



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

Институт кибербезопасности и цифровых технологий

(наименование института, филиала)

Кафедра информатики

(наименование кафедры)

## КУРСОВАЯ РАБОТА

по дисциплине

**Основы машинного обучения**

(наименование дисциплины)

Тема курсовой работы **Реализация алгоритмов оптимизации линейной регрессии  
для улучшения точности прогнозов стоимости недвижимости на языке  
программирования**

Студент группы

**БФБО-01-23, Ерофеев Олег Игоревич**

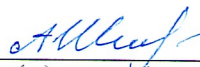
(учебная группа, фамилия, имя, отчество студента)

  
(подпись студента)

Руководитель курсовой  
работы

**зав. каф., к.ф.-м.н., доцент, Шмелева А.Г.**

(должность, звание, ученая степень, ФИО)

  
(подпись руководителя)

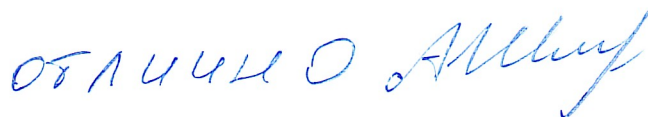

Рецензент (при наличии)

—  
(должность, звание, учёная степень)

—  
(подпись рецензента)

Работа представлена к защите «06» 05 2024 г.

Допущен к защите «06» 05 2024 г.

  
07.06.2024  




МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

Институт кибербезопасности и цифровых технологий

(наименование института, филиала)

Кафедра информатики

(наименование кафедры)

Утверждаю

заведующий кафедрой информатики

  
Подпись

Шмелева А.Г.  
ФИО

« 27 » 02 2024 г.

### ЗАДАНИЕ

на выполнение курсовой работы по дисциплине

«Основы машинного обучения»

Студент Ерофеев Олег Игоревич

Группа БФБО-01-23

**Тема** «Реализация алгоритмов оптимизации линейной регрессии для улучшения точности прогнозов стоимости недвижимости на языке программирования»


**Исходные данные:** база данных о стоимости аренды недвижимости

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:** изучение библиотек языка программирования, методов регрессионного анализа данных, метода наименьших квадратов, реализация алгоритмов, оценка точности моделей, визуализация результатов решения, работа с категориальными данными, работа с нормированием данных, размер тестовой выборки, мультиколлинеарность, работа с выбросами в данных.

**Срок предоставления к защите курсового проекта (работы):**

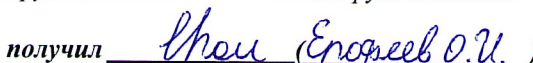
до « 06 » 05 2024 г.

**Задание на курсовой проект (работу) выдал**

  
Подпись руководителя

(Шмелева А.Г.)  
Ф.И.О. руководителя

« 26 » 02 2024 г. **Задание на курсовой проект (работу) получил**

  
Подпись обучающегося Ф.И.О. исполнителя

## Оглавление

Введение .....	4
1. Теоретические основы машинного обучения алгоритмов регрессионного анализа данных.....	6
1.1. Теоретические аспекты и сферы применения регрессионного анализа данных .....	7
1.2. Построение полиномов первой и второй степени, аппроксимирующие результаты эксперимента .....	28
2. Практическое применение и реализация алгоритмов оптимизации линейной регрессии для улучшения точности прогнозов стоимости недвижимости.....	32
2.1. Постановка задачи, обработки массива данных с применением модели множественной регрессии .....	33
2.2. Решение поставленной задачи .....	35
Заключение .....	52
Список использованных источников .....	54
Приложение 1 (листинг программы) .....	54

## **Введение**

Цель данной курсовой работы заключается в исследовании различных методов оптимизации линейной регрессии с целью улучшения точности прогнозов. В работе будут рассмотрены различные подходы и техники, такие как отбор признаков, обработка выбросов, работа с текстовыми данными и т.д.

Линейная регрессия является одним из наиболее широко используемых методов прогнозирования и анализа данных в различных областях. Она позволяет строить простые и понятные модели, которые могут быть использованы для предсказания значений зависимой переменной на основе набора независимых переменных.

Однако, в реальных условиях часто возникает необходимость улучшения точности таких прогнозов. Это может быть вызвано различными факторами, такими как некорректная спецификация модели, наличие выбросов в данных, мультиколлинеарность, ошибки или пропуски в данных или недостаточная обучающая выборка. В таких случаях оптимизация линейной регрессии становится необходимой для повышения качества прогнозов и улучшения эффективности модели.

Актуальность исследования различных методов оптимизации линейной регрессии с целью улучшения точности прогнозов обуславливается рядом факторов, отражающих ключевые тенденции в современной науке о данных, компьютерной статистике и машинном обучении. Важность развития и улучшения методов линейной регрессии реализации и оптимизации обусловлена потребностью в повышении точности и эффективности прогнозирования в различных областях — от экономики и социологии до медицины и экологии.

Растущий объём данных, их многообразие и сложность ставят перед наукой о данных задачу улучшения существующих и разработки новых методов анализа. В частности, оптимизация алгоритмов, таких как линейная регрессия, направлена на минимизацию ошибок прогнозирования и повышение общей устойчивости прогнозных моделей к изменениям в исходных данных.

Дополнительное значение исследования обусловлено всё более глубокой интеграцией машинного обучения и искусственного интеллекта в реальные бизнес-процессы и потребностью в разработке чётких, надёжных и масштабируемых решений для предприятий различного уровня. В этом контексте оптимизация линейной регрессии воспринимается как реальный вклад в развитие аналитической культуры и повышение качества принятия решений на основе данных.

Объектом исследования в данной курсовой работе является база данных со стоимостью аренды недвижимости в четырех городах России. Предмет исследования – методы оптимизации алгоритмов линейной регрессии для повышения точности прогнозов стоимости аренды недвижимости.

Для достижения поставленной цели будет проведён комплексный анализ сущности линейной регрессии, чтобы добиться поставленных задач, её основных параметров и факторов, влияющих на её качество. В ходе исследования будут рассмотрены практические примеры применения оптимизации линейной регрессии в разных сферах, демонстрируя значимость этих методов для повышения точности аналитических моделей. Внимание будет уделено проблеме переобучения и методам её предотвращения, способам оценки эффективности и точности моделей. Анализ существующих методик и подходов позволит сформулировать рекомендации по выбору оптимального набора инструментов оптимизации линейной регрессии в зависимости от исходных данных. В работе будет акцентировано внимание на важности предварительной обработки данных, включая нормализацию, масштабирование признаков и устранение выбросов, для улучшения качества моделирования.

## **1. Теоретические основы машинного обучения алгоритмов регрессионного анализа данных**

Эта глава будет посвящена теоретическим основам, лежащим в основе алгоритмов регрессионного анализа в контексте машинного обучения, начиная от основ регрессии, включая корреляции и базовые понятия, и завершая более сложными аспектами, такими как выбор признаков и анализ качества моделей.

Мы начнем с основ регрессии, затронем важность понимания корреляций между переменными, их взаимоотношений и влияния на результаты анализа. Далее, особое внимание будет уделено линейной регрессии - одному из наиболее простых и широко используемых видов регрессионного анализа. Пройдем путь от определения и применения линейной регрессии в задачах прогнозирования до методов оптимизации моделей, включая популярный метод наименьших квадратов.

Не менее важным аспектом является подготовка данных для анализа, включая процессы предобработки, нормализации, обработки выбросов и пропущенных значений, а также аспекты работы с текстовыми данными. Подбор и выбор признаков станут ключевыми моментами в оценке эффективности и точности аналитических моделей, а обсуждение мультиколлинеарности поможет избежать распространенных ошибок в анализе.

Завершая главу, мы углубимся в анализ моделей регрессии, затронем различные метрики качества линейных регрессионных моделей, что позволит не только создавать, но и критично оценивать и сравнивать различные подходы к решению задач прогнозирования.

Цель данной главы - предоставить комплексное представление о теоретических аспектах разработки и применения алгоритмов регрессионного анализа в машинном обучении, заложить основу для понимания более сложных концепций и методов.

## **1.1. Теоретические аспекты и сферы применения регрессионного анализа данных**

Прежде чем приступить к изучению регрессии необходимо разобраться с корреляцией.

Величины могут быть либо независимыми, либо связанными функциональной или стохастической (вероятностной) зависимостью [1].

Функциональная связь между величинами возникает, когда каждому значению одного измерения может быть однозначно приписано значение другого измерения. Примером может служить площадь круга, которая зависит от радиуса или площадь квадрата, зависящая от длины стороны. Однако, чаще всего, между переменными устанавливаются отношения, в которых одному значению аргумента не соответствует единственное значение другой величины, а целый набор потенциальных значений. Этот тип взаимосвязи называется стохастической или вероятностной зависимостью. Представим ситуацию, когда человек планирует свой отпуск на море. Он выбрал место, забронировал отель, купил билеты на самолет. Однако, успешность его отдыха будет зависеть от множества случайных факторов. Возможны непредвиденные обстоятельства, такие как отмена или задержка рейса, проблемы с проживанием в отеле, заболевание на отдыхе и т.д.

Корреляционная зависимость представляет собой особый пример вероятностной зависимости, выражающийся через стохастическую связь между случайными переменными. Эта зависимость характеризуется тем, что значения одной переменной функционально связаны со средними значениями другой переменной. Понятие "корреляция" (что в переводе с латинского означает соотношение или связь) ввели в XIX веке английский математик Карл Пирсон и антрополог Френсис Гальтон. Точки на координатной плоскости, обозначающие значения двух переменных, формируют корреляционное поле. Регрессионная функция, представляющая собой аналитическое приближение эмпирических данных, обозначает как изменяется одна переменная в ответ на изменения другой и стремится максимально точно приблизить положение эмпирических точек в

корреляционном поле. Эта функция может принимать различные формы, включая линейную, параболическую, гиперболическую, логарифмическую и другие.

Корреляционное поле – изображенные на координатной плоскости точки, которые обозначают значения первой и второй переменных (рис. 1.1.1.)  
Функциональная зависимость – аналитическая функция, аппроксимирующая наблюдаемые эмпирические значения (рис. 1.1.2.) [1].

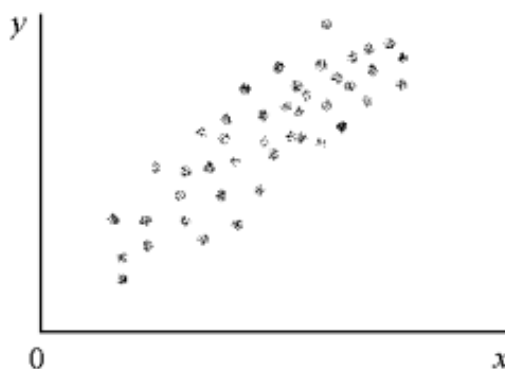


Рисунок 1.1.1. – Корреляционное поле [1]

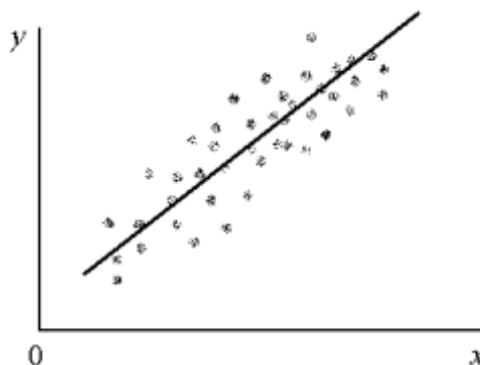


Рисунок 1.1.2. – Функция регрессии [1]

Кроме того, хотелось бы отметить следующие важные особенности корреляции:

1) Присутствие корреляционной связи между двумя переменными не обязательно указывает на их прямую взаимосвязь: зафиксированное взаимодействие может быть обусловлено влиянием других переменных, а



отношения между анализируемыми величинами могут проходить через скрытые переменные, не видимые исследователю.

2) В отличие от функциональной связи, корреляционная связь лишь указывает на тенденцию изменений одной переменной в ответ на изменения другой и позволяет сделать выводы об интенсивности взаимоотношений между ними, но не подтверждает наличие прямой причинно-следственной связи. С другой стороны, наличие корреляции дает основания для формулирования гипотезы о такой зависимости [2].

Понимание корреляции между переменными является фундаментальным шагом, который подготавливает почву для более глубокого анализа их взаимосвязей через регрессию

Регрессия – это метод, который позволяет исследователям определить, как прогнозы или средние значения выхода (outcome) модели различаются для разных объектов, определенных набором входных данных – предикторов (predictor) [3]. Было разработано множество методов для определения различных параметрических зависимостей между зависимой переменной и независимыми переменными. Эти методы обычно зависят от формы параметрической функции регрессии и ошибки в регрессионной модели. Например, линейная регрессия, логистическая регрессия, регрессия Пуассона, пробит-регрессия и т.д. Эти конкретные регрессионные модели предполагают различные регрессионные функции и условия ошибки из соответствующих распределений подчеркивания. Обобщение моделей линейной регрессии было формализовано в “обобщенной линейной модели”, и для этого требуется указать функцию связи, которая обеспечивает взаимосвязь между линейным предиктором и средним значением функции распределения [4].

Регрессионная модель часто в значительной степени зависит от выполнения лежащих в ее основе допущений. Существует три типа регрессии, а именно:

Простая линейная регрессия изучает линейную зависимость между двумя переменными, т.е. одной зависимой и одной независимой переменной.

Множественная линейная регрессия изучает линейную зависимость между одной зависимой переменной и более чем одной независимой переменной.

Нелинейная регрессия предполагает, что взаимосвязь между зависимой переменной и независимыми переменными не является линейной по параметрам регрессии [5].

Простая линейная регрессия (также известная как двумерная регрессия) — это линейное уравнение, описывающее взаимосвязь между объясняющей переменной и переменной результата, в частности, с предположением, что объясняющая переменная влияет на переменную результата, а не наоборот [6].

Основная цель линейной регрессии — определить наилучшую прямую линию (в случае одной независимой переменной) или гиперплоскость (в случае множества независимых переменных), которая наилучшим образом аппроксимирует взаимосвязи между переменными.

Простая линейная регрессия исследует связь между двумя переменными, предполагая, что изменение одной ( $X$ ) повлечет изменение другой ( $Y$ ), что может быть как увеличением, так и уменьшением  $Y$  в зависимости от  $X$ . Для изучения взаимозависимости этих переменных также используется понятие корреляции, описанное в предыдущем разделе. В отличие от корреляции, которая оценивает степень связи между переменными, регрессия дает количественную оценку этой связи, указывая на ее характер [7].

Самой ранней формой линейной регрессии был метод наименьших квадратов, который был опубликован Лежандром в 1805 году в книге «Nouvelles méthodes pour la détermination des orbites des comètes» и Гауссом в 1809 году, о чем будет рассказано чуть позже. Термин “наименьшие квадраты” происходит от термина Лежандра. Лежандр и Гаусс оба применили этот метод к задаче определения орбит тел вокруг Солнца на основе астрономических наблюдений. Эйлер безуспешно работал над той же задачей (1748). Однако стоит отметить, что Карлу Фридриху Гауссу приписывают разработку основ анализа наименьших квадратов в 1795 году в возрасте восемнадцати лет, который просто не публиковал исследования до 1809 года, в котором вышла книга «Theoria motvs

corporum coelestium in sectionibus conicis solem ambientium». Гаусс опубликовал дальнейшее развитие теории наименьших квадратов в 1821 году, включая версию хорошо известной сегодня теоремы Гаусса-Маркова, которая является фундаментальной теоремой в области общих линейных моделей [4].

Простая регрессионная модель (1.1.1) часто записывается в следующем виде:

$$y = b_0 + b_1x + \varepsilon, \quad (1.1.1)$$

где  $y$  - зависимая переменная,  $b_0$  - пересечение  $y$ ,  $b_1$  - градиент или наклон линии регрессии,  $x$  - независимая переменная, а  $\varepsilon$  -случайная ошибка [4].

Общая модель множественной линейной регрессии (1.1.2):

$$y = b_0 + b_1x_1 + \dots + b_px_p + \varepsilon, \quad (1.1.2)$$

где  $y$  - зависимая переменная,  $b_0, b_1, \dots, b_p$  - коэффициенты регрессии, а  $x_1, \dots, x_p$  - независимые переменные в модели, а  $\varepsilon$  -случайная ошибка [4].

Линейная регрессия и множественная линейная регрессия являются мощными статистическими методами, широко применяемыми в различных областях для прогнозирования и анализа данных, например, в таких как:

1) Прогнозирование ВВП: использование множественной линейной регрессии для прогнозирования ВВП страны на основе таких факторов, как инвестиции, уровень безработицы, инфляция и другие экономические показатели.

2) Оценка стоимости акций: прогнозирование цен акций на основе финансовых показателей компании, таких как прибыль на акцию, отношение цены к прибыли и объем торгов.

3) Кредитный скоринг: определение кредитоспособности клиента на основе его кредитной истории, дохода, существующих долгов и других финансовых показателей.

4) Недвижимость

Оценка стоимости объектов недвижимости: множественная линейная регрессия используется для прогнозирования стоимости недвижимости на

основе характеристик, таких как площадь, местоположение, количество комнат и доступность инфраструктуры.

#### 5) Медицина

Прогнозирование риска сердечных заболеваний: анализ факторов риска, таких как кровяное давление, уровень холестерина, индекс массы тела (ИМТ) и др., чтобы предсказать вероятность развития сердечных заболеваний у пациентов.

#### 6) Маркетинг

Анализ эффективности рекламы: оценка влияния различных рекламных кампаний на увеличение продаж или узнаваемости бренда, используя расходы на рекламу, количество показов, кликов и другие метрики.

Поведение потребителей: прогнозирование спроса на продукт на основе цены, доходов потребителей, сезонных трендов и маркетинговых кампаний.

#### 7) Занятость и труд

Прогнозирование зарплат: использование множественной линейной регрессии для определения справедливого уровня зарплаты на основе квалификации, опыта работы, образования и отрасли.

#### 8) Спорт

Оценка производительности спортсменов: прогнозирование успехов спортсменов на основе статистических данных о предыдущих достижениях, физическом состоянии и других переменных.

#### 9) Транспорт

Прогнозирование спроса на перевозки: анализ данных о пассажиропотоке для прогнозирования спроса на транспортные услуги, что помогает в оптимизации маршрутов и расписаний.

Рассмотрим подробнее метод наименьших квадратов, который был описан выше [21].

Метод наименьших квадратов — такой способ проведения регрессионной линии, чтобы сумма квадратов отклонений отдельных значений зависимой переменной от неё была минимальной [8].

Чтобы метод наименьших квадратов давал наиболее точные результаты, необходимо соблюдение следующих предположений касательно случайных отклонений:

Ожидается, что среднее значение всех случайных отклонений будет равно нулю:  $M(\varepsilon_i) = 0$ , где  $\varepsilon_i$  – дисперсия случайных отклонений

Разброс (дисперсия) случайных отклонений должен быть одинаковым (1.1.3):

$$M(\varepsilon_i) = M(\varepsilon_j) = 0, \quad (1.1.3)$$

где  $M(\varepsilon_i), M(\varepsilon_j)$  – среднее значение всех случайных отклонений;  $\varepsilon_i, \varepsilon_j$  – дисперсия случайных отклонений.

Если это условие соблюдается, говорят о гомоскедастичности, т.е. о стабильности разброса. В противном случае, когда разброс не постоянен, имеет место гетероскедастичность (дисперсия неодинакова для различных уровней независимой переменной или через время).

Любые два случайных отклонения  $\varepsilon_i$  и  $\varepsilon_j$  для  $i \neq j$  должны быть независимы друг от друга, что гарантирует отсутствие автокорреляции (значения временного ряда (или данных) независимы друг от друга).

Случайные отклонения должны быть независимы от объясняющих переменных. Это предположение обычно выполнимо, если в модели объясняющие переменные не случайны [9].

Далее перейдем к основной теме курсовой, оптимизации линейной регрессии. Начнем с поиска набора данных для анализа.

Формирование выборки является важной составляющей в методологии проведения научных исследований. Только те выборки, которые созданы на основе принципов теории вероятностей, позволяют распространять полученные в исследовании результаты на всё исследуемое множество. Статистики рассматривают добычу данных как средство построения статистической модели, т. е. закона, в соответствии с которым распределены видимые данные. Для создания вероятностных выборок необходимо иметь перечень всех элементов

генеральной совокупности, что бывает не всегда осуществимо. Поэтому начнем обсуждение с разбора тех выборок, которые формируются без учёта теории вероятностей, и рассмотрим их недостатки [10].

Первый метод — это формирование выборки по удобству. Примером может служить формирование выборки путём опроса людей, заходящих на рынок в обеденное время. Такой метод подходит для исследования определенной группы в конкретное время или если нет возможности создать более репрезентативную выборку. Обычно его используют в академической среде для опроса студентов на лекциях. Этот способ позволяет протестировать методы сбора данных, однако его результаты не могут быть экстраполированы на более широкую совокупность.

Второй метод — целенаправленная выборка, когда элементы отбираются на основе предположений о их соответствии целям исследования. Такой подход хорош для изучения специфической подгруппы в более обширной совокупности, которую трудно полностью перечислить, или для анализа нетипичных случаев для лучшего понимания стандартных ситуаций. Целенаправленную выборку используют для тестирования исследовательских инструментов или предварительного изучения темы, но результаты такого исследования нельзя экстраполировать на всю совокупность.

Третий тип — это "снежный ком", применяемый для исследования труднодоступных групп, когда невозможно составить список всех элементов генеральной совокупности. Этот метод часто используется для изучения закрытых сообществ, например, проблемы бездомности. Выборка "снежный ком" подходит для предварительного изучения темы, но также не позволяет обобщать выводы на всю совокупность.

Четвертый метод — формирования выборки, который не основывается на теории вероятностей, это квотное исследование. Этот тип выборки также стремится к репрезентативности аналогично вероятностной выборке. В процессе формирования квотной выборки используется таблица, в которой указаны доли населения по возрасту, полу или социальному классу. Составление квотной

выборки предполагает создание выборки, которая точно отражает процентное распределение определенных характеристик в общей популяции. Тем не менее, при работе с квотной выборкой могут возникнуть сложности, такие как необходимость актуализации информации в таблицах, что может потребовать значительных усилий, а также возможные отклонения в данных из-за особенностей поведения интервьюеров.

Выборки, которые научно обоснованы и чьи результаты могут быть экстраполированы на всю генеральную совокупность, должны формироваться на основе теории вероятностей. Эта теория обеспечивает представительность выборки для исследования. Существуют четыре основных типа вероятностных выборок: простая случайная, систематическая, стратифицированная и многоуровневая кластерная. Формирование каждого из этих типов выборок требует наличия полного списка всех элементов в генеральной совокупности [11].

После того как подходящий набор данных для анализа был выбран, следующий критически важный этап заключается в его предобработке. Начнем погружение в этот критически важный этап с введения в предобработку данных, раскрывая ее ключевые аспекты и методологии.

Данные, собранные из разнообразных источников и процессов для дальнейшей обработки, могут включать в себя различные неточности и дефекты, которые могут отрицательно сказаться на общем качестве данных. Вот некоторые из распространенных проблем, связанных с качеством данных:

- 1) Отсутствие данных: Отдельные атрибуты или значения отсутствуют в датасете.
- 2) Зашумленность: Присутствие некорректных данных или аномалий.
- 3) Противоречивость: Различия или конфликты в записях данных.

Для того чтобы разработать точные и надежные прогностические модели, наличие высококачественных данных является критически важным. Чтобы предотвратить ситуацию "плохие данные приводят к плохим результатам" и повысить качество данных, а таким образом и производительность моделей,

важно осуществлять тщательный контроль за состоянием данных с целью своевременного выявления и устранения возникающих проблем. Это подразумевает необходимость в проведении действий по проверке, предобработке и очистке данных, чтобы обеспечить их максимальную корректность и полноту перед началом аналитической работы.

Добавляя к вышеуказанному, улучшение качества данных может включать в себя методы нормализации данных, обработку пропущенных значений, выявление и удаление выбросов, а также устранение дубликатов записей. Эти шаги помогают минимизировать потенциальные искажения и ошибки в анализе, способствуя созданию более точных и эффективных моделей машинного обучения и аналитики [12].

После того как мы ознакомились с основными принципами предобработки данных, давайте перейдем к разбору одного из наиболее фундаментальных процессов в этой стадии - нормализации данных, который является неотъемлемой частью подготовки данных к эффективному анализу.

Нормализация данных — это общепринятая задача предобработки в машинном обучении. Данный термин означает преобразование данных в диапазон с крайними значениями. Многие алгоритмы предполагают, что все признаки находятся в единой шкале, как правило, от 0 до 1 или от -1 до 1 [13]. Значимость стандартизации возрастает, когда исследуемые переменные измеряются в сильно различающихся масштабах, например, когда речь идет о значениях, колеблющихся от микрон до миллиардов единиц [14]. Применение стандартизации позволяет устранить потенциальное искажение результатов, вызванное различиями в масштабах измерений, и повысить точность статистического анализа.

Самыми популярными методами для нормализации данных служат:

- 1) Минимакс

В данном шкалировании минимальное и максимальное значения признака используются для шкалирования значений в пределах диапазона (1.1.4) [13].



$$x'_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}, \quad (1.1.4)$$

где  $x$  – вектор признака,  $x_i$  – отдельный элемент признака  $x$ :  $x'_i$  – прошкалированный элемент [15].

## 2) Z-масштабирование

Одной из популярных альтернатив шкалированию данных методом минимакс, служит процесс стандартизации признаков (Z-масштабирование). Стандартизация заключается в трансформации данных таким образом, чтобы их распределение приближалось к стандартному нормальному распределению. Каждый элемент при данном шкалировании преобразуется по следующей формуле:

$$x'_i = \frac{x_i - \mu}{\sigma}, \quad (1.1.5)$$

где  $x'_i$  – стандартизированная форма  $x_i$ ,  $\mu$  – среднее значение,  $\sigma$  – стандартное отклонение [16].

После того как мы рассмотрели нормализацию и уравнили масштаб наших данных, следующим важным шагом в предобработке является обнаружение и обработка выбросов в данных, которые могут существенно повлиять на точность моделей.

Выбросы в данных — это наблюдения, которые сильно отличаются от других наблюдений в наборе данных и могут существенно исказить результаты анализа, в том числе привести к неверным выводам при использовании линейной регрессии. Эти аномалии могут возникать по разным причинам: из-за ошибок при сборе данных, необычных условий эксперимента, ошибок ввода или обработки данных и других факторов.

Игнорирование наличия выбросов может привести к значительным искажениям в оценках параметров модели, а также к ухудшению качества прогностической способности модели линейной регрессии.

Boxplot, также известный как "ящик с усами", является графическим инструментом, используемым в статистике для визуализации распределения

числовых данных через квартили. В основе Boxplot лежит понятие пятикратного обзора данных: минимум, первый квартиль (Q1), медиана, третий квартиль (Q3) и максимум. Эти пятикратные точки позволяют пользователям эффективно оценивать основные тенденции в данных, включая их распределение, центральную тенденцию, вариабельность и наличие выбросов. Пример данного способа визуализации приведен на рисунке 1.1.3 [14].

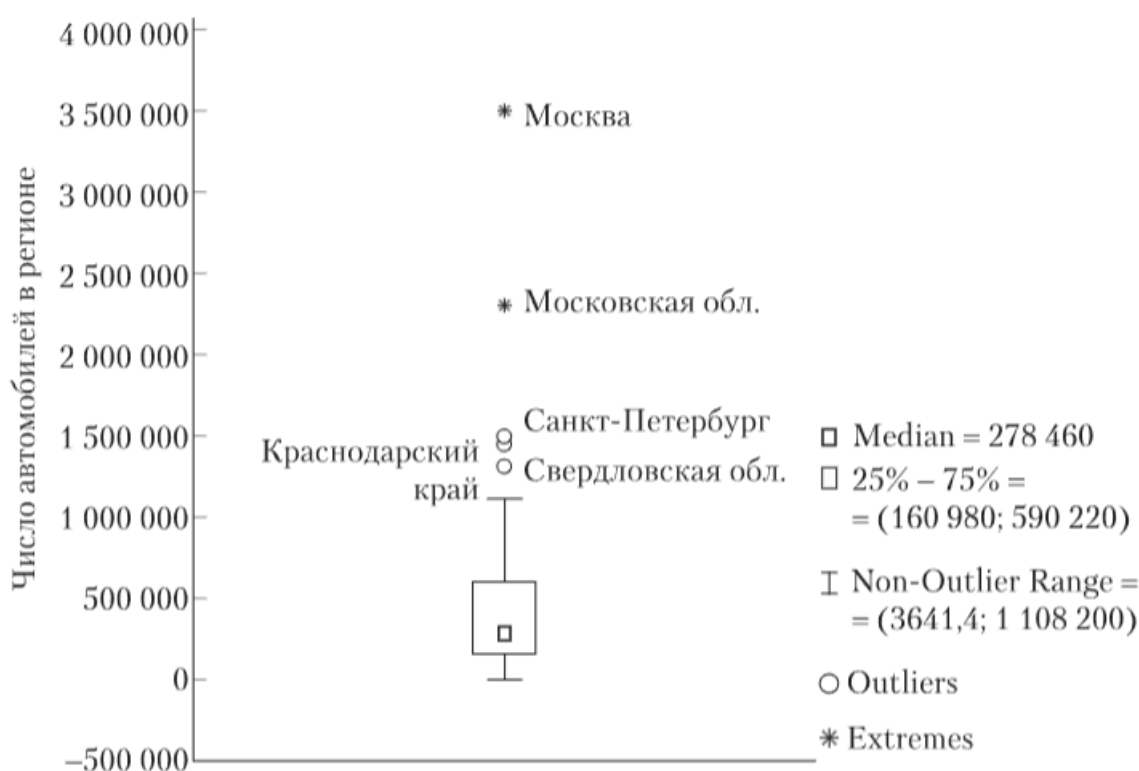


Рисунок 1.1.3. – Диаграмма «ящик с усами» вида «медиана – интерквартильных размах – размах» для данных по количеству автомобилей в регионе по регионам России [14]

Рассмотрим подробнее работу «ящика с усами». Центральная линия ящика представляет медиану набора данных. Медиана делит набор данных на две равные части и является центральной точкой данных при их упорядочивании. Ящик (интерквартильный диапазон, IQR) — это диапазон, который охватывает 50% данных, расположенных от первого квартиля (Q1, нижний край ящика) до третьего квартиля (Q3, верхний край ящика). Разница между Q3 и Q1 представляет собой IQR, который используется для определения вариативности

данных. Усы - линии, исходящие из верхней и нижней части ящика до наиболее удаленных точек данных, не считая выбросов. Длина "усов" часто рассчитывается как  $1.5 * IQR$  выше  $Q3$  и ниже  $Q1$  [13]. Точки за пределами усов считаются потенциальными выбросами. Выбросы при данном способе визуализации – точки данных, лежащие за пределами "усов", рассматриваются как выбросы и обычно обозначаются отдельными точками на графике. Любые данные, находящиеся за пределами усов, могут рассматриваться как потенциальные выбросы. Однако стоит помнить, что не все точки за пределами усов обязательно являются истинными аномалиями. Их стоит дополнительно анализировать в контексте исследования [14].

Завершая обсуждение выбросов, следующий этап пути по предобработке данных подводит к важной задаче работы с пропущенными и текстовыми данными.

Обработка пропущенных значений в датасете представляет собой важный этап предварительной подготовки данных для аналитической обработки и моделирования. Прежде чем приступить к любым манипуляциям с пропущенными значениями, критически важно понять причины, по которым данные могли не быть записаны. Это может быть связано как с техническими особенностями сбора информации, так и с особенностями объектов исследования. Понимание причин позволит выбрать наиболее адекватный подход к обработке пропусков.

Варианты методик обработки пропущенных данных весьма разнообразны и должны подбираться с учетом специфики анализа и задачи.

- 1) Удаление записей - самый радикальный способ, который предполагает исключение из анализа всех записей, содержащих хотя бы одно пропущенное значение. Хотя такой подход может существенно сократить объем данных, в некоторых случаях он остается единственно возможным для сохранения чистоты и точности исследования.

- 2) Фиктивная подстановка - метод заключается в замене пропущенных значений предварительно определенными маркерами, например, "unknown" для

категориальных переменных или "0" для числовых. Этот способ позволяет сохранить все записи в датасете, однако может ввести искажения в анализ.

3) Подстановка среднего значения - является распространенным подходом для числовых данных, когда пропущенные значения заменяются средним по столбцу. Этот метод помогает сохранить общую статистическую структуру данных, но может также скрыть вариабельность исходных данных.

4) Подстановка медианы или моды - альтернативы среднему значению, которые могут быть предпочтительнее в случаях, когда распределение данных сильно искажено или категориально.

5) Подстановка наиболее часто используемого элемента - аналогичен подстановке моды для категориальных переменных и подходит для случаев, когда существует явно выраженное доминирование одного из значений.

Для каждого набора данных и конкретной задачи исследования подход к обработке пропущенных значений следует выбирать индивидуально, учитывая потенциальное влияние каждого из методов на результаты анализа и принимаемые решения [12].

Во многих случаях более целесообразно классифицировать объекты на основе их характеристик, а не численных показателей. Такая классификация обычно выражается через присвоение объекту определенной категории по его качественным признакам, например, по полу, цвету или марке машины. Категориальные данные могут быть различны по типу. Категории, которые не предполагают никакого внутреннего порядка, называются номинальными. Примеры номинальных категорий включают:

- 1) цвета: синий, красный, зеленый;
- 2) пол: мужчина, женщина;
- 3) фрукты: банан, клубника, яблоко.

В отличие от номинальных, существуют категории с определенной последовательностью, которые называются порядковыми. В таких случаях, категории выражают некоторую степень или уровень. Например:

- 1) уровень высоты: низкий, средний, высокий;

- 2) возрастная категория: молодые, старые;
- 3) уровень согласия: согласен, нейтрален, не согласен [16].

Унификация – процесс преобразования формата данных в единый [17].

Базовый метод работы с категориальными данными — это присвоение числового значения категориям. Это связано с тем, что почти все алгоритмы машинного обучения требуют ввода числа. Использовать данный способ может быть особенно уместно в случаях, когда категории можно упорядочить по какому-то критерию, имеющему отношение к целевой переменной в модели линейной регрессии. Такой способ более подходит к второй из приведенных выше типов категориальных данных. Например, если категориальный признак представляет уровень образования (без образования, среднее, высшее), ему можно присвоить значения 0, 1 и 2 соответственно. Такая трансформация, называемая порядковым кодированием (ordinal encoding), предполагает, что между категориями существует линейная зависимость.

Однако, все же чаще информация представлена в виде текстовых значений, которые сложно или невозможно заменить на числовые показатели. Цель состоит в том, чтобы осуществить такое преобразование, которое адекватно отражает информацию, содержащуюся в категориях, включая их упорядоченность и относительные различия между ними. Если же последнее сделать невозможно, то необходимо искать альтернативные способы, например, можно посмотреть, сильное ли влияние оказывает данный параметр на исходное значение или его можно убрать в принципе [18].

После того как мы тщательно прошли этапы предобработки данных, подготовив и очистив информацию до оптимального состояния для анализа, наступает время перехода к следующей критически важной главе в нашем исследовании - выбору признаков. Этот этап так же важен, как и предыдущие, поскольку правильный выбор признаков напрямую определяет успех моделирования. На этом этапе мы обратим особое внимание на общие рекомендации к выбору признаков и исследование мультиколлинеарности, что позволит нам повысить предсказательную способность моделей.

При сборе данных зачастую лучше собирать большее количество параметров, чем может потребоваться для анализа. После предобработки данных следует исключить менее значимые признаки. Правильный выбор может значительно улучшить качество модели и сделать модель более интерпретируемой [17].

Самое главное, что хотелось бы отметить – оставшиеся после выбора признаки должны быть значимы по отношению к целевой переменной, иначе производительность модели ухудшится. Далее следует избегать мультиколлинеарность, о которой подробнее рассказывается в следующем разделе. Ведь в ее присутствии данные могут привести к большим изменениям в оценках коэффициентов, что делает модель неустойчивой.

Большое количество признаков может привести к переобучению. Модель начинает "запоминать" обучающие данные вместо того, чтобы "понимать" их. Это ведет к высокой точности на обучающих данных, но к плохой обобщающей способности на уже тестовых данных (рис. 1.1.5.). Одной из причин переобучения является неправильная интерпретация обучающих данных. Таким образом, модель выдает менее точные результаты для невидимых данных. Однако переоснащенная модель дает очень высокие показатели точности на этапе обучения. Аналогичным образом, недостаточно подготовленные модели неэффективно отражают взаимосвязь между входными и выходными данными, потому что они слишком просты. В результате недостаточно подготовленная модель плохо работает даже с обучающими данными [19].

Разработка признаков (feature engineering) требует углубления в задачу и в ее проблемную сферу, а также большого опыта в машинном обучении. Из-за всего вышеизложенного в совокупности следуют не пренебрегать правильным выбором признаков (feature selection) [17].

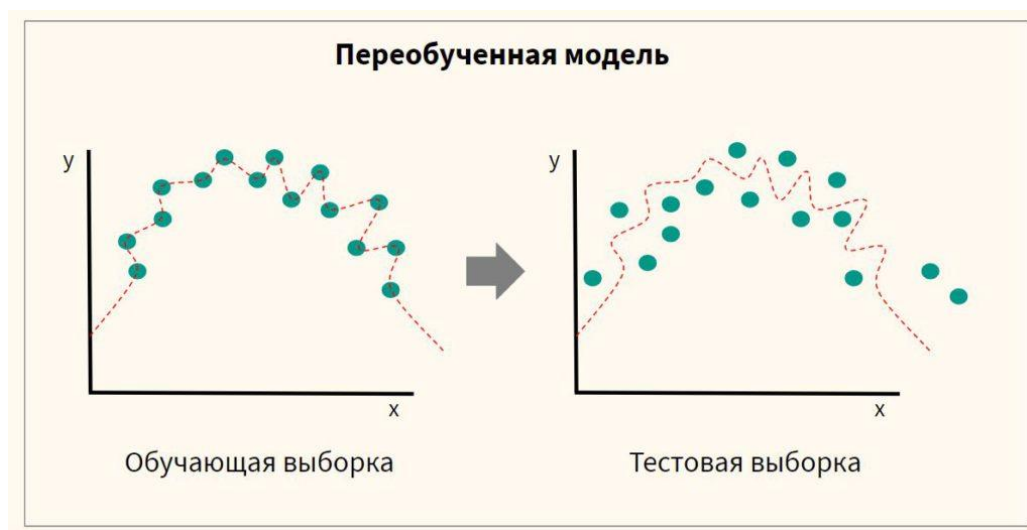


Рисунок 1.1.5. – Переобучение на обучающей и на тестовой выборке [20]

Исследовав общие рекомендации по выбору признаков, которые подчеркивают важность взвешенного подхода к их отбору, теперь перейдем к другому аспекту, который влияет на выбор признаков – мультиколлинеарности.

В некоторых случаях в данных может проявляться явление, известное как мультиколлинеарность, когда два или более параметра показывают высокую степень корреляции.

Чтобы определить степень взаимосвязи между двумя количественными переменными, используют коэффициент корреляции Пирсона. Этот коэффициент показывает, как одна переменная связана с другой и может изменяться от +1 (положительная корреляция) до -1 (отрицательная корреляция), где 0 означает отсутствие связи. Если две переменные имеют достаточно высокий коэффициент корреляции, можно с некоторой точностью предсказать значение одной переменной, исходя из значения другой.

Считается, что коллинеарность присутствует в данных, когда коэффициент корреляции между двумя переменными превышает 0,7 или составляет менее -0,7. Исключение одного из линейно связанных параметров из модели помогает устранить коллинеарность, причем отсев происходит с учетом наименьшей связи с другими параметрами, при достаточно сильной корреляции с результатом.

Для определения коллинеарности переменных создается матрица парных коэффициентов корреляции, изучающая взаимосвязь каждого признака с результативным и друг с другом.

Когда в данных присутствует мультиколлинеарность, для обеспечения адекватности модели можно провести отбор признаков, исключая нерелевантные переменные, демонстрирующие линейную зависимость или высокую корреляцию с другими. Также применяется линейная регрессия с регуляризацией L1 или L2, что способствует борьбе с мультиколлинеарностью [22, 23].

После тщательного выбора и анализа признаков, необходимых для создания наших регрессионных моделей, мы переходим к следующей важной стадии нашего исследования - анализу полученных моделей. Далее мы сосредоточим внимание на оценке эффективности и качества линейных регрессионных моделей, используя различные метрики качества. Это позволит нам не только понять, насколько хорошо наши модели работают, но и выявить возможные направления для их дальнейшего улучшения.

В этом разделе курсовой работы мы сосредоточим внимание на метриках качества линейной регрессии. Эти метрики играют критически важную роль в оценке эффективности регрессионных моделей, поскольку предоставляют объективную количественную оценку точности предсказаний модели. От выбора и правильного применения этих метрик зависит не только оценка текущего качества модели, но и понимание путей её улучшения. Далее рассмотрим самые популярные метрики подробнее:

Среднеквадратичная ошибка (MSE) (формула 1.1.6): одна из самых популярных мер оценки модели линейной регрессии. В основном применяется, чтобы выделить большие ошибки. Однако использование только данной метрики может негативно сказаться на итоговом результате, так как возведение одной ошибки в квадрат уже оказывает огромное влияние на значение среднеквадратичную ошибку.



$$MSE = \frac{1}{m} \sum_{i=1}^m |a_i - y_i|^2, \quad (1.1.6)$$

где  $m$  – количество наблюдений,  $a_i$  – фактическое значение зависимой переменной для  $i$ -го измерения,  $y_i$  – значение зависимой переменной для  $i$ -го измерения [24].

Корень из среднеквадратичной ошибки (RMSE) (формула 1.1.7): представляет собой корень из MSE. Помогает понять размер ошибки в тех же единицах измерения, что и данные. Однако с MSE более проще работать и анализировать модель. Кроме того, большие ошибки оказывают непропорциональное влияние [25].

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{m} \sum_{i=1}^m |a_i - y_i|^2}, \quad (1.1.7)$$

где  $m$  – количество наблюдений,  $a_i$  – фактическое значение зависимой переменной для  $i$ -го измерения,  $y_i$  – значение зависимой переменной для  $i$ -го измерения [25].

Среднеквадратичная ошибка в процентах (MSPE) (формула 1.1.8): описывает, насколько в среднем ошибаются прогнозы в процентном соотношении. Однако, если значение зависимой переменной оказывается 0, то возникает неопределенность [25].

$$MSPE = \frac{100}{m} \sum_{i=1}^m \left( \frac{a_i - y_i}{a_i} \right)^2, \quad (1.1.8)$$

где  $m$  – количество наблюдений,  $a_i$  – фактическое значение зависимой переменной для  $i$ -го измерения,  $y_i$  – значение зависимой переменной для  $i$ -го измерения [25].

Средняя абсолютная ошибка (MAE) (формула 1.1.9): это средняя величина ошибок без учета их направления. Является линейной оценкой, поэтому четче видно разницу между значениями, легче определить качество модели. Все это

сказалось на том, что данная метрика, как и MSE, является одной из самых популярных [25].

$$MAE = \frac{1}{m} \sum_{i=1}^m |a_i - y_i|, \quad (1.1.9)$$

где  $m$  – количество наблюдений,  $a_i$  – фактическое значение зависимой переменной для  $i$ -го измерения,  $y_i$  – значение зависимой переменной для  $i$ -го измерения [25].

Средняя абсолютная ошибка в процентах (MAPE) (формула 1.1.10): измеряет средний процент ошибки прогноза относительно реальных значений. Это помогает понять, насколько велика ошибка в контексте размера данных [25].

$$MAPE = 100\% * \frac{1}{m} \sum_{i=1}^m \frac{|y_i - a_i|}{|y_i|}, \quad (1.1.10)$$

где  $m$  – количество наблюдений,  $a_i$  – фактическое значение зависимой переменной для  $i$ -го измерения,  $y_i$  – значение зависимой переменной для  $i$ -го измерения [25].

Симметричная средняя абсолютная процентная ошибка (SMAPE) (формула 1.1.11): аналогично MAPE, но более устойчивая к экстремальным значениям, так как учитывает размер как прогнозируемых, так и фактических значений в знаменателе. Одна из проблем – неполная симметрия, т.е. метрика выдаст разные значения при вычитании предсказанного и действительного и наоборот [25].

$$SMAPE = 100\% * \frac{1}{m} \sum_{i=1}^m \frac{|y_i - a_i|}{(y_i - a_i) / 2}, \quad (1.1.11)$$

где  $m$  – количество наблюдений,  $a_i$  – фактическое значение зависимой переменной для  $i$ -го измерения,  $y_i$  – значение зависимой переменной для  $i$ -го измерения [25].

Средняя относительная ошибка (MRE) (формула 1.1.12): иногда называют средней относительной абсолютной ошибкой. Она показывает, насколько в среднем относительно действительных значений ошибается прогноз. Это

помогает понять масштаб ошибки относительно величины реальных данных. Исключение – нельзя применять, если фактическое значение равно 0 [25].

$$\text{MRE} = \frac{1}{n} \sum_{i=1}^n \frac{|a_i - y_i|}{|a_i|}, \quad (1.1.12)$$

где  $n$  – количество наблюдений,  $a_i$  – фактическое значение зависимой переменной для  $i$ -го измерения,  $y_i$  – значение зависимой переменной для  $i$ -го измерения [25].

В рамках исследования метрик качества линейных регрессионных моделей были рассмотрены ключевые инструменты оценки модели. Эти метрики играют важнейшую роль в оценке точности и эффективности моделей линейной регрессии, позволяя исследователям и аналитикам оценить, насколько хорошо модель предсказывает зависимую переменную. Несмотря на детальное изучение упомянутых показателей, следует подчеркнуть, что рассматриваемый обзор не охватывал всего спектра существующих метрик оценки. Существуют и другие метрики, включая, среднеквадратичная логарифмическая ошибка,  $R^2$  и многие другие, каждая из которых может предоставлять уникальные взгляды на свойства моделей и имеет свои недостатки и преимущества.

## 1.2. Построение полиномов первой и второй степени, аппроксимирующие результаты эксперимента

Когда между данными прослеживается явная связь, можно визуализировать процесс ручной настройки линии. В реальности же регрессионная линия представляет собой оценочную модель, которая стремится минимизировать сумму квадратов разностей между наблюдаемыми и предсказанными значениями, что также известно как остаточная сумма квадратов или RSS (формула 1.2.1):

$$RSS = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^n (Y_i - \hat{b}_0 - \hat{b}_1 X_i)^2, \quad (1.2.1)$$

$\hat{b}_0$  и  $\hat{b}_1$  являются значениями, которые как раз и минимизируют RSS,  $\hat{b}_1$  – наклон,  $\hat{b}_0$  – смещение в уравнении  $y = \hat{b}_0 + \hat{b}_1 X$  [7, 26]

Очевидно, что уменьшение погрешностей по их абсолютному значению способствует более точному представлению зависимости между экспериментальными значениями  $x$  и  $y$  с помощью линейной функции  $y = kx + b$ . Суть метода наименьших квадратов заключается в выборе таких коэффициентов  $k$  и  $b$ , чтобы минимизировать общую сумму квадратов этих погрешностей:

$$S = \delta_1^2 + \delta_1^2 + \dots + \delta_n^2 = \sum_{i=1}^n \delta_i^2 = \sum_{i=1}^n (y_i - (kx_i + b))^2 \rightarrow \min, \quad (1.2.2)$$

где  $n$  – количество данных (+ в формулах текст другого шрифта),  $\delta_i$  – квадрат погрешности,  $x_i$  и  $y_i$  (буквы тоже другого шрифта) – заданные числа,  $k$  и  $b$  – коэффициенты,  $S$  – сумма, функция двух переменных  $k$  и  $b$ :  $S = S(k, b)$  [27].

Исследуем функцию двух переменных  $k$  и  $b$  на экстремум:

$$\begin{cases} \frac{\partial S}{\partial b} = 0, \\ \frac{\partial S}{\partial k} = 0; \end{cases} \quad (1.2.3)$$

где  $n$  – количество данных,  $k$  и  $b$  – коэффициенты,  $S$  – сумма, функция двух переменных  $k$  и  $b$ :  $S = S(k, b)$  [27].

$$\frac{\partial S}{\partial k} = 2 \sum_{i=1}^n (y_i - (kx_i + b))(-x_i) = -2 \sum_{i=1}^n (y_i - (kx_i + b))x_i, \quad (1.2.4)$$

где  $n$  – количество данных,  $x_i$  и  $y_i$  – заданные числа,  $k$  и  $b$  – коэффициенты,  $S$  – сумма, функция двух переменных  $k$  и  $b$ :  $S = S(k, b)$  [27].

$$\frac{\partial S}{\partial b} = 2 \sum_{i=1}^n (y_i - (kx_i + b))(-1) = -2 \sum_{i=1}^n (y_i - (kx_i + b)), \quad (1.2.5)$$

где  $n$  – количество данных,  $x_i$  и  $y_i$  – заданные числа,  $k$  и  $b$  – коэффициенты,  $S$  – сумма, функция двух переменных  $k$  и  $b$ :  $S = S(k, b)$  [27].

$$\begin{cases} -2 \sum_{i=1}^n (y_i - (kx_i + b))x_i = 0, \\ 2 \sum_{i=1}^n (y_i - (kx_i + b)) = 0; \end{cases} \quad (1.2.6)$$

где  $n$  – количество данных,  $x_i$  и  $y_i$  – заданные числа,  $k$  и  $b$  – коэффициенты [27].

$$\begin{cases} k \sum_{i=1}^n x_i^2 + b \sum_{i=1}^n x_i = \sum_{i=1}^n y_i x_i, \\ k \sum_{i=1}^n x_i + bn = \sum_{i=1}^n y_i; \end{cases} \quad (1.2.7)$$

где  $n$  – количество данных,  $x_i$  и  $y_i$  – заданные числа,  $k$  и  $b$  – коэффициенты [27].

Из данной системы, получая коэффициенты  $k$  и  $b$ , можем составить уравнение линейной регрессии. Пример построения прямой приведен на рисунке 1.2.1., однако вместо коэффициента  $b$  используется  $a$ , что не меняет смысл применения формул.

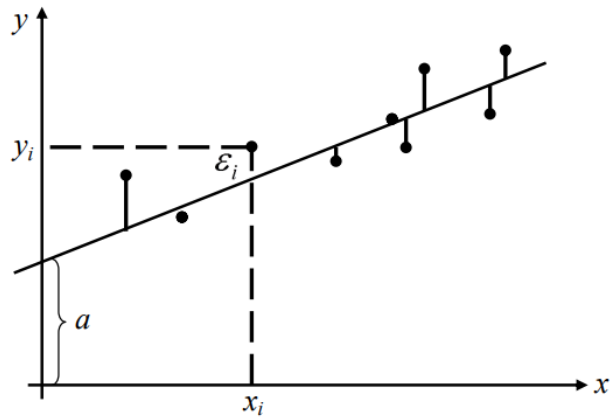


Рисунок 1.2.1. – Построение полинома первой степени [9]

Тот факт, что функция  $S = S(k, b)$  в найденной точке  $(k, b)$  имеет именно минимум, устанавливается с помощью частных производных второго порядка.

$$\frac{\partial^2 S}{\partial k^2} = -2 \sum_{i=1}^n (-x_i)x_i = 2 \sum_{i=1}^n (x_i)^2, \quad (1.2.8)$$

где  $n$  – количество данных,  $x_i$  – заданное число,  $k$  – коэффициент,  $S$  – сумма, функция двух переменных  $k$  и  $b$ :  $S = S(k, b)$  [27].

$$\frac{\partial^2 S}{\partial b^2} = -2 \sum_{i=1}^n (-1) = 2n, \quad (1.2.9)$$

где  $n$  – количество данных,  $b$  – коэффициент,  $S$  – сумма, функция двух переменных  $k$  и  $b$ :  $S = S(k, b)$  [27].

$$\frac{\partial^2 S}{\partial k \partial b} = -2 \sum_{i=1}^n (-x_i) = 2 \sum_{i=1}^n x_i, \quad (1.2.10)$$

где  $n$  – количество данных,  $x_i$  – заданное число,  $k$  и  $b$  – коэффициенты,  $S$  – сумма, функция двух переменных  $k$  и  $b$ :  $S = S(k, b)$  [27].

Вычислим  $\Delta$ :

$$\Delta = \frac{\partial^2 S}{\partial k^2} \cdot \frac{\partial^2 S}{\partial b^2} - \left( \frac{\partial^2 S}{\partial k \partial b} \right)^2, \quad (1.2.11)$$

где  $k$  и  $b$  – коэффициенты,  $S$  – сумма, функция двух переменных  $k$  и  $b$ :  $S = S(k, b)$  [27].

$$\Delta = 4n \sum_{i=1}^n (x_i)^2 - \left( 2 \sum_{i=1}^n x_i \right)^2 = 2 \sum_{i=1}^n \sum_{j=1}^n (x_i - x_j)^2, \quad (1.2.12)$$

где  $n$  – количество данных,  $x_i$  – заданное число,  $k$  и  $b$  – коэффициенты,  $S$  – сумма, функция двух переменных  $k$  и  $b$ :  $S = S(k, b)$  [27].

«Очевидно,  $\Delta > 0$ , следовательно, в найденной точке  $(k, b)$  функция  $S = S(k, b)$  имеет экстремум; а так как  $\frac{\partial^2 S}{\partial k^2} > 0$ , то, согласно достаточному условию экстремума функции двух переменных, в точке  $(k, b)$  функция имеет минимум» [27].

Если отношение между данными, полученными в ходе эксперимента, напоминает квадратичную (или параболическую) зависимость, то основная задача заключается в вычислении коэффициентов  $a_2, a_1, a_0$ , чтобы сформировать уравнение вида  $y = a_2 x^2 + a_1 x + a_0$ . Для того, чтобы найти эти коэффициенты, необходимо решить определенную систему уравнений:

$$\begin{cases} na_0 + a_1 \sum_{i=1}^n x_i + a_2 \sum_{i=1}^n x_i^2 = \sum_{i=1}^n y_i, \\ a_0 \sum_{i=1}^n x_i + a_1 \sum_{i=1}^n x_i^2 + a_2 \sum_{i=1}^n x_i^3 = \sum_{i=1}^n x_i y_i, \\ a_0 \sum_{i=1}^n x_i^2 + a_1 \sum_{i=1}^n x_i^3 + a_2 \sum_{i=1}^n x_i^4 = \sum_{i=1}^n x_i^2 y_i; \end{cases} \quad (1.2.13)$$

где  $n$  – количество данных,  $x_i, y_i$  – заданные числа,  $a_2, a_1, a_0$  – коэффициенты [27].

Ясно, что после определения типа аппроксимирующего уравнения, задача упрощается до нахождения значений его параметров.

## **2. Практическое применение и реализация алгоритмов оптимизации линейной регрессии для улучшения точности прогнозов стоимости недвижимости**

В данном разделе целью является обработка массива данных с использованием множественной регрессии для минимизации ошибки предсказания стоимости недвижимости в различных городах России на практике. Для достижения этой цели будет поставлена общая задача, которая разделена на подзадачи.

Мы подробно рассмотрим методы и техники, направленные на повышение эффективности предсказательной модели. Исходя из того, что основные препятствия для достижения максимальной точности прогнозов связаны с наличием выбросов, неправильной обработкой текстовых данных, проблемами мультиколлинеарности и необходимостью нормализации данных, наша задача заключается в разработке и внедрении комплекса оптимизационных мер.

Прежде всего, основное внимание будет уделено методам идентификации и обработки выбросов, так как они могут существенно искажать результаты модели. Для выявления выбросов предложим использовать как статистические методы, например, IQR (межквартильный размах), так и визуальные средства, включая диаграммы.

Далее, разберём процесс нормализации данных, необходимый для устранения возможных искажений, вызванных различными масштабами признаков. Обоснуем выбор метода нормализации, будь то минимаксное масштабирование или стандартизация, и продемонстрируем его практическое применение на наших данных.

Важная часть оптимизации модели связана с обработкой текстовых данных. Здесь мы опишем подходы к преобразованию текста в числовые эквиваленты.

Также, нельзя игнорировать проблему мультиколлинеарности, которая возникает, когда два или более предиктора сильно коррелируют друг с другом, что ведёт к нестабильности модели. После чего избавимся от таких параметров, используя диаграмму мультиколлинеарности.



## **2.1. Постановка задачи, обработки массива данных с применением модели множественной регрессии**

Оптимизация линейной регрессии, включая методы нормализации данных, устранения выбросов и мультиколлинеарности, широко применяется в самых разных сферах для повышения качества моделирования и точности прогнозов. Эти методы используются в финансах для анализа рыночных трендов, в маркетинге для оценки эффективности рекламных кампаний, в здравоохранении для прогнозирования исходов лечения, в недвижимости для оценки стоимости объектов и в экономике для анализа влияния экономических показателей. Эффективное использование этих методов обеспечивает более глубокое понимание данных и помогает в принятии обоснованных решений, так все это в совокупности уменьшает ошибки предсказания.

Чтобы создать постановку задачи для множественной линейной регрессии, сначала нужно определить цель исследования - какую зависимую переменную мы хотим предсказать или объяснить на основе ряда независимых переменных. Включение в анализ релевантных и доступных данных является ключевым, поэтому следует собрать достаточный датасет, в котором присутствуют как зависимая переменная, так и все заинтересовавшие нас независимые переменные. Затем необходимо проверить эти данные на наличие выбросов, пропущенных значений и мультиколлинеарности между независимыми переменными, потому что эти факторы могут влиять на точность и надёжность получаемой модели. После чистки и подготовки данных можно построить модель множественной линейной регрессии, где зависимая переменная будет представлена как линейная комбинация независимых. Важно также определить критерии оценки эффективности модели, чтобы можно было измерить, насколько хорошо наша модель объясняет вариативность зависимой переменной. Последний этап включает анализ результатов, интерпретацию значимости независимых переменных и, при необходимости, доработку модели для повышения её точности и предсказательной способности.

Для практической части курсовой работы была поставлена глобальная задача по достижению уменьшения ошибки предсказания стоимости аренды недвижимости в городах России, которая была разбита на множество подзадач.

Они включают в себя предобработку данных (заполнение пропущенных данных, удаление строк с множеством неизвестных, обработка выбросов и аномалий в данных, работа с текстовыми данными, нормализация числовых признаков и многое другое), оценку мультиколлинеарности (провести анализ мультиколлинеарности между переменными, исключить коррелированные признаки и осуществить отбор наиболее значимых факторов для построения модели), оценку качества модели (по различным популярным метрикам для множественной линейной регрессии).

## 2.2. Решение поставленной задачи

Целью данного пункта курсовой работы является решение поставленной задачи по оптимизации линейной регрессии с использованием различных методов и подходов. Задача включает в себя отбор наиболее значимых переменных, оценку качества построенной модели, минимизацию ошибок прогнозирования и т.д. В данной части работы будут исследованы теоретические основы линейной регрессии, методы оценки ее параметров, а также подходы к оптимизации модели с целью повышения точности и надежности прогнозов.

Для начала работы над поставленной задачей необходимо для упрощения работы и уменьшения количества кода было создать класс и импортировать библиотеки.

Pandas — это мощная библиотека на языке Python, предназначенная для анализа данных, обработки и анализа структур данных, таких как Series и DataFrame, позволяя удобно выполнять операции импорта, экспорта, очистки и сложных аналитических преобразований.

NumPy — это фундаментальная библиотека для научных вычислений в Python, предоставляющая поддержку многомерных массивов и матриц, а также большой набор функций для работы с этими данными, что делает её незаменимым инструментом в области линейной алгебры, математики и инженерных расчётов.

Seaborn — это библиотека визуализации данных для Python, построенная на основе matplotlib, предназначенная для создания информативных и привлекательных статистических графиков, облегчая визуализацию сложных данных с помощью высокоуровневого интерфейса.

sklearn.preprocessing используемой для предварительной обработки данных в Python. MinMaxScaler масштабирует и нормализует данные, приводя их к диапазону между заданным минимальным и максимальным значениями (обычно от 0 до 1), что полезно для алгоритмов, чувствительных к масштабу переменных. StandardScaler стандартизирует данные, удаляя среднее и масштабируя их к

единичной дисперсии, что помогает в случаях, когда различные признаки имеют разные масштабы, делая алгоритмы нечувствительными к масштабу признаков.

`matplotlib.pyplot` является модулем в библиотеке `Matplotlib` для Python, который предоставляет функциональность построения графиков и визуализации данных в образе интерфейса. Импортируя его под псевдонимом `plt`, получаем доступ к удобным функциям для создания широкого спектра статических, анимированных и интерактивных графиков и диаграмм, обеспечивая эффективное средство для анализа и представления данных.

`from sklearn.model_selection import train_test_split` импортирует функцию из библиотеки `scikit-learn`, предназначенную для разделения наборов данных на обучающие и тестовые подвыборки, что является ключевым этапом в процессе обучения и тестирования моделей машинного обучения.

`from sklearn.linear_model import LinearRegression` позволяет использовать линейную регрессию, один из основных методов в машинном обучении для прогнозирования численных значений на основе предыдущих наблюдений, которая будет использоваться в данной курсовой работе.

`from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error, mean_squared_log_error` предоставляет набор метрик для оценки точности моделей машинного обучения, включая среднюю абсолютную ошибку, среднеквадратичную ошибку, среднюю абсолютную процентную ошибку и среднеквадратичную логарифмическую ошибку, каждая из которых имеет своё применение в зависимости от конкретной задачи и характеристик данных.

Библиотека `time` в Python используется для работы с временем, включая измерения времени выполнения кода, а также для получения текущего времени. Это полезный модуль для мониторинга производительности программ или создания задержек в выполнении программ, что может помочь, чтобы понять скорость выполнения блока программы.

С другой стороны, `import itertools` предоставляет доступ к модулю `itertools`, который содержит функции для создания итераторов, предназначенных для

эффективного перебора данных. Модуль `itertools` включает в себя мощные инструменты, такие как комбинации (`combinations`), перестановки (`permutations`) и бесконечные итераторы, что делает его очень полезным для алгоритмических задач, обработки данных и для реализации сложных циклических конструкций без необходимости писать большой объём кода.

После импорта библиотек, создадим класс `Predict_price`, в котором будем описывать методы, которые понадобятся при реализации алгоритмов оптимизации и обучения модели. Первый метод `__init__` может считывать csv файл, а также создавать список столбцов датасета. Далее `pop_column` позволяет удалить столбец. `dataset` позволяет вернуть таблицу данных. `describe_dataset` возвращает описание датасета. `unique_of_column` возвращает уникальные значения для указанной колонки набора данных. Это может быть полезно для выявления различных категорий или значений в колонке. `delete_nan_rows` заменяет значения в наборе данных, эквивалентные 0, '0', NaN (`np.nan`), None или 'Не заполнено', на NaN, а затем удаляет строки, которые содержат количество NaN значений больше или равно `count_nan`. Это помогает очистить данные от неполных записей. `popular_name` вернет наиболее часто встречающееся значение в указанной колонке. Это может использоваться для определения самых распространённых категорий или значений. `change_nan_to_popular` заменяет все пропущенные значения (NaN) в указанной колонке на наиболее часто встречающееся значение. Это стратегия обработки пропусков в данных, когда удаление строк с пропусками может привести к значимой потере данных. `drop_row_by_word` удаляет все строки, в которых в указанной колонке содержится определённое слово `word`. `remove_outliers` удаляет выбросы из указанной колонки на основе межквартильного размаха (IQR). Выбросы определяются как значения, находящиеся за пределами 1.5 IQR от нижнего и верхнего квартилей. `text_to_numbers` заменяет текстовые значения в указанной колонке на числовые согласно предоставленному словарю `dict`. Это важно для подготовки категориальных данных к обработке численными методами машинного обучения. `scaler` масштабирует данные в указанной колонке,

используя или `MinMaxScaler`, или `StandardScaler`, в зависимости от параметра `scaler_type`, что необходимо для нормализации данных перед использованием в большинстве алгоритмов машинного обучения. `fractions_into_numbers` преобразует дробные и целые числа из текстового представления в числовое в указанной колонке. Сначала находится наиболее часто встречающееся значение (модальное значение) в колонке. Затем, в зависимости от того, обнаружен ли в этом значении символ точки (представление десятичной дроби) или слеш (представление обыкновенной дроби), осуществляется разбиение строки на числитель и знаменатель, их преобразование в числа и вычисление результата деления с округлением до трех знаков после запятой. Далее, для каждого значения в колонке аналогичным образом вычисляется десятичное представление, и в случае обнаружения значений, эквивалентных нулю или NaN, они заменяются на вычисленное модальное значение. `fillna_row` заполняет отсутствующие (NaN) значения в указанной колонке средним арифметическим всех ненулевых значений колонки. `sb_pairplot` вызывает функцию `pairplot` из библиотеки `seaborn` для построения матрицы графиков, которые показывают отношения между всеми парами колонок в датасете. Это полезно для быстрого визуального анализа многомерных данных и обнаружения возможных корреляций между колонками. `training` этот метод позволяет обучить линейную регрессионную модель. Он принимает названия колонок независимых переменных (`X_columns`), зависимой переменной (`y_column`) и размер тестовой выборки (`size_of_test`) как долю от всего набора данных. В методе используется функция `train_test_split` из библиотеки `sklearn.model_selection` для разделения данных на обучающие и тестовые наборы с заданным размером тестовой выборки и параметром `random_state=123`, который гарантирует, что выборка будет одной и той же. Затем создается объект линейной регрессии, на котором метод `fit` обучает модель на обучающем наборе данных. `coefficient` возвращает датафрейм `pandas`, содержащий коэффициенты линейной регрессии модели для каждой независимой переменной, которые представляют собой важность каждой переменной для предсказания зависимой переменной. `price_pred` метод

предназначен для получения прогнозных значений модели на тестовом наборе данных. Результаты фактических и прогнозируемых значений собираются в датафрейм `pandas` для удобства сравнения. `print_errors` выводит в консоль значения основных метрик качества модели, включая среднеквадратичную ошибку (Mean Squared Error), среднюю абсолютную ошибку (Mean Absolute Error) и среднюю абсолютную процентную ошибку (Mean Absolute Percentage Error). Эти метрики помогают оценить, насколько точно модель предсказывает данные. `return_errors` возвращает значения основных метрик качества модели в виде списка. `return_test_split` возвращает объекты разделения данных на обучающие и тестовые наборы, включая `X_train`, `X_test`, `y_train`, и `y_test`, что может быть полезно для дальнейшего анализа вне этого класса.

Создадим объект класса `price_apart1`, затем выведем таблицу с помощью метода `dataset`. Далее посмотрим, при каком количестве `nan` в строке будет наименьшая ошибка. Понятно, что наилучший результат можно добиться только при количестве неизвестных данных равном нулю, однако тогда не будет пропусков в данных. Поэтому временно уберем `random_state=123` в методе `training`. Хотелось бы сразу заметить, что многие блоки программ запускались множество раз, чтобы понять наиболее подходящий вариант ответа. `global_count_nan_in_line` предназначена для хранения информации о количестве пропусков в строке, которое создает наименьшую ошибку. `global_sum_errors` используется как сравнительная метрика для определения наилучшего набора данных после обработки пропусков. После чего идет циклическое удаление строк с пропусками: рассматривается различное количество допустимых пропусков в строке от 0 до 10. Сначала инициализируется объект `price_apart1` класса `Predict_price` с загрузкой данных из CSV файла. Удаление нерелевантных столбцов из набора данных удаляются столбцы, которые были признаны несущественными для анализа и дальнейшего использования. Это могут быть столбцы с координатами, именами агентств и датами. Массив с названиями колонок: создаётся список с названиями колонок, которые будут использоваться после предобработки данных. Удаление строк с пропусками: применяется метод

`delete_nan_rows`, который удаляет строки с количеством пропусков больше заданного значения в цикле.

Обработка и преобразование признаков. Будем идти постепенно, корректируя каждый столбец, количество пропущенных данных в строке и размер тестовой выборки. Столбец `type` преобразуется из текстовых значений в числовые, удаляются строки с редким значением признака, пропуски заменяются на наиболее частое значение, значение «Специализированный жилищный фонд» отбрасывается, так как в датасете только одна позиция. Для столбцов `rooms`, `kitchen_area`, `living_area` и `area` пропуски заменяются на среднее значение, после чего данные масштабируются с помощью минимаксного масштабирования (`MinMaxScaler`). Позже будем рассматривать, какое масштабирование наилучшее. `level` - обработка дробных значений и замена неизвестных значений на наиболее встречаемое. Столбцы, в которых содержатся текстовые данные, заменим параметры на числовые эквиваленты для начала примерно, а позже в программе с помощью циклов. Однако уже на данном этапе каждый из этих шагов улучшает качество данных за счет приведения их к формату, который может быть более эффективно использован в моделях машинного обучения, поскольку модели работают лучше, когда данные представлены числами, пропуски заполнены адекватным образом, и признаки масштабированы для обеспечения согласованности между различными значениями. Важно отметить, что фрагмент кода является частью более крупной процедуры, где результаты каждой итерации цикла по `count_nan_in_line` сравниваются для выбора наилучшего варианта обработки данных (используя `global_sum_errors` как критерий). В конце получаем `global_count_nan_in_line`, то есть наилучшее значение количества пропущенных в данных в строке. Практически в каждом блоке кода используется библиотека `time` для вывода времени работы программы. Наилучшее количество пропусков в данных оказалось меньше или равным пяти.

В следующих блоках начнем обрабатывать каждый признак отдельно, так как язык программирования Python не поддерживает более 20 вложенных циклов.



Начнем с параметра `type` (тип помещения). Сначала объявляются глобальные переменные для хранения оптимальных числовых значений (`global_change_type_flat` – Квартира, `global_change_type_residential_building` – Жилой дом блокированной застройки, `global_change_type_apartment_building` – Многоквартирный дом в `type`), соответствующих различным типам жилья. Затем проводится замена пропущенных значений в столбце «`type`» на наиболее часто встречающееся значение с последующим удалением строк, относящихся к "Специализированному жилищному фонду", что упрощает анализ.

Для того чтобы преобразовать текстовые значения типов недвижимости в числовые, создается копия столбца «`type`», которая будет использоваться для последующих откатов к исходным значениям. Для каждого типа недвижимости генерируются числовые значения в определенных диапазонах, после чего происходит непосредственное преобразование текстовых значений в числовые по заранее определенным правилам.

После преобразования типов недвижимости в числовые данные проводится этап обучения модели множественной линейной регрессии, где в качестве предикторов выступают различные характеристики недвижимости, а целевой переменной является цена. На основе результатов обучения и анализа ошибок модели выбираются оптимальные числовые кодировки для различных типов недвижимости, соответствующие наименьшим значениям ошибок MSE (среднеквадратическая ошибка) и MAE (средняя абсолютная ошибка).

Такой подход позволяет не только автоматически изменять параметры модели для достижения наилучших результатов прогнозирования, но и обеспечивает гибкость экспериментирования с различными числовыми кодировками текстовых категорий, чтобы уменьшить ошибку и улучшить предсказание модели. Поэтому будем использовать и для последующих замен текстовых данных на числовые.

Теперь рассмотрим, какое масштабирование (`StandartScaler` или `MinMaxScaler`) использовать для различных признаков. В этом фрагменте кода реализуется процесс подбора наилучших масштабировщиков для признаков в

задаче прогнозирования цены на недвижимость. Используя два типа масштабировщиков – MinMaxScaler и StandardScaler, цель состоит в том, чтобы определить, какой из них лучше справляется с масштабированием каждого конкретного признака для достижения наименьшей ошибки в прогнозе.

Сначала объявляются глобальные переменные для каждого признака, которые будут использоваться для хранения наиболее подходящего масштабировщика на основе результатов обучения модели. В данном блоке программы будем подбирать масштабировщик для следующих параметров: rooms (количество комнат), kitchen\_area (площадь кухни), living\_area (жилая площадь), area (общая площадь), price (цена), build\_year (год постройки здания), areaRating (рейтинг местности), remoute\_from\_center (удаленность от центра).

Вложенные циклы перебирают различные комбинации масштабировщиков для признаков. Для каждой уникальной комбинации масштабировщиков применяется метод `price_apart1.scaler`, который, изменяет масштаб указанного признака в соответствии с заданным масштабировщиком, выбор которого зависит от ошибки после обучения модели, которое происходит на каждом этапе цикла. Это позволяет адаптировать данные для обучения модели, сокращая потенциальные искажения, вызванные различиями в масштабах признаков.

Чтобы определить наилучшее значение для переменной `material` (материал) при прогнозировании цены на жилье, выполняется перебор различных числовых значений для вариантов: дерево (`wood`), монолит (`monolith`), панель (`panel`), блоки (`blocks`), кирпич (`brick`). В этом процессе используется подход, аналогичный ранее использованному (в признаке `type`) для определения наилучших масштабировщиков. Сначала создаются переменные для хранения текущего наилучшего типа материала и соответствующего значения минимальной ошибки. Процесс включает в себя перебор каждого типа материала, при каждой итерации обновляется набор данных модели, задавая переменной `material` текущее значение. Далее модель обучается на этих данных, и оцениваются результаты с помощью метрик оценки качества, таких как среднеквадратичная ошибка (MSE) и средняя абсолютная ошибка (MAE).

Если найденное комбинированное значение ошибок превосходит текущий показатель минимума, информация о лучшем значении для материала и ошибках обновляется. В конечном итоге, после выполнения перебора всех вариантов, остается наилучший тип материала, который имеет ключевое значение для точности прогнозирования цены на жилье, подтвержденное наименьшими значениями ошибок. Этот опыт помогает выявить, какие характеристики строительных материалов наиболее существенно влияют на ценообразование в недвижимости.

Те же самые действия, описанные в последних абзацах, проведем с остальными признаками и их значениями:

wooden (дерево). Mixed (смешанное), Reinforced Concrete (Железобетон)

1. build\_overlap (Перекрытие)
  - a. Wooden (Дерево)
  - b. Mixed (Смешанное)
  - c. Reinforced Concrete (Железобетон)
2. city (Город)
  - a. Yoshkar\_Ola (Йошкар-Ола)
  - b. Cheboksary (Чебоксары)
  - c. Kazan (Казань)
  - d. Moscow (Москва)
3. build\_oldest (Старость здания)
  - a. new (Новое)
  - b. middle (Средней старости)
  - c. old (Старое)

Если заменить только неизвестные значения и текстовые значение признаков на числовые с помощью интуиции, то получаем следующие ошибки:

Mean Squared Error: 0.00013412042104013597

Mean Absolute Error: 0.0015191447085022157

В следующем блоке кода будет описан процесс подбора наилучшего размера тестовой выборки для модели машинного обучения. Правильный подбор

размера тестовой выборки является критически важным шагом в процессе разработки модели, поскольку он влияет на оценку её эффективности и обобщающей способности.

Выбор слишком маленькой тестовой выборки может не дать достаточного количества данных для адекватного тестирования модели, что может привести к недообучению модели, нестабильной оценке производительности. В таком случае, результаты на тестовой выборке могут значительно варьироваться при разных запусках или разбиениях данных, что затруднит понимание истинной эффективности модели.

С другой стороны, выбор слишком большой тестовой выборки означает, что для обучения модели будет использоваться меньше данных. Это может привести к переобучению на обучающей выборке, когда модель хорошо работает на данных, на которых она обучалась, но плохо адаптируется к новым, неизвестным данным. Сильно переобученная модель может показывать оптимистичные результаты на обучающей выборке, но разочаровывать при запусках на новых данных.

Оптимальный размер тестовой выборки зависит от множества факторов, в том числе от общего объема данных, сложности модели и вариативности данных. Обычная практика включает использование деления данных в соотношениях 70:30, 80:20 или даже 90:10 (обучение:тест) для больших наборов данных

Важно помнить, что идеальный размер тестовой выборки не существует в абсолютном смысле и его нужно подбирать исходя из конкретных условий задачи и доступных данных.

В программе с помощью цикла увеличиваем каждый раз обучающую выборку на 5%. В конечном итоге узнаем при какой тестовой выборке мы получим минимальную ошибку. В данном датасете лучше всего оказалось взять 85% данных в тестовую выборку.

В следующем блоке кода рассмотрена работа с мультиколлинеарностью. Мультиколлинеарность возникает, когда два или более предикторов в множественной регрессии модели сильно коррелированы, что может привести к

трудностям в интерпретации результатов регрессионного анализа. Чтобы понять какие данные сильно коррелируют между друг другом выведем диаграмму корреляции (рис. 2.2.1).

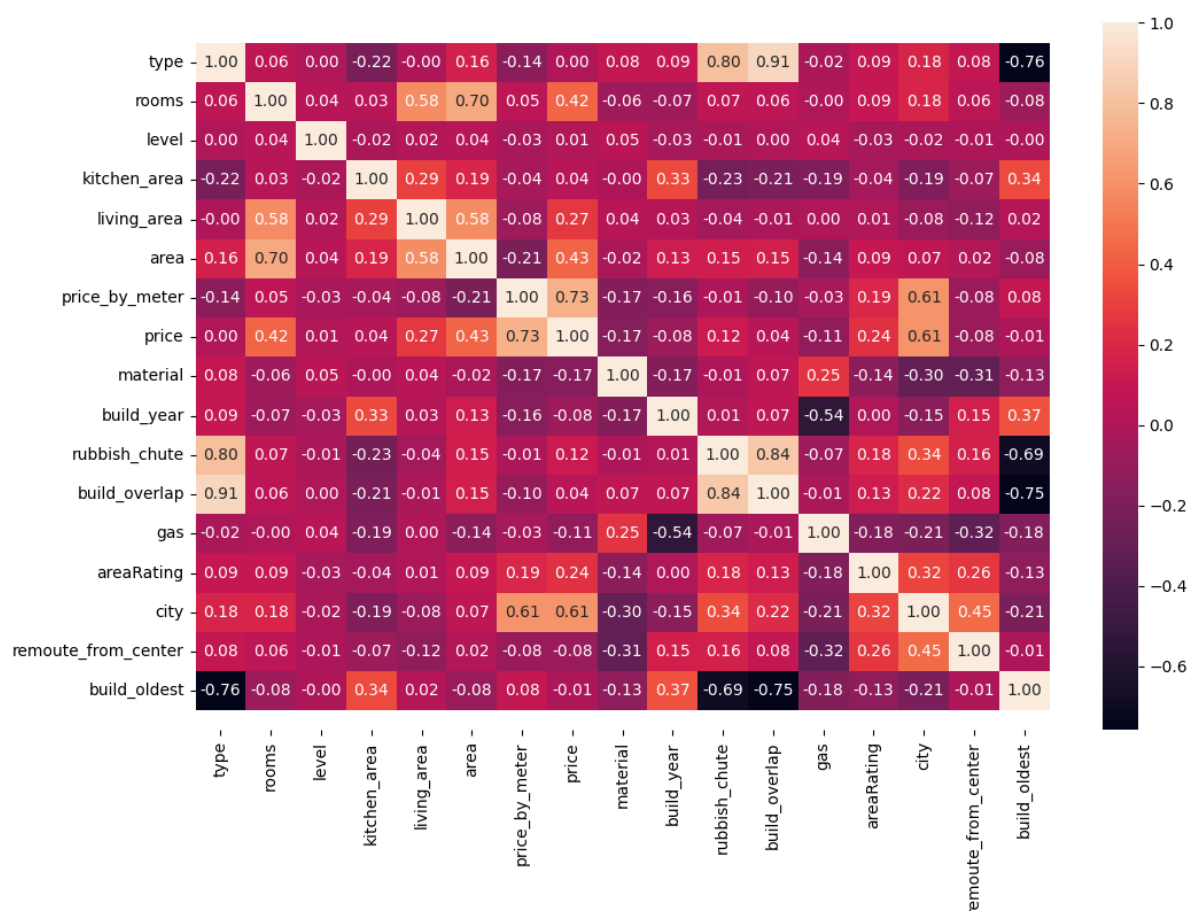


Рисунок 2.2.1. – диаграмма корреляции

Если значение стремится к -1 или к 1, то это означает, что между двумя признаками присутствует сильная корреляция, которая портит уравнение регрессии, что приводит к увеличению ошибки предсказанного значения.

Используя цикл, мы планируем обучать модель, комбинируя различные признаки. Этот подход позволит одновременно проверить несколько важных аспектов работы с регрессионными моделями в условиях большого количества признаков. В частности, мы сможем понять, способна ли регрессия адекватно обучаться, когда ей предоставляется множество переменных, а также выясним, как мультиколлинеарность влияет на точность предсказаний модели. Последнее выясним следующим образом: если модель не будет брать в обучение признаки,

который сильно коррелируют, то это будет означать, что мультиколлинеарность увеличивает ошибку.

Кроме того, данный подход даст возможность выявить, насколько хорошо модель способна отбирать действительно значимые признаки, влияющие на предсказываемый исход, и всегда ли эти признаки должны иметь существенный вес в уравнении множественной регрессии. Так мы увидим, удастся ли модели сосредоточить внимание на признаках, оказывающих существенное влияние на результат, минимизируя при этом воздействие мультиколлинеарности и избыточности данных. Перед тем как запустить цикл, выведем коэффициенты различных признаков (рис. 2.2.2.), оставляя на данный момент мультиколлинеарность и большое количество переменных.

	Коэффициенты
Тип (вид)	-5.236169e-04
Кол-во комнат	-3.398794e-04
Этаж	7.025317e-07
Кухонная площадь	-1.740980e-04
Жилая площадь	-8.804323e-05
Общая площадь	2.801984e-01
Материал	-6.423142e-04
Год постройки	-1.107646e-02
Мусоропровод (есть ли)	-4.595423e-04
Перекрытие (вид)	2.987656e-04
Газ (есть ли)	-6.799103e-05
Рейтинг местности	4.995671e-04
Город	3.610214e-03
Удаленность от центра	-6.927921e-04
Старость здания	6.171066e-04

Рисунок 2.2.2. – коэффициенты признаков

При многократном запуске цикла стало очевидно, что оптимальное количество признаков для достижения наименьшей ошибки предсказания составляет 6-7 из изначально предложенных 13. Интересно отметить, что при увеличении числа переменных до 9, точность модели начинала снижаться, и ошибка увеличивалась. Это наблюдение подчеркивает важность правильного отбора признаков в процессе обучения регрессионных моделей. Оно также показывает, что добавление избыточных переменных ведет не к улучшению, а к

ухудшению производительности модели, делая предсказания менее точными из-за возрастания шума и избыточности информации.

При этом программа выбрала такие параметры как тип (вид), рейтинг местности, старость здания, город, общая площадь, удаленность от центра и материал. Получились следующие ошибки:  $MSE = 0.0001537815993054434$ ,  $MAE = 0.0012837906695264128$ .

Теперь мы собираемся сосредоточить внимание на отслеживании выбросов для каждого отдельно взятого признака. Этот шаг позволит нам улучшить качество модели, идентифицировав и обработав аномально высокие или низкие значения, которые могут исказить результаты обучения и, соответственно, точность предсказаний. Понимание и корректировка выбросов является критически важной задачей, поскольку они могут оказывать значительное влияние на анализ данных тем, что сильно изменяют веса различных параметров, что может критически исказить уравнение регрессии. В зависимости от контекста данных и целей исследования, выбросы могут быть как удалены, так и подвергнуты дальнейшему анализу для выяснения причин их появления. Однако так как курсовая работа заключается в оптимизации модели, то просто уберем выбросы из данных.

Создадим для каждого признака диаграмму типа «ящик с усами» (boxplot) для визуализации распределения значений в столбцах датасета, ассоциированного с объектом `price_apart1`. Диаграмма позволяет наглядно увидеть медиану, межквартильный размах и выбросы, то есть значения, лежащие за пределами 1.5 межквартильных размахов от квартилей. Затем вызов `plt.show()` выводит диаграмму на экран.

Второй шаг состоит в вызове метода `remove_outliers`, который предназначен для удаления выбросов из столбца `rooms` на основе квартильного анализа. Метод определяет выбросы как значения, выходящие за пределы определенного интервала квартилей, и исключает соответствующие записи из датасета.

Ниже в виде примера будут приведены «ящики с усами» для таких признаков, как тип (type) (рис. 2.2.3.), в котором выбросов нет выбросов, и количество комнат (rooms) (рис. 2.2.4.), где есть значения, выходящие за пределы, от которых избавимся и заново выведем график уже без выбросов (рис. 2.2.5.).

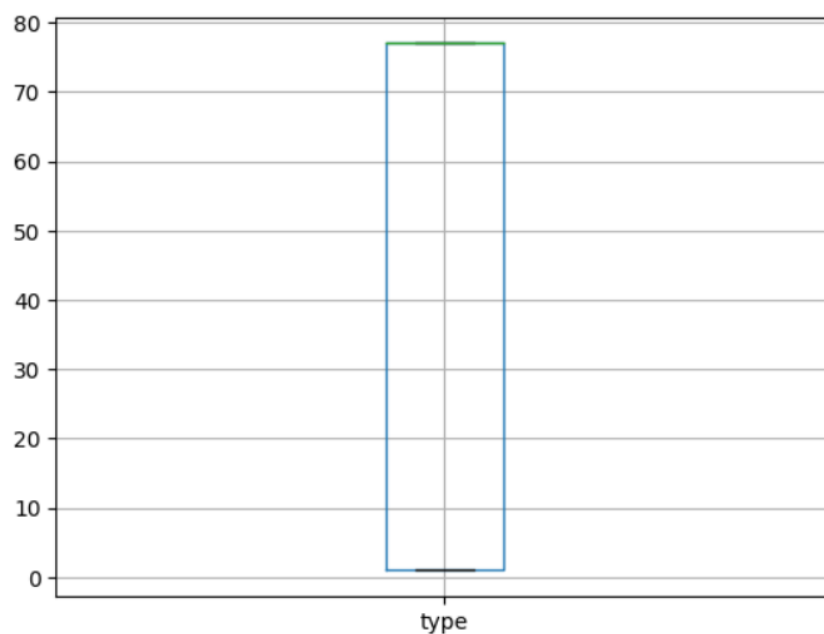


Рисунок 2.2.3. – Диаграмма «ящик с усами» без выбросов для признака «type»

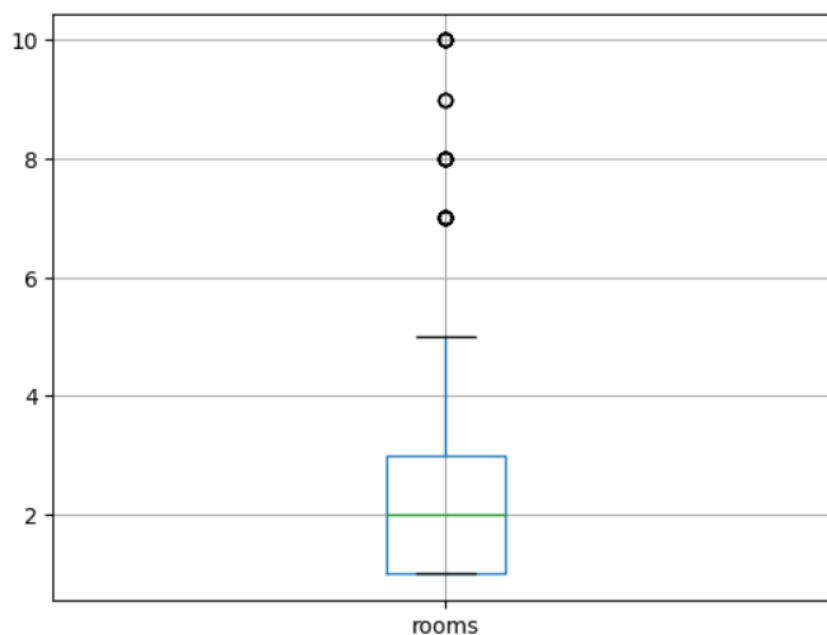


Рисунок 2.2.4. – Диаграмма «ящик с усами» с выбросами для признака «rooms»



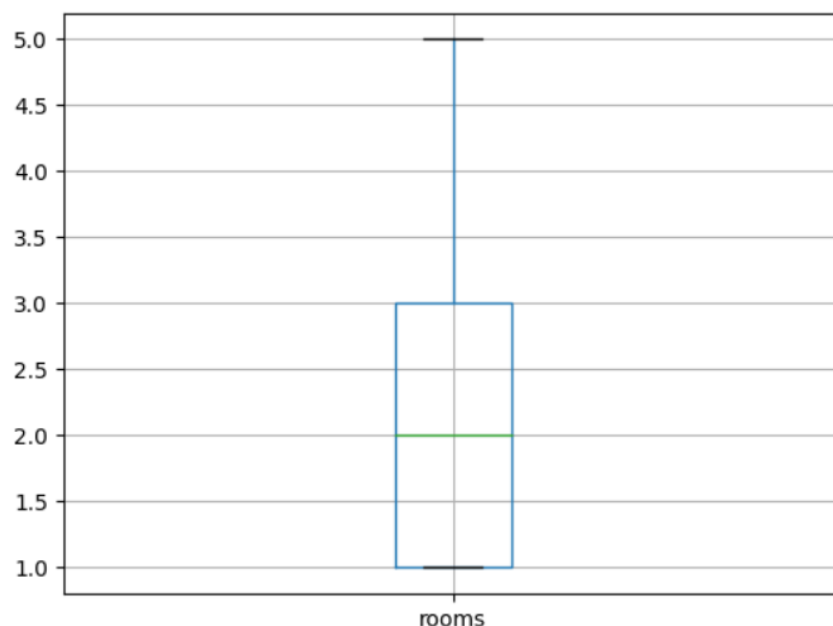


Рисунок 2.2.5. – Диаграмма «ящик с усами» без выбросов для признака «rooms»

Чтобы оптимизировать модель линейной регрессии, был применён ряд методов обработки и анализа данных. Сначала все текстовые данные были преобразованы в числовой формат, что позволило модели корректно обрабатывать категориальные переменные. Затем было принято решение о методе нормализации данных — между `StandardScaler`, стандартизирующим данные до среднего значения 0 и стандартного отклонения 1, и `MinMaxScaler`, масштабирующим данные к заданному интервалу между минимумом и максимумом. Выбор был сделан на основе конкретного распределения данных и желаемых характеристик нормализованных признаков.

Процесс отбора признаков был успешно проведён с целью идентифицировать и оставить только те переменные, которые оказывают существенное влияние на зависимую переменную. Кроме того, большое количество признаков было убрано, так как иначе это увеличило бы ошибку.

Также было уделено внимание удалению выбросов из набора данных, поскольку они негативно влияли на точность модели, искажая оценку параметров. Для обнаружения и удаления выбросов использовались методы, основанные на межквартильном размахе.

После подготовки данных, модель готова к обучению. Однако, перед началом обучения модели, предстоит визуально изучить связи между переменными, построив график pairplot с помощью библиотеки seaborn. Этот график покажет распределение каждой переменной и парные взаимосвязи между всеми переменными в датасете, предоставляя ценное предварительное представление о данных, которое может направить дальнейший выбор признаков для моделирования. Эффективность модели линейной регрессии оценим с помощью метрик, таких как среднеквадратичная ошибка (MSE), средняя абсолютная ошибка (MAE), для понимания, насколько хорошо модель будет справляться с предсказаниями на тестовых данных.

После выполнения программы мы получили результаты, которые представлены в двух ключевых аспектах: коэффициентах модели и ошибках (метриках ошибок). Результаты коэффициентов модели отображены на рисунке 2.2.6., а значения ошибок: MSE - 0.0122, MAE - 0.0802. Эти результаты позволяют нам судить о точности и надёжности модели линейной регрессии.

	Коэффициенты
Тип (вид)	-0.009246
Рейтинг местности	0.096803
Старость здания	0.065696
Город	0.643208
Общая площадь	0.458097
Удаленность от центра	-0.100387
Материал	-0.022653

Рисунок 2.2.6. – Коэффициенты признаков итоговой модели

На первый взгляд может показаться, что величина ошибок увеличилась, однако все метрики имеют кроме плюсов огромное количество минусов. Именно поэтому при анализе данных и оценке работы модели машинного обучения, особенно линейной регрессии, крайне важно не полагаться исключительно только на метрики ошибок. Это обусловлено тем, что отрицательные значения в предсказаниях модели могут искусственно уменьшать величину ошибки, создавая впечатление большей точности, чем есть на самом деле. Поэтому, даже

при наличии кажущихся малыми ошибок, необходимо провести более глубокий анализ.

Для более качественной оценки работы модели требуется также ручную сравнить предсказанные значения и действительные данные. Это позволит выявить потенциальные недостатки модели, такие как систематическое недооценивание или переоценивание результатов для определённых групп данных. Такой подход даст более полное представление о производительности модели в различных условиях и поможет идентифицировать направления для дальнейшего улучшения. Если просмотреть предсказанные значения до оптимизации, то там присутствовала большая разница между действительными и предсказанными значениями, последние зачастую даже оказывались отрицательными. После оптимизации количество значений меньше нуля стремится к нулю, а сами значения более приближены к реальным.

После завершения анализа и выполнения программы ошибки, полученные в результате, могли показаться достаточно немаленькими (более 5% неверных предсказаний), однако с учётом того, что в данных присутствовали выбросы, многие переменные были категориальными, также значимым фактором была мультиколлинеарность среди объясняющих переменных, то можно утверждать, что итоговая ошибка достаточно мала. Кроме того, если просматривать ручную, оптимизация улучшила предсказанные значения цены, которые приблизились к реальным.

## **Заключение**

В ходе выполнения курсовой работы были рассмотрены и применены различные методы и подходы к улучшению качества регрессионных моделей. Основное внимание было уделено нормализации данных, преобразованию текстовых данных в числовые, обработке выбросов с помощью квартильного анализа, борьбе с мультиколлинеарностью.

Применение нормализации позволило уменьшить разброс значений признаков, что, в свою очередь, способствовало повышению стабильности и точности множественной линейной регрессии.

Замена текстовых данных на числовые оказалась необходимым шагом для включения этих информационных источников в регрессионную модель, что расширило её аналитические возможности.

Особенно важным аспектом работы было выделение и обработка выбросов. Использование квартильного анализа позволило эффективно идентифицировать и корректировать аномалии в данных, что значительно улучшило общее качество моделирования.

Дополнительно, проблема мультиколлинеарности была решена путём анализа корреляции между переменными и последующего исключения сильно коррелированных признаков, что повысило интерпретируемость и стабильность модели.

В практической части курсовой работы был проведен всесторонний анализ и применение методов оптимизации к множественной линейной регрессии, включая все изложенные выше подходы, а также анализ влияния количества признаков, задействованных в обучении модели, на её ошибку. Хотелось бы отметить, что увеличение количества признаков в модели не всегда ведет к повышению её качества. Было показано, что добавление избыточных параметров может привести к увеличению ошибки из-за переобучения. Таким образом, выбор оптимального набора признаков остается ключевым элементом в процессе моделирования множественной линейной регрессии.

Внимание было уделено тому, как каждый из этих методов влияет на повышение точности и эффективности прогнозируемых результатов модели множественной линейной регрессии.

Применение нормализации данных помогло уравнивать масштабы признаков, что важно для корректного восприятия их значимости моделью. Преобразование текстовых данных в числовые обеспечило возможность интеграции этих важных переменных в модель, делая анализ более всесторонним. Через квартильный анализ выбросы были успешно обнаружены и скорректированы, предотвращая их искажающее влияние на результаты. Борьба с мультиколлинеарностью позволила избежать влияния избыточной корреляции между переменными на стабильность модели. Наконец, обнаружение оптимального числа признаков минимизировало риск переобучения и повысило точность прогнозирования.

В результате применения этих оптимизационных подходов было достигнуто общее снижение ошибки модели, в ручном анализе было заметно более подходящие к реальным предсказанные значения, чем это было до оптимизации, что свидетельствует о значительном улучшении точности предсказаний. Полученные результаты подтверждают эффективность испытанных методик оптимизации и их важность для улучшения качества аналитических моделей в работе со сложными данными.

В заключение хочется подчеркнуть, что выполненная работа демонстрирует значительный потенциал оптимизационных техник в регрессионном анализе. Реализованные методы и подходы могут быть использованы для улучшения точности и надежности множественной линейной регрессии в различных областях науки и промышленности. Это, в свою очередь, открывает новые возможности для исследования и разработки более эффективных алгоритмов обработки и анализа информации, что представляет значительный интерес для научного сообщества и практики применения статистических методов.

### Список использованных источников

- 1) Харченко М.А. Корреляционный анализ: Учебн. пособие. – Воронеж: Издательско-полиграфический центр Воронежского государственного университета, 2008. – 31 с.
- 2) Баврина А.П., Борисов И.Б. Современные правила применения корреляционного анализа / Медицинский альманах. – 2021. – № 5. – с. 70-79.
- 3) Гельман Э., Хилл Д., Вехтари А. Регрессия: теория и практика. С примерами на R и Stan / Пер. с англ. В.С. Яценкова. – М.: ДМК Пресс, 2022. – 748 с.
- 4) Yan X., Gang Su X. Linear Regression Analysis: Theory and Computing. – Singapore: World Scientific Publishing Co. Pte. Ltd., 2009. – 349 с.
- 5) A New Modified Liu Ridge-Type Estimator for the Linear Regression Model: Simulation and Application / Olasunkanmi J.O. и др. / International Journal of Clinical Biostatistics and Biometrics. – 2022. – Т. 8, № 2. – с. 1-14.
- 6) URL: [https://www.murraylax.org/rtutorials/regression\\_intro.html](https://www.murraylax.org/rtutorials/regression_intro.html) (Дата обращения: 17.02.2024)
- 7) Bruce P., Bruce A. Practical statistics for data scientists: 50 essential concepts. – Sebastopol: O'Reilly Media, Inc., 2017. – 409 с.
- 8) URL: <https://education.yandex.ru/handbook/data-analysis/article/linejnaya-regressiya-metod-naimenshih-kvadratov> (Дата обращения: 20.01.2024)
- 9) Исмагилов И.И., Кадочникова Е.И., Костромин А.В. Эконометрика: Конспект лекций. – Казань: Казан. ун-т, 2014. – 235 с.
- 10) Современные методы обработки и анализа данных / Орлов Г.М. и др. – СПб.: Университет ИТМО, 2021. – 147 с.
- 11) URL: <https://w1wlaa.github.io/analysis.pdf> (Дата обращения: 20.01.2024)
- 12) URL: <https://learn.microsoft.com/ru-ru/azure/architecture/data-science-process/prepare-data> (Дата обращения: 27.01.2024)
- 13) Старовойтов В.В. Нормализация данных в машинном обучении / Информатика. – 2021. – Т. 18, № 3. – с. 83-96.

- 14) Анализ данных: учебник для вузов / Мхитарян В.С. и др. – М.: Издательство Юрайт, 2024. – 490 с.
- 15) URL: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html> (Дата обращения: 09.12.2023)
- 16) Машинное обучение с использованием Python. Сборник рецептов: Пер. с англ. – СПб.: БХВ-Петербург, 2019. – 384 с.
- 17) Платонов А.В. Машинное обучение: Учебн. пособие. – М.: Издательство Юрайт, 2024. – 85 с.
- 18) Поручиков М.А. Анализ данных: Учебн. пособие. – Самара: Изд-во Самарского университета, 2016. – 88 с.
- 19) URL: <https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning> (Дата обращения: 17.02.2024)
- 20) URL: <https://www.dmitrymakarov.ru/opt/regularization-07/?ysclid=lsqglx9wda350508788#0-pereobuchenie-i-nedoobuchenie-modeli> (Дата обращения: 17.02.2024)
- 21) Галочкин В.Т. Эконометрика: учебник и практикум для вузов. – М.: Издательство Юрайт, 2024. – 293 с.
- 22) URL: <https://www.kaggle.com/code/nadezhda2019/regression> (Дата обращения: 24.01.2024)
- 23) Daoud J. Multicollinearity and Regression Analysis / Journal of Physics: Conference Series. – 2017. – Series 949. – 6 с.
- 24) URL: [https://alexanderdyakonov.wordpress.com/wp-content/uploads/2018/10/book\\_08\\_metrics\\_12\\_blog1.pdf](https://alexanderdyakonov.wordpress.com/wp-content/uploads/2018/10/book_08_metrics_12_blog1.pdf) (Дата обращения: 02.03.2024)
- 25) URL: <https://loginom.ru/blog/quality-metrics?ysclid=lt0dh9rym3331396363> (Дата обращения: 02.03.2024)
- 26) URL: [web.stanford.edu/class/stats305a/additional-reading/Shalizi19.pdf](http://web.stanford.edu/class/stats305a/additional-reading/Shalizi19.pdf) (Дата обращения: 06.03.2024)
- 27) URL: <https://lib.kgmtu.ru/wp-content/uploads/no-category/5260.pdf> (Дата обращения: 06.03.2024)

## Приложение 1 (листинг программы)

```
import pandas as pd
import numpy as np
import seaborn as sb
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_absolute_percentage_error, mean_squared_log_error
import time
import itertools

class Predict_price:
    def __init__(self, dataset):
        self.dataset = pd.read_csv(dataset, header=0)
        self.dataset.columns = self.dataset.columns.str.strip()
    def pop_column(self, column):
        self.dataset.pop(column)
    def dataset(self):
        return self.dataset
    def describe_dataset(self):
        return self.dataset.describe()
    def unique_of_column(self, column):
        return self.dataset[column].unique()
    def delete_nan_rows(self, count_nan):
        arr_columns = np.array(self.dataset.columns)
        for index, row in self.dataset.iterrows():
            for column in arr_columns:
                if row[column] == '0' or row[column] == 0 or row[column] == np.nan or
row[column] == None or row[column] == 'Не заполнено':
```



```

        self.dataset.at[index, column] = np.nan

    self.dataset.dropna(thresh=count_nan, inplace=True)

    def popular_name(self, column):

        return self.dataset[column].value_counts()[1].index.tolist()

    def change_nan_to_popular(self, column):

        popular_value = self.dataset[column].value_counts()[1].index.tolist()[0]

        self.dataset[column].fillna(popular_value, inplace=True)

    def drop_row_by_word(self, column, word):

        for index, row in self.dataset.iterrows():

            if row[column] == word:

                self.dataset.drop(labels=[index], inplace=True)

    def remove_outliers(self, column):

        Q1 = np.percentile(self.dataset[column], 25, method='midpoint') # Нижний
квартиль
        Q3 = np.percentile(self.dataset[column], 75, method='midpoint') # Верхний
квартиль

        IQR = Q3 - Q1 # Межквартильный размах (МКР)
        lower = Q1 - 1.5 * IQR
        upper = Q3 + 1.5 * IQR

        for index, row in self.dataset.iterrows():

            if row[column] > upper or row[column] < lower:

                self.dataset.drop(labels=[index], inplace=True)

    def text_to_numbers(self, dict, column):

        self.dataset[column] = self.dataset[column].replace(dict)

    def scaler(self, scaler_type, column):

        scaler = MinMaxScaler() if scaler_type == 'MinMaxScaler' else StandardScaler()

        self.dataset[[column]] = scaler.fit_transform(self.dataset[[column]])

    def fractions_into_numbers(self, column):

        popular_value = self.dataset[column].value_counts()[1].index.tolist()[0]

        if '.' in popular_value:

```

```

        first,        second        =        popular_value[:popular_value.index('.')],
popular_value[popular_value.index('.') + 1:]
        first_number, second_number = ", "
        for i in range(len(first)):
            first_number += first[i]
        for i in range(len(second)):
            second_number += second[i]
        first_number = int(first_number)
        second_number = int(second_number)
    elif '/' in popular_value:
        first,        second        =        popular_value[:popular_value.index('/')],
popular_value[popular_value.index('/') + 1:]
        first_number, second_number = ", "
        for i in range(len(first)):
            first_number += first[i]
        for i in range(len(second)):
            second_number += second[i]
        first_number = int(first_number)
        second_number = int(second_number)
    popular_value = round(first_number / second_number, 3)
    for index, row in self.dataset.iterrows():
        if row[column] == 0 or row[column] == '0' or row[column] == np.nan or
row[column] == None:
            self.dataset.at[index, column] = popular_value
        else:
            values = list(str(self.dataset.loc[index, column]))
            if '.' in values:
                first, second = values[:values.index('.')], values[values.index('.') + 1:]
                first_number, second_number = ", "
                for i in range(len(first)):

```

```

        first_number += first[i]
    for i in range(len(second)):
        second_number += second[i]
    first_number = int(first_number)
    second_number = int(second_number)
elif '/' in values:
    first, second = values[:values.index('/')], values[values.index('/') + 1:]
    first_number, second_number = "", ""
    for i in range(len(first)):
        first_number += first[i]
    for i in range(len(second)):
        second_number += second[i]
    first_number = int(first_number)
    second_number = int(second_number)
if second_number == 0:
    self.dataset.at[index, column] = popular_value
else:
    self.dataset.at[index, column] = round(first_number / second_number, 3)
def fillna_row(self, column):
    mean = self.dataset[column].mean()
    self.dataset[column].fillna(mean, inplace=True)
def sb_pairplot(self):
    return sb.pairplot(self.dataset)
def training(self, X_columns, y_column, size_of_test):
    X = self.dataset[X_columns]
    y = self.dataset[y_column]
    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, y,
test_size=size_of_test, random_state=123)
    self.regressor = LinearRegression()
    self.regressor.fit(self.X_train, self.y_train)

```

```

def coefficient(self, rows, column):
    return pd.DataFrame(self.regressor.coef_, rows, columns = column)
def price_pred(self):
    self.y_pred = self.regressor.predict(self.X_test)
    return pd.DataFrame({'Actual': self.y_test, 'Predicted': self.y_pred})
def print_errors(self):
    print(f'Mean Squared Error: {mean_squared_error(self.y_test, self.y_pred) :.3}')
    print(f'Mean Absolute Error: {mean_absolute_error(self.y_test, self.y_pred) :.3}')
    print(f'Mean                Absolute                Percentage                Error:
{mean_absolute_percentage_error(self.y_test, self.y_pred)})')
def return_errors(self):
    return [mean_squared_error(self.y_test, self.y_pred),
mean_absolute_error(self.y_test, self.y_pred),
mean_absolute_percentage_error(self.y_test, self.y_pred)]
def return_test_split(self):
    return [self.X_train, self.X_test, self.y_train, self.y_test]
price_apart1 = Predict_price('E:/University/Основы машинного
обучения/Datasets/rent_apartments.csv')
price_apart1.dataset
start_time = time.time()
# Создадим переменные для наилучших значений
global_count_nan_in_line = None # Количество nan в строке
# Создадим переменные для хранения данных, с которыми будем сравнивать
global_sum_errors = 10 ** 10
# Удалим строки, если в них количество неизвестных величин больше n
for count_nan_in_line in range(11):
    price_apart1 = Predict_price('E:/University/Основы машинного
обучения/Datasets/rent_apartments.csv')

```

```

for column in 'object_type', 'build_type', 'build_walls', 'build_serias', 'published',
'updated', 'longitude', 'latitude', 'agencyName', 'enterances', 'max_levels','min_levels',
'heating':

    price_apart1.pop_column(column)

# Создадим массив с названиями колонок
arr_columns = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area',
'price_by_meter', 'price', 'material', 'build_year', 'rubbish_chute', 'build_overlap',
'heating', 'gas', 'areaRating', 'city', 'remoute_from_center', 'build_oldest']

# Удаляем строки по кол-ву nan
price_apart1.delete_nan_rows(count_nan_in_line)

# type

# Так как специализированного жилищного фонда одна позиция, то удалим
такую строку
for index, row in price_apart1.dataset.iterrows():
    if row['type'] == 'Специализированный жилищный фонд':
        price_apart1.dataset.drop(labels=[index], inplace=True)

# Преобразование в числовые признаки
change_type = {'Квартира': 1, 'Жилой дом блокированной застройки': 2,
'Множкквартирный дом': 3}

price_apart1.text_to_numbers(change_type, 'type')

# Заменяем nan на наиболее повторяющуюся позицию
price_apart1.change_nan_to_popular('type')

#rooms

# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rooms')

# Нормализуем данные
rooms_scaler = 'MinMaxScaler'
price_apart1.scaler(rooms_scaler, 'rooms')

#level

```

# Заменяем дроби на поделенные дроби, неизвестные данные на самое встречаемое значение

```
price_apart1.fractions_into_numbers('level')
```

```
#kitchen_area
```

# Заменяем пропуски на среднее значение

```
price_apart1.fillna_row('kitchen_area')
```

```
kitchen_area_scaler = 'MinMaxScaler'
```

```
price_apart1.scaler(kitchen_area_scaler, 'kitchen_area')
```

```
#living_area
```

# Заменяем пропуски на среднее значение

```
price_apart1.fillna_row('living_area')
```

```
living_area_scaler = 'MinMaxScaler'
```

```
price_apart1.scaler(living_area_scaler, 'living_area')
```

```
#area
```

# Заменяем пропуски на среднее значение

```
price_apart1.fillna_row('area')
```

```
area_scaler = 'MinMaxScaler'
```

```
price_apart1.scaler(area_scaler, 'area')
```

```
#price
```

# Заменяем пропуски на среднее значение

```
price_apart1.fillna_row('price')
```

```
price_scaler = 'MinMaxScaler'
```

```
price_apart1.scaler(price_scaler, 'price')
```

```
#material
```

```
change_material = {'Панель': 7, 'Кирпич': 10, 'Блоки': 8, 'Монолит': 6, 'Дерево':
```

```
5}
```

```
price_apart1.text_to_numbers(change_material, 'material')
```

# Заменяем пропуски на среднее значение

```
price_apart1.fillna_row('material')
```

# Нормализуем данные

```

material_scaler = 'MinMaxScaler'
price_apart1.scaler(material_scaler, 'material')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#rubbish_chute
change_rubbish_chute = {'Нет': 0, 'Есть': 1}
price_apart1.text_to_numbers(change_rubbish_chute, 'rubbish_chute')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rubbish_chute')
# Нормализуем данные
rubbish_chute_scaler = 'MinMaxScaler'
price_apart1.scaler(rubbish_chute_scaler, 'rubbish_chute')
#build_overlap
build_overlap_change = {'Железобетонные': 10, 'Смешанные': 5, 'Деревянные':
1}
price_apart1.text_to_numbers(build_overlap_change, 'build_overlap')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_overlap')
# Нормализуем данные
build_overlap_scaler = 'MinMaxScaler'
price_apart1.scaler(build_overlap_scaler, 'build_overlap')
#gas
change_gas = {'Нет': 0, 'Да': 1}
price_apart1.text_to_numbers(change_gas, 'gas')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('gas')
# Нормализуем данные

```

```

gas_scaler = 'MinMaxScaler'
price_apart1.scaler(gas_scaler, 'gas')
#areaRating
# Нормализуем
areaRating_scaler = 'MinMaxScaler'
price_apart1.scaler(areaRating_scaler, 'areaRating')
#city
change_city = {'Москва': 10, 'Казань': 8, 'Чебоксары': 6, 'Йошкар-Ола': 3}
price_apart1.text_to_numbers(change_city, 'city')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('city')
# Нормализуем данные
city_scaler = 'MinMaxScaler'
price_apart1.scaler(city_scaler, 'city')
#remoute_from_center
remoute_from_center_scaler = 'MinMaxScaler'
price_apart1.scaler(remoute_from_center_scaler, 'remoute_from_center')
#build_oldest
build_oldest_change = {'new': 3, 'middle': 2, 'old': 1}
price_apart1.text_to_numbers(build_oldest_change, 'build_oldest')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_oldest')
# Нормализуем данные
build_oldest_scaler = 'MinMaxScaler'
price_apart1.scaler(build_oldest_scaler, 'build_oldest')
# Обучение
num_test_size = 0.9
X = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area', 'material', 'build_year',
'rubbish_chute', 'build_overlap', 'gas', 'areaRating', 'city', 'remoute_from_center',
'build_oldest']

```



```

y = 'price'
price_apart1.training(X, y, num_test_size)
coeff_rows = ["Тип (вид)", "Кол-во комнат", "Этаж", "Кухонная площадь",
"Жилая площадь", "Общая площадь", "Материал", "Год постройки",
"Мусоропровод (есть ли)", "Перекрытие (вид)", "Газ (есть ли)", "Рейтинг
местности", "Город", "Удаленность от центра", "Старость здания"]

coeff = price_apart1.coefficient(coeff_rows, ['Коэффициенты'])
pred = price_apart1.price_pred()
MSE, MAE = price_apart1.return_errors()[0], price_apart1.return_errors()[1]
if MSE + MAE < global_sum_errors:
    global_count_nan_in_line = count_nan_in_line
    global_sum_errors = MSE + MAE
print(global_count_nan_in_line, global_sum_errors)
print("--- %s seconds ---" % (time.time() - start_time))
start_time = time.time()

# Удалим строки, если в них количество неизвестных величин больше 5
global_change_type_flat = None # Квартира в type
global_change_type_residential_building = None # Жилой дом блокированной
застройки в type
global_change_type_apartment_building = None # Многоквартирный дом в type
count_nan_in_line = 5
global_sum_errors = 10 ** 10
price_apart1 = Predict_price('E:/University/Основы машинного
обучения/Datasets/rent_apartments.csv')
for column in 'object_type', 'build_type', 'build_walls', 'build_serias', 'published',
'updated', 'longitude', 'latitude', 'agencyName', 'enterances', 'max_levels', 'min_levels',
'heating':
    price_apart1.pop_column(column)
# Создадим массив с названиями колонок

```

```

arr_columns = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area',
'price_by_meter', 'price', 'material', 'build_year', 'rubbish_chute', 'build_overlap',
'heating', 'gas', 'areaRating', 'city', 'remoute_from_center', 'build_oldest']

# Удаляем строки по кол-ву nan
price_apart1.delete_nan_rows(count_nan_in_line)

#rooms

# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rooms')

# Нормализуем данные
rooms_scaler = 'MinMaxScaler'
price_apart1.scaler(rooms_scaler, 'rooms')

#level

# Заменяем дроби на поделенные дроби, неизвестные данные на самое
встречаемое значение
price_apart1.fractions_into_numbers('level')

#kitchen_area

# Заменяем пропуски на среднее значение
price_apart1.fillna_row('kitchen_area')
kitchen_area_scaler = 'MinMaxScaler'
price_apart1.scaler(kitchen_area_scaler, 'kitchen_area')

#living_area

# Заменяем пропуски на среднее значение
price_apart1.fillna_row('living_area')
living_area_scaler = 'MinMaxScaler'
price_apart1.scaler(living_area_scaler, 'living_area')

#area

# Заменяем пропуски на среднее значение
price_apart1.fillna_row('area')
area_scaler = 'MinMaxScaler'
price_apart1.scaler(area_scaler, 'area')

```

```

#price
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('price')
price_scaler = 'MinMaxScaler'
price_apart1.scaler(price_scaler, 'price')

#material
change_material = {'Панель': 7, 'Кирпич': 10, 'Блоки': 8, 'Монолит': 6, 'Дерево': 5}
price_apart1.text_to_numbers(change_material, 'material')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('material')
# Нормализуем данные
material_scaler = 'MinMaxScaler'
price_apart1.scaler(material_scaler, 'material')

#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')

#rubbish_chute
change_rubbish_chute = {'Нет': 0, 'Есть': 1}
price_apart1.text_to_numbers(change_rubbish_chute, 'rubbish_chute')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rubbish_chute')
# Нормализуем данные
rubbish_chute_scaler = 'MinMaxScaler'
price_apart1.scaler(rubbish_chute_scaler, 'rubbish_chute')

#build_overlap
build_overlap_change = {'Железобетонные': 10, 'Смешанные': 5, 'Деревянные': 1}
price_apart1.text_to_numbers(build_overlap_change, 'build_overlap')
# Заменяем пропуски на среднее значение

```

```

price_apart1.fillna_row('build_overlap')
# Нормализуем данные
build_overlap_scaler = 'MinMaxScaler'
price_apart1.scaler(build_overlap_scaler, 'build_overlap')
#gas
change_gas = {'Нет': 0, 'Да': 1}
price_apart1.text_to_numbers(change_gas, 'gas')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('gas')
# Нормализуем данные
gas_scaler = 'MinMaxScaler'
price_apart1.scaler(gas_scaler, 'gas')
#areaRating
# Нормализуем
areaRating_scaler = 'MinMaxScaler'
price_apart1.scaler(areaRating_scaler, 'areaRating')
#city
change_city = {'Москва': 10, 'Казань': 8, 'Чебоксары': 6, 'Йошкар-Ола': 3}
price_apart1.text_to_numbers(change_city, 'city')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('city')
# Нормализуем данные
city_scaler = 'MinMaxScaler'
price_apart1.scaler(city_scaler, 'city')
#remoute_from_center
remoute_from_center_scaler = 'MinMaxScaler'
price_apart1.scaler(remoute_from_center_scaler, 'remoute_from_center')
#build_oldest
build_oldest_change = {'new': 3, 'middle': 2, 'old': 1}
price_apart1.text_to_numbers(build_oldest_change, 'build_oldest')

```

```

# Заменим пропуски на среднее значение
price_apart1.fillna_row('build_oldest')

# Нормализуем данные
build_oldest_scaler = 'MinMaxScaler'
price_apart1.scaler(build_oldest_scaler, 'build_oldest')

# type

# Заменим nan на наиболее повторяющуюся позицию
price_apart1.change_nan_to_popular('type')

# Так как специализированного жилищного фонда одна позиция, то удалим
такую строку
for index, row in price_apart1.dataset.iterrows():
    if row['type'] == 'Специализированный жилищный фонд':
        price_apart1.dataset.drop(labels=[index], inplace=True)

# Преобразование в числовые признаки
type_copy = price_apart1.dataset['type'].copy()
for change_type_flat in range(1, 1 + 1):
    for change_type_residential_building in range(change_type_flat + 1, 50 + 1):
        for change_type_apartment_building in range(change_type_residential_building
+ 1, 100 + 1):
            price_apart1.dataset['type'] = type_copy
            change_type = {'Квартира': change_type_flat, 'Жилой дом блокированной
застройки': change_type_residential_building, 'Многоквартирный дом':
change_type_apartment_building}
            price_apart1.text_to_numbers(change_type, 'type')

# Обучение
num_test_size = 0.9
X = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area', 'material',
'build_year', 'rubbish_chute', 'build_overlap', 'gas', 'areaRating', 'city',
'remoute_from_center', 'build_oldest']
y = 'price'

```

```

price_apart1.training(X, y, num_test_size)

coeff_rows = ["Тип (вид)", "Кол-во комнат", "Этаж", "Кухонная площадь",
"Жилая площадь", "Общая площадь", "Материал", "Год постройки",
"Мусоропровод (есть ли)", "Перекрытие (вид)", "Газ (есть ли)", "Рейтинг
местности", "Город", "Удаленность от центра", "Старость здания"]

coeff = price_apart1.coefficient(coeff_rows, ['Коэффициенты'])

pred = price_apart1.price_pred()

MSE, MAE = price_apart1.return_errors()[0], price_apart1.return_errors()[1]

if MSE + MAE < global_sum_errors:

    global_change_type_residential_building =
change_type_residential_building

    global_change_type_apartment_building =
change_type_apartment_building

    global_change_type_flat = change_type_flat

    global_sum_errors = MSE + MAE

print(global_sum_errors)

print('change_type_residential_building', global_change_type_residential_building)
print('change_type_apartment_building', global_change_type_apartment_building)
print('change_type_flat', global_change_type_flat)

print("--- %s seconds ---" % (time.time() - start_time))

start_time = time.time()

global_rooms_scaler = None
global_kitchen_area_scaler = None
global_living_area_scaler = None
global_area_scaler = None
global_price_scaler = None
global_build_year_scaler = None
global_areaRating_scaler = None
global_remoute_from_center_scaler = None

count_nan_in_line = 5

```

```

global_sum_errors = 10 ** 10

price_apart1 = Predict_price('E:/University/Основы машинного
обучения/Datasets/rent_apartments.csv')

for column in 'object_type', 'build_type', 'build_walls', 'build_serias', 'published',
'updated', 'longitude', 'latitude', 'agencyName', 'enterances', 'max_levels','min_levels',
'heating':

    price_apart1.pop_column(column)

# Создадим массив с названиями колонок
arr_columns = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area',
'price_by_meter', 'price', 'material', 'build_year', 'rubbish_chute', 'build_overlap',
'heating', 'gas', 'areaRating', 'city', 'remoute_from_center', 'build_oldest']

# Удаляем строки по кол-ву nan
price_apart1.delete_nan_rows(count_nan_in_line)

# type

# Так как специализированного жилищного фонда одна позиция, то удалим
такую строку
for index, row in price_apart1.dataset.iterrows():

    if row['type'] == 'Специализированный жилищный фонд':

        price_apart1.dataset.drop(labels=[index], inplace=True)

# Преобразование в числовые признаки
change_type = {'Квартира': 1, 'Жилой дом блокированной застройки': 46,
'Многоквартирный дом': 77}
price_apart1.text_to_numbers(change_type, 'type')

# Заменяем nan на наиболее повторяющуюся позицию
price_apart1.change_nan_to_popular('type')

#level

# Заменяем дроби на поделенные дроби, неизвестные данные на самое
встречаемое значение
price_apart1.fractions_into_numbers('level')

#material

```

```

change_material = {'Панель': 7, 'Кирпич': 10, 'Блоки': 8, 'Монолит': 6, 'Дерево': 5}
price_apart1.text_to_numbers(change_material, 'material')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('material')
# Нормализуем данные
material_scaler = 'MinMaxScaler'
price_apart1.scaler(material_scaler, 'material')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#rubbish_chute
change_rubbish_chute = {'Нет': 0, 'Есть': 1}
price_apart1.text_to_numbers(change_rubbish_chute, 'rubbish_chute')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rubbish_chute')
# Нормализуем данные
rubbish_chute_scaler = 'MinMaxScaler'
price_apart1.scaler(rubbish_chute_scaler, 'rubbish_chute')
#build_overlap
build_overlap_change = {'Железобетонные': 10, 'Смешанные': 5, 'Деревянные': 1}
price_apart1.text_to_numbers(build_overlap_change, 'build_overlap')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_overlap')
# Нормализуем данные
build_overlap_scaler = 'MinMaxScaler'
price_apart1.scaler(build_overlap_scaler, 'build_overlap')
#gas
change_gas = {'Нет': 0, 'Да': 1}

```



```

price_apart1.text_to_numbers(change_gas, 'gas')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('gas')
# Нормализуем данные
gas_scaler = 'MinMaxScaler'
price_apart1.scaler(gas_scaler, 'gas')
#city
change_city = {'Москва': 10, 'Казань': 8, 'Чебоксары': 6, 'Йошкар-Ола': 3}
price_apart1.text_to_numbers(change_city, 'city')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('city')
# Нормализуем данные
city_scaler = 'MinMaxScaler'
price_apart1.scaler(city_scaler, 'city')
#build_oldest
build_oldest_change = {'new': 3, 'middle': 2, 'old': 1}
price_apart1.text_to_numbers(build_oldest_change, 'build_oldest')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_oldest')
# Нормализуем данные
build_oldest_scaler = 'MinMaxScaler'
price_apart1.scaler(build_oldest_scaler, 'build_oldest')
price_apart1.fillna_row('rooms')
price_apart1.fillna_row('kitchen_area')
price_apart1.fillna_row('living_area')
price_apart1.fillna_row('area')
price_apart1.fillna_row('price')
price_apart1.fillna_row('build_year')
for rooms_scaler in 'MinMaxScaler', 'StandartScaler':
    for kitchen_area_scaler in 'MinMaxScaler', 'StandartScaler':

```

```

for living_area_scaler in 'MinMaxScaler', 'StandartScaler':
    for area_scaler in 'MinMaxScaler', 'StandartScaler':
        for price_scaler in 'MinMaxScaler', 'StandartScaler':
            for build_year_scaler in 'MinMaxScaler', 'StandartScaler':
                for areaRating_scaler in 'MinMaxScaler', 'StandartScaler':
                    for remoute_from_center_scaler in 'MinMaxScaler', 'StandartScaler':
                        price_apart1.scaler(rooms_scaler, 'rooms')
                        price_apart1.scaler(kitchen_area_scaler, 'kitchen_area')
                        price_apart1.scaler(living_area_scaler, 'living_area')
                        price_apart1.scaler(area_scaler, 'area')
                        price_apart1.scaler(price_scaler, 'price')
                        price_apart1.scaler(build_year_scaler, 'build_year')
                        price_apart1.scaler(areaRating_scaler, 'areaRating')
                        price_apart1.scaler(remoute_from_center_scaler,
'remoute_from_center')
                        # Обучение
                        num_test_size = 0.9
                        X = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area',
'material', 'build_year', 'rubbish_chute', 'build_overlap', 'gas', 'areaRating', 'city',
'remoute_from_center', 'build_oldest']
                        y = 'price'
                        price_apart1.training(X, y, num_test_size)
                        coeff_rows = ["Тип (вид)", "Кол-во комнат", "Этаж", "Кухонная
площадь", "Жилая площадь", "Общая площадь", "Материал", "Год постройки",
"Мусоропровод (есть ли)", "Перекрытие (вид)", "Газ (есть ли)", "Рейтинг
местности", "Город", "Удаленность от центра", "Старость здания"]
                        coeff = price_apart1.coefficient(coeff_rows, ['Коэффициенты'])
                        pred = price_apart1.price_pred()
                        MSE, MAE = price_apart1.return_errors()[0],
price_apart1.return_errors()[1]

```

```

        if MSE + MAE < global_sum_errors:
            global_rooms_scaler = rooms_scaler
            global_kitchen_area_scaler = kitchen_area_scaler
            global_living_area_scaler = living_area_scaler
            global_area_scaler = area_scaler
            global_price_scaler = price_scaler
            global_build_year_scaler = build_year_scaler
            global_areaRating_scaler = areaRating_scaler
            global_remoute_from_center_scaler =
remoute_from_center_scaler

            global_sum_errors = MSE + MAE

print(global_sum_errors)
print('rooms_scaler', global_rooms_scaler)
print('kitchen_area_scaler', global_kitchen_area_scaler)
print('living_area_scaler', global_living_area_scaler)
print('area_scaler', global_area_scaler)
print('price_scaler', global_price_scaler)
print('build_year', global_build_year_scaler)
print('areaRating', global_areaRating_scaler)
print('remoute_from_center', global_remoute_from_center_scaler)
print("--- %s seconds ---" % (time.time() - start_time))

# Заменяем категориальные данные числовыми для оставшихся параметров
# material
global_change_material_Wood = None
global_change_material_Monolith = None
global_change_material_Panel = None
global_change_material_Blocks = None
global_change_material_Brick = None
start_time = time.time()
count_nan_in_line = 5

```

```

global_sum_errors = 10 ** 10

price_apart1 = Predict_price('E:/University/Основы машинного
обучения/Datasets/rent_apartments.csv')

for column in 'object_type', 'build_type', 'build_walls', 'build_serias', 'published',
'updated', 'longitude', 'latitude', 'agencyName', 'enterances', 'max_levels','min_levels',
'heating':

    price_apart1.pop_column(column)

# Создадим массив с названиями колонок
arr_columns = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area',
'price_by_meter', 'price', 'material', 'build_year', 'rubbish_chute', 'build_overlap',
'heating', 'gas', 'areaRating', 'city', 'remoute_from_center', 'build_oldest']

# Удаляем строки по кол-ву nan
price_apart1.delete_nan_rows(count_nan_in_line)

# type

# Так как специализированного жилищного фонда одна позиция, то удалим
такую строку
for index, row in price_apart1.dataset.iterrows():

    if row['type'] == 'Специализированный жилищный фонд':

        price_apart1.dataset.drop(labels=[index], inplace=True)

# Преобразование в числовые признаки
change_type = {'Квартира': 1, 'Жилой дом блокированной застройки': 46,
'Многоквартирный дом': 77}
price_apart1.text_to_numbers(change_type, 'type')

# Заменяем nan на наиболее повторяющуюся позицию
price_apart1.change_nan_to_popular('type')

#level

# Заменяем дроби на поделенные дроби, неизвестные данные на самое
встречаемое значение
price_apart1.fractions_into_numbers('level')

#rooms

```

```
# Заменим пропуски на среднее значение
price_apart1.fillna_row('rooms')

# Нормализуем данные
rooms_scaler = 'StandartScaler'
price_apart1.scaler(rooms_scaler, 'rooms')

#kitchen_area

# Заменим пропуски на среднее значение
price_apart1.fillna_row('kitchen_area')
kitchen_area_scaler = 'StandartScaler'
price_apart1.scaler(kitchen_area_scaler, 'kitchen_area')

#living_area

# Заменим пропуски на среднее значение
price_apart1.fillna_row('living_area')
living_area_scaler = 'StandartScaler'
price_apart1.scaler(living_area_scaler, 'living_area')

#area

# Заменим пропуски на среднее значение
price_apart1.fillna_row('area')
area_scaler = 'MinMaxScaler'
price_apart1.scaler(area_scaler, 'area')

#price

# Заменим пропуски на среднее значение
price_apart1.fillna_row('price')
price_scaler = 'MinMaxScaler'
price_apart1.scaler(price_scaler, 'price')

#build_year

# Заменим пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
```

```

#areaRating
# Нормализуем
areaRating_scaler = 'MinMaxScaler'
price_apart1.scaler(areaRating_scaler, 'areaRating')

#remoute_from_center
remoute_from_center_scaler = 'StandartScaler'
price_apart1.scaler(remoute_from_center_scaler, 'remoute_from_center')

#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')

#rubbish_chute
change_rubbish_chute = {'Нет': 0, 'Есть': 1}
price_apart1.text_to_numbers(change_rubbish_chute, 'rubbish_chute')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rubbish_chute')
# Нормализуем данные
rubbish_chute_scaler = 'MinMaxScaler'
price_apart1.scaler(rubbish_chute_scaler, 'rubbish_chute')

#build_overlap
build_overlap_change = {'Железобетонные': 10, 'Смешанные': 5, 'Деревянные': 1}
price_apart1.text_to_numbers(build_overlap_change, 'build_overlap')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_overlap')
# Нормализуем данные
build_overlap_scaler = 'MinMaxScaler'
price_apart1.scaler(build_overlap_scaler, 'build_overlap')

#gas
change_gas = {'Нет': 0, 'Да': 1}

```

```

price_apart1.text_to_numbers(change_gas, 'gas')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('gas')
# Нормализуем данные
gas_scaler = 'MinMaxScaler'
price_apart1.scaler(gas_scaler, 'gas')
#city
change_city = {'Москва': 10, 'Казань': 8, 'Чебоксары': 6, 'Йошкар-Ола': 3}
price_apart1.text_to_numbers(change_city, 'city')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('city')
# Нормализуем данные
city_scaler = 'MinMaxScaler'
price_apart1.scaler(city_scaler, 'city')
#build_oldest
build_oldest_change = {'new': 3, 'middle': 2, 'old': 1}
price_apart1.text_to_numbers(build_oldest_change, 'build_oldest')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_oldest')
# Нормализуем данные
build_oldest_scaler = 'MinMaxScaler'
price_apart1.scaler(build_oldest_scaler, 'build_oldest')
material_copy = price_apart1.dataset['material']
for change_material_Wood in range(1, 1 + 1):
    for change_material_Monolith in range(change_material_Wood + 1, 10 + 1):
        print(change_material_Monolith, 'Monolith in for')
        print("--- %s seconds ---" % (time.time() - start_time))
    for change_material_Panel in range(change_material_Monolith + 1, 20 + 1):
        for change_material_Blocks in range(change_material_Panel + 1, 30 + 1):
            for change_material_Brick in range(change_material_Blocks + 1, 40 + 1):

```

```

price_apart1.dataset['material'] = material_copy

change_material = {'Панель': change_material_Panel, 'Кирпич':
change_material_Brick, 'Блоки': change_material_Blocks, 'Монолит':
change_material_Monolith, 'Дерево': change_material_Wood}

price_apart1.text_to_numbers(change_material, 'material')
price_apart1.fillna_row('material')
material_scaler = 'MinMaxScaler'
price_apart1.scaler(material_scaler, 'material')

# Обучение
num_test_size = 0.9

X = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area', 'material',
'build_year', 'rubbish_chute', 'build_overlap', 'gas', 'areaRating', 'city',
'remoute_from_center', 'build_oldest']

y = 'price'
price_apart1.training(X, y, num_test_size)

coeff_rows = ["Тип (вид)", "Кол-во комнат", "Этаж", "Кухонная
площадь", "Жилая площадь", "Общая площадь", "Материал", "Год постройки",
"Мусоропровод (есть ли)", "Перекрытие (вид)", "Газ (есть ли)", "Рейтинг
местности", "Город", "Удаленность от центра", "Старость здания"]

coeff = price_apart1.coefficient(coeff_rows, ['Коэффициенты'])
pred = price_apart1.price_pred()

MSE, MAE = price_apart1.return_errors()[0],
price_apart1.return_errors()[1]

if MSE + MAE < global_sum_errors:

    global_change_material_Wood = change_material_Wood
    global_change_material_Monolith = change_material_Monolith
    global_change_material_Panel = change_material_Panel
    global_change_material_Blocks = change_material_Blocks
    global_change_material_Brick = change_material_Brick
    global_sum_errors = MSE + MAE

```



```

print('\n')
print(global_change_material_Wood, 'Wood')
print(global_change_material_Monolith, 'Monolith')
print(global_change_material_Panel, 'Panel')
print(global_change_material_Blocks, 'Blocs')
print(global_change_material_Brick, 'Brick')
print(global_sum_errors)
print("--- %s seconds ---" % (time.time() - start_time))
# Заменяем категориальные данные числовыми для оставшихся параметров
# build_overlap
global_build_overlap_Wooden = None
global_build_overlap_Mixed = None
global_build_overlap_Reinforced_Concrete = None
start_time = time.time()
count_nan_in_line = 5
global_sum_errors = 10 ** 10
price_apart1 = Predict_price('E:/University/Основы машинного
обучения/Datasets/rent_apartments.csv')
for column in 'object_type', 'build_type', 'build_walls', 'build_serias', 'published',
'updated', 'longitude', 'latitude', 'agencyName', 'enterances', 'max_levels', 'min_levels',
'heating':
    price_apart1.pop_column(column)
# Создадим массив с названиями колонок
arr_columns = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area',
'price_by_meter', 'price', 'material', 'build_year', 'rubbish_chute', 'build_overlap',
'heating', 'gas', 'areaRating', 'city', 'remoute_from_center', 'build_oldest']
# Удаляем строки по кол-ву nan
price_apart1.delete_nan_rows(count_nan_in_line)
# type

```

```

# Так как специализированного жилищного фонда одна позиция, то удалим
такую строку
for index, row in price_apart1.dataset.iterrows():
    if row['type'] == 'Специализированный жилищный фонд':
        price_apart1.dataset.drop(labels=[index], inplace=True)
# Преобразование в числовые признаки
change_type = {'Квартира': 1, 'Жилой дом блокированной застройки': 46,
'Mногоквартирный дом': 77}
price_apart1.text_to_numbers(change_type, 'type')
# Заменяем nan на наиболее повторяющуюся позицию
price_apart1.change_nan_to_popular('type')
#level
# Заменяем дроби на поделенные дроби, неизвестные данные на самое
встречаемое значение
price_apart1.fractions_into_numbers('level')
#rooms
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rooms')
# Нормализуем данные
rooms_scaler = 'StandartScaler'
price_apart1.scaler(rooms_scaler, 'rooms')
#kitchen_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('kitchen_area')
kitchen_area_scaler = 'StandartScaler'
price_apart1.scaler(kitchen_area_scaler, 'kitchen_area')
#living_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('living_area')
living_area_scaler = 'StandartScaler'

```

```

price_apart1.scaler(living_area_scaler, 'living_area')
#area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('area')
area_scaler = 'MinMaxScaler'
price_apart1.scaler(area_scaler, 'area')
#price
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('price')
price_scaler = 'MinMaxScaler'
price_apart1.scaler(price_scaler, 'price')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#areaRating
# Нормализуем
areaRating_scaler = 'MinMaxScaler'
price_apart1.scaler(areaRating_scaler, 'areaRating')
#remoute_from_center
remoute_from_center_scaler = 'StandartScaler'
price_apart1.scaler(remoute_from_center_scaler, 'remoute_from_center')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#material
change_material = {'Панель': 7, 'Кирпич': 10, 'Блоки': 8, 'Монолит': 6, 'Дерево': 5}

```

```

price_apart1.text_to_numbers(change_material, 'material')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('material')
# Нормализуем данные
material_scaler = 'MinMaxScaler'
price_apart1.scaler(material_scaler, 'material')
#rubbish_chute
change_rubbish_chute = {'Нет': 0, 'Есть': 1}
price_apart1.text_to_numbers(change_rubbish_chute, 'rubbish_chute')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rubbish_chute')
# Нормализуем данные
rubbish_chute_scaler = 'MinMaxScaler'
price_apart1.scaler(rubbish_chute_scaler, 'rubbish_chute')
#gas
change_gas = {'Нет': 0, 'Да': 1}
price_apart1.text_to_numbers(change_gas, 'gas')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('gas')
# Нормализуем данные
gas_scaler = 'MinMaxScaler'
price_apart1.scaler(gas_scaler, 'gas')
#city
change_city = {'Москва': 10, 'Казань': 8, 'Чебоксары': 6, 'Йошкар-Ола': 3}
price_apart1.text_to_numbers(change_city, 'city')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('city')
# Нормализуем данные
city_scaler = 'MinMaxScaler'
price_apart1.scaler(city_scaler, 'city')

```

```

#build_oldest
build_oldest_change = {'new': 3, 'middle': 2, 'old': 1}
price_apart1.text_to_numbers(build_oldest_change, 'build_oldest')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_oldest')
# Нормализуем данные
build_oldest_scaler = 'MinMaxScaler'
price_apart1.scaler(build_oldest_scaler, 'build_oldest')
build_overlap_copy = price_apart1.dataset['build_overlap']
for change_build_overlap_Wooden in range(1, 1 + 1):
    for change_build_overlap_Mixed in range(change_build_overlap_Wooden + 1, 10
+ 1):
        for change_build_overlap_Reinforced_Concrete in
range(change_build_overlap_Mixed + 1, 20 + 1):
            price_apart1.dataset['build_overlap'] = build_overlap_copy
            build_overlap_change = {'Железобетонные':
change_build_overlap_Reinforced_Concrete, 'Смешанные':
change_build_overlap_Mixed, 'Деревянные': change_build_overlap_Wooden}
            price_apart1.text_to_numbers(build_overlap_change, 'build_overlap')
            price_apart1.fillna_row('build_overlap')
            build_overlap_scaler = 'MinMaxScaler'
            price_apart1.scaler(build_overlap_scaler, 'build_overlap')
            # Обучение
            num_test_size = 0.9
            X = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area', 'material',
'build_year', 'rubbish_chute', 'build_overlap', 'gas', 'areaRating', 'city',
'remoute_from_center', 'build_oldest']
            y = 'price'
            price_apart1.training(X, y, num_test_size)

```

```

coeff_rows = ["Тип (вид)", "Кол-во комнат", "Этаж", "Кухонная площадь",
"Жилая площадь", "Общая площадь", "Материал", "Год постройки",
"Мусоропровод (есть ли)", "Перекрытие (вид)", "Газ (есть ли)", "Рейтинг
местности", "Город", "Удаленность от центра", "Старость здания"]

coeff = price_apart1.coefficient(coeff_rows, ['Коэффициенты'])
pred = price_apart1.price_pred()
MSE, MAE = price_apart1.return_errors()[0], price_apart1.return_errors()[1]
if MSE + MAE < global_sum_errors:
    global_build_overlap_Wooden = change_build_overlap_Wooden
    global_build_overlap_Mixed = change_build_overlap_Mixed
    global_build_overlap_Reinforced_Concrete =
change_build_overlap_Reinforced_Concrete
    global_sum_errors = MSE + MAE
print('\n')
print(global_build_overlap_Wooden, 'Wooden')
print(global_build_overlap_Mixed, 'Mixed')
print(global_build_overlap_Reinforced_Concrete, 'Reinforced_Concrete')
print(global_sum_errors)
print("--- %s seconds ---" % (time.time() - start_time))
# Заменяем категориальные данные числовыми для оставшихся параметров
# city
global_city_Yoshkar_Ola = None
global_city_Cheboksary = None
global_city_Kazan = None
global_city_Moscow = None
start_time = time.time()
count_nan_in_line = 5
global_sum_errors = 10 ** 10
price_apart1 = Predict_price('E:/University/Основы машинного
обучения/Datasets/rent_apartments.csv')

```

```

for column in 'object_type', 'build_type', 'build_walls', 'build_serias', 'published',
'updated', 'longitude', 'latitude', 'agencyName', 'enterances', 'max_levels','min_levels',
'heating':

    price_apart1.pop_column(column)
# Создадим массив с названиями колонок
arr_columns = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area',
'price_by_meter', 'price', 'material', 'build_year', 'rubbish_chute', 'build_overlap',
'heating', 'gas', 'areaRating', 'city', 'remoute_from_center', 'build_oldest']
# Удаляем строки по кол-ву nan
price_apart1.delete_nan_rows(count_nan_in_line)
# type
# Так как специализированного жилищного фонда одна позиция, то удалим
такую строку
for index, row in price_apart1.dataset.iterrows():
    if row['type'] == 'Специализированный жилищный фонд':
        price_apart1.dataset.drop(labels=[index], inplace=True)
# Преобразование в числовые признаки
change_type = {'Квартира': 1, 'Жилой дом блокированной застройки': 46,
'Многоквартирный дом': 77}
price_apart1.text_to_numbers(change_type, 'type')
# Заменяем nan на наиболее повторяющуюся позицию
price_apart1.change_nan_to_popular('type')
#level
# Заменяем дроби на поделенные дроби, неизвестные данные на самое
встречаемое значение
price_apart1.fractions_into_numbers('level')
#rooms
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rooms')
# Нормализуем данные

```

```

rooms_scaler = 'StandartScaler'
price_apart1.scaler(rooms_scaler, 'rooms')
#kitchen_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('kitchen_area')
kitchen_area_scaler = 'StandartScaler'
price_apart1.scaler(kitchen_area_scaler, 'kitchen_area')
#living_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('living_area')
living_area_scaler = 'StandartScaler'
price_apart1.scaler(living_area_scaler, 'living_area')
#area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('area')
area_scaler = 'MinMaxScaler'
price_apart1.scaler(area_scaler, 'area')
#price
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('price')
price_scaler = 'MinMaxScaler'
price_apart1.scaler(price_scaler, 'price')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#areaRating
# Нормализуем
areaRating_scaler = 'MinMaxScaler'

```



```

price_apart1.scaler(areaRating_scaler, 'areaRating')
#remoute_from_center
remoute_from_center_scaler = 'StandartScaler'
price_apart1.scaler(remoute_from_center_scaler, 'remoute_from_center')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#material
change_material = {'Панель': 7, 'Кирпич': 10, 'Блоки': 8, 'Монолит': 6, 'Дерево': 5}
price_apart1.text_to_numbers(change_material, 'material')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('material')
# Нормализуем данные
material_scaler = 'MinMaxScaler'
price_apart1.scaler(material_scaler, 'material')
#rubbish_chute
change_rubbish_chute = {'Нет': 0, 'Есть': 1}
price_apart1.text_to_numbers(change_rubbish_chute, 'rubbish_chute')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rubbish_chute')
# Нормализуем данные
rubbish_chute_scaler = 'MinMaxScaler'
price_apart1.scaler(rubbish_chute_scaler, 'rubbish_chute')
#build_overlap
build_overlap_change = {'Железобетонные': 11, 'Смешанные': 6, 'Деревянные': 1}
price_apart1.text_to_numbers(build_overlap_change, 'build_overlap')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_overlap')

```

```

# Нормализуем данные
build_overlap_scaler = 'MinMaxScaler'
price_apart1.scaler(build_overlap_scaler, 'build_overlap')

#gas
change_gas = {'Нет': 0, 'Да': 1}
price_apart1.text_to_numbers(change_gas, 'gas')

# Заменим пропуски на среднее значение
price_apart1.fillna_row('gas')

# Нормализуем данные
gas_scaler = 'MinMaxScaler'
price_apart1.scaler(gas_scaler, 'gas')

#build_oldest
build_oldest_change = {'new': 3, 'middle': 2, 'old': 1}
price_apart1.text_to_numbers(build_oldest_change, 'build_oldest')

# Заменим пропуски на среднее значение
price_apart1.fillna_row('build_oldest')

# Нормализуем данные
build_oldest_scaler = 'MinMaxScaler'
price_apart1.scaler(build_oldest_scaler, 'build_oldest')

city_copy = price_apart1.dataset['city']
for change_city_Yoshkar_Ola in range(1, 1 + 1):
    for change_city_Cheboksary in range(change_city_Yoshkar_Ola + 1, 30 + 1):
        for change_city_Kazan in range(change_city_Cheboksary + 1, 60 + 1):
            for change_city_Moscow in range(change_city_Kazan + 1, 90 + 1):
                price_apart1.dataset['city'] = city_copy
                change_city = {'Москва': change_city_Moscow, 'Казань':
change_city_Kazan, 'Чебоксары': change_city_Cheboksary, 'Йошкар-Ола':
change_city_Yoshkar_Ola}
                price_apart1.text_to_numbers(change_city, 'city')
                price_apart1.fillna_row('city')

```

```

city_scaler = 'MinMaxScaler'
price_apart1.scaler(city_scaler, 'city')
# Обучение
num_test_size = 0.9
X = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area', 'material',
'build_year', 'rubbish_chute', 'build_overlap', 'gas', 'areaRating', 'city',
'remoute_from_center', 'build_oldest']
y = 'price'
price_apart1.training(X, y, num_test_size)
coeff_rows = ["Тип (вид)", "Кол-во комнат", "Этаж", "Кухонная площадь",
"Жилая площадь", "Общая площадь", "Материал", "Год постройки",
"Мусоропровод (есть ли)", "Перекрытие (вид)", "Газ (есть ли)", "Рейтинг
местности", "Город", "Удаленность от центра", "Старость здания"]
coeff = price_apart1.coefficient(coeff_rows, ['Коэффициенты'])
pred = price_apart1.price_pred()
MSE, MAE = price_apart1.return_errors()[0], price_apart1.return_errors()[1]
if MSE + MAE < global_sum_errors:
    global_city_Yoshkar_Ola = change_city_Yoshkar_Ola
    global_city_Cheboksary = change_city_Cheboksary
    global_city_Kazan = change_city_Kazan
    global_city_Moscow = change_city_Moscow
    global_sum_errors = MSE + MAE
print('\n')
print(global_city_Yoshkar_Ola, 'Yoshkar_Ola')
print(global_city_Cheboksary, 'Cheboksary')
print(global_city_Kazan, 'Kazan')
print(global_city_Moscow, 'Moscow')
print(global_sum_errors)
print("--- %s seconds ---" % (time.time() - start_time))
# Заменяем категориальные данные числовыми для оставшихся параметров

```

```

# build_oldest
global_build_oldest_new = None
global_build_oldest_middle = None
global_build_oldest_old = None
start_time = time.time()
count_nan_in_line = 5
global_sum_errors = 10 ** 10
price_apart1 = Predict_price('E:/University/Основы машинного
обучения/Datasets/rent_apartments.csv')
for column in 'object_type', 'build_type', 'build_walls', 'build_serias', 'published',
'updated', 'longitude', 'latitude', 'agencyName', 'enterances', 'max_levels','min_levels',
'heating':
    price_apart1.pop_column(column)
# Создадим массив с названиями колонок
arr_columns = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area',
'price_by_meter', 'price', 'material', 'build_year', 'rubbish_chute', 'build_overlap',
'heating', 'gas', 'areaRating', 'city', 'remoute_from_center', 'build_oldest']
# Удаляем строки по кол-ву nan
price_apart1.delete_nan_rows(count_nan_in_line)
# type
# Так как специализированного жилищного фонда одна позиция, то удалим
такую строку
for index, row in price_apart1.dataset.iterrows():
    if row['type'] == 'Специализированный жилищный фонд':
        price_apart1.dataset.drop(labels=[index], inplace=True)
# Преобразование в числовые признаки
change_type = {'Квартира': 1, 'Жилой дом блокированной застройки': 46,
'Многоквартирный дом': 77}
price_apart1.text_to_numbers(change_type, 'type')
# Заменяем nan на наиболее повторяющуюся позицию

```

```

price_apart1.change_nan_to_popular('type')
#level
# Заменяем дроби на поделенные дроби, неизвестные данные на самое
встречаемое значение
price_apart1.fractions_into_numbers('level')
#rooms
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rooms')
# Нормализуем данные
rooms_scaler = 'StandartScaler'
price_apart1.scaler(rooms_scaler, 'rooms')
#kitchen_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('kitchen_area')
kitchen_area_scaler = 'StandartScaler'
price_apart1.scaler(kitchen_area_scaler, 'kitchen_area')
#living_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('living_area')
living_area_scaler = 'StandartScaler'
price_apart1.scaler(living_area_scaler, 'living_area')
#area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('area')
area_scaler = 'MinMaxScaler'
price_apart1.scaler(area_scaler, 'area')
#price
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('price')
price_scaler = 'MinMaxScaler'

```

```

price_apart1.scaler(price_scaler, 'price')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#areaRating
# Нормализуем
areaRating_scaler = 'MinMaxScaler'
price_apart1.scaler(areaRating_scaler, 'areaRating')
#remoute_from_center
remoute_from_center_scaler = 'StandartScaler'
price_apart1.scaler(remoute_from_center_scaler, 'remoute_from_center')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#material
change_material = {'Панель': 7, 'Кирпич': 10, 'Блоки': 8, 'Монолит': 6, 'Дерево': 5}
price_apart1.text_to_numbers(change_material, 'material')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('material')
# Нормализуем данные
material_scaler = 'MinMaxScaler'
price_apart1.scaler(material_scaler, 'material')
#rubbish_chute
change_rubbish_chute = {'Нет': 0, 'Есть': 1}
price_apart1.text_to_numbers(change_rubbish_chute, 'rubbish_chute')
# Заменяем пропуски на среднее значение

```

```

price_apart1.fillna_row('rubbish_chute')
# Нормализуем данные
rubbish_chute_scaler = 'MinMaxScaler'
price_apart1.scaler(rubbish_chute_scaler, 'rubbish_chute')
#build_overlap
build_overlap_change = {'Железобетонные': 11, 'Смешанные': 6, 'Деревянные': 1}
price_apart1.text_to_numbers(build_overlap_change, 'build_overlap')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_overlap')
# Нормализуем данные
build_overlap_scaler = 'MinMaxScaler'
price_apart1.scaler(build_overlap_scaler, 'build_overlap')
#gas
change_gas = {'Нет': 0, 'Да': 1}
price_apart1.text_to_numbers(change_gas, 'gas')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('gas')
# Нормализуем данные
gas_scaler = 'MinMaxScaler'
price_apart1.scaler(gas_scaler, 'gas')
#city
change_city = {'Москва': 87, 'Казань': 15, 'Чебоксары': 9, 'Йошкар-Ола': 1}
price_apart1.text_to_numbers(change_city, 'city')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('city')
# Нормализуем данные
city_scaler = 'MinMaxScaler'
price_apart1.scaler(city_scaler, 'city')
build_oldest_copy = price_apart1.dataset['build_oldest']
for change_build_oldest_old in range(1, 1 + 1):

```

```

for change_build_oldest_middle in range(change_build_oldest_old + 1, 10 + 1):
    for change_build_oldest_new in range(change_build_oldest_middle + 1, 20 + 1):
        price_apart1.dataset['build_oldest'] = build_oldest_copy
        build_oldest_change = {'new': change_build_oldest_new, 'middle':
change_build_oldest_middle, 'old': change_build_oldest_old}
        price_apart1.text_to_numbers(build_oldest_change, 'build_oldest')
        price_apart1.fillna_row('build_oldest')
        build_oldest_scaler = 'MinMaxScaler'
        price_apart1.scaler(build_oldest_scaler, 'build_oldest')
        # Обучение
        num_test_size = 0.9
        X = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area', 'material',
'build_year', 'rubbish_chute', 'build_overlap', 'gas', 'areaRating', 'city',
'remoute_from_center', 'build_oldest']
        y = 'price'
        price_apart1.training(X, y, num_test_size)
        coeff_rows = ["Тип (вид)", "Кол-во комнат", "Этаж", "Кухонная площадь",
"Жилая площадь", "Общая площадь", "Материал", "Год постройки",
"Мусоропровод (есть ли)", "Перекрытие (вид)", "Газ (есть ли)", "Рейтинг
местности", "Город", "Удаленность от центра", "Старость здания"]
        coeff = price_apart1.coefficient(coeff_rows, ['Коэффициенты'])
        pred = price_apart1.price_pred()
        MSE, MAE = price_apart1.return_errors()[0], price_apart1.return_errors()[1]
        if MSE + MAE < global_sum_errors:
            global_build_oldest_new = change_build_oldest_new
            global_build_oldest_middle = change_build_oldest_middle
            global_build_oldest_old = change_build_oldest_old
            global_sum_errors = MSE + MAE
print('\n')
print(global_build_oldest_new, 'new')

```



```

print(global_build_oldest_middle, 'middle')
print(global_build_oldest_old, 'old')
print(global_sum_errors)
print("--- %s seconds ---" % (time.time() - start_time))
# Найдем наилучшее значение для размера тестовой выборки
global_num_test_size = None
start_time = time.time()
global_sum_errors = 10 ** 10
count_nan_in_line = 5
price_apart1 = Predict_price('E:/University/Основы машинного
обучения/Datasets/rent_apartments.csv')
for column in 'object_type', 'build_type', 'build_walls', 'build_serias', 'published',
'updated', 'longitude', 'latitude', 'agencyName', 'enterances', 'max_levels', 'min_levels',
'heating':
    price_apart1.pop_column(column)
# Создадим массив с названиями колонок
arr_columns = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area',
'price_by_meter', 'price', 'material', 'build_year', 'rubbish_chute', 'build_overlap',
'heating', 'gas', 'areaRating', 'city', 'remoute_from_center', 'build_oldest']
# Удаляем строки по кол-ву nan
price_apart1.delete_nan_rows(count_nan_in_line)
# type
# Так как специализированного жилищного фонда одна позиция, то удалим
такую строку
for index, row in price_apart1.dataset.iterrows():
    if row['type'] == 'Специализированный жилищный фонд':
        price_apart1.dataset.drop(labels=[index], inplace=True)
# Преобразование в числовые признаки
change_type = {'Квартира': 1, 'Жилой дом блокированной застройки': 46,
'Многоквартирный дом': 77}

```

```

price_apart1.text_to_numbers(change_type, 'type')
# Заменяем nan на наиболее повторяющуюся позицию
price_apart1.change_nan_to_popular('type')
#level
# Заменяем дроби на поделенные дроби, неизвестные данные на самое
встречаемое значение
price_apart1.fractions_into_numbers('level')
#rooms
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rooms')
# Нормализуем данные
rooms_scaler = 'StandartScaler'
price_apart1.scaler(rooms_scaler, 'rooms')
#kitchen_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('kitchen_area')
kitchen_area_scaler = 'StandartScaler'
price_apart1.scaler(kitchen_area_scaler, 'kitchen_area')
#living_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('living_area')
living_area_scaler = 'StandartScaler'
price_apart1.scaler(living_area_scaler, 'living_area')
#area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('area')
area_scaler = 'MinMaxScaler'
price_apart1.scaler(area_scaler, 'area')
#price
# Заменяем пропуски на среднее значение

```

```

price_apart1.fillna_row('price')
price_scaler = 'MinMaxScaler'
price_apart1.scaler(price_scaler, 'price')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#areaRating
# Нормализуем
areaRating_scaler = 'MinMaxScaler'
price_apart1.scaler(areaRating_scaler, 'areaRating')
#remoute_from_center
remoute_from_center_scaler = 'StandartScaler'
price_apart1.scaler(remoute_from_center_scaler, 'remoute_from_center')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#material
change_material = {'Панель': 7, 'Кирпич': 10, 'Блоки': 8, 'Монолит': 6, 'Дерево': 5}
price_apart1.text_to_numbers(change_material, 'material')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('material')
# Нормализуем данные
material_scaler = 'MinMaxScaler'
price_apart1.scaler(material_scaler, 'material')
#rubbish_chute
change_rubbish_chute = {'Нет': 0, 'Есть': 1}

```

```

price_apart1.text_to_numbers(change_rubbish_chute, 'rubbish_chute')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rubbish_chute')
# Нормализуем данные
rubbish_chute_scaler = 'MinMaxScaler'
price_apart1.scaler(rubbish_chute_scaler, 'rubbish_chute')
#build_overlap
build_overlap_change = {'Железобетонные': 10, 'Смешанные': 5, 'Деревянные': 1}
price_apart1.text_to_numbers(build_overlap_change, 'build_overlap')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_overlap')
# Нормализуем данные
build_overlap_scaler = 'MinMaxScaler'
price_apart1.scaler(build_overlap_scaler, 'build_overlap')
#gas
change_gas = {'Нет': 0, 'Да': 1}
price_apart1.text_to_numbers(change_gas, 'gas')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('gas')
# Нормализуем данные
gas_scaler = 'MinMaxScaler'
price_apart1.scaler(gas_scaler, 'gas')
#city
change_city = {'Москва': 10, 'Казань': 8, 'Чебоксары': 6, 'Йошкар-Ола': 3}
price_apart1.text_to_numbers(change_city, 'city')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('city')
# Нормализуем данные
city_scaler = 'MinMaxScaler'
price_apart1.scaler(city_scaler, 'city')

```

```

#build_oldest
build_oldest_change = {'new': 17, 'middle': 3, 'old': 1}
price_apart1.text_to_numbers(build_oldest_change, 'build_oldest')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_oldest')
# Нормализуем данные
build_oldest_scaler = 'MinMaxScaler'
price_apart1.scaler(build_oldest_scaler, 'build_oldest')
# Обучение
for num_test_size in range(5, 95, 5):
    num_test_size = num_test_size / 100
    X = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area', 'material', 'build_year',
'rubbish_chute', 'build_overlap', 'gas', 'areaRating', 'city', 'remoute_from_center',
'build_oldest']
    y = 'price'
    price_apart1.training(X, y, num_test_size)
    coeff_rows = ["Тип (вид)", "Кол-во комнат", "Этаж", "Кухонная площадь",
"Жилая площадь", "Общая площадь", "Материал", "Год постройки",
"Мусоропровод (есть ли)", "Перекрытие (вид)", "Газ (есть ли)", "Рейтинг
местности", "Город", "Удаленность от центра", "Старость здания"]
    coeff = price_apart1.coefficient(coeff_rows, ['Коэффициенты'])
    pred = price_apart1.price_pred()
    MSE, MAE = price_apart1.return_errors()[0], price_apart1.return_errors()[1]
    if MSE + MAE < global_sum_errors:
        global_num_test_size = num_test_size
        global_sum_errors = MSE + MAE
print("\n")
print(global_num_test_size, 'test size')
print(global_sum_errors)
print("--- %s seconds ---" % (time.time() - start_time))

```

```

count_nan_in_line = 5

price_apart1 = Predict_price('E:/University/Основы машинного
обучения/Datasets/rent_apartments.csv')

for column in 'object_type', 'build_type', 'build_walls', 'build_serias', 'published',
'updated', 'longitude', 'latitude', 'agencyName', 'enterances', 'max_levels','min_levels',
'heating':

    price_apart1.pop_column(column)

# Создадим массив с названиями колонок
arr_columns = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area',
'price_by_meter', 'price', 'material', 'build_year', 'rubbish_chute', 'build_overlap',
'heating', 'gas', 'areaRating', 'city', 'remoute_from_center', 'build_oldest']

# Удаляем строки по кол-ву nan
price_apart1.delete_nan_rows(count_nan_in_line)

# type

# Так как специализированного жилищного фонда одна позиция, то удалим
такую строку
for index, row in price_apart1.dataset.iterrows():

    if row['type'] == 'Специализированный жилищный фонд':

        price_apart1.dataset.drop(labels=[index], inplace=True)

# Преобразование в числовые признаки
change_type = {'Квартира': 1, 'Жилой дом блокированной застройки': 46,
'Многоквартирный дом': 77}
price_apart1.text_to_numbers(change_type, 'type')

# Заменяем nan на наиболее повторяющуюся позицию
price_apart1.change_nan_to_popular('type')

type_scaler = 'MinMaxScaler'
price_apart1.scaler(type_scaler, 'type')

#level

# Заменяем дроби на поделенные дроби, неизвестные данные на самое
встречаемое значение

```

```

price_apart1.fractions_into_numbers('level')
#rooms
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rooms')
# Нормализуем данные
rooms_scaler = 'StandardScaler'
price_apart1.scaler(rooms_scaler, 'rooms')
#kitchen_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('kitchen_area')
kitchen_area_scaler = 'StandardScaler'
price_apart1.scaler(kitchen_area_scaler, 'kitchen_area')
#living_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('living_area')
living_area_scaler = 'StandardScaler'
price_apart1.scaler(living_area_scaler, 'living_area')
#area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('area')
area_scaler = 'MinMaxScaler'
price_apart1.scaler(area_scaler, 'area')
#price
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('price')
price_scaler = 'MinMaxScaler'
price_apart1.scaler(price_scaler, 'price')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')

```

```

build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#areaRating
# Нормализуем
areaRating_scaler = 'MinMaxScaler'
price_apart1.scaler(areaRating_scaler, 'areaRating')
#remoute_from_center
remoute_from_center_scaler = 'StandartScaler'
price_apart1.scaler(remoute_from_center_scaler, 'remoute_from_center')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#material
change_material = {'Панель': 7, 'Кирпич': 10, 'Блоки': 8, 'Монолит': 6, 'Дерево': 5}
price_apart1.text_to_numbers(change_material, 'material')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('material')
# Нормализуем данные
material_scaler = 'MinMaxScaler'
price_apart1.scaler(material_scaler, 'material')
#rubbish_chute
change_rubbish_chute = {'Нет': 0, 'Есть': 1}
price_apart1.text_to_numbers(change_rubbish_chute, 'rubbish_chute')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rubbish_chute')
# Нормализуем данные
rubbish_chute_scaler = 'MinMaxScaler'
price_apart1.scaler(rubbish_chute_scaler, 'rubbish_chute')

```



```

#build_overlap
build_overlap_change = {'Железобетонные': 10, 'Смешанные': 5, 'Деревянные': 1}
price_apart1.text_to_numbers(build_overlap_change, 'build_overlap')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_overlap')
# Нормализуем данные
build_overlap_scaler = 'MinMaxScaler'
price_apart1.scaler(build_overlap_scaler, 'build_overlap')

#gas
change_gas = {'Нет': 0, 'Да': 1}
price_apart1.text_to_numbers(change_gas, 'gas')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('gas')
# Нормализуем данные
gas_scaler = 'MinMaxScaler'
price_apart1.scaler(gas_scaler, 'gas')

#city
change_city = {'Москва': 10, 'Казань': 8, 'Чебоксары': 6, 'Йошкар-Ола': 3}
price_apart1.text_to_numbers(change_city, 'city')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('city')
# Нормализуем данные
city_scaler = 'MinMaxScaler'
price_apart1.scaler(city_scaler, 'city')

#build_oldest
build_oldest_change = {'new': 17, 'middle': 3, 'old': 1}
price_apart1.text_to_numbers(build_oldest_change, 'build_oldest')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_oldest')
# Нормализуем данные

```

```

build_oldest_scaler = 'MinMaxScaler'
price_apart1.scaler(build_oldest_scaler, 'build_oldest')
# Обучение
num_test_size = 0.85
X = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area', 'material', 'build_year',
'rubbish_chute', 'build_overlap', 'gas', 'areaRating', 'city', 'remoute_from_center',
'build_oldest']
y = 'price'
price_apart1.training(X, y, num_test_size)
coeff_rows = ["Тип (вид)", "Кол-во комнат", "Этаж", "Кухонная площадь",
"Жилая площадь", "Общая площадь", "Материал", "Год постройки",
"Мусоропровод (есть ли)", "Перекрытие (вид)", "Газ (есть ли)", "Рейтинг
местности", "Город", "Удаленность от центра", "Старость здания"]
coeff = price_apart1.coefficient(coeff_rows, ['Коэффициенты'])
pred = price_apart1.price_pred()
MSE, MAE = price_apart1.return_errors()[0], price_apart1.return_errors()[1]
print(coeff)
print('\n')
print(pred)
print('\n')
print(f'MSE - {MSE :.3}')
print(f'MAE - {MAE :.3}')
start_time = time.time()
count_nan_in_line = 5
global_sum_errors = 10 ** 10
global_coeff_rows = None
global_pred = None
global_MSE = None
global_MAE = None

```

```

price_apart1 = Predict_price('E:/University/Основы машинного
обучения/Datasets/rent_apartments.csv')
for column in 'object_type', 'build_type', 'build_walls', 'build_serias', 'published',
'updated', 'longitude', 'latitude', 'agencyName', 'enterances', 'max_levels','min_levels',
'heating':
    price_apart1.pop_column(column)
# Создадим массив с названиями колонок
arr_columns = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area',
'price_by_meter', 'price', 'material', 'build_year', 'rubbish_chute', 'build_overlap',
'heating', 'gas', 'areaRating', 'city', 'remoute_from_center', 'build_oldest']
# Удаляем строки по кол-ву nan
price_apart1.delete_nan_rows(count_nan_in_line)
# type
# Так как специализированного жилищного фонда одна позиция, то удалим
такую строку
for index, row in price_apart1.dataset.iterrows():
    if row['type'] == 'Специализированный жилищный фонд':
        price_apart1.dataset.drop(labels=[index], inplace=True)
# Преобразование в числовые признаки
change_type = {'Квартира': 1, 'Жилой дом блокированной застройки': 46,
'Многоквартирный дом': 77}
price_apart1.text_to_numbers(change_type, 'type')
# Заменяем nan на наиболее повторяющуюся позицию
price_apart1.change_nan_to_popular('type')
type_scaler = 'StandartScaler'
price_apart1.scaler(type_scaler, 'type')
#level
# Заменяем дроби на поделенные дроби, неизвестные данные на самое
встречаемое значение
price_apart1.fractions_into_numbers('level')

```

```
#rooms
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rooms')
# Нормализуем данные
rooms_scaler = 'StandardScaler'
price_apart1.scaler(rooms_scaler, 'rooms')
#kitchen_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('kitchen_area')
kitchen_area_scaler = 'StandardScaler'
price_apart1.scaler(kitchen_area_scaler, 'kitchen_area')
#living_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('living_area')
living_area_scaler = 'StandardScaler'
price_apart1.scaler(living_area_scaler, 'living_area')
#area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('area')
area_scaler = 'MinMaxScaler'
price_apart1.scaler(area_scaler, 'area')
#price
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('price')
price_scaler = 'MinMaxScaler'
price_apart1.scaler(price_scaler, 'price')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
```

```

price_apart1.scaler(build_year_scaler, 'build_year')
#areaRating
# Нормализуем
areaRating_scaler = 'MinMaxScaler'
price_apart1.scaler(areaRating_scaler, 'areaRating')
#remoute_from_center
remoute_from_center_scaler = 'StandartScaler'
price_apart1.scaler(remoute_from_center_scaler, 'remoute_from_center')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#material
change_material = {'Панель': 7, 'Кирпич': 10, 'Блоки': 8, 'Монолит': 6, 'Дерево': 5}
price_apart1.text_to_numbers(change_material, 'material')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('material')
# Нормализуем данные
material_scaler = 'MinMaxScaler'
price_apart1.scaler(material_scaler, 'material')
#rubbish_chute
change_rubbish_chute = {'Нет': 0, 'Есть': 1}
price_apart1.text_to_numbers(change_rubbish_chute, 'rubbish_chute')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rubbish_chute')
# Нормализуем данные
rubbish_chute_scaler = 'MinMaxScaler'
price_apart1.scaler(rubbish_chute_scaler, 'rubbish_chute')
#build_overlap

```

```

build_overlap_change = {'Железобетонные': 10, 'Смешанные': 5, 'Деревянные': 1}
price_apart1.text_to_numbers(build_overlap_change, 'build_overlap')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_overlap')
# Нормализуем данные
build_overlap_scaler = 'MinMaxScaler'
price_apart1.scaler(build_overlap_scaler, 'build_overlap')

#gas
change_gas = {'Нет': 0, 'Да': 1}
price_apart1.text_to_numbers(change_gas, 'gas')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('gas')
# Нормализуем данные
gas_scaler = 'MinMaxScaler'
price_apart1.scaler(gas_scaler, 'gas')

#city
change_city = {'Москва': 10, 'Казань': 8, 'Чебоксары': 6, 'Йошкар-Ола': 3}
price_apart1.text_to_numbers(change_city, 'city')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('city')
# Нормализуем данные
city_scaler = 'MinMaxScaler'
price_apart1.scaler(city_scaler, 'city')

#build_oldest
build_oldest_change = {'new': 17, 'middle': 3, 'old': 1}
price_apart1.text_to_numbers(build_oldest_change, 'build_oldest')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_oldest')
# Нормализуем данные
build_oldest_scaler = 'MinMaxScaler'

```

```

price_apart1.scaler(build_oldest_scaler, 'build_oldest')
# Обучение
num_test_size = 0.85
X = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area', 'material', 'build_year',
'rubbish_chute', 'build_overlap', 'gas', 'areaRating', 'city', 'remoute_from_center',
'build_oldest']
y = 'price'
for cnt_column in range(1, 9):
    mean_global_sum_errors = []
    print(cnt_column, ' - cnt_column')
    print("--- %s seconds ---" % (time.time() - start_time))
    variants = []
    for X_columns in itertools.permutations(X, cnt_column):
        if list(set(X_columns)) not in variants:
            price_apart1.training(list(X_columns), y, num_test_size)
            variants.append(list(set(X_columns)))
            pred = price_apart1.price_pred()
            MSE, MAE = price_apart1.return_errors()[0], price_apart1.return_errors()[1]
            mean_global_sum_errors.append(MSE + MAE)
            if MSE + MAE < global_sum_errors:
                global_coeff_rows = price_apart1.coefficient(X_columns,
['Коэффициенты'])
                global_sum_errors = MSE + MAE
                global_pred = price_apart1.price_pred()
                global_MSE = MSE
                global_MAE = MAE
                print(f'MSE = {global_MSE}, MAE = {global_MAE}, Количество
столбцов - {cnt_column}, MSE + MAE = {global_MSE + global_MAE}, columns =
{list(set(X_columns))}')
    print(len(variants), ' - количество вариантов')

```

```

    print(sum(mean_global_sum_errors) / len(mean_global_sum_errors), ' - средняя
сумма MSE и MAE при данном количестве столбцов')
print('\n')
print(global_coeff_rows)
print('\n')
print(global_pred)
print('\n')
print(f'MSE - {global_MSE :.3}')
print(f'MAE - {global_MAE :.3}')
print("--- %s seconds ---" % (time.time() - start_time))
count_nan_in_line = 5
price_apart1 = Predict_price('E:/University/Основы машинного
обучения/Datasets/rent_apartments.csv')
for column in 'object_type', 'build_type', 'build_walls', 'build_serias', 'published',
'updated', 'longitude', 'latitude', 'agencyName', 'enterances', 'max_levels', 'min_levels',
'heating':
    price_apart1.pop_column(column)
# Создадим массив с названиями колонок
arr_columns = ['type', 'rooms', 'level', 'kitchen_area', 'living_area', 'area',
'price_by_meter', 'price', 'material', 'build_year', 'rubbish_chute', 'build_overlap',
'heating', 'gas', 'areaRating', 'city', 'remoute_from_center', 'build_oldest']
# Удаляем строки по кол-ву nan
price_apart1.delete_nan_rows(count_nan_in_line)
# type
# Так как специализированного жилищного фонда одна позиция, то удалим
такую строку
for index, row in price_apart1.dataset.iterrows():
    if row['type'] == 'Специализированный жилищный фонд':
        price_apart1.dataset.drop(labels=[index], inplace=True)
# Преобразование в числовые признаки

```



```

change_type = {'Квартира': 1, 'Жилой дом блокированной застройки': 46,
'Mногоквартирный дом': 77}
price_apart1.text_to_numbers(change_type, 'type')
# Заменяем nan на наиболее повторяющуюся позицию
price_apart1.change_nan_to_popular('type')
#level
# Заменяем дроби на поделенные дроби, неизвестные данные на самое
встречаемое значение
price_apart1.fractions_into_numbers('level')
#rooms
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rooms')
#kitchen_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('kitchen_area')
#living_area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('living_area')
#area
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('area')
#price
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('price')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')
#build_year
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_year')

```

```

#material
change_material = {'Панель': 7, 'Кирпич': 10, 'Блоки': 8, 'Монолит': 6, 'Дерево': 5}
price_apart1.text_to_numbers(change_material, 'material')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('material')
#rubbish_chute
change_rubbish_chute = {'Нет': 0, 'Есть': 1}
price_apart1.text_to_numbers(change_rubbish_chute, 'rubbish_chute')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('rubbish_chute')
#build_overlap
build_overlap_change = {'Железобетонные': 10, 'Смешанные': 5, 'Деревянные': 1}
price_apart1.text_to_numbers(build_overlap_change, 'build_overlap')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_overlap')
#gas
change_gas = {'Нет': 0, 'Да': 1}
price_apart1.text_to_numbers(change_gas, 'gas')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('gas')
#city
change_city = {'Москва': 10, 'Казань': 8, 'Чебоксары': 6, 'Йошкар-Ола': 3}
price_apart1.text_to_numbers(change_city, 'city')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('city')
#build_oldest
build_oldest_change = {'new': 17, 'middle': 3, 'old': 1}
price_apart1.text_to_numbers(build_oldest_change, 'build_oldest')
# Заменяем пропуски на среднее значение
price_apart1.fillna_row('build_oldest')

```

```
boxplot = price_apart1.dataset.boxplot(column=['type'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['rooms'])
plt.show()
price_apart1.remove_outliers('rooms')
boxplot = price_apart1.dataset.boxplot(column=['rooms'])
plt.show()
price_apart1.remove_outliers('kitchen_area')
boxplot = price_apart1.dataset.boxplot(column=['kitchen_area'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['living_area'])
plt.show()
price_apart1.remove_outliers('living_area')
boxplot = price_apart1.dataset.boxplot(column=['living_area'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['area'])
plt.show()
price_apart1.remove_outliers('area')
boxplot = price_apart1.dataset.boxplot(column=['area'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['price'])
plt.show()
price_apart1.remove_outliers('price')
boxplot = price_apart1.dataset.boxplot(column=['price'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['material'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['build_year'])
plt.show()
price_apart1.remove_outliers('build_year')
```

```

boxplot = price_apart1.dataset.boxplot(column=['build_year'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['rubbish_chute'])
plt.show()
price_apart1.remove_outliers('rubbish_chute')
boxplot = price_apart1.dataset.boxplot(column=['rubbish_chute'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['build_overlap'])
plt.show()
price_apart1.remove_outliers('build_overlap')
boxplot = price_apart1.dataset.boxplot(column=['build_overlap'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['gas'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['areaRating'])
plt.show()
price_apart1.remove_outliers('areaRating')
boxplot = price_apart1.dataset.boxplot(column=['areaRating'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['city'])
plt.show()
price_apart1.remove_outliers('city')
boxplot = price_apart1.dataset.boxplot(column=['city'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['remoute_from_center'])
plt.show()
price_apart1.remove_outliers('remoute_from_center')
boxplot = price_apart1.dataset.boxplot(column=['remoute_from_center'])
plt.show()
boxplot = price_apart1.dataset.boxplot(column=['build_oldest'])

```

```

plt.show()
# type
type_scaler = 'MinMaxScaler'
price_apart1.scaler(type_scaler, 'type')
#rooms
rooms_scaler = 'StandartScaler'
price_apart1.scaler(rooms_scaler, 'rooms')
#kitchen_area
kitchen_area_scaler = 'StandartScaler'
price_apart1.scaler(kitchen_area_scaler, 'kitchen_area')
#living_area
living_area_scaler = 'StandartScaler'
price_apart1.scaler(living_area_scaler, 'living_area')
#area
area_scaler = 'MinMaxScaler'
price_apart1.scaler(area_scaler, 'area')
#price
price_scaler = 'MinMaxScaler'
price_apart1.scaler(price_scaler, 'price')
#build_year
build_year_scaler = 'MinMaxScaler'
price_apart1.scaler(build_year_scaler, 'build_year')
#areaRating
areaRating_scaler = 'MinMaxScaler'
price_apart1.scaler(areaRating_scaler, 'areaRating')
#remoute_from_center
remoute_from_center_scaler = 'StandartScaler'
price_apart1.scaler(remoute_from_center_scaler, 'remoute_from_center')
#build_year
build_year_scaler = 'MinMaxScaler'

```

```

price_apart1.scaler(build_year_scaler, 'build_year')
#material
material_scaler = 'MinMaxScaler'
price_apart1.scaler(material_scaler, 'material')
#rubbish_chute
rubbish_chute_scaler = 'MinMaxScaler'
price_apart1.scaler(rubbish_chute_scaler, 'rubbish_chute')
#build_overlap
build_overlap_scaler = 'MinMaxScaler'
price_apart1.scaler(build_overlap_scaler, 'build_overlap')
#gas
gas_scaler = 'MinMaxScaler'
price_apart1.scaler(gas_scaler, 'gas')
#city
city_scaler = 'MinMaxScaler'
price_apart1.scaler(city_scaler, 'city')
#build_oldest
build_oldest_scaler = 'MinMaxScaler'
price_apart1.scaler(build_oldest_scaler, 'build_oldest')
price_apart1.sb_pairplot()
plt.figure(figsize = (12,8))
ax = sb.heatmap(price_apart1.dataset.corr(), annot = True, fmt = ".2f")
i, k = ax.get_ylim()
ax.set_ylim(i+0.5, k-0.5)
plt.show()
price_apart1.dataset.set_index([[x for x in range(len(price_apart1.dataset['rooms']))]],
inplace=True)
price_apart1.dataset
num_test_size = 0.85
X = ['type', 'areaRating', 'build_oldest', 'city', 'area', 'remoute_from_center', 'material']

```

```

y = 'price'
price_apart1.training(X, y, num_test_size)
#coeff_rows = ["Тип (вид)", "Кол-во комнат", "Этаж", "Кухонная площадь",
"Жилая площадь", "Общая площадь", "Материал", "Год постройки",
"Мусоропровод (есть ли)", "Перекрытие (вид)", "Газ (есть ли)", "Рейтинг
местности", "Город", "Удаленность от центра", "Старость здания"]
coeff_rows = ["Тип (вид)", "Рейтинг местности", "Старость здания", "Город",
"Общая площадь", "Удаленность от центра", "Материал"]
coeff = price_apart1.coefficient(coeff_rows, ['Коэффициенты'])
pred = price_apart1.price_pred()
MSE, MAE, MAPE = price_apart1.return_errors()[0], price_apart1.return_errors()[1],
price_apart1.return_errors()[2]
print(coeff)
print('\n')
print(pred)
print('\n')
print(f'MSE - {MSE :.3}')
print(f'MAE - {MAE :.3}')

```