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

Отчёт по кейсу «Самолёт»:

Работу выполнил:
Студент группы № ИСП-22
Флейшгауэр Александр Михайлович

Долгопрудный, 2024

Введение

Данный отчет посвящен разработке системы прогнозирования цен на жилье, которая может стать ключевым инструментом для участников рынка недвижимости. В условиях нестабильной экономики и высокой волатильности цен возможность предсказать стоимость жилья позволяет покупателям, инвесторам и девелоперам принимать более обоснованные решения. Отчет направлен на анализ взаимосвязей между характеристиками объектов и их стоимостью, а также на разработку модели, обеспечивающей высокую точность предсказаний.

Цель

Собрать данные и произвести аналитическую работу над ними для будущих работ, например, создание модели на основе выводов.

Задачи

- Используя открытые источники собрать список данных.
- На основе полученной информации произвести удаление ненужных данных, дополнение необходимых, выявление аномалий и их блокировка.
- Визуализация данных при помощи, как минимум, двух инструментов для подобных задач. Нахождение взаимосвязей между данными или их полное отсутствие, усреднённых показателей для уверенного отчёта.
- Создание и тестирование модели для точного прогнозирования цены жилья

Методология

Исследование основывается на данных о недвижимости, собранных с помощью API DomClick. Набор данных включает разнообразные параметры объектов: их площадь, местоположение, тип, наличие инфраструктуры и ремонт, что позволяет провести комплексный анализ факторов, влияющих на стоимость жилья. Аналитический процесс включал следующие этапы:

1. **Сбор данных:** Данные получены с использованием API, что позволяет автоматизировать сбор информации и обеспечивает актуальность данных.

```
2 import requests
3 import hashlib
4 from datetime import datetime   5 from itertools import product
6  import json
7  import pandas as pd
8  import os
9  from concurrent.futures import ThreadPoolExecutor, as_completed
10 from ratelimit import limits, RateLimitException
11 from tenacity import retry, stop_after_attempt, wait_fixed
12
13             class DomClickApi:
14             def __init__(self):
15             self.session = requests.Session()
16             self.session.headers.update({
17             "X-Service": "true",
18             "Connection": "Keep-Alive",
19             "User-Agent": "Android; 12; Google; google_pixel_5; 8.72.0; 8720006; ; NONAUTH"
20             })
21
22         # Инициализация (получение cookies)
23         self.get("https://api.domclick.ru/core/no-auth-zone/api/v1/ensure_session")
24         self.get("https://ipoteka.domclick.ru/mobile/v1/feature_toggles")
25
26     @retry(stop=stop_after_attempt(3), wait=wait_fixed(2))
27     @limits(calls=10, period=1)
28    def get(self, url, **kwargs):
29         self.__update_headers(url,
**kwargs)
30         result =
self.session.get(url, **kwargs)  31
if result.status_code != 200:
32             raise RateLimitException("API response: {}".format(result.status_code), result)
33         return result
 34
35             def __update_headers(self, url, **kwargs):
36             url = self.__get_prepared_url(url, **kwargs)
37             sault = "ad65f331b02b90d868cbdd660d82aba0"
38             timestamp = str(int(datetime.now().timestamp()))
```

```
            encoded = (sault + url + timestamp).encode("UTF-8")
            h = hashlib.md5(encoded).hexdigest()
            self.session.headers.update({
            "Timestamp": timestamp,
            "Hash": "v1:" + h,
            })

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

    def fetch_offers(dca, params):
    offers_url = 'https://offers-service.domclick.ru/research/v5/offers/'
    offset = 0  54    limit = 10
     offers_list = []

    while True:
    res = dca.get(offers_url, params={**params, 'offset': offset, 'limit': limit})
      59        try:
                data = res.json()
                if not data['success']:
                break

            offers = data.get("result",
{}).get("items", [])  65            if not offers:
                break

            offers_list.extend(offers)
            offset += limit

            except json.JSONDecodeError:
            break

     return offers_list

        def generate_param_combinations():
        param_grid = {
        'address': ['77fa5072-181d-4264-97be-02498a8f0d5d', '1d1463ae-c80f-4d19-9331-
        a1b68a85b553',"9930cc20-32c
        'deal_type': ['sale'],
        'category': ['living'],
        'offer_type': [['flat'], ['layout']],
        'rooms': [None, 'st', '1', '2', '3', '4+'],
        }

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

def save_to_csv(df, filename):
new_data = df  # Default in case the file does not exist  90    if
  os.path.exists(filename):
        existing_df = pd.read_csv(filename)
        existing_ids = set(existing_df['id'].tolist())
        new_data = df[~df['id'].isin(existing_ids)]  # Filter new data based on existing IDs  94
        updated_df = pd.concat([existing_df, new_data], ignore_index=True)  95    else:
        updated_df = df  # If file doesn't exist, just use the
current dataframe  97
    updated_df.to_csv(filename, index=False)

     def process_dataframe(temp_df, columns_needed, unique_ids):
     available_columns = [col for col in columns_needed if col in temp_df.columns]
     temp_df = temp_df[available_columns]
     temp_df = temp_df[~temp_df['id'].isin(unique_ids)]
```

```
    return temp_df

        def main():
        dca = DomClickApi()
        unique_ids = set()
        max_unique_records = 100000
        columns_needed = [
        'object_info.area', 'object_info.floor', 'object_info.rooms',
        'object_info.is_apartment', 'category',
        'has_advance_payment', 'monthly_payment', 'trade_in', 'price_info.price',
        'price_info.square_price',
        'price_info.commission', 'price_info.price_for_year',
        'price_info.square_price_for_year',
        'legal_options.is_owner', 'legal_options.is_agent_owner_approved', 'offer_type',
        'discount_status.status 115        'discount_status.value', 'seo_info.subways',
        'seo_info.display_name_parts', 'deal_type', 'status',
       'is_auction', 'id', 'payment_order_id', 'description', 'updated_dt', 'source',
'duplicates_offer_count', 117        'address.id', 'address.kind', 'address.guid', 'address.name',
'address.display_name', 'address.parent_id 118        'address.position.lat',
'address.position.lon', 'address.locality.id', 'address.locality.kind',
        'address.locality.guid', 'address.locality.name', 'address.locality.display_name',
        'address.locality.par
        'address.info.timezone', 'address.info.timezone_offset', 'address.short_display_name',
        'ipoteka_rate',
        'pessimization.pessimized', 'published_dt', 'assignment_sale', 'house.floors',
        'renovation.type',
        'renovation.display_name', 'backwash',
        ]

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

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

                if 'id' not in temp_df.columns:
                continue

            temp_df = process_dataframe(temp_df, columns_needed, unique_ids)
            unique_ids.update(temp_df['id'].tolist())
            df = pd.concat([df, temp_df], ignore_index=True)

                if len(unique_ids) >= max_unique_records:
                break

    save_to_csv(df, 'offers_part_0.csv')

    if __name__ == "__main__":
    main()
```

2. **Предобработка данных**: На этом этапе данные очищались от дубликатов, пропусков и выбросов, что улучшило качество данных

для анализа и позволило избежать искажений в итоговых выводах. Код для каждого этапа обработки, включая фильтрацию, коррекцию пропусков и удаление выбросов, приведен далее в отчете.

```
print(f'Data has {df_copy.shape[0]} rows, {df_copy.shape[1]} columns.')
```

```
Data has 35255 rows, 73 columns.
```

```
# Удаляем дубликаты для минимизации избыточных данных
df_no_duplicates = df_copy.drop_duplicates()
print(f'Data has {df_no_duplicates.shape[0]} rows, {df_no_duplicates.shape[1]} columns.')
```

```
Data has 35226 rows, 73 columns.
```

```
df_no_duplicates.info()
```

3. Формирование признаков: Из исходного описания объектов были выделены дополнительные характеристики, такие как наличие мебели, инфраструктуры, типа отопления и состояния ремонта, что позволило учесть все аспекты, которые могут влиять на рыночную стоимость.

```
# Загрузка данных из CSV-файла
df = pd.read_csv('/content/drive/MyDrive/offers_part_0.csv')

# Функция для
извлечения информации
def
extract_info(description)
:
    # Проверка, что description
- это строка    if not
isinstance(description, str):
        return {}

    info = {}

        # Проверка наличия ключевых характеристик и присвоение значений
        info['repair'] = 'euro_repair' if re.search(r'евроремонт', description, re.IGNORECASE)
        else (
        'cosmetic_repair' if re.search(r'косметический ремонт', description, re.IGNORECASE) else
        'no_repair')

    info['furniture_and_appliances'] = 'fully_furnished' if re.search(r'полностью меблированная',
description, r

    info['balcony_loggia'] = 'loggia' if re.search(r'лоджия', description, re.IGNORECASE) else
'balcony'

        info['layout'] = 'open_plan' if re.search(r'свободная планировка', description,
        re.IGNORECASE) else (
```

```
        'studio' if re.search(r'студия', description, re.IGNORECASE) else 'separate_rooms')

    info['parking'] = 'underground_parking' if re.search(r'подземная парковка', description,
re.IGNORECASE) else

    info['courtyard'] = 'playgrounds' if re.search(r'детские площадки', description,
re.IGNORECASE) else 'green_

    info['sports_and_recreation'] = 'fitness_center' if re.search(r'фитнес-центр', description,
re.IGNORECASE) e

    info['shopping_centers'] = 'shopping_mall' if re.search(r'торговый центр', description,
re.IGNORECASE) else

    info['schools_and_kindergartens'] = 'school' if re.search(r'школа', description,
re.IGNORECASE) else 'kinder

    info['medical_facilities'] = 'hospital' if re.search(r'больница', description, re.IGNORECASE)
else 'clinic'

    info['property_history'] = 'new_building' if re.search(r'новостройка', description,
re.IGNORECASE) else 'sec

    info['garbage_chute'] = bool(re.search(r'мусоропровод', description, re.IGNORECASE))

    kitchen_size_match = re.search(r'кухня\s+(\d+(\.\d+)?)\s*кв\.?\s*м', description,
    re.IGNORECASE)
    info['kitchen_area'] = float(kitchen_size_match.group(1)) if kitchen_size_match else None

    info['heating'] = 'central_heating' if re.search(r'центральное отопление', description,
re.IGNORECASE) else

    info['water_supply'] = 'central_water_supply' if re.search(r'центральное водоснабжение',
description, re.IGN

    info['sewage'] = 'central_sewerage' if re.search(r'центральная канализация', description,
re.IGNORECASE) els

    total_area_match = re.search(r'площадью\s+(\d+(\.\d+)?)\s*кв\.?\s*м', description,
    re.IGNORECASE)
    info['total_area'] = float(total_area_match.group(1)) if total_area_match else None

    living_area_match = re.search(r'жилая площадь\s+(\d+(\.\d+)?)\s*кв\.?\s*м', description,
    re.IGNORECASE)
    info['living_area'] = float(living_area_match.group(1)) if living_area_match else None

    return info

# Создание копии DataFrame для обработки
df_copy = df.copy()

# Применение функции ко всем описаниям и добавление столбца
df_copy['characteristics'] = df_copy['description'].apply(extract_info)

# Разделение данных словаря на отдельные столбцы
df_copy = pd.concat([df_copy.drop(columns=['characteristics']),
  df_copy['characteristics'].apply(pd.Series)], ax

# Сохранение результатов в CSV-файл
df_copy.to_csv('processed_offers.csv', index=False, encoding='utf-8-sig')
```

4. Анализ корреляций и важности признаков: Оценивалась корреляция каждого признака с целевой переменной — ценой за квадратный метр. Выявлены признаки с высокой и средней степенью влияния, что позволило сократить объем данных, исключив второстепенные характеристики.

5. Моделирование: Для построения модели использовался алгоритм XGBoost Regressor, который показывает высокую точность на задачах регрессии с использованием большого количества признаков.

Основная часть: Анализ данных

Описание данных

Набор данных состоит из 35226 записей с 73 характеристиками. Включены такие параметры, как площадь квартиры, количество комнат, цена, расположение, инфраструктура в районе, наличие мебели и ремонт. Ниже приведены ключевые признаки, отобранные для анализа и построения модели.

- price_info.price — общая цена квартиры
- price_info.square_price — цена за квадратный метр(целевой показатель)..
- object_info.area — площадь квартиры.
- object_info.rooms — количество комнат.
- address.locality.display_name — название населенного пункта.
- house.floors — количество этажей в здании.
- category — категория объекта (квартира, студия и т.д.).
- deal_type — тип сделки (продажа, аренда).
- heating — тип отопления (центральное или индивидуальное).
- infrastructure_availability — наличие инфраструктуры, включающей школы, магазины, детские сады.
- sports_infrastructure — наличие спортивной инфраструктуры (спортивные центры, парки).

Каждый из перечисленных признаков имеет свои подкатегории и значения, которые были нормализованы и перекодированы для использования в модели.

Обработка данных

Удаление дубликатов

Удаление дубликатов позволило избежать избыточности данных. В результате осталось 35226 уникальных записей, что улучшает точность анализа.

Работа с выбросами

Для повышения качества данных была проведена очистка от выбросов на основе интерквартильного размаха (IQR) для ключевых признаков, таких как площадь и цена. Это позволило убрать аномально высокие или низкие значения, которые могли искажать прогнозы модели. Вот результаты сколько было удалено пропусков

Пример кода для удаления выбросов:

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

     # Вычисляем количество выбросов
     outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    num_outliers = len(outliers)

    # Фильтруем DataFrame, чтобы удалить выбросы
    df_cleaned = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

     return df_cleaned, num_outliers

# Пример использования
df_cleaned = df_baby.copy()
total_outliers = 0

    for column in df_cleaned.select_dtypes(include=[np.number]).columns:
    df_cleaned, num_outliers = remove_outliers_iqr(df_cleaned, column)
    total_outliers += num_outliers
    print(f"Столбец: {column}, Выбросы удалены: {num_outliers}")

print(f"Всего выбросов удалено: {total_outliers}")
```

```
Столбец: object_info.area, Выбросы удалены: 1333
Столбец: object_info.floor, Выбросы удалены: 618
Столбец: object_info.rooms, Выбросы удалены: 3
Столбец: price_info.price, Выбросы удалены: 1400
Столбец: price_info.square_price, Выбросы удалены: 397
Столбец: price_info.commission,
Выбросы удалены: 230 Столбец:
price_info.price_for_year, Выбросы
удалены: 269
Столбец: status, Выбросы удалены: 8
Столбец: address.position.lat, Выбросы удалены: 423
Столбец: address.position.lon, Выбросы удалены: 330
Столбец: ipoteka_rate, Выбросы удалены: 0
Столбец: house.floors, Выбросы удалены: 254
Столбец: infrastructure_availability,
Выбросы удалены: 0 Столбец:
sports_infrastructure, Выбросы удалены: 0
Всего выбросов удалено: 5265
```

Обработка пропусков

Часть данных содержала пропуски, которые были заполнены на основе моды, медианы или путем удаления строк с критически важными для анализа отсутствующими значениями. Например, для переменной object_info.is_apartment использовалось заполнение модой.

Пример кода для заполнения пропусков:

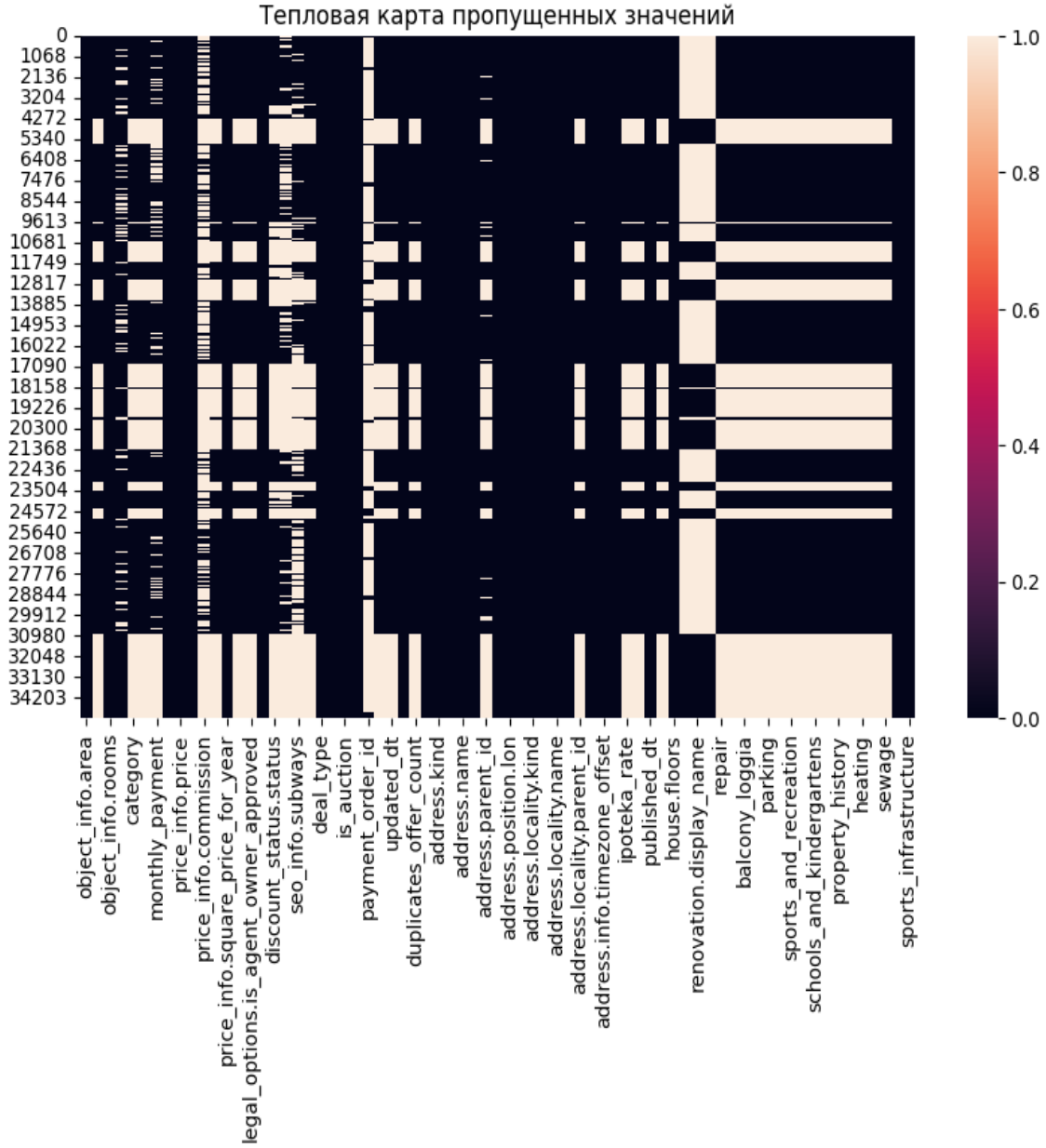
```
# Заполнение пропущенных значений в 'object_info.is_apartment' модой
mode_value = df_baby['object_info.is_apartment'].mode()[0]
df_baby['object_info.is_apartment'] = df_baby['object_info.is_apartment'].fillna(mode_value)

# Заполнение пропущенных значений в 'price_info.commission' нулями
df_baby['price_info.commission'] = df_baby['price_info.commission'].fillna(0)
```

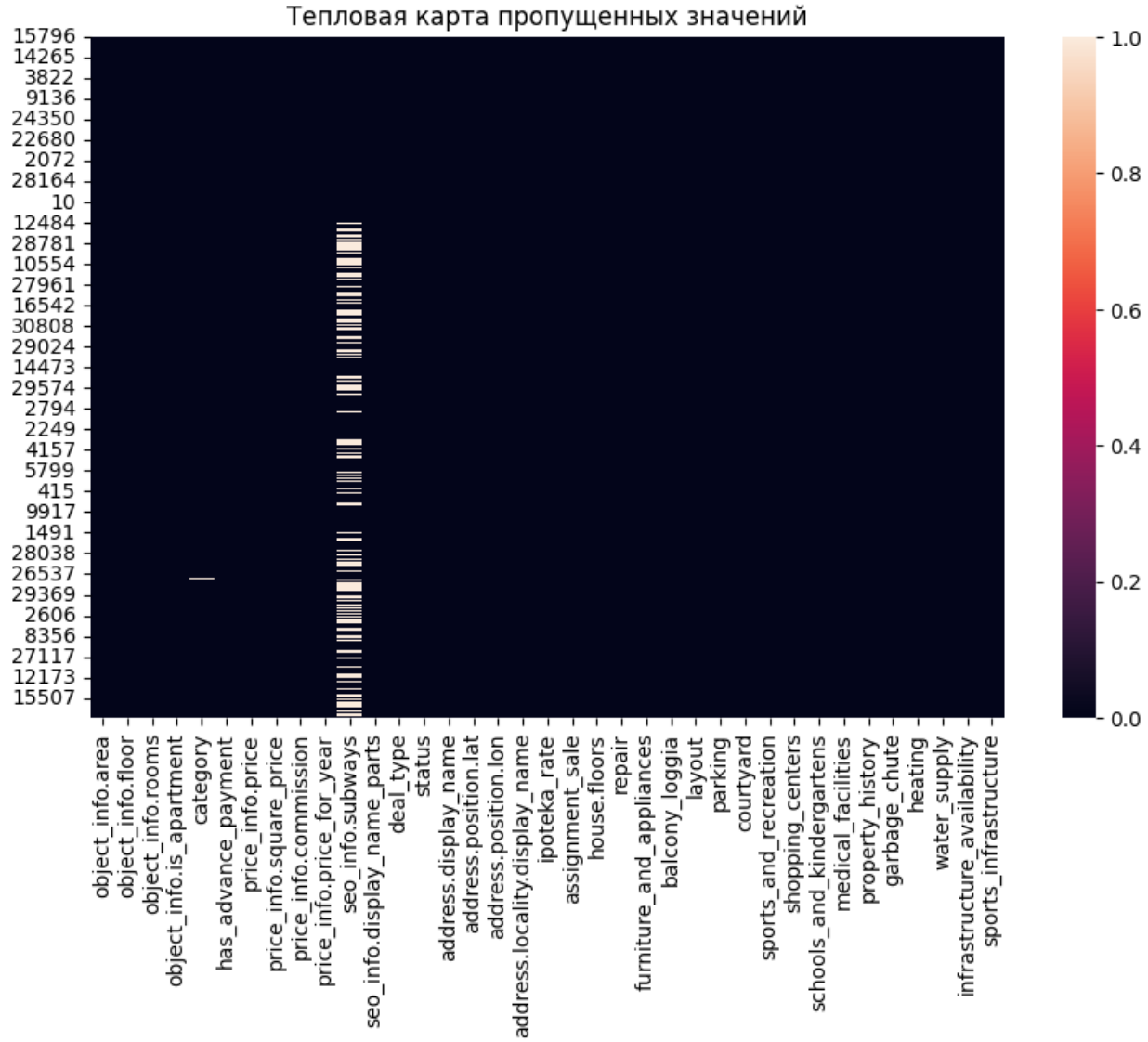
Визуализация пропущенных данных

Для анализа пропусков была построена тепловая карта, которая позволила оценить, в каких признаках наибольшее количество пропусков и принять решение об их обработке.

Тепловая карта пропущенных данных до заполнения пропусков:


Тепловая карта пропущенных значений
0
1068
2136
3204
4272
5340
6408
7476
8544
9613
10681
11749
12817
13885
14953
16022
17090
18158
19226
20300
21368
22436
23504
24572
25640
26708
27776
28844
29912
30980
32048
33130
34203
1.0
0.8
0.6
0.4
0.2
0.0
object_info.area
object_info.rooms
category
monthly_payment
price_info.price
price_info.commission
price_info.square_price_for_year
legal_options.is_agent_owner_approved
discount_status.status
seo_info.subways
deal_type
is_auction
payment_order_id
updated_dt
duplicates_offer_count
address.kind
address.name
address.parent_id
address.position.lon
address.locality.kind
address.locality.name
address.locality.parent_id
address.info.timezone_offset
ipoteka_rate
published_dt
house.floors
renovation.display_name
repair
balcony_loggia
parking
sports_and_recreation
schools_and_kindergartens
property_history
heating
sewage
sports_infrastructure

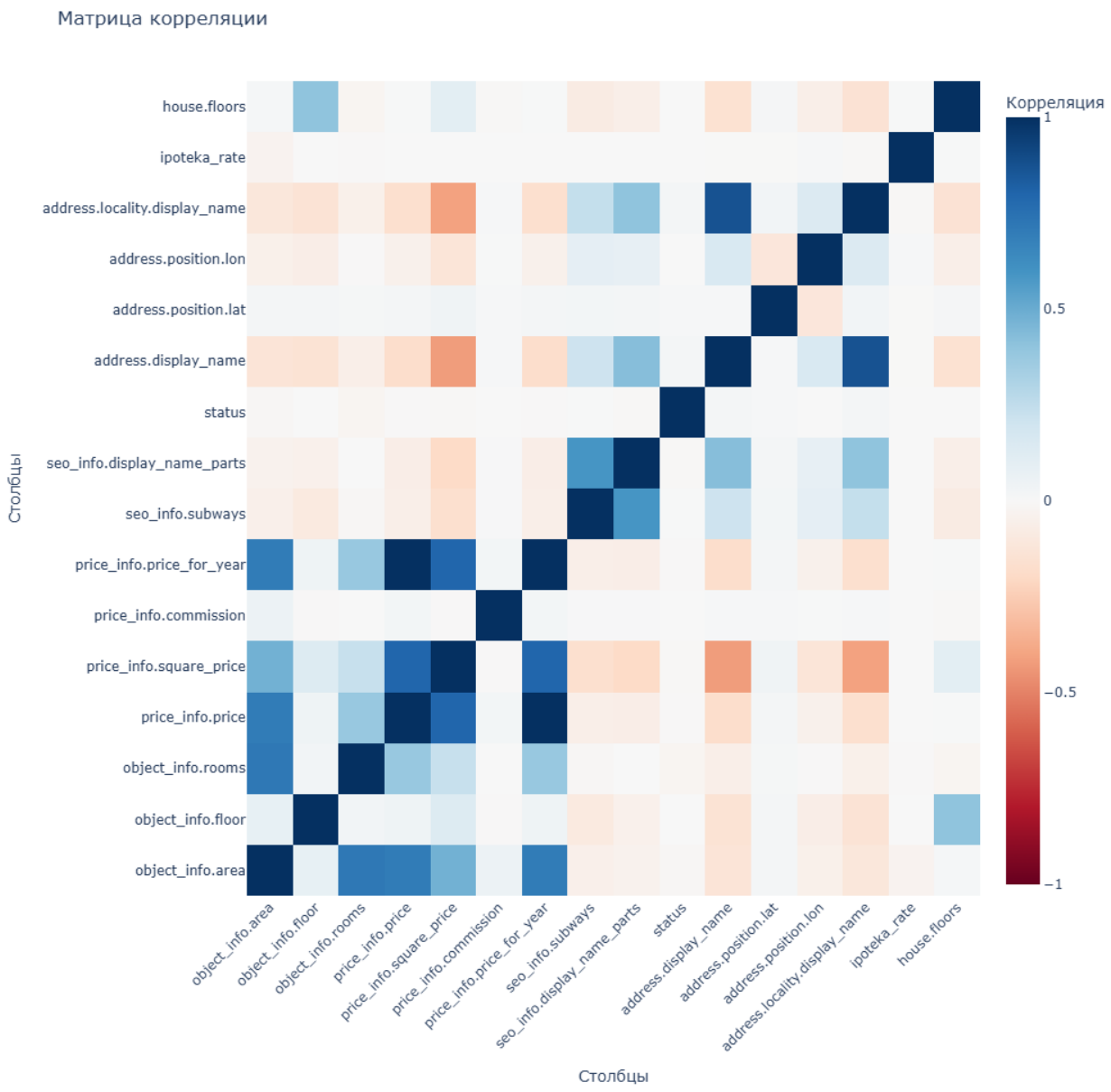
Тепловая карта пропущенных данных после заполнения пропусков:



Корреляционный анализ

Корреляционный анализ выявил признаки, которые имеют значительное влияние на целевую переменную — цену за квадратный метр. Наиболее значимые характеристики включают общую стоимость объекта, площадь, количество комнат и местоположение. Эти признаки показывают наибольшую корреляцию с ценой за квадратный метр.

Матрица корреляции:



Наиболее значимые признаки:

1. price_info.price (0.81) — высокая положительная корреляция с ценой за квадратный метр.

2. price_info.price_for_year (0.81) — аналогичная высокая корреляция, что может указывать на тесную связь с текущей ценой.

3. object_info.area (0.52) — положительная корреляция, указывающая, что большая площадь чаще всего ассоциируется с высокой стоимостью.

4. address.locality.display_name (-0.35) — отрицательная корреляция, отражающая влияние местоположения на снижение цен в определенных районах.

5. address.display_name (-0.32) — сходная отрицательная корреляция с местоположением, подкрепляющая значимость района.

6. house.floors (0.29) — положительная корреляция: высота дома часто положительно влияет на стоимость.

7. object_info.rooms (0.29) — небольшая положительная корреляция, показывающая связь количества комнат с ценой.

8. schools_and_kindergartens_kindergarten (0.17) — небольшая положительная корреляция с инфраструктурой для детей, что может быть важным для семей.

Моделирование

Для построения модели использовался алгоритм XGBoost Regressor, который обеспечивает высокую точность на задачах регрессии и хорошо справляется с большим числом признаков. Данные были разделены на обучающую и тестовую выборки (80% и 20% соответственно).

Пример кода для построения модели:

```
# Импортируем необходимые библиотеки
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Разделяем данные на обучающую и тестовую выборки
X = data_cleaned.drop(columns=['price_info.square_price'])  # Матрица признаков
y = data_cleaned['price_info.square_price']  # Целевая переменная
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Инициализация XGBoost регрессора с базовыми параметрами
model = XGBRegressor(objective='reg:squarederror', random_state=42)
# Обучение модели
model.fit(X_train, y_train)

# Предсказание на тестовой выборке
y_pred = model.predict(X_test)

# Вычисление метрик
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Вывод метрик
print(f"Средняя абсолютная ошибка (MAE): {mae:.2f}")
print(f"Корень из среднеквадратичной ошибки (RMSE): {rmse:.2f}")
```

```
Средняя абсолютная ошибка (MAE): 11647.84
Корень из среднеквадратичной ошибки (RMSE): 42396.15
```

Результаты модели:

Средняя абсолютная ошибка (MAE): 11647.84 — показывает среднюю погрешность предсказания модели.

Корень из среднеквадратичной ошибки (RMSE): 42396.15 — указывает на разброс предсказаний модели относительно истинных значений, что подтверждает высокую точность.

Рекомендации

1. Уделить внимание параметрам инфраструктуры и местоположения: обнаружено, что такие параметры, как наличие школ, детских садов и спортивной инфраструктуры, положительно влияют на цену за квадратный метр. Учет этих факторов может повысить точность прогнозов.

2. Введение временных данных для улучшения прогноза: включение сезонных данных и временных рядов в анализ позволит отслеживать сезонные колебания цен и учесть тренды, что может существенно улучшить модель.

3. Дополнительная работа с выбросами и пропусками: улучшение фильтрации выбросов и расширение методов заполнения пропусков позволит сократить ошибки и повысить качество данных, что также может улучшить точность предсказаний.

Заключение

В данном отчете была исследована взаимосвязь различных характеристик объектов недвижимости с их ценой за квадратный метр. Выявлены значимые признаки, такие как общая стоимость объекта, его площадь и расположение, которые оказывают значительное влияние на цену. Разработанная модель XGBoost показала удовлетворительную точность, достигнув MAE в 11647.84 и RMSE 42396.15. Для дальнейшего улучшения точности рекомендуется вводить временные данные и оптимизировать обработку пропусков и выбросов