In [133]:

```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score as r2
```

In [134]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [135]:

```
def evaluate_preds(true_values, pred_values):
    print("R2:\t" + str(round(r2(true_values, pred_values), 3)))

    plt.figure(figsize=(10,10))

    sns.scatterplot(x=pred_values, y=true_values)

    plt.xlabel('Predicted values')
    plt.ylabel('True values')
    plt.title('True vs Predicted values')
    plt.show()
```

In [136]:

```
TRAIN_DATASET_PATH = 'train.csv'
TEST_DATASET_PATH = 'test.csv'
PREDICTIONS_PATH = 'predictions.csv '
```

Загрузка данных

In [137]:

```
train_df = pd.read_csv(TRAIN_DATASET_PATH)
test_df = pd.read_csv(TEST_DATASET_PATH)
```

Обзор обучающего датасета¶

In [138]:

```
print(train_df.shape)
```

```
(10000, 20)
```

In [139]:

```
print(train_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 20 columns):
Id               10000 non-null int64
DistrictId       10000 non-null int64
Rooms            10000 non-null float64
Square           10000 non-null float64
LifeSquare       7887 non-null float64
KitchenSquare    10000 non-null float64
Floor            10000 non-null int64
HouseFloor       10000 non-null float64
HouseYear        10000 non-null int64
Ecology_1        10000 non-null float64
Ecology_2        10000 non-null object
Ecology_3        10000 non-null object
Social_1         10000 non-null int64
Social_2         10000 non-null int64
Social_3         10000 non-null int64
Healthcare_1     5202 non-null float64
Helthcare_2      10000 non-null int64
Shops_1          10000 non-null int64
Shops_2          10000 non-null object
Price            10000 non-null float64
dtypes: float64(8), int64(9), object(3)
memory usage: 1.5+ MB
None
```

In [140]:

```
train_df.describe()
```

Out[140]:

	Id	DistrictId	Rooms	Square	LifeSquare	KitchenSquare	
count	10000.00000	10000.000000	10000.000000	10000.000000	7887.000000	10000.000000	10C
mean	8383.40770	50.400800	1.890500	56.315775	37.199645	6.273300	
std	4859.01902	43.587592	0.839512	21.058732	86.241209	28.560917	
min	0.00000	0.000000	0.000000	1.136859	0.370619	0.000000	
25%	4169.50000	20.000000	1.000000	41.774881	22.769832	1.000000	
50%	8394.50000	36.000000	2.000000	52.513310	32.781260	6.000000	
75%	12592.50000	75.000000	2.000000	65.900625	45.128803	9.000000	
max	16798.00000	209.000000	19.000000	641.065193	7480.592129	2014.000000	

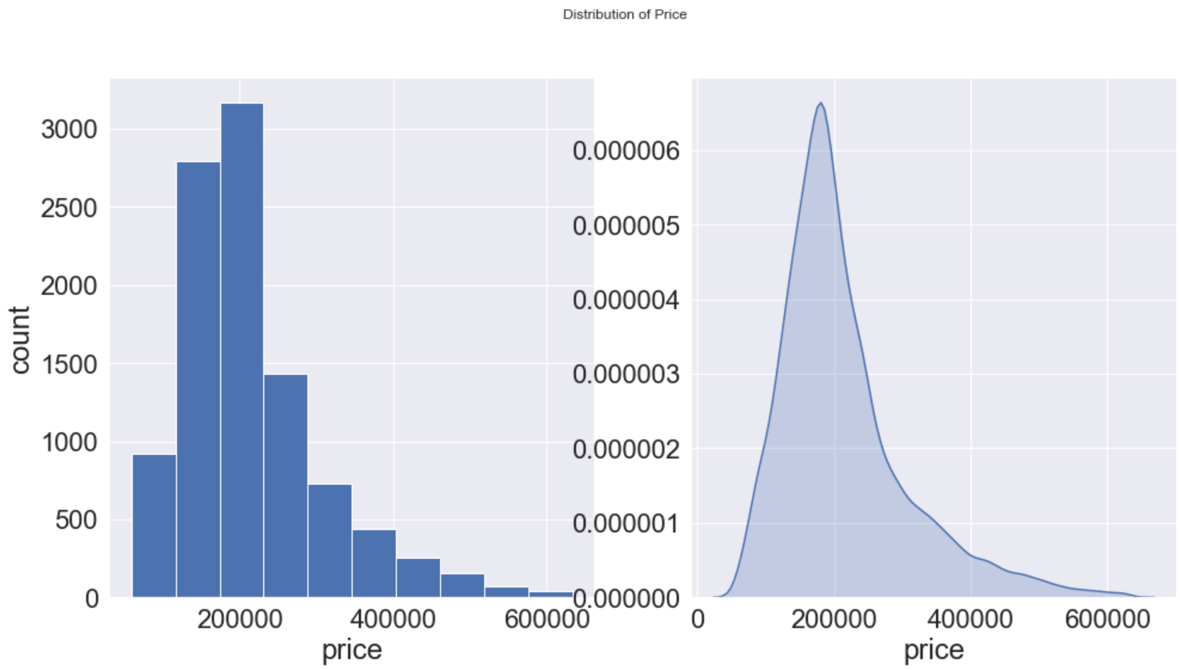
In [141]:

```
plt.figure(figsize = (16, 8))

plt.subplot(121)
train_df['Price'].hist()
plt.ylabel('count')
plt.xlabel('price')

plt.subplot(122)
sns.kdeplot(train_df['Price'], shade=True, legend=False)
plt.xlabel('price')

plt.suptitle('Distribution of Price')
plt.show()
```



Подготовка обучающего датасета

Исключаем признак "Healthcare_1", т.к. по нему почти 50% пропусков

In [142]:

```
train_df = train_df.drop('Healthcare_1', axis=1)
```

Преобразуем категориальные признаки "Ecology_2", "Ecology_3", "Shops_2" в бинарные

In [143]:

```
print(train_df['Ecology_2'].value_counts())
print(train_df['Ecology_3'].value_counts())
print(train_df['Shops_2'].value_counts())
```

```
B    9903
A      97
Name: Ecology_2, dtype: int64
B    9725
A     275
Name: Ecology_3, dtype: int64
B    9175
A     825
Name: Shops_2, dtype: int64
```

In [144]:

```
train_df['Ecology_2_bin'] = train_df['Ecology_2'].replace({'A':0, 'B':1})
train_df['Ecology_3_bin'] = train_df['Ecology_3'].replace({'A':0, 'B':1})
train_df['Shops_2_bin'] = train_df['Shops_2'].replace({'A':0, 'B':1})
```

Работаем с выбросами признака "Rooms"

In [145]:

```
rooms_med = train_df['Rooms'].median()
train_df.loc[train_df['Rooms'].isin([0, 10, 19]), 'Rooms'] = rooms_med
```

Работаем с выбросами признаков "LifeSquare" и "KitchenSquare"

In [146]:

```
lifesq_med = train_df['LifeSquare'].median()
kitchsq_med = train_df['KitchenSquare'].median()
train_df.loc[train_df['LifeSquare'].isnull(), 'LifeSquare'] = lifesq_med
train_df.loc[train_df['LifeSquare'] < 10, 'LifeSquare'] = lifesq_med
train_df.loc[train_df['LifeSquare'] > 400, 'LifeSquare'] = lifesq_med
train_df.loc[train_df['KitchenSquare'] < 5, 'KitchenSquare'] = kitchsq_med
train_df.loc[train_df['KitchenSquare'] > 80, 'KitchenSquare'] = kitchsq_med
```

Работаем с выбросами признака "Square"

In [147]:

```
square_med = train_df['Square'].median()
train_df.loc[train_df['Square'] < 16, 'Square'] = square_med
train_df.loc[train_df['Square'] > 400, 'Square'] = square_med
```

Работаем с выбросами признака "HouseYear"

In [148]:

```
train_df.loc[train_df['HouseYear'] > 2020, 'HouseYear'] = 2020
```

Работаем с выбросами признаков "Floor" и "HouseFloor"

In [149]:

```
hfloor_med = train_df['HouseFloor'].median()
train_df.loc[train_df['HouseFloor'] == 0, 'HouseFloor'] = hfloor_med
```

In [150]:

```
ind = train_df[train_df['Floor'] > train_df['HouseFloor']].index
train_df.loc[ind, 'Floor'] = train_df.loc[ind, 'HouseFloor']
```

Вычисляем "m_2_Price" - стоимость квадратного метра общей площади

In [151]:

```
train_df['m_2_Price'] = train_df['Price'] / train_df['Square']
```

На его основе создаем новые признаки m_2_MedPriceByDistrict - медианная стоимость квадратного метра в зависимости от района и m_2_MedPriceByHouseYear - медианная стоимость квадратного метра в зависимости от возраста дома

In [152]:

```
m_2_MedPriceByDistrict = train_df.groupby(['DistrictId'], as_index=False).agg({'m_2_Price':
                    .rename(columns={'m_2_Price':'m_2_MedPriceByDistrict'})
m_2_MedPriceByHouseYear = train_df.groupby(['HouseYear'], as_index=False).agg({'m_2_Price':
                    .rename(columns={'m_2_Price':'m_2_MedPriceByHouseYear'})
```

Добавляем новые признаки к датасету

In [153]:

```
train_df = train_df.merge(m_2_MedPriceByDistrict, on=['DistrictId'], how='left')
train_df = train_df.merge(m_2_MedPriceByHouseYear, on=['HouseYear'], how='left')
train_df.head()
```

Out[153]:

	Id	DistrictId	Rooms	Square	LifeSquare	KitchenSquare	Floor	HouseFloor	HouseY
0	14038	35	2.0	47.981561	29.442751	6.0	7.0	9.0	19
1	15053	41	3.0	65.683640	40.049543	8.0	7.0	9.0	19
2	4765	53	2.0	44.947953	29.197612	6.0	8.0	12.0	19
3	5809	58	2.0	53.352981	52.731512	9.0	8.0	17.0	19
4	10783	99	1.0	39.649192	23.776169	7.0	11.0	12.0	19

5 rows × 25 columns

In [154]:

```
train_df.describe()
```

Out[154]:

	Id	DistrictId	Rooms	Square	LifeSquare	KitchenSquare	
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10
mean	8383.40770	50.400800	1.888800	56.228457	35.980607	7.490600	
std	4859.01902	43.587592	0.812096	19.058793	15.378518	3.290409	
min	0.00000	0.000000	1.000000	16.117154	10.523868	5.000000	
25%	4169.50000	20.000000	1.000000	41.800063	27.654813	6.000000	
50%	8394.50000	36.000000	2.000000	52.513310	32.781260	6.000000	
75%	12592.50000	75.000000	2.000000	65.889736	41.415441	9.000000	
max	16798.00000	209.000000	6.000000	275.645284	263.542020	78.000000	

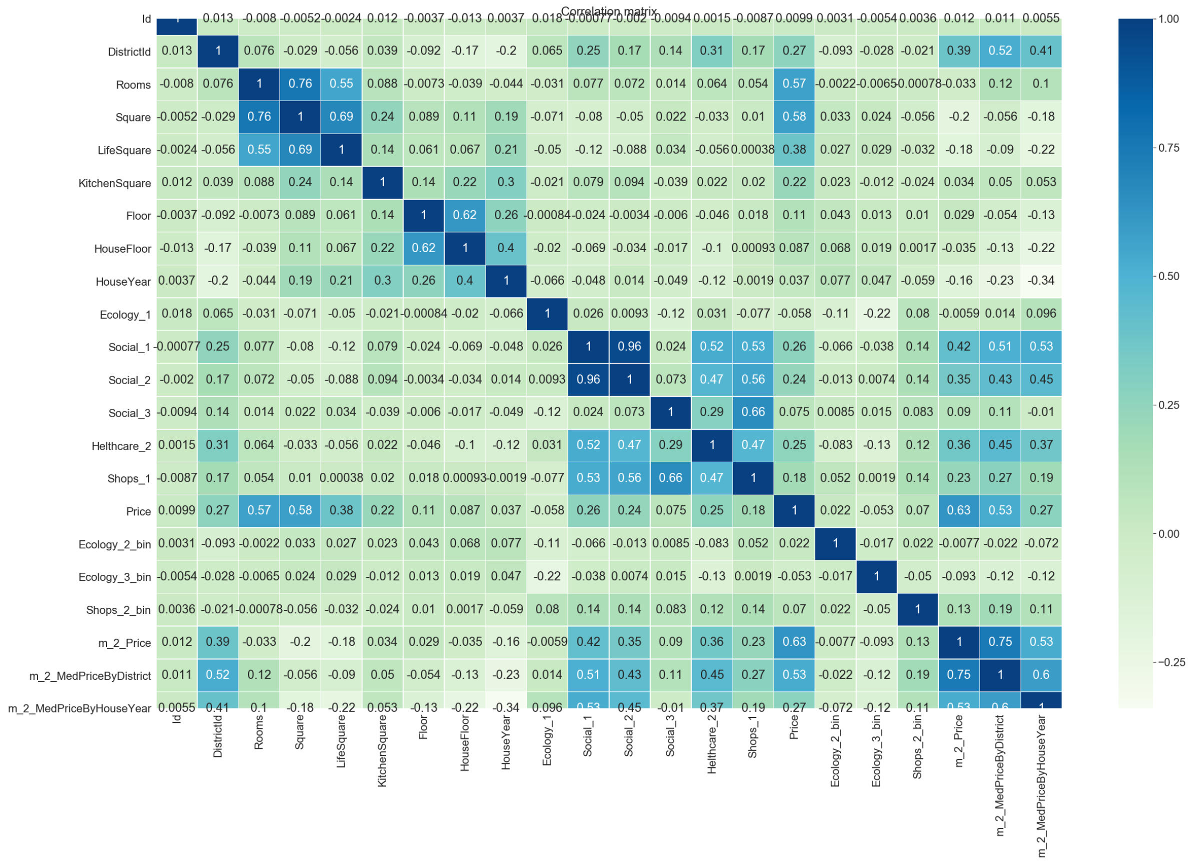
8 rows × 22 columns

In [155]:

```
plt.figure(figsize = (40,25))

sns.set(font_scale=2)
sns.heatmap(train_df.corr(), annot=True, linewidths=.5, cmap='GnBu')

plt.title('Correlation matrix')
plt.show()
```



Отбираем признаки для модели

In [156]:

```
feature_names = ['Rooms', 'Square', 'LifeSquare', 'KitchenSquare', 'Floor', 'HouseFloor', '
                 'm_2_MedPriceByDistrict', 'm_2_MedPriceByHouseYear',
                 'Ecology_1','Social_1', 'Social_3', 'Helthcare_2', 'Shops_1',
                 'Ecology_2_bin', 'Ecology_3_bin', 'Shops_2_bin',]
target_name = 'Price'
```

In [157]:

```
df = train_df[feature_names + [target_name]]
df.head()
```

Out[157]:

	Rooms	Square	LifeSquare	KitchenSquare	Floor	HouseFloor	HouseYear	m_2_MedPrice
0	2.0	47.981561	29.442751	6.0	7.0	9.0	1969	43
1	3.0	65.683640	40.049543	8.0	7.0	9.0	1978	44
2	2.0	44.947953	29.197612	6.0	8.0	12.0	1968	48
3	2.0	53.352981	52.731512	9.0	8.0	17.0	1977	29
4	1.0	39.649192	23.776169	7.0	11.0	12.0	1976	39

Масштабируем признаки

In [158]:

```
scaler = StandardScaler()
stand_features = scaler.fit_transform(df[feature_names])
```

In [159]:

```
df[feature_names] = pd.DataFrame(stand_features, columns=feature_names)
df.head()
```

Out[159]:

	Rooms	Square	LifeSquare	KitchenSquare	Floor	HouseFloor	HouseYear	m_2_Me
0	0.136936	-0.432730	-0.425150	-0.453036	-0.177049	-0.614468	-0.861908	
1	1.368379	0.496131	0.264599	0.154821	-0.177049	-0.614468	-0.373187	
2	0.136936	-0.591909	-0.441092	-0.453036	0.015773	-0.148856	-0.916210	
3	0.136936	-0.150882	1.089295	0.458750	0.015773	0.627163	-0.427489	
4	-1.094506	-0.869945	-0.793643	-0.149107	0.594239	-0.148856	-0.481792	

Разбиваем на обучающую и валидационную выборку

In [160]:

```
X = df[feature_names]
y = df[target_name]
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.67, shuffle=True,
```

Моделируем

In [161]:

```
gb_model = GradientBoostingRegressor(max_depth=5, n_estimators=200, random_state=42)
gb_model.fit(X_train, y_train)
```

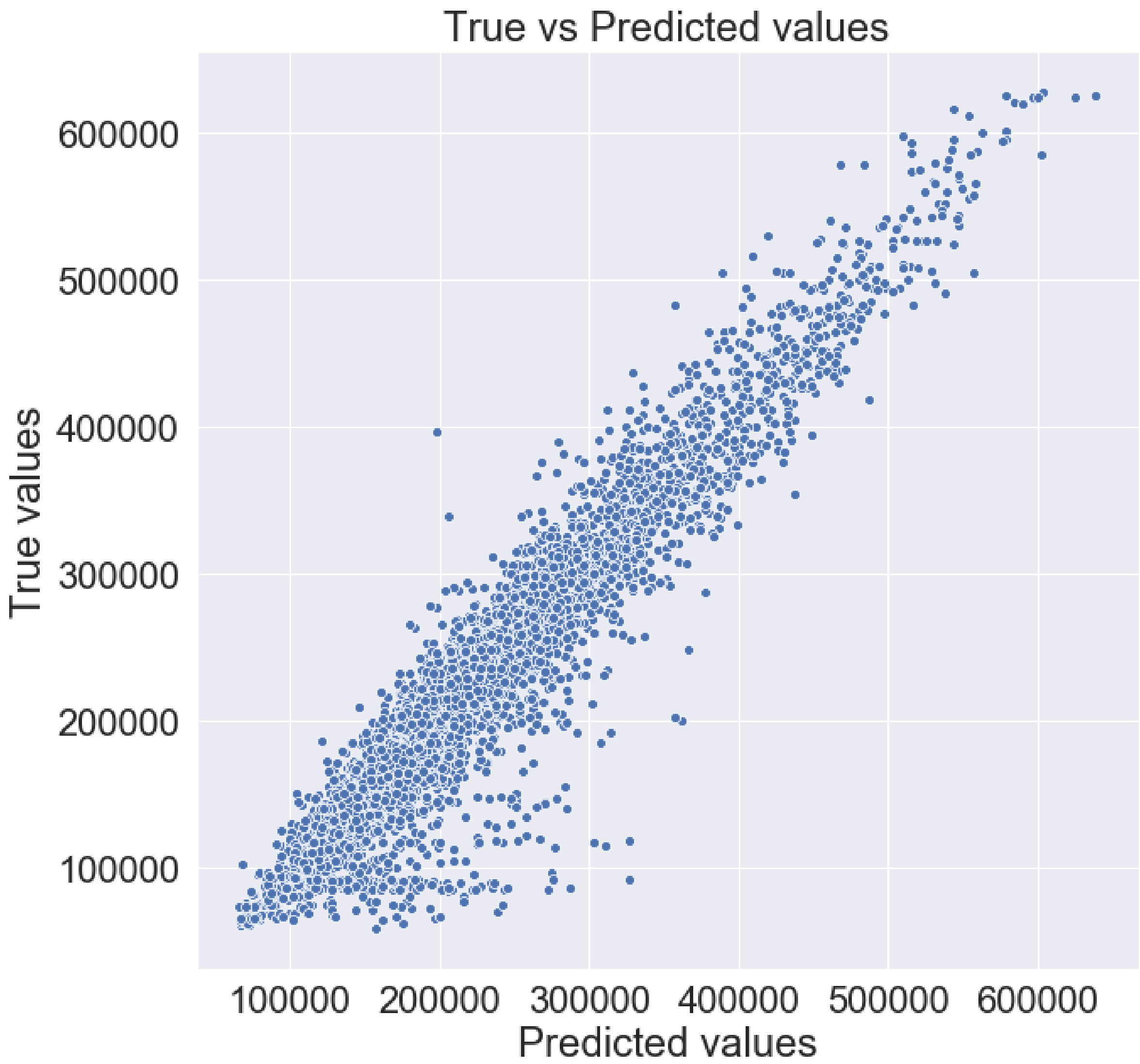
Out[161]:

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.1, loss='ls', max_depth=5,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=Non
e,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=200,
                          n_iter_no_change=None, presort='auto',
                          random_state=42, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=Fal
se)
```

In [162]:

```
y_train_preds = gb_model.predict(X_train)
evaluate_preds(y_train, y_train_preds)
```

```
R2:     0.91
```

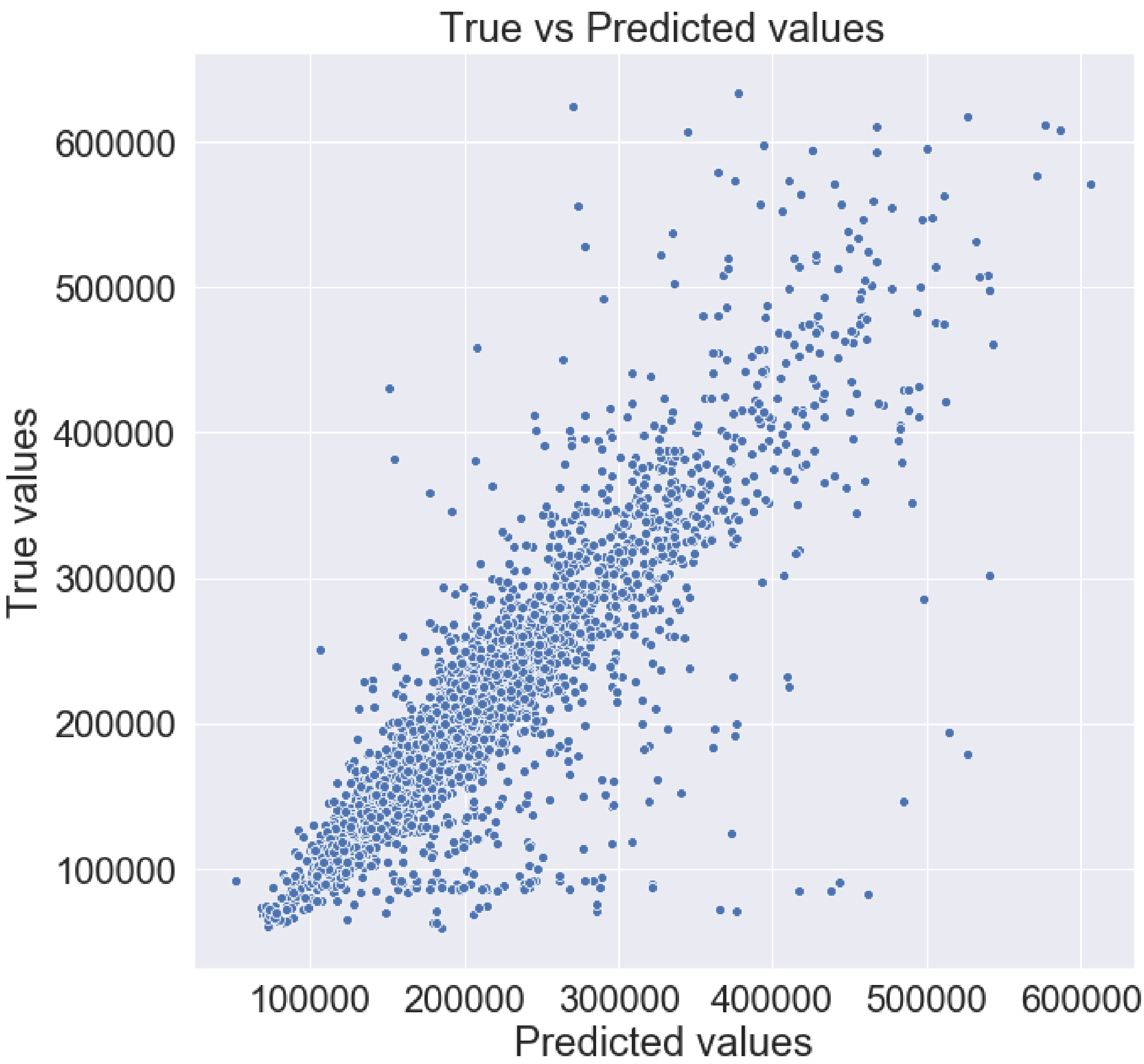


Проверка на валидационной выборке

In [163]:

```
y_valid_preds = gb_model.predict(X_valid)
evaluate_preds(y_valid, y_valid_preds)
```

```
R2:     0.743
```



Важность признаков

In [164]:

```
feature_importances = pd.DataFrame(zip(X_train.columns, gb_model.feature_importances_),
                                   columns=['feature_name', 'importance'])

feature_importances.sort_values(by='importance', ascending=False)
```

Out[164]:

	feature_name	importance
1	Square	0.451549
7	m_2_MedPriceByDistrict	0.366754
8	m_2_MedPriceByHouseYear	0.029613
0	Rooms	0.024442
10	Social_1	0.019055
2	LifeSquare	0.018196
6	HouseYear	0.015947
9	Ecology_1	0.015769
5	HouseFloor	0.014159
3	KitchenSquare	0.013155
4	Floor	0.011156
11	Social_3	0.009935
13	Shops_1	0.005568
12	Helthcare_2	0.003250
16	Shops_2_bin	0.000604
15	Ecology_3_bin	0.000502
14	Ecology_2_bin	0.000346

Обзор тестового датасета

In [165]:

```
print(test_df.shape)
```

```
(5000, 19)
```

In [166]:

```
print(test_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 19 columns):
Id                5000 non-null int64
DistrictId        5000 non-null int64
Rooms             5000 non-null float64
Square            5000 non-null float64
LifeSquare        3959 non-null float64
KitchenSquare     5000 non-null float64
Floor             5000 non-null int64
HouseFloor        5000 non-null float64
HouseYear         5000 non-null int64
Ecology_1         5000 non-null float64
Ecology_2         5000 non-null object
Ecology_3         5000 non-null object
Social_1          5000 non-null int64
Social_2          5000 non-null int64
Social_3          5000 non-null int64
Healthcare_1      2623 non-null float64
Helthcare_2       5000 non-null int64
Shops_1           5000 non-null int64
Shops_2           5000 non-null object
dtypes: float64(7), int64(9), object(3)
memory usage: 742.3+ KB
None
```

In [167]:

```
test_df.describe()
```

Out[167]:

	Id	DistrictId	Rooms	Square	LifeSquare	KitchenSquare	
count	5000.000000	5000.000000	5000.000000	5000.000000	3959.000000	5000.000000	5000.
mean	8412.595400	51.279200	1.910000	56.449500	36.158810	5.976800	8.
std	4832.674037	44.179466	0.838594	19.092787	17.825287	9.950018	5.
min	1.000000	0.000000	0.000000	1.378543	0.333490	0.000000	1.
25%	4221.750000	21.000000	1.000000	41.906231	23.092026	1.000000	4.
50%	8320.500000	37.000000	2.000000	52.921340	32.925087	6.000000	7.
75%	12598.250000	77.000000	2.000000	66.285129	45.174091	9.000000	12.
max	16795.000000	212.000000	17.000000	223.453689	303.071094	620.000000	78.

Подготовка тестового датасета

Исключаем признак "Healthcare_1", т.к. по нему почти 50% пропусков

In [168]:

```
test_df = test_df.drop('Healthcare_1', axis=1)
```

Преобразуем категориальные признаки "Ecology_2", "Ecology_3", "Shops_2"

In [169]:

```
print(test_df['Ecology_2'].value_counts())
print(test_df['Ecology_3'].value_counts())
print(test_df['Shops_2'].value_counts())
```

```
B    4952
A      48
Name: Ecology_2, dtype: int64
B    4851
A     149
Name: Ecology_3, dtype: int64
B    4588
A     412
Name: Shops_2, dtype: int64
```

In [170]:

```
test_df['Ecology_2_bin'] = test_df['Ecology_2'].replace({'A':0, 'B':1})
test_df['Ecology_3_bin'] = test_df['Ecology_3'].replace({'A':0, 'B':1})
test_df['Shops_2_bin'] = test_df['Shops_2'].replace({'A':0, 'B':1})
```

Работаем с выбросами признака "Rooms"

In [171]:

```
test_df.loc[test_df['Rooms'].isin([0, 17]), 'Rooms'] = rooms_med
```

Работаем с выбросами признаков "LifeSquare" и "KitchenSquare"

In [172]:

```
test_df.loc[test_df['LifeSquare'].isnull(), 'LifeSquare'] = lifesq_med
test_df.loc[test_df['LifeSquare'] < 10, 'LifeSquare'] = lifesq_med
test_df.loc[test_df['LifeSquare'] > 200, 'LifeSquare'] = lifesq_med
test_df.loc[test_df['KitchenSquare'] < 5, 'KitchenSquare'] = kitchsq_med
test_df.loc[test_df['KitchenSquare'] > 80, 'KitchenSquare'] = kitchsq_med
```

Работаем с выбросами признака "Square"

In [173]:

```
test_df.loc[test_df['Square'] < 16, 'Square'] = square_med
test_df.loc[test_df['Square'] > 400, 'Square'] = square_med
```

Работаем с выбросами признаков "Floor" и "HouseFloor"

In [174]:

```
test_df.loc[test_df['HouseFloor'] == 0, 'HouseFloor'] = hfloor_med
```

In [175]:

```
ind = test_df[test_df['Floor'] > test_df['HouseFloor']].index
test_df.loc[ind, 'Floor'] = test_df.loc[ind, 'HouseFloor']
```

Добавляем новые признаки к датасету

In [176]:

```
test_df = test_df.merge(m_2_MedPriceByDistrict, on=['DistrictId'], how='left')
test_df = test_df.merge(m_2_MedPriceByHouseYear, on=['HouseYear'], how='left')
```

Заполняем возможные пропуски

In [177]:

```
test_df.loc[test_df['m_2_MedPriceByDistrict'].isnull(), 'm_2_MedPriceByDistrict'] =\
                    m_2_MedPriceByDistrict['m_2_MedPriceByDistrict'].median()
```

In [178]:

```
test_df.loc[(test_df['m_2_MedPriceByHouseYear'].isnull()) & (test_df['HouseYear'] < 1950),
                     'm_2_MedPriceByHouseYear'] =\
                      m_2_MedPriceByHouseYear.loc[m_2_MedPriceByHouseYear['HouseYear'] < 19
                                                   'm_2_MedPriceByHouseYear'].median()
```

In [179]:

```
test_df.loc[(test_df['m_2_MedPriceByHouseYear'].isnull()) & (test_df['HouseYear'] >= 1990),
                     'm_2_MedPriceByHouseYear'] =\
                      m_2_MedPriceByHouseYear.loc[m_2_MedPriceByHouseYear['HouseYear'] >= 1
                                                   'm_2_MedPriceByHouseYear'].median()
```

In [180]:

```
test_df.loc[(test_df['m_2_MedPriceByHouseYear'].isnull()) & (test_df['HouseYear'] >= 1950)
                    (test_df['HouseYear'] < 1990), 'm_2_MedPriceByHouseYear'] =\
                      m_2_MedPriceByHouseYear.loc[(m_2_MedPriceByHouseYear['HouseYear'] >=
                                                   (m_2_MedPriceByHouseYear['HouseYear'] < 1
                                                   'm_2_MedPriceByHouseYear'].median()
```

In [181]:

```
test_df.describe()
```

Out[181]:

	Id	DistrictId	Rooms	Square	LifeSquare	KitchenSquare	
count	5000.000000	5000.000000	5000.00000	5000.000000	5000.000000	5000.000000	5000.0
mean	8412.595400	51.279200	1.90780	56.543749	36.047463	7.425800	8.0
std	4832.674037	44.179466	0.81008	18.955344	14.706864	3.038674	5.3
min	1.000000	0.000000	1.00000	16.319015	10.692499	5.000000	1.0
25%	4221.750000	21.000000	1.00000	41.951045	27.990919	6.000000	4.0
50%	8320.500000	37.000000	2.00000	52.921340	32.781260	6.000000	7.0
75%	12598.250000	77.000000	2.00000	66.285129	41.760597	9.000000	12.0
max	16795.000000	212.000000	6.00000	223.453689	169.901701	65.000000	46.0

Масштабируем признаки

In [182]:

```
stand_features = scaler.fit_transform(test_df[feature_names])
```

In [183]:

```
test_df[feature_names] = pd.DataFrame(stand_features, columns=feature_names)
```

Предсказываем цены для тестового датасета¶

In [184]:

```
X_test = test_df[feature_names]
```

In [185]:

```
y_test_preds = gb_model.predict(X_test)
```

Сохраняем результаты

In [186]:

```
test_df['Price'] = y_test_preds
```

In [187]:

```
test_df.to_csv(PREDICTIONS_PATH, columns=['Id', 'Price'], index=False, encoding='utf-8')
```

In [188]:

```
print(y_test_preds)
```

```
[164458.43017737 216300.55543514 272844.1635563  ... 287805.07697688
 201779.6857984  176929.98550873]
```

In [189]:

```
print(X_test)
```

```
         Rooms    Square  LifeSquare  KitchenSquare     Floor  HouseFloor  \
0     0.113827 -0.351446   -0.177804      -0.469265 -0.384893    0.163681
1     0.113827  0.671088   -0.222109      -0.469265 -1.321830   -1.846763
2    -1.120741 -0.212649   -1.366792       1.505478 -1.134442   -1.228165
3     0.113827  0.870705    1.080786       0.518107  2.613303    1.400877
4    -1.120741 -0.475725    0.499144      -0.469265  1.676366    0.627630
...        ...       ...         ...            ...       ...         ...
4995  1.348396  0.558746    1.003872      -0.469265 -0.572281   -0.609567
4996 -1.120741 -0.862391   -0.968380       0.847230  0.739430    0.627630
4997  1.348396  1.123723    0.832019       0.518107  2.613303    1.400877
4998  0.113827  1.306436   -0.222109      -0.469265 -0.759668    0.009032
4999  0.113827  0.211674   -0.222109      -0.469265  0.364656    0.627630

      HouseYear  m_2_MedPriceByDistrict  m_2_MedPriceByHouseYear  Ecology_1  \
0     -0.667299               -1.011548                 0.650315   1.585280
1     -0.398066               -0.636732                -1.167186  -0.367283
2     -4.059632                4.768055                 2.275193  -0.998468
3      1.217331                0.238699                 0.802375  -0.149940
4      1.755796               -1.205267                -0.971467  -0.397445
...         ...                     ...                      ...        ...
4995  -0.613452                0.190383                 0.736246  -0.997055
4996   1.755796               -0.705405                -0.971467  -0.939144
4997   0.248093                0.218683                 0.522057  -0.242173
4998  -0.398066               -1.353291                -1.167186  -0.397445
4999  -0.398066               -1.132932                -1.167186  -0.997817

      Social_1  Social_3  Helthcare_2   Shops_1  Ecology_2_bin  Ecology_3_bin  \
0    -0.794834 -0.304366    -0.891612 -0.888194       0.098453       0.175258
1    -1.080052 -0.220549    -0.891612 -0.469511       0.098453       0.175258
2     0.288994  3.299785     2.487242  0.158513       0.098453       0.175258
3    -0.110311 -0.220549     1.135700 -0.260169       0.098453       0.175258
4    -1.308227 -0.304366    -0.891612 -0.888194       0.098453       0.175258
...        ...       ...          ...       ...            ...            ...
4995  0.631256 -0.346275    -0.215841 -0.678852       0.098453       0.175258
4996 -1.365270 -0.346275    -0.891612 -0.678852       0.098453       0.175258
```

```
4997  2.798913 -0.262458     2.487242  2.251927       0.098453       0.175
258
4998 -1.308227 -0.304366    -0.891612 -0.888194       0.098453       0.175
258
4999 -0.167355  5.562856     1.135700  3.926658       0.098453       0.175
258

      Shops_2_bin
0        0.299666
1        0.299666
2        0.299666
3        0.299666
4       -3.337053
...           ...
4995     0.299666
4996     0.299666
4997     0.299666
4998    -3.337053
4999     0.299666

[5000 rows x 17 columns]
```

In [ ]: