Государственное бюджетное профессиональное
образовательное учреждение Московской области
«Физико-технический колледж»

Аналитический отчет

«Анализ Циана»

Работу выполнила:
Студентка группы ИСП-22
Лесницкая Татьяна

Долгопрудный, 2024

ВВЕДЕНИЕ

Цена на недвижимость является динамичным и многогранным показателем, на который влияют разнообразные факторы.

Основной целью данного анализа является выявление ключевых тенденций роста цен на недвижимость и определение параметров, которые наиболее сильно влияют на цену квартир. Это позволит не только сформировать более точные прогнозы будущих изменений, но и предоставить полезную информацию для принятия обоснованных решений всеми участниками рынка.

Постановка задачи

Цель: Собрать данные и провести разведочный исследовательский анализ данных (EDA) для построения модели, которая будет оценивать цену квадратного метра недвижимости в Московском регионе (Москва, Московская область).

Актуальность:

Изменения в цене за квадратный метр затрагивают широкий круг заинтересованных сторон, включая обычных покупателей, собственников, застройщиков и строительные компании. В условиях постоянно меняющегося рынка важно понимать текущие тенденции и факторы, которые оказывают наибольшее влияние на стоимость жилья.

Задачи:

- Составить список параметров, значительно влияющих на цену квадратного метра жилой площади
- Произвести парсинг с сайта циана с помощью библиотеки cianparser
- Подготовить данные для анализа: проверить на пропуски, выбросы и ошибки. Обработать выявленные аномалии (удалить / заполнить)
- Проведите Исследовательский Анализ Данных (EDA). Построить распределение основных параметров; визуализировать взаимосвязи

между ними; определить признаки, оказывающие наиболее сильное влияние на целевую переменную (цену за квадратный метр)

ВЛИЯЮЩИЕ ПАРАМЕТРЫ

Таких параметров существует много. Цены могут сильно различаться в зависимости от города, самой площади, типа жилья, близости от метро, улицы и количества комнат.

СБОР ДАННЫХ

Данные были собраны циана, так как это самая популярная платформа по продаже жилья.

При парсинге использовалась библиотека cianparser, но сам код библиотеки был изменен в одном файле под названием page.py, в папке flat, которая является методом который я использовала когда парсила данные.

Такое решение обусловлено тем, что сама библиотека в какой-то мере устарела. Часть данных, которые указаны на сайте, а именно: object_type, parking_type, have_loggia не было. Структура сайта меняется, в какой-то мере.

Программа где выполнялся парсинг: Pycharm

Измененный код библиотеки page.py:

```
import bs4
import re
import time
import random


class FlatPageParser:
    def __init__(self, session, url):
        self.session = session
        self.url = url

    def __load_page__(self):
        res = self.session.get(self.url)
```

```
        if res.status_code == 429:
            time.sleep(10)
        res.raise_for_status()
        self.offer_page_html = res.text
        self.offer_page_soup = bs4.BeautifulSoup(self.offer_page_html,
'html.parser')

    def __parse_flat_offer_page_json__(self):
        page_data = {
            "year_of_construction": -1,
            "object_type": -1,
            # "ceiling_height": -1,
            # "ceiling_height": -1,
            "have_loggia": -1,
            "parking_type": -1,
            "house_material_type": -1,
            "heating_type": -1,
            "finish_type": -1,
            "living_meters": -1,
            "kitchen_meters": -1,
            "floor": -1,
            "floors_count": -1,
            "phone": "",
        }
        ot =
self.offer_page_soup.select_one('[data-name="OfferSummaryInfoItem"]
p:nth-of-type(2)').get_text()
        page_data["object_type"] = ot
        time.sleep(random.uniform(0, 5))
        # ch =
self.offer_page_soup.select_one('[data-name="OfferSummaryInfoItem"]
p:nth-of-type(10)').get_text()
        # page_data["ceiling_height"] = ch
        # it =
self.offer_page_soup.select_one('[data-name="OfferSummaryInfoItem"]
p:nth-of-type(10)').get_text()
        # page_data["ceiling_height"] = it
        # et =
self.offer_page_soup.select_one('[data-name="OfferSummaryInfoItem"]
p:nth-of-type(14)').get_text()
        # page_data["have_loggia"] = et
```

```
        pt_elements =
self.offer_page_soup.select('[data-name="OfferSummaryInfoItem"] p')
        for i, p_element in enumerate(pt_elements):
            if "Парковка" in p_element.get_text():
                parking_type_element = pt_elements[i + 1]
                print(i)
                page_data["parking_type"] =
parking_type_element.get_text()
                time.sleep(random.uniform(0, 5))
                break
        else:
            page_data["parking_type"] = -1

        hl_elements =
self.offer_page_soup.select('[data-name="OfferSummaryInfoItem"] p')
        for i, hl_element in enumerate(hl_elements):
            if "Балкон/лоджия" in hl_element.get_text():
                have_loggia_element = hl_elements[i + 1]
                print(i)
                page_data["have_loggia"] =
have_loggia_element.get_text()
                time.sleep(random.uniform(0, 5))
                break
        else:
            page_data["have_loggia"] = -1
        ch_elements =
self.offer_page_soup.select('[data-name="OfferSummaryInfoItem"] p')
        for i, ch_element in enumerate(ch_elements):
            if "Высота потолков" in ch_element.get_text():
                ceiling_height_element = ch_elements[i + 1]
                print(i)
                page_data["ceiling_height"] =
ceiling_height_element.get_text()
                time.sleep(random.uniform(0, 5))
                break
        else:
            page_data["ceiling_height"] = -1


        # et =
self.offer_page_soup.select_one('[data-name="OfferSummaryInfoItem"]
p:nth-of-type(2)').get_text()
        # page_data["heating_type"] = et
```

```
        spans = self.offer_page_soup.select("span")
        for index, span in enumerate(spans):
            # if "Тип жилья" == span.text:
            #     page_data["object_type"] = spans[index + 1].text

            if "Тип дома" == span.text:
                page_data["house_material_type"] = spans[index + 1].text
                time.sleep(random.uniform(0, 5))

            # if "Отопление" == span.text:
            #     page_data["heating_type"] = spans[index + 1].text

            if "Отделка" == span.text:
                page_data["finish_type"] = spans[index + 1].text
                time.sleep(random.uniform(0, 5))



            if "Площадь кухни" == span.text:
                page_data["kitchen_meters"] = spans[index + 1].text
                time.sleep(random.uniform(0, 5))

            if "Жилая площадь" == span.text:
                page_data["living_meters"] = spans[index + 1].text
                time.sleep(random.uniform(0, 5))

            if "Год постройки" in span.text:
                page_data["year_of_construction"] = spans[index +
1].text
                time.sleep(random.uniform(0, 5))

            if "Год сдачи" in span.text:
                page_data["year_of_construction"] = spans[index +
1].text
                time.sleep(random.uniform(0, 5))

            if "Этаж" == span.text:
                ints = re.findall(r'\d+', spans[index + 1].text)
                if len(ints) == 2:
                    page_data["floor"] = int(ints[0])
                    page_data["floors_count"] = int(ints[1])
```

```
                time.sleep(random.uniform(0, 5))

        if "+7" in self.offer_page_html:
            page_data["phone"] =
self.offer_page_html[self.offer_page_html.find("+7"):
self.offer_page_html.find("+7") + 16].split('"')[0]. \
                replace(" ", ""). \
                replace("-", "")
            time.sleep(random.uniform(0, 5))

        return page_data

    def parse_page(self):
        self.__load_page__()
        return self.__parse_flat_offer_page_json__()
```

Код файла parser.py, в котором использовалась измененная библиотека

```
import cianparser

# формат кода

locations = [ 'Москва', 'Черноголовка', 'Одинцово', 'Электросталь',
'Щёлково',
    'Дрезна', 'Клин', 'Егорьевск', 'Высоковск', 'Лыткарино', 'Чехов',
    'Хотьково', 'Сергиев Посад', 'Павловский Посад', 'Красногорск',
    'Химки', 'Дмитров', 'Яхрома', 'Долгопрудный', 'Троицк', 'Балашиха',
    'Подольск', 'Мытищи', 'Люберцы', 'Королёв', 'Домодедово',
'Серпухов',
    'Коломна', 'Раменское', 'Реутов', 'Пушкино', 'Жуковский', 'Видное',
    'Орехово-Зуево', 'Ногинск', 'Воскресенск', 'Ивантеевка', 'Лобня',
    'Дубна', 'Котельники', 'Фрязино', 'Дзержинский', 'Краснознаменск',
    'Кашира', 'Звенигород', 'Истра', 'Красноармейск', 'Волоколамск',
    'Озёры', 'Кубинка', 'Пущино', 'Талдом', 'Руза', 'Краснозаводск',
    'Пересвет', 'Можайск']

for location in locations:
    parser = cianparser.CianParser(location=location)

    data = parser.get_flats(deal_type='sale', rooms=(1, 2, 3, 4),
with_saving_csv=True, additional_settings={'start_page': 1, 'end_page':
50})
```

В этом коде могли меняться количество комнат, начальная и конечная страница а также города.

После того как данные спарсились, можно приступать к очистке и анализу

АНАЛИЗ ДАННЫХ

ЦИАН

Далее я перехожу в google collab. Я подключаю гугл диск и загружаю файл своей таблицы с данными циана.

Смотрю информацию о датасете и колонках, вывожу первые строки. Перевожу все колонки с английского на русский для наглядности.

Затем проверяю на дубликаты и так как они все-таки оказались я их удаляю.

```
[ ]  df.drop_duplicates(inplace=True)
     print(f'Осталось {df.shape[0]} строчек ')
```

```
Осталось 9575 строчек
```

Смотрю на нули, заменяю “-1”на np.nan чтобы видеть все пропущенные значения.

```
for column in df.columns:
  df[column].replace('-1', np.nan, inplace=True)
```

Смотрю на них по колонкам в процентном отношении. Удаляю в связи с этим колонки ‘heating_type’ и ‘house_material_type’, а также house_number, url и phone поскольку они не имеют влияния на анализ.

```
useless_columns = ['house_material_type', 'heating_type', 'finish_type', 'house_number', 'url', 'phone']
for column in useless_columns:
  df.drop([column], axis=1, inplace=True)
```

Дальше мне необходимо преобразовать часть данных из object в float, чтобы я могла посмотреть на выбросы и для дальнейших вычислений.

```
[ ]  possible_outliers = df[['floor', 'floors_count', 'rooms_count', 'total_meters', 'price', 'year_of_construction', 'living_meters', 'kitchen_meters', 'ceili

     possible_outliers.describe()
```

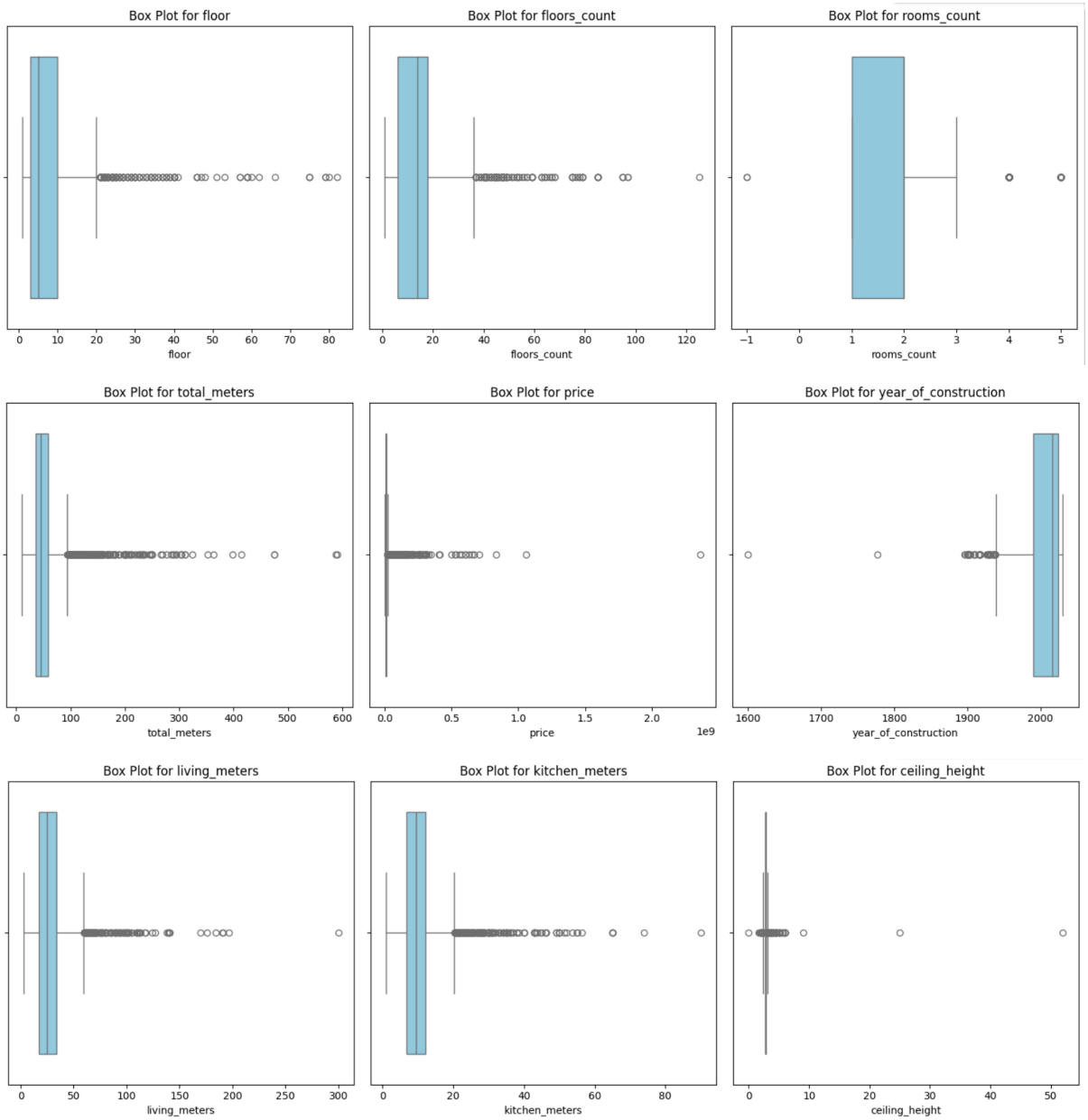
Здесь я создала копию основного датафрейма и передала в список все числовые значения, так как в них могут быть выбросы.

	floor	floors_count	rooms_count	total_meters	price	year_of_construction	living_meters	kitchen_meters	ceiling_height
count	9573.000000	9573.000000	9477.000000	9573.000000	9.560000e+03	6209.000000	5875.000000	6299.000000	4912.000000
mean	7.326126	13.890630	1.703598	50.775833	1.362327e+07	2006.099372	27.437719	10.538324	2.828031
std	6.501501	9.094196	0.781568	27.474646	4.160334e+07	23.984940	15.643088	5.857255	0.826707
min	1.000000	1.000000	-1.000000	10.800000	8.300000e+05	1600.000000	3.000000	1.000000	0.000000
25%	3.000000	6.000000	1.000000	36.000000	5.400000e+06	1990.000000	17.000000	6.700000	2.700000
50%	5.000000	14.000000	2.000000	44.700000	7.750000e+06	2016.000000	25.000000	9.500000	2.750000
75%	10.000000	18.000000	2.000000	59.200000	1.090000e+07	2024.000000	34.000000	12.100000	2.900000
max	82.000000	125.000000	5.000000	590.300000	2.361200e+09	2031.000000	300.000000	90.000000	52.000000

Пока я не выводила boxplot чтобы явно посмотреть на диапазон выбросов, но уже по этой таблице можно заметить аномальные значения, такие как 2031 год, количество этажей 125 или высота потолка 52 м.

Дальше я понимаю, что выбросы действительно есть во всех колонках:

```
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))

# Создание box plots для каждого столбца
for i, column in enumerate(possible_outliers.columns):
    row = i // 3
    col = i % 3
    sns.boxplot(x=possible_outliers[column], ax=axes[row, col], color='skyblue')
    axes[row, col].set_title(f'Box Plot for {column}')


plt.subplots_adjust(hspace=0.9)
plt.tight_layout()
plt.show()
```

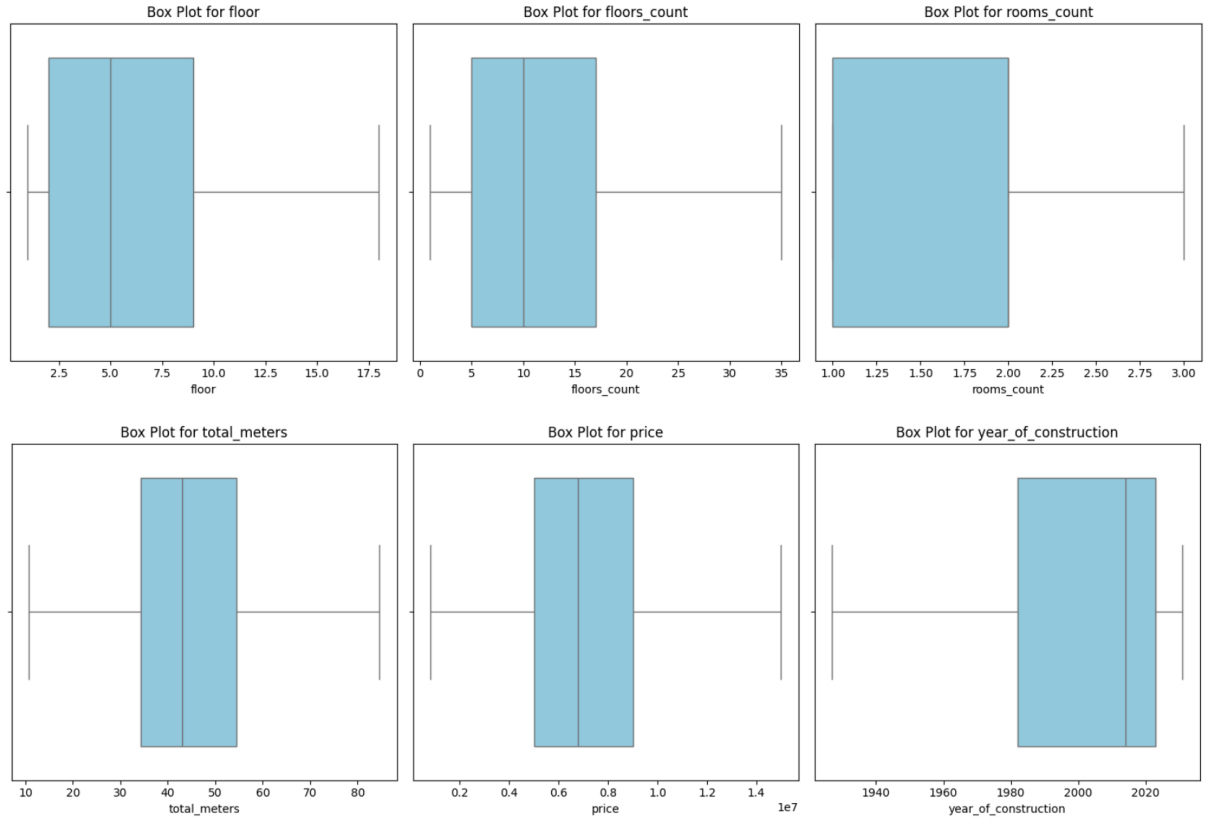


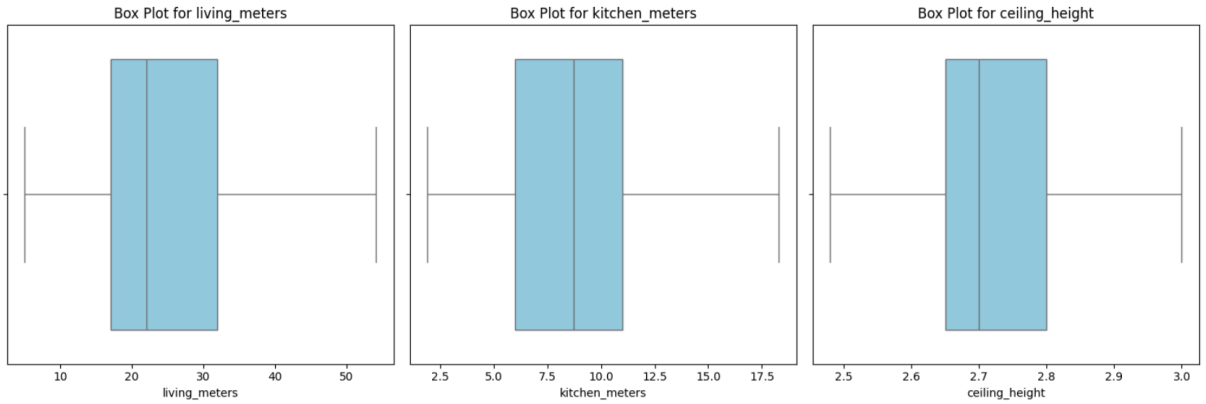
Все то, что обведено в кружочек это выбросы. И судя по графикам, их очень много, что сильно повлияет на предсказание цены.

Я удаляю выбросы с помощью статистических методов, рассчитывая первый и третий квартиль, межквартильный размах и границы выбросов и опять проверяю по графиками чтобы все выбросы были удалены.

```python
def detect_outliers_iqr(possible_outliers, column):
    Q1 = possible_outliers[column].quantile(0.25)
    Q3 = possible_outliers[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = possible_outliers[(possible_outliers[column] < lower_bound) | (possible_outliers[column] > upper_bound)]
    return outliers

# функцияя для удаления выбросов
def remove_outliers(possible_outliers, column):
    outliers = detect_outliers_iqr(possible_outliers, column)
    possible_outliers = possible_outliers.drop(outliers.index)
    return possible_outliers

# список количественных переменных
quantitative_columns = ['total_meters', 'price', 'living_meters', 'kitchen_meters', 'ceiling_height']

# категориальные переменные
categorical_columns = ['floor', 'floors_count', 'rooms_count', 'year_of_construction']

# Визуализация выбросов
def visualize_outliers(possible_outliers, column):
    plt.figure(figsize=(10, 6))
    sns.boxplot(x=possible_outliers[column], color='skyblue')
    plt.title(f'Box Plot for {column}')
    plt.show()

# Определение и удаление выбросов
for column in quantitative_columns + categorical_columns:
    possible_outliers = remove_outliers(possible_outliers, column)
    visualize_outliers(possible_outliers, column)

# Вывод обработанных данных
print("Обработанные данные:")
print(possible_outliers)
```

Выбросы были успешно удалены:





Теперь мне надо только закрепить эти изменения

```
[ ]  remaining_indices = possible_outliers.index

     # Обновляем df, оставляя только строки, которые не были удалены как выбросы
     df = df.loc[remaining_indices]
```

Я возвращаюсь к заполнению пропущенных значений. Я разделяю колонки на числовые и категориальные.

В числовых заменяю пропущенные значения медианой, а в категориальных - где это возможно, заменяю модой или значением ‘unknown’

```
]  median = ['total_meters', 'living_meters', 'kitchen_meters',
              'ceiling_height', 'price']

   for m in median:
     df[m] = df[m].fillna(df[m].median())
```

```
]  columns_to_fill = ['floors_count', 'rooms_count', 'floor']


   for column in columns_to_fill:
       mode_value = df[column].mode()[0]
       df[column] = df[column].fillna(mode_value)
```

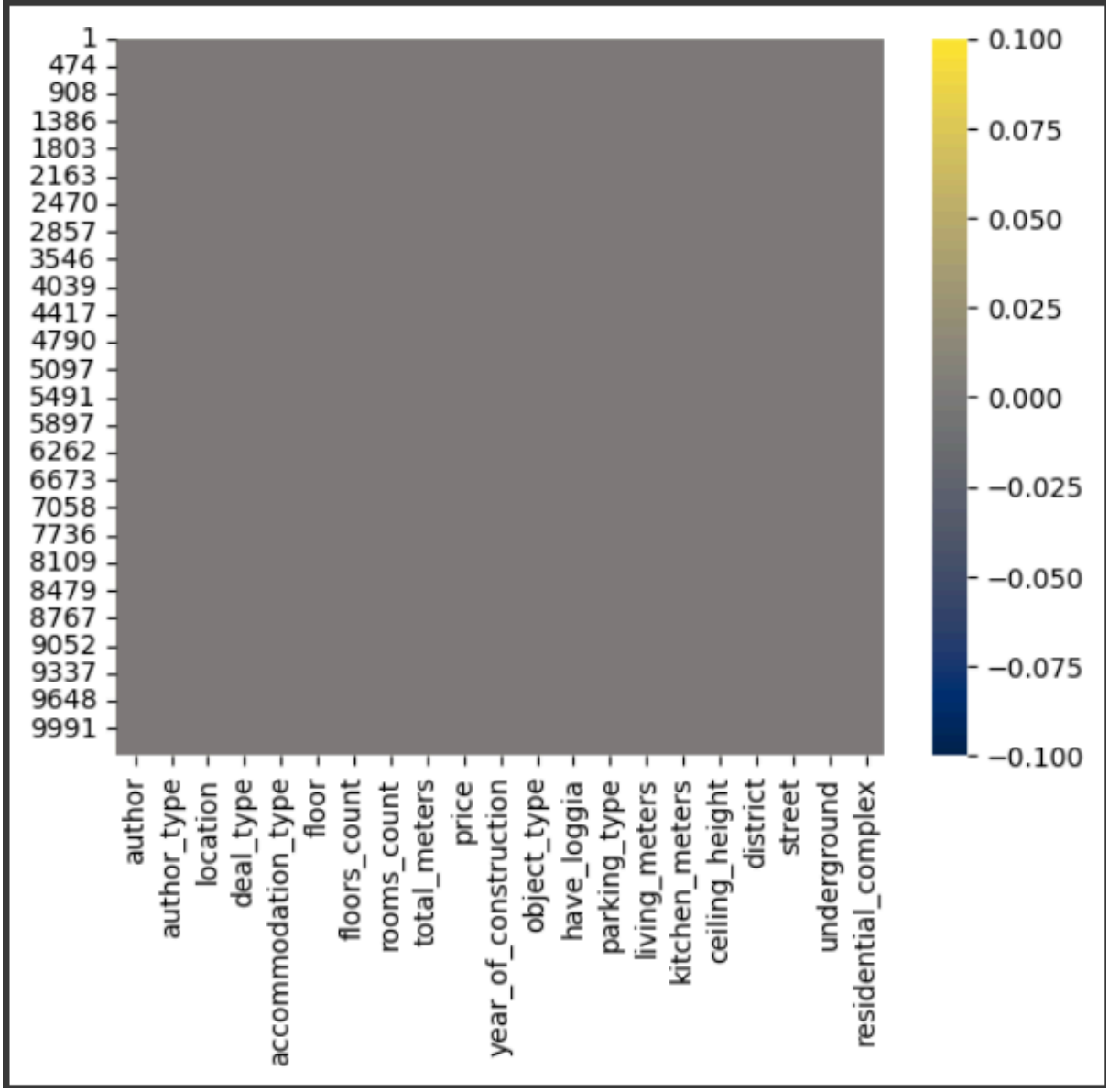
```
]  median_value = df['year_of_construction'].median()
   median_value = int(median_value)


   df['year_of_construction'] = df['year_of_construction'].fillna(median_value)
```

```
cat_columns = ['object_type', 'have_loggia', 'parking_type', 'district',
       'street', 'underground', 'residential_complex', 'author',
              'author_type', 'accommodation_type', 'deal_type', 'location']


for column in cat_columns:
     df[column] = df[column].apply(lambda x: 'unknown' if pd.isna(x) else x)
```

И смотрю на тепловую карту. Все что заполнено серым означает, что пропущенных значений не осталось

```
sns.heatmap(df.isnull(), cmap='cividis')
```

```
<Axes: >
```



Затем проверяю численно, что пропусков точно не осталось, и привожу часть колонок в тип int

```
to_int_columns = ['floor', 'floors_count', 'rooms_count', 'year_of_construction']


for columni in to_int_columns:
  df[columni] = df[columni].astype(int)
```

После очистки данных от дубликатов, выбросов, и приведения данных к нужному типу, я могу посчитать целевую переменную square_price

```
df['square_price'] = df['price']/df['total_meters']
df.head()
```

Затем сохраняю данные без дубликатов и выбросов в новый csv файл чтобы можно было строить графики в Power BI

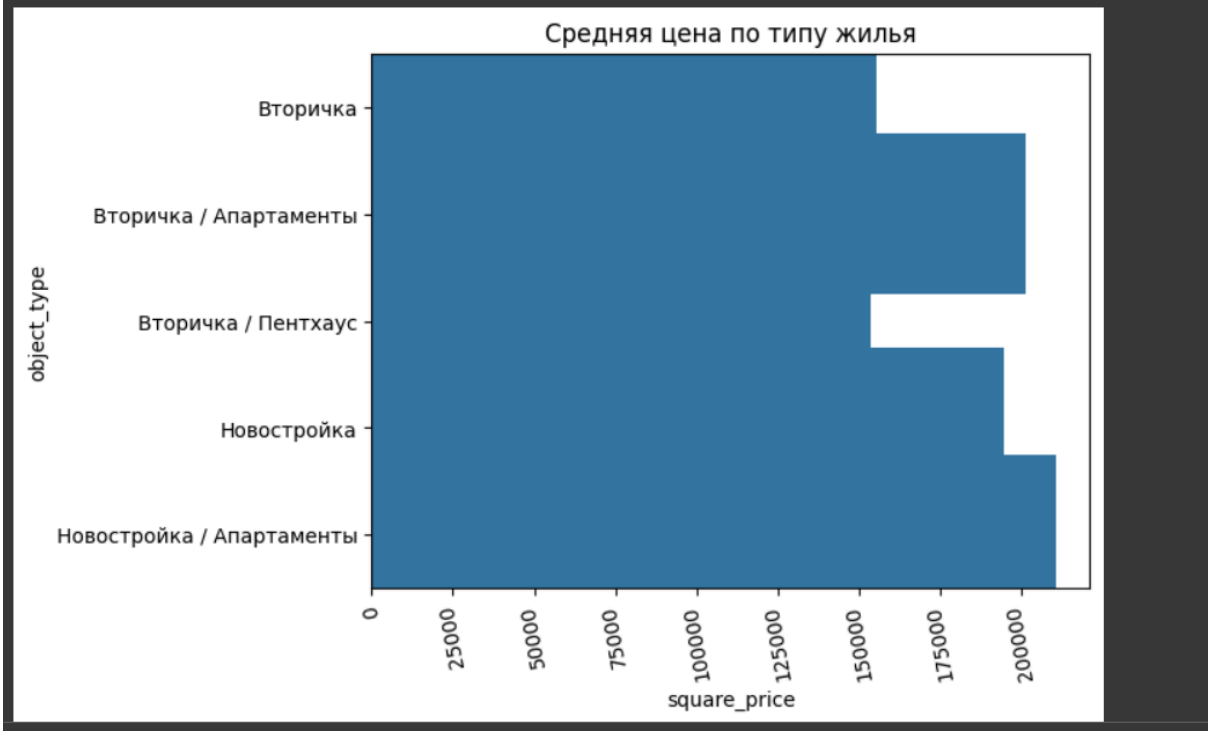
```
df.to_csv('data_without_outliers.csv')
```

ВИЗУАЛИЗАЦИЯ

Google Colab

Теперь у меня есть все для начала анализа.
График зависимости цены за квадратный метр от года постройки здания:

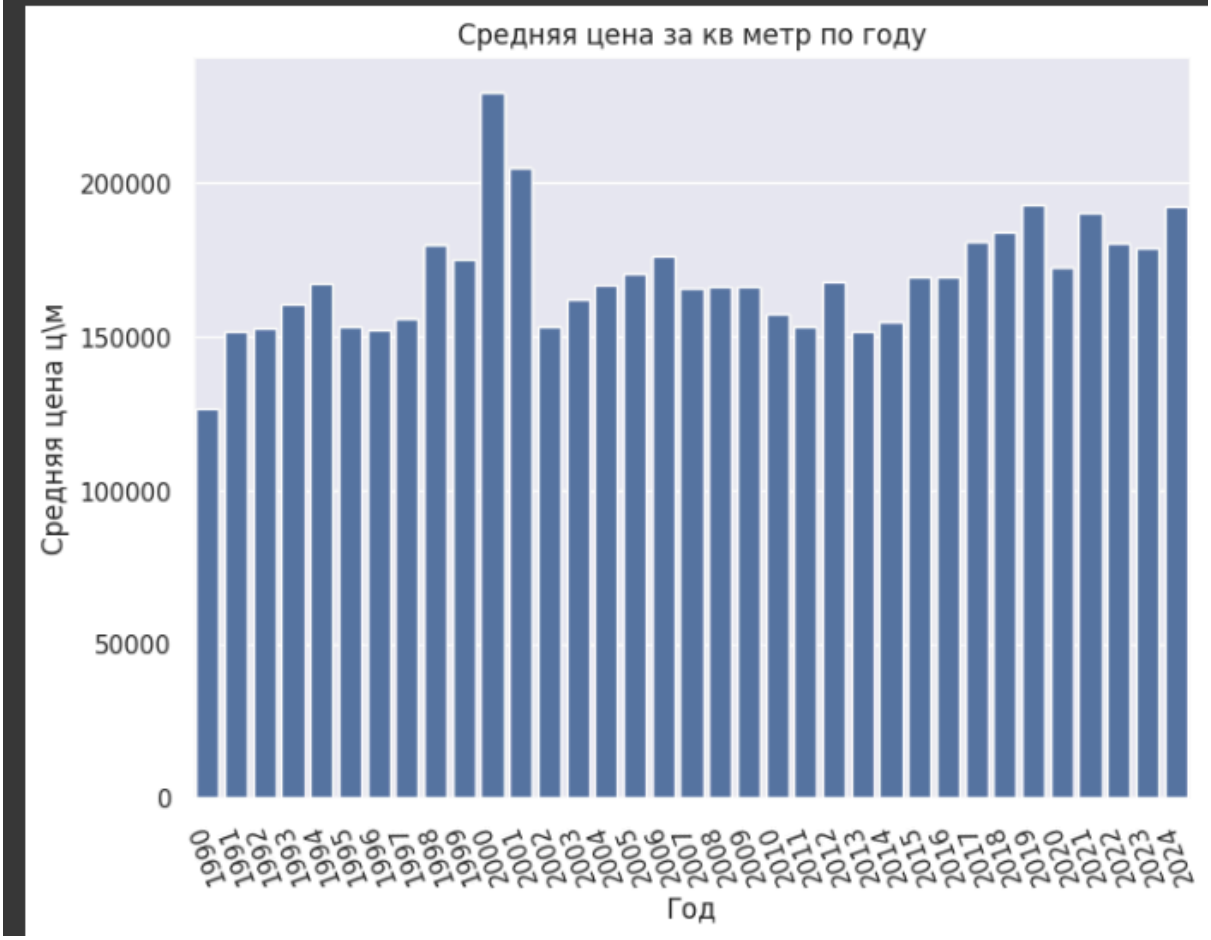
```
sns.barplot(x='square_price', y='object_type', data=ot, width=1.5)  # Увеличиваем ширину столбцов до 120%
plt.title('Средняя цена по типу жилья')
plt.xticks(rotation=100)  # Поворот подписей по оси X для лучшей читаемости
plt.show()
```



Из этого графика я делаю вывод, что дороже всего стоит тип жилья Новостройка / Апартаменты, а дешевле всего Вторички

График изменения цены за квадратный метр от года:

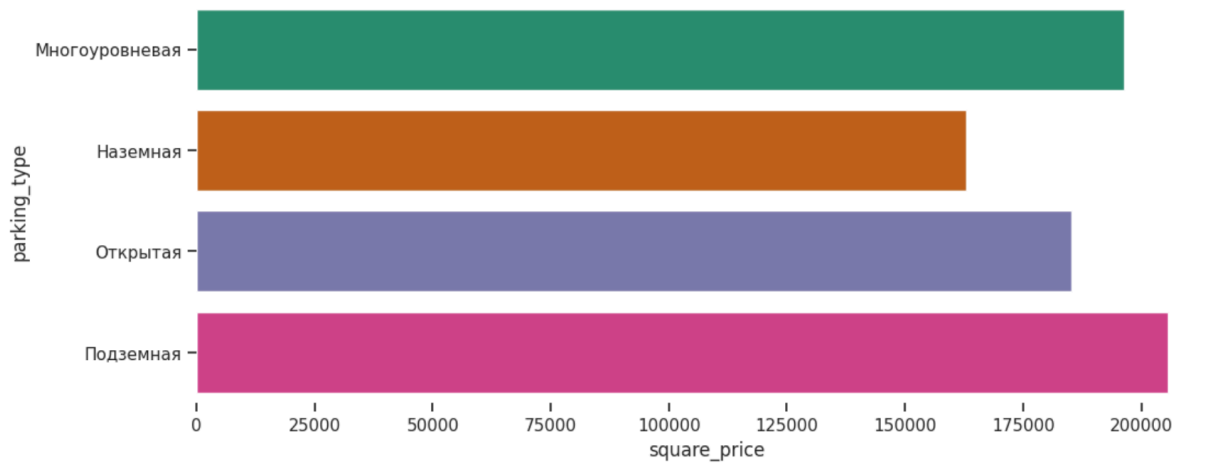
```
sns.set(style="darkgrid")
plt.figure(figsize=(8, 6))
sns.barplot(x='year_of_construction', y='square_price', data=average_prices)
plt.title('Средняя цена за кв метр по году')
plt.xlabel('Год')
plt.ylabel('Средняя цена ц\м')
plt.xticks(rotation=110)

plt.show()
```



Максимальное значение цены находится в 2000 году, а минимальное - в 1990

Смотрю какая парковка является самой дорогой:

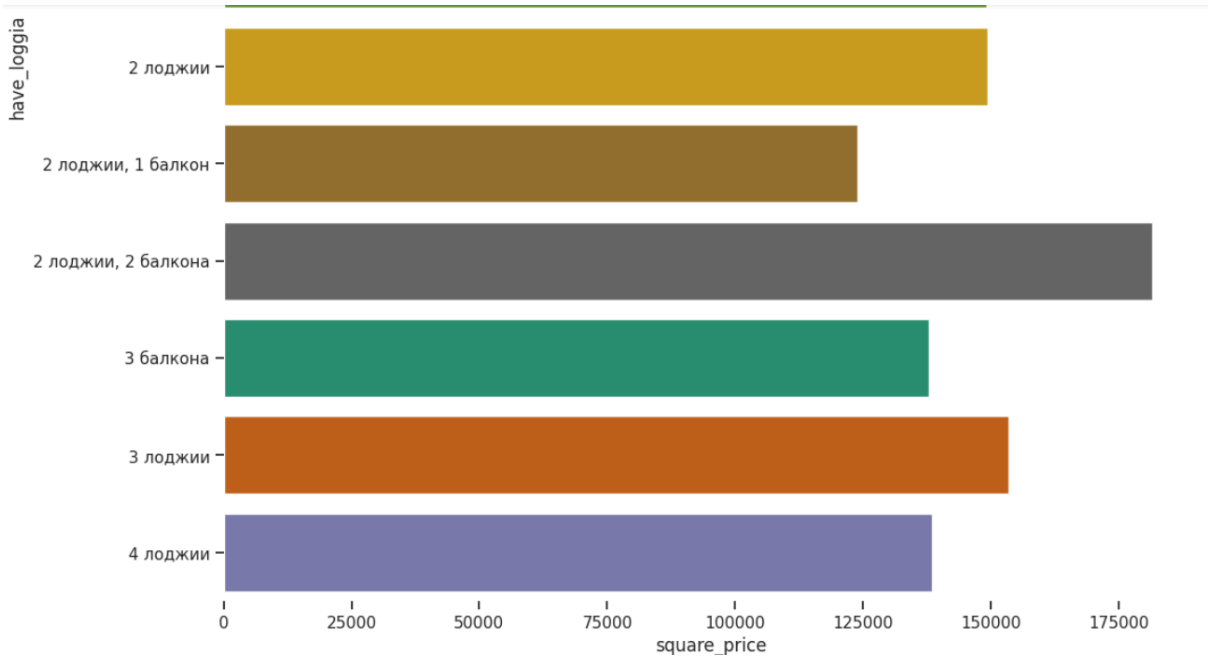
```
figsize = (12, 1.2 * len(parking['parking_type'].unique()))
plt.figure(figsize=figsize)


sns.barplot(data=parking, x='square_price', y='parking_type', palette='Dark2', estimator=np.mean)

# Удаление лишних осей
sns.despine(top=True, right=True, bottom=True, left=True)


plt.show()
```



Дороже всего подземная парковка, а самая дешевая - наземная

Наличие балкона или лоджии на цену:

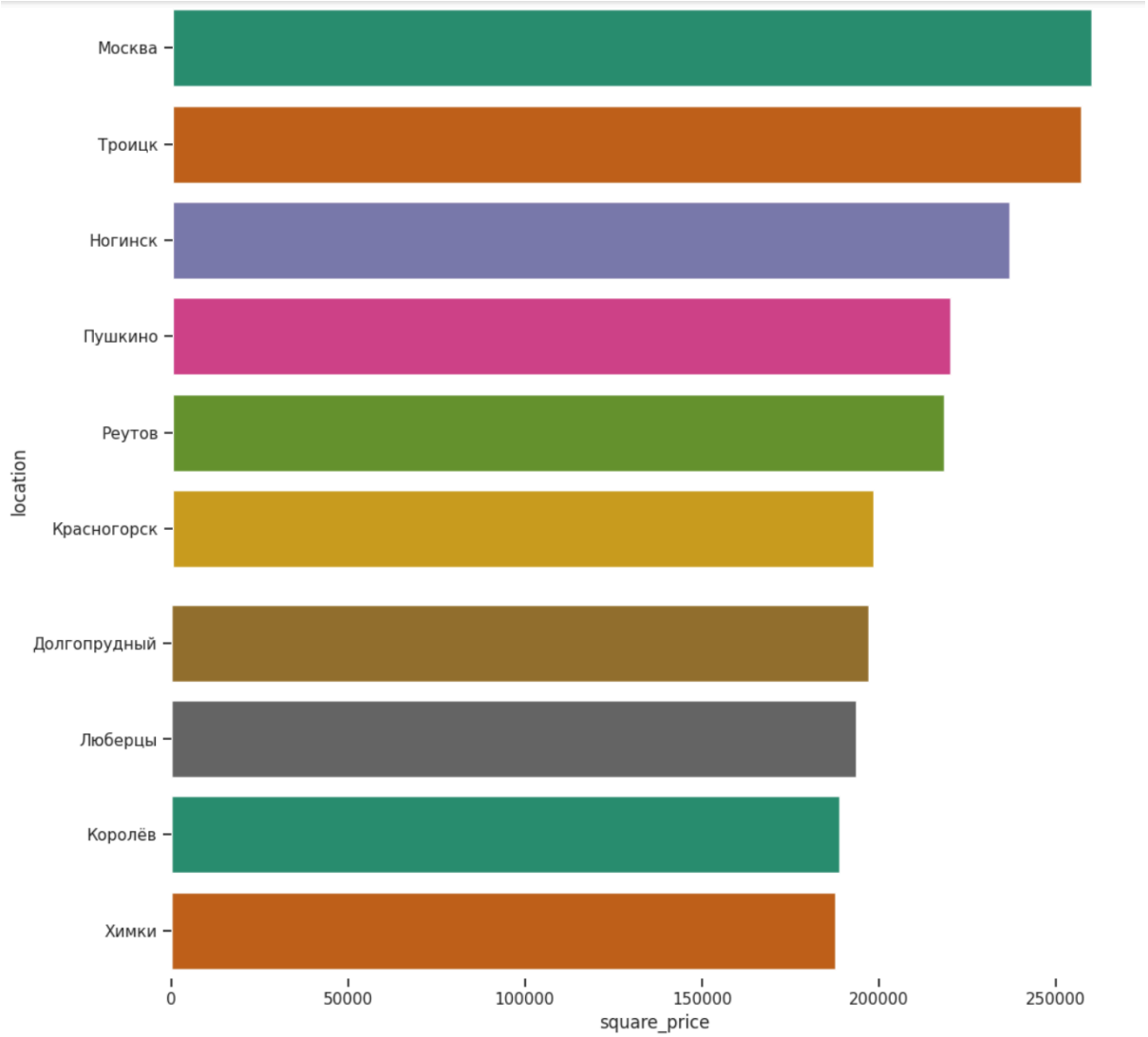
```
figsize = (12, 1.2 * len(logia['have_loggia'].unique()))
plt.figure(figsize=figsize)


sns.barplot(data=logia, x='square_price', y='have_loggia', palette='Dark2', estimator=np.mean)


sns.despine(top=True, right=True, bottom=True, left=True)


plt.show()
```



Балкон стоит в разы больше чем лоджия из-за больших размеров площади

Вывожу топ 10 городов по стоимости:

```
top_10_locations = locations.sort_values(by='square_price', ascending=False).head(10)


figsize = (12, 1.2 * len(top_10_locations))
plt.figure(figsize=figsize)

# топ 10 городов
sns.barplot(data=top_10_locations, x='square_price', y='location', palette='Dark2', estimator=np.mean)

sns.despine(top=True, right=True, bottom=True, left=True)
```



ЗАВИСИМОСТИ

Для того чтобы увидеть зависимости и корреляцию, нужно сначала закодировать данные в числовой формат

```
] from sklearn.preprocessing import OrdinalEncoder

  categorical_columns = df.select_dtypes(include=['object']).columns.tolist()
  ordinal_encoder = OrdinalEncoder()
  encoded_data = ordinal_encoder.fit_transform(df[categorical_columns])
  df[categorical_columns] = encoded_data.astype(int)
  df
```

	author	author_type	location	deal_type	accommodation_type	floor	floors_count	rooms_count	total_meters	price	...	h
2	701	5	31	1	1	4	4	1	14.0	1700000.0	...	
5	1883	4	31	1	1	2	3	1	46.7	6599999.0	...	
10	1839	5	31	1	1	1	5	2	42.8	2999000.0	...	
13	1955	4	31	1	1	1	3	1	25.3	2500000.0	...	
14	1581	5	31	1	1	2	3	1	36.0	2790000.0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
10384	1943	5	4	1	1	7	16	1	38.8	6290000.0	...	
10385	1985	5	4	1	1	4	12	2	53.6	9300000.0	...	

После вывожу корреляцию по целевому признаку square_price

```
corr_matrix = df.corr()['square_price'].round(2)
sort = corr_matrix.sort_values(ascending=False)
print(sort)
```

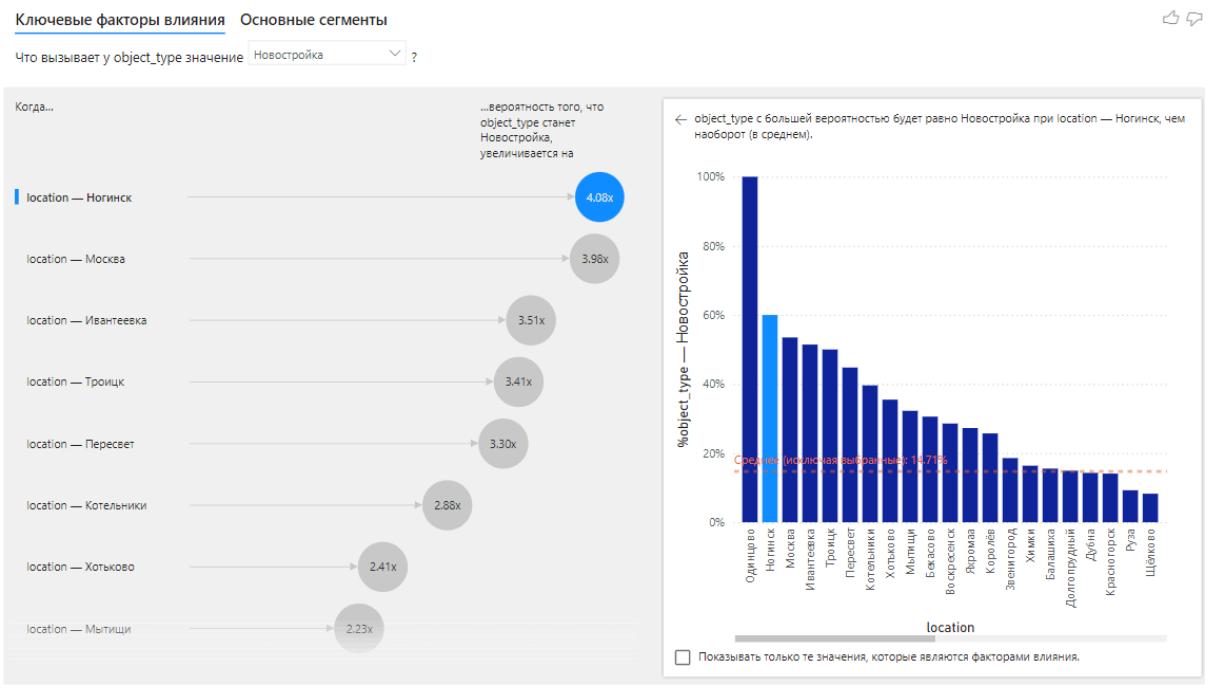
```
square_price            1.00
price                   0.60
floors_count            0.43
underground             0.35
floor                   0.24
residential_complex     0.20
ceiling_height          0.12
parking_type            0.12
year_of_construction    0.11
object_type             0.10
street                  0.08
have_loggia             0.06
kitchen_meters          0.01
accommodation_type      0.00
deal_type               0.00
location               -0.03
author_type            -0.09
author                 -0.11
total_meters           -0.19
living_meters          -0.19
rooms_count            -0.20
district               -0.32
Name: square_price, dtype: float64
```

Делаю вывод, что больше всего влияют количество этажей, метро и жилой комплекс.

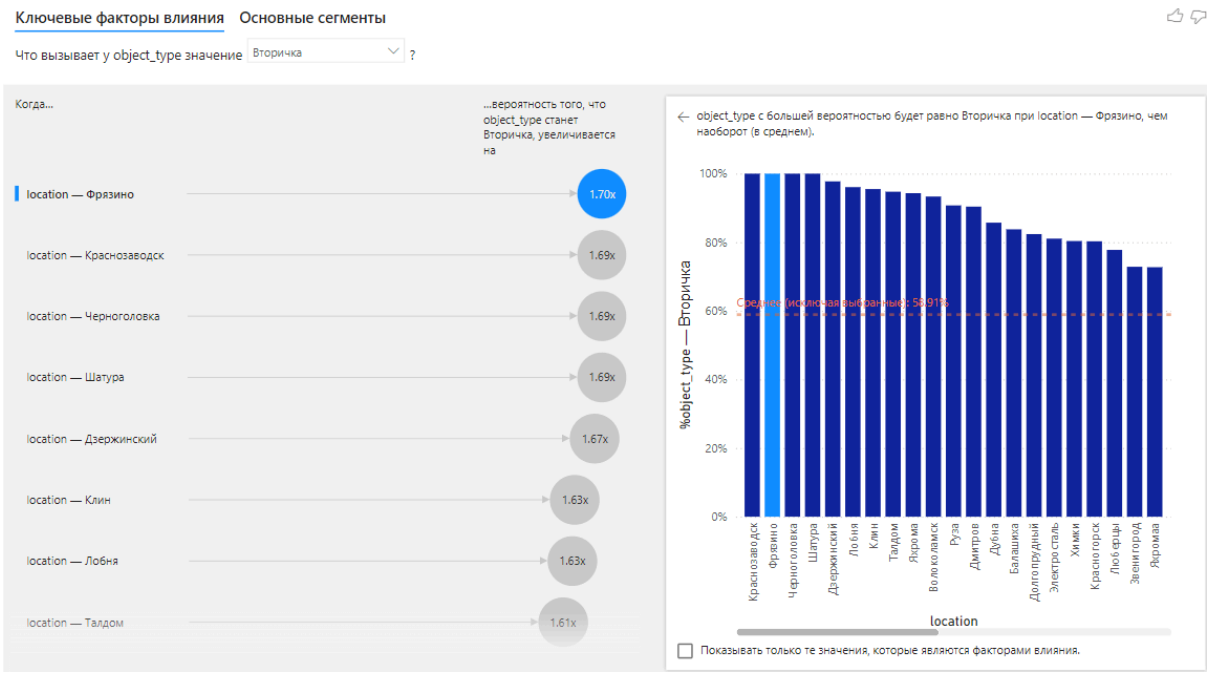
Меньше влияют высота потолка, тип жилья, тип парковки, улица, наличие лоджии / балкона и площадь кухни. А что тип предложения абсолютно не влияет на цену.

Отрицательную корреляцию имеют город, тип автора, а также, что странно - общая и жилая площадь и количество комнат. Вполне возможно такие данные получились из-за каких-то оставшихся выбросов.

Power BI

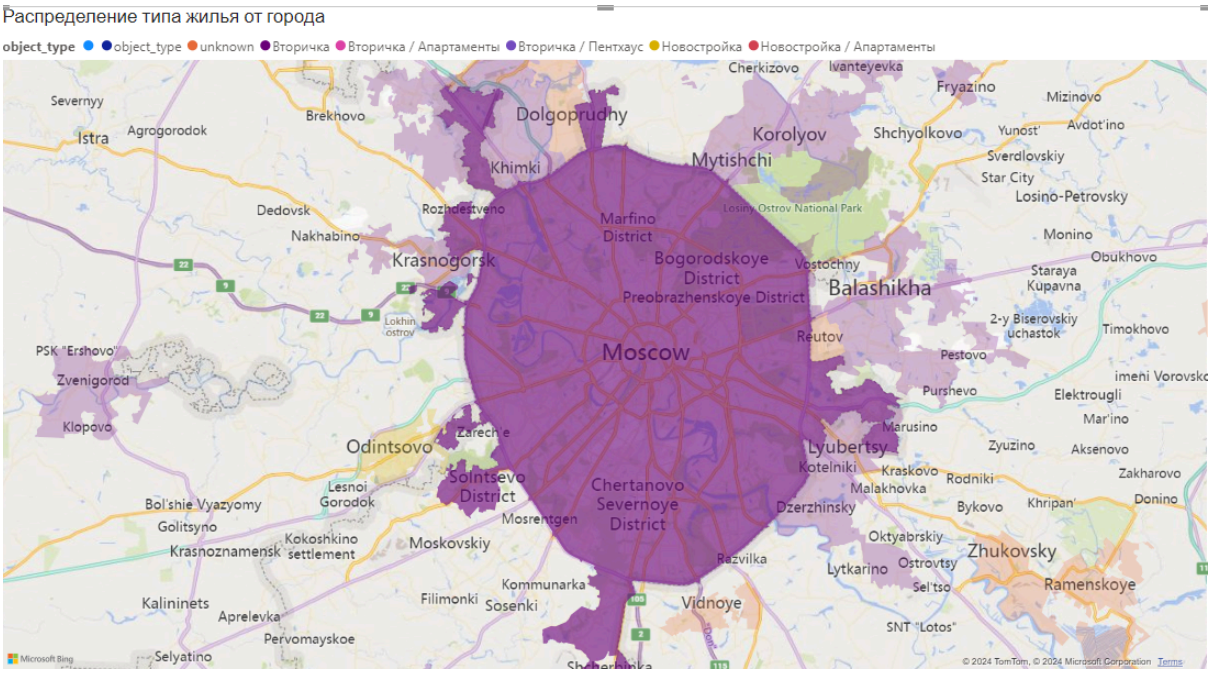


Больше всего Новостроек находится в Ногинске и Москве



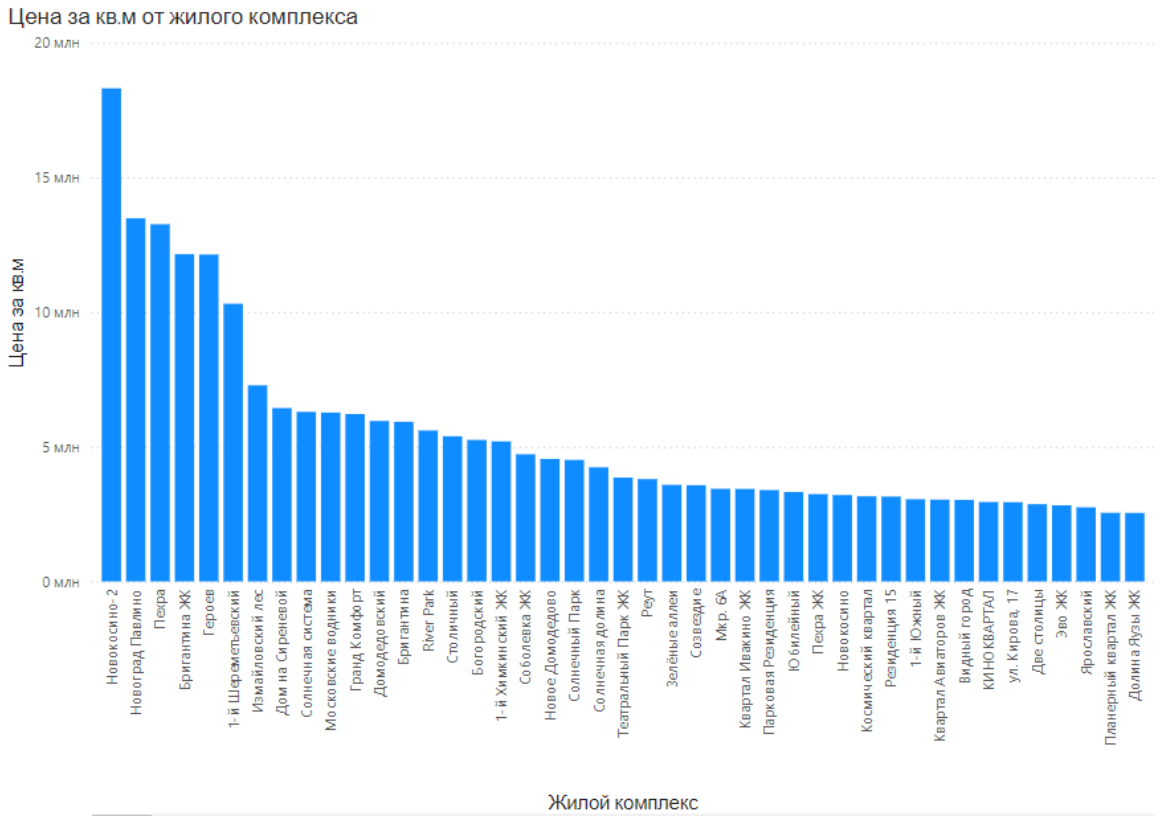
Очень много Вторичек находятся во Фрязино

Распределение типов жилья по городам на карте:

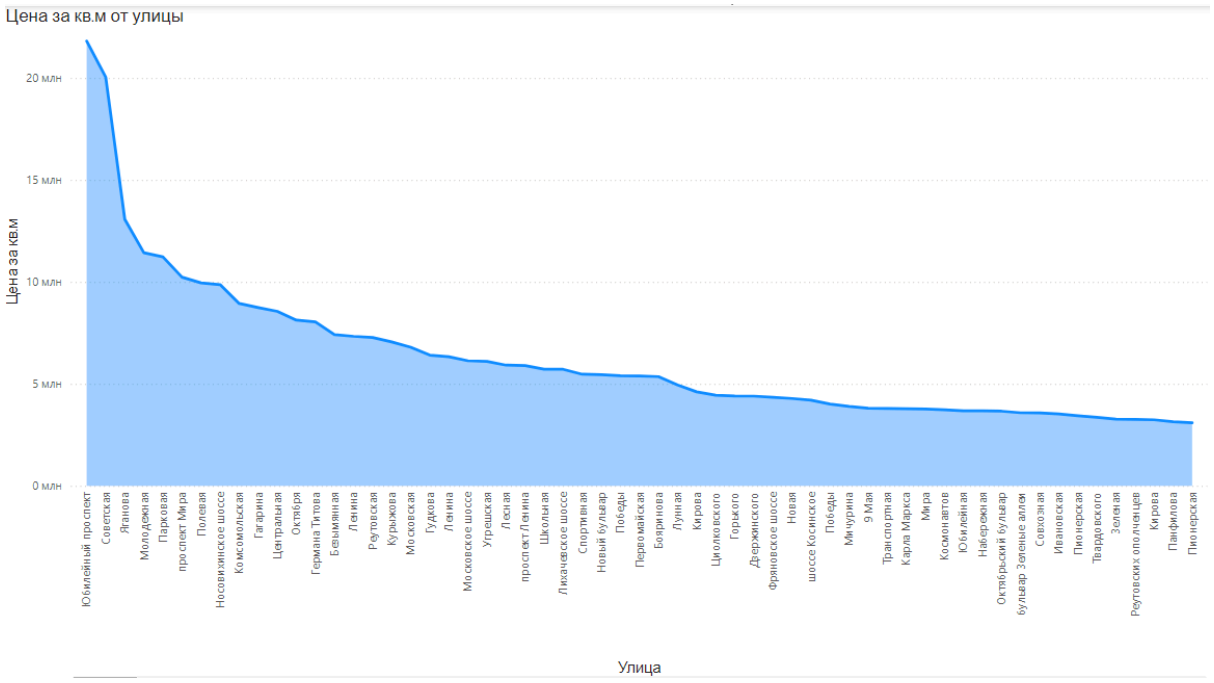


Как цена зависит от метро:

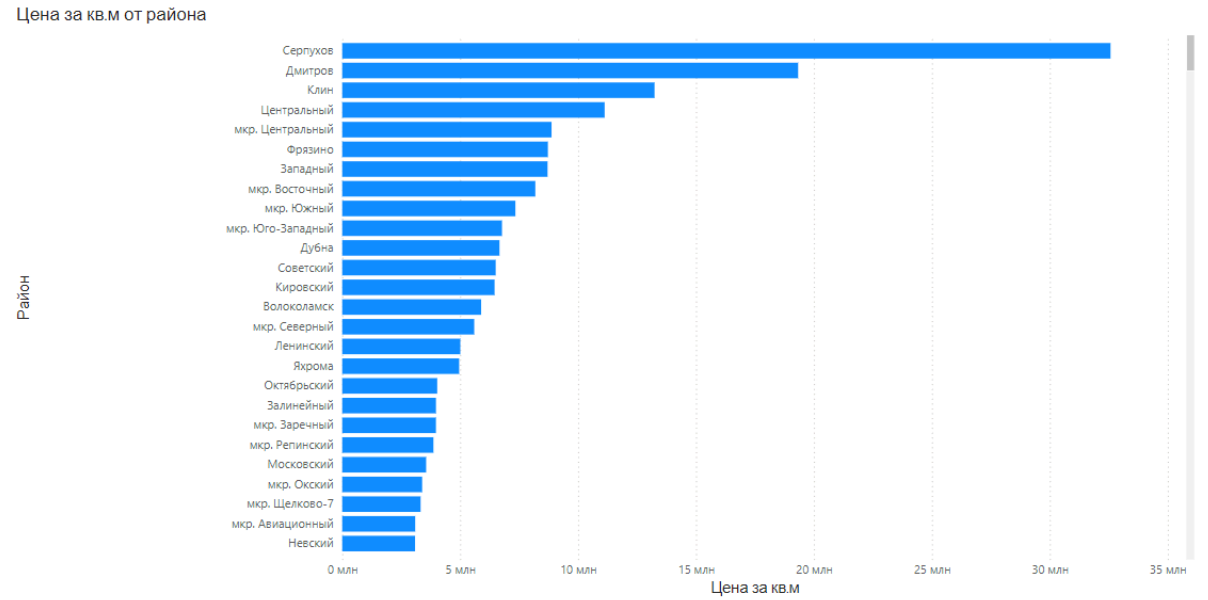
Как цена зависит от жилого комплекса:

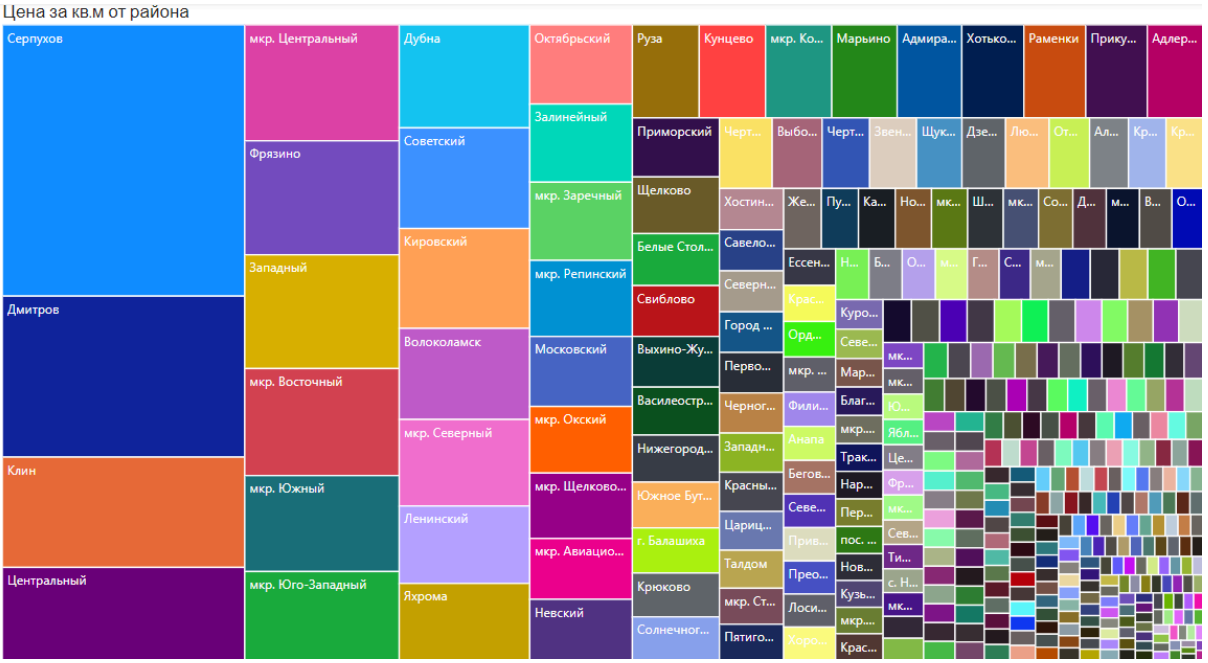


Зависимость от улицы:

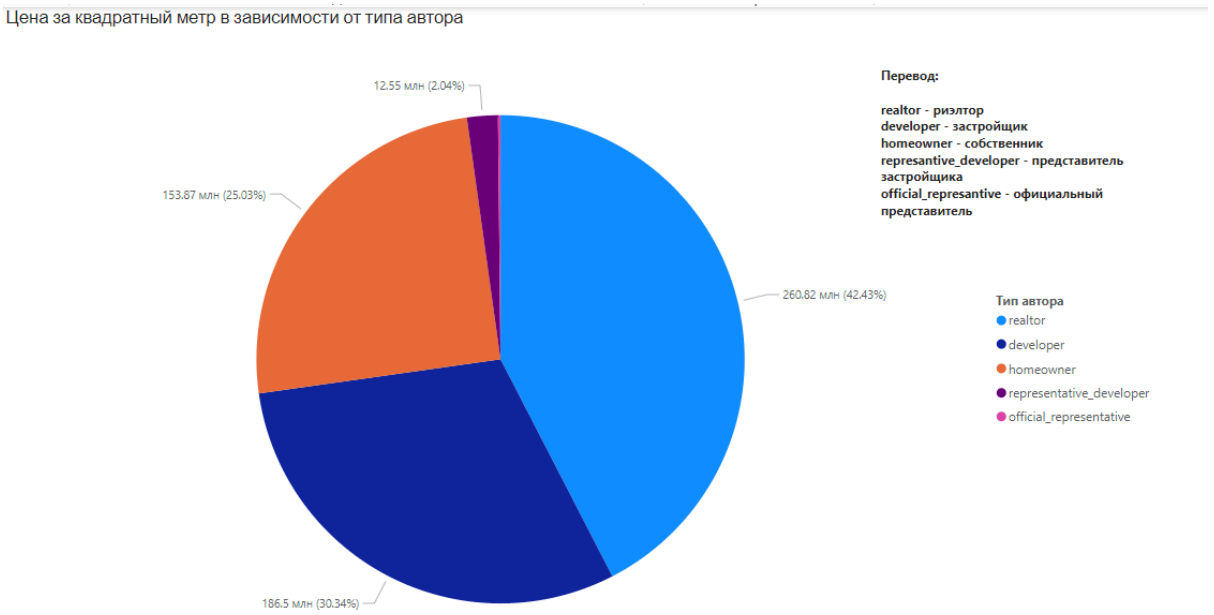


От района (2 вида графиков):





От автора:


Цена за квадратный метр в зависимости от типа автора
12.55 млн (2.04%)
153.87 млн (25.03%)
260.82 млн (42.43%)
186.5 млн (30.34%)
Перевод:
realtor - риэлтор
developer - застройщик
homeowner - собственник
represantive_developer - представитель застройщика
official_represantive - официальный представитель
Тип автора
realtor
developer
homeowner
representative_developer
official_representative

ЗАКЛЮЧЕНИЕ

В ходе исследования анализировались цены на недвижимость в Московском регионе для выявления факторов, влияющих на стоимость квадратного метра жилья. Данные были очищены от дубликатов, преобразованы в нужные типы и очищены от выбросов. Нулевые значения заполнены медианой, модой или 'unknown'. Добавлена целевая переменная 'square_price'.

Визуализация показала зависимость цены от типа жилья, года постройки, парковки и локации. Матрица корреляций выявила главные влияющие признаки.

Несмотря на удаление некоторых колонок (тип отделки, тип отопления) из-за отсутствия данных, и наличия небольшой части выбросов, результаты предоставляют ценную информацию для участников рынка недвижимости.

Из всех выведенных графиков было выявлено очень много интересных зависимостей: в том числе от улицы, жилого комплекса, метро и типа автора.