

# ИССЛЕДОВАНИЕ ОБЪЯВЛЕНИЙ О ПРОДАЖЕ КВАРТИР

Проект на тему "Исследовательский анализ данных" учебного курса "[Специалист по Data Science](#)" от Яндекс.Практикум.

Выполнил **Денис Абрашин** ([e-mail](#))

Чек-лист готовности проекта Версия 1.1

## Оглавление

- Задание
  - Общие требования
  - Описание полей набора данных
- Подключение и настройка необходимых библиотек
  - Настройка pandas
  - Настройка matplotlib и seaborn
  - Проверка версий подключенных библиотек
- Шаг 1. Открытие и анализ файла с данными
  - Задание шага 1
  - Выполнение шага 1
  - Замысел выполнения шага 1
  - Словарь полей данных и условий их автоматизированной проверки
    - Информация об одном поле
    - Информация о всех полях и их типах
    - Информация об одном условии проверки набора данных
  - Вспомогательный класс для загрузки и проверки файла данных
    - Замысел вспомогательного класса
    - Код вспомогательного класса
    - Наиболее употребимые элементы условий проверки
    - Специфические для полученного набора данных условия проверки
  - Параметры загрузки и открытия файлов
  - Загрузка файлов данных
  - Ручная проверка файлов данных
  - Вывод краткой информации о наборе данных
  - Просмотр первых строк набора данных
  - Изучение параметров распределения числовых данных
  - Вывод по шагу 1
- Шаг 2. Предобработка данных
  - Задание шага 2
  - Поиск пропусков, ошибок и аномалий в данных
    - Первая автоматизированная проверка типов полей
    - Первая автоматизированная проверка значений
  - Обработка пропусков, ошибок и аномалий в данных
    - Оценка количества пропусков
    - Разбиение названия населенного пункта

- Обработка нереальных и маловероятных значений
  - Устранение замечаний к данным о расстояниях
  - Фильтрация неадекватных потолков
  - Восстановление этажности
  - Устранение неопределенности суммы площадей
  - Восстановление пропусков в жилой площади
  - Корректировка сведений о балконах
  - Устранение других пропусков
  - Ложные тревоги
  - Устранение аномалий с близкими объектами
  - Обработка замечаний к количеству комнат
  - Повторная автоматизированная проверка типов полей
  - Повторная автоматизированная проверка значений
  - Удаление объявлений с неадекватной жилой площадью
  - Снятие с контроля расстояний до объектов
  - Контрольная автоматизированная проверка
  - Оценка количества пропусков после предобработки
  - Оценка количества полных дубликатов
  - Ручной контроль преобразования типов данных
  - Вывод по шагу 2
- Шаг 3. Вычисление новых показателей
    - Задание шага 3
    - Вычисление цены квадратного метра
    - Определение дня недели, месяца и года объявления
      - Учет даты совершения сделки
      - День недели объявления
      - Месяц объявления
      - Год объявления
      - Год и месяц объявления
    - Категорирование этажа квартиры
    - Определение соотношения жилой и общей площади
    - Вычисление отношения площади кухни к общей
    - Расстояние до центра в километрах
    - Описание новых полей
    - Проверка типов и значений новых полей
    - Вывод по шагу 3
  - Шаг 4. Исследовательский анализ данных
    - Задание шага 4
    - Функция построения гистограмм
    - Построение гистограмм
      - Гистограмма общей площади
      - Гистограмма площади кухни
      - Гистограмма цены квартиры
      - Гистограмма цены квадратного метра
      - Гистограмма количества комнат
      - Гистограмма высоты потолков
      - Гистограммы времени продажи квартиры
        - Гистограмма срока публикации объявления
        - Гистограмма года совершения сделки

- Гистограмма месяца совершения сделки
- Гистограмма дня недели совершения сделки
- Удаление редких и выбивающихся значений
- Какие факторы больше всего влияют на стоимость квартиры
  - Площадь, число комнат и удалённость от центра
  - Этаж квартиры
  - Динамика изменения цены квадратного метра по месяцам
  - День недели публикации объявления
  - Расстояние до объектов
- 10 населённых пунктов с наибольшим числом объявлений
- Анализ квартир в Санкт-Петербурге
  - Поиск центра города
  - Факторы влияния на квартиры в центре
- Сравнение динамики в разных районах
  - Динамика средней цены квадратного метра
  - Динамика средней цены квадратного метра
  - Динамика скорости продажи
  - Динамика количества продаж
- Вывод по шагу 4
- Шаг 5. Общий вывод
- Чек-лист готовности проекта
- Таблица контроля версий
- Бонус
  - Визуальная проверка правильности определения центра города
    - Установка библиотеки Folium
    - Подключение библиотеки Folium
    - Настройка карты
  - Карта центра города

## Задание

В вашем распоряжении данные сервиса Яндекс.Недвижимость — архив объявлений о продаже квартир в Санкт-Петербурге и соседних населённых пунктах за несколько лет. Нужно научиться определять **рыночную стоимость объектов недвижимости**. Ваша задача — установить параметры. Это позволит построить автоматизированную систему: она отследит аномалии и мошенническую деятельность.

По каждой квартире на продажу доступны два вида данных. Первые вписаны пользователем, вторые — получены автоматически на основе картографических данных. Например, расстояние до центра, аэропорта, ближайшего парка и водоёма.

## Общие требования

Полный текст задания можно скачать по [ссылке](#). Задания для отдельных шагов приведены ниже.

Выполните задание в Jupyter Notebook. Заполните программный код в ячейках типа code, текстовые пояснения — в ячейках типа markdown. Примените форматирование и заголовки.

## Описание полей набора данных

Имя поля	Описание поля
<b>airports_nearest</b>	расстояние до ближайшего аэропорта в метрах (м)
<b>balcony</b>	число балконов
<b>ceiling_height</b>	высота потолков (м)
<b>cityCenters_nearest</b>	расстояние до центра города (м)
<b>days_exposition</b>	сколько дней было размещено объявление (от публикации до снятия)
<b>first_day_exposition</b>	дата публикации
<b>the_floor</b>	этаж
	Указанное в файле данных название поля <b>floor</b> не может быть использовано в функции <code>query</code> Pandas, т.к. совпадает с уже зарегистрированным ключевым словом. В работе будем использовать другое название.
<b>floors_total</b>	всего этажей в доме
<b>is_apartment</b>	апартаменты (булев тип)
	Апартаменты — это нежилые помещения, не относящиеся к жилому фонду, но имеющие необходимые условия для проживания.
<b>kitchen_area</b>	площадь кухни в квадратных метрах ( $\text{м}^2$ )
<b>last_price</b>	цена на момент снятия с публикации
<b>living_area</b>	жилая площадь в квадратных метрах ( $\text{м}^2$ )
<b>locality_name</b>	название населённого пункта
<b>open_plan</b>	свободная планировка (булев тип)
<b>parks_around3000</b>	число парков в радиусе 3 км
<b>parks_nearest</b>	расстояние до ближайшего парка (м)
<b>ponds_around3000</b>	число водоёмов в радиусе 3 км
<b>ponds_nearest</b>	расстояние до ближайшего водоёма (м)
<b>rooms</b>	число комнат
<b>studio</b>	квартира-студия (булев тип)
<b>total_area</b>	площадь квартиры в квадратных метрах ( $\text{м}^2$ )
<b>total_images</b>	число фотографий квартиры в объявлении

## Подключение и настройка необходимых библиотек

```
In [1]: # отключение предупреждений при желании
# import warnings; warnings.filterwarnings('once')

In [2]: import os
import errno

In [3]: from collections import namedtuple

In [4]: from urllib.request import urlretrieve

In [5]: from IPython.display import HTML, display
```

## Настройка pandas

```
In [6]: # обновление библиотеки Pandas для исключения ошибок версий  
!pip3 install --upgrade --user pandas
```

```
Requirement already up-to-date: pandas in c:\users\bomba\appdata\roaming\python\python37\site-packages (1.1.4)  
Requirement already satisfied, skipping upgrade: python-dateutil>=2.7.3 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from pandas) (2.8.1)  
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from pandas) (2020.1)  
Requirement already satisfied, skipping upgrade: numpy>=1.15.4 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from pandas) (1.19.2)  
Requirement already satisfied, skipping upgrade: six>=1.5 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
```

```
In [7]: import numpy as np  
import pandas as pd
```

```
In [8]: pd.set_option('display.notebook_repr_html', True)  
pd.set_option('display.max_columns', 8)  
pd.set_option('display.max_rows', 10)  
pd.set_option('display.width', 80)
```

## Настройка matplotlib и seaborn

```
In [9]: # обновление библиотек  
!pip3 install --upgrade --user matplotlib seaborn
```

```
Requirement already up-to-date: matplotlib in c:\users\bomba\appdata\roaming\python\python37\site-packages (3.3.3)  
Requirement already up-to-date: seaborn in c:\users\bomba\appdata\roaming\python\python37\site-packages (0.11.0)  
Requirement already satisfied, skipping upgrade: cycler>=0.10 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from matplotlib) (0.10.0)  
Requirement already satisfied, skipping upgrade: numpy>=1.15 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from matplotlib) (1.19.2)  
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from matplotlib) (2.8.1)  
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from matplotlib) (2.4.7)  
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from matplotlib) (1.3.0)  
Requirement already satisfied, skipping upgrade: pillow>=6.2.0 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from matplotlib) (8.0.1)  
Requirement already satisfied, skipping upgrade: scipy>=1.0 in c:\users\bomba\appdata\roaming\python\python37\site-packages (from seaborn) (1.5.4)  
Requirement already satisfied, skipping upgrade: pandas>=0.23 in c:\users\bomba\appdata\roaming\python\python37\site-packages (from seaborn) (1.1.4)  
Requirement already satisfied, skipping upgrade: six in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from cycler>=0.10->matplotlib) (1.15.0)  
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from pandas>=0.23->seaborn) (2020.1)
```

```
In [10]: import matplotlib as mpl  
import matplotlib.pyplot as plt  
import seaborn as sns
```

```
In [11]: %matplotlib inline
```

```
In [12]: small, medium, large = 12, 16, 22  
params = {'figure.figsize': (12, 6),  
          'figure.titlesize': large,  
          'legend.fontsize': medium,  
          'axes.titlesize': medium,  
          'axes.labelsize': medium,  
          'xtick.labelsize': medium,  
          'ytick.labelsize': medium,
```

```
'legend.loc':      'best'}
```

```
plt.rcParams.update(params)
```

```
In [13]: # повышение четкости графиков для больших мониторов
```

```
%config InlineBackend.figure_format = 'retina'
```

```
In [14]: # включение цветового оформления Seaborn
```

```
#plt.style.use('seaborn-whitegrid')
```

```
sns.set_style("white")
```

## Проверка версий подключенных библиотек

```
In [15]: def lib_versions(libs):
```

```
    for lib in libs: print('Версия', lib.__name__, '-', lib.__version__)
```

```
In [16]: lib_versions([np, pd, mpl, sns])
```

Версия numpy - 1.19.2

Версия pandas - 1.1.4

Версия matplotlib - 3.3.3

Версия seaborn - 0.11.0

## Шаг 1. Открытие и анализ файла с данными

### Задание шага 1

Откройте файл с данными и изучите общую информацию. Путь к файлу:

/datasets/real\_estate\_data.csv . [Скачать датасет](#). (Примечание исполнителя: датасет скачивается ниже автоматически).

### Выполнение шага 1

#### Замысел выполнения шага 1

Не будет преувеличением сказать, что прохождение учебного курса и приближение к реальным задачам сопровождается увеличением количества источников данных и их объемов. Проведение предобработки информации только вручную будет отнимать много времени и сил, что в свою очередь может привести к снижению качества анализа или срыву сроков сдачи проекта.

Для нивелирования описанной проблемы создадим вспомогательные структуры данных, классы и методы для автоматической загрузки массива данных, его первоначальной проверки и подготовки удобочитаемого отчета об обнаруженных ошибках в значениях полей.

Полученный опыт будем использовать не только в решении конкретных задач, но и в создании (совершенствовании) средств автоматической предобработки данных, которые, например, могут быть оформлены в виде отдельных библиотек.

Если Вам не терпится посмотреть результат, можете сразу перейти к [загрузке файлов данных](#).

## Словарь полей данных и условий их автоматизированной проверки

Опишем словарь со списком всех полей нашего набора данных в соответствии с их [описанием в задании](#). Эх, получить бы сразу такую информацию от поставщика данных в удобном формате типа JSON:

### Информация об одном поле

Используем `namedtuple` для описания структуры информации об отдельном поле набора данных. В коде программ к этим полям будем обращаться через точку:

- `desc` - смысловое описание поля;
- `desc_short` - смысловое описание поля в кратком формате для небольших графиков;
- `dtype` - тип поля;
- `try_to_fix` - если `True`, то попытаться автоматически исправить тип поля на указанный в `dtype`,
- `name_in_csv` - имя поля в файле данных.

```
In [17]: DataField = namedtuple('DataField', ['desc', 'desc_short', 'dtype', 'try_to_fix', 'name_in_csv'],
                           defaults=[None, None, None, True,
```

## Информация о всех полях и их типах

Сведем все имена полей, их описания и требования к типам в один словарь для последующей алгоритмизации. В учебных целях код оставим в этом ноутбуке. В будущем можем вынести его в отдельный файл.

```
In [18]: data_fields = \
{
    'rooms': DataField('число комнат', 'число\пкомнат', np.int64),
    'living_area': DataField('жилая площадь в м2', 'жилая\площадь', np.float64),
    'kitchen_area': DataField('площадь кухни в м2', 'площадь\пкухни', np.float64),
    'total_area': DataField('площадь квартиры в м2', 'общая\площадь', np.float64),
    'ceiling_height': DataField('высота потолков в метрах', 'потолок', np.float64),

    'studio': DataField('квартира-студия', 'студия', np.bool),
    'open_plan': DataField('свободная планировка', 'свободный\план', np.bool),
    'is_apartment': DataField('апартаменты', 'аппарт.', np.bool),
    'balcony': DataField('число балконов', 'балконов', np.int64),
    'the_floor': DataField('этаж', 'этаж', np.int64, True, 'floor'), # need to convert to int
    'floors_total': DataField('всего этажей в доме', 'этажей\в доме', np.int64),

    'locality_name': DataField('название населённого пункта', 'место', np.object),
    'cityCenters_nearest': DataField('расстояние до центра города в метрах', 'расстояние\центра'),
    'parks_around3000': DataField('число парков в радиусе 3 км', 'число\ппарков', np.int64),
    'parks_nearest': DataField('расстояние до ближайшего парка в метрах', 'расстояние\бпарка'),
    'ponds_around3000': DataField('число водоёмов в радиусе 3 км', 'число\ппрудов', np.int64),
    'ponds_nearest': DataField('расстояние до ближайшего водоёма в метрах', 'расстояние\бводоёма'),
    'airports_nearest': DataField('расстояние до ближайшего аэропорта в метрах', 'расстояние\баэропорта'),

    'first_day_exposition': DataField('дата публикации', 'опубликовано', np.dtype('date')),
    'days_exposition': DataField('сколько дней было размещено объявление', 'срок\года'),
    'last_price': DataField('цена на момент снятия с публикации', 'цена\пква'),
    'total_images': DataField('число фотографий квартиры в объявлении', 'количество\фото')
}
```

Опишем вспомогательные функции, которые пригодятся при подписывании осей графиков и именовании столбцов таблиц:

```
In [19]: def get_desc_long(field_names):
    """Возвращает список смысловых описаний для переданного списка полей"""
    return list(map(lambda x: data_fields[x].desc, field_names))

def get_desc_short(field_names):
    """Возвращает список смысловых описаний в кратком формате для переданного списка полей"""
    return list(map(lambda x: data_fields[x].desc_short, field_names))
```

## Информация об одном условии проверки набора данных

Аналогично опишем информацию об одном конкретном условии проверки значений набора

данных:

- `fields` - список имен проверяемых полей;
- `desc_template` - шаблон смыслового описания условия проверки, которое будет выводится в дальнейшем на экран с использованием значения `fields` в качестве параметра к `str.format`;
- `query_template` - шаблон строки запроса на выборку неудовлетворяющих условию записей через `pandas.query` с использованием значения `fields` в качестве параметра к `str.format`;
- `fix` - функция корректировки найденных ошибок (получает на вход ссылку на набор данных). Если нет необходимости исправлять ошибки, то присвойте `fix` значение `None`. Функцию удобно задавать через `lambda`. `fix` можно применять не только для исправления ошибок, но и просто для вывода дополнительной информации после проверки очередного условия.

```
In [20]: DataChecking = namedtuple('DataChecking', ['fields', 'desc_template', 'query_template'])
```

## Вспомогательный класс для загрузки и проверки файла данных

Опишем класс, облегчающий предобработку одного файла данных. В учебных целях его код оставим в этом ноутбуке. В будущем можем вынести его в отдельный файл. Заказчика кодом не удивить!

### Замысел вспомогательного класса

К разрабатываемому классу предъявим следующие требования:

- загрузка файла должна осуществляться с локального или сетевого источника (при отсутствии локального);
- параметры открытия файла через Pandas должны быть настраиваемыми;
- проверку файла по заданным критериям можно осуществлять повторно (вдруг в будущих проектах машинного обучения валидационную выборку нам сразу не предоставят или обучающую дополнят);
- отчет о проверке файла должен быть удобочитаемым.

### Код вспомогательного класса

```
In [21]: class PandasFile(object):  
    """Класс для описания одного файла, который необходимо загрузить для дальнейшего анализа"""  
  
    def __init__(self, name, file_path, file_url, pandas_params={}, data_fields=None, fix=None):  
        """Конструктор"""  
  
        self.__name = name  
        self.__file_path = file_path  
        self.__file_url = file_url  
        self.__pandas_params = pandas_params  
        self.data_fields = data_fields  
        self.data_checkings = data_checkings  
        self.__df = None  
  
        # назначение файла  
        # локальный путь к файлу  
        # сетевой адрес файла  
        # список параметров pandas.read_csv  
        # словарь информации о полях и их типах  
        # список условий автоматизированной проверки  
        # ссылка на созданный из файла датафрейм  
  
        # проверить наличие файла, загрузить его при отсутствии  
        if self.exists():  
            print(f'{self.__name} ({self.__file_path}) был загружен ранее.')  
        else:  
            self.load_from_url()  
  
        # создать датафрейм из файла  
        self.get_data_frame(reread=True)
```

```

def exists(self):
    """Проверка наличия локального файла"""
    return os.path.isfile(self.__file_path)

def load_from_url(self):
    """Загрузка файла из сетевого источника"""
    # создаем папку для сохранения файла с данными, если она ещё не создана
    folder_path = os.path.dirname(self.__file_path)
    if not os.path.exists(folder_path):
        print(f'Создаю папку {folder_path}')
        os.makedirs(folder_path)

    print(f'Загружаю "{self.__name__}" из {self.__file_url} в {self.__file_path} ...')
    result = urlretrieve(url=self.__file_url, filename=self.__file_path)
    print('OK')
    return result

def get_data_frame(self, reread=False):
    """Загрузка датафрейма Pandas из локального файла"""
    if reread or self.__df is None:
        print(f'Открываю "{self.__name__}" с помощью Pandas ... ', end=' ')
        self.__df = pd.read_csv(self.__file_path, **self.__pandas_params)
        print('OK')
        # переименование полей при необходимости
        for field_name, field_info in self.data_fields.items():
            if field_info.name_in_csv is not None:
                print(f'Переименовываю поле "{field_info.name_in_csv}" в "{field_name}"')
                self.__df.rename(columns={field_info.name_in_csv: field_name}, inplace=True)
                print('OK')
        # пересортировка столбцов в порядке их следования в списке
        self.__df = self.__df[self.data_fields.keys()]
    return self.__df

def check_data_frame_fields(self, check_try_to_fix=True, data_fields=None, prefix=''):
    """Проверка полей датасета на соответствие типам, указанным в data_fields.
    Возвращает количество несоответствий заданным требованиям.
    prefix - строка-префикс для создания ссылок на ошибки с использованием тега <a>"""
    display(HTML(f'  
Протокол автоматизированной проверки типов полей набора {self.__name__}'))

    # выбираем список с информацией о полях для проверки
    if data_fields is None:
        data_fields = self.data_fields

    # если для набора данных не заданы описания полей...
    if data_fields is None:
        display(HTML(f'В наборе данных <b>"{self.__name__}"</b> нет описания полей. Готово'))
        return -1

    # обнулим счетчик ошибок
    errors = 0

    # полный префикс ссылки тега <a>
    link_prefix = f'{self.__name__}-{prefix}'.replace(' ', '-')

    # для каждого описания поля...
    for field_name, field_info in data_fields.items():
        try:
            # проверка существования поля
            field_dtype = self.__df[field_name].dtypes

            # если тип поля отличается от предполагаемого...
            if field_dtype != field_info.dtype:
                errors += 1
                display(HTML(f'<a id="{link_prefix}{errors}"></a>'))

```

```

        f'<b>{errors}</b> Тип <b>{field_dtype}</b> поля <b>"{field_name}"</b>
        f'</b> ({field_info.desc}) не соответствует заявленному
        f'<b>{({field_info.dtype.__name__})}</b>.')
    # если необходимо попытаться изменить тип поля...
    if check_try_to_fix and field_info.try_to_fix:
        try:
            # попытаемся изменить тип поля
            self.__df[field_name] = self.__df[field_name].astype(field_info.dtype)
        except Exception as e:
            display(HTML(f'При попытке изменения типа поля произошла ошибка: {e}'))

            # проверяем тип поля после попытки его изменения
            field_dtype = self.__df.dtypes[field_name]
            if field_dtype == field_info.dtype:
                display(HTML(f'Тип поля <b>"{field_name}"</b> ({field_info.dtype.__name__})'
                            f' изменен на <b>{({field_info.dtype.__name__})}</b>.'))

    # обработка исключительных ситуаций
    except KeyError as e:
        errors += 1
        display(HTML(f'<a id="{link_prefix}{errors}"></a>' 
                    f'<b>{errors}</b> Поле с именем <b>"{field_name}"</b> в наборе данных '
                    f'набора данных не определено.'))

    if errors == 0:
        display(HTML(f'<a id="{link_prefix}{errors}"></a>' 
                    f'В наборе данных <b>"{self.__name__}"</b> ошибок в типах не обнаружено.'))

    # возвращаем общее количество обнаруженных замечаний
    return errors

def check_data_frame_values(self, check_fix=True, example_count=3, data_fields=None,
                           prefix='value error'):
    """Проверка значений датасета на соответствие условиям, указанным в data_checkings.
    Возвращает количество несоответствий заданным требованиям."""
    display(HTML(f'<br><b>Протокол автоматизированной проверки значений набора данных</b>'))

    # выбираем список с информацией об условиях проверки значений
    if data_checkings is None:
        data_checkings = self.data_checkings

    # если для набора данных не заданы условия проверки значений...
    if data_checkings is None:
        display(HTML(f'В наборе данных <b>"{self.__name__}"</b> нет условий проверки '
                    f'Проверка не проводилась.'))

    return -1

    # выбираем список с информацией о полях для проверки
    if data_fields is None:
        data_fields = self.data_fields

    # если для набора данных не заданы описания полей...
    if data_fields is None:
        display(HTML(f'В наборе данных <b>"{self.__name__}"</b> нет описания полей.
                    f'Расширенные пояснения не будут выводиться.'))

    # обнулим счетчик ошибок
    errors = 0

    # полный префикс ссылки тега <a>
    link_prefix = f'{self.__name__}-{prefix}'.replace(' ', '-')

    # для каждого условия проверки...
    for check in data_checkings.copy():
        try:
            # проверим существование полей
            for f in check.fields: field_type = self.__df[f].dtype

```

```

# отберем записи набора, которые не удовлетворяют условиям проверки
check_df = self.__df.query(check.query_template.format(*check.fields))

# определим их количество
lines_count = check_df.shape[0]

# если обнаружены нарушения...
if check_df.shape[0] > 0:
    errors += 1

# выведем текст ошибки с указанием количества записей и их доли в %
percent = lines_count / self.__df.shape[0]

# сформируем текст с перечислением
try:
    fields_list = ', '.join(map(lambda f: f"<b>'{f}'</b> " +
                                f"({{data_fields[{f}].desc}})", check.fields))
except:
    fields_list = ', '.join(map(lambda f: f"<b>'{f}'</b>", check.fields))

display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
            f'<b>{errors}</b> Обнаружены замечания к значениям {fields_list}' +
            f'{check.desc_template.format(*check.fields)}.' +
            f'Количество записей с нарушением: <b>{lines_count}</b>'))

# если обработчик ошибки задан...
if check_fix and check.fix is not None:
    # вызовем его, передав ссылку на условие проверки и датасет с ошибками
    check.fix(check, check_df, self.__df)
else:
    # иначе выведем пример ошибок в табличном виде
    if example_count > 0:
        display(check_df[check.fields].head(example_count))

# "назад в будущее" - создадим гиперссылку на блок исправления ошибки
display(HTML(f'<p><a href="#{link_prefix}{errors}-fix">См. исправление <b>{errors}</b></a></p>'))

# обработка исключительных ситуаций
except KeyError as e:
    errors += 1
    display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                f'<b>{errors}</b> Поле с именем <b>"{e}"</b> в датасете не существует'))

except NameError as e:
    errors += 1
    # определим имя поля, совпадающее с зарезервированным словом,
    # из последних скобок сообщения NameError
    text = str(e)[-1]
    reserved_name = text[text.find('(') + 1 : text.find(')')[-1]]
    display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                f'<b>{errors}</b> Имя поля <b>{reserved_name}</b> совпадает с зарезервированным <b>{reserved_name}</b> словом. Переименуйте его для возможности использовать в датафрейме.' +
                f'<b>pandas.query</b>. Проверка поля <b>{reserved_name}</b> в датафрейме не прошла'))

if errors == 0:
    display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                f'В наборе данных <b>"{self.__name}"</b> ошибок в значениях нет'))

# возвращаем общее количество обнаруженных замечаний
return errors

@property
def df(self):

```

```
"""Возвращает ссылку на последнюю созданную копию датасета Pandas"""
return self.get_data_frame(reread=False)
```

## Наиболее употребимые элементы условий проверки

Опишем наиболее часто употребимые элементы условий проверки. Вместо `{0}`, `{1}` и так далее программа ([см. ниже](#)) будет подставлять имя проверяемого поля из `fields` по индексу, начиная с нуля:

```
In [22]: # проверка на пустое значение одного поля
desc_template_nan      = 'Значение {0} не должно быть пустым'
query_template_nan     = '`{0}` != `{0}`'

# проверка на пустое значение двух полей одновременно
desc_template_nan_2    = 'Значения {0} и {1} не должны быть пустыми одновременно'
query_template_nan_2   = '(`{0}` != `{0}`) and (`{1}` != `{1}`)'

# другие частые проверки
desc_template_g_zero   = 'Значение "{0}" должно быть больше нуля'
query_template_g_zero  = '`{0}` <= 0'

desc_template_ge_zero  = 'Значение "{0}" должно быть не меньше нуля'
query_template_ge_zero = '`{0}` < 0'

# Этажность
desc_template_floors   = 'Значение {0}, как правило, от 1 до 60'
query_template_floors  = '(`{0}` == 0) or (`{0}` > 60)'

# количество объектов рядом
desc_template_nearest   = 'Значение {0}, как правило, не больше 5'
query_template_nearest  = '(`{0}` > 5)'

# расстояние до объектов
desc_template_dist      = 'Значение {0} должно быть не более 3000 км'
query_template_dist     = '`{0}` > 3000'

# расстояние до центра Санкт-Петербурга
desc_template_centre    = 'Значение {0}, как правило, до 50 км'
query_template_centre   = '(`{0}` >= 0) and (`{0}` > 50000)'
```

## Специфические для полученного набора данных условия проверки

Проверочные условия зададим, исходя из здравого смысла и опыта специалистов в сфере торговли недвижимостью. Например, не может квартира находится выше последнего этажа дома, а дом на отрицательном расстоянии от центра, ближе 5 км к аэропорту и т.п.

Недоверие должна вызывать и жилая площадь меньше социальных норм.

Выявление подобных аномалий на ранней стадии очень важно как для анализа и генерации новых атрибутов по имеющимся данным. Их искаженные значения могут отрицательно сказаться на качестве машинного обучения, к которому мы когда-нибудь подберемся.

В следующих проектах попробуем реализовать запись условий проверки в форме, максимально приближенной к естественному языку. А пока воспользуемся описанной [выше](#) структурой `DataChecking`.

```
In [23]: # создадим пустой список условий проверки
data_checkings = []

# для всех полей проверим наличие пропусков
for field in data_fields.keys():
    data_checkings += [DataChecking([field], desc_template_nan, query_template_nan,
                                    lambda _, __, ___: None)] # не будем выводить таблицу примеров

# для некоторых полей проверим наличие пропусков в них одновременно
```

```

for fields in [['total_area', 'living_area'], ['total_area', 'kitchen_area'], ['living_area', 'floors_total', 'the_floor']]:
    data_checkings += [DataChecking(fields, desc_template_nan_2, query_template_nan_2,
                                    lambda _, __, ___: None)]      # не будем выводить таблицу примеров

# для некоторых числовых полей проверим наличие отрицательных значений
for field in ['total_images', 'kitchen_area', 'living_area', 'airports_nearest', 'ceilings_height',
              'cityCenters_nearest', 'parks_nearest', 'ponds_nearest', 'parks_around3000',
              'ponds_around3000', 'rooms', 'balcony', 'last_price']:
    data_checkings += [DataChecking([field], desc_template_ge_zero, query_template_ge_zero)]

# для некоторых числовых полей проверим наличие нулевых и отрицательных значений
for field in ['floors_total', 'the_floor', 'days_exposition', 'total_area']:
    data_checkings += [DataChecking([field], desc_template_g_zero, query_template_g_zero)]

# проверим поле расстояния до центра города
locality_check = DataChecking(['cityCenters_nearest'], desc_template_centre, query_template_centre,
                               lambda c, cdf, __: (cdf.hist(c.fields, bins=20), plt.show(),
                                                    cdf.groupby('locality_name')[c.fields]
                                                    .count().plot.pie(y=c.fields[0],
                                                                      legend=None),
                                                    plt.show()), data_checkings.remove(locality_check))

data_checkings += [locality_check]

# проверим поля расстояний до объектов
distances_checkings = \
[
    DataChecking(['parks_nearest'], desc_template_dist, query_template_dist, None),
    DataChecking(['ponds_nearest'], desc_template_dist, query_template_dist, None)
]

data_checkings += distances_checkings

# проверим даты публикаций
date_check = DataChecking(['first_day_exposition'],
                          'Дата публикации предположительно должна быть с 2016 по 2020 год',
                          '(`{0}`.dt.year < 2016) or (`{0}` > "2020-10-01")',
                          lambda _, cdf, __: (cdf.first_day_exposition.hist(bins=20), plt.show(),
                                               data_checkings.remove(date_check)))

data_checkings += [date_check]

# добавим остальные условия проверки, принимая во внимание смысл переменных
data_checkings += \
[
    DataChecking(['airports_nearest'],
                'Расстояние до аэропорта должно быть не менее 5000 м по санитарным нормам',
                '(`{0}` >= 0) and (`{0}` < 5000)',
                lambda _, cdf, __: display(cdf[['airports_nearest', 'locality_name']]),
    DataChecking(['ceiling_height'], 'Высота потолков не меньше 2.3 м по санитарным нормам',
                '(`{0}` >= 0) and (`{0}` < 2.3)',
                lambda c, cdf, __: (cdf.hist(c.fields), plt.show())),
    DataChecking(['parks_around3000'], desc_template_nearest, query_template_nearest,
                lambda c, cdf, __: (cdf.hist(c.fields, bins=20), plt.show())),
    DataChecking(['ponds_around3000'], desc_template_nearest, query_template_nearest,
                lambda c, cdf, __: (cdf.hist(c.fields, bins=20), plt.show())),
    DataChecking(['floors_total'], desc_template_floors, query_template_floors,
                lambda c, cdf, __: (cdf.hist(c.fields), plt.show())),
    DataChecking(['the_floor'], desc_template_floors, query_template_floors,
                lambda c, cdf, __: (cdf.hist(c.fields), plt.show())),
    DataChecking(['the_floor', 'floors_total'],
                'Этаж не может быть выше крыши дома',
                '(`{0}` > 0) and (`{1}` > 0) and (`{0}` > `{1}`)', None),
    DataChecking(['rooms', 'studio'],
                'Количество комнат в обычной квартире или со свободной планировкой (не должны быть нулевым',
                '(`{0}` == 0) and (not `{1}`)', lambda _, __, ___: None),      # не будем выводить таблицу примеров
]

```

```

DataChecking(['rooms', 'studio'],
            'В студии по определению должна быть одна комната',
            '(`{0}` > 0) and (`{0}` != 1)) and `{1}`',
            lambda _, __, ___: None),      # не будем выводить таблицу примеров
DataChecking(['studio', 'open_plan'],
            'Странно, что помещение имеет свободную планировку и одновременно обозначено как комната',
            '``{0}` and `{1}`', lambda _, __, ___: None),
DataChecking(['balcony', 'rooms', 'studio', 'open_plan'],
            'Количество балконов, как правило, не больше количества комнат + 2 (когда есть комната и балкон, а также студия)',
            '(`{0}` > 0) and (`{0}` > 0) and (`{0}` > (`{1}` + 2)) and `(`not `{2}`) and (not `{3}`)', None),
DataChecking(['balcony', 'the_floor'],
            'Количество балконов на первом этаже, как правило, не больше одного',
            '(`{0}` > 0) and (`{0}` != 1) and (`{1}` == 1)',
            lambda c, cdf, __ : (cdf.hist(c.fields[0]), plt.show())),
DataChecking(['last_price'],
            'Цена, как правило, больше 100 тыс. по здравому смыслу',
            '(`{0}` > 0) and (`{0}` <= 100000)', None),
DataChecking(['last_price'],
            'Цена, как правило, не больше 500 млн. по здравому смыслу',
            '(`{0}` > 5000000000)',  

            lambda c, cdf, __ : display(cdf[['last_price', 'rooms', 'living_area',
                                              'kitchen_area', 'total_area', 'ceiling_height',
                                              'days_exposition']])),
DataChecking(['total_images'],
            'Число фотографий квартиры в объявлении, как правило, от 0 до 50',
            '(`{0}` >= 0) and (`{0}` > 50)', None),
DataChecking(['days_exposition'],
            'Срок публикации, как правило, не больше 5 лет',
            '(`{0}` > 5*395)', None),
DataChecking(['kitchen_area', 'studio', 'open_plan'],
            'Площадь кухни в обычной квартире, как правило, не меньше 5 м2 по санитарным нормам',
            '(`{0}` >= 0) and (`{0}` < 5) and (not `{1}`) and (not `{2}`)',  

            lambda c, cdf, __ : (cdf.hist(c.fields[0]), plt.show())),
DataChecking(['kitchen_area', 'total_area'],
            'Площадь кухни должна быть меньше общей площади',
            '(`{0}` >= 0) and (`{0}` >= `{1}`)', None),
DataChecking(['living_area'],
            'Жилая площадь, как правило, не меньше 8 м2 по санитарным нормам',
            '(`{0}` >= 0) and (`{0}` < 8)',  

            lambda c, cdf, __ : (cdf.hist(c.fields), plt.show())),
DataChecking(['living_area', 'total_area'],
            'Жилая площадь должна быть не больше общей',
            '(`{0}` >= 0) and (`{0}` > `{1}`)', None),
DataChecking(['total_area'],
            'Общая площадь, как правило, не меньше 12 м2 по санитарным нормам',
            '(`{0}` > 0) and (`{0}` < 12)',  

            lambda c, cdf, __ : (cdf.hist(c.fields), plt.show())),
DataChecking(['total_area'],
            'Общая площадь, как правило, не больше 500 м2 по здравому смыслу',
            '``{0}` > 500',  

            lambda c, cdf, __ : display(cdf[['last_price', 'rooms', 'living_area',
                                              'kitchen_area', 'total_area', 'ceiling_height']])),
DataChecking(['living_area'],
            'Жилая площадь, как правило, не больше 300 м2 по здравому смыслу',
            '``{0}` > 300',  

            lambda c, cdf, __ : display(cdf[['last_price', 'rooms', 'living_area',
                                              'kitchen_area', 'total_area', 'ceiling_height']])),
DataChecking(['total_area', 'living_area', 'kitchen_area'],
            'Общая площадь не должна быть меньше суммы жилой и кухни',
            '(`{0}` >= 0) and (`{1}` >= 0) and (`{2}` >= 0) and `(`(`{1}` + `{2}` - `{0}`) > 0.1)', None),
DataChecking(['ceiling_height', 'studio'],
            'Высота потолков в обычной квартире (не студии), как правило, не больше 3 м',
            '(`{0}` > 10) and (not `{1}`)',  

            lambda c, cdf, __ : display(cdf[['last_price', 'rooms', 'living_area',
                                              'kitchen_area', 'total_area', 'ceiling_height']])),

```

```

        'kitchen_area', 'total_area', 'ceiling_height'
DataChecking(['ceiling_height', 'studio'],
             'Высота потолков в студии, как правило, не больше 7 м',
             '{0} > 15) and {1}',
             lambda c, cdf, __ : display(cdf[['last_price', 'rooms', 'living_area',
                                                'kitchen_area', 'total_area', 'ceiling_height']])
DataChecking(['kitchen_area'],
             'Площадь кухни, как правило, не больше 70 м2',
             '{0} > 70',
             lambda c, cdf, __ : display(cdf[['last_price', 'rooms', 'living_area',
                                                'kitchen_area', 'total_area', 'ceiling_height']])
]

```

Отсортируем наши условия проверки по названию полей для удобства их дальнейшей обработки:

```
In [24]: data_checkings.sort(key = lambda x : ' '.join(x.fields))
```

```
In [25]: len(data_checkings)
```

```
Out[25]: 74
```

## Параметры загрузки и открытия файлов

Зададим необходимый набор параметров для открытия файла данных нашего проекта с помощью `pandas.read_csv`:

```
In [26]: pandas_params = \
{
    'sep': '\t',                                     # разделитель полей - табуляция
    'parse_dates': ['first_day_exposition'],          # список полей даты и времени
}
```

Зададим локальную папку для хранения файлов данных (по умолчанию `'datasets'`). При необходимости укажите более удобное место:

```
In [27]: local_folder = 'datasets'
```

## Загрузка файлов данных

Зададим список всех файлов данных нашего проекта. В данном случае файл один, но в будущем их будет гораздо больше (тогда список можно будет легко пополнить). Выполнение следующей ячейки должно привести к загрузке всех отсутствующих локально файлов и созданию датафреймов, как было описано в [замысле выполнения шага 1](#) практической работы:

```
In [28]: data_files = \
{
    'Основной файл':
        PandasFile(
            (
                'Информация об объявлениях',                      # назначение
                os.path.join(local_folder, 'real_estate_data.csv'), # локальный
                '',                                                 # сетевой адрес файла не показан в отчёте из-за правовых ограничений
                pandas_params,                                     # список параметров
                data_fields,                                       # словарь полей
                data_checkings                                    # список проверок
            )
}
```

"Информация об объявлениях" (datasets\real\_estate\_data.csv) был загружен ранее.  
Открываю "Информация об объявлениях" с помощью Pandas ... OK  
Переименовываю поле "floor" в "the\_floor" ... OK

# Ручная проверка файлов данных

Сохраним для удобства короткую ссылку на загруженный набор данных Pandas:

```
In [29]: ads = data_files['Основной файл'].df
```

## Вывод краткой информации о наборе данных

Выведем краткую информацию о наборе данных с помощью `info`:

```
In [30]: ads.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23699 entries, 0 to 23698
Data columns (total 22 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   rooms            23699 non-null   int64  
 1   living_area      21796 non-null   float64 
 2   kitchen_area     21421 non-null   float64 
 3   total_area       23699 non-null   float64 
 4   ceiling_height  14504 non-null   float64 
 5   studio           23699 non-null   bool    
 6   open_plan        23699 non-null   bool    
 7   is_apartment     2775 non-null   object  
 8   balcony          12180 non-null   float64 
 9   the_floor         23699 non-null   int64  
 10  floors_total    23613 non-null   float64 
 11  locality_name   23650 non-null   object  
 12  cityCenters_nearest  18180 non-null   float64 
 13  parks_around3000 18181 non-null   float64 
 14  parks_nearest    8079 non-null   float64 
 15  ponds_around3000 18181 non-null   float64 
 16  ponds_nearest    9110 non-null   float64 
 17  airports_nearest 18157 non-null   float64 
 18  first_day_exposition 23699 non-null   datetime64[ns] 
 19  days_exposition  20518 non-null   float64 
 20  last_price       23699 non-null   float64 
 21  total_images     23699 non-null   int64  
dtypes: bool(2), datetime64[ns](1), float64(14), int64(3), object(2)
memory usage: 3.7+ MB
```

Можно сделать вывод, что файл успешно загружен в память. В таблице содержится **23699 строк, 22 столбца**. Многие поля имеют **пропуски**. Так, для 49 объявлений не указан населенный пункт, придётся от них избавиться. Некоторые поля имеют неверный тип - **вещественный** вместо **целого** (количество этажей, балконов, дней публикации объявлений и т.д.). После предобработки мы должны получить массив с преобразованными типами полей, адекватных смыслу хранящихся в них значений. ([спойлер](#))

## Просмотр первых строк набора данных

Разобьем поля набора данных **на несколько групп** для удобства их отображения и анализа:

```
In [31]: feature_groups = \
{
    'Размеры': ['rooms', 'living_area', 'kitchen_area', 'total_area', 'ceiling_height'],
    'Планировка': ['studio', 'open_plan', 'is_apartment', 'balcony', 'the_floor', 'floors_total'],
    'Место': ['locality_name', 'cityCenters_nearest', 'parks_around3000', 'parks_nearest',
              'ponds_around3000', 'ponds_nearest', 'airports_nearest'],
    'Объявление': ['first_day_exposition', 'days_exposition', 'last_price', 'total_images']
}
```

Опишем функцию `process_field_groups`, выполняющую другую функцию-параметр `func` над группами полей:

```
In [32]: def process_field_groups(func, feature_groups=feature_groups):
    for group_name, columns in feature_groups.items():
        func(group_name, ads[columns])
```

Выведем **первые несколько записей** таблицы с помощью `head()`:

```
In [33]: process_field_groups(func=lambda n, c: (display(HTML(f'  
{n}')), display(c.head(n))))
```

### Размеры

	rooms	living_area	kitchen_area	total_area	ceiling_height
0	3	51.0	25.0	108.0	2.70
1	1	18.6	11.0	40.4	NaN
2	2	34.3	8.3	56.0	NaN
3	3	NaN	NaN	159.0	NaN
4	2	32.0	41.0	100.0	3.03

### Планировка

	studio	open_plan	is_apartment	balcony	the_floor	floors_total
0	False	False	NaN	NaN	8	16.0
1	False	False	NaN	2.0	1	11.0
2	False	False	NaN	0.0	4	5.0
3	False	False	NaN	0.0	9	14.0
4	False	False	NaN	NaN	13	14.0

### Место

	locality_name	cityCenters_nearest	parks_around3000	parks_nearest	ponds_around3000	ponds_nearest
0	Санкт-Петербург	16028.0	1.0	482.0	2.0	751.0
1	посёлок Шушары	18603.0	0.0	NaN	0.0	NaN
2	Санкт-Петербург	13933.0	1.0	90.0	2.0	57.0
3	Санкт-Петербург	6800.0	2.0	84.0	3.0	23.0
4	Санкт-Петербург	8098.0	2.0	112.0	1.0	48.0

### Объявление

	first_day_exposition	days_exposition	last_price	total_images
0	2019-03-07	NaN	13000000.0	20
1	2018-12-04	81.0	3350000.0	7
2	2015-08-20	558.0	5196000.0	10

	first_day_exposition	days_exposition	last_price	total_images
3	2015-07-24	424.0	64900000.0	0
4	2018-06-19	121.0	10000000.0	2

## Изучение параметров распределения числовых данных

Изучим параметры распределения числовых данных с помощью функции `describe`.

In [34]: `process_field_groups(func=lambda n, c: (display(HTML(f'  
{n}')), display(c.describe())) for n, c in zip(names, df.select_dtypes('number')))`

### Размеры

	rooms	living_area	kitchen_area	total_area	ceiling_height
<b>count</b>	23699.000000	21796.000000	21421.000000	23699.000000	14504.000000
<b>mean</b>	2.070636	34.457852	10.569807	60.348651	2.771499
<b>std</b>	1.078405	22.030445	5.905438	35.654083	1.261056
<b>min</b>	0.000000	2.000000	1.300000	12.000000	1.000000
<b>25%</b>	1.000000	18.600000	7.000000	40.000000	2.520000
<b>50%</b>	2.000000	30.000000	9.100000	52.000000	2.650000
<b>75%</b>	3.000000	42.300000	12.000000	69.900000	2.800000
<b>max</b>	19.000000	409.700000	112.000000	900.000000	100.000000

### Планировка

	balcony	the_floor	floors_total
<b>count</b>	12180.000000	23699.000000	23613.000000
<b>mean</b>	1.150082	5.892358	10.673824
<b>std</b>	1.071300	4.885249	6.597173
<b>min</b>	0.000000	1.000000	1.000000
<b>25%</b>	0.000000	2.000000	5.000000
<b>50%</b>	1.000000	4.000000	9.000000
<b>75%</b>	2.000000	8.000000	16.000000
<b>max</b>	5.000000	33.000000	60.000000

### Место

	cityCenters_nearest	parks_around3000	parks_nearest	ponds_around3000	ponds_nearest	airport
<b>count</b>	18180.000000	18181.000000	8079.000000	18181.000000	9110.000000	18181.000000
<b>mean</b>	14191.277833	0.611408	490.804555	0.770255	517.980900	2879.000000
<b>std</b>	8608.386210	0.802074	342.317995	0.938346	277.720643	1261.000000
<b>min</b>	181.000000	0.000000	1.000000	0.000000	13.000000	181.000000
<b>25%</b>	9238.000000	0.000000	288.000000	0.000000	294.000000	1851.000000
<b>50%</b>	13098.500000	0.000000	455.000000	1.000000	502.000000	2671.000000

	cityCenters_nearest	parks_around3000	parks_nearest	ponds_around3000	ponds_nearest	airport
75%	16293.000000	1.000000	612.000000	1.000000	729.000000	372
max	65968.000000	3.000000	3190.000000	3.000000	1344.000000	848

## Объявление

	days_exposition	last_price	total_images
count	20518.000000	2.369900e+04	23699.000000
mean	180.888634	6.541549e+06	9.858475
std	219.727988	1.088701e+07	5.682529
min	1.000000	1.219000e+04	0.000000
25%	45.000000	3.400000e+06	6.000000
50%	95.000000	4.650000e+06	9.000000
75%	232.000000	6.800000e+06	14.000000
max	1580.000000	7.630000e+08	50.000000

Бегло изучим список **населенных пунктов**:

In [35]: `ads.locality_name.value_counts()`

Out[35]:

Санкт-Петербург	15721
посёлок Мурино	522
посёлок Шушары	440
Всеволожск	398
Пушкин	369
...	
деревня Луполово	1
поселок Коробицыно	1
деревня Раполово	1
посёлок Петро-Славянка	1
деревня Тойворово	1

Name: locality\_name, Length: 364, dtype: int64

Обратим внимание на то, что в поле `'locality_name'` фактически хранится две категориальные переменные:

- тип населенного пункта;
- название населенного пункта.

Для повышения качества анализа разобъем их на два поля [на этапе предобработки данных](#).

## Вывод по шагу 1

- Полученный файл имеет корректный формат, удобный для обработки с помощью библиотеки Pandas.
- Файл успешно [загружен](#) в память. В таблице содержится 23699 строк и 22 столбца.
- Названия столбцов не искажены, соответствуют полученному в задании описанию и могут быть использованы для упрощенной адресации Pandas в виде `data.locality_name`, а не только `data['locality_name']`. Столбец `'floor'` был переименован в `'the_floor'` для возможности использования его имени в запросах `pandas.DataFrame.query`.

- Массив данных в целом пригоден для анализа, но требует предобработки, включая как минимум:
  - устранение пропусков во многих полях (некоторые записи придется удалить целиком, например, 49 объявлений без указания населенного пункта);
  - приведение вещественных типов к целому для нескольких полей (например, количество этажей, балконов, дней публикации объявлений).
  - контроль правильности значений в соответствии с ограничениями на характеристики квартир. Так, например, имеются нереальные или маловероятные данные о:
    - кухнях слишком большой площади ( $112\text{ м}^2$ );
    - потолках высотой в 1 м;
    - нулевом расстоянии до аэропорта;
    - подозрительном соотношении общей и жилой площади, а также размера кухни;
    - слишком большом удалении от центра (65968 м).
- Для упрощения анализа столбец с названием населенного пункта необходимо разбить на два: тип (например, 'поселок', 'деревня') и собственно название.
- Для облегчения выполнения предобработки был [подготовлен](#) класс, который в соответствии со списком ограничений на данные построит отчет о найденных ошибках и аномалиях. Их устранение целесообразно осуществить на следующем шаге.
- После корректировки значений набора данных следует проверить наличие дубликатов и принять решение об их удалении.

## Шаг 2. Предобработка данных

### Задание шага 2

Определите и изучите пропущенные значения:

- для некоторых пропущенных значений можно предположить логичную замену. Например, если человек не указал число балконов — скорее всего, их нет. Такие пропуски правильно заменить на 0. Для других типов данных нет подходящего значения на замену. В этом случае правильно оставить эти значения пустыми. Отсутствие значения — тоже важный сигнал, который не нужно прятать;
- заполните пропуски, где это уместно. Опишите, почему вы решили заполнить пропуски именно в этих столбцах и как выбрали значения;
- укажите причины, которые могли привести к пропускам в данных.

Приведите данные к нужным типам: поясните, в каких столбцах нужно изменить тип данных и почему.

### Поиск пропусков, ошибок и аномалий в данных

#### Первая автоматизированная проверка типов полей

Выполнив ручную проверку, мы убедились в наличии достаточно большого количества замечаний к массиву данных. Убедимся в эффективности придуманной автоматизированной проверки. Начнем с изучения типов полей загруженных файлов с помощью разработанного выше класса:

```
In [36]: def auto_check_all_fields(check_try_to_fix=True):
    for f in data_files.values():
        f.check_data_frame_fields(check_try_to_fix=check_try_to_fix)
```

```
In [37]: auto_check_all_fields()
```

## Протокол автоматизированной проверки типов полей набора данных "Информация об объявлениях":

1. Тип **object** поля "**is\_apartment**" (апартаменты) не соответствует заявленному в описании **bool**.

Тип поля "**is\_apartment**" (апартаменты) изменен на **bool**.

2. Тип **float64** поля "**balcony**" (число балконов) не соответствует заявленному в описании **int64**.

При попытке изменения типа поля произошла ошибка: *Cannot convert non-finite values (NA or inf) to integer*.

3. Тип **float64** поля "**floors\_total**" (всего этажей в доме) не соответствует заявленному в описании **int64**.

При попытке изменения типа поля произошла ошибка: *Cannot convert non-finite values (NA or inf) to integer*.

4. Тип **float64** поля "**parks\_around3000**" (число парков в радиусе 3 км) не соответствует заявленному в описании **int64**.

При попытке изменения типа поля произошла ошибка: *Cannot convert non-finite values (NA or inf) to integer*.

5. Тип **float64** поля "**ponds\_around3000**" (число водоёмов в радиусе 3 км) не соответствует заявленному в описании **int64**.

При попытке изменения типа поля произошла ошибка: *Cannot convert non-finite values (NA or inf) to integer*.

6. Тип **float64** поля "**days\_exposition**" (сколько дней было размещено объявление) не соответствует заявленному в описании **int64**.

При попытке изменения типа поля произошла ошибка: *Cannot convert non-finite values (NA or inf) to integer*.

Получили внушительный список замечаний по набору данных. Их поиск вручную мог бы затянуться. Обратим внимание, что автоматическое преобразование большинства полей не произошло из-за наличия в них пропусков. А вот поле `is_apartment` (апартаменты) теперь имеет булев тип. Повторный запуск проверки не выявит замечаний к нему (попытаться изменить тип других полей пока не будем):

```
In [38]: auto_check_all_fields(check_try_to_fix=False)
```

## Протокол автоматизированной проверки типов полей набора данных "Информация об объявлениях":

1. Тип **float64** поля "**balcony**" (число балконов) не соответствует заявленному в описании **int64**.

2. Тип **float64** поля "**floors\_total**" (всего этажей в доме) не соответствует заявленному в описании **int64**.

3. Тип **float64** поля "**parks\_around3000**" (число парков в радиусе 3 км) не соответствует заявленному в описании **int64**.

4. Тип **float64** поля "**ponds\_around3000**" (число водоёмов в радиусе 3 км) не соответствует заявленному в описании **int64**.

5. Тип **float64** поля "**days\_exposition**" (сколько дней было размещено объявление) не соответствует заявленному в описании **int64**.

Вот и оправдались временные затраты на реализацию [замысла выполнения первого шага](#). На следующих этапах предобработки данных после исправления очередной порции ошибок и

пропусков будем периодически осуществлять автоматическую проверку и радостно наблюдать за сокращением протокола.

## Первая автоматизированная проверка значений

Проверим значения массивов данных на соответствие заданным [выше](#) ограничениям, сопроводив результаты поясняющими графиками и диаграммами при необходимости:

```
In [39]: def auto_check_all_values(check_fix, prefix):
    for f in data_files.values():
        f.check_data_frame_values(check_fix=check_fix, prefix=prefix)
```

Обнаруженные замечания будем автоматически помечать перекрестными гиперссылками с идентификаторами в формате `test 1 value error-{N}` (обнаруженная ошибка) и `test 1 value error-{N}-fix` (обработка ошибки), где `{N}` - номер позиции в протоколе. Это позволит нам удобно продолжить предобработку данных, переходя по этим ссылкам.

```
In [40]: auto_check_all_values(check_fix=True, prefix='test 1 value error')
```

### Протокол автоматизированной проверки значений набора данных "Информация об объявлениях":

1. Обнаружены замечания к значениям '`airports_nearest`' (расстояние до ближайшего аэропорта в метрах). Значение `airports_nearest` не должно быть пустым. Количество записей с нарушением: **5542 (23.38%)**.

[См. исправления и комментарии по п. 1](#)

2. Обнаружены замечания к значениям '`airports_nearest`' (расстояние до ближайшего аэропорта в метрах). Расстояние до аэропорта должно быть не менее 5000 м по санитарным нормам. Количество записей с нарушением: **1 (0.00%)**.

<code>airports_nearest</code>	<code>locality_name</code>	<code>living_area</code>
21085	0.0 Санкт-Петербург	19.8

[См. исправления и комментарии по п. 2](#)

3. Обнаружены замечания к значениям '`balcony`' (число балконов). Значение `balcony` не должно быть пустым. Количество записей с нарушением: **11519 (48.61%)**.

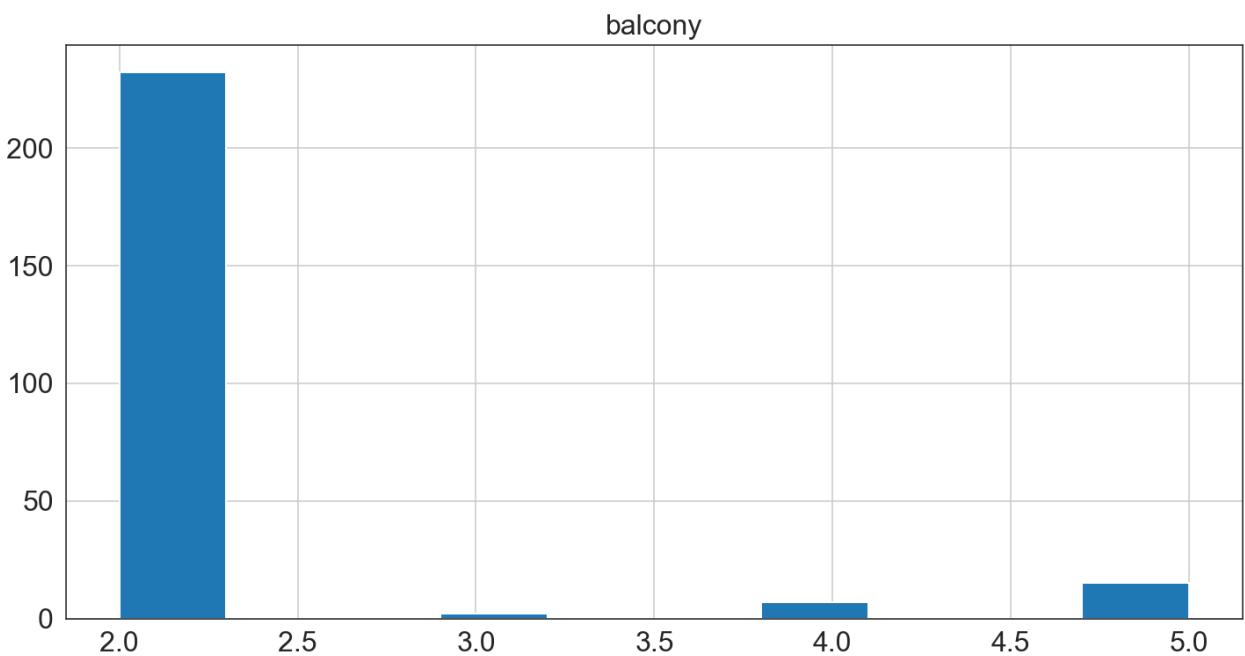
[См. исправления и комментарии по п. 3](#)

4. Обнаружены замечания к значениям '`balcony`' (число балконов), '`rooms`' (число комнат), '`studio`' (квартира-студия), '`open_plan`' (свободная планировка). Количество балконов, как правило, не больше количества комнат + 2 (кухня, коридор), если не свободная планировка и не студия. Количество записей с нарушением: **124 (0.52%)**.

	<code>balcony</code>	<code>rooms</code>	<code>studio</code>	<code>open_plan</code>
371	5.0	2	False	False
654	4.0	1	False	False
845	5.0	2	False	False

[См. исправления и комментарии по п. 4](#)

5. Обнаружены замечания к значениям '`balcony`' (число балконов), '`the_floor`' (этаж). Количество балконов на первом этаже, как правило, не больше одного. Количество записей с нарушением: **256 (1.08%)**.



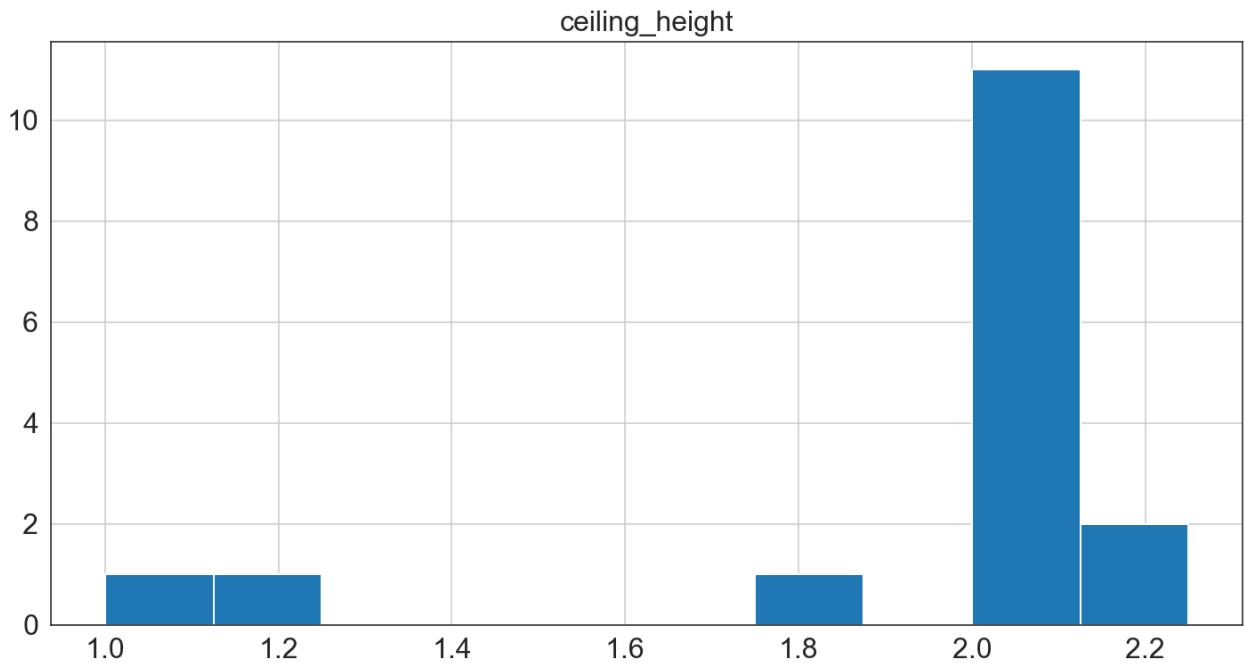
[См. исправления и комментарии по п. 5](#)

**6.** Обнаружены замечания к значениям '**ceiling\_height**' (высота потолков в метрах).

Значение **ceiling\_height** не должно быть пустым. Количество записей с нарушением: **9195 (38.80%)**.

[См. исправления и комментарии по п. 6](#)

**7.** Обнаружены замечания к значениям '**ceiling\_height**' (высота потолков в метрах). Высота потолков не меньше 2.3 м по санитарным нормам. Количество записей с нарушением: **16 (0.07%)**.



[См. исправления и комментарии по п. 7](#)

**8.** Обнаружены замечания к значениям '**ceiling\_height**' (высота потолков в метрах), '**studio**' (квартира-студия). Высота потолков в обычной квартире (не студии), как правило, не больше 10 м. Количество записей с нарушением: **25 (0.11%)**.

	<u>last_price</u>	<u>rooms</u>	<u>living_area</u>	<u>kitchen_area</u>	<u>total_area</u>	<u>ceiling_height</u>
<b>355</b>	3600000.0	2	32.0	NaN	55.2	25.0
<b>3148</b>	2900000.0	3	53.0	8.0	75.0	32.0
<b>4643</b>	4300000.0	2	30.0	7.0	45.0	25.0

	last_price	rooms	living_area	kitchen_area	total_area	ceiling_height
<b>4876</b>	3000000.0	0	17.0	NaN	25.0	27.0
<b>5076</b>	3850000.0	1	19.5	5.5	30.5	24.0
...	...	...	...	...	...	...
<b>21824</b>	2450000.0	2	38.0	8.6	44.0	27.0
<b>22309</b>	5300000.0	1	15.5	NaN	45.0	10.3
<b>22336</b>	9999000.0	2	55.5	16.5	92.4	32.0
<b>22869</b>	15000000.0	1	14.0	11.0	25.0	100.0
<b>22938</b>	4000000.0	4	73.0	9.0	98.0	27.0

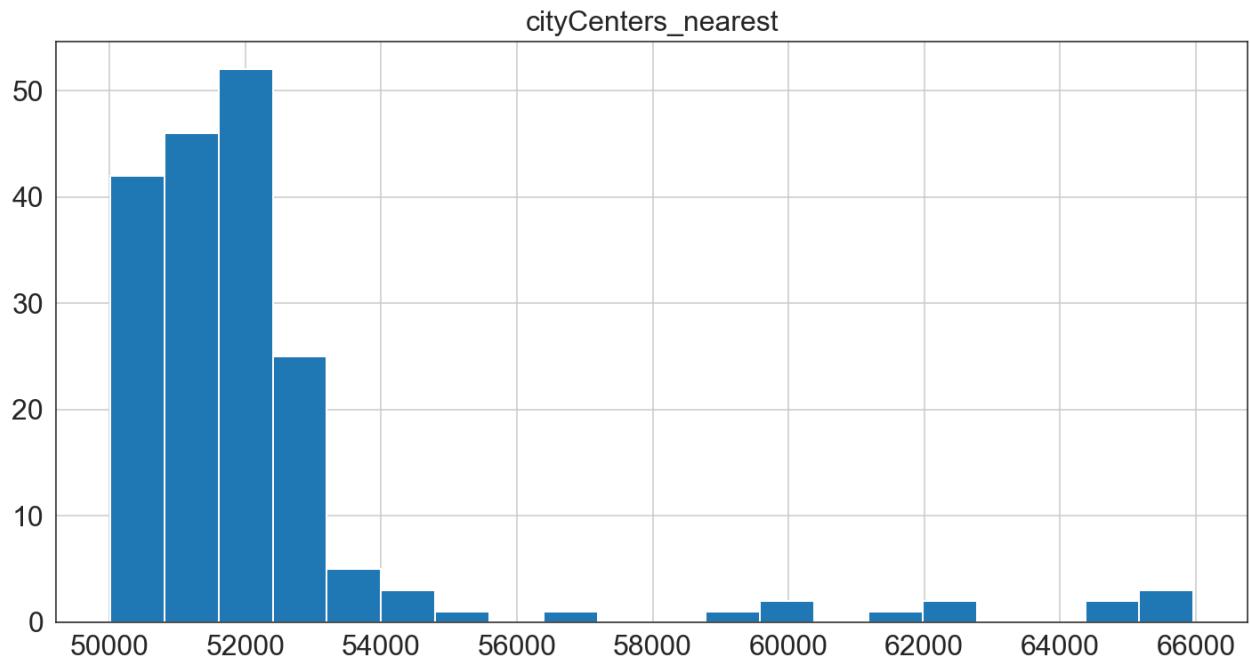
25 rows × 6 columns

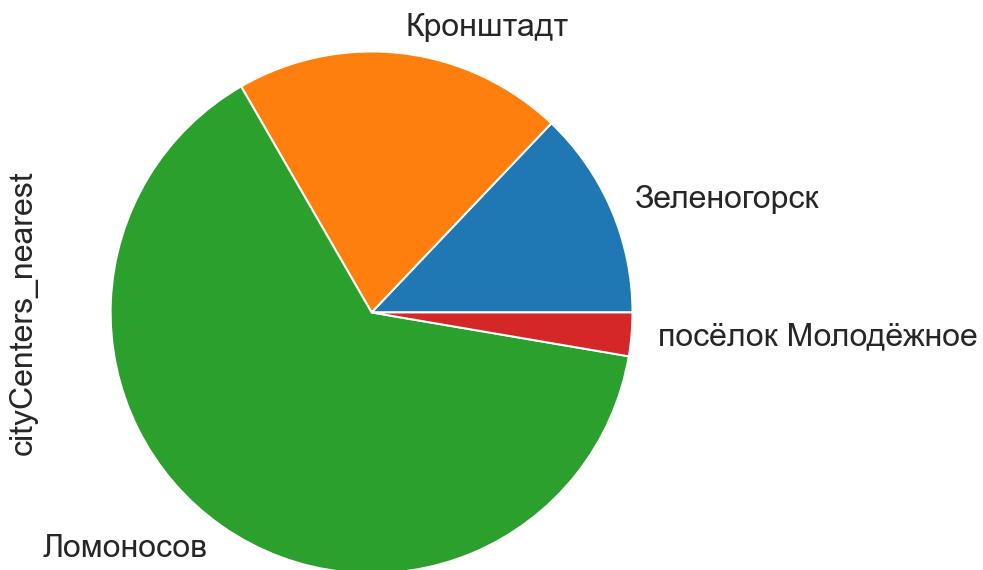
[См. исправления и комментарии по п. 8](#)

**9.** Обнаружены замечания к значениям '**cityCenters\_nearest**' (расстояние до центра города в метрах). Значение cityCenters\_nearest не должно быть пустым. Количество записей с нарушением: **5519 (23.29%)**.

[См. исправления и комментарии по п. 9](#)

**10.** Обнаружены замечания к значениям '**cityCenters\_nearest**' (расстояние до центра города в метрах). Значение cityCenters\_nearest, как правило, до 50 км. Количество записей с нарушением: **186 (0.78%)**.



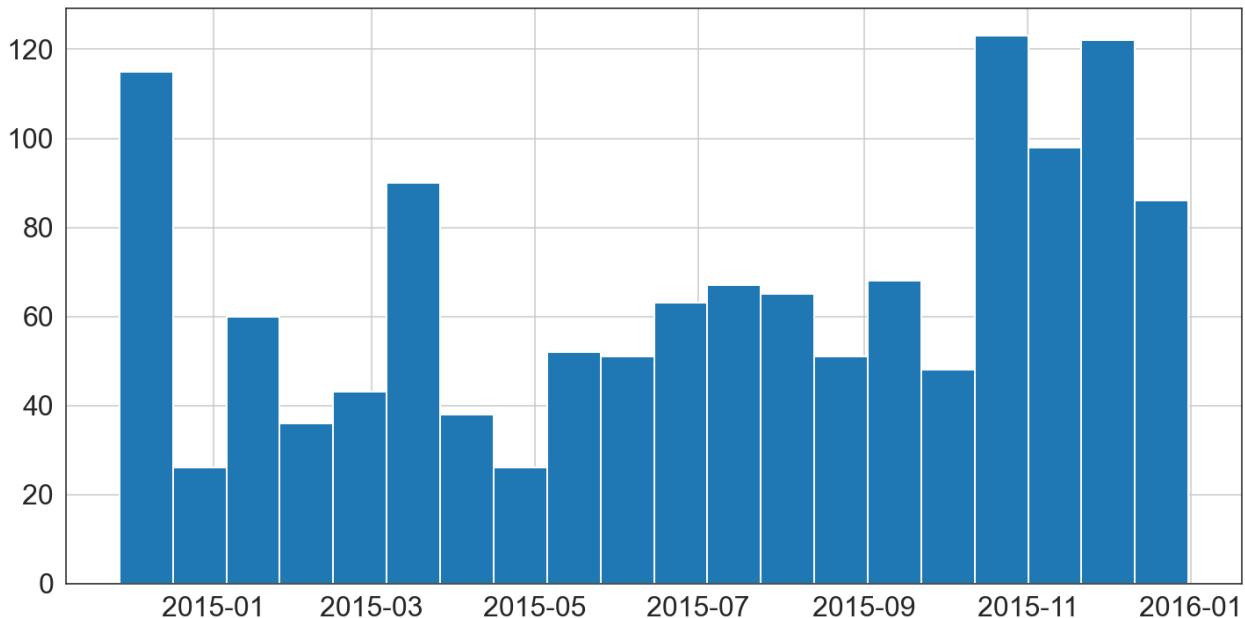


[См. исправления и комментарии по п. 10](#)

**11.** Обнаружены замечания к значениям '**days\_exposition**' (сколько дней было размещено объявление). Значение days\_exposition не должно быть пустым. Количество записей с нарушением: **3181 (13.42%)**.

[См. исправления и комментарии по п. 11](#)

**12.** Обнаружены замечания к значениям '**first\_day\_exposition**' (дата публикации). Дата публикации предположительно должна быть с 2016 по 2020 год. Количество записей с нарушением: **1328 (5.60%)**.



[См. исправления и комментарии по п. 12](#)

**13.** Обнаружены замечания к значениям '**floors\_total**' (всего этажей в доме). Значение floors\_total не должно быть пустым. Количество записей с нарушением: **86 (0.36%)**.

[См. исправления и комментарии по п. 13](#)

**14.** Обнаружены замечания к значениям '**kitchen\_area**' (площадь кухни в м<sup>2</sup>). Значение kitchen\_area не должно быть пустым. Количество записей с нарушением: **2278 (9.61%)**.

[См. исправления и комментарии по п. 14](#)

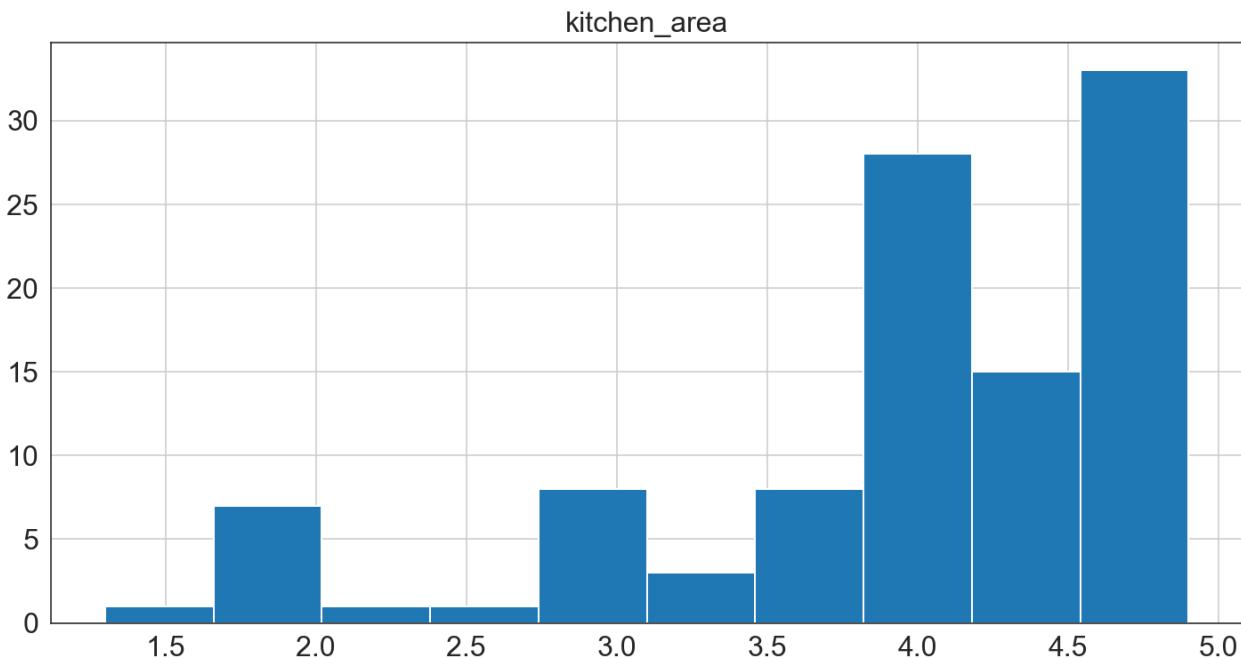
**15.** Обнаружены замечания к значениям '**kitchen\_area**' (площадь кухни в м<sup>2</sup>). Площадь кухни, как правило, не больше 70 м<sup>2</sup>. Количество записей с нарушением: **11 (0.05%)**.

	<b>last_price</b>	<b>rooms</b>	<b>living_area</b>	<b>kitchen_area</b>	<b>total_area</b>	<b>ceiling_height</b>
<b>492</b>	95000000.0	4	86.0	77.0	216.0	3.0
<b>2489</b>	12500000.0	3	153.9	100.7	255.0	NaN
<b>4394</b>	57000000.0	3	95.0	77.0	174.0	3.2
<b>5358</b>	65000000.0	15	409.0	100.0	590.0	3.5
<b>10867</b>	56844500.0	3	87.2	87.2	177.9	NaN
...	...	...	...	...	...	...
<b>16239</b>	82400000.0	3	58.2	93.2	181.1	3.9
<b>16647</b>	122000000.0	3	36.0	72.0	145.0	3.3
<b>16797</b>	65850000.0	2	40.0	93.0	146.0	NaN
<b>19540</b>	420000000.0	12	409.7	112.0	900.0	2.8
<b>20215</b>	85000000.0	3	72.0	107.0	249.7	NaN

11 rows × 6 columns

[См. исправления и комментарии по п. 15](#)

**16.** Обнаружены замечания к значениям '**kitchen\_area**' (площадь кухни в м<sup>2</sup>), '**studio**' (квартира-студия), '**open\_plan**' (свободная планировка). Площадь кухни в обычной квартире, как правило, не меньше 5 м<sup>2</sup> по санитарным нормам. Количество записей с нарушением: **105 (0.44%)**.



[См. исправления и комментарии по п. 16](#)

**17.** Обнаружены замечания к значениям '**last\_price**' (цена на момент снятия с публикации). Цена, как правило, больше 100 тыс. по здравому смыслу. Количество записей с нарушением: **1 (0.00%)**.

	<b>last_price</b>
<b>8793</b>	12190.0

[См. исправления и комментарии по п. 17](#)

**18.** Обнаружены замечания к значениям '**last\_price**' (цена на момент снятия с публикации). Цена, как правило, не больше 500 млн. по здравому смыслу. Количество записей с

нарушением: 1 (0.00%).

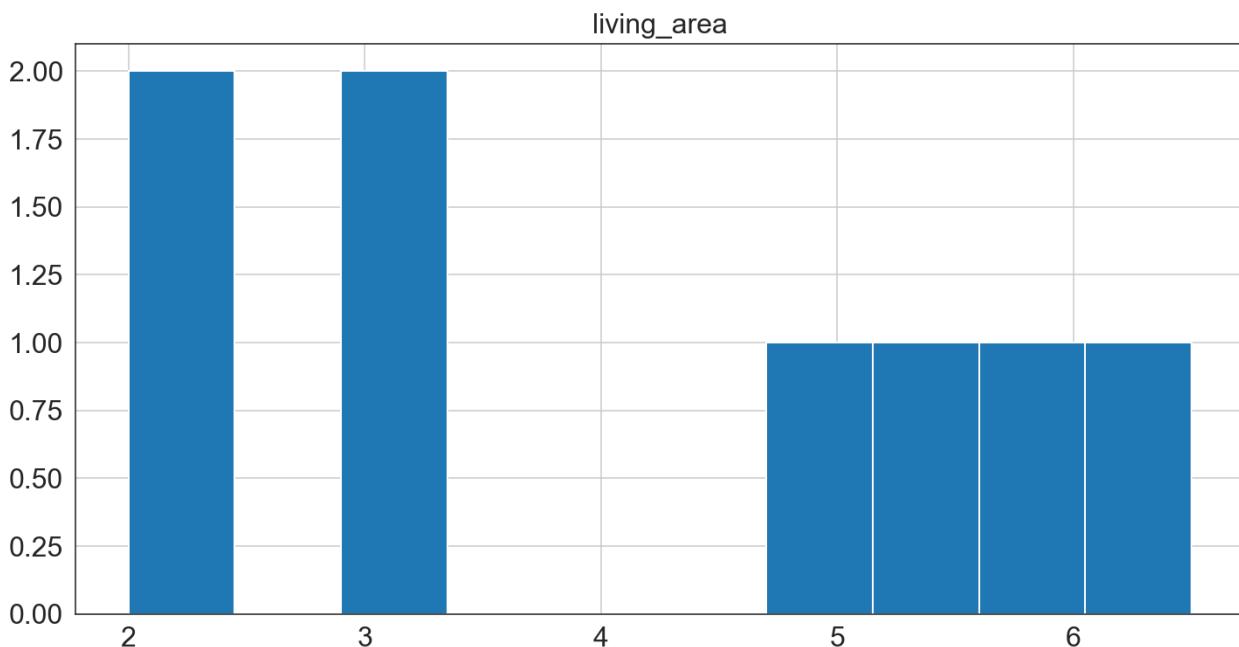
	last_price	rooms	living_area	kitchen_area	total_area	ceiling_height	days_exposition
12971	763000000.0	7	250.0	NaN	400.0	NaN	33.0

[См. исправления и комментарии по п. 18](#)

**19.** Обнаружены замечания к значениям '**living\_area**' (жилая площадь в м<sup>2</sup>). Значение living\_area не должно быть пустым. Количество записей с нарушением: **1903 (8.03%)**.

[См. исправления и комментарии по п. 19](#)

**20.** Обнаружены замечания к значениям '**living\_area**' (жилая площадь в м<sup>2</sup>). Жилая площадь, как правило, не меньше 8 м<sup>2</sup> по санитарным нормам. Количество записей с нарушением: **8 (0.03%)**.



[См. исправления и комментарии по п. 20](#)

**21.** Обнаружены замечания к значениям '**living\_area**' (жилая площадь в м<sup>2</sup>). Жилая площадь, как правило, не больше 300 м<sup>2</sup> по здравому смыслу. Количество записей с нарушением: **7 (0.03%)**.

	last_price	rooms	living_area	kitchen_area	total_area	ceiling_height
660	49950000.0	7	312.5	26.4	483.9	3.20
4237	50000000.0	7	332.0	22.0	517.0	NaN
5358	65000000.0	15	409.0	100.0	590.0	3.50
8018	84000000.0	5	301.5	45.5	507.0	4.45
12401	91500000.0	7	347.5	25.0	495.0	4.65
12859	140000000.0	7	322.3	19.5	631.2	3.90
19540	420000000.0	12	409.7	112.0	900.0	2.80

[См. исправления и комментарии по п. 21](#)

**22.** Обнаружены замечания к значениям '**living\_area**' (жилая площадь в м<sup>2</sup>), '**kitchen\_area**' (площадь кухни в м<sup>2</sup>). Значения living\_area и kitchen\_area не должны быть пустыми одновременно. Количество записей с нарушением: **1464 (6.18%)**.

[См. исправления и комментарии по п. 22](#)

**23.** Обнаружены замечания к значениям '**locality\_name**' (название населённого пункта).

Значение locality\_name не должно быть пустым. Количество записей с нарушением: **49 (0.21%)**.

[См. исправления и комментарии по п. 23](#)

**24.** Обнаружены замечания к значениям '**parks\_around3000**' (число парков в радиусе 3 км). Значение parks\_around3000 не должно быть пустым. Количество записей с нарушением: **5518 (23.28%)**.

[См. исправления и комментарии по п. 24](#)

**25.** Обнаружены замечания к значениям '**parks\_nearest**' (расстояние до ближайшего парка в метрах). Значение parks\_nearest не должно быть пустым. Количество записей с нарушением: **15620 (65.91%)**.

[См. исправления и комментарии по п. 25](#)

**26.** Обнаружены замечания к значениям '**parks\_nearest**' (расстояние до ближайшего парка в метрах). Значение parks\_nearest должно быть не более 3000 км. Количество записей с нарушением: **4 (0.02%)**.

parks_nearest	
1590	3064.0
10959	3190.0
19208	3013.0

[См. исправления и комментарии по п. 26](#)

**27.** Обнаружены замечания к значениям '**ponds\_around3000**' (число водоёмов в радиусе 3 км). Значение ponds\_around3000 не должно быть пустым. Количество записей с нарушением: **5518 (23.28%)**.

[См. исправления и комментарии по п. 27](#)

**28.** Обнаружены замечания к значениям '**ponds\_nearest**' (расстояние до ближайшего водоёма в метрах). Значение ponds\_nearest не должно быть пустым. Количество записей с нарушением: **14589 (61.56%)**.

[См. исправления и комментарии по п. 28](#)

**29.** Обнаружены замечания к значениям '**rooms**' (число комнат), '**studio**' (квартира-студия). Количество комнат в обычной квартире или со свободной планировкой (не студии), не должно быть нулевым. Количество записей с нарушением: **59 (0.25%)**.

[См. исправления и комментарии по п. 29](#)

**30.** Обнаружены замечания к значениям '**total\_area**' (площадь квартиры в м<sup>2</sup>). Общая площадь, как правило, не больше 500 м<sup>2</sup> по здравому смыслу. Количество записей с нарушением: **7 (0.03%)**.

	last_price	rooms	living_area	kitchen_area	total_area	ceiling_height
3117	140000000.0	7	Nan	60.0	631.0	Nan
4237	50000000.0	7	332.0	22.0	517.0	Nan
5358	65000000.0	15	409.0	100.0	590.0	3.50
8018	84000000.0	5	301.5	45.5	507.0	4.45
12859	140000000.0	7	322.3	19.5	631.2	3.90
15651	300000000.0	7	258.0	70.0	618.0	3.40
19540	420000000.0	12	409.7	112.0	900.0	2.80

[См. исправления и комментарии по п. 30](#)

**31.** Обнаружены замечания к значениям '**total\_area**' (площадь квартиры в м<sup>2</sup>), '**living\_area**' (жилая площадь в м<sup>2</sup>), '**kitchen\_area**' (площадь кухни в м<sup>2</sup>). Общая площадь не должна быть меньше суммы жилой и кухни. Количество записей с нарушением: **64 (0.27%)**.

	<b>total_area</b>	<b>living_area</b>	<b>kitchen_area</b>
<b>184</b>	30.20	26.10	6.20
<b>545</b>	23.80	20.00	5.00
<b>551</b>	31.59	30.55	9.28

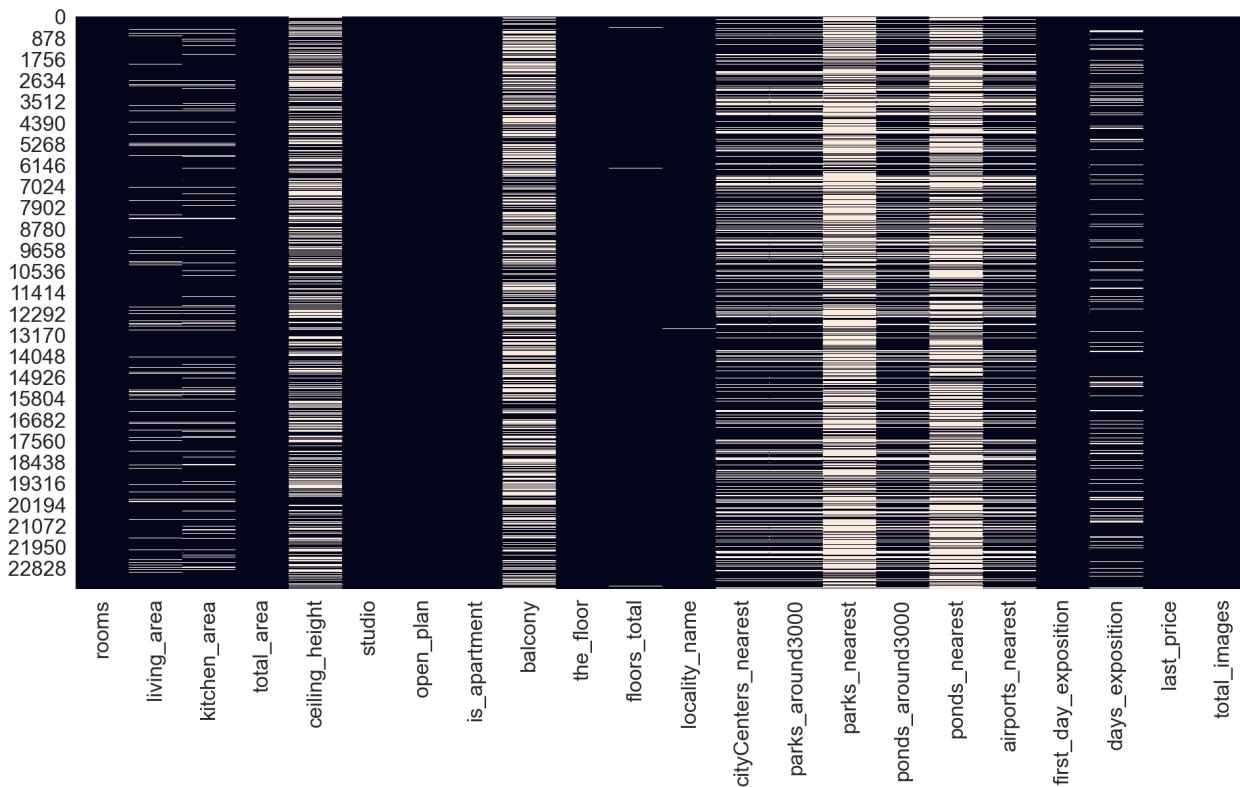
[См. исправления и комментарии по п. 31](#)

## Обработка пропусков, ошибок и аномалий в данных

### Оценка количества пропусков

Бросим взгляд на распределение пропусков по набору данных с помощью `sns.heatmap`:

```
In [41]: plt.figure(figsize = (16, 8))
sns.heatmap(ads[data_fields.keys()].isnull(), cbar=False);
```



Кажется, что пропуски больше сосредоточены в расстояниях, количестве балконов и высоте потолков. Интересно будет взглянуть на картину после завершения предобработки данных ([спойлер](#)).

Запомним размеры нашей таблицы до начала предобработки:

```
In [42]: shape_0 = ads.shape
shape_0
```

```
Out[42]: (23699, 22)
```

### Разбиение названия населенного пункта

Ранее была обоснована необходимость разбиения поля с названием населенного пункта на два отдельных: тип населенного пункта и название. Сделаем это в первую очередь, так как

пропуски в некоторых полях, возможно, будет необходимо заполнять по средним значениям показателей квартир того же населенного пункта или его типа.

```
In [43]: ads.locality_name.value_counts()
```

```
Out[43]: Санкт-Петербург      15721
посёлок Мурино        522
посёлок Шушары        440
Всеволожск            398
Пушкин                 369
...
деревня Луполово       1
поселок Коробицыно     1
деревня Раполово       1
посёлок Петро-Славянка 1
деревня Тойвороно       1
Name: locality_name, Length: 364, dtype: int64
```

Значения не требуют сложной лемматизации - поле уже было кем-то хорошо предобработано. Для городов задается только название, а для более мелких населенных пунктов указывается ещё и их тип. Имена собственные начинаются с большой буквы.

Заменим букву ё на е:

```
In [44]: ads.locality_name = ads.locality_name.str.replace('ё', 'е')
ads.locality_name = ads.locality_name.str.replace('Ё', 'Е')
```

Разберем поле `locality_name` регулярным выражением и получим новый набор данных с типами населенного пункта:

```
In [45]: splited = ads.locality_name.str.extractall(r'^((?:(?:[а-я]+) *)*) *([А-Я].*)$')
display(splited[0].unique())
display(splited[1].unique())
```

```
array(['поселок ', 'городской поселок ', 'деревня ',
       'поселок городского типа ', 'садовое товарищество ', 'село ',
       'поселок городского типа имени ', 'поселок станции ',
       'садоводческое некоммерческое товарищество ',
       'поселок при железнодорожной станции ', 'коттеджный поселок '],
      dtype=object)
array(['Санкт-Петербург', 'Шушары', 'Янино-1', 'Парголово', 'Мурино',
       'Ломоносов', 'Сертолово', 'Петергоф', 'Пушкин', 'Кудрово',
       'Коммунар', 'Колпино', 'Красный Бор', 'Гатчина', 'Федоровское',
       'Выборг', 'Кронштадт', 'Кировск', 'Новое Девяткино',
       'Металлострой', 'Лебяжье', 'Сиверский', 'Молодцово',
       'Кузьмоловский', 'Новая Ропша', 'Павловск', 'Пикколо',
       'Всеволожск', 'Волхов', 'Кингисепп', 'Приозерск', 'Сестрорецк',
       'Куттузи', 'Аннино', 'Ефимовский', 'Плодовое', 'Заклинье',
       'Торковичи', 'Первомайское', 'Красное Село', 'Понтонный',
       'Сясьстрой', 'Старая', 'Лесколово', 'Новый Свет', 'Сланцы',
       'Путилово', 'Ивангород', 'Шлиссельбург', 'Никольское',
       'Зеленогорск', 'Сосновый Бор', 'Оржицы', 'Кальтино', 'Романовка',
       'Бугры', 'Рошино', 'Кириши', 'Луга', 'Волосово', 'Отрадное',
       'Павлово', 'Оредеж', 'Копорье', 'Молодежное', 'Тихвин', 'Победа',
       'Нурма', 'Синявино', 'Тосно', 'Стрельна', 'Бокситогорск',
       'Александровская', 'Лопухинка', 'Пикалево', 'Терволово',
       'Советский', 'Подпорожье', 'Петровское', 'Токсово', 'Сельцо',
       'Вырица', 'Кипень', 'Келози', 'Вартемяги', 'Тельмана',
       'Севастьяново', 'Большая Ижора', 'Агалатово', 'Новогорелово',
       'Лесогорский', 'Лаголово', 'Цвелодубово', 'Рахья', 'Белогорка',
       'Заводской', 'Новоселье', 'Большие Колпаны', 'Горбунки', 'Батово',
       'Заневка', 'Иссад', 'Приморск', 'Мистолово', 'Новая Ладога',
       'Зимитицы', 'Барышево', 'Разметелево', 'Свердлова', 'Пеники',
       'Рябово', 'Пудомяги', 'Корнево', 'Низино', 'Бегуницы', 'Поляны',
       'Мга', 'Елизаветино', 'Кузнечное', 'Колтуши', 'Запорожское',
       'Гостилицы', 'Малое Карлино', 'Мичуринское', 'Морозова',
       'Песочный', 'Сосново', 'Аро', 'Ильичево', 'Тайцы', 'Малое Верево',
       'Извара', 'Вещево', 'Паша', 'Калитино', 'Ульяновка', 'Чудской Бор',
       'Дубровка', 'Мины', 'Войсковицы', 'Коркино', 'Ропша',
```

```
'Приладожский', 'Щеглово', 'Гаврилово', 'Лодейное Поле',
'Рабитицы', 'Никольский', 'Кузьмолово', 'Малые Колпаны',
'Петро-Славянка', 'Назия', 'Репино', 'Углово', 'Старая Малукса',
'Меньково', 'Старые Бегуницы', 'Саперный', 'Семирно', 'Глажево',
'Кобринское', 'Гарболово', 'Юкки', 'Приветнинское', 'Мануйлово',
'Пчева', 'Цвылево', 'Мельниково', 'Пудость', 'Усть-Луга',
'Светогорск', 'Любань', 'Селеzнево', 'Каменногорск', 'Кривко',
'Глебычево', 'Парицы', 'Жилпоселок', 'Войскорово', 'Стеклянный',
'Важины', 'Мыза-Ивановка', 'Русско-Высоцкое', 'Форносово',
'Старая Ладога', 'Житково', 'Виллози', 'Лампово', 'Шпаньково',
'Лаврики', 'Сумино', 'Возрождение', 'Старосиверская', 'Кикерино',
'Старое Хинколово', 'Пригородный', 'Торфяное', 'Будогощь',
'Суходолье', 'Красная Долина', 'Хапо-Ое', 'Дружная Горка',
'Лисий Нос', 'Яльгелево', 'Рождествено', 'Старополье', 'Левашово',
'Сяськелево', 'Камышовка', 'Лесная Поляна', 'Хязельки',
'Жилгородок', 'Ялгино', 'Новый Учхоз', 'Гончарово', 'Почап',
'Саперное', 'Платформа 69-й километр', 'Каложицы', 'Фалилеево',
'Пельгора', 'Торошковичи', 'Белоостров', 'Алексеевка',
'Серебрянский', 'Лукаши', 'Тарасово', 'Кингисеппский', 'Ушаки',
'Котлы', 'Сижно', 'Торосово', 'Форт Красная Горка', 'Новолисино',
'Громово', 'Глинка', 'Старая Пустошь', 'Коммунары', 'Починок',
'Вознесенье', 'Разбегаево', 'Гладкое', 'Тесово-4', 'Бор',
'Коробицыно', 'Большая Вруда', 'Курковицы', 'Кобралово',
'Суоранда', 'Кондратьево', 'Счастье', 'Реброво', 'Тойворово',
'Семизерье', 'Лесное', 'Совхозный', 'Ленинское', 'Судя',
'Нижние Осельки', 'Свирь', 'Перово', 'Высоцк', 'Шум', 'Котельский',
'Лужайка', 'Большая Пустомержа', 'Красносельское', 'Вахнова Кара',
'Пижма', 'Кивеннапа Север', 'Ромашки', 'Каськово', 'Куровицы',
'Плоское', 'Кирпичное', 'Ям-Тесово', 'Раздолье', 'Терпилицы',
'Шугозеро', 'Ваганово', 'Пушное', 'Садко', 'Усть-Ижора',
'Выскатка', 'Свирьстрой', 'Кисельня', 'Трубников Бор',
'Высокоключевой', 'Пансионат Зеленый Бор', 'Ненимаки',
'Снегиревка', 'Рапполово', 'Пустынка', 'Большой Сабск', 'Русско',
'Луполово', 'Большое Рейзино', 'Малая Романовка', 'Дружноселье',
'Пчевжа', 'Володарское', 'Нижняя', 'Тихковицы', 'Борисова Грива',
'Дзержинского'], dtype=object)
```

Обратим внимание на два слова ('имени' и 'станции'), которые имеют отношение к названию, а не типу. Усовершенствуем регулярное выражение для их исключения:

```
In [46]: splited = ads.locality_name.str.extractall(r'^((?:(?:?!имени )(?!станции )[а-я]+)*)')
display(splited[0].unique())

array([nan, 'поселок ', 'городской поселок ', 'деревня ',
       'поселок городского типа ', 'садовое товарищество ', 'село ',
       'садоводческое некоммерческое товарищество ',
       'поселок при железнодорожной ', 'коттеджный поселок '],
      dtype=object)
```

Среди типов населенных пунктов есть одинаковые по смыслу. Объединим их через замену подстрок:

```
In [47]: ads.locality_name = ads.locality_name.str.replace('городской ', '')
ads.locality_name = ads.locality_name.str.replace(' городского типа', '')
ads.locality_name = ads.locality_name.str.replace(' при железнодорожной', '')
ads.locality_name = ads.locality_name.str.replace('садоводческое некоммерческое', 'садо')
ads.locality_name = ads.locality_name.str.replace('село ', 'деревня ')
```

```
In [48]: splited = ads.locality_name.str.extractall(r'^((?:(?:?!имени )(?!станции )[а-я]+)*)')
display(splited[0].unique())

array([nan, 'поселок ', 'деревня ', 'садовое товарищество ',
       'коттеджный поселок '], dtype=object)
```

Занесем **новое поле** в анализируемую таблицу:

```
In [49]: ads[['locality_type']] = splited[0].unstack()
ads.locality_type = ads.locality_type.str.strip()
```

Населенные пункты без указанного типа трактуем в качестве города:

```
In [50]: ads['locality_type'] = ads['locality_type'].fillna('город')
```

Проверим полученные данные:

```
In [51]: ads[['locality_type','locality_name']].value_counts().sort_values(ascending=False)
```

locality_type	locality_name	Length: 320, dtype: int64
город	Санкт-Петербург	15721
поселок	поселок Мурино	556
	поселок Шушары	440
город	Всеволожск	398
	Пушкин	369
...		
поселок	поселок Высокоключевой	1
	поселок Гладкое	1
	поселок Гончарово	1
	поселок Дзержинского	1
коттеджный поселок	коттеджный поселок Лесное	1

Преобразуем тип населенного пункта в **категориальную переменную** средствами `pandas.CategoricalDtype`, отсортировав возможные значения по импирической значимости:

```
In [52]: locality_type_dtype = pd.CategoricalDtype(['садовое товарищество', 'деревня',
                                                'коттеджный поселок', 'поселок', 'город'], or

In [53]: ads['locality_type'] = ads.locality_type.astype(locality_type_dtype)

In [54]: display(ads.locality_type.dtype)

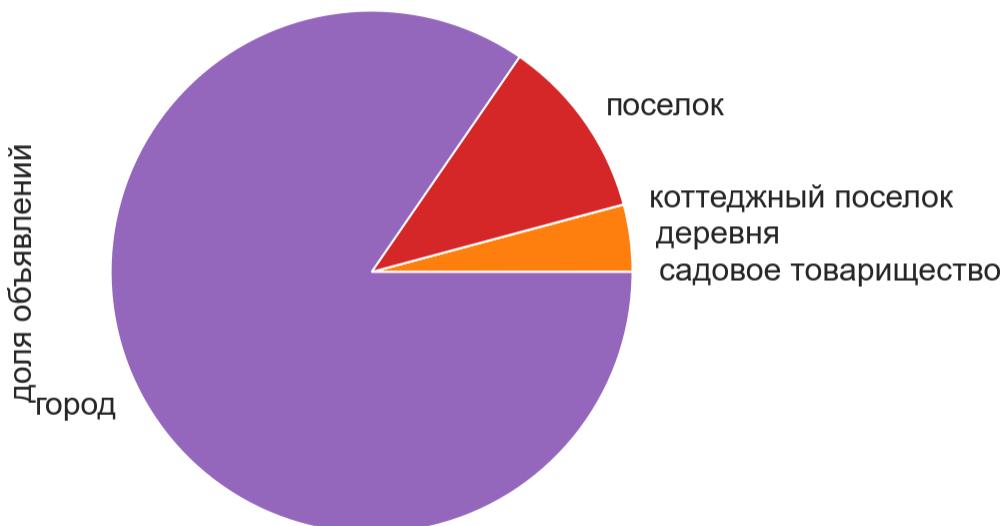
CategoricalDtype(categories=['садовое товарищество', 'деревня', 'коттеджный поселок',
                            'поселок', 'город'],
                  ordered=True)
```

Добавим описание нового поля в [список](#) полей набора данных для последующего использования:

Построим круговую диаграмму типов населенных пунктов. убедимся в её адекватности:

```
In [56]: ads.groupby('locality_type')['locality_name'].count().plot.pie(title=data_fields['locality_type'], ylabel='доля объявлений')
```

## тип населённого пункта



```
In [57]: ads['locality_type'].value_counts().sort_values(ascending=False)
```

```
Out[57]: город          20057  
поселок        2657  
деревня         977  
садовое товарищество    5  
коттеджный поселок    3  
Name: locality_type, dtype: int64
```

Среди объявлений преобладают предложения о продаже квартир в городе. Меньше всего на рынке представлена недвижимость в садовых товариществах и коттеджных поселках.

Подавляющее количество объявлений из Санкт-Петербурга:

```
In [58]: ads['locality_name'].value_counts().sort_values(ascending=False).head(10)
```

```
Out[58]: Санкт-Петербург      15721  
поселок Мурино        556  
поселок Шушары        440  
Всеволожск            398  
Пушкин                  369  
Колпино                  338  
поселок Парголово      327  
Гатчина                  307  
деревня Кудрово        299  
Выборг                  237  
Name: locality_name, dtype: int64
```

## Обработка нереальных и маловероятных значений

В первую очередь разберемся с нереальными и маловероятными значениями. Они могут негативно повлиять на результат анализа. Пройдемся последовательно по всем пунктам полученного ранее протокола проверки.

Автоматизированная проверка обнаружила **49 объявлений (0.21%) без указания населенного пункта**. Определим для них максимальное удаление от центра города:

```
In [59]: ads[ads.locality_name.isnull()]['cityCenters_nearest'].max()
```

```
Out[59]: 41294.0
```

Протяжённость Санкт-Петербурга в административных границах: с севера на юг в пределах КАД — 32 км (за пределами КАД — 52 км), с северо-запада на юго-восток за пределами КАД,

— около 90 км. Объявления с указанным расстоянием до центра не более 25 км разместим в северной столице, остальные удалим (либо пользователи ошиблись, либо - поставщики данных).

```
In [60]: ads.locality_name.fillna(ads.locality_name.where(ads.cityCenters_nearest > 25000, other
```

Посчитаем теперь количество "анонимок" за пределами Санкт-Петербурга:

```
In [61]: ads.locality_name.isnull().sum()
```

```
Out[61]: 3
```

Удалим их:

```
In [62]: ads.dropna(subset=['locality_name'], inplace=True)
```

Убедимся, что пропусков в названии населенного пункта больше не осталось:

```
In [63]: ads.locality_name.isnull().sum()
```

```
Out[63]: 0
```

Автоматизированная проверка выявила по одной квартире [дешевле 100 тысяч и дороже 500 миллионов](#). Удалим их:

```
In [64]: ads.drop(ads[(ads.last_price < 100000) | (ads.last_price > 5e8)].index, inplace=True)
```

Автоматизированная проверка [обнаружила 105 \(0.44%\)](#) обычных квартир **с площадью кухни меньше 5 м<sup>2</sup>**, что является нарушением санитарных норм и является маловероятным. Удалим подобные объявления:

```
In [65]: ads.drop(ads[~(ads.studio | ads.open_plan) & (ads.kitchen_area < 5)].index, inplace=True)
```

Автоматизированная проверка [выявила 11 \(0.05%\)](#) объектов недвижимости, **с аномально большими кухнями** площадью больше 70 м<sup>2</sup>. Это либо элитные объекты, структура цены, которых отличается от изучаемых нами квартир, либо ошибки. Принимая во внимание небольшое количество строк, принимаем решение удалить их:

```
In [66]: ads.drop(ads[ads.kitchen_area > 70].index, inplace=True)
```

Автоматизированная проверка [обнаружила](#) замечания к **жилой площади 8 (0.03%) квартир**, у которых она меньше 8 м<sup>2</sup>, что нарушает санитарные нормы. Удалим такие записи:

```
In [67]: ads.drop(ads[ads.living_area < 8].index, inplace=True)
```

Автоматизированная проверка выявила по семь квартир с аномально большой **жилой площадью** (больше 300 м<sup>2</sup>) и **общей площадью** (больше 500 м<sup>2</sup>). Удалим такие объявления:

```
In [68]: ads.drop(ads[ads.living_area > 300].index, inplace=True)
```

```
In [69]: ads.drop(ads[ads.total_area > 500].index, inplace=True)
```

Автоматизированная проверка [выявила 64 \(0.27%\)](#) квартиры, у которых **общая площадь меньше суммы жилой и кухни**. Удалим эти явные ошибки:

```
In [70]: ads.drop(ads[ads.total_area < (ads.living_area + ads.kitchen_area)].index, inplace=True)
```

## Устранение замечаний к данным о расстояниях

Автоматизированная проверка [обратила внимание](#) на **5542 (23.38%)** объявлений **с пустым**

**расстоянием до ближайшего аэропорта в метрах.** Это достаточно много для простого удаления строк. Попробуем восстановить пропуски по другим полям.

А пока вспомним, что автоматизированная проверка также [выявила](#) для одной квартиры **нулевую близость к аэропорту**, хотя расстояние до аэропорта должно быть не менее 5000 м по санитарным нормам. Можно предположить, что хозяин офиса в Пулково выставил его на продажу, или петербуржец забыл заполнить это поле. Причин может быть много. Одно показание можно было бы и удалить, но в учебном проекте, как в жизни, нам важен каждый клиент.

Поэтому заменим расстояние до аэропорта для пустых значений и меньших нормы в 5 километров на другое значение, вычисленное на основе данных соседей по городу.

```
In [71]: ads.loc[ads.airports_nearest < 5000, 'airports_nearest'] = np.nan
```

Для начала посмотрим распределение расстояний до аэропорта в различных населенных пунктах:

```
In [72]: # соберем статистику по всем населенным пунктам
stat_names = ['min', 'max', 'median', 'count']
D = ads[['locality_name', 'airports_nearest']].groupby(by='locality_name', as_index=True)
D.columns = stat_names
# выведем населенные пункты, в объявлениях которых есть информация о расстоянии до аэропорта
# отсортируем по ближайшему минимальному расстоянию
D = D[D.loc[:, 'count'] > 0].sort_values(by='min')
display(D)
```

		min	max	median	count
	locality_name				
	<b>Санкт-Петербург</b>	6450.0	54784.0	26757.0	15559
	<b>поселок Шушары</b>	9294.0	26293.0	17439.5	434
	<b>Пушкин</b>	12157.0	21055.0	15766.0	366
	<b>поселок Александровская</b>	12781.0	13012.0	12896.5	2
	<b>Павловск</b>	19380.0	24291.0	20529.5	38
	...	...	...	...	...
	<b>поселок Репино</b>	61451.0	64127.0	61797.5	4
	<b>поселок Щеглово</b>	61908.0	61908.0	61908.0	1
	<b>Кронштадт</b>	64931.0	69785.0	67847.0	92
	<b>Зеленогорск</b>	70016.0	81607.0	72280.0	23
	<b>поселок Молодежное</b>	83758.0	84869.0	84665.0	5

27 rows × 4 columns

Из данных таблицы видно, что расстояние до аэропорта в массиве данных, действительно, указывается для Пулково. Достаточно посмотреть на карту, например, для [Кронштадта](#), [поселка Шушары](#) или [Павловска](#). Это значит, что для пропущенных значений расстояния до аэропорта мы можем взять какие-нибудь усредненные значения того же населенного пункта (если они имеются). Тогда в дальнейшем, например, сможем точнее построить модель линейной регрессии или дерева решений для предсказания стоимости квартиры, в том числе с учетом транспортной доступности.

Сохраним на всякий случай первоначальное значение поля с расстоянием до аэропорта:

```
In [73]: ads['airports_nearest_o'] = ads['airports_nearest']
```

Присвоим пропускам медианные средние расстояния до аэропорта от населенного пункта при их наличии (точного адреса объявления ведь у нас нет). Заодно потренируем навыки замены пропусков значениями, полученными с помощью группировки. Уверен, они нам еще понадобятся:

```
In [74]: ads['airports_nearest'] = ads.groupby('locality_name')['airports_nearest'] \  
       .transform(lambda x: x.fillna(x.mean()))
```

Убедимся, что количество пропусков в расстояниях до аэропорта уменьшилось:

```
In [75]: ads.airports_nearest.isnull().sum()
```

```
Out[75]: 4816
```

Заполнить их можно было бы по справочникам. Действительно: название населенного пункта Ленинградской области известно, значит известны его географические координаты, значит можно подсчитать расстояние. Возникает вопрос, где взять справочники. Есть несколько вариантов решения:

- запросить у поставщиков данных;
- написать скрапер, который по названию населенного пункта "вытянет" с какого-нибудь сайта необходимые данные. Оставим эту задачу на следующие проекты. В данном случае пока просто "оштрафуем" объявления с этими пропусками условным максимальным расстоянием, которое сделает их на 10% менее привлекательным по сравнению с самыми далекими из известных:

```
In [76]: distance_penalty = round(ads.airports_nearest.max() * 1.1)  
distance_penalty
```

```
Out[76]: 93356
```

```
In [77]: ads.airports_nearest.fillna(distance_penalty, inplace = True)
```

Убедимся, что пропусков в расстояниях до аэропорта больше нет:

```
In [78]: ads.airports_nearest.isnull().sum()
```

```
Out[78]: 0
```

Автоматизированная проверка [предупредила](#) о возможных аномалиях в сведениях о **расстояниях до центра**. Как видим на [круговой диаграмме](#), все **186 значений достоверны**, так как указанные на ней населенные пункты находятся на расстоянии от 50 до 70 километров от Санкт-Петербурга. Яндекс.Карты это [подтверждают](#). Оставим данные без изменения.

Автоматизированная проверка **выявила 5519 (23.29%) объявлений с пустым расстоянием до центра города**. Это достаточно много для простого их удаления. Попробуем восстановить пропуски по другим полям аналогично расстояниям до аэропорта.

Сохраним на всякий случай первоначальное значение поля с расстоянием до центра:

```
In [79]: ads['cityCenters_nearest_o'] = ads['cityCenters_nearest']
```

Присвоим пропускам медианные средние расстояния до центра от населенного пункта при их наличии:

```
In [80]: ads['cityCenters_nearest'] = ads.groupby('locality_name')['cityCenters_nearest'] \  
       .transform(lambda x: x.fillna(x.mean()))
```

```
.transform(lambda x: x.fillna(x.mean()))
```

Убедимся, что количество пропусков в расстояниях до центра уменьшилось:

```
In [81]: ads.cityCenters_nearest.isnull().sum()
```

```
Out[81]: 4816
```

Заполнить их можно было бы также по справочникам. В их отсутствии пока просто "оштрафуем" объявления с этими пропусками условным максимальным расстоянием, которое сделает их на 10% менее привлекательным по сравнению с самыми далекими из известных:

```
In [82]: distance_penalty = round(ads.cityCenters_nearest.max() * 1.1)  
distance_penalty
```

```
Out[82]: 72565
```

```
In [83]: ads.cityCenters_nearest.fillna(distance_penalty, inplace=True)
```

Убедимся, что пропусков в расстояниях до центра больше нет:

```
In [84]: ads.cityCenters_nearest.isnull().sum()
```

```
Out[84]: 0
```

## Фильтрация неадекватных потолков

Автоматизированная проверка выявила **16 объявлений**, декларирующих **неадекватно низкие потолки**, которые не соответствуют санитарным нормам в 2.3 метра. Причинами ошибок могут быть: ошибки измерений и ошибки при ручном вводе показаний.

Удалим записи с потолками ниже 2 метров. Их количество ничтожно мало по сравнению с размером всего массива данных:

```
In [85]: ads.drop(ads[ads.ceiling_height < 2.0].index, inplace=True)
```

Остальным нарушениям присвоим санитарную норму (в учебных целях):

```
In [86]: ads.loc[ads.ceiling_height < 2.3, 'ceiling_height'] = 2.3
```

Убедимся в правильности выполнения операций. Низких потолков теперь нет:

```
In [87]: ads[ads.ceiling_height < 2.3].size
```

```
Out[87]: 0
```

Автоматизированная проверка выявила **неадекватно высокие потолки** (больше 10 метров) в обычных квартирах (не студиях). Количество записей с нарушением: **25 (0.11%)**.

Предположим, что основной причиной является ошибка ввода данных пользователем - пропущен разделитель целой и дробной части. Исправим показатели для ошибок от 25 до 50 делением на 10, остальные аномалии удалим:

```
In [88]: ads.loc[(~ads.studio) & (ads.ceiling_height >= 25) & (ads.ceiling_height <= 50), 'ceiling_height'] = 2.3
```

```
In [89]: ads.drop(ads[(~ads.studio) & (ads.ceiling_height > 10)].index, inplace=True)
```

Автоматизированная проверка выявила **9195 (38.80%)** объявлений с пустой высотой потолков в метрах.

Есть острое желание заменить пропуски каким-нибудь новым методом Pandas, например, предыдущим непустым значением с заглядыванием назад на не более, чем 20 записей:

```
ads.ceiling_height.fillna(2.3, inplace=True)
```

Лучше мы не будем поолагаться на случай и заменим пропуски в высоте потолка на минимально возможные по санитарной норме - 2.3 метра. Скорее всего, владельцы квартир стесняются указывать именно это значение:

```
In [90]: ads.ceiling_height.fillna(2.3, inplace=True)
```

Оценим параметры распределения высоты потолков после заполнения пропусков:

```
In [91]: ads.ceiling_height.describe()
```

```
Out[91]: count    23490.000000
mean      2.561903
std       0.308688
min       2.300000
25%      2.300000
50%      2.500000
75%      2.700000
max       8.300000
Name: ceiling_height, dtype: float64
```

Убеждаемся, что оно соответствует здравому смыслу.

## Восстановление этажности

Автоматизированная проверка выявила **86 (0.36%)** объявлений с **пустым количеством этажей в доме**.

В реальной задаче можно было бы восстановить этажность здания по его адресу и электронным массивам [Роскадастра](#). В качестве второго варианта целесообразно рассмотреть восстановление пропущенных данных о высоте зданий по другим заполненным объявлениям. Можно так даже проверять качество подаваемых объявлений.

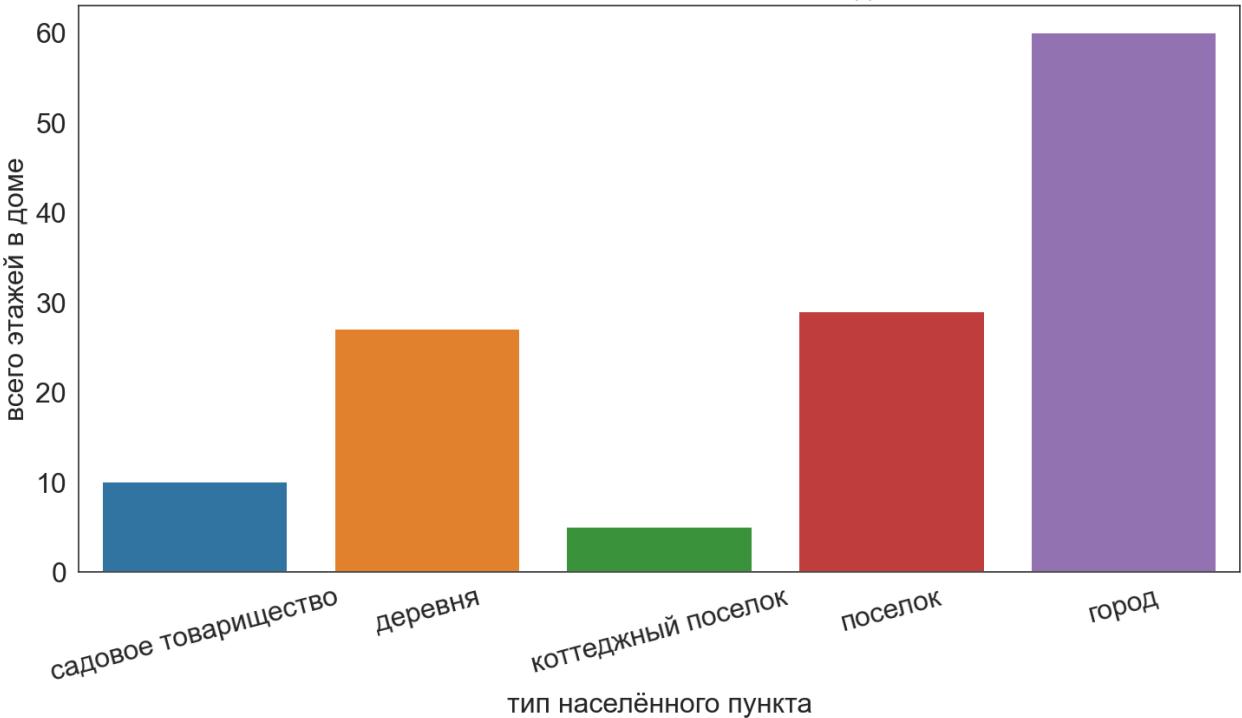
В нашей учебной задаче восстановим этажность дома по этажу квартиры (пропусков в этом поле нет). Некоторые владельцы недвижимости могут обидеться за это, ведь они теперь, по нашим данным, будут жить на последнем этаже. Но они сами виноваты, каждый несет ответственность за полноту предоставляемых сведений. Да и небольшая доля таких объявлений (0.36% от общего количества) не испортит статистику.

```
In [92]: ads.floors_total.fillna(ads.the_floor, inplace=True)
```

Изучим максимальную высоту домов в населенных пунктах различных типов:

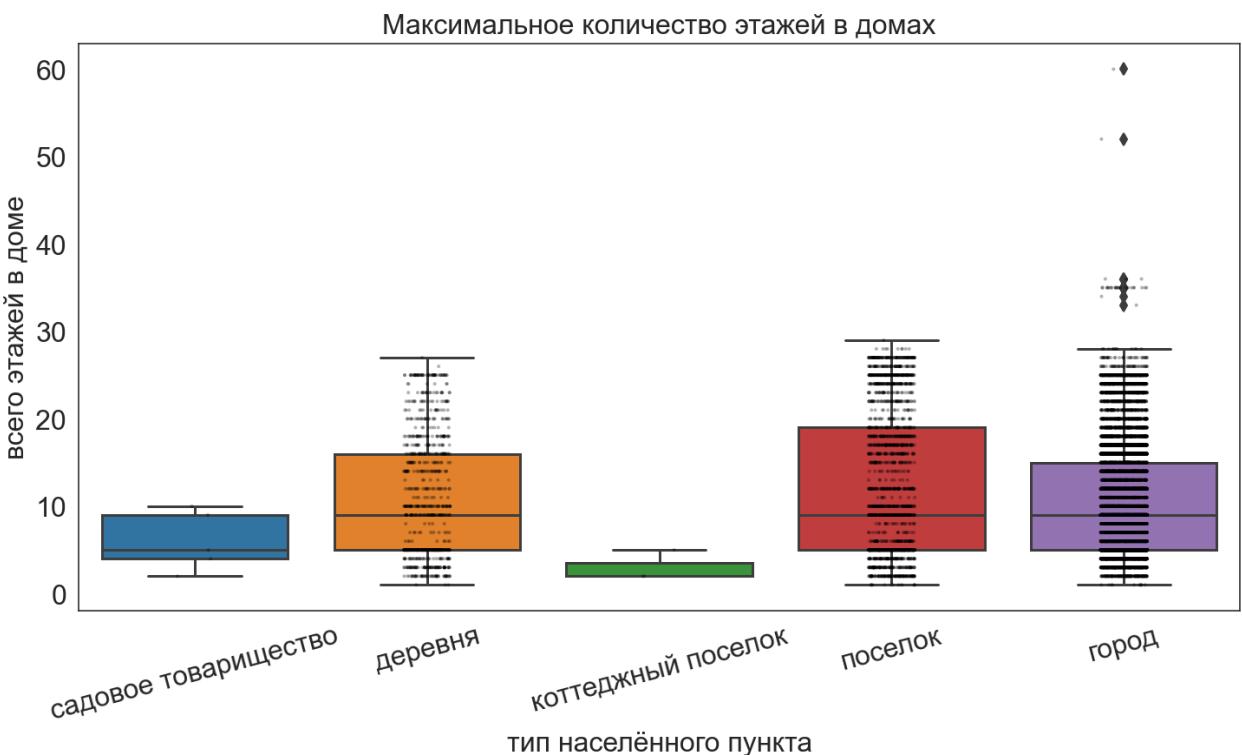
```
In [93]: graph = sns.barplot(data=ads.groupby('locality_type', as_index = False)[['floors_total']]
                           .x='locality_type', y='floors_total')
graph.set(title='Максимальное количество этажей в домах',
          xlabel=data_fields['locality_type'].desc,
          ylabel=data_fields['floors_total'].desc)
graph.set_xticklabels(graph.get_xticklabels(), rotation=15);
```

## Максимальное количество этажей в домах



Вот это поворот! В нашем наборе объявлений есть деревни и поселки с домами под тридцать этажей.

```
In [94]: #small_Localities = ads[ads.locality_type.isin(['деревня', 'поселок', 'садовое товарищество'])]
graph = sns.boxplot(x='locality_type', y='floors_total', data=ads)
sns.stripplot(x='locality_type', y='floors_total', color='black', size=2, alpha=0.3, data=ads)
graph.set(title='Максимальное количество этажей в домах',
          xlabel=data_fields['locality_type'].desc,
          ylabel=data_fields['floors_total'].desc)
graph.set_xticklabels(graph.get_xticklabels(), rotation=15);
```



При этом небоскребов в городе меньше, чем многоэтажек в маленьких населенных пунктах. Изучим, где имеются дома высотой больше 10 этажей:

```
In [95]: ads[(ads.locality_type != 'город') & (ads.floors_total > 10)] \
    .groupby('locality_name', as_index=False)[['floors_total']] \
    .agg(['min', 'max'])
```

Out[95]:

locality_name	floors_total	
	min	max
деревня Колтуши	16.0	16.0
деревня Кудрово	11.0	27.0
деревня Куттузи	12.0	15.0
деревня Лаврики	17.0	17.0
деревня Малые Колпаны	12.0	12.0
...	...	...
поселок Стрельна	13.0	15.0
поселок Тельмана	14.0	16.0
поселок Шушары	11.0	27.0
поселок Янино-1	12.0	19.0
поселок имени Свердлова	15.0	27.0

26 rows × 2 columns

Обнаружили **26** населенных пунктов. Неужели мы ошиблись, когда выделяли тип населенного пункта? Обратимся к Яндекс.Картам для проверки на примере [деревни Колтуши](#). Изучим поисковые выдачи по картинкам и обнаружим 16-этажку, которую мы незаслуженно подозревали!



Будем считать, что с этажностью разобрались, а для себя сделаем вывод: в малых населенных пунктах активно строятся комфортабельные высотные дома.

## Устранение неопределенности суммы площадей

Автоматизированная проверка выявила **2278 (9.61%)** объявлений с пропущенными значениями площади кухни. Кроме того, были обнаружены **1464 (6.18%)** объявлений, в которых **указана только общая площадь, а жилая площадь и размер кухни пустые**. Это делает невозможным выполнение анализа для довольно большой доли квартир. Попробуем восстановить пропуски, заполнив площади кухни средним значением для групп квартир, примерно одинаковых по общей площади. Как правило, кухни более стандартизованы, чем жилая зона, и восстановление пропусков по площади именно места приготовления пищи не должно испортить распределение и внести ошибки в статистику изучаемого набора данных.

В квартирах-студиях нет отдельных кухонь. В аппартаментах со свободной планировкой кухней могли пожертвовать ради спорт-зала или терриума. Примем во внимание только квартиры (не студии и не в свободной планировке):

```
In [96]: flats = ads[~(ads.studio | ads.open_plan)]
print('Всего объявлений об обычных квартирах:', flats.shape[0])
```

Всего объявлений об обычных квартирах: 23276

Сгруппируем квартиры по [перцентилям](#) размера общей площади с шагом 0.01, воспользовавшись возможностями функции `pandas.Series.rank`. Подсчитаем для получившихся групп средние значения площадей кухни:

```
In [97]: flats_kitchen_group = flats.groupby(by=flats.total_area.rank(pct=True).round(2))
flats_kitchen = flats_kitchen_group \
    .agg(ads_count=('total_area', 'count'), # количество объявлений
         ads_with_kitchen=('kitchen_area', 'count'), # с кухнями
         total_area_min=('total_area', 'min'), # мин. общая площадь
         total_area_max=('total_area', 'max'), # макс. общая площадь
         kitchen_min=('kitchen_area', 'min'), # мин. площадь кухни
         kitchen_median=('kitchen_area', 'median'), # медиана площади
         kitchen_mean=('kitchen_area', 'mean'), # средняя площадь
         kitchen_max=('kitchen_area', 'max')) # макс. площадь кухни
display(flats_kitchen)
```

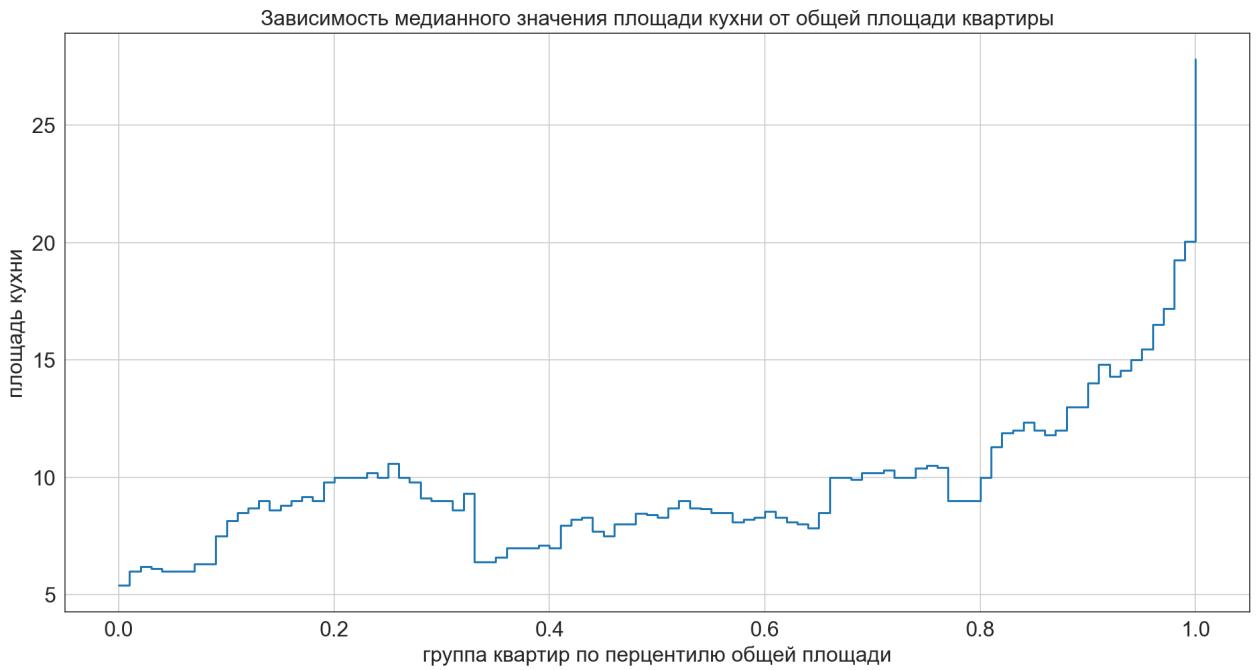
total_area	ads_count	ads_with_kitchen	total_area_min	total_area_max	kitchen_min	kitchen_median	kitchen_max
<b>0.00</b>	111	23	12.00	24.90	5.0	5.40	
<b>0.01</b>	272	166	25.00	29.00	5.0	6.00	
<b>0.02</b>	296	266	29.10	30.00	5.0	6.20	
<b>0.03</b>	141	135	30.10	30.60	5.0	6.10	
<b>0.04</b>	91	90	30.61	30.90	5.1	6.00	
...	...	...	...	...	...	...	...
<b>0.96</b>	224	210	120.10	130.21	7.8	16.50	
<b>0.97</b>	233	217	130.30	147.90	8.0	17.20	
<b>0.98</b>	232	206	147.98	170.70	7.2	19.25	
<b>0.99</b>	234	210	170.90	240.00	5.0	20.05	
<b>1.00</b>	116	98	241.00	500.00	10.0	27.80	

101 rows × 8 columns



Видим, что в каждой группе достаточно количество объявлений. Построим график медианы:

```
In [98]: plt.figure(figsize=(16, 8))
plt.step(flats_kitchen.index, flats_kitchen.kitchen_median, '--', where='post')
plt.title('Зависимость медианного значения площади кухни от общей площади квартиры')
plt.xlabel('группа квартир по перцентилю общей площади')
plt.ylabel('площадь кухни')
plt.grid()
plt.show()
```



Наша группировка уловила закономерность: размер кухонь в квартирах не зависит линейно от общей площади, а растет волнами. Это объясняется тем, что представленная на рынке недвижимость, как правило, соответствует конечному числу проектов. Это не противоречит здравому смыслу, значит мы можем заполнить пропуски групповыми медианными значениями.

Заполним пропуски в площадях кухонь для объявлений, в которых указана только общая площадь:

```
In [99]: # пометим нужные квартиры
flats_to_fillna = (~(flats.studio | flats.open_plan)) & \
                  (flats.living_area.isna()) & (flats.kitchen_area.isna())
# вычислим для них новые значения
new_values = flats_kitchen_group[['kitchen_area']] \
              .transform(lambda x: x.fillna(x.median()))
```

```
In [100...]: # занесем новые значения в таблицу в соответствии с индексами
ads.loc[new_values.index, 'kitchen_area'] = new_values
```

Обнулим площадь кухонь для квартир-студий и проектов со свободной планировкой, если в этих объявлениях не указаны размеры зоны для приготовления пищи:

```
In [101...]: flats_to_fillna = ads.studio | ads.open_plan
```

```
In [102...]: ads.loc[flats_to_fillna, 'kitchen_area'] = ads.loc[flats_to_fillna, 'kitchen_area'].fi
```

Попытаемся восстановить площадь кухни по известным общей и жилой площадям (за вычетом санузлов и коридоров с некоторым коэффициентом). Если восстановленная площадь кухни не будет удовлетворять санитарной норме в  $5 \text{ м}^2$  или окажется неадекватно большой, то такие объявления удалим, так как в них либо ошибка, либо аномалия:

```
In [103...]: ads.kitchen_area = ads.kitchen_area.fillna(ads.total_area - ads.living_area / 0.6 - 6)

In [104...]: ads.drop(ads[~(ads.studio | ads.open_plan) & (ads.kitchen_area < 5)].index, inplace=True)

In [105...]: ads.drop(ads[~(ads.studio | ads.open_plan) & (ads.kitchen_area > 70)].index, inplace=True)
```

## Восстановление пропусков в жилой площади

Автоматизированная проверка выявила **1903 (8.03%) объявлений с пустой жилой площадью**. Мы можем попробовать восстановить эти значения, зная общую площадь квартиры и размер кухни, а также учитывая санузлы и коридоры с некоторым коэффициентом:

```
In [106...]: ads['living_area'] = ads['living_area'].fillna((ads['total_area'] - (ads['kitchen_area'] + ads['bathroom_area'] + ads['corridor_area'])) * 0.6)
```

При выбранном подходе могут появиться значения жилой площади, неудовлетворяющие социальной норме 8 м<sup>2</sup>. Такие записи соответствуют объявлениям, которые, скорее всего, изначально содержат ошибки и требуют удаления. Вот и хорошо! Используем их для пользы дела и проверим, как справится наша функция автоматизированной проверки при повторном запуске. Это имеет важное значение, так как специалист в области больших данных должен проверять себя на каждом шаге изменения данных, чтобы не плодить ошибки. Разработанная функция должна облегчить этот итерационный процесс. ([спойлер](#))

## Корректировка сведений о балконах

Автоматизированная проверка выявила **11519 (48.61%) объявлений без указания количества балконов**. Логично предположить, что их и нет. В таком случае обнулим пропуски:

```
In [107...]: ads.balcony.fillna(0, inplace=True)
```

Автоматизированная проверка обнаружила замечания к **256 (1.08%)** значениям **количества балконов на первом этаже**. В нашей задаче будем считать что, их не должно быть больше одного:

```
In [108...]: ads.loc[ads[((ads.the_floor == 1) & (ads.balcony != 1))].index, 'balcony'] = 1
```

Автоматизированная проверка выявила **124 случая (0.52%)**, когда **количество балконов в обычной квартире (если не свободная планировка и не студия) больше количества комнат + 2 (кухня, коридор)**. Удалим записи об этих объявлениях, как о недостоверных:

```
In [109...]: ads.drop(ads[(~ads.studio) & (~ads.open_plan) & (ads.balcony > ads.rooms + 2)].index, inplace=True)
```

## Устранение других пропусков

Автоматизированная проверка обратила внимание на **3181 пропуск (13.42%)** в **продолжительности размещения объявлений в днях**. Данное поле не может быть пустым, так как у всех строк заполнен столбец **цены на момент снятия с публикации**. То есть в нашем наборе данных нет активных объявлений. Поэтому заполним пропуски срока публикации минимальным возможным значением в 1 день.

```
In [110...]: d1 = ads.days_exposition.describe()
```

```
In [111...]: ads.days_exposition.fillna(1, inplace=True)
```

```
In [112...]: d2 = ads.days_exposition.describe()
```

Сравним распределения продолжительности размещения объявления до и после заполнения пропусков. Убедимся, что оно изменилось в пределах здравого смысла:

```
In [113...]: pd.DataFrame({'до': d1, 'после': d2, 'разница': d2 - d1})
```

	до	после	разница
count	19773.000000	22847.000000	3074.000000
mean	181.037526	156.813980	-24.223546
std	220.100319	213.776454	-6.323865
min	1.000000	1.000000	0.000000
25%	45.000000	22.000000	-23.000000
50%	96.000000	74.000000	-22.000000
75%	231.000000	198.500000	-32.500000
max	1580.000000	1580.000000	0.000000

## Ложные тревоги

Опасения по поводу **дат публикации** оказались напрасными. Оставляем этот столбец без изменений.

## Устранение аномалий с близкими объектами

Автоматизированная проверка выявила:

- **5518 (23.28%)** объявлений с пустым числом парков в радиусе 3000 м;
- **15620 (65.91%)** пустых расстояний до ближайшего парка в метрах;
- **4 (0.02%)** расстояния до ближайшего парка более 3000 м;
- **5518 (23.28%)** объявлений с пустым числом водоёмов в радиусе 3000 м;
- **14589 (61.56%)** пустых расстояний до ближайшего водоёма в метрах.

Причиной такого явления может быть то, что пользователи иногда вводили либо количество близлежащих объектов, либо расстояние до них.

Сократим расстояния больше 3000 метров до допустимого минимума (скорее всего, в данные вкралась погрешность округления):

```
In [114...]: ads.parks_nearest.where(ads.parks_nearest.isna() | (ads.parks_nearest <= 3000), other=3000).dropna()
ads.ponds_nearest.where(ads.ponds_nearest.isna() | (ads.ponds_nearest <= 3000), other=3000).dropna()
```

Если задано расстояние до ближайшего объекта, установим для пустого числа объектов единицу. Если есть расстояние, то существует хотя бы один объект:

```
In [115...]: ads.parks_around3000 = ads[ads.parks_nearest.notna()].parks_around3000.fillna(1)
ads.ponds_around3000 = ads[ads.ponds_nearest.notna()].ponds_around3000.fillna(1)
```

Если задано количество близайших объектов, установим для пустого расстояния максимально допустимое - 3000 метров:

```
In [116...]: ads.parks_nearest = ads[ads.parks_around3000.notna()].parks_nearest.fillna(3000)
ads.ponds_nearest = ads[ads.ponds_around3000.notna()].ponds_nearest.fillna(3000)
```

Теперь занулим пустые значения в количествах близких объектов:

```
In [117...]: ads['parks_around3000'].fillna(0, inplace=True)
ads['ponds_around3000'].fillna(0, inplace=True)
```

А пустые расстояния "оштрафуем" тройным максимально допустимым значением (9000 метров) для удобства работы алгоритма машинного обучения, который мы когда-нибудь применим к этому набору данных:

```
In [118...]: ads['parks_nearest'].fillna(9000, inplace=True)
ads['ponds_nearest'].fillna(9000, inplace=True)
```

Убедимся в успешном завершении операций по данным о распределениях:

```
In [119...]: objects_stat = ads[['parks_around3000', 'parks_nearest', 'ponds_around3000', 'ponds_nearest']]
objects_stat.describe()
```

```
Out[119...]:
```

	parks_around3000	parks_nearest	ponds_around3000	ponds_nearest
<b>count</b>	22847.000000	22847.000000	22847.000000	22847.000000
<b>mean</b>	0.470084	6087.464131	0.591194	5737.41349
<b>std</b>	0.748201	4042.467973	0.884114	4130.28785
<b>min</b>	0.000000	1.000000	0.000000	13.00000
<b>25%</b>	0.000000	598.000000	0.000000	636.00000
<b>50%</b>	0.000000	9000.000000	0.000000	9000.00000
<b>75%</b>	1.000000	9000.000000	1.000000	9000.00000
<b>max</b>	3.000000	9000.000000	3.000000	9000.00000

## Обработка замечаний к количеству комнат

Автоматизированная проверка выявила **59 (0.25%) квартир (обычных и со свободной планировкой), в которых нет комнат**. В соответствующем столбце записан ноль. Удалим эти объявления как недостоверные:

```
In [120...]: ads.drop(ads[~ads.studio] & (ads.rooms == 0)].index, inplace=True)
```

## Повторная автоматизированная проверка типов полей

Вызовем повторно функцию проверки и автоматического преобразования типов столбцов нашей таблицы:

```
In [121...]: auto_check_all_fields()
```

## Протокол автоматизированной проверки типов полей набора данных "Информация об объявлениях":

**1.** Тип **float64** поля "**balcony**" (число балконов) не соответствует заявленному в описании **int64**.

Тип поля "**balcony**" (число балконов) изменен на **int64**.

**2.** Тип **float64** поля "**floors\_total**" (всего этажей в доме) не соответствует заявленному в описании **int64**.

Тип поля "**floors\_total**" (всего этажей в доме) изменен на **int64**.

**3.** Тип **float64** поля "**parks\_around3000**" (число парков в радиусе 3 км) не соответствует заявленному в описании **int64**.

Тип поля "**parks\_around3000**" (число парков в радиусе 3 км) изменен на **int64**.

**4.** Тип **float64** поля "**ponds\_around3000**" (число водоёмов в радиусе 3 км) не соответствует заявленному в описании **int64**.

Тип поля "**ponds\_around3000**" (число водоёмов в радиусе 3 км) изменен на **int64**.

5. Тип **float64** поля "**days\_exposition**" (сколько дней было размещено объявление) не соответствует заявленному в описании **int64**.

Тип поля "**days\_exposition**" (сколько дней было размещено объявление) изменен на **int64**.

После удаления пропусков все типы успешно приведены в соответствие с описанием набора данных.

## Повторная автоматизированная проверка значений

Повторно запустим функцию проверки значений полей.

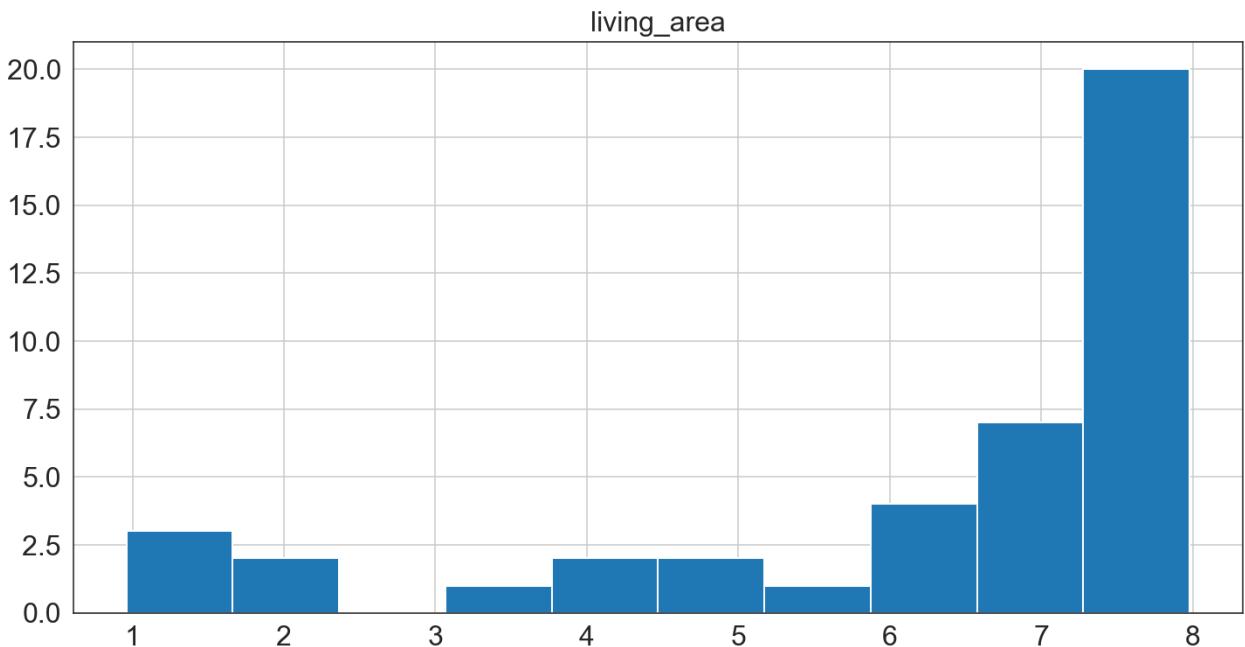
По идеи, у неё возникнут вопросы только к расстояниям до парков и озер, которые мы [заполнили значением 9000 метров](#) для тех объявлений, у которых количество объектов рядом нулевое. Если решим в дальнейшем применить метод линейной регрессии для прогнозирования, то такой подход будет "штрафовать" квартиры без информации об объектах инфраструктуры поблизости.

Ну, и не забываем о том, что мы [сознательно оставили](#) некоторые аномалии в восстановленных значениях жилой площади, чтобы протестировать, как с ними справится функция автоматизированной проверки.

```
In [122...]: auto_check_all_values(check_fix=True, prefix='test 2 value error')
```

### Протокол автоматизированной проверки значений набора данных "Информация об объявлениях":

1. Обнаружены замечания к значениям '**living\_area**' (жилая площадь в м<sup>2</sup>). Жилая площадь, как правило, не меньше 8 м<sup>2</sup> по санитарным нормам. Количество записей с нарушением: **42 (0.18%)**.



[См. исправления и комментарии по п. 1](#)

2. Обнаружены замечания к значениям '**parks\_nearest**' (расстояние до ближайшего парка в метрах). Значение parks\_nearest должно быть не более 3000 км. Количество записей с нарушением: **14977 (65.72%)**.

parks_nearest	
1	9000.0
5	9000.0

### **parks\_nearest**

<b>6</b>	9000.0
----------	--------

[См. исправления и комментарии по п. 2](#)

**3.** Обнаружены замечания к значениям '**ponds\_nearest**' (расстояние до ближайшего водоёма в метрах). Значение ponds\_nearest должно быть не более 3000 км. Количество записей с нарушением: **14021 (61.53%)**.

### **ponds\_nearest**

<b>1</b>	9000.0
<b>5</b>	9000.0
<b>6</b>	9000.0

[См. исправления и комментарии по п. 3](#)

## Удаление объявлений с неадекватной жилой площадью

Повторная автоматизированная проверка справилась с [обнаружением](#) нереальных значений **жилой площади меньше 8 м<sup>2</sup>**, нарушающих санитарные нормы. Обнаружено **8 (0.03%) объявлений**, которые можно смело удалить:

```
In [123...]: ads.drop(ads[ads.living_area < 8].index, inplace=True)
```

## Снятие с контроля расстояний до объектов

Выявленные [повторной автоматизированной проверкой](#) замечания к расстояниям до [парков](#) и [прудов](#) ошибками не являются, так как мы сами ранее [задали штраф](#) для объектов недвижимости, у которых рядом нет объектов. Снимем с контроля поля '**parks\_nearest**' и '**ponds\_nearest**' перед заключительной проверкой:

```
In [124...]: for checking in distances_checkings: data_checkings.remove(checking)
```

## Контрольная автоматизированная проверка

После завершения предобработки данных осуществим контрольную автоматизированную проверку, чтобы окончательно убедиться в отсутствии ошибок в данных и полезности разработки [вспомогательного класса](#):

```
In [125...]: auto_check_all_fields()
auto_check_all_values(check_fix=True, prefix='test 3 value error')
```

## Протокол автоматизированной проверки типов полей набора данных "Информация об объявлениях":

В наборе данных "**Информация об объявлениях**" ошибок в типах не обнаружено.

## Протокол автоматизированной проверки значений набора данных "Информация об объявлениях":

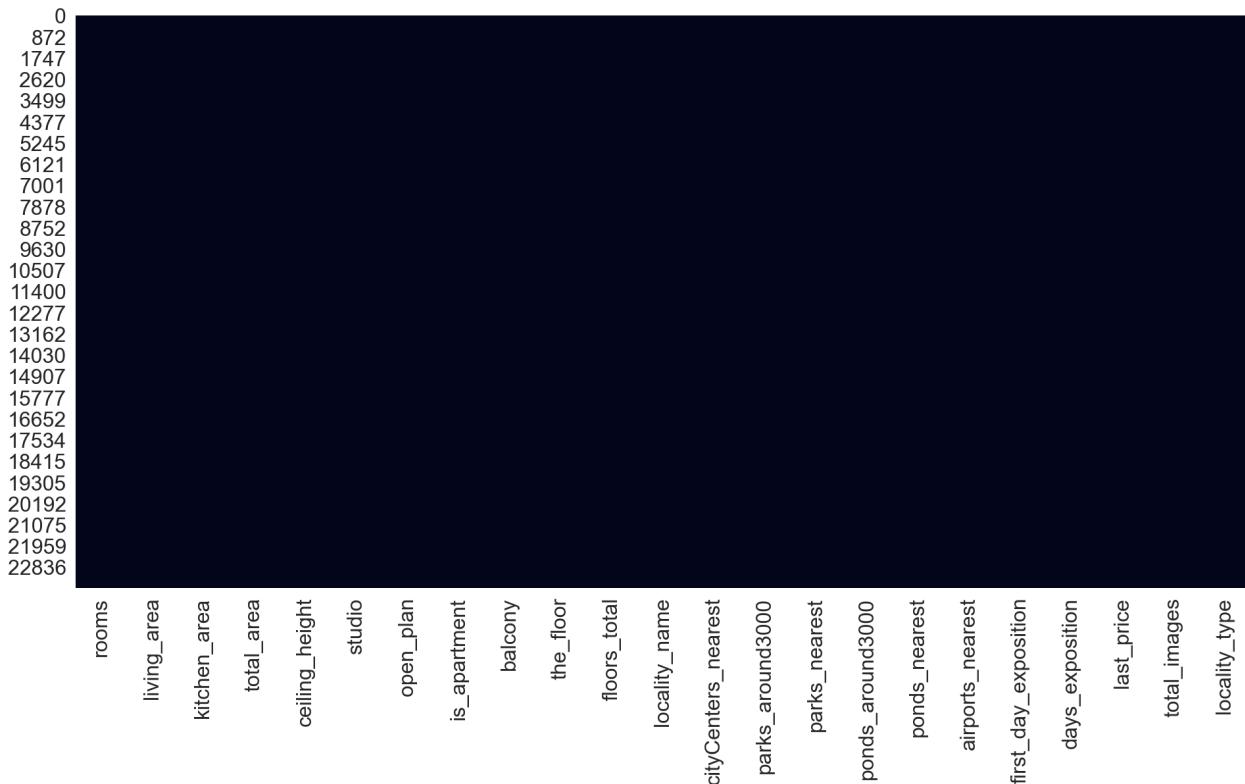
В наборе данных "**Информация об объявлениях**" ошибок в значениях не обнаружено.

## Оценка количества пропусков после предобработки

Ранее мы уже строили тепловую карту пропусков. Посмотрим на её изменения после

предобработки данных:

```
In [126...]: plt.figure(figsize = (16, 8))
sns.heatmap(ads[data_fields.keys()].isnull(), cbar=False);
```



Раскрыт источник вдохновения [Малевича](#) - Pandas и предобработка данных! А если серьезно, то были устраниены все пропуски.

Сравним размер таблицы с первоначальным:

```
In [127...]: shape_1 = ads.shape
shape_1
```

```
Out[127...]: (22746, 25)
```

На этапе предобработки пришлось удалить некоторое число записей:

```
In [128...]: delta = shape_0[0] - shape_1[0]
delta
```

```
Out[128...]: 953
```

Процент удаленных записей составляет:

```
In [129...]: delta / shape_0[0] * 100
```

```
Out[129...]: 4.021266720114773
```

Нам удалось провести предобработку, включая восстановление данных по известным значениям и удаление аномалий, без существенного ущерба для размера выборки.

## Оценка количества полных дубликатов

```
In [130...]: ads.duplicated(data_fields.keys()).sum()
```

```
Out[130...]: 0
```

Полных дубликатов в нашем наборе нет.

## Ручной контроль преобразования типов данных

Убедимся, что все типы полей были преобразованы в соответствии со смыслом хранящихся в них значений:

In [131...]: ads.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 22746 entries, 0 to 23698
Data columns (total 25 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   rooms            22746 non-null   int64  
 1   living_area      22746 non-null   float64 
 2   kitchen_area     22746 non-null   float64 
 3   total_area       22746 non-null   float64 
 4   ceiling_height   22746 non-null   float64 
 5   studio           22746 non-null   bool    
 6   open_plan         22746 non-null   bool    
 7   is_apartment     22746 non-null   bool    
 8   balcony          22746 non-null   int64  
 9   the_floor         22746 non-null   int64  
 10  floors_total     22746 non-null   int64  
 11  locality_name    22746 non-null   object  
 12  cityCenters_nearest  22746 non-null   float64 
 13  parks_around3000  22746 non-null   int64  
 14  parks_nearest    22746 non-null   float64 
 15  ponds_around3000  22746 non-null   int64  
 16  ponds_nearest    22746 non-null   float64 
 17  airports_nearest  22746 non-null   float64 
 18  first_day_exposition  22746 non-null   datetime64[ns] 
 19  days_exposition   22746 non-null   int64  
 20  last_price        22746 non-null   float64 
 21  total_images      22746 non-null   int64  
 22  locality_type     22746 non-null   category 
 23  airports_nearest_o 17488 non-null   float64 
 24  cityCenters_nearest_o 17510 non-null   float64 
dtypes: bool(3), category(1), datetime64[ns](1), float64(11), int64(8), object(1)
memory usage: 3.9+ MB
```

Обратим внимание на изменения по сравнению с [первоначальным состоянием набора данных](#).

## Вывод по шагу 2

- Определены и изучены [пропущенные значения](#):
  - наибольшее их число было сосредоточено в расстояниях, количестве балконов и высоте потолков;
  - отмечено отсутствие данных в значимых для оценки объявлений характеристиках:
    - [этаж](#) квартиры и этажность здания;
    - жилая площадь и площадь кухни ([иногда одновременно](#));
    - [срок публикации](#) объявления;
    - [населенный пункт](#);
    - другие поля.
- Для всех пропущенных значений оценены возможные причины, а также предположена замена или способ восстановления по известным показателям. Например:
  - пустые количества балконов [обнулены](#);
  - для 46 объявлений без указания населенного пункта [подтверждено](#) нахождение квартиры в Санкт-Петербурге по указанному расстоянию до центра города, 3 квартиры в Ленинградской области пришлось удалить из массива;
  - вместо пустых значений в полях расстояний до парков и прудов записывался "штраф" в виде большого значения, который гипотетически может учитываться

алгоритмом машинного обучения в модели оценки спроса на жилье.

- Все столбцы **автоматизированно проверены** на адекватность значений по сформулированным условиям, исходя из смысла хранимых значений. Например:
  - обнаружено 16 объявлений, декларирующих **неадекватно низкие потолки**, которые не соответствуют санитарным нормам в 2.3 метра. Потолкам выше 2 метров присвоено минимально допустимое значение, остальные были удалены;
  - объявления с неадекватными значениями площадей **удалены**.
- Все столбцы приведены к **нужным типам** в соответствии со смыслом хранящихся в них значений:
  - тип полей "balcony" (число балконов), "floors\_total" (всего этажей в доме), "parks\_around3000" (число парков в радиусе 3 км), "ponds\_around3000" (число водоёмов в радиусе 3 км), "days\_exposition" (сколько дней было размещено объявление) **изменен** с вещественного на целочисленный;
  - полу "is\_apartment" (апартаменты) **присвоен** логический тип.
- На основе данных в названии населенного пункта **создана** новая категориальная переменная 'locality\_type' (тип населенного пункта - город, деревня, поселок городского типа и т.д.). В столбце объединены смысловые дубликаты (например, "садовое товарищество" и "садоводческое некоммерческое товарищество").
- Таким образом, предобработка данных выполнена. Массив готов к дальнейшему анализу.

## Шаг 3. Вычисление новых показателей

### Задание шага 3

Посчитайте и добавьте в таблицу:

- цену квадратного метра;
- день недели, месяц и год публикации объявления;
- этаж квартиры (варианты - первый, последний, другой);
- соотношение жилой и общей площади, а также отношение площади кухни к общей.

### Вычисление цены квадратного метра

Цену квадратного метра квартиры определим через отношение её стоимости к общей площади:

```
In [132]: ads['meter_price'] = ads['last_price'] / ads['total_area']
```

### Определение дня недели, месяца и года объявления

#### Учет даты совершения сделки

Перед продолжением работы обойдем одну ловушку для аналитика - рассинхронизацию времени регистрации данных, над которыми он совершает свои аналитические действия.

Целью нашего исследования является анализ рынка недвижимости, в том числе динамики цен. Но в нашем наборе есть стоимость квартиры, но нет даты, на которую эта стоимость была актуальной. Да, да! Есть дата публикации объявления и цена на момент его снятия после совершения сделки. На лицо неопределенность, которую необходимо устранить. На помощь приходит другое поле - иногда достаточно долгий срок публикации объявления в днях.

Прибавим его значение к дате публикации и получим правильное соответствие цены квартиры и времени.

Создадим новый столбец `price_date`, в который занесем дату совершения сделки - дату

актуальности цены квартиры. Воспользуемся `pandas.Timedelta`:

```
In [133...]: # опишем функцию, которая прибавляет дни к дате
def add_days_to_date(date, days):
    return date + pd.Timedelta(value=days-1, unit='D')
```

```
In [134...]: ads[['price_date']] = ads[['first_day_exposition', 'days_exposition']] \
    .apply(lambda v: add_days_to_date(*v), axis=1)
```

Проверим правильность вычислений на примере нескольких значений:

```
In [135...]: ads[['first_day_exposition', 'days_exposition', 'price_date']].head()
```

```
Out[135...]:
```

	first_day_exposition	days_exposition	price_date
0	2019-03-07	1	2019-03-07
1	2018-12-04	81	2019-02-22
2	2015-08-20	558	2017-02-27
3	2015-07-24	424	2016-09-19
4	2018-06-19	121	2018-10-17

В дальнейшем анализ объявлений и вычисление новых показателей будем проводить в контексте `price_date` - даты актуальности цены квартиры. Для краткости будем называть его датой совершения сделки.

## День недели объявления

День недели и ряд других новых показателей целесообразно создать в виде категориальной переменной с помощью `pandas.CategoricalDtype`. Это позволит задавать им привычные для человека значения без расходования памяти, как в случае со строковым типом:

Опишем упорядоченную последовательность дней недели:

```
In [136...]: weekday_dtype = pd.CategoricalDtype(['понедельник', 'вторник', 'среда', 'четверг', 'пятница',
                                             'суббота', 'воскресенье'], ordered=True)
```

В принципе, мы можем пользоваться функцией `pandas.Series.dt.day_name` для получения названия дней недели, но тогда станем зависеть от ее внутренней реализации. В учебном проекте отработаем работу с помощью пользовательских категориальных переменных.

Извлечем день недели из поля даты совершения сделки с помощью `pandas.Series.dt.weekday` и сформируем новое поле, переведя числа в значения определенных выше категорий через `pandas.Categorical.from_codes`:

```
In [137...]: ads['weekday'] = pd.Categorical.from_codes(codes=ads['price_date'].dt.weekday,
                                                   dtype=weekday_dtype)
```

Проверим правильность вычислений на нескольких примерах по календарю:

```
In [138...]: ads[['price_date', 'weekday']].head()
```

```
Out[138...]:
```

	price_date	weekday
0	2019-03-07	четверг
1	2019-02-22	пятница
2	2017-02-27	понедельник
3	2016-09-19	понедельник

	price_date	weekday
4	2018-10-17	среда

## Месяц объявления

Опишем упорядоченную последовательность месяцев года:

```
In [139...]: month_dtype = pd.CategoricalDtype(['январь', 'февраль', 'март', 'апрель',
                                             'май', 'июнь', 'июль', 'август',
                                             'сентябрь', 'октябрь', 'ноябрь', 'декабрь'],
                                             ordered=True)
```

Извлечем месяц из даты совершения сделки с помощью `pandas.Series.dt.month` и сформируем новое поле, переведя числа в значения определенных выше категорий через `pandas.Categorical.from_codes`. Из месяца будем вычитать единицу, так как в категориальной переменной они нумеруются с нуля:

```
In [140...]: ads['month'] = pd.Categorical.from_codes(codes=ads['price_date'].dt.month - 1,
                                                dtype=month_dtype)
```

Проверим правильность вычислений на нескольких примерах по календарю:

```
In [141...]: ads[['price_date', 'month']].head()
```

	price_date	month
0	2019-03-07	март
1	2019-02-22	февраль
2	2017-02-27	февраль
3	2016-09-19	сентябрь
4	2018-10-17	октябрь

## Год объявления

Год является частью поля с датой совершения сделки и может использоваться для анализа активности на рынке недвижимости. Чем больше сделок в год, тем выше активность:

```
In [142...]: ads['year'] = ads['price_date'].dt.year
```

Проверим правильность вычислений на нескольких примерах:

```
In [143...]: ads[['price_date', 'year']].head()
```

	price_date	year
0	2019-03-07	2019
1	2019-02-22	2019
2	2017-02-27	2017
3	2016-09-19	2016
4	2018-10-17	2018

## Год и месяц объявления

Для анализа динамики изменения цен или других показателей недвижимости во времени целесообразно разбивать общую выборку на временные интервалы и считать для них

метрики. Для этого заведем новый столбец `year_month`, приведя дату сделки к первому числу следующего месяца:

```
In [144...]: ads['year_month'] = ads.price_date + pd.offsets.MonthBegin(0)
```

Проверим правильность вычислений на нескольких примерах:

```
In [145...]: ads[['price_date', 'year_month']].head()
```

```
Out[145...]:   price_date  year_month
```

0	2019-03-07	2019-04-01
1	2019-02-22	2019-03-01
2	2017-02-27	2017-03-01
3	2016-09-19	2016-10-01
4	2018-10-17	2018-11-01

Если сгруппировать данные по новому полю, то получим все объявления за прошедший месяц. В дальнейшем используем такой подход для анализа изменения цен во времени.

## Категорирование этажа квартиры

Опишем возможные значения категории этажа:

- одноэтажный (например, в загородном доме - фактически первый и последний одновременно);
- первый;
- промежуточный (между первым и последним);
- последний.

В нашем массиве одноэтажных домов мало, но мы будем их учитывать для возможности масштабирования алгоритма для выхода на рынок элитного жилья на отечественных и зарубежных курортах 😊:

```
In [146...]: ads[(ads.the_floor == 1) & (ads.floors_total == 1)].the_floor.count()
```

```
Out[146...]: 23
```

Опишем упорядоченную последовательность категорий этажа. По приоритету начнем с промежуточного:

```
In [147...]: floor_dtype = pd.CategoricalDtype(['промежуточный', 'первый', 'одноэтажный',
                                             'последний', 'ошибка классификации'], ordered=True)
```

```
In [148...]: floor_dtype.categories
```

```
Out[148...]: Index(['промежуточный', 'первый', 'одноэтажный', 'последний',
                     'ошибка классификации'],
                     dtype='object')
```

Опишем функцию определения категории этажа:

```
In [149...]: def get_floor_category(the_floor, floors_total):
    """Функция вычисления категории этажа по его номеру и высоте здания"""
    if (floors_total >= 1) and (the_floor >= 1) and (the_floor <= floors_total):
        if the_floor != floors_total:
            if the_floor == 1:
                return 'первый'
```

```

else:
    return 'промежуточный'
elif floors_total > 1:
    return 'последний'
else:
    return 'одноэтажный'
else:
    return 'ошибка классификации'

```

Присвоим каждому объявлению новую категорию `floor_category` на основе значений `the_floor` и `floors_total`:

```
In [150...]: ads['floor_category'] = pd.Categorical(
    ads[['the_floor', 'floors_total']].apply(lambda v: get_floor_category(*v), axis=1),
    dtype=floor_dtype)
```

Проверим правильность классификации по нескольким значениям каждой категории:

```
In [151...]: for the_floor in floor_dtype.categories:
    display(HTML(f'{  
<b>Примеры категории "{the_floor}"</b>'}'))
    display(ads[ads.floor_category == the_floor][['floor_category', 'the_floor', 'floors_total']])
```

**Примеры категории "промежуточный":**

	floor_category	the_floor	floors_total
<b>0</b>	промежуточный	8	16
<b>2</b>	промежуточный	4	5
<b>3</b>	промежуточный	9	14
<b>4</b>	промежуточный	13	14
<b>5</b>	промежуточный	5	12

**Примеры категории "первый":**

	floor_category	the_floor	floors_total
<b>1</b>	первый	1	11
<b>14</b>	первый	1	6
<b>16</b>	первый	1	5
<b>24</b>	первый	1	3
<b>28</b>	первый	1	6

**Примеры категории "одноэтажный":**

	floor_category	the_floor	floors_total
<b>3076</b>	одноэтажный	1	1
<b>4379</b>	одноэтажный	1	1
<b>5300</b>	одноэтажный	1	1
<b>5698</b>	одноэтажный	1	1
<b>5787</b>	одноэтажный	1	1

**Примеры категории "последний":**

<b>floor_category</b>	<b>the_floor</b>	<b>floors_total</b>
12	последний	5
13	последний	5
23	последний	9
36	последний	9
43	последний	4

**Примеры категории "ошибка классификации":**

<b>floor_category</b>	<b>the_floor</b>	<b>floors_total</b>
-----------------------	------------------	---------------------

Ошибок классификации по этажу не обнаружено. Бросим взгляд на количество объявлений каждой категории и убедимся в адекватности распределения по здравому смыслу (одноэтажных должно быть меньше всего, первых и последних - примерно одинаково, промежуточных - в разы больше остальных):

```
In [152...]: ads.groupby(['floor_category']).agg(category_count = ('floor_category', 'count'))
```

```
Out[152...]: category_count
```

<b>floor_category</b>	<b>category_count</b>
промежуточный	16675
первый	2776
одноэтажный	23
последний	3272
ошибка классификации	0

## Определение соотношения жилой и общей площади

Соотношение жилой и общей площади вычислим делением значений соответствующих столбцов:

```
In [153...]: ads['living_div_total'] = ads['living_area'] / ads['total_area']
```

## Вычисление отношения площади кухни к общей

Отношение площади кухни к общей вычислим делением значений соответствующих столбцов:

```
In [154...]: ads['kitchen_div_total'] = ads['kitchen_area'] / ads['total_area']
```

## Расстояние до центра в километрах

Для анализа зависимости цены жилья от расстояния до центра города удобнее пользоваться не метрами, а километрами. Определим новое поле `cityCenters_km`:

```
In [155...]: ads['cityCenters_km'] = ads.cityCenters_nearest.div(1000).round()
```

## Описание новых полей

В набор данных были добавлены новые поля:

Имя нового поля	Описание нового поля
<b>meter_price</b>	цена квадратного метра
<b>weekday</b>	день недели публикации объявления
<b>month</b>	месяц публикации объявления
<b>year</b>	год публикации объявления
<b>year_month</b>	первый рабочий день следующего месяца - отчетная дата, на статистику которой будет влиять объявление
<b>floor_category</b>	категория этажа квартиры (промежуточный, первый, одноэтажный, последний, ошибка классификации)
<b>living_div_total</b>	соотношение жилой и общей площади квартиры
<b>kitchen_div_total</b>	отношения площади кухни к общей
<b>locality_type</b>	тип населенного пункта (садовое товарищество, деревня, коттеджный поселок, поселок,город)
<b>cityCenters_km</b>	расстояние до центра в километрах

Добавим описание новых полей в [список](#) набора данных для последующего использования при построении графиков и проверки значений:

```
In [156...]: data_fields['meter_price'] = DataField('цена квадратного метра', 'цена м2', np.float64)
data_fields['price_date'] = DataField('дата совершения сделки', 'дата сделки', np.datetime64)
data_fields['weekday'] = DataField('день недели совершения сделки', 'день\пнедель')
data_fields['month'] = DataField('месяц совершения сделки', 'месяц\псделки', np.int64)
data_fields['year'] = DataField('год совершения сделки', 'год\псделки', np.int64)
data_fields['year_month'] = DataField('отчетная дата', 'отчетная\пдата', np.datetime64)
data_fields['floor_category'] = DataField('категория этажа квартиры', 'категория\пподиума')
data_fields['living_div_total'] = DataField('соотношение жилой и общей площади квартиры', 'жилая\общая')
data_fields['kitchen_div_total'] = DataField('отношения площади кухни к общей', 'кухня\общая')
data_fields['cityCenters_km'] = DataField('расстояние до центра в километрах', 'расстояние\центра')
```

```
In [157...]: ads.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 22746 entries, 0 to 23698
Data columns (total 35 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   rooms            22746 non-null   int64  
 1   living_area      22746 non-null   float64 
 2   kitchen_area     22746 non-null   float64 
 3   total_area       22746 non-null   float64 
 4   ceiling_height   22746 non-null   float64 
 5   studio           22746 non-null   bool    
 6   open_plan        22746 non-null   bool    
 7   is_apartment     22746 non-null   bool    
 8   balcony          22746 non-null   int64  
 9   the_floor         22746 non-null   int64  
 10  floors_total     22746 non-null   int64  
 11  locality_name    22746 non-null   object  
 12  cityCenters_nearest  22746 non-null   float64 
 13  parks_around3000  22746 non-null   int64  
 14  parks_nearest    22746 non-null   float64 
 15  ponds_around3000  22746 non-null   int64  
 16  ponds_nearest    22746 non-null   float64 
 17  airports_nearest 22746 non-null   float64 
 18  first_day_exposition 22746 non-null   datetime64[ns]
 19  days_exposition  22746 non-null   int64  
 20  last_price        22746 non-null   float64 
 21  total_images      22746 non-null   int64  
 22  locality_type     22746 non-null   category
```

```

23 airports_nearest_o      17488 non-null  float64
24 cityCenters_nearest_o   17510 non-null  float64
25 meter_price             22746 non-null  float64
26 price_date              22746 non-null  datetime64[ns]
27 weekday                 22746 non-null  category
28 month                   22746 non-null  category
29 year                     22746 non-null  int64
30 year_month               22746 non-null  datetime64[ns]
31 floor_category           22746 non-null  category
32 living_div_total          22746 non-null  float64
33 kitchen_div_total         22746 non-null  float64
34 cityCenters_km            22746 non-null  float64
dtypes: bool(3), category(4), datetime64[ns](3), float64(15), int64(9), object(1)
memory usage: 5.8+ MB

```

## Проверка типов и значений новых полей

Наложим на значения новых полей ограничения, руководствуясь здравым смыслом, и занесем их в созданный ранее список автоматизированных проверок:

- отношение жилой площади к общей должно находиться в интервале 0.3 .. 1.0 (Квартира предназначена для жизни, а не для хождения по коридорам. Могут быть и исключения, например, загородные дома. Тем более будет интересно проверить еще раз автоматизированную проверку);
- отношение площади кухни к общей не должно быть больше 0.5 - маловероятно, что кухня занимает половину квартиры). Некоторые аномальные данные мы уже удалили на этапе предобработки. Поищем другие возможные выбросы перед переходом к исследованию на шаге 4.

```
In [158...]: # создадим ограничения для некоторых новых полей
new_checkings = [
    DataChecking(['living_div_total'],
                'Значение {0} должно быть от 0.3 до 1.0 включительно',
                '{`{0}` < 0.3} or ({`{0}` > 1}'),
                lambda _, cdf, __ : display(cdf[['living_div_total',
                                                    'living_area',
                                                    'total_area']])),
    DataChecking(['kitchen_div_total'],
                'Значение {0} не должно быть больше 0.5',
                '`{0}` > 0.5',
                lambda _, cdf, __ : display(cdf[['kitchen_div_total',
                                                    'kitchen_area',
                                                    'total_area']])))
]
data_checkings += new_checkings
```

После добавления новых полей осуществим контрольную автоматизированную проверку, чтобы выявить возможные аномалии:

```
In [159...]: auto_check_all_fields()
auto_check_all_values(check_fix=True, prefix='test 4 value error')
```

## Протокол автоматизированной проверки типов полей набора данных "Информация об объявлениях":

1. Тип **float64** поля "**cityCenters\_km**" (расстояние до центра в километрах) не соответствует заявленному в описании **int64**.

Тип поля "**cityCenters\_km**" (расстояние до центра в километрах) изменен на **int64**.

## Протокол автоматизированной проверки значений набора данных "Информация об объявлениях":

1. Обнаружены замечания к значениям '**living\_div\_total**' (соотношение жилой и общей площади квартиры). Значение living\_div\_total должно быть от 0.3 до 1.0 включительно. Количество записей с нарушением: **197 (0.87%)**.

	<b>living_div_total</b>	<b>living_area</b>	<b>total_area</b>
<b>6</b>	0.284182	10.60	37.30
<b>51</b>	0.236025	38.00	161.00
<b>123</b>	0.291667	14.00	48.00
<b>176</b>	0.292237	15.36	52.56
<b>219</b>	0.289608	17.00	58.70
...	...	...	...
<b>23359</b>	0.276778	10.00	36.13
<b>23388</b>	0.279070	12.00	43.00
<b>23394</b>	0.141304	13.00	92.00
<b>23481</b>	0.270314	10.08	37.29
<b>23573</b>	0.285085	13.59	47.67

197 rows × 3 columns

[См. исправления и комментарии по п. 1](#)

2. Обнаружены замечания к значениям '**kitchen\_div\_total**' (отношения площади кухни к общей). Значение kitchen\_div\_total не должно быть больше 0.5. Количество записей с нарушением: **48 (0.21%)**.

	<b>kitchen_div_total</b>	<b>kitchen_area</b>	<b>total_area</b>
<b>511</b>	0.724638	50.0	69.0
<b>680</b>	0.534884	23.0	43.0
<b>1326</b>	0.615385	32.0	52.0
<b>1336</b>	0.530303	35.0	66.0
<b>2309</b>	0.657051	41.0	62.4
...	...	...	...
<b>21399</b>	0.523810	22.0	42.0
<b>21908</b>	0.561555	26.0	46.3
<b>22087</b>	0.574026	22.1	38.5
<b>22252</b>	0.508021	19.0	37.4
<b>23208</b>	0.594595	22.0	37.0

48 rows × 3 columns

[См. исправления и комментарии по п. 2](#)

Автоматизированная проверка в очередной раз помогла нам без лишних затрат на написание кода выявить аномалии в наборе данных. Оправдал себя и подход создания новых показателей. Были выявлены квартиры с как минимум **странным соотношением жилой и**

**общей площадей**. Возможно, не следует принимать их во внимание на этапе исследовательского анализа.

Не стоит доверять и объявлениям **о квартирах, состоящих по площади практически из одной кухни**.

## Вывод по шагу 3

- Перед вычислением новых показателей **устранена неопределенность даты совершения сделки** и актуальности цены объектов недвижимости. Создано новое поле `price_date`, которое было вычислено добавлением дней публикации объявления к дате его первого появления. Учет этого фактора позволит обойти логическую ошибку в анализе динамики рынка через временные ряды.
- Для упрощения предстоящего исследовательского анализа данных (группировки, фильтрации, поиска аномалий, построения графиков и т.п.) были вычислены новые показатели:
  - **цена квадратного метра**;
  - **день недели, месяц и год** публикации объявления;
  - **год и месяц объявления**;
  - **категория этажа квартиры**;
  - **соотношение жилой и общей площади**;
  - **отношение площади кухни к общей**;
  - **расстояние до центра города в километрах**.
- Информация о назначении и типе новых показателей внесена в **таблицу** для удобства дальнейшего использования.
- Новые значения **проверены** на наличие аномалий с помощью автоматизированной проверки:
  - **обнаружены** 197 квартир с подозрительным соотношением жилой и общей площадей;
  - **отмечены** 48 объявлений, в которых больше половины общей площади квартиры приходится на кухню.
- Таким образом, массив данных подготовлен для проведения исследовательского анализа.

## Шаг 4. Исследовательский анализ данных

### Задание шага 4

Проведите исследовательский анализ данных и выполните инструкции:

- Изучите следующие параметры: **площадь, цена, число комнат, высота потолков**. Постройте гистограммы для каждого параметра.
- Изучите **время продажи квартиры**. Постройте гистограмму.
- Посчитайте среднее и медиану.
  - Опишите, сколько обычно занимает продажа.
  - Когда можно считать, что продажи прошли очень быстро, а когда необычно долго?
- **Уберите** редкие и выбивающиеся значения. Опишите, какие особенности обнаружили.
- Какие факторы больше всего **влияют на стоимость квартиры**?
  - Изучите, зависит ли цена от **площади, числа комнат, удалённости от центра**.
  - Изучите зависимость цены от того, на каком **этаже** расположена квартира: первом, последнем или другом.
  - Также изучите зависимость от **даты размещения**: дня недели, месяца и года.

- Выберите 10 населённых пунктов с наибольшим числом объявлений.
  - Посчитайте среднюю цену квадратного метра в этих населённых пунктах.
  - Выделите среди них населённые пункты с самой высокой и низкой стоимостью жилья. Эти данные можно найти по имени в столбце `'locality_name'`.
- Изучите предложения квартир: для каждой квартиры есть информация о расстоянии до центра.
  - Выделите квартиры в Санкт-Петербурге (`'locality_name'`).
  - Ваша задача — выяснить, какая область входит в центр. Создайте столбец с расстоянием до центра в километрах: округлите до целых значений.
  - После этого посчитайте среднюю цену для каждого километра.
  - Постройте график: он должен показывать, как цена зависит от удалённости от центра.
  - Определите границу, где график сильно меняется — это и будет [центральная зона](#).
- Выделите сегмент квартир в центре.
  - [Проанализируйте](#) эту территорию и изучите следующие параметры: площадь, цена, число комнат, высота потолков.
  - Также выделите факторы, которые влияют на стоимость квартиры (число комнат, этаж, удалённость от центра, дата размещения объявления).
  - Сделайте [выводы](#). Отличаются ли они от общих выводов по всему городу?

## Функция построения гистограмм

Опишем вспомогательную функцию для построения гистограммы или другого графика одного столбца набора данных, исходя из специфики решаемой задачи. На этапе предобработки данных мы обратили внимание, что в некоторых объявлениях встречаются редкие значения, слишком выбивающиеся из общей массы - не попадающие в доверительный интервал. Напишем функцию так, чтобы она строила две гистограммы, если мы задаем доверительные интервалы, первую - для всей выборки, а вторую - только для значений из доверительного интервала. Это поможет нам в динамике оценивать наличие выбросов:

```
In [160...]: def plot_column_hist(data, column_name, mean, perc, **kwargs):
    """Функция построения гистограммы или другого графика одного поля набора данных,
    его среднего арифметического, медианы с указанием названия поля и подписыванием осей
    data - датасет Pandas;
    column_name - имя столбца;
    mean - True (строить средние), False (не строить средние);
    perc - None или доверительный интервал (одним значением или списком);
    kwargs - именованные параметры для передачи в pandas.hist
    например:
        kind='hist', bins=20 - для гистограммы с 20 интервалами
        kind='bar' - столбчатая диаграмма
    """
    # отберем значения для построения гистограммы
    x = data[column_name]

    # определим значения перцентилей, если заданы доверительные интервалы
    if perc is not None:
        if isinstance(perc, list):
            if len(perc) == 1:
                perc = [0, perc[0]]
                print(perc)
        else:
            perc = [0, perc]
            xp = np.percentile(x, perc)

    # построим гистограмму, передав именованные переменные для её особого форматирования
    plt.figure(figsize=(10, 6))
    plt.hist(x, bins=20, density=True, alpha=0.7, color='blue', edgecolor='black')
    if mean:
        plt.axvline(mean, color='red', linestyle='dashed', linewidth=2)
    if xp:
        plt.axvline(xp[0], color='green', linestyle='solid', linewidth=2)
        plt.axvline(xp[1], color='green', linestyle='solid', linewidth=2)
    plt.title(f'Гистограмма для столбца {column_name}')
    plt.xlabel(column_name)
    plt.ylabel('Плотность вероятности')
    plt.grid(True)
    plt.show()
```

```

#hist = x.hist(**kwargs, label='количество')
hist = x.plot(**kwargs, label='количество')

# если необходимо строить среднее и медиану
if mean:
    # определим среднее и медиану
    x_mean = x.mean()
    x_median = x.median()

    # построим линии среднего арифметического и медианы
    plt.axvline(x_mean, color='r', linestyle='--', linewidth=2,
                 label=f'среднее: {x_mean:.02f}')
    plt.axvline(x_median, color='k', linestyle='--', linewidth=2,
                 label=f'медиана: {x_median:.02f}')

# отобразим линии доверительного интервала, если они заданы
if perc is not None:
    for p_i, xp_i in zip(perc, xp):
        if p_i > 0:
            plt.axvline(xp_i, linestyle='-.', linewidth=2,
                         label=f'доверительный интервал {p_i}%%: {xp_i:.02f}')

# подготовим подписи
plt.title(f'Гистограмма поля "{column_name}" - {data_fields[column_name].desc}')
plt.xlabel(f'{data_fields[column_name].desc}')
plt.ylabel('количество значений')
plt.legend()

# выведем рисунок на экран
plt.show()

# при необходимости рекурсивно вызовем построение гистограммы для доверительного инт
if perc is not None:
    display(HTML(f'<br>Построим эту же гистограмму, но только для значений из довер
                f'интервала {min(perc)}% - {max(perc)}%:'))
    plot_column_hist(data[(data[column_name] >= xp.min()) & (data[column_name] <=
        column_name, mean, None, **kwargs)

# возвратим ссылку на гистограмму
return hist

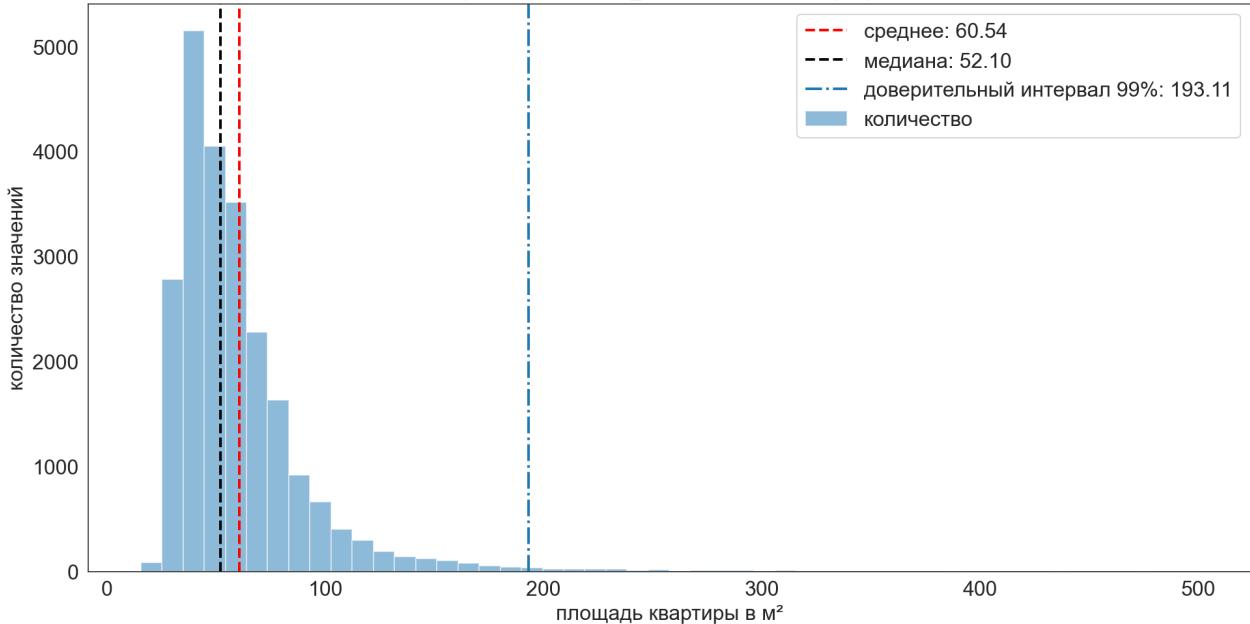
```

## Построение гистограмм

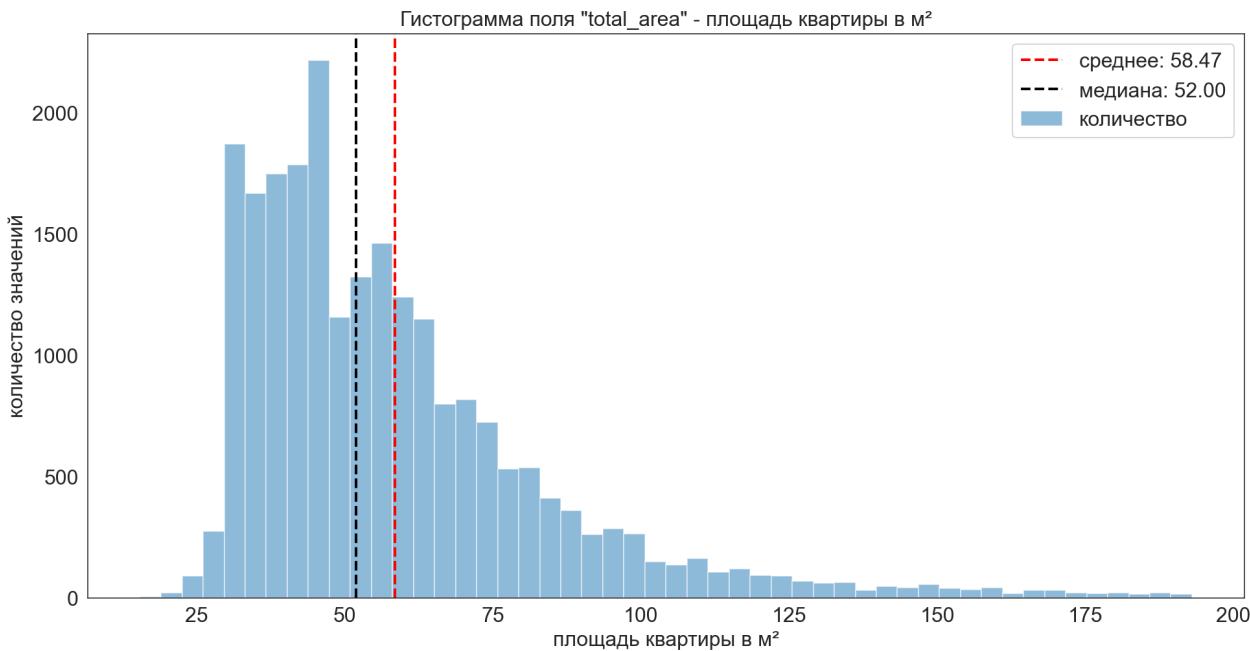
### Гистограмма общей площади

Построим гистограммы в доверительном интервале 99%, т.е. отделим от основной выборки 1% самых больших по общей площади квартир.

```
In [161]: plot_column_hist(ads, 'total_area', True, 99, kind='hist', bins=50, alpha=0.5, figsize=
```

Гистограмма поля "total\_area" - площадь квартиры в м<sup>2</sup>

Построим эту же гистограмму, но только для значений из доверительного интервала 0% - 99%:



Видим, что гистограмма приобрела детали, когда мы отбросили слишком большие значения. В дальнейшем при определенных видах анализа можем отбросить квартиры площадью более 193 м<sup>2</sup> (в соответствии с [графиком](#) они не попали в доверительный интервал).

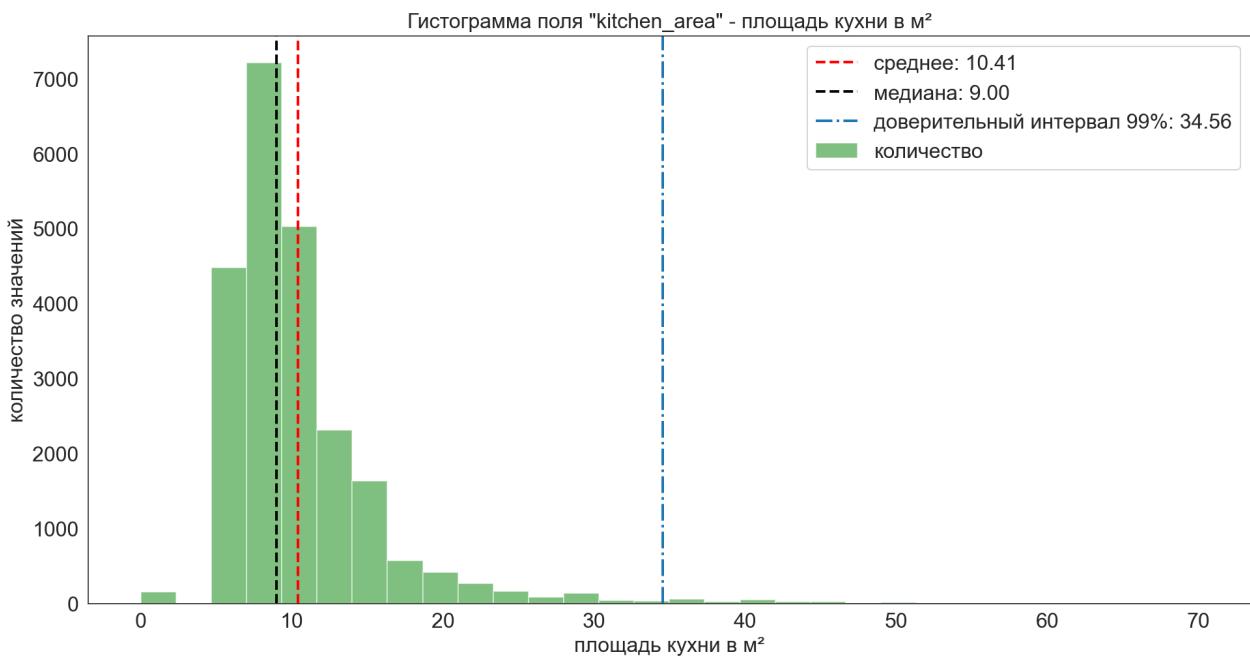
При отбрасывании выбивающихся значений немного изменились средняя и медиана, они стали точнее характеризовать основную выборку.

Чаще всего встречаются объявления о продаже квартир общей площадью 52 квадратных метра.

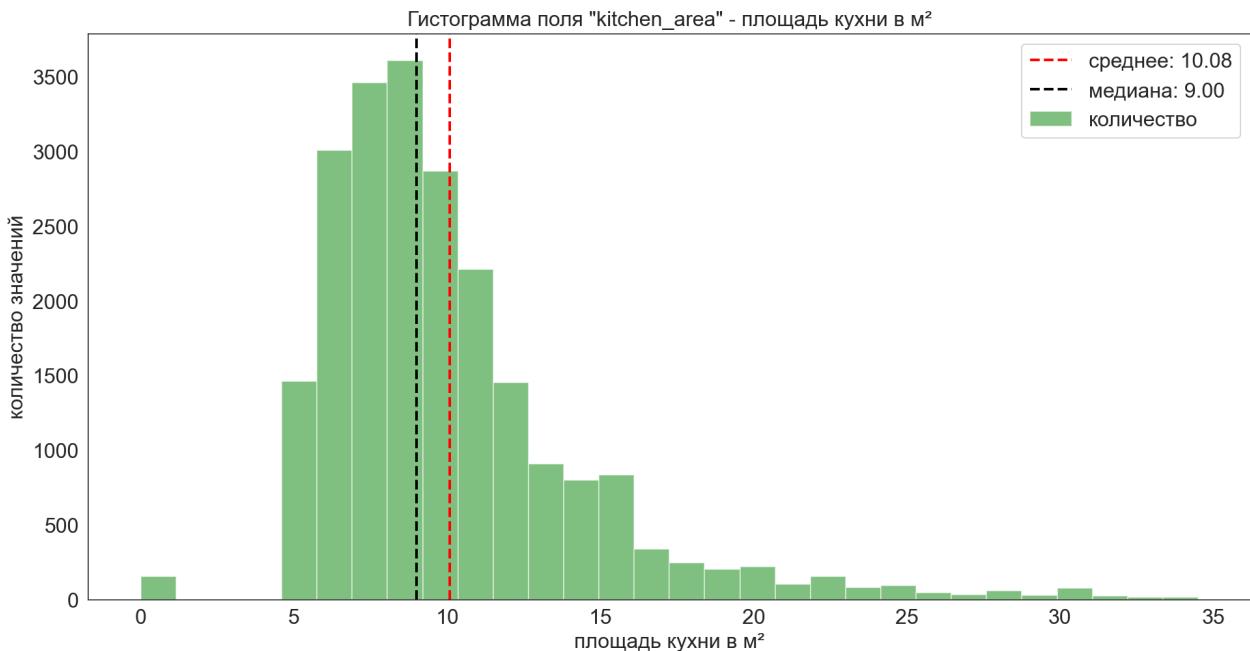
Аналогичным образом построим гистограммы других параметров.

## Гистограмма площади кухни

```
In [162]: plot_column_hist(ads, 'kitchen_area', True, 99, kind='hist', bins=30, color='g', alpha=
```



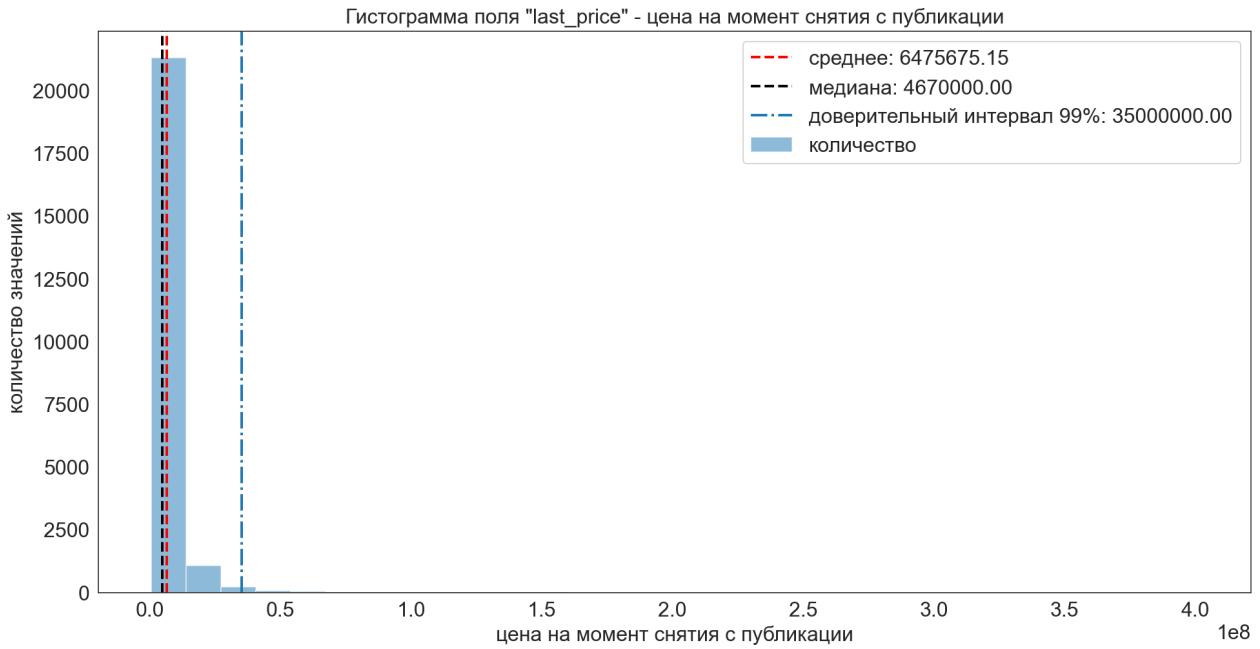
Построим эту же гистограмму, но только для значений из доверительного интервала 0% - 99%:



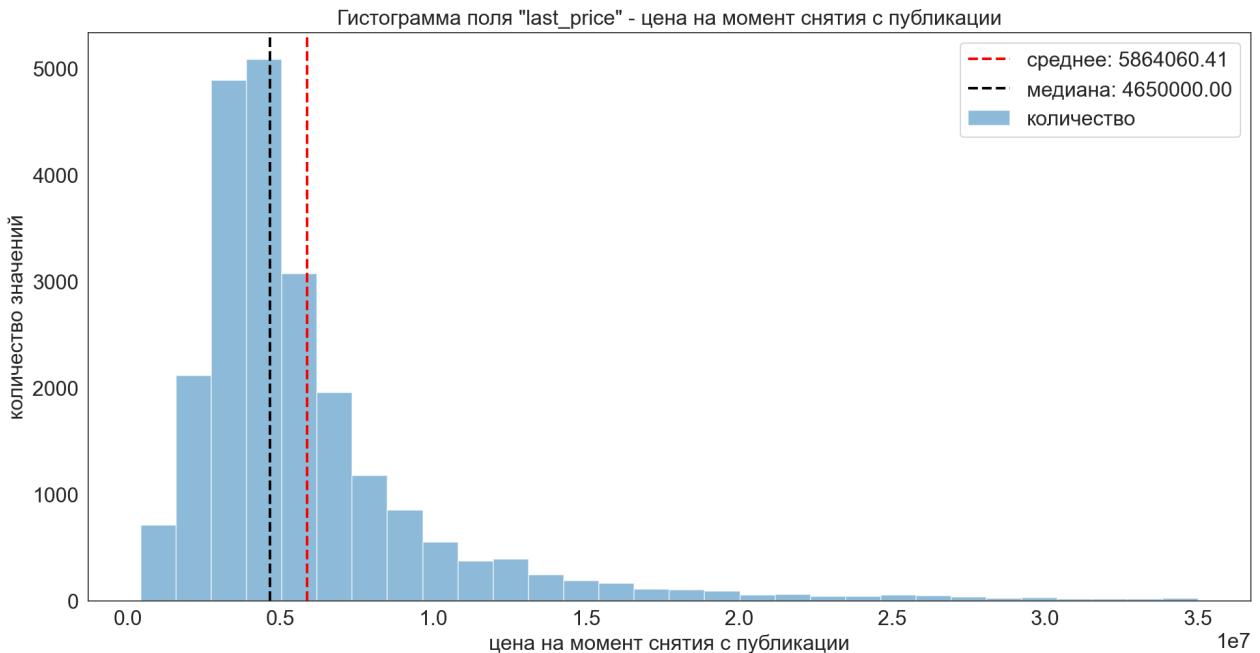
Особняком стоят квартиры-студии с нулевой площадью кухни. Наиболее часто встречаются кухни в 9 квадратных метров.

### Гистограмма цены квартиры

```
In [163]: plot_column_hist(ads, 'last_price', True, 99, kind='hist', bins=30, alpha=0.5, figsize=
```



Построим эту же гистограмму, но только для значений из доверительного интервала 0% - 99%:

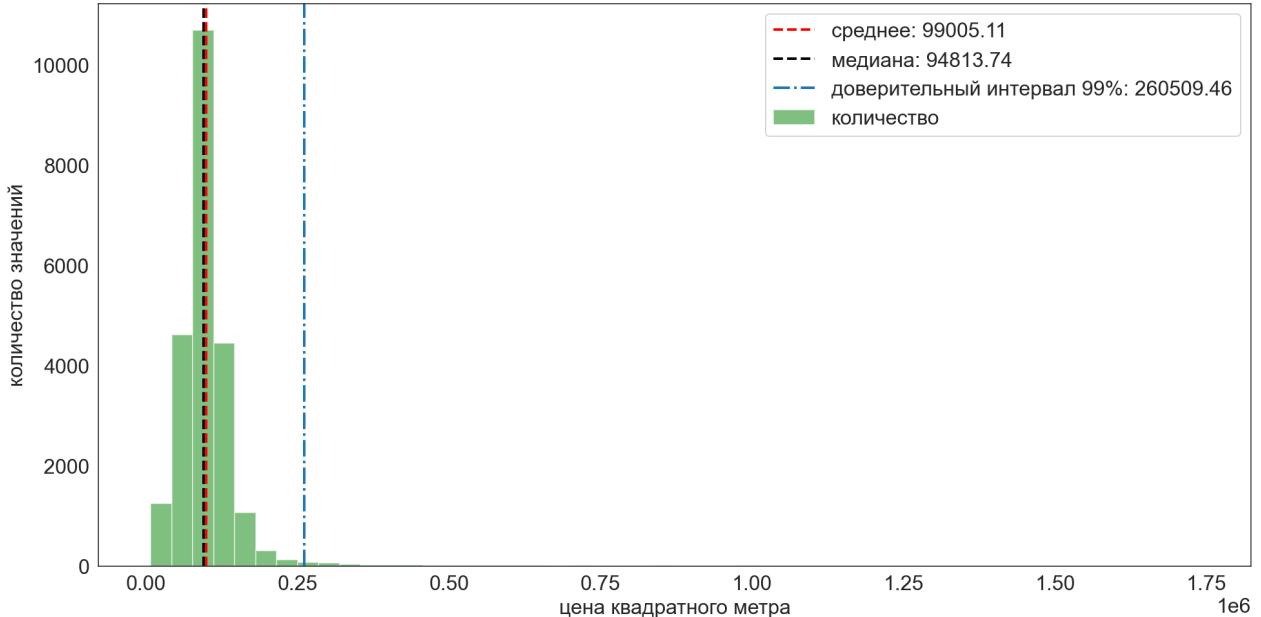


Разброс цен на квартиры достаточно большой. Среднее значение - 5.9 млн.

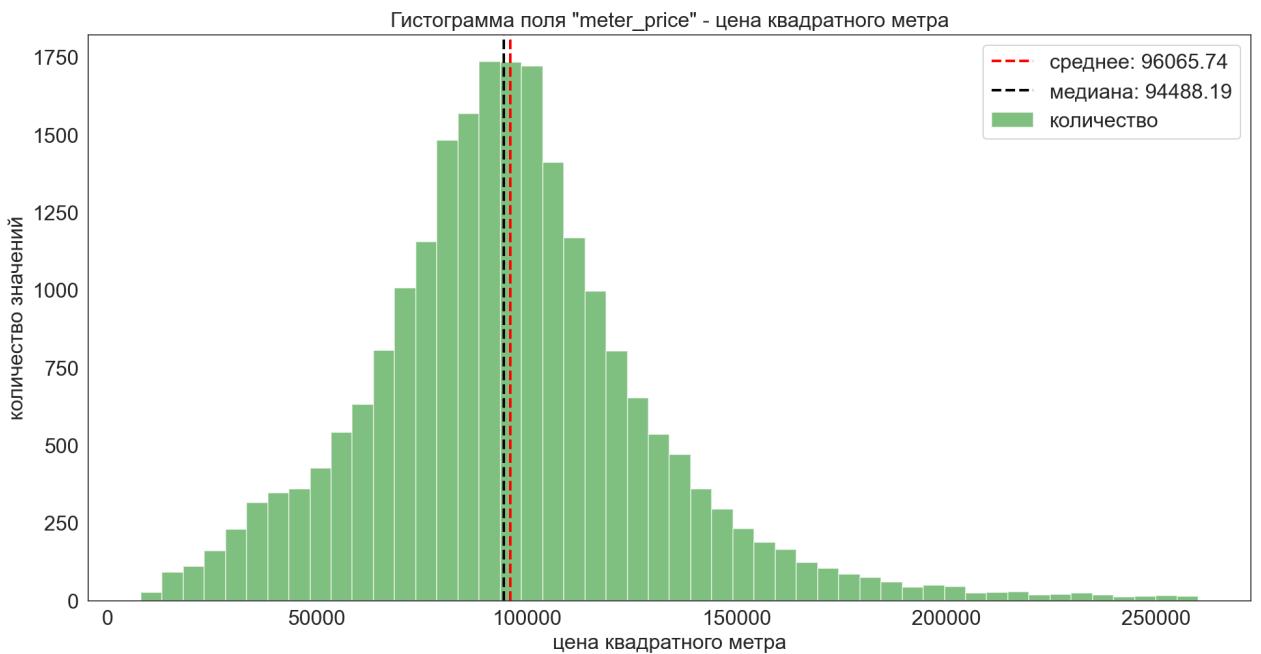
### Гистограмма цены квадратного метра

```
In [164]: plot_column_hist(ads, 'meter_price', True, 99, kind='hist', bins=50, color='g', alpha=0.5)
```

Гистограмма поля "meter\_price" - цена квадратного метра



Построим эту же гистограмму, но только для значений из доверительного интервала 0% - 99%:

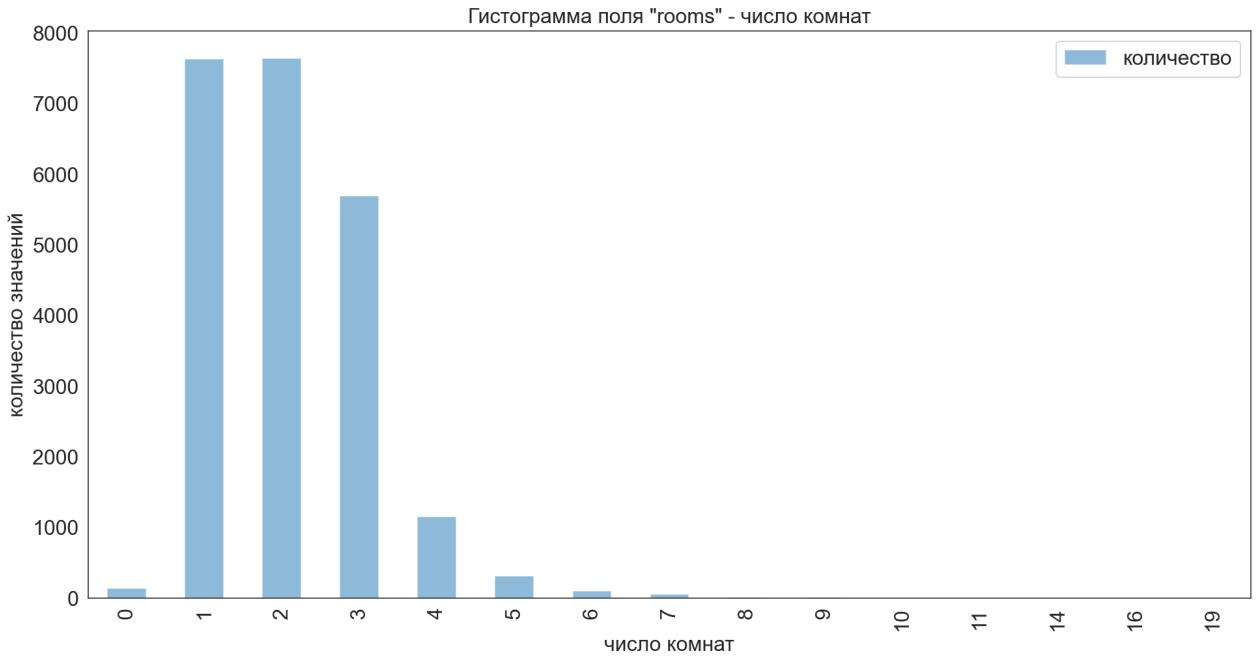


Гистограмма цены квадратного метра выглядит компактнее гистограммы общей стоимости квартиры. Исключение аномально редких значений цены квадратного метра улучшило качество гистограммы. Среднее арифметическое и медиана не сильно отстают друг от друга. Значит цена квадратного метра - хороший показатель для применения в анализе рынка.

Среднее значение цены квадратного метра: 94-96 тыс.

### Гистограмма количества комнат

```
In [165...]: plot_column_hist(ads.groupby('rooms').agg(rooms=('rooms', 'count')), 'rooms', False, None, kind='bar', alpha=0.5, figsize=(16,8));
```

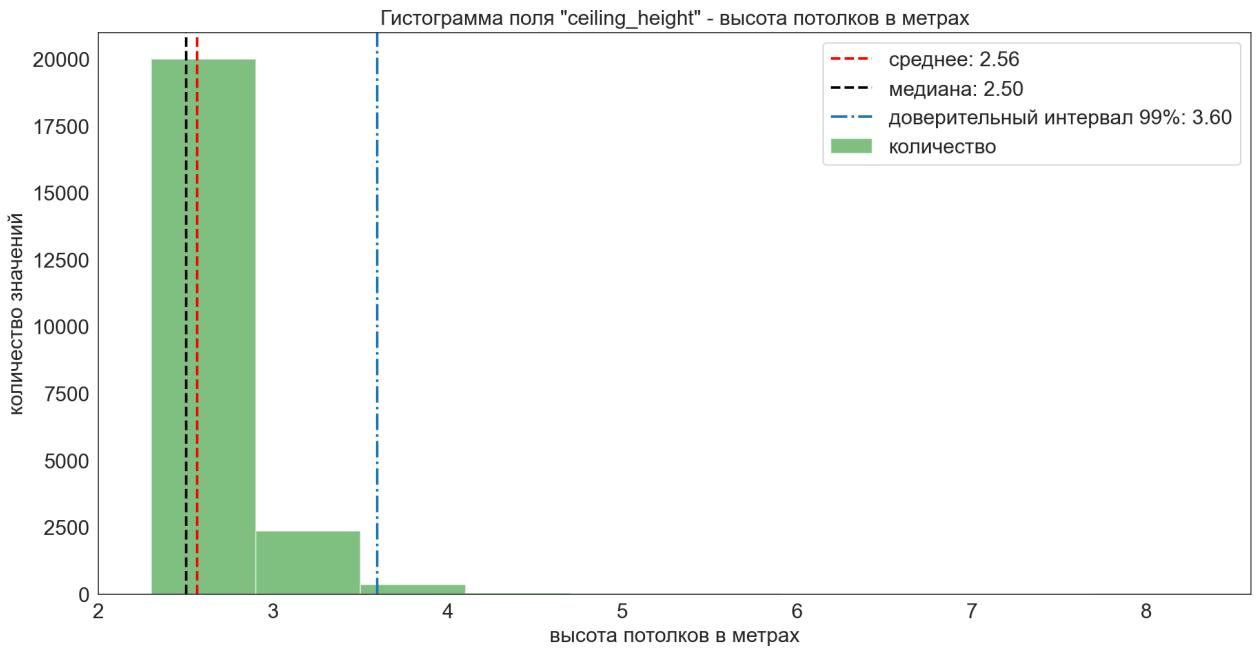


Интересные наблюдения:

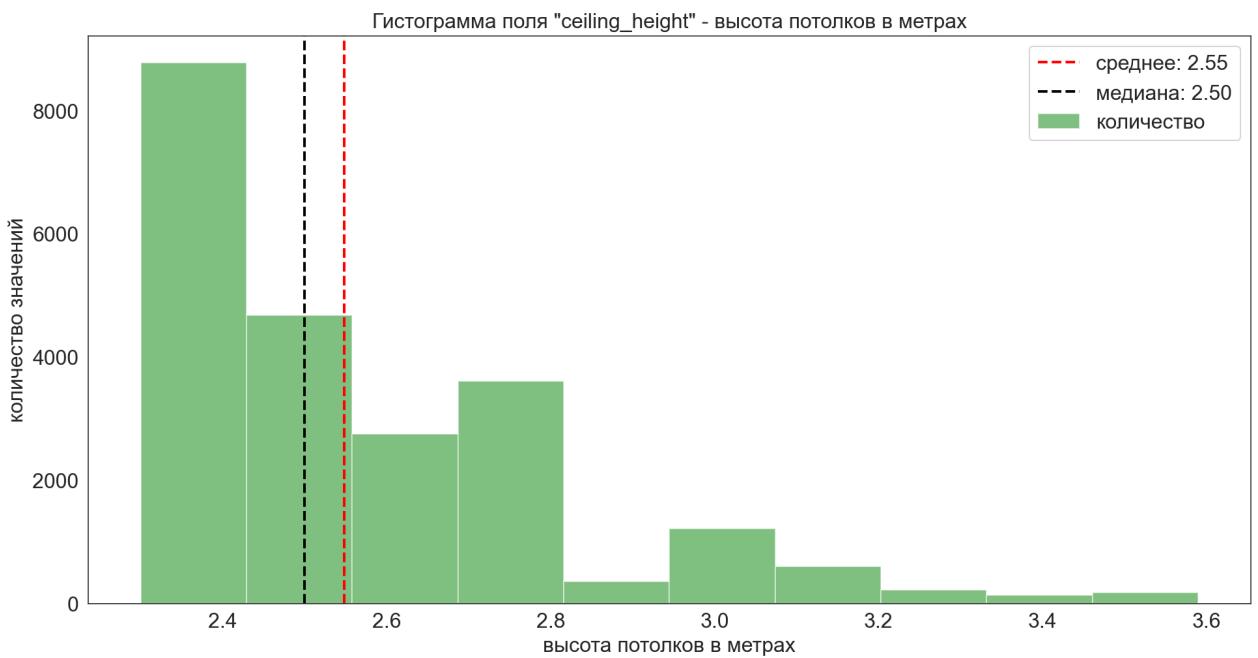
- в основном торгуются однокомнатные и двухкомнатные квартиры;
- квартир-студий на рынке совсем немного;
- реже встречаются только квартиры от шести и более комнат.

### Гистограмма высоты потолков

```
In [166]: plot_column_hist(ads, 'ceiling_height', True, 99, kind='hist', bins=10, color='g', alpha=0.8)
```



Построим эту же гистограмму, но только для значений из доверительного интервала 0% - 99%:

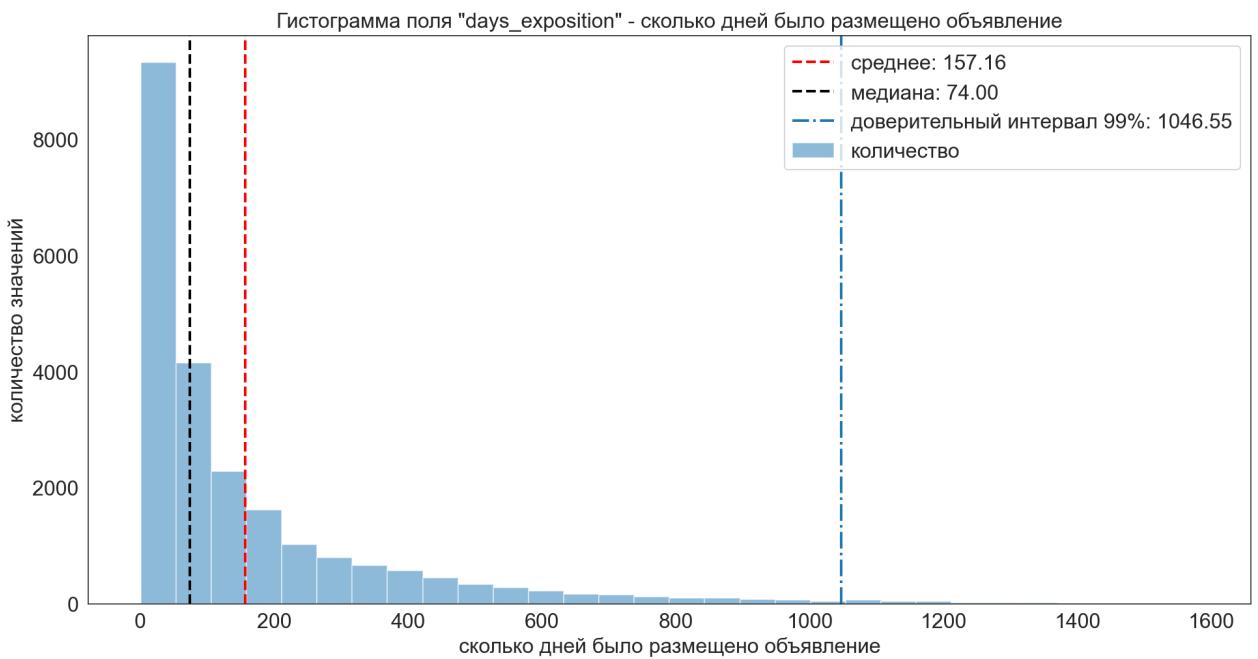


Типовая квартира имеет потолок высотой около 2.5 метров. У 99% квартир потолки не выше 3.6 метров.

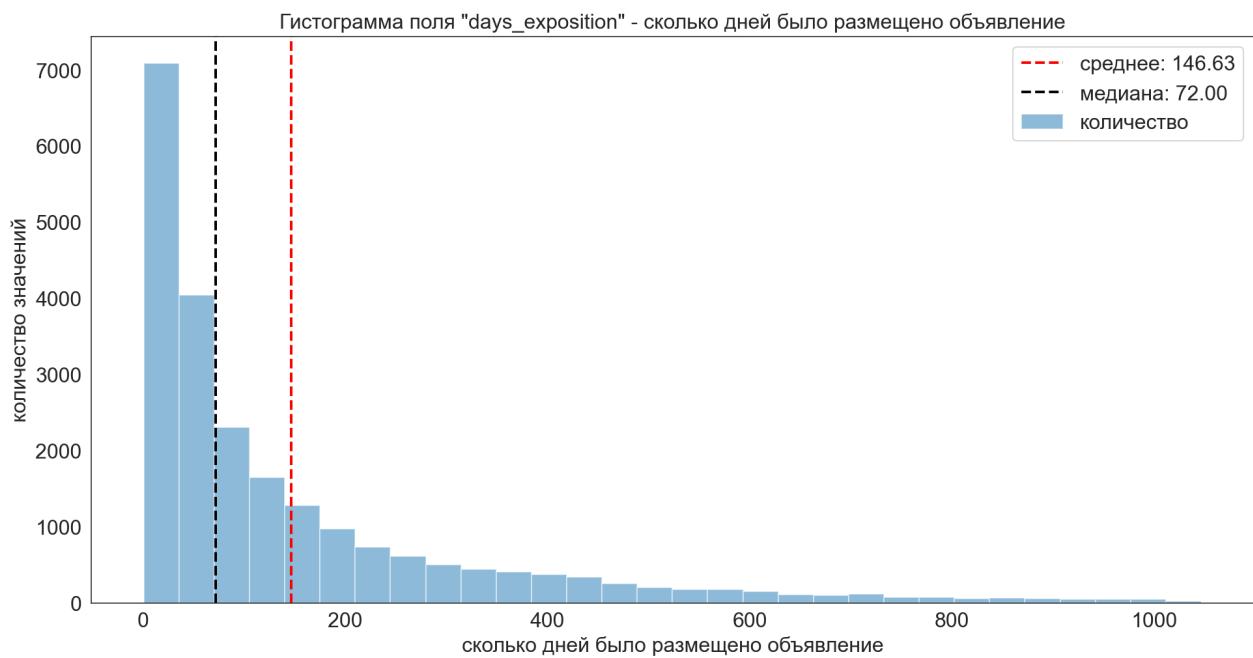
## Гистограммы времени продажи квартиры

### Гистограмма срока публикации объявления

```
In [167]: plot_column_hist(ads, 'days_exposition', True, 99, kind='hist', bins=30, alpha=0.5, fig
```



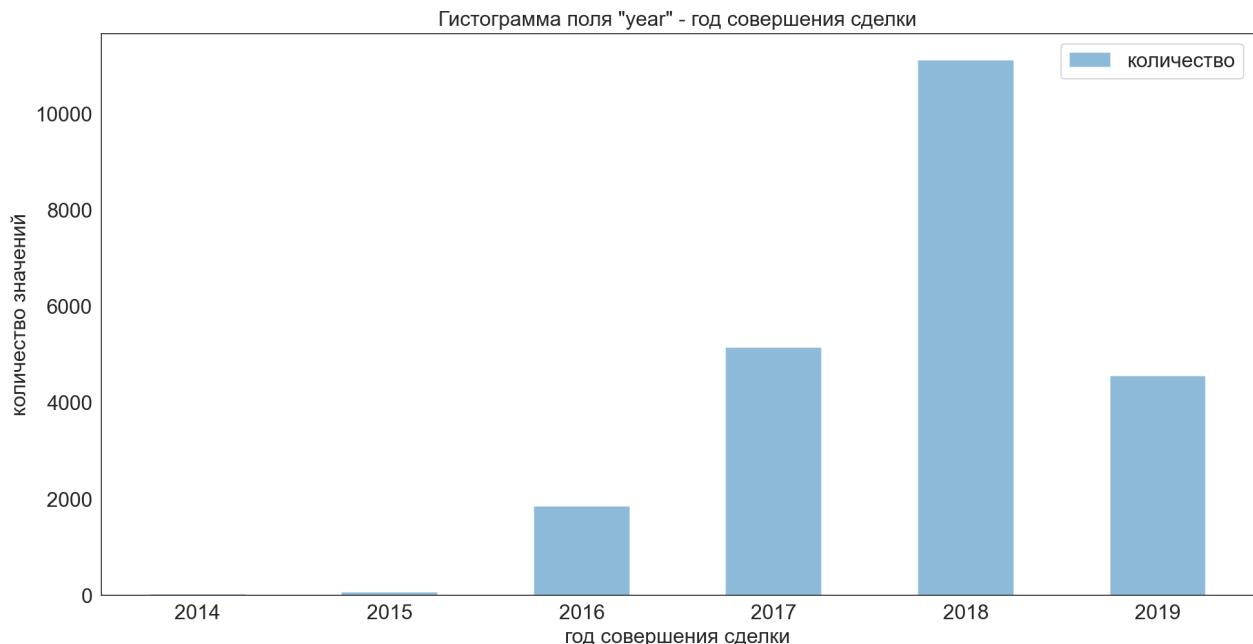
Построим эту же гистограмму, но только для значений из доверительного интервала 0% - 99%:



Судя по медиане, **половина объявлений реализуется за 74 дня**. Остальные менее привлекательные объявления гораздо дольше ожидают своего покупателя и утягивают среднее значение вправо от медианы. Если удается реализовать квартиру за два месяца (быстрее, чем большая часть других вариантов), то можно считать это быстрой сделкой.

### Гистограмма года совершения сделки

```
In [168...]: plot_column_hist(ads.groupby('year').agg(year=('year', 'count')), 'year', False, None, kind='bar', alpha=0.5, rot=0, figsize=(16,8));
```



С 2016 по 2018 год наблюдается тенденция к активизации рынка недвижимости, количество сделок растет. 2019 год в расчет не берем, массив данных не содержит информацию по нему за весь период. Наблюдения собраны не за весь 2019 год, а только по май (видимо, массив данных был подготовлен в середине 2019 года):

```
In [169...]: ads.price_date.min()
```

```
Out[169...]: Timestamp('2014-11-27 00:00:00')
```

```
In [170...]: ads.price_date.max()
```

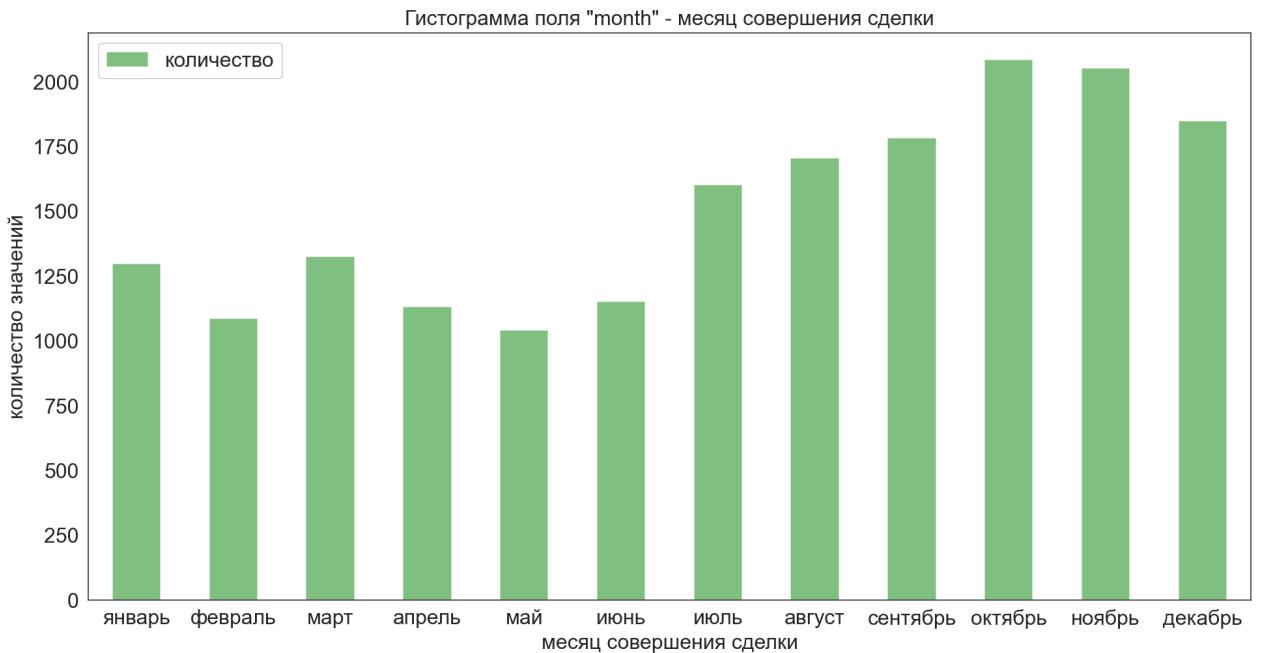
```
Out[170...]: Timestamp('2019-05-03 00:00:00')
```

Объявлений за 2014-2015 год слишком мало, возможно, они содержат ошибочные сведения. В дальнейшем анализе динамики цен их можно не учитывать.

### Гистограмма месяца совершения сделки

Проанализируем активность на рынке недвижимости по месяцам на основе информации за полные 2016-2018 гг.

```
In [171...]: plot_column_hist(ads[(ads.year >= 2016) & (ads.year <= 2018)].groupby('month').agg(month=False, None, kind='bar', color='g', alpha=0.5, rot=0, figsize=(16,8));
```



Пики активности на рынке в 2016-2018 гг. приходились на последние три месяца года.

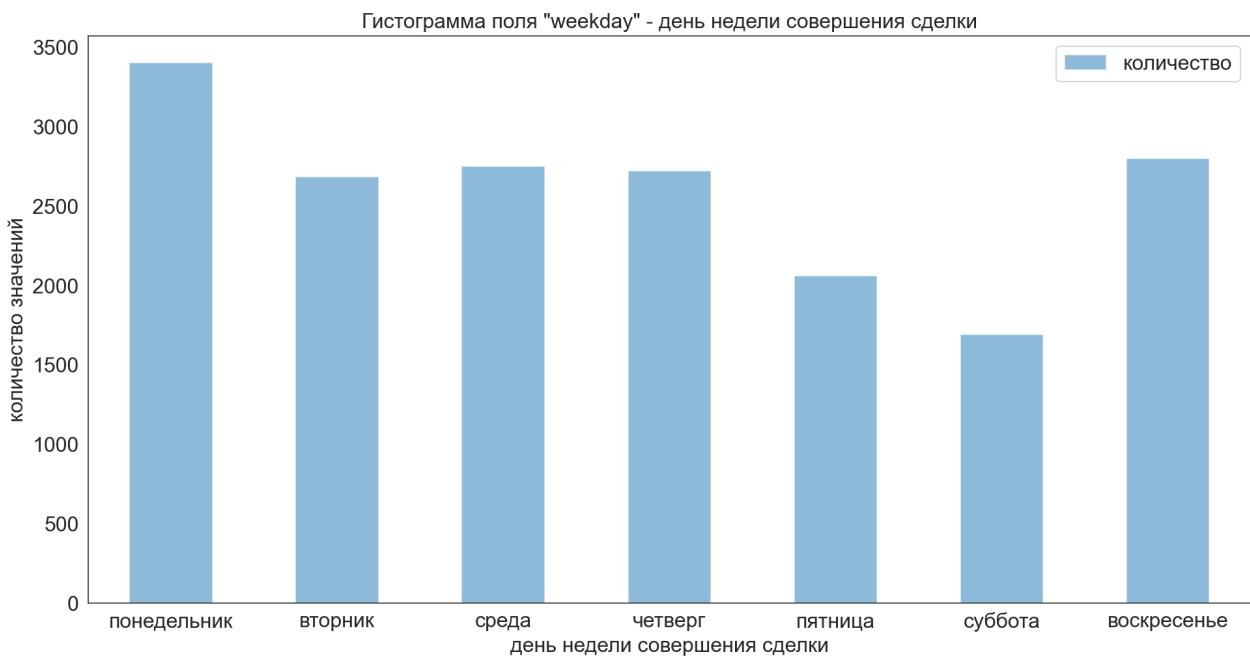
Назовем это явление предновогодним ажиотажем.

Эти выводы [подтверждаются и экспертами ЦИАН](#). Значит мы приняли правильное решение привязывать цены не на момент публикации объявления, а на момент его снятия. Результаты анализа будут точнее.

### Гистограмма дня недели совершения сделки

Пик снятия объявлений о недвижимости приходится на понедельник. Эксперты по недвижимости говорят, что именно в этот день чаще заключаются сделки:

```
In [172...]: plot_column_hist(ads[(ads.year >= 2016) & (ads.year <= 2018)].groupby('weekday').agg(weekday=('weekday', 'count')), 'weekday', False, None, kind='bar', alpha=0.5, rot=0, figsize=(16,8));
```



## Удаление редких и выбивающихся значений

Построенные выше гистограммы позволяют удалить редкие и выбивающиеся значения. В дальнейшем анализе не будем учитывать следующие объявления о квартирах:

- за 2014-2015 годы (объявлений [слишком мало](#));
- с общей площадью более 193 м<sup>2</sup> (они не попадают в выбранный [доверительный интервал](#));
- с площадью кухни больше 35 м<sup>2</sup> (они не попадают в выбранный [доверительный интервал](#));
- с потолками выше 3.6 м (такие объекты недвижимости [редки](#) и их не следует смешивать с обычными квартирами);
- с ценой 1 м<sup>2</sup> выше 260510 (они не попадают в выбранный [доверительный интервал](#)).

Ссылки на объявления, не выбивающиеся из общей массы, занесем в отдельный датафрейм `valid_ads` для дальнейшего анализа:

```
In [173...]: valid_ads = ads[(ads.price_date >= '2016.01.01') &
                     (ads.total_area <= 193) &
                     (ads.kitchen_area <= 35) &
                     (ads.ceiling_height <= 3.6) &
                     (ads.meter_price <= 260510)]
```

```
In [174...]: print(f'Было отброшено объявлений: {ads.shape[0] - valid_ads.shape[0]} из {ads.shape[0]}')
print(f'Осталось объявлений для дальнейшего анализа: {valid_ads.shape[0]}')
print(f'({valid_ads.shape[0] / ads.shape[0]:.1%} от первоначального количества)')
```

Было отброшено объявлений: 767 из 22746

Осталось объявлений для дальнейшего анализа: 21979 (96.6% от первоначального количества)

## Какие факторы больше всего влияют на стоимость квартиры

### Площадь, число комнат и удалённость от центра

Для проверки, зависит ли цена от площади, числа комнат, высоты потолков и удалённости от центра, построим парные диаграммы:

```
In [175...]: def plot_pairof_with_desc(df):
    save_columns = df.columns
    try:
```

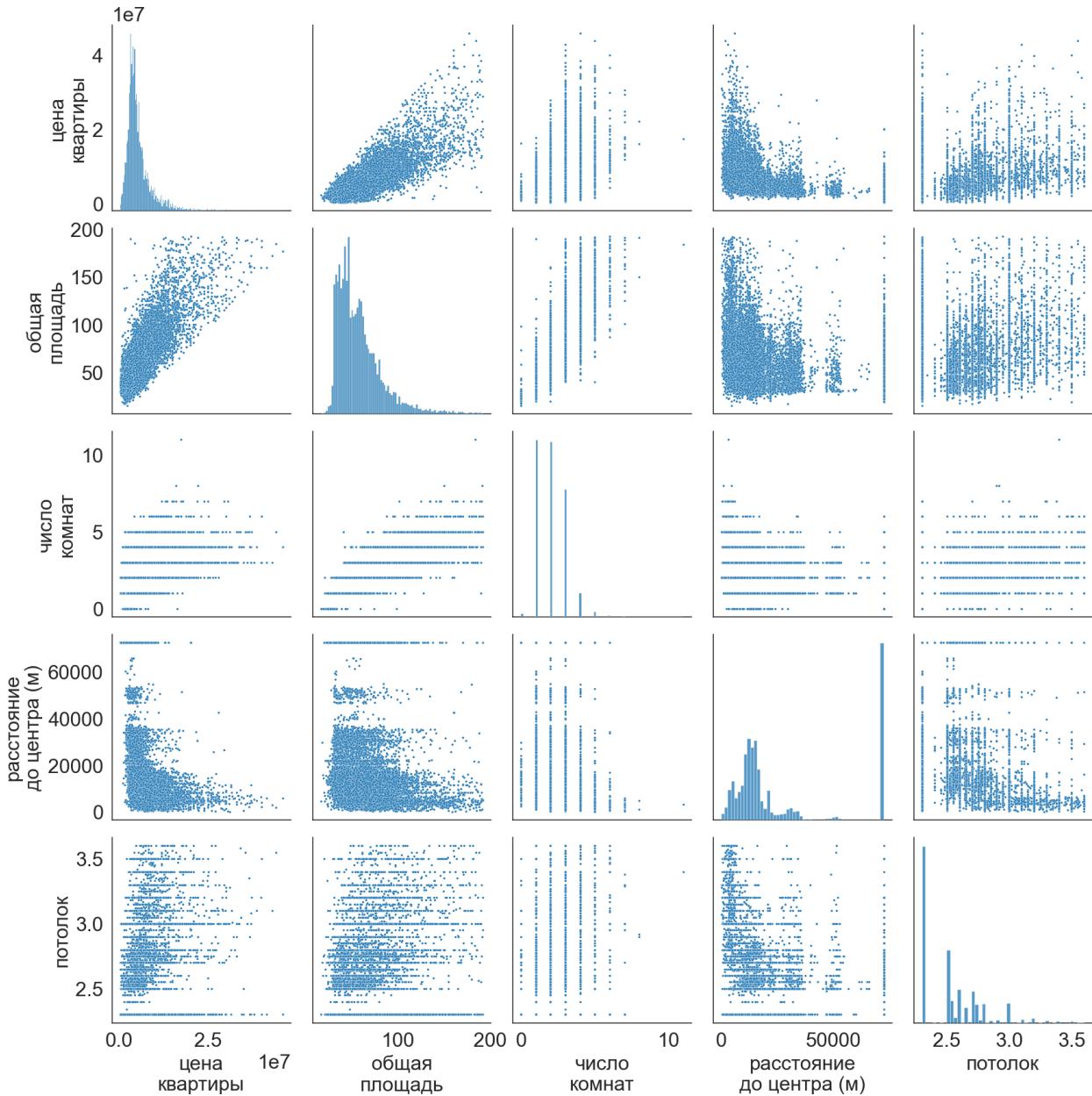
```

df.columns = get_desc_short(df.columns)
sns.pairplot(df, plot_kws={'s': 3});
finally:
    df.columns = save_columns

```

In [176]: pair\_data = valid\_ads[['last\_price', 'total\_area', 'rooms', 'cityCenters\_nearest', 'cei']]

In [177]: plot\_pairepot\_with\_desc(pair\_data)



Мы можем убедиться, что **три фактора (площадь, число комнат и удалённость от центра) в разной степени оказывают влияние на стоимость квартиры:**

- Общая площадь квартиры линейно повышает верхнюю границу возможной цены недвижимости (зная площадь, мы сможем определить её максимально возможную цену, но практически всегда найдется недвижимость такого же размера, но дешевле).
- Число комнат оказывает чуть меньшее влияние на цену квартиры и тоже больше влияет на верхнюю границу цены.
- А вот удаление от центра снижает потолок цены: в центре есть и дорогие и дешевые квартиры, но на периферии дорогих квартир уже меньше, и меньше. Степень влияния расстояния меньше, чем у остальных показателей.
- Высота потолка оказывает наименьшее влияние на стоимость квартиры.

Эти умозаключения подтверждаются и матрицей корреляций:

```
In [178... def display_df_corr(df):
    save_columns = df.columns
    try:
        df.columns = get_desc_long(df.columns)
        display(df.corr().iloc[1:, [0]])
    finally:
        df.columns = save_columns
```

```
In [179... display_df_corr(pair_data)
```

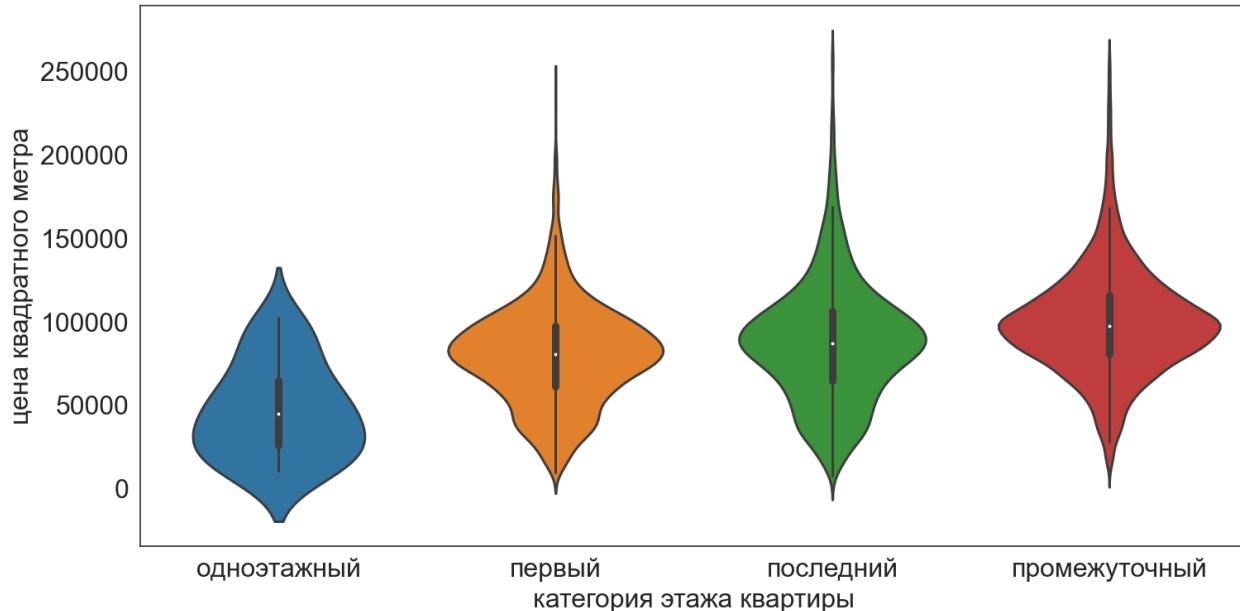
цена на момент снятия с публикации	
площадь квартиры в м <sup>2</sup>	0.792302
число комнат	0.499692
расстояние до центра города в метрах	-0.439827
высота потолков в метрах	0.326778

## Этаж квартиры

**На стоимость жилья оказывает влияние его этаж.** Самые дешевые - одноэтажные дома.

Квартиры на первом этаже в общем случае значительно дороже, но чуть уступают помещениям на последних этажах. Наибольшей ценой за квадратный метр могут похвастаться промежуточные этажи:

```
In [180... pair_data = valid_ads[['floor_category', 'meter_price']]
pair_data.columns = list(map(lambda x: data_fields[x].desc, pair_data.columns))
sns.violinplot(x='категория этажа квартиры', y='цена квадратного метра', data=pair_data,
                order=['одноэтажный', 'первый', 'последний', 'промежуточный']);
```

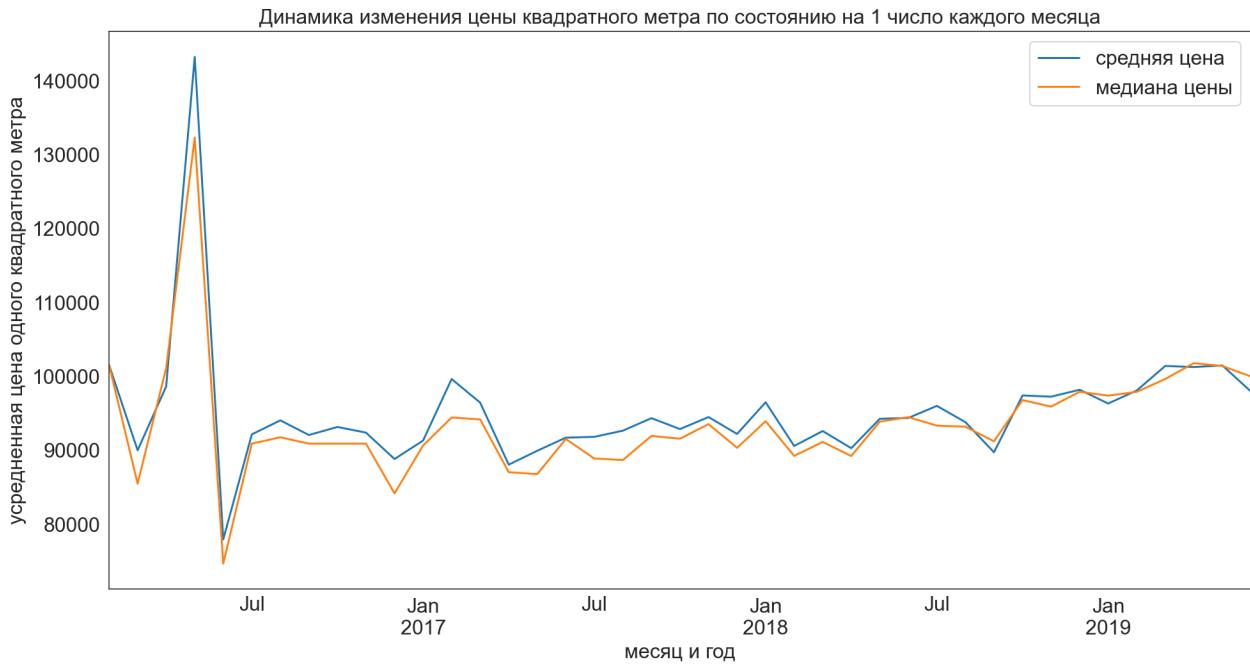


## Динамика изменения цены квадратного метра по месяцам

Построим графики зависимости средних значений цены квадратного метра жилья по времени:

```
In [181... data = valid_ads.groupby('year_month').agg(price_mean=('meter_price', 'mean'),
                                                price_median=('meter_price', 'median'))
ax = data.plot(figsize=(16,8))
plt.title('Динамика изменения цены квадратного метра по состоянию на 1 число каждого ме')
plt.xlabel('месяц и год')
plt.ylabel('усредненная цена одного квадратного метра')
```

```
plt.legend(['средняя цена', 'медиана цены'])
plt.show()
```

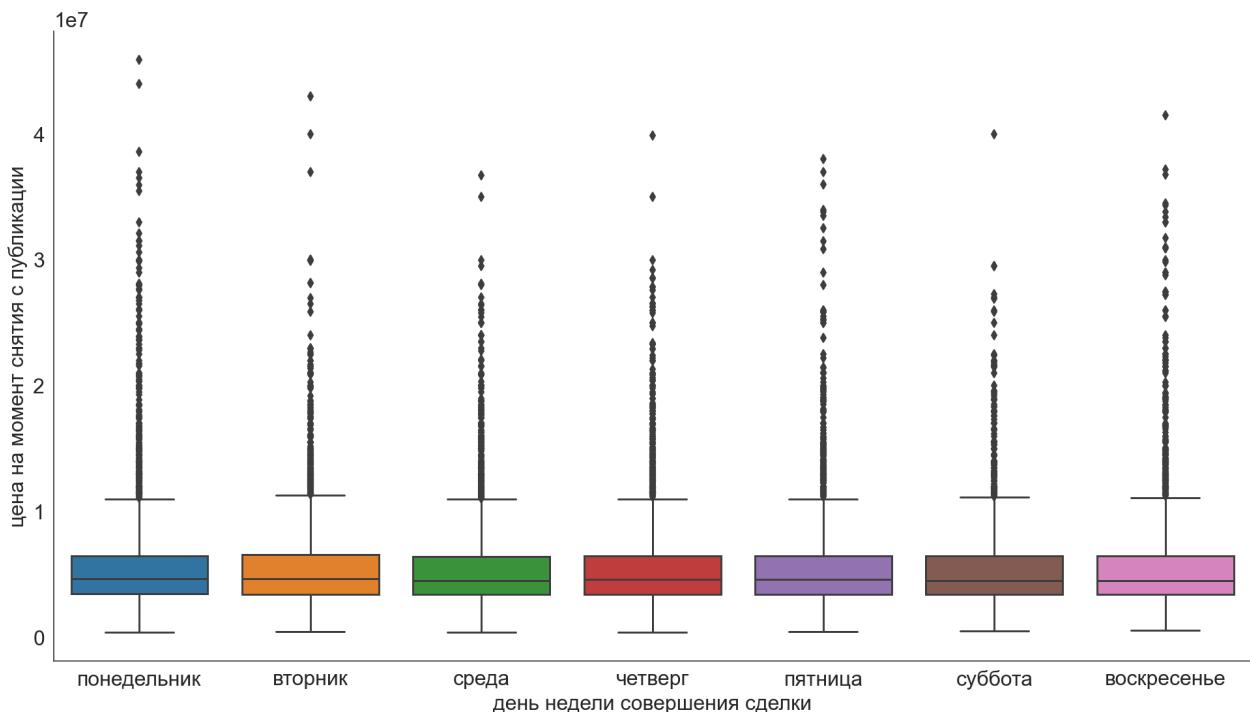


**Видим, что в 2016 году цены на недвижимость демонстрировали повышенную волатильность. С января 2017 года рынок стабилизировался - стоимость одного квадратного метра демонстрирует тенденцию к росту, но колеблется уже не так резко, как это было в 2016 году.**

### День недели публикации объявления

А вот никакой зависимости стоимости от дня недели нет. Средние цены для всех семи дней одинаковые:

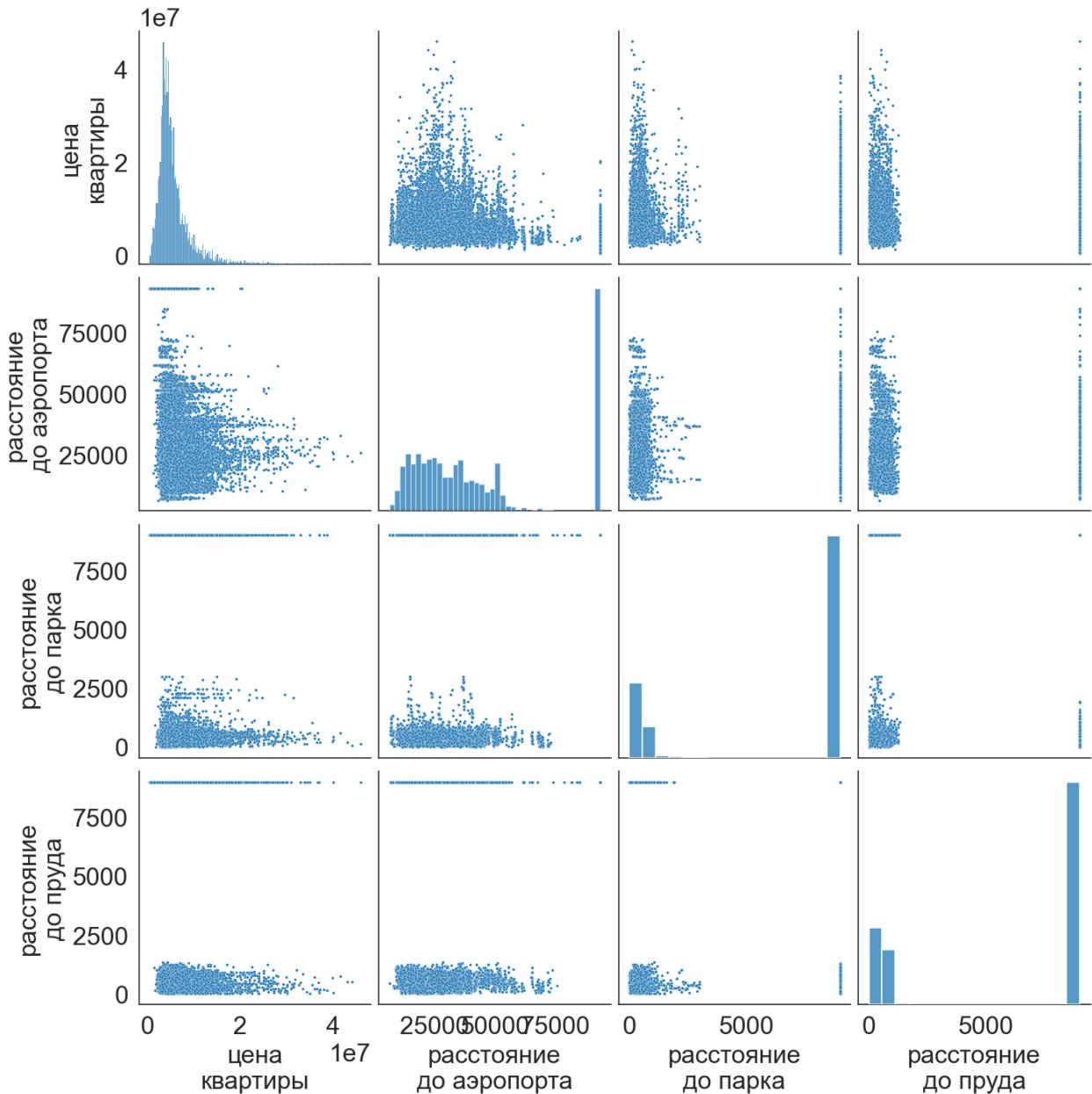
```
In [182...]: pair_data = valid_ads[['last_price', 'weekday', 'month', 'year']]
pair_data.columns = list(map(lambda x: data_fields[x].desc, pair_data.columns))
g = sns.catplot(x='день недели совершения сделки', y='цена на момент снятия с публикации')
g.fig.set_figwidth(16); g.fig.set_figheight(8);
```



### Расстояние до объектов

```
In [183...]: pair_data = valid_ads[['last_price', 'airports_nearest', 'parks_nearest', 'ponds_neares
```

```
In [184...]: plot_paiрpot_with_desc(pair_data)
```



```
In [185...]: display_df_corr(pair_data)
```

цена на момент снятия с публикации	
расстояние до ближайшего аэропорта в метрах	-0.352578
расстояние до ближайшего парка в метрах	-0.274358
расстояние до ближайшего водоёма в метрах	-0.261006

**Расстояния до инфраструктурных объектов оказывает меньшее влияние на стоимость жилья. Удаленность от аэропорта имеет особенность: в непосредственной близости к этому шумному месту цены существенно ниже.**

## 10 населённых пунктов с наибольшим числом объявлений

Отберем 10 населённых пунктов с наибольшим числом объявлений первым способом:

```
In [186...]: locality_stat_series = valid_ads.locality_name.value_counts()[:10]
display(locality_stat_series)
```

Санкт-Петербург	14478
поселок Мурино	492

поселок Шушары	424
Всеволожск	380
Пушкин	345
Колпино	329
поселок Парголово	317
Гатчина	293
деревня Кудрово	266
Выборг	226

Name: locality\_name, dtype: int64

Отберем 10 населённых пунктов с наибольшим числом объявлений вторым способом, сразу собрав статистику, необходимую для выполнения [задания](#), и отсортировав результат по убыванию средней цены квадратного метра жилья:

```
In [187...]: locality_stat_df = valid_ads.groupby('locality_name') \
            .agg(N=('locality_name', 'count'),
                  meter_price_mean=('meter_price', 'mean')) \
            .sort_values('N', ascending=False).iloc[:10] \
            .sort_values('meter_price_mean', ascending=False)
locality_stat_df.columns = ('количество объявлений', 'средняя цена за м2')
display(locality_stat_df)
```

locality_name	количество объявлений	средняя цена за м <sup>2</sup>
Санкт-Петербург	14478	109231.017223
Пушкин	345	102369.361333
деревня Кудрово	266	92295.553337
поселок Парголово	317	90378.699994
поселок Мурино	492	84909.292166
поселок Шушары	424	78572.643423
Колпино	329	75257.686330
Гатчина	293	68704.955484
Всеволожск	380	67139.154454
Выборг	226	57907.536018

Выделим среди отобранных населённые пункты с самой высокой и низкой стоимостью жилья:

```
In [188...]: # определим индексы экстремумов
locality_top_1 = locality_stat_df['средняя цена за м2'].idxmax()
locality_top_10 = locality_stat_df['средняя цена за м2'].idxmin()

# выведем результат
display(HTML(f'{locality_top_1} - лидер топ-10 по цене квадратного метра ' \
            f'({{locality_stat_df["средняя цена за м2"].max():.0f}}).'))
display(HTML(f'{locality_top_10} замыкает десятку ' \
            f'({{locality_stat_df["средняя цена за м2"].min():.0f}}).'))
```

**Санкт-Петербург** - лидер топ-10 по цене квадратного метра (109231).

**Выборг** замыкает десятку (57908).

## Анализ квартир в Санкт-Петербурге

### Поиск центра города

Выясним, какая область входит в центр. Воспользуемся столбцом с расстоянием до центра в километрах, [созданным ранее](#). Постройте ступенчатый график средней цены для каждого

километра. Он показывает, как цена зависит от удалённости от центра. Определим границу, где график сильно меняется — это и будет центральная зона.

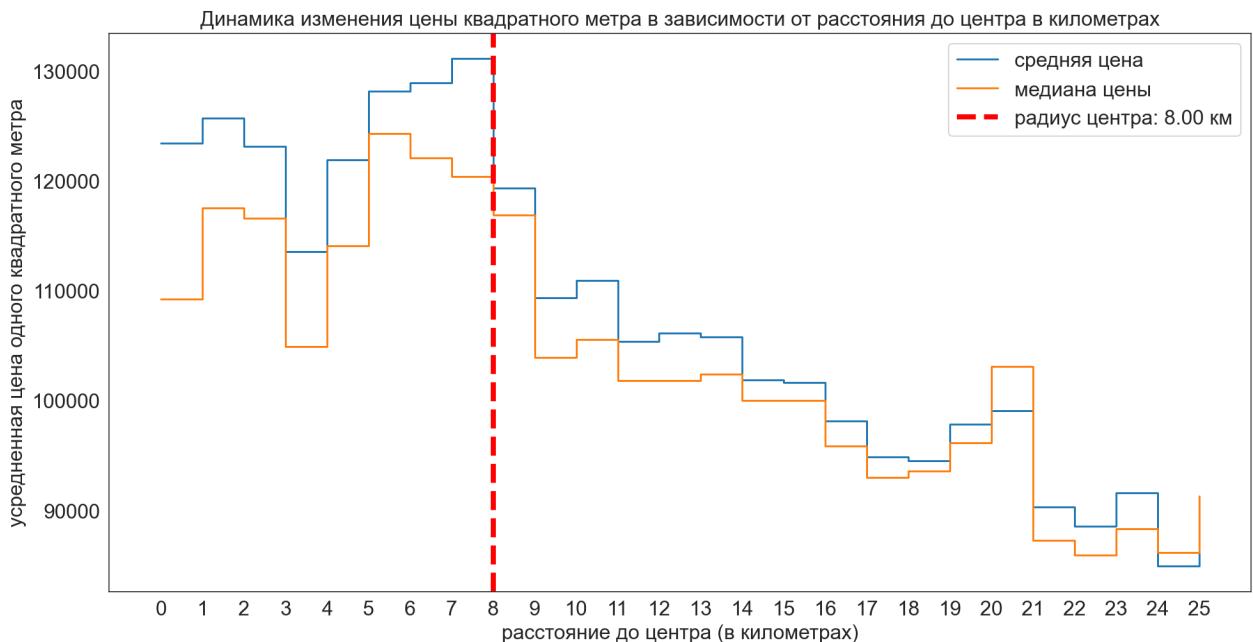
Для нивелирования возможных ошибок, связанных с отмеченной нами [волатильностью цен в 2017 году](#), примем в расчет данные по Санкт-Петербургу только за 2017-2018 годы.

Ограничим поиски двадцатью пятью километрами, чтобы не заниматься лишними вычислениями.

Выдвинем гипотезу, что центральная зона ограничена окружностью радиусом 8 километров. Проверим её построив графики:

```
In [189...]: city_centre_radius_km = 8.0
```

```
In [190...]: data = valid_ads[(valid_ads.locality_name == 'Санкт-Петербург') &
                     (valid_ads.cityCenters_km <= 25) &
                     (valid_ads.year.isin([2017, 2018]))] \
                     .groupby('cityCenters_km').agg(price_mean=('meter_price', 'mean'),
                                         price_median=('meter_price', 'median'))
plt.figure(figsize=(16, 8))
plt.step(data.index, data.price_mean, '-', where='post')
plt.step(data.index, data.price_median, '-', where='post')
plt.title('Динамика изменения цены квадратного метра в зависимости от расстояния до центра')
plt.xticks(data.index, data.index)
plt.xlabel('расстояние до центра (в километрах)')
plt.ylabel('усредненная цена одного квадратного метра')
plt.axvline(city_centre_radius_km, color='r', linestyle='--', linewidth=4)
plt.legend(['средняя цена', 'медиана цены', f'радиус центра: {city_centre_radius_km:.02f}'])
plt.show()
```



Как видим, что, начиная с отметки в 8 километров, цена за квадратный метр из максимального положения резко падает. Наша гипотеза о радиусе центральной зоны оказалась верна. Колебания цен внутри зоны объясняются тем что жилье в историческом центре иногда дешевеет из-за аварийного состояния домов и устаревших планировок квартир.

Интересно было бы взглянуть на расположение центральной зоны на карте. ([спойлер](#))

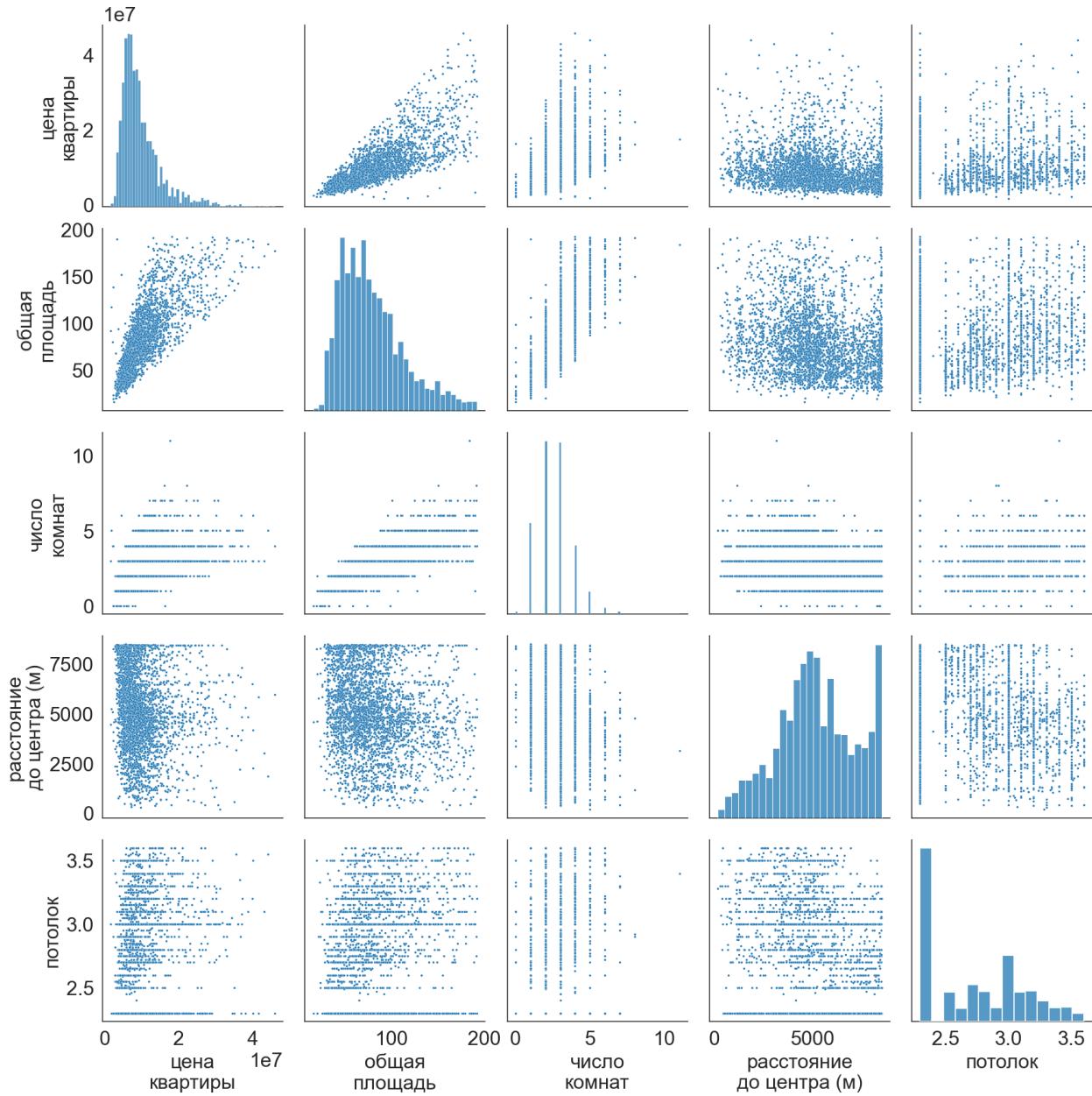
## Факторы влияния на квартиры в центре

Отберем квартиры в пределах центральной зоны Санкт-Петербурга:

```
In [191...]: ads_spb_centre = valid_ads[(valid_ads.locality_name == 'Санкт-Петербург') & (valid_ads.cityCenters_km <= city_centre_radius_km)]
```

Для выяснения, как зависит цена квартир в сердце города от площади, числа комнат, высоты потолков и удалённости от центра, построим парные диаграммы:

```
In [192...]: plot_pairof_with_desc(ads_spb_centre[['last_price', 'total_area', 'rooms', 'cityCenters_nearest_km', 'ceiling_height_m']])
```



По сравнению со всей совокупностью жилых помещений **цена квартир в сердце города стала меньше зависеть от высоты потолков и расстояния до центра**. Действительно, сложно уловить разницу между близко расположенными объектами. На их привлекательность влияют другие факторы.

Эти выводы подтверждаются и матрицей корреляций:

```
In [193...]: display_df_corr(ads_spb_centre[['last_price', 'total_area', 'rooms', 'cityCenters_nearest_km', 'ceiling_height_m']])
```

цена на момент снятия с публикации	
площадь квартиры в м <sup>2</sup>	0.776347
число комнат	0.482908
расстояние до центра города в метрах	-0.114122
высота потолков в метрах	0.157722

## Сравнение динамики в разных районах

Опишем функцию построения графика усредненных значений выбранного поля датасетов (резов набора данных о квартирах по различным критериям, объединенных в словарь:

```
In [194...]: def plot_df_field_by_date(df_dict, field_name, func, title, ylabel):
    """Функция построения графика усредненных значений выбранного поля датасетов
    (резов набора данных о квартирах по различным критериям, объединенных в словарь
    df_dict - словарь датасетов (ключ - подпись графика)
    field_name - имя поля для применения к нему функции func
    title - элемент заголовка всего рисунка
    ylabel - подпись вертикальной оси"""

    fig, ax = plt.subplots(figsize=(16, 8))
    for name, df in df_dict.items():
        data = df.groupby('year_month').agg(field_mean=(field_name, func))
        data.plot(ax=ax)

    plt.title(f'Динамика изменения {title} по месяцам')
    plt.xlabel('месяц и год')
    plt.ylabel(ylabel)
    plt.legend(df_dict.keys())
    plt.show()
```

Сравним три группы:

- центр Санкт-Петербурга;
- перефтерия Санкт-Петербурга;
- Ленинградская область.

```
In [195...]: ads_dic = \
{
    'центр Санкт-Петербурга':
        ads_spb_centre[ads_spb_centre.year_month < '2019.06.01'],

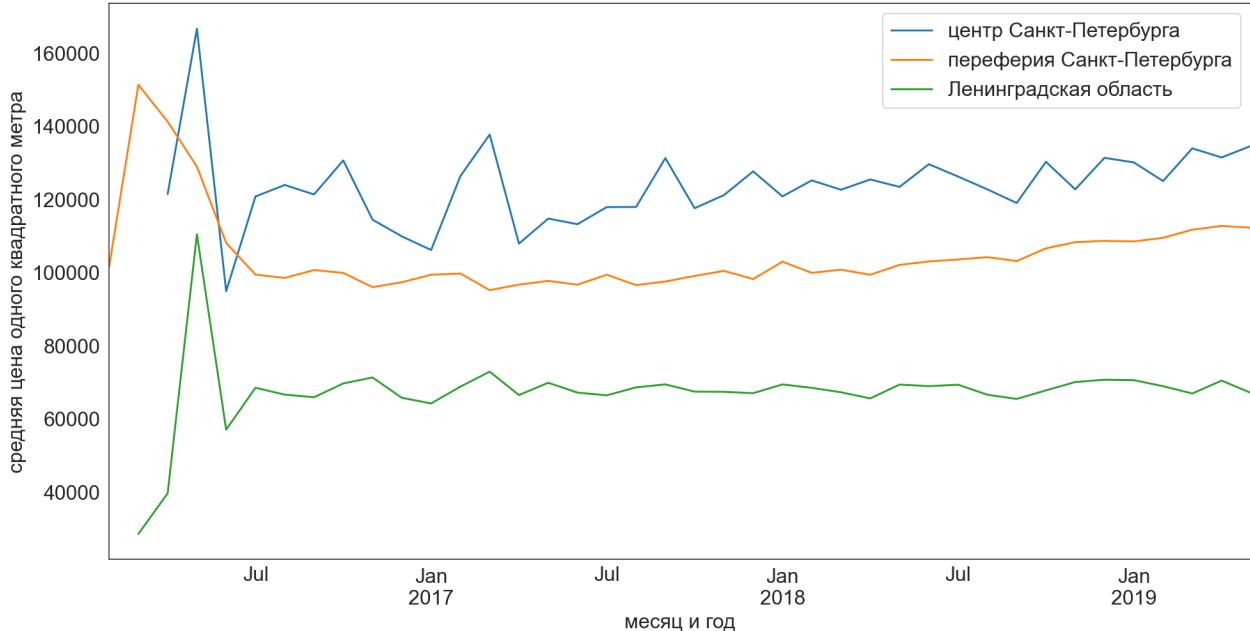
    'перефтерия Санкт-Петербурга':
        valid_ads[(valid_ads.locality_name == 'Санкт-Петербург') &
                   (valid_ads.cityCenters_km > city_centre_radius_km) &
                   (valid_ads.year_month < '2019.06.01')],

    'Ленинградская область':
        valid_ads[(valid_ads.locality_name != 'Санкт-Петербург') &
                   (valid_ads.year_month < '2019.06.01')]
}
```

### Динамика средней цены квадратного метра

```
In [196...]: plot_df_field_by_date(ads_dic, 'meter_price', 'mean',
                                'средней цены квадратного метра', 'средняя цена одного квадратного метра')
```

### Динамика изменения средней цены квадратного метра по месяцам

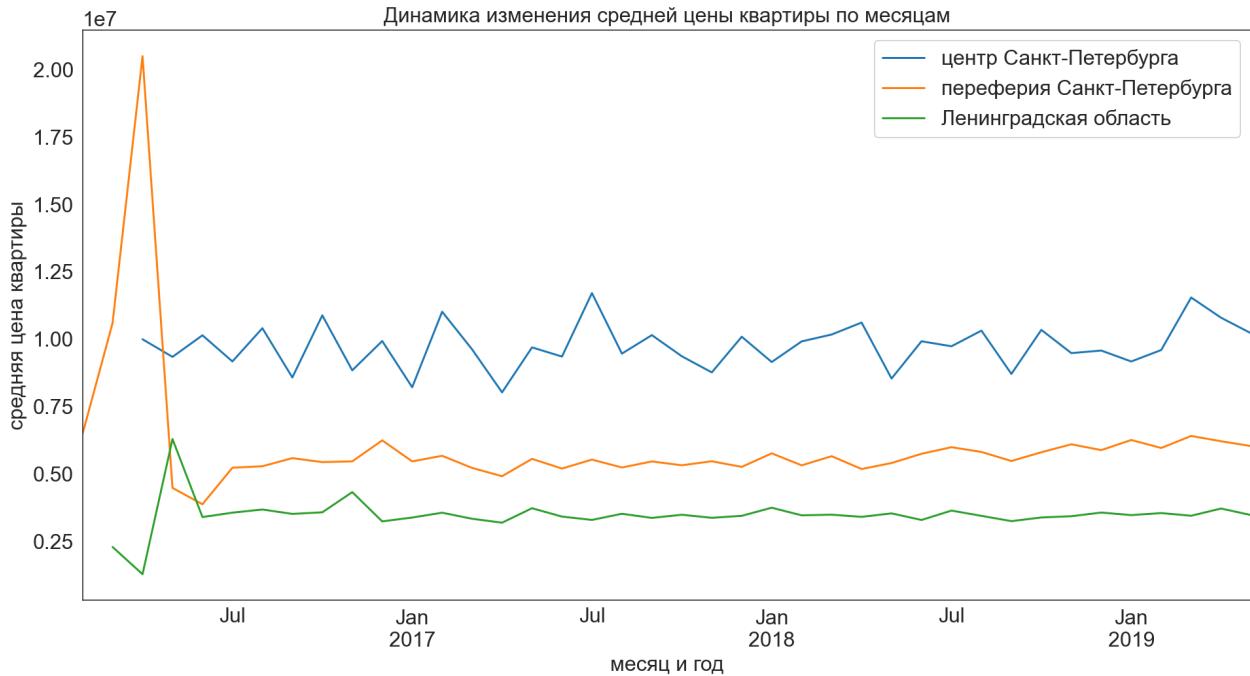


Квартиры в центре в среднем дороже остальных районов города и области.

Стоимость квадратного метра в центре подвержена более сильным колебаниям. В последние годы жилье дорожает во всех районах.

### Динамика средней цены квадратного метра

```
In [197...]: plot_df_field_by_date(ads_dic, 'last_price', 'mean',
                           'средней цены квартиры', 'средняя цена квартиры')
```

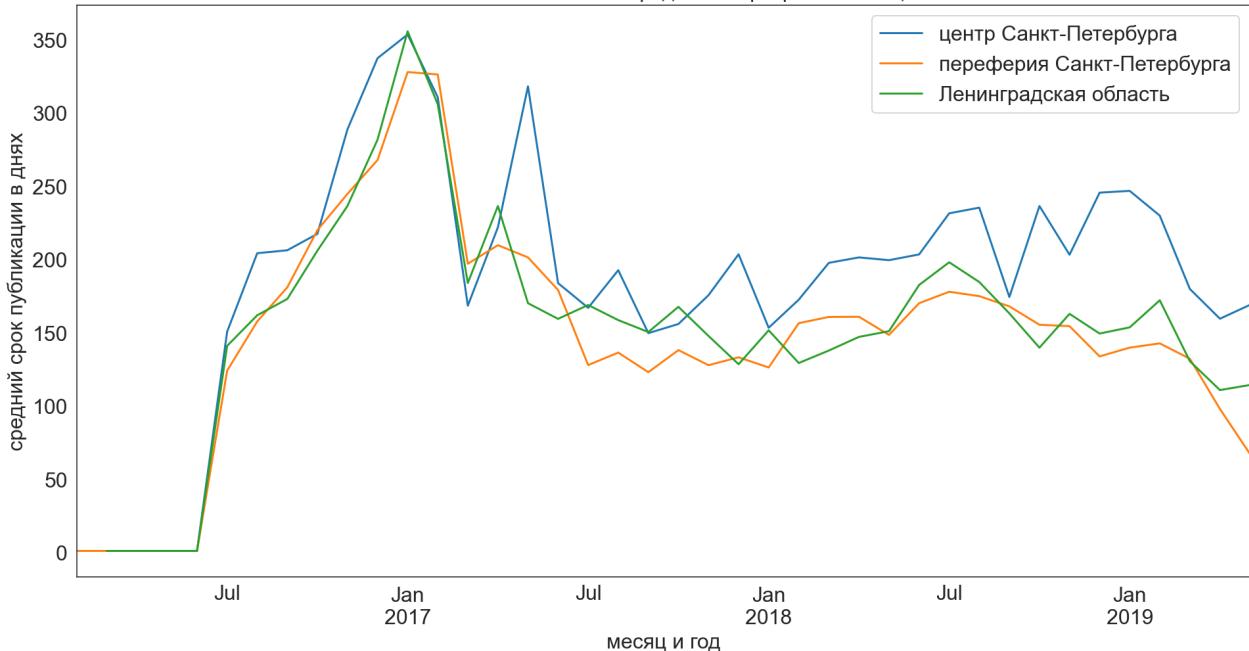


Колебания стоимости квартиры целиком в центре ещё заметнее.

### Динамика скорости продажи

```
In [198...]: plot_df_field_by_date(ads_dic, 'days_exposition', 'mean',
                           'среднего срока публикации\побывления на момент продажи квартиры', 'средни
```

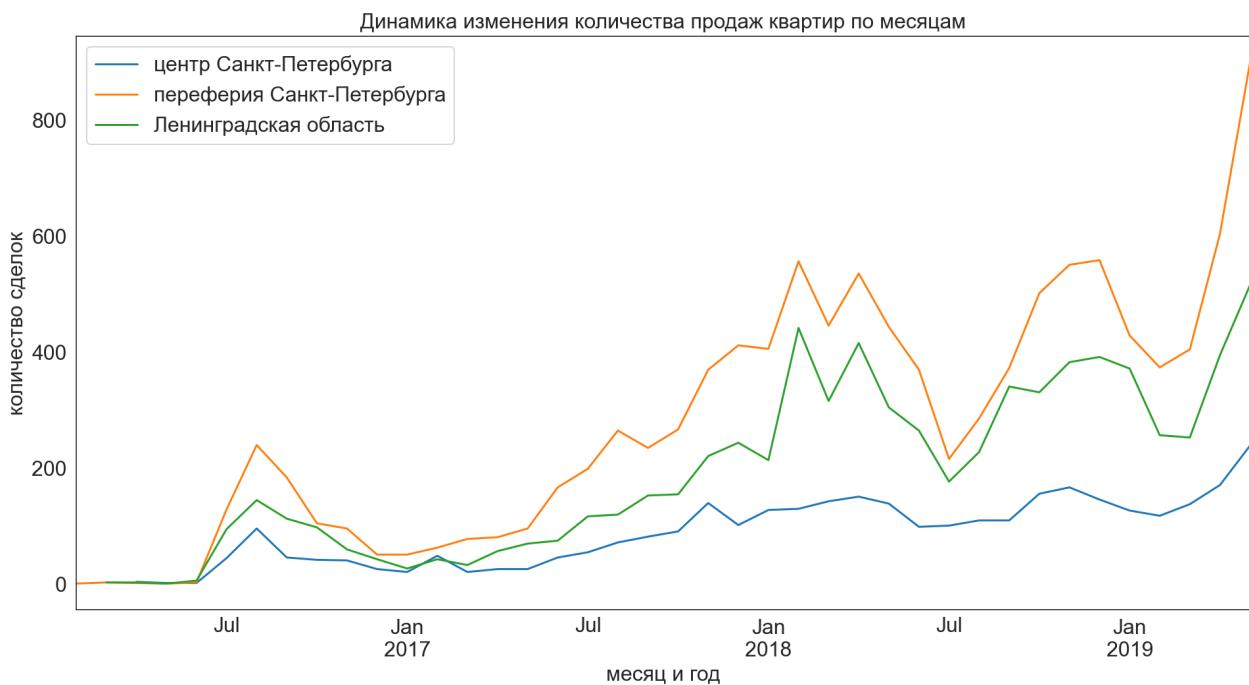
### Динамика изменения среднего срока публикации объявления на момент продажи квартиры по месяцам



В 2019 году наблюдаем тенденцию к сокращению сроков публикации объявлений - значит растет спрос на недвижимость.

### Динамика количества продаж

```
In [199]: plot_df_field_by_date(ads_dic, 'price_date', 'count',
                           'количество продаж квартир', 'количество сделок')
```



Сокращается срок публикации, значит увеличивается количество продаж. Спрос на рынке недвижимости в последнее время резко пошел вверх во всех районах. Больше всего сделок совершается не в центральных районах Санкт-Петербурга.

## Вывод по шагу 4

- В массиве данных имеются редкие, выбивающиеся по своему размеру значения. Гистограммы приобрели детали, когда мы отбросили слишком большие значения:
  - за 2014-2015 годы (объявлений [слишком мало](#));

- с общей площадью более 193 м<sup>2</sup> (они не попадают в выбранный [доверительный интервал](#));
- с площадью кухни больше 35 м<sup>2</sup> (они не попадают в выбранный [доверительный интервал](#));
- с потолками выше 3.6 м (такие объекты недвижимости [редки](#) и их не следует смешивать с обычными квартирами);
- с ценой 1 м<sup>2</sup> выше 260510 (они не попадают в выбранный [доверительный интервал](#)).
- При отбрасывании выбивающихся значений немного изменились средняя и медиана, они стали точнее характеризовать основную выборку.
- Гистограмма цены квадратного метра выглядит компактнее гистограммы общей стоимости квартиры. Исключение аномально редких значений цены квадратного метра улучшило качество гистограммы. Среднее арифметическое и медиана не сильно отстают друг от друга. Значит цена квадратного метра - хороший показатель для применения в анализе рынка.
- Выполнены все пункты [задания 4](#). На каждом шаге формулировались заключения, основное содержание которых можно перенести в [общий вывод](#).

## Шаг 5. Общий вывод

Исследование проведено на основе массива из 23699 объявлений о продаже квартир в Санкт-Петербурге и Ленинградской области. Из массива были удалены редкие выбывающиеся и ошибочные показатели. Их малая доля не повлияла на качество анализа.

Установлено, что чаще всего встречаются объявления о продаже квартир общей площадью 52 м<sup>2</sup> и кухней 9 м<sup>2</sup>. Типовая квартира имеет потолок высотой около 2.5 м, а у 99% квартир она не больше 3.6 м. В основном торгуются однокомнатные и двухкомнатные квартиры, квартир-студий на рынке совсем немного, реже встречаются только квартиры от шести и более комнат.

Разброс цен на квартиры достаточно большой. Среднее значение - 5.9 млн. Среднее значение цены квадратного метра (94-96 тыс.) позволяет лучше оценивать рынок, особенно в динамике. Санкт-Петербург - лидер топ-10 по цене квадратного метра (109231). Выборг замыкает десятку (57908).

На стоимость квартиры основное влияние в разной степени оказывают три фактора - площадь, число комнат, удалённость от центра.

Общая площадь квартиры линейно повышает верхнюю границу возможной цены недвижимости (зная площадь, мы сможем определить её максимально возможную цену, но практически всегда найдется недвижимость такого же размера, но дешевле).

Число комнат оказывает чуть меньшее влияние на цену квартиры и тоже больше влияет на верхнюю границу цены.

Удаление от центра снижает потолок цены: в центре есть и дорогие и дешевые квартиры, но на периферии дорогих квартир уже меньше, и меньше. Степень влияния расстояния меньше, чем у остальных показателей.

Имеет значение и этаж. Самые дешевые - одноэтажные дома. Квартиры на первом этаже в общем случае значительно дороже, но чуть уступают помещениям на последних этажах. Наибольшей ценой за квадратный метр могут похвастаться промежуточные этажи.

Высота потолка оказывает наименьшее влияние на стоимость квартиры. Никакой зависимости стоимости от дня недели публикации объявления нет.

Центральная зона Санкт-Петербурга (окружность радиусом 8 км) имеет некоторые особенности. Квартиры в центре в среднем дороже остальных районов города и области. Стоимость квадратного метра в центре подвержена более сильным колебаниям. Колебания стоимости квартиры целиком в центре ещё заметнее. По сравнению со всей совокупностью жилых помещений цена квартир в сердце города меньше зависит от высоты потолков и расстояния до центра.

С 2016 наблюдается тенденция к активизации рынка недвижимости, количество сделок растет. Пики активности на рынке в 2016-2018 гг. приходились на последние три месяца года - предновогодний ажиотаж. Больше всего сделок совершается с квартирами Санкт-Петербурга за пределами его центральной зоны.

В последние годы жилье дорожает во всех районах. Несмотря на это спрос растет. В 2019 году наблюдается тенденция к сокращению сроков публикации объявлений. Половина объявлений реализуется за 74 дня. Если удается реализовать квартиру за два месяца (быстрее, чем большая часть других вариантов), то можно считать это быстрой сделкой.

## Чек-лист готовности проекта

- [x] открыт файл
- [x] файлы изучены (выведены первые строки, метод `info()`)
- [x] определены пропущенные значения ([пример 1](#), [пример 2](#), [пример 3](#))
- [x] заполнены пропущенные значения ([пример 1](#), [пример 2](#), [пример 3](#))
- [x] есть пояснение, какие пропущенные значения обнаружены ([пример 1](#), [пример 2](#))
- [x] изменены типы данных
- [x] есть **пояснение**, в каких столбцах изменены типы и почему
- [x] посчитано и добавлено в таблицу: [цена квадратного метра](#)
- [x] посчитано и добавлено в таблицу: [день недели, месяц и год](#) публикации объявления
- [x] посчитано и добавлено в таблицу: [этаж квартиры](#); варианты — первый, последний, другой
- [x] посчитано и добавлено в таблицу: [соотношение жилой и общей площади](#), а также [отношение площади кухни к общей](#)
- [x] изучены следующие параметры: [площадь, цена, число комнат, высота потолков](#)
- [x] построены [гистограммы](#) для каждого параметра
- [x] выполнено задание: "Изучите время продажи квартиры. Постройте [гистограмму](#). Посчитайте среднее и медиану. Опишите, сколько обычно занимает продажа. Когда можно считать, что продажи прошли очень быстро, а когда необычно долго?"
- [x] выполнено задание: "Уберите редкие и выбивающиеся значения. Опишите, какие особенности обнаружили."
- [x] выполнено задание: "Какие факторы больше всего влияют на стоимость квартиры? Изучите, зависит ли цена от квадратного метра, числа комнат, этажа (первого или последнего), удалённости от центра. Также изучите зависимость от даты размещения: дня недели, месяца и года. "Выберите [10 населённых пунктов](#) с наибольшим числом объявлений. Посчитайте среднюю цену квадратного метра в этих населённых пунктах. Выделите населённые пункты с самой высокой и низкой стоимостью жилья. Эти данные можно найти по имени в столбце '`locality_name`'. "

- [x] выполнено задание: "Изучите предложения квартир: для каждой квартиры есть информация о расстоянии до центра. [Выделите квартиры в Санкт-Петербурге](#) ('*locality\_name*'). Ваша задача — выяснить, какая область входит в центр. Создайте [столбец с расстоянием до центра в километрах](#): округлите до целых значений. После этого посчитайте среднюю цену для каждого километра. Постройте график: он должен показывать, как цена зависит от удалённости от центра. Определите границу, где график сильно меняется — это и будет центральная зона. "
- [x] выполнено задание: "[Выделите сегмент квартир в центре](#). Проанализируйте эту территорию и изучите следующие параметры: площадь, цена, число комнат, высота потолков. Также выделите факторы, которые влияют на стоимость квартиры (число комнат, этаж, удалённость от центра, дата размещения объявления). Сделайте выводы. Отличаются ли они от общих выводов по всему городу?"
- [x] в каждом этапе есть выводы ([вывод 1](#), [вывод 2](#), [вывод 3](#), [вывод 4](#))
- [x] есть [общий вывод](#)

## Таблица контроля версий

Версия	Дата	Описание
1.0	07.11.2020	Направлена на первичную проверку
1.1	08.11.2020	Для исправления ошибок, возникших из-за разных версий библиотек, установленных у разработчика и ревьюера, в начало ноутбука добавлены <a href="#">команды обновления</a> библиотеки Pandas

[К оглавлению](#)

## Бонус

### Визуальная проверка правильности определения центра города

Мы вычислили радиус центральной зоны Санкт-Петербурга. Хотелось бы убедиться, что его значение адекватно реальности. Для проверки попробуем отразить окружность на карте с помощью библиотеки `folium`.

#### Установка библиотеки Folium

```
In [200...]: !pip install folium
Collecting folium
  Using cached folium-0.11.0-py2.py3-none-any.whl (93 kB)
Requirement already satisfied: jinja2>=2.9 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from folium) (2.11.2)
Requirement already satisfied: numpy in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from folium) (1.19.2)
Requirement already satisfied: requests in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from folium) (2.25.0)
Collecting branca>=0.3.0
  Using cached branca-0.4.1-py3-none-any.whl (24 kB)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from jinja2>=2.9->folium) (1.1.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from requests->folium) (1.25.11)
Requirement already satisfied: idna<3,>=2.5 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from requests->folium) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from requests->folium) (2020.6.20)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\users\bomba\anaconda3\envs\python37\lib\site-packages (from requests->folium) (3.0.4)
```

```
Installing collected packages: branca, folium
Successfully installed branca-0.4.1 folium-0.11.0
```

## Подключение библиотеки Folium¶

```
In [201...]: import folium
```

```
In [202...]: lib_versions([folium])
```

Версия folium - 0.11.0

## Настройка карты

Зададим географические координаты Санкт-Петербурга:

```
In [203...]: city_centre_coords = (59.938951, 30.315635)
```

Создадим карту OpenStreetMap:

```
In [204...]: m = folium.Map(location = city_centre_coords, zoom_start = 11)
```

```
In [205...]: folium.TileLayer('openstreetmap').add_to(m);
```

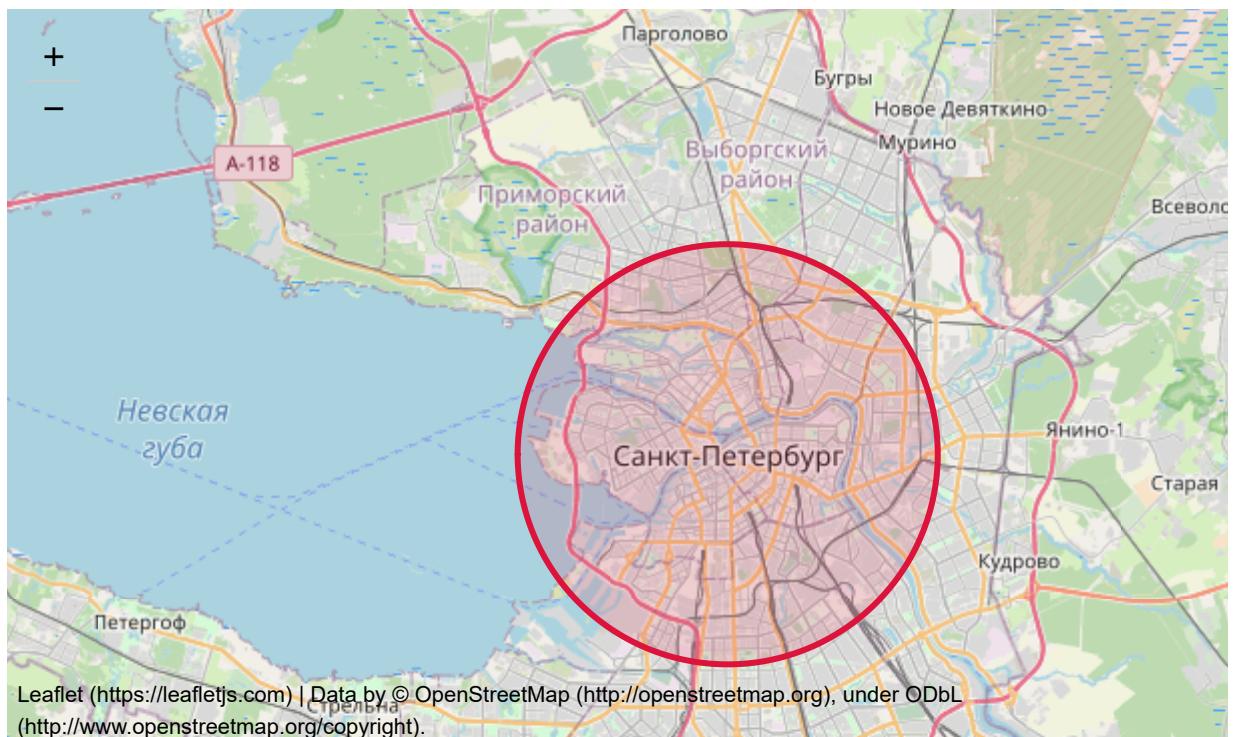
Нанесем на карту окружность центральной части города:

```
In [206...]: folium.Circle(
    location=city_centre_coords,
    radius=city_centre_radius_km * 1000,
    color='crimson',
    fill=True, fill_opacity=0.15
).add_to(m);
```

## Карта центра города

```
In [208...]: m
```

```
Out[208...]:
```



Найденный нами центр города вполне укладывается в общие представления об этом понятии.

После просмотра бонуса самое время пересмотреть [чек-лист готовности проекта](#). Спасибо за потраченное на проверку время!

