МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра анализа данных и искусственного интеллекта

Допустить к защите
заведующий кафедрой
д-р тех. наук, доцент
_____ А.В. Коваленко
_____ 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

RESTFUL СЕРВИС ДЛЯ ПРЕДСКАЗАНИЯ СТОИМОСТИ КВАРТИР С ИСПОЛЬЗОВАНИЕМ МАШИННОГО ОБУЧЕНИЯ

Работу выполнил(а) _____ П.Ю. Захаров

Направление подготовки 09.03.03 Прикладная информатика в экономике

Направленность (профиль) Прикладная информатика

Научный руководитель
доцент _____ Г.А. Кесиян

Нормоконтролер
канд. физ.-мат. наук, доцент _____ Г.В. Калайдина

Краснодар
2022

РЕФЕРАТ

Выпускная квалификационная работа 40 с., 41 рис.,10 источников.

МАШИННОЕ ОБУЧЕНИЕ, ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ, ИНС, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, ПАРСИНГ, ЗАДАЧА РЕГРЕССИИ, РАЗВЁРТЫВАНИЕ НЕЙРОННОЙ СЕТИ, РАЗРАБОТКА API, RESTFUL СЕРВИС, РАЗРАБОТКА САЙТА, ТИПИЗАЦИЯ

Предметом исследования является рынок вторичного жилья, а также создание инструментов для получения, предоставления и предсказания (дополнения) информации по нему.

Целью данной работы является разработка групп связанных друг с другом приложений, использующих реальные предложения для оценки рыночной стоимости той или иной квартиры.

В итоге, были реализованы программные модули для получения и анализа данных рынка, а также сервис для обработки и представления этих данных. Для демонстрации работы сервиса дополнительно разработан клиентский сайт.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Вторичный рынок недвижимости – это совокупность участников и договоров между ними по покупке, продаже, аренде и т. д. объектов вторичной недвижимости. Дом или квартира являются вторичным жильём, если право собственности на них оформляется более одного раза [1].

Вторичное жилье пользуется большим спросом в России. По данным Росреестра, в декабре 2020 года впервые превышен показатель в 20000 продаж вторичного жилья за месяц в Москве [2]. Количество сделок на вторичном рынке превышает количество сделок на первичном за этот же период в Москве, и это при общем росте спроса на недвижимость за последний год [3].

Это актуально и для других городов России. По данным Росриэлт для Краснодара, количество объявлений о продаже квартир на вторичном рынке почти в 5 раз превышает количество объявлений для новостроек [4].

Одним из самых важных критериев квартиры на рынке является её стоимость. Для покупателя важно знать рыночную стоимость рассматриваемого жилья, чтобы из множества предложений купить с наибольшей выгодой. Продавец же может ставить разную стоимость для своей квартиры относительно рыночной, в зависимости от того, как быстро он хочет её продать. Но как покупателю и продавцу узнать рыночную стоимость той или иной квартиры?

Обычно за выяснением этого обращаются к оценщикам и риэлторам. Стоимость оценивается по характеристикам жилья, таким как: площадь, количество комнат, этаж, ремонт, расположение и т.д. Достаточно знать схожие предложения, чтобы сделать вердикт по оцениваемой квартире. Можно убедиться, что перед нами стоит задача регрессии – предсказание непрерывной величины, цены квартиры в нашем случае, на основе выборки объектов со схожими характеристиками.

1 Постановка задачи

В работе поставлена задача создания инструмента для предсказания рыночной стоимости квартиры. Эта задача делится на следующие подзадачи:

– изучение сайта с предложениями по продаже квартир;
– разработка средства для извлечения данных с сайта (парсер);
– анализ полученных данных;
– подготовка данных для передачи в нейронную сеть;
– первичное обучение сети, поиск оптимальных гипер-параметров;
– обучение нейронной сети и оценка ошибки;
– разработка RESTful-сервиса для оценки стоимости квартир;
– демонстрация работы сервиса через сайт.

Изобразим схематично порядок решения общей задачи на рисунке 1.1.

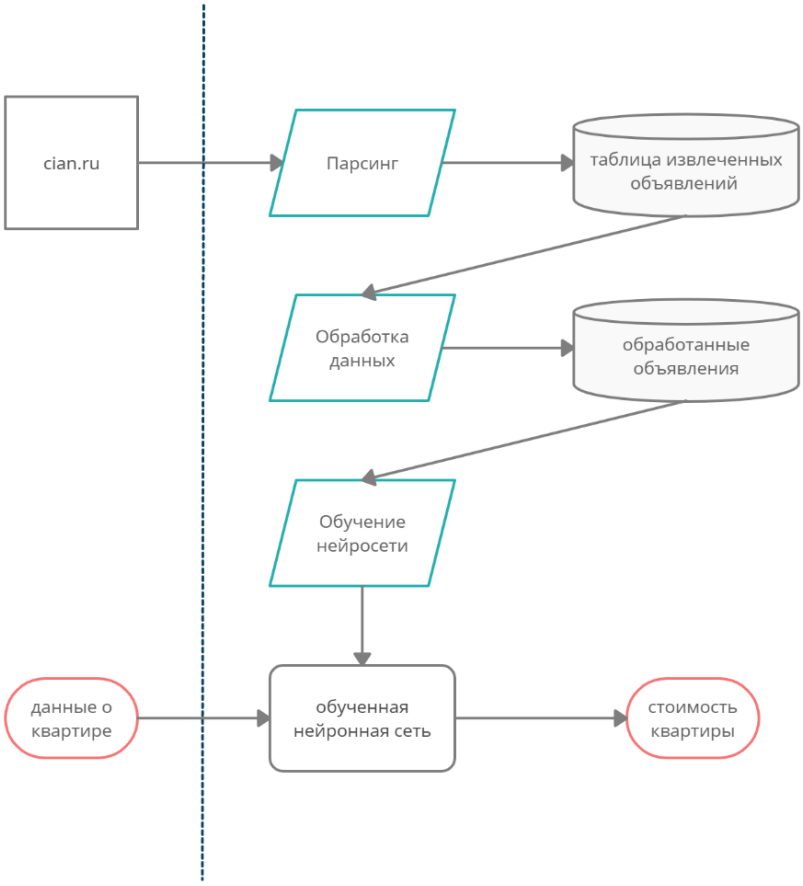


Рисунок 1.1 – Схема решения общей задачи

2 Парсинг сайта

2.1 Изучение сайта с объявлениями

Как говорилось ранее, предсказание стоимости квартиры можно свести к задаче регрессии, которая относится к методам обучения с учителем. Множеству наборов характеристик квартир соответствует множество цен. Есть некоторая зависимость между характеристиками и ценой, но она неизвестна. Нам ещё предстоит выяснить её, но для начала необходимо получить данные об объявлениях продаж.

В интернете можно найти довольно много онлайн-площадок по продаже квартир. Циан[5] – один самых популярных сайтов с обширной базой предложений. Воспользуемся им как источником данных.

Процесс автоматического сбора данных с сайтов и их приведением в нужный формат называют парсингом или скрайпингом, а программу, которая это делает – парсером.

В нашем случае разработать парсер для одного сайта будет рациональнее, чем для нескольких. Помимо проблем с приведением данных к общему виду, сбор данных из нескольких источников может привести к дублированию одних и тех же объявлений и, как следствие, просачиванию объектов тестирующей выборки в обучающую, что приведёт к искажению оценки нейронной сети.

Парсинг основывается на анализе структуры веб-документа программистом и выделении нужных блоков. Перейдём на сайт
https://krasnodar.cian.ru/kupit-kvartiru-vtorichka/
и изучим его. Как видно на рисунке 2.1, перед нами список объявлений по продаже квартир с краткой информацией по каждой из них. Будем брать информацию с веб-страниц самих предложений, поэтому просто отметим блоки с гиперссылками на них.

Основная часть страницы с объявлением изображена на рисунке 2.2, характеристики квартиры и дома на рисунке 2.3.

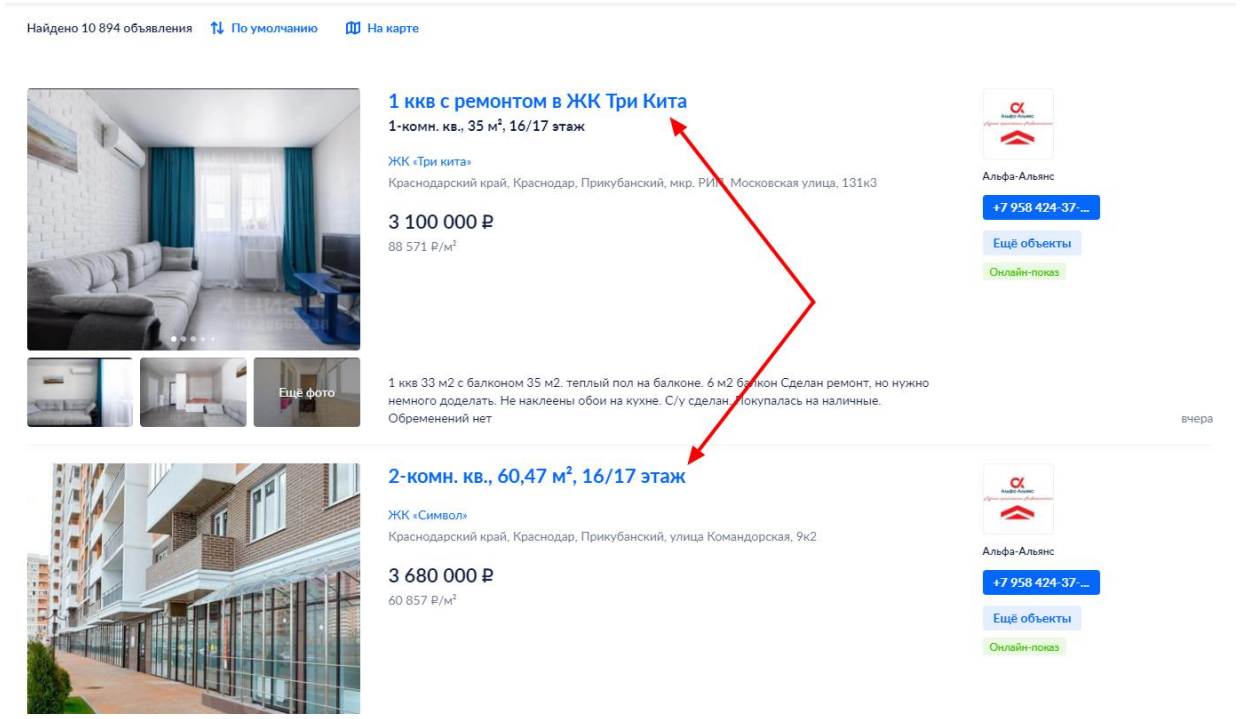


Рисунок 2.1 – Список объявлений

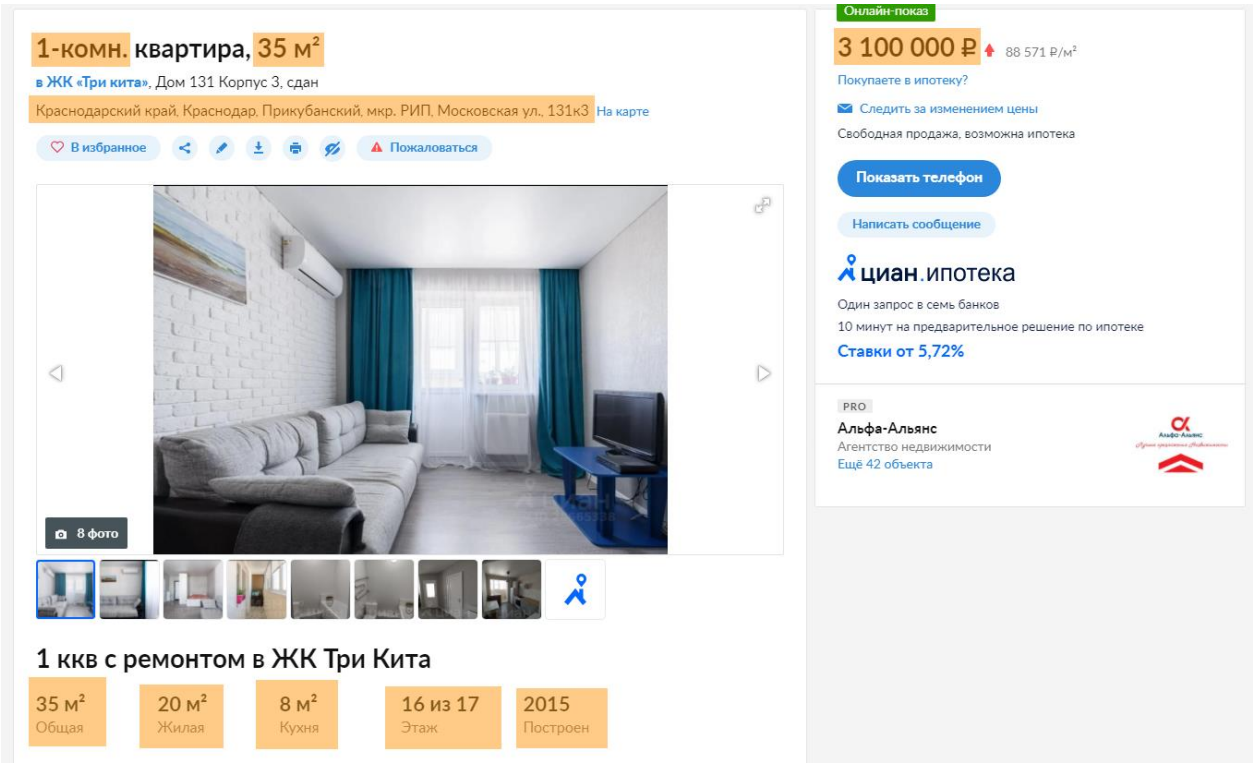


Рисунок 2.2 – Основная часть страницы объявления

Нужные блоки выделены прямоугольниками. Самый важный – блок с ценой.

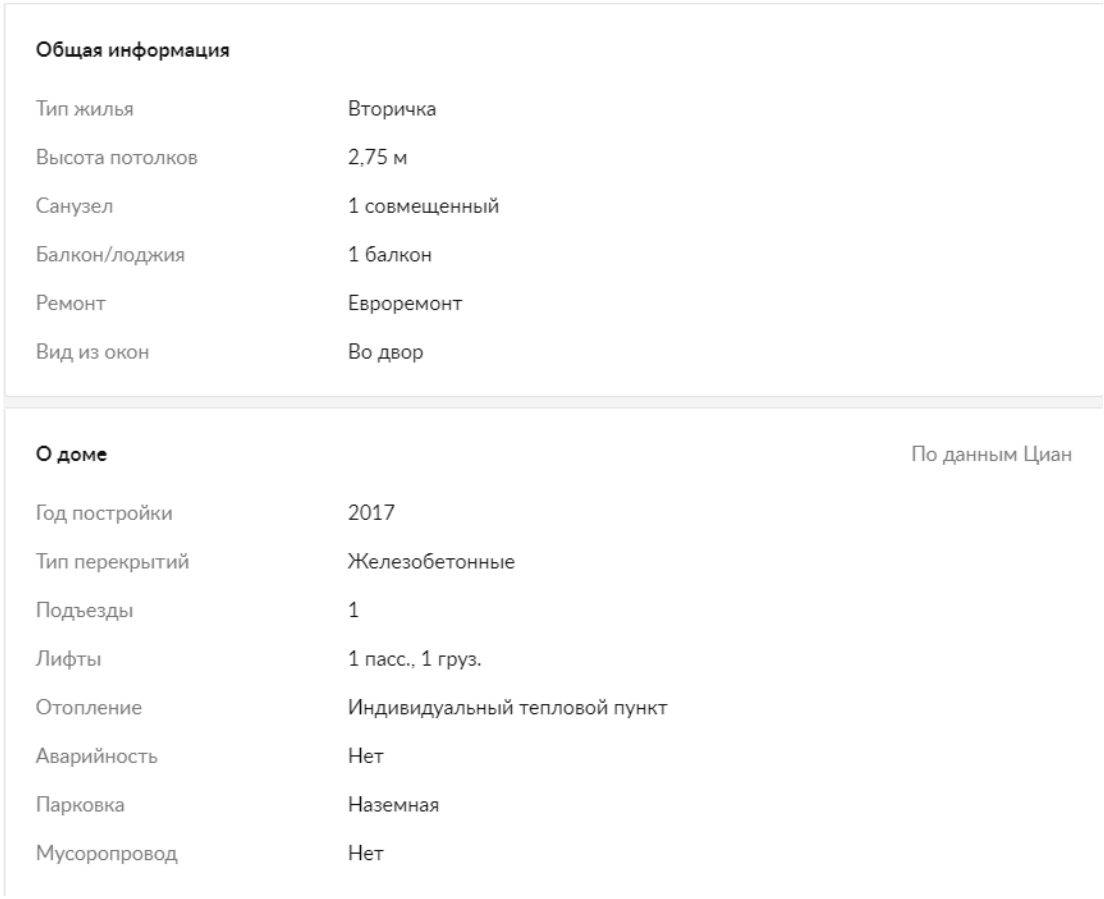


Рисунок 2.3 – Характеристики квартиры и дома

В качестве потенциальных факторов подходят выделенные на рисунке 2.2 блоки и большая часть характеристик на рисунке 2.3. Теперь определимся с порядком обхода сайта.

Начать стоит с первой страницы списка объявлений. Извлекаем ссылки на квартиры, после чего парсим каждую из них. Переходим на следующую страницу при помощи пагинатора и URL поменялся на *https://krasnodar.cian.ru/cat.php?deal_type=sale&engine_version=2&object_type%5B0%5D=1&offer_type=flat&p=2&region=4820*

Параметр *p* отвечает за номер текущей страницы, алгоритм обхода страниц будет итерироваться по нему.

На сайте есть одно существенное ограничение – максимальное количество страниц равно 54. Если указать *p* больше, то происходит переход на первую страницу. На каждой странице выводится не более 28 объявлений. Получается, что наибольшее количество объявлений, которое можно получить, проходя только по одному параметру *p*: 54 * 28 = 1512 объявлений.

Это значение можно увеличить, если указать дополнительные фильтры. Фильтр должен быть таким, чтобы объявление из одной группы фильтра не попадало в другие. Один из таких фильтров – «Комнатность». На рисунке 2.4 видно, что квартира может иметь от одной до шести комнат или являться студией или иметь свободную планировку.

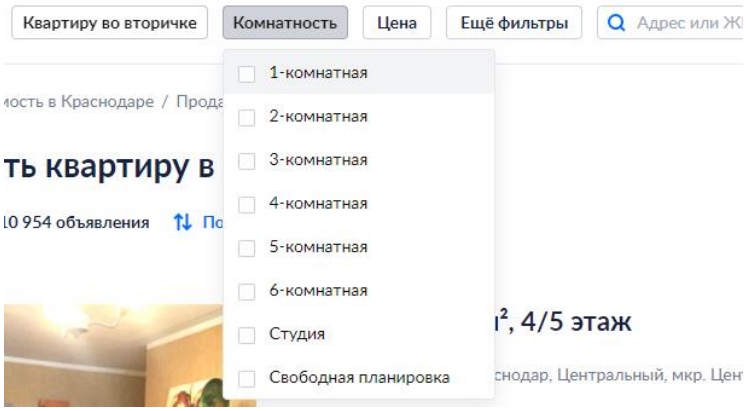


Рисунок 2.4 – Фильтр «Комнатность»

В URL добавляется параметр *room<n>=1*, где <*n*> = 1, 2, …, 6, 9. Значениям от 1 до 6 соответствует количество комнат, а 9 обозначается студия. Квартиры со свободной планировкой в расчёт не берём. Забегая наперёд, итерация страниц по типу комнаты *room<n>* и номеру страницы *p* позволяет извлекать от 5000 объявлений (учитывая вычет дублей).

2.2 Разработка парсера

После того, как мы изучили структуру сайта, выделили места откуда будет извлекаться информация и описали примерный алгоритм обхода, можно приступить к написанию парсера.

Код парсера написан на языке программирования Python с использованием внешних библиотек Requests и BeautifulSoup4. Requests – это удобный HTTP-клиент, а BeautifulSoup4 – высокоуровневая библиотека для парсинга HTML документов и предоставлению их структуры в удобной форме.

Подключение этих и других библиотек изображено на рисунке 2.5.

```
1  import csv
2  import datetime
3  import re
4  from itertools import product
5
6  import requests
7  from bs4 import BeautifulSoup
8  from requests.exceptions import HTTPError
```

Рисунок 2.5 – Импорт модулей

На рисунке 2.6 определим константы и опишем их.

```
10 CITY = {
11     'name': 'krasnodar',
12     'cian_region': 4820
13 }
14 MAX_PAGE = 55
15 BASE_URL = 'https://{city_name}.cian.ru/cat.php?region={cian_region}&{tale}'.format(
16     city_name=CITY['name'],
17     cian_region=CITY['cian_region'],
18     tale='deal_type=sale&engine_version=2&object_type%5B0%5D=1&offer_type=flat&p={page}&room{room_type}=1'
19 )
20 ROOM_TYPES = {
21     1: (1, '1-комнатная'),
22     2: (2, '2-комнатная'),
23     3: (3, '3-комнатная'),
24     4: (4, '4-комнатная'),
25     5: (5, '5-комнатная'),
26     6: (6, '6-комнатная'),
27     9: (0, 'студия')
28 }
```

Рисунок 2.6 – Константы парсера

Константа *CITY* задаёт словарь с названием населённого пункта и номером региона в базе Циана, *MAX_PAGE* – предел для итерации по страницам, *BASE_URL* – базовый URL с подставленными значениями параметров для получения списка объявлений по продаже квартир на вторичном рынке. *ROOM_TYPES* нужен для преобразования значения «Комнатность» на Циан с нашим представлением.

При желании, задачу определения рыночной цены для квартир в Краснодаре можно легко переделать под любой другой город России, просто поменяв значения словаря *CITY* на нужные. Например, для Сочи это параметры *name = 'sochi', cian_region = 4998*.

На рисунке 2.7 создаётся сессия. Такие заголовки запроса нужны для того, чтобы обойти защиту сайта от ботов, симулируя обычный браузер вместо HTTP-клиента Requests.

```
session = requests.Session()
session.headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) '
                  'Chrome/90.0.4430.93 Safari/537.36 OPR/76.0.4017.123',
    'Accept-Language': 'ru'
}
```

Рисунок 2.7 – Создание сессии и определение заголовков

Функция *scrape_page* на рисунке 2.8 принимает на вход URL страницы и возвращает список URL объявлений.

```
def scrape_page(url):
    response = session.get(url)
    response.raise_for_status()
    if response.status_code // 100 == 3:
        raise NoMorePagesException()

    page_soup = BeautifulSoup(response.text, 'lxml')
    offers = page_soup.select('div[data-name="LinkArea"] a[href*="/flat/"]')
    offers_links = [offer['href'] for offer in offers]
    return offers_links
```

Рисунок 2.8 – Код функции *scrape_page*

Код функций *str_square_to_float* и *scrape_offer* изображен на рисунке 2.9.

```
42  def str_square_to_float(str_square):
43      """
44      Функция для перевода строки в определенном формате в число с плавающей точкой
45      ex.: '1-комн. квартира, 44,22 м²' -> 44.22
46      """
47      return float(re.search(r'(\d+,?\d*)\sм', str_square)[1].replace(',', '.'))
48
49
50  def scrape_offer(url):
51      response = session.get(url)
52      response.raise_for_status()
53      offer_soup = BeautifulSoup(response.text, 'lxml')
54      return offer_soup
55
```

Рисунок 2.9 – Код функций *str_square_to_float* и *scrape_offer*

Первая функция использует регулярное выражение для выделения площади из строки и приводит её к типу float. Функция *scrape_offer* получает HTML страницы объявления по данному URL и возвращает объект BeautifulSoup для дальнейшего парсинга.

Перед парсингом создадим файл в формате таблиц CSV и запишем туда названия столбцов. Для удобства, добавим в имя файла временную метку, как изображено на рисунке 2.10.

Пройдёмся по функции *init_parsing*.

Как видно на рисунке 2.11, она принимает на вход два атрибута – *file_name* и *yield_rows*. Первый указывается, если нужно сохранить результат работы парсера в файл с переданным именем. Если значение второго аргумента *yield_rows* равен True, тогда функция работает в режиме генератора последовательности. Это позволяет не нагружать модуль парсера логикой, связанной с базами данных. Можно совмещать оба режима.

```
237 def main():
238     current_time = datetime.datetime.now()
239     file_stamp = current_time.strftime('%Y%m%d_%H%M')
240     file_name = 'cian_flats_{}.csv'.format(file_stamp)
241     with open(file_name, 'w', newline='', encoding='utf-8') as flats_file:
242         flats_writer = csv.writer(flats_file, delimiter=';', quotechar='|', quoting=csv.QUOTE_MINIMAL)
243         flats_writer.writerow([
244             'ID', 'Link', 'Price', 'Rooms', 'Views', 'Square', 'Live Square', 'Kitchen', 'Floor', 'Total Floors',
245             'Residental Complex', 'District', 'Flat number', 'Street', 'Microdistrict', 'Microistrict 2', 'Description',
246             'Flat Type', 'Toilet', 'Balcony', 'Repair', 'Window View', 'Ceiling', 'House Type', 'House Year', 'Lifts',
247             'Parking', 'Gas'
248         ])
249     init_parsing(file_name)
250
251
252 if __name__ == '__main__':
253     main()
254
```

Рисунок 2.10 – Создание файла для сохранения результатов парсинга

```
145 def init_parsing(file_name=None, yield_rows=False):
146     timer_start = datetime.datetime.now()
147     flats_hashes = set()
148     for room_type in ROOM_TYPES.keys():
149         for page_number in range(1, MAX_PAGE):
150             page_timer_start = datetime.datetime.now()
151             offers_data = {}
152             try:
153                 page_offers_links = scrape_page(BASE_URL.format(page=page_number, room_type=room_type))
154                 print('Got offers on page {}, room_type {}'.format(page_number, room_type))
155             except NoMorePagesException:
156                 print('[NMP]: Skip page {} for room_type {}'.format(page_number, room_type))
157                 break
```

Рисунок 2.11 – Функция *init_parsing*

Также на рисунке 2.11 видно, как происходит обход сайта. Сначала *room_type* принимает некое значение, например 1 – тип «комнатности» по Циану (соответствует количеству комнат кроме значения для девяти, обозначающее студию). Затем для данного типа проходятся номера страниц *page_number*, от одного до *MAX_PAGE* – 1. Когда все номера страниц пройдены, получаем следующее значение *room*_type и так далее. Это позволяет реализовать механизм пропуска ненужных запросов, если при достижении какого-то номера страницы для текущего количества комнат происходит перенаправление на первую страницу (строки с 155 по 157).

Собственно, в строке 153 происходит изъятие списка ссылок на объявления. В базовый URL подставляются текущие значения номера страницы и «комнатности». Ниже, каждый из полученных URL объявлений,

передаётся в *scrape_offer* и добавляется в словарь *offers_data* вместе со своим объектом BeautifulSoup.

На рисунке 2.12 изображено получение интересующей нас информации об объявлениях через функцию *parse_offer* и механизм пропуска дублей.

```
192         for offer_id, offer_data in offers_data.items():
193             try:
194                 flat_info = parse_offer(offer_data['soup'])
195             except Exception as e:
196                 print('[DNG] Exception during parsing. ', e)
197                 continue
198 
199             flat_hash = '{}{}{}{}{}{}'.format(
200                 flat_info['total_square'],
201                 offer_data['room_type'],
202                 flat_info['floor'],
203                 flat_info['district'],
204                 flat_info['microdistrict_main'],
205                 flat_info['street'],
206                 flat_info['flat_number']
207             )
208             if flat_hash in flats_hashes:
209                 print('[DOUBLE] Skip offer saving')
210                 continue
211 
212             flats_hashes.add(flat_hash)
```

Рисунок 2.12 – Получение информации из объявления и проверка уникальности

Как было замечено ранее, желательно не иметь дублей объектов из-за возможности утечки информации из тестирующей выборки в обучающую. Чтобы отфильтровать дубли, после извлечения информации об объявлении создаётся строка *flat_hash*, позволяющая отличить одни объявления от других по основным характеристикам (строка 199). Если объявление с такими характеристиками уже встречалось, пропускается сохранение информации для дубля и цикл переходит на следующую итерацию (строки с 208 по 210). Иначе, строка *flat_hash* добавляется в множество *flats_hashes*. Стоит заметить, что множества в Python могут хранить только хешируемые

объекты. При добавлении строки в множество вычисляется её хеш. Это позволяет производить поиск по множеству за константное время O(1).

Перед тем как описать последние строки рассматриваемой функции *init_parsing*, обратимся к функции *parse_offer*. На вход *parse_offer* принимает объект BeautifulSoup некоторого объявления. Из этого объекта извлекаются интересующая нас информация, выделенная на рисунке 2.2. Поиск блоков с такой информацией осуществляется при помощи CSS-селекторов. На рисунке 2.13 есть пример извлечения цены продажи квартиры, количества просмотров и общей площади квартиры.

```
56  def parse_offer(offer_soup):
57      price: str = offer_soup.select_one('span[itemprop="price"]')['content']
58      price_value = int(price.replace(' ', '')[:-1])  # ex.: '2 700 000 ₽' -> 2700000
59      views = offer_soup.select_one('div[data-name="OfferStats"] a')
60      views_total_value = int(views.get_text().split()[0])
61
62      flat_desc_dict = {}
63      flat_description = offer_soup.select('div[data-name="Description"] div[itemscope=""] > div')
64      for flat_desc_block in [desc_block.get_text(separator='::') for desc_block in flat_description]:
65          desc_value, desc_key = flat_desc_block.split('::')
66          flat_desc_dict[desc_key] = desc_value
67
68      flat_square = flat_desc_dict.get('Общая')
69      flat_square_value = str_square_to_float(flat_square) if flat_square else ''
```

Рисунок 2.13 – Получение информации при помощи CSS-селекторов

Выбор подходящих селекторов был сделан в предыдущем разделе.

Пропустим описание кода всей функции и сразу перейдём к её результату на рисунке 2.14. Результат парсинга пишем аддитивно в результирующую таблицу на рисунке 2.15. Парсер готов. Запустим его и посмотрим на результат на рисунке 2.16. Таким образом, за 9097 секунд (примерно два с половиной часа) была извлечена информация о 4981 объявлении.

```
    return {
        'price': price_value,
        'views': views_total_value,
        'total_square': flat_square_value,
        'live_square': live_square_value,
        'kitchen_square': kitchen_square_value,
        'floor': flat_floor, 'total_floors': total_floors,
        'residental_complex': residental_complex_value,
        'district': district_value,
        'flat_number': flat_number,
        'street': street,
        'microdistrict_main': microdistrict_main,
        'microdistrict_local': microdistrict_local,
        'description': description_value,
        'flat_type': dict_main_info.get('Тип жилья', ''),
        'toilet': dict_main_info.get('Санузел', ''),
        'balcony': dict_main_info.get('Балкон/лоджия', ''),
        'repair_status': dict_main_info.get('Ремонт', ''),
        'window_view': dict_main_info.get('Вид из окон', ''),
        'ceiling': dict_main_info.get('Высота потолков', ''),
        'house_type': dict_cian_info.get('Тип дома', ''),
        'house_year': flat_desc_dict.get('Построен', ''),
        'lifts': dict_cian_info.get('Лифты', ''),
        'parking': dict_cian_info.get('Парковка', ''),
        'gas': dict_cian_info.get('Газоснабжение', '')
    }
```

Рисунок 2.14 – Переменные получаемые при парсинге

```
            row = [
                offer_id, offer_data['link'], flat_info['price'], ROOM_TYPES[offer_data['room_type']][0],
                flat_info['views'], flat_info['total_square'], flat_info['live_square'],
                flat_info['kitchen_square'], flat_info['floor'], flat_info['total_floors'],
                flat_info['residental_complex'], flat_info['district'], flat_info['flat_number'],
                flat_info['street'], flat_info['microdistrict_main'], flat_info['microdistrict_local'],
                flat_info['description'].replace(';', ' ').replace('\n', ' '), flat_info['flat_type'],
                flat_info['toilet'], flat_info['balcony'], flat_info['repair_status'], flat_info['window_view'],
                flat_info['ceiling'], flat_info['house_type'], flat_info['house_year'], flat_info['lifts'],
                flat_info['parking'], flat_info['gas']
            ]
            if file_name:
                with open(file_name, 'a', newline='', encoding='utf-8') as csv_file:
                    csv_writer = csv.writer(csv_file, delimiter=';', quotechar='|', quoting=csv.QUOTE_MINIMAL)
                    csv_writer.writerow(row)

            if yield_rows:
                yield row

        print('[I] Batch has been parsed and saved.')
```

Рисунок 2.15 – Сохранение полученной информации об объявлении в файл

```
Scraping offer on url https://krasnodar.cian.ru/sale/flat/250847391/
Scraping offer on url https://krasnodar.cian.ru/sale/flat/257493174/
Scraping offer on url https://krasnodar.cian.ru/sale/flat/255780244/
Scraping offer on url https://krasnodar.cian.ru/sale/flat/255034434/
Scraping offer on url https://krasnodar.cian.ru/sale/flat/256818718/
Scraping offer on url https://krasnodar.cian.ru/sale/flat/256875834/
Scraping offer on url https://krasnodar.cian.ru/sale/flat/254757435/
Scraping offer on url https://krasnodar.cian.ru/sale/flat/247526701/
[I] Scraping done for page 25, room_type 9 in 30 seconds. Start parsing...
[DOUBLE] Skip offer saving
[DOUBLE] Skip offer saving
[DOUBLE] Skip offer saving
[DOUBLE] Skip offer saving
[I] Batch has been parsed and saved.
[NMP]: Skip page 26 for room_type 9
Scraping past in 9097 seconds. Fetched 4981 offers.
```

Рисунок 2.16 – Результат работы парсера

Как было оговорено ранее, за счёт изменения значений словаря *CITY* можно парсить предложения и для других городов. Помимо Краснодара, я получил таблицы для Санкт-Петербура, Калининграда, Сочи, Новосибирска и Хабаровска. В общем получилось 32 132 объявления!

После разработки RESTful сервиса и отправки парсером данных на сервер, было решено оставить также режим сохранения в CSV файл, так как это понадобится в дальнейшем при обучении нейронной сети в среде Google Colaboratory.

3 Анализ данных и обучение нейронной сети

3.1 Анализ полученных данных

После того как данные получены, их можно изучить. Так мы узнаем, какие факторы стоит включить в нейронную сеть, а от каких следует избавиться.

Для этих целей воспользуемся Python библиотеками NumPy, Pandas и Matplotlib. Анализ будет проводиться в среде Jupyter Notebook. Все необходимые компоненты доступны при установке пакета Anaconda или же можно воспользоваться онлайн-платформой Google Colaboratory.

Десериализуем таблицу в объект Pandas DataFrame и выведем часть таблицы на рисунке 3.1.

```
path = '/content/drive/My Drive/calculus_data/shuffled_flats_data.csv'
pd_flats = pd.read_csv(path, sep=';', header=0)
```

```
[ ] pd_flats
```

	ID	Link	Price	Rooms	Views	Square	Live Square	Kitchen	Floor	Total Floors	Residental Complex	District	Flat number
0	256194855	https://krasnodar.cian.ru/sale/flat/256194855/	2280000	1	1	39.2	NaN	NaN	4	7	NaN	Прикубанский	20
1	252607744	https://krasnodar.cian.ru/sale/flat/252607744/	2900000	1	2	43.0	18.0	14.0	19	26	ЖК «Москва»	Прикубанский	79/3к2
2	253570915	https://krasnodar.cian.ru/sale/flat/253570915/	2700000	1	857	44.0	NaN	NaN	16	24	ЖК «Галактика»	Карасунский	71к1

Рисунок 3.1 – Десериализация таблицы

Для начала посмотрим в каких столбцах больше всего пропусков данных. Подсчитаем их количество и процент от общего числа объявлений, отсортируем и выведем на рисунке 3.2.

В дальнейшем исключим факторы, у которых больше 40% пустых значений. Также выбросим столбцы, которые не относятся к оценке стоимости квартиры (ID, Link и Views).

	Total	Percent
Microistrict 2	4778	0.937230
Gas	4437	0.870341
Parking	4043	0.793056
Residental Complex	2923	0.573362
Ceiling	2915	0.571793
House Year	2910	0.570812
Window View	1907	0.374068
Lifts	1832	0.359357
Flat number	1513	0.296783
Microdistrict	1404	0.275402
Balcony	1117	0.219106
Kitchen	622	0.122009
Toilet	529	0.103766
House Type	523	0.102589
Live Square	464	0.091016
Repair	257	0.050412
Price	0	0.000000

Рисунок 3.2 – Количество пропущенных значений по столбцам

Для анализа количественных переменных выведем так называемую «тепловую» корреляционную матрицу (heatmap) на рисунке 3.3. Чем ярче квадрат на пересечении двух переменных, тем выше корреляция между ними.

Видна высокая зависимость целевой переменной от значения общей площади квартиры (Square). Исключим экзогенные переменные Live Square (жилая площадь) и Kitchen (площадь кухни), так как они сильно коррелируют с Square (общая площадь).

С другой стороны, корреляции переменных House Year (год постройки жилья) и Floor Edge (искусственная переменная, обозначает минимальное количество промежуточных этажей от этажа квартиры до первого или последнего этажей дома) близки к нулю, поэтому от них можно избавиться.

Теперь рассмотрим категориальные переменные. Говорить о их значимости для целевой переменной будем основываясь на диаграмму размаха, также известную как ящик с усами (box and whiskers plot).

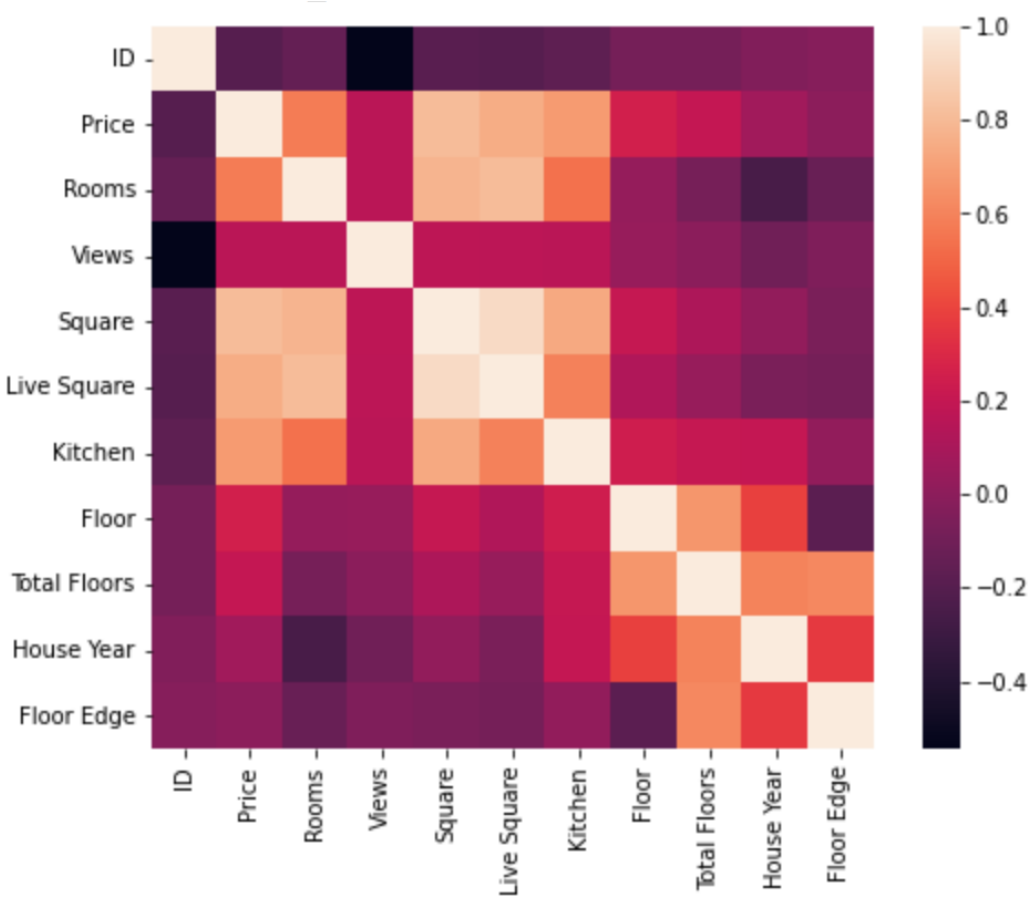


Рисунок 3.3 – Тепловая матрица корреляций

Выберем любую категориальную переменную, у которой меньше 40% пропусков значений, например Repair (ремонт) и посмотрим на её диаграмму размаха на рисурке 3.4. По вертикали цена квартиры, по горизонтали какой в ней сделан ремонт. Закрашенные прямоугольники ограничены сверху и снизу, соответственно, верхней и нижней квартилями, а чёрная черта внутри них – это медиана. «Усы» – края статистически значимой выборки, а чёрные точки – это выбросы, которые вышли за эти края.

Распределение значений считается хорошим, если диаграмма выглядит симметричной и компактной. Помимо этого, для нас критерием использования категориальной переменной является то как расположены диаграммы относительно друг друга в рамках одной переменной. Например для Repair видно, что квартиры с дизайнерским ремонтом обычно сильно дороже остальных квартир. Поэтому мы можем её использовать.

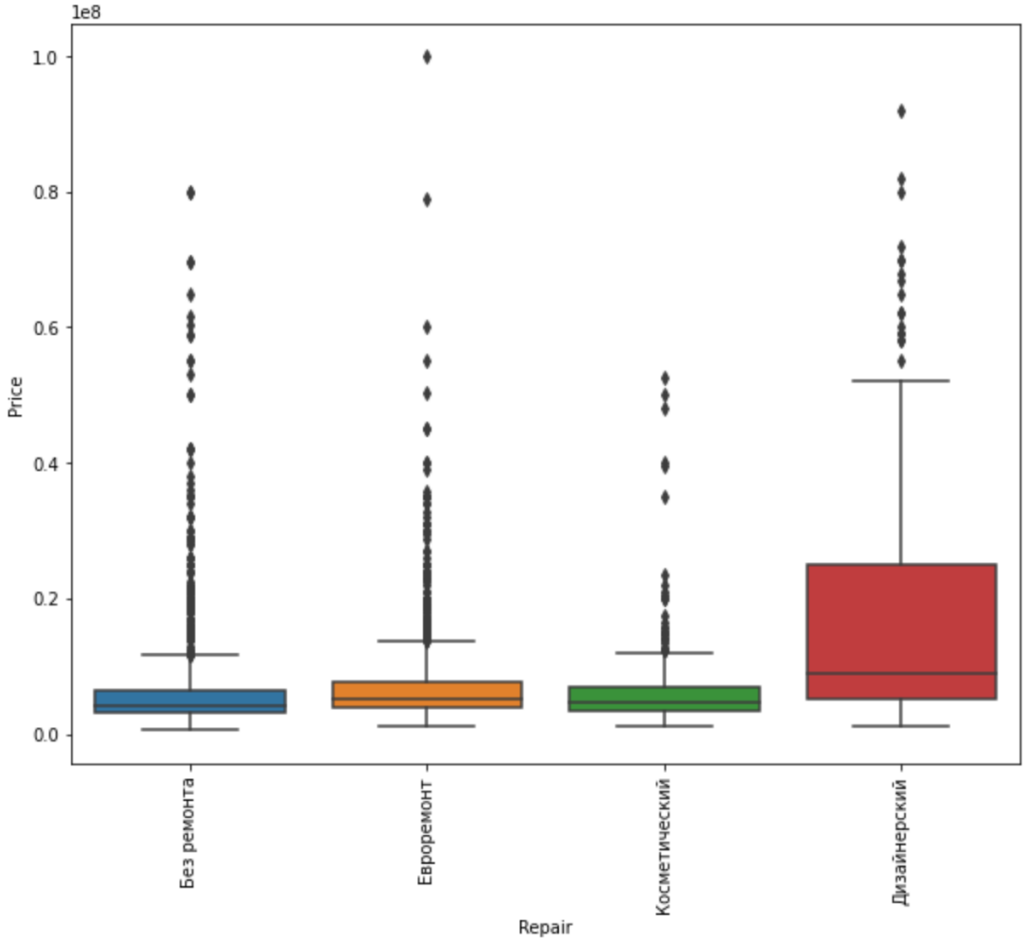


Рисунок 3.4 – Диаграммы размаха Repair

Для фактора Balcony (тип балкона) на рисунке 3.5 можно увидеть большое количество выбросов, но, так как в общем случае диаграммы размаха отличаются для разных значений, мы также можем его использовать.

Не будем использовать такие переменные, как Window View (вид из окна), диаграммы размаха которых изображены на рисунке 3.6. У значений этой переменной большое количество выбросов, а сами диаграммы несильно отличаются друг от друга.

Вместе с этим, не стоит использовать те категориальные переменные, в которых какое-нибудь значение встречается сильно чаще других, так как нельзя говорить о её репрезентативности.

Выбросим факторы со слишком большим количеством значений, таких как Street (улица), диаграммы размаха которой невозможно прочесть (рисунок 3.7).

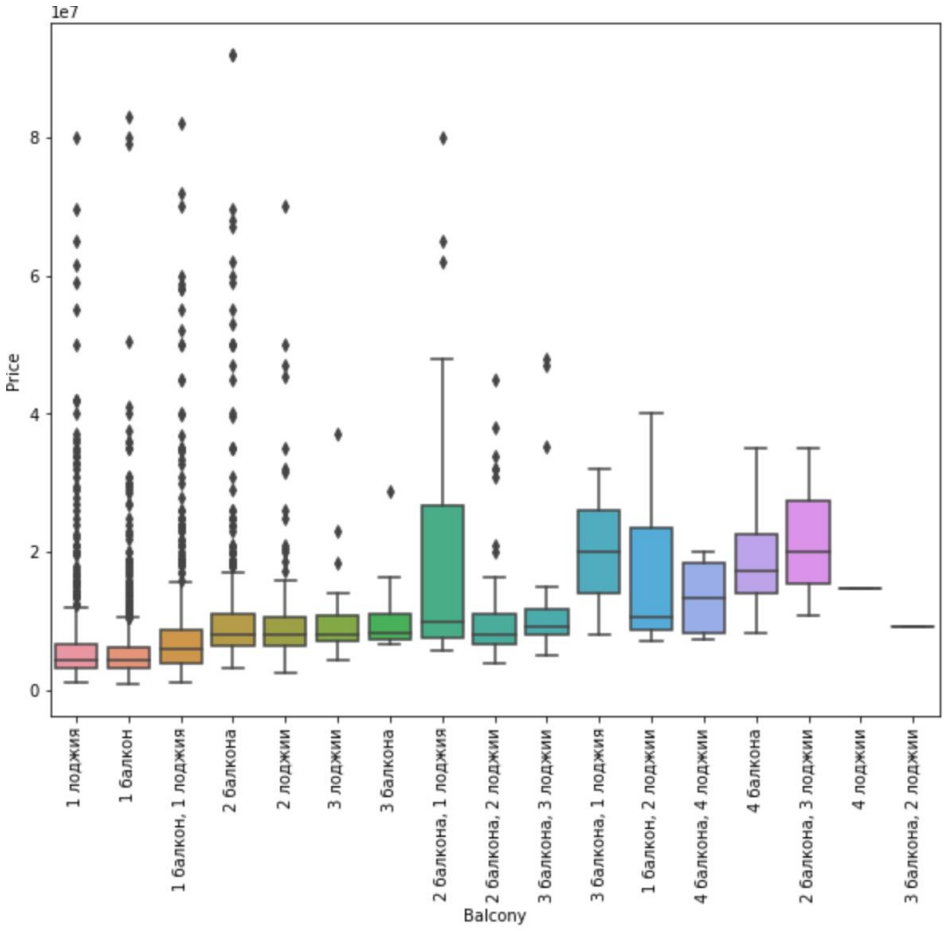


Рисунок 3.5 – Диаграммы размаха Balcony

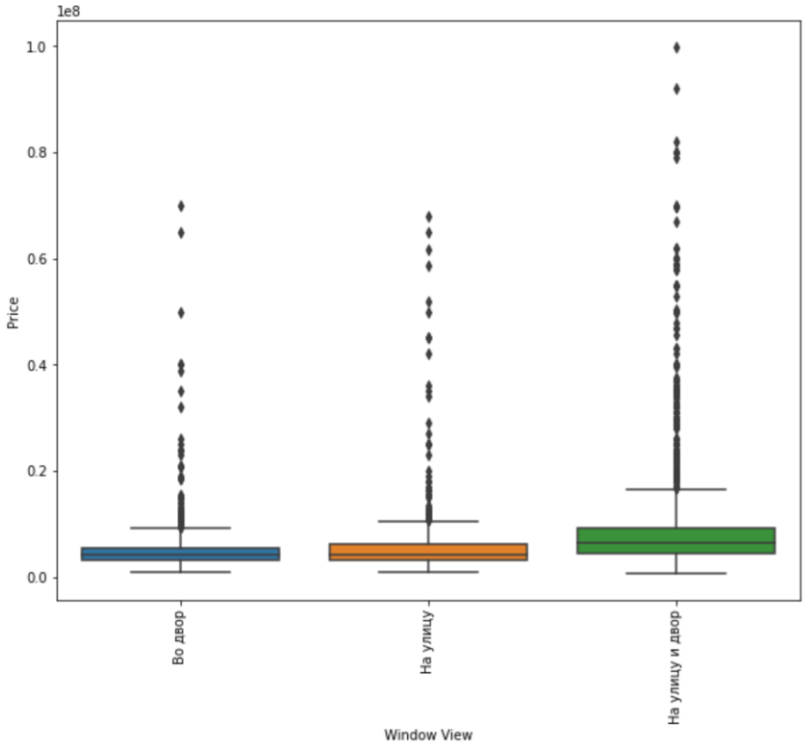


Рисунок 3.6 – Диаграммы размаха Window View

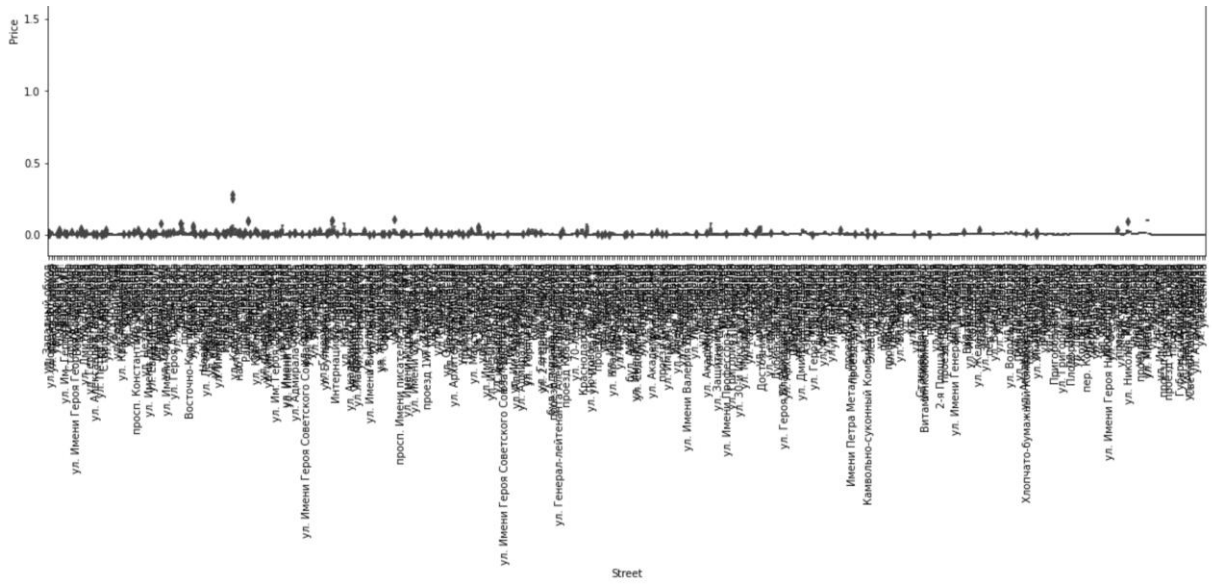


Рисунок 3.7 – Нечитаемые диаграммы

После описанных выше процедур, для обучения нейронной сети были отобраны следующие параметры:

- Price – целевая переменная, стоимость квартиры в объявлении;
- Rooms – количество комнат (студии соответствуют 0);
- Square – общая площадь;
- Floor – этаж, на котором расположена квартира;
- Total Floors – количество этажей в доме;
- District – район или округ, в котором находится квартира;
- Microdistrict – микрорайон или жилмассив;
- Toilet – количество и типы санузлов;
- Balcony – количество балконов и/или лоджий;
- Repair – тип ремонта;
- House Type – материал, из которого построен дом;
- Lifts – количество и типы лифтов в доме.

3.2 Обучение нейронной сети

Для обучения нейронной сети был использован фреймворк Keras на базе Tensorflow. Также воспользовался вспомогательными библиотеками:

– pandas – для десериализации и динамической обработки таблицы с результатом парсинга;

– Matplotlib – для построения графиков;

– NumPy – для работы с тензорами;

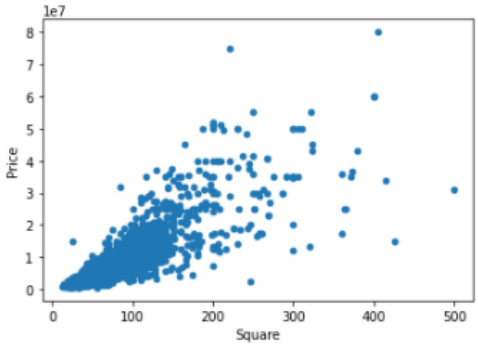
– sklearn – для создания препроцессора данных;

– joblib – для сериализации объектов и дальнейшего использования при деплое нейронной сети на сервере.

После десериализации таблицы с объявлениями и разделения её на обучающую и проверочную выборки, избавимся от «выбросов». «Выброс» – это такой объект выборки, который очень сильно отличается от других объектов и, скорее всего, просто является ошибкой наблюдения. В нашем случае избавимся от таких объявлений, цена за квадратный метр которых отличается от средней цены в два стандартных отклонения. На рисунке 3.8 изображен график частот переменных Price и Square, а также процесс избавления от выбросов. После этого объём выборки снизился с 4981 до 4770 объявлений.

Для тренировки сети выделим из обучающей выборки выборку для валидации. Она нужна для сверки результатов предсказания модели на этапе обучения, а проверочная (тестовая) выборка используется на уже обученной сети для получения представления об её качестве. На рисунке 3.9 изображено деление всей таблицы на обучающую и тестовую, затем выделения из обучающей валидационной выборки, а также процесс нормализации. Все категориальные переменные до этого были преобразованы при помощи OneHotEncoder.

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff2403fba50>
```




```
[ ]  pps = pd_flats['Price'] / pd_flats['Square']
```

```
[ ]  pps.mean() + 2 * pps.std()
```

```
153814.03692483326
```

```
[ ]  holy_drop = pd_flats.drop(pd_flats[(pd_flats['Price']/pd_flats['Square']>pps.mean()+2*pps.std())
     |(pd_flats['Price']/pd_flats['Square']<pps.mean()-1.5*pps.std())].index)
```

```
[ ]  holy_drop.shape[0]
```

```
4770
```

Рисунок 3.8 – График частот и избавление от выбросов

```
[ ]  from sklearn.model_selection import train_test_split

     data_train, data_test = train_test_split(np_flats, test_size=0.15, random_state=123)
```

```
[ ]  print('Data Train shape: {}\nData Test shape: {}'.format(data_train.shape, data_test.shape))
```

```
Data Train shape: (4054, 162)
Data Test shape: (716, 162)
```

```
[ ]  y_train, X_train = data_train[:, 0], data_train[:, 1:]
     y_test, X_test = data_test[:, 0], data_test[:, 1:]
```

```
[ ]  mean = X_train.mean(axis=0)
     X_train -= mean
     X_test -= mean
     std = X_train.std(axis=0)
     X_train /= std
     X_test /= std
```

```
[ ]  X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=123)
```

Рисунок 3.9 – Выделение выборок и нормализация

Архитектура нейронной сети представлена на рисунке 3.10.

```
model = models.Sequential()
model.add(
    layers.Dense(
      1024,
      activation='relu',
      input_shape=(X_train.shape[1],),
      kernel_regularizer=regularizers.l1_l2(0.001, 0.001)
      )
)
model.add(layers.Dropout(0.5))
model.add(
    layers.Dense(
        2048,
        activation='tanh',
        kernel_regularizer=regularizers.l1_l2(0.001, 0.001)
    )
)
model.add(layers.Dropout(0.25))
model.add(
    layers.Dense(
        512,
        activation='relu',
        kernel_regularizer=regularizers.l1_l2(0.001, 0.001)
    )
)
model.add(layers.Dropout(0.5))
model.add(
    layers.Dense(
      256,
      activation='relu',
      kernel_regularizer=regularizers.l1_l2(0.001, 0.001)
    )
)
model.add(layers.Dense(1))
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

Рисунок 3.10 – Архитектура нейронной сети

Наша нейронная сеть – это многослойный перцептрон, на каждом слое которого присутствует регуляризация весов, а между скрытыми слоями происходит прореживание. Эти два способа предназначены для уменьшения переобучения сети. Гиперпараметры, такие как количество нод в каждом слое и функции активации, были подобраны экспериментально. В качестве оптимизатора используется алгоритм «Adam», ошибка рассчитывается при помощи среднеквадратичного отклонения, а в процессе обучения сети нам будет отображаться среднее абсолютное отклонение.

Запустим обучение сети со следующими параметрами (рисунок 3.11):

```
history = model.fit(
    X_tr,
    y_tr,
    epochs=64,
    batch_size=256,
    verbose=0,
    validation_data=(X_val, y_val)
)
val_mse_score, val_mae_score = model.evaluate(X_val, y_val)
```

```
14/14 [==============================] - 0s 3ms/step - loss:
```

```
val_mae_score
```

```
901415.375
```

Рисунок 3.11 – Параметры обучения сети

Видно, что на выборке валидации средняя абсолютная ошибка (MAE) составила чуть больше 900000 рублей. Рассмотрим ниже, приемлем ли такой результат.

На рисунке 3.12 изображен способ подсчёта процентов отклонений реальных значений и предсказанных: разность этих значений делится на настоящую стоимость квартиры, и, если отклонение составляет более 100%, то такие значения аккумулируются (для последнего столбца диаграммы).

Отобразим график и выведем распределение ошибки на рисунке 3.13. Из графика и таблицы результатов видно, что отклонение составляет ~17% по среднему арифметическому и ~11% по медиане. Также видно, что верхняя квартиль отклонения приблизительно равна 21%.

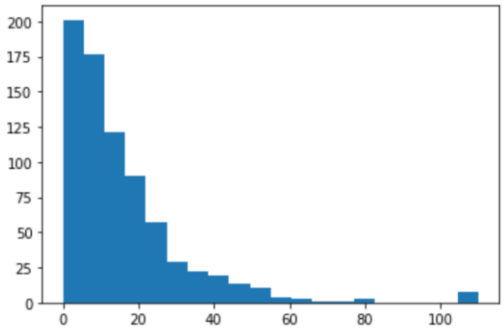
Можно сказать, что в среднем на миллион рублей стоимости квартиры отклонение составляет 110 тысяч рублей. Будем считать, что результат приемлем.

```
counter = 0
error_percents = []
for y_pr, y_real, x in zip(y_pred, y_test, X_test):
    y_pr = y_pr[0]
    e = y_pr - y_real
    err_percent = 100 * e / y_real
    error_percents.append(abs(err_percent))
    if err_percent > 100:
      error_percents.append(105)  # аккумулируем все больше 100
      print(y_real, '->', y_pr, '; %', err_percent, '\tx:', x)
error_percents = np.array(error_percents)
```

Рисунок 3.12 – Код расчёта отклонений

```
[70] plt.hist(error_percents, bins=20, range=(0, 110))

(array([201., 176., 121.,  90.,  57.,  29.,  22.,  19.,  13.,  10.,   4.,
          3.,   1.,   1.,   3.,   0.,   0.,   0.,   0.,   8.]),
 array([  0. ,   5.5,  11. ,  16.5,  22. ,  27.5,  33. ,  38.5,  44. ,
        49.5,  55. ,  60.5,  66. ,  71.5,  77. ,  82.5,  88. ,  93.5,
        99. , 104.5, 110. ]),
 <a list of 20 Patch objects>)
```




```
[71] err_pd = pd.DataFrame(error_percents)
     err_pd.describe()
```

	0
count	766.000000
mean	17.244539
std	22.257036
min	0.002500
25%	5.159666
50%	11.441902
75%	21.104266
max	243.364393

Рисунок 3.13 – График и описание отклонений

Сохраним модель для деплоя на сервер. Вместе с моделью также необходимо сохранить различные объекты, которые были нужны для

преобразования данных перед обучением сети. Сущность *transformer* содержит способ кодирования категориальных переменных. На рисунке 3.14 сохраняется модель и объект для преобразований. Помимо запуска блока кода, нужно скачать файлы из окружения Google Colab.

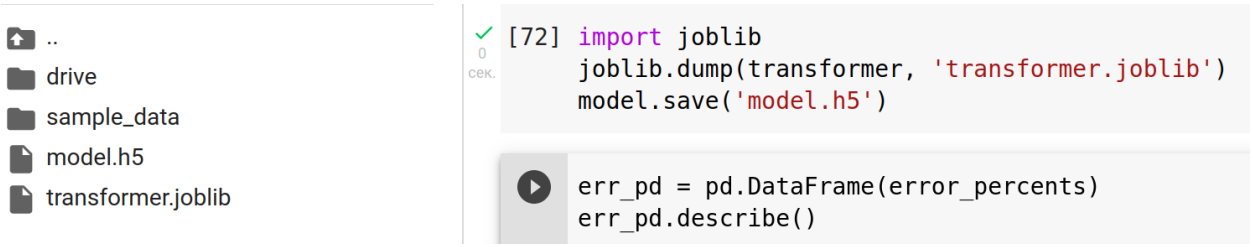


Рисунок 3.14 – Сохранение данных

4 Разработка RESTful сервиса и демонстрационного сайта

4.1 Что такое RESTful?

REST или Representational State Transfer– это архитектурный стиль приложений, определяющей работу API (программный интерфейс приложения). REST API базируется на протоколе HTTP и использует его методы. Архитектуру REST можно описать следующими шестью ограничениями, введённых Роем Филдингом:

– Единый интерфейс – определяет взаимодействие между клиентом и сервером;

– Отсутствие состояний – вся нужная информация для запроса содержится в самом запросе (например, нет сессий);

– Кеширование – возможность клиента кешировать ответы;

– Клиент-сервер – строгое разделение задач клиента от задач сервера;

– Многоуровневая система – сервис может состоять из нескольких серверов;

– Код по требованию (опционально) – возможность передать программный код клиенту.

Сервис является RESTful, если подходит под все вышеперечисленные ограничения, кроме, может быть, последнего.

4.2 Разработка сервиса

В качестве фреймворка используется набирающий популярность FastAPI. Его главная фишка – разработка ведётся вокруг подсказок типов (type hints), представленных в Python 3.6. Благодаря им легко настраивается валидация и обработка входных и выходных данных. Также из коробки доступна автогенерируемая документация на Swagger и ReDoc.

Работа с базой данных ведётся через ORM SqlAlchemy, СУБД PostgreSQL.

На сервер также нужно поставить ML библиотеки, которые использовались в предыдущем разделе: Keras, sklearn и joblib.

Выведем на рисунке 4.1 структуру проекта.

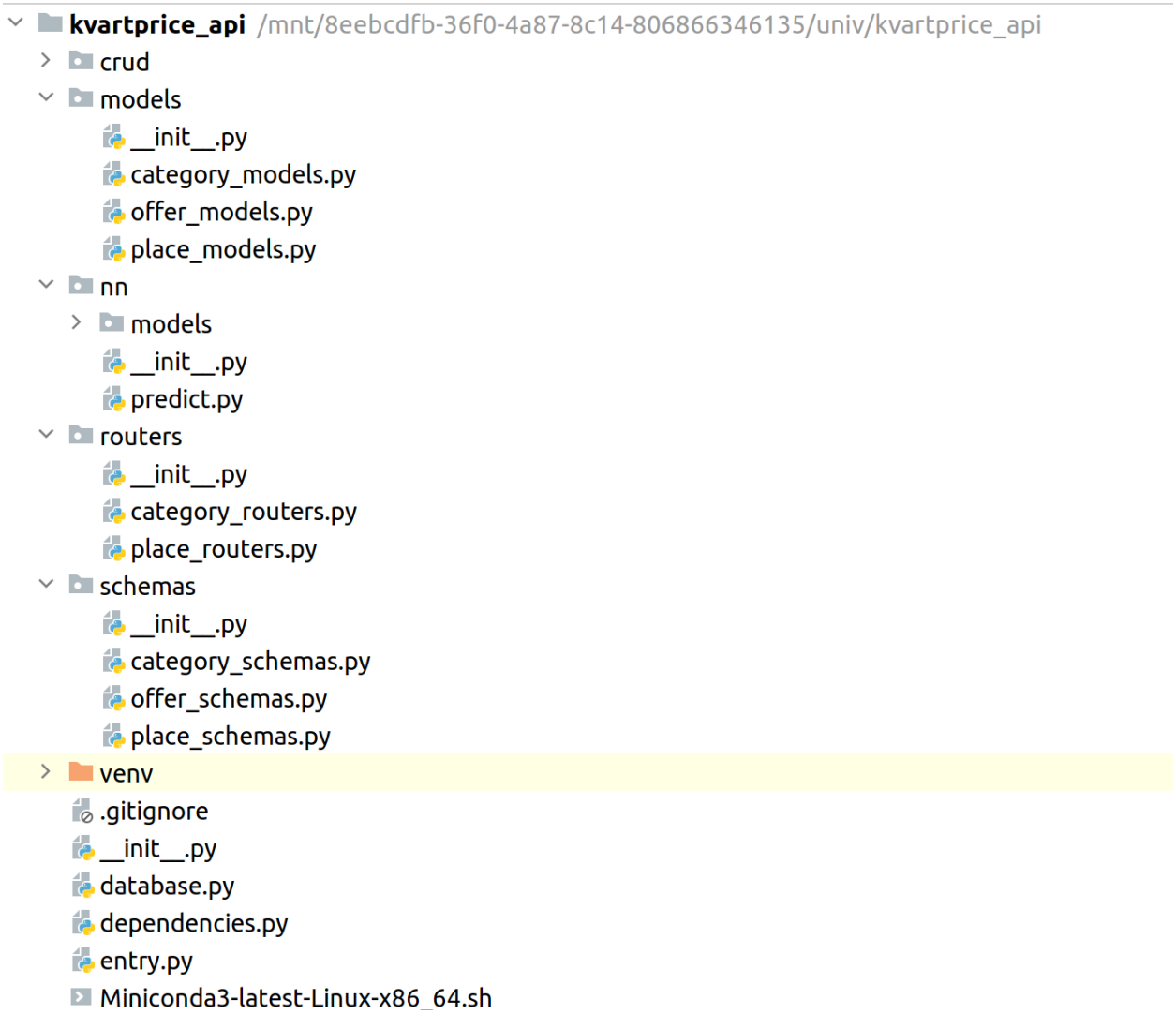
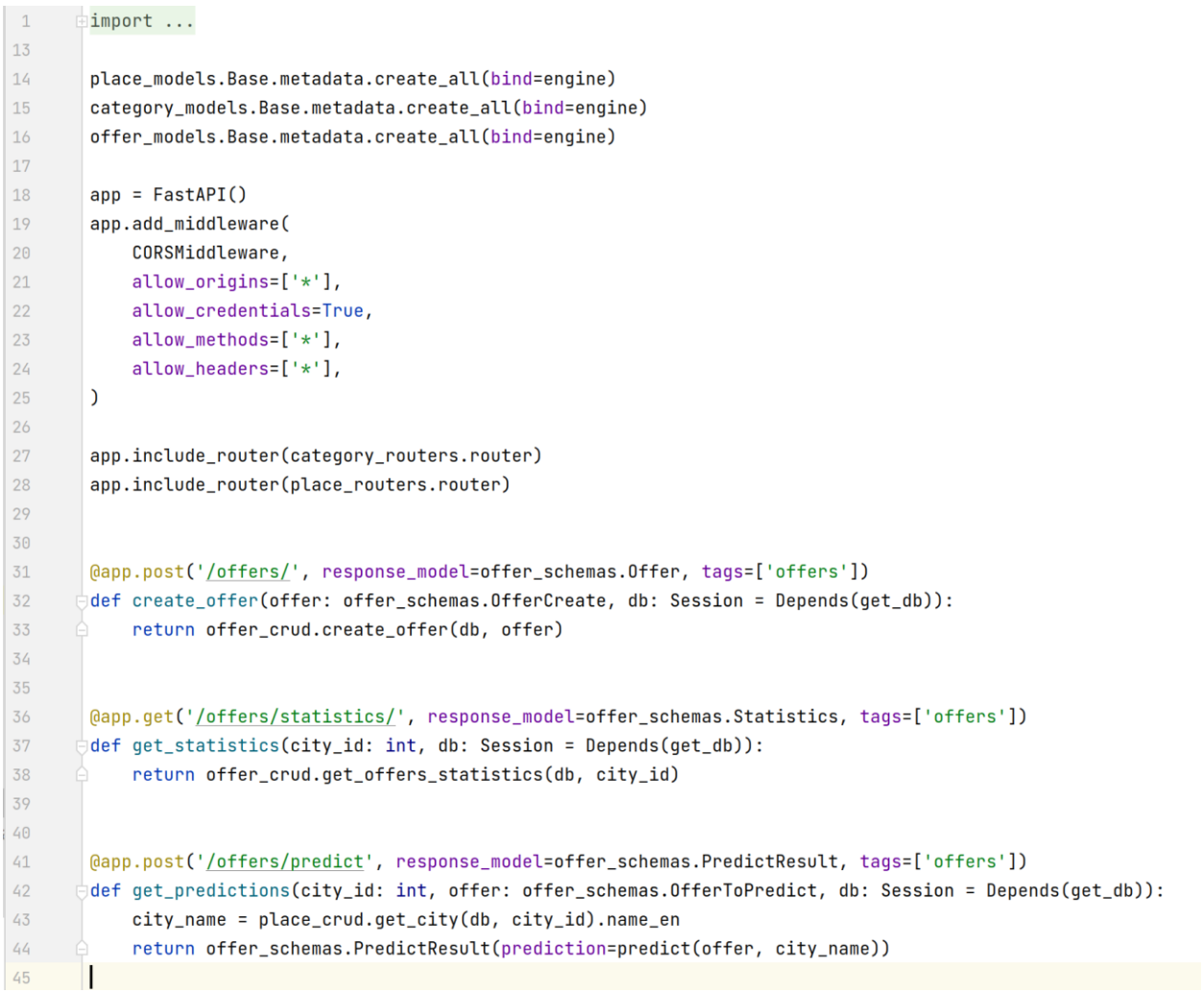


Рисунок 4.1 – Структура проекта

Пробежимся по назначениям модулей и пакетов.

Модуль *entry.py* является точкой входа. В нём объявляется инстанс приложения FastAPI и подключаются пути до функций-обработчиков запросов. Выведем его на рисунке 4.2.



```python
import ...

place_models.Base.metadata.create_all(bind=engine)
category_models.Base.metadata.create_all(bind=engine)
offer_models.Base.metadata.create_all(bind=engine)

app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=['*'],
    allow_credentials=True,
    allow_methods=['*'],
    allow_headers=['*'],
)

app.include_router(category_routers.router)
app.include_router(place_routers.router)


@app.post('/offers/', response_model=offer_schemas.Offer, tags=['offers'])
def create_offer(offer: offer_schemas.OfferCreate, db: Session = Depends(get_db)):
    return offer_crud.create_offer(db, offer)


@app.get('/offers/statistics/', response_model=offer_schemas.Statistics, tags=['offers'])
def get_statistics(city_id: int, db: Session = Depends(get_db)):
    return offer_crud.get_offers_statistics(db, city_id)


@app.post('/offers/predict', response_model=offer_schemas.PredictResult, tags=['offers'])
def get_predictions(city_id: int, offer: offer_schemas.OfferToPredict, db: Session = Depends(get_db)):
    city_name = place_crud.get_city(db, city_id).name_en
    return offer_schemas.PredictResult(prediction=predict(offer, city_name))
```

Рисунок 4.2 – Код модуля entry.py

Настройки для подключения к базе данных осуществляется в *database.py*, а в пакете *models* находятся классы-модели. Выведем один из них, например *place_models.py*, на рисунке 4.3. Такие классы выступают посредниками между объектами Python и отношений в БД. Позже можно будет работать с БД не используя при этом SQL! В этом и заключается суть ORM.

Теперь перейдём к типизации. FastAPI работает с подсказками типов через библиотеку Pydantic. Создаются схемы, которые определяют, какие поля ждём от пользователя и какие возвращаем сами. Продемонстрируем файл *place_schemas.py* из пакета *shemas* на рисунке 4.4.

```python
import ...


class City(Base):
    __tablename__ = 'cities'

    city_id = Column(Integer, primary_key=True, index=True)
    city_code = Column(Integer, unique=True)
    name_en = Column(String, index=True)
    name_ru = Column(String)

    districts = relationship('District', back_populates='city')


class District(Base):
    __tablename__ = 'districts'

    district_id = Column(Integer, primary_key=True, index=True)
    name = Column(String)
    city_id = Column(Integer, ForeignKey('cities.city_id'))

    city = relationship('City', back_populates='districts')
    microdistricts = relationship('Microdistrict', back_populates='district')


class Microdistrict(Base):
    __tablename__ = 'microdistricts'

    microdistrict_id = Column(Integer, primary_key=True, index=True)
    name = Column(String)
    district_id = Column(Integer, ForeignKey('districts.district_id'))

    district = relationship('District', back_populates='microdistricts')

```

Рисунок 4.3 – Код модуля place_models.py

В пакете *crud* лежат модули с функциями-описаниями запросов к БД через ORM.

В модулях пакета *routers* описаны руты и эндпоинты сервиса. Разберём часть файла *place_routers.py* для примера на рисунке 4.5. Видим, что узел и HTTP-метод, передающиеся в эндпоинт, задаются через декораторы. Здесь же указывается схема, в которую будет обёрнут ответ. Если же нужно задать схему для запроса, то она указывается у параметра эндпоинта.

```python
from pydantic import BaseModel

class MicrodistrictBase(BaseModel):...

class MicrodistrictCreate(MicrodistrictBase):...

class Microdistrict(MicrodistrictBase):
    microdistrict_id: int
    district_id: int

    class Config:
        orm_mode = True


class DistrictBase(BaseModel):...

class DistrictCreate(DistrictBase):...

class District(DistrictBase):
    district_id: int
    city_id: int
    microdistricts: list[Microdistrict] = []

    class Config:
        orm_mode = True

class CityBase(BaseModel):
    name_en: str
    name_ru: str
    city_code: int

class CityCreate(CityBase):...

class City(CityBase):
    city_id: int
    districts: list[District] = []

    class Config:
        orm_mode = True

```

Рисунок 4.4 – Код модуля place_schemas.py

Теперь можем запустить сервер и перейти на страницу сгенерированной документации (по пути */docs/*). Можем проверить работоспособность любого из эндпоинтов, как на рисунке 4.6. На рисунке 4.7. изображен список всех доступных эндпоинтов.

```python
import ...

router = APIRouter()


@router.post('/cities/', response_model=place_schemas.City, tags=['places'])
def create_city(city: place_schemas.CityCreate, db: Session = Depends(get_db)):
    db_city = place_crud.get_city_by_name(db, city.name_en)
    if db_city:
        raise HTTPException(status_code=400, detail='City already exists')

    return place_crud.create_city(db, city)


@router.get('/cities/', response_model=list[place_schemas.City], tags=['places'])
def read_cities(db: Session = Depends(get_db)):
    return place_crud.get_cities(db)


@router.get('/cities/{city_id}', response_model=place_schemas.City, tags=['places'])
def read_city(city_id: int, db: Session = Depends(get_db)):
    db_city = place_crud.get_city(db, city_id)
    if not db_city:
        raise HTTPException(status_code=404, detail='City not found')

    return db_city

```

Рисунок 4.5 – Код модуля place_routers.py

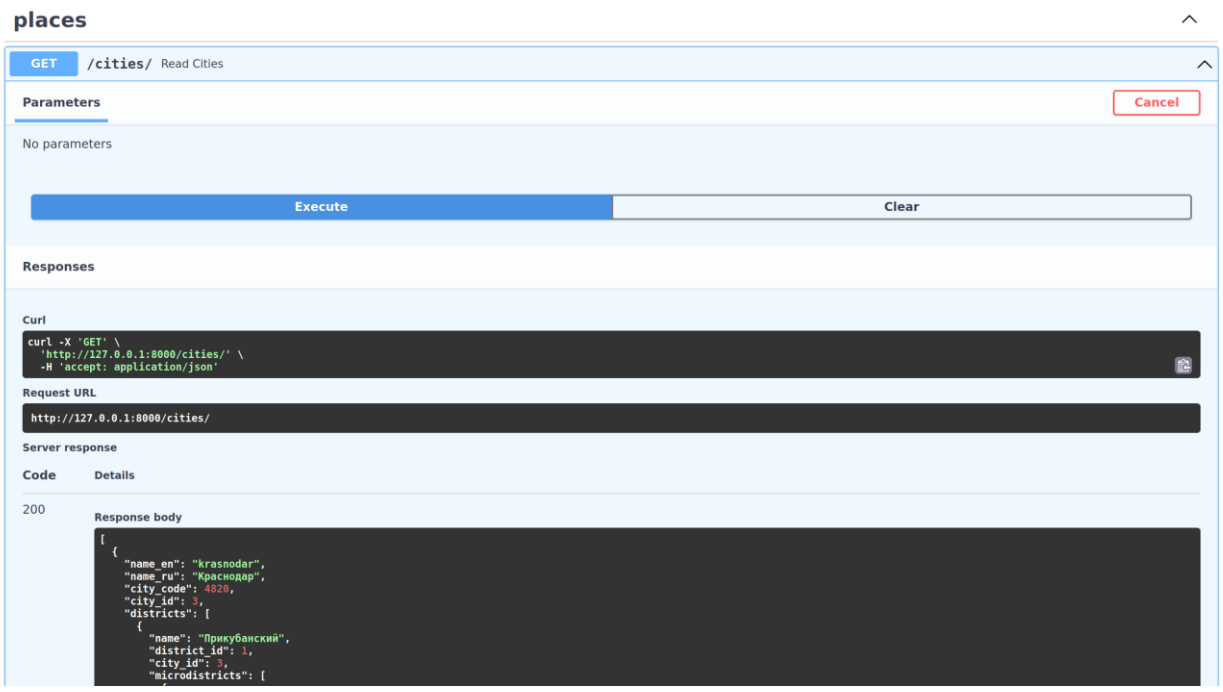


Рисунок 4.6 – Работа с API через Swagger

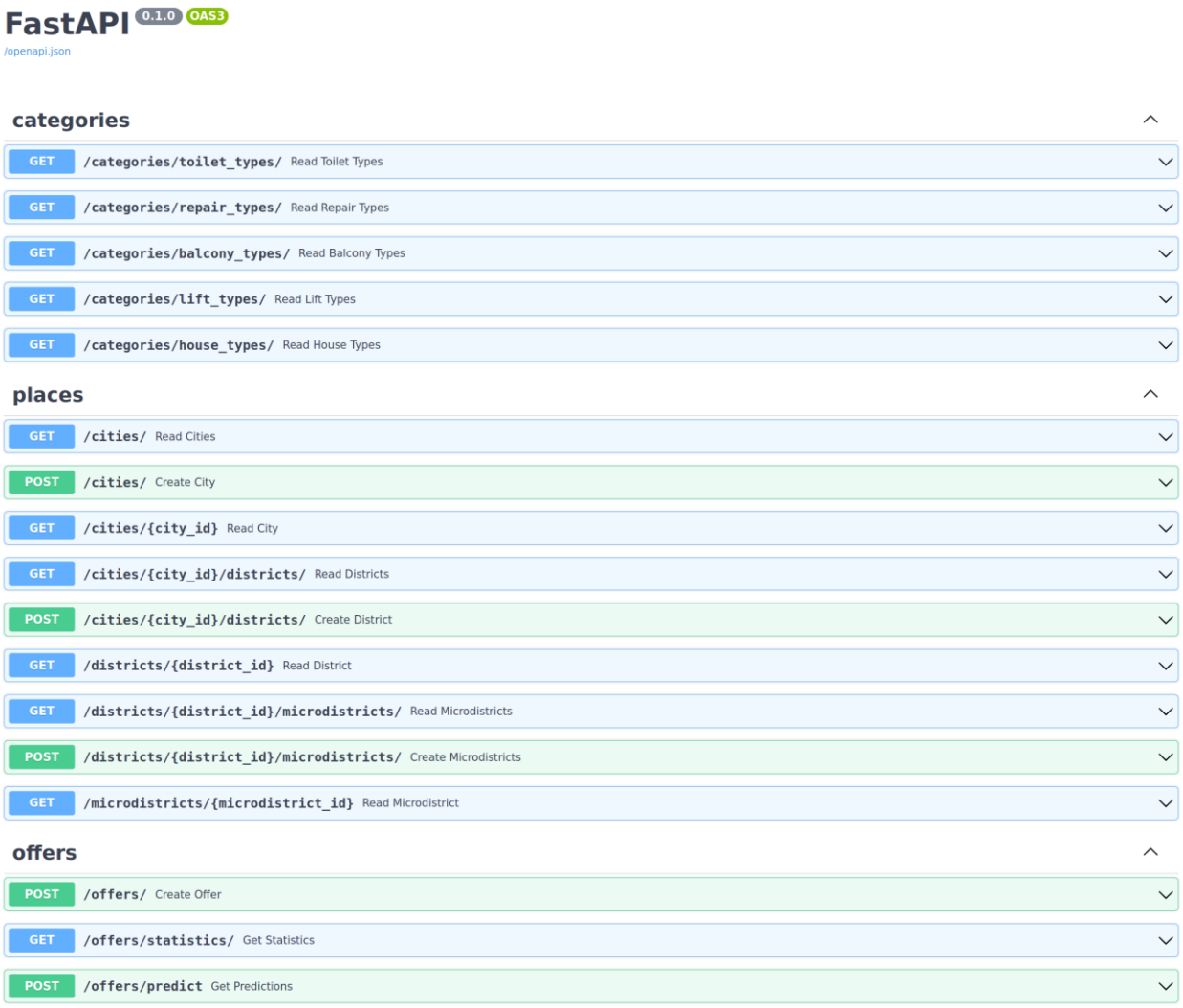


Рисунок 4.6 – Список всех эндпоинтов

4.3 Демонстрационный сайт

Заполнение данных на сайте, таких как статистика парсинга и опции для выпадающих меню, происходит при помощи кода на JavaScript, который делает AJAX запросы к API сервера и далее заполняет необходимые элементы DOM. Например, для получения списка городов на рисунке 4.6 идёт GET запрос к /*cities*/.

Внешний вид сайта изображен на рисунках 4.7 (форма) и 4.8 (предсказанной стоимостью жилья).

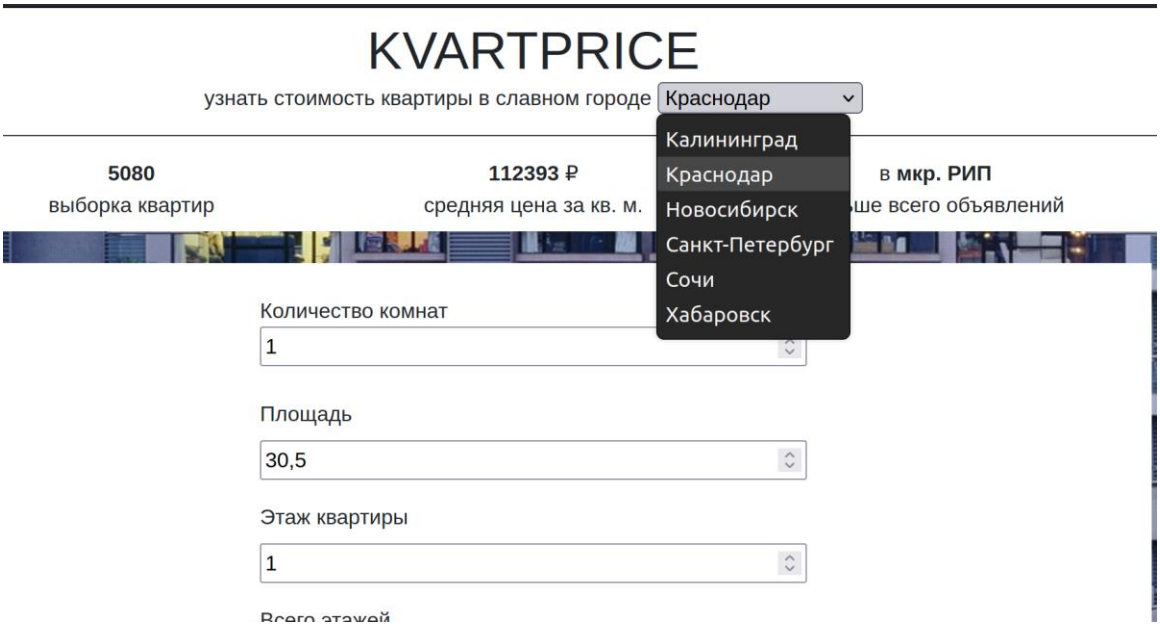


Рисунок 4.6 – Список городов

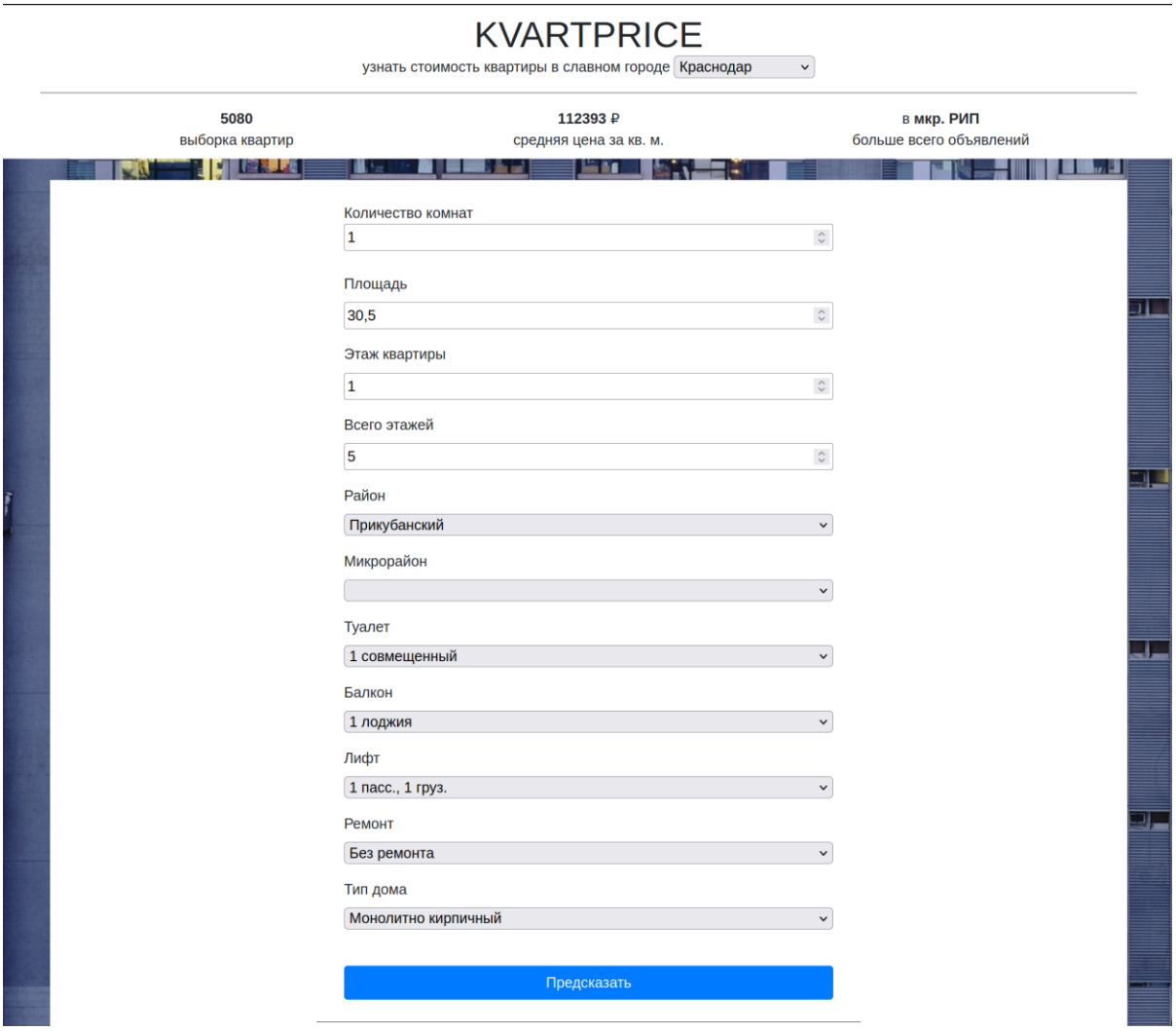


Рисунок 4.7 – Основная страница сайта

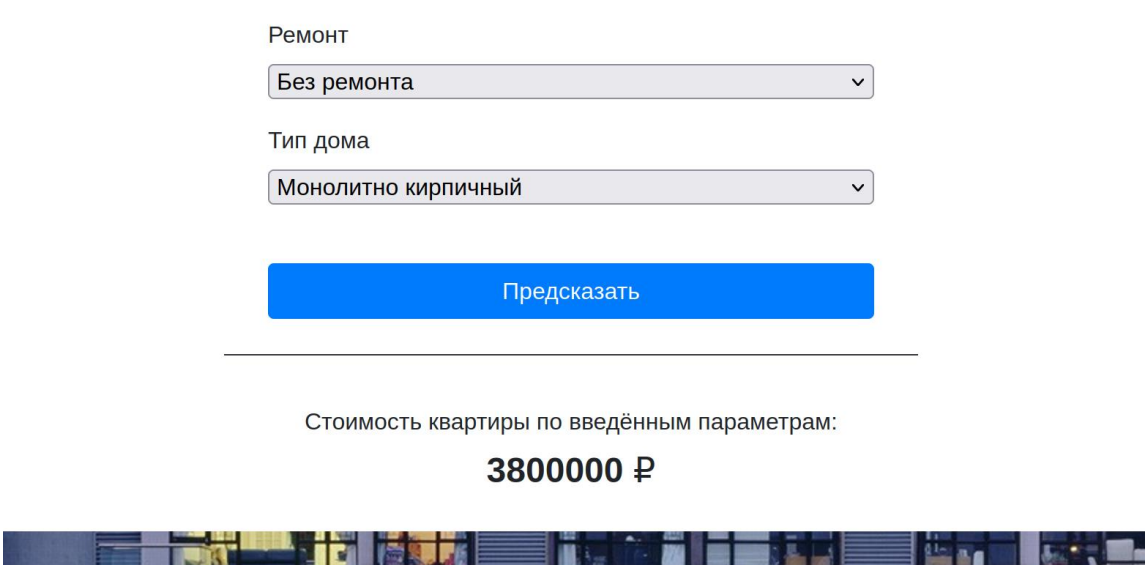


Рисунок 4.8 – Предсказнная стоимость

ЗАКЛЮЧЕНИЕ

Была проделана масштабная работа по созданию инструмента для предсказания стоимости квартир по их характеристикам – от разработки парсера до обучения нейронной сети и создания сервиса под неё.

В парсер заложена возможность получения данных для разных городов и его легко можно адаптировать при помощи добавления новых фильтров. В итоге сам сайт cian.ru оказался не лучшим решением для получения данных об объявлениях, так как выдача предложений искусственно ограничена, и защита сайта не позволяет использовать свой сервер для парсинга с определённой периодичностью (не проходит проверку на бота, даже с headless браузером через 100 запросов просит подтверждения).

Нейронная сеть дала хороший результат, но он может быть улучшен за счёт другого подхода к прореживанию выбросов (было достаточно много отклонений >100%, чтобы так считать) или применением иной, более подходящей архитектуры самой сети.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Статья агентства недвижимости Трансферт о вторичном жилье. URL – https://transfert-vrn.ru/novosti/pervichnoe-i-vtorichnoe-zhile-raznica/ (дата обращения 22.05.2022)
2. Статья Росреестра о рекорде продаж на вторичном рынке жилья за месяц. URL: – https://rosreestr.gov.ru/site/press/news/v-moskve-vpervye-prevyshen-pokazatel-v-20-tys-registratsiy-na-vtorichnom-rynke-zhilya-/ (дата обращения 24.05.2022)
3. Статья РБК о рекордном спросе на новостройки в Москве. URL: – https://realty.rbc.ru/news/60092b869a7947e5c4a109bb (дата обращения 24.05.2022)
4. Росриэлт. Цены на недвижимость в Краснодаре. URL: – https://rosrealt.ru/krasnodar/cena (дата обращения 24.05.2022)
5. Циан. Купить вторичное жилье – квартиры в Краснодаре. URL: – https://krasnodar.cian.ru/kupit-kvartiru-vtorichka/ (дата обращения 25.05.2022)
6. Шолле Ф. Глубокое обучение на Python / Ф. Шолле. – СПб.: Питер, 2020. – 400 с.
7. Вьюгин В. В. Математические основы машинного обучения и прогнозирования / В. В. Вьюгин. – М.: МЦНМО, 2013. – 304 с.
8. Samsung Research Russia. Онлайн–курс «Нейронные сети и компьютерное зрение». URL: – https://stepik.org/course/50352/info (дата обращения 27.02.2022)
9. Типология задач обучения по прецедентам. URL: – http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение (дата обращения 27.02.2022)
10. Что такое RESTful API?. URL: – https://aws.amazon.com/ru/what-is/restful-api/ (дата обращения 27.05.2022)

ОТЗЫВ

на выпускную квалификационную работу студента
ФГБОУ ВО «КубГУ» Захарова Павла Юрьевича направления 09.03.03
Прикладная информатика в экономике на тему:
«RESTful сервис для предсказания стоимости квартир с использованием машинного обучения»

Актуальность работы П.Ю. Захарова определяется высоким интересом к рынку вторичной недвижимости и вытекающей отсюда проблемой честной оценки предложений на нём. Одним из путей решения данной проблемы является создание модели машинного обучения, которая, могла бы оценивать квартиры по их характеристикам.

Целью данной работы является разработка групп связанных друг с другом приложений, использующих реальные предложения для оценки рыночной стоимости той или иной квартиры. При этом были решены следующие задачи:

- разработка средства для извлечения данных с сайта (парсера);
- обучение нейронной сети и оценка ошибки;
- разработка RESTful-сервиса для оценки стоимости квартир;
- демонстрация работы сервиса через сайт.

Работа выполнена самостоятельно, приведена авторская интерпретация результатов анализа, выводы обоснованы оценкой эффективности предлагаемых к внедрению решений.

Работа оформлена в соответствии с требованиями, содержащимися в методических указаниях по выполнению ВКР. Работа содержит совокупность результатов, свидетельствующих о приобретении выпускником необходимых общекультурных и профессиональных компетенций, а также о способности решать задачи прикладного характера. В ходе подготовки и защиты ВКР Захаров П.Ю. показал высокий уровень сформированности необходимых компетенций. ВКР не содержит существенных недостатков.

Выпускная квалификационная работа соответствует требованиям, предъявляемым к выпускным квалификационным работам, и может быть рекомендована к защите на заседании государственной аттестационной комиссии, заслуживает оценки «отлично».

Научный руководитель дипломной работы,
доцент кафедры анализа данных и
искусственного интеллекта КубГУ Г. А. Кесиян

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

СПРАВКА

о результатах проверки текстового документа
на наличие заимствований

Кубанский Государственный университет

ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

Автор работы: Захаров Павел Юрьевич
Самоцитирование рассчитано для: Захаров Павел Юрьевич
Название работы: RESTFUL СЕРВИС ДЛЯ ПРЕДСКАЗАНИЯ СТОИМОСТИ КВАРТИР С ИСПОЛЬЗОВАНИЕМ МАШИННОГО ОБУЧЕНИЯ
Тип работы: Выпускная квалификационная работа
Подразделение: ФКТиПМ, кафедра анализа данных и искусственного интеллекта

РЕЗУЛЬТАТЫ

■ **ОТЧЕТ О ПРОВЕРКЕ КОРРЕКТИРОВАЛСЯ:** НИЖЕ ПРЕДСТАВЛЕНЫ РЕЗУЛЬТАТЫ ПРОВЕРКИ ДО КОРРЕКТИРОВКИ

ЗАИМСТВОВАНИЯ	2.46%		ЗАИМСТВОВАНИЯ	2.46%	
ОРИГИНАЛЬНОСТЬ	90.76%		ОРИГИНАЛЬНОСТЬ	90.76%	
ЦИТИРОВАНИЯ	6.78%		ЦИТИРОВАНИЯ	6.78%	
САМОЦИТИРОВАНИЯ	0%		САМОЦИТИРОВАНИЯ	0%	

ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 23.06.2022

ДАТА И ВРЕМЯ КОРРЕКТИРОВКИ: 23.06.2022 07:20

Модули поиска: ИПС Адилет; Библиография; Сводная коллекция ЭБС; Интернет Плюс; Сводная коллекция РГБ; Цитирование; Переводные заимствования (RuEn); Переводные заимствования по eLIBRARY.RU (EnRu); Переводные заимствования по Интернету (EnRu); Переводные заимствования издательства Wiley (RuEn); eLIBRARY.RU; СПС ГАРАНТ; Модуль поиска "КубГУ"; Медицина; Диссертации НББ; Перефразирования по eLIBRARY.RU; Перефразирования по Интернету; Перефразирования по коллекции издательства Wiley; Патенты СССР, РФ, СНГ; СМИ России и СНГ; Шаблонные фразы; Кольцо вузов; Издательство Wiley; Переводные заимствования

Работу проверил: Калайдина Галина Вениаминовна
ФИО проверяющего

Дата подписи: 23.06.2022

Подпись проверяющего

Чтобы убедиться
в подлинности справки, используйте QR-код,
который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.