_gbdt_best,X_train_transformed,[0])
```

Out[187…
```
<sklearn.inspection._plot.partial_dependence.PartialDependenceDisplay at 0x
16c858f40>
```



In [190…
```
plot_partial_dependence(lgbm_gbdt_best,X_train_transformed,[1])
```

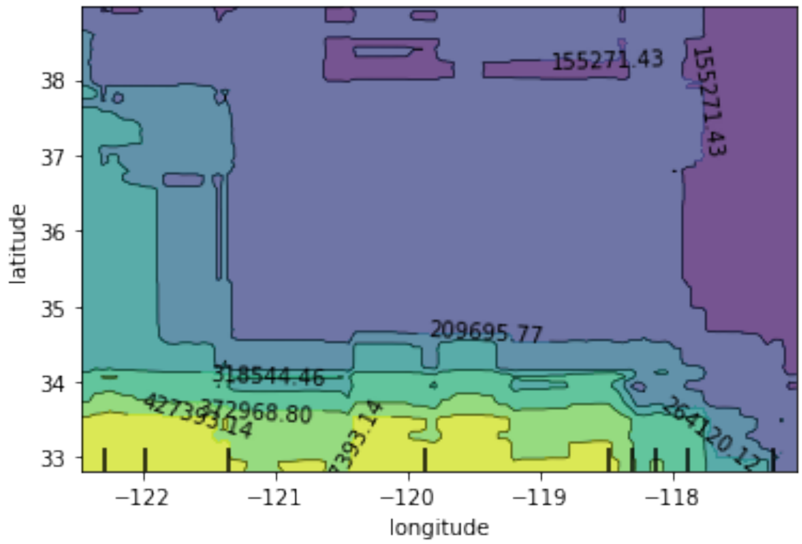
Out[190…
```
<sklearn.inspection._plot.partial_dependence.PartialDependenceDisplay at 0x
16c8f66d0>
```



In [234…
```
plot_partial_dependence(lgbm_gbdt_best, X_train_transformed,[(0,1)])
```

Out[234…

```
<sklearn.inspection._plot.partial_dependence.PartialDependenceDisplay at 0x
2906370d0>
```



In [ ]:

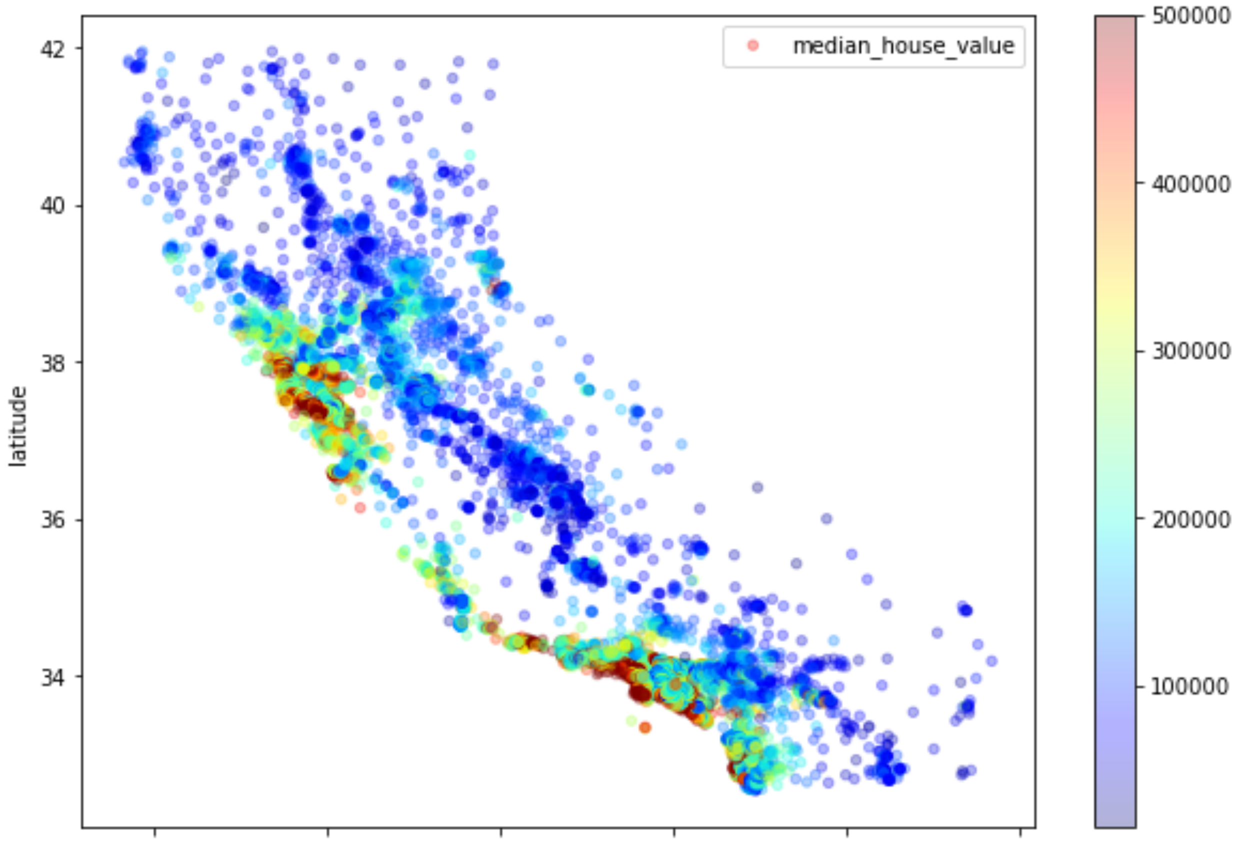
```
#위도와 경도가 높아지면 price는 낮아짐. 확인하기 위해 plot
```

In [39]:

```
import matplotlib.pyplot as plt
housing.plot(kind='scatter',x='longitude',y='latitude',alpha=.3,
             figsize=(10,7),c=housing['median_house_value'],label='median_h
plt.legend()
```

Out[39]:

```
<matplotlib.legend.Legend at 0x14404ffa0>
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [40]:

```
#lgbm regressor
lgbm_goss = LGBMRegressor(boosting_type='goss', random_state=1)
```

In [149…

```
grid_search(lgbm_goss, {'n_estimators': range(50, 400, 50)})
```

```
Best params: {'n_estimators': 350}
Best score: 46722.63
```

In [151…

```
grid_search(lgbm_goss, {'learning_rate': [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, (
```

```
Best params: {'learning_rate': 0.1}
Best score: 47996.34
```

In [153…

```
grid_search(lgbm_goss, {'max_depth': [3, 4, 6, 8, None]})
```

```
Best params: {'max_depth': None}
Best score: 47996.34
```

In [156…

```
grid_search(lgbm_goss, {'min_child_weight': range(1,5)})
```

```
Best params: {'min_child_weight': 4}
Best score: 47940.1
```

In [157…

```
grid_search(lgbm_goss, {'subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1]})
```

```
Best params: {'subsample': 0.5}
Best score: 47996.34
```

In [158…

```
grid_search(lgbm_goss, {'colsample_bytree': [0.5, 0.6, 0.7, 0.8, 0.9, 1]})
```

```
Best params: {'colsample_bytree': 0.8}
Best score: 47516.79
```

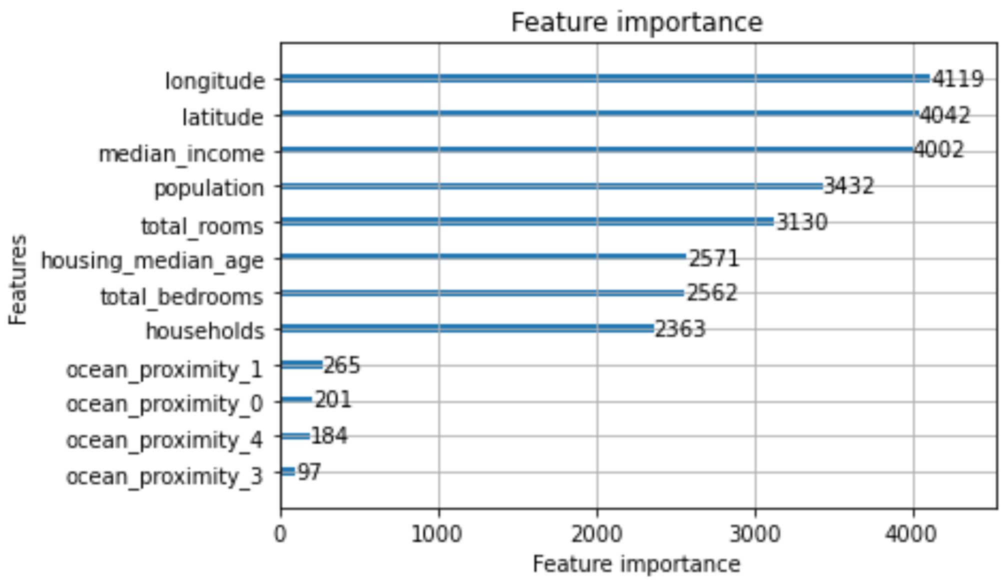
In [41]:
```
#All in one & param 세분화 with randomizedcv
param_grid = {'n_estimators': range(250, 3500, 10), 'learning_rate': [0.05
              'max_depth': [8, 9, None],'min_child_weight': [3, 4, 5], 'sul
              'colsample_bytree': [0.7, 0.75, 0.8, 0.85]}
lgbm_goss_best = randomized_search(lgbm_goss, param_grid)
```

```
Best params: {'subsample': 0.55, 'n_estimators': 900, 'min_child_weight': 3
, 'max_depth': 9, 'learning_rate': 0.05, 'colsample_bytree': 0.8}
Training score: 45888.371
```

In [42]:
```
model_evaluation(lgbm_goss_best)
```

```
Train rmse: 30153.758666898404
Test rmse: 45959.273536084926
```

In [43]:
```
from lightgbm import plot_importance
from matplotlib import pyplot
plot_importance(lgbm_goss_best,max_num_features=None)
pyplot.show()
```



In [ ]:
```

```

4. Ensemble

In [55]:
```
from sklearn.ensemble import VotingRegressor

estimators = []
estimators.append(('rf', rf_best))
estimators.append(('xgb', xgb_best))
estimators.append(('lgbm_gbdt',lgbm_gbdt_best))
estimators.append(('lgbm_goss',lgbm_goss_best))
ensemble = VotingRegressor(estimators)
scores = cross_val_score(ensemble, X_train_transformed, y_train, scoring='r
rmse = np.sqrt(-scores)
print(rmse.mean())
```

```
45222.352208828226
```

In [57]:
```
ensemble.fit(X_train_transformed, y_train)
model_evaluation(ensemble)
```

```
Train rmse: 27277.147221947936
Test rmse: 45467.0632380657
```

In [58]:
```
ensemble_pipeline = Pipeline([('null_imputer', NullValueImputer()),
                              ('ohe',OHEMatrix()),
                              ('ensemble',VotingRegressor([('rf', RandomFor
                                                           ('xgb', XGBRegre
                                                           ('lgbm_gbdt', LG
                                                           ('lgbm_goss', LG
```

In [60]:
```
ensemble_pipeline.fit(X_train,y_train)
y_pred=ensemble_pipeline.predict(X_test)
rmse_cat = MSE(y_test, y_pred)**0.5
print(rmse_cat)
```

```
46224.428506777025
```

In [ ]: