

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

Аналитический отчёт

Работу выполнила:
студент группы № ИСП-22
Титова София

Долгопрудный, 2024

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
ВВЕДЕНИЕ	3
МЕТОДОЛОГИЯ	4
СБОР ДАННЫХ	5
Циан	5
Домклик	9
АНАЛИЗ ДАННЫХ	11
ДомКлик	11
Циан	12
Power BI	23
ЗАКЛЮЧЕНИЕ	24

ВВЕДЕНИЕ

Оценка недвижимости — важная составляющая девелоперского бизнеса. Информация о реальной цене квартиры интересна для покупателей, продавцов, застройщиков, агентов и др.

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

Актуальность: оценка стоимости квадратного метра недвижимости крайне важна для анализа рынка недвижимости, принятия взвешенных решений, развития бизнеса и исследований.

Задачи:

- Используя открытые источники и личный опыт, составить список параметров, значительно влияющих на цену квадратного метра жилой площади;
- С учётом выявленных выше факторов произвести парсинг данных по квартирам на продажу, используя различные парсеры. Данные получаем с различных сайтов с объявлениями о продаже недвижимости: Циан, Авито, ДомКлик и др;
- Произвести подготовку данных для анализа: проверка на пропуски, выбросы и ошибки. Обработать выявленные аномалии (удалить / заполнить);
- Провести Исследовательский Анализ Данных (EDA). Построить распределение основных параметров; визуализировать взаимосвязи между ними; определить признаки, оказывающие наиболее сильное влияние на целевую переменную;
- Построить модель для предсказания цены на квадратный метр.

Теория: на цену за квадратный метр недвижимости влияют город, площадь, этаж, квартиры, высота потолков и различные дополнительные факторы, к примеру район, наличие парковки, тип дома и так далее.

МЕТОДОЛОГИЯ

Используемые инструменты и технологии:

- Библиотеки для парсинга: cianparser и Domclick Parser
- Среда разработки для парсинга и формирования (сбора датасета): Visual Studio Code (VS Code) , расширение Jupyter для VS Code
- Библиотеки Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn
- Среда для анализа данных и построения модели: Google Colaboratory
- Power BI

СБОР ДАННЫХ

Циан

Устанавливаем библиотеку `ciانparser` и первым делом переписываем файл `page.py` в файлах библиотеки, чтобы метод `get_flats()` потом смог более корректно спарсить дополнительную информацию из объявлений по продаже. Изменённый код файла `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(5 + 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(5 + 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(5 + 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(5 + 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(5 + 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(5 + random.uniform(0, 5))
            if "Площадь кухни" == span.text:
                page_data["kitchen_meters"] = spans[index + 1].text
                time.sleep(5 + random.uniform(0, 5))
            if "Жилая площадь" == span.text:
                page_data["living_meters"] = spans[index + 1].text
                time.sleep(5 + random.uniform(0, 5))
            if "Год постройки" in span.text:
                page_data["year_of_construction"] = spans[index + 1].text
                time.sleep(5 + random.uniform(0, 5))
            if "Год сдачи" in span.text:
                page_data["year_of_construction"] = spans[index + 1].text
                time.sleep(5 + 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(5 + 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(5 + random.uniform(0, 5))
        return page_data
    def parse_page(self):
        self.__load_page__()
        return self.__parse_flat_offer_page_json__()

```

Сам парсинг осуществляется через такой код:

```

parser = cianparser.CianParser(location="Серпухов")
data = parser.get_flats(deal_type="sale", rooms=(1),
additional_settings={"start_page": 1, "end_page": 54}, with_extra_data=True,
with_saving_csv=True)

```

Меняются значения location (город), rooms (количество комнат) в зависимости от того, какие объявления нужно спарсить.

Каждый раз при запуске кода создаётся файл в формате csv, в который записываются спарсенные данные. Чтобы соединить все эти файлы в один DataFrame, устанавливаем и импортируем библиотеку pandas, считываем все csv файлы с помощью read_csv() и соединяем это всё через метод concat().

```

# пример чтения файла
df_ser1 = pd.read_csv('serpuhov1.csv', sep = ';')
# метод concat(), в самом коде соединяется большее кол-во файлов
df = pd.concat([df_ser1, df_ser2, df_ser3, df_ser4, df_dub1, df_dub2, df_dub3,
df_msk1])

```

Выгружаем собранный датасет:

```

df.to_csv('war_is_over.csv', sep=',', index=False, encoding='utf-8')

```


Домклик

Код для парсинга с ДомКлика:

```
import json
import requests
import hashlib
from datetime import datetime
import pandas as pd
from itertools import product
from concurrent.futures import ThreadPoolExecutor, as_completed

class DomClickApi:
    def __init__(self):
        self.session = requests.Session()
        self.session.headers.update({
            "X-Service": "true",
            "Connection": "Keep-Alive",
            "User-Agent": "Android; 12; Google; google_pixel_5; 8.72.0; 8720006; ; NONAUTH"
        })

        # Инициализация (получение cookies)
        self.get("https://api.domclick.ru/core/no-auth-zone/api/v1/ensure_session")
        self.get("https://ipoteka.domclick.ru/mobile/v1/feature_toggles")

    def get(self, url, **kwargs):
        self.__update_headers(url, **kwargs)
        result = self.session.get(url, **kwargs)
        return result

    def __update_headers(self, url, **kwargs):
        url = self.__get_prepared_url(url, **kwargs)
        sault = "ad65f331b02b90d868cbdd660d82aba0"
        timestamp = str(int(datetime.now().timestamp()))
        encoded = (sault + url + timestamp).encode("UTF-8")
        h = hashlib.md5(encoded).hexdigest()
        self.session.headers.update({
            "Timestamp": timestamp,
            "Hash": "v1:" + h,
        })

    def __get_prepared_url(self, url, **kwargs):
        p = requests.models.PreparedRequest()
        p.prepare(method="GET", url=url, **kwargs)
        return p.url

    def fetch_offers(dca, params):
        offers_url = 'https://offers-service.domclick.ru/research/v5/offers/'
        offset = 0
        limit = 30
        offers_list = []

        while True:
            res = dca.get(offers_url, params={**params, 'offset': offset, 'limit': limit})
            try:
                data = res.json()
                if not data['success']:
                    print(f"Ошибка API: {data['errors']}")
                    break
                offers = data.get("result", {}).get("items", [])
                if not offers:
                    break
                offers_list.extend(offers)
                offset += limit
            except json.JSONDecodeError:
                print(f"Ошибка: Не удалось декодировать JSON. Ответ сервера:\n{res.text}")
                break

        print(f"Загружено {len(offers_list)} предложений с параметрами: {params}")
        return offers_list

    def generate_param_combinations():
        param_grid = {
            'address': ['1d1463ae-c80f-4d19-9331-a1b68a85b553'],
            'deal_type': ['sale'],
            'category': ['living'],
        }
```

```

        'offer_type': [['flat'], ['layout']],
        'rooms': [None, 'st', '1', '2', '3', '4+'],
        'time_on_foot_lte': [None, 5, 10, 15, 20],
        'time_by_car_lte': [None, 5, 10, 15, 20],
    }

    for params in product(*param_grid.values()):
        yield dict(zip(param_grid.keys(), params))

def main():
    dca = DomClickApi()
    df = pd.DataFrame()
    max_offers = 20000
    columns_needed = [
        'id',
        'offer_type',
        'object_info.floor',
        'object_info.rooms',
        'object_info.area',
        'price_info.price',
        'price_info.square_price',
        'address.position.lat',
        'address.position.lon',
        'seller.agent.is_agent',
        'published_dt',
        'updated_dt',
        'trade_in',
        'source',
        'ipoteka_rate',
        'has_advance_payment',
        'is_exclusive',
        'status',
        'assignment_sale',
        'online_show',
        'last_price_history_state',
        'is_placement_paid',
        'discount_status.status',
        'discount_status.value',
        'duplicates_offer_count',
        'chat_available',
        'is_auction',
        'address.id',
        'address.kind',
        'address.guid',
        'address.parent_id',
        'address.locality.id',
        'address.locality.kind',
        'address.locality.subkind',
        'address.locality.parent_id',
        'legal_options.is_owner',
        'legal_options.is_agent_owner_approved',
        'pessimization.pessimized',
        'pessimization.pessimization_type',
        'house.floors',
    ]

    with ThreadPoolExecutor(max_workers=10) as executor:
        futures = [executor.submit(fetch_offers, dca, params) for params in
generate_param_combinations()]

        for future in as_completed(futures):
            offers = future.result()
            temp_df = pd.json_normalize(offers)

            temp_df = temp_df[[col for col in columns_needed if col in temp_df.columns]]

            df = pd.concat([df, temp_df])
            df.drop_duplicates(subset=['id'], keep='first', inplace=True)

            if len(df) >= max_offers:
                break

    df.to_csv('ml_intensiv_oct.csv', index=False)
    print("данные сохранены")

if __name__ == "__main__":
    main()

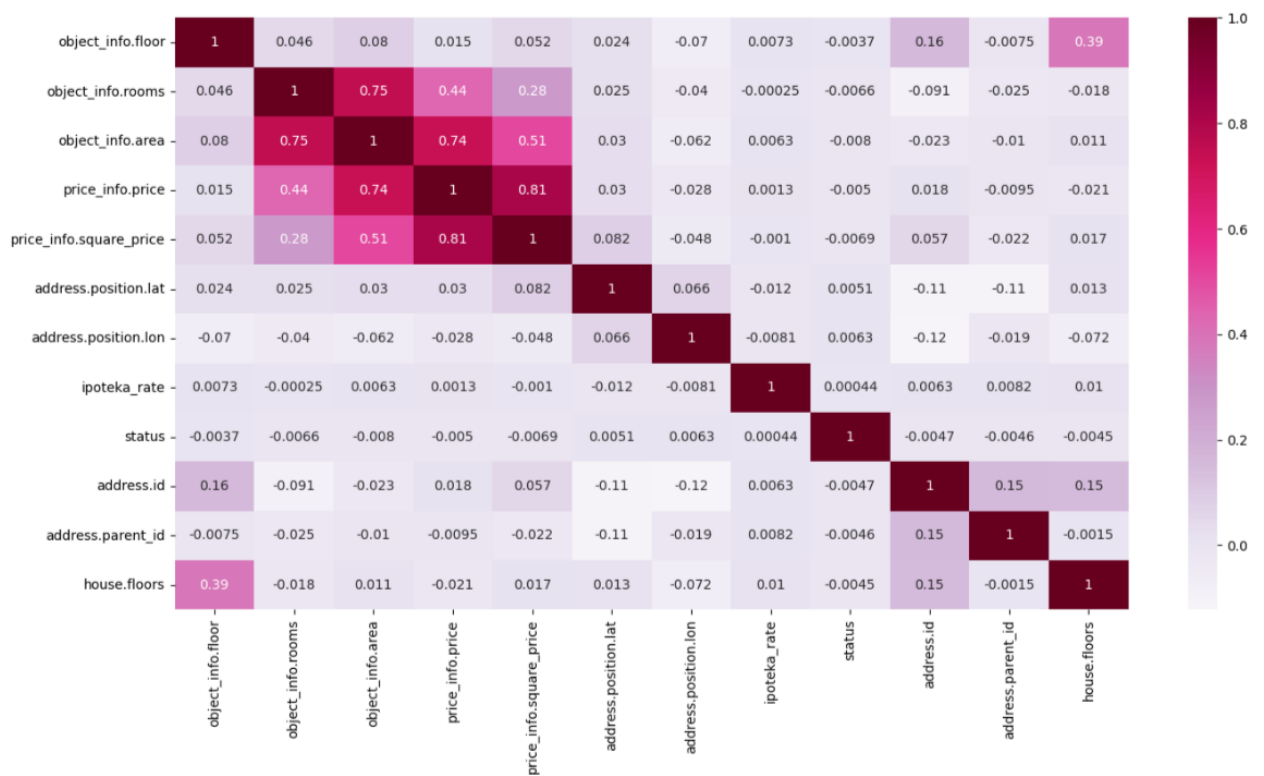
```

АНАЛИЗ ДАННЫХ

ДомКлик

Примечание: основной анализ и обработка данных делалась только на том, что было спарсено с циана, так как там вышли более понятные данные для обучения и построения модели.

В файле с данными с Домклик я удалила много колонок, убрала пропуски и вывела матрицу корреляции:

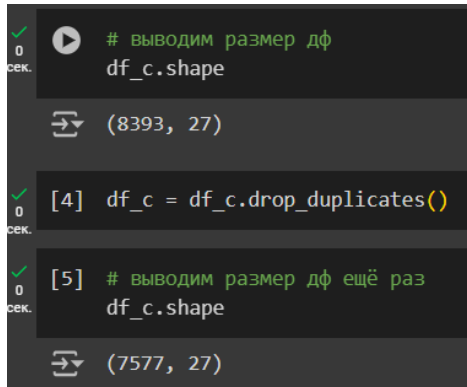


Матрица корреляции показывает, что:

- Количество комнат больше всего зависит от площади квартиры
- Площадь квартиры зависит от её стоимости и количества комнат
- Стоимость квартиры больше зависит от площади и цены за квадратный метр, чем от количества комнат
- Цена за квадратный метр зависит от стоимости квартиры и от площади.

Циан

Загружаем файл с данными, смотрим его размер, удаляем дубликаты и снова выводим размер таблицы:



```
# выводим размер дф
df_c.shape

(8393, 27)

[4] df_c = df_c.drop_duplicates()

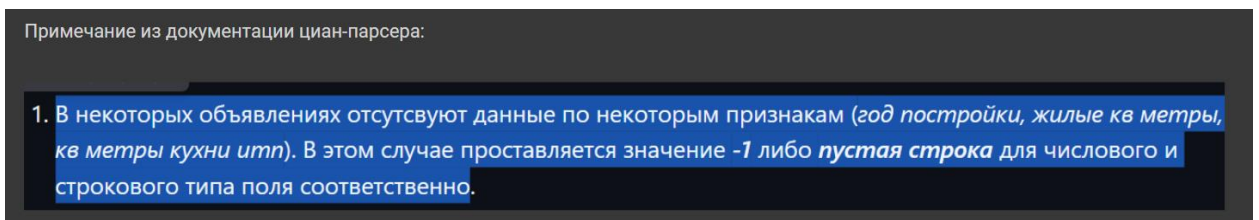
[5] # выводим размер дф ещё раз
df_c.shape

(7577, 27)
```

Было 8393 строки, осталось 7577. Колонок на данном этапе 27 штук. Первые две колонки хранят информацию о том, кто выложил объявление (имя автора и кто он), это не нужно для анализа, эти колонки удаляем. После этого выводим информацию о табличке.

Во всех колонках тип данных у значений object, надо будет кодировать данные по всем колонкам.

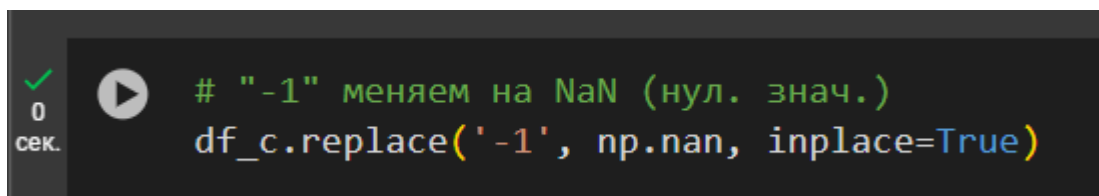
В документации библиотеки `ciانparser` есть следующее примечание:



Примечание из документации циан-парсера:

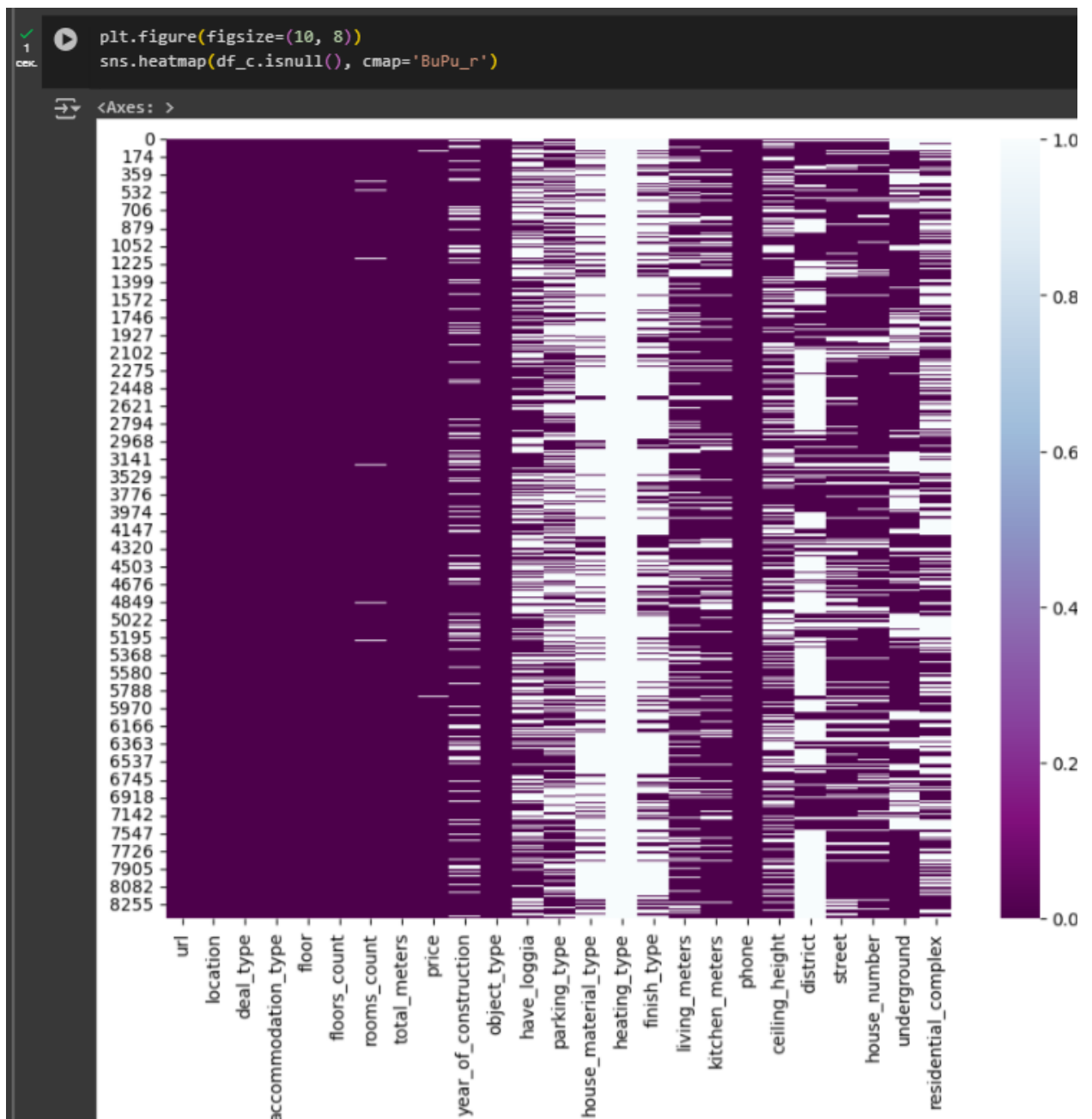
1. В некоторых объявлениях отсутствуют данные по некоторым признакам (*год постройки, жилые кв метры, кв метры кухни итп*). В этом случае проставляется значение *-1* либо *пустая строка* для числового и строкового типа поля соответственно.

Исходя из этого, надо заменить все значения, равные «-1» на NaN и вывести тепловую карту, по которой можно увидеть количество пропусков по колонкам.



```
# "-1" меняем на NaN (нул. знач.)
df_c.replace('-1', np.nan, inplace=True)
```

Тепловая карта:



Смотрим сколько пропусков в каждой колонке и начинаем работать с ними.

```
# выводим кол-во нул. значений ещё раз
null_counts_cian = df_c.isnull().sum().reset_index()
null_counts_cian.columns = ['Column', 'Null Count']

print(null_counts_cian)
```

	Column	Null Count
0	url	1
1	location	1
2	deal_type	1
3	accommodation_type	1
4	floor	1
5	floors_count	1
6	rooms_count	97
7	total_meters	1
8	price	14
9	year_of_construction	1315
10	object_type	1
11	have_loggia	3243
12	parking_type	3334
13	house_material_type	6191
14	heating_type	7576
15	finish_type	5668
16	living_meters	1701
17	kitchen_meters	1277
18	phone	1
19	ceiling_height	2664
20	district	4583
21	street	1575
22	house_number	1133
23	underground	2277
24	residential_complex	3778

Удаляем колонки "heating_type", "house_material_type", "finish_type" и 'district', т.к. в них очень много пропусков. Так же удаляем строки, в которых не указаны атрибуты "rooms_count" и "url".

Ещё удаляем колонки "house_number" и "residential_complex", из адреса (местоположения) оставляем метро\мдд и улицу. Удаляем колонку с номером телефона, т.к. она не нужна для анализа.

С помощью метода describe() выводим информацию о колонке url. Видим, что уникальных значений меньше, чем строк, оставляем по ней только уникальные значения.

Пропуски по колонке price заполняем вручную, т.к. их мало.

Удаляем колонку с ссылкой. Так же удаляем колонки deal_type и accommodation_type т.к. в них только одно уникальное значение.

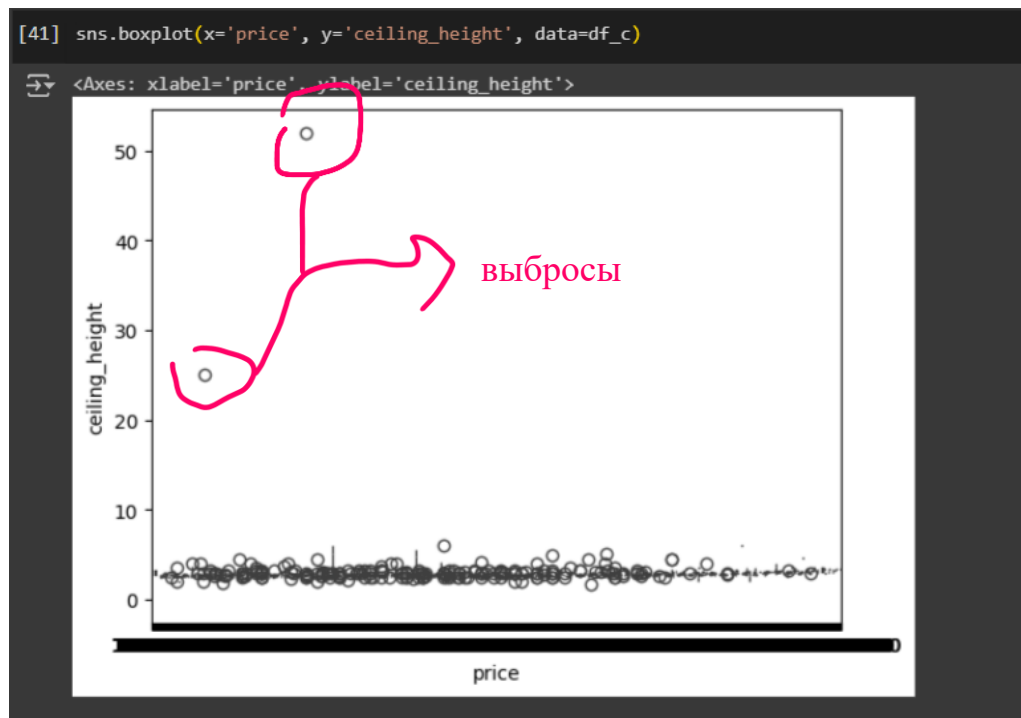
```
[24] print(df_c['deal_type'].unique())
      print(df_c['accommodation_type'].unique())

      ['sale' 'deal_type']
      ['flat' 'accommodation_type']

[25] # удаляем их, тк они содержат одно значение и ни на что не влияют
      df_c.drop(['deal_type', 'accommodation_type'], axis=1, inplace=True)
```

Дальше начинаем параллельно с обработкой пропусков кодировать данные. Сначала кодируем столбцы, которые по смыслу хранят числовую информацию, такие как количество комнат, цена, этаж, площадь, высота потолка.

Пропуски по колонке `ceiling_height` (высота потолков) можно заполнить средним значением, но сначала нужно посмотреть на выбросы и убрать их.



После удаления выбросов смотрим на числовые характеристики по колонке и заполняем пропуски средним значением, округлённым до сотых.

```
[45] df_c['ceiling_height'].describe()
```

ceiling_height	
count	4317.000000
mean	2.814443
std	0.305040
min	0.000000
25%	2.700000
50%	2.750000
75%	2.900000
max	9.000000

После этого смотрим на колонку object_type и проделываем следующую работу:

```
[48] print(f'Уникальные значения: {df_c["object_type"].unique()}')
```

```
↪ Уникальные значения: ['Вторичка' 'Новостройка' 'Вторичка / Апартаменты'  
 'Новостройка / Апартаменты' 'Вторичка / Пентхаус'  
 'Новостройка / Пентхаус']
```

Делаем так, чтобы было всего два варианта: новостройка и вторичка

```
[49] # функция для замены значений  
def values(value):  
    if value == 'Вторичка / Апартаменты':  
        return 'Вторичка'  
    elif value == 'Вторичка / Пентхаус':  
        return 'Вторичка'  
    elif value == 'Новостройка / Пентхаус':  
        return 'Новостройка'  
    elif value == 'Новостройка / Апартаменты':  
        return 'Новостройка'  
    else:  
        return value  
  
df_c['object_type'] = df_c['object_type'].apply(values)  
  
print(f'Уникальные значения: {df_c["object_type"].unique()}')
```

```
↪ Уникальные значения: ['Вторичка' 'Новостройка']
```

После работаем со столбцом year_of_construction:

```
[51] # выводим уник. знач.  
df_c['year_of_construction'].unique()
```

```
↪ array(['1917', '1975', nan, '1968', '2009', '1969', '1980', '1990',  
        '2025', '2015', '1970', '1984', '1965', '2010', '1964', 'Аукцион',  
        '1992', '2011', '1986', '1994', '1956', '1966', '1973', '1971',  
        '1957', '2013', '1981', '1972', '1959', '2004', '1962', '2019',  
        '1978', '1977', '2003', '1963', '1952', '2024', '2007', '1995',  
        '1976', '1982', '1953', '2008', '2026', '2023', '1941', '2028',  
        '1993', '2016', '2027', '2017', '2022', '1932', '2012', '1960',  
        '2006', '1926', '2020', '1943', '2014', '2021', '1967', '1904',  
        '1991', '2018', '1987', '1988', '1974', 'Напишите автору', '1979',  
        '1989', '1985', '1930', '2005', '1958', '1996', '1940', '1948',  
        '1961', '1938', '1997', '2002', '2001', '1999', '1983', '1998',  
        '1902', '2000', '1915', '1910', '1939', '1929', '1954', '1897',  
        '1900', 'Позвоните автору', '1955', '1949', '1947', '1901', '1909',  
        '1600', '1951', '1936', '1931', '1950', '1934', '1928', '1945',  
        '1937', '1927', '1896', '1777', '2031', '1935'], dtype=object)
```



```
[53] print(f'Пропусков в столбце "year_of_construction": {df_c["year_of_construction"].isna().sum()}')
print(f'Процент пропущенных значений: {(df_c["year_of_construction"].isna().sum() / len(df_c["year_of_construction"])).round(2) * 100}')

Пропусков в столбце "year_of_construction": 1120
Процент пропущенных значений: 17.0

[54] df_c['year_of_construction'].describe()

year_of_construction
count      5407
unique      112
top        2025
freq       589

dtype: object

Больше всего квартир с годом постройки\сдачи 2025
```

Снова выводим информацию и кодируем все столбцы, у которых тип данных object, кроме площади кухни и жилой площади; с помощью LabelEncoder. Отдельно кодируем living_meters и kitchen_meters.

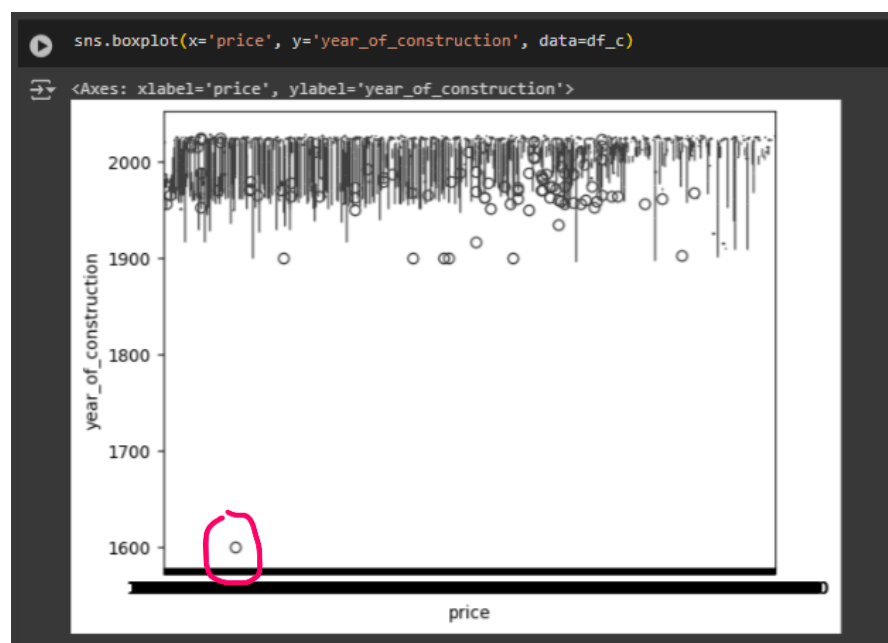
Колонку underground кодируем след. образом: если в ней пустое значение, проставляем 0, если нет – 1:

```
def replace_values(value):
    if pd.isna(value):
        return 0
    else:
        return 1

df_c['underground'] = df_c['underground'].apply(replace_values)
```

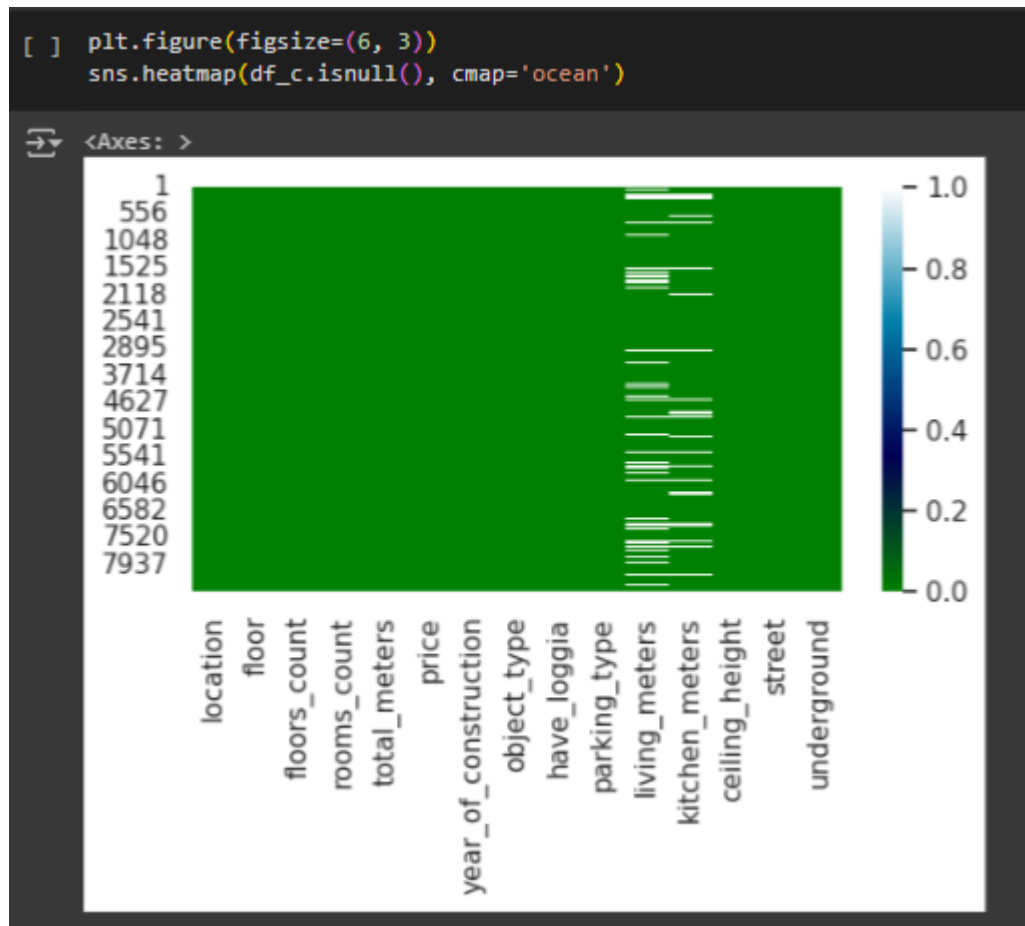
ПРИМЕЧАНИЕ: до этого я кодировала колонку underground через LabelEncoding() и корреляция между ценой и этим атрибутом была меньше.

Возвращаемся к колонке year_of_construction и визуализируем выбросы:



В итоге оставляем строки, в которых год входит в период с 1970-го по 2024ый, и удаляем 1998, так как это выброс (p.s.: я решила оставить остальные графики по этой колонке в коллабе, тк они довольно большие и отсюда их читать неудобно + они отобразятся в плохом качестве)

Выводим тепловую карту с пропусками, они остались в двух колонках:



Для заполнения площади кухни и жилой площади я сначала удаляю строки, где стоят нулевые значения в обеих этих колонках. Среди тех строк, которые остались, нулевое значение стоит только в одном из этих столбцов, поэтому я его заполняю разницей общей площади и известной площади (жилая\кухонная). Значение будет не точным, но приближенным к реальному.

```
[ ] # удаляем строки где обе площади заполнены нул. значением
    df_c.drop(df_c[(pd.isna(df_c['living_meters'])) & (pd.isna(df_c['kitchen_meters']))].index, inplace=True)

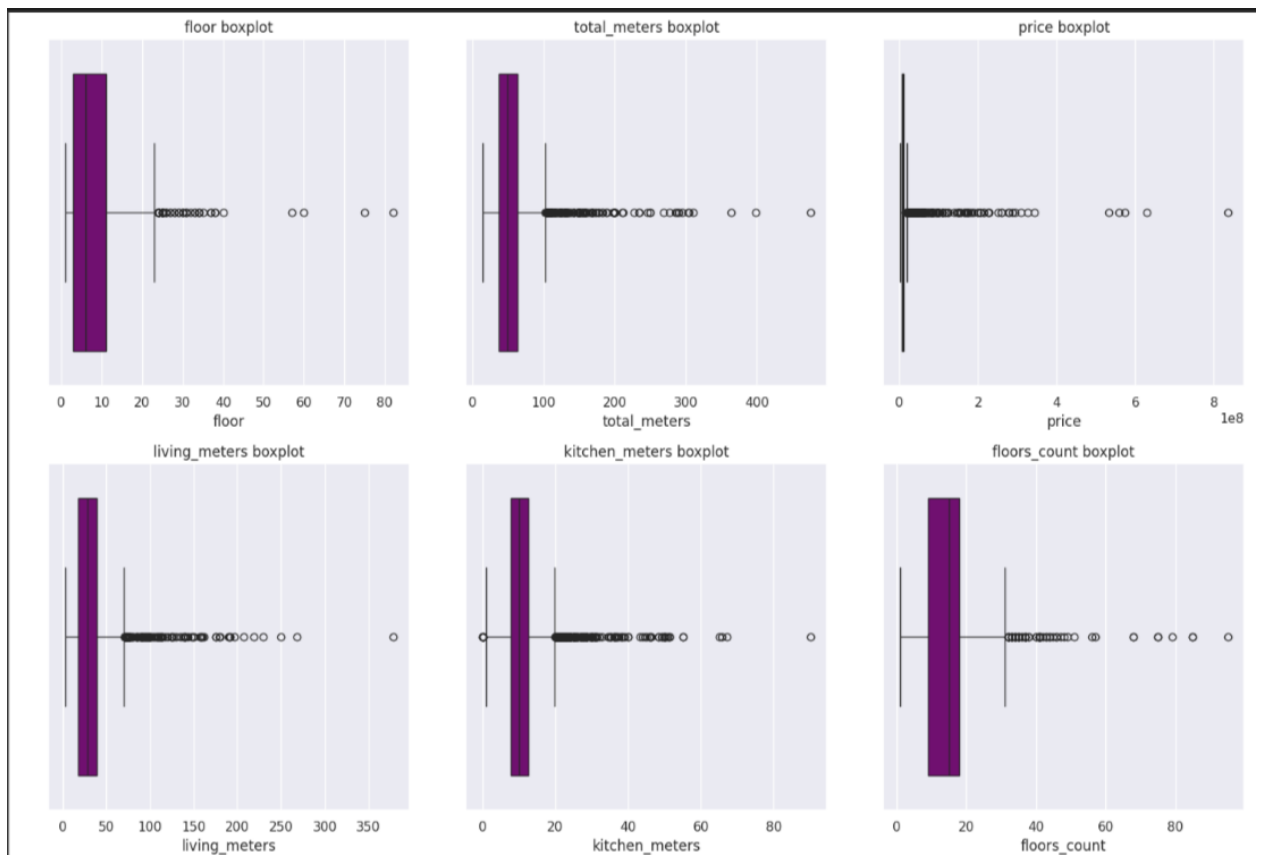
[ ] print(f'Пропусков в столбце "living_meters": {df_c["living_meters"].isna().sum()}')
    print(f'Пропусков в столбце "kitchen_meters": {df_c["kitchen_meters"].isna().sum()}')

Пропусков в столбце "living_meters": 431
Пропусков в столбце "kitchen_meters": 110

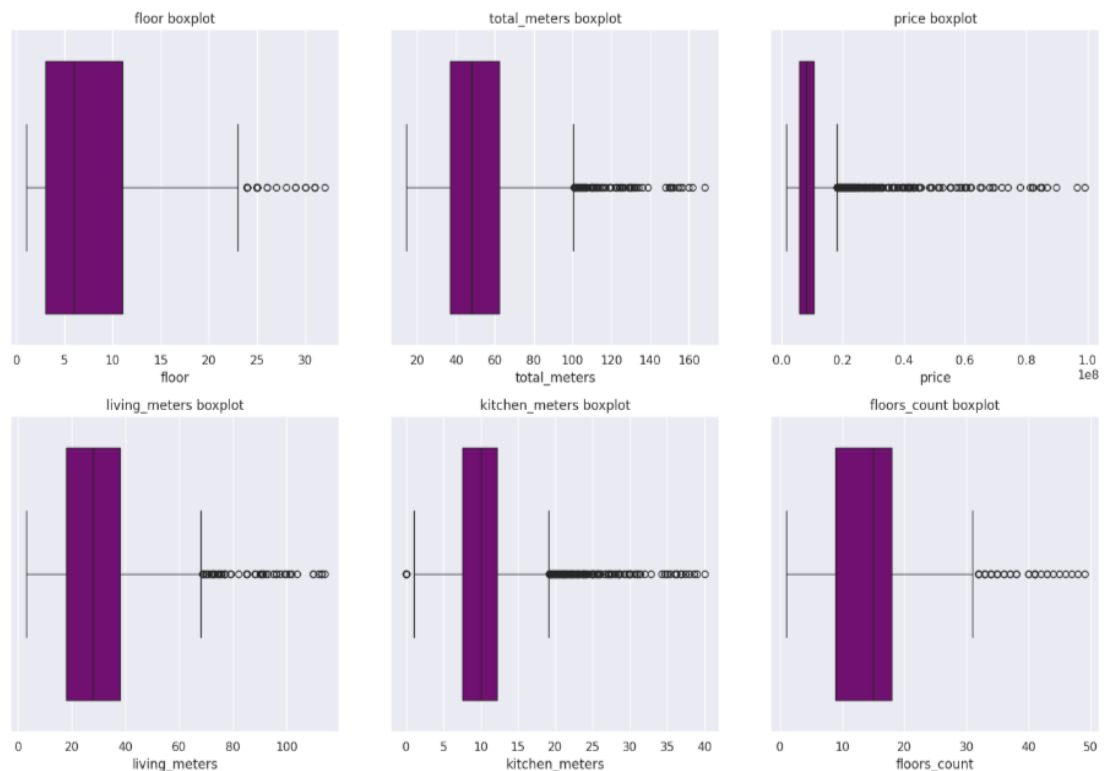
# Заполняем пропуски в 'living_meters'
df_c['living_meters'].fillna(df_c['total_meters'] - df_c['kitchen_meters'], inplace=True)

# Заполняем пропуски в 'kitchen_meters'
df_c['kitchen_meters'].fillna(df_c['total_meters'] - df_c['living_meters'], inplace=True)
```

Пропусков больше не осталось, продолжаем работать с выбросами. Выводим графики с выбросами по колонкам, в которых они могут быть (атрибуты, хранящие численные по смыслу данные).



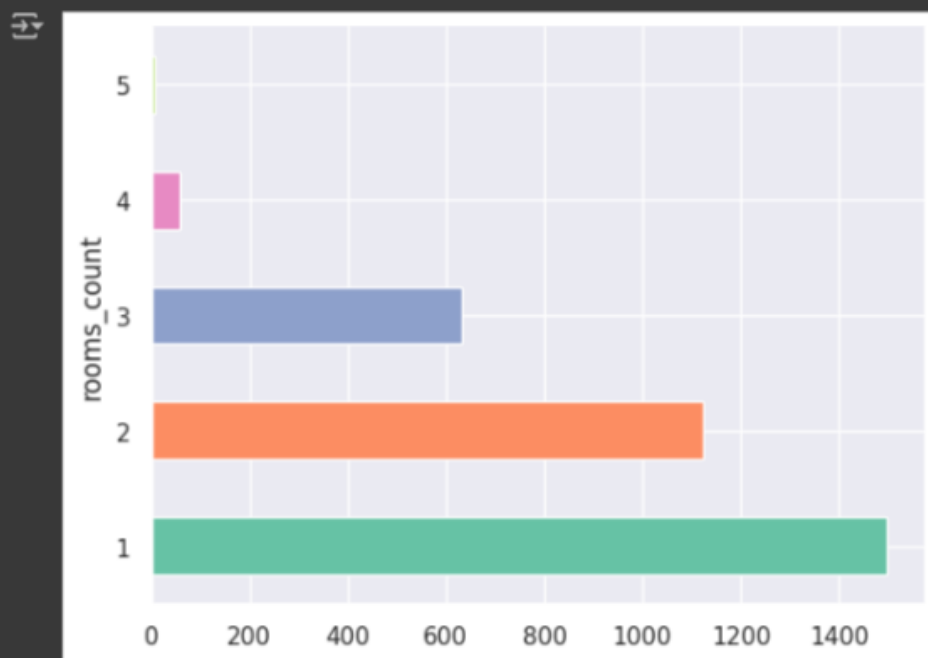
Удаляем выбросы, выводим такой же график:



Теперь вывожу пару графики для анализа:

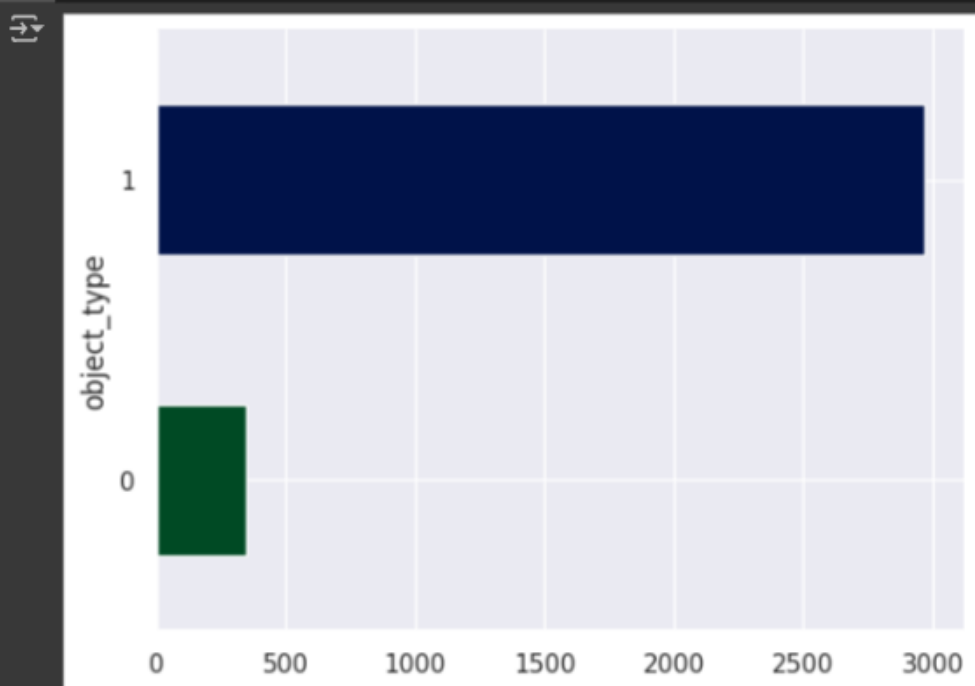
- Количество n-комнатных квартир:

```
df_c.groupby('rooms_count').size().plot(kind='barh', color=sns.palettes.mpl_palette('Set2'))  
plt.gca().spines[['top', 'right']].set_visible(False)
```



- График по колонке object_type

```
df_c.groupby('object_type').size().plot(kind='barh', color=sns.palettes.mpl_palette('ocean'))  
plt.gca().spines[['top', 'right']].set_visible(False)
```



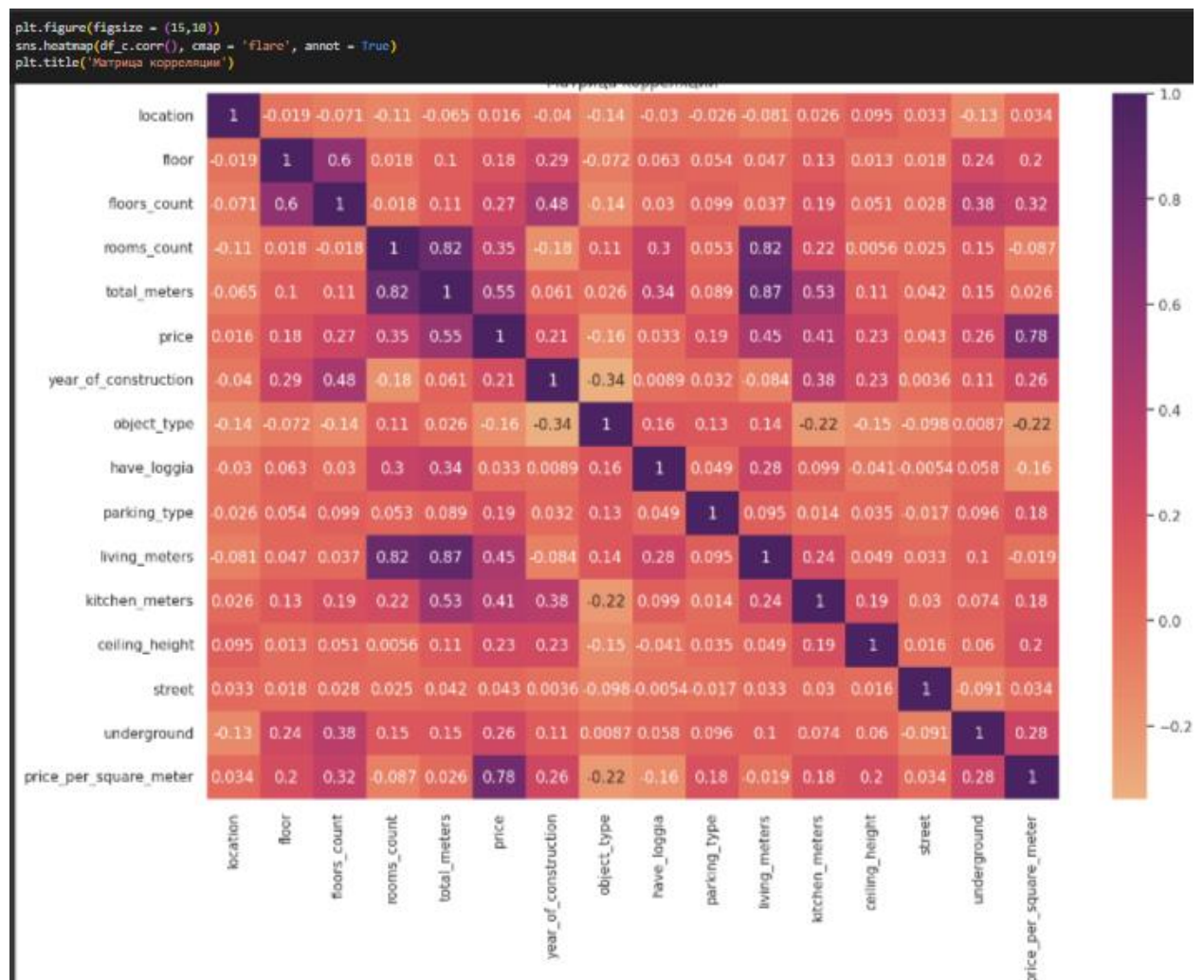
Вторичка = 1, Новостройка = 0 → Новостроек намного меньше (примерно в 9 раз)

Так же я вывела количество квартир по городам (по колонке location), но из-за размера графика решила его не вставлять в отчёт.

После всего этого добавляем колонку с целевой переменной – стоимостью за квадратный метр. Для этого нужно поделить стоимость всей квартиры на её полную площадь:

```
df_c = df_c.assign(price_per_square_meter = df_c['price'] // df_c['total_meters'])
```

Выводим матрицу корреляции и формулируем вывод:



Вывод по матрице корреляции ➡★

- Стоимость квартиры зависит от этажа, этажности дома, года постройки, количества комнат, общей площади квартиры, площади кухни и жилой площади, высоты потолка, немного от наличия метро\мцк (0.11) + есть небольшая корреляция с типом парковки
 - Стоимость за квадратный метр зависит от этажа, этажности дома, площади (но маловато, если опираться на эту матрицу), года постройки здания, высоты потолка + от наличия парковки, но немного (0.18) + от наличия рядом метро\мцд (0.28)
- Стоимость квартиры и стоимость за квадратный метр зависят друг от друга

После этого удаляем колонку с ценой, так как она не нужна для модели, смотрим на размер обработанного датасета и сохраняем его:

```
✓ [210] df_c.drop(['price'], axis=1, inplace=True)
```

```
✓ [216] df_c.shape
0      (3291, 15)
сек.
```

После обработки данных у нас осталось 3291 строка и 15 колонок

Сохраняем обработанный датасет на гугл диск

```
[212] #df_c.to_csv('for_model_ver2.csv')
```

```
[213] #from google.colab import drive
      #drive.mount('drive')
```

Drive already mounted at drive; to attempt to forcibly remount, call drive.mount("drive", force_remount=True).

```
[215] #!cp for_model_ver2.csv "drive/My Drive/"
```

Power BI



ЗАКЛЮЧЕНИЕ

Стоимость квартиры за один квадратный метр зависит от этажа, этажности дома, цены, площади (общая площадь, площадь кухни и жилая), года постройки здания, высоты потолка, а так же от наличия метро или МЦД, парковки и других признаков. Из-за того, что спарсилось слишком неравномерное количество квартир по городам и количеству комнат (второе особенно хорошо видно на графике в Power Bi и в коллабе), анализ получился не совсем корректным.