목차

In [1]:

```
# 분석 기본 도구
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')

# 통계 분석 도구
from scipy import stats

# 시각화 도구
import folium
import geojson
from folium import plugins
from shapely.geometry import shape, Point, multipolygon
from shap import TreeExplainer, summary_plot

# 전처리 도구
from sklearn.preprocessing import LabelEncoder

# 학습 도구
# from sklearn.ensemble import RandomForestRegressor
# from sklearn.ensemble import GradientBoostingRegressor
# import lightgbm as lgb
import xgboost as xgb

# 검증 도구
# from sklearn.model_selection import KFold
# from sklearn.model_selection import cross_val_score
# from sklearn.metrics import make_scorer
```

Data Load

In [2]:

```
train = pd.read_csv('input/train.csv')

print(train.shape)
train.head()
```

```
(15035, 21)
```

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_ba
0	0	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	
1	1	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	
2	2	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	
3	3	20140627T000000	257500.0	3	2.25	1715	6819	2.0	0	0	...	7	1715	
4	4	20150115T000000	291850.0	3	1.50	1060	9711	1.0	0	0	...	7	1060	

5 rows × 21 columns

In [3]:

```
test = pd.read_csv('input/test.csv')

print(test.shape)
test.head()
```

```
(6468, 20)
```

Out[3]:

	id	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_ba
0	15035	20141209T000000	3	2.25	2570	7242	2.0	0	0	3	7	2170	
1	15036	20141209T000000	4	3.00	1960	5000	1.0	0	0	5	7	1050	
2	15037	20140512T000000	4	4.50	5420	101930	1.0	0	0	3	11	3890	
3	15038	20150415T000000	3	1.00	1780	7470	1.0	0	0	3	7	1050	
4	15039	20150312T000000	3	2.50	1890	6560	2.0	0	0	3	7	1890	

In [4]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15035 entries, 0 to 15034
Data columns (total 21 columns):
id               15035 non-null int64
date             15035 non-null object
price            15035 non-null float64
bedrooms         15035 non-null int64
bathrooms        15035 non-null float64
sqft_living      15035 non-null int64
sqft_lot         15035 non-null int64
floors           15035 non-null float64
waterfront       15035 non-null int64
view             15035 non-null int64
condition        15035 non-null int64
grade            15035 non-null int64
sqft_above       15035 non-null int64
sqft_basement    15035 non-null int64
yr_built         15035 non-null int64
yr_renovated     15035 non-null int64
zipcode          15035 non-null int64
lat              15035 non-null float64
long             15035 non-null float64
sqft_living15    15035 non-null int64
sqft_lot15       15035 non-null int64
dtypes: float64(5), int64(15), object(1)
memory usage: 2.4+ MB
```

In [5]:

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6468 entries, 0 to 6467
Data columns (total 20 columns):
id               6468 non-null int64
date             6468 non-null object
bedrooms         6468 non-null int64
bathrooms        6468 non-null float64
sqft_living      6468 non-null int64
sqft_lot         6468 non-null int64
floors           6468 non-null float64
waterfront       6468 non-null int64
view             6468 non-null int64
condition        6468 non-null int64
grade            6468 non-null int64
sqft_above       6468 non-null int64
sqft_basement    6468 non-null int64
yr_built         6468 non-null int64
yr_renovated     6468 non-null int64
zipcode          6468 non-null int64
lat              6468 non-null float64
long             6468 non-null float64
sqft_living15    6468 non-null int64
sqft_lot15       6468 non-null int64
dtypes: float64(4), int64(15), object(1)
memory usage: 1010.7+ KB
```

EDA

1. Check Features

In [6]:

```
data = pd.merge(train, test, how='outer')

print(data.shape)
data.head()
```

```
(21503, 21)
```

Out[6]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_ba
0	0	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	
1	1	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	
2	2	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	
3	3	20140627T000000	257500.0	3	2.25	1715	6819	2.0	0	0	...	7	1715	
4	4	20150115T000000	291850.0	3	1.50	1060	9711	1.0	0	0	...	7	1060	

5 rows × 21 columns

In [7]:

```
data['year'] = data['date'].apply(lambda x:x[:4]).astype(int)
data['month'] = data['date'].apply(lambda x:x[4:6]).astype(int)
data['day'] = data['date'].apply(lambda x:x[6:8]).astype(int)
```

In [8]:

```
data.columns
```

Out[8]:

```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15', 'year', 'month', 'day'],
      dtype='object')
```

In [9]:

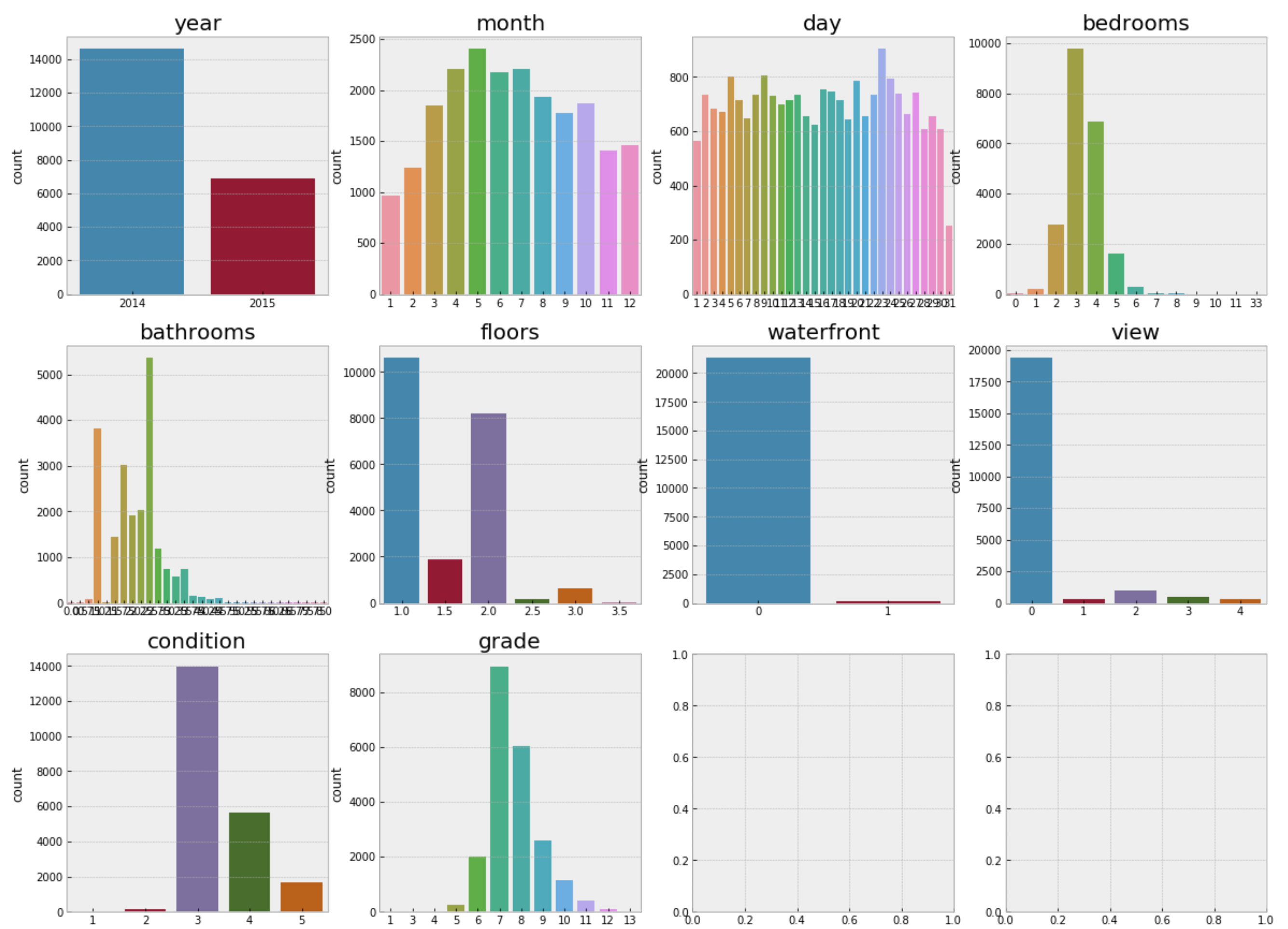
```
data['yr_built'].min()
```

Out[9]:

```
1900
```

In [10]:

```
count_info = ['year', 'month', 'day', 'bedrooms', 'bathrooms', 'floors', 'waterfront', 'view', 'con
dition', 'grade']
fig, axes = plt.subplots(3, 4, figsize=(20, 15))

for r in range(3):
    for c in range(4):
        index = 4 * r + c
        if index == len(count_info):
            break
        sns.countplot(data=data, x=count_info[index], ax=axes[r, c])
        axes[r, c].set_xlabel('')
        axes[r, c].set_title(count_info[index], fontsize=20)
```



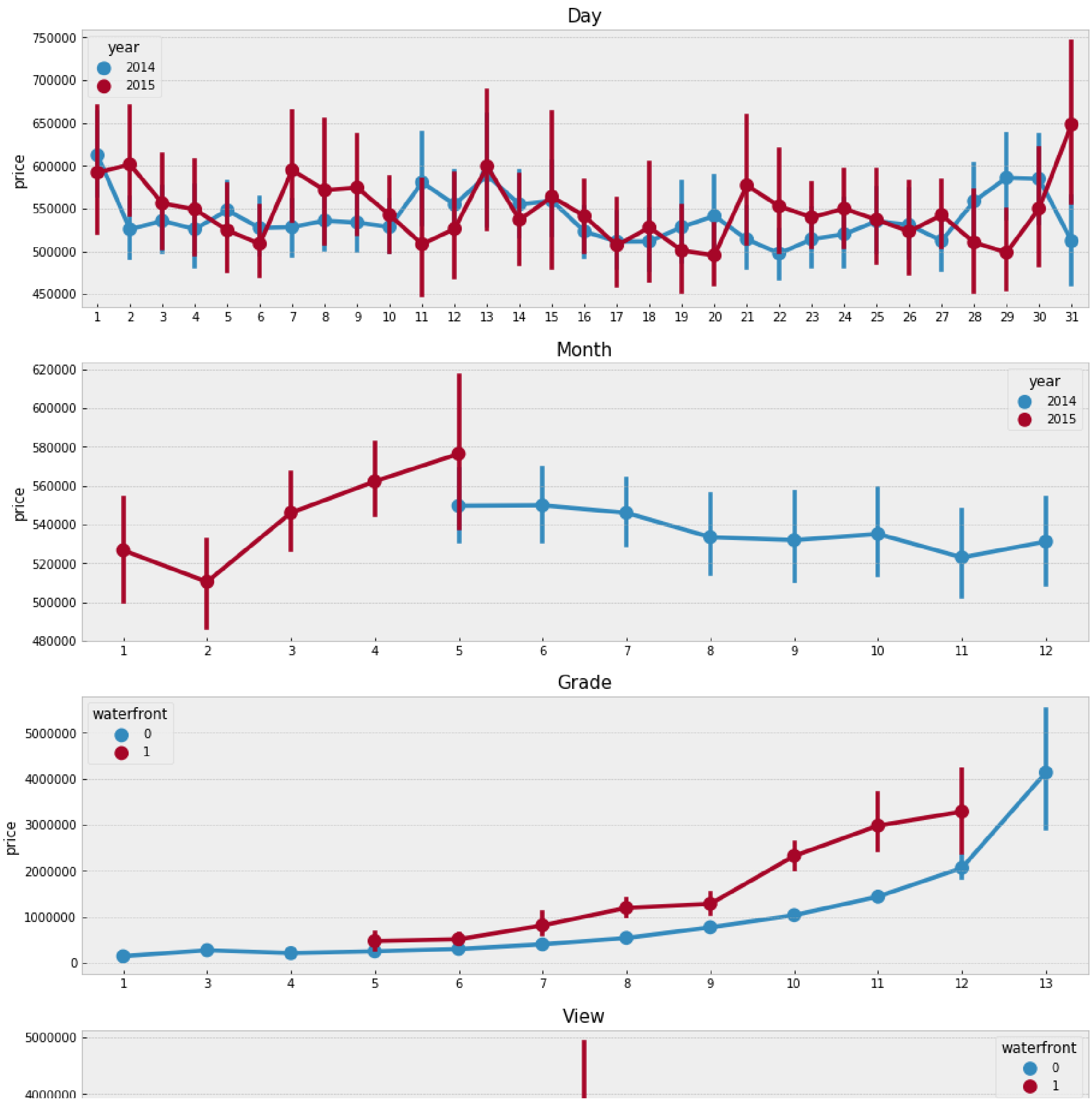
2. Correlation between Discrete Variables

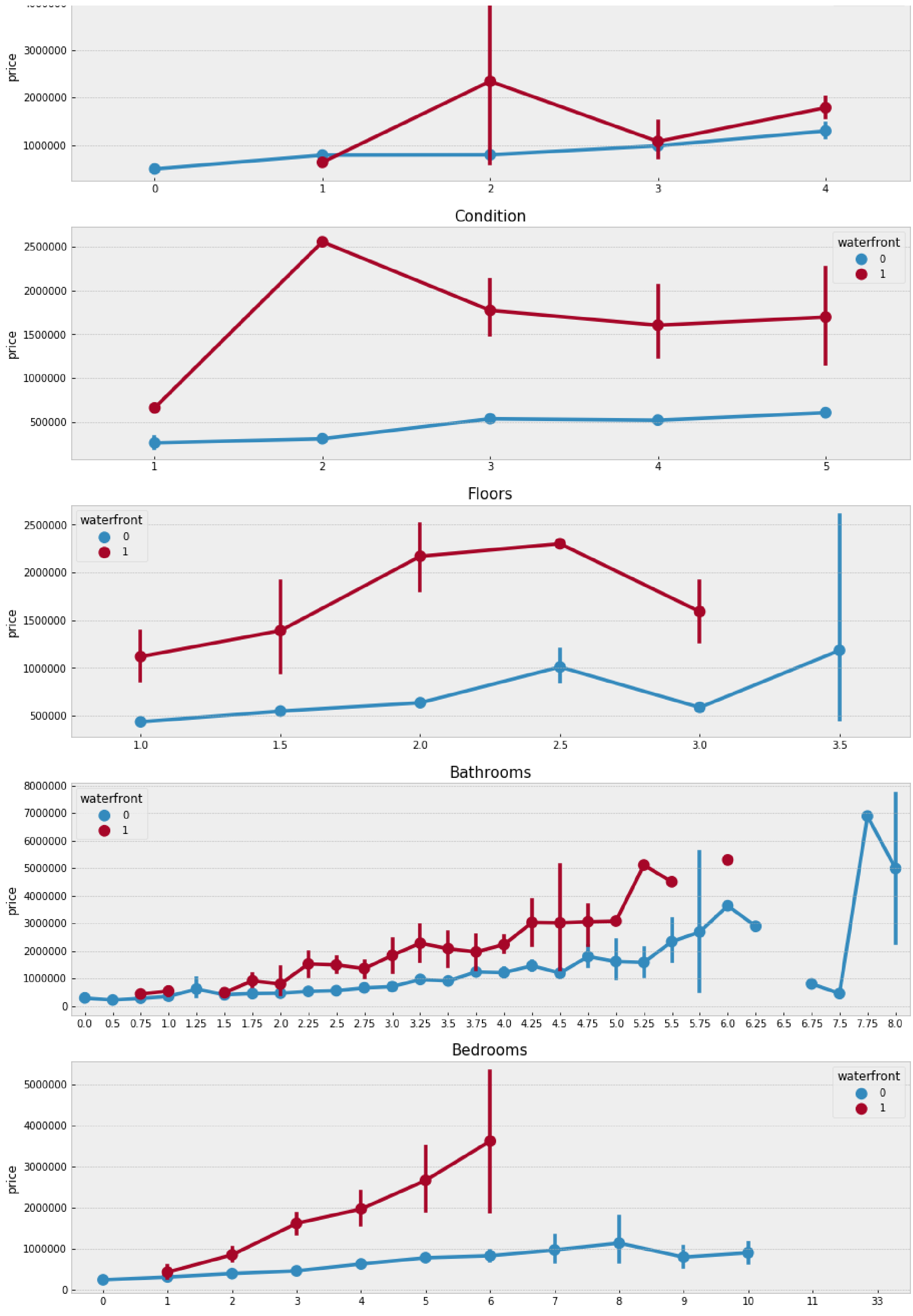
2. Correlation between Discrete Variables

- 데이터는 2014년 5월부터 2015년 5월까지의 집 정보와 가격 정보를 담고 있습니다.
- grade, view, condition에 비례하여 집 가격이 높아지고 있음을 볼 수 있습니다.
- waterfront인지 아닌지에 따른 가격 차이가 존재합니다.
- floors의 소수점은 다락방 등을 의미하는 것으로 보입니다.
- bathrooms는 Taemyung Heo님의 discussion를 참고하면 좋습니다.

In [11]:

```
fig, axes = plt.subplots(nrows=8, figsize=(15, 40))

sns.pointplot(data=data, x='day', y='price', hue='year', ax=axes[0])
sns.pointplot(data=data, x='month', y='price', hue='year', ax=axes[1])
sns.pointplot(data=data, x='grade', y='price', hue='waterfront', ax=axes[2])
sns.pointplot(data=data, x='view', y='price', hue='waterfront', ax=axes[3])
sns.pointplot(data=data, x='condition', y='price', hue='waterfront', ax=axes[4])
sns.pointplot(data=data, x='floors', y='price', hue='waterfront', ax=axes[5])
sns.pointplot(data=data, x='bathrooms', y='price', hue='waterfront', ax=axes[6])
sns.pointplot(data=data, x='bedrooms', y='price', hue='waterfront', ax=axes[7])

title = ['Day', 'Month', 'Grade', 'View', 'Condition', 'Floors', 'Bathrooms', 'Bedrooms']
for i in range(8):
    axes[i].set_xlabel('')
    axes[i].set_title(title[i], fontsize=15)
plt.show()
```





3. Correlation between Continuous Variables

- sqft_living은 실질적으로 살 수있는 공간을 뜻합니다.
- sqft_living > sqft_lot인 집이 많은 것으로 보아, sqft_living은 gross floor area로 계산이 되고 있습니다.

- 만약 어떤 주택의 1층의 면적이 500이고 2층의 면적이 300이라면, sqft_living은 800으로 계산합니다.
    - [참고 링크]()
    - 추가적으로, sqft_living = sqft_above + sqft_basement이며, sqft_basement는 지하실의 면적입니다.
    - sqft_above = 연면적(미국에선 바닥면적)
    - sqft_living을 층 수로 나누어도 여전히 sqft_lot보다 큰 집들이 있습니다.
- 건축 관련 수치들:
    - 건폐율(building coverage) = 건축면적 / 대지면적
    - 연면적(floor area ratio) = 연면적(바닥면적) / 대지면적
- sqft_lot과 sqft_lot15은 단일 속성으로서 price와 상관관계가 적은 것으로 보입니다.

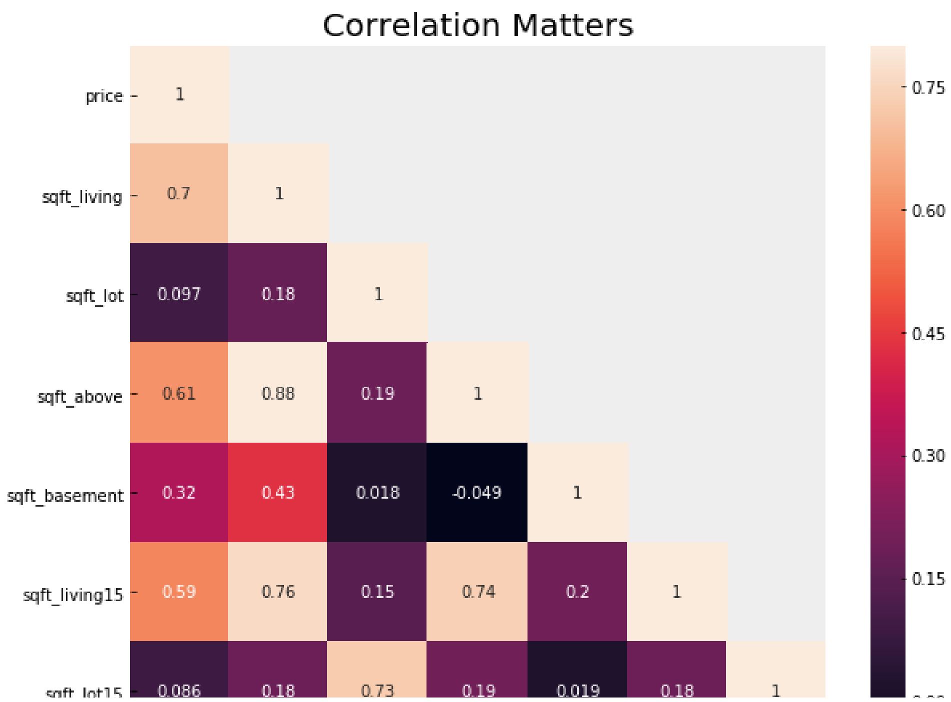
In [12]:

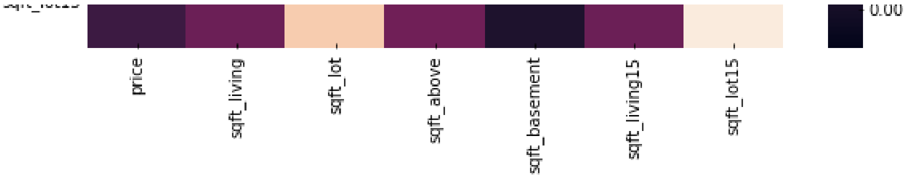
```
corrMatt = train[['price', 'sqft_living', 'sqft_lot', 'sqft_above', 'sqft_basement', 'sqft_living15
', 'sqft_lot15']]
corrMatt = corrMatt.corr()

print(corrMatt)

mask = np.array(corrMatt)
mask[np.tril_indices_from(mask)] = False
```

```
                  price  sqft_living  sqft_lot  sqft_above  sqft_basement  \
price          1.000000     0.702899  0.096793    0.608577       0.322218
sqft_living    0.702899     1.000000  0.176500    0.878736       0.434017
sqft_lot       0.096793     0.176500  1.000000    0.186242       0.017818
sqft_above     0.608577     0.878736  0.186242    1.000000      -0.048623
sqft_basement  0.322218     0.434017  0.017818   -0.048623       1.000000
sqft_living15  0.586419     0.760271  0.147562    0.737795       0.198380
sqft_lot15     0.086384     0.184176  0.728458    0.194226       0.018813

               sqft_living15  sqft_lot15
price               0.586419    0.086384
sqft_living         0.760271    0.184176
sqft_lot            0.147562    0.728458
sqft_above          0.737795    0.194226
sqft_basement       0.198380    0.018813
sqft_living15       1.000000    0.183599
sqft_lot15          0.183599    1.000000
```

In [13]:

```
fig, ax = plt.subplots(figsize=(10, 8))

sns.heatmap(corrMatt, mask=mask, vmax=0.8, square=True, annot=True)
ax.set_title('Correlation Matters', fontsize=20)
plt.show()
```



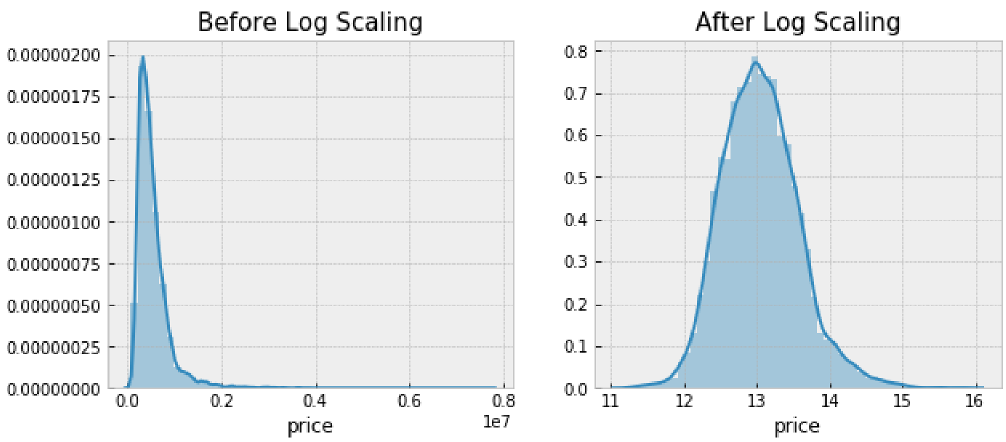


4. Distribution of Price

- log scaling(1을 더한 후 log)한 후에 normal distribution에 더욱 가까워지는 것을 확인할 수 있습니다.

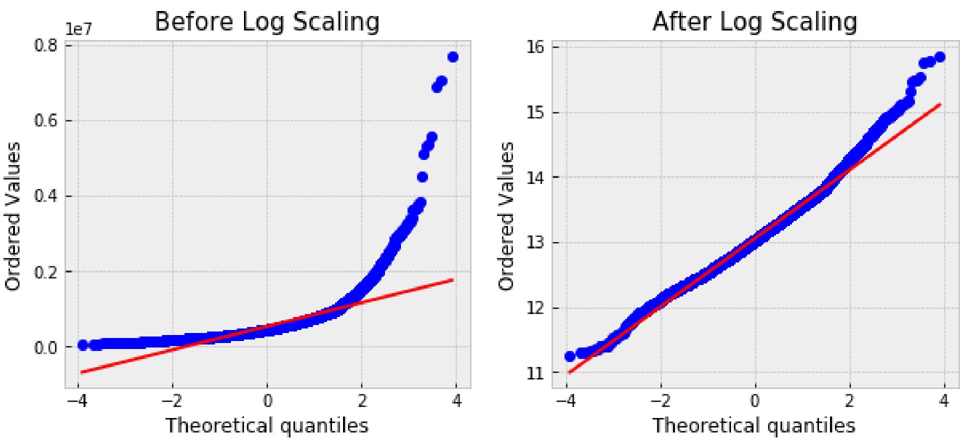
In [14]:

```
fig, axes = plt.subplots(ncols=2, figsize=(10, 4))

sns.distplot(train['price'], ax=axes[0])
axes[0].set_title('Before Log Scaling', fontsize=15)
sns.distplot(np.log1p(train['price']), ax=axes[1])
axes[1].set_title('After Log Scaling', fontsize=15)
plt.show()
```



In [15]:

```
fig, axes = plt.subplots(ncols=2, figsize=(10, 4))

reg = stats.probplot(train['price'], plot=axes[0])
axes[0].set_title('Before Log Scaling', fontsize=15)
reg = stats.probplot(np.log1p(train['price']), plot=axes[1])
axes[1].set_title('After Log Scaling', fontsize=15)
plt.show()
```

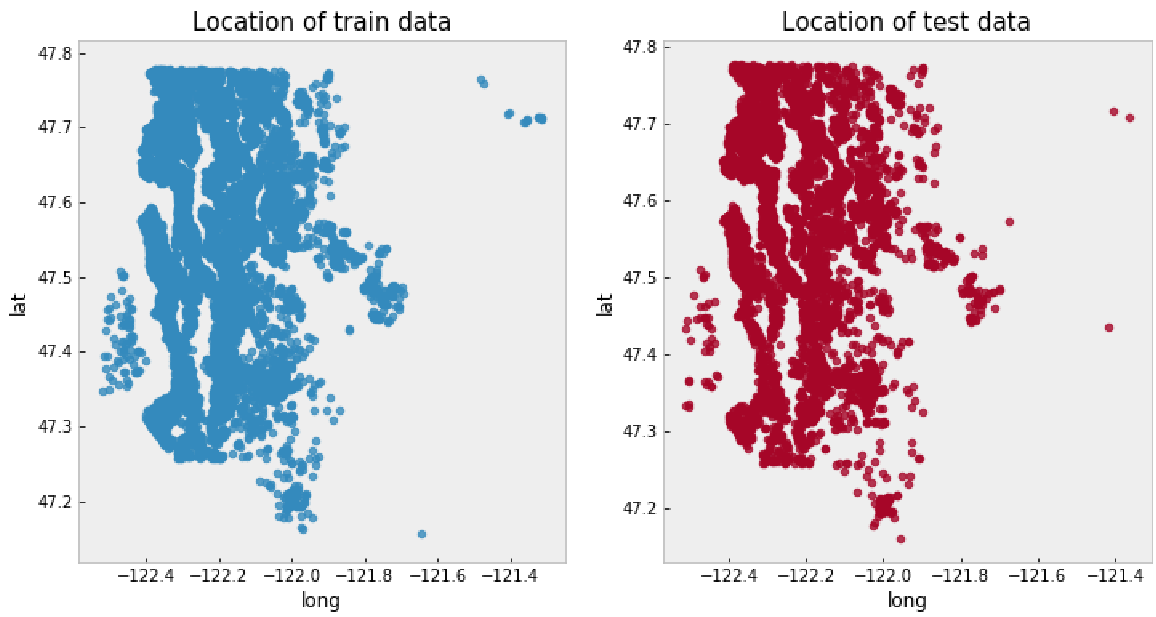


5. Geometry

- 특정 지역의 집 가격이 높음을 관찰할 수 있습니다.

In [16]:

```
fig, axes = plt.subplots(ncols=2, figsize=(12, 6))

sns.regplot(data=train, x='long', y='lat',
            fit_reg=False,
            scatter_kws={'s': 20},
            ax=axes[0])
axes[0].grid(False)
axes[0].set_title('Location of train data', fontsize=15)

sns.regplot(data=test, x='long', y='lat',
            fit_reg=False,
            scatter_kws={'s': 20},
            ax=axes[1])
axes[1].grid(False)
axes[1].set_title('Location of test data', fontsize=15)

plt.show()
```



In [17]:

```
train[(train['long'] > -121.8) & (train['lat'] < 47.2)]
```

Out[17]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	s
2354	2354	20140624T000000	380000.0	3	2.25	1860	15559	2.0	0	0	...	7	1860	

1 rows × 21 columns

In [18]:

```
test[(test['long'] > -121.6) & (test['lat'] < 47.5)]
```

Out[18]:

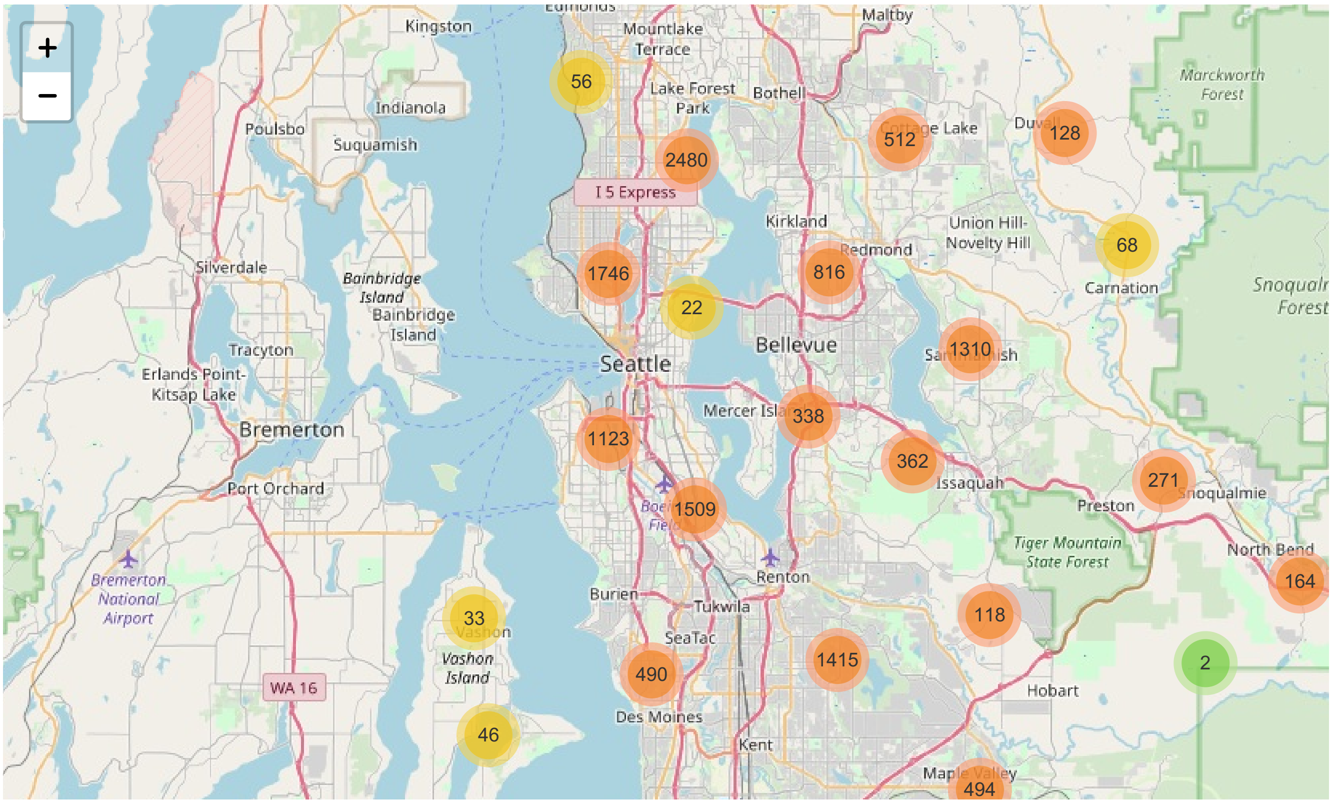
	id	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft
1378	16413	20140521T000000	3	2.75	2100	10362	2.0	0	0	3	9	1510	

In [19]:

```
map_count = folium.Map(location=[train['lat'].mean(), train['long'].mean()],
```

```
map_count = folium.Map(location=[train['lat'].mean(), train['long'].mean()],
                       min_zoom=8,
                       #max_zoom=11,
                       width=960,  # map size scaling
                       height=540)

lat_long_data = train[['lat', 'long']].values.tolist()
h_cluster = folium.plugins.FastMarkerCluster(lat_long_data).add_to(map_count)

map_count
```

Out[19]:



In [20]:

```
zipcode_data = train.loc[:, ['price', 'lat', 'long', 'zipcode']]
zipcode_data['coord'] = [(round(x,6), round(y,6)) for x, y in zip(train['long'], train['lat'])]
del zipcode_data['lat']
del zipcode_data['long']

print(zipcode_data.shape)
zipcode_data.head()
```

(15035, 3)

Out[20]:

	price	zipcode	coord
0	221900.0	98178	(-122.257, 47.5112)
1	180000.0	98028	(-122.233, 47.7379)
2	510000.0	98074	(-122.045, 47.6168)
3	257500.0	98003	(-122.327, 47.3097)
4	291850.0	98198	(-122.315, 47.4095)

In [21]:

```
geo_zipcode = geojson.load(open('geodata/zipcode_king_county.geojson', encoding='utf-8'))
```

In [22]:

```
def zipcode(coord):
    point = Point(coord)
    for feature in geo_zipcode['features']:
        polygon = shape(feature['geometry'])
        if polygon.contains(point):
            return feature['properties']['ZCTA5CE10']
    return 'Outlier'
```

In [23]:

```
# 생성
# zipcode_data['zipcode'] = zipcode_data.coord.apply(zipcode) #주의: 2~3분정도 걸림
# zipcode_data.head()

# 저장
# zipcode_data.to_csv('savefiles/zipcode_data.csv', index=None, encoding='utf-8')

# 불러오기
zipcode_data = pd.read_csv('savefile/zipcode_data.csv', index_col=None, header=0, encoding='utf-8')

print(zipcode_data.shape)
zipcode_data.head()
```

```
(15035, 3)
```

Out[23]:

	price	zipcode	coord
0	221900.0	98178	(-122.257, 47.5112)
1	180000.0	98028	(-122.233, 47.7379)
2	510000.0	98074	(-122.045, 47.6168)
3	257500.0	98003	(-122.327, 47.3097)
4	291850.0	98198	(-122.315, 47.4095)

In [24]:

```
zipcode = pd.pivot_table(zipcode_data, index=['zipcode'])

print(zipcode.shape)
zipcode.head()
```

```
(70, 1)
```

Out[24]:

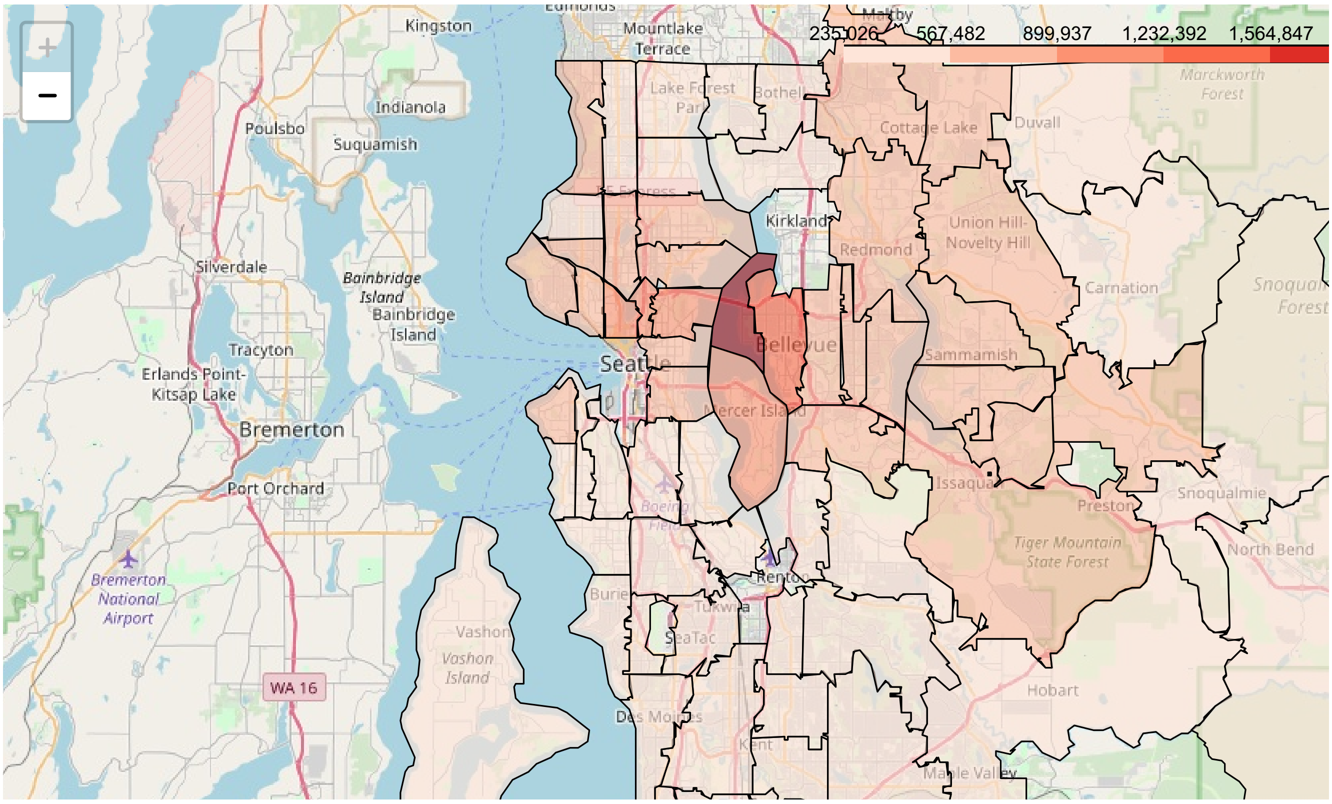
	price
zipcode	
98001	2.800475e+05
98002	2.355189e+05
98003	2.869325e+05
98004	1.395841e+06
98005	7.963691e+05

In [25]:

```
map_price = folium.Map(location=[train['lat'].mean(), train['long'].mean()],
```

```
                       min_zoom=8, # 고정
                       max_zoom=10,
                       #tiles='Stamen Toner',
                       width=960,
                       height=540,
                      )

folium.Choropleth(
    geo_data=geo_zipcode,
    data=zipcode['price'],
    columns=[zipcode.index, zipcode['price']],
    fill_color='Reds',
    fill_opacity=0.6,
    line_opacity=0.6,
    key_on='feature.properties.ZCTA5CE10',
).add_to(map_price)

map_price
```

Out[25]:



Preprocessing

- 저 같은 경우에는 특별한 이상치가 없다고 생각했고, 이상치를 제거할 때 성능 상승의 효과를 보지 못했기 때문에 전혀 제거하지 않고 학습을 시행하였습니다.

1. Feature Engineering

- 좌표와 가격만을 가지고 clustering하던 중에, **zipcode**를 이용한 **평방 대비 가격(price per footage)**을 다룬 Hyun woo Kim님의 Kernel을 참고하게 되었습니다.
- 연도를 쓰지 않고 2014년 5월부터 2015년 5월까지의 시간을 열 세달(1~13)로 쪼개어 재정렬 → 성능 상승
- 건축 관련 수치 추가
  - floor_area_ratio(sqft_living / sqft_lot) 추가 → 성능 상승
  - building_coverage(sqft_above / (floor * sqft_lot)) 추가 → 효과 없음
  - 그 외 여러가지 시도
- 지하실 유무, 다락방 유무, 재건축 유무 등에 대한 feature를 추가할 수도 있으나 역효과
  - feature의 개수가 많아져서 model complexity가 증가하면 overfitting이 강력해지기 때문으로 추측
- 방 관련 feature 추가
  - 방 총 개수 추가 → 성능 상승
  - 다락방, 지하실 유무 등 추가 → 효과 없음
- sqft_lot15 제거 → **상황마다 다른 걸로 결론**
- how_old: 건물이 건축되고 리모델링 된 후, 팔리기까지 걸린 시간

- yr_renovated 제거

2. Log Scaling

- 평방 관련 수치에 대해 log scaling → 성능 상승

3. Label Encoding

- zipcode에 Label Encoding → 성능 상승
- yr_built = yr_built − 1900

In [26]:

```
le = LabelEncoder()
le.fit(train['zipcode'])
le.fit(test['zipcode'])

train['zipcode'] = le.transform(train['zipcode'])
test['zipcode'] = le.transform(test['zipcode'])
```

In [27]:

```
train['price_per_land_area'] = train['price'] / (train['sqft_living'])
price_per_ft = train.groupby(['zipcode'])['price_per_land_area'].agg({'mean', 'std',
'count'}).reset_index()

train = pd.merge(train, price_per_ft, how='left', on='zipcode')
test = pd.merge(test, price_per_ft, how='left', on='zipcode')

del train['price_per_land_area']
```

In [28]:

```
X_train = train.drop(['id', 'price'], axis=1)
y_train = train['price']
y_train = np.log1p(y_train)
X_test = test.drop(['id'], axis=1)
```

In [29]:

```
# Adding features
for df in [X_train, X_test]:
    df['date(new)'] = df['date'].apply(lambda x: int(x[4:8])+800 if x[:4] == '2015' else int(x[4:8])
-400)
    df['how_old'] = df['date'].apply(lambda x: x[:4]).astype(int) - df[['yr_built', 'yr_renovated']
].max(axis=1)
    del df['date']
    del df['yr_renovated']
    df['yr_built'] = df['yr_built'] - 1900
    df['sqft_floor'] = df['sqft_above'] / df['floors']
    df['floor_area_ratio'] = df['sqft_living'] / df['sqft_lot']
    df['rooms'] = df['bedrooms'] + df['bathrooms']

# Log Scaling
log_features = ['sqft_lot', 'sqft_living', 'sqft_above', 'sqft_basement', 'sqft_living15', 'sqft_lo
t15', 'sqft_floor',
                 'mean', 'floor_area_ratio',
                ]
for df in [X_train, X_test]:
    for feature in log_features:
        df[feature] = np.log1p(df[feature])
```

In [30]:

```
train.columns
```

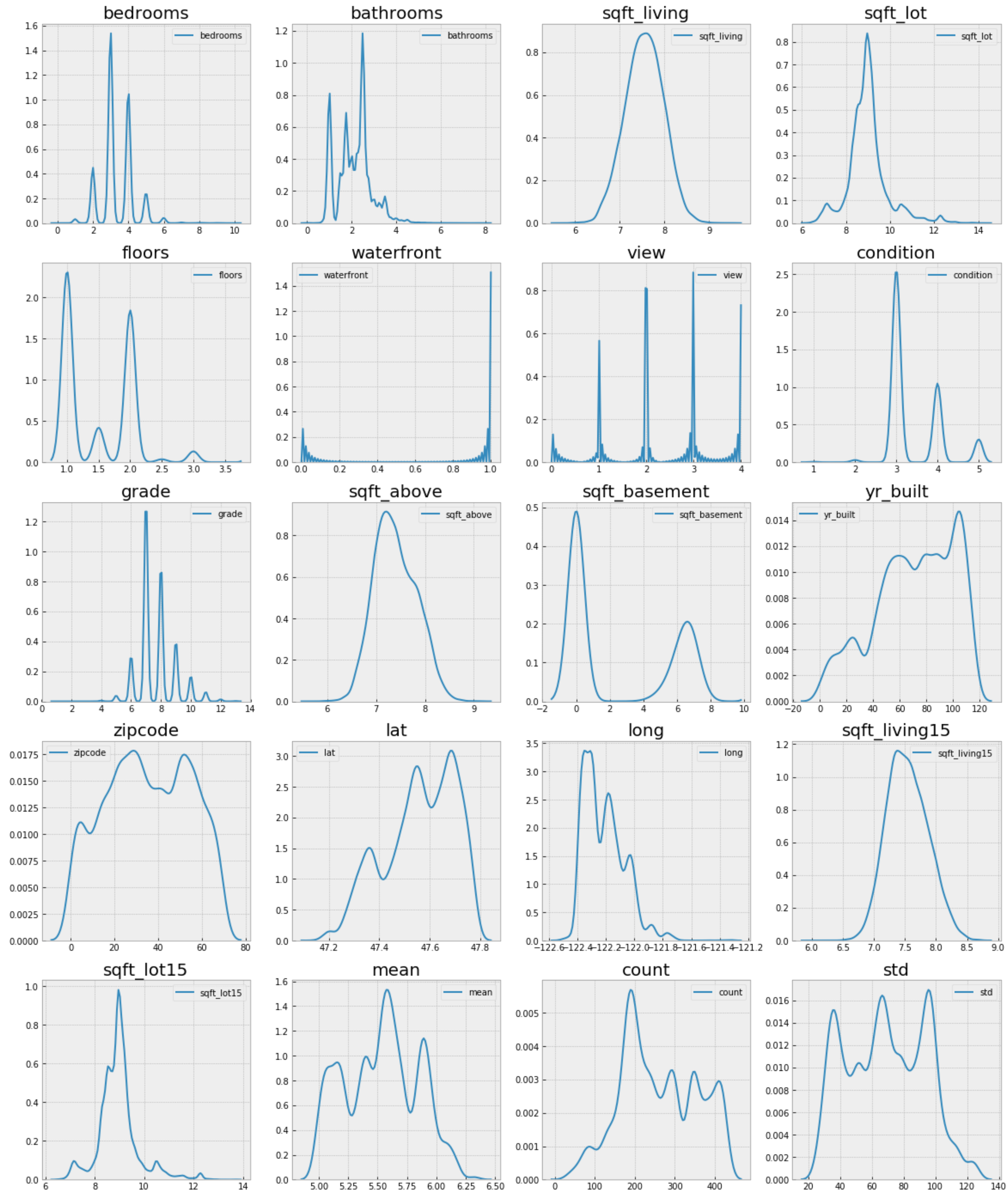
Out[30]:

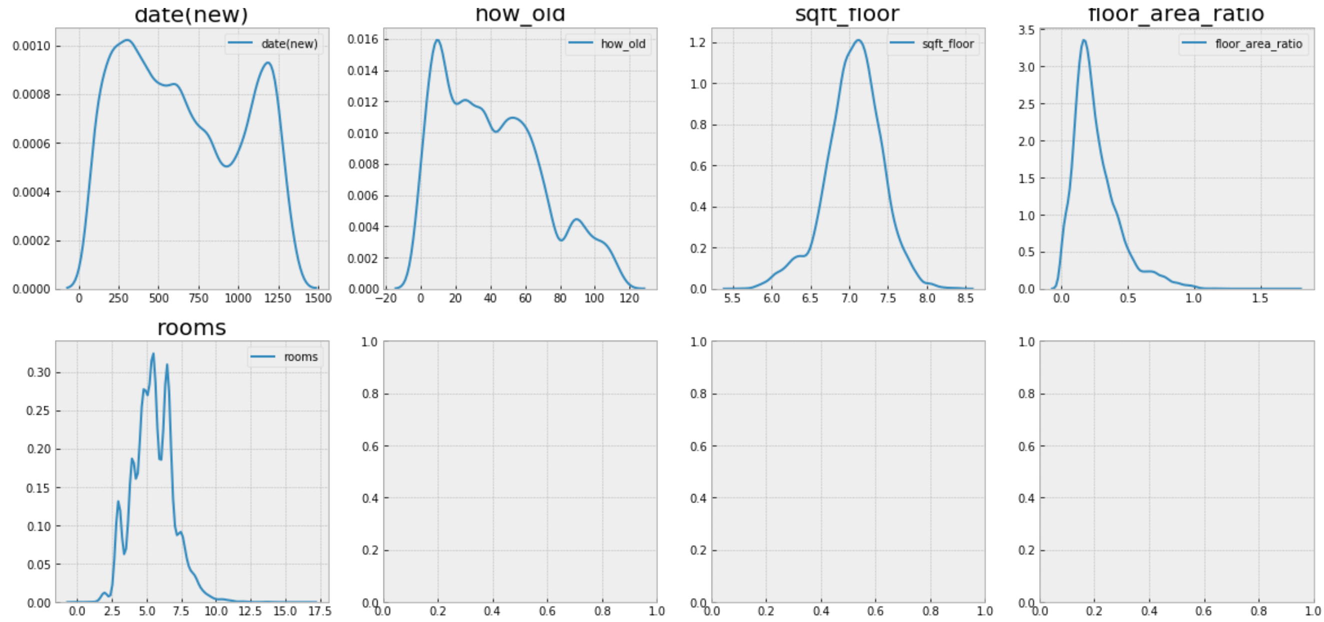
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
```

```
       'lat', 'long', 'sqft_living15', 'sqft_lot15', 'mean', 'count', 'std'],
      dtype='object')
```

In [31]:

```python
rows = (X_train.shape[1]+3) // 4
fig, axes = plt.subplots(rows, 4, figsize=(20, rows*5))
cols = X_train.columns

for r in range(rows):
    for c in range(4):
        index = 4 * r + c
        if index == len(cols):
            break
        sns.kdeplot(X_train[cols[index]], ax=axes[r, c])
        axes[r, c].set_title(cols[index], fontsize=20)
```





Learning

- 여러가지 학습 모델을 사용해 보았는데, 단일 모델로는 xgboost의 성능이 가장 좋았던 것 같습니다.
- 검증 모델은 RMSE(Root Mean Square Error)이며, 수식은 다음과 같습니다.

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

where $y = (y_1, y_2, \ldots, y_n)$ and $\bar{y} = (\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_n)$ denote the vector of actual values and the vector of our predicted values, respectively.

In [32]:

```
def rmse_exp(predictions, dmat):
    labels = dmat.get_label()
    diffs = np.expm1(predictions) - np.expm1(labels)
    mean_squared_diffs = np.mean(np.square(diffs))
    return ('rmse_exp', np.sqrt(mean_squared_diffs))
```

1. Hyperparameter

- 튜닝에는 Grid Search를 이용하였습니다. (과정은 생략)

In [33]:

```
xgb_params = {
    'eta': 0.02,
    'max_depth': 6,
    'subsample': 0.8,
    'colsample_bytree': 0.4,
#      'tree_method': 'gpu_hist',     # 최적 분할점을 찾는 알고리즘 설정 및 GPU 사용
#      'predictor': 'gpu_predictor', # 예측 시에도 GPU 사용
    'objective': 'reg:linear',     # 회귀
    'eval_metric': 'rmse',         # kaggle에서 요구하는 검증모델
    'silent': True,                # 학습 동안 메세지 출력할지 말지
    #'alpha': 0.05,
    #'lambda': 1,
}
```

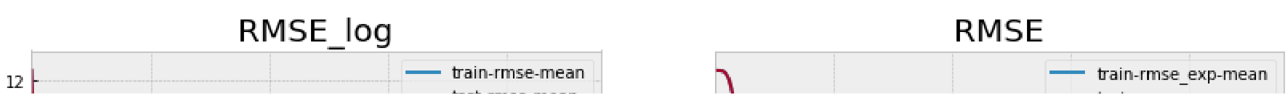
2. Validation

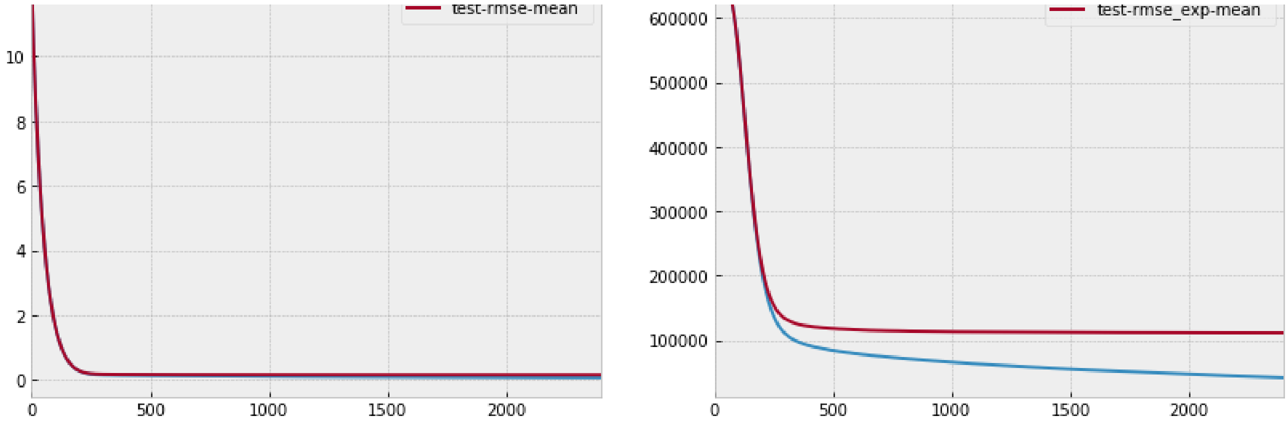
- 우리는 price 대신 $\log(\text{price} + 1)$을 사용했으므로, scoring을 할 때 feval function을 이용하여 $\exp(\bar{y}) - 1$를 취해줍니다.

In [34]:

```
%%time
# transform
print('Start Transforming...')
dtrain = xgb.DMatrix(X_train, y_train)
dtest = xgb.DMatrix(X_test)

# cross validation
print('Valid the Model...')
cv_output = xgb.cv(xgb_params,
                   dtrain,
                   num_boost_round=5000,          # 학습 횟수
                   early_stopping_rounds=100,     # overfitting 방지
                   nfold=5,                       # 높을 수록 실제 검증값에 가까워지고 낮을 수록 빠름
                   verbose_eval=100,              # 몇 번째마다 메세지를 출력할 것인지
                   feval=rmse_exp,                # price 속성을 log scaling 했기 때문에, 다시
exponential
                   maximize=False,
                   show_stdv=False,               # 학습 동안 std(표준편차) 출력할지 말지
                   )

# scoring
best_rounds = cv_output.index.size
score = round(cv_output.iloc[-1]['test-rmse_exp-mean'], 2)

print(f'\nBest Rounds: {best_rounds}')
print(f'Best Score: {score}')


# plotting
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14,5))
cv_output[['train-rmse-mean', 'test-rmse-mean']].plot(ax=ax1)
ax1.set_title('RMSE_log', fontsize=20)
cv_output[['train-rmse_exp-mean', 'test-rmse_exp-mean']].plot(ax=ax2)
ax2.set_title('RMSE', fontsize=20)

plt.show()

print('Running Time is...')
```

```
Start Transforming...
Valid the Model...
[0] train-rmse:12.3084 train-rmse_exp:656012 test-rmse:12.3084 test-rmse_exp:655932
[100] train-rmse:1.65044 train-rmse_exp:557823 test-rmse:1.65116 test-rmse_exp:558065
[200] train-rmse:0.275138 train-rmse_exp:209002 test-rmse:0.282518 test-rmse_exp:217181
[300] train-rmse:0.153358 train-rmse_exp:109622 test-rmse:0.171248 test-rmse_exp:132846
[400] train-rmse:0.141202 train-rmse_exp:91410.3 test-rmse:0.165007 test-rmse_exp:121488
[500] train-rmse:0.133748 train-rmse_exp:83779.4 test-rmse:0.162475 test-rmse_exp:117957
[600] train-rmse:0.127532 train-rmse_exp:78753.4 test-rmse:0.160893 test-rmse_exp:116014
[700] train-rmse:0.122018 train-rmse_exp:74700.3 test-rmse:0.159666 test-rmse_exp:114725
[800] train-rmse:0.117261 train-rmse_exp:71307.4 test-rmse:0.15889 test-rmse_exp:113967
[900] train-rmse:0.113064 train-rmse_exp:68455.4 test-rmse:0.158292 test-rmse_exp:113394
[1000] train-rmse:0.109147 train-rmse_exp:65654.6 test-rmse:0.157852 test-rmse_exp:112919
[1100] train-rmse:0.105448 train-rmse_exp:63110.1 test-rmse:0.157488 test-rmse_exp:112552
[1200] train-rmse:0.101989 train-rmse_exp:60769.3 test-rmse:0.157209 test-rmse_exp:112233
[1300] train-rmse:0.0987172 train-rmse_exp:58666.7 test-rmse:0.156946 test-rmse_exp:111997
[1400] train-rmse:0.0956554 train-rmse_exp:56743.4 test-rmse:0.156748 test-rmse_exp:111908
[1500] train-rmse:0.0927266 train-rmse_exp:54883.3 test-rmse:0.156616 test-rmse_exp:111845
[1600] train-rmse:0.090004 train-rmse_exp:53129 test-rmse:0.156513 test-rmse_exp:111700
[1700] train-rmse:0.0873404 train-rmse_exp:51420.2 test-rmse:0.156456 test-rmse_exp:111655
[1800] train-rmse:0.084853 train-rmse_exp:49865.2 test-rmse:0.156405 test-rmse_exp:111569
[1900] train-rmse:0.082513 train-rmse_exp:48373.3 test-rmse:0.15637 test-rmse_exp:111496
[2000] train-rmse:0.0801854 train-rmse_exp:46960.5 test-rmse:0.156326 test-rmse_exp:111413
[2100] train-rmse:0.0779942 train-rmse_exp:45536.7 test-rmse:0.156318 test-rmse_exp:111299
[2200] train-rmse:0.075811 train-rmse_exp:44179.4 test-rmse:0.156309 test-rmse_exp:111248
[2300] train-rmse:0.0737592 train-rmse_exp:42963.6 test-rmse:0.156304 test-rmse_exp:111199
[2400] train-rmse:0.071711 train-rmse_exp:41709.2 test-rmse:0.156278 test-rmse_exp:111160

Best Rounds: 2395
Best Score: 111155.81
```






```
Running Time is...
Wall time: 4min 15s
```

3. Prediction

- 마찬가지로 예측값에도 $\exp(\bar{y}) + 1$를 취해줍니다.

In [35]:

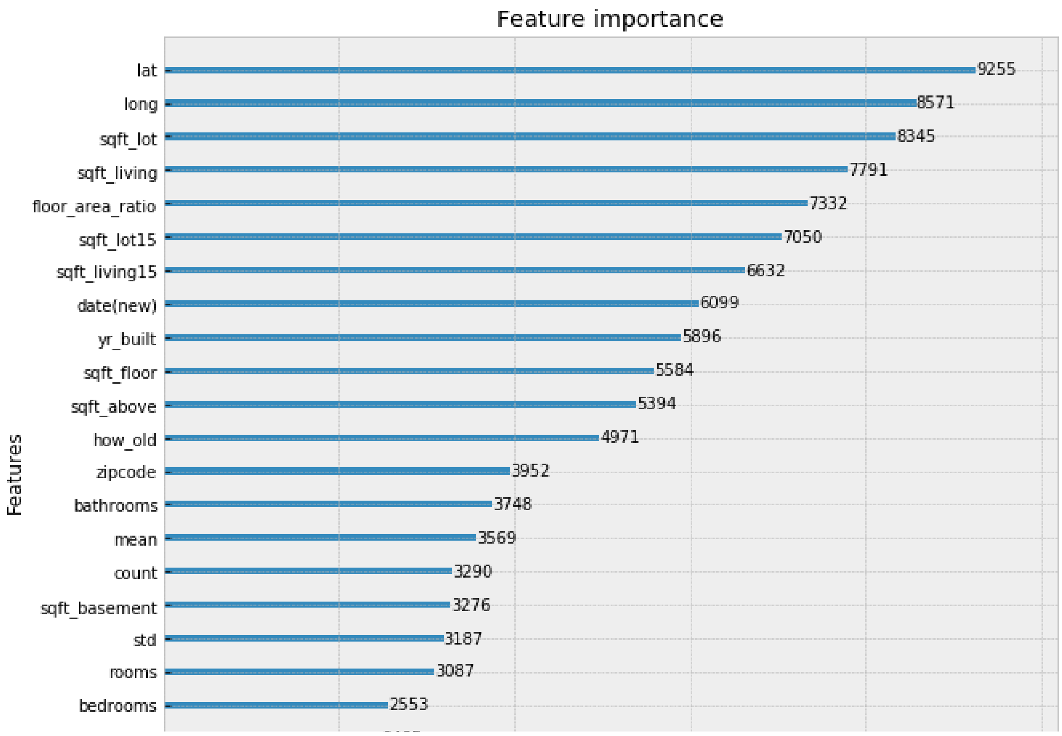
```
model = xgb.train(xgb_params, dtrain, num_boost_round = best_rounds)
y_pred = model.predict(dtest)
y_pred = np.expm1(y_pred)
```

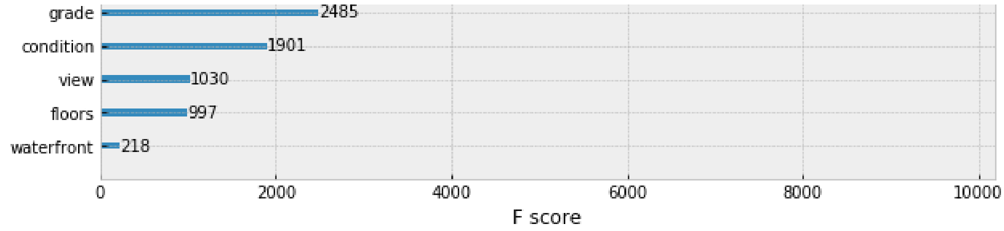
4. Feature Importance

- waterfront는 feature importance가 낮은데도 불구하고 시각화나 실험단계에서 집 가격에 상당한 영향을 미치는 것을 확인하였는데, 이는 전처리를 덜한 탓도 있지만 waterfront인 집과 아닌 집의 수 차이가 많이 나기 때문으로 생각합니다.
- EDA에서 sqft_lot 속성은 price와 낮은 관계성을 보였는데, feature importance에서는 굉장히 높은 중요도를 띕니다.

In [36]:

```
fig, ax = plt.subplots(figsize=(10,10))
xgb.plot_importance(model, ax=ax)

plt.show()
```

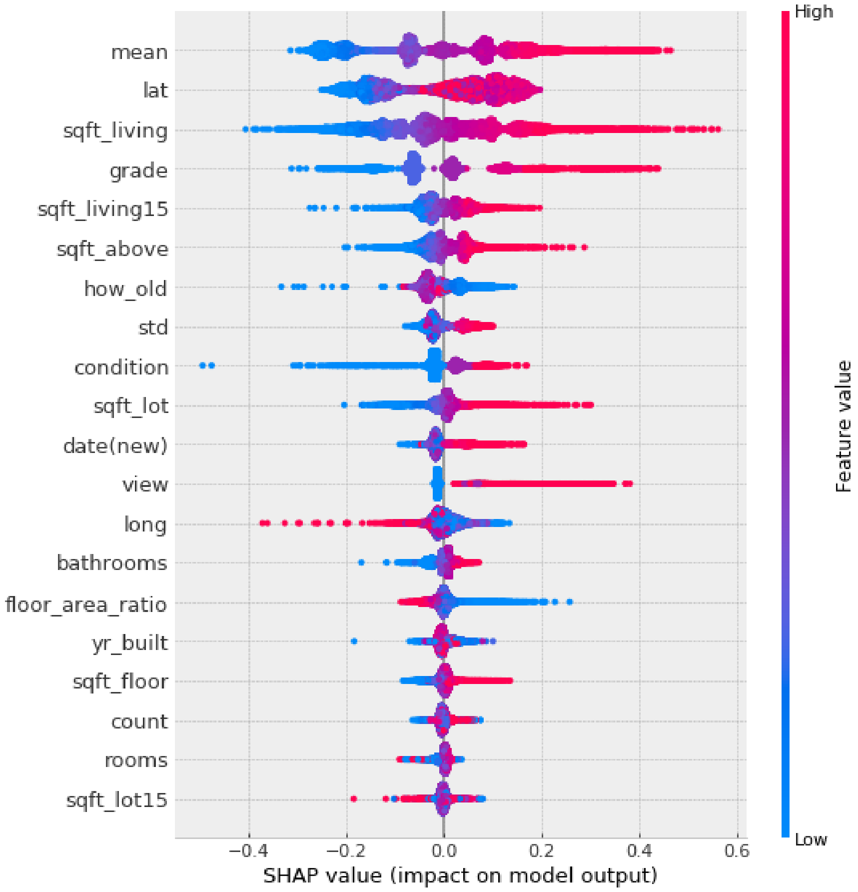




- Shap 라이브러리를 이용하여 각 Feature마다 회귀에 어떻게 영향을 주는지 확인합니다.

In [37]:

```
explainer = TreeExplainer(model)
shap_values = explainer.shap_values(X_train)
summary_plot(shap_values, X_train, title='Train')
```



Submission

- submission sample를 확인하고 그 양식에 맞게 submission 파일을 만들어 제출합니다.

In [38]:

```
sample_submission = pd.read_csv('input/sample_submission.csv')

print(sample_submission.shape)
sample_submission.head()
```

```
(6468, 2)
```

Out[38]:

	id	price
0	15035	100000

	id	price
1	15036	100000
2	15037	100000
3	15038	100000
4	15039	100000

In [39]:

```
submission = pd.DataFrame(data = {'id': test['id'], 'price': y_pred})

print(submission.shape)
submission.head()
```

```
(6468, 2)
```

Out[39]:

	id	price
0	15035	5.188972e+05
1	15036	4.811997e+05
2	15037	1.387298e+06
3	15038	3.068885e+05
4	15039	3.219448e+05

In [40]:

```
submission.to_csv(f'result/submission({score}).csv', index=False)
```