

```
In [14]: 1 import numpy as np
2 import pandas as pd
3 from datetime import datetime
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 import scipy.stats as sps
7
8 from sklearn.model_selection import train_test_split, RandomizedSearchCV
9 from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder
10 from sklearn.linear_model import LinearRegression, Lasso, Ridge
11 from sklearn.metrics import mean_squared_error, make_scorer
12 from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
13 import xgboost as xgb
```

## 1. Знакомство с данными

```
In [2]: 1 data = pd.read_csv('houses_train.csv')
```

```
In [3]: 1 data.head()
```

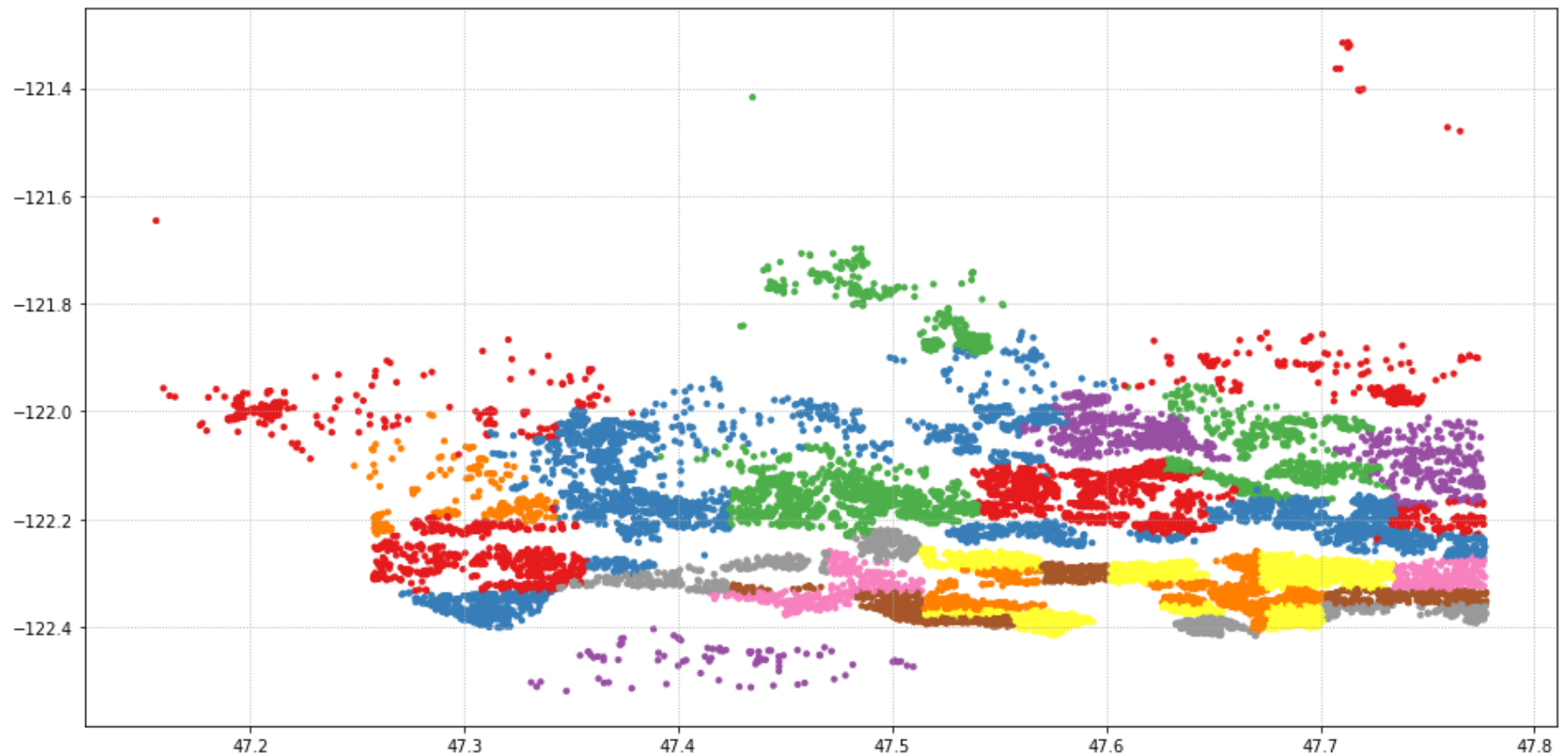
```
Out[3]:
```

|   | id    | date            | price    | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | ... | grade | sqft_above | sqft_basement |
|---|-------|-----------------|----------|----------|-----------|-------------|----------|--------|------------|------|-----|-------|------------|---------------|
| 0 | 3392  | 20151013T000000 | 221900.0 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0          | 0    | ... | 7     | 1180       | 0             |
| 1 | 18295 | 20151209T000000 | 538000.0 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0          | 0    | ... | 7     | 2170       | 400           |
| 2 | 14569 | 20151209T000000 | 604000.0 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0          | 0    | ... | 7     | 1050       | 910           |
| 3 | 14081 | 20160218T000000 | 510000.0 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0          | 0    | ... | 8     | 1680       | 0             |
| 4 | 6725  | 20150627T000000 | 257500.0 | 3        | 2.25      | 1715        | 6819     | 2.0    | 0          | 0    | ... | 7     | 1715       | 0             |

5 rows × 21 columns

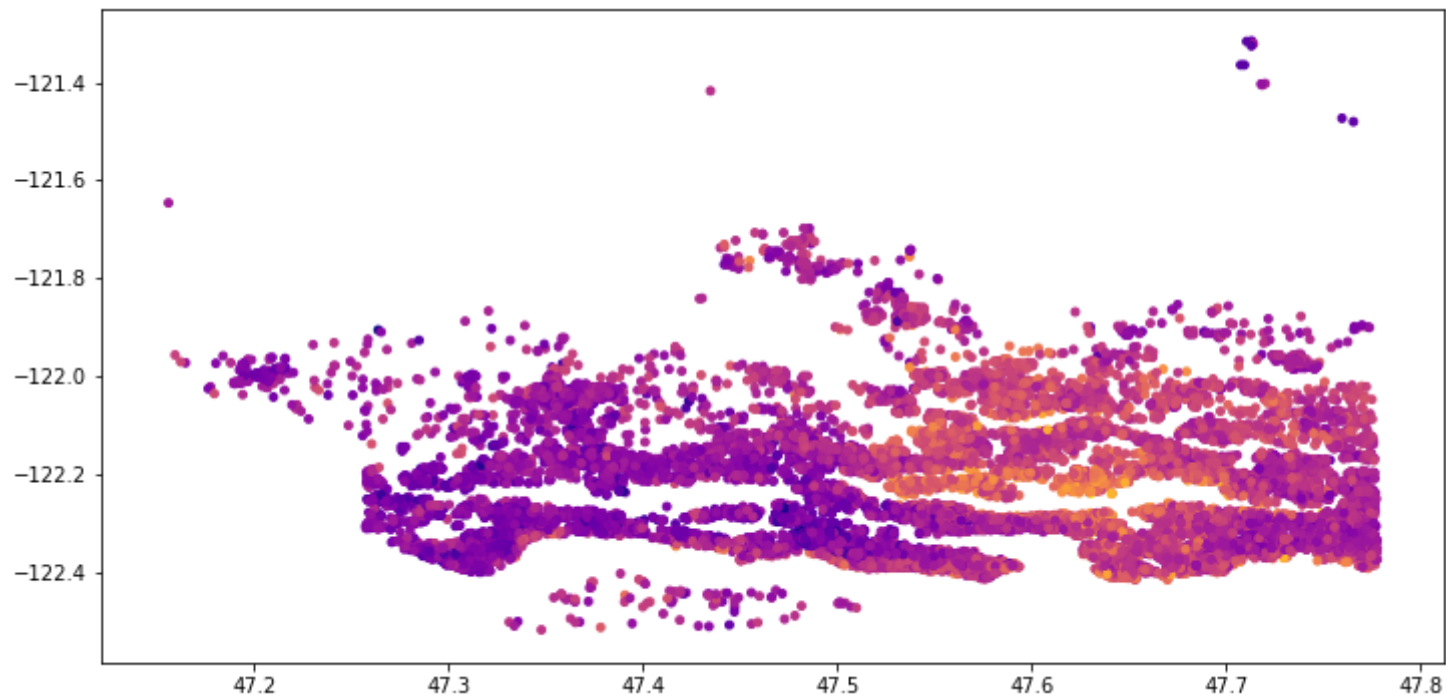
Посмотрим, как данные разделены по районам:

```
In [4]: 1 plt.figure(figsize=(16, 8))
        2 plt.scatter(data['lat'], data['long'], c=data['zipcode'], cmap='Set1', s=10)
        3 plt.grid(linestyle=':');
```



Нарисуем карту, где цвет отражает цену, и увидим самый дорогой район города.

```
In [5]: 1 plt.figure(figsize=(12, 6))
2         plt.scatter(data['lat'], data['long'], c=np.log(data['price']), s=15, cmap='plasma')
3         max_price = data['price'].idxmax()
```



`sqft_above + sqft_basement` почти везде равно `sqft_living` (с небольшой погрешностью), поэтому столбец `sqft_above` удалим

```
In [6]: 1 target = data['price']
2         print((abs(data['sqft_above'] + data['sqft_basement']
3                   - data['sqft_living']) / data['sqft_living']
4               < 0.04).all())
```

True

## 2. Работа с признаками

Преобразование фичей:

- вместо категориального признака `zipcode` введём два признака - один равный усреднённой цене на все дома с данным `zipcode` (т.е. в данном районе) и другой - то же, только средняя цена за квадратный метр. Сохраним отображение из `zipcode` в средние цены, чтобы потом использовать на тестовой выборке.
- Найдём самый "элитный" район города (по средней цене домов), найдём его центр и посчитаем расстояние от каждого из домов до этого центра.
- С признаками-датами сделаем следующее:
  - заменим дату постройки на возраст дома на момент продажи
  - заменим год реновации на два признака - перестраивался ли дом в принципе (бинарный) и время, прошедшее с реновации на момент продажи (или время с постройки, если `was_renovated = 0`)
  - также добавим отдельно год и месяц, в которые был продан дом

Добавим следующие признаки (поможем модели делить и умножать существующие):

- какой процент площади занимает подвальное помещение
- во сколько раз площадь жилья и участка соседей больше/меньше собственного
- среднюю площадь комнаты
- и, наконец, основная идея: так все такие признаки, как оценка состояния, вид и т.д. влияют на цену квадратного метра, а общая цена дома пропорциональна ей (т.е. маленький хороший дом может стоить так же, как большая развалюха), то логично добавить признаки, умноженные на площадь жилья и площадь участка. Но я не стала ограничиваться логичными (как `grade` и `view`), а умножила все, и это дало неплохой результат. Кроме того, для признаков `condition`, `bedrooms`, `bathrooms`, `floors` и `grade` я добавила их экспоненциальную шкалу. Но потом, увидев в итоговой модели, что их значимость практически нулевая, удалила, и они остались только в произведении с площадью.

In [7]:

```
1  def transform_data(data,
2                      prices_by_zipcode,
3                      prices_per_meter_by_zipcode,
4                      lat_max, long_max):
5      # преобразования
6      data['prices_by_zipcode'] = data['zipcode'].map(prices_by_zipcode)
7      data['prices_per_meter_by_zipcode'] = data['zipcode'].map(prices_per_meter_by_zipcode)
8      data['distance_from_rublevka'] = np.sqrt((data['lat'] - lat_max)**2 + (data['long'] - long_max)**2)
9      data = data.drop(['id', 'sqft_above', 'zipcode'], axis=1)
10
11     # даты
12     data['age'] = pd.to_datetime(data['date']).dt.year - data['yr_built']
13     data['was_renovated'] = (data['yr_renovated'] != 0)
14     data.loc[data['yr_renovated'] == 0, 'yr_renovated'] = data['yr_built']
15     data['years_from_renovation'] = pd.to_datetime(data['date']).dt.year - data['yr_renovated']
16     data['year_sold'] = pd.to_datetime(data['date']).dt.year
17     data['month_sold'] = pd.to_datetime(data['date']).dt.month
18     data = data.drop(['date', 'yr_built', 'yr_renovated'], axis=1)
19
20     # просто те же признаки, но в качестве показателей степени
21     for feature in ['condition', 'bedrooms', 'bathrooms', 'floors']:
22         data[feature + '_exp_scale'] = 10.0**((data[feature] - 1)
23         data['grade_exp_scale'] = 2.0**((data['grade'] - 1)
24
25     # новые признаки
26     data['basement_percentage'] = data['sqft_basement'] / data['sqft_living']
27     data['compare_to_neighbour_sqft_living'] = data['sqft_living'] / data['sqft_living15']
28     data['compare_to_neighbour_sqft_lot'] = data['sqft_lot'] / data['sqft_lot15']
29     data['mean_room_sqft'] = data['sqft_living'] / data['bedrooms']
30
31     # все признаки, увеличенные пропорционально площади дома
32     for feature in ['bedrooms', 'bathrooms',
33                     'floors',
34                     'waterfront', 'view',
35                     'condition', 'grade',
36                     'lat', 'long', 'sqft_living15', 'sqft_lot15',
37                     'prices_by_zipcode', 'prices_per_meter_by_zipcode',
38                     'distance_from_rublevka',
39                     'age', 'was_renovated', 'years_from_renovation'] + [
40     f + '_exp_scale' for f in ['condition', 'bedrooms', 'bathrooms', 'floors', 'grade']]:
41         data[feature + '_dot_sqft_living'] = data[feature] * data['sqft_living']
```

```

42         data[feature + '_dot_sqft_lot'] = data[feature] * data['sqft_lot']
43     data = data.drop([f + '_exp_scale' for f in [
44         'condition', 'bedrooms', 'bathrooms', 'floors', 'grade']], axis=1)
45
46     return data

```

В преобразовании данных мы используем расстояние до центра самого дорогого района города. Координаты центра передаются в функцию преобразования ( lat\_max , long\_max ), поэтому сейчас нам необходимо их найти. Для этого сгруппируем цены по признаку zipcode , найдём среднее для каждого, из средних возьмём максимум и усредним координаты уже отдельно для него.

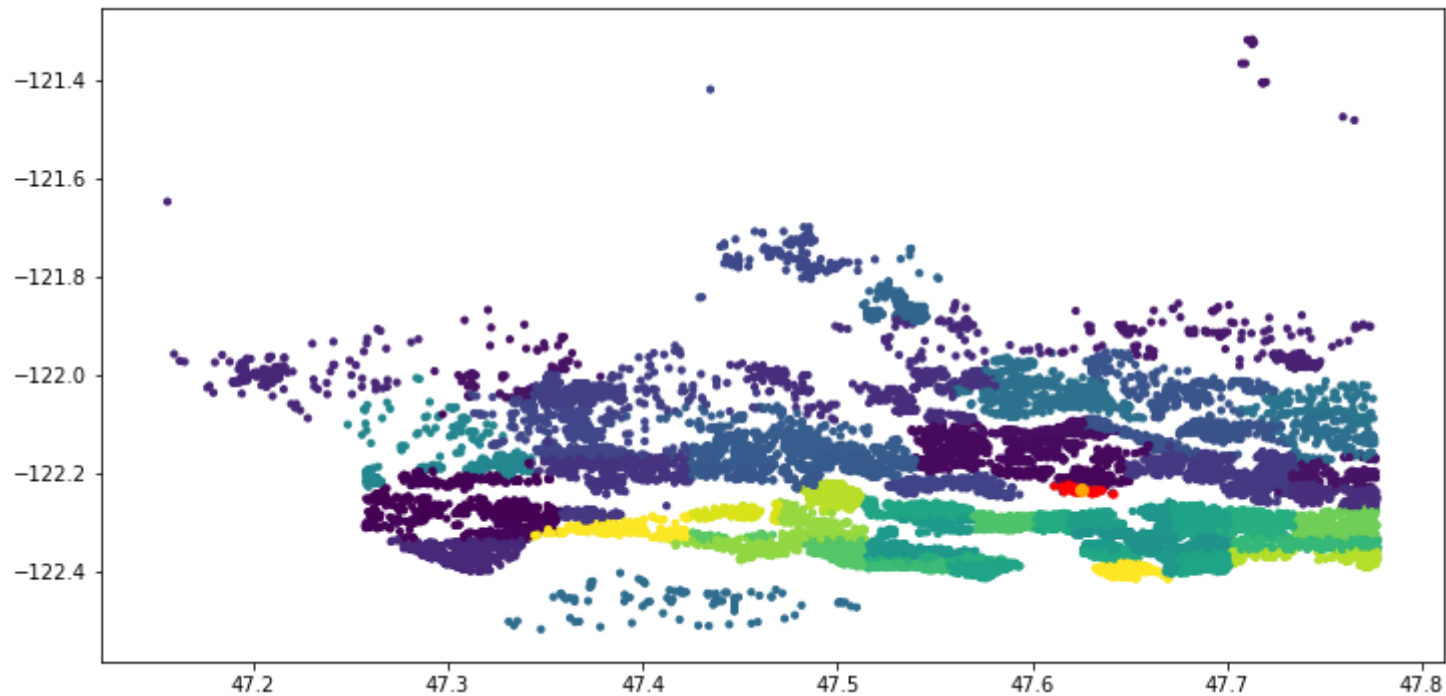
```

In [8]: 1 prices_by_zipcode = dict(data.groupby(['zipcode']).mean()['price'])
2       data['price_per_meter'] = data['price'] / data['sqft_living']
3       prices_per_meter_by_zipcode = dict(data.groupby(['zipcode']).mean()['price_per_meter'])
4       elite_zipcode = [k for k, v in prices_by_zipcode.items() if v == max(prices_by_zipcode.values())[0]]
5       lat_max, long_max = data.groupby(['zipcode']).mean().loc[elite_zipcode, 'lat'], \
6                           data.groupby(['zipcode']).mean().loc[elite_zipcode, 'long']

```

Нарисуем то что получилось. Элитный район - красный, его центр - оранжевая точка. Как видим, найденное место выделялось уже в изначальной визуализации, так что всё верно.

```
In [9]: 1 plt.figure(figsize=(12, 6))
2 plt.scatter(data['lat'], data['long'], c=data['zipcode'], s=10)
3 ▼ plt.scatter(data[data['zipcode'] == elite_zipcode]['lat'],
4             data[data['zipcode'] == elite_zipcode]['long'],
5             c='r', s=10)
6 plt.scatter(lat_max, long_max, c='orange');
```



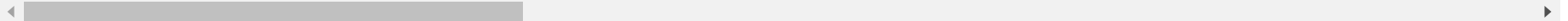
В следующей ячейке преобразуем тестовые данные и уберём из них лишние поля.

```
In [10]: 1 data = transform_data(data,
2           prices_by_zipcode,
3           prices_per_meter_by_zipcode,
4           lat_max, long_max)
5 data = data.drop(['price_per_meter', 'price'], axis=1)
6 data.head()
```

```
Out[10]:
```

|   | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition | grade | sqft_basement | ... | condition_exp | scale_dot | sqft_living |
|---|----------|-----------|-------------|----------|--------|------------|------|-----------|-------|---------------|-----|---------------|-----------|-------------|
| 0 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0          | 0    | 3         | 7     | 0             | ... |               |           | 118000.0    |
| 1 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0          | 0    | 3         | 7     | 400           | ... |               |           | 257000.0    |
| 2 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0          | 0    | 5         | 7     | 910           | ... |               |           | 19600000.0  |
| 3 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0          | 0    | 3         | 8     | 0             | ... |               |           | 168000.0    |
| 4 | 3        | 2.25      | 1715        | 6819     | 2.0    | 0          | 0    | 3         | 7     | 0             | ... |               |           | 171500.0    |

5 rows × 70 columns



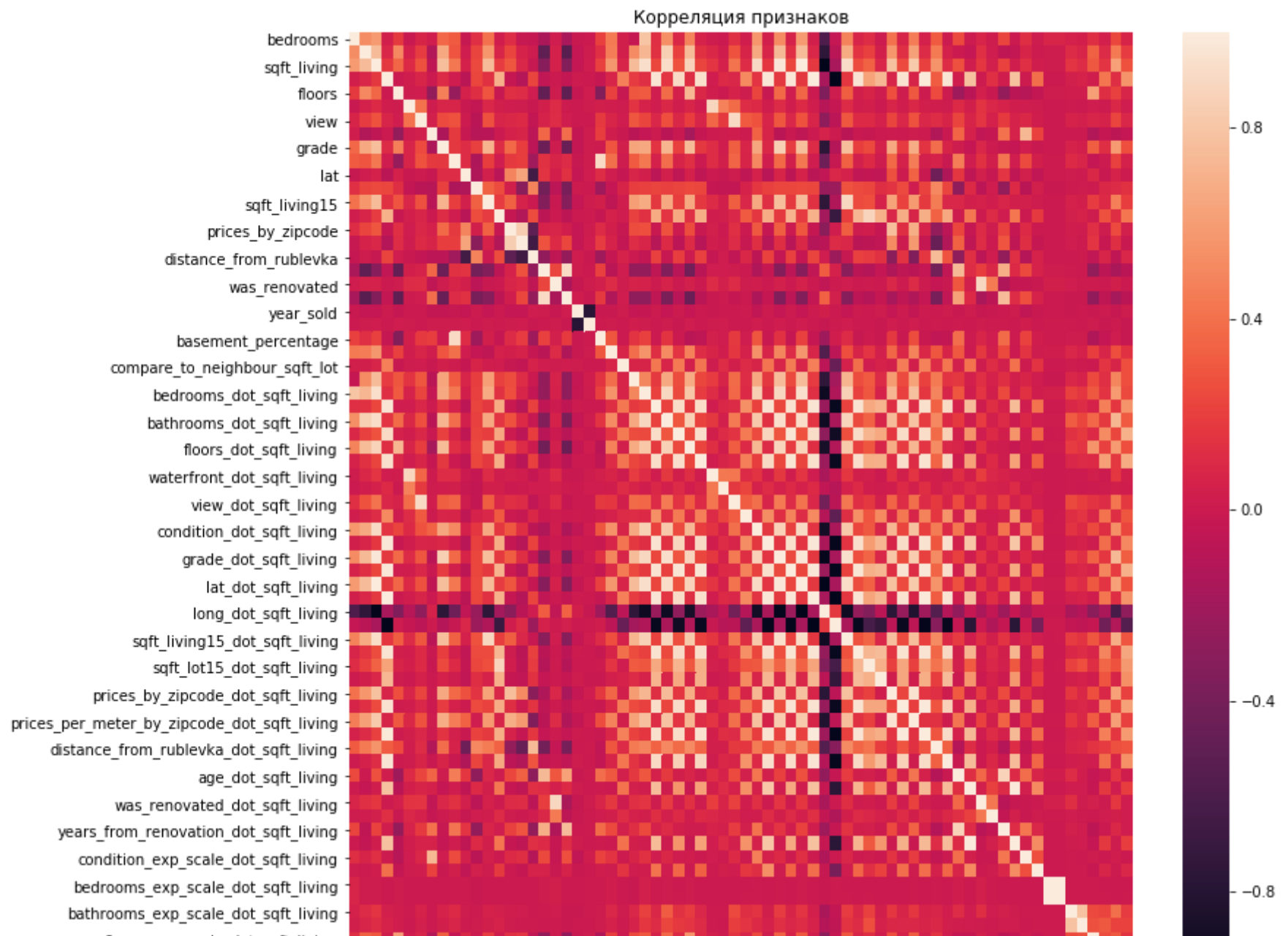
```
In [11]: 1 print(data.shape)

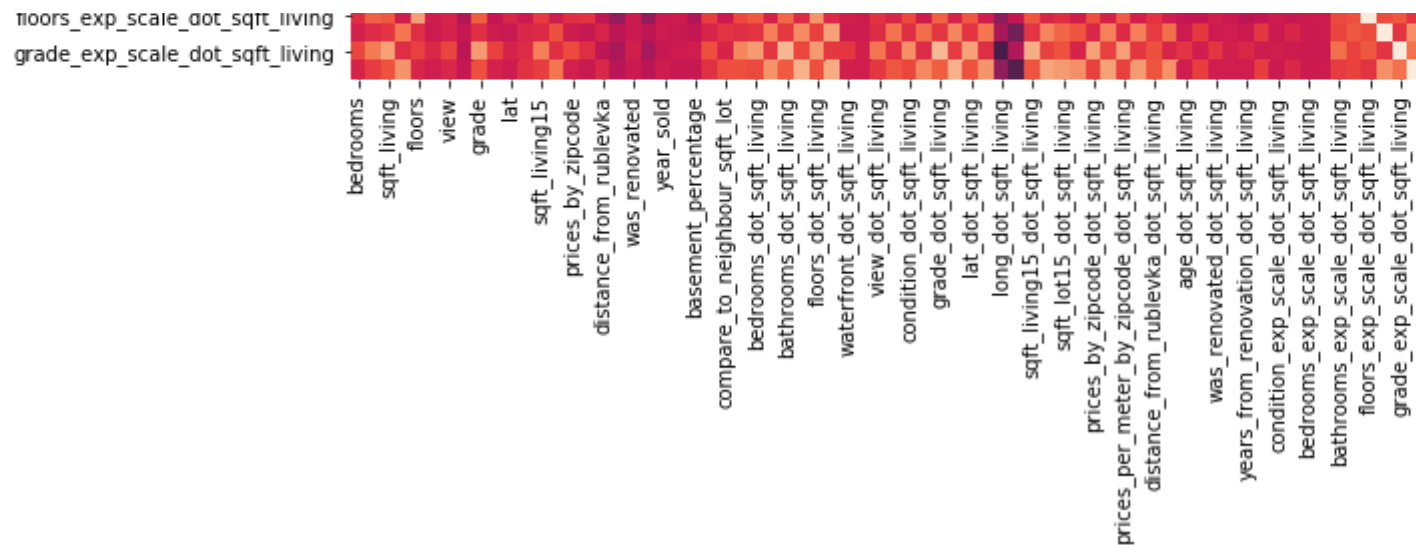
(15626, 70)
```

Далее посмотрим, насколько коррелируют наши признаки (здесь ничего неожиданного: сильная корреляция между теми, которые мы домножали на один и тот же коэффициент)



```
In [12]: 1 plt.figure(figsize=(12,12))
2         sns.heatmap(data.corr())
3         t = plt.title("Корреляция признаков")
```





### 3. Выбор моделей

Для сравнения качества нам пригодится метрика MAPE (хотя оптимизировать её напрямую, конечно, нельзя - для обучения используем просто MSE )

```
In [13]: 1 ▾ def mape_score(y_true, y_predicted):
          2     return np.mean(np.abs(y_true - y_predicted) / y_true)
```

С выбором модели вопросов не возникало, ведь известно, что градиентный бустинг - лучший помощник в соревнованиях на катле. Осталось только подобрать параметры: сделаем это случайным поиском, так как параметров целых 3, и в случае gridsearch -а сетка получится или очень большая, или неточная.

```
In [15]: 1 random_search = RandomizedSearchCV(xgb.XGBRegressor(),
2     param_distributions={
3         'n_estimators': sps.randint(low=2000, high=6000),
4         'learning_rate': [0.01],
5         'max_depth': sps.randint(low=4, high=10),
6         'subsample': sps.uniform(loc=0.5, scale=0.3),
7     },
8     n_iter=10,
9     cv=3,
10    n_jobs=-1,
11    return_train_score=True,
12    scoring=make_scorer(mape_score, greater_is_better=False),
13    verbose=10
14 )
15 random_search.fit(np.array(data), np.array(target))
```

Fitting 3 folds for each of 10 candidates, totalling 30 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  2 tasks      | elapsed:  5.1min
[Parallel(n_jobs=-1)]: Done  9 tasks      | elapsed: 15.5min
[Parallel(n_jobs=-1)]: Done 19 out of  30 | elapsed: 24.8min remaining: 14.3min
[Parallel(n_jobs=-1)]: Done 23 out of  30 | elapsed: 26.2min remaining:  8.0min
[Parallel(n_jobs=-1)]: Done 27 out of  30 | elapsed: 30.2min remaining:  3.4min
[Parallel(n_jobs=-1)]: Done 30 out of  30 | elapsed: 32.9min finished
```

```
Out[15]: RandomizedSearchCV(cv=3, error_score='raise-deprecating',
    estimator=XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
    colsample_bytree=1, gamma=0, importance_type='gain',
    learning_rate=0.1, max_delta_step=0, max_depth=3,
    min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
    nthread=None, objective='reg:linear', random_state=0, reg_alpha=0,
    reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
    subsample=1),
    fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
    param_distributions={'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7fa
    6b460e438>, 'learning_rate': [0.01], 'max_depth': <scipy.stats._distn_infrastructure.rv_frozen object at 0x
    7fa6b460e470>, 'subsample': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7fa6b460e908>},
    pre_dispatch='2*n_jobs', random_state=None, refit=True,
    return_train_score=True,
    scoring=make_scorer(mape_score, greater_is_better=False),
    verbose=10)
```

In [26]:

```
1 best_params = random_search.best_params_  
2 print(best_params)
```

```
{'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 4958, 'subsample': 0.5095764820380737}
```

Посмотрим, на каких наборах параметров получились лучшие результаты. Если бы мы задали какие-то совсем неадекватные границы для случайного поиска, можно было бы таким образом это заметить и улучшить. Но тут всё в порядке:

In [18]:

```
1 result = pd.DataFrame(random_search.cv_results_) [  
2     'param_learning_rate', 'param_max_depth',  
3     'param_n_estimators', 'param_subsample',  
4     'mean_test_score', 'rank_test_score', 'mean_train_score'  
5 ]]  
6 result[result['rank_test_score'] < 10]
```

Out[18]:

|   | param_learning_rate | param_max_depth | param_n_estimators | param_subsample | mean_test_score | rank_test_score | mean_train_score |
|---|---------------------|-----------------|--------------------|-----------------|-----------------|-----------------|------------------|
| 0 | 0.01                | 6               | 2522               | 0.72314         | -0.119729       | 5               | -0.071424        |
| 1 | 0.01                | 5               | 5250               | 0.633437        | -0.119222       | 4               | -0.063938        |
| 2 | 0.01                | 6               | 4958               | 0.509576        | -0.118952       | 1               | -0.049089        |
| 3 | 0.01                | 5               | 5795               | 0.665886        | -0.118986       | 2               | -0.059500        |
| 4 | 0.01                | 4               | 5805               | 0.56346         | -0.120611       | 9               | -0.083192        |
| 5 | 0.01                | 7               | 3943               | 0.589378        | -0.119190       | 3               | -0.037588        |
| 6 | 0.01                | 9               | 2444               | 0.639167        | -0.120399       | 7               | -0.028256        |
| 8 | 0.01                | 9               | 3101               | 0.60583         | -0.120454       | 8               | -0.020214        |
| 9 | 0.01                | 6               | 2554               | 0.595773        | -0.120140       | 6               | -0.074183        |

Теперь обучим с `XGBRegressor` с лучшими параметрами. Можно было бы взять и `random_search.best_estimator_`, но он обучен не на полных данных, а с исключением подмножества для кросс-валидации. А мы боремся за каждую тысячную балла на катге!

```
In [21]: 1 ▾ grad_boost = xgb.XGBRegressor(  
2         n_estimators=best_params['n_estimators'],  
3         learning_rate=best_params['learning_rate'],  
4         max_depth=best_params['max_depth'],  
5         subsample=best_params['subsample'],  
6         n_jobs=-1  
7     )  
8     grad_boost.fit(np.array(data), np.array(target))
```

```
Out[21]: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,  
    colsample_bytree=1, gamma=0, importance_type='gain',  
    learning_rate=0.01, max_delta_step=0, max_depth=6,  
    min_child_weight=1, missing=None, n_estimators=4958, n_jobs=-1,  
    nthread=None, objective='reg:linear', random_state=0, reg_alpha=0,  
    reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,  
    subsample=0.5095764820380737)
```

Ещё можно взглянуть на значимость признаков. Если какие-то из них мы добавляли вручную, то можно увидеть, что некоторые из них, вопреки предположениям, не особо помогают (но это не значит, что их обязательно удалять)

```
In [23]: 1 sorted(list(zip(grad_boost.feature_importances_, data.columns)))
```

```
Out[23]: [(0.0006207382, 'bedrooms'),
(0.0010073672, 'condition'),
(0.0011593606, 'floors'),
(0.0013671934, 'bathrooms'),
(0.0014149902, 'lat_dot_sqft_lot'),
(0.001556657, 'long_dot_sqft_lot'),
(0.0015827516, 'month_sold'),
(0.0017012606, 'bedrooms_dot_sqft_lot'),
(0.0017145465, 'sqft_lot15'),
(0.0017322748, 'sqft_lot15_dot_sqft_living'),
(0.0017358004, 'prices_by_zipcode_dot_sqft_lot'),
(0.0017609553, 'compare_to_neighbour_sqft_lot'),
(0.0017696175, 'sqft_lot15_dot_sqft_lot'),
(0.001784059, 'sqft_lot'),
(0.0018463883, 'prices_per_meter_by_zipcode_dot_sqft_lot'),
(0.0018692205, 'condition_dot_sqft_lot'),
(0.0019310656, 'grade_dot_sqft_lot'),
(0.0019577672, 'bedrooms_exp_scale_dot_sqft_lot'),
(0.0019748379, 'mean_room_sqft'),
(0.002077991, 'sqft_living15_dot_sqft_lot'),
(0.002094447, 'age_dot_sqft_lot'),
(0.0021026565, 'bathrooms_dot_sqft_lot'),
(0.0021344007, 'long_dot_sqft_living'),
(0.0024474922, 'floors_exp_scale_dot_sqft_lot'),
(0.0024577565, 'was_renovated'),
(0.0024608881, 'distance_from_rublevka_dot_sqft_lot'),
(0.00247638, 'lat_dot_sqft_living'),
(0.0026120418, 'condition_exp_scale_dot_sqft_living'),
(0.0026890214, 'sqft_living15'),
(0.0027092162, 'bathrooms_exp_scale_dot_sqft_lot'),
(0.0027512785, 'floors_dot_sqft_lot'),
(0.0028556604, 'floors_exp_scale_dot_sqft_living'),
(0.0028817952, 'age_dot_sqft_living'),
(0.0029450995, 'distance_from_rublevka_dot_sqft_living'),
(0.0030020857, 'years_from_renovation_dot_sqft_living'),
(0.0030165392, 'condition_exp_scale_dot_sqft_lot'),
(0.003020444, 'years_from_renovation_dot_sqft_lot'),
(0.0030780362, 'bedrooms_exp_scale_dot_sqft_living'),
(0.00329922, 'bathrooms_exp_scale_dot_sqft_living'),
```

```
(0.0036233806, 'sqft_basement'),
(0.0036307129, 'long'),
(0.0036395718, 'lat'),
(0.0037270274, 'basement_percentage'),
(0.0039031804, 'sqft_living'),
(0.004343715, 'age'),
(0.004412267, 'floors_dot_sqft_living'),
(0.0045310142, 'years_from_renovation'),
(0.0048723486, 'compare_to_neighbour_sqft_living'),
(0.0049182237, 'distance_from_rublevka'),
(0.0050949035, 'year_sold'),
(0.005937423, 'sqft_living15_dot_sqft_living'),
(0.0060910815, 'prices_by_zipcode'),
(0.0061045852, 'prices_per_meter_by_zipcode'),
(0.00614252, 'was_renovated_dot_sqft_lot'),
(0.0063414993, 'grade_exp_scale_dot_sqft_lot'),
(0.0064209877, 'condition_dot_sqft_living'),
(0.00672584, 'bedrooms_dot_sqft_living'),
(0.0078235185, 'bathrooms_dot_sqft_living'),
(0.00990392, 'was_renovated_dot_sqft_living'),
(0.010139683, 'view_dot_sqft_lot'),
(0.016207336, 'grade_dot_sqft_living'),
(0.01841262, 'grade_exp_scale_dot_sqft_living'),
(0.019183058, 'prices_by_zipcode_dot_sqft_living'),
(0.023757199, 'view_dot_sqft_living'),
(0.025073098, 'view'),
(0.029036393, 'grade'),
(0.075739756, 'waterfront_dot_sqft_lot'),
(0.09406733, 'waterfront_dot_sqft_living'),
(0.11733189, 'waterfront'),
(0.37926468, 'prices_per_meter_by_zipcode_dot_sqft_living')]
```

Наконец, сделаем предсказания. Это может показаться очевидным, но тестовые данные мы преобразуем точно так же, как и `data` :)

```
In [24]: 1 test_data = pd.read_csv('houses_test.csv')
2 test_data = transform_data(test_data,
3                             prices_by_zipcode,
4                             prices_per_meter_by_zipcode,
5                             lat_max, long_max)
6 assert sum(test_data.columns == data.columns) == len(data.columns)
```

```
In [25]: 1 predicted = grad_boost.predict(np.array(test_data))
2 test_predictions = pd.DataFrame(predicted)
3 test_predictions.insert(0, 'index', 1+np.arange(len(test_predictions)))
4 test_predictions.columns = ['index', 'price']
5 test_predictions.to_csv(f'answer_{datetime.now():%Y%m%d-%H%M-%S}.csv', index=False)
6 test_predictions.head()
```

```
Out[25]:
```

|   | index | price        |
|---|-------|--------------|
| 0 | 1     | 291629.18750 |
| 1 | 2     | 906873.81250 |
| 2 | 3     | 187603.46875 |
| 3 | 4     | 443859.68750 |
| 4 | 5     | 283426.06250 |