

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Московский авиационный институт
(национальный исследовательский университет)

Институт № 8
Компьютерные науки и прикладная математика
Кафедра 806 «Компьютерная математика»

КУРСОВАЯ РАБОТА
по дисциплине «Системы интеллектуальной поддержки»
на тему: Интеллектуальная система оценки
стоимости квартиры

Выполнил: студент группы М8О-311Б-20

Иваненков Лев Михайлович

(Фамилия, имя, отчество)

(подпись)

Принял: доцент кафедры 806

Чумакова Екатерина Витальевна

(Фамилия, имя, отчество)

(подпись)

Оценка:

Дата:

Москва, 2023

Содержание

ВВЕДЕНИЕ	3
1 Цели, задачи и методологическая база исследования.	3
2 Разведочное анализ данных	4
2.1 Отбор признаков и выбор таргета	5
2.2 Работа с пропусками	5
2.2.1 Некатегориальные колонки	5
2.2.2 Категориальные колонки	7
2.2.3 Временные признаки	8
2.3 Исследование распределения признаков	8
2.3.1 Распределения таргета по месяцам	8
2.3.2 Распределения таргета по этажу	9
2.3.3 Распределения таргета по количеству магазинов в радиусе 3км .	10
2.3.4 Распределения таргета по типу недвижимости	11
3 Построение модели	11
3.1 Разделение данных	12
3.2 Регуляризация	12
3.3 Анализ выбросов	14
3.4 Сегментация данных	14
ВЫВОД	15
ЗАКЛЮЧЕНИЕ	16
СПИСОК ЛИТЕРАТУРЫ	17
ПРИЛОЖЕНИЕ	18

ВВЕДЕНИЕ

В современной эпохе все чаще возникает необходимость в быстром и точном определении стоимости недвижимости как для продавцов, так и для покупателей. Определение стоимости недвижимости - это один из самых сложных и многоплановых процессов. Существует множество факторов, которые влияют на цену недвижимости, такие как: район, количество комнат, эмоциональная составляющая, общая площадь и т.д.

В связи с этим создание системы, способной оценить стоимость недвижимости на основе таких факторов, является актуальной проблемой. Одним из возможных решений данной проблемы является создание системы оценки стоимости квартиры на основе датасета из kaggle: Sberbank Russian Housing Market.

1 Цели, задачи и методологическая база исследования.

К основным целям создания такой модели можно отнести:

1. Облегчение процесса оценки стоимости квартиры: система поможет быстро и точно определить цену недвижимости, что позволит сократить время на продажу или покупку квартиры и сэкономить ресурс
2. Установление более объективной цены недвижимости: система будет учитывать множество факторов, что позволит установить более реалистичную и объективную цену недвижимости.
3. Повышение эффективности рынка недвижимости: создание такой системы сможет улучшить ситуацию на рынке недвижимости, поскольку будет предоставлять более точный и объективный показатель цен на квартиры.
4. Усиление конкурентоспособности риэлторов: риэлторы смогут использовать систему в своей работе, чтобы быстро и точно оценивать стоимость недвижимости, повышая тем самым свою конкурентоспособность.

5. Уменьшение рисков для покупателей: покупатели смогут доверять более объективным ценам на квартиры и не рисковать переплатить из-за недостаточной информации о рынке.

Данная система сможет выполнять разный спектр задач, к примеру:

1. Автоматическая оценка стоимости квартиры на основе множества факторов, таких как географическое расположение, количество комнат, наличие балкона, состояние квартиры и другие.
2. Предоставление пользователям качественной визуализации и интерактивных инструментов для анализа цен на недвижимость и других рыночных данных.
3. Создание отчетов и аналитических заметок для риэлторов, инвесторов и других заинтересованных лиц, которые помогут им принимать обоснованные решения на основе рыночных данных и прогнозов.
4. Улучшение процесса продажи и покупки недвижимости, позволяя пользователям быстрее и эффективнее находить и заключать сделки, уменьшая количество обращений к риэлторам и ускоряя процесс оценки недвижимости.

2 Разведочное анализ данных

Разведочный анализ данных (EDA) проводится для более глубокого понимания данных и выявления связей между переменными в датасете Sberbank Russian Housing Market. Во время EDA можно ответить на следующие вопросы:

- Какие признаки наиболее коррелируют с целевой переменной?
- Какие признаки имеют наибольшую дисперсию или имеют выбросы?
- Какие признаки являются категориальными, а какие числовыми, и какую обработку нужно провести для каждой категории?
- Существуют ли в данных пропущенные значения, и как они могут повлиять на модель?
- Какие признаки могут быть удалены из датасета из-за нерелевантности или мультиколлинеарности?

В результате EDA можно получить представление о структуре данных, которые помогут в построении более точной модели на датасете Sberbank Russian Housing Market.

Исходное описание признаков можно посмотреть в [приложение а](#).

2.1 Отбор признаков и выбор таргета

Эти колонки заранее удалены т.к. эти id описывают местоположение квартиры, а у нас и так есть такие колонки

```
1
2 df = df.drop(['ID_metro',
3 'ID_railroad_station_walk',
4 'ID_railroad_station_avto',
5 'ID_big_road1',
6 'ID_big_road2',
7 'ID_railroad_terminal',
8 'ID_bus_terminal'], axis=1)
```

Так как в соревновании использовали метрику - RMSLE. По факту, это корень от MSLE, поэтому неважно, какую из них оптимизировать!

```
1
2 import numpy as np
3
4 df = df.assign(log_price_doc=np.log1p(df['price_doc']))
5 df = df.drop('price_doc', axis=1)
6
```

2.2 Работа с пропусками

2.2.1 Некатегориальные колонки

Некатегориальные колонки заполним средним

```
1
2 numeric_columns = df.loc[:,df.dtypes!=np.object].columns
3 df.describe()
4 for col in numeric_columns:
5     df[col] = df[col].fillna(df[col].mean())
```

	full_sq	life_sq	floor	max_floor	material	build_year	num_room	kitch_sq	state
count	30471.000000	24088.000000	30304.000000	20899.000000	20899.000000	1.686600e+04	20899.000000	20899.000000	16912.000000
mean	54.214269	34.403271	7.670803	12.558974	1.827121	3.068057e+03	1.909804	6.399301	2.107025
std	38.031487	52.285733	5.319989	6.756550	1.481154	1.543878e+05	0.851805	28.265979	0.880148
min	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000e+00	0.000000	0.000000	1.000000
25%	38.000000	20.000000	3.000000	9.000000	1.000000	1.967000e+03	1.000000	1.000000	1.000000
50%	49.000000	30.000000	6.500000	12.000000	1.000000	1.979000e+03	2.000000	6.000000	2.000000
75%	63.000000	43.000000	11.000000	17.000000	2.000000	2.005000e+03	2.000000	9.000000	3.000000
max	5326.000000	7478.000000	77.000000	117.000000	6.000000	2.005201e+07	19.000000	2014.000000	33.000000

Рис. 1: Некатегориальные колонки

Корелляции вещественных признаков

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

	full_sq	life_sq	floor	max_floor	material	build_year	num_room	kitch_sq	state	area_m
full_sq	1.000000	0.153837	0.089450	0.057390	0.015820	-0.002532	0.334760	0.009640	-0.038544	0.056303
life_sq	0.153837	1.000000	0.038603	0.034483	0.010200	-0.002324	0.169211	0.000651	-0.058546	0.043262
floor	0.089450	0.038603	1.000000	0.373873	-0.007439	0.000855	-0.004654	-0.006957	-0.083337	-0.019321
max_floor	0.057390	0.034483	0.373873	1.000000	0.045915	-0.000215	-0.014220	0.020345	-0.061203	-0.079591
material	0.015820	0.010200	-0.007439	0.045915	1.000000	-0.004029	-0.026924	0.038747	-0.031320	0.001012
...
mosque_count_5000	0.021568	0.008870	-0.012222	-0.048229	0.041689	0.014699	0.051980	0.010915	0.068570	-0.086786
leisure_count_5000	0.030218	0.019099	-0.044093	-0.040977	0.037472	-0.000601	0.049396	-0.005270	-0.002915	-0.195067
sport_count_5000	0.001580	-0.012124	-0.101769	-0.083772	0.082620	0.004257	0.075427	0.014149	0.143328	-0.416222
market_count_5000	-0.041254	-0.043462	-0.123534	-0.094960	0.063992	0.005333	0.051672	0.022122	0.203451	-0.449849
log_price_doc	0.271408	0.119971	0.117870	0.078692	0.011807	0.003203	0.347790	0.009964	0.067892	-0.156493

Рис. 2: Корелляции вещественных признаков

Фильтрация признаков

```

1 def get_redundant_pairs(df):
2     pairs_to_drop = set()
3     cols = df.columns
4     for i in range(0, df.shape[1]):
5         for j in range(0, i+1):
6             pairs_to_drop.add((cols[i], cols[j]))
7     return pairs_to_drop
8
9 def get_top_abs_correlations(df, n=5):
10     au_corr = df.corr().abs().unstack()
11     labels_to_drop = get_redundant_pairs(df)
12     au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
13     return au_corr[0:n]
14
15 print("Top Absolute Correlations")
16 print(get_top_abs_correlations(df[numeric_columns], 50))
17

```

Удаление колонок, где корреляция оказывается > 0.9

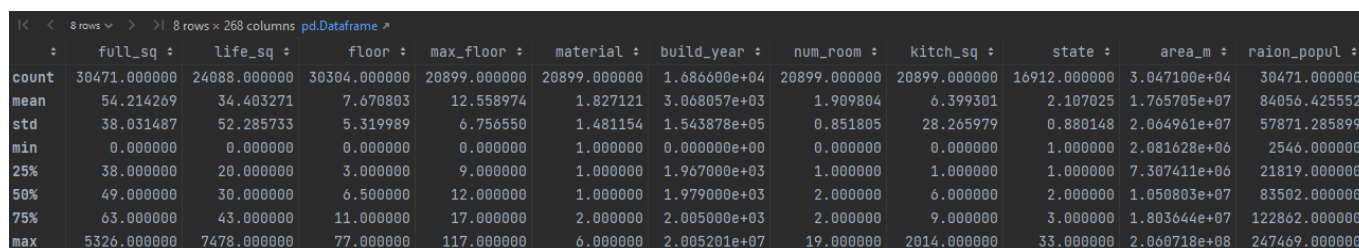
```
1 def correlation(dataset, threshold):
2     col_corr = set() # Set of all the names of deleted columns
3     corr_matrix = dataset.corr()
4     for i in range(len(corr_matrix.columns)):
5         for j in range(i):
6             if (corr_matrix.iloc[i, j] >= threshold) and (corr_matrix.columns[j] not in col_corr):
7                 colname = corr_matrix.columns[i] # getting the name of column
8                 col_corr.add(colname)
9                 if colname in dataset.columns:
10                     del dataset[colname] # deleting the column from the dataset
11
12 correlation(df, 0.9)
13
14 numeric_columns = df.loc[:, df.dtypes != np.object].columns
15
16 df.shape
```

У нас остаось 150 признаков.

2.2.2 Категориальные колонки

Категориальные колонки

```
1 categorical_columns = df.loc[:, df.dtypes == np.object].columns
2
3 df.describe()
```



	full_sq	life_sq	floor	max_floor	material	build_year	num_room	kitch_sq	state	area_m	raion_popul
count	30471.000000	24088.000000	30304.000000	20899.000000	20899.000000	1.686600e+04	20899.000000	20899.000000	16912.000000	3.047100e+04	30471.000000
mean	54.214269	34.403271	7.670803	12.558974	1.827121	3.068057e+03	1.909804	6.399301	2.107025	1.765705e+07	84056.425552
std	38.031487	52.285733	5.319989	6.756550	1.481154	1.543878e+05	0.851805	28.265979	0.880148	2.064961e+07	57871.285899
min	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000e+00	0.000000	0.000000	1.000000	2.081628e+06	2546.000000
25%	38.000000	20.000000	3.000000	9.000000	1.000000	1.967000e+03	1.000000	1.000000	1.000000	7.307411e+06	21819.000000
50%	49.000000	30.000000	6.500000	12.000000	1.000000	1.979000e+03	2.000000	6.000000	2.000000	1.050803e+07	83502.000000
75%	63.000000	43.000000	11.000000	17.000000	2.000000	2.005000e+03	2.000000	9.000000	3.000000	1.803644e+07	122862.000000
max	5326.000000	7478.000000	77.000000	117.000000	6.000000	2.005201e+07	19.000000	2014.000000	33.000000	2.060718e+08	247469.000000

Рис. 3: Категориальные колонки

Преобразуем категориальные колонки

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

```

5         df = pd.concat((df.drop(col, axis=1), one_hot), axis=1)
6
7     else:
8         mean_target = df.groupby(col)['log_price_doc'].mean()
9         df[col] = df[col].map(mean_target)

```

2.2.3 Временные признаки

```

1 df['timestamp'] = pd.to_datetime(df['timestamp'])
2 df['month'] = df.timestamp.dt.month
3 df['year'] = df.timestamp.dt.year
4 df.head()
5 df = df.sort_values('timestamp')
6 df.head()

```

Отсортируем по timestamp

	timestamp	full_sq	life_sq	floor	max_floor	material	build_year	num_room	kitch_sq	state	sub_area	area_m
0	2011-08-20	43	27.0	4.0	12.558974	1.827121	3068.057097	1.909804	6.399301	2.107025	15.594247	6.407578e+06
1	2011-08-23	34	19.0	3.0	12.558974	1.827121	3068.057097	1.909804	6.399301	2.107025	15.864842	9.589337e+06
2	2011-08-27	43	29.0	2.0	12.558974	1.827121	3068.057097	1.909804	6.399301	2.107025	15.613141	4.808270e+06
3	2011-09-01	89	50.0	9.0	12.558974	1.827121	3068.057097	1.909804	6.399301	2.107025	15.914449	1.258354e+07
4	2011-09-05	77	77.0	4.0	12.558974	1.827121	3068.057097	1.909804	6.399301	2.107025	16.091227	8.398461e+06

Рис. 4: Отсортированные данные по timestamp

2.3 Исследование распределения признаков

2.3.1 Распределения таргета по месяцам

```

1 fig = plt.figure()
2 fig.set_size_inches(16, 10)
3
4 sns.boxplot(y='log_price_doc', x=df['month'].astype('category'), data=df)
5 plt.show()
6 one_hot = pd.get_dummies(df['month'], prefix='month', drop_first=True)
7 df = pd.concat((df.drop('month', axis=1), one_hot), axis=1)

```

Закодируем данную колонку с месяцем через One-Hot

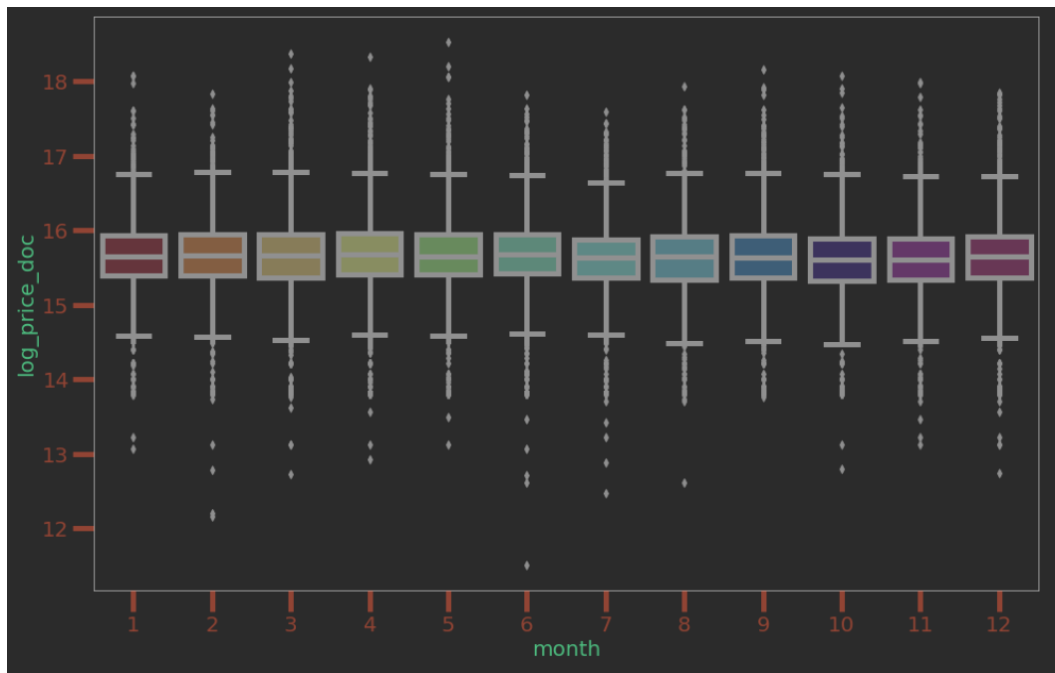


Рис. 5: Распределения таргета по месяцам

2.3.2 Распределения таргета по этажу

```

1 fig = plt.figure()
2 fig.set_size_inches(16, 10)
3
4 sns.boxplot(y='log_price_doc', x=df['floor'].astype('category'), data=df)
5 plt.show()

```

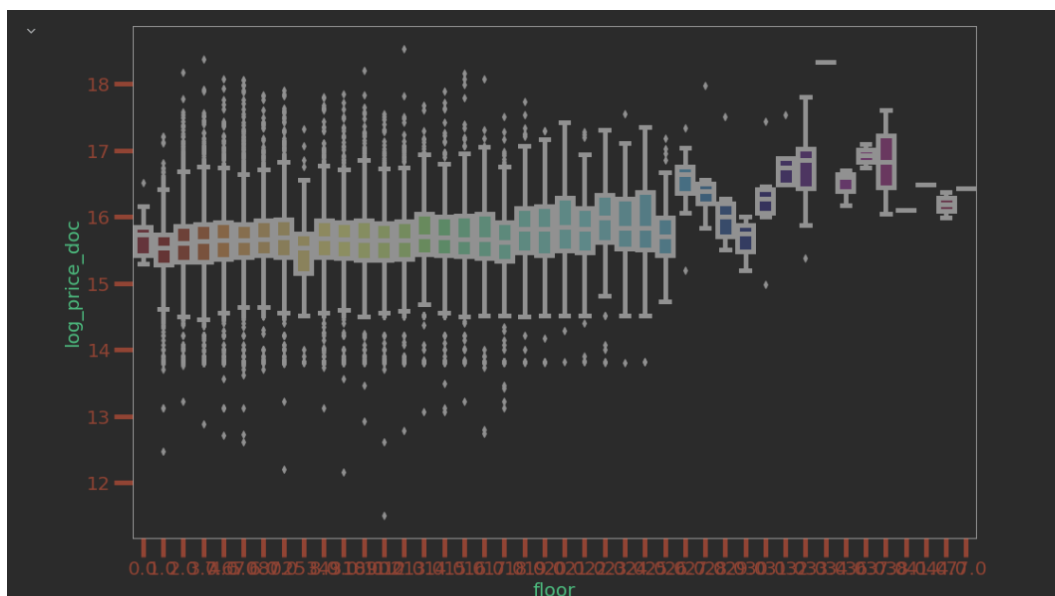


Рис. 6: Распределения таргета по этажу

2.3.3 Распределения таргета по количеству магазинов в радиусе 3км

```
1 fig = plt.figure()
2 fig.set_size_inches(16, 10)
3
4 sns.boxplot(y='log_price_doc', x=df['market_count_3000'].astype('category'), data=df)
5 plt.show()
```

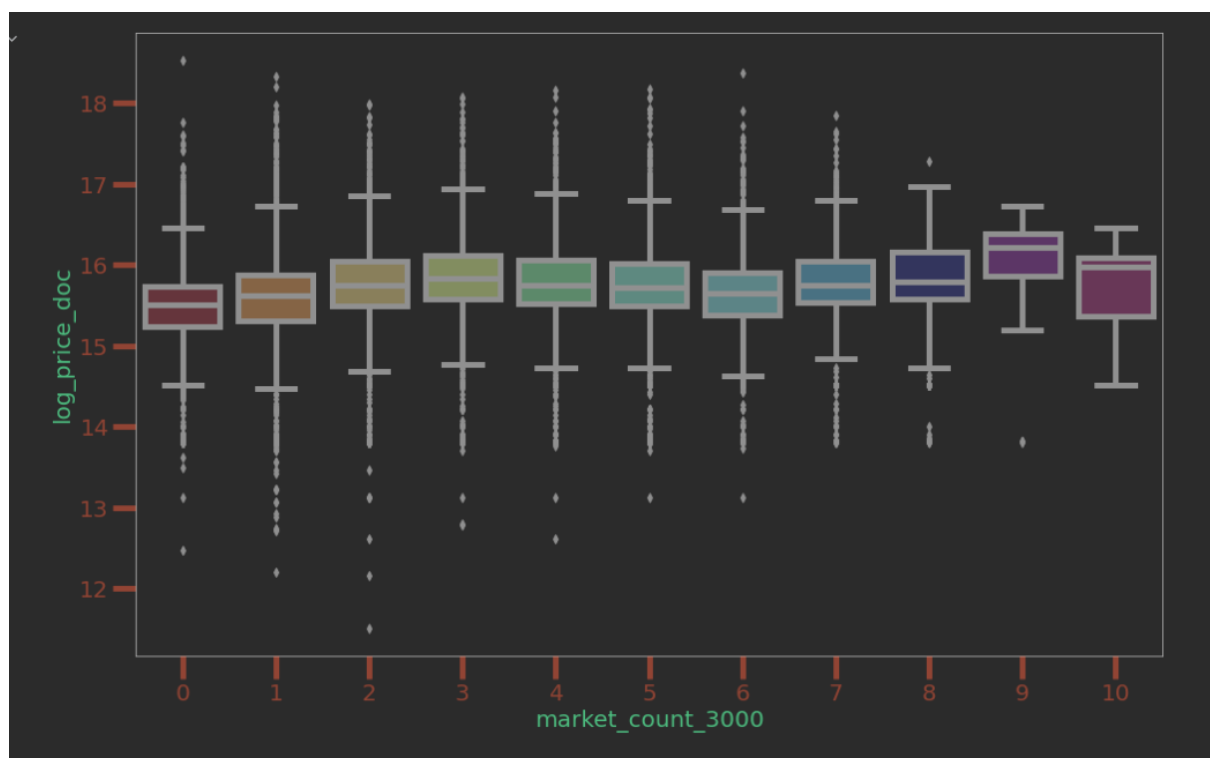


Рис. 7: Распределения таргета по количеству магазинов в радиусе 3км

2.3.4 Распределения таргета по типу недвижимости

```
1 fig = plt.figure()
2 fig.set_size_inches(16, 10)
3
4 sns.boxplot(y='log_price_doc', x=df['product_type_OwnerOccupier'].astype('category'), data=df)
5 plt.show()
```

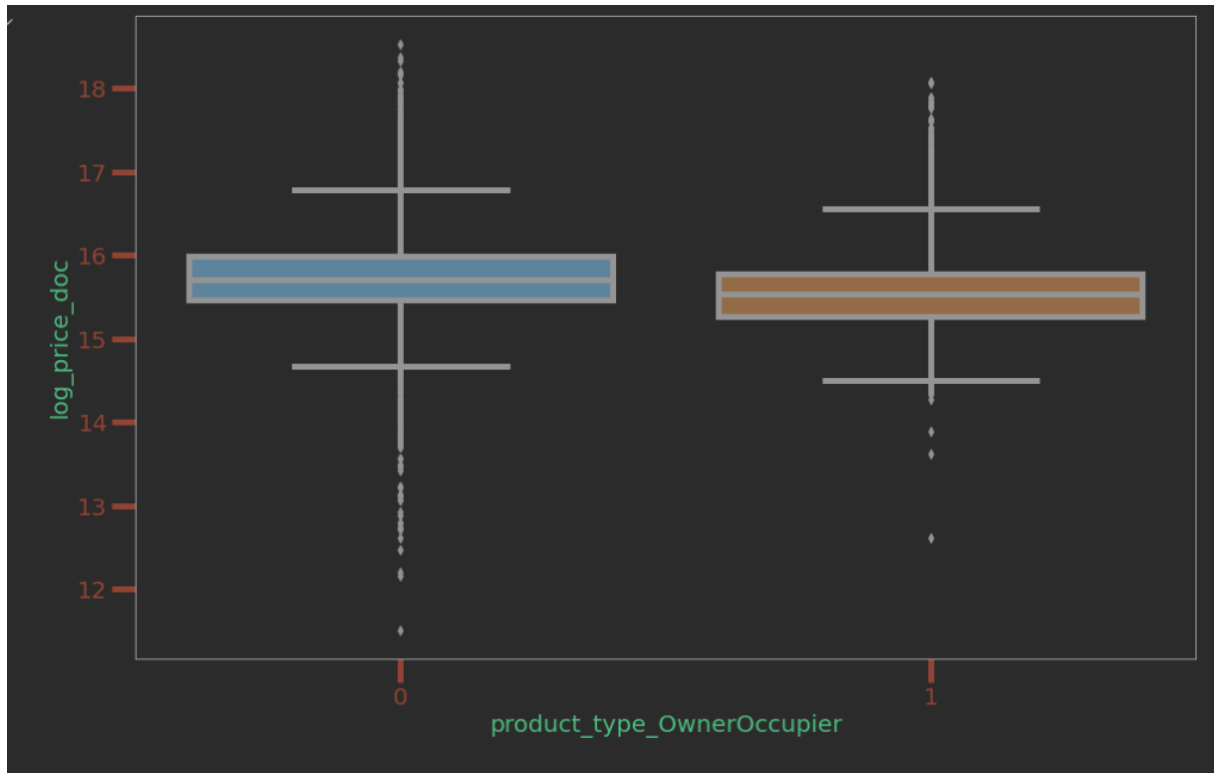


Рис. 8: Распределения таргета по типу недвижимости

3 Построение модели

После EDA данные обладают временной структурой. Поэтому, чтобы получить хорошую обобщающую способность, нужно построить не просто модель, хорошо работающую на новых данных, а модель, которая угадывает распределение данных в будущем хотя бы на коротком горизонте. Поэтому, когда мы валидируем дизайн модели, нам важно делить на каждом шаге трейн и тест таким образом, чтобы по временной шкале они не пересекались, и точки из второго множества появлялись позже точек из первого. В качестве модели используется линейная регрессия с регуляризацией.

3.1 Разделение данных

Для этого разделим выборку на валидацию и тест

```
1 from sklearn.model_selection import TimeSeriesSplit
2
3 splitter = TimeSeriesSplit(n_splits=4)
4
5 from sklearn.model_selection import KFold
6 from sklearn.linear_model import LinearRegression
7
8
9 test_losses = []
10 train_losses = []
11
12 for train_index, test_index in splitter.split(X):
13
14     x_train, x_test = X.values[train_index], X.values[test_index]
15     y_train, y_test = Y.values[train_index], Y.values[test_index]
16
17     model = LinearRegression()
18     model.fit(x_train, y_train)
19
20     preds_test = model.predict(x_test)
21     preds_train = model.predict(x_train)
22
23     error_test = np.mean((preds_test - y_test)**2)
24     error_train = np.mean((preds_train - y_train)**2)
25
26     test_losses.append(error_test)
27     train_losses.append(error_train)
28
29 print(f"Среднее MSLE на тренировочных фолдах: {np.mean(train_losses).round(3)}")
30 print(f"Среднее MSLE на тестовых фолдах: {np.mean(test_losses).round(3)}")
```

Среднее MSLE на тренировочных фолдах: 0.24

Среднее MSLE на тестовых фолдах: 1.455

3.2 Регуляризация

Регуляризация используется в линейной регрессии, чтобы избежать проблемы переобучения (overfitting) модели на тренировочных данных. Переобучение происходит, когда модель слишком хорошо подстраивается под тренировочные данные, заучивая

их, вместо обобщения правильных закономерностей в данных. В результате, модель становится менее точной на новых, незнакомых данных.

Регуляризация работает путем добавления штрафа к функции потери, который направлен на минимизацию весов модели. Два основных вида регуляризации, используемые в линейной регрессии - L1 (лассо) и L2 (гребневая).

L1 регуляризация предотвращает переобучение через отбор признаков. Она ограничивает коэффициенты признаков, уменьшая нулевые коэффициенты и удаляя нерелевантные признаки.

L2 регуляризация, с другой стороны, снижает веса коэффициентов признаков, уменьшая их, но не приводя их к нулю. Она часто применяется для борьбы с корреляцией признаков.

Использование регуляризации при обучении модели линейной регрессии позволяет получить более обобщенную модель, которая предпочитает уменьшать веса признаков, снижая возможность переобучения и улучшая качество предсказаний на новых данных.

```
1 from sklearn.linear_model import Lasso, Ridge
2
3 model_lasso = Lasso(max_iter=100000)
4
5 cv_result_lasso = cross_validate(model_lasso, X, Y,
6                                 scoring='neg_mean_squared_error',
7                                 cv=splitter, return_train_score=True)
8
9 print(f"Среднее MSLE на тренировочных фолдах: {-np.mean(cv_result_lasso['train_score']).round(3)}")
10 print(f"Среднее MSLE на тестовых фолдах: {-np.mean(cv_result_lasso['test_score']).round(3)}")
```

Среднее MSLE на тренировочных фолдах: 0.435

Среднее MSLE на тестовых фолдах: 0.446

Видим что разница в score на тестовой выборке и на трейне уменьшилась, соответственно теперь модель не переобучается.

Также добавим этап нормировки данных.

```
1
2 from sklearn.pipeline import Pipeline
3 from sklearn.preprocessing import StandardScaler
4
5 pipe = Pipeline([('scaler', StandardScaler()), ('Lasso', Lasso(max_iter=100000))])
6 pipe.fit(X, Y)
7
```

```

8 print(pipe.predict(X.head(1)))
9
10 cv_result_pipe = cross_validate(pipe, X, Y,
11                                 scoring='neg_mean_squared_error',
12                                 cv=splitter, return_train_score=True)

```

Среднее MSLE на тренировочных фолдах: 0.365

Среднее MSLE на тестовых фолдах: 0.366

Скор повысился.

3.3 Анализ выбросов

```

1 top_quantile = data['log_price_doc'].quantile(0.975)
2 low_quantile = data['log_price_doc'].quantile(0.025)
3
4 print(f"Топ 2,5% значение таргета: {top_quantile.round(2)}")
5 print(f"Топ 97,5% значение таргета: {low_quantile.round(2)}")

```

Топ 2,5 процент значение таргета: 16.7 Топ 97,5 процент значение таргета: 13.82

Выбросим объекты со значениями вне отрезка [top 2,5; top97,5]

```

1 data = data[(data['log_price_doc']>low_quantile)&(data['log_price_doc']<top_quantile)]
2
3 X_new, Y_new = data.drop('log_price_doc', axis=1), data['log_price_doc']

```

3.4 Сегментация данных

Разделим квартиры по типу недвижимости.

Для первички и вторички будем строить разные модели.

```

1 Owner_Occupier = data[data['product_type_OwnerOccupier'] == 1].copy()
2 Investment = data[data['product_type_OwnerOccupier'] == 0].copy()
3 X_Occupier = Owner_Occupier.drop('log_price_doc', axis=1)
4 X_Investment = Investment.drop('log_price_doc', axis=1)
5
6 Y_Occupier = Owner_Occupier['log_price_doc']
7 Y_Investment = Investment['log_price_doc']
8
9 search_Owner_Occupier = GridSearchCV(pipe, param_grid,
10                                     cv=splitter, scoring='neg_mean_squared_error')
11
12 search_Owner_Occupier.fit(X_Occupier, Y_Occupier)
13

```

```

14 print(f"Best parameter (CV score={search_Owner_Occupier.best_score_:.5f}):")
15 print(search_Owner_Occupier.best_params_)
16
17 pipe.set_params(Lasso__alpha=search_Owner_Occupier.best_params_['Lasso__alpha'])
18
19 cv_result_pipe = cross_validate(pipe, X_Occupier, Y_Occupier,
20                                 scoring='neg_mean_squared_error',
21                                 cv=splitter, return_train_score=True)
22
23 error_Occupier_train = -np.mean(cv_result_pipe['train_score'])
24 error_Occupier_test = -np.mean(cv_result_pipe['test_score'])
25
26 print(f"Среднее MSLE на тренировочных фолдах: {error_Occupier_train.round(3)}")
27 print(f"Среднее MSLE на тестовых фолдах: {error_Occupier_test.round(3)}")

```

Среднее взвешенное MSLE на тренировочных фолдах: 0.154

Среднее взвешенное MSLE на тестовых фолдах: 0.16

ВЫВОД

Модель линейной регрессии, построенная на датасете Sberbank Russian Housing Market, может быть использована для предсказания цен на недвижимость. Эта модель может быть полезна для риэлторских агентств, инвесторов, застройщиков и других заинтересованных сторон для принятия решений, связанных с рынком недвижимости в России. Например, риэлторские агентства могут использовать данную модель для оценки рыночной стоимости недвижимости при проведении сделок, инвесторы могут использовать модель для прогнозирования цен на недвижимость и принятия решений об инвестициях, застройщики могут использовать модель для оценки цен на жилье при планировании новых строительных проектов. Важно учитывать, что результаты модели линейной регрессии могут не всегда быть точными и нужно проводить дополнительный анализ данных и уточнение предсказаний модели перед принятием окончательного решения.

ЗАКЛЮЧЕНИЕ

Таким образом, была построена модель, которая может определять цену по заданным параметрам. Были использованы такие приемы для создания модели как:

1. Разведочный анализ данных (EDA).
2. Исследование распределения признаков.
3. Регуляризация.
4. Сегментация данных.

СПИСОК ЛИТЕРАТУРЫ

1. Николенко С., Кадурын А. Глубокое обучение. Погружение в мир нейронных сетей - 2016.- 237с.
2. Флах П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. – М., 2009.

ПРИЛОЖЕНИЕ

ПРИЛОЖЕНИЕ А

```
1
2 # train.csv and test.csv
3
4 price_doc: sale price (this is the target variable)
5 id: transaction id
6 timestamp: date of transaction
7 full_sq: total area in square meters, including loggias, balconies and other non-residential areas
8 life_sq: living area in square meters, excluding loggias, balconies and other non-residential areas
9 floor: for apartments, floor of the building
10 max_floor: number of floors in the building
11 material: wall material
12 build_year: year built
13 num_room: number of living rooms
14 kitch_sq: kitchen area
15 state: apartment condition
16 product_type: owner-occupier purchase or investment
17 sub_area: name of the district
18
19 The dataset also includes a collection of features about each propertys surrounding 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.
20
21 full_all: subarea population
22 male_f, female_f: subarea population by gender
23 young_*: population younger than working age
24 work_*: working-age population
25 ekder_*: retirement-age population
26 n_m_{all|male|female}: population between n and m years old
27 build_count_*: buildings in the subarea by construction type or year
28 x_count_500: the number of x within 500m of the property
29 x_part_500: the share of x within 500m of the property
30 _sqm_: square meters
31 cafe_count_d_price_p: number of cafes within d meters of the property that have an average bill under
   ↪ p RUB
32 trc_: shopping malls
33 prom_: industrial zones
34 green_: green zones
35 metro_: subway
36 _avto_: distances by car
37 mkad_: Moscow Circle Auto Road
38 ttk_: Third Transport Ring
39 sadovoe_: Garden Ring
```

```

40 bulvar_ring_: Boulevard Ring
41 kremlin_: City center
42 zd_vokzaly_: Train station
43 oil_chemistry_: Dirty industry
44 ts_: Power plant
45
46
47 # macro.csv
48
49 A set of macroeconomic indicators, one for each date.
50
51 timestamp: Transaction timestamp
52 oil_urals: Crude Oil Urals ($/bbl)
53 gdp_quart: GDP
54 gdp_quart_growth: Real GDP growth
55 cpi: Inflation - Consumer Price Index Growth
56 ppi: Inflation - Producer Price index Growth
57 gdp_deflator: Inflation - GDP deflator
58 balance_trade: Trade surplus
59 balance_trade_growth: Trade balance (as a percentage of previous year)
60 usdrub: Ruble/USD exchange rate
61 eurrub: Ruble/EUR exchange rate
62 brent: London Brent ($/bbl)
63 net_capital_export: Net import / export of capital
64 gdp_annual: GDP at current prices
65 gdp_annual_growth: GDP growth (in real terms)
66 average_provision_of_build_contract: Provision by orders in Russia (for the developer)
67 average_provision_of_build_contract_moscow: Provision by orders in Moscow (for the developer)
68 rts: Index RTS / return
69 micex: MICEX index / return
70 micex_rghi_tr: MICEX index for government bonds (MICEX RGI TR) / yield
71 micex_cbi_tr: MICEX Index corporate bonds (MICEX CBI TR) / yield
72 deposits_value: Volume of household deposits
73 deposits_growth: Volume growth of populations deposits
74 deposits_rate: Average interest rate on deposits
75 mortgage_value: Volume of mortgage loans
76 mortgage_growth: Growth of mortgage lending
77 mortgage_rate: Weighted average rate of mortgage loans
78 grp: GRP of the subject of Russian Federation where Apartment is located
79 grp_growth: Growth of gross regional product of the subject of the Russian Federation where Apartment
    ↪ is located
80 income_per_cap: Average income per capita
81 real_dispos_income_per_cap_growth: Growth in real disposable income of Population
82 salary: Average monthly salary
83 salary_growth: Growth of nominal wages
84 fixed_basket: Cost of a fixed basket of consumer goods and services for inter-regional comparisons of
    ↪ purchasing power

```

85 retail_trade_turnover: Retail trade turnover
86 retail_trade_turnover_per_cap: Retail trade turnover per capita
87 retail_trade_turnover_growth: Retail turnover (in comparable prices in% to corresponding period of
↪ previous year)
88 labor_force: Size of labor force
89 unemployment: Unemployment rate
90 employment: Employment rate
91 invest_fixed_capital_per_cap: Investments in fixed capital per capita
92 invest_fixed_assets: Absolute volume of investments in fixed assets
93 profitable_enterpr_share: Share of profitable enterprises
94 unprofitable_enterpr_share: The share of unprofitable enterprises
95 share_own_revenues: The share of own revenues in the total consolidated budget revenues
96 overdue_wages_per_cap: Overdue wages per person
97 fin_res_per_cap: The financial results of companies per capita
98 marriages_per_1000_cap: Number of marriages per 1,000 people
99 divorce_rate: The divorce rate / growth rate
100 construction_value: Volume of construction work performed (million rubles)
101 invest_fixed_assets_phys: The index of physical volume of investment in fixed assets (in comparable
↪ prices in% to the corresponding month of Previous year)
102 pop_natural_increase: Rate of natural increase / decrease in Population (1,000 persons)
103 pop_migration: Migration increase (decrease) of population
104 pop_total_inc: Total population growth
105 childbirth: Childbirth
106 mortality: Mortality
107 housing_fund_sqm: Housing Fund (sqm)
108 lodging_sqm_per_cap: Lodging (sqm / pax)
109 water_pipes_share: Plumbing availability (pax)
110 baths_share: Bath availability (pax)
111 sewerage_share: Canalization availability
112 gas_share: Gas (mains, liquefied) availability
113 hot_water_share: Hot water availability
114 electric_stove_share: Electric heating for the floor
115 heating_share: Heating availability
116 old_house_share: Proportion of old and dilapidated housing, percent
117 average_life_exp: Average life expectancy
118 infant_mortality_per_1000_cap: Infant mortality rate (per 1,000 children aged up to one year)
119 perinatal_mort_per_1000_cap: Perinatal mortality rate (per 1,000 live births)
120 incidence_population: Overall incidence of the total population
121 rent_price_4+room_bus: rent price for 4-room apartment, business class
122 rent_price_3room_bus: rent price for 3-room apartment, business class
123 rent_price_2room_bus: rent price for 2-room apartment, business class
124 rent_price_1room_bus: rent price for 1-room apartment, business class
125 rent_price_3room_eco: rent price for 3-room apartment, econom class
126 rent_price_2room_eco: rent price for 2-room apartment, econom class
127 rent_price_1room_eco: rent price for 1-room apartment, econom class
128 load_of_teachers_preschool_per_teacher: Load of teachers of preschool educational institutions (number
↪ of children per 100 teachers);

129 child_on_acc_pre_school: Number of children waiting for the determination to pre-school educational
 ↳ institutions, for capacity of 100
 130 load_of_teachers_school_per_teacher: Load on teachers in high school (number of pupils in high school
 ↳ for 100 teachers)
 131 students_state_oneshift: Proportion of pupils in high schools with one shift, of the total number of
 ↳ pupils in high schools
 132 modern_education_share: Share of state (municipal) educational organizations, corresponding to modern
 ↳ requirements of education in the total number of high schools;
 133 old_education_build_share: The share of state (municipal) educational organizations, buildings are in
 ↳ disrepair and in need of major repairs of the total number.
 134 provision_doctors: Provision (relative number) of medical doctors in area
 135 provision_nurse: Provision of nursing staff
 136 load_on_doctors: The load on doctors (number of visits per physician)
 137 power_clinics: Capacity of outpatient clinics
 138 hospital_beds_available_per_cap: Availability of hospital beds per 100 000 persons
 139 hospital_bed_occupancy_per_year: Average occupancy rate of the hospital beds during a year
 140 provision_retail_space_sqm: Retail space
 141 provision_retail_space_modern_sqm: Provision of population with retail space of modern formats, square
 ↳ meter
 142 retail_trade_turnover_per_cap: Retail trade turnover per capita
 143 turnover_catering_per_cap: Turnover of catering industry per person
 144 theaters_viewers_per_1000_cap: Number of theaters viewers per 1000 population
 145 seats_theater_rfmin_per_100000_cap: Total number of seats in Auditorium of the Ministry of Culture
 ↳ Russian theaters per 100,000 population
 146 museum_visits_per_100_cap: Number of visits to museums per 1000 of population
 147 bandwidth_sports: Capacity of sports facilities
 148 population_reg_sports_share: Proportion of population regularly doing sports
 149 students_reg_sports_share: Proportion of pupils and students regularly doing sports in the total
 ↳ number
 150 apartment_build: City residential apartment construction
 151 apartment_fund_sqm: City residential apartment fund
 152
 153
 154 # Complete description of neighbourhood features
 155
 156 area_m Area mun. area, sq.m.
 157 raion_popul Number of municipality population. district
 158 green_zone_part Proportion of area of greenery in the total area
 159 indust_part Share of industrial zones in area of the total area
 160 children_preschool Number of pre-school age population
 161 preschool_quota Number of seats in pre-school organizations
 162 preschool_education_centers_raion Number of pre-school institutions
 163 children_school Population of school-age children
 164 school_quota Number of high school seats in area
 165 school_education_centers_raion Number of high school institutions
 166 school_education_centers_top_20_raion Number of high schools of the top 20 best schools in Moscow
 167 hospital_beds_raion Number of hospital beds for the district

168 healthcare_centers_raion Number of healthcare centers in district
169 university_top_20_raion Number of higher education institutions in the top ten ranking of the
↔ Federal rank
170 sport_objects_raion Number of higher education institutions
171 additional_education_raion Number of additional education organizations
172 culture_objects_top_25 Presence of the key objects of cultural heritage (significant objects for the
↔ level of the RF constituent entities, city)
173 culture_objects_top_25_raion Number of objects of cultural heritage
174 shopping_centers_raion Number of malls and shopping centres in district
175 office_raion Number of malls and shopping centres in district
176 thermal_power_plant_raion Presence of thermal power station in district
177 incineration_raion Presence of incinerators
178 oil_chemistry_raion Presence of dirty industries
179 radiation_raion Presence of radioactive waste disposal
180 railroad_terminal_raion Presence of the railroad terminal in district
181 big_market_raion Presence of large grocery / wholesale markets
182 nuclear_reactor_raion Presence of existing nuclear reactors
183 detention_facility_raion Presence of detention centers, prisons
184 full_all Total number of population in the municipality
185 male_f Male population
186 female_f Female population
187 young_all Population younger than working age
188 young_male Male population younger than working age
189 young_female Female population younger than working age
190 work_all Working-age population
191 work_male Male working-age population
192 work_female Female working-age population
193 ekder_all Population older than working age
194 ekder_male Male population older than working age
195 ekder_female Female population older than working age
196 0_6_all Population aged 0-6
197 0_6_male Male population aged 0-7
198 0_6_female Female population aged 0-8
199 7_14_all Population aged 7-14
200 7_14_male Male population aged 7-14
201 7_14_female Female population aged 7-14
202 0_17_all Population aged 0-17
203 0_17_male Male population aged 0-17
204 0_17_female Female population aged 0-17
205 16_29_all Population aged 16-19
206 16_29_male Male population aged 16-19
207 16_29_female Female population aged 16-19
208 0_13_all Population aged 0-13
209 0_13_male Male population aged 0-13
210 0_13_female Female population aged 0-13
211 raion_build_count_with_material_info Number of building with material info in district
212 build_count_block Share of block buildings

```

213 build_count_wood    Share of wood buildings
214 build_count_frame   Share of frame buildings
215 build_count_brick   Share of brick buildings
216 build_count_monolith Share of monolith buildings
217 build_count_panel   Share of panel buildings
218 build_count_foam    Share of foam buildings
219 build_count_slag    Share of slag buildings
220 build_count_mix     Share of mixed buildings
221 raion_build_count_with_builddate_info  Number of building with build year info in district
222 build_count_before_1920  Share of before_1920 buildings
223 build_count_1921-1945    Share of 1921-1945 buildings
224 build_count_1946-1970    Share of 1946-1970 buildings
225 build_count_1971-1995    Share of 1971-1995 buildings

```

ПРИЛОЖЕНИЕ Б

```

1  ###
2  import warnings
3  warnings.filterwarnings('ignore')
4  import matplotlib as mlp
5
6  mlp.rcParams['lines.linewidth'] = 5
7  mlp.rcParams['xtick.major.size'] = 20
8  mlp.rcParams['xtick.major.width'] = 5
9  mlp.rcParams['xtick.labelsize'] = 20
10 mlp.rcParams['xtick.color'] = '#FF5533'
11
12 mlp.rcParams['ytick.major.size'] = 20
13 mlp.rcParams['ytick.major.width'] = 5
14 mlp.rcParams['ytick.labelsize'] = 20
15 mlp.rcParams['ytick.color'] = '#FF5533'
16
17 mlp.rcParams['axes.labelsize'] = 20
18 mlp.rcParams['axes.titlesize'] = 20
19 mlp.rcParams['axes.titlecolor'] = '#00B050'
20 mlp.rcParams['axes.labelcolor'] = '#00B050'
21 ###
22 import pandas as pd
23 pd.options.display.max_columns = 500
24
25 df = pd.read_csv("train.csv")
26 ###
27 # Эти колонки заранее удалены т.к. эти id описывают местоположение квартиры, а у нас и так есть такие
   ↪ колонки
28

```

```

29 df = df.drop(['ID_metro',
30   'ID_railroad_station_walk',
31   'ID_railroad_station_avto',
32   'ID_big_road1',
33   'ID_big_road2',
34   'ID_railroad_terminal',
35   'ID_bus_terminal'], axis=1)
36 ###
37 df = df.drop('id', axis=1)
38
39 print(df.shape)
40
41 df.head()
42 ### md
43 ### В соревновании использовали метрику - RMSLE. По факту, это корень от MSLE, поэтому неважно, какую
44 ↪ из них оптимизировать!
45 ###
46
47 import numpy as np
48
49 df = df.assign(log_price_doc=np.log1p(df['price_doc']))
50 df = df.drop('price_doc', axis=1)
51 ###
52 df
53 ### md
54 ### Работа с пропусками и и элементы EDA
55 ###
56 ### Посмотрим на некатегориальные колонки
57
58 numeric_columns = df.loc[:,df.dtypes!=np.object].columns
59
60 df.describe()
61 ###
62 ### Заполним средним
63
64 for col in numeric_columns:
65     df[col] = df[col].fillna(df[col].mean())
66 ###
67 ### Изучим корреляции вещественных признаков
68
69 df[numeric_columns].corr()
70 ###
71 ### Отфильтруем признаков
72
73 def get_redundant_pairs(df):
74     pairs_to_drop = set()
75     cols = df.columns
76     for i in range(0, df.shape[1]):

```



```

75         for j in range(0, i+1):
76             pairs_to_drop.add((cols[i], cols[j]))
77     return pairs_to_drop
78
79 def get_top_abs_correlations(df, n=5):
80     au_corr = df.corr().abs().unstack()
81     labels_to_drop = get_redundant_pairs(df)
82     au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
83     return au_corr[0:n]
84
85 print("Top Absolute Correlations")
86 print(get_top_abs_correlations(df[numeric_columns], 50))
87 ###
88 ### Удалим колонки, где корреляция оказывается > 0.9
89
90 def correlation(dataset, threshold):
91     col_corr = set() # Set of all the names of deleted columns
92     corr_matrix = dataset.corr()
93     for i in range(len(corr_matrix.columns)):
94         for j in range(i):
95             if (corr_matrix.iloc[i, j] >= threshold) and (corr_matrix.columns[j] not in col_corr):
96                 colname = corr_matrix.columns[i] # getting the name of column
97                 col_corr.add(colname)
98                 if colname in dataset.columns:
99                     del dataset[colname] # deleting the column from the dataset
100
101 correlation(df, 0.9)
102 ###
103 numeric_columns = df.loc[:,df.dtypes!=np.object].columns
104
105 df.shape
106 ###
107 ### Посмотрим на квазиконстантные признаки
108
109 from sklearn.feature_selection import VarianceThreshold
110
111
112 cutter = VarianceThreshold(threshold=0.1)
113 cutter.fit(df[numeric_columns])
114 constant_cols = [x for x in numeric_columns if x not in cutter.get_feature_names_out()]
115
116 df[constant_cols]
117 ###
118 ### Посмотрим на категориальные колонки
119
120 categorical_columns = df.loc[:,df.dtypes==np.object].columns
121

```

```

122 ### Изучим их
123
124 df.describe(include='object')
125 ###
126 ### Преобразуем категориальные колонки
127
128 for col in categorical_columns:
129     if col != 'timestamp':
130         if df[col].nunique() < 5:
131             one_hot = pd.get_dummies(df[col], prefix=col, drop_first=True)
132             df = pd.concat((df.drop(col, axis=1), one_hot), axis=1)
133
134         else:
135             mean_target = df.groupby(col)['log_price_doc'].mean()
136             df[col] = df[col].map(mean_target)
137 ###
138 df.head()
139 ###
140 ### Изменим вид даты
141
142 df['timestamp'] = pd.to_datetime(df['timestamp'])
143
144 df['month'] = df.timestamp.dt.month
145 df['year'] = df.timestamp.dt.year
146 ###
147 df.head()
148 ### Уберем timestamp
149
150 df = df.drop('timestamp', axis=1)
151 ###
152 ### Отделим таргеты от объектов
153
154 X = df.drop('log_price_doc', axis=1)
155 Y = df['log_price_doc']
156 ### md
157 ## Построим пару базовых моделей в качестве бэйзлайна
158 ### md
159
160 Теперь наши данные обладают временной структурой. Поэтому, чтобы получить хорошую обобщающую
    ↳ способность, мы хотим построить не просто модель, хорошо работающую на новых данных, а модель,
    ↳ которая угадывает распределение данных в будущем хотя бы на коротком горизонте. Поэтому, когда мы
    ↳ валидируем дизайн модели, нам важно делить на каждом шаге трейн и тест таким образом, чтобы по
    ↳ временной шкале они не пересекались, и точки из второго множества появлялись позже точек из
    ↳ первого.
161
162 ###
163 ### Разделим выборку на валидацию и тест

```

```

164
165 from sklearn.model_selection import TimeSeriesSplit
166
167 splitter = TimeSeriesSplit(n_splits=4)
168
169 ###
170 alphas = np.linspace(start=0.01, stop=1, num=30)
171 alphas
172 ###
173 from sklearn.model_selection import GridSearchCV
174
175 param_grid = {
176     "Lasso__alpha": alphas
177 }
178
179 ### Передадим в GridSearchCV
180
181 search = GridSearchCV(pipe, param_grid,
182                       cv=splitter, scoring='neg_mean_squared_error')
183
184 search.fit(X, Y)
185
186 print(f"Best parameter (CV score={search.best_score_:.5f}):")
187 print(search.best_params_)
188 ###
189 ### Убедимся, что все ок!
190
191 pipe.set_params(Lasso__alpha=search.best_params_['Lasso__alpha'])
192 ###
193 cv_result_pipe = cross_validate(pipe, X, Y,
194                                 scoring='neg_mean_squared_error',
195                                 cv=splitter, return_train_score=True)
196
197 print(f"Среднее MSLE на тренировочных фолдах: {-np.mean(cv_result_pipe['train_score']).round(3)}")
198 print(f"Среднее MSLE на тестовых фолдах: {-np.mean(cv_result_pipe['test_score']).round(3)}")
199 ### md
200 ### Анализ выбросов
201 ###
202 data = pd.concat((X, Y), axis=1)
203 ###
204 top_quantile = data['log_price_doc'].quantile(0.975)
205 low_quantile = data['log_price_doc'].quantile(0.025)
206
207 print(f"Топ 2,5% значение таргета: {top_quantile.round(2)}")
208 print(f"Топ 97,5% значение таргета: {low_quantile.round(2)}")
209 ###
210 ### Выбросим объекты со значениями вне отрезка [top 2,5%; top97,5%]

```

```

211
212 data = data[(data['log_price_doc']>low_quantile)&(data['log_price_doc']<top_quantile)]
213
214 X_new, Y_new = data.drop('log_price_doc', axis=1), data['log_price_doc']
215 ###
216 ### Как подобрать коэффициент регуляризации?
217
218 new_splitter = TimeSeriesSplit(n_splits=4)
219
220 param_grid = {
221     "Lasso__alpha": alphas
222 }
223
224 ### Передадим в GridSearchCV
225
226 search = GridSearchCV(pipe, param_grid,
227                       cv=new_splitter, scoring='neg_mean_squared_error')
228
229 search.fit(X_new, Y_new)
230
231 print(f"Best parameter (CV score={search.best_score_:.5f}):")
232 print(search.best_params_)
233 ###
234 ### Убедимся, что все ок!
235
236 pipe.set_params(Lasso__alpha=search.best_params_['Lasso__alpha'])
237 ###
238 cv_result_pipe = cross_validate(pipe, X_new, Y_new,
239                                 scoring='neg_mean_squared_error',
240                                 cv=splitter, return_train_score=True)
241
242 print(f"Среднее MSLE на тренировочных фолдах: {-np.mean(cv_result_pipe['train_score']).round(3)}")
243 print(f"Среднее MSLE на тестовых фолдах: {-np.mean(cv_result_pipe['test_score']).round(3)}")
244 ### md
245 ### Сегментация данных
246
247 ###
248 ### Разделим квартиры по типу недвижимости
249 ### Для первички и вторички будем строить разные модели
250
251 Owner_Occupier = data[data['product_type_OwnerOccupier'] == 1].copy()
252 Investment = data[data['product_type_OwnerOccupier'] == 0].copy()
253 ###
254 X_Occupier = Owner_Occupier.drop('log_price_doc', axis=1)
255 X_Investment = Investment.drop('log_price_doc', axis=1)
256
257 Y_Occupier = Owner_Occupier['log_price_doc']

```

```

258 Y_Investment = Investment['log_price_doc']
259 ###
260 ### Соберем модель для Owner_Occupier
261
262 search_Owner_Occupier = GridSearchCV(pipe, param_grid,
263                                     cv=splitter, scoring='neg_mean_squared_error')
264
265 search_Owner_Occupier.fit(X_Occupier, Y_Occupier)
266
267 print(f"Best parameter (CV score={search_Owner_Occupier.best_score_:.5f}):")
268 print(search_Owner_Occupier.best_params_)
269
270 pipe.set_params(Lasso__alpha=search_Owner_Occupier.best_params_['Lasso__alpha'])
271
272 cv_result_pipe = cross_validate(pipe, X_Occupier, Y_Occupier,
273                                 scoring='neg_mean_squared_error',
274                                 cv=splitter, return_train_score=True)
275
276 error_Occupier_train = -np.mean(cv_result_pipe['train_score'])
277 error_Occupier_test = -np.mean(cv_result_pipe['test_score'])
278
279 print(f"Среднее MSLE на тренировочных фолдах: {error_Occupier_train.round(3)}")
280 print(f"Среднее MSLE на тестовых фолдах: {error_Occupier_test.round(3)}")
281 ###
282 ### Соберем модель для Investment
283
284 search_Investment = GridSearchCV(pipe, param_grid,
285                                  cv=splitter, scoring='neg_mean_squared_error')
286
287 search_Investment.fit(X_Investment, Y_Investment)
288
289 print(f"Best parameter (CV score={search_Investment.best_score_:.5f}):")
290 print(search_Investment.best_params_)
291
292 pipe.set_params(Lasso__alpha=search_Investment.best_params_['Lasso__alpha'])
293
294 cv_result_pipe = cross_validate(pipe, X_Investment, Y_Investment,
295                                 scoring='neg_mean_squared_error',
296                                 cv=splitter, return_train_score=True)
297
298 error_Investment_train = -np.mean(cv_result_pipe['train_score'])
299 error_Investment_test = -np.mean(cv_result_pipe['test_score'])
300
301 print(f"Среднее MSLE на тренировочных фолдах: {error_Investment_train.round(3)}")
302 print(f"Среднее MSLE на тестовых фолдах: {error_Investment_test.round(3)}")
303 ###
304 ### Перевзвесим скоры с учетом количества объектов

```

```

305 ### в обоих типах жилья
306
307 n_Occupier = Owner_Occupier.shape[0]
308 n_Investment = Investment.shape[0]
309
310 ## Посчитаем доли категорий в общей выборке
311
312 share_Occupier = n_Occupier / data.shape[0]
313 share_Investment = n_Investment / data.shape[0]
314
315 weighted_error_train = share_Occupier * error_Occupier_train + \
316                         share_Investment * error_Investment_train
317
318 weighted_error_test = share_Occupier * error_Occupier_test + \
319                       share_Investment * error_Investment_test
320
321 print(f"Среднее взвешенное MSLE на тренировочных фолдах: {weighted_error_train.round(3)}")
322 print(f"Среднее взвешенное MSLE на тестовых фолдах: {weighted_error_test.round(3)}")

```
