KARPOV.COURSES >>>
КОНСПЕКТ

# Конспект > 9 урок > Housing market: практика

## >Оглавление


>Оглавление
>**Разведочный анализ данных**
    **Описание данных**
>Обработка вещественных признаков
>Обработка категориальных признаков
    One Hot Encoding
    Mean Target Encoding
>Работа с датой/временем
>Построение базовой модели. Валидация.
>Стандартизация и масштабирование признаков
>Анализ выбросов
>Сегментация данных


## >Разведочный анализ данных

**Разведочный анализ данных** (EDA - exploratory data analysis) — первичное исследование данных, нахождение общих закономерностей, поиск аномалий, построение базовых моделей.

### Описание данных

Разберем кейс реального соревнования с kaggle - Sberbank Russian Housing Market по предсказанию стоимости жилья.

Датасет, включающий тренировочный сет с информацией о более чем 21 тысяче транзакций по продаже недвижимости в период с августа 2011 по июнь 2015 и тестовый сет с более 7 тысячей записей о транзакциях, а также файл с макроэкономическими показателями за соответствующий период.

Всего в датафрейме `284` признака, перечислим лишь часть из них:

`timestamp` -время совершения сделки

`full_sq` -общая площадь

`life_sq`-жилая площадь

`floor` - этаж

`max_floor` - количество этажей в доме

`material` - материал

`build_year` - год постройки

`...`

Для начала определимся с таргетом и функционалом качества, который будем использовать для оценки модели.

Если мы посмотрим на распределение таргетной переменной- она принимает значения порядка нескольких миллионов, в этом случае есть смысл ее логарифмировать и после считать уже не MSE, а MSLE.

Как уже помним из предыдущих лекций, для того чтобы посчитать MSLE достаточно прологорифмировать таргетную переменную.

Для логарифмирования используем функцию `log1p()` библиотеки numpy:

```
import numpy as np
df = df.assign(log_price_doc=np.log1p(df['price_doc']))
df = df.drop('price_doc', axis=1)
```

# >Обработка вещественных признаков

Для начала, познакомимся с данными. Для этого вопользуемся функцией `describe()`, которая позволяет посмотреть базовые статистические характеристики по каждой колонке(признаку): `count` - количество присутствующих значений (не NaN), `min` и `mean` - его минимальные и средние значения, `std` - стандартное отклонение, а также 25%, 50% , 75% квантили, `max`- максимальное значение по каждому признаку.

Подробнее о методе `describe()` в документации

```
df.describe() #крайне полезная функция, позволяет посмотреть статистики по всему датафрейм
у
```

Отберем все вещественные признаки:

```
pynumeric_columns = df.loc[:,df.dtypes!=np.object].columns
```

Работа с пропусками:

Существует множество вариантов работы с пропущенными значениями(NaN). Воспользуемся одним из простых способов - заполненим пропуски средними значениями по каждому признаку:

```
for col in numeric_columns:
    df[col] = df[col].fillna(df[col].mean())
```

Поиск и избавление от коррелирующих признаков:

Как мы помним наличие коррелирующих признаков в датасете приводит к избыточности информации и, как следствие, часто к переобучению. Для выявления корреляции (линейной зависимости между признаками) используется функция `corr()`:

```
df[numeric_columns].corr()
```

Подробнее о `corr()` в документации

Однако такая функция не всегда удобна в применении, поскольку признаков может быть очень много.
На этот случай у нас есть “секретный код” со Stack Overflow:

```
def get_redundant_pairs(df):
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop

def get_top_abs_correlations(df, n=5):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
    return au_corr[0:n]

print("Top Absolute Correlations")
print(get_top_abs_correlations(df[numeric_columns], 50)) #выведем топ 50 коррелирующих пар
```

Следующий код удалит все колонки, где корреляция выше 0.9:

```
def correlation(dataset, threshold):
    col_corr = set() # Set of all the names of deleted columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if (corr_matrix.iloc[i, j] >= threshold) and (corr_matrix.columns[j] not in co
l_corr):
                colname = corr_matrix.columns[i] # getting the name of column
                col_corr.add(colname)
                if colname in dataset.columns:
                    del dataset[colname] # deleting the column from the dataset

correlation(df, 0.9)
```

Теперь, когда мы избавились от веществеых признаков, избавимся также признаков, являющихся константными (квазиконстантными), то есть не меняющимися от объекта к объекту.

Для этого будем использовать функцию `VarianceThreshold` из библиотеки sklearn. В параметры передадим `threshold` =0.1, что отфильтрует все признаки с

коэффициентом дисперсии (отклонением значений) меньше заданного параметра.

```
from sklearn.feature_selection import VarianceThreshold

cutter = VarianceThreshold(threshold=0.1)
cutter.fit(df[numeric_columns])
constant_cols = [x for x in numeric_columns if x not in cutter.get_feature_names_out()]

df[constant_cols]
```

! Данный метод следует применять ориентируясь на масштаб величин, иначе есть риск потерять значимые признаки.

# >Обработка категориальных признаков

Теперь отберем категориальные признаки по аналогии с тем, как делали ранее:

```
categorical_columns = df.loc[:,df.dtypes==np.object].columns

df.describe(include='object')
```

Используем функцию `describe()` и посмотрим на характеристики признаков, для категориальных они будут отличаться:

`count` - количество непропущенных значений, `unique`- количество уникальных значений признака, `top` - самый часто встречаемый признак, `freq` - число объектов с этим признаком в выборке.

## One Hot Encoding

One Hot Encoding - метод позволяющий преобразовать каждое значение признака в отдельный признак, где каждому объекту будет соответствовать 0 или 1, в зависимости от того, обладает объект этим признаком или нет.

| id | color |
|---|---|
| 1 | red |
| 2 | blue |
| 3 | green |
| 4 | blue |


| id | color_red | color_blue | color_green |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 |

*Ссылка на источник:https://towardsdatascience.com/building-a-one-hot-encoding-layer-with-tensorflow-f907d686bf39*

!В конце преобразования признаков не забываем выкинуть одну колонку, чтобы избежать избыточной информации и переобучения (если было 12 значений признака, оставляем 11 колонок).

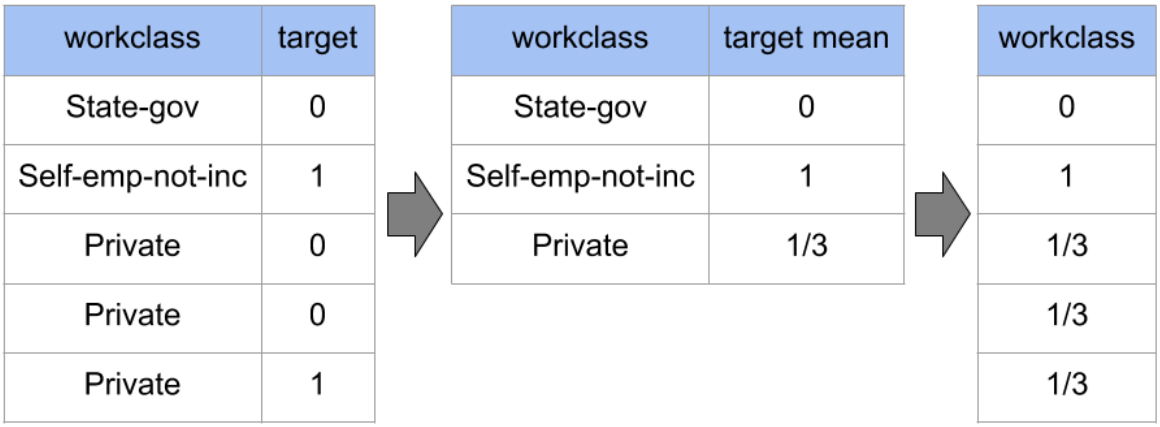
Подробнее о OneHotEncoding

Основной недостаток One Hot Encoding- генерация огромного количества новых признаков и все связанные с этим сложности, например, многократное увеличение объема памяти и времени обучения модели, появление квазиконстантных признаков (для редко встречающихся значений). Поэтому этот метод подходит для признаков с изначально небольшим числом значений.

## Mean Target Encoding

Другой способ дать категориальным признаком численный эквивалент - Mean Target Encoding - когда каждому признаку поставим в соответствие среднее значение таргета по данной категории.

## Target Encoding



| workclass | target |
|---|---|
| State-gov | 0 |
| Self-emp-not-inc | 1 |
| Private | 0 |
| Private | 0 |
| Private | 1 |

| workclass | target mean |
|---|---|
| State-gov | 0 |
| Self-emp-not-inc | 1 |
| Private | 1/3 |

| workclass |
|---|
| 0 |
| 1 |
| 1/3 |
| 1/3 |
| 1/3 |

*Ссылка на источник: https://www.machinelearningmastery.ru/all-about-categorical-variable-encoding-305f3361fd02/*

К недостаткам этого метода можно отнести прежде всего риск переобучения, так как значения рассчитываются на основе таргета.

Используем комбинацию данных методов - для признаков с числом категорий менее пяти - One Hot Encoding, для остальных применим Mean target Encoding:

```
for col in categorical_columns:
    if col != 'timestamp':
        if df[col].nunique() < 5:
            one_hot = pd.get_dummies(df[col], prefix=col, drop_first=True)
            df = pd.concat((df.drop(col, axis=1), one_hot), axis=1)

        else:
            mean_target = df.groupby(col)['log_price_doc'].mean()
            df[col] = df[col].map(mean_target)
```

Дополнительно об One Hot Encoding и Mean target Encoding.

# >Работа с датой/временем

Есть несколько путей работы с датой - во первых, можно перевести ее в категориальные признаки, выделив в отдельные колонки - год, месяц, день недели, время суток в зависимости от важности данного признака.

В случае, когда в данных встречается более одного признака с датой или речь идет о временных промежутках, удобно сразу перевести дату в числовое значение просто взяв разницу даты/времени.

Выделим в отдельные колонки месяц и год покупки жилья:

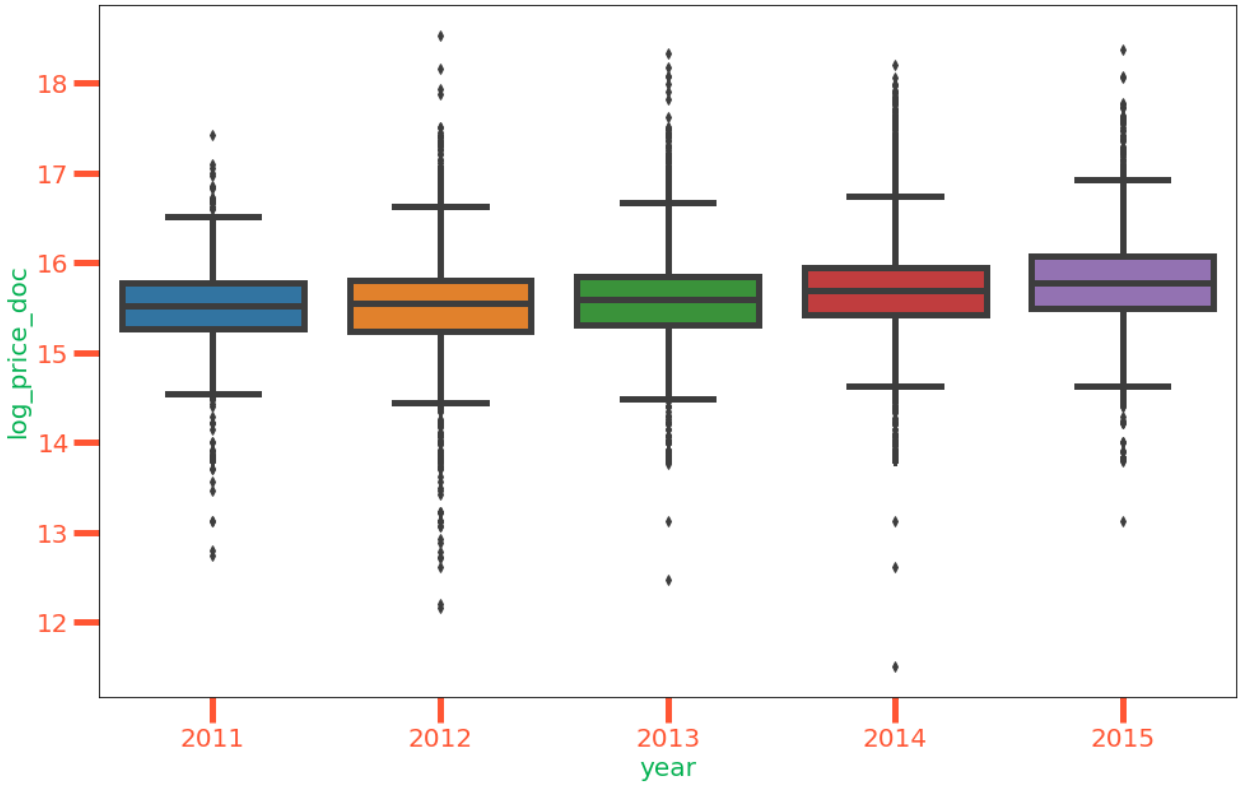
```
df['timestamp'] = pd.to_datetime(df['timestamp'])

df['month'] = df.timestamp.dt.month
df['year'] = df.timestamp.dt.year
```

**Как можно понять, насколько важна та или иная категория?**

Воспользуемся визуализацией, чтобы оценить значимость признака.

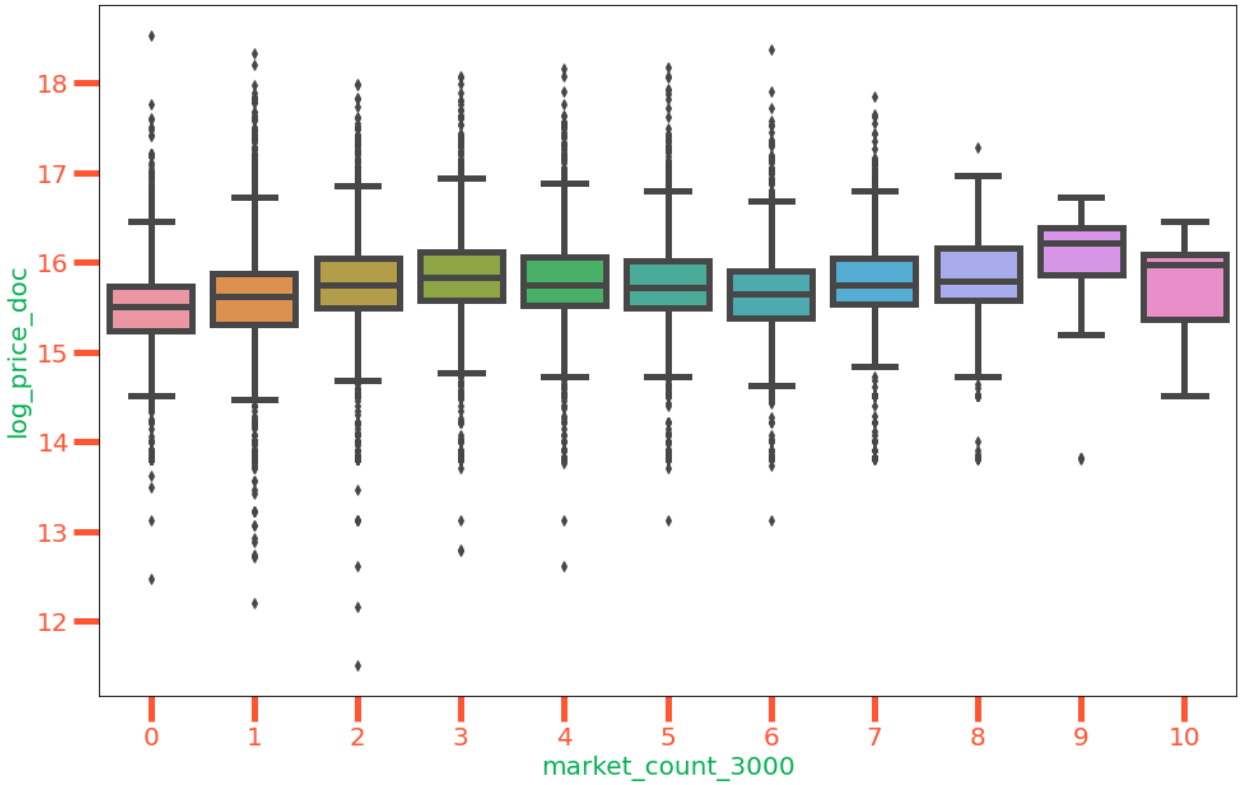
Посмотрим на распределение таргета по годам:

```
import matplotlib.pyplot as plt
import seaborn as sns
fig = plt.figure()
fig.set_size_inches(16, 10)
sns.boxplot(y='log_price_doc', x=df['year'].astype('category'), data=df)
plt.show()
```




По графику можно сделать вывод, что цена из года в год становится выше, поэтому признак имеет смысл оставить.

Ту же операцию с боксплотом можно проделать по каждому из категориальных признаков.




Помимо исследования каждого признака отдельно полезно посмотреть на их различные комбинации, попробовать объединить какие-либо признаки в один либо выделить новые.

# >Построение базовой модели. Валидация.

Важной особенностью нашего датасета является наличие временной структуры. В таком случае привычную кросс-валидацию проводить нельзя, так как в ходе такой операции получится, что будут использоваться данные “из будущего” для предсказания прошлого. Поэтому при валидации моделей с временной структурой важно, чтобы мы обучали модель на ранних и тестировали предсказание на более поздних данных.

Для этого существует модицифицованный вариант кросс-валидации - time-series split validation.

```
from sklearn.model_selection import TimeSeriesSplit

splitter = TimeSeriesSplit(n_splits=4)
```

Подробнее о  time-series split validation в статье на медиуме.

Встроенная функция кросс-валидации на примере модели Линейной регрессии:

```
from sklearn.model_selection import cross_validate

model = LinearRegression()

cv_result = cross_validate(model, X, Y,
                           scoring='neg_mean_squared_error',
                           cv=splitter, return_train_score=True)
```

# >Стандартизация и масштабирование признаков

Как показывает практика, лучше всего масштабировать данные на тренировочном датасете.

Отмасштабировать данные при использовании модели регуляризации можно с помощью класса Pipeline:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

pipe = Pipeline([('scaler', StandardScaler()), ('Lasso', Lasso(max_iter=100000))])
pipe.fit(X, Y)

print(pipe.predict(X.head(1)))

cv_result_pipe = cross_validate(pipe, X, Y,
                                scoring='neg_mean_squared_error',
                                cv=splitter, return_train_score=True)
```

`pipe.get_params()` - метод, позволяющий посмотреть все возможные параметры Pipeline

Подробнее о Pipeline в документации

Чтобы подобрать подходящий параметр регуляризации воспользуемся еще одним классом библиотеки sklearn - GridSearchCV, который перебирает все возможные параметры и отбирает их наилучшие комбинации:

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    "Lasso__alpha": alphas
}

### Передадим в GridSearchCV

search = GridSearchCV(pipe, param_grid,
                      cv=splitter, scoring='neg_mean_squared_error')

search.fit(X, Y)

print(f"Best parameter (CV score={search.best_score_:.5f}):")
print(search.best_params_)
```

Подробнее о `GridSearchCV` в документации.

# >Анализ выбросов

**Выбросы(англ. outliers)** – это наблюдения, удаленные от других в выборке. Из-за наличия выбросов в тестовой выборке модель может показывать худший результат, особенно когда нам нужно предсказывать значения в среднестатистических, типичных ситуациях.

Чтобы иметь лучшее  представление о работе с выбросами, обратимся к понятию из статистики.

**Квантиль(quantile)** -  число, которое заданная случайная величина не превышает с фиксированной вероятностью. Например, 0,025-квантиль – число, ниже которого лежит примерно 2,5% выборки.

Давайте очистим данные от выбросов и посмотрим, как это отразится на качестве модели:

```
top_quantile = data['log_price_doc'].quantile(0.975)
low_quantile = data['log_price_doc'].quantile(0.025)
```

```
print(f"Топ 2,5% значение таргета: {top_quantile.round(2)}")
print(f"Топ 97,5% значение таргета: {low_quantile.round(2)}")
```

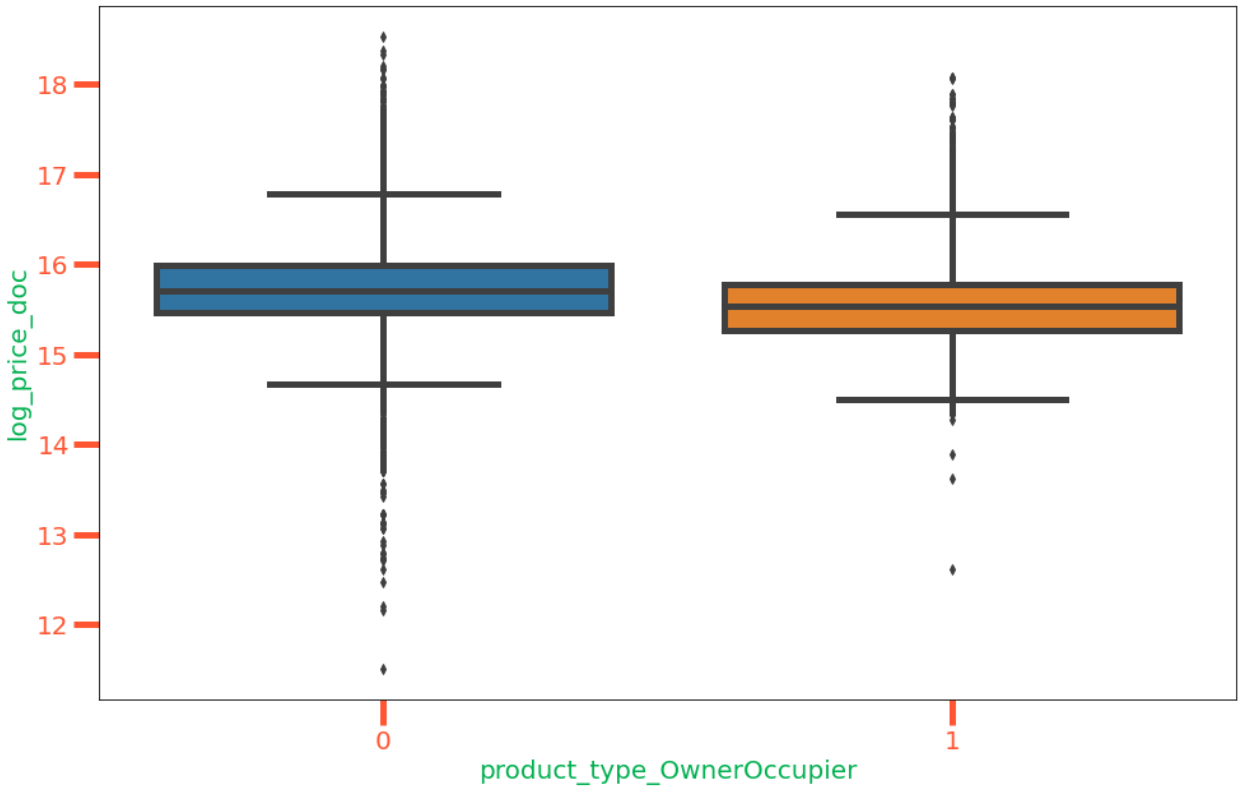
```
data = data[(data['log_price_doc']>low_quantile)&(data['log_price_doc']<top_quantile)]

X_new, Y_new = data.drop('log_price_doc', axis=1), data['log_price_doc']
```

Хорошая статья про выбросы.

# >Сегментация данных

Еще один механизм, который позволяет улучшить качество модели - сегментация данных.

Суть его в том, чтобы разбить данные по одному из признаков на категории и построить отдельные модели по каждому набору данных.



Разобьем данные о покупках жилья на первичном и вторичном рынке на два отдельных датафрейма и построим по ним отдельные модели:

```
Owner_Occupier = data[data['product_type_OwnerOccupier'] == 1].copy()
Investment = data[data['product_type_OwnerOccupier'] == 0].copy()

X_Occupier = Owner_Occupier.drop('log_price_doc', axis=1)
X_Investment = Investment.drop('log_price_doc', axis=1)

Y_Occupier = Owner_Occupier['log_price_doc']
Y_Investment = Investment['log_price_doc']
```

Обучим первую модель:

```
#модель для Owner_Occupier

search_Owner_Occupier = GridSearchCV(pipe, param_grid,
                                     cv=splitter, scoring='neg_mean_squared_error')

search_Owner_Occupier.fit(X_Occupier, Y_Occupier)

print(f"Best parameter (CV score={search_Owner_Occupier.best_score_:.5f}):")
print(search_Owner_Occupier.best_params_)

pipe.set_params(Lasso__alpha=search_Owner_Occupier.best_params_['Lasso__alpha'])

cv_result_pipe = cross_validate(pipe, X_Occupier, Y_Occupier,
                                scoring='neg_mean_squared_error',
                                cv=splitter, return_train_score=True)

error_Occupier_train = -np.mean(cv_result_pipe['train_score'])
error_Occupier_test = -np.mean(cv_result_pipe['test_score'])
```

Обучим вторую модель:

```
#модель для Investment

search_Investment = GridSearchCV(pipe, param_grid,
                                 cv=splitter, scoring='neg_mean_squared_error')

search_Investment.fit(X_Investment, Y_Investment)

print(f"Best parameter (CV score={search_Investment.best_score_:.5f}):")
```

```
print(search_Investment.best_params_)

pipe.set_params(Lasso__alpha=search_Investment.best_params_['Lasso__alpha'])

cv_result_pipe = cross_validate(pipe, X_Investment, Y_Investment,
                                scoring='neg_mean_squared_error',
                                cv=splitter, return_train_score=True)

error_Investment_train = -np.mean(cv_result_pipe['train_score'])
error_Investment_test = -np.mean(cv_result_pipe['test_score'])
```

Перевзвесим показатели качества с учетом количества объектов в обоих типах жилья:

```
n_Occupier = Owner_Occupier.shape[0] #количество объектов для первого типа жилья
n_Investment = Investment.shape[0]   #количество объектов для второго типа жилья

# Посчитаем доли категорий в общий выборке

share_Occupier = n_Occupier / data.shape[0]
share_Investment = n_Investment / data.shape[0]

#умножим MSLE полученных моделей на долю соответствующего вида жилья:
weighted_error_train = share_Occupier * error_Occupier_train + \\
                       share_Investment * error_Investment_train

weighted_error_test = share_Occupier * error_Occupier_test + \\
                       share_Investment * error_Investment_test
```

В итоге на выходе получаем две рабочих модели, которые в сумме дают меньшую ошибку и когда поступает очередной объект, по которому нужно сделать предсказание, то в зависимости от его сегмента, применяем соответствующую модель.