Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления» Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Технологии машинного обучения»
Отчет по лабораторной работе №4
«Линейные модели, SVM и деревья решений.»

Выполнил:
Студент(ка) группы ИУ5-65Б
Тазенков Иван Дмитриевич

Подпись:_____

Дата:_____

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий Евгеньевич

Подпись:_____

Дата:_____

Москва, 2023 г.

Задание:

1 Выберите набор данных (датасет) для решения задачи классификации или регрессии.

2 В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.

3 С использованием метода train_test_split разделите выборку на обучающую и тестовую.

4 Обучите следующие модели:

- одну из линейных моделей (линейную или полиномиальную регрессию при решении задачи регрессии, логистическую регрессию при решении задачи классификации);
- SVM;
- дерево решений.

5 Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

6 Постройте график, показывающий важность признаков в дереве решений.

7 Визуализируйте дерево решений или выведите правила дерева решений в текстовом виде.

Homework

In [1]:

```
!wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1OKFSv2GpuUFDphO0r8LdM7bl6MAWwBfX' -O data.csv
```

```
--2022-02-19 07:31:11--  https://docs.google.com/uc?export=download&id=1OKFSv2GpuUFDphO0r8LdM7bl6MAWwBfX
Resolving docs.google.com (docs.google.com)... 74.125.124.139, 74.125.124.100, 74.125.124.113, ...
Connecting to docs.google.com (docs.google.com)|74.125.124.139|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-04-ak-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1/msvaed2k552t88j2dur5t8jfqv7rj89l/1645255800000/03856158561714992485/*/1OKFSv2GpuUFDphO0r8LdM7bl6MAWwBfX?e=download [following]
Warning: wildcards not supported in HTTP.
--2022-02-19 07:31:13--  https://doc-04-ak-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1/msvaed2k552t88j2dur5t8jfqv7rj89l/1645255800000/03856158561714992485/*/1OKFSv2GpuUFDphO0r8LdM7bl6MAWwBfX?e=download
Resolving doc-04-ak-docs.googleusercontent.com (doc-04-ak-docs.googleusercontent.com)... 74.125.70.132, 2607:f8b0:4001:c02::84
Connecting to doc-04-ak-docs.googleusercontent.com (doc-04-ak-docs.googleusercontent.com)|74.125.70.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 33322228 (32M) [text/csv]
Saving to: ‘data.csv’

data.csv            100%[===================>]  31.78M  89.7MB/s    in 0.4s

2022-02-19 07:31:13 (89.7 MB/s) - ‘data.csv’ saved [33322228/33322228]
```

В этой домашней работе вы будете предсказывать стоимость домов по их характеристикам.

Метрика качества: `RMSE`

Оценивание:

- **Baseline - 2** балла
- **Feature Engineering - 2** балла
- **Model Selection - 3** балла
- **Ensemble v.1 - 3** балла
- **(*) Ensemble v.2 -** дополнительно**, 2** балла

Описание датасета

Короткое описание данных:

```
price: sale price (this is the target variable)
id: transaction id
timestamp: date of transaction
full_sq: total area in square meters, including loggias, balconies and other non-residential areas
life_sq: living area in square meters, excluding loggias, balconies and other non-residential areas
floor: for apartments, floor of the building
max_floor: number of floors in the building
material: wall material
build_year: year built
num_room: number of living rooms
kitch_sq: kitchen area
```

```
state: apartment condition
product_type: owner-occupier purchase or investment
sub_area: name of the district

The dataset also includes a collection of features about each property's surroundin
g neighbourhood, and some features that are constant across each sub area (known as
a Raion). Most of the feature names are self explanatory, with the following notes.
See below for a complete list.

full_all: subarea population
male_f, female_f: subarea population by gender
young_*: population younger than working age
work_*: working-age population
ekder_*: retirement-age population
n_m_{all|male|female}: population between n and m years old
build_count_*: buildings in the subarea by construction type or year
x_count_500: the number of x within 500m of the property
x_part_500: the share of x within 500m of the property
_sqm_: square meters
cafe_count_d_price_p: number of cafes within d meters of the property that have an
average bill under p RUB
trc_: shopping malls
prom_: industrial zones
green_: green zones
metro_: subway
_avto_: distances by car
mkad_: Moscow Circle Auto Road
ttk_: Third Transport Ring
sadovoe_: Garden Ring
bulvar_ring_: Boulevard Ring
kremlin_: City center
zd_vokzaly_: Train station
oil_chemistry_: Dirty industry
ts_: Power plant
```

Setup

In [158]:

```
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tqdm import tqdm
import warnings
warnings.filterwarnings("ignore")

#ML algorithms
from sklearn.model_selection import KFold
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import HuberRegressor
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from catboost import CatBoostRegressor
import xgboost
```

```
import lightgbm
```

In [4]:

```
df = pd.read_csv("data.csv", parse_dates=["timestamp"])
```

Разделите имеющиеся у вас данные на обучающую и тестовую выборки. В качестве обучающей выборки возьмите первые **80%** данных, последние **20%** - тестовая выборка.

In [5]:

```
df.head()
```

Out[5]:

	id	timestamp	full_sq	life_sq	floor	max_floor	material	build_year	num_room	kitch_sq	state	product_type	sub_ar
0	**0**	**2014-12-26**	**1**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**OwnerOccupier**	**Nagorn**
1	**1**	**2012-10-04**	**64**	**64.0**	**16.0**	**NaN**	**NaN**	**NaN**	**NaN**	**NaN**	**NaN**	**OwnerOccupier**	**Poselei Sosensk**
2	**2**	**2014-02-05**	**83**	**44.0**	**9.0**	**17.0**	**1.0**	**1985.0**	**3.0**	**10.0**	**3.0**	**Investment**	**Krylatsk**
3	**3**	**2012-07-26**	**71**	**49.0**	**2.0**	**NaN**	**NaN**	**NaN**	**NaN**	**NaN**	**NaN**	**Investment**	**Matushki**
4	**4**	**2014-10-29**	**60**	**42.0**	**9.0**	**9.0**	**1.0**	**1970.0**	**3.0**	**6.0**	**2.0**	**Investment**	**Gol'janc**

5 rows × 292 columns

In [6]:

```
X = df.drop(columns='price')
y = df['price']
```

In [7]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
)
X_TRAIN = X_train.copy() # сохраним исходные данные
X_TEST = X_test.copy()
```

Возможно в ваших моделях вам придется указывать, какие колонки являются категориальными (например, в бустингах). Для упрощения предлагается разделить колонки по следующему принципу:

```
drop_columns = [
    'id',           # May leak information
    'timestamp',    # May leak information
]
cat_columns = [
    'product_type',              #
    'material',                  # Material of the wall
    'state',                     # Satisfaction level
    'sub_area',                  # District name
    'culture_objects_top_25',    #
    'thermal_power_plant_raion', #
    'incineration_raion',        #
    'oil_chemistry_raion',       #
    'radiation_raion',           #
    'railroad_terminal_raion',   #
    'big_market_raion',          #
    'nuclear_reactor_raion',     #
    'detention_facility_raion',  #
```

```
        'ID_metro',                 #
        'ID_railroad_station_walk',  #
        'ID_railroad_station_avto',  #
        'water_1line',              #
        'ID_big_road1',             #
        'big_road1_1line',          #
        'ID_big_road2',             #
        'railroad_1line',           #
        'ID_railroad_terminal',     #
        'ID_bus_terminal',          #
        'ecology',                  #
    ]
    num_columns = list(set(df.columns).difference(set(cat_columns + drop_columns)))
```

In [8]:

```
drop_columns = [
    'id',           # May leak information
    'timestamp',    # May leak information
]
cat_columns = [
    'product_type',              #
    'material',                  # Material of the wall
    'state',                     # Satisfaction level
    'sub_area',                  # District name
    'culture_objects_top_25',    #
    'thermal_power_plant_raion', #
    'incineration_raion',        #
    'oil_chemistry_raion',       #
    'radiation_raion',           #
    'railroad_terminal_raion',   #
    'big_market_raion',          #
    'nuclear_reactor_raion',     #
    'detention_facility_raion',  #
    'ID_metro',                  #
    'ID_railroad_station_walk',  #
    'ID_railroad_station_avto',  #
    'water_1line',               #
    'ID_big_road1',              #
    'big_road1_1line',           #
    'ID_big_road2',              #
    'railroad_1line',            #
    'ID_railroad_terminal',      #
    'ID_bus_terminal',           #
    'ecology',                   #
]
num_columns = list(set(X_train.columns).difference(set(cat_columns + drop_columns)))
```

Baseline (2 балла)

В качестве **Baseline** обучите `DecisionTreeRegressor` из `sklearn`.

Preparing dataset: пропуски

In [9]:

```
print(X.shape)
```

```
(20000, 291)
```

In [10]:

```
X.isna().any().sum(axis = 0) #  сколько признаков содержат неизвестные значения
```

Out[10]:

51

In [11]:

```
X.isna().sum().sort_values(ascending=False)[:51]
```

Out[11]:

```
hospital_beds_raion                        9404
state                                      8907
build_year                                 8905
cafe_avg_price_500                         8778
cafe_sum_500_max_price_avg                 8778
cafe_sum_500_min_price_avg                 8778
max_floor                                  6303
material                                   6303
num_room                                   6303
kitch_sq                                   6303
cafe_sum_1000_max_price_avg                4285
cafe_avg_price_1000                        4285
cafe_sum_1000_min_price_avg                4285
preschool_quota                            4279
school_quota                               4277
life_sq                                    4103
build_count_foam                           3175
build_count_wood                           3175
build_count_frame                          3175
build_count_brick                          3175
build_count_monolith                       3175
build_count_panel                          3175
build_count_1971-1995                      3175
build_count_slag                           3175
build_count_mix                            3175
raion_build_count_with_builddate_info      3175
build_count_before_1920                    3175
build_count_1921-1945                      3175
raion_build_count_with_material_info       3175
build_count_after_1995                     3175
build_count_block                          3175
build_count_1946-1970                      3175
cafe_sum_1500_min_price_avg                2769
cafe_sum_1500_max_price_avg                2769
cafe_avg_price_1500                        2769
cafe_avg_price_2000                        1134
cafe_sum_2000_min_price_avg                1134
cafe_sum_2000_max_price_avg                1134
cafe_avg_price_3000                         641
cafe_sum_3000_min_price_avg                 641
cafe_sum_3000_max_price_avg                 641
cafe_sum_5000_min_price_avg                 196
cafe_sum_5000_max_price_avg                 196
cafe_avg_price_5000                         196
prom_part_5000                              118
floor                                       113
metro_min_walk                               16
metro_km_walk                                16
railroad_station_walk_km                     16
railroad_station_walk_min                    16
ID_railroad_station_walk                     16
dtype: int64
```

In [12]:

```
columns_nan = X.isna().sum().sort_values(ascending=False)[:51].index
```

In [13]:

```
cols_nan_num = set(columns_nan) & set(num_columns)
```

In [14]:

```
print(len(cols_nan_num))
```

```
print(len(cols_nan_num)) # пропуски в числовых признаках
```

```
48
```

In [15]:

```
cols_nan_cat = set(columns_nan) & set(cat_columns)
```

In [16]:

```
print(len(cols_nan_cat)) # пропуски в категорилаьных признаках
```

```
3
```

In [17]:

```
X_train_num = X_train[cols_nan_num]
```

In [18]:

```
# разобьем данные на группы и в каждой группе заполним средним все числовые пропуски
kf = KFold(n_splits=5)
for other, idxs in kf.split(X_train_num):
    selection = X_train_num.iloc[idxs] # берем часть данных
    imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean') # стратегия заполнен
ия
    clean = pd.DataFrame(imp_mean.fit_transform(selection), columns = selection.columns)
# запоминаем очищенную выборку
    X_train_num.iloc[idxs] = clean
```

In [19]:

```
X_train[list(cols_nan_num)] = X_train_num # исправляем числовые пропуски в трейне
```

In [20]:

```
X_train.isna().any().sum(axis=0) # осталось три категориальных признака с NaN
```

Out[20]:

```
3
```

In [21]:

```
X_train_cat = X_train[cols_nan_cat]
```

In [22]:

```
kf = KFold(n_splits=5)
for other, idxs in kf.split(X_train_cat): # заполним пропуски для категориальных данных
    selection = X_train_cat.iloc[idxs] # берем часть данных
    imp_mean = SimpleImputer(missing_values=np.nan, strategy='most_frequent') # стратегия
заполнения
    clean = pd.DataFrame(imp_mean.fit_transform(selection), columns = selection.columns)
# запоминаем очищенную выборку
    X_train_cat.iloc[idxs] = clean
```

In [23]:

```
X_train[list(cols_nan_cat)] =  X_train_cat
```

In [24]:

```
X_train.isna().any().sum().sum()
```

Out[24]:

```
0
```

In [25]:

```
# убираем все категориальные признаки
```

```
# убираем все категориальные признаки
Xcat = X_train[cat_columns]
X_train.drop(columns = cat_columns, inplace=True)
```

In [26]:

```
X_cat = pd.get_dummies(Xcat)
print(Xcat.shape) # было
print(X_cat.shape) # one-hot encoding стало
```

```
(16000, 24)
(16000, 186)
```

In [27]:

```
X_train = pd.concat([X_train, X_cat], axis = 1)
X_train.drop(columns = drop_columns, inplace=True)
```

In [28]:

```
print(X_train.shape)
```

```
(16000, 451)
```

Сделаем тоже самое для тестовой выборки. Мемори лики будут отсутствовать.

In [29]:

```
# разобьем данные на группы и в каждой группе заполним средним все числовые пропуски
X_test_num = X_test[cols_nan_num]
kf = KFold(n_splits=5)
for other, idxs in kf.split(X_test_num):
    selection = X_test_num.iloc[idxs] # берем часть данных
    imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean') # стратегия заполнен
ия
    clean = pd.DataFrame(imp_mean.fit_transform(selection), columns = selection.columns)
# запоминаем очищенную выборку
    X_test_num.iloc[idxs] = clean
X_test[list(cols_nan_num)] = X_test_num # исправляем числовые пропуски в тесте

X_test_cat = X_test[cols_nan_cat]
kf = KFold(n_splits=5)
for other, idxs in kf.split(X_test_cat): # заполним пропуски для категориальных данных
    selection = X_test_cat.iloc[idxs] # берем часть данных
    imp_mean = SimpleImputer(missing_values=np.nan, strategy='most_frequent') # стратегия
заполнения
    clean = pd.DataFrame(imp_mean.fit_transform(selection), columns = selection.columns)
# запоминаем очищенную выборку
    X_test_cat.iloc[idxs] = clean
X_test[list(cols_nan_cat)] =  X_test_cat # исправляем числовые пропуски в тесте
print(X_test.isna().any().sum().sum()) # должно остаться ноль пропусков

# убираем все категориальные признаки
Xcat = X_test[cat_columns]
X_test.drop(columns = cat_columns, inplace=True)
X_cat = pd.get_dummies(Xcat)
X_test = pd.concat([X_test, X_cat], axis = 1)

X_test.drop(columns = drop_columns, inplace=True) # убираем ненужные признаки
```

```
0
```

Бейзлайн - линейная регрессия

In [30]:

```
res = set(X_train.columns) - set(X_test.columns)
res # те колонки, которых нет в тестовой выбрке
```

Out[30]:

```
{'sub_area_Molzhaninovskoe',
 'sub_area_Poselenie Kievskij',
 'sub_area_Poselenie Klenovskoe',
 'sub_area_Poselenie Mihajlovo-Jarcevskoe',
 'sub_area_Poselenie Shhapovskoe',
 'sub_area_Poselenie Voronovskoe',
 'sub_area_Vostochnoe'}
```

In [31]:

```
X_test[list(res)] = np.zeros((X_test.shape[0], len(res)))
```

In [32]:

```
res = set(X_test.columns) - set(X_train.columns)
res # те колонки, которых нет в трейне
```

Out[32]:

```
set()
```

In [33]:

```
X_train[list(res)] = np.zeros((X_train.shape[0], len(res)))
```

In [34]:

```
print(X_test.shape) # теперь shape у трейна и теста должен быть одинаковым
print(X_train.shape)
```

```
(4000, 451)
(16000, 451)
```

In [35]:

```
# расположим колонки в тесте в таком же порядке, как в трейне
X_test = X_test.reindex(columns=X_train.columns)
```

In [36]:

```
def RMSE(y_test, y_pred):
    return mean_squared_error(y_test, y_pred, squared=False)
```

In [37]:

```
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred = lin_reg.predict(X_test)
print(RMSE(y_test, y_pred))
```

```
3222702.329998895
```

Обучение **Decision Tree**

In [38]:

```
clf = DecisionTreeRegressor(random_state=42)
clf.fit(X_train, y_train)
```

Out[38]:

```
DecisionTreeRegressor(random_state=42)
```

In [39]:

```
y_pred = clf.predict(X_test)
print(RMSE(y_test, y_pred))
```

```
3919562.231522684
```

Нормализация признаков

In [40]:

```
X_train
```

Out[40]:

	full_sq	life_sq	floor	max_floor	build_year	num_room	kitch_sq	area_m	raion_popul	green_zone_part	i
5894	96	96.000000	3.0	12.759690	1869.313165	1.914273	6.865937	7.307411e+06	75377	0.065444	
3728	64	33.000000	2.0	17.000000	2014.000000	2.000000	12.000000	3.879802e+06	81980	0.157332	
8958	73	34.277756	17.0	17.000000	1869.313165	2.000000	1.000000	1.139168e+07	19940	0.055644	
7671	60	34.277756	4.0	18.000000	1.000000	2.000000	1.000000	1.084231e+07	85219	0.062172	
5999	40	34.277756	9.0	12.759690	1869.313165	1.914273	6.865937	1.139168e+07	19940	0.055644	
...	...	...	...	...	...	...	...	...	...	...	
11284	46	28.000000	4.0	12.000000	1972.000000	2.000000	9.000000	8.087656e+06	116742	0.048011	
11964	46	46.000000	10.0	12.429025	1879.667040	1.883447	5.908844	6.677245e+07	9553	0.336177	
5390	72	72.000000	2.0	25.000000	2015.000000	2.000000	1.000000	1.163805e+07	123280	0.068202	
860	82	34.137147	8.0	12.429025	1879.667040	1.883447	5.908844	6.677245e+07	9553	0.336177	
15795	38	19.000000	11.0	17.000000	1987.000000	1.000000	8.000000	1.216448e+07	78507	0.297166	

16000 rows × 451 columns

In [41]:

```
std = StandardScaler()
std.fit(X_train[num_columns])
X_train_norm = X_train.copy()
X_train_norm[num_columns] = std.transform(X_train[num_columns])
```

In [42]:

```
X_train_norm = pd.DataFrame(X_train_norm)
X_train_norm
```

Out[42]:

	full_sq	life_sq	floor	max_floor	build_year	num_room	kitch_sq	area_m	raion_popul	green_zone_part	ind
5894	0.876474	3.303707	-0.881160	0.023950	-0.014701	0.013531	0.010672	-0.493396	-0.157809	-0.875787	-(
3728	0.204198	-0.063847	-1.070905	0.786055	-0.013789	0.137529	0.193906	-0.658102	-0.043610	-0.353380	(
8958	0.393276	0.004453	1.775269	0.786055	-0.014701	0.137529	-0.198682	-0.297134	-1.116587	-0.931507	1
7671	0.120164	0.004453	-0.691415	0.965783	-0.026486	0.137529	-0.198682	-0.323533	0.012408	-0.894389	(
5999	-0.300008	0.004453	0.257310	0.023950	-0.014701	0.013531	0.010672	-0.297134	-1.116587	-0.931507	1
...	...	...	...	...	...	...	...	...	...	...	
11284	-0.173956	-0.331113	-0.691415	-0.112588	-0.014054	0.137529	0.086837	-0.455902	0.557596	-0.974901	-(
11964	-0.173956	0.631045	0.447055	-0.035480	-0.014636	-0.031056	-0.023486	2.364074	-1.296230	0.663406	-(
5390	0.372267	2.020829	-2.223883	-0.013782	0.137529	-0.670670	-0.860108	-(			

	full_sq	life_sq	floor	max_floor	build_year	num_room	kitch_sq	area_m	raion_popul	green_zone_part	ind
860	0.582353	-0.003063	0.067565	-0.035480	-0.014636	-0.031056	-0.023486	2.364074	-1.296230	0.663406	-(
15795	-0.342025	-0.812192	0.636800	0.786055	-0.013959	-1.308894	0.051147	-0.259999	-0.103676	0.441620	-(

16000 rows × 451 columns

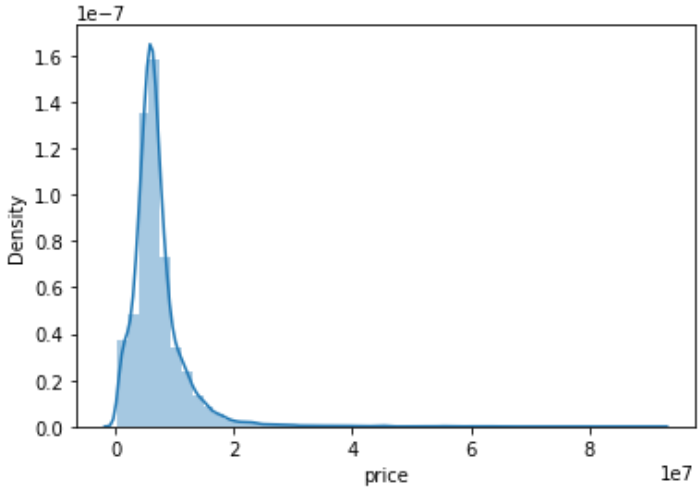
In [43]:

```
X_test_norm = X_test.copy()
X_test_norm[num_columns] = std.transform(X_test[num_columns]) # нормализуем тестовую выборку
X_test_norm = pd.DataFrame(X_test_norm)
```

Распределение таргета

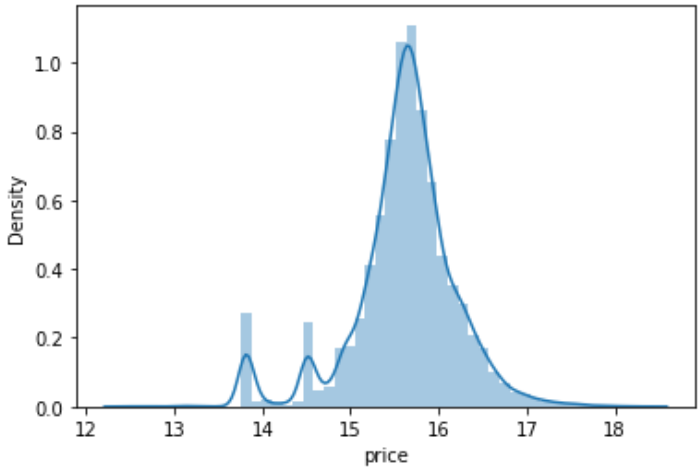
In [44]:

```
sns.distplot(y_train); # длинный хвост
```



In [45]:

```
sns.distplot(np.log(y_train));
```



In [46]:

```
y_test_norm = np.log(y_test)
y_train_norm = np.log(y_train)
```

In [47]:

```
print(y_train_norm.max())
```

16.397061305017715

18.327096130501715

In [48]:

```
#  значение моделей на нормализованных данных:
lin_reg = LinearRegression()
lin_reg.fit(X_train_norm, y_train)
y_pred = lin_reg.predict(X_test_norm)
print(f'Линейная регрессия на нормализованных данных: {RMSE(y_test, y_pred)}') # значение немного ухудшилось
```

```
Линейная регрессия на нормализованных данных: 3229498.2876013517
```

In [49]:

```
clf = DecisionTreeRegressor(random_state=42)
clf.fit(X_train_norm, y_train)
y_pred = clf.predict(X_test_norm)
print(f'Дерево решений на нормализованных данных: {RMSE(y_test, y_pred)}') # значение ухудшилось, значит нормализуя данные - мы теряем информацию
```

```
Дерево решений на нормализованных данных: 3919655.0854102164
```

Отбор признаков с помощью **L1** регуляризации

In [50]:

```
history = []
cs = np.linspace(0.001, 1200, 7)
for c in tqdm(cs):
    log_reg = Lasso(alpha=c)
    log_reg.fit(X_train_norm, y_train)
    y_pred = log_reg.predict(X_test_norm)
    history.append(mean_squared_error(y_test, y_pred, squared=False))
```

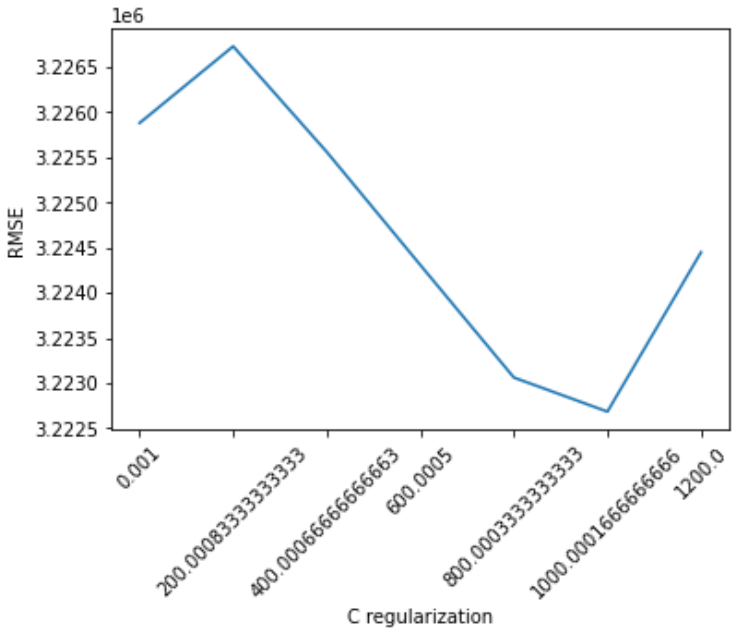
```
100%|██████████| 7/7 [01:53<00:00, 16.26s/it]
```

In [51]:

```
plt.plot(np.arange(len(history)), history)
plt.xticks(np.arange(len(history)), cs, rotation=45)
plt.xlabel('C regularization')
plt.ylabel('RMSE')
```

Out[51]:

```
Text(0, 0.5, 'RMSE')
```



In [52]:

```
idx = np.argmin(history)
print(f'лучший скор: {history[idx]}')
print(f'C = {cs[idx]}')
```

```
лучший скор: 3222683.3005640176
C = 1000.0001666666666
```

In [53]:

```
c = cs[idx]
lin_reg = Lasso(alpha=c)
lin_reg.fit(X_train_norm, y_train)
```

Out[53]:

```
Lasso(alpha=1000.0001666666666)
```

In [54]:

```
coefs = lin_reg.coef_
```

In [55]:

```
print(sorted(coefs, key=lambda x: abs(x))) # видим что L1 регуляризация занулила огромное кол-во коеффициентов
```

```
[0.0, 0.0, -0.0, 0.0, 0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 0.0, 0.0, 0.0, -0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, -0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.0, 0.0, -0.0, -0.0, 0.0,
-0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, -0.0, 0.0, -0.0, -0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, -0.0, -0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, 0.0, 0.0, -0.0, 0.0, 0.0, -0.0, -0.0, -0
.0, 0.0, -0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, -0.0, -0.0, -0.0, 0.0, -0.0, -0.0, -
0.0, 0.0, 0.0, -0.0, 0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 0.0, -0.0, 0.0, -0.0, -0.0,
0.0, -0.0, 0.0, -0.0, -0.0, -0.0, 0.0, 0.0, -0.0, -0.0, 0.0, 0.0, -0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, -0.0, -0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.0, -0.0, 0.0, 0.0, -0.0, 0.0, -0.0,
-0.0, 0.0, -0.0, -0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, 0.0, -0.0, -0.0, -0.0, 0.0, -0.0,
-0.0, -0.0, 0.0, 0.0, -0.0, 0.0, -0.0, -0.0, -0.0, 3.248979333650869e-11, -5.388208986745
37e-11, 5.665688113831023e-11, -1.7729822477560606e-10, 3.4505070250776887e-10, 4.6004095
7146179e-10, 4.4220054059882376e-08, 1.090040468846951e-07, -2.0353716317800923e-07, 4.70
8680638577788e-05, 740.310851729834, 1021.1571705426471, 1120.3468200136372, 1228.0707440
170074, -1879.0730245704362, 2172.9334271959315, -2341.545470757709, 2944.6709302458103,
3263.7884075659326, -4962.5827286168, 6031.915902712775, 6436.577838304006, -6547.7584790
091605, -7382.047279651775, 7731.193456809204, 7827.005938555373, 8686.087197221319, 9354
.903987807103, -9743.319977272256, 9852.81168896097, -14928.117156248889, 15393.234451514
058, 16613.01684501812, 18312.919294061234, 20325.459490806392, -21144.664141256348, -215
44.28214968585, -23780.477870187733, 25959.32268783659, 26733.7888154035, -26928.97531909
0736, 28552.069239334043, -30093.88758699063, -30274.380704141793, -30551.27279879667, -3
1858.737322432455, -31879.224954081867, -32292.235544811367, -33391.250145673424, -33553.
4044195188, 34256.45602569641, 35644.184932961805, -35788.63420503195, -36780.13348567867
, -37140.21856158515, 39352.88842657382, 39403.17180807036, 39898.244586443696, 40909.164
4727261, -43715.51193048809, -44738.803552700665, -46622.50653816041, 47620.89506097069,
47896.40348122064, -48872.559920125124, -49583.4493697894, 49888.52553047616, 53107.57564
680037, 53302.93657700867, -56745.540028132105, -59540.879730059365, 64821.5421826028, 65
056.73169244892, 66011.47480250834, -66327.68490557552, -68859.61518228683, -69346.030470
76882, 69704.04702287354, -70882.31190395483, -73141.15554135741, 78006.8849637887, -7880
6.29044981698, -79780.06595001572, -80187.74434141719, -81963.10370942247, -83021.8712506
621, -93265.17946924998, 94061.48624454951, -94087.67177921429, -95450.92065453493, 95885
.34173531909, 98427.51313361716, 99931.17264535905, 100861.67991077463, -102119.792562538
63, 102461.44470362019, 102641.15555752642, -103109.77145703186, 106258.50896554308, -108
294.14147864857, 109471.11862301339, 110385.4070011263, -111144.08229909277, -111539.2508
4602574, -113498.29612981025, -115755.24741447356, -117689.13998588324, 118001.4943893718
7, -119002.67405521985, -124932.25547600086, -127803.01511808428, 130139.18528502066, 130
734.4214178559, 131603.8876275834, -133278.3086658134, 134568.92145977434, 134710.8373444
9482, 136150.65949873382, 137077.28784996588, -141679.46876829918, 142840.86709986624, -1
44871.65331266512, 145563.22891620977, 151421.01069165205, 153052.7548082325, 153072.3559
0953383, -157391.86851862358, -157830.18955668824, -161814.54371824674, 162757.3986503965
, 162870.81485733978, 166847.79632749103, 167503.65611975294, 170057.92273967457, -177616
.45198967753, -180973.3434311007, 181562.03048739926, 182446.2022563708, -184618.30555698
01, -187706.81616041288, -188088.082010424, -193780.5105001727, -195472.20425814067, -200
145.81241041675, -201531.39339488832, 201579.6526440923, -205780.65543006518, -209889.078
01302287, -212251.39767040362, 213116.8350369348, 213287.3039754787, 221634.37985543328,
-221969.7202597324, -228542.3525604151, 230017.47848727627, -230721.20496814518, 231796.5
32608297, 234711.87864163995, -242264.73221282763, -244460.32780208197, 245680.8905527373
```

```
4, -246935.43555852753, 248236.0540225906, 250679.33058463238, -250977.43878599204, -2528
22.4710262492, 265776.76470362773, 268433.14529845223, 271878.3631061366, -272989.3456632
0915, -279551.0132702829, 281351.4766145951, 281624.5105322086, -284771.4203107987, -2865
41.9891231097, 288654.3437381549, -289776.8201264437, -301183.4631689288, 303200.59067831
485, 303499.2433312109, -310250.0675579825, -311692.5376521749, -312092.53036261554, -318
464.89171534876, -321804.417080138, 325067.16041038284, 326600.395936322, 331449.76894432
12, 332197.63733858644, -340955.5315538161, -350996.1974706448, 351190.57838128076, 35150
1.5369136472, -351645.3444680572, 352697.04161324835, 355750.6507150374, -362770.02979234
03, 369984.8075791187, 370105.704603643, -380527.150411613, -385336.4425306734, -389547.3
6987970746, -399192.7017630508, -405522.56042815454, 406806.821611788, -407154.7386032761
4, 407359.6740594252, 410126.99348010926, -419746.1981568255, 433204.5028505175, 439472.5
99594639, -446585.3270740354, 460090.9973842468, 471841.0544268299, 480809.25981752516, 4
82671.8386523433, 485161.2009322616, 504352.5346703483, -511565.32875928504, 518734.94970
956014, -519833.87004869763, -522957.088448294, 523581.3467134718, 532233.5967806064, -54
3194.2872138413, -570431.1324324759, -573747.4250416622, -577742.4311142784, 584020.76599
70385, 609886.4717414035, -615000.7049529392, 617755.1654880536, -621000.9691577187, -626
307.7612234139, 627118.3717450985, -631342.7693317209, 633786.8055803347, 643041.45365054
04, -645710.859905615, 659278.1374381335, 689697.9309345831, 693064.851390167, -704991.62
4141999, 710447.6030498138, -731528.9589919316, -733861.7196857082, -739601.3266365335, -
742994.1244069815, 758616.4617693351, 769433.2127472993, 787347.8976946939, 796055.858276
3053, 804806.364247307, 812799.1204897747, -826193.317215243, -835437.6198454965, 836377.
5522814358, -849403.9800754256, 866196.7998300134, 886839.251670158, 889975.8153955742, -
914088.303914573, 916726.1097646871, -936390.0287008589, -987649.1359110117, 1001175.7998
067228, -1013093.5781166552, 1015140.0912630019, 1063307.1675239196, 1066753.6248840406,
1067312.9346303833, -1119395.1161488807, -1125685.8843750318, -1132521.2799593103, -11702
78.927898034, -1241832.0559948043, -1316653.7628335077, -1355908.141608875, -1405224.9989
804612, -1411204.7376052313, 1423438.3135065867, 1446201.358178395, 1522293.5681722416, -
1600765.6490306412, 1653330.8775730678, 1656818.9296604707, 1710019.9048218552, 1734633.0
233904414, 1748452.1895653796, 1767860.7159132964, 1819416.7587529637, 1922963.6444428263
, -2057986.4965950302, 2792028.771252112, -2960147.84637206, 3095654.8531363676]
```

In [56]:

```
coefs_pd = pd.DataFrame({'w' : coefs, 'abs_w' : np.abs(coefs)}) # отсортируем коэффициен
ты по возрастанию их модуля
coefs_pd.sort_values(by=['abs_w'])
```

Out[56]:

	w	abs_w
450	-0.000000e+00	0.000000e+00
233	0.000000e+00	0.000000e+00
413	0.000000e+00	0.000000e+00
412	0.000000e+00	0.000000e+00
411	-0.000000e+00	0.000000e+00
...	...	...
358	1.922964e+06	1.922964e+06
188	-2.057986e+06	2.057986e+06
310	2.792029e+06	2.792029e+06
339	-2.960148e+06	2.960148e+06
327	3.095655e+06	3.095655e+06

451 rows × 2 columns

In [57]:

```
print(len(coefs_pd[coefs_pd['abs_w'] < 1])) # ненужные признаки
```

```
165
```

In [58]:

```
idxs = coefs_pd[coefs_pd['abs_w'] < 1].index
```

In [59]:

```
bad_columns = X_train.columns[idxs]
X_train_r = X_train.drop(columns=bad_columns) # reduced X_train
X_test_r = X_test.drop(columns = bad_columns)
```

In [60]:

```
print(X_train_r.shape)
print(X_test_r.shape)
```

```
(16000, 286)
(4000, 286)
```

In [61]:

```
lin_reg = LinearRegression()
lin_reg.fit(X_train_r, y_train)
y_pred = lin_reg.predict(X_test_r)
print(f'Линеная регрессия: с отбором признаков {RMSE(y_test, y_pred)}')
```

```
Линеная регрессия: с отбором признаков 3232438.1395197045
```

In [62]:

```
clf = DecisionTreeRegressor(random_state=42)
clf.fit(X_train_r, y_train)
y_pred = clf.predict(X_test_r)
print(f'Дерево решений с отбором признаков: {RMSE(y_test, y_pred)}')
```

```
Дерево решений с отбором признаков: 3755930.01382082
```

Проверьте качество на отложенной выборке.

Подберем признаки для решающего дерева

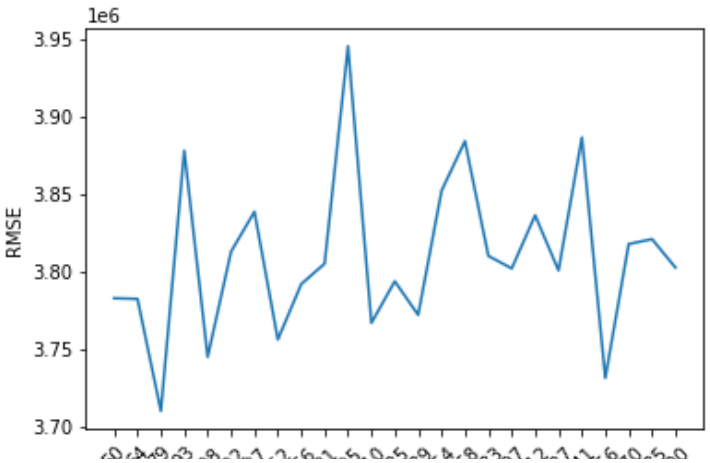
In [63]:

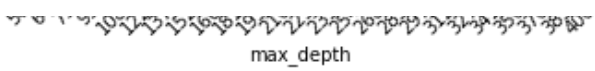
```
depths = np.linspace(50, 400, 25, dtype=int)
hist1 = []
for d in tqdm(depths): # подбор параметра max_depth
    clf = DecisionTreeRegressor(max_depth=d)
    clf.fit(X_train_r, y_train)
    y_pred = clf.predict(X_test_r)
    hist1.append(RMSE(y_test, y_pred))
```

```
100%|██████████| 25/25 [01:09<00:00,  2.78s/it]
```

In [64]:

```
plt.plot(np.arange(len(hist1)), hist1)
plt.xlabel('max_depth')
plt.ylabel('RMSE')
plt.xticks(np.arange(len(hist1)), depths, rotation=45);
```





In [65]:

```
idx = 5
print(hist1[idx])
print(depths[idx])
```

```
3813002.3975572465
122
```

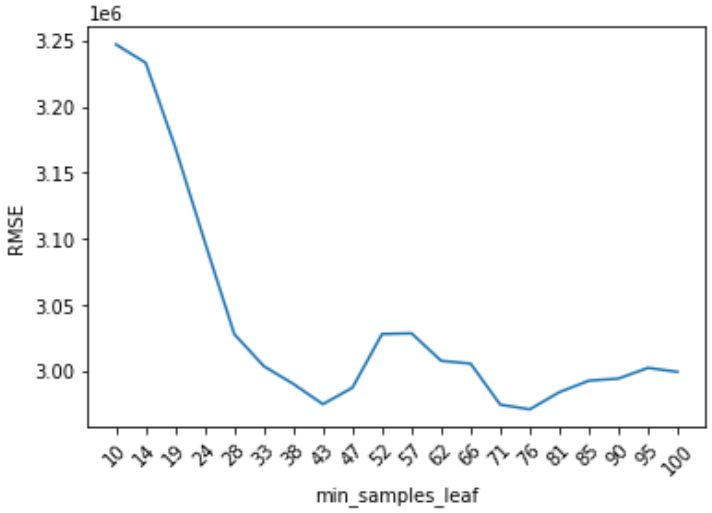
In [66]:

```
leafs = np.linspace(10, 100, 20, dtype=int)
hist2 = []
for leaf in tqdm(leafs): # подбор параметра min_samples_leaf
    clf = DecisionTreeRegressor(min_samples_leaf=leaf)
    clf.fit(X_train_r, y_train)
    y_pred = clf.predict(X_test_r)
    hist2.append(RMSE(y_test, y_pred))
```

```
100%|██████████| 20/20 [00:28<00:00,  1.44s/it]
```

In [67]:

```
plt.plot(np.arange(len(leafs)), hist2)
plt.xlabel('min_samples_leaf')
plt.ylabel('RMSE')
plt.xticks(np.arange(len(hist2)), leafs, rotation=45);
```



In [68]:

```
idx = np.argmin(hist2)
print(hist2[idx])
print(leafs[idx])
```

```
2971821.3080791165
76
```

In [69]:

```
clf = DecisionTreeRegressor(max_depth=79, min_samples_leaf=76)
clf.fit(X_train_r, y_train)
y_pred = clf.predict(X_test_r)
print(RMSE(y_test, y_pred))
```

```
2971821.3080791165
```

Feature Engineering (2 балла)

Часто улучшить модель можно с помощью аккуратного **Feature Engineering.**

Добавим в модель дополнительные признаки:

- "Как часто в этот год и этот месяц появлились объявления"
- "Как часто в этот год и эту неделю появлялись объявления"

In [70]:

```
month_year = (df.timestamp.dt.month + df.timestamp.dt.year * 100) # хешируем год + месяц
month_year_cnt_map = month_year.value_counts().to_dict()
df["month_year_cnt"] = month_year.map(month_year_cnt_map)

week_year = (df.timestamp.dt.weekofyear + df.timestamp.dt.year * 100)
week_year_cnt_map = week_year.value_counts().to_dict()
df["week_year_cnt"] = week_year.map(week_year_cnt_map)
```

Добавьте следующие дополнительные признаки:

- Месяц (из колонки `timestamp`)
- День недели (из колонки `timestamp`)
- Отношение "этаж / максимальный этаж в здании" (колонки `floor` и `max_floor`)
- Отношение "площадь кухни / площадь квартиры" (колонки `kitch_sq` и `full_sq`)

По желанию можно добавить и другие признаки.

In [71]:

```
df['month'] = df.timestamp.dt.month
df['weekday'] = df.timestamp.dt.weekday
df['height%'] = df['floor'] / df['max_floor']
df['kitchen%'] = df['kitch_sq'] / df['full_sq']
```

In [72]:

```
feature_cols = ['month_year_cnt', 'week_year_cnt', 'month', 'weekday', 'height%', 'kitch
en%']
df[feature_cols].isna().sum()
```

Out[72]:

```
month_year_cnt       0
week_year_cnt        0
month                0
weekday              0
height%           6305
kitchen%          6304
dtype: int64
```

In [73]:

```
X_train_r.index
```

Out[73]:

```
Int64Index([ 5894,  3728,  8958,  7671,  5999,  5751,  1688,  6836,  6536,
             4842,
            ...
            11363, 14423,  4426, 16850,  6265, 11284, 11964,  5390,   860,
            15795],
           dtype='int64', length=16000)
```

Разделите выборку на обучающую и тестовую еще раз (потому что дополнительные признаки созданы для исходной выборки).

In [74]:

```
eps = 1e-5
```

In [75]:

```
X_train_r['max_floor'] += eps # чтобы не делить на нули
X_train_r['full_sq'] += eps
X_test_r['max_floor'] += eps
X_test_r['full_sq'] += eps
```

In [76]:

```
X_train_r['height%'] = np.clip(X_train_r['floor'] / X_train_r['max_floor'], 0, 1) # прео
бразуем трейн и тест отдельно, чтобы не было мемори ликов
X_train_r['kitchen%'] = np.clip(X_train_r['kitch_sq'] / X_train_r['full_sq'], 0, 1)
X_train_r['month'] = df.loc[X_train_r.index]['month']
X_train_r['weekday'] = df.loc[X_train_r.index]['weekday']

X_test_r['height%'] = np.clip(X_test_r['floor'] / X_test_r['max_floor'], 0, 1)
X_test_r['kitchen%'] = np.clip(X_test_r['kitch_sq'] / X_test_r['full_sq'], 0, 1)
X_test_r['month'] = df.loc[X_test_r.index]['month']
X_test_r['weekday'] = df.loc[X_test_r.index]['weekday']
```

Model Selection (3 балла)

Посмотрите, какого качества можно добиться если использовать разные модели:

- `DecisionTreeRegressor` из `sklearn`
- `RandomForestRegressor` из `sklearn`
- `CatBoostRegressor`

Также вы можете попробовать линейные модели, другие бустинги (`LigthGBM` и `XGBoost`).

Почти все библиотеки поддерживают удобный способ подбора гиперпараметров: посмотрите как это делать в **sklearn** или в **catboost**.

Проверяйте качество каждой модели на тестовой выборке и выберите наилучшую.

In [77]:

```
lin_reg = LinearRegression() # Линейная регрессия
lin_reg.fit(X_train_r, y_train)
y_pred = lin_reg.predict(X_test_r)
print(f'Линеная регрессия: с отбором признаков и feature инжинирингом {RMSE(y_test, y_pre
d)}') # стало чуть лучше
```

```
Линеная регрессия: с отбором признаков и feature инжинирингом 3229098.0433697538
```

In [78]:

```
clf = DecisionTreeRegressor(min_samples_leaf=76) # Дерево решений
clf.fit(X_train_r, y_train)
y_pred = clf.predict(X_test_r)
print(f'Дерево решений с отбором признаков и feature инжинирингом: {RMSE(y_test, y_pred)}
')
```

```
Дерево решений с отбором признаков и feature инжинирингом: 2976903.9330579336
```

In [79]:

```
clf = RandomForestRegressor(n_estimators=100, max_features = X_train.shape[1]//3, max_sa
mples=0.5)
clf.fit(X_train_r, y_train)
y_pred = clf.predict(X_test_r)
print(f' Случайный лес с отбором признаков и feature инжиннирингом: {RMSE(y_test, y_pred)
}')
```

```
 Случайный лес с отбором признаков и feature инжиннирингом: 2622298.2445838535
```

In [82]:

```
cat = CatBoostRegressor(verbose=0, n_estimators = 500)
cat.fit(X_train_r, y_train)
```

```
y_pred = cat.predict(X_test_r)
print(f'Градиентный бустинг с отбором признаков и feature инжиниринг: {RMSE(y_test, y_pred)}')
```

```
Градиентный бустинг с отбором признаков и feature инжиниринг: 2528349.0040742285
```

In [86]:

```
xgb = xgboost.XGBRegressor(n_estimators = 500, objective ='reg:squarederror')
xgb.fit(X_train_r, y_train)
y_pred = xgb.predict(X_test_r)
print(f'Другой градиентный бустинг с отбором признаков и feature инжиниринг: {RMSE(y_test, y_pred)}')
```

```
Другой градиентный бустинг с отбором признаков и feature инжиниринг: 2659004.717902067
```

In [96]:

```
lgbm = lightgbm.LGBMRegressor(n_estimators=500)
lgbm.fit(X_train_r, y_train)
y_pred = lgbm.predict(X_test_r)
print(f'lightGBM градиентный бустинг с отбором признаков и feature инжиниринг: {RMSE(y_test, y_pred)}')
```

```
lightGBM градиентный бустинг с отбором признаков и feature инжиниринг: 2601250.7611257746
```

Ensemble v.1 (3 балла)

Ансамбли иногда оказываются лучше чем одна большая модель.

В колонке `product_type` содержится информация о том, каким является объявление: `Investment` (продажа квартиры как инвестиции) или `OwnerOccupier` (продажа квартиры для жилья). Логично предположить, что если сделать по модели на каждый из этих типов, то качество будет выше.

Обучите свои лучшие модели на отдельно на `Investment` и `OwnerOccupier` (т.е. у вас будет `model_invest`, обученная на `(invest_train_X, invest_train_Y)` и `model_owner`, обученная на `(owner_train_X, owner_train_Y)`) и проверьте качество на отложенной выборке (т.е. на исходном `test_split`).

In [102]:

```
mask = np.char.startswith(list(X_train_r.columns),'product_type')
print(X_train_r.columns[mask])
```

```
Index(['product_type_Investment'], dtype='object')
```

In [124]:

```
X_train_inv = X_train_r[X_train_r['product_type_Investment'] == 1] # разделение всех выборок
y_train_inv = y_train.loc[X_train_inv.index]
X_train_own = X_train_r[X_train_r['product_type_Investment'] == 0]
y_train_own = y_train.loc[X_train_own.index]

X_test_inv = X_test_r[X_test_r['product_type_Investment'] == 1]
X_test_own = X_test_r[X_test_r['product_type_Investment'] == 0]
```

In [125]:

```
cat_inv = CatBoostRegressor(verbose=0, n_estimators = 500) # обучение моделей
cat_inv.fit(X_train_inv, y_train_inv)
cat_own = CatBoostRegressor(verbose=0, n_estimators = 500)
cat_own.fit(X_train_own, y_train_own)
```

Out[125]:

```
<catboost.core.CatBoostRegressor at 0x7fcb4de26150>
```

In [136]:

```
y_inv_idxs = np.where(X_test_r['product_type_Investment'] == 1) # индексы
y_own_idxs = np.where(X_test_r['product_type_Investment'] == 0)
```

In [137]:

```
y_inv = cat_inv.predict(X_test_inv)  # ответы для индексов
y_own = cat_own.predict(X_test_own)
```

In [142]:

```
y_pred = np.zeros_like(y_test) # единый массив ответов
y_pred[y_inv_idxs] = y_inv
y_pred[y_own_idxs] = y_own
```

In [143]:

```
print(f'Ансамбль из двух catboost для разных типов жилья RMSE = {RMSE(y_test, y_pred)}')
```

```
Ансамбль из двух catboost для разных типов жилья RMSE = 2491412.515572773
```

In [148]:

```
print(f'средняя абсолютная ошибка: {mean_absolute_error(y_test, y_pred)}')
print(f'средняя цена: {np.mean(y_test)}')
```

```
средняя абсолютная ошибка: 1392836.81275
средняя цена: 7140915.0145
```

(*) Ensemble v.2 (дополнительно, 2 балла)

Попробуйте сделать для `Investment` более сложную модель: обучите `CatBoostRegressor` и `HuberRegressor` из `sklearn`, а затем сложите их предсказания с весами `w_1` и `w_2` (выберите веса сами; сумма весов равняется 1).

In [153]:

```
y_inv_test = np.array(y_test)[y_inv_idxs] # истинные ответы для двух типов жилья
y_own_test = np.array(y_test)[y_own_idxs]
```

In [154]:

```
print(f'Investment предсказания RMSE = {RMSE(y_inv_test, y_inv)}')
print(f'Owner предсказания RMSE = {RMSE(y_own_test, y_own)}')
```

```
Investment предсказания RMSE = 2986175.0463391463
Owner предсказания RMSE = 1091165.1002023178
```

In [157]:

```
cat = CatBoostRegressor(n_estimators=500, verbose=0)
cat.fit(X_train_inv, y_train_inv)
y_inv_cat = cat.predict(X_test_inv)
```

```
Custom logger is already specified. Specify more than one logger at same time is not thre
ad safe.
```

In [170]:

```
hub_reg = HuberRegressor()
hub_reg.fit(X_train_inv, y_train_inv)
y_inv_hub = hub_reg.predict(X_test_inv)
```

In [171]:

```
print(RMSE(y_inv_test, y_inv_hub))
```

```
4865521.848348602
```

In [178]:

```
ws = np.linspace(0, 1, 100)
h = []
for w in tqdm(ws):
    y_pred = y_inv_cat * w + y_inv_hub * (1 - w)
    h.append(RMSE(y_inv_test, y_pred))
```

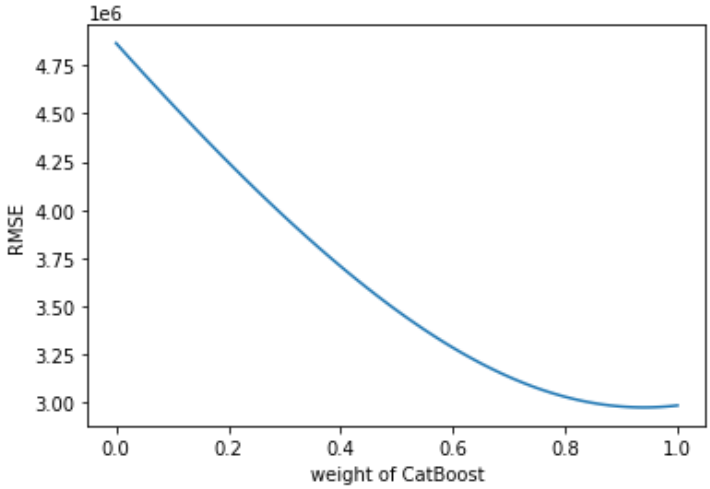
```
100%|██████████| 100/100 [00:00<00:00, 1673.12it/s]
```

In [182]:

```
plt.plot(ws, h)
plt.xlabel('weight of CatBoost')
plt.ylabel('RMSE')
```

Out[182]:

```
Text(0, 0.5, 'RMSE')
```



Более сложная модель, не стала лучше, чем просто **catboost**

Выводы:

Лучший скор дал ансамбль из двух **catboost** отдельно для двух типов объявлений: `Investment` и `OwnerOccupier`. Также мы произвели отбор признаков с помощью **L1** линейной регрессии, очистили данные от пропусков с помощью техники **KFold**, и покрафтили фичи.

Итоговый RMSE = 2491412.5