3.1 Wczytaj do Pandas Data Frame

In [57]:
```
import pandas as pd
import numpy as np

# Wczytaj do DataFrame
df = pd.read_csv('kc_house_data.csv',parse_dates=['date'])
df.head()
```

Out[57]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
0	7129300520	2014-10-13	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	2014-12-09	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	
3	2487200875	2014-12-09	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	

5 rows × 21 columns

In [58]:
```
print(len(df))
print(df.shape)

df.info()
print(df.columns)
```

```
21613
(21613, 21)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   id             21613 non-null  int64         
 1   date           21613 non-null  datetime64[ns]
 2   price          21613 non-null  float64       
 3   bedrooms       21613 non-null  int64         
 4   bathrooms      21613 non-null  float64       
 5   sqft_living    21613 non-null  int64         
 6   sqft_lot       21613 non-null  int64         
 7   floors         21613 non-null  float64       
 8   waterfront     21613 non-null  int64         
 9   view           21613 non-null  int64         
 10  condition      21613 non-null  int64         
 11  grade          21613 non-null  int64         
 12  sqft_above     21613 non-null  int64         
 13  sqft_basement  21613 non-null  int64         
 14  yr_built       21613 non-null  int64         
 15  yr_renovated   21613 non-null  int64         
 16  zipcode        21613 non-null  int64         
 17  lat            21613 non-null  float64       
 18  long           21613 non-null  float64       
 19  sqft_living15  21613 non-null  int64         
 20  sqft_lot15     21613 non-null  int64         
dtypes: datetime64[ns](1), float64(5), int64(15)
memory usage: 3.5 MB
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipco
de',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

3.2 Wyświetl informacje o zbiorze danych

In [59]:

```
    # informacje o danych
print('--- Min values ---')
print(df.min())
print('--- Max values ---')
print(df.max())
print('--- Mean values ---')
print(df.mean())
```

```
--- Min values ---
id                          1000102
date            2014-05-02 00:00:00
price                       75000.0
bedrooms                          0
bathrooms                       0.0
sqft_living                     290
sqft_lot                        520
floors                          1.0
waterfront                        0
view                              0
condition                         1
grade                             1
sqft_above                      290
sqft_basement                     0
yr_built                       1900
yr_renovated                      0
zipcode                       98001
lat                         47.1559
long                       -122.519
sqft_living15                   399
sqft_lot15                      651
dtype: object
--- Max values ---
id                       9900000190
date            2015-05-27 00:00:00
price                     7700000.0
bedrooms                         33
bathrooms                       8.0
sqft_living                   13540
sqft_lot                    1651359
floors                          3.5
waterfront                        1
view                              4
condition                         5
grade                            13
sqft_above                     9410
sqft_basement                  4820
yr_built                       2015
yr_renovated                   2015
zipcode                       98199
lat                         47.7776
long                       -121.315
sqft_living15                  6210
sqft_lot15                   871200
dtype: object
--- Mean values ---
id               4.580302e+09
price            5.400881e+05
bedrooms         3.370842e+00
bathrooms        2.114757e+00
sqft_living      2.079900e+03
sqft_lot         1.510697e+04
floors           1.494309e+00
waterfront       7.541757e-03
view             2.343034e-01
condition        3.409430e+00
grade            7.656873e+00
sqft_above       1.788391e+03
sqft_basement    2.915090e+02
```

```
yr_built         1.971005e+03
yr_renovated     8.440226e+01
zipcode          9.807794e+04
lat              4.756005e+01
long            -1.222139e+02
sqft_living15    1.986552e+03
sqft_lot15       1.276846e+04
dtype: float64
```

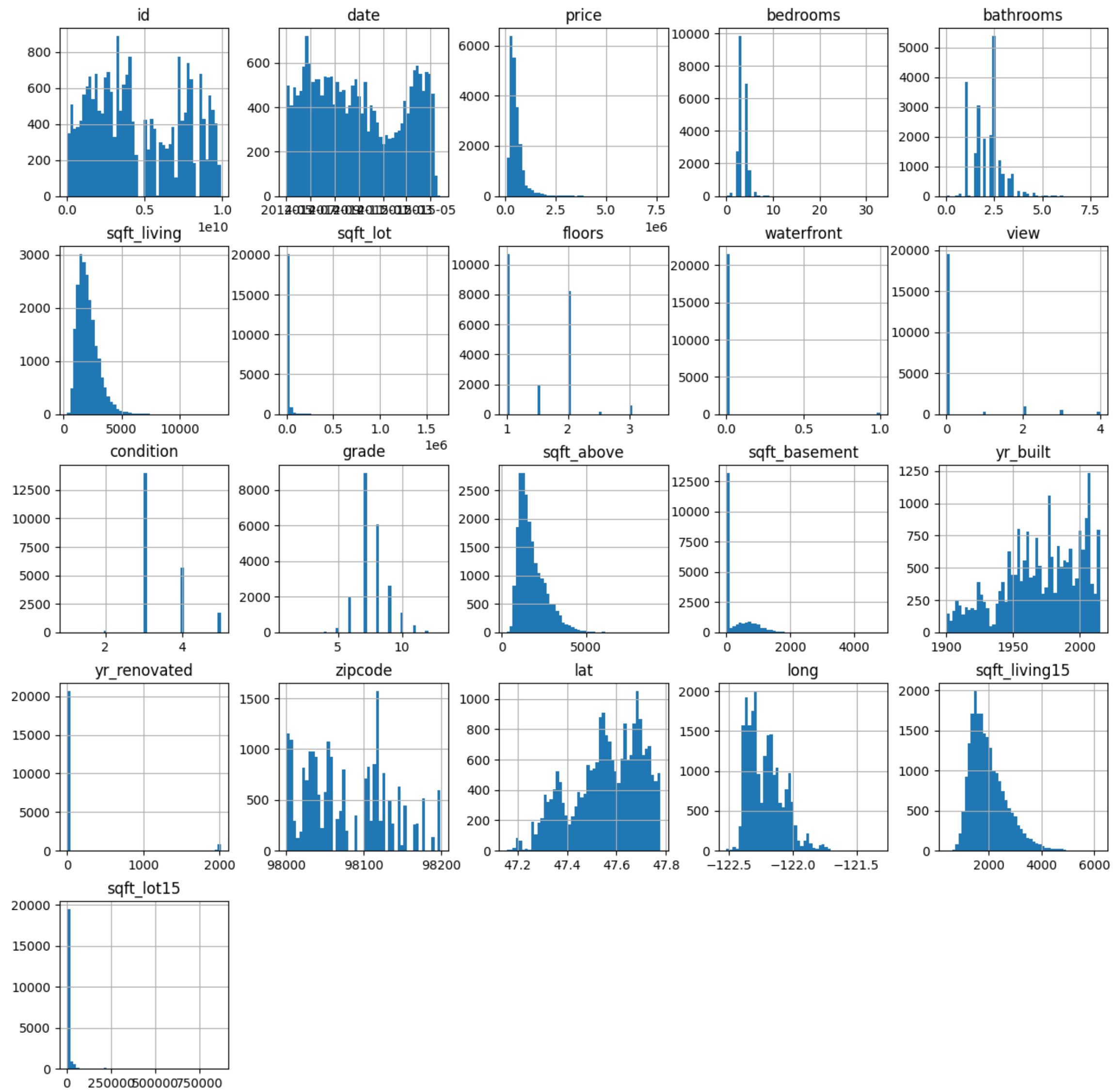
```
/var/folders/db/mk99b90s0f95rb60mwwyfc9m0000gq/T/ipykernel_71255/4151175
719.py:7: FutureWarning: DataFrame.mean and DataFrame.median with numeri
c_only=None will include datetime64 and datetime64tz columns in a future
version.
  print(df.mean())
```

In [60]:

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (15,15)
df.hist(bins=50)
```

Out[60]:

```
array([[<Axes: title={'center': 'id'}>, <Axes: title={'center': 'date'}
>,
        <Axes: title={'center': 'price'}>,
        <Axes: title={'center': 'bedrooms'}>,
        <Axes: title={'center': 'bathrooms'}>],
       [<Axes: title={'center': 'sqft_living'}>,
        <Axes: title={'center': 'sqft_lot'}>,
        <Axes: title={'center': 'floors'}>,
        <Axes: title={'center': 'waterfront'}>,
        <Axes: title={'center': 'view'}>],
       [<Axes: title={'center': 'condition'}>,
        <Axes: title={'center': 'grade'}>,
        <Axes: title={'center': 'sqft_above'}>,
        <Axes: title={'center': 'sqft_basement'}>,
        <Axes: title={'center': 'yr_built'}>],
       [<Axes: title={'center': 'yr_renovated'}>,
        <Axes: title={'center': 'zipcode'}>,
        <Axes: title={'center': 'lat'}>,
        <Axes: title={'center': 'long'}>,
        <Axes: title={'center': 'sqft_living15'}>],
       [<Axes: title={'center': 'sqft_lot15'}>, <Axes: >, <Axes: >,
        <Axes: >, <Axes: >]], dtype=object)
```



3.3 Usuwanie i konwersja danych

In [61]:
```
df2 = df.drop(columns=['id','zipcode'])
df2['date']=pd.to_numeric(df2['date'])
df2.head()
```

Out[61]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
0	1413158400000000000	221900.0	3	1.00	1180	5650	1.0
1	1418083200000000000	538000.0	3	2.25	2570	7242	2.0
2	1424822400000000000	180000.0	2	1.00	770	10000	1.0
3	1418083200000000000	604000.0	4	3.00	1960	5000	1.0
4	1424217600000000000	510000.0	3	2.00	1680	8080	1.0

3.4 Regresja

In [62]:
```
import sklearn.linear_model
regr = sklearn.linear_model.LinearRegression()

# y to wektor wartości wyjściowych
y =df2['price'].to_numpy()
```

```
# nie chcemy używać ceny w regresji bo otrzymalibyśmy równanie price=1.0*

df2_noprice=df2.drop(columns=['price'])
X=df2_noprice.to_numpy()

regr.fit(X, y)
regr.score(X,y)
```

Out[62]: 0.5424973730881661

3.4.2 Obliczamy metryki

In [63]:
```
import sklearn.metrics

y_pred = regr.predict(X)

scores={'r2':sklearn.metrics.r2_score,
        'mse':sklearn.metrics.mean_squared_error,
        'rmse':lambda y_true,y_pred : np.sqrt(sklearn.metrics.mean_square
        'maxe':sklearn.metrics.max_error,
        'med':sklearn.metrics.median_absolute_error,
        'mae':sklearn.metrics.mean_absolute_error,
        }
for k in scores:
  r = scores[k](y,y_pred)
  print(f'{k}:{r}')
```

```
r2:0.5424973730881661
mse:61660439113.24238
rmse:248315.20113203375
maxe:4198757.304702904
med:121134.33837447874
mae:164179.0515596507
```

3.4.3 Train Test Split

In [64]:
```
from sklearn.model_selection import train_test_split
print(X.shape)
print(y.shape)
for i in range(0,10):
  i=i+1
  test_size = 1-i*len(df.columns)/len(df)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = t
  regr = sklearn.linear_model.LinearRegression()

  regr.fit(X_train, y_train)
  y_pred = regr.predict(X_test)

  for k in scores:
    r = scores[k](y_test,y_pred)
    print(f'{k} : {r}')
  print('\n-------------------------------\n')
```

```
(21613, 18)
(21613,)
r2 : -12.045920017932007
mse : 1757089369440.7966
rmse : 1325552.4770603376
maxe : 57645025.436886765
med : 198300.9103661268
mae : 401036.9947502121

------------------------------

r2 : -0.31352106822298964
mse : 176948091478.3863
rmse : 420651.98380417307
maxe : 10577072.912042513
med : 146124.73288576258
mae : 227635.63579658527

------------------------------

r2 : 0.38805441149533737
mse : 82399501169.4458
rmse : 287053.13300754235
maxe : 4303932.515188776
med : 142581.31230199616
mae : 192723.091279381

------------------------------

r2 : 0.4243600344974464
mse : 77545617822.80075
rmse : 278470.1381168199
maxe : 4372490.119800981
med : 139531.3888453599
mae : 187526.07054353764

------------------------------

r2 : 0.4723233604881253
mse : 71114291179.79428
rmse : 266672.6292287873
maxe : 4342862.875569306
med : 131693.8518817611
mae : 178007.7694801556

------------------------------

r2 : 0.4718118035748162
mse : 71209352626.38864
rmse : 266850.8059316828
maxe : 4363442.418498281
med : 129940.65737752011
mae : 176592.452890065

------------------------------

r2 : 0.48887564613279877
mse : 68908639466.1662
rmse : 262504.5513246698
maxe : 4358620.046643781
```

```
med : 128378.3547745464
mae : 174753.99860505055

------------------------------

r2 : 0.5179594897942638
mse : 64987736047.0358
rmse : 254926.92295447298
maxe : 4285267.502140472
med : 126174.04506579245
mae : 170382.04668709982

------------------------------

r2 : 0.4981450834679396
mse : 67587506885.001724
rmse : 259975.97366872526
maxe : 4195880.8614141485
med : 131951.76562171313
mae : 177376.81045385328

------------------------------

r2 : 0.5001977291054609
mse : 67316148651.67134
rmse : 259453.55779343506
maxe : 4265197.892705477
med : 130413.25346377445
mae : 175867.1014084718

------------------------------
```

Tak, wyniki zaleza od wielkosci zbioru uczacego.

k	r2	rmse
k=1	-15.183	2.118
k=2	-0.180	0.572
k=5	0.446	0.391
k=7	0.443	0.392
k=10	0.502	0.371
k=50	0.532	0.360
k=100	0.536	0.359

Dla zbyt malej ilosci danych, otrzymane przebiegi nie sa zadawalajace. Dla wiekszych ilosci danych (np. od 5 do 20 razy wiekszych od liczby atrybutow) wyniki staja sie coraz lepsze.

3.4.4 Wydruk równania regresji

Czy na podstawie wag można określić, które atrybuty mają mały/duży wpływ na wynik?

- na wyniki duzy wplyw mialy na pewno: lazienki, wielkosc mieszkania, warunki, wielkosc piwnicy

Jaką operację należałoby przeprowadzić, aby uzyskać wiarygodny wynik?

- nalezaloby wyelimowac zbyt silnie oraz slabo skorelowane atrybuty

In [65]:
```
def print_formula(regr,labels,target):
  print(f'{target} = ')
  for i in range(len(regr.coef_)):
    print(f'\t{regr.coef_[i]: .3g}\t* {labels[i]} +')
  print(f'\t{regr.intercept_:.8}')

print_formula(regr,df2_noprice.columns,'price')
```

```
price =
         7.69e-13       * date +
         2.43   * bedrooms +
        -7.12   * bathrooms +
         171    * sqft_living +
         0.742  * sqft_lot +
        -5.52   * floors +
        -0.186  * waterfront +
         0.666  * view +
         6.29   * condition +
        -6.86   * grade +
         73.7   * sqft_above +
         97.5   * sqft_basement +
        -782    * yr_built +
         188    * yr_renovated +
         0.635  * lat +
        -1.06   * long +
         93     * sqft_living15 +
        -2.57   * sqft_lot15 +
        315419.05
```

3.4.5 k-fold CrossValidation

In [66]:
```
from sklearn.model_selection import KFold
from sklearn.utils import Bunch

# słownik na składowanie wyników
b = Bunch()
for k in scores:
  b[k]=[]
  # b[k].append(0)

print(b)

# train_index, test_index to indeksy wierszy użytych jako zbiory treningo

kf = KFold(n_splits=10)
for train_index, test_index in kf.split(X, y):
  X_train,y_train = X[train_index],y[train_index]
  X_test,y_test = X[test_index],y[test_index]

  regr = sklearn.linear_model.LinearRegression()

  regr.fit(X_train, y_train)
```

```
  y_pred = regr.predict(X_test)

  for k in scores:
    r = scores[k](y_test,y_pred)
    b[k].append(r)

for k in b:
    print(k, sum(b[k])/len(b[k]))
```

```
{'r2': [], 'mse': [], 'rmse': [], 'maxe': [], 'med': [], 'mae': []}
r2 0.5343584071378065
mse 62621502827.79281
rmse 249280.66829693364
maxe 3011899.6752815684
med 123699.88749120357
mae 165615.99406906642
```

oprzec sie na mierze wspolczynnika determinacji (niezej)

3.4.6 Przetestuj Ridge i Lasso

In [67]:
```
def train_and_test(X,y,regr=sklearn.linear_model.LinearRegression()):
  # print(regr)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0

  regr.fit(X_train, y_train)
  y_pred = regr.predict(X_test)

  for k in scores:
    r = scores[k](y_test,y_pred)
    print(f'{k}:{r}')
```

In [68]:
```
train_and_test(X, y)
```

```
r2:0.5545286393159092
mse:60494838652.817444
rmse:245956.98537105517
maxe:4197978.569396781
med:122056.52731704153
mae:163797.5277726916
```

In [69]:
```
train_and_test(X, y, sklearn.linear_model.Ridge())
```

```
r2:0.706706905616217
mse:39829088890.216866
rmse:199572.26483210752
maxe:4121780.551017642
med:88994.11757771671
mae:126485.7419669327
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-p
ackages/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-condition
ed matrix (rcond=6.9088e-37): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

In [70]:
```
train_and_test(X, y, sklearn.linear_model.Lasso())
```

```
r2:0.7066067631913553
mse:39842688192.81853
rmse:199606.33304787334
maxe:4122756.9140735045
med:88976.38624636456
mae:126506.01996811922
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-p
ackages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarn
ing: Objective did not converge. You might want to increase the number o
f iterations, check the scale of the features or consider increasing reg
ularisation. Duality gap: 3.129e+14, tolerance: 2.032e+11
  model = cd_fast.enet_coordinate_descent(
```

Rodzaj	r2	mse	rmse	maxe	
Linear	0.554	60494838652.817444	245956.98537105517	4197978.569396781	122056.!
Ridge	0.706	39829088890.216866	199572.26483210752	4121780.551017642	88994.1
Lasso	0.706	39842688192.81853	199606.33304787334	4122756.9140735045	88976.3

3.5 Analiza danych

3.5.1 Czy atrybuty są skorelowane?

In [71]:
```python
import scipy.stats

n = len(df2.columns)
rs = np.zeros((n,n))
# print(rs)

# print(df.iloc[:,0])
for i in range(n):
  for j in range(n):
    r,p = scipy.stats.pearsonr(df2.iloc[:,i],df2.iloc[:,j])
    rs[i,j]= r
```

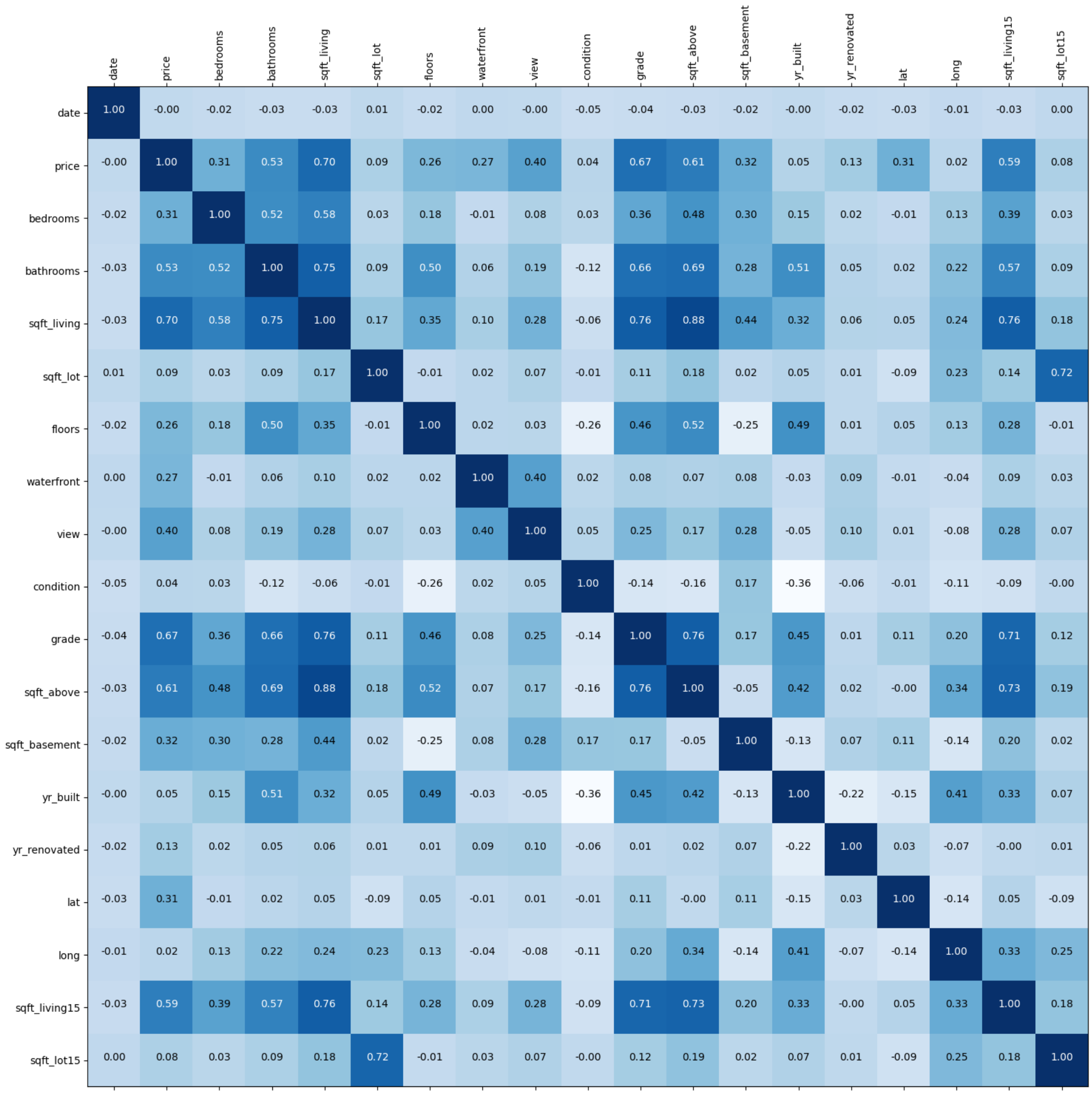
In [72]:
```python
import itertools
def plot_matrix(cm, labels,
                    normalize=False,
                    title='',
                    cmap=plt.cm.Blues):
    fig = plt.figure()
    ax=fig.add_subplot(111)
    ax.matshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    # plt.colorbar()
    tick_marks = np.arange(len(labels))
    plt.xticks(tick_marks, labels, rotation=90)
    ax.set_xticks(tick_marks)
    ax.set_yticks(tick_marks)
    ax.set_xticklabels(labels)
    ax.set_yticklabels(['']+labels)

    fmt = '.2f' #if normalize else 'd'
    thresh = 0.5
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1]))
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
```

```
                    color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.show()

plot_matrix(rs,labels=df2.columns)
```



powinninsmy zostawic silnie skorelowane atrybuty z cena

3.5.2 Przetestuj działanie dla podzbiorów atrybutów

In [73]:
```
df3 = df2_noprice.drop(columns = ['date', 'sqft_above', 'sqft_lot'])
X = df3.to_numpy()
regr = sklearn.linear_model.LinearRegression()
train_and_test(X,y,regr)
print_formula(regr,df3.columns,'price')
```

```
r2:0.705076188313879
mse:40050539669.77271
rmse:200126.30928934034
maxe:4113148.1846670434
med:89576.62161863968
mae:126911.42679537342
price =
        -3.23e+04       * bedrooms +
         4.05e+04       * bathrooms +
         177    * sqft_living +
        -2.62e+03       * floors +
         6.37e+05       * waterfront +
         4.81e+04       * view +
         3.01e+04       * condition +
         1.03e+05       * grade +
        -33.6   * sqft_basement +
        -2.5e+03        * yr_built +
         21     * yr_renovated +
         5.58e+05       * lat +
        -1.19e+05       * long +
         24.6   * sqft_living15 +
        -0.242  * sqft_lot15 +
        -3.6863234e+07
```

In [74]:
```
df3 = df2_noprice.drop(columns = ['date', 'sqft_above', 'sqft_lot'])
X = df3.to_numpy()
regr = sklearn.linear_model.Ridge()
train_and_test(X,y,regr)
print_formula(regr,df3.columns,'price')
```

```
r2:0.7051816871105201
mse:40036212973.953094
rmse:200090.51195384827
maxe:4112228.608455725
med:89433.52876338735
mae:126888.53478888674
price =
        -3.23e+04       * bedrooms +
         4.04e+04       * bathrooms +
         177    * sqft_living +
        -2.5e+03        * floors +
         6.3e+05        * waterfront +
         4.84e+04       * view +
         3.01e+04       * condition +
         1.03e+05       * grade +
        -33.5   * sqft_basement +
        -2.51e+03       * yr_built +
         21.1   * yr_renovated +
         5.55e+05       * lat +
        -1.19e+05       * long +
         24.6   * sqft_living15 +
        -0.243  * sqft_lot15 +
        -3.6700434e+07
```

In [75]:
```
df3 = df2_noprice.drop(columns = ['date', 'sqft_above', 'sqft_lot'])
X = df3.to_numpy()
regr = sklearn.linear_model.Lasso()
train_and_test(X,y,regr)
print_formula(regr,df3.columns,'price')
```

```
r2:0.7050790677798002
mse:40050148639.413216
rmse:200125.3323280519
maxe:4113168.6752446145
med:89569.64516450092
mae:126910.25553445228
price =
        -3.23e+04       * bedrooms +
         4.05e+04       * bathrooms +
         177    * sqft_living +
        -2.6e+03        * floors +
         6.37e+05       * waterfront +
         4.81e+04       * view +
         3.01e+04       * condition +
         1.03e+05       * grade +
        -33.6   * sqft_basement +
        -2.5e+03        * yr_built +
         21     * yr_renovated +
         5.58e+05       * lat +
        -1.19e+05       * long +
         24.6   * sqft_living15 +
        -0.242  * sqft_lot15 +
        -3.6851391e+07
```

In [76]:
```
df3 = df2_noprice.drop(columns = ['date', 'yr_renovated', 'sqft_lot15', '
X = df3.to_numpy()
regr = sklearn.linear_model.LinearRegression()
train_and_test(X,y,regr)
print_formula(regr,df3.columns,'price')
```

```
r2:0.7042401135907008
mse:40164078294.12488
rmse:200409.77594450046
maxe:4102222.511117488
med:89412.29947353154
mae:127211.10858862997
price =
        -3.2e+04        * bedrooms +
         4.33e+04       * bathrooms +
         142    * sqft_living +
        -0.0325 * sqft_lot +
        -717    * floors +
         6.42e+05       * waterfront +
         4.82e+04       * view +
         2.78e+04       * condition +
         1.03e+05       * grade +
         32.7   * sqft_above +
        -2.63e+03       * yr_built +
         5.57e+05       * lat +
        -1.23e+05       * long +
         23.3   * sqft_living15 +
        -3.7140331e+07
```

In [77]:
```
df3 = df2_noprice.drop(columns = ['date', 'yr_renovated', 'sqft_lot15', '
X = df3.to_numpy()
regr = sklearn.linear_model.Ridge()
train_and_test(X,y,regr)
print_formula(regr,df3.columns,'price')
```

```
r2:0.7043458435416984
mse:40149720207.63298
rmse:200373.95092085443
maxe:4101236.303773068
med:89454.63531115279
mae:127188.690665646
price =
        -3.21e+04       * bedrooms +
         4.33e+04       * bathrooms +
         142    * sqft_living +
        -0.0336 * sqft_lot +
        -600    * floors +
         6.35e+05       * waterfront +
         4.85e+04       * view +
         2.77e+04       * condition +
         1.03e+05       * grade +
         32.6   * sqft_above +
        -2.64e+03       * yr_built +
         5.55e+05       * lat +
        -1.23e+05       * long +
         23.3   * sqft_living15 +
        -3.6971684e+07
```

In [78]:
```
df3 = df2_noprice.drop(columns = ['date', 'yr_renovated', 'sqft_lot15', '
X = df3.to_numpy()
regr = sklearn.linear_model.Lasso()
train_and_test(X,y,regr)
print_formula(regr,df3.columns,'price')
```

```
r2:0.7042430056136694
mse:40163685558.523224
rmse:200408.79611065783
maxe:4102241.178395182
med:89406.00863466412
mae:127209.92418547647
price =
        -3.2e+04        * bedrooms +
         4.33e+04       * bathrooms +
         142    * sqft_living +
        -0.0325 * sqft_lot +
        -703    * floors +
         6.42e+05       * waterfront +
         4.82e+04       * view +
         2.78e+04       * condition +
         1.03e+05       * grade +
         32.7   * sqft_above +
        -2.63e+03       * yr_built +
         5.57e+05       * lat +
        -1.23e+05       * long +
         23.3   * sqft_living15 +
        -3.7128453e+07
```

In [79]:
```
df3 = df2_noprice.drop(columns = ['date', 'sqft_above', 'condition', 'lor
X = df3.to_numpy()
regr = sklearn.linear_model.LinearRegression()
train_and_test(X,y,regr)
print_formula(regr,df3.columns,'price')
```

```
r2:0.7013129134070679
mse:40561590948.00557
rmse:201399.08378144517
maxe:4170712.9730094224
med:89356.96975279786
mae:127573.78567347403
price =
        -3.15e+04       * bedrooms +
         4.13e+04       * bathrooms +
         164    * sqft_living +
         0.113  * sqft_lot +
         6.87e+03       * floors +
         6.44e+05       * waterfront +
         4.8e+04        * view +
         1.08e+05       * grade +
        -2.98e+03       * yr_built +
         12     * yr_renovated +
         5.47e+05       * lat +
         20.1   * sqft_living15 +
        -0.43   * sqft_lot15 +
        -2.0794413e+07
```

In [80]:
```
df3 = df2_noprice.drop(columns = ['date', 'sqft_above', 'condition', 'lon
X = df3.to_numpy()
regr = sklearn.linear_model.Ridge()
train_and_test(X,y,regr)
print_formula(regr,df3.columns,'price')
```

```
r2:0.7014178643032892
mse:40547338656.85125
rmse:201363.69746518673
maxe:4169536.5803499967
med:89375.49332806468
mae:127552.48567638772
price =
        -3.15e+04       * bedrooms +
         4.13e+04       * bathrooms +
         164    * sqft_living +
         0.113  * sqft_lot +
         6.94e+03       * floors +
         6.36e+05       * waterfront +
         4.82e+04       * view +
         1.08e+05       * grade +
        -2.98e+03       * yr_built +
         12     * yr_renovated +
         5.44e+05       * lat +
         20.1   * sqft_living15 +
        -0.43   * sqft_lot15 +
        -2.0689411e+07
```

In [81]:
```
df3 = df2_noprice.drop(columns = ['date', 'sqft_above', 'condition', 'lon
X = df3.to_numpy()
regr = sklearn.linear_model.Lasso()
train_and_test(X,y,regr)
print_formula(regr,df3.columns,'price')
```

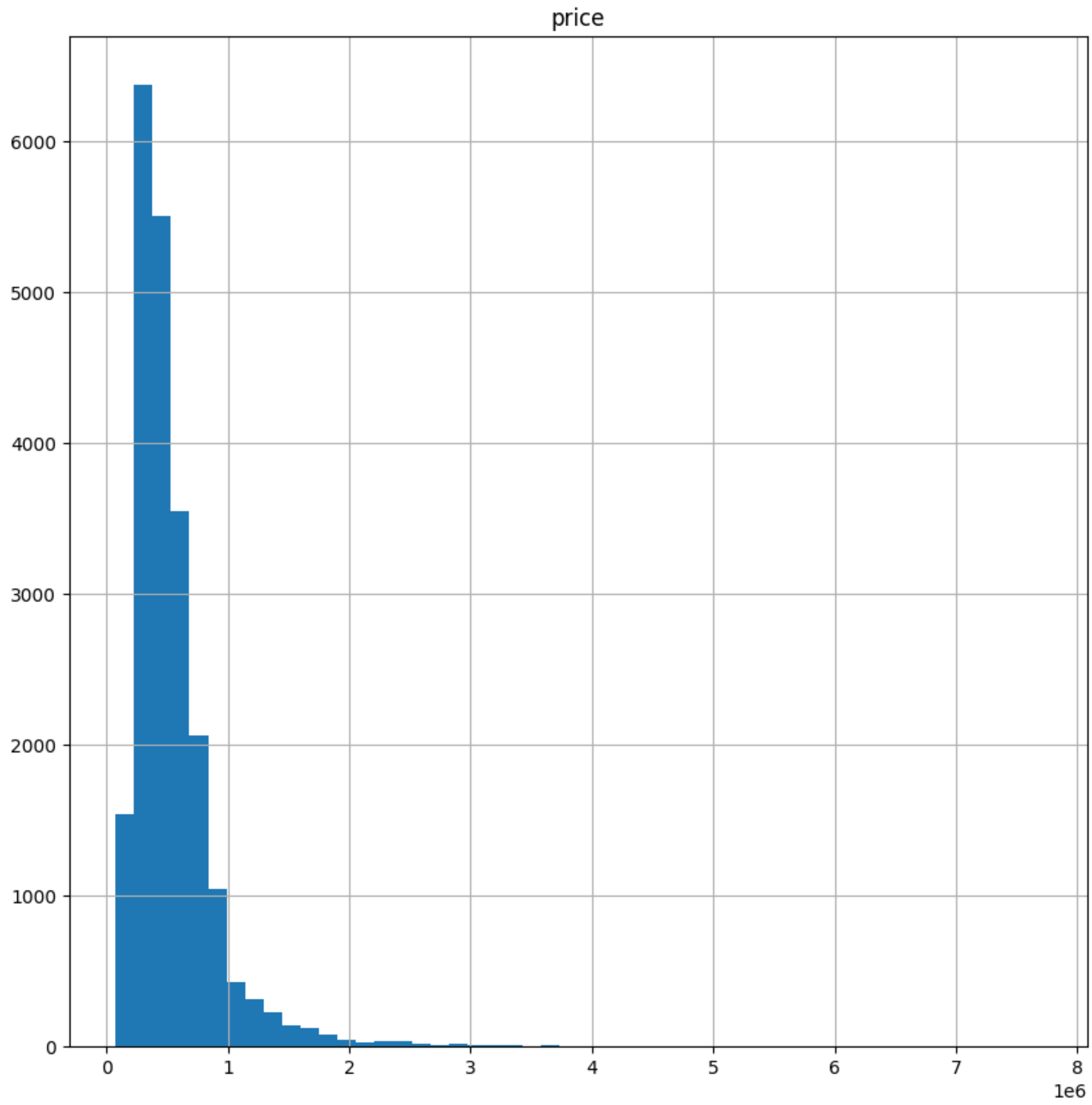
```
r2:0.7013153559042447
mse:40561259257.832726
rmse:201398.26031481187
maxe:4170694.476348508
med:89353.6371756699
mae:127573.23992220999
price =
        -3.15e+04       * bedrooms +
         4.13e+04       * bathrooms +
         164    * sqft_living +
         0.113  * sqft_lot +
         6.86e+03       * floors +
         6.43e+05       * waterfront +
         4.8e+04        * view +
         1.08e+05       * grade +
        -2.98e+03       * yr_built +
         12     * yr_renovated +
         5.46e+05       * lat +
         20.1   * sqft_living15 +
        -0.43   * sqft_lot15 +
        -2.079179e+07
```

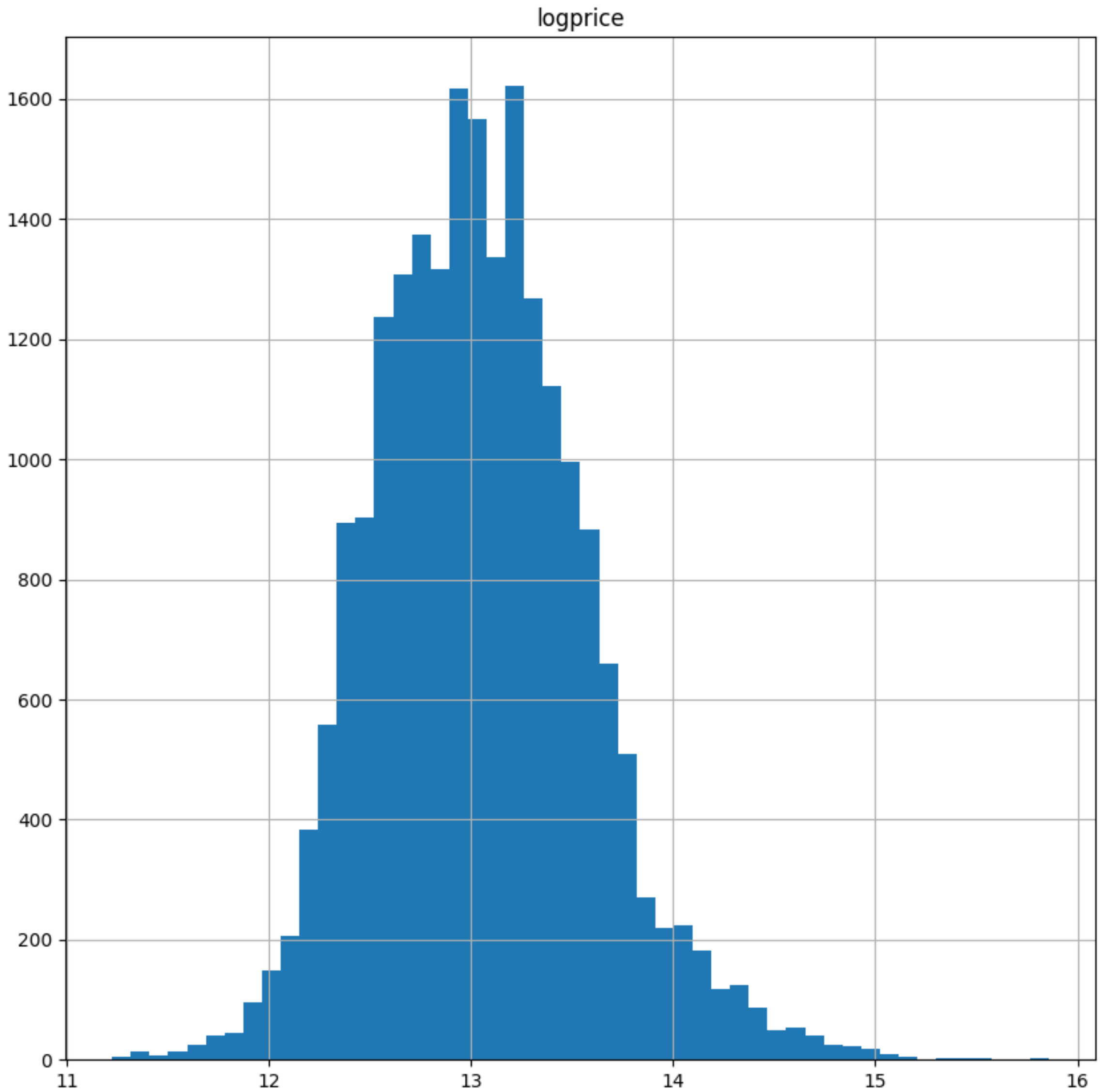
Rodzaj	r2	usuniete kolumny
Linear	0.772	'date', 'sqft_above', 'sqft_lot'
Ridge	0.773	'date', 'sqft_above', 'sqft_lot'
Lasso	0.541	'date', 'sqft_above', 'sqft_lot'
Linear	0.772	'date', 'yr_renovated', 'sqft_lot15', 'sqft_basement'
Ridge	0.772	'date', 'yr_renovated', 'sqft_lot15', 'sqft_basement'
Lasso	0.537	'date', 'yr_renovated', 'sqft_lot15', 'sqft_basement'
Linear	0.767	'date', 'sqft_above', 'condition', 'long', 'sqft_basement'
Ridge	0.767	'date', 'sqft_above', 'condition', 'long', 'sqft_basement'
Lasso	0.540	'date', 'sqft_above', 'condition', 'long', 'sqft_basement'

3.5.3 A może transformacja danych?

In [82]:
```
plt.rcParams["figure.figsize"] = (10,10)
df2.hist('price',bins=50)
df2_log=pd.DataFrame(df2)
df2_log['logprice']=np.log(df2['price'])
df2_log.hist('logprice',bins=50)
```

Out[82]:
```
array([[<Axes: title={'center': 'logprice'}>]], dtype=object)
```


price
6000
5000
4000
3000
2000
1000
0
0
1
2
3
4
5
6
7
8
1e6



Rozkladem otrzymanyn byl wykres ktory ksztaltem przypomina rozklad Gaussa. Jest to zgodne z zalozeniami poniewaz jest to wynik oczekiwany dla metody regresji.

In [83]:
```
def train_and_test_log(X,y,regr=sklearn.linear_model.LinearRegression()):

  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0


  regr.fit(X_train, y_train)
  y_pred = regr.predict(X_test)
  #y_pred=np.log() logarytm ceny costam

  for k in scores:
    r = scores[k](y_test,y_pred)
    print(f'{k}:{r}')

df3=df2_noprice
X = df3.to_numpy()
y = np.log(y)
regr = sklearn.linear_model.Ridge(solver='svd')
train_and_test_log(X,y,regr)
```

```
r2:0.7750220834768263
mse:0.06388142885555524
rmse:0.2527477573699819
maxe:1.2336732650456756
med:0.16109330707012237
mae:0.19584137928146822
```

3.5.4 Normalizacja cech

Wykonaj testy używając X_n zamiast X. Czy poprawiły się wyniki? Wydrukuj równanie regresji. Które współczynniki mają teraz największy wpływ na cenę?

- wspolczynnik skorelowania poprawil sie, wartosci osiagnely wyniki podobne jak w przypadku braku normalizacji dla algorytmow Ridge oraz Lasso

In [84]:
```
from sklearn.preprocessing import StandardScaler
X=df2_noprice.to_numpy()
scaler = StandardScaler() #poprzesuwa dane i podzieli
X_n=scaler.fit_transform(X)
```

In [85]:
```
train_and_test_log(X_n, y)
```

```
r2:0.7751128491865857
mse:0.0638556563561023
rmse:0.2526967676012147
maxe:1.2257037723762796
med:0.15935460039096316
mae:0.19564393991887086
```

3.6 Regresja wielomianowa

In [86]:
```
from sklearn.preprocessing import PolynomialFeatures

# df3 = df2_noprice.drop(columns = [....])
df3 = df2_noprice
X = df3.to_numpy()
poly = PolynomialFeatures(degree=2)
X=poly.fit_transform(X)

regr = sklearn.linear_model.Lasso()
train_and_test(X,y,regr)
# print_formula(regr,df3.columns,'price')
print(f'Liczba parametrów modelu: {len(regr.coef_)+1}')
```

```
r2:0.7846219376118118
mse:0.06115559510073597
rmse:0.24729657316820217
maxe:1.7513077248876048
med:0.15445119241894378
mae:0.19085301353212134
Liczba parametrów modelu: 191
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-p
ackages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarn
ing: Objective did not converge. You might want to increase the number o
f iterations, check the scale of the features or consider increasing reg
ularisation. Duality gap: 4.578e+02, tolerance: 4.154e-01
  model = cd_fast.enet_coordinate_descent(
```

In [87]:
```
# df3 = df2_noprice.drop(columns = [....])
df3 = df2_noprice
X = df3.to_numpy()
poly = PolynomialFeatures(degree=2)
X=poly.fit_transform(X)

regr = sklearn.linear_model.LinearRegression()
train_and_test(X,y,regr)
# print_formula(regr,df3.columns,'price')
print(f'Liczba parametrów modelu: {len(regr.coef_)+1}')
```

```
r2:0.5429647818591246
mse:0.12977301605129848
rmse:0.36024021992456434
maxe:1.3910780858977887
med:0.2551659598954892
mae:0.2914436991342734
Liczba parametrów modelu: 191
```

In [88]:
```
#df3 = df2_noprice.drop(columns = [....])
df3 = df2_noprice
X = df3.to_numpy()
poly = PolynomialFeatures(degree=2)
X=poly.fit_transform(X)

regr = sklearn.linear_model.Ridge()
train_and_test(X,y,regr)
# print_formula(regr,df3.columns,'price')
print(f'Liczba parametrów modelu: {len(regr.coef_)+1}')
```

```
r2:0.5878613290785988
mse:0.11702485111411676
rmse:0.3420889520491955
maxe:1.3545782502082098
med:0.23986234742846957
mae:0.275326022265641
Liczba parametrów modelu: 191
```

In [89]:
```
#df3 = df2_noprice.drop(columns = [....])
df3 = df2_noprice
X = df3.to_numpy()
poly = PolynomialFeatures(degree=3)
X=poly.fit_transform(X)

regr = sklearn.linear_model.Ridge()
train_and_test(X,y,regr)
# print_formula(regr,df3.columns,'price')
print(f'Liczba parametrów modelu: {len(regr.coef_)+1}')
```

```
r2:0.5949902285638817
mse:0.11500063339388286
rmse:0.33911743304330855
maxe:1.3069211482291383
med:0.23885593138210037
mae:0.273350635511974
Liczba parametrów modelu: 1331
```

In [90]:
```
#df3 = df2_noprice.drop(columns = [....])
df3 = df2_noprice
X = df3.to_numpy()
```

```
poly = PolynomialFeatures(degree=3)
X=poly.fit_transform(X)

regr = sklearn.linear_model.Lasso()
train_and_test(X,y,regr)
# print_formula(regr,df3.columns,'price')
print(f'Liczba parametrów modelu: {len(regr.coef_)+1}')
```

```
r2:0.625644057607769
mse:0.10629662177585489
rmse:0.3260316269564272
maxe:16.86770786626537
med:0.15060985308719665
mae:0.1910851280249474
Liczba parametrów modelu: 1331
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-p
ackages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarn
ing: Objective did not converge. You might want to increase the number o
f iterations, check the scale of the features or consider increasing reg
ularisation. Duality gap: 4.273e+02, tolerance: 4.154e-01
  model = cd_fast.enet_coordinate_descent(
```

In [91]:
```
#df3 = df2_noprice.drop(columns = [....])
df3 = df2_noprice
X = df3.to_numpy()
poly = PolynomialFeatures(degree=3)
X=poly.fit_transform(X)

regr = sklearn.linear_model.LinearRegression()
train_and_test(X,y,regr)
# print_formula(regr,df3.columns,'price')
print(f'Liczba parametrów modelu: {len(regr.coef_)+1}')
```

```
r2:0.5429691451403853
mse:0.12977177711795895
rmse:0.3602385003271568
maxe:1.38987855564792
med:0.25509838868420776
mae:0.2914457394752219
Liczba parametrów modelu: 1331
```

rzad	rodzaj	r2
2 stopien	Linear	0.542
2 stopien	Lasso	0.784
2 stopien	Ridge	0.587
3 stopien	Linear	0.542
3 stopien	Lasso	0.625
3 stopien	Ridge	0.594

3.7 Inne algorytmy regresji

In [95]:
```
from sklearn.linear_model import ElasticNet, LassoLars, BayesianRidge, SG
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.pipeline import make_pipeline
```

```
from sklearn.preprocessing import StandardScaler

df3 = df2_noprice
X = df3.to_numpy()

regressors = [
              ElasticNet(),
              LassoLars(alpha=.1),
              BayesianRidge(),
              make_pipeline(StandardScaler(),SGDRegressor(max_iter=1000,
              SVR(kernel='poly',degree=2),
              DecisionTreeRegressor(),
              XGBRegressor(),
              ]

for regr in regressors:
  print(f'------------ {regr.__class__.__name__} ---------------')
  train_and_test(X,y,regr)
```

```
------------ ElasticNet ---------------
r2:0.5432289667904523
mse:0.12969800197366751
rmse:0.3601360881301227
maxe:1.3578520865827048
med:0.25927390029040875
mae:0.2923591161734078
------------ LassoLars ---------------
r2:0.5511145407078872
mse:0.12745893008173825
rmse:0.3570139074066139
maxe:1.3595795321700557
med:0.25446613742515733
mae:0.28891528675432504
------------ BayesianRidge ---------------
r2:0.77501759436767
mse:0.06388270351713814
rmse:0.2527502789655001
maxe:1.2341520940193167
med:0.16094010925301472
mae:0.19583897247928744
------------ Pipeline ---------------
r2:0.7734218577663614
mse:0.0643358054737393
rmse:0.25364503833850033
maxe:1.2645880290065445
med:0.1599144426879775
mae:0.19629713676744132
------------ SVR ---------------
r2:-0.002500118390242001
mse:0.28465522741222227
rmse:0.5335309057704364
maxe:2.7219502288846122
med:0.35142045721728987
mae:0.4188620920169213
------------ DecisionTreeRegressor ---------------
r2:0.7732240327346688
mse:0.06439197696773845
rmse:0.2537557427285902
maxe:1.570217199280819
med:0.13155236859032815
mae:0.18206044707565328
------------ XGBRegressor ---------------
r2:0.9022166134520077
mse:0.02776513600781529
rmse:0.16662873704080966
maxe:1.1854580998076312
med:0.08669298231840994
mae:0.11877524142451434
```

rodzaj	r2
ElasticNet	0.543
LassoLars	0.551
BayesianRidge	0.775
Pipeline	0.773
SVR	-0.002

rodzaj	r2
DecisionTreeRegressor	0.770
XGBRegressor	0.902

XGBRegressor dal zdecydowanie najlpesze rezultaty. Rowniez metody takie jak DecisionTreeRegressor, Pipeline oraz BayessianRidge daly dobre wyniki. Najgorzej sprawdzil sie SVR.